



www.mohandesyar.com

عنوان

کتاب آموزش جامع

زبان برنامه نویسی C#

اسلام احمد زاده

هرگونه انتقاد و پیشنهاد در مورد مطالب کتاب را با شماره ۰۹۱۷۷۱۱۲۱۶۱ در میان بگذارید تا در

نسخه های بعدی اصلاحات لازم صورت گیرد. با تشکر

این کتاب الکترونیکی رایگان است و هرگونه نسخه برداری از مطالب آن بصورت کلی یا جزئی مجاز

میباشد. این کتاب در واقع هدیه ای است به هموطنان عزیز.

www.IrPDF.com

(من اسلام احمد زاده در زمان نوشتن این کتاب استاد آموزشکده فنی باهنر شیراز و موسسه غیر انتفاعی زند شیراز میباشم و دست تمام دانشجویانی را میبوسم که با رفتار خود به من بزرگ منشی را یاد داده اند و از تمام دانشجویانی که احساس میکنند در نمره دادن به آنها ظلم کرده ام عذر خواهی میکنم.)

فهرست مطالب

۱-۲-۳-۱-دستورات ۵۸if	۱-۱-مروری بر ساختار. ۱۳NET
۲-۲-۳-۲-بلوک های تک دستوری ۵۹if	۱-۱-۱-NET. - و استانداردهای ۱۴CLI
۳-۲-۳-۳-ارزیابی کوتاه ۶۲	۲-۱-CLR ۱۵-
۴-۲-۳-۴-دستورات ۶۳if.. else	۱-۲-۱-کامپایل کردن کد. ۱۵NET
۵-۲-۳-۵-دستورات if تو در تو ۶۴	۲-۲-۱-۱۷-CTS
۶-۲-۳-۶-دستورات ۶۵switch	۳-۲-۱-اسمبلی ها ۱۸
۸-۲-۳-۸-دستورات switch روی رشته ها ۶۸	۳-۱-FCL ۲۲-
۳-۳-۳-دستورات تکرار ۶۹	۴-۱-کار با چارچوب NET و ۲۵SDK
۱-۳-۳-۱-ایجاد حلقه ها با ۶۹goto	۱-۴-۱-بروزآوری چارچوب. ۲۵NET
۳-۳-۳-۳-حلقه ۷۱do... while	۲-۴-۱-ابزار چارچوب. ۲۵NET
۴-۳-۳-۴-حلقه ۷۲for	۳-۴-۱-ابزار پیکربندی چارچوب ۲۹
۴-۳-۴-۴-خلاصه ۷۷	۵-۱-فهم کامپایلر ۲۱C#
۱-۴-۱-۱-ایجاد مدل ها ۸۰	۱-۵-۱-محل کامپایلر ۳۱
۲-۴-۲-۲-کلاس ها و اشیاء ۸۰	۲-۵-۱-کامپایل کردن از طریق خط فرمان ۳۱
۳-۴-۳-۳-تعریف یک کلاس: ۸۱	۶-۱-خلاصه ۳۳
۴-۴-۴-۴-روابط کلاس: ۸۲	۱-۲-چیدمان یک برنامه ۲۶C#
۵-۴-۵-۵-ارکان سه گانه ی برنامه نویسی شی گرا ۸۲	۱-۱-۲-تذکرات عمومی برنامه نویسی ۳۷C#
۱-۵-۴-۱-۱-کپسوله کردن ۸۳	۲-۲-انواع داده اولیه ۳۹
۲-۵-۴-۲-۲-تخصص ۸۳	۳-۲-عملگرهای ریاضی، منطق و شرطی ۴۲
۳-۵-۴-۳-۳-چندریختی ۸۴	۱-۳-۲-عملگرهای ریاضی ۴۲
۶-۴-۶-۶-تحلیل و طراحی شی گرا ۸۴	۲-۳-۲-عملگرهای شرطی و رابطه ای ۴۳
۷-۴-۷-۷-خلاصه ۸۴	۴-۲-راهنماهای پیش پردازش ۴۴C#
۱-۵-۱-۱-تعریف کلاس ۸۷	۱-۴-۲-کامپایل شرطی ۴۴
۱-۱-۵-۱-۱-نمونه سازی اشیاء ۸۸	۲-۴-۲-راهنماهای تشخیص ۴۵
۲-۱-۵-۲-۱-ایجاد کلاس ۸۹Time	۵-۲-نوع داده ی شمارشی ۴۵
۳-۱-۵-۳-۱-معرف های دسترسی ۹۰	۱-۵-۲-کار با نوع داده شمارشی ۴۶
۲-۵-۲-۲-آرگومان های متد ۹۱	۲-۵-۲-۲-متدهای ۴۷System.Enum
۳-۵-۳-۳-سازنده ها ۹۲	۳-۵-۲-۳-انواع شمارشی و flagهای بیتی ۴۷
۴-۵-۴-۴-مقدار دهنده های اولیه ۹۳	۶-۲-۶-۲-انواع داده ی مقداری و ارجاعی ۴۸
۵-۵-۵-۵-کلمه کلیدی ۹۴this	۶-۲-۶-۲-۱-System.Object و ۴۸System.ValueType
۶-۵-۶-۵-اعضای نمونه و ایستا ۹۵	۲-۶-۲-۲-تخصیص حافظه برای انواع داده مقداری و ارجاعی ۴۸
۱-۶-۵-۱-۶-احضار متدهای ایستا ۹۶	۳-۶-۲-۳-جعبه بندی ۴۹
۲-۶-۵-۲-۶-کاربرد فیلدهای ایستا ۹۸	۷-۲-۷-۲-فضاهای نامی ۵۰
۷-۵-۷-۵-خراب کردن اشیاء ۹۹	۱-۷-۲-۱-دستور ۵۱using
۸-۵-۸-۵-تخصیص حافظه ۱۰۱	۲-۷-۲-۲-اسامی مستعار فضای اسمی ۵۲
۹-۵-۹-۵-خلاصه ۱۰۴	۸-۲-۸-۲-کنسول ۵۳I/O
فصل ششم ۱۰۶	۹-۲-۹-۲-خلاصه ۵۴
وراثت و چند ریختی ۱۰۶	۱-۳-۱-۳-دستورات انشعاب غیرشرطی ۵۷
۱-۶-۱-۶-تخصص و تعمیم ۱۰۶	۲-۳-۲-۳-دستورات انشعاب شرطی ۵۸

- ۲-۶- وراثت ۱۰۸
- ۱-۲-۶- پیاده‌سازی وراثت ۱۰۹
- ۲-۲-۶- فراخوانی سازنده‌های کلاس پایه ۱۱۰
- ۳-۲-۶- کنترل دسترسی ۱۱۰
- ۳-۶- چند ریختی ۱۱۱
- ۱-۳-۶- ایجاد انواع داده‌ی چندریختی ۱۱۱
- ۲-۳-۶- نسخه‌سازی با new و override ۱۱۴
- ۴-۶- کلاس‌های انتزاعی ۱۱۵
- ۵-۶- کلاس‌های مهرشده ۱۱۷
- ۶-۶- ریشه‌ی همه کلاس‌ها () Object ۱۱۷
- ۷-۶- خلاصه ۱۱۹
- فصل هفتم ۱۲۰
- متدهای داخلی ۱۲۰
- ۷-overload- کردن متدها ۱۲۰
- ۲-۷- کپسوله کردن داده‌ها با خصوصیات ۱۲۲
- ۱-۲-۷- معاون ۱۲۴get
- ۲-۲-۷- معاون ۱۲۵set
- ۳-۷- برگرداندن چندین مقدار ۱۲۵
- ۱-۳-۷- ارسال انواع داده‌ی مقداری بوسیله ارجاع ۱۲۶
- ۲-۳-۷- پارامترهای out و انتساب روشن ۱۲۸
- ۴-۷- خلاصه ۱۲۹
- ۱-۱-۸- کاربرد آرایه‌ها ۱۳۰
- ۱-۱-۸- اعلان آرایه‌ها ۱۳۱
- ۲-۱-۸- فهم مقادیر پیش فرض ۱۳۱
- ۳-۱-۸- دسترسی به عناصر آرایه ۱۳۲
- ۲-۸- دستور foreach ۱۳۳
- ۳-۸- مقداردهی اولیه عناصر آرایه ۱۳۴
- ۴-۸- کلید کلیدی ۱۳۴params
- ۵-۸- آرایه‌های چندبعدی ۱۳۵
- ۱-۵-۸- آرایه‌های مستطیلی ۱۳۵
- ۲-۵-۸- آرایه‌های ناهموار ۱۳۷
- ۶-۸- متدهای آرایه ۱۴۰
- ۷-۸- مرتب‌کردن آرایه‌ها ۱۴۰
- ۸-۸- خلاصه ۱۴۱
- ۱-۹- تعریف ساختارها ۱۴۴
- ۲-۹- ایجاد ساختارها ۱۴۵
- ۱-۲-۹- ساختارها به صورت انواع داده‌ی مقداری ۱۴۶
- ۲-۲-۹- ایجاد ساختارها بدون new ۱۴۶
- ۳-۹- خلاصه ۱۴۸
- ۱-۱۰- کلاس ArrayList ۱۵۰
- ۲-۱۰- کلاس Queue ۱۵۱
- ۳-۱۰- کلاس Stack ۱۵۲
- ۴-۱۰- کلاس HashTable ۱۵۲
- ۵-۱۰- کلاس SortedList ۱۵۳
- ۶-۱۰- کلاس BitArray ۱۵۴
- ۷-۱۰- مقایسه‌ی آرایه‌ها و کلکسیون‌ها ۱۵۷
- ۸-۱۰- کاربرد کلاس‌های کلکسیون برای بازی کارت‌ها ۱۵۷
- ۹-۱۰- خلاصه ۱۵۷
- ۱-۱۱- تعریف کل ۱۵۸
- ۲-۱۱- پیاده‌سازی کل‌ها ۱۶۰
- ۳-۱۱- اعمال کردن کل‌ها ۱۶۱
- ۴-۱۱- انواع داده کل چندگانه ۱۶۱
- ۵-۱۱- محدودیت‌های کل ۱۶۳
- ۱-۵-۱۱- محدودیت‌های مشتق ۱۶۴
- ۲-۵-۱۱- محدودیت سازنده ۱۶۶
- ۳-۵-۱۱- محدودیت نوع مقداری / ارجاعی ۱۶۶
- ۶-۱۱- کلاس‌ها و کلکسیون‌های کل در FCL ۱۶۶
- ۱-۶-۱۱- مروری بر کلکسیون‌های کل ۱۶۷
- ۷-۱۱- خلاصه ۱۷۰
- ۱-۱۲- اندیس‌گذار ۱۷۲
- ۱-۱-۱۲- مثال بدون کاربرد اندیس‌گذار ۱۷۲
- ۲-۱-۱۲- کاربرد اندیس‌گذارها در مثال قبلی ۱۷۳
- ۲-۱۲- مقایسه آرایه‌ها و اندیس‌گذارها ۱۷۴
- ۳-۱۲- خصوصیات آرایه‌ها و اندیس‌گذارها ۱۷۵
- ۴-۱۲- اندیس‌گذارها در واسط‌ها ۱۷۶
- ۵-۱۲- خلاصه ۱۷۷
- ۱-۱۳- کاربرد کلمه‌ی کلیدی operator ۱۷۹
- ۲-۱۳- پشتیبانی دیگر زبان‌های . NET ۱۷۹
- ۳-۱۳- ایجاد عملگرهای مفید ۱۷۹
- ۴-۱۳- عملگرهای دوتایی منطقی ۱۸۰
- ۵-۱۳- عملگر تساوی ۱۸۰
- ۶-۱۳- عملگرهای تبدیل ۱۸۰
- ۷-۱۳- خلاصه ۱۸۴
- ۱-۱۴- برنامه‌نویسی یک فرم ویندوز ۱۸۶
- ۱-۱-۱۴- ایجاد دستی یک برنامه کاربردی ویندوز ۱۸۶
- ۲-۱۴- کلاس‌های کنترل در Windows.Forms ۱۸۸
- ۱-۲-۱۴- کلاس Control ۱۸۸
- خصوصیات Control ۱۸۹
- ۲-۲-۱۴- کار با کنترل‌ها ۱۹۰
- اندازه و موقعیت ۱۹۰
- چگونه یک کنترل را لنگر بیاندازیم و بچسبانیم ۱۹۱
- ترتیب Tab و کانون ۱۹۲
- طی‌کردن همه کنترل‌های روی یک فرم ۱۹۳
- ۳-۲-۱۴- رویدادهای Control ۱۹۴
- اداره‌کردن رویدادهای ماوس ۱۹۴
- اداره‌کردن رویدادهای صفحه کلید ۱۹۶

۱۹۸Form کلاس	۱۴-۳-۵-۲-کلاس Panel ۲۲۶
۱۹۹-۱-تنظیم ظاهر یک فرم	۱۵-۲-۶-کنترل FlowLayoutPanel ۲۲۶
کد ری فرم ۲۰۰	۱۵-۲-۷-کنترل TableLayoutPanel ۲۲۷
شفافیت فرم ۲۰۰	۱۵-۲-۸-کلاس Label ۲۲۸
تنظیم اندازه و موقعیت فرم ۲۰۲	۱۵-۳-کنترل‌های PictureBox و TextBox ۲۲۸
۱۴-۳-۲-نمایش فرم‌ها ۲۰۳	۱۵-۳-۱-کلاس PictureBox ۲۲۸
چرخه‌ی زندگی یک فرم ۲۰۳modeless	۱۵-۲-۲-کلاس TextBox ۲۳۰
ایجاد و نمایش فرم ۲۰۳	کادرهای متنی و کاراکترهای بازگشت به سر سطر ۲۳۱
۱۴-۳-۳-فعال سازی و غیرفعال سازی فرم ۲۰۴	۱۵-۴-کلاس‌های CheckedListBox، ListBox و
بستن فرم ۲۰۴	۲۳۲ComboBox
۱۴-۴-۳-فعل و انفعال فرم‌ها - یک برنامه کاربردی نمونه ۲۰۵	۱۵-۴-۱-کلاس ListBox ۲۳۲
کد فرم اصلی ۲۰۶	اضافه کردن اقلام به یک ۲۳۲ListBox
کد فرم جستجو ۲۰۶	انتخاب و جستجوی اقلام در یک ۲۳۳ListBox
۱۴-۳-۵-فرم‌های مالک و ملک ۲۰۷	سفارشی کردن ظاهر یک ۲۳۴ListBox
فرم‌های ۲۰۸MDI	۱۵-۴-۲-کنترل‌های دیگری از لیست: ComboBox و
ایجاد یک منو و فرم ۲۰۸MDI	۲۳۶CheckedListBox
ایجاد یک منوی MDI با استفاده از ۲۱۰VS.NET	۱۵-۵-کلاس‌های TreeView و ListView ۲۳۷
۱۴-۴-۱-کار با منوها ۲۱۱	۱۵-۵-۱-کلاس ListView ۲۳۷
۱۴-۴-۱-۱-خصوصیات ۲۱۱MenuItem	ایجاد یک شی ۲۳۷ListView
۱۴-۴-۲-منوهای زمینه ۲۱۱	تعریف ظاهر شی ۲۳۷ListView
۱۴-۴-۳-ساختن یک منوی زمینه ۲۱۲	تنظیم سرآیندهای ستون ۲۳۸
۱۴-۵-۱-اضافه کردن کمک به یک فرم ۲۱۳	ایجاد قلم داده‌های ۲۳۹ListView
۱۴-۵-۱-۵-ToolTip-ها ۲۱۳	تعیین آیکون‌ها ۲۳۹
۱۴-۵-۲-پاسخ به ۱۴ دکمه ۲۱۴Help	کار با کنترل ۲۴۰ListView
۱۴-۵-۳-کنترل ۲۱۶HelpProvider	طی کردن همه قلم داده‌ها یا قلم داده‌های انتخاب شده ۲۴۰
۱۴-۶-۱-وراثت فرم‌ها ۲۱۶	تشخیص قلم داده انتخاب شده جاری ۲۴۱
۱۴-۶-۱-۱-ایجاد و استفاده یک کتابخانه از فرم‌ها ۲۱۶	مرتب‌سازی قلم داده‌های یک کنترل ۲۴۱ListView
۱۴-۶-۲-کاربرد فرم ارث‌بری شده ۲۱۷	۱۵-۵-۲-کلاس TreeView ۲۴۲
۳۶-۶-۳-Override کردن رویدادها ۲۱۷	کلاس ۲۴۲TreeNode
۱۴-۶-۴-ایجاد فرم‌های ارث‌بری شده با ۲۱۸VS.NET	اضافه کردن و حذف کردن گره‌ها ۲۴۳
۱۴-۷-خلاصه ۲۱۸	طی کردن همه گره‌ها در یک ۲۴۴TreeView
۱۵-۱-مطالعه کنترل‌های NET فرم‌های ویندوز ۲۲۰	تشخیص گره انتخاب شده ۲۴۴
۱۵-۲-کلاس‌های Button، GroupBox، Panel و Label	یک مثال TreeView با کاربرد انعکاس ۲۴۴
۲۲۲	۱۵-۶-کلاس‌های Timer، ProgressBar و StatusStrip
۱۵-۲-۱-کلاس Button ۲۲۲	۲۴۶
تنظیم ظاهر یک دکمه ۲۲۲	ایجاد یک ۲۴۷StatusStrip
اداره کردن رویدادهای ۲۲۳Button	۱۵-۷-ایجاد کنترل‌های سفارشی ۲۴۸
۱۵-۲-۲-کلاس CheckBox ۲۲۳	۱۵-۷-۱-بسط یک کنترل ۲۴۹
تنظیم ظاهر ۲۲۳CheckBox	۱۵-۷-۲-ایجاد یک UserControl سفارشی ۲۴۹
۱۵-۳-۲-کلاس RadioButton ۲۲۴	یک مثال از ۲۴۹User Control
قراردادن دکمه‌های رادیویی در یک گروه ۲۲۴	استفاده از UserControl سفارشی ۲۵۰
۱۵-۴-۲-کلاس GroupBox ۲۲۶	کار با UserControl در زمان طراحی ۲۵۱

۱۵-۸- استفاده از کشیدن و انداختن بوسیله کنترل‌ها ۲۵۲

مروری بر کشیدن و انداختن ۲۵۲

مسئولیت‌های کنترل مبدأ ۲۵۴

مسئولیت‌های کنترل هدف ۲۵۴

۱۵-۹- کاربرد منابع ۲۵۶

۱۵-۹-۱- کار با فایل‌های منبع ۲۵۷

ایجاد رشته‌های منبع از روی یک فایل متنی ۲۵۷

کاربرد کلاس ResourceWriter برای ایجاد یک فایل

. ۲۵۸Resources

کاربرد کلاس ResourceManager برای دستیابی به منابع

۲۵۸

کاربرد کلاس ResXResourceWriter برای ایجاد یک فایل

. ۲۵۹resx

کاربرد کلاس ResXResourceReader برای خواندن یک

فایل . ۲۵۹resx

تبدیل یک فایل .resx به یک فایل . ۲۵۹resources

۱۵-۹-۲-VS.NET و منابع ۲۵۹

کاربرد فایل‌های منبع برای ایجاد فرم‌های محلی ۲۶۰

محلی کردن منابع با استفاده از VS.NET ۲۶۰

تعیین منابع محلی در زمان اجرا ۲۶۱

ایجاد یک اسمبلی پیرو بدون VS.NET ۲۶۱

۱۵-۱۰- خلاصه ۲۶۱

۱۶-۱- کادر محاوره‌ای ۲۶۳MessageBox

دکمه‌های موجود برای کادر پیغام: ۲۶۴

تنظیم دکمه‌ی پیش فرض: ۲۶۵

گزینه‌های مختلف کادر پیغام ۲۶۵

حالت‌های مختلف استفاده از متد Show ۲۶۵

نمونه‌هایی از کادر پیغام ۲۶۷

۱۶-۲- کنترل OpenFileDialog ۲۶۸

خصوصیت‌های کنترل OpenFileDialog ۲۶۹

متدهای OpenFileDialog ۲۷۱

استفاده از کنترل OpenFileDialog ۲۷۱

بررسی نکات مهم برنامه ۲۷۳

۱۶-۳- کنترل SaveFileDialog ۲۷۴

خصوصیت‌های کنترل SaveFileDialog ۲۷۴

متدهای کنترل SaveFileDialog ۲۷۵

استفاده از کنترل SaveFileDialog ۲۷۵

۱۶-۴- کنترل FontDialog ۲۷۷

خصوصیت‌های کنترل FontDialog ۲۷۷

متدهای کنترل FontDialog ۲۷۸

استفاده از کنترل FontDialog ۲۷۸

۱۶-۵- کنترل ColorDialog ۲۸۰

خصوصیت‌های کنترل ColorDialog ۲۸۱

استفاده از کنترل ColorDialog ۲۸۲

۱۶-۶- کنترل PrintDialog ۲۸۳

خصوصیت‌های کنترل PrintDialog ۲۸۴

استفاده از کنترل PrintDialog ۲۸۴

۱۶-۶-۱- کلاس PrintDocument ۲۸۴

خصوصیات کلاس PrintDocument ۲۸۴

چاپ یک سند ۲۸۵

بررسی مثال چاپ ۲۸۸

۱۶-۷- کنترل FolderBrowserDialog ۲۹۱

خصوصیت‌های کنترل FolderBrowser ۲۹۲

استفاده از کنترل FolderBrowser ۲۹۲

۱۶-۸- خلاصه ۲۹۴

۱۷-۱- تعریف و پیاده‌سازی یک واسط ۲۹۷

۱۷-۱-۱- پیاده‌سازی بیش از یک واسط ۲۹۹

۱۷-۱-۲- بسط دادن واسط‌ها ۲۹۹

۱۷-۲- دستیابی به متدهای واسط ۳۰۳

۱۷-۲-۱- قالب‌بندی به یک واسط ۳۰۳

۱۷-۲-۲- عملگر Fis ۳۰۴

۱۷-۲-۳- عملگر as ۳۰۶

۱۷-۲-۴- مقایسه عملگرهای as و Vis ۳۰۷

۱۷-۲-۵- مقایسه کلاس انتزاعی و واسط ۳۰۷

۱۷-۳- override- کردن پیاده‌سازی‌های واسط ۳۰۷

۱۷-۴- پیاده‌سازی صریح واسط ۳۱۰

۱۷-۴-۲- پنهان کردن اعضا ۳۱۲

۱۷-۴-۳- دستیابی به کلاس‌های مهرشده و انواع داده‌ی

مقداری ۳۱۲

۱۷-۵- خلاصه ۳۱۴

۱۸-۱- اعلان و کاربرد نماینده‌ها ۳۱۶

۱۸-۱-۱- سناریوی کارخانه اتوماتیک ۳۱۷

۱۸-۱-۲- پیاده‌سازی کارخانه بدون کاربرد نماینده‌ها ۳۱۷

۱۸-۱-۳- پیاده‌سازی کارخانه با استفاده یک نماینده ۳۱۷

۱۸-۱-۴- متدها و نماینده‌های بی‌نام ۳۱۹

۱۸-۲- اعلان یک رویداد ۳۲۱

۱۸-۲-۱- متعهد شدن به یک رویداد ۳۲۲

۱۸-۲-۲- غیر متعهد شدن از یک رویداد ۳۲۲

۱۸-۲-۳- رها کردن یک رویداد ۳۲۲

۱۸-۳- رویدادهای GUI ۳۲۳

۱۹-۱- مقدمه ۳۲۵

۱۹-۲- کلاس System.Exception ۳۲۶

۱۹-۳- کدنویسی برای اداره کردن استثناها ۳۲۷

۱۹-۴- چگونه یک کلاس استثناء سفارشی ایجاد کنیم؟ ۳۲۹

۱۹-۵- استثناء‌های اداره نشده ۳۳۲

۲۰-۱- کاراکترها و یونیکد ۳۳۳

- ۲۰-۱-۱-۱ یونیکد ۳۳۴
- ۲۰-۱-۲-۱ کار با کاراکترها ۳۳۵
- انتساب یک مقدار به یک نوع داده‌ی char ۳۳۵
- تبدیل یک مقدار Char به یک مقدار عددی ۳۳۵
- ۲۰-۱-۲-۲ کاراکترها و محلی کردن ۳۳۵
- ۲۰-۱-۲-۳ کاراکترها و دسته‌های یونیکد آنها ۳۳۶
- ۲۰-۲-۱ کلاس رشته ۳۳۷
- ۲۰-۱-۲-۱-۱ ایجاد رشته‌ها ۳۳۷
- ۲۰-۲-۲-۱ داخل کردن رشته‌ها ۳۳۸
- ۲۰-۲-۲-۲ مروری بر عملیات رشته‌ها ۳۳۹
- ۲۰-۳-۱ مقایسه‌ی رشته‌ها ۳۳۹
- ۲۰-۳-۲-۱ کاربرد String.Compare ۳۴۰
- ۲۰-۳-۲-۲ کاربرد String.CompareOrdinal ۳۴۱
- ۲۰-۴-۱ جستجو، تغییر و کدگذاری محتوای یک رشته ۳۴۲
- ۲۰-۴-۱-۱ جستجوی محتویات یک رشته ۳۴۲
- ۲۰-۴-۲-۱ جستجوی رشته‌ی جانشین‌دار ۳۴۳
- ۲۰-۴-۲-۲ تبدیل رشته‌ها ۳۴۳
- ۲۰-۴-۲-۳ کدگذاری رشته ۳۴۵
- ۲۰-۵-۱ کلاس StringBuilder ۳۴۶
- ۲۰-۵-۲ مقایسه‌ی StringBuilder و الحاق رشته ۳۴۷
- ۲۰-۶-۱ فرمت‌دهی مقادیر عددی، تاریخ و زمان ۳۴۸
- ۲۰-۶-۱-۱ ساختن یک عنصر فرمت ۳۴۸
- ۲۰-۶-۲-۱ فرمت‌دهی مقادیر عددی ۳۴۹
- ۲۰-۶-۲-۲ فرمت‌دهی تاریخ و زمان ۳۵۰
- ۲۰-۶-۲-۳ تاریخ‌ها و فرهنگ ۳۵۲
- کلاس‌های DateTimeFormatInfo و
- ۳۵۲ NumberFormatInfo
- ۲۰-۷-۱ عبارات منظم ۳۵۳
- ۲۰-۷-۱-۱ کلاس Regex ۳۵۴
- ۲۰-۷-۲-۱ ایجاد عبارات منظم ۳۵۷
- ۲۰-۷-۳-۱ مثال‌هایی از کاربرد عبارات منظم ۳۶۱
- ۲۰-۸-۱ خلاصه ۳۶۲
- ۲۱-۱-۱ مدیریت سیستم فایل ۳۶۳
- ۲۱-۱-۲-۱ کلاس‌های مربوط به پوشه‌ها و فایل‌ها در .NET ۳۶۴
- ۲۱-۲-۱-۱ کلاس Path ۳۶۶
- ۲۱-۳-۱-۱ مثال File Browser ۳۶۶
- ۲۱-۲-۲-۱ انتقال، کپی و حذف فایل‌ها ۳۷۰
- ۲۱-۱-۲-۱ مثال FilePropertiesAndMovement ۳۷۰
- ۲۱-۲-۲-۱ بررسی کد برنامه FilePropertiesAndMovement ۳۷۱
- ۲۱-۳-۱ خواندن و نوشتن در فایل‌ها ۳۷۴
- ۲۱-۳-۲-۱ خواندن یک فایل ۳۷۴
- ۲۱-۳-۲-۲ نوشتن به یک فایل ۳۷۵
- ۲۱-۳-۲-۳ جریان‌های بافر شده ۳۷۸
- ۲۱-۳-۲-۴ خواندن و نوشتن در فایل‌های دودویی ۳۷۸
- کلاس FileStream ۳۷۸
- ۲۱-۳-۲-۵ خواندن و نوشتن در فایل‌های متنی ۳۸۰
- کلاس StreamReader ۳۸۰
- متدهای کلاس StreamReader ۳۸۰
- کلاس StreamWriter ۳۸۱
- متدهای کلاس StreamWriter ۳۸۱
- ۲۱-۳-۲-۷ رمزنگاری با کلاس CryptoStream ۳۸۱
- ۲۱-۴-۱ خواندن اطلاعات درایو ۳۸۱
- ۲۱-۵-۱ امنیت فایل ۳۸۳
- ۲۱-۵-۱-۱ خواندن ACLهای یک فایل ۳۸۳
- ۲۱-۵-۲-۱ اضافه کردن و حذف ACLهای یک فایل ۳۸۴
- ۲۱-۶-۱ خلاصه ۳۸۵
- ۲۱-۲-۱ مقدمه ۳۸۷
- اشیای موجود در Access ۳۸۸
- ۲۱-۳-۱ مقید کردن داده‌ها ۳۹۴
- خلاصه ۴۰۰
- ۲۱-۲-۱ ADO.NET ۴۰۲
- ۲۱-۲-۱-۱ فضای نامی Data ۴۰۳
- ۲۱-۲-۲-۱ کلاس SqlConnection ۴۰۴
- ایجاد بخش‌های مختلف Fring ConnectionSt ۴۰۴
- ۲۱-۲-۳-۱ کلاس SqlCommand ۴۰۶
- خاصیت Connection ۴۰۶
- خاصیت CommandText ۴۰۶
- خاصیت Parameters ۴۰۷
- متد ExecuteNonQuery ۴۰۸
- ۲۱-۲-۳-۲ کلاس SqlDataAdapter ۴۰۹
- خاصیت SelectCommand ۴۰۹
- تنظیم خاصیت SelectCommand با استفاده از دستور SQL ۴۱۰
- تنظیم خاصیت SelectCommand با استفاده از پروسیجر ذخیره شده ۴۱۱
- استفاده از CommandBuilder برای ایجاد دستورات SQL دیگر ۴۱۱
- متد Fill ۴۱۲
- ۲۱-۲-۳-۳ کلاس DataSet ۴۱۳
- ۲۱-۲-۳-۴ کلاس DataView ۴۱۴
- خاصیت Sort ۴۱۵
- خاصیت RowFilter ۴۱۵

۴۸۴-۳-۱-۲۶-مقدمه‌ای بر پورت‌ها	متد Find ۴۱۶
۴۸۵System.Net-۴-۱-۲۶-فضای نامی	۴۱۷-۳-۲۳-استفاده از کلاس‌های ADO.NET در عمل
۴۸۵System.Net.Sockets-۵-۱-۲۶-فضای نامی	۴۱۷-۳-۲۳-کاربرد DataSet در برنامه
۴۸۵TCP-۲-۲۶-مثال انتقال و پردازش دستورات در	۴۲۲-۴-۲۳-اتصال داده‌ها
۴۸۷Net.-۱-۲-۲۶-کاربرد کلاس‌های معمول	۴۲۲-CurrencyManager, BindingContext: ۱-۴-۲۳
۴۸۸-۲-۲۶-سرویس دهنده	۴۲۳-۲-۴-۲۳-اتصال کنترل‌ها
۴۸۹-۳-۲۶-سرویس گیرنده	۴۲۷-۵-۲۳-خلاصه
۴۹۲-۴-۲۶-کامپایل کردن و اجرای برنامه	۴۴۰-۱-۲۴-ریسمان‌ها
۴۹۲UDP-۳-۲۶-مثال انتقال و پردازش دستور با	۴۴۰-۱-۲۴-شروع ریسمان‌ها
۴۹۲NET-۱-۳-۲۶-کاربرد کلی کلاس‌های ضروری	۴۴۲-۲-۱-۲۰-پیوندزدن ریسمان‌ها
۴۹۳-۲-۳-۲۶-سرور	۴۴۳Sleep-۳-۱-۲۰-بلوکه کردن ریسمان‌ها با
۴۹۴-۳-۳-۲۶-سرویس گیرنده	۴۴۴-۴-۱-۲۴-از بین بردن ریسمان‌ها
۴۹۵-۴-۳-۲۶-کامپایل کردن و اجرای مثال	۴۴۷-۲-۲۴-همگام سازی
۴۹۶UDP-۴-۲۶-ایجاد یک تلگراف اخبار بوسیله چندپخشی	۴۴۹Interlocked-۱-۲-۲۰-کاربرد
۴۹۶NET-۱-۴-۲۶-کاربرد کلی کلاس‌های مورد نیاز	۴۵۱-۲-۲۴-کاربرد قفل‌ها
۴۹۸-۲-۴-۲۶-سرور	۴۵۱-۳-۲۰-کاربرد مانیتورها
۵۰۰-۳-۴-۲۶-سرویس گیرنده	۴۵۶-۳-۲۰-خلاصه
۵۰۳-۴-۴-۲۶-کامپایل کردن و اجرای مثال	۴۵۸WebClient-۱-۲۵-کلاس
فصل بیست و هفتم ۵۰۴	۴۵۸-۱-۲۵-گرفتن فایل‌ها
۵۰۴NET-۱-۲۷-کلاس Socket چارچوب	۴۵۸WebClient-۲-۱-۲۵-مثالی از
۵۰۴Socket-۱-۱-۲۷-سرویس گیرنده TCP با کلاس	۴۶۰WebResponse و WebRequest-۴-۱-۲۵-کلاس‌های
۵۰۴Socket-کلاس	۴۶۰WebResponse و WebRequest-۵-۱-۲۵-ویژگی‌های دیگر
۵۰۵-سازنده	۴۶۱
۵۰۵-متدها	۴۶۱-۶-۲۵-تقاضاهای ناهمگام
۵۰۷-خصوصیات	۴۶۲-۲-۲۵-نمایش خروجی بصورت یک صفحه HTML
۵۰۷SocketOptionLevel-کلاس شمارشی	۴۶۳-۱-۲۵-کاوش کردن ساده وب از طریق برنامه کاربردی
۵۰۷SocketOptionName-کلاس شمارشی	۴۶۵IE-۲-۲۵-شروع نمونه‌های
۵۰۸SocketFlags-کلاس شمارشی	۴۶۵-۳-۲۵-اعمال کردن بیشتر ویژگی‌های IE روی برنامه
۵۰۹SocketException-کلاس شمارشی	۴۶۵-کاربردی
۵۱۰TCP-مثال برنامه سرویس گیرنده	۴۶۵-۴-۲۵-نمایش مستندات با استفاده از کنترل
۵۱۰TcpEchoClientSockets.cs-کد برنامه‌ی	۴۷۱WebBrowser
۵۱۰Socket-۲-۱-۲۷-سرویس دهنده با کلاس	۴۷۲WebBrowser-۵-۲۵-چاپ کردن بوسیله کنترل
۵۱۱TCP-مثال برنامه سرویس دهنده	۴۷۳-۶-۲۵-نمایش کد یک صفحه درخواست شده
۵۱۲-۳-۱-۲۷-گزینه‌های سوکت	۴۷۴Web-۳-۲۵-سلسله مراتب کلاس‌های
۵۱۳UDP-مثال سرویس گیرنده	۴۷۵-۱-۳-۲۵-کلاس‌های سودمند
۵۱۴-۴-۱-۲۷-پرچم‌های سوکت	۴۷۵-URI‌ها
۵۱۵-۱-۲۷-I/O-بدون وقفه	۴۷۵DNS-۲-۳-۲۵-آدرس‌های IP و اسامی
۵۱۶I/O-۶-۱-۲۷-بررسی وضعیت	۴۷۶IP-۳-۳-۲۵-کلاس‌های NET برای آدرس‌های
۵۱۷-۷-۱-۲۷-فراخوانی‌های مسدود کننده با مهلت زمانی	۴۷۷DNSLookup-۴-۳-۲۵-مثال
۵۱۹-مثال سرویس دهنده Echo با مهلت زمانی معین	۴۷۸-۴-۲۵-خلاصه
۵۲۰-۸-۱-۲۷-تسهیم سازی	۴۷۹-۲-۲۶-مقدمه
۵۲۰Socket-متد Select() کلاس	۴۸۱TCP-۱-۱-۲۶-مقدمه‌ای بر
۵۲۱-مثال سرویس دهنده چند پورته	۴۸۲UDP-۲-۱-۲۶-مقدمه‌ای بر

۳۰-۲-۱- ایجاد یک اسمبلی چند ماژولی ۵۶۶

۳۰-۲-۲- آزمایش اسمبلی ۵۷۰

۳۰-۲-۳- اسمبلی‌های خصوصی ۵۷۲

۳۰-۲-۴- اسمبلی‌های اشتراکی ۵۷۲

۳۰-۲-۵- پایان جهنم ۵۷۳DLL

۳۰-۳- نسخه‌ها ۵۷۳

۳۰-۳-۱- اسامی قوی ۵۷۳

۳۰-۳-۲- GAC- ۵۷۴

۳۰-۳-۳- ایجاد یک اسمبلی اشتراکی ۵۷۴

۳۰-۳-۴- اسمبلی‌های مورد نیاز دیگر ۵۷۶

ضمیمه ۱ ۵۷۸

نصب ویزوال #C ۵۷۸ ۲۰۰۵

محیط توسعه‌ی #VC ۵۸۰ ۲۰۰۵

۲۷-۱-۰۹-۱- ناهمگام ۵۲۲

مثال سرویس گیرنده‌ی ناهمگام ۵۲۵

مثال سرویس دهنده‌ی Tcp ناهمگام ۵۲۷

۲۸-۱- مقدمه ۵۲۹

۲۸-۱-۱- چه زمانی صف‌بندی پیام را به کار ببریم؟ ۵۳۰

۲۸-۱-۲- ویژگی‌های صف‌بندی پیام ۵۳۱

۲۸-۱-۳- محصولات صف بندی پیام ۵۳۱

۲۸-۲- معماری ۵۳۲MQ

۲۸-۲-۱- پیام‌ها ۵۳۲

۲۸-۲-۲- صف پیام ۵۳۳

ایجاد صف‌های پیام ۵۳۴

خصوصیات صف پیام ۵۳۴

۲۸-۳- برنامه‌نویسی صف‌بندی پیام ۵۳۵

۲۸-۳-۱- ایجاد یک صف پیام ۵۳۶

۲۸-۳-۲- برنامه کاربردی CourseOrder ۵۳۶

کتابخانه‌ی کلاس CourseOrder ۵۳۶

ارسال کننده‌ی پیام تکلیف درس ۵۳۸

ارسال پیام‌های قابل ترمیم و اولویت‌دار ۵۳۹

دریافت کننده‌ی پیام تکلیف درس ۵۴۰

صف‌های تصدیق ۵۴۳

صف‌های جواب ۵۴۳

صف‌های تراکنشی ۵۴۴

۲۸-۴- نصب MessageQueue ۵۴۵

۲۸-۵- خلاصه ۵۴۵

۲۹-۱- وارد کردن کنترل‌های ActiveX ۵۴۶

۲۹-۲- وارد کردن یک کنترل به .NET ۵۴۹

۲۹-۲-۱- وارد کردن یک کنترل ۵۴۹

۲۹-۲-۲- وارد کردن کنترل به صورت دستی ۵۵۰

۲۹-۲-۳- اضافه کردن کنترل به فرم ۵۵۱

۲۹-۳- وارد کردن قطعات Com ۵۵۲

۲۹-۳-۱- کدنویسی برنامه‌ی ComTestForm ۵۵۳

۲۹-۳-۲- وارد کردن COM DLL به .NET ۵۵۴

۲۹-۳-۳- وارد کردن کتابخانه نوع داده ۵۵۴

۲۲-۳-۵- ایجاد یک برنامه آزمایشی ۵۵۴

۲۹-۴- صادر کردن قطعات .NET ۵۵۸

۲۲-۴-۱- ایجاد یک کتابخانه‌ی نوع داده ۵۶۰

۲۹-۵- P/Invoke ۵۶۱

۳۰-۱- فایل‌های PPE ۵۶۴

۳۰-۱-۱- فراداده ۵۶۴

۳۰-۱-۲- محدوده‌های امنیت ۵۶۵

۳۰-۱-۳- اظهارنامه‌ها ۵۶۵

۳۰-۲- اسمبلی‌های چند ماژولی ۵۶۶

فصل یک

مقدمه‌ای بر NET و C#

در این فصل یاد خواهید گرفت:

- مروری بر چارچوب NET: معماری و ویژگی‌ها
 - CLR: مروری بر کارهای قابل انجام توسط قسمت زمان اجرای چارچوب.
 - کامپایلر NET^۲ JIT. (فقط در لحظه بارگذاری اسمبلی‌ها و تایید کد لازم است).
 - CTS^۴ و CLS^۳: قوانینی که سازگاری CLR و تعامل بین زبانی را مدیریت می‌کنند.
 - اسمبلی‌ها: نگاهی کوتاه به ساختار یک اسمبلی، فلسفه‌ی پشت آن و تفاوت مابین اسمبلی‌های خصوصی و اشتراکی.
 - FCL^۵: این کتابخانه صدها کلاس پایه‌ی گروه‌بندی شده در فضاها‌ی نامی منطقی را فراهم می‌کند.
 - ابزار توسعه: بوسیله‌ی NET چندین ابزار با هدف توسعه‌ی کد فراهم شده است. این ابزار شامل ILdasm برای نمایش کد، WinCV برای مشاهده خصوصیات یک کلاس و ابزار دیگر پیکربندی چارچوب.
 - کامپایل کردن و اجرای برنامه‌های C#: کاربرد کامپایلر C# از خط فرمان و گزینه‌هایی برای پیکربندی یک برنامه.
- استفاده‌ی کارای یک زبان، یادگیری گرامر و ویژگی‌های زبان را نیاز دارد. در حقیقت بخش اعظم منحنی یادگیری برای تکنولوژی جدید، به محیط برنامه‌نویسی مرتبط است. حرفه‌ای شدن در C# کافی نیست. معمار نرم‌افزار و توسعه دهنده‌ی موفق باید کتابخانه‌های اصیل کلاس و ابزار تولید آنها را بشناسد.
- از منظر برنامه‌نویسی NET Platform شامل یک محیط اجرایی پیوند خورده به یک کتابخانه کلاس پایه است، که در این فصل CLR، FCL و طرز کار با NET Platform بررسی خواهد شد.

۱-۱ مروری بر ساختار NET

ساختار NET بصورت یک محیط مجتمع برای توسعه و اجرای برنامه‌های اینترنتی، برنامه‌های کاربردی ویندوز (میزکار) و حتی دستگاه‌های موبایل طراحی شده است. اهداف اصلی آن بصورت زیر است:

- فراهم ساختن یک محیط شی‌گرایی مابین دامنه‌ای از کاربردها.
- با فراهم ساختن این محیط، تداخل نسخه‌های DLL را کم کرده و پروسه توزیع و نصب کد را ساده می‌کند.

^۱ Common Language RunTime

^۲ Just In Time

^۳ Common Language Specification

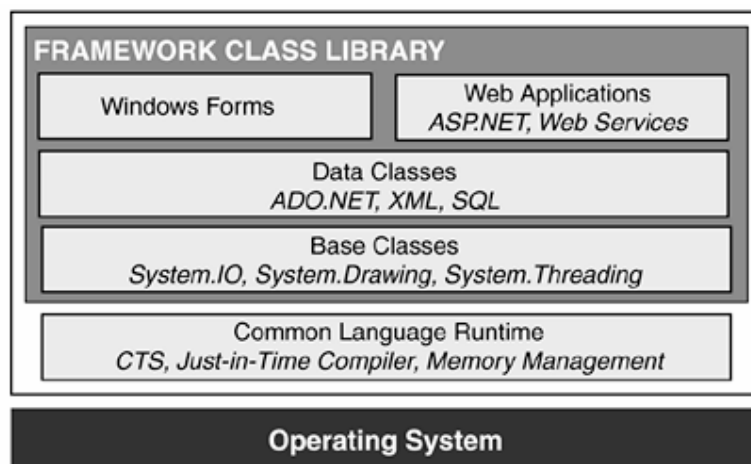
^۴ Common Type System

^۵ Framework Common Library

- = یک محیط قابل حمل^۱ براساس استانداردهای تایید شده آماده می سازد تا بتوانند توسط هر سیستم عاملی میزبانی شوند. در حال حاضر C# و یک بخش اصلی از زمان اجرای NET به نام CLI^۲ بوسیله ECMA^۳ استاندارد شده اند.
- فراهم ساختن یک محیط مدیریت شده، که اجرای امن را به سادگی تایید می کند.

طراحان چارچوب NET برای رسیدن به این اهداف بزرگ روی یک معماری به توافق رسیدند که چارچوب را به دو بخش تجزیه می کند: CLR و FCL. شکل ۱-۱ آن را ارائه می کند.

شکل ۱-۱ معماری NET Framework.



CLR (پیاده سازی استاندارد CLI توسط میکروسافت) اجرای کد و همه کارهای تخصیص یافته به آن همچون کامپایل، مدیریت حافظه، امنیت، مدیریت ریسمان و ایمنی از نوع داده^۴، کد اجرا شده تحت نظر CLR (که کد مدیریت شده می نامند)، کد مدیریت نشده توسط CLR همچون COM و API را اداره می کند.

FCL قطعه اصلی دیگر می باشد. یک کتابخانه کد قابل استفاده مجدد (شامل کلاس ها، ساختارها و غیره)، که برای برنامه های اجرا شده تحت NET در دسترس است. همه زبان های NET این کتابخانه کلاس مشترک را استفاده می کنند. پس این مفاهیم در همه زبان های NET مشترک خواهد بود.

۱-۱-۱ NET و استانداردهای CLI

یک توسعه دهنده قبل از صرف زمان برای یادگیری NET و C# می پرسد: آیا این مهارت را می تواند به Platform های دیگر تبدیل کند؟ آیا محصول NET میکروسافت فقط مختص سیستم عامل ویندوز است؟ یا آیا آن قابل حمل اجرایی است و آن برای حمل روی سیستم عامل های دیگر نیز پیاده سازی شده است؟ برای جواب دادن به این سئوال، فهمیدن رابطه ی مابین NET، C# و استانداردهای CLI ضروری است.

CLI یک محیط اجرایی مجازی مستقل از Platform را تعریف می کند. آن هیچ سیستم عاملی را تعیین نمی کند و برای لینوکس همانند ویندوز راحت است. بخش استاندارد مرکزی، تعریف یک CIL^۵ است که باید توسط کامپایلرهای مطیع CLI

^۱ Portable

^۲ Common Language Infrastructure

^۳ European computer Manufactures Association

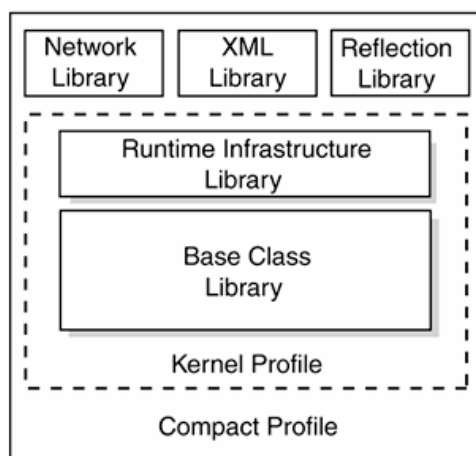
^۴ Type Safety

^۵ Common Intermediate Language

تولید شود. بخش دیگر سیستم نوع^۱ است که همه انواع داده‌ای پشتیبان شده توسط هر زبان مطیع CLI را تعریف می‌کند. این کد میانی به زبان اصلی سیستم عامل میزبان کامپایل می‌شود.

CLI استانداردهایی را برای زبان #C در بر دارد که بوسیله مایکروسافت توسعه و ارتقاء یافته‌اند. (Fortran, Pascal, Pythen, Defacto)

چارچوب NET ارائه شده در شکل ۱-۱، پیاده‌سازی مایکروسافت از استانداردهای CLI است. این پیاده‌سازی ویژگی‌های زیادی دارد که بوسیله معماری CLI مشخص می‌شوند. برای فهم بیشتر، آن را با معماری استاندارد های CLI شکل ۱-۲ مقایسه کنید.



شکل ۱-۲ معماری تعریف شده بوسیله مشخصه CLI

این کتاب پیاده‌سازی مایکروسافت را تشریح می‌کند. فرض بر آن است که می‌توانیم از کدهای قبلی نوشته شده نیز استفاده کنیم و کدهای پیاده‌سازی شده توسط NET به سیستم عامل دیگری منتقل نخواهد شد و محیط مجازی گفته شده نیز شفاف است.

۱-۲- CLR

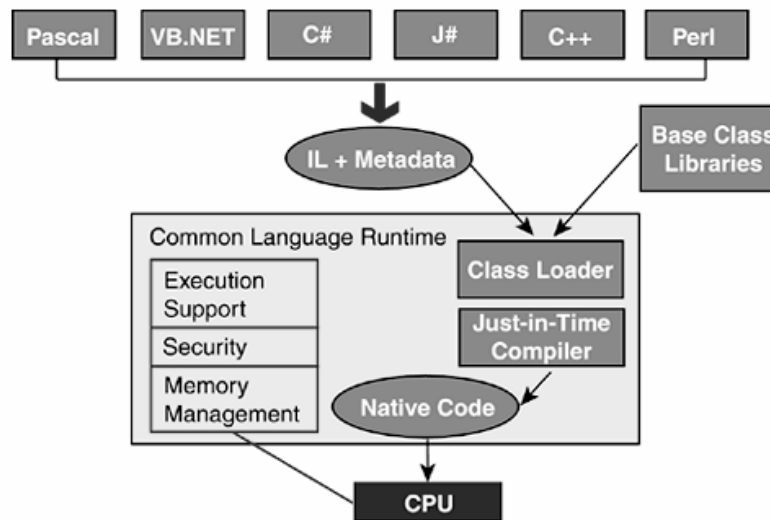
CLR تمام چرخه‌ی زندگی یک برنامه کاربردی را مدیریت می‌کند. آن کد را یافته و کامپایل می‌کند. کلاس‌های تخصیص یافته را بارگذاری می‌کند. اجرایش را مدیریت می‌کند و مدیریت اتوماتیک حافظه را مطمئن می‌سازد. آن ارتباط بین زبانی را پشتیبانی می‌کند، تا تعامل مابین کدهای نوشته شده در زبان‌های مختلف را مجاز دارد. این بخش کارکرد داخلی CLR را نشان می‌دهد. آن یک بحث عمقی نیست و فقط می‌خواهد شما را با اصطلاح آن آشنا سازد.

۱-۲-۱- کامپایل کردن NET.

کامپایلرهای مطیع CLR کدی تولید می‌کنند که کد هدف زمان اجرا بوده و برای یک CPU خاص پیشنهاد شده است. این کد به نام CIL است. IL یا MSIL یک زبان از نوع اسمبلی است که در یک فایل DLL یا EXE بسته‌بندی می‌شود. توجه کنید که اسمبلی‌ها فایل‌هایی با استاندارد اجرایی نیستند. لازم است یک کامپایلر زمان اجرا به نام JIT، IL را به یک کد ماشین خاص تبدیل کند. (زمانی که برنامه واقعاً اجرا می‌شود). چون CLR مسئول مدیریت این IL است، این کد میانی یکی از کلیدهای رویارویی با اهداف اسمی چارچوب NET از نظر سازگاری زبان است. همانطور که شکل ۱-۳ نشان می‌دهد، CLR

^۱ Type System

نمی‌داند چه زبانی این برنامه را ایجاد کرده است. تعامل آن با IL مستقل از زبان می‌باشد. چون برنامه‌ها از طریق IL با هم ارتباط برقرار می‌کنند، پس خروجی یک کامپایلر می‌تواند با خروجی کامپایلر متفاوت دیگر مجتمع شود.



شکل ۳-۱ عملکرد CLR

هدف دیگر .NET، قابلیت حمل Platform است، که با محلی کردن^۱ ایجاد کد ماشین در کامپایلر JIT فراهم می‌شود. بدین معنی که IL تولید شده روی یک Platform می‌تواند روی Platform دیگری که چارچوب خاص خودش و یک کامپایلر JIT با کد ماشین خاص خودش را دارد، اجرا شود.

کامپایلرهایی که کد هدف آنها CLR است، برای هر ماژول علاوه بر تولید IL، باید فراداده^۲ای را نیز صادر کنند. در فراداده‌ها مجموعه‌ای از جداول قرار می‌گیرند تا هر ماژول، کد خود - توصیف^۳ داشته باشد. در جداول علاوه بر توصیف کامل کد، اطلاعاتی درباره اسمبلی‌ها نیز وجود دارد. این اطلاعات شامل موارد دیگر نیز هستند: چه نوع داده‌هایی در دسترس هستند؟ نام هر نوع داده، اعضای نوع داده، دامنه یا میدان دید نوع داده و ویژگی‌های هر نوع داده دیگر. فراداده‌های دیگری که کاربردهای زیادی دارند:

- مهمترین کاربرد آن بوسیله کامپایلر JIT است، که اطلاعات همه‌ی نوع داده‌های مورد نیاز برای کامپایل کردن را مستقیماً از فراکد^۴ جمع‌آوری می‌کند. این اطلاعات را برای بررسی کد بکار می‌برد تا مطمئن شود برنامه عملیات را به درستی انجام می‌دهد. برای مثال، JIT از طریق مقایسه‌ی پارامترهای متد، فراخوانی صحیح را مطمئن می‌سازد.
- فراداده‌ها در پروسه جمع‌آوری زباله^۵ استفاده می‌شوند. جمع‌کننده زباله، برای شناسایی فیلدها و ارجاعات آنها از فراداده استفاده می‌کند و می‌تواند تعیین کند حافظه‌ی چه اشیایی می‌توانند آزاد شوند یا نه؟
- NET یک مجموعه از کلاس‌ها برای خواندن فراداده‌های یک برنامه فراهم می‌کند. این توانایی به نام انعکاس^۶ شناخته می‌شود، که یک ویژگی قدرتمند است و اجازه می‌دهد یک برنامه در زمان اجرا، کد را مورد جستجو قرار دهد و براساس اطلاعات یافته شده تصمیم‌گیری کند. می‌توان صفات سفارشی را به فراداده اضافه کرد.

^۱ Localizing

^۲ Metadata

^۳ Self descriptive

^۴ Meta code

^۵ Garbage Collection

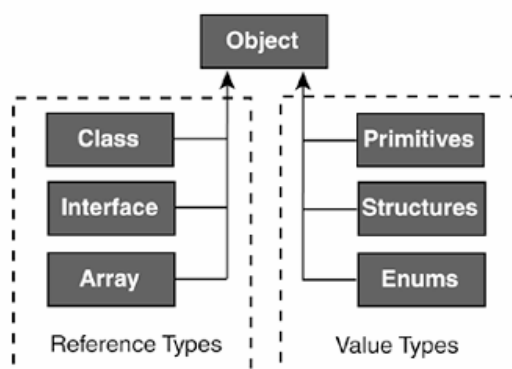
^۶ Reflection

IL و فراداده برای فراهم ساختن ارتباط بین زبانی بسیار مهم هستند. اما دنیای واقعی به همه کامپایلرهای .NET که یک مجموعه‌ی مشترک از انواع داده‌ای و توصیف زبان را پشتیبانی می‌کنند، منوط است. برای مثال، دو زبان در IL سازگار نیستند، اگر یکی عدد صحیح علامت‌دار ۳۲ بیتی را پشتیبانی کند و دیگری آن را پشتیبانی نکند. آنها ممکن است گرامر متفاوتی داشته باشند، اما باید روی انواع داده‌ای پایه که پشتیبانی می‌کنند، توافق داشته باشند.

همانطور که قبلاً بحث شده، CLI یک توصیف رسمی به نام CTS تعریف می‌کند که بخش مکمل CLR است. آن شرح می‌دهد که چگونه انواع داده‌ای تعریف می‌شوند و چگونه باید رفتار کنند تا بوسیله CLR پشتیبانی شوند.

CTS-۲-۲-۱

CTS یک مجموعه‌ی پایه از انواع داده‌ای برای زبان‌های تحت .NET فراهم می‌کند. علاوه بر این نحوه اعلان و ایجاد انواع داده‌ای سفارشی و نحوه مدیریت مدت زمان عمر نمونه‌های این نوع داده‌ها را تعیین می‌کند. شکل ۱-۴ نشان می‌دهد چگونه .NET، CTS را سازمان‌دهی می‌کند.



شکل ۱-۴ - انواع داده‌ای پایه تعریف شده در CTS

از این شکل دو چیز به دست می‌آید. واضح‌ترین مورد اینکه، انواع مقداری^۱ یا ارجاعی^۲ گروه‌بندی می‌شوند. این طبقه‌بندی بر اساس نحوه‌ی ذخیره و دسترسی در حافظه است. انواع ارجاعی در یک ناحیه‌ی خاص حافظه به نام Heap، از طریق اشاره‌گرها دستیابی می‌شوند، در حالیکه انواع مقداری مستقیماً در Stack برنامه قرار می‌گیرند. مورد دیگر اینکه، همه انواع داده‌ای اعم از سفارشی و انواع داده‌ای .NET از یک نوع داده‌ی پیش تعریف شده بنام System.Object ارث‌بری می‌کنند. پس مطمئناً همه انواع داده‌ای، یک مجموعه‌ی پایه از متدها و خصوصیات را ارث‌بری می‌کنند.

در .NET، نوع داده^۳ یک عبارت کلی است که به یک کلاس، ساختار، نوع شمارشی یا واسط یا نماینده اشاره می‌کند.

کامپایلری که مطیع مشخصه CTS است، تضمین می‌کند که انواع داده‌ای آن می‌توانند بوسیله CLR میزبانی شوند. این به تنهایی ضامن ارتباط یک زبان با زبان دیگر نیست. یک مجموعه‌ی محدود کننده از مشخصات به نام CLS وجود دارد که قوانینی را برای ارتباط بین زبانی تعریف می‌کنند. این مشخصات، ویژگی‌های حداقلی تعریف می‌کنند که یک کامپایلر با هدف CLR باید شامل باشد. جدول ۱-۱ بعضی از قوانین CLS را نشان می‌دهد:

جدول ۱-۱ - قوانین و ویژگی‌های CLS

^۱ ValueType

^۲ Reference Type

^۳ Type

میدان دید ^۱	این قوانین فقط به آن اعضای از یک نوع داده اعمال می‌شوند که از بیرون اسمبلی تعریف کننده در دسترس است.
کاراکترها و حالت آنها	در دو متغیر متمایز، باید اختلاف آنها بیشتر از حالت کاراکترها در آنها باشد.
انواع اصلی	انواع داده‌ای اصلی مطیع CLS هستند. Byte, Int۱۶, Int۳۲, Int۶۴, Single, Double, Boolean, char, Decimal, IntPtr, String
سازنده	یک سازنده قبل از دسترسی به هر داده از کتابخانه، باید سازنده کلاس پایه را فراخوانی کند.
محدوده‌های آرایه	همه ابعاد آرایه‌ها باید از اندیس صفر شروع شوند.
نوع شمارشی	نوع داده اصلی یک نوع شمارشی باید از نوع Byte و Int۱۶, Int۳۲, Int۶۴ باشد.
متد	انواع داده‌ی پارامترها و مقدار بازگشتی استفاده شده در متد باید مطیع CLS باشند.

این قوانین مشخص و واضح هستند. قطعه کدی از #C را در نظر بگیرید تا نحوه‌ی اعمال این قوانین را ببینیم:

```
public Class Conversion
{
    public double Metric (double inches)
    {
        return (۲.۵۴ * inches) ;
    }
    public double metric (double miles)
    {
        return (miles/۰.۶۲) ;
    }
}
```

اگرچه با کد #C آشنا نیستید، ولی می‌توانید به راحتی ببینید که با قوانین CLS مغایرت دارد. چون دو متد Metric و metric قانون را نقض کرده‌اند. اگرچه در #C جواب می‌دهد، ولی در صورت تعامل با کدنویسی VB.NET با شکست مواجه می‌شود.

۱-۲-۳-اسمبلی‌ها

همه کدهای مدیریت شده که تحت .NET اجرا می‌شوند، باید در یک اسمبلی قرار گیرند. بطور منطقی اسمبلی یک فایل EXE یا DLL است. از نظر فیزیکی ممکن است شامل کلکسیون‌ی از یک یا چند فایل باشد، که هر فایل می‌تواند شامل کد یا منابعی همچون تصاویر یا XML باشند.

زمانی که یک کامپایلر سازگار .NET یک فایل کد منبع را به یک DLL یا EXE تبدیل می‌کند، یک اسمبلی ایجاد می‌شود. همانطور که در شکل ۱-۵ مشاهده می‌کنید، یک اسمبلی شامل یک اظهارنامه^۲، فرا داده و IL است. این موارد را بیشتر بررسی می‌کنیم.

اظهار نامه

^۱ Visiblity (Scope)

^۲ manifest

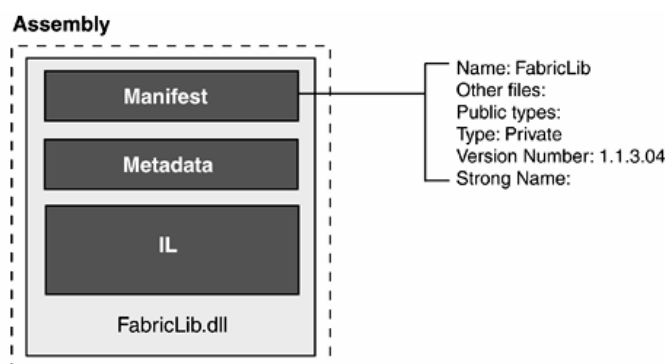
هر اسمبلی باید یک فایل برای در برداشتن اظهارنامه داشته باشد. اظهارنامه شامل جداولی است که در این جداول، اسامی همه فایل‌های موجود در اسمبلی، ارجاعات به اسمبلی‌های بیرونی و اطلاعاتی همچون نام و نسخه اسمبلی لیست می‌شوند. اسمبلی‌های نامگذاری شده، یک امضاء دیجیتالی^۱ منحصر به فرد دارند. زمانی که یک اسمبلی بارگذاری می‌شود، ابتدا فایل manifest توسط CLR بارگذاری می‌شود تا بتواند اعضای اسمبلی را تشخیص دهد.

فرا داده

علاوه بر جداول manifest کامپایلر #C جداول تعریف نوع داده و ارجاع را تولید می‌کند. جداول تعریف، یک توصیف کامل از انواع داده‌ای موجود در IL فراهم می‌کنند. برای مثال، جداولی برای تعریف انواع داده‌ای، متدها، فلیدها و خصوصیات وجود دارند. جداول ارجاع، اطلاعاتی را در مورد ارجاعات به انواع داده‌ای و اسمبلی‌های دیگر شامل هستند. کامپایلر JIT بر پایه‌ی این جداول، IL را به کد ماشین موردنظر تبدیل می‌کند.

IL

نقش IL در حال حاضر بحث شده است. قبل از اینکه CLR بتواند IL را بکاربرد، باید در یک اسمبلی EXE یا DLL بسته‌بندی شود. این دو یکسان نیستند. یک اسمبلی EXE نقطه‌ی ورودی دارد که آن را قابل اجرا می‌سازد. یک اسمبلی DLL بصورت یک کتابخانه از تعاریف انواع داده طراحی می‌شود.



شکل ۱-۵ اسمبلی تک فایلی

اسمبلی، چیزی بالاتر از یک روش منطقی برای بسته‌بندی کد قابل اجرا است. آن قلب مدل NET را برای توسعه‌ی کد، کنترل نسخه و امنیت تشکیل می‌دهد.

- تمام کد مدیریت شده‌ی مربوط به یک برنامه مستقل، یک کنترل یا یک کتابخانه DLL شامل انواع داده‌ای قابل استفاده مجدد، در یک اسمبلی بسته‌بندی می‌شوند. آن تجربه‌ناپذیرترین واحد است که می‌تواند روی یک سیستم بکار گرفته شود. زمانی که یک برنامه آغاز می‌شود، باید فقط اسمبلی‌هایی که برای مقداردهی اولیه لازم هستند، در حافظه باشند. اسمبلی‌های دیگر براساس نیاز بارگذاری می‌شوند. یک توسعه‌دهنده این مزیت را برای تقسیم یک برنامه به چندین اسمبلی براساس فرکانس استفاده آنها بکار می‌برد.
- در NET، اسمبلی یک محدوده‌ی نسخه تشکیل می‌دهد. فیلد نسخه در اظهارنامه روی تمام انواع داده و منابع اسمبلی اعمال می‌گردد. همه فایل‌های تشکیل دهنده اسمبلی بصورت یک واحد منفرد با نسخه‌ی یکسان در نظر گرفته می‌شوند. با جدا کردن بسته فیزیکی از منطقی، NET می‌تواند یک صفت منطقی را مابین چندین فایل فیزیکی به اشتراک گذارد. این یک ویژگی پایه است که یک اسمبلی را از سیستم مبتنی بر DLL سنتی متمایز می‌کند.

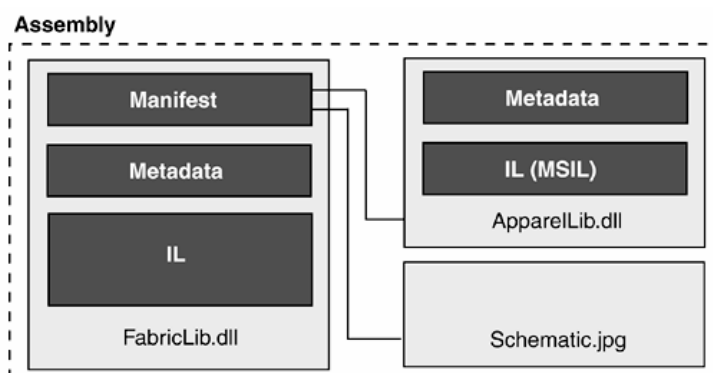
^۱ Digital signature

- اسمبلی یک محدوده امنیت روی جوازهای دسترسی تشکیل می‌دهد. #C معرف‌های دسترسی را برای کنترل نحوه دسترسی انواع داده و اعضای انواع داده در یک اسمبلی بکار می‌برد. دو مورد از کاربرد اسمبلی به عنوان محدوده بصورت زیر است:

— public: دسترسی نامحدود به هر اسمبلی را مجاز می‌شمارد.

— internal: دسترسی را به انواع داده و اعضای داخل آن اسمبلی محدود می‌کند.

همانطور که شرح داده شد، ممکن است یک اسمبلی چندین فایل را شامل شود که این فایل‌ها محدود به ماژول‌های کد نیستند. ممکن است فایل‌های منبع همچون تصاویر گرافیکی و فایل‌های متنی باشند. یک کاربرد عمومی این فایل‌ها، مجاز داشتن منابعی است که یک واسط برای کشور یا زبان یک کاربر فراهم می‌کند. هیچ محدودیتی روی تعداد فایل‌ها در اسمبلی نیست. شکل ۱-۶ طرح یک اسمبلی چند فایل را در دیاگرام اسمبلی چند فایل نشان می‌دهد. توجه داشته باشید که اظهارنامه‌ی اسمبلی شامل اطلاعاتی است که همه فایل‌های اسمبلی را تعیین می‌کند.



شکل ۱-۶-اسمبلی چند فایل

- اگرچه بیشتر اسمبلی‌ها، یک فایل منفرد را شامل هستند، ولی در چندین حالت، اسمبلی‌های چند فایل مزایایی دارند.
- آنها ترکیب ماژول‌های تولید شده در زبان‌های برنامه‌سازی مختلف را مجاز می‌دارند. اگر کدهایی در VB.NET و کدهایی در #C نوشته شده باشند، این دو کد می‌توانند در یک فایل اسمبلی .NET باهم تعامل داشته باشند.
- برای بهینه کردن نحوه‌ی بارگذاری به CLR، ماژول‌های کد می‌توانند تقسیم‌بندی شوند. بایستی کد پر استفاده و مرتبط بهم در یک ماژول قرار گیرند. CLR ماژول‌ها را در صورت نیاز بارگذاری می‌کند. هنگام ایجاد یک کتابخانه‌ی کلاس، لازم است قطعات کد براساس چرخه زندگی و نسخه و امنیت مشترک در اسمبلی‌های مجزایی گروه‌بندی شوند.
- فایل‌های منبع می‌توانند در ماژول‌های مجزایی از ماژول‌های IL قرار بگیرند، تا چندین برنامه منابع مشترک خود را به راحتی به اشتراک گذارند.

اسمبلی‌های چند فایل می‌توانند با اجرای کامپایلر #C از خط فرمان یا برنامه سودمند (Assembly Linker) AL.exe ایجاد شوند. یک مثال کاربرد کامپایلر #C از خط فرمان در بخش‌های بعدی آمده است. توجه داشته باشید که VS.NET ۲۰۰۵ ایجاد اسمبلی‌های چند فایل را پشتیبانی نمی‌کند.

۱-۲-۴-اسمبلی‌های خصوصی^۱ و اشتراکی^۲

^۱ Private

^۲ Shared

اسمبلی‌ها به دو روش ممکن (خصوصی یا سراسری) ایجاد می‌شوند. اسمبلی‌هایی که در فهرست اصلی برنامه یا در یک زیرفهرست آن قرار می‌گیرند، اسمبلی‌های خصوصی خوانده می‌شوند. نصب و بهنگام سازی یک اسمبلی ساده است. فقط لازم است اسمبلی به یک فهرست به نام AppBase در آن برنامه کپی شود. هیچ تنظیم رجیستری لازم نیست. علاوه بر این برای override کردن تنظیمات پیکربندی برنامه می‌توان یک اظهارنامه به برنامه کاربردی اضافه کرد و اجازه داد یک فایل اسمبلی به فهرست AppBase منتقل شود.

یک اسمبلی اشتراکی در یک موقعیت سراسری نصب می‌شود، که ^۱GAC نامیده می‌شود و بوسیله چندین برنامه قابل دستیابی است. مهمترین ویژگی GAC مجاز داشتن اجرای چندین نسخه از اسمبلی در کنار همدیگر می‌باشند. NET برای پشتیبانی از این امر، مشکل تداخل اسمی را با استفاده از چهار صفت جهت شناختن یک اسمبلی رفع می‌کند. نام فایل، مشخصه فرهنگ، شماره نسخه، نشانه کلید عمومی.

معمولاً اسمبلی‌های عمومی در زیرفهرست Assembly از فهرست سیستم عامل (winnt) قرار می‌گیرند. همانطور که در شکل ۱-۷ نشان داده شده است، اسمبلی‌ها در یک قالب خاص لیست می‌شوند که چهار صفت آن نشان داده می‌شود. چارچوب NET یک فایل DLL دارد که Windows Explorer را قادر می‌سازد محتوای GAC را نمایش دهد.

نگاهی سریع به این چهار خصوصیت داریم:

- نام اسمبلی: این همان نام فایل اسمبلی بدون پسوند آن است.
- نسخه: هر اسمبلی یک شماره نسخه دارد که به همه فایل‌های اسمبلی اعمال می‌گردد. آن شامل چهار عدد به قالب زیر است:

<major number>.<minor number>.<build>.<revision>

معمولاً شماره‌های اصلی و فرعی نسخه برای تغییرات بروز می‌شوند، چون سازگاری را تحت تاثیر قرار می‌دهند. شماره نسخه بوسیله یک صفت به نام AssemblyVersion در کد منبع اسمبلی به آن تخصیص داده می‌شود.

- تنظیم فرهنگ: ممکن است محتوای یک اسمبلی به یک زبان و فرهنگ خاصی تخصیص داده شود، که با صفت AssemblyCulture در کد منبع اسمبلی مشخص می‌گردد.

```
[("assembly: AssmblyCulture ("fr-CA]
```

- نشانه کلید عمومی: برای اطمینان از اینکه یک اسمبلی اشتراکی، منحصر بفرد و تصدیق شده است، در NET باید ایجاد کننده اسمبلی آن را با یک نام قوی نشانه‌گذاری کند. این پروسه را امضاء کردن^۲ گویند که جفت کلید عمومی/خصوصی را نیاز دارد. در زمان کامپایل اسمبلی، کلید خصوصی برای تولید یک نام قوی بکار می‌رود، کلید عمومی برای نشانه بزرگ است، پس با عمل درهم‌سازی کلید عمومی، ۸ بایت آخر آن را انتخاب می‌کنند. این نشانه در اظهارنامه‌ی هر اسمبلی سرویس‌گیرنده که به یک اسمبلی اشتراکی ارجاع دارد جای می‌گیرد و برای تشخیص اسمبلی در حین اجرا بکار می‌رود.

^۱ Global Assembly Cache

^۲ Signing

Assembly Name	Version	Culture	Public Key Token
Accessibility	2.0.3600.0		b03f5f7f11d50a3a
ADODB	7.0.3300.0		b03f5f7f11d50a3a
Apphost	2.0.3600.0		b03f5f7f11d50a3a
AspNetMMCExt	2.0.3600.0		b03f5f7f11d50a3a
CRVsPackageLib	1.0.0.0		692fba5521e1304
CrystalDecisions.CrystalReports.Engine	9.1.3300.0		692fba5521e1304

شکل ۱-۷- بخشی از فهرست اسمبلی سراسری

۱-۲-۴- از پیش کامپایل کردن یک اسمبلی

بعد از بارگذاری یک اسمبلی، بایستی IL آن به کد ماشین جاری کامپایل شود. اگر شما با فایل های قابل اجرا در فرمت کد ماشین کار می کنید، سئوالاتی در مورد بهره وری و اینکه آیا ایجاد فایل های قابل اجرای معادل در .NET امکان پذیر است، پیش می آید. جواب سئوال قسمت دوم بله است. .NET یک روش برای از پیش کامپایل کردن یک اسمبلی فراهم می کند.

چار چوب .NET (ابزاری به نام Native Image Generator (Ngen دارد، که برای کامپایل یک اسمبلی به یک "تصویر محلی" بکار گرفته می شود، که در کش تصویر محلی (یک فضای رزرو شده از GAC) ذخیره می شود. هر زمانی که CLR یک اسمبلی را بارگذاری می کند، کش را برای وجود یک تصویر محلی از آن اسمبلی بررسی می کند، اگر باشد آن کد از پیش کامپایل شده را بارگذاری می کند. ظاهراً، این یک ایده ی خوب برای بهبود کارایی به نظر می رسد. اما چندین ایراد دارد.

Ngen یک تصویر برای معماری ماشین فرضی اجرا کننده ایجاد می کند. برای مثال روی هر ماشین سازگار با پردازنده x86، زمانی که JIT در .NET اجرا می گردد، آن از نوع ماشین آگاه بوده و می تواند نکات بهره وری را در نظر بگیرد. نتیجه اینکه اغلب اوقات خروجی آن خارج از عملکرد اسمبلی از پیش کامپایل شده است. ایراد دیگر کاربرد یک تصویر محلی این است که تغییرات پیکربندی سخت افزار یا سیستم عامل یک سیستم، اغلب اوقات اسمبلی از پیش تعریف شده را نامعتبر می کند.

تایید کد^۲

به عنوان بخشی از پروسه ی کامپایل JIT، CLR دو نوع تایید انجام می دهد. تایید IL و ارزیابی فراداده. هدف آن اطمینان از قابل قبول بودن کد نوع امن است. در عمل، بدین معنی است که پارامترهای موجود در یک فراخوانی و متد فراخوانی شده، همنوع هستند یا نوع مقدار بازگشتی یک متد همان نوع برگشتی در اعلان است. خلاصه اینکه CLR از طریق IL و فراداده، سازگاری نوع داده ها را مطمئن می سازد. اگر به غیر از این باشد، یک خطا رخ می دهد.

مزیت کد تایید شده این است که CLR یقین دارد، کد از طریق دسترسی به حافظه ی خارج از محدوده ی مجاز خود نمی تواند برنامه های دیگر را تحت تاثیر قرار دهد. بدین ترتیب CLR برای اجرای امن چندین برنامه در یک پروسه یا فضای آدرس واحدی است و کارایی و کاهش استفاده از منابع سیستم عامل را بهبود می بخشد.

۱-۳- FCL

FCL کلکسیونی از کلاس ها و انواع داده ی دیگر (نوع شمارشی، ساختارها، واسطها) است که برای تمام کدهای مدیریت شده ی نوشته شده در هر زبانی با کد هدف CLR در دسترس هستند. این بسیار مهم است، بدین معنی که این کتابخانه ها مختص کامپایلرهای خاصی نیستند. به عنوان یک توسعه دهنده، شما می توانید با انواع داده موجود در کتابخانه ها آشنا شوید، که این دانش در هر زبان .NET برای شما قابل استفاده است.

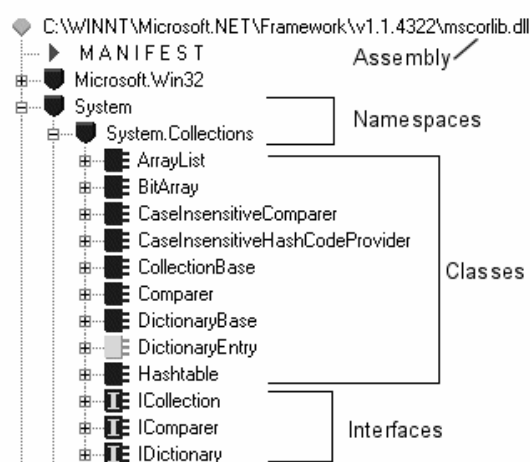
^۱ Native Image

^۲ Code verification

منابع بوسیله FCL از طریق گروه‌بندی‌های منطقی به نام فضای نامی سازمان‌دهی می‌شوند. این گروه‌بندی‌ها براساس محدوده عملکرد می‌باشند. برای مثال، انواع داده‌ای مورد استفاده برای عملیات گرافیکی در فضاهای نامی `System.Drawing` و `System.Drawing.Drawing2D` گروه‌بندی می‌شوند. انواع داده‌ای مورد نیاز برای ورود و خروج فایل‌ها، اعضایی از فضای نامی `System.IO` هستند. فضاهای نامی، یک مفهوم منطقی نه فیزیکی هستند.

FCL صدها اسمبلی DLL را در برمی‌گیرد. هر اسمبلی ممکن است چند فضای نامی را شامل شود. به علاوه، ممکن است یک فضای نامی چندین اسمبلی را بهم ببافد^۱. برای ارائه این مطلب به داخل یک اسمبلی FCL نگاه کنید.

شکل ۱-۸ بخشی از خروجی تولید شده با برنامه `ILDasm.exe` جهت کنترل محتوای اسمبلی `mscorlib` را نمایش می‌دهد. اگرچه یک لیست ناقص است، شما می‌توانید ببینید که `mcorlib` فضای نامی `System` را شامل است و انواع داده‌ای پایه‌ی .NET در آن قرار دارند و فضای نامی `System.Collections` را نیز در بر می‌گیرد که کلاس‌ها و واسطه‌های مورد استفاده برای دستکاری کلکسیون‌های داده را شامل می‌شود.



جدول ۱-۲ بعضی از مهمترین فضاهای نامی .NET را لیست می‌کند.

جدول ۱-۲ تعدادی از فضاهای نامی متداول

کاربرد	فضای اسمی
شامل انواع داده‌ی پایه است که بوسیله همه برنامه‌ها استفاده می‌شود. آن کلاس‌های استثناء، خصوصیتی از پیش تعریف شده، کتابخانه‌ی Math و کلاس‌های مدیریت محیط برنامه را نیز شامل است.	System
واسطه‌ها و کلاس‌های استفاده شده جهت مدیریت کلکسیون‌هایی از اشیا. این کلکسیون‌ها شامل ArrayList, Stack, Hashtable و ... هستند.	System.Collections System.Collections.Specialized System.Collections.Generic

^۱ Span

<p>کلاس‌های مورد استفاده برای عملیات پایگاه داده (ADO.NET). فضاهای نامی سرویس گیرنده‌ی Oracle و SQLServer را پشتیبانی می‌کنند و OleDb, Odbc اتصال داده مورد استفاده را تعریف می‌کنند.</p>	<p>System.Data System.Data.OracleClient System.Data.OleDb System.Data.Odbc</p>
<p>کلاس‌هایی را شامل است که می‌توانند اجرای برنامه، اشکال‌یابی، کار با log های سیستم و شمارنده‌های بهره‌وری را پیگیری کنند.</p>	<p>System.Diagnostics</p>
<p>عملکردهای گرافیکی را برای GDI+ فراهم می‌کند. این فضاهای نامی یک کلاس ترسیم به خوبی fonts, pens, geometric shapes, brushes را در بر دارند.</p>	<p>System.Drawing System.Drawing.Drawing2D System.Drawing.Printing System.Drawing.Text</p>
<p>کلاس‌هایی در ارتباط با اطلاعات مرتبط با فرهنگ دارد که روش مقداردهی تاریخ‌ها، واحد پول و سمبل‌های نمایشی را تحت تاثیر قرار می‌دهد.</p>	<p>System.Globalization</p>
<p>عملیات ورود و خروج فایل و جریان داده را فراهم می‌کند. این کلاس‌ها یک روش برای دسترسی به سیستم‌های فایل سیستم عامل میزبان فراهم می‌کنند.</p>	<p>System.IO</p>
<p>کلاس‌هایی که عملیات و پروتکل‌های شبکه را پشتیبانی می‌کنند. برای مثال WebRequest و WebResponse که یک صفحه وب را درخواست و واکنشی می‌کنند.</p>	<p>System.Net</p>
<p>انواع داده‌ای که تغییر فراداده را در زمان اجرا مجاز می‌دارند، شامل است. فضای نامی Emit به یک کامپایلر یا ابزار، تولید پویای IL و فراداده را اجازه می‌دهد.</p>	<p>System.Reflection System.Reflection.Emit</p>
<p>ارتباط داخلی مابین کد مدیریت شده و کد مدیریت نشده همچون DLL یا COM را فراهم می‌سازد.</p>	<p>System.Runtime.InteropServices</p>
<p>کلاس‌های استفاده شده برای مدیریت امنیت .NET. کلاس‌هایی تعریف می‌کنند که دسترسی به عملیات و منابع را کنترل می‌کنند.</p>	<p>System.Security System.Security.Permission System.Security.Cryptography</p>
<p>کلاس‌هایی که موتور عبارت منظم .NET را پشتیبانی می‌کنند.</p>	<p>System.Text.RegularExpressions</p>
<p>فعالیت‌های برنامه‌نویسی ریسمان یعنی ایجاد ریسمان، همگام‌سازی و دسترسی به استخر ریسمان را مدیریت می‌کنند.</p>	<p>System.Threading System.Threading.Thread</p>
<p>کلاس‌های مرتبط با اینترنت که به ASP.NET معروف هستند. آنها نیازهای ارتباط با سرور، دستکاری کوکی‌ها را مدیریت می‌کنند.</p>	<p>System.Web System.Web.Services</p>
<p>Web.UI دارای کلاس‌ها و واسطه‌هایی است که برای ایجاد کنترل‌ها و صفحات مرتبط با فرم‌های وب استفاده می‌شوند.</p>	<p>System.Web.UI System.Web.UI.WebControls System.Web.Security</p>

کلاس‌هایی که برای ایجاد برنامه‌های GUI میزکار ویندوز استفاده می‌شوند. این کنترل‌ها عبارت هستند از
 ListBox, TextBox, DataGrid, Buttons:

System.Windows.Forms

یک مجموعه از انواع داده‌ای برای پردازش XML

System.XML

فضاهای اسمی یک نقشه برای هدایت کردن FCL هستند.

۴-۱- کار با چارچوب .NET و SDK

SDK^۱ مربوط به چارچوب .NET، ابزارها، کامپایلرها و مستندسازی را شامل می‌شود که جهت ایجاد نرم‌افزار روی هر ماشینی که چارچوب .NET روی آن نصب شده است، لازم هستند. SDK به اندازه ۱۰۰MB بصورت رایگان از سایت مایکروسافت روی ویندوز XP و ۲۰۰۳ و سرور و دنباله‌ای از سیستم‌عامل‌های ویندوز قابل نصب می‌باشد. اگر .NET vs را روی سیستم خود نصب کرده باشید، نیازی به این کار نخواهد بود. لازم است نسخه چارچوب .NET خود را متناسب با نرم‌افزارهای توسعه یافته بروز کنید.

۱-۴-۱- بروزآوری چارچوب .NET

برخلاف بسیاری از محیط‌های توسعه، نصب یک نسخه جدید از چارچوب مشکل‌ساز نیست. پروسه‌ی نصب، نسخه‌ی توسعه یافته را در یک فهرست جدید با نام نسخه قرار می‌دهد. مهمتر اینکه هیچ وابستگی فایلی مابین نسخه‌های جدید و قدیمی وجود ندارد. پس همه نسخه‌ها روی سیستم، عملیاتی هستند. اگرچه در سیستم‌عامل‌های مختلف متفاوت است، ولی معمولاً در مسیرهایی شبیه زیر قرار می‌گیرند:

```
\wint\Microsoft.NET\Framework\V۱,۰,۳۷۰۵
```

```
\wint\Microsoft.NET\Framework\V۱,۱,۴۳۲۲
```

```
\wint\Microsoft.NET\Fromwork\۷۲,۰,۴۰۶۰۷
```

در نصب نسخه‌های جدید یک نرم‌افزار، سئوالی در مورد سازگاری با برنامه‌های توسعه‌یافته توسط نسخه قدیمی پیش می‌آید. .NET اجرای برنامه‌های موجود را ساده‌تر می‌کند. نکته کلیدی این کار به فایل پیکربندی برنامه مربوط می‌شود.

۱-۴-۲- ابزار چارچوب .NET

چارچوب .NET تا حد امکان بسیاری از عملیات را اتوماتیک می‌کند و جزئیات را از دید توسعه‌دهنده پنهان می‌سازد. با این وجود، بعضی اوقات دخالت دستی مورد نیاز است. احتمال دارد به فهم بهتر جزئیات یک اسمبلی و آماده‌سازی یک برنامه برای توسعه نیاز داشته باشیم. چندین نمونه از این کارها را در اینجا عنوان می‌کنیم:

- اضافه کردن یک فایل به یک اسمبلی
- مشاهده محتویات یک اسمبلی
- مشاهده جزئیات یک کلاس خاص
- تولید زوج کلید عمومی / خصوصی برای ایجاد یک نام اسمبلی قوی

^۱ Software development Kit

بیشتر این موارد در فصل‌های بعدی بهتر بررسی خواهند شد، ولی آگاهی از وجود این ابزار مفید است.
جدول ۱-۳ بعضی از ابزارهای موجود برای توسعه و توزیع برنامه‌های شما را لیست می‌کند. سه مورد از آنها را در این قسمت بررسی خواهیم کرد.

جدول ۱-۳- ابزار انتخابی از چارچوب .NET

ابزار	توصیف
AL.exe AssemblyLinker	می‌تواند برای ایجاد یک اسمبلی مرکب از ماژول‌های تولید شده توسط کامپایلرهای مختلف استفاده شود. همچنین برای ایجاد اسمبلی‌هایی استفاده می‌شود که فقط شامل منبع هستند (satellite).
Fuslogrw.exe Assembly Binding log Viewer	برای اشکال‌یابی پروسه بارگذاری اسمبلی استفاده می‌شود. آن مراحل تلاش برای بارگذاری یک اسمبلی را پی‌گیری می‌کند.
Gacutil.exe Global Assembly Cache Tool	برای نصب یا حذف یک اسمبلی از GAC استفاده می‌شود و همچنین برای لیست کردن محتویات GAC نیز استفاده می‌شود.
Ildasm.exe MSSL disassembler	برای کاوش یک اسمبلی، IL آن و فراداده‌ی آن استفاده می‌شود.
Mscorcfg.msc NET.Framework Configuration Tool	یک MMC که برای پیکربندی یک اسمبلی استفاده می‌شود. در حالیکه از تغییرات دستی مستقیم به یک فایل پیکربندی برنامه را در نظر نمی‌گیرد. مختص مدیران سیستم طراحی شده است. برای تک تک برنامه‌نویسان در دسترس است.
Ngan.exe Native Image Generator	IL یک اسمبلی را به کد ماشین محلی کامپایل می‌کند. این تصویر در کش تصویر محلی قرار می‌گیرد.
Sn.exe Strong name tool	کلیدهایی تولید می‌کند که برای ایجاد یک نام قوی استفاده می‌شود.
Wincv.exe Windows Forms class Viewer	یک واسط مجازی برای نمایش اطلاعات قابل جستجو درباره یک کلاس است.
Wsdll.exe	اطلاعات توصیفی درباره یک سرویس وب تولید می‌کند که بوسیله یک سرویس‌گیرنده جهت دسترسی به سرویس استفاده می‌شود.

بیشتر این ابزار در زیرفهرست SDK قرار گرفته‌اند:

C:\program files\Microsoft.NET\SDK\ V۲.۰\Bin

برای اینکه از هر مسیری بتوان اجرا کرد، ضروری است که این مسیر را به متغیر سیستمی path اضافه کنید. مراحل کار بصورت زیر است:

۱. روی My Computer کلیک راست کرده و Properties را انتخاب کنید.

۲. از برگه Advanced گزینه Enviroment Variables را انتخاب کنید.

۳. متغیر Path را برگزیده و زیرفهرست SDK را به آن اضافه کنید.

اگر VS را نصب کرده‌اید، یک روش ساده استفاده از خط فرمان از پیش پیکربندی شده‌ی VS است. آن بطور اتوماتیک اطلاعات مسیر را مقداردهی اولیه کرده و دسترسی به ابزار خط فرمان را قادر می‌سازد.

ILDasm.exe

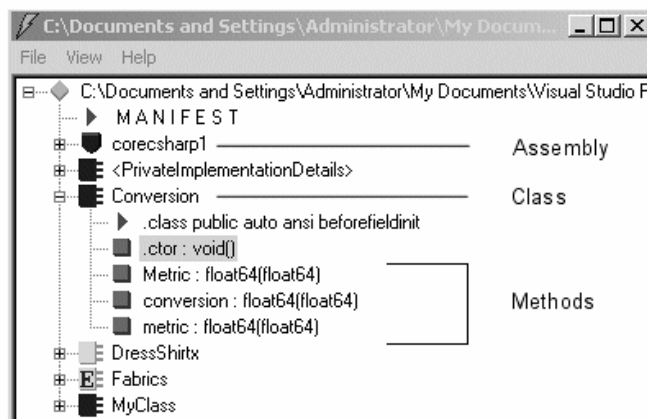
ابزار Intermediate Language Disassembler توسط SDK چارچوب NET فراهم شده و در مسیر نصب SDK قرار گرفته است. آن برای بررسی محیط اسمبلی NET گرانبهاست و یکی از اولین ابزاری است که جهت کار با کد و اسمبلی‌های NET باید با آن آشنا شوید.

ساده‌ترین راه استفاده از آن ابزار در خط فرمان بصورت زیر است:

C:\ILDasm /adv

سوئیچ اختیاری adv/ گزینه‌های پیشرفته را نیز نشان می‌دهد. این دستور محیط GUI را احضار می‌کند تا یک منوی File برای انتخاب اسمبلی مورد نظرش فراهم کند. توجه کنید که آن ابزار فایل‌های GAC را باز نمی‌کند.

شکل ۱-۹ یک مثال از خروجی این ابزار را نشان می‌دهد. محتوای خروجی در فرمت سلسله مراتبی قابل خواندن نمایش داده می‌شود، که نام اسمبلی و همه اعضای آن را در بردارد.



شکل ۱-۹ مشاهده محتویات اسمبلی بوسیله ILDasm.exe

این سلسله مراتب می‌تواند تا رسیدن به دستورات IL مربوط به یک عضو خاص باز شود. به عنوان مثال، کلاس Conversion را در نظر بگیرید. شکل نشان می‌دهد که آن سه متد Metric, conversion, metric دارد. کد منبع اصلی، این را تصدیق می‌کند.

```
public class Conversion
{
    public double Metric(double inches)
    {
        return (۲,۵*inches);
    }

    [CLSCompliantAttribute(false)]

    public double metric(double miles)
    {
        return (miles/۰,۶۲);
    }

    public double Conversion(double pounds)
    {

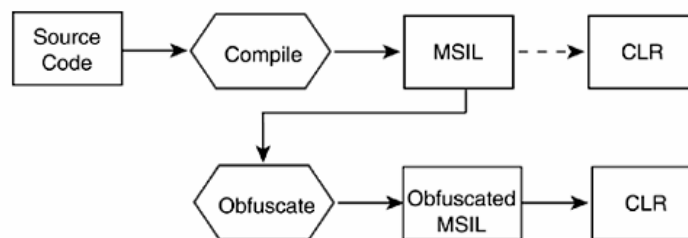
```

```
return (pounds*۴۵۴);
}
}
```

با دابل کلیک کردن روی متد `metric` صفحه‌ای باز شده و IL مربوط به متد را نشان می‌دهد. می‌توان از ILdasm به عنوان یک ابزار یادگیری مفاهیم IL و اسمبلی‌ها استفاده کرد. همچنین چندین کاربرد عملی دارد. فرض کنید یک قطعه‌ی ثالث دارید که هیچ مستنداتی در مورد آن ندارید. ILdasm یک نقطه‌ی شروع مفیدی برای کشف جزئیات واسط اسمبلی فراهم می‌کند. در ILdasm یک منوی `File-Dump` وجود دارد که می‌توانیم مستندات برنامه را در یک فایل متنی ذخیره کنیم.

ILdasm و ابهام^۱

با وجود چنین برنامه‌ای، سؤال اینجاست که چگونه می‌توانیم کد خود را در برابر چنین برنامه‌هایی محافظت کنیم. یک راه حل، کاربرد ابهام است. یک تکنیک برای تغییر نام و دستکاری کد، بطوریکه محتوای یک اسمبلی توسط انسان خوانا نباشد. ابهام همان رمزنگاری نیست. رمزگذاری یک مرحله رمزگشایی نیاز دارد تا کامپایلر JIT بتواند آن را پردازش کند. عمل ابهام، کد IL را به یک شکلی تبدیل می‌کند که می‌تواند توسط ابزار محیط توسعه شما کامپایل شود. شکل ۱-۱۰ مراحل کار را نشان می‌دهد.



شکل ۱-۱۰

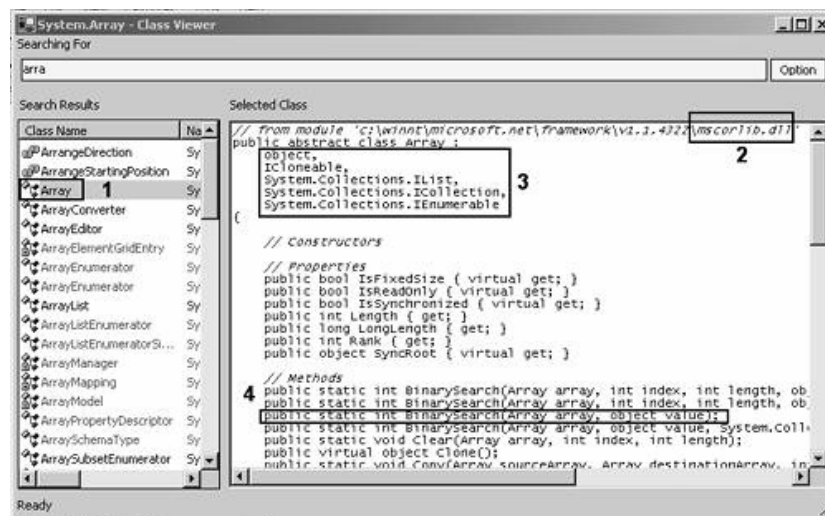
کد ابهام عملاً معادل با کد IL اسمبلی است و در زمان اجرا توسط CLR کد یکسانی را تولید می‌کند. آن چگونه این کار را انجام می‌دهد؟ تغییر نام انواع داده‌ای بامعنی و اعضای آنها به اسامی بی‌معنا معمول‌ترین حيله است. اگر به کد ابهام نظری بیاندازید، مقدار زیادی انواع داده به نام‌های "a"، "b" خواهید دید. البته الگوریتم ابهام باید به اندازه‌ی کافی هوشمند باشد، بطوریکه انواع داده‌ای استفاده شده از خارج اسمبلی به اسم اصلی وابسته نباشند. حيله معمول دیگر، تغییر کنترل جریان کد بدون تغییر منطق آن می‌باشد. برای مثال یک دستور `while` را با ترکیبی از دستورات `if, goto` جایگزین کند.

در .NET SDK ابهام‌کننده‌ای وجود ندارد. Dotfuscator Community Edition یک نسخه‌ی محدود از یک محصول تجاری با .NET VS همراه است.

Wincv.exe

Wincv یک ناظر کلاس، مشابه Visual Studio Object Viewer است. آن در فهرست `program` قرار گرفته است و از خط فرمان قابل اجراست. زمانی که پنجره ظاهر می‌گردد، نام کلاس مورد جستجو را وارد کنید. شکل ۱-۱۱ را ببینید

^۱ obfuscation



شکل ۱-۱۱

WinCV اطلاعات زیادی درباره هر نوع داده از کتابخانه‌های کلاس پایه را فراهم می‌کند. چهار ناحیه مشخص شده در شکل قبلی یک نمونه از اطلاعات موجود را فراهم می‌کند:

۱. کلاس System.Array کاوش می‌شود.
۲. این کلاس در اسمبلی mscorlib.dll قرار دارد. این اسمبلی انواع داده‌ای مدیریت شده NET را در بر دارد.
۳. این لیست شامل کلاس، شی و واسطه‌هایی است که کلاس Array از آنها ارث می‌برد.
۴. تعریف متدهای کلاس را شامل است. دسترس‌پذیری، نوع داده‌ها و پارمترهای متد در این تعریف قرار دارند، که امضاء متد نامیده می‌شود.

۱-۴-۳- ابزار پیکربندی چارچوب

این ابزار جهت دستیابی کد یک روش آسان برای مدیریت و پیکربندی اسمبلی‌ها فراهم می‌کنند. این ابزار بصورت یک Sanp-in در MMC بسته‌بندی می‌شوند. برای دسترسی به آن در Control Panel گزینه Administrative Tools را انتخاب کرده و سپس Microsoft.NET Framework Configuration را اجرا کنید. این ابزار برای مدیرانی که کارهای زیر را نیاز دارند طراحی شده است.

- مدیریت اسمبلی‌ها: اسمبلی‌ها می‌توانند به GAC اضافه یا از آن حذف شوند.
- پیکربندی: زمانی که یک اسمبلی بروز می‌شود، ناشر اسمبلی مسئول بروزآوری سیاست مقید کردن^۱ اسمبلی است. این سیاست به CLR می‌گوید، در هنگام بارگذاری یک اسمبلی، کدام نسخه از آن را بارگذاری کند. برای مثال، اگر یک اسمبلی با نسخه ۱.۱ اسمبلی نسخه ۱.۰ را جایگزین کند. این سیاست، بارگذاری نسخه ۱.۰ را به ۱.۱ هدایت می‌کند. این اطلاعات راهنما در یک فایل پیکربندی قرار می‌گیرند.
- مشاهده امنیت چارچوب NET و تغییر امنیت یک اسمبلی: در امنیت NET به یک اسمبلی جوازها و حقوق معین انتساب داده می‌شود. به علاوه، یک اسمبلی در مورد اسمبلی‌های دیگری که به آن دسترسی دارند، می‌تواند تعیین کند که جوازهای معینی را نیاز داشته باشد.

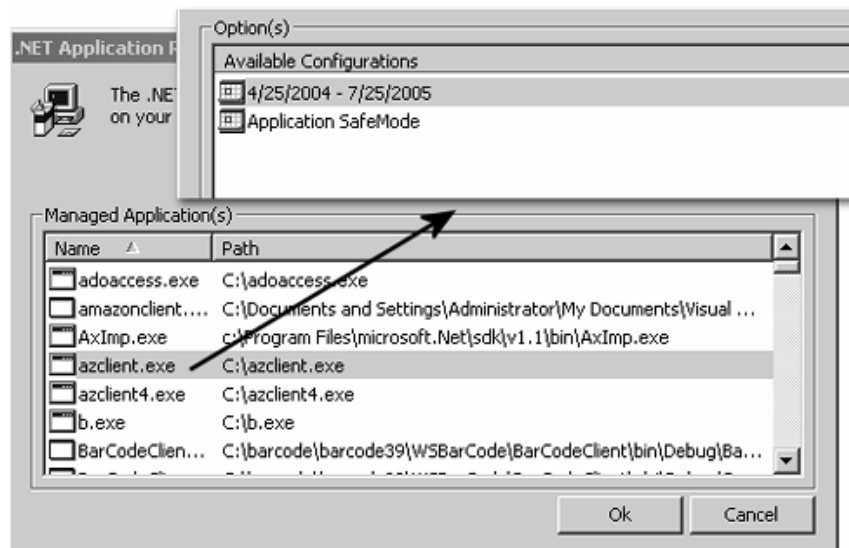
^۱ Binding

- مدیریت نحوه‌ی تعامل تک تک برنامه‌ها با یک اسمبلی یا مجموعه‌ای از اسمبلی‌ها: شما می‌توانید لیستی از اسمبلی‌های مورد استفاده یک برنامه را دیده و نسخه آن اسمبلی‌ها را تنظیم کنید.

برای ارائه یک کاربرد عملی از ابزار پیکربندی، توجه کنید که آن چگونه یکی از عمومی‌ترین مسایل آزاردهنده پروسه توسعه نرم‌افزار را اداره می‌کند. (زمانی که برنامه جاری با شکست مواجه می‌شود نیاز است به نسخه قبلی برگردد.) زمانی که سرویس‌دهنده با DLL‌ها یا اسمبلی‌ها درگیر می‌شود، می‌تواند یک کار مشکلی شود. NET یک راه‌حل جالب برای این مسئله پیشنهاد می‌کند: هر زمانی که یک برنامه اجرا می‌شود، مجموعه اسمبلی‌های مورد استفاده‌ی برنامه ثبت می‌شوند، اگر آنها نسبت به قبل تغییر نکرده باشند، CLR از آنها صرف‌نظر می‌کند، در غیر این صورت یک کپی از مجموعه‌ی جدید اسمبلی‌ها ذخیره می‌شود.

زمانی که یک برنامه با شکست مواجه می‌شود، برگشت به نسخه قبلی یک انتخاب برای برنامه‌نویس است. ابزار پیکربندی برای هدایت برنامه به نسخه اخیر می‌تواند استفاده شوند. با این وجود، احتمال درگیری با چندین اسمبلی وجود دارد. اینجاست که ابزار پیکربندی به کار می‌آیند. به شما اجازه می‌دهد پیکربندهای اسمبلی قبلی را دیده و اسمبلی‌هایی که باید استفاده شوند را انتخاب کنید.

برای مشاهده و انتخاب پیکربندی‌های قبلی از منوی Configuration گزینه Fix an Application را اجرا کنید. شکل ۱۲-۱ دو کادر باز شده‌ی متوالی را ترکیب می‌کند. برنامه‌هایی که اجرا و ثبت شده‌اند را در پنجره اصلی لیست می‌کند. هنگام کلیک بر روی یک برنامه، پنجره‌ی کوچکتری باز می‌شود. این پنجره اخیرترین پیکربندی‌های تخصیص یافته به برنامه را لیست می‌کند. حال می‌توانید پیکربندی‌های اسمبلی مورد دلخواه خود را انتخاب کنید.



شکل ۱۲-۱ کاربرد ابزار پیکربندی برای انتخاب نسخه‌ی اسمبلی

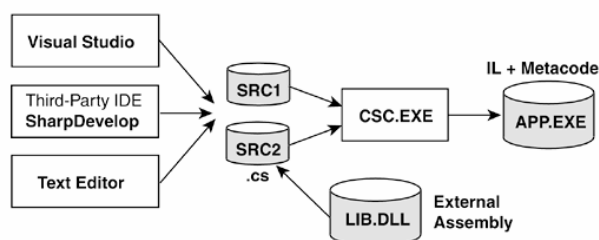
این ابزار پیکربندی مخصوص مدیران هستند. توسعه‌دهندگان باید به سه ویزارد^۱ از این ابزار استناد کنند. Fix an Application, Trust An Assembly, Adjust .NET Security. این ویزاردها را از طریق Framework wizards در Administrative Tools دستیابی کنید.

^۱ wizard

۵-۱- فهم کامپایلر C#

اغلب توسعه‌دهندگان، برنامه‌های .NET بزرگ خود را با استفاده از VS یا هر IDE دیگر جهت وارد کردن کد منبع، پیوند اسمبلی‌های خارجی، اشکال‌یابی، ایجاد خروجی کامپایل شده نهایی می‌نویسند. اگر در این شاخه کار می‌کنید، فهمیدن نحوه کار با .NET SDK و کامپایلر #C ضروری نیست. با این وجود، این عمل فهم شما از پروسه کامپایل .NET را افزایش داده و به شما یک احساس بهتر برای کار با اسمبلی‌ها می‌دهد. این بخش شما را با برنامه‌های فرعی SDK در خط فرمان آشنا می‌سازد. بیشتر برنامه‌های ارائه شده در بخش قبلی از خط فرمان در دسترس هستند. شما کار با آنها را نسبت به محیط IDE مفید خواهید یافت.

شکل ۱-۱۳ مراحل اصلی تبدیل کد منبع به کد کامپایل شده نهایی را نشان می‌دهد. هدف این بخش نشان دادن نحوه ایجاد یک برنامه بوسیله یک ویرایشگر متنی و کامپایلر #C می‌باشد. در ادامه، بعضی از سوئیچ‌های کامپایلر که از دید IDE پنهان هستند را خواهیم دید.



شکل ۱-۱۳- پروسه کامپایل

۵-۱-۱ محل کامپایلر

کامپایلر (CSC.exe) #C در مسیری که چارچوب .NET نصب می‌شود، قرار می‌گیرد.

C:\winnt\Microsoft.NET\Framework\V2.0.40607

البته این مسیر در سیستم‌عامل‌های مختلف و نسخه‌های مختلف چارچوب متفاوت است. برای تایید وجود کامپایلر #C (اگر مسیر آن را در متغیر سیستمی Path قرار داده باشید)، دستور زیر را در خط فرمان اجرا کنید.

C:\>CSC /help

۵-۱-۲ کامپایل کردن از طریق خط فرمان

برای کامپایل کردن برنامه کنسولی Client.cs به برنامه اجرایی Client.exe، دستورات زیر را در خط فرمان وارد کنید.

C:\>CSC Client.cs

C:\>CSC /t:exe client.cs

هر دو دستور کد منبع را به یک فایل اجرایی کامپایل می‌کنند. همانطور که جدول ۴-۱ نشان می‌دهد با سوئیچ /t: می‌توان نوع خروجی را مشخص کرد.

در صورتی که برای برنامه‌های ویندوزی، t:exe/ استفاده شود، کنسول به عنوان پیش زمینه برای آن ظاهر خواهد شد.

جدول ۴-۱- گزینه‌های انتخابی از کامپایلر #C

گزینه توصیف

/addModule	یک ماژول را برای درج در اسمبلی ایجاد شده مشخص می کند. ساده ترین روش ایجاد اسمبلی چند فایلی است.
/debug	باعث می شود اطلاعات اشکال یابی تولید شوند.
/define	راهنماهای پیش پردازنده جهت انتقال به کامپایلر را مشخص می کند. <code>define:DEBUG/</code>
/delaySign	یک اسمبلی را با به تاخیر انداختن امضاء نام قوی ایجاد می کند.
/doc	برای تعیین یک فایل خروجی جهت در برگرفتن مستندات XML استفاده می شود.
/keyfile	مسیر فایل <code>snk</code> را تعیین می کند، که زوج کلید استفاده شده برای امضاء نام قوی را در بردارد.
/lib	محل اسمبلی هایی که در گزینه <code>reference/</code> شامل هستند را تعیین می کند.
/out	نام فایل خروجی کامپایل شده را مشخص می کند. بطور پیش فرض نام همان فایل ورودی با پسوند <code>exe</code> است.
/reference	به یک اسمبلی خارجی ارجاع می کند.
/resource	برای تعبیه کردن فایل های منبع به یک اسمبلی استفاده می شود.
/target (t)	نوع فایل خروجی تولید شده را تعیین می کند.
t:exe/	یک برنامه کنسولی <code>exe</code> می سازد. این خروجی پیش فرض است.
t:library/	یک اسمبلی <code>dll</code> ایجاد می کند.
t:module/	یک ماژول فایل اجرایی قابل حمل بدون اظهارنامه ایجاد می کند.
t:winexe/	یک اسمبلی ویندوزی <code>exe</code> ایجاد می کند.

ارزش واقعی کار با کامپایلر خام، توانایی آن در کار کردن با چندین فایل اسمبلی است. برای نشان دادن این مورد، دو فایل منبع #C به نام های `Client.CS` و `ClientLib.CS` را ایجاد کنید.

```
//Client.CS
using Sysyem ;
public class MyApp
{
    static Vold Main (string[ ] args)
    {
        SnowName.ShowMe ( "Core C#");
    }
}

//Clientlib.CS
using System ;
public class ShowName
{
    public static void ShowMe (string MyName)
    {
        Console .write line (MyName);
    }
}
```

فهم جزئیات کد مهم نیست، فقط بدانید که روتین `Client`، یک تابع از `Clientlib` را برای نوشتن یک پیام به کنسول فراخوانی می کند. با استفاده از کامپایلر #C می توانیم این رابطه را در چندین روش نمایش دهیم.

مثال ۱: کامپایل کردن چندین فایل

کامپایلر C# هر تعداد فایل منبع ورودی را پذیرفته و آنها را در یک فایل اسمبلی واحد ترکیب می‌کند.

```
CSC /out:client.exe client.CS clientlib.CS
```

مثال ۲: ایجاد یک کتابخانه کد و استفاده آن

کد Clientlib می‌تواند در یک کتابخانه‌ی مجزا قرار گیرد و توسط هر سرویس‌گیرنده‌ای دستیابی شود.

```
CSC /t:library Clientlib.CS
```

خروجی یک اسمبلی به نام Clientlib.dll است. حال کد Client را کامپایل کرده و به این اسمبلی خارجی ارجاع دهید.

```
CSC /r:Clientlib.dll Client.CS
```

خروجی یک اسمبلی به نام Client.exe است. اگر شما این اسمبلی را با ILdasm باز کنید. می‌بینید که اظهارنامه‌ی آن یک ارجاع به اسمبلی Clientlib دارد.

مثال ۳: ایجاد اسمبلی چندفایلی

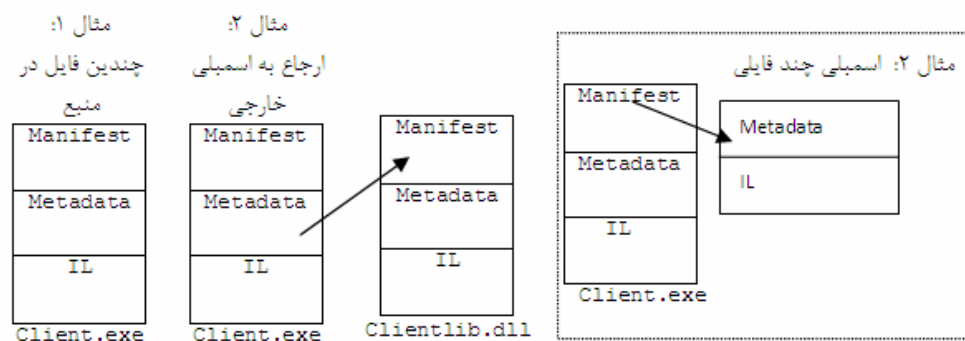
به جای ایجاد یک اسمبلی مجزا، Clientlib می‌تواند در داخل اسمبلی Client.exe بصورت یک فایل مجزا بسته‌بندی شود. چون فقط یک فایل در اسمبلی اظهارنامه امکان‌پذیر است. ابتدا لازم است Clientlib.CS را به یک ماژول کامپایل کنیم. این عمل با انتخاب t:Module/ انجام می‌گیرد.

```
CSC /t:module Clientlib.CS
```

فایل خروجی به نام Clientlib.NETModule است. حال می‌توانید با سوئیچ addModule/ این ماژول را به اسمبلی موردنظر اضافه کنید.

```
CSC /addModule:Clientlib.NETModule Client.CS
```

اسمبلی منتج شده شامل دو فایل Client.exe و Clientlib.NETModule است. این مثال‌های ساده در شکل ۱-۱۴ نشان داده شده‌اند.



شکل ۱-۱۴

۱-۶- خلاصه

- چارچوب NET شامل CLR و FCL است. CLR همه کارهای مرتبط با کد اجرایی را مدیریت می‌کند. ابتدا مطمئن می‌شود که کد براساس استاندارد CLS مطیع CLR است. سپس یک برنامه کاربردی را بارگذاری کرده و همه اسمبلی‌های وابسته آن را پیدا می‌کند.
- JIT، IL موجود در اسمبلی برنامه کاربردی (کوچکترین واحد کد قابل توسعه در NET) را به کد ماشین محلی تبدیل می‌کند. CLR در حین اجرای برنامه واقعی امنیت، ریسمان‌ها، حافظه و جمع‌آوری زباله را مدیریت می‌کند.

- همه‌ی کدها جهت استفاده توسط CLR، باید در یک اسمبلی بسته‌بندی شوند.
- اسمبلی یک فایل واحد یا گروهی از چندین فایل فیزیکی است که به عنوان یک واحد منفرد در نظر گرفته می‌شود. آن مازول‌های کد را همانند فایل‌های منبع شامل می‌شود.
- FCL یک مجموعه از کلاس‌ها و انواع داده‌ای دیگر قابل استفاده مجدد را فراهم می‌کند، که برای همه کدهای مطیع CLR در دسترس هستند. آن نیاز به کتابخانه‌های مختص یک کامپایلر را حذف می‌کند.
- اگرچه FCL شامل چندین فایل DLL فیزیکی است که هزاران نوع داده را در بر می‌گیرند، آن به کمک فضاهای نامی، یک ساختار منطقی روی همه انواع داده فراهم می‌کند.
- برای کمک به توسعه‌دهنده در امر اشکال‌یابی و آماده‌سازی نرم‌افزار، .NET مجموعه‌ای از برنامه‌های سودمند دارد که مدیر را قادر می‌سازد کارهایی از قبیل مدیریت اسمبلی‌ها، اسمبلی‌های از پیش کامپایل شده، اضافه کردن فایل‌ها به یک اسمبلی و مشاهده جزئیات یک کلاس را انجام دهد.
- به علاوه تعداد زیادی ابزار .NET. OpenSource با هدف کمک به پروسه توسعه فراهم ساخته است.

فصل دوم

اصول زبان #C

آنچه که در این فصل یاد خواهید گرفت:

- مروری بر برنامه #C: علاوه بر عناصر اصلی یک برنامه #C، لازم است یک توسعه‌دهنده از ویژگی‌های دیگر .NET، همچون توضیحات و قراردادهای نامگذاری پیشنهاد شده آگاه باشد.
- انواع داده‌ی اولیه^۱: انواع داده‌ی اولیه همان انواع داده پایه‌ای تعریف شده بوسیله FCL برای نمایش اعداد، کاراکتر و تاریخ‌ها است.
- عملگرها: #C گرامر عملگرهای سنتی را برای انجام عملیات ریاضی و شرطی به کار می‌برد.
- نوع شمارشی: نوع شمارشی یک روش مناسب برای تخصیص توصیف است که می‌تواند برای ارجاع به یک مجموعه اصیل مقادیر استفاده شود.
- انواع داده مقداری و ارجاعی: همه انواع داده‌ای در #C از نوع مقداری یا ارجاعی هستند. فهم تفاوت مابین این دو نوع داده مهم است و اینکه چگونه کارایی برنامه را تحت تاثیر قرار می‌دهند.

در سپتامبر ۲۰۰۰، گروه کاری ECMA یک استاندارد پیشنهاد شده برای زبان برنامه‌نویسی #C تعریف کردند، که هدف طراحی این زبان را تولید یک زبان برنامه‌نویسی آسان، پیشرفته، همه‌منظوره و شی‌گرا بیان کرد. استاندارد تعریف شده ECMA-۳۳۴ است که یک زبان مرتب با گرامر زبان جاوا و قوانینی از ++C است. آن یک زبان برای ارتقاء قدرت نرم‌افزار با کنترل محدوده‌ی آرایه، کنترل نوع داده قوی^۲ و ممانعت از متغیرهای مقداردهی اولیه نشده طراحی شده است.

این فصل اصول زبان #C را به شما معرفی می‌کند و بخش‌های اصلی یک برنامه #C را نشان می‌دهد. انواع داده‌ی مقداری و ارجاعی را مقایسه می‌کند و گرامر اپراتورها و دستورات برنامه را شرح می‌دهد.

برنامه‌نویس حرفه‌ای با این مفاهیم آشنا است. با این وجود، بخش انواع داده مقداری و ارجاعی توجه بیشتری لازم دارد. فهم اختلاف در نحوه‌ی اداره‌ی انواع داده مقداری و ارجاعی روی طراحی برنامه شما تاثیرگذار خواهد بود.

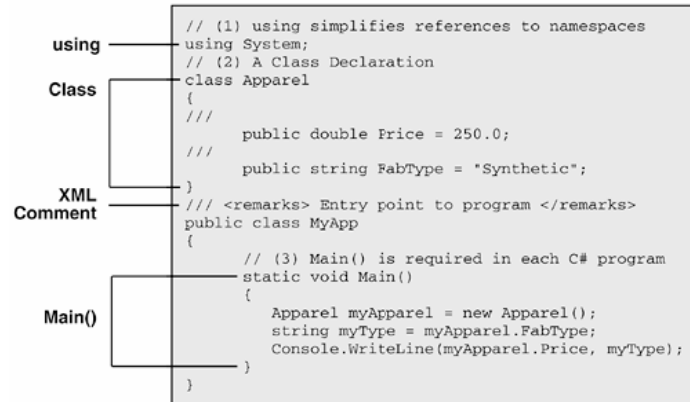
^۱ Primitive

^۲ Strong Type cheking

۱-۲- چیدمان یک برنامه C#

شکل ۱-۲ بعضی از ویژگی‌های یک برنامه C# را نشان می‌دهد.

شکل ۱-۲



کد موجود در شکل ۱-۲ شامل یک کلاس بنام myApp است که منطق برنامه را در بر دارد و یک کلاس Apparel که داده‌هایی را در بر می‌گیرد. این برنامه یک نمونه از Apparel را ایجاد کرده و آن را myApparel نامگذاری می‌کند. سپس این شی برای چاپ مقادیر اعضای کلاس بنام‌های FabType و Price در کنسول استفاده می‌شود. مهمترین ویژگی‌های مورد توجه بصورت زیر هستند:

۱- دستور using فضای نامی System را مشخص می‌کند. همانطور که می‌دانید فضای نامی System همه انواع داده‌ای پایه را شامل می‌شود. دستور using به کامپایلر می‌گوید زمان رفع ارجاع‌ها این فضای نامی را جستجو کنند، که استفاده از اسامی کامل را غیرضروری می‌کند. برای مثال، به جای System.web.UI.WebControls.Label می‌توانید به Label ارجاع کنید.

۲- همه داده‌ها و منطق برنامه باید در یک تعریف نوع داده قرار گیرند. همه داده‌ها و منطق برنامه باید در یک کلاس، ساختار، نوع شمارشی، واسط یا نماینده تعبیه شوند. برخلاف VB، در C# متغیر سراسری خارج از دامنه وجود ندارد. دسترسی به انواع داده و اعضای آنها به وسیله معرف‌ها^۱ کنترل می‌شود. در این مثال معرف public به کلاس‌های خارجی اجازه می‌دهد به دو عضو کلاس Apparel دسترسی داشته باشند.

۳- متد Main () برای هر برنامه اجرایی C# لازم است. این متد به عنوان نقطه ورودی برنامه عمل می‌کند. آن باید همواره معرف static را داشته باشد و حرف M آن بزرگ باشد. شکل‌های overload شده‌ی Main یک نوع داده بازگشتی و یک لیست پارامتر ورودی را تعریف می‌کنند. نمونه‌ی زیر یک مقدار صحیح بر می‌گرداند.

```

static int Main()
{
    return ۰; // must return an integer value
}
  
```

مثال زیر، لیست آرگون‌های خط فرمان را به عنوان پارامتر دریافت کرده و یک مقدار صحیح بر می‌گرداند.

```

static int Main(string[] args)
{
    // loop through arguments
    foreach (string myArg in args)
        Console.WriteLine(myArg);
    return ۰;
}
  
```

^۱ modifier

پارامتر آن یک آرایه‌ی رشته‌ای از محتوای خط فرمان است که برای احضار برنامه استفاده می‌شود. برای مثال، این خط فرمان برنامه MyApparel را اجرا می‌کند و دو مقدار از طریق پارامتر بر آن ارسال می‌کند.

```
C:\> MyApparel ۰ ۱
```

توجه: خصوصیت System.Environment.CommandLine می‌تواند محتویات خط فرمان را بدست آورد.

۲-۱-۱- تذکرات عمومی برنامه‌نویسی #C

حساسیت به حالت حروف

همه متغیرها و کلمات کلید با حساسیت به حالت حروف متمایز می‌شوند. در شکل ۲-۱ class را با Class جایگزین کنید، کد کامپایل نخواهد شد.

قراردادهای نامگذاری

استاندارد ECMA قراردادهای نام‌گذاری فراهم می‌کند که در کد #C شما رعایت می‌شود. علاوه بر ارتقاء سازگاری، رعایت سیاست نامگذاری، خطاهای مرتبط با حساسیت به حالت حروف را می‌تواند کاهش دهد. جدول ۲-۱ بعضی از توصیه‌های مهم را خلاصه کرده است.

جدول ۲-۱- قراردادهای نامگذاری

نوع	توضیحات
کلاس	<ul style="list-style-type: none"> اسم یا عبارت اسمی سعی کنید با حرف I شروع نشود. زیر خط را بکار نبرید.
ثابت	<pre>public const double GramToPound = ۰.۰۰۴۵۴;</pre>
نوع شمارشی	<ul style="list-style-type: none"> برای اسامی شمارشی نام واحدی بکار برید. <pre>public enum WarmColor { Orange, Yellow, Brown }</pre>
رویداد	<ul style="list-style-type: none"> متدی که رویدادها را هدایت می‌کند، باید پیشوند EventHandler داشته باشد. کلاس‌های آرگومان Event باید پیشوند EventArgs داشته باشند.
استثناء	<ul style="list-style-type: none"> پسوند Exception دارد.
واسط	<ul style="list-style-type: none"> پیشوند I را دارد. IDisposable
متغیرهای محلی	<ul style="list-style-type: none"> متغیرهایی که معرف public دارند، بصورت Pascal نوشته می‌شوند. <pre>int myIndex</pre>
متد	<ul style="list-style-type: none"> فعل یا عبارات فعلی را برای نامگذاری استفاده کنید.
فضای اسمی	<ul style="list-style-type: none"> نباید یک فضای اسمی و کلاس هم نام باشند. برای دوری از اسامی یکسان از پیشوندها استفاده کنید. برای مثال از نام شرکت به عنوان پیشوند برای فضاهای نامی پروژه‌های خود استفاده کنید. <pre>Arshia.GraphicsLib</pre>
خصوصیت	<ul style="list-style-type: none"> اسم یا عبارت اسمی بکار برید.
پارامتر	<ul style="list-style-type: none"> اسامی با معنی که هدف پارامتر را شرح می‌دهند، بکار برید.

همه موارد بالا از حالت Pascal استفاده می‌کنند به غیر از پارامتر و متغیر محلی که از حالت Camel استفاده می‌کنند.

توجه کنید که حالت یک نام می‌تواند بصورت زیر بر اساس دو طرح نوشتن حروف بزرگ باشد:

۱- Pascal: کاراکتر اول هر کلمه بزرگ نوشته می‌شود.

۲- Camel: به استثناء کلمه‌ی اول، حرف اول بقیه کلمات بزرگ نوشته می‌شود.

توضیحات در برنامه #C

کامپایلر #C سه نوع توضیح تعبیه شده را پشتیبانی می‌کند: یک نسخه Xml، توضیح یک خطی (//) و توضیحات چند خطی (/* */).

```
// for a single line
/* for one or more lines
*/
/// <remarks> XML comment describing a class </remarks>
```

یک توضیح Xml با سه خط کج (///) آغاز می‌شود و معمولاً شامل برچسب‌های XML است که یک قطعه کد همچون یک ساختار، یک کلاس یا عضو کلاس را مستندسازی می‌کند. #C می‌تواند برچسب‌های XML را برای بدست آوردن اطلاعات اضافی و صادر کردن^۱ آنها به یک فایل خارجی گسترش دهد.

برچسب <remarks> برای شرح دادن یک نوع داده استفاده می‌شود. کامپایلر #C هشت برچسب اصلی دیگر را تشخیص می‌دهد که به عنصر خاصی از برنامه تخصیص داده شده‌اند. این برچسب‌ها در بالای خطوط کد مربوطه قرار می‌گیرند.

جدول ۲-۲- برچسب‌های XML

برچسب	شرح
<example>	یک مثال از نحوه استفاده از ویژگی خاص برنامه مابین برچسب شروع و پایان نشان می‌دهد.
<exception cref="Excep">	خصوصیت cref نام استثناء را شامل است.
</exception>	///<exception cref="NoParmException">
<include file="myXML">	در خصوصیت file، نام یک فایل xml دیگر مشخص می‌شود که مستندسازی این کد در آن فایل قرار گرفته است.
<param name="parm\">	خصوصیت name نام پارامتر را شامل می‌شود.
<permission cref= ">	بیشتر مواقع دارای مقادیر زیر است:
</permission>	///<permission cref="System.Security.PermissionSet">
<returns>	یک توصیف متنی در مورد نوع مقدار برگشتی از یک متد یا خصوصیت را مشخص می‌کند.
<remarks>	اطلاعات اضافی در مورد نوع داده‌ای که در بخش <summary> نیست فراهم می‌کند.
<seealso cref="price">	در خصوصیت cref نام یک خصوصیت، متد یا هر عضو دیگر قرار می‌گیرد.
<summary>	توصیف یک کلاس را در بر می‌گیرد که به وسیله intellisense در VS.NET استفاده می‌شود

^۱ export

مقایر توضیحات XML در حقیقت برای آن است که می‌توانند به یک فایل XML مجزا صادر شوند و سپس به وسیله تکنیک‌های پارس کردن XML استاندارد پردازش شوند. کامپایلر بطور پیش‌فرض این کار را انجام نمی‌دهد، باید آن را تنظیم کرد.

خط زیر کد منبع ConsoleApp.cs را کامپایل کرده و یک فایل XML بنام ConsoleXML.xml ایجاد می‌کند.

```
C:\> csc consoleapp.cs /doc:consoleXML.xml
```

اگر کد شکل ۲-۱ را کامپایل کنید، هشدار زیر را برای همه اعضای public کدتان خواهید دید.

```
Warning CS۱۰۹۱: Missing XML comment for publicly visible type ...
```

برای منع این هشدار گزینه /nowarn:۱۵۹۱ را به خط کامپایل اضافه کنید. در این گزینه می‌توانیم شماره‌های مختلفی را با کاما از هم جدا کنیم.

۲-۲- انواع داده اولیه

سه بخش بعدی ویژگی‌هایی را شرح می‌دهند که در اکثر زبانهای برنامه نویسی خواهید یافت. متغیرها و انواع داده‌ها، عملگرها، عبارات و دستورات. این بحث با انواع داده اولیه شروع می‌شود که اینها انواع داده‌ای هسته‌ای #C برای ساختن کلاس‌ها یا ساختارهای پیچیده‌تر هستند. متغیرهای این نوع داده‌ها، یک مقدار منفرد دارند و اندازه آنها از پیش تعریف شده است. جدول ۲-۳ یک لیست رسمی از انواع داده اولیه فراهم می‌کند.

جدول ۲-۳- لیست انواع داده‌ی اولیه در #C

نوع داده اولیه #C	نوع داده FCL	توصیف داده
object	System.Object	نوع داده‌ی پایه برای هر نوع داده‌ی دیگر
string	System.String	یک دنباله از کارکترهای یونیکد
decimal	System.Decimal	اعداد اعشاری مختصر با ۲۸ رقم معنی‌دار
bool	System.Boolean	یک مقدار که به صورت true یا false نشان داده می‌شود.
char	System.Char	یک کارکتر یونیکد ۱۶ بیتی
byte	System.Byte	نوع داده صحیح ۸ بیتی بدون علامت
sbyte	System.SByte	نوع داده صحیح ۸ بیتی علامت‌دار
short	System.Int۱۶	عدد صحیح علامت‌دار ۱۶ بیتی
int	System.Int۳۲	عدد صحیح علامت‌دار ۳۲ بیتی
long	System.Int۶۴	عدد صحیح علامت‌دار ۶۴ بیتی
ushort	System.UInt۱۶	عدد صحیح بدون علامت ۱۶ بیتی
uint	System.UInt۳۲	عدد صحیح بدون علامت ۳۲ بیتی
ulong	System.UInt۶۰	عدد صحیح بدون علامت ۶۴ بیتی
(single(float	System.Single	عدد اعشاری
double	System.Double	عدد اعشاری با دقت مضاعف

همانطور که جدول نشان می‌دهد، انواع داده‌ای اولیه به ^۱BCL نگاشت شده‌اند، که می‌توانند به جای همدیگر استفاده شوند. دستورات زیر را ملاحظه کنید:

```
System.Int32 age = new System.Int32(۱۷);
int age = ۱۷;
System.Int32 age = ۱۷;
```

هر سه دستور، کد IL یکسانی تولید می‌کنند. نسخه‌ی کوتاه در C# از کلمه کلیدی `int` به عنوان نام مستعار^۲ برای نوع داده `System.Int32` استفاده می‌کند. C# برای همه انواع داده اولیه اسامی مستعار در نظر گرفته است.

چند نکته زیر را هنگام کار با انواع داده اولیه بخاطر داشته باشید:

- کلمات کلیدی که انواع داده‌ی مقداری اولیه را تعیین می‌کنند، در واقع اسامی مستعار یک ساختار معینی هستند. اعضای خاصی از این ساختارها برای دستکاری انواع داده اولیه بکار برده می‌شوند. برای مثال ساختار `Int32` فیلدی که بزرگترین مقدار صحیح ۳۲ بیتی را بر می‌گرداند و یک متد که رشته عددی را به یک عدد صحیح تبدیل می‌کند، را در بر دارد.

```
int iMax = int.MaxValue; // Return largest integer
int pVal = int.Parse("۱۰۰"); // converts string to int
```

کامپایلر C# تبدیل ضمنی را پشتیبانی می‌کند. اگر تبدیل یک تبدیل امن باشد، در نتیجه‌ی حاصله هیچ داده‌ای از دست نمی‌رود. این زمانی اتفاق می‌افتد که مقصد تبدیل دقت بالایی نسبت به شی مبداء داشته باشد، که تبدیل گسترش^۳ نام دارد. در تبدیل کاهشی^۴ که مقصد دقت کمتری دارد، باید عمل تبدیل به صورت صریح باشد، قالب‌بندی برای تبدیل یک مقدار از یک نوع به یک نوع دیگر استفاده می‌شود. این عمل با قرار دادن نوع داده مقصد در داخل پرانتزهایی قبل از مقدار مورد نظر انجام می‌شود.

```
short i۱۶ = ۵۰; // ۱۶-bit integer
int i۳۲ = i۱۶; // Okay: int has greater precision
i۱۶ = i۳۲; // Fails: short is ۱۶ bit, int is ۳۲
i۱۶ = (short) i۳۲; // Okay since casting used
```

- مقادیر حرفی^۵ تخصیص یافته به انواع داده‌ی `float`، `decimal` یک حرف به دنبال مقدار مورد نظر نیاز دارند. `Float` به `F` یا `f`، `double` به `D` یا `d` و `decimal` به `M` یا `m` نیاز دارد.

- `decimal pct = .۱۰M; // M is required for literal value`

بقیه این بخش مروری بر مفیدترین انواع داده اولیه به استثناء `String` دارد.

Decimal

نوع داده‌ی `decimal` یک عدد اعشاری ۱۲۸ بیتی با دقت بسیار بالا است. دقت آن ۲۸ رقم اعشار است و در محاسبات مالی که گرد کردن قابل تحمل نیست، بکار برده می‌شود. این مثال سه متد برای نوع داده `decimal` ارائه می‌دهد.

```
decimal iRate = ۳,۹۸۳۴M; // decimal requires M
iRate = decimal.Round(iRate,۲); // Returns ۳,۹۸
decimal dividend = ۵۱۲,۰M;
decimal divisor = ۵۱,۰M;
decimal p = decimal.Parse("۱۰۰,۰۵");
// Next statement returns remainder = ۲
decimal rem = decimal.Remainder(dividend,divisor);
```

bool

^۱ Base Class Library

^۲ Alias

^۳ Widening conversion

^۴ Narrowing conversion

^۵ literal

false و true تنها مقادیر ممکن یک نوع داده‌ی bool هستند. قالب‌بندی یک مقدار bool به یک عدد صحیح ممکن نیست. برای مثال تبدیل true به ۱ یا قالب‌بندی ۱ یا ۰ به یک مقدار bool ممکن نیست.

```
bool bt = true;
string bStr = bt.ToString(); // returns "true"
bt = (bool) ۱; // fails
```

Char

نوع داده char یک کاراکتر یونیکد^۱ ۱۶ بیتی را نشان می‌دهد و به صورت یک عدد صحیح بدون علامت پیاده‌سازی می‌شود. یک نوع داده char، عملیات انتساب متعددی را می‌پذیرد: یک مقدار کاراکتری مابین دو علامت تک کوتیشن (` `)، یک مقدار عددی قالب‌بندی شده، یک دنباله escape. همانطور که مثال زیر نشان می‌دهد، ساختار System.char تعدادی متد مفید فراهم می‌کند.

```
myChar = 'B'; // 'B' has an ASCII value of ۶۶
myChar = (char) ۶۶; // Equivalent to 'B'
myChar = '\u۰۰۴۲'; // Unicode escape sequence
myChar = '\x۰۰۴۲'; // Hex escape sequence
myChar = '\t'; // Simple esc sequence:horizontal tab
bool bt;
string pattern = "\۲۳abcd?";
myChar = pattern[۰]; // '\۲'
bt = char.IsLetter(pattern,۳); // true ('a')
bt = char.IsNumber(pattern,۳); // false
bt = char.IsLower(pattern,۰); // false ('\۲')
bt = char.IsPunctuation(pattern,۷); // true ('?')
bt = char.IsLetterOrDigit(pattern,۱); // true
bt = char.IsNumber(pattern,۲); // true ('۳')
string kstr="K";
char k = char.Parse(kstr);
```

Byte , sbyte

یک نوع داده‌ی byte عدد صحیح بدون علامت با مقداری از ۰ تا ۲۵۵ است و sbyte عدد صحیح ۸ بیتی علامت‌دار با مقداری از ۱۲۸- تا ۱۲۷ است.

```
byte[] b = {۰x۰۰, ۰x۱۲, ۰x۳۴, ۰x۵۶, ۰xAA, ۰x۵۵, ۰xFF};
string s = b[۴].ToString(); // returns ۱۷۰
char myChar = (char) b[۳];
```

Short و long

اینها به ترتیب اعداد صحیح علامت‌دار ۱۶ و ۳۲ و ۶۴ بیتی را نشان می‌دهند. نسخه‌های بدون علامت آنها به ترتیب ushort و uint و ulong هستند.

```
short i۱۱ = ۲۰۰;
i۱۱ = ۰xCA; // hex value for ۲۰۰
int i۲۲ = i۱۱; // no casting required
```

double و single

اینها قالب‌های ۳۲ بیتی با دقت معمولی و ۶۴ بیتی دقت مضاعف را نمایش می‌دهند. در NET.X نوع داده‌ی Single معادل float است.

- نوع داده single یک مقدار در محدوده‌ی ۱,۵*۱۰^{-۴۵} تا ۳,۴*۱۰^{۳۸} با ۷ رقم اعشار دارد.
- نوع داده‌ی double یک مقدار در محدوده‌ی ۵*۱۰^{-۳۲۴} تا ۱,۷*۱۰^{۳۰۸} با ۱۵ الی ۱۶ رقم اعشار دارد.
- عملیات ممیز شناور برای نشان‌دادن اینکه نتیجه‌ی یک محاسبات تعریف نشده است، Nan برمی‌گردانند. تقسیم ۰,۰ بر ۰,۰ NaN است.

^۱ unicode

- متد System.Convert را برای تبدیل یک نوع داده‌ی ممیز شناور به هر نوع دیگر بکار ببرید.

```
float xFloat = ۲۴۵۱۷,۱۱F;
int xInt = Convert.ToInt۳۲(xFloat); // returns ۲۴۵۱۷
int xInt۲ = (int) xFloat;
if(xInt == xInt۲) { } // False
string xStr = Convert.ToString(xFloat);
single zero = ۰;
if (Single.IsNaN(۰ / zero)) { } // True
double xDouble = ۱۲۴,۵۱D;
```

متدهای Parse و TryParse را برای تبدیل یک رشته عددی به نوع داده‌ی مشخص به کار می‌برند.

```
short shParse = Int۱۶.Parse("\۰۰");
int iParse = Int۳۲.Parse("\۰۰");
long lparse = Int۶۴.Parse("\۰۰");
decimal dParse = decimal.Parse("۹۹,۹۹");
float sParse = float.Parse("۹۹,۹۹");
double dbParse = double.Parse("۹۹,۹۹");
```

TryParse در NET ۲,۰، پارس کردن شرطی را فراهم می‌کند. برای تعیین موفق بودن عمل پارس یک پارامتر بولین بر می‌گرداند، که روشی برای دوری از کد کنترل رسمی استثناء فراهم می‌سازد.

```
int result;
// parse string and place result in result parameter
bool ok = Int۳۲.TryParse("\۰۰", out result);
bool ok = Int۳۲.TryParse("\۰۰", NumberStyles.Integer, null, out result);
```

در فرم دوم از این متد، اولین پارامتر یک رشته عددی مورد نظر جهت پارس کردن، پارامتر دوم نوع عدد در رشته را مشخص می‌کنند. مقدار بازگشتی در پارامتر چهارم قرار می‌گیرد.

۲-۳- عملگرهای ریاضی، منطق و شرطی

عملگرهای C# برای عملیات ریاضی، دستکاری بیت‌ها و کنترل شرطی برنامه استفاده می‌شوند، که باید همه برنامه‌نویسان با آنها آشنا باشند.

۲-۳-۱- عملگرهای ریاضی

جدول ۲-۴ عملگرهای عددی پایه را خلاصه می‌کند. اولویت این عملگرها در حین ارزیابی یک عبارت در پرانتزها اعمال می‌شود. ۱ بالاترین سطح اولویت است.

جدول ۲-۴

مثال	توصیف	عملگر
<code>int x = y + ۱۰;</code>	جمع	<code>(۳) +</code>
	تفریق	<code>-</code>
<code>int x = ۱۰;</code> <code>int y = ۱۵;</code>	ضرب	<code>(۲) *</code>
<code>int z = x * y / ۲; // ۴۵۰</code>	تقسیم	<code>/</code>
<code>y = x % ۲۹; // remainder is ۲</code>	باقیمانده	<code>%</code>
<code>x = ۵;</code> <code>Console.WriteLine(x++) // x = ۵</code> <code>Console.WriteLine(++x) // x = ۶</code>	افزایش / کاهش پیشوندی/پسوندی	<code>(۱) ++</code> <code>--</code>
<code>int x = ~۱۲۷; // returns -۱۲۸</code>	مکمل بیتی	<code>(۱) ~</code>

byte x = ۱۰; // binary ۱۰ is ۰۱۱۰	شیفت به راست	<< (۴)
int result = x << ۱; // ۲۰ = ۱۰۱۰۰		
result = x >> ۲; // ۵ = ۰۰۱۰۱	شیفت به چپ	>>
byte x = ۱۲; // ۰۰۱۱۰۰	And بیتی	& (۵)
byte y = ۱۱; // ۰۰۱۰۱۱		
int result = x & y; // ۸ = ۰۰۱۰۰۰	Or بیتی	(۶)
result = x ^ y; // ۷ = ۰۰۰۱۱۱	Xor بیتی	^ (۷)

عملوندهای این عملگر ها باید صحیح باشند.

توجه: #C عملگر توان ندارد. در عوض، متد *Math.Power* را برای رساندن یک عدد به توان بکار می‌برند و متد *Math.Exp()* را به توان می‌رساند.

۲-۳-۲- عملگرهای شرطی و رابطه‌ای

عملگرهای رابطه‌ای برای مقایسه دو مقدار و تعیین رابطه مابین آنها استفاده می‌شوند. آنها عموماً در ارتباط با عملگرهای شرطی، برای ایجاد ساختارهای تصمیم‌گیری پیچیده استفاده می‌شوند. جدول ۲-۵ خلاصه‌ای از عملگرهای رابطه‌ای و شرطی #C را تهیه می‌کند.

جدول ۲-۵- عملگرهای رابطه‌ای و شرطی

مثال	توصیف	عملگر
if (x == y) {...}	مساوی	==
	نامساوی	!=
if (x <= y) {...}	کوچکتر	>
	کوچکتر یا مساوی	>=
	بزرگتر	<
	بزرگتر یا مساوی	<=
if (x == y && y < ۲۰) {...}	And منطقی	&&
اگر عبارت اول false باشد، عبارت دوم ارزیابی نمی‌شود.	Or منطقی	
if (x== y y < ۲۰) {...}	And منطقی	&
همیشه عبارت دوم ارزیابی می‌شود.	Or منطقی	
if !(x ==y && y < ۲۰) {...}	نقیض منطقی	!

به دو شکل عملیات منطقی AND/OR توجه کنید. در عملگرهای || یا &&، اگر عبارت اول به ترتیب true یا false باشد، دومی ارزیابی نمی‌گردد. (تکنیک ارزیابی کوتاه)

عملگرهای & و | همواره هر دو عبارت را ارزیابی می‌کنند. اینها زمانی استفاده می‌شوند که مقادیر عبارات از یک تابع برمی‌گردند و می‌خواهیم از فراخوانی متدها مطمئن شویم.

علاوه بر عملگرهای جدول ۲-۵، #C برای انتساب شرطی یک مقدار به یک متغیر عملگر ?: را پشتیبانی می‌کند. همانطور که این مثال نشان می‌دهد آن ساده شده دستور if else است.

```
string pass;
int grade=۷۴;
If(grade >= ۷۰) pass="pass"; else pass="fail";
// expression ? op\ : op۲
pass = (grade >= ۷۰) ? "pass" : "fail";
```

اگر عبارت true باشد، عملگر ! مقدار اول را بر می‌گرداند، در غیر این صورت مقدار دوم برگردانده می‌شود.

۲-۴- راهنماهای پیش پردازش C#

این راهنماها، دستوراتی هستند که در مرحله تحلیل حرفی توسط کامپایلر C# خوانده می‌شوند. آنها می‌توانند کامپایلر را برای در برگرفتن یا ننگرفتن یک تکه کد یا حتی رد عمل کامپایل بر اساس مقادیر راهنماهایی پیش پردازش^۲ تذکر دهند. یک راهنمای پیش پردازش با کارکتر # در ابتدای خط معین می‌شود. فضاهای خالی قبل و بعد از سمبل # مجاز هستند. جدول ۲-۶ راهنماهایی از C# را لیست می‌کند.

جدول ۲-۶- راهنماهای پیش پردازش C#

سمبل پیش پردازش C#	توصیف
define#	برای تعریف و حذف یک سمبل بکار می‌رود. با تعریف یک سمبل، در صورتی که در راهنمای
undef#	if# استفاده شود، آن به true ارزیابی می‌گردد.
if#	شبیه if-else در C# عمل می‌کند.
	#if #elif #else #endif
line#	شماره ترتیب خط را تغییر می‌دهد و می‌تواند تعیین کند منبع این خط کدام فایل است.
region#	برای تعیین یک بلاک کد استفاده می‌شود که به هنگام کد نویسی در محیط IDE مربوط به
endregion#	VS۲۰۰۵ می‌توان این بلاک را جمع کرده یا باز کرد.
error#	باعث می‌شود کامپایلر یک خطای مخرب (بزرگ) گزارش دهد.
warning#	باعث می‌شود کامپایلر یک هشدار گزارش داده و سپس پردازش را ادامه دهد.

از عمومی ترین کاربردهای راهنماهای پیش پردازش، انجام کامپایل شرطی، اضافه کردن تشخیص‌ها برای گزارش خطاها و هشدارها و تعریف ناحیه‌های کد است.

۲-۴-۱- کامپایل شرطی

با استفاده از راهنماهای مرتبط با if# می‌توان تعیین کرد، کدام قسمت از کد در طول کامپایل در گرفته شود. هر کدی که مابین دستور if# و دستور endif# قرار دارند، بر اساس اینکه شرط if به true یا false ارزیابی می‌شود، در هنگام کامپایل در برگرفته می‌شوند یا در برگرفته نمی‌شوند. این یک ویژگی قدرتمند، اغلب برای اهداف رفع اشکال استفاده می‌شوند. حال یک مثال از این مفهوم ارائه می‌گردد.

```
#define DEBUG
using System;
public class MyApp
{
```

^۱ Directive

^۲ Preprocess

فصل دوم- اصول #C

```
public static void Main()
{
    #if (DEBUG)
    Console.WriteLine("Debug Mode");
    #else
    Console.WriteLine("Release Mode");
    #endif
}
```

همه راهنماهای #define بایستی در آغاز فایل cs. قرار گیرند. سمبل کامپایل شرطی دو حالت دارد: تعریف شده یا تعریف نشده. در این مثال سمبل DEBUG تعریف می‌شود و دستور #DEBUG (if) به true ارزیابی می‌شود. کاربرد واضح راهنمای #define کنترل حالت Debug هر فایل منبع را مجاز می‌دارد. اگر VS ۲۰۰۵ را بکار می‌برید شما می‌توانید یک تولید برنامه بصورت Debug را تعیین کنید، در نتیجه سمبل DEBUG برای هر فایل در پروژه به طور اتوماتیک تعریف می‌گردد. هیچ راهنمای #define صریح لازم نیست.

در خط فرمان کامپایل #C نیز می‌توان با استفاده از سوییچ /Define یک سمبل را تعریف کرد.

```
csc /Define:DEBUG myproject.cs
```

کامپایل کردن با این دستور، معادل این است که دستور #DEBUG Define در کد منبع نوشته شده باشد.

۲-۴-۲- راهنماهای تشخیص

راهنماهای تشخیص، پیام‌های هشدار و خطا می‌فرستند که شبیه دیگر خطاها و هشدارهای فرمان کامپایل با آنها رفتار می‌شود. راهنمای #warning ادامه کامپایل را مجاز می‌دارد، در حالیکه #error آن را متوقف می‌کند.

```
#define CLIENT
#define DEBUG
using System;
public class MyApp
{
    public static void Main()
    {
        #if DEBUG && INHOUSE
        #warning Debug is on.
        #elif DEBUG && CLIENT
        #error Debug not allowed in Client Code.
        #endif
        // Rest of program follows here
    }
}
```

در این مثال، کامپایل با یک پیام خطا پایان خواهد داد، چون CLIENT و DEBUG تعریف می‌شوند.

ناحیه‌های کد

راهنماهای ناحیه برای نشانه‌گذاری قسمتهایی از کد به عنوان ناحیه استفاده می‌شوند. راهنمای ناحیه برای کامپایلر #C هیچ معنایی ندارند، اما به وسیله VS.NET تشخیص داده می‌شوند.

```
#region
// any C# statements
#endregion
```

۲-۵- نوع داده‌ی شمارشی

نوع داده شمارشی که در #C به enum معروف است، یک روش مناسب برای ایجاد یک مجموعه ساخت‌یافته^۱ از سمبل‌ها جهت نمایش مقادیر ثابت پیشنهاد می‌کند.

گرامر:

^۱ Structured

```
[access modifiers]enum <identifier> [:enum-base]{enum body}
```

مثال:

```
enum Fabric :short {
    Cotton = ۱,
    Silk = ۲,
    Wool = ۴,
    Rayon = ۸,
    Other = ۱۲۸
}
```

توجه: اگر سمبل‌های enum به یک مقدار خاصی تنظیم نشده باشند، آنها به طور اتوماتیک با دنباله ۰ و ۳ و ۰ و ۳ تنظیم می‌شوند.

معرف‌های دسترسی، دامنه‌ی enum را تعریف می‌کنند. به طور پیش فرض internal است و اجازه می‌دهد فقط کلاس‌های داخل آن اسمبلی به آن دسترسی داشته باشند. معرف public آن را برای هر کلاسی در هر اسمبلی قابل دسترسی می‌سازد. گزینه‌ی اختیاری enum-base نوع داده پایه ثابت‌ها را بر اساس مقادیر تخصیص یافته به آنها مشخص می‌کند. آن فقط می‌تواند یکی از انواع صحیح byte, sbyte, short, ushort, int, unit, long یا ulong باشند. به طور پیش فرض int است.

۲-۵-۱- کار با نوع داده شمارشی

انواع شمارشی نه تنها خوانایی برنامه را بهتر می‌کنند، بلکه هنگام تغییر مقدار اصلی، تغییرات کد را نیز کاهش می‌دهند و همه ارجاعات به آن مقدار، معتبر می‌مانند. مزیت دیگر اینکه انواع شمارشی، نوع داده قوی هستند. بدین معنی که هر وقت یک نوع enum به عنوان پارامتر ارسال شود، متد دریافت کننده باید یک پارامتر مطابق با همان نوع داشته باشد. در غیر اینصورت کامپایلر خطا رخ می‌دهد. قطعه کد ۲-۲ این اهداف را با استفاده از نوع داده شمارشی Fabric نشان می‌دهد.

قطعه کد ۲-۲

```
static double GetPrice(Fabric fab)
{
    switch(fab)
    {
        case Fabric.Cotton: return(۳,۰۰);
        case Fabric.Silk: return(۵,۶۰);
        case Fabric.Wool: return(۴,۰۰);
        case Fabric.Rayon: return(۳,۲۰);
        case Fabric.Other: return(۲,۰۰);
        default: return(۰,۰);
    }
}
static void Main()
{
    Fabric fab = Fabric.Cotton;
    int fabNum = (int) fab; // \
    string fabType = fab.ToString(); // "Cotton"
    string fabVal = fab.ToString("D"); // "\
    double cost = GetPrice(fab); // ۳,۰۰
}
```

نکات:

- برای مقداردهی enum به یک مقدار صحیح، قالب‌بندی آن لازم است.
- با متد ToString() و پارامتر "D" می‌توان مقدار رشته‌ای یک ثابت را بدست آورد.
- در هنگام ارسال یک نمونه از نوع شمارشی Fabric به متد GetPrice لازم است پارامتر متد از همان نوع اعلان شده باشد.

این مثال نشان می‌دهد که نحوه‌ی به دست آوردن نام سمبل یا مقدار ثابت نوع شمارشی ساده است. زمانی که نمونه‌ای از یک نوع داده شمارشی مشخص باشد، به راحتی می‌توان اعضاء یک نوع داده شمارشی را به کمک کلاس System.Enum و حلقه foreach به دست آورد.

۲-۵-۲- متدهای System.Enum

متدهای Enum.Parse، Enum.IsDefined و Enum.GetName، سه متد مفید System.Enum هستند. اغلب دو متد اول با هم به کار می‌روند تا مشخص کنند آیا یک مقدار یا سمبل عضو یک enum است و یک نمونه از آن ایجاد می‌کنند. برای فهم بهتر مطلب، مثال‌های زیر را در نظر بگیرید. در این مثال نوع شمارشی Fabric برای وجود یک مقدار رشته‌ای در آن جستجو می‌شود و متد GetName یکی از مقادیر آن را چاپ می‌کند.

```
string fabStr = "Cotton";
// Determine if symbol Cotton exists in Fabric enum
if (Enum.IsDefined(typeof(Fabric), fabStr))
{
    // Create enum instance
    Fabric fab = (Fabric)Enum.Parse( typeof(Fabric) , fabStr);
    // Output from the following statement is: "Silk"
    Console.WriteLine("Second value of Fabric Enum is: " + Enum.GetName(typeof(Fabric), ۲));
}
```

متد IsDefined دو پارامتر می‌گیرد: یک نوع شمارشی که عملگر typeof بر می‌گرداند و یک رشته که سمبل مورد نظر جهت تست را نشان می‌دهد. شکل دیگر این متد با مقدار عددی برای پارامتر دوم است.

متد Parse همان پارامترهای IsDefined را می‌گیرد و یک نمونه از نوع داده‌ی شمارشی ایجاد می‌کند. متغیر fab ایجاد شده در اینجا معادل همان متغیر قطعه کد ۲-۲ است. اطمینان از وجود عضو enum، قبل از کاربرد متد Parse مهم است، اگر آن عضو وجود نداشته باشد یک استثناء روی می‌دهد.

متد GetName یک مقدار رشته‌ای از enum که مقدار آن در پارامتر دوم ارسال می‌شود بر می‌گرداند. در این مثال "Silk" برگردانده می‌شود، چون مقدار ثابت آن ۲ است.

۲-۵-۳- انواع شمارشی و flagهای بیتی

تنظیم مقادیر نوع شمارشی Fabric با توان‌هایی از ۲ تصادفی نیست. بیشتر موارد، اعضاء نوع شمارشی در عملیات منطقی استفاده می‌شوند. این مقادیر در نگاشت به مقادیر بیتی منحصر به فرد مفید هستند. ممکن است ترکیبی از این مقادیر را در کدنویسی تعیین کنید.

```
Fabric cotWool = Fabric.Cotton | Fabric.Wool;
Console.WriteLine(cotWool.ToString()); // Output: ۰
```

خروجی در صورتی بامعنی خواهد بود، که آن ترکیبی از Cotton و Wool باشد. می‌توان این کار را با اضافه کردن صفت [Flags] به اعلان نوع شمارشی انجام داد.

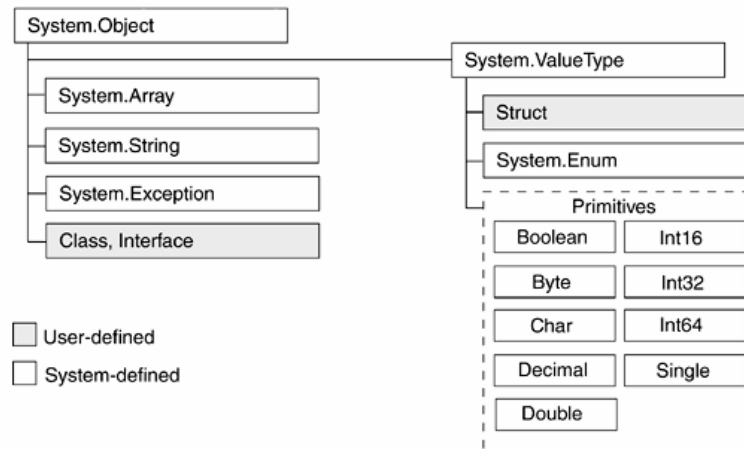
```
[Flags]
enum Fabric :short {
```

متد ToString() اعلان نوع شمارشی را بررسی می‌کند. اگر این صفت موجود باشد، آن نوع شمارشی را به صورت یک مجموعه از اعضاء نگاشت بیتی در نظر می‌گیرد. در این مثال نمی‌توان یک مقدار سمبلیک معادل ۵ پیدا کرد، پس متد ToString() الگوی بیتی "۱۰۱" را بکار می‌برد و سمبل‌هایی که الگوهای بیتی "۰۰۱" و "۱۰۰" دارند را چاپ می‌کند. خروجی جدید یک لیست جدا شده با کاما است. "cotton, wool"

۲-۶-انواع داده‌ی مقداری و ارجاعی

CLR دو نوع داده را پشتیبانی می‌کند: انواع مقداری و انواع ارجاعی (شکل ۲-۲). انواع داده‌ی ارجاعی شامل کلاس‌ها، آرایه‌ها، واسط‌ها و نماینده‌ها هستند. انواع داده‌ی مقداری شامل انواع داده‌ی اولیه همچون `int`، `char`، `byte` و همچنین انواع داده‌ی `enum` و `struct` هستند. انواع داده‌ی ارجاعی و مقداری به وسیله محل‌شان در سلسله مراتب کلاس .NET و روش تخصیص حافظه متمایز می‌شوند.

شکل ۲-۲- سلسله مراتب کلاس‌های .NET



۲-۶-۱- System.ValueType و System.Object

هر دو نوع داده‌ی مقداری و ارجاعی از کلاس `System.Object` ارث‌بری می‌کند. تفاوت این است که انواع داده‌ی ارجاعی مستقیماً از آن ارث‌بری می‌کنند. در حالی که انواع داده‌ی مقداری از کلاس `System.ValueType` ارث‌بری می‌کنند.

`System.Object` به عنوان پایه‌ی همه‌ی انواع داده، یک مجموعه متد فراهم می‌کند، که می‌توانید در هر نوع داده بیابید. متد `ToString()` عضوی از این مجموعه است. همچنین متدهای ایجاد یک کپی از یک نوع داده و یک کد درهم‌سازی منحصر به فرد برای یک نوع داده و مقایسه دو نمونه از یک نوع داده نیز وجود دارد.

`System.ValueType` از `System.Object` ارث‌بری می‌کند و هیچ عضوی اضافه نمی‌کند. اما یک سری از متدهای ارث‌بری شده را جهت ایجاد تناسب با انواع داده‌ی مقداری `override` می‌کند. برای مثال، متد `Equals()` برای مقایسه‌ی دو شی هم‌نوع `override` می‌شود. بنابه تعریف، همه‌ی انواع داده‌ی مقداری به صورت ضمنی از کلاس `ValueType` ارث‌بری می‌کنند.

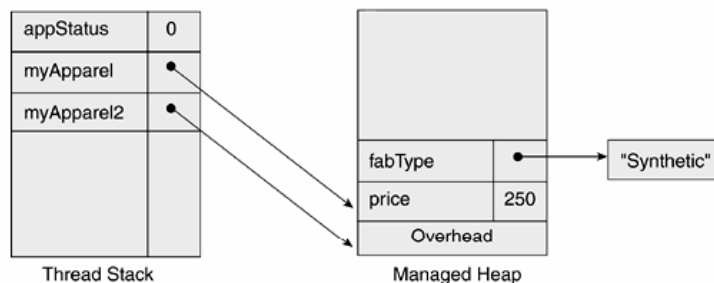
۲-۶-۲- تخصیص حافظه برای انواع داده مقداری و ارجاعی

تفاوت اصلی انواع داده مقداری و ارجاعی در روش مدیریت CLR برای نیازهای حافظه‌ای آنها می‌باشد. در زمان اجرا به انواع داده مقداری روی یک `Stack` حافظه تخصیص داده می‌شود و انواع داده ارجاعی روی یک `Heap` مدیریت شده قرار می‌گیرند که از `Stack` ارجاع داده می‌شوند.

شکل ۲-۳ نحوه تخصیص حافظه را برای انواع داده‌ی ارجاعی و مقداری نشان می‌دهد و اینکه زمان ایجاد یک نمونه از یک نوع داده ارجاعی و تخصیص آن به یک متغیر دومی چه اتفاقی می‌افتد را دنبال کنید.

```
Apparel myApparel = new Apparel();
Apparel myApparel۲ = myApparel;
```

شکل ۳-۲- نحوه‌ی تخصیص حافظه



۱- CLR در بالای Heap مدیریت شده به شی مورد نظر حافظه تخصیص می‌دهد.

۲- اطلاعات سربار شی به Heap اضافه می‌شوند. این اطلاعات شامل یک اشاره‌گر به جدول متد شی و یک SyncBlockIndex است که برای همزمان‌سازی دسترسی به شی مابین چند متد استفاده می‌شود.

۳- شی myApparel به عنوان یک نمونه از کلاس Apparel ایجاد می‌شود و فیلدهای Price و FabType آن روی Heap قرار می‌گیرند.

۴- ارجاع به myApparel روی Stack قرار می‌گیرد.

۵- زمانی که یک متغیر ارجاع جدید myApparel₂ ایجاد می‌شود، آن روی Stack قرار می‌گیرد و یک اشاره‌گر به شی موجود می‌دهد. هر دو متغیر ارجاعی myApparel و myApparel₂ به شی یکسانی اشاره می‌کنند.

ایجاد یک شی ارجاعی به دلیل داشتن چندین مرحله و سربار اضافی مربوط به زمان و منابع، گران هستند. با این وجود، تنظیم ارجاع‌های اضافی به یک شی موجود کاملاً کارآمد است، چون نیازی به ایجاد یک کپی فیزیکی از یک شی نیست. عکس آن برای انواع داده مقداری صحیح است.

۲-۶-۳- جعبه‌بندی^۱

.NET یک نوع داده خاص Object دارد که مقادیر هر نوع داده‌ای را می‌پذیرد. در صورتیکه نوع یک مقدار مشخص نباشد، آن یک روش کلی برای ارسال پارامترها و انتساب مقادیر فراهم می‌کند. با هر چیز منتسب شده به Object بایستی به صورت یک نوع داده ارجاعی رفتار شود و روی Heap ذخیره می‌گردد. دستورات زیر را ملاحظه نمایید:

```
int age = ۱۷;
object refAge = age;
```

دستور اول متغیر age را ایجاد می‌کند و مقدار آن را روی Stack قرار می‌دهد. دستور دومی مقدار age را به یک نوع داده‌ی ارجاعی منتسب می‌کند. آن دستور مقدار ۱۷ را روی Heap قرار می‌دهد و اشاره‌گرهای سربار را اضافه می‌کند و در Stack یک ارجاع به آن اضافه می‌کند، این عمل را جعبه‌بندی گویند. تبدیل یک نوع داده ارجاعی به یک نوع داده مقداری^۲ از جعبه در آوردن^۲ گفته می‌شود و با قالب‌بندی یک شی به نوع داده اصلی آن انجام می‌شود. حال شی ایجاد شده در مثال قبلی را از جعبه در می‌آوریم.

```
int newAge = (int) refAge;
string newAge = (string) refAge; // Fails. InvalidCastException
```

^۱ Boxing

^۲ Unboxing

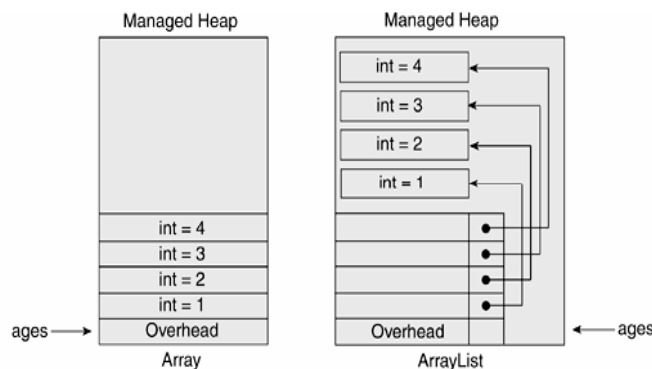
توجه داشته باشید مقداری که از جعبه درآورده می‌شود، باید از همان نوع داده قالب‌بندی شده باشد. در کل، جعبه‌بندی می‌تواند صرف‌نظر شود، چون CLR جزئیات را بطور شفاف کنترل می‌کند. با این وجود، زمان طراحی کد، ذخیره‌ی مقدار زیادی از داده‌های عددی در حافظه را بررسی کنید. برای این منظور کلاس‌های `System.Array` و `ArrayList` را ملاحظه کنید. هر دو از انواع داده ارجاعی هستند، اما از نظر ذخیره مقادیر داده‌ای ساده کاملاً متفاوت عمل می‌کنند.

متدهای `ArrayList` برای کار روی نوع داده کلی `Object` طراحی شده‌اند. در نتیجه، `ArrayList` همه عناصر را به صورت انواع داده ارجاعی ذخیره می‌کند. اگر داده‌ی مورد نظر جهت ذخیره، یک نوع داده‌ی مقداری باشد، قبل از ذخیره باید جعبه‌بندی شود. از طرف دیگر آرایه می‌تواند هر دو نوع داده‌ی مقداری و ارجاعی را نگه دارد. آن با انواع داده ارجاعی همانند `ArrayList` رفتار می‌کند، ولی انواع داده مقداری را جعبه‌بندی نمی‌کند.

کد زیر یک آرایه و یک `ArrayList` از مقادیر صحیح ایجاد می‌کند. همانطور که در شکل ۲-۴ نشان داده شده است، مقادیر در حافظه کاملاً با روش متفاوتی ذخیره می‌شوند.

```
// Create array with four values
int[] ages = {۱,۲,۳,۴};
// Place four values in ArrayList
ArrayList ages = new ArrayList();
for (int i=۰; i<۴; i++)
{
    ages.add(i); // expects object parameter
}
```

شکل ۲-۴ - مقایسه‌ی `Array` و `ArrayList` در حافظه



متغیر آرایه، مقادیر را به صورت مقادیر `int` ذخیره می‌کند. `ArrayList` هر مقداری را جعبه‌بندی می‌کند. آن سپس سربرار مورد نیاز بوسیله انواع داده‌ی ارجاعی را اضافه می‌کند. اگر برنامه کاربردی شما مقدار زیادی داده در حافظه ذخیره می‌کند و ویژگی‌های خاص `ArrayList` را نیاز ندارد، آرایه یک پیاده‌سازی کاراتری است. اگر از .NET ۲.۰ استفاده می‌کنید، کلاس `List` بهترین انتخاب است، چون جعبه‌بندی را حذف می‌کند و ویژگی‌های انعطاف‌پذیر `ArrayList` را شامل است.

نکته: زمانی که یک متغیر از میدان خارج می‌گردد، حافظه‌ی آن از روی `Stack` آزاد می‌شود. در صورتی که حد مشخص حافظه سیستم استفاده گردد، GC حافظه‌ی آشغال روی `Heap` را آزاد می‌کند. GC به وسیله‌ی .NET کنترل می‌گردد و بطور اتوماتیک در فاصله‌های غیرقابل پیش‌بینی اجرا می‌گردد.

۲-۷- فضاهای نامی

همانطور که می‌دانید، فضاهای نامی یک روش برای سازمان‌دهی کلاس‌های مرتبط و انواع داده‌ای دیگر فراهم می‌کند. برخلاف یک فایل یا یک قطعه، فضای نامی به جای گروه‌بندی فیزیکی، یک گروه‌بندی منطقی انجام می‌دهد. زمانی که یک کلاس تعریف می‌کنند، می‌توانید آن را در یک تعریف فضای نامی قرار دهید. سپس اگر بخواهید یک کلاس دیگر در فایل دیگری تعریف کنید که کار مرتبط با آن انجام می‌دهد، می‌توانید آن را در همان فضای نامی قرار دهید. ایجاد یک گروه‌بندی منطقی، تشخیص کلاس‌های مورد استفاده را برای توسعه‌دهندگان دیگر ساده می‌کند.

#C فصل دوم - اصول

```
namespace CustomerPhoneBookApp
{
    using System;
    public struct Subscriber
    {
        // Code for struct here...
    }
}
```

قراردادن یک نوع داده در یک فضای نامی، یک نام طولانی به آن نوع داده می‌دهد که شامل چندین اسم جدا شده با نقطه می‌باشد، که در انتها نام کلاس قرار می‌گیرد. در مثال قبلی نام کامل ساختار Subscriber، CustomerPhoneBookApp.Subscriber است. این عمل کاربرد کلاسهای هم نام در فضاهای نامی مختلف را در یک برنامه بدون هیچ ابهامی مجاز می‌دارد.

می‌توانید فضاهای نامی تودرتو را ایجاد کنید و یک ساختار سلسله مراتبی برای انواع داده‌ای خودتان بسازید.

```
namespace Arshia
{
    namespace ProCSharp
    {
        namespace Basics
        {
            class NamespaceExample
            {
                // Code for the class here...
            }
        }
    }
}
```

نام هر فضای نامی از نام های فضاهای نامی که آن را در بر می‌گیرند تشکیل شده است، که با نقطه از هم جدا شده‌اند. با نام بیرونی ترین فضای نامی شروع شده و به نام کوتاه خودش ختم می‌شود.

پس نام کامل فضای نامی ProCSharp، Arshia.ProCSharp است و نام کامل کلاس NamespaceExample، Arshia.ProCSharp.Basics.NamespaceExample است.

می‌توانید در تعاریف فضای نامی خودتان، این قاعده را برای سازمان‌دهی فضاهای نامی بکار ببرید. بنابراین کد قبلی می‌تواند بصورت زیر نوشته شود.

```
namespace Arshia.ProCSharp.Basics
{
    class NamespaceExample
    {
        // Code for the class here...
    }
}
```

توجه: اعلان یک فضای نامی چندبخشی به صورت تو در تو، در داخل فضای اسمی دیگر مجاز نیست.

فضاهای نامی به اسمبلی‌ها مرتبط نیستند. در یک اسمبلی می‌توان فضاهای نامی مختلفی تعریف کرد یا در فضاهای نامی انواع داده‌ای تعریف کرد که در اسمبلی‌های مختلفی قرار دارند.

۲-۷-۱- دستور using

همانطور که قبلاً گفته شد، #C اختصار نوشتن نام کامل یک کلاس را اجازه می‌دهد. برای انجام این کار، فضای نامی کلاس را در بالای فایل با کلمه کلیدی using به برنامه ربط دهید. در ادامه‌ی فایل می‌توانید به انواع داده‌ای موجود در آن فضای نامی بوسیله نام نوع داده ارجاع کنید.

```
using System;
using Arshia.ProCSharp;
```

بطور مجازی همه کدهای منبع #C با دستور `using System` شروع می‌شوند. چون بیشتر کلاس‌های مفید تهیه شده توسط مایکروسافت در فضای اسمی `System` دربرگرفته شده‌اند.

اگر در فضاهای نامی ارجاع شده با دستورات `using` یک نوع داده هم نام داشته باشند، شما نام کامل را برای آنها استفاده خواهید کرد تا مطمئن شوید کامپایلر کدام نوع داده را دستیابی می‌کند. برای مثال، فرض کنید کلاس‌هایی بنام `NamespaceExample` در فضاهای نامی `Arshia.ProCSharp.Basic` و `Arshia.ProCSharp.OOP` وجود دارند. اگر شما یک کلاس بنام `Test` در فضای نامی `Arshia.ProCSharp` ایجاد کنید و یک نمونه از کلاس `NamespaceExample` در این کلاس ایجاد کنید، باید مشخص کنید که کدام یک از دو کلاس بالا را مد نظر دارید.

```
using Arshia.ProCSharp;
class Test
{
    public static int Main()
    {
        Basics.NamespaceExample nSEx = new Basics.NamespaceExample();
        // do something with the nSEx variable
        return 0;
    }
}
```

دستور `using` هیچ عمل پیوند فیزیکی بین فایل‌ها انجام نمی‌دهد و فضاهای نامی معادل فایل‌های `header` در ++C نیستند.

بیشتر سازمان‌ها مدت زمانی را برای توسعه یک فضای نامی صرف می‌کنند تا توسعه‌دهندگان بتوانند به راحتی کلاس‌های مورد نیاز خود را پیدا کنند و از تداخل نامی در کتابخانه‌های خود جلوگیری کنند.

۲-۷-۲-اسامی مستعار فضای اسمی

کاربرد دیگر کلید کلیدی `using`، انتساب اسامی مستعار به کلاس‌ها و فضاهای نامی است. استفاده از نام طولانی یک فضای نامی در برنامه به دفعات زیاد خسته کننده است. با استفاده از دستور ساده `using` می‌توانید یک نام مستعار به فضای نامی انتساب دهید.

```
using alias = NamespaceName;
```

مثال زیر نام مستعار `Introduction` را به فضای نامی `Arshia.ProCSharp.Basic` انتساب می‌دهد و این را برای ایجاد یک شی از کلاس `NamespaceExample` به کار می‌برد. به کاربرد توصیف کننده^۱ نام مستعار فضای نامی (::) توجه کنید. آن به اجبار نام مستعار فضای نامی را جستجو می‌کند. اگر یک کلاس به نام `Introduction` در همان میدان تولید شده باشد، تداخل رخ می‌دهد. عملگر :: حتی در صورت وجود تداخل، امکان استفاده از آن نام مستعار را فراهم می‌سازد. کلاس `NamespaceExample` یک متد به نام `GetNamespace()` دارد که از متد `GetType()` استفاده می‌کند. متد `GetType()` به یک شی از نوع `Type` دسترسی دارد که نوع کلاس را نشان می‌دهد. این شی را برای برگرداندن نام فضای نامی کلاس بکار می‌برید.

```
using System;
using Introduction = Arshia.ProCSharp.Basics;
class Test
{
    public static int Main()
    {
        Introduction::NamespaceExample nSEx =
        new Introduction::NamespaceExample();
        Console.WriteLine(nSEx.GetNamespace());
        return 0;
    }
}
```

^۱ Qualifier

```
namespace Arshia.ProCSharp.Basics
{
    class NamespaceExample
    {
        public string GetNamespace()
        {
            return this.GetType().Namespace;
        }
    }
}
```

۲-۸- کنسول I/O

تا اینجا باید یک آشنایی پایه‌ای از انواع داده‌ی C# بدست آورده باشید. در این قسمت چندین متد ایستای کلاس Console را برای خواندن و نوشتن داده استفاده می‌کنیم، چون این متدها زمان نوشتن برنامه‌های پایه‌ای C# مفید هستند.

برای خواندن یک خط متنی از پنجره کنسول، متد Console.ReadLine() را استفاده می‌کنید. این متد یک جریان ورودی را از پنجره کنسول خوانده و رشته ورودی را بر می‌گرداند. همچنین دو متد برای نوشتن کنسول وجود دارد.

```
Console.Write()
```

مقدار ارجاعی را به پنجره کنسول می‌نویسد.

```
Console.WriteLine()
```

همان کار را انجام می‌دهد، اما یک کاراکتر خط جدید به انتهای خروجی اضافه می‌کند.

شکل‌های متنوع این متدها برای انواع داده‌ای از پیش تعریف شده وجود دارند. پس در بیشتر موارد، تبدیل مقادیر به رشته‌ها قبل از نوشتن آنها لازم نیست. برای مثال، کد زیر به کاربر اجازه می‌دهد یک خط متنی وارد کند و آن را نمایش می‌دهد.

```
string s = Console.ReadLine();
Console.WriteLine(s);
```

متد Console.WriteLine() به شما اجازه می‌دهد، یک خروجی فرمت‌دهی شده را همانند تابع printf زبان C نمایش دهید. برای استفاده از این رفتار WriteLine، تعدادی پارامتر به آن ارسال می‌کنید. اولین پارامتر رشته‌ای است که در آن رشته از نشانه‌گذارها استفاده می‌شود. نشانه‌گذارها محل قرارگیری خروجی‌ها در داخل متن را با {} مشخص می‌کنند. هر نشانه‌گذار یک اندیس با پایه صفر را برای مشخص کردن تعداد پارامترها استفاده می‌کند. برای مثال {۰} اولین پارامتر لیست است. کد زیر را ملاحظه کنید.

```
int i = ۱۰;
int j = ۲۰;
Console.WriteLine("{۰} plus {۱} equals {۲}", i, j, i + j);
```

این کد خروجی زیر را نمایش می‌دهد.

```
۱۰ plus ۲۰ equals ۳۰
```

می‌توانید یک طول برای مقدار مشخص کنید و متن را با آن طول تنظیم کنید. مقادیر مثبت برای تنظیم راست و مقادیر منفی برای تنظیم چپ استفاده می‌شوند. برای این کار فرمت {n,w} را به کار می‌برید که n اندیس پارامتر و w طول مقدار است.

```
int i = ۹۴۰;
int j = ۷۳;
Console.WriteLine(" {۰,۴}\n+{۱,۴}\n ----\n {۲,۴}", i, j, i + j);
```

نتیجه به صورت زیر است.

```
۹۴۰
+ ۷۳
----
```

^۱ Stream

در نهایت می‌توانید یک رشته فرمت به همراه دقت (اختیاری) اضافه کنید. لیستی از فرمت‌بندی‌های مورد استفاده در انواع داده‌ی پیش تعریف شده را به صورت زیر می‌بینید:

رشته	توصیف
C	فرمت پول محلی
D	فرمت دسیمال. یک عدد صحیح را به مبنای ۱۰ بر می‌گرداند و اگر دقت آن مشخص باشد صفرهایی را به آن اضافه می‌کند.
E	فرمت علمی (توانی). مشخص کننده دقت، تعداد ارقام اعشار را مشخص می‌کند (پیش‌فرض آن ۶ است). بزرگی یا کوچکی حرف رشته فرمت (e یا E) حالت سمبل توان را تعیین می‌کند.
F	فرمت اعشاری ثابت. مشخص کننده دقت، ارقام اعشار را کنترل می‌کند. صفر قابل قبول است.
G	فرمت کلی. فرمت‌دهی f یا e را بر اساس حالت مقدار به کار می‌برد.
N	فرمت عدد، عدد را با جدا کننده هزار (کاما) فرمت می‌کند. برای مثال ۳۲۰,۷۶۷,۴۴
P	فرمت درصد
X	فرمت هگزا دسیمال. مشخص کننده دقت برای تعیین تعداد صفرهای اضافی به کار می‌رود.

توجه کنید که رشته فرمت به استثناء e/E در بقیه موارد به حالت حروف حساس نیست. اگر می‌خواهید یک رشته فرمت را بکار ببرید، بایستی آن را بلافاصله بعد از نشانه‌گذار مربوط به آن پارامتر قرار دهید و شماره پارامتر و طول مقدار را با کاما از هم جدا کنید. برای مثال، جهت فرمت‌دهی یک مقدار دسیمال به صورت واحد پول محلی کامپیوتر با دقت دو رقم اعشار، ۲C را به کار می‌برید.

```
decimal i = ۹۴۰,۲۲m;
decimal j = ۷۳,۷m;
Console.WriteLine(" {۰,۹:C۲}\n+{۱,۹:C۲}\n -----\n {۲,۹:C۲}", i, j, i + j);
```

خروجی این کد در ایالت متحده به صورت زیر است.

```
$۹۴۰,۲۲
$۷۳,۷۰
-----
$۱,۰۱۳,۹۳
```

به عنوان نکته نهایی، شما می‌توانید جهت نگاشت فرمت‌بندی کاراکترهای جایگزین را به جای رشته‌های فرمت به کار ببرید. به عنوان مثال:

```
double d = ۰,۲۳۴;
Console.WriteLine("{۰:#.۰۰}", d);
```

خروجی ۲۳. را نشان می‌دهد. اگر کاراکتری در آن محل نباشد، سمبل # صرف‌نظر می‌شود و صفرها با کاراکترهای موجود در آن موقعیت جایگزین می‌شوند و اگر کاراکتری نباشد، صفر چاپ می‌شود.

۹-۲- خلاصه

- انواع داده‌ی اولیه، پایه‌ی هر نوع داده‌ی تعریف شده در C# است.
- C# یک زبان برنامه‌نویسی ساده، مدرن و شی‌گرا و... است.
- هر قطعه کد C# باید داخل یک کلاس قرار گیرد.

- متد Main () نقطه‌ی ورودی هر برنامه‌ی C# است.
- C# در تعریف متغیرها به حالت حروف حساس است.
- در C# همه‌ی انواع داده به صورت Pascal نوشته می‌شوند، با استثناء متغیرهای محلی و پارامترها که با حالت Camel نوشته می‌شوند.
- سه روش برای مشخص کردن توضیحات در C# وجود دارد. //، /* */، ///
- هر نوع داده‌ی پایه در C# نام مستعار یک ساختار داده از FCL است.
- هر ساختار داده تعدادی متد مفید همچون Parse، TryParse و... دارند.
- مقادیر حرفی انتساب داده شده به متغیرهایی از نوع داده‌ی float، decimal و single به ترتیب حرف‌های F و M را به صورت پسوند مقدار عددی خود استفاده می‌کنند.
- عملگرهای ریاضی، منطقی و رابطه‌ای C# همانند زبان C است.
- عملگرهای & و | همواره هر دو عبارت را ارزیابی می‌کنند. اینها زمانی استفاده می‌شوند که مقادیر عبارات از یک تابع برمی‌گردند و می‌خواهیم از فراخوانی متدها مطمئن شویم.
- راهنماهای پیش‌پردازش برای راهنمایی کامپایلر C# استفاده می‌شوند.
- برای کامپایل شرطی می‌توان از راهنماهای #def و #undef استفاده کرد.
- یک روش مناسب برای ایجاد یک مجموعه ساخت یافته از سمبل‌ها جهت نمایش مقادیر ثابت پیشنهاد می‌کند.
- نوع شمارشی خوانایی برنامه را بالا می‌برد و تغییرات برنامه را راحت تر و منسجم تر می‌سازد.
- می‌توان یک مقدار شمارشی را به نام رشته‌ای آن تبدیل کرد.
- مقادیر شمارشی می‌توانند بصورت ترکیبی استفاده شود، بنابراین مقادیر عددی آنها را توان‌هایی از ۲ در نظر می‌گیرند تا نگاشت بیتی امکان پذیر باشد.
- انواع داده‌ی مقداری و ارجاعی از نظر تخصیص حافظه و مدیریت با هم فرق دارند. انواع مقداری در Stack و انواع ارجاعی در Heap مدیریت شده ذخیره می‌گردند.
- انواع داده‌ی مقداری از System.ValueType و انواع داده‌ی ارجاعی از System.Object ارث‌بری می‌کنند.
- جعبه‌بندی مقدار هر متغیری را به نوع object تبدیل می‌کند.
- از جعبه درآوردن، مقدار یک متغیر از نوع object را به یک نوع خاص تبدیل می‌کند.
- فضای نامی یک گروه‌بندی منطقی از کلاس‌ها و انواع داده‌ای دیگر فراهم می‌سازد. ممکن است این کلاس‌ها محل فیزیکی متفاوتی داشته باشند.
- در هنگام استفاده‌ی کلاس، باید نام کامل کلاس را نوشت. ولی در صورت استفاده از کلمه‌ی کلیدی using در ابتدای فایل، می‌توان فقط به نام کوچک کلاس اشاره کرد.
- برای دوری از اسامی طولانی فضاها نامی، می‌توان از اسامی مستعار برای فضاها نامی طولانی استفاده کرد و برای دسترسی به کلاس‌های آن فضای نامی بوسیله‌ی نام مستعار از عملگر :: استفاده می‌کنند.

فصل سوم

انشعاب

آنچه که در این فصل یاد خواهید گرفت

- نحوه‌ی استفاده از انشعاب برای کنترل روند اجرای برنامه
- دستورات انشعاب شرطی و غیرشرطی (if, else, if, goto و ...)
- دستورات تکرار برنامه به تعداد معین و نامعین (do while, for, while)
- خروج از بدنه حلقه و نادیده گرفتن برخی از دستورات حلقه و برگشت به ابتدای حلقه (continue break)

مقدمه

همه دستورات برنامه‌ی شما به ترتیب اجرا می‌شوند. متأسفانه آن زیاد سودمند نیست، مگر اینکه شما بخواهید برنامه‌تان در هر بار اجرا دقیقاً کار یکسانی انجام دهد. در حقیقت اغلب نمی‌خواهید همه کد اجرا شوند، بلکه می‌خواهید متناسب با مقدار یک متغیر کاری انجام شود یا نشود. این بدین معنی است که شما نیاز دارید برنامه‌ی خود را قادر سازید در زمان اجرا براساس شرایط برنامه، دستوراتی را انتخاب کرده و اجرا کند، این پروسه انشعاب^۱ خوانده می‌شود و دو روش برای انجام آن وجود دارد: غیرشرطی^۲ یا شرطی^۳. همانطور که از نامش پیداست، انشعاب غیرشرطی در هر زمان که اجرای برنامه به آنجا برسد اتفاق می‌افتد. برای مثال، زمانی که کامپایلر با فراخوانی یک متد مواجه می‌شود، یک انشعاب غیرشرطی اتفاق می‌افتد. کامپایلر اجرای متد جاری را متوقف می‌کند و به متد فراخوانی شده می‌رود. زمانی که متد فراخوانی شده تمام شود، اجرای برنامه به دستور بعد از فراخوانی متد در متد اصلی بر می‌گردد.

انشعاب شرطی، پیچیده‌تر است. در زمان اجرا متدها براساس ارزیابی شرطی‌های معینی می‌توانند منشعب شوند. برای مثال، ممکن است یک انشعاب ایجاد کنید تا فقط در صورتی که درآمد کارمندان پیمانی از یک میزان بیشتر باشد، مالیات آن را محاسبه کند. #C تعدادی دستور فراهم می‌سازد که انشعاب شرطی را پشتیبانی می‌کنند، همچون if, else, switch. کاربرد این دستورات بعداً در این فصل بررسی می‌شوند.

روش دیگری که پردازش گام‌به‌گام دستورات را می‌شکند، بوسیله ایجاد حلقه است. یک حلقه باعث می‌شود، یک مجموعه از دستورات تا زمانی که شرطی برقرار است تکرار گردند. ("برنامه داده‌هایی را از ورودی می‌گیرد تا زمانی که کاربر دستور توقف

^۱ Branching

^۲ unconditional

^۳ Conditional

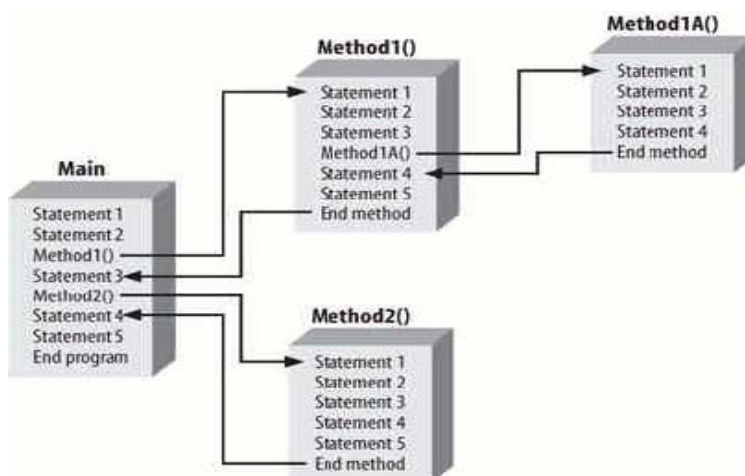
دهد یا ۱۰ عدد دریافت شده باشد"). C# چندین دستور برای ایجاد حلقه فراهم می‌کند، که do while, while, for در این فصل بررسی خواهند شد.

۱-۳- دستورات انشعاب غیرشرطی

ساده‌ترین مثال برای انشعاب غیرشرطی فراخوانی متد است. زمانی که یک فراخوانی متد فرا می‌رسد، هیچ عمل تست برای ارزیابی حالت شی انجام نمی‌شود. برنامه برای شروع متد جدید فوراً منشعب می‌شود. می‌توانید یک متد را بوسیله نوشتن نام آن فراخوانی کنید. برای مثال:

```
UpdateSalary ( ) ; // invokes the method up date salary
```

همانطور که قبلاً در این فصل شرح دادیم، زمانی که کامپایلر با یک فراخوانی متد مواجه می‌شود، اجرای متد جاری را متوقف می‌کند و به متد جدید منشعب می‌شود. زمانی که متد جدید اجراش را کامل می‌کند، کامپایلر به متد اصلی بر می‌گردد. این پروسه در شکل ۱-۳ بصورت شماتیک نشان داده می‌شود.



شکل ۱-۳- اجرای یک برنامه دارای چند متد

همانطور که شکل ۱-۳ پیش‌بینی می‌کند، انشعاب غیرشرطی چندین متد کاملاً طبیعی و معمول است. در شکل ۱-۳ اجرای برنامه در متدی بنام Main شروع می‌شود. دستور اول و دوم خود را اجرا می‌کند. سپس کامپایلر فراخوانی متد Method1() را می‌بیند. برنامه بصورت غیرشرطی به خط اول متد Method1() انشعاب می‌کند، تا سه دستور اول آن اجرا می‌شود. با رسیدن به فراخوانی MethodA() مجدداً اجرای برنامه منشعب می‌شود. این انشعاب برای شروع اجرای MethodA() است. چهار دستور MethodA() اجرا می‌گردد و از MethodA() بر می‌گردد. اجرای برنامه با اولین دستور بعد از فراخوانی در Method1() ادامه می‌یابد. اجرای برنامه تا انتهای Method1() ادامه می‌یابد. در این لحظه اجرای برنامه به دستور سوم در Main بر می‌گردد. مجدداً در فراخوانی Method2() اجرای برنامه منشعب می‌شود. همه دستورات Method2() اجرا می‌گردد و سپس دستور چهارم Main ادامه می‌یابد. زمانی که Main پایان می‌یابد، خود برنامه پایان می‌پذیرد.

می‌توانید تاثیر فراخوانی‌های متد را در مثال ۱-۳ ببینید. اجرا در Main شروع می‌شود، اما به یک متد بنام SomeMethod() انشعاب می‌کند. دستورات WriteLine() در هر متد برای نشان دادن محلی است که برنامه در حال اجرا است.

مثال ۱-۳

```
using System;
class Functions
{
    static void Main( )
```

```

{
    Console.WriteLine( "In Main! Calling SomeMethod( )..." );
    SomeMethod( );
    Console.WriteLine( "Back in Main( )." );
}
static void SomeMethod( )
{
    Console.WriteLine( "Greetings from SomeMethod!" );
}
}

```

خروجی بصورت زیر است :

```

In Main! Calling SomeMethod( )...
Greetings from SomeMethod!
Back in Main( ).

```

جریان برنامه در Main() شروع می‌شود و تا احضار SomeMethod() پیش می‌رود. (بعضی مواقع احضار یک متد را بصورت فراخوانی متد بیان می‌کنند). در آن نقطه جریان برنامه به متد منشعب می‌شود. زمانی که متد کامل می‌شود، جریان برنامه با دستور بعد از فراخوانی متد ادامه می‌یابد.

می‌توانید با استفاده از یکی از کلمات کلیدی انشعاب غیرشرطی همچون *goto* و *break*، *continue* یا *return* یا *throw* یک انشعاب غیرشرطی را ایجاد کنید. چهار مورد اول در این فصل بررسی می‌شوند.

۳-۲- دستورات انشعاب شرطی

اگرچه متدها بصورت غیرشرطی انشعاب می‌کنند، اغلب می‌خواهید براساس شرطی که در طول اجرای برنامه ارزیابی می‌کنید، انشعاب نمایید، این عمل با عنوان انشعاب شرطی معروف است. دستورات انشعاب شرطی به شما اجازه می‌دهند، منطقی همچون "اگر سن شما بالای ۲۵ سال است، ممکن است یک ماشین کرایه کرده باشید" را بنویسید. #C تعدادی ساختار فراهم می‌کند که به شما اجازه می‌دهند، انشعاب‌های شرطی در برنامه‌هایتان بنویسید. این ساختارها در بخش‌های بعدی شرح داده می‌شوند.

۳-۲-۱- دستورات if

if ساده‌ترین دستور انشعاب است. دستور if می‌گوید: اگر یک شرط برقرار است، پس دستوراتی را اجرا کن، در غیر اینصورت از آنها بگذر. شرط یک عبارت بولین است. این عبارت شامل دستوراتی است که به یک مقدار ارزیابی می‌شود. یک عبارت بولین با مقدار true یا false ارزیابی می‌شود.

توصیف رسمی دستور if بصورت زیر است:

(عبارت) if

دستور ۱

این ساختار نوعی توصیف از دستور if است که در مستندات کامپایلر می‌یابید. آن به شما نشان می‌دهد، دستور if عبارت را در داخل پرانتزها می‌گیرد و اگر عبارت به مقدار true ارزیابی گردد، دستورا اجرا می‌شود. توجه کنید که دستور ۱ می‌تواند یک بلوک از دستورات در داخل {} باشد. (همانطور که در مثال ۳-۲ ارائه شده است.)

مثال ۳-۲

```

using System;
namespace Branching
{
class Test
{

```

```

static void Main( )
{
    int valueOne = ۱۰;
    int valueTwo = ۲۰;
    int valueThree = ۳۰;
    Console.WriteLine( "Testing valueOne against valueTwo..." );
    if ( valueOne > valueTwo )
    {
        Console.WriteLine( "ValueOne: {۰} larger than ValueTwo: {۱}",valueOne,
                           valueTwo );
    }
    Console.WriteLine( "Testing valueThree against valueTwo..." );
    if ( valueThree > valueTwo )
    {
        Console.WriteLine( "ValueThree: {۰} larger than ValueTwo: {۱}",
                           valueThree, valueTwo );
    }
    // end if
}
// end Main
}
// end class
}
// end namespace

```

در هر جایی از C# که می‌توانید یک دستور بنویسید، می‌توانید بجای آن بلوکی از دستورات را در داخل {} بنویسید.

در این برنامه‌ی ساده، سه متغیر valueOne, valueTwo, valueThree به ترتیب با مقادیر ۱۰، ۲۰، و ۳۰ اعلان گردید. دستور if اول تست می‌کند، آیا valueOne بزرگتر از valueTwo است:

```

if (ValueOne>ValueTwo)
{
    Console Write Line("Value One:{۰}Laser then Value Two: {۱}"
                      ValueOne , ValueTwo);
}

```

چون (valueOne < valueTwo) کوچکتر از (valueTwo < valueTwo) است، دستور if شکست می‌خورد (مقدار false بر می‌گرداند) و بدنه‌ی دستور if را اجرا نمی‌کند. سپس تست می‌کند آیا valueThree بزرگتر از valueTwo است:

```

if (ValueThree>ValueTwo)
{
    Console. Write Line("Value Three:{۰}longer than Value Two:{۱}",
                      Value Three , Value Two);
}

```

چون (valueThree < valueTwo) بزرگتر از (valueTwo < valueTwo) است، عبارت بولین مقدار true بر می‌گرداند، سپس دستور را اجرا می‌کند. دستور در این مثال بلوکی است که متد WriteLine فراخوانی می‌شود. خروجی منعکس می‌کند که if اول شکست می‌خورد، ولی if دوم موفق است.

```

Testing Value One against Value Two...
Testing Value Three against Value Two...
Value Three : ۳۰ larger than Value Two : ۲۰

```

۳-۲-۲- بلوک های تک دستوری if

توجه کنید که بلوک دستورهای if نشان داده شده در مثال ۳-۲ هر کدام فقط یک دستور واحد در بردارند (یک فراخوانی از WriteLine()). در چنین مواردی می‌توانید آکولادهای باز و بسته بلوک if را ننویسید. پس ممکن است مثال ۳-۲ را بصورتی که در مثال ۳-۳ نشان داده شده، رونویسی کنید.

```
using System;
namespace Branching
{
    class Test
    {
        static void Main( )
        {
            int valueOne = ۱۰;
            int valueTwo = ۲۰;
            int valueThree = ۳۰;
            Console.WriteLine( "Testing valueOne against valueTwo..." );
            if ( valueOne > valueTwo )
                Console.WriteLine( "ValueOne: {۰} larger than ValueTwo: {۱}",
                                    valueOne, valueTwo );
            Console.WriteLine( "Testing valueThree against valueTwo..." );
            if ( valueThree > valueTwo )
                Console.WriteLine( "ValueThree: {۰} larger than ValueTwo: {۱}",
                                    valueThree, valueTwo );
        } // end Main
    } // end class
} // end namespace
```

حتی در صورتی که فقط یک دستور در بلوک if باشد، استفاده از آکولادها ایده خوبی است. دو دلیل وجود دارد: اول اینکه، خواندن و فهم کدها بوسیله آکولادها راحت تر است. کدی که خواندن آن راحت تر باشد نگهداری آن نیز راحت تر است.

منظور از نگهداری کد تغییراتی است که با تغییر نیازها یا رفع عیوب بر روی دستورات داده می شود.

دلیل دوم استفاده از آکولادها، دوری از یک خطای معمول است. وقتی دستور دیگری به if اضافه می شود، اضافه کردن آکولادها فراموش می شود. کد نشان داده شده در مثال ۳-۴ را ملاحظه کنید. برنامه نویس مقدار valueThree را به ۳ تغییر داده است و یک دستور دوم به بلوک if اضافه کرده است.

مثال ۳-۴

```
using System;
namespace Branching
{
    class Test
    {
        static void Main( )
        {
            int valueOne = ۱۰;
            int valueTwo = ۲۰;
            int valueThree = ۱۰;
            Console.WriteLine( "Testing valueOne against valueTwo..." );
            if ( valueOne > valueTwo )
                Console.WriteLine( "ValueOne: {۰} larger than ValueTwo: {۱}",
                                    valueOne, valueTwo );
            Console.WriteLine( "Testing valueThree against valueTwo..." );
            if ( valueThree > valueTwo )
                Console.WriteLine( "ValueThree: {۰} larger than ValueTwo: {۱}",
                                    valueThree, valueTwo );
            Console.WriteLine( "Good thing you tested again!" );
        } // end Main
    } // end class
} // end namespace
```

حال قبل از خواندن هر چیزی، کد را مرور کرده و خودتان تصمیم بگیرید که خروجی چه چیزی باید باشد. با نگاه کردن به پاراگراف قبلی گول نخورید. پس با اینکه فکر می کنید که می دانید خروجی چه چیزی خواهد شد، به موارد زیر نگاهی بیاندازید.

```
Testing Value One against Value Two
Testing Value Three against Value Two
Good thing you tested again
```

آیا متعجب شدید ؟

برنامه نویس به دلیل فقدان آکولادها و فرورفتگی فریب می خورد. بخاطر دارید که فرورفتگی یک فضای خالی است و بوسیله کامپایلر نادیده گرفته می شود. از نگاه برنامه نویس دستور دوم (" Good thing ...") بخشی از بلوک if است.

```
if ( valueThree > valueTwo )
    Console.WriteLine( "ValueThree: {0} larger than ValueTwo: {1}",
                        valueThree, valueTwo);
    Console.WriteLine("Good thing you tested again!");
```

کامپایلر فقط اولین دستور بعد از if را به عنوان بدنه ای آن اجرا می کند. دستور دوم بخشی از دستور if نیست. از نظر کامپایلر دستور if شبیه زیر است .

```
if ( valueThree > valueTwo )
    Console.WriteLine("ValueThree: {0} larger than ValueTwo:{1}", valueThree,
                        valueTwo);
Console.WriteLine("Good thing you tested again!");
```

اگر می خواهید دستور دوم بخشی از if باشد، بایستی همانطور که در زیر می بینید، آکولادها را بکار برید.

```
if ( valueThree > valueTwo )
{
    Console.WriteLine( "ValueThree: {0} larger than ValueTwo: {1}",
                        valueThree, valueTwo);
    Console.WriteLine("Good thing you tested again!");
}
```

بیشتر برنامه نویسان C# به دلیل جلوگیری از سردرگمی، در همه دستورات if آکولادها را بکار می برند، حتی اگر بدنه if فقط یک خط باشد.

سبک های آکولادها

چندین روش برای شکل دادن آکولادها در دستور if وجود دارد، اما بیشتر برنامه نویسان یکی از سه سبک را بکار خواهند برد.

سبک اول

```
if (condition)
{
    // statement
}
```

سبک دوم

```
if (condition)
{
    // statement
}
```

سبک سوم

```
if (condition) {
    // statement
}
```

سه سبک مختلف

سبک اول که در این کتاب استفاده می‌شود، آکولادها را در زیر کلمه کلیدی `if` قرار می‌دهد و محتوای بلوک `if` را تو رفته می‌کند. سبک دوم کمتر استفاده می‌شود. آکولادها را به همراه محتوای آنها تو رفته می‌کند. سبک سوم آکولاد باز را در همان خط دستور `if` و آکولاد بسته را در زیر دستور `if` قرار می‌دهد. سبک سوم به سبک K&R معروف است. (اختصار کلمات Kernighan , Ritchie) که نویسندگان اولین کتاب زبان برنامه‌نویسی C هستند. کتاب آنها بسیار بانفوذ بود که بیشتر برنامه‌نویسان یک تعهد قوی به این سبک آکولادها احساس می‌کنند. چون سبک K&R کمتر واضح است، این کتاب اولین سبک را بکار خواهد برد.

۳-۲-۳- ارزیابی کوتاه^۱

قطعه کد زیر را ملاحظه کنید:

```
int x=۸;
int y= ۱۵;
if ( ( x==۸) // ( y==۱۲) )
```

در اینجا دستور `if` کمی پیچیده است. کل دستور `if` پرانتزگذاری شده است، پس باید پارانته‌های بیرونی به `true` ارزیابی شوند تا اینکه دستور `if` به `true` ارزیابی شود. در داخل پارانته‌های بیرونی دو عبارت وجود دارد: `(x==۸)` و `(y==۱۲)` که بوسیله یک عملگر `or` (`||`) مجزا می‌شوند. چون `x` مساوی `۸` است، عبارت اول `(x==۸)` به `true` ارزیابی می‌شود و لازم نیست عبارت دوم `(y==۱۲)` ارزیابی گردد و اینکه آیا `y` مساوی `۱۲` است، مهم نیست و کل عبارت به `true` ارزیابی می‌شود. بطور مشابه این قطعه کد را ملاحظه کنید:

```
int x=۸;
int y=۱۲;
if ( ( x==۵) && (y==۱۲) )
```

باز نیازی نیست عبارت دوم ارزیابی شود، چون عبارت اول `false` است، عمل `and` به `false` ارزیابی خواهد شد. در چنین مواردی کامپایلر `#C` ارزیابی کوتاه انجام خواهد داد. تست دوم هرگز انجام نخواهد شد. این ویژگی به شما اجازه می‌دهد، آن را قبل از انجام هر کاری روی مقدار تست کنید که از احتمال یک استثناء دوری می‌کند. این یک مثال کوتاه است.

```
public bool QuotientOverTwenty(float dividend, float divisor)
{
    if ( divisor != ۰ && dividend / divisor > ۲۰ )
    {
        return true;
    }
    return false;
}
```

در این کد فقط می‌خواهیم تصمیم بگیریم آیا باقیمانده بزرگتر از `۲۰` است. اما برای دوری از استثناء تقسیم بر صفر بایستی مطمئن شویم که مقسوم علیه صفر نباشد. با ارزیابی کوتاه در صورتی که بخش اول `if` به `false` ارزیابی شود، هرگز بخش دوم اجرا نخواهد شد. دوماً اینکه مختصر بوده و فهم آن ساده‌تر از نوشتن آن است.

```
public bool QuotientOverTwenty(float dividend, float divisor)
{
    bool retVal = false;
```

^۱ Short circuit evaluation


```

if ( divisor != ۰ )
{
    if ( dividend / divisor > ۲۰ )
        retVal = true;
}
return retVal;
}

```

۳-۲-۴- دستورات if... else

اغلب می‌خواهید با توجه به اینکه نتیجه یک شرط true است، یک مجموعه از دستورات اجرا شود، در غیر اینصورت مجموعه دیگری اجرا گردد. این عمل یک منطق همانند این را مجاز می‌دارد: "اگر سن شما بیشتر از ۲۵ باشد، ممکن است یک ماشین کرایه کرده باشید، در غیر این صورت بایستی از قطار استفاده کنید".

بخش مربوط به "در غیر اینصورت" منطق بالا، در دستور else اجرا می‌شود. برای مثال، می‌توانید مثال ۳-۲ را تغییر دهید تا با اینکه آیا valueOne بزرگتر از valueTwo است یا نه؟ پیام مناسبی چاپ کند. همانطور که در مثال ۳-۵ می‌بینید.

مثال ۳-۵

```

using System;
namespace Branching
{
    class Test
    {
        static void Main( )
        {
            int valueOne = ۱۰;
            int valueTwo = ۲۰;
            Console.WriteLine( "Testing valueOne against valueTwo..." );
            if ( valueOne > valueTwo )
            {
                Console.WriteLine( "ValueOne: {۰} larger than ValueTwo: {۱}",
                                    valueOne, valueTwo );
            } // end if
            else
            {
                Console.WriteLine("Nope, ValueOne: {۰} is NOT larger than ValueTwo:
                                    {۱}", valueOne, valueTwo );
            } // end else
        } // end Main
    } // end class
} // end namespace

```

خروجی بصورت زیر است :

```

Testing valueOne against valueTwo...
Nope, ValueOne: ۱۰ is NOT larger than ValueTwo: ۲۰

```

چون عمل تست در دستور if شکست می‌خورد، بدنه دستور if رد شده و بدنه دستور else اجرا می‌شود و در صورتی که تست موفق‌آمیز باشد، بدنه دستور if اجرا می‌شود و بدنه دستور else رد خواهد شد.

برای اداره کردن شرط‌های پیچیده، تودرتو کردن دستورهای if امکان‌پذیر است. برای مثال، فرض کنید یک برنامه‌ای نیاز دارید که دما را ارزیابی کرده و اطلاعات زیر را برگرداند:

- اگر دما کمتر از ۳۲ درجه باشد، باید یخ زدگی جاده را هشدار دهد.
 - اگر دما دقیقاً مساوی ۳۲ درجه باشد، باید به شما درباره احتمال یخ‌زدگی گزارش دهد.
 - اگر دما بزرگتر از ۳۲ درجه باشد، باید اطمینان دهد که یخ‌زدگی اصلاً وجود ندارد.
- راه‌های مناسب زیادی برای نوشتن این برنامه وجود دارد. مثال ۳-۶ یکی از این روش‌ها را با استفاده از دستور if تودرتو ارائه می‌کند.

مثال ۳-۶

```
using System;
class Values
{
    static void Main( )
    {
        int temp = ۳۲;
        if ( temp <= ۳۲ )
        {
            Console.WriteLine( "Warning! Ice on road!" );
            if ( temp == ۳۲ )
            {
                Console.WriteLine("Temp exactly freezing, beware of water." );
            }
            else
            {
                Console.WriteLine( "Watch for black ice! Temp: {۰}", temp );
            }
        }
    }
}
```

منطق مثال ۳-۶ بدین صورت است: ابتدا تست می‌کند آیا دما کمتر یا مساوی ۳۲ است. اگر باشد، یک هشدار چاپ می‌کند:

```
if ( temp<=۳۲)
Console. Write line ( " warning Ice on road " );
```

سپس برنامه بررسی می‌کند، آیا دما مساوی ۳۲ درجه است، اگر باشد یک پیام چاپ می‌کند و اگر نباشد، باید دما کمتر از ۳۲ بوده و برنامه پیام بعدی را چاپ می‌کند.

توجه کنید که دستور if دوم درون دستور if اول است. پس منطق else بدین صورت است: " چون آن انجام نشده پس دما کمتر یا مساوی ۳۲ است و چون مساوی ۳۲ نیست، پس باید کمتر از ۳۲ باشد".

روش دیگر، زنجیر کردن بیش از یک دستور if بوسیله عبارت else if است، که بعضی از برنامه‌نویسان #C بکار می‌برند. برنامه شرط دستور if اول را تست می‌کند. اگر false باشد، کنترل برنامه به else عبور می‌کند، که با دستور if دیگری دنبال می‌شود که شرط متفاوتی را تست می‌کند. برای مثال، می‌توانید مثال ۳-۶ را رونویسی کنید تا با سه تست مشخص کند، آیا دما از درجه یخبندان بزرگتر، کوچکتر یا مساوی است. همانطور که در مثال ۳-۷ نشان داده می‌شود.

مثال ۳-۷

```
using System;
class Values
```

```

{
    static void Main( )
    {
        int temp = ۳۲;
        if ( temp < ۳۲ )
        {
            Console.WriteLine( "Warning! Ice on road!" );
        }
        else if ( temp == ۳۲ )
        {
            Console.WriteLine("Temp exactly freezing, beware of water." );
        }
        else
        {
            Console.WriteLine( "Watch for black ice! Temp: {۰}", temp );
        }
    }
}

```

این مثال ابتدا شرط if اول را تست می‌کند، آیا دما کوچکتر از ۳۲ است؟ چون دما دقیقاً ۳۲ است. عبارت اول false است و کنترل به دستور else if عبور می‌کند. دستور if دوم true است، پس دستور سوم (دستور else) هرگز اجرا نمی‌گردد. لطفاً توجه کنید که این کد معادل کد زیر است:

```

using System;
class Values
{
    static void Main( )
    {
        int temp = ۳۲;
        if ( temp < ۳۲ )
        {
            Console.WriteLine( "Warning! Ice on road!" );
        }
        else
        {
            if ( temp == ۳۲ )
            {
                Console.WriteLine("Temp exactly freezing, beware of water." );
            }
            else
            {
                Console.WriteLine( "Watch for black ice! Temp: {۰}", temp );
            }
        }
    }
}

```

در هر حالت، اگر از عبارت else if استفاده می‌کنید، مطمئن شوید که دستور تست نهایی به جای else if از else استفاده کرده باشد تا اگر هیچ کدام از شرطها برقرار نبود، این دستور اجرا گردد.

۳-۲-۶- دستورات switch

دستورات if تودرتو برای خواندن، اشکال‌یابی و کنترل درست‌بودن مشکل هستند. زمانی که یک مجموعه پیچیده از انتخاب‌ها وجود داشته باشد، دستور switch یک روش قدرتمندی است. منطق دستور switch این است: "یک مقدار معین را برگزین و بر طبق آن عمل کن"

```

switch ( expression )
{

```

```

case constant- expression
        Statement
Jump- Statement
[ default: statement \]
}

```

عبارتی که براساس آن سویچ می‌کنید، در بالای دستور switch در داخل پرانتزها گذاشته می‌شود. هر دستور case مقدار ثابتی را با عبارت مقایسه می‌کند. عبارات ثابت می‌توانند یک مقدار ثابت، شمارشی، سمبل یا حرف باشند.

کامپایلر از اولین دستور case شروع می‌کند و تا پایین ادامه می‌دهد و یک مقدار منطبق با عبارت را جستجو می‌کند. اگر یکی از case ها منطبق گردد، دستور یا بلوک دستورات اختصاص یافته به آن اجرا می‌شوند. باید بلوک case به یک دستور پرش خاتمه یابد. زمانی که یک break در یک دستور switch اجرا می‌گردد، اجرای دستور بعد از آکولاد بسته switch ادامه می‌یابد.

در مثال ۳-۸ از کاربر سوال می‌شود، وابستگی سیاسی خود را از میان دموکرات، جمهوری خواه یا مترقی انتخاب کند. برای سادگی مثال، دموکرات بصورت ثابت انتخاب شده است.

مثال ۳-۸

```

class Values
{
    enum Party
    {
        Democrat,
        Republican,
        Progressive
    }
    static void Main()
    {
        // hard wire to Democratic
        Party myChoice = Party.Democrat;
        // switch on the value of myChoice
        switch ( myChoice )
        {
            case Party.Democrat:
                Console.WriteLine( "You voted Democratic." );
                break;
            case Party.Republican:
                Console.WriteLine( "You voted Republican." );
                break;
            case Party.Progressive:
                Console.WriteLine( "You voted Progressive." );
                break;
        }
        Console.WriteLine( "Thank you for voting." );
    }
}

```

خروجی شبیه زیر است :

```

You voted Democratic.
Thank you for voting

```

مثال ۳-۸ به جای یک دستور if پیچیده، یک دستور switch بکار می‌برد. در ابتدا، انتخاب کاربر بوسیله‌ی دستور switch ارزیابی می‌شود و بلوکی از دستورات اجرا می‌شوند که case آن با مقدار مورد نظر منطبق می‌باشد. (در این مثال Democrat).

دستورات مابین دستور case و break به ترتیب اجرا می‌شوند. می‌توانید بدون اینکه از آکولادها استفاده کنید، بیش از یک دستور قرار دهید. دستور case و break به ترتیب جای آکولاد باز و بسته می‌باشند. ممکن است یک کاربر چیزی به غیر از

دموکرات، جمهوری خواه و مترقی انتخاب کند و همانطور که در مثال ۳-۹ مشاهده می کنید، ممکن است بخواهید یک case پیش فرض ایجاد کنید که با یک انتخاب نامعتبر دستورات آن اجرا گردد..

مثال ۳-۹

```
using System;
class Values
{
    enum Party
    {
        Democrat,
        Republican,
        Progressive
    }
    static void Main()
    {
        // hard wire to Democratic
        Party myChoice = Party.Democrat;
        // switch on the value of myChoice
        switch ( myChoice )
        {
            case Party.Democrat:
                Console.WriteLine( "You voted Democratic." );
                break;
            case Party.Republican:
                Console.WriteLine( "You voted Republican." );
                break;
            case Party.Progressive:
                Console.WriteLine( "You voted Progressive." );
                break;
            default:
                Console.WriteLine( "You did not make a valid choice." );
                break;
        }
        Console.WriteLine( "Thank you for voting." );
    }
}
```

خروجی شبیه زیر است :

```
You did not make a valid choice.
Thank you for voting.
```

اگر کاربر یکی از مقادیر متناسب با یک دستور case را انتخاب نکند، دستورات default را اجرا خواهد کرد. در این مثال یک پیام چاپ می شود که به کاربر می گوید انتخاب معتبری انجام نداده است. در کد برنامه اصلی، این قطعه کد داخل یک حلقه while قرار می گیرد تا زمانی که یک انتخاب معتبر صورت گیرد.

۳-۲-۷- سرازیر شدن یا پرش به case ها

اگر دو case دستورات یکسانی اجرا کنند، یک case "سرازیر شدنی" ایجاد می کنیم که چندین دستور case با کد یکسان را گروه بندی می کند. همانطور که در اینجا می بینید:

```
case Compassionate Republican :
case Republican :
    Console.WriteLine( "you voted Republican.\n");
    Console.WriteLine( "do not you feel compassion");
```

در این مثال اگر کاربر یکی از دو مورد جمهوری یا جمهوری دلسوز را انتخاب کند، مجموعه دستورات یکسانی اجرا خواهند شد.

توجه کنید فقط در صورتی که case اولی هیچ کدی نداشته باشد، می‌توانید سرازیر شوید. در این مثال case اول این معیار را برآورده می‌کند. پس می‌توانید به case دوم سرازیر شوید. با این وجود، اگر بخواهید در case اول دستوری اجرا کرده، سپس به case بعدی سرازیر شود. باید کلمه کلیدی goto را جهت پرش به case بعدی مورد نظر بکار ببرید. کلمه کلیدی goto یک انشعاب غیرشرطی است. زمانی که کامپایلر این دستور را می‌بیند، کنترل برنامه فوراً به جایی که آن اشاره می‌کند انتقال می‌یابد. حتی اگر آن در داخل یک دستور انشعاب شرطی گذاشته شده باشد.

برای مثال، اگر یک بخش NewLeft ایجاد کنید. ممکن است بخواهید با انتخاب NewLeft یک پیام چاپ شده و به Democrat سرازیر گردد. ممکن است کد زیر را بنویسید :

```
case NewLeft:
Console. write line("the New Left members are Voting Democratic");
case Democrat :
Console. write line(" you Voted Democratic./n");
Break;
```

این کد کامپایل نخواهد شد و با یک خطا شکست می‌خورد:

Control cannot fall through from one case label(case '۴') To another

این یک پیام خطای همراه کننده است. در صورتی که برچسب اولی دستوری نداشته باشد، کنترل اجرا می‌تواند از یک برچسب به برچسب دیگری سرازیر گردد. توجه کنید که پیام خطا نام case را با مقدار شمارشی آن (۴) به جای مقدار سمبلیک آن (NewLeft) نمایش می‌دهد. بخاطر دارید که NewLeft فقط نام ثابت است.

```
const int Democrat = ۰;
const int Compassioned Republican = ۱;
const int Republican = ۲;
const int Progressive = ۳;
const int NewLeft = ۴;
```

چون case مربوط به NewLeft یک دستور دارد، برای سرازیر شدن به case بعدی باید یک دستور goto بکار ببرید:

```
case NewLeft:
Console.write line("the New Left members are Voting Democratic");
Goto case Democrat;
case Democrat;
Console. write line("you Voted Democratic./n");
Break;
```

همانطور که انتظار دارید این کد کامپایل و اجرا خواهد شد. توجه کنید که لازم نیست Democrat بعد از NewLeft قرار گیرد.

۳-۲-۸- دستورات switch روی رشته‌ها

در مثال‌های قبلی مقدار switch یک ثابت صحیح بود. C# قابلیت switch روی یک رشته را نیز پیشنهاد می‌کند. پس همانطور که در مثال ۳-۱۰ می‌بینید، می‌توانید مثال ۳-۹ را برای switch روی رشته‌ها رونویسی کنید

مثال ۳-۱۰

```
using System;
class Values
{
    static void Main( )
    {
        String myChoice = "NewLeft";
        // switch on the string value of myChoice
        switch ( myChoice )
```

```

{
    case "NewLeft":
        Console.WriteLine(
            "The NewLeft members are voting Democratic." );
        goto case "Democrat";
    case "Democrat":
        Console.WriteLine( "You voted Democratic.\n" );
        break;
    case "CompassionateRepublican": // fall through
    case "Republican":
        Console.WriteLine( "You voted Republican.\n" );
        Console.WriteLine( "don't you feel compassionate?" );
        break;
    case "Progressive":
        Console.WriteLine( "You voted Progressive.\n" );
        break;
    default:
        Console.WriteLine( "You did not make a valid choice." );
        break;
}
Console.WriteLine( "Thank you for voting." );
}
}

```

۳-۳- دستورات تکرار

موقعیت‌های زیادی وجود دارد که می‌خواهید یک کار را چندین بار انجام دهید، بطوریکه در هر مرحله شاید یک مقدار تغییر کند. این عمل را تکرار یا حلقه گویند. معمولاً عمل یکسانی را روی هر قلم داده از یک کلکسیون (مجموعه‌ای از اقلام داده) تکرار خواهید کرد.

این عمل همانند برنامه‌نویسی یک خط مونتاژ است. در روی خط مونتاژ ممکن است یک صد بدنه ماشین را گرفته و روی هر کدام یک شیشه نصب کند. در یک برنامه تکراری ممکن است روی همه کادرهای متنی فرم کار کنید، بطوریکه مقدار هر کدام را گرفته و با استفاده از آن مقادیر یک پایگاه داده را بروز کنید.

#C یک دنباله وسیع از دستورات تکرار همچون حلقه‌های `while`، `do... while`، `for` و `foreach` فراهم می‌کند. همچنین با استفاده از دستور `goto` می‌توانید حلقه ایجاد کنید. بقیه این فصل کاربرد `goto`، `for`، `while`، `do...` را بررسی می‌کند.

۳-۳-۱- ایجاد حلقه‌ها با `goto`

قبلاً در این فصل دستور `goto` به عنوان یک انشعاب غیرشرطی در دستور `switch` استفاده شد. ایجاد یک حلقه، معمول‌ترین کاربرد `goto` است. در حقیقت دستور `goto` ریشه همه دستورهایی حلقه دیگر است.

در برنامه‌نویسی ساخت یافته توصیه می‌شود از دستور `goto` استفاده نشود. چون خطایابی برنامه را کاهش داده و اشکال‌یابی برنامه را نیز مشکل‌تر می‌کند. چون این مشکلات بوسیله دستور `goto` ایجاد می‌شوند. آن به ندرت در خارج از دستور `switch` استفاده می‌شوند. حال نحوه ایجاد حلقه با دستور `goto` را بررسی می‌کنیم:

۱- ایجاد یک برچسب ۲- دستور `goto` به آن برچسب

برچسب، شناسه‌ای است که بعد از آن یک کالن قرار می‌دهند. می‌توانید در کد خود برچسب قرار داده و کلمه کلیدی goto را برای پرش به آن برچسب بکار ببرید. همانطور که در مثال ۱۱-۳ می‌بینید، معمولاً دستور goto با یک دستور if مرتبط می‌گردد.

مثال ۱۱-۳

```
using System;
public class Tester
{
    public static void Main( )
    {
        int counterVariable = ۰;
        repeat: // the label
        Console.WriteLine(
            "counterVariable: {۰}", counterVariable );
        // increment the counter
        counterVariable++;
        if ( counterVariable < ۱۰ )
            goto repeat; // the dastardly deed
```

خروجی شبیه زیر است :

```
counterVariable: ۰
counterVariable: ۱
counterVariable: ۲
counterVariable: ۳
counterVariable: ۴
counterVariable: ۵
counterVariable: ۶
counterVariable: ۷
counterVariable: ۸
counterVariable: ۹
```

این کد بسیار پیچیده نیست. شما فقط یک دستور goto بکار بردید. با این وجود، در صورت استفاده از چندین برچسب و دستور goto دنبال کردن اجرای برنامه بسیار مشکل است.

۵-۳-۲ حلقه while

مفهوم حلقه while این است: "تا زمانی که شرط درست است، این کار را انجام بده" گرامر دستور به صورت زیر است :

```
while ( Boolean Expression ) Statement
```

معمولاً عبارت بولین عبارتی است که به یک مقدار false یا true ارزیابی می‌گردد. دستور اجرا شده توسط while، می‌تواند بلوکی از دستورات در داخل آکولادها باشد. مثال ۱۲-۳ کاربرد حلقه while را نشان می‌دهد.

مثال ۱۲-۳

```
using System;
public class Tester
{
    public static void Main( )
    {
        int counterVariable = ۰;
        // while the counter variable is less than ۱۰
        // print out its value
```



```

while ( counterVariable < ۱۰ )
{
    Console.WriteLine( "counterVariable: {۰}", counterVariable );
    counterVariable++;
}
}
}

```

خروجی شبیه زیر است :

```

counterVariable: ۰
counterVariable: ۱
counterVariable: ۲
counterVariable: ۳
counterVariable: ۴
counterVariable: ۵
counterVariable: ۶
counterVariable: ۷
counterVariable: ۸
counterVariable: ۹

```

نتایج کد مثال ۱۲-۳ با مثال ۱۱-۳ یکسان هستند، اما منطق کد کمی واضح تر است. دستور while خود-توصیف است و آن شبیه یک جمله انگلیسی خوانده می شود "تا زمانی که counterVariable از ۱۰ کوچکتر است، این پیام را چاپ کن و counterVariable را افزایش بده .

توجه کنید که حلقه while قبل از وارد شدن به حلقه، مقدار counterVariable را تست می کند. این عمل اطمینان می دهد که اگر شرط نادرست باشد، حلقه اجرا نخواهد شد. پس اگر counterVariable با ۱۱ مقداردهی اولیه شده باشد، هرگز حلقه اجرا نخواهد شد.

۳-۳-۳ حلقه do... while

در بعضی مواقع یک حلقه while هدف شما را برآورد نمی کند. در موقعیت های خاصی ممکن است بخواهید مفهوم while را معکوس کنید. عبارت " اجرا کن تا زمانی که شرط برقرار است" با " تا زمانی که شرط برقرار است اجرا کن" کاملاً متفاوت است. کد اولی مفهوم do... while را و دومی مفهوم while را می رساند. به عبارت دیگر حلقه do while ابتدا کاری را انجام داده، سپس شرط را بررسی می کند. این نوع حلقه، حداقل یک بار اجرا خواهد شد.

برای اینکه مطمئن شوید ابتدا عمل انجام شده، سپس شرط تست می شود حلقه do... while را بکار ببرید. گرامر بدین صورت است که ابتدا کلمه کلیدی do نوشته می شود و به دنبال آن دستور یا دستورات بدنه حلقه نوشته می شود. بعد از دستورات بدنه کلمه کلیدی while نوشته می شود و شرط به دنبال آن در داخل پرانتزها مشخص می شود و بعد از شرط یک سمی کالن (;) قرار می گیرد.

```
do Statement while(Boolean Expression);
```

مثال ۱۳-۳ برای استفاده از حلقه do... while مثال ۱۲-۳ را رونویسی می کند.

مثال ۱۳-۳

```

using System;
public class Tester
{
    public static void Main( )
    {
        int counterVariable = ۱۱;
    }
}

```

```
// display the message and then test that the value is
// less than ۱۰
do
{
    Console.WriteLine( "counterVariable: {۰}", counterVariable );
    counterVariable++;
} while ( counterVariable < ۱۰ );
}
```

خروجی بصورت زیر است :

```
counter variable : ۱۱
```

در مثال ۳-۱۳، ابتدا counterVariable را با ۱۱ مقداردهی اولیه می‌کند و تست while شکست می‌خورد و بدنه حلقه فقط یکبار اجرا می‌شود.

۳-۳-۴ - حلقه for

بررسی دقیق حلقه while در مثال ۳-۱۲، الگوی دستورات تکراری را آشکار می‌کند. یک متغیر را مقداردهی اولیه می‌کند (counterVariable)، متغیر را تست می‌کند (counterVariable < ۱۰). حلقه for اجازه می‌دهد همه این مراحل را در یک دستور واحد ترکیب کنیم. یک حلقه for با کلمه کلیدی for نوشته می‌شود که گرامر آن بصورت زیر است :

```
for([initializes]; [BooleanExpression]; [iterates];)
Statement
```

بخش اول سرآیند، مقداردهنده اولیه است که یک متغیر را مقداردهی اولیه می‌کند. بخش دوم عبارت بولین است و شرط پایان را مشخص می‌کند. بخش سوم تکرار کننده است که مقدار متغیر شمارنده را بروز می‌کند. همه اینها در داخل پرانتزهای باز و بسته قرار گرفته‌اند. یک نمونه از حلقه for در مثال ۳-۱۴ نشان داده می‌شود.

مثال ۳-۱۴

```
using System;
public class Tester
{
    public static void Main( )
    {
        for ( int counter = ۰; counter < ۱۰; counter++ )
        {
            Console.WriteLine(
                "counter: {۰} ", counter );
        }
    }
}
```

خروجی بصورت زیر است :

```
counter: ۰
counter: ۱
counter: ۲
counter: ۳
counter: ۴
counter: ۵
counter: ۶
```

```

counter: ۷
counter: ۸
counter: ۹

```

در قسمت اول سرآیند متغیر counter با صفر مقداردهی اولیه می‌شود. مقدار counter در بخش عبارت بولین سرآیند تست می‌گردد. در نهایت در بخش تکرارکننده‌ی سرآیند، مقدار counter افزایش داده می‌شود.

بخش مقداردهی اولیه فقط یکبار در شروع حلقه اجرا می‌گردد. شمارنده صحیح ایجاد شده و با صفر مقداردهی می‌شود و سپس عمل تست اجرا می‌شود. چون counter از ۱۰ کوچکتر است. بدنه حلقه for اجرا شده و مقداری نمایش داده می‌شود. بعد از تکمیل حلقه بخش تکرار کننده سرآیند اجرا می‌گردد و counter افزایش داده می‌شود. مقدار counter تست می‌شود. اگر شرط به true ارزیابی گردد، بدنه حلقه for مجدداً اجرا می‌شود.

در قسمت تکرارکننده، لازم نیست حتماً عملگر ++ استفاده شود. می‌توان از هر عبارتی استفاده کرد. در حلقه for دستورات ++counter و ++counter نتیجه یکسانی خواهند داشت.

منطق دستور for بدین صورت است "برای هر مقداری از counter که با صفر شروع می‌شود، اگر تست به true ارزیابی گردد، عملی را انجام بده و مقدار counter را بروز کن"

خارج شدن از حلقه for (دستور break)

با وجود درست بودن شرط، ممکن است بخواهیم از حلقه for خارج شویم. برای اینکه حلقه for را بصورت دائمی خاتمه دهید، دستور انشعاب غیرشرطی break را بکار ببرید.

دستور break حلقه for را متوقف می‌کند و اجرای برنامه به دستور بعد از حلقه for منسحب می‌شود. مثال ۳-۱۶ را ملاحظه کنید.

مثال ۳-۱۶

```

using System;
public class Tester
{
    public static void Main( )
    {
        for ( int counter = ۰; counter < ۱۰; counter++ )
        {
            Console.WriteLine( "counter: {۰} ", counter );
            // if condition is met, break out.
            if ( counter == ۵ )
            {
                Console.WriteLine( "Breaking out of the loop" );
                break;
            }
            Console.WriteLine( "for loop ended" );
        }
    }
}

```

خروجی بصورت زیر است :

```

counter: ۰
counter: ۱
counter: ۲
counter: ۳

```

```
counter: ۴
counter: ۵
Breaking out of the loop
for loop ended
```

در این مثال حلقه for تست می‌کند آیا شمارنده از ۵ کوچکتر است؟ اگر مقداری کوچکتر از ۵ پیدا شود از حلقه خارج می‌گردد.

دستور `continue`

در بعضی از مواقع ممکن است به جای خارج شدن از حلقه، مفهومی بصورت زیر را پیاده کنید "بقیه دستورات را تا آخر حلقه اجرا نکرده و از بالای حلقه مجدداً شروع کنید. برای پیاده‌سازی این مفهوم دستور انشعاب غیرشرطی `continue` را بکار برید.

کاربرد `break` و `continue` چندین نقطه خروج برای حلقه ایجاد می‌کنند، که فهم و نگهداری کد را مشکل‌تر می‌سازند و بهتر است با دقت استفاده شوند.

مثال ۳-۱۷ مکانیزم کار هر دو دستور `break` و `continue` را ارائه می‌کند. این کد یک سیستم پردازش ترافیک سیگنالی ایجاد می‌کند.

مثال ۳-۱۷

```
using System;
public class Tester
{
    public static int Main( )
    {
        string signal = "."; // initialize to neutral
        while ( signal != "X" ) // X indicates stop
        {
            Console.Write( "Enter a signal. X = stop. A = Abort: " );
            signal = Console.ReadLine( );
            // do some work here, no matter what signal you
            // receive
            Console.WriteLine( "Received: {0}", signal );
            if ( signal == "A" )
            {
                // faulty - abort signal processing
                // Log the problem and abort.
                Console.WriteLine( "Fault! Abort\n" );
                break;
            }
            if ( signal == "." )
            {
                // normal traffic condition
                // log and continue on
                Console.WriteLine( "All is well.\n" );
                continue;
            }
            // Problem. Take action and then log the problem
            // and then continue on
            Console.WriteLine( "{0} -- raise alarm!\n", signal );
        }
        return 0;
    }
}
```

سیگنال‌ها بوسیله واردکردن کارکترهای حرف بزرگ و اعداد از صفحه کلید شبیه‌سازی می‌شوند. بوسیله متد `Console.ReadLine()` یک خط متنی خوانده و در متغیر رشته‌ای قرار می‌دهد. با فشار دادن A رشته پایان می‌یابد.

الگوریتم ساده است، دریافت یک "0" (صفر) به معنی شرایط عادی است و فقط یک عمل ثبت رویداد لازم است. با دریافت سیگنال Abort (با حرف بزرگ "A" شبیه‌سازی شده) مشکلی ثبت شده و پروسه خاتمه می‌یابد. در نهایت برای هر رویداد دیگر، یک هشدار داده می‌شود. اگر سیگنال "X" باشد، هشدار داده می‌شود، اما حلقه `while` نیز خاتمه می‌یابد. این نمونه‌ای از خروجی مثال است:

```
Enter a signal. X = stop. A = Abort: *
Received: *
All is well.
Enter a signal. X = stop. A = Abort: \
Received: \
\ -- raise alarm!
Enter a signal. X = stop. A = Abort: X
Received: X
X -- raise alarm!
```

این دومین نمونه از خروجی است :

```
Enter a signal. X = stop. A = Abort: A
Received: A
Fault! Abort
```

نکته‌ی این تمرین مربوط به زمان دریافت سیگنال A است. دستور `if` اجرا شده و بدون دادن هشدار به برنامه از حلقه خارج می‌شود. زمانی که سیگنال 0 است. یک هشدار نامطلوب داده شده و برنامه از بالای حلقه ادامه می‌یابد. مطمئن باشید که حروف بزرگ X و A را استفاده می‌کنید.

عناصر اختیاری سرآیند حلقه `for`

به خاطر دارید که سرآیند حلقه `for` سه بخش مقداردهی اولیه، عبارت بولین و تکرارکننده را دارد و گرامر آن بصورت زیر است:

```
for([Initializes];[BooleanExpression]; [Aerators])
Statement
```

همه بخش‌های سرآیند حلقه `for` اختیاری هستند. همانطور که در مثال ۱۸-۳ می‌بینید، می‌توانید خارج از سرآیند حلقه `for`، مقدارها را مشخص کنید.

مثال ۱۸-۳

```
using System;
public class Tester
{
    public static void Main( )
    {
        int counter = 0;
        // some work here
        counter = 3;
        // more work here
        for ( ; counter < 10; counter++ )
        {
            Console.WriteLine("counter: {0} ", counter );
        }
    }
}
```

خروجی شبیه زیر است :

```

counter: ۳
counter: ۴
counter: ۵
counter: ۶
counter: ۷
counter: ۸
counter: ۹

```

در این مثال متغیر counter قبل از شروع حلقه for مقداردهی اولیه شده و تغییر داده شده است. توجه کنید که یک ; برای نگه داشتن محل مربوط به دستور مقداردهی اولیه گذاشته شده است.

اگر دلیلی برای افزایش مقدار counter در داخل حلقه for ندارید، می‌توانید بخش تکرار کننده را خالی بگذارید. مثال ۳-۱۹ را ببینید.

مثال ۳-۱۹

```

using System;
public class Tester
{
    public static void Main( )
    {
        for ( int counter = ۰; counter < ۱۰; ) // no increment
        {
            Console.WriteLine( "counter: {۰} ", counter );
            // do more work here
            counter++; // increment counter
        }
    }
}

```

در صورتی که حلقه for را بدون مقدار اولیه و تکرار کننده، بصورت زیر ایجاد کنیم:

```
for ( ; counter < ۱۰; )
```

شما یک حلقه while بوسیله for ایجاد کرده‌اید و البته اغلب استفاده نمی‌شود. نوشتن تمام بخش‌های سرآیند for امکان‌پذیر است، که یک حلقه بی‌نهایت ایجاد می‌گردد:

```
for ( ; ; )
```

می‌توانید حلقه بی‌نهایت را با حلقه (while true) ایجاد کنید. از یک حلقه بی‌نهایت با دستور break خارج می‌شوند. یک حلقه بی‌نهایت در مثال ۳-۲۰ نشان داده شده است.

مثال ۳-۲۰

```

using System;
public class Tester
{
    public static void Main( )
    {
        int counterVariable = ۰; // initialization
        for ( ; ; ) // forever
        {
            Console.WriteLine( "counter: {۰} ", counterVariable++ ); // increment
            if ( counterVariable > ۱۰ ) // test

```

```

        break;
    }
}
}

```

خروجی شبیه زیر است :

```

counter: ۰
counter: ۱
counter: ۲
counter: ۳
counter: ۴
counter: ۵
counter: ۶
counter: ۷
counter: ۸
counter: ۹
1۰counter:

```

برای مشخص کردن حالتی که یک حلقه بی‌شمار مرحله ادامه یابد، یک حلقه بی‌نهایت را بکار ببرید. برای مثال، اگر برنامه شما در انتظار یک رویداد از سیستم است، شرایط خارج شدن از حلقه می‌تواند استثناء بوده و در داخل بدنه حلقه مدیریت شود.

۳-۴- خلاصه

- انشعاب باعث می‌شود برنامه از حالت اجرای ترتیبی دستورات خارج شود.
- فراخوانی متد، معمول‌ترین انشعاب غیرشرطی است. بعد از کامل شدن اجرای متد، اجرای برنامه به نقطه بعد از فراخوانی متد بر می‌گردد.
- با دستورات انشعاب شرطی، برنامه شما قادر است براساس شرایط زمان اجرا انشعاب کند. معمولاً براساس مقدار یک یا چند متغیر یا شی عمل می‌کند.
- اگر یک شرط true باشد، ساختار if یک دستور را اجرا می‌کند، در غیر اینصورت آن را رد می‌کند.
- اگر در یک دستور if دو شرط بوسیله عملگر or متصل شده باشند. اگر شرط اول به true ارزیابی گردد، شرط دوم هرگز ارزیابی نخواهد شد. این ویژگی را میان بر زدن گویند.
- ساختار if else اجازه می‌دهد اگر یک شرط true باشد، مجموعه‌ای از دستورات اجرا گردند و اگر false باشد، مجموعه متفاوت دیگری اجرا شوند.
- برای ارزیابی شرط‌های پیچیده‌تر، می‌توانیم دستورات if تودرتو بکار ببریم.
- دستور switch اجازه می‌دهد مقدار یک عبارت با چند مقدار ثابت مقایسه گردد و عمل خاصی متناسب با مقدار منطبق شده انجام گیرد.
- در عمل بهتر است دستورات switch بخش default را داشته باشند تا اگر هیچ مقدار منطبق پیدا نشد، دستورات آن بخش اجرا شوند.
- تکرار یا حلقه به شما اجازه می‌دهد یک عمل چندین بار انجام شود. معمولاً تکرارها با یک عبارت شرطی کنترل می‌شوند.

- دستور goto برای هدایت اجرای برنامه به نقطه دیگر استفاده می‌شود و معمولاً استفاده از آن توصیه نمی‌شود.
- حلقه while تا زمانی که شرط آن true ارزیابی شود، بلوکی از دستورات را اجرا می‌کند. قبل از هر تکرار شرط تست می‌شود.
- حلقه do...while شبیه while است، با این تفاوت که شرط در انتهای تکرار تست می‌شود. پس تضمین می‌شود که بدنه حلقه حداقل یکبار اجرا خواهد شد.
- حلقه for برای اجرای دستور به تعداد معینی استفاده می‌شود. سرآیند حلقه for می‌تواند یک یا چند متغیر را مقداردهی اولیه کرده، یک شرط منطقی را تست کند و متغیرها را تغییر دهد. کاربرد معمول حلقه for مقداردهی اولیه یک شمارنده است. شرط حلقه قبل از شروع هر تکرار تست می‌شود و بعد از هر تکراری شمارنده را تغییر می‌دهد.

فصل چهار

برنامه‌نویسی شی‌گرا

آنچه که در این فصل یاد می‌گیرید:

- مفاهیم شی‌گرایی
- کلاس، شی و روابط آنها
- مدل سازی، نحوه‌ی تعریف کلاس
- ارکان سه‌گانه‌ی شی‌گرایی: کیسوله‌سازی، تخصص، چند ریختی
- مفهوم تحلیل و طراحی شی‌گرا

برنامه‌های ویندوزی و وب، برنامه‌های پیچیده‌ای هستند که در روشهای گرافیکی اطلاعات را به کاربران نمایش می‌دهند و واسطه‌های کاربران را مرکب از منوهای آبخاری و بازشو^۱، دکمه‌ها، کادر لیست^۲ و پیشنهاد می‌کنند. برنامه‌ها در پشت این واسطه^۳ها، روابط مفهومی پیچیده همچون روابط مابین مشتری‌ها و کالاها و سفارش‌ها و دارائی را مدل می‌کنند.

برای مدیریت این پیچیدگی عظیم، برنامه‌نویس‌ها یک تکنیک به نام برنامه‌نویسی شی‌گرا^۴ توسعه داده‌اند. آن بر اساس یک فرضیه‌ی خیلی ساده است. شما پیچیدگی را با مدل کردن جنبه‌های ضروری سیستم مدیریت کنید. برنامه‌ی شما برای مدل کردن مسئله‌ای است که سعی دارید حل کنید و ساده‌ترین راه فهم آن برنامه است.

برنامه‌نویس‌ها به مسئله‌ای که سعی دارید حل کنید و همه اطلاعاتی که مربوط به این مسئله هستند (دامنه مسئله^۵)، مراجعه می‌کنند. برای مثال، اگر شما یک برنامه برای مدیریت دارائی و فروش یک شرکت بنویسید. دامنه‌ی برنامه شامل هر چیزی درباره نحوه‌ی بدست آوردن و مدیریت دارائی، فروش و بکار بردن سود فروش و پی‌گیری ارقام فروش و... است. باید مدیر فروش و مدیر سرمایه، کارشناسان دامنه‌ی مسئله باشند که در فهم دامنه می‌توانند موثرتر عمل کنند.

یک برنامه‌ی شی‌گرا با طراحی خوب، با اشیائی از دامنه‌ی مسئله پر می‌شود. برای مثال، اگر دامنه‌ی مسئله برای بانکداری است، ممکن است اشیاء دامنه شامل مشتریان و حساب‌ها و صورت‌حساب‌های ماهیانه و... باشند.

در اولین سطح از طراحی، شما درباره نحوه‌ی تکامل اشیاء، حالت آنها، توانایی‌ها و مسئولیت‌های آنها فکر خواهید کرد.

^۱ PopUp

^۲ Listbox

^۳ Interface

^۴ Object oriented programming

^۵ problem domain

حالت

یک برنامه‌نویس به شرائط جاری و مقادیر یک شی، حالت شی^۱ می‌گوید. برای مثال، شیئی که یک مشتری را نشان می‌دهد، در نظر بگیرد. حالت مشتری با آدرس مشتری، شماره تلفن و پست الکترونیکی و نرخ اعتبار مشتری نشان داده می‌شود.

توانایی‌ها

مشتری توانایی‌های زیادی دارد. اما توسعه‌دهنده فقط مواردی که مرتبط با دامنه‌ی مسئله هستند را مدل می‌کند. پس احتمالاً مشتری قادر است سپرده‌گذاری، انتقال وجه و برداشت از صندوق و... را انجام دهد.

مسئولیت‌ها

در امتداد توانایی‌ها، مسئولیت‌هایی می‌آیند. شی مشتری مسئول مدیریت آدرس خود است. در یک برنامه با طراحی خوب، هیچ شی دیگری لازم ندارد به جزئیات آدرس مشتری دسترسی داشته باشد. آدرس به عنوان یک داده با شی مشتری ذخیره می‌شود. اما شی مشتری باید بداند چگونه آدرس خودش را بازیابی و بروز کند. توانایی یک شی برای مسئولیت‌پذیری در برابر حالت و رفتارش به کپسوله‌سازی^۲ معروف است.

۴-۱-۱ ایجاد مدل‌ها

انسانها سازندگان مدل هستند. ما مدل‌های دنیا را برای مدیریت پیچیدگی و کمک به فهم مسائل برای حل آنها ایجاد می‌کنیم. شما همواره مدل‌ها را می‌بینید. نقشه‌ها مدل‌هایی از جاده‌ها هستند. گوی‌ها مدل‌هایی از زمین هستند. در اصل مدل‌ها نوعی ساده‌سازی^۳ بشمار می‌آیند. اگر یک نقشه از ایالت متحده دارید، در صورتی که بخواهید همه چیز را دقیق روی نقشه بیاورید، نقشه به بزرگی ایالت متحده خواهد بود، ولی در روی نقشه اطلاعات خاص و دقیق قرار می‌گیرند. یک طراحی شی‌گرای خوب، مدل دقیقی از مسئله‌ای است که سعی دارید حل کنید. انتخاب شی در طراحی، نه تنها نحوه‌ی حل یک مسئله را تحت تاثیر قرار می‌دهد، بلکه نحوه‌ی تفکر شما روی مسئله را نیز تحت تاثیر قرار می‌دهد. یک طراحی خوب، شبیه یک مدل خوب به شما اجازه می‌دهد، جزئیات مرتبط با مسئله را بدون سردرگمی بررسی کنید.

۴-۲-۲ کلاس‌ها و اشیاء

ما دنیا را مرکب از اشیاء درک می‌کنیم. شما چیزی جز قطعات پلاستیکی و شیشه‌ای ادغام شده با محیط را نمی‌بینید، شما بطور طبیعی اشیاء متمایز را می‌بینید. یک کامپیوتر، یک صفحه کلید، یک مانیتور، اسپیکرها، مداد و کاغذ. مطلب مهم اینکه، قبل از اینکه خودتان تصمیم بگیرید، این اشیاء را به گروه مبدل کرده‌اید. شما کامپیوتر روی میزتان را به عنوان یک نمونه‌ی خاص از یک نوع گروه‌بندی می‌کنید. این کامپیوتر یک نمونه از نوع کامپیوتر است.

^۱ object state^۲ Encapsulation^۳ simplification

نظریه‌ی پشت پرده برای برنامه‌سازی شی‌گرا، مدل‌سازی صحیح دنیا برای برنامه‌های کامپیوتری است. این برنامه‌ها باید تمایل انسان را در مورد نمایش تک‌تک اشیاء و نوع آنها منعکس سازد. در C# این کار را با ایجاد یک کلاس^۱ جهت تعریف یک نوع داده و ایجاد یک نمونه^۲ از آن کلاس برای مدل کردن چیزی انجام می‌دهید.

یک کلاس یک نوع داده‌ی جدید را تعریف می‌کند. هر کلاسی خصوصیات مشترک هر شی از آن نوع جدید را تعریف می‌کند.

برای مثال، کلاس Car را در نظر بگیرید. ماشین من و شما هر دو به کلاس Car تعلق دارند. آنها از نوع داده‌ی Car هستند. یک شی، یک نمونه‌ی منحصر به فرد از یک کلاس است. هر ماشین منحصر به فرد، یک نمونه از کلاس Car است. پس یک شی است و شی نیز یک چیز است.

۴-۳- تعریف یک کلاس:

در زمان تعریف یک کلاس، ویژگی‌ها و رفتار اشیاء آن نوع داده را تعریف می‌کنید. در C# ویژگی‌ها را با فیلد عضو^۳ شرح می‌دهید.

```
class Dog
{
private int weight; // member field
private String name; // member field
```

فیلدهای کلاس برای نگهداری حالت شی استفاده می‌شوند. برای مثال، حالت Dog با وزن و نام جاری آن تعریف می‌شود. حالت یک کارمند با حقوق و سطح مدیریت و نرخ کارائی آن تعریف می‌شود. شما رفتار نوع داده‌ی جدیدتان را با متدها تعریف می‌کنید. متدها کدی را برای انجام یک عمل در بر می‌گیرند.

```
class Dog
{
private int weight;
private String name;
public void Bark( ) // member method
{
// code here to bark
}
```

کلمات کلیدی public و private معرف‌های دسترسی هستند که سطح دسترسی به آن فیلد یا متد را مشخص می‌کنند. اعضای private فقط در داخل همان کلاس قابل دسترسی هستند، ولی اعضای public از بیرون کلاس نیز قابل دسترسی خواهند بود.

^۱ class
^۲ instance
^۳ Member field

یک کلاس تعدادی متد برای کار با آن کلاس تعریف می‌کند. کلاس Dog احتمالا متدهایی برای پارس کردن^۱ و خوردن و چرت زدن^۲ و... دارد. کلاس کارمند متدهایی برای تسویه حقوق، بازدیدهای سالیانه و ارزیابی اهداف کارائی در بر می‌گیرد.

متدها با تغییر دادن مقادیر فیلدهای عضو می‌توانند حالت شی را دستکاری کنند. یک متد می‌تواند با اشیاء دیگر از همان نوع یا انواع دیگر تعامل داشته باشند. تعامل بین اشیاء برای برنامه‌نویسی شی‌گرا بسیار مهم است. برای مثال، یک متد در کلاس سگ، حالت آن را تغییر می‌دهد. (یک متد Feed())، وزن سگ را تغییر می‌دهد. با سگهای دیگر تعامل دارد (Bark و Sniff) و یا با انسان تعامل دارد (BegForFeed). ممکن است یک شی کالا با شی مشتری تعامل داشته باشد.

در یک برنامه‌ی شی‌گرای خوب، شما اشیائی را طرح می‌کنید که در دامنه مسئله شما چیزهائی را نشان می‌دهند. سپس کار برنامه را با تخصیص مسئولیت به اشیاء (بر اساس توانایی آنها)، مابین اشیاء تقسیم خواهید کرد.

۳-۴- روابط کلاس:

برقراری روابط مابین کلاس‌ها، قلب طراحی شی‌گرا است. کلاس‌ها با روش‌های مختلفی با بقیه در تعامل بوده و مرتبط هستند. ساده ترین تعامل زمانی است که متدی از یک کلاس، متد کلاس دیگری را فراخوانی می‌کند. برای مثال، کلاس Manager، متد UpdateSalary یک شی از کلاس کارمند را فراخوانی می‌کند. می‌گوئیم کلاس Manager و کلاس Employee انجمن^۳ هستند. انجمن ما بین کلاس‌ها به معنی تعامل ما بین آنهاست.

بعضی از انواع داده‌های پیچیده از انواع دیگری مرکب می‌شوند. برای مثال، یک خودرو، از چرخ‌ها، موتور، سیستم انتقال و... تشکیل شده است. برای مدلسازی خودرو، یک کلاس چرخ، یک کلاس موتور و یک کلاس انتقال ایجاد می‌شود. پس می‌توانید کلاس ماشین را ایجاد کنید. هر ماشین چهار نمونه از کلاس چرخ، یک نمونه از کلاس موتور و یک نمونه از کلاس انتقال دارد. این ترکیب عموماً رابطه Has-a خوانده می‌شود. روش دیگر نمایش این رابطه آن است که کلاس خودرو، کلاس چرخ، موتور و انتقال را تجمع می‌کند. یا کلاس کار از اشیاء چرخ، موتور و انتقال تشکیل می‌شود. این رابطه را Is-a گویند.

در بعضی از زبان‌ها همچون ++C مابین روابط is-a و has-a تفاوت وجود دارد، ولی در #C این تفاوت اعمال نمی‌شود.

رابطه‌ی تجمع به شما اجازه می‌دهد کلاس‌های پیچیده را با اسمبل کردن و ترکیب کلاس‌های ساده ایجاد کنید. چارچوب NET. یک کلاس String برای بکار بردن رشته‌های متنی فراهم می‌کند. ممکن است کلاس Address خود را با پنج رشته متنی ایجاد کنید. سپس کلاس Employee را طوری ایجاد کنید که یک عضو آن از نوع Address است.

۳-۵- ارکان سه‌گانه‌ی برنامه‌نویسی شی‌گرا

برنامه نویسی شی‌گرا، بر پایه سه رکن ساخته می‌شود: کپسوله کردن، تخصص، چند ریختی. هر کلاس باید کاملاً کپسوله باشد. آن باید بطور کامل حالت و مسئولیتهای نوع داده را تعریف کند. تخصص برقراری ارتباط‌های سلسله مراتبی مابین

^۱ Bark

^۲ nap

^۳ associate

کلاسهای خودتان را مجاز می‌دارد. چندریختی، اجازه می‌دهد یک گروه از اشیا مرتبط در سلسله مراتب، رفتار مشابهی نشان دهند.

۴-۵-۱-کپسوله کردن

کپسوله کردن، اولین رکن برنامه‌سازی شی‌گرا است. هدف پشت پرده کپسوله‌سازی این است که هر نوع داده یا کلاس را محرم‌مانه و تودار (self-contains) نگه دارید. بنابراین بدون اینکه هر کلاس دیگری را تحت تاثیر قرار دهد، می‌توانید پیاده‌سازی یک کلاس را تغییر دهید. متدی که توسط یک کلاس برای استفاده در کلاس‌های دیگر ایجاد می‌شود، سرویس‌دهنده گفته می‌شود و کسی که این متد را بکار می‌برد، یک سرویس‌گیرنده گفته می‌شود. کپسوله کردن به شما اجازه می‌دهد جزئیات مربوط به پیاده‌سازی نحوه‌ی کارکرد سرور را تغییر دهید، بدون اینکه پیاده‌سازی سرویس‌گیرنده را با شکست مواجه کنید.

خصوصیات و متدهای public کلاس که از بیرون کلاس قابل دسترسی هستند را واسط کلاس می‌نامند. یک سرویس‌گیرنده باید مطمئن باشد که واسط عمومی کلاس تغییر نمی‌کند. اگر واسط عمومی کلاس تغییر یابد، پس سرویس‌گیرنده نیز باید مجدداً طراحی و کامپایل شود.

به عبارت دیگر، پیاده‌سازی خصوصی مختص خود سرویس‌دهنده است. طراح کلاس سرویس‌دهنده برای تغییر واسط عمومی کلاس خود آزاد است، ولی آن باید پارامترهای معینی را بگیرد، کار وعده داده شده را انجام دهد و مقدار وعده داده شده را برگرداند و دسترسی به خصوصیات public را مجاز دارد.

برای مثال، فرض کنید یک متد به نام NetPresentValue() وجود دارد که یک مقدار بر حسب دلار و تعداد سال را می‌گیرد و مقدار کنونی شبکه را بر می‌گرداند. نحوه‌ی محاسبه به تجارت ما بستگی دارد.

۴-۵-۲-تخصص

تخصص^۱، رکن دوم برنامه‌نویسی شی‌گرا است که در C# با وراثت^۲ پیاده‌سازی می‌شود (با اعلان یک کلاس جدید که از یک کلاس موجود ارث‌بری می‌کند). کلاس تخصصی شده، ویژگی‌های کلاس کلی را ارث‌بری می‌کند. کلاس تخصصی شده، یک کلاس مشتق شده^۳ نامیده می‌شود. در حالیکه کلاس کلی، به یک کلاس پایه معروف است.

از رابطه تخصص به عنوان یک رابطه Is-a اشاره می‌شود. سگ یک پستاندار است. ماشین یک وسیله نقلیه است (سگ از کلاس پایه‌ی پستاندار و ماشین از کلاس پایه‌ی وسیله نقلیه مشتق می‌شود).

برای مثال، Manager یک نوع خاصی از Employee است. مدیر توانایی‌های جدید (استخدام، اخراج) و حالت جدید (اهداف سالیانه، سطح مدیریت) را اضافه می‌کند. Manager ویژگی‌های کلی همه کارمندان را به ارث می‌برد. پس کلاس Manager یک آدرس، یک نام، یک شماره استخدام دارد و Manager می‌تواند ارتقاء یا تنزل یابد.

تخصص، ایجاد یک خانواده از اشیا را مجاز می‌دارد. در ویندوز، Button یک کنترل است. ListBox یک کنترل است. کنترل‌ها ویژگی‌های معین (رنگ، اندازه، موقعیت) و توانایی‌های معین (رسم، انتخاب و...) دارند. این ویژگی‌ها و توانایی‌ها به وسیله همه انواع داده‌ای مشتق شده از آنها ارث‌بری می‌شود. به جای paste, cut کردن از یک نوع به نوع دیگر، نوع داده مشتق شده فیلدها و متدهای مشترک را به ارث می‌برند.

^۱ specialization

^۲ inheritance

^۳Derived

۴-۵-۳- چندریختی

چند ریختی، رکن سوم برنامه‌نویسی شی‌گرا است که بسیار مرتبط با وراثت است. poly به معنی چند و morph به معنی شکل است. چندریختی یعنی اینکه یک کلاس یا یک نوع داده‌ی منفرد، چندین شکل داشته باشند.

در بعضی مواقع شما یک کلکسیون داده از نوع داده کلی دارید. برای مثال، یک کلکسیون از کنترل‌ها را در نظر بگیرید که نمی‌دانید زیر نوع خاص هر کدام چیست؟ (Button, ListBox). مهمترین چیز این است که می‌دانید همه آنها توانایی‌های مشترکی را به ارث می‌برند. با همه آنها به عنوان کنترل رفتار می‌کنید. اگر شما یک دستور بنویسید که هر کنترل خودش را رسم کند، چون متد Draw بطور مناسب روی هر کنترل پیاده‌سازی می‌شود، نیازی نیست بدانید که هر کنترل چگونه خود را رسم می‌کند. فقط کافی است بدانید هر کنترل توانایی رسم خود را دارد.

چندریختی به شما اجازه می‌دهد با یک کلکسیون از انواع داده مشتق شده‌ی مجزا بصورت یک گروه رفتار کنید.

۴-۶- تحلیل و طراحی شی‌گرا

قبل از برنامه‌نویسی هر چیزی، لازم است دو مرحله‌ی تحلیل و طراحی را انجام دهید. تحلیل، همان پروسه‌ی فهم و تشریح مسئله‌ای است که سعی دارید حل کنید. طراحی، همان برنامه‌ریزی واقعی راه‌حل‌تان است.

در مسائل کوچک، ممکن است یک دوره‌ی تحلیل گسترده نیاز نباشد. اما در مسئله‌های تجاری پیچیده، پروسه‌ی تحلیل هفته‌ها، حتی ماه‌ها طول می‌کشد. یک تکنیک تحلیل قدرتمند، ایجاد مواردی بنام سناریوهای use-case است که شما بعضی از جزئیات نحوه استفاده از سیستم را شرح می‌دهید. در میان بررسی‌های دیگر، تعیین فاکتورهای موفقیت و نوشتن مشخصه‌ی نیازمندی‌های برنامه شما وجود دارد.

بعد از اینکه مسئله را تحلیل کردید، شما راه حل را طراحی می‌کنید. کلاس‌هایی که استفاده خواهید کرد و روابط ما بین آنها را در پروسه طراحی تصور می‌کنید. ممکن است یک برنامه ساده را بدون برنامه‌ریزی دقیق آن طراحی کنید.

چندین تکنیک طراحی قدرتمند برای استفاده شما وجود دارد. مقدار زمانی که برای طراحی قبل از شروع کدنویسی صرف می‌کنند به پیچیدگی سازمان، اندازه‌ی تیم شما، تجربه و آموزش بستگی دارد.

۴-۷- خلاصه

- برنامه‌نویسی شی‌گرا به برنامه‌نویس‌ها در جهت مدیریت پیچیدگی با مدل کردن جنبه‌های ضروری مسئله واقعی کمک می‌کند.
- کلاس یک نوع داده جدید در برنامه‌ی شما تعریف می‌کند و نوعاً برای نمایش یک نوع چیز در دامنه‌ی مسئله بکار برده می‌شود.
- یک شی نمونه‌ای از یک کلاس است.
- حالت شی، شرایط جاری یک شی است.

- بیشتر کلاس‌ها فیلدهای عضو تعریف می‌کنند که متغیرهای خصوصی بوده و فقط برای متدهای داخل آن کلاس نمایان هستند.
- رفتار کلاس با متدها تعریف می‌شود که قطعه کدی را برای انجام یک عمل در بر دارد. متدها حالت شی را دستکاری می‌کنند و با اشیاء دیگر تعامل می‌کنند.
- کپسوله‌کردن، تخصص و چندریختی، سه رکن اساسی برنامه‌نویسی شی‌گرا هستند.
- در کپسوله‌کردن لازم است هر کلاسی محرمانه و تودار باشد. تخصص به وسیله‌ی وراثت پیاده‌سازی می‌شود.
- چندریختی به شما اجازه می‌دهد با یک کلکسیون از اشیائی که انواع آنها از کلاس پایه مشترکی مشتق شده‌اند، بصورت یک گروه رفتار کنید.
- تحلیل، همان پروسه تشریح مسئله‌ای است که سعی دارید حل کنید.
- طراحی، همان برنامه‌ریزی حل مسئله است.

فصل پنجم

کلاس‌ها و اشیاء

آنچه که در این فصل یاد خواهید گرفت:

- نحوه‌ی تعریف یک کلاس و استفاده از آن
- معرف‌های دسترسی و تاثیر آنها روی اعضای کلاس
- سازنده‌ها و مخرب کلاس
- اعضای نمونه و ایستای کلاس و تفاوت‌های کاربردی آنها
- تخصیص حافظه به نمونه‌های کلاس
- کلمه‌ی کلیدی `this`

در فصل‌های قبلی در مورد انواع داده درونی (ساخته شده در `#c`) صحبت کردیم. به یاد دارید که این انواع داده برای نگهداری و دستکاری مقادیر عددی و رشته‌ای استفاده می‌شدند. قدرت واقعی `#c` تعریف انواع داده جدید برای انطباق با مسایل خاص است. توانایی ایجاد انواع داده‌ای جدید، یک زبان شی‌گرا را مشخص می‌کند. در `#c` با اعلان و تعریف کلاس‌ها، انواع داده‌ی جدید را مشخص می‌کنید.

نمونه‌های خاصی از یک کلاس، اشیاء نامیده می‌شوند. فرق بین یک کلاس و یک شی همانند اختلاف ما بین مفهوم یک سگ و سگ خاصی به نام جو است. شما نمی‌توانید اطلاعات را از یک کلاس سگ واکنشی کنید، فقط با یک نمونه از آن کلاس می‌توانید کار کنید.

یک کلاس سگ شرح می‌دهد، آنها چه چیزهای مشابهی دارند. آنها وزن، طول، رنگ چشم، رنگ مو، مزاج و غیره دارند. آنها می‌توانند کارهای مشابهی انجام دهند، همچون خوردن، راه رفتن، پارس کردن و خوابیدن. هر سگ خاصی، وزن و قد، رنگ چشم و مو و مزاج خاصی دارد.

مزیت بزرگ برنامه‌نویسی شی‌گرا آن است که کلاس‌ها، ویژگی‌ها و توانایی‌های یک نوع داده را در یک واحد منحصر به فرد و تودار کپسوله می‌کنند.

برای مثال، فرض کنید می‌خواهید محتویات یک نمونه از کنترل `ListBox` ویندوز را مرتب کنید. کنترل `ListBox` به عنوان یک کلاس تعریف می‌شود. یکی از ویژگی‌های کلاس این است که آن می‌داند چگونه خود را مرتب کند. مرتب‌سازی در کلاس کپسوله شده و کلاس‌های دیگر از نحوه‌ی کار آن مطلع نیستند. اگر بخواهید `ListBox` مرتب شود، فقط به آن می‌گویید که خودش را مرتب کند. شما خود را با نحوه‌ی مرتب‌سازی آن درگیر نمی‌کنید. این همان کپسوله‌کردن است.

این فصل ویژگی‌های زبان C# را برای تعریف کلاس‌های جدید شرح می‌دهد. اجزاء یک کلاس از قبیل رفتارهای آن و حالت آن به اعضای کلاس معروف هستند. رفتار کلاس با نوشتن متدها ایجاد می‌شود. یک متد امر عادی است که هر شیئی از آن کلاس می‌تواند اجرا کند. برای مثال، یک کلاس Dog می‌تواند متدی به نام Bark داشته باشد و کلاس ListBox می‌تواند متدی به نام Sort داشته باشد.

حالت کلاس با فیلدها نگهداری می‌شود. ممکن است فیلدها از انواع داده اولیه یا اشیائی از کلاس‌های دیگر باشند.

در نهایت، ممکن است کلاس خصوصیات خاصی داشته باشد که برای سرویس‌دهنده شبیه متدها عمل می‌کنند، اما برای سرویس‌گیرنده کلاس شبیه فیلدهای کلاس به نظر می‌رسند. یک سرویس‌گیرنده، شیئی است که با هر نمونه از کلاسی تعامل دارد.

۵-۱- تعریف کلاس

در زمان تعریف یک کلاس، ویژگی‌های همه اشیاء کلاس را به خوبی رفتارهایشان تعریف می‌کنید. برای مثال، اگر پنجره‌سازی سیستم‌عامل خودتان را ایجاد می‌کنید، ممکن است بخواهید چیزهای صفحه نمایش را ایجاد کنید. کنترل جالب ListBox در نمایش لیستی از انتخاب‌ها و توانایی کاربر برای انتخاب از لیست مفید است. ListBox ها ویژگی‌های متعددی همچون طول، عرض، موقعیت و رنگ متن دارند.

برنامه‌نویس‌ها رفتارهای معینی از کادرلیست‌ها انتظار دارند. آنها می‌توانند باز، بسته، مرتب و غیره شوند.

برنامه‌نویسی شی‌گرا به شما اجازه می‌دهد نوع داده‌ی جدیدی بنام ListBox ایجاد کنید که این ویژگی‌ها و توانایی‌ها را کپسوله می‌کند.

برای تعریف یک نوع داده یا کلاس جدید، ابتدا آن را اعلان کرده و سپس متدها و فیلدهای آن را تعریف کنید. یک کلاس را با استفاده از کلمه‌ی کلیدی class اعلان کنید. گرامر کامل آن به صورت زیر است.

```
[attributes] [access-modifiers] class identifier [:base-class] {class-body}
```

صفات^۲ برای فراهم کردن متاداده خاصی درباره یک کلاس استفاده می‌شوند (اطلاعاتی درباره ساختار یا کاربرد کلاس). صفات را برای برنامه‌نویسی عادی C# نیاز ندارید. معرف‌های دسترسی بعداً بررسی می‌شوند (معمولاً کلاس‌های شما از کلمه کلیدی public به عنوان معرف دسترسی خود استفاده می‌کنند).

شناسه‌ی^۳ کلاس همان نام کلاس است. کلاس‌های C# با اسامی نامگذاری می‌شوند (سگ، کارمند، کادرلیست). قوانین نامگذاری از نمادگذاری پاسکال استفاده می‌کنند. در نمادگذاری پاسکال خط فاصله یا زیرخط استفاده نمی‌کنند و در شناسه‌های یک یا چندکلمه‌ای، حرف اول هر کلمه را بزرگ می‌نویسند (GoldenRetriever).

همانطور که قبلاً گفته شد. وراثت یکی از ارکان برنامه‌نویسی شی‌گرا است. گزینه اختیاری کلاس پایه در بحث وراثت شرح داده می‌شود. تعریف اعضای کلاس در بدنه‌ی کلاس و در داخل آکولادهای باز و بسته قرار می‌گیرند.

```
class Dog
{
    int age; // the dog's age
    int weight; // the dog's weight
    Bark( ) { //... }
```

^۱ listBox

^۲ Attributes

^۳ identifier

```
Eat( ) { // ... }
}
```

متدهای موجود در تعریف کلاس Dog، همه رفتار سگ را شرح می‌دهند. فیلدهایی همچون سن و وزن، حالت یا همه خصوصیات سگ را شرح می‌دهند.

۵-۱-۱- نمونه‌سازی^۱ اشیاء

برای ایجاد یک نمونه‌ی واقعی یا یک شی از کلاس سگ، باید شی را اعلان کرده و برای شی حافظه تخصیص دهید. این دو مرحله برای ایجاد یک شی یا نمونه‌سازی لازم هستند. برای اعلان یک شی، ابتدا نام کلاس و به دنبال آن نام شی را بنویسید.

```
Dog milo; // declare milo to be an instance of Dog
```

این عمل به ایجاد یک متغیر محلی شباهت ندارد. در تعریف شی از قرارداد نمادگذاری Camel استفاده می‌کنیم. پس نام یک متغیر یا شی می‌تواند مانند myDog باشد.

عمل اعلان به تنهایی یک نمونه‌ی واقعی ایجاد نمی‌کند. برای ایجاد یک نمونه از یک کلاس، باید با استفاده از کلمه کلیدی new به آن شی حافظه تخصیص دهیم.

```
milo=new Dog();//allocate memory for milo
```

می‌توانید اعلان یک شی و تخصیص حافظه به آن را ترکیب کنید و در یک خط بنویسید.

```
;()Dog milo=new Dog
```

این کد milo را به عنوان شیئی از نوع Dog اعلان می‌کند و یک نمونه‌ی جدید از Dog را ایجاد می‌کند. دلیل وجود پرانتزها را بعداً خواهید فهمید.

در C# هر چیزی بوسیله‌ی یک کلاس اتفاق می‌افتد. هیچ متدی نمی‌تواند خارج از کلاس اجرا شود. متد Main() نقطه‌ی ورود برنامه‌ی شما است، که به وسیله‌ی سیستم عامل فراخوانی می‌شود و آن جایی است که اجرای برنامه آغاز می‌شود. شما یک کلاس کوچک برای جا دادن متد Main() ایجاد خواهید کرد، چون متد Main() شبیه هر متد دیگر باید در یک کلاس باشد.

```
public class Tester
{
public static void Main( )
{
//...
}
}
```

اگرچه کلاس Tester برای جا دادن متد Main() ایجاد شده است، شما هنوز هیچ نمونه‌ای از نوع Tester ندارید. برای این کار خواهیم نوشت:

```
;()Tester myTester = new Tester
```

ایجاد یک نمونه از کلاس Tester، فراخوانی متدهای دیگر کلاس را از طریق شی ایجادشده ممکن می‌سازد.

^۱ Instantiate

حال، کلاسی که زمان را پیگیری و نمایش می‌دهد، ملاحظه کنید. حالت داخلی کلاس باید قادر به نمایش سال، ماه، روز، ساعت، دقیقه و ثانیه جاری باشد. ممکن است این کلاس را برای نمایش زمان در قالب‌های مختلف بکار ببرید.

چارچوب .NET یک کلاس عملیاتی به نام *DateTime* فراهم کرده است.

شما این کلاس را با تعریف یک متد و ۶ متغیر (مثال ۵-۱) پیاده‌سازی کنید.

مثال ۵-۱ کلاس Time

```
using System;
public class Time
{
    // private variables
    private int year;
    private int month;
    private int date;
    private int hour;
    private int minute;
    private int second;
    // public methods
    public void DisplayCurrentTime( )
    {
        Console.WriteLine( "stub for DisplayCurrentTime" );
    }
}
public class Tester
{
    static void Main( )
    {
        Time timeObject = new Time( );
        timeObject.DisplayCurrentTime( );
    }
}
```

این کد یک نوع داده جدید کاربری به نام *Time* ایجاد می‌کند. تعریف کلاس *Time* با اعلان تعدادی متغیر عضو *second* *minute* , *hour* , *day* , *month* , *year* شروع می‌شود.

کلمه‌ی کلیدی *private* تعیین می‌کند که این فیلدها فقط در متدهای داخل کلاس قابل استفاده هستند. کلمه کلیدی *private* یک معرف دسترسی است.

بیشتر برنامه نویسان #C ترجیح می‌دهند اعلان فیلدها در کنار هم (بالا یا پایین) کلاس قرار گیرند. البته ضروری نیست.

تنها متد اعلان‌شده در کلاس *Time*، متد *DisplayCurrentTime* است. این متد مقدار *void* بر می‌گرداند، به عبارت دیگر هیچ مقدار برگشتی ندارد.

ممکن است بخواهیم در ابتدای برنامه‌نویسی، ساختار کلی برنامه را بدون در نظر گرفتن جزئیات آن ایجاد کنیم، در اینصورت یک دستور ساده در متدها می‌نویسیم که نشان‌دهنده‌ی اجرای این متد باشد. مانند متد `DisplayCurrentTime` که فقط یک پیام را در خروجی چاپ می‌کند.

بعد از کلاس `Time`، کلاس `Tester` تعریف می‌شود که متد `Main()` را در بردارد. در `Main()` یک نمونه از `Time` بنام `timeObject` ایجاد می‌شود.

```
Time timeObject = new Time( );
```

چون `timeObject` یک نمونه از کلاس `Time` است، متد `Main()` می‌تواند از متد `DisplayCurrentTime` آن شی استفاده کرده و آن را برای نمایش زمان فراخوانی کند.

```
timeObject.DisplayCurrentTime( );
```

متدی از یک کلاس را با نوشتن نام شی و به دنبال آن نام متد (که با نقطه از هم جدا شده اند)، می‌توان احضار کرد و پارامترهای متد در داخل پرانتزها قرار می‌گیرند. اگر متدی پارامتر نداشته باشد، پرانتز خالی نیز اجباری است.

۵-۱-۳- معرفی‌های دسترسی

یک معرف دسترسی نحوه دستیابی به اعضای کلاس، از بیرون یا داخل کلاس را تعیین می‌کند. جدول ۱-۶ معرفی‌های دستیابی C# را خلاصه می‌کند.

جدول ۱-۵- معرفی‌های دستیابی

محدودیت‌ها ^۱	معرف دسترسی
بدون محدودیت. اعضای که با <code>public</code> علامت‌گذاری شده‌اند، برای هر متد از هر کلاسی نمایان هستند.	<code>public</code>
این اعضا فقط به متدهای همان کلاس نمایان هستند.	<code>private</code>
این اعضا برای متدهای همان کلاس و کلاس‌هایی که از آن مشتق شده‌اند، نمایان هستند	<code>protected</code>
برای همه کلاس‌های موجود در اسمبلی مربوطه نمایان هستند	<code>internal</code>
برای متدهای همان کلاس و کلاسی که از آن مشتق شده و در اسمبلی مربوطه قرار دارند، نمایان است.	<code>Protected Internal</code>
بخشی از واسط عمومی کلاس هستند. آنها نحوه‌ی رفتار کلاس را تعریف می‌کنند.	متدهای <code>public</code>
متدهای کمکی هستند که توسط متدهای <code>public</code> برای انجام کاری استفاده می‌شوند. چون کار داخل کلاس خصوصی است، نیاز نیست متدهای کمکی به کلاس‌های دیگر نمایان باشند.	متدهای <code>private</code>

^۱ Restrictions

توجه: اگر چند معرف دسترسی در چندین سطح روی یک عنصری اعمال شوند، محدودترین آنها در نظر گرفته می‌شود.

کلاس Time و متد DisplayCurrentTime() هر دو public اعلان می‌شوند تا هر کلاس دیگر بتواند از آن استفاده کند. اگر متد مورد نظر private باشد، در این حالت متدهای مربوط به کلاس‌های دیگر نمی‌توانند آنرا احضار کنند. در مثال ۶-۲ متد DisplayCurrentTime() از طریق یک متد کلاس Tester احضار شده است. این عمل معقول است، چون کلاس Time و متد DisplayCurrentTime هر دو بصورت public علامت‌گذاری شده‌اند.

۵-۲- آرگومان‌های متد

رفتار یک کلاس با متدهای آن کلاس تعریف می‌شود. برای انعطاف‌پذیر کردن متدها، می‌توانید پارامترهایی را تعریف کنید. پارامترها اطلاعاتی هستند که هنگام احضار متد به آن ارسال می‌شوند. پس به جای تعریف متدهای مختلف برای مرتب‌سازی صعودی و نزولی یک ListBox، می‌توانید در متد Sort() پارامتری تعریف کنید که نوع عمل مرتب‌سازی را به آن ارسال کند.

متدها می‌توانند هر تعداد پارامتر دریافت کنند. لیست پارامتر به دنبال نام متد در داخل پارانتهز قرار می‌گیرد. نوع هر پارامتر قبل از نام آن پارامتر تعیین می‌شود. برای مثال، اعلان زیر یک متد به نام MyMethod() تعریف می‌کند که مقدار void بر می‌گرداند و دو پارامتر می‌گیرد.

```
void MyMethod (int firstParam, Button secondParam)
{
// ...
}
```

در بدنه‌ی متد، پارامترها همانند متغیرهای محلی عمل می‌کنند. مثال ۵-۲ نحوه‌ی ارسال مقادیر به یک متد را نشان می‌دهد.

مثال ۵-۲

```
using System;
public class MyClass
{
public void SomeMethod( int firstParam, float secondParam )
{
Console.WriteLine("Here are the parameters received: {0}, {1}",firstParam,
secondParam );
}
}
public class Tester
{
static void Main( )
{
int howManyPeople = ۰;
float pi = ۳.۱۴f;
MyClass mc = new MyClass( );
mc.SomeMethod( howManyPeople, pi );
}
}
```

خروجی آن به صورت زیر است:

Here are the parameters received ۵.۳,۱۴

در متد فراخوانی‌کننده (Main) دو متغیر محلی howManyPeople و pi ایجاد و مقداردهی اولیه می‌شوند. این متغیرها به عنوان پارامترهایی به SomeMethod ارسال می‌شوند. کامپایلر متغیر howManyPeople را به firstParam و pi را به secondParam نگاشت می‌کند.

۵-۳- سازنده‌ها

توجه کنید که در مثال ۶-۱ دستور ایجاد شیئی از نوع Time شبیه احضار متد Time() است.

```
Time timeObject = new Time( );
```

در واقع در زمان نمونه‌سازی یک شی، یک متد احضار می‌شود، این متد سازنده نامیده می‌شود. زمان تعریف یک کلاس، تعریف سازنده‌ی آن دلخواه است. اما اگر این کار را انجام ندهید، کامپایلر یک سازنده‌ی پنهان و اتوماتیک فراهم خواهد کرد. کار سازنده، ایجاد یک نمونه از کلاس و قرار دادن آن در یک حالت معتبر است. قبل از اجرای سازنده، شی بصورت یک حباب حافظه است، بعد از اجرای سازنده، حافظه یک نمونه‌ی معتبر از کلاس را نگه می‌دارد.

در مثال ۵-۲ کلاس Time، سازنده تعریف نمی‌کند، پس کامپایلر به طور اتوماتیک آن را ایجاد می‌کند. سازنده‌ی فراهم شده توسط کامپایلر فقط شی را ایجاد می‌کند و هیچ کاری انجام نمی‌دهد. هر سازنده‌ای که آرگومانی ندارد، سازنده‌ی پیش فرض است. سازنده‌ی فراهم شده توسط کامپایلر هیچ آرگومانی ندارد.

اگر شما متغیرهای عضو را به طور صریح مقداردهی اولیه نکنید، آنها به مقادیر بی‌ضرر مقداردهی اولیه می‌شوند. جدول ۵-۲ مقادیر پیش فرض تخصیص یافته به انواع داده‌ی مختلف را لیست می‌کند.

جدول ۵-۲

نوع داده	مقدار پیش فرض
عددی	صفر
bool	false
char	'\0' null
Enum	.
Refrence	null

معمولاً شما سازنده‌ی کلاس‌تان را تعریف و آرگومان‌های آن را فراهم خواهید کرد تا سازنده بتواند حالت اولیه‌ی شی را تنظیم کند. در مثال ۵-۳ می‌توانید برای ایجاد یک شی با داده‌ی بامعنی، مقادیر سال، ماه و روز جاری را به سازنده ارسال کنید.

یک سازنده را شبیه هر متد دیگر اعلان کنید به استثناء موارد زیر:

۱- نام سازنده باید هم نام کلاس باشد.

۲- سازنده‌ها مقدار بازگشتی ندارند.

اگر بخواهید آرگومان‌هایی را به سازنده ارسال کنید، لیست آرگومان‌ها را شبیه هر متد دیگر اعلان کنید. مثال ۵-۳ یک سازنده برای کلاس Time اعلان می‌کند که ۶ آرگومان می‌پذیرد.

مثال ۵-۳

```
using System;
public class Time
{
    // private member variables
    int year;
    int month;
    int date;
    int hour;
```

```
int minute;
int second;
// public method
public void DisplayCurrentTime( )
{
    System.Console.WriteLine( "{0}/{1}/{2} {3}:{4}:{5}", month, date, year, hour,
                                minute, second );
}
// constructor
public Time( int theYear, int theMonth, int theDate, int theHour, int theMinute,
            int theSecond )
{
    year = theYear;
    month = theMonth;
    date = theDate;
    hour = theHour;
    minute = theMinute;
    second = theSecond;
}
}
public class Tester
{
    static void Main( )
    {
        Time timeObject = new Time( ۲۰۰۸, ۸, ۱, ۹, ۳۵, ۲۰ );
        timeObject.DisplayCurrentTime( );
    }
}
```

خروجی به صورت زیر است:

۹:۳۵:۲۰ ۲۰۰۸/۱/۸

در این مثال، سازنده یک دنباله از اعداد صحیح را گرفته و همه متغیرهای عضو را براساس این پارامترها مقداردهی اولیه می‌کند.

زمانی که سازنده پایان می‌یابد، شی Time وجود دارد و مقادیر آن مقداردهی اولیه شده‌اند. زمانی که متد DisplayCurrentTime() در Main() فراخوانی می‌شود، مقادیر نمایش داده می‌شوند. اگر متغیر عددی را مقداردهی نکنید، توسط کامپایلر به مقدار صفر مقداردهی اولیه می‌شود. به یاد دارید که انواع داده‌ی مقداری باید مقداردهی اولیه شده باشند، در غیر اینصورت سازنده به آنها مقادیر بی‌ضرر می‌دهد.

۴-۵- مقداردهنده‌های اولیه^۱

مقداردهی اولیه متغیرهای عضو بوسیله‌ی یک مقدار اولیه دهنده امکان‌پذیر است. به جای مقداردهی اولیه در سازنده، یک مقداردهنده اولیه را با تخصیص یک مقدار اولیه به یک عضو کلاس ایجاد می‌کنید.

```
Private int second = ۳۰ ; //initializer
```

فرض کنید در شی Time مقدار زمان مهم نیست، اما ثانیه‌ها همواره به مقدار ۳۰ مقداردهی اولیه می‌شوند. شاید کلاس Time را رونویسی کنید تا یک مقداردهنده اولیه، مقدار Second را مشخص کند. مثال ۴-۵ را ببینید.

مثال ۴-۵

```
using System;
public class Time
{
    // private member variables
```

^۱ Initializers

```

int year;
int month;
int date;
int hour;
int minute;
int second = ۳۰;
// public method
public void DisplayCurrentTime( )
{
    System.Console.WriteLine( "{۰}/{۱}/{۲} {۳}:{۴}:{۵}",month, date, year, hour,
                                minute, second );
}
// constructor
public Time( int theYear, int theMonth, int theDate,int theHour, int theMinute )
{
    year = theYear;
    month = theMonth;
    date = theDate;
    hour = theHour;
    minute = theMinute;
}
}
public class Tester
{
    static void Main( )
    {
        Time timeObject = new Time( ۲۰۰۸, ۸, ۱, ۹, ۳۰ );
        timeObject.DisplayCurrentTime( );
    }
}

```

خروجی شبیه زیر است

۹:۳۵:۳۰ ۲۰۰۸/۱/۸

اگر مقداردهنده‌ی اولیه را فراهم نکنید، سازنده هر متغیر صحیح را به صفر مقداردهی اولیه می‌کند. در حالت بالا، عضو Second به ۳۰ مقدار دهی اولیه می‌شود.

در بخش بعدی خواهیم دید که می‌توانید بیش از یک سازنده داشته باشید. اگر بخواهید در بیش از یک مورد از آنها Second را به ۳۰ مقداردهی اولیه کنید، به جای مقداردهی اولیه جداگانه در هر سازنده، استفاده از مقدار دهنده اولیه بهترین راه است.

۵-۵- کلمه کلیدی this

کلمه کلیدی this به نمونه جاری از یک شی ارجاع می‌کند. ارجاع this یک پارامتر پنهان در هر متد غیرایستای یک کلاس است (متدهای ایستا بعداً بررسی می‌شوند). معمولاً ارجاع this به سه روش استفاده می‌شود: روش اول برای تمایز پارامترهای هم نام با اعضای نمونه‌ی کلاس است.

```

public void SomeMethod (int hour)
{
    this.hour = hour;
}

```

در این مثال متد SomeMethod یک پارامتر هم نام عضوی از کلاس می‌گیرد. ارجاع this برای حل این ابهام بکار برده می‌شود. this.hour به متغیر عضو و hour به پارامتر ارجاع می‌کند.

می‌توانید ارجاع this را برای انتساب صریح بکار ببرید.


```
public void SetTime(year, month, date, newHour, newMinute, newSecond)
{
    this.year = year; // use of "this" required
    this.month = month; // required
    this.date = date; // required
    this.hour = hour; // use of "this" optional
    this.minute = newMinute; // optional
    second = newSecond; // also ok
}
```

در صورتی که یک پارامتر با عضوی از کلاس هم نام باشد، استفاده از `this` ضروری است، در غیر اینصورت اختیاری می‌باشد.

کاربرد دوم ارجاع `this` برای ارسال شی جاری به عنوان پارامتر به متد دیگری می‌باشد. به عنوان مثال :

```
Class SomeClass
{
    public void FirstMethod(OtherClass otherObject)
    {
        otherObject.SecondMethod(this);
    }
    // ...
}
```

این قطعه کد دو کلاس را بنا می‌نهد: `SomeClass` و `OtherClass`. یک متد بنام `FirstMethod` و `OtherClass` یک متد بنام `SomeMethod` دارد. متد `FirstMethod`، متد `SecondMethod` را احضار کرده و شی جاری را برای پردازش بعدی به آن ارسال می‌کند. برای این کار، ارجاع `this` را به عنوان پارامتر به آن ارسال می‌کند.

کاربرد دیگر `this` در `indexer` ها است .

۵-۶- اعضای نمونه و ایستا

فیلدها، خصوصیات و متدهای یک کلاس می‌توانند اعضای نمونه^۱ یا اعضا ایستا^۲ باشند. اعضای نمونه به نمونه‌های یک نوع داده اختصاص داده می‌شوند، در حالیکه اعضای ایستا به کلاسی اختصاص داده می‌شوند و به نمونه خاصی از کلاس اختصاص داده نمی‌شوند. متدها بطور پیش فرض متدهای نمونه هستند، مگر اینکه به طور واضح کلمه‌ی کلیدی `static` با آنها بکار برده شود.

اکثریت متدها، متدهای نمونه خواهند بود. متد نمونه بدین معنی است که یک عمل روی یک شی خاصی رخ می‌دهد. در حال حاضر به منظور توانایی احضار یک متد بدون داشتن یک نمونه از آن مناسب است که متد ایستا تعریف کنید.

بعد از اعلان یک کلاس، دسترسی به عضو ایستای آن ممکن است. برای مثال، فرض کنید یک کلاس بنام `Button` دارید و اشیای معرفی شده از این کلاس به نام های `btnUpdate` و `btnDelete` را دارید.

فرض کنید کلاس `Button` یک متد نمونه بنام `Draw()` و یک متد ایستا به نام `GetButtonCount()` دارد. کار متد `Draw()` ترسیم دکمه جاری است و کار متد `GetButtonCount` برگرداندن تعداد دکمه‌های قابل رویت روی فرم است.

دسترسی به یک متد نمونه از طریق یک نمونه از آن شی انجام می‌شود.

```
btnUpdate.SomeMethod( );
```

یک متد ایستا از طریق نام کلاس نه نام یک نمونه دستیابی می‌شود.

```
Button.GetButtonCount( );
```

^۱ Instance

^۲ Static

۵-۶-۱- احضار متدهای ایستا

متدهای ایستا به جای یک نمونه از کلاس روی یک کلاس عمل می‌کنند. آنها نمی‌توانند کلمه‌ی کلیدی `this` را بکار ببرند، چون هیچ شی جاری در کار نیست.

متدهای ایستا نمی‌توانند مستقیماً به اعضای غیرایستا دسترسی داشته باشند. به خاطر دارید که `Main()` یک متد ایستا است. اگر `Main()` بخواهد به متد غیر ایستای هر کلاسی دسترسی داشته باشد، باید یک نمونه از آن کلاس را معرفی کند.

برای مثال بعدی `vs2005` را استفاده کرده و یک برنامه کاربردی کنسولی بنام `StaticTester` ایجاد کنید. `VS.NET` یک فضای نامی `StaticTester` و یک کلاس به نام `Class1` ایجاد می‌کند. `Class1` را به `Tester` تغییر نام دهید و همه توضیحات و صفت `[STAThread]` را که `VS.NET` در بالای `Main()` قرار می‌دهد حذف کنید. پارامتر `args` را از متد `Main()` حذف کنید. بعد از این کارها، کد شما به صورت زیر می‌باشد:

```
using System;
namespace StaticTester
{
    class Tester
    {
        static void Main( )
        {
        }
    }
}
```

آن نقطه شروع خوبی است. تا به حال، همه کارهای برنامه در متد `Main()` انجام می‌شد، اما حالا یک متد نمونه بنام `Run()` ایجاد خواهیم کرد. کار برنامه در متد `Run()` انجام خواهد شد.

متد نمونه جدید به نام `Run()` را در داخل کلاس اعلان کنید. در اعلان متد `Run`، معرف دسترسی `public` را نوشته و به دنبال آن نوع مقدار بازگشتی، شناسه و سپس پرانتزها را بنویسید.

```
public void Run( )
```

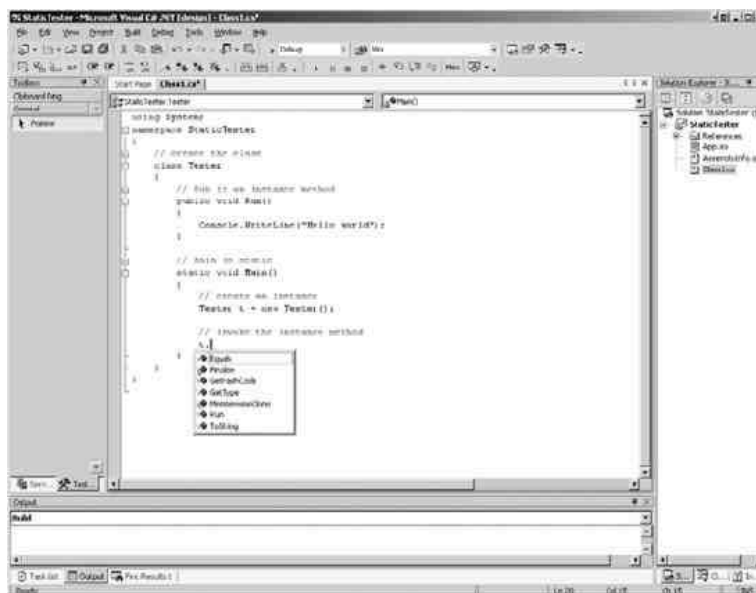
پرانتزها، پارامترها را نگه خواهند داشت، اما `Run()` هیچ پارامتری ندارد. پس می‌توانید پرانتزها را خالی رها کنید. آکولادها را برای متد باز کرده و دستور چاپ `"Hello world"` روی کنسول را بنویسید.

```
public void Run( )
{
    Console.WriteLine("Hello world");
}
```

`Run()` یک متد نمونه است و `Main()` یک متد ایستا است. پس نمی‌تواند مستقیماً `Run()` را احضار کند. بنابراین یک نمونه از کلاس `Tester` را ایجاد خواهید کرد و `Run()` را روی آن نمونه فراخوانی خواهید کرد.

```
;Tester t=new Tester
```

شکل ۵-۱



زمانی که کلمه کلیدی new را تایپ می‌کنید، سیستم هوشمند^۱ سعی می‌کند با نام کلاس به شما کمک کند. در خط بعدی متد Run () را روی شی t از کلاس Tester احضار کنید. هنگام تایپ عملگر نقطه بعد از t، سیستم هوشمند لیست متدهای کلاس Tester را نمایش می‌دهد.

زمانی که برنامه شما کامل شد، شبیه مثال ۵-۵ خواهد شد.

مثال ۵-۵

```
using System;
namespace StaticTester
{
    // create the class
    class Tester
    {
        // Run is an instance method
        public void Run( )
        {
            Console.WriteLine( "Hello world" );
        }
        // Main is static
        static void Main( )
        {
            // create an instance
            Tester t = new Tester( );
            // invoke the instance method
            t.Run( );
        }
    }
}
```

خروجی به صورت زیر است:

Hello world

این مدل را در بیشتر برنامه‌های کنسولی بکار خواهید برد. متد Main () به معرفی یک شی محدود می‌شود و سپس متد Run () را احضار می‌کند.

^۱ Intellisense

۵-۶-۲- کاربرد فیلدهای ایستا

یک کاربرد عمومی متغیرها یا فیلدهای ایستا، نگهداری تعداد نمونه‌های جاری موجود از یک کلاس است. در مثال بعدی، یک شی از کلاس Cat را برای شبیه‌سازی یک فروشگاه حیوانات اهلی ایجاد خواهید کرد.

در این مثال، فقط خصوصیات ضروری کلاس Cat آمده است. لیست کامل برنامه در مثال ۵-۶ نشان داده می‌شود.

مثال ۵-۶

```
using System;
namespace Test
{
    // declare a Cat class
    // stripped down
    public class Cat
    {
        // a private static member to keep
        // track of how many Cat objects have
        // been created
        private static int instances = 0;
        private int weight;
        private String name;

        // cat constructor
        // increments the count of Cats
        public Cat( String name, int weight )
        {
            instances++;
            this.name = name;
            this.weight = weight;
        }
        // Static method to retrieve
        // the current number of Cats
        public static void HowManyCats( )
        {
            Console.WriteLine( "{0} cats adopted", instances );
        }
        public void TellWeight( )
        {
            Console.WriteLine( "{0} is {1} pounds", name, weight );
        }
    }
    class Tester
    {
        public void Run( )
        {
            Cat.HowManyCats( );
            Cat frisky = new Cat( "Frisky", 0 );
            frisky.TellWeight( );
            Cat.HowManyCats( );
            Cat whiskers = new Cat( "Whisky", 7 );
            whiskers.TellWeight( );
            Cat.HowManyCats( );
        }
        static void Main( )
        {
            Tester t = new Tester( );
            t.Run( );
        }
    }
}
```

خروجی:



```
• cats adopted
Frisky is ۵ pounds
\ cats adopted
Whisky is ۷ pounds
۲ cats adopted
```

کلاس Cat با تعریف یک فیلد ایستا بنام instances شروع می‌شود، که با صفر مقداردهی اولیه می‌شود. این فیلد ایستا تعداد اشیاء Cat ایجاد شده را نگه می‌دارد. هر زمان که سازنده اجرا می‌شود، مقدار آن یک واحد افزایش می‌یابد.

کلاس Cat دو فیلد نمونه تعریف می‌کند: weight , name. این دو فیلد نام و وزن تک به تک اشیاء Cat را پی‌گیری می‌کنند.

کلاس Cat دو متد HowManyCats , TellWeight() تعریف می‌کند. متد HowManyCats() ایستا است. تعدادگره‌ها یک خصوصیت از یک گره نیست. آن یک خصوصیت از کل کلاس است. متد TellWeight() یک متد نمونه است، چون هر گره نام و وزن خودش را دارد. متد Main() از طریق کلاس Cat مستقیماً به متد HowManyCats دسترسی می‌کند.

```
Cat.HowManyCats( );
```

سپس Main() یک نمونه از Cat ایجاد می‌کند و از طریق نمونه‌ی Cat به متد نمونه TellWeight() دسترسی می‌کند.

```
Cat frisky = new Cat( )
frisky.TellWeight( );
```

هر زمانی که یک نمونه از کلاس Cat ایجاد می‌شود. HowManyCats() افزایش گره‌ها را گزارش می‌دهد.

۵-۷- خراب کردن اشیاء

#C برخلاف بیشتر زبان‌های برنامه‌نویسی ++C, C و پاسکال)، جمع‌آوری زباله را فراهم می‌کند. بعد از کار بر روی اشیاء، آنها بطور اتوماتیک خراب می‌شوند و نیازی نیست نگران پاک‌شدن اشیاء باشید. مگر اینکه منابع مدیریت‌نشده یا نادراً استفاده کرده باشید. یک منبع نادر، منبعی است که شما تعداد کمتری از آنرا در اختیار دارید (مانند اتصالات به پایگاه داده).

در صورت استفاده از یک منبع مدیریت‌نشده، باید به طور صریح آنرا آزاد کنید. کنترل صریح روی این منبع با یک مخرب فراهم می‌شود که در حین خراب کردن شی به وسیله جمع‌کننده‌ی زباله فراخوانی می‌شود.

مخرب #C را با علامت ~ اعلان کنید.

```
~{ } MyClass
```

این گرامر توسط کامپایلر به صورت زیر ترجمه می‌شود.

```
protected override void Finalize()
{
    try
    {
        // do work here
    }
    finally
    {
        base.Finalize();
    }
}
```

بدین دلیل بعضی از برنامه‌نویسان، مخرب را همانند یک تمام‌کننده^۱ می‌دانند. فراخوانی صریح یک مخرب نامعقول است. مخرب به وسیله جمع‌کننده زباله فراخوانی خواهد شد. اگر می‌خواهید خراب کردن منابع مدیریت نشده‌ای را اداره کنید، شما باید واسطه `IDisposable` را پیاده‌سازی کنید. بدین منظور لازم است یک متد به نام `Dispose` پیاده‌سازی کنید که به وسیله سرویس‌گیرنده‌ها فراخوانی خواهد شد.

در صورت فراهم کردن یک متد `Dispose()` باید جمع‌کننده زباله را از فراخوانی مخرب شی متوقف سازید. برای متوقف کردن جمع‌کننده زباله متد ایستای `GC.SuppressFinalize()` را فراخوانی کنید و `this` را به عنوان پارامتر به آن ارسال کنید. پس مخرب شی می‌تواند متد `Dispose()` شما را فراخوانی کند، احتمالاً می‌نویسید:

```
using System;
class Testing : IDisposable
{
    bool is_disposed = false;
    protected virtual void Dispose( bool disposing )
    {
        if ( !is_disposed ) // only dispose once!
        {
            if ( disposing )
            {
                Console.WriteLine( "Not in destructor,
                OK to reference other objects" );
            }
            // perform cleanup for this object
            Console.WriteLine( "Disposing..." );
        }
        this.is_disposed = true;
    }
    public void Dispose( )
    {
        Dispose( true );
        // tell the GC not to finalize
        GC.SuppressFinalize( this );
    }
    ~Testing( )
    {
        Dispose( false );
        Console.WriteLine( "In destructor." );
    }
}
```

در بعضی از اشیاء، سرویس‌گیرنده‌ها متد `Close()` را فراخوانی می‌کنند. می‌توانید یک متد `Dispose()` خصوصی و یک متد عمومی `Close()` ایجاد کنید و در متد `Close()` متد `Dispose()` را احضار کنید.

به دلیل عدم اطمینان از فراخوانی `Dispose()` توسط کاربر برنامه و اتمام غیرقطعی برنامه، #C یک دستور `using` برای اطمینان از فراخوانی `Dispose()` در نزدیک‌ترین زمان ممکن فراهم می‌کند. برای اشیایی که به کار می‌برید، برای آنها در بین آکولادها یک میدان ایجاد می‌کنید. زمانی که به آکولاد بسته رسیدیم، متد `Dispose()` به طور اتوماتیک روی آن شی فراخوانی خواهد شد. همانطور که در اینجا می‌بینید:

```
using System.Drawing;
class Tester
{
    public static void Main( )
    {
        using (Font theFont = new Font("Arial", ۱۰, f))
        {
            // use the font
        }
    }
}
```

^۱ Finalizer

}

}

چون ویندوز فقط تعداد کمی از اشیاء Font را مجاز می‌دارد، ما می‌خواهیم در نزدیکترین فرصت آن را از بین ببریم. در این قطعه کد، شی Font با دستور using ایجاد می‌شود. زمانی که دستور using پایان می‌یابد، فراخوانی Dispose() روی شی Font (تضمین می‌شود).

۵-۸- تخصیص حافظه

اشیاء ایجاد شده در متدها، متغیرهای محلی خوانده می‌شوند. آنها در متد به صورت محلی هستند. شی ایجاد شده در متد، در همان متد استفاده می‌شود و در زمان پایان متد خراب می‌شود. اشیاء محلی، بخشی از حالت شی نیستند، آنها مقادیر موقت را نگه می‌دارند.

متغیرهای محلی انواع داده اصلی، همچون int روی بخشی از حافظه بنام Stack ایجاد می‌شوند. زمانی که متدها احضار می‌شوند، در روی Stack به پارامترها و متغیرهای محلی آنها حافظه تخصیص داده می‌شود و در انتهای متد آزاد می‌شود. زمانی که یک متد شروع می‌شود، همه متغیرهای محلی روی Stack ایجاد می‌شوند. زمانی که متد پایان می‌یابد، متغیرهای محلی خراب می‌شوند.

این متغیرها، محلی بیان می‌شوند، چون فقط در طول زندگی متد وجود دارند (یعنی میدان محلی دارند). زمانی که متد پایان می‌یابد، متغیر از میدان بیرون رفته و خراب می‌شود.

#C انواع داده‌ها را به دو گروه مقداری و ارجاعی تقسیم می‌کند. انواع داده‌ی مقداری روی Stack ایجاد می‌شوند. همه انواع داده اصلی (int , long) انواع داده مقداری هستند و روی Stack ایجاد می‌شوند.

کلاسها انواع داده‌ی ارجاعی هستند. انواع داده ارجاعی روی یک بلاک مشتق شده از حافظه بنام Heap ایجاد می‌شوند. زمانی که یک نمونه از نوع داده ارجاعی اعلان می‌کنید، در واقع یک متغیر به یک شی دیگر اشاره می‌کند. ارجاع، همانند یک نام مستعار برای شی عمل می‌کند. پس زمانیکه شما می‌نویسید:

```
;(Dog milo=new Dog
```

عملکرد new یک شی Dog روی Heap ایجاد می‌کند و یک ارجاع به آن بر می‌گرداند. آن ارجاع به milo انتساب داده می‌شود. بنابراین milo یک شی ارجاعی است که به یک شی Dog روی Heap اشاره می‌کند. اما از نظر تکنیکی آن نادرست است. در واقع milo یک شی ارجاعی است که به یک شی Dog (بدون نام) روی Heap اشاره می‌کند. ارجاع milo همانند یک نام مستعار برای شی بدون نام عمل می‌کند. در اهداف عملی با milo همانند خود شی Dog برخورد می‌گردد.

مفهوم کاربرد ارجاع‌ها این است که می‌توانید به یک شی چندین ارجاع داشته باشید. برای فهم تفاوت ما بین انواع داده مقداری و ارجاعی مثال ۵-۷ را بررسی کنید. تحلیل کامل به دنبال مثال می‌آید.

مثال ۵-۷

```
using System;
namespace heap
{
public class Dog
{
public int weight;
}
class Tester
{
public void Run( )
{
// create an integer
```

```

int firstInt = ۰;
// create a second integer
int secondInt = firstInt;
// display the two integers
Console.WriteLine( "firstInt: {۰} secondInt: {۱}",firstInt, secondInt );
// modify the second integer
secondInt = ۷;
// display the two integers
Console.WriteLine( "firstInt: {۰} secondInt: {۱}",firstInt, secondInt );
// create a dog
Dog milo = new Dog( );
// assign a value to weight
milo.weight = ۰;
// create a second reference to the dog
Dog fido = milo;
// display their values
Console.WriteLine( "Milo: {۰}, fido: {۱}",milo.weight, fido.weight );
// assign a new weight to the second reference
fido.weight = ۷;
// display the two values
Console.WriteLine( "Milo: {۰}, fido: {۱}",milo.weight, fido.weight );
}
static void Main( )
{
    Tester t = new Tester( );
    t.Run( );
}
}
}

```

خروجی به صورت زیر است:

```

firstInt: ۰ secondInt: ۰
firstInt: ۰ secondInt: ۷
Milo: ۰, fido: ۰
Milo: ۷, fido: ۷

```

برنامه با ایجاد یک متغیر صحیح بنام firstInt و مقداردهی آن به ۵ آغاز می‌شود. متغیر دوم بنام secondInt را ایجاد و با firstInt مقداردهی می‌کند. مقادیر آنها در خروجی به صورت زیر نمایش داده می‌شوند:

```
firstInt: ۰ secondInt: ۰
```

شکل ۵-۲



این مقادیر یکسان هستند. چون int یک نوع داده‌ی مقداری است که یک کپی از مقدار firstInt ساخته و به secondInt انتساب می‌دهد. همانطور که شکل ۵-۲ نشان می‌دهد، متغیر secondInt یک متغیر مستقل است.

سپس برنامه یک مقدار جدید به secondInt انتساب می‌دهد.

```
secondInt = ۷;
```

چون این متغیرها از نوع مقداری بوده و مستقل از بقیه هستند، پس متغیر اول را تحت تاثیر قرار نمی‌دهد و فقط کپی آن تغییر داده می‌شود.

شکل ۵-۳



زمانی که داده‌ها نمایش داده می‌شوند، مقادیر آنها متفاوت هستند.

```
firstInt: ۵ secondInt: ۷
```

گام بعدی ایجاد یک کلاس ساده بنام Dog با فقط یک فیلد بنام weight است. این فیلد public بوده و هر متد از هر کلاسی می‌تواند به آن دسترسی داشته باشد. شما یک شی Dog را معرفی کرده و یک ارجاع به آن در milo ذخیره می‌کنید.

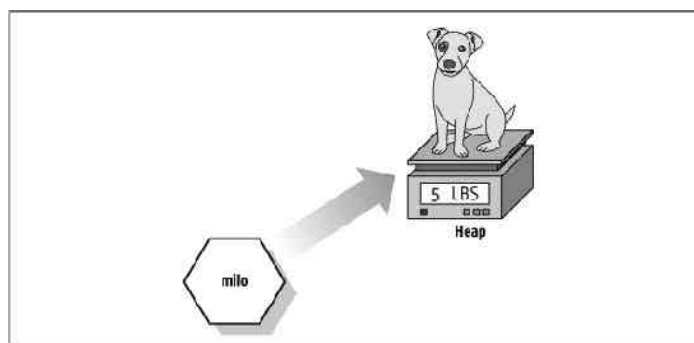
```
Dog milo = new Dog( );
```

مقدار ۵ را به فیلد وزن milo انتساب دهید.

```
milo.weight = ۵;
```

بطور معمول گفته می‌شود، وزن milo مقدار ۵ قرار داده شده است. اما در واقع وزن شی بدون نام روی heap که milo به آن ارجاع می‌کند مقداردهی شده است. همانطور که در شکل ۵-۴ نشان داده شده است.

شکل ۵-۴



در گام بعدی ارجاع دیگری به Dog ایجاد کرده و آنرا با milo مقداردهی کنید. این عمل ارجاع جدیدی به آن شی روی heap ایجاد می‌کند.

```
Dog fido = milo;
```

توجه کنید که قانون ایجاد این متغیر و مقداردهی اولیه‌ی آن، شبیه ایجاد متغیر secondInt و مقداردهی اولیه آن می‌باشد

```
int secondInt = firstInt;
```

```
Dog fido = milo;
```

با این تفاوت که چون Dog یک نوع داده‌ی ارجاعی است، پس fido یک کپی از milo نیست. آن یک ارجاع به همان شی milo است. پس همانطور که در شکل ۵-۵ می‌بینید، در حال حاضر یک شی با دو ارجاع‌کننده به آن روی Heap وجود دارد. زمانی که وزن شی fido را تغییر می‌دهید.

```
fido.weight = ۷;
```

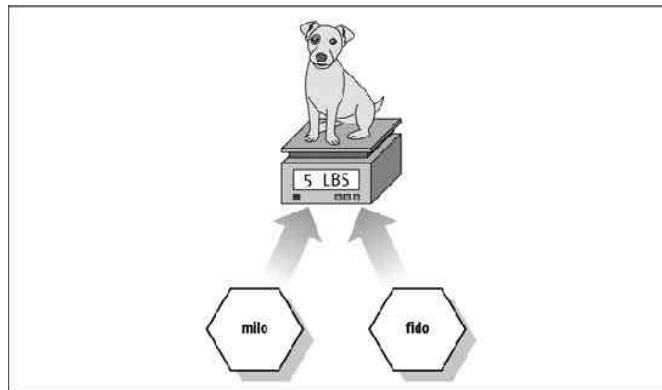
وزن شی مربوط به milo نیز تغییر می‌یابد و خروجی زیر منعکس می‌گردد.

```
Milo : ۷, Fido : ۷
```

شیء fido، شی milo را تغییر نمی‌دهد. آن شی بدون نام روی Heap را تغییر می‌دهد. پس بطور همزمان مقدار شی milo نیز تغییر می‌یابد.

توجه: اگر زمان ایجاد *fido* کلمه کلیدی *new* را بکار ببرید، یک نمونه‌ی جدید از *Dog* روی *Heap* ایجاد می‌کنید و *fido* و *milu* به شی یکسانی اشاره نمی‌کنند.

شکل ۵-۵



اگر کلاسی نیاز دارید که شبیه نوع داده مقداری رفتار کند، بایستی ساختار^۱ ایجاد کنید.

۵-۹- خلاصه

- زمان تعریف یک کلاس جدید، نام کلاس را با کلمه کلیدی *class* اعلان کرده و سپس متدها، فیلدها، نماینده‌ها^۲، رویدادها و خصوصیات آن را تعریف کنید.
- برای معرفی یک شی، همانند یک متغیر محلی نام شی را به دنبال نام کلاس اعلان کنید. سپس کلمه‌ی کلیدی *new* را برای تخصیص حافظه به آن شی روی *Heap* لازم دارید.
- فراخوانی یک متد روی شی با نوشتن عملگر نقطه بعد از نام شی و نام متد و پارامترهای آن بعد از نقطه امکان پذیر است.
- معرف‌های دسترسی نمایان بودن متدها و متغیرهای کلاس را برای کلاس‌های دیگر مشخص می‌کنند. همه اعضای کلاس برای همه متدهای آن کلاس نمایان هستند.
- اعضای که با *public* نشانه‌گذاری شده‌اند، هیچ محدودیتی ندارند و برای هر متد از هر کلاسی نمایان هستند.
- اعضای *private* فقط به متدهای همان کلاس نمایان هستند.
- اعضای *protected* به متدهای همان کلاس و متدهای کلاس مشتق شده از آن کلاس نمایان هستند.
- سازنده، متد خاصی است که زمان ایجاد شی احضار می‌شود. اگر سازنده‌ای برای کلاس خود فراهم نکنید، کامپایلر به طور اتوماتیک این کار را انجام می‌دهد. سازنده‌ی پیش فرض هیچ پارامتری ندارد.
- زمان تعریف اعضای کلاس می‌توانید به آنها مقدار اولیه دهید. کلمه کلیدی *this* به نمونه‌ی جاری یک شی اشاره می‌کند.
- یک متغیر صریح *this* به هر متد غیرایستای یک کلاس ارسال می‌شود.

^۱ Struct

^۲ Delegate

- اعضای ایستا به خود کلاس نه به نمونه‌ی خاصی از آن کلاس انتساب داده می‌شوند. اعضای ایستا با کلمه کلیدی `static` اعلان می‌شوند و از طریق نام کلاس احضار می‌شوند. متدهای ایستا کلمه کلیدی `this` را بکار نمی‌برند، چون هیچ نمونه‌ای از کلاس وجود ندارد.
- در `#C` متد مخرب ضروری نیست، چون چارچوب `.NET` به کمک `GC` هر شی بدون استفاده را خراب می‌کند.
- اگر یک کلاس از منابع مدیریت نشده استفاده کند، باید یک متد `Dispose()` فراهم کنید.
- متغیرهای محلی نوع مقداری روی `Stack` ایجاد می‌شوند. زمانی که متد پایان می‌یابد، این متغیرها از میدان خارج شده و خراب می‌شوند.
- اشیاء از نوع داده‌ی ارجاعی هستند و روی `Heap` ایجاد می‌شوند. زمان اعلان یک نمونه از نوع ارجاعی، در واقع یک اشاره‌گر به شی موجود در `Heap` ایجاد می‌کنید. اگر این ارجاع را در داخل یک متد اعلان کنید، پس از پایان یافتن متد، ارجاع مورد نظر خراب می‌شود. در صورتی که هیچ ارجاعی به آن شی روی `Heap` نمانده باشد، آن شی به وسیله `GC` خراب می‌شود.

وراثت و چند ریختی

آنچه که در این فصل یاد خواهید گرفت:

- مفاهیم ارث‌بری همچون تخصص و تعمیم
- تشخیص و ایجاد سلسله مراتب ارث‌بری
- دسترسی به سازنده‌های کلاس پایه
- چندریختی و نحوه پیاده‌سازی آن
- ایجاد نسخه‌های مختلف از یک متد به کمک `override` کردن متدها
- آشنایی با کلاس‌های انتزاعی و مهر شده و پیاده‌سازی این کلاس‌ها

در فصل‌های قبلی نحوه‌ی تعریف یک کلاس جدید و ارتباط ما بین کلاس‌ها، انجمن، تخصص^۱ و تجمع را بررسی کردیم. این فصل روی تخصص تمرکز دارد که از طریق وراثت در `#C` پیاده‌سازی می‌شود. در این فصل نحوه‌ی برخورد با کلاس‌های تخصصی بصورت کلاس‌های کلی پایه نیز شرح داده می‌شود که پروسه‌ی چندریختی^۲ نام دارد.

۶-۱- تخصص و تعمیم^۳

کلاس‌ها و نمونه‌های آنها در خلاء وجود ندارند، بلکه در یک دنیا از رابطه‌ها و دسته‌ها زندگی می‌کنند. تخصص، یکی از مهمترین رابطه‌های مابین اشیاء در دنیای واقعی است که بصورت یک رابطه `is-a` تشریح می‌شود. زمانی که می‌گوییم سگ یک پستاندار است، بدین معنی است که سگ نوع خاصی از پستاندار است و آن همه مشخصه‌های یک پستاندار را دارد. اما این مشخصه‌ها را به وسیله مشخصه‌های جنس سگ اختصاص می‌کنند. گربه نیز یک پستاندار است، پس انتظار داریم بعضی از ویژگی‌های سگ را به اشتراک گذارند که در عموم پستانداران وجود دارند.

^۱ Specialization

^۲ Polymorphism

^۳ Generalization

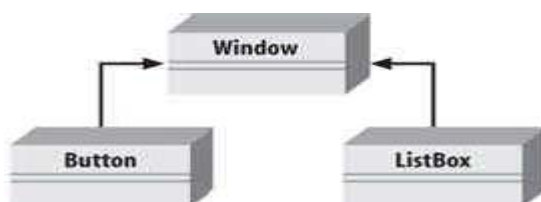
^۴ characteristic

رابطه‌های اختصاصی کردن و تعمیم دادن. سلسله مراتبی و دوسویه هستند. اختصاصی کردن روی دیگر سکه‌ی تعمیم است. پستاندار، موارد مشترک مابین سگ‌ها و گربه‌ها را تعمیم می‌دهد و سگ‌ها و گربه‌ها، پستانداران را به انواع داده خاص خود اختصاصی می‌کنند.

چون این رابطه‌ها یک درخت رابطه‌ای ایجاد می‌کنند، پس رابطه‌ها سلسله مراتبی هستند که انواع داده‌ها ی اختصاصی از انواع تعمیم شده انشعاب شده اند. همان طور که از سلسله مراتب به سمت بالا حرکت می‌کنید، تعمیم بیشتر مشاهده می‌کنید. همان طور که از سلسله مراتب به سمت پایین حرکت می‌کنید، آن را اختصاصی می‌کنید. پس گربه، پستاندار را در داشتن چنگ‌ها و خرخر کردن اختصاصی می‌کند.

بطور مشابه زمانیکه می‌گویید `ListBox`, `Button` پنجره هستند، نشان می‌دهید که ویژگی‌ها و رفتارهایی از پنجره وجود دارند که انتظار دارید در این دو نوع داده بیابند. بعبارت دیگر، پنجره، ویژگی‌های مشترک `ListBox`, `Button` را تعمیم می‌کند، در حالی که هر کدام ویژگی‌ها و رفتارهای مخصوص خود را اختصاصی می‌کنند.

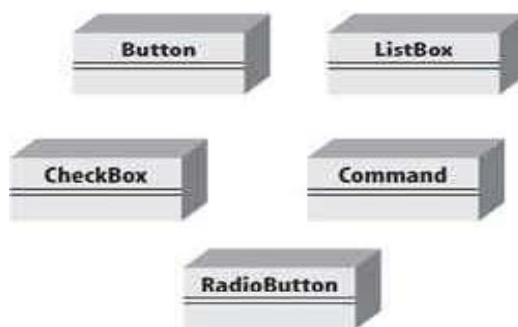
UML، یک زبان استاندارد برای تشریح یک سیستم شی‌گراست. UML چندین علامت بصری^۱ دارد که کلاس‌ها را با کادر مستطیلی نمایش می‌دهند. نام کلاس در بالای کادر ظاهر می‌گردد و متدها و اعضا در داخل کادر نمایش داده می‌شوند. در UML، روابط اختصاصی کردن را بصورت شکل ۶-۱ مدل کنید. به خط جهت‌دار از کلاس اختصاصی شده به کلاس تعمیم شده توجه کنید. در شکل زیر، کلاس‌های اختصاصی شده‌ی `Button` و `ListBox` به کلاس کلی `Window` اشاره می‌کنند.



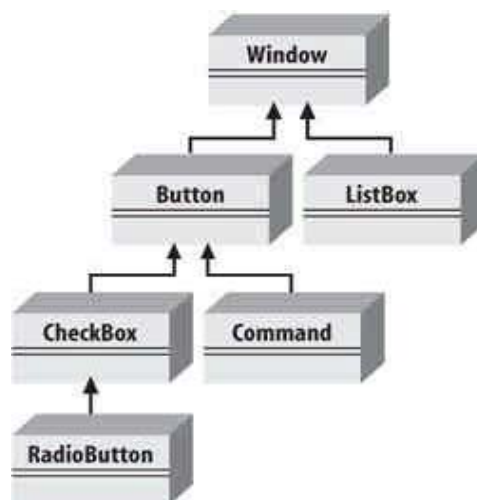
شکل ۶-۱- روابط اختصاصی کردن

به اشتراک گذاشتن عملکرد دو کلاس، غیرمعمول است. چون می‌توانید این مشترکات را در یک کلاس پایه قرار دهید که نسبت به کلاس‌های اختصاصی شده، کلی تر است. این عمل، استفاده مجدد کد مشترک را ممکن می‌سازد و نگهداری کد ساده‌تر می‌شود، چون به جای اینکه تغییرات در هر کلاس جداگانه انجام شود در کلاس منفردی رخ می‌دهد.

برای مثال: فرض کنید ایجاد یک دنباله از اشیاء (شکل ۶-۲) را آغاز کردید. بعد از کار با `CheckBox` , `RadioButton` , `CommandButton` ها می‌فهمید که آنها ویژگی‌ها و رفتارهای معینی را به اشتراک می‌گذارند که نسبت به پنجره اختصاصی‌تر هستند و ممکن است این رفتار و ویژگی‌های مشترک را به یک کلاس پایه مشترک بنام `Button` تجزیه کنید و سلسله مراتب وراثت را مجدداً مرتب کنید (شکل ۶-۳). این مثال نحوه‌ی استفاده از تعمیم در توسعه‌ی شی‌گرایی را نشان می‌دهد.



شکل ۶-۲- ایجاد یک دنباله از اشیاء



شکل ۶-۳- رابطه‌ی ما بین کلاس‌های تجزیه‌شده

دیاگرام UML شکل ۶-۳ رابطه‌ی ما بین کلاس‌های تجزیه‌شده را نشان می‌دهد و دو کلاس ListBox و Button مشتق شده از Window را نمایش می‌دهد که Button به Command و CheckBox اختصاصی می‌شود. در نهایت، RadioButton از CheckBox مشتق می‌شود.

این بهترین یا ضروری‌ترین روش سازمان‌دهی این اشیاء نیست، اما یک نقطه‌ی شروع معقول برای فهم نحوه‌ی ارتباط این نوع داده‌ها با دیگری است.

۶-۲- وراثت

در #C، رابطه‌ی اختصاصی کردن با استفاده از یک قاعده بنام وراثت پیاده‌سازی می‌شود. این تنها راه پیاده‌سازی اختصاصی کردن نیست، بلکه عمومی‌ترین و طبیعی‌ترین راه پیاده‌سازی این رابطه است. این گفته که ListBox از Window، ارث‌بری می‌کند، نشان می‌دهد که آن window را اختصاصی می‌کند.

Window یک کلاس پایه^۱ بیان می‌شود و ListBox به عنوان یک کلاس مشتق شده اشاره می‌شود و ListBox ویژگی‌ها و رفتار Window را مشتق گرفته و نیازهای خاص خود را اختصاصی می‌کند. اغلب به کلاس پایه، کلاس پدر^۲ گفته می‌شود و کلاس مشتق شده، کلاس فرزند^۳ خوانده می‌شود و بالاترین کلاس (object) ریشه گفته می‌شود.

^۱ Base class

^۲ Parent

^۳ Child

#c یک کلاس مشتق شده را با اضافه کردن یک کالان بعد از نام کلاس که به دنبال آن نام کلاس پایه قرار می گیرد، ایجاد می کند.

```
public class ListBox : Window
```

این کد یک کلاس جدید بنام ListBox اعلان می کند که از Window مشتق می شود. می توان کالان را بصورت "مشتق می شود از" خواند.

کلاس مشتق شده همگی اعضای کلاس پایه را به ارث می برد و متدهای کلاس مشتق شده به همگی اعضای عمومی و حفاظت شده ی کلاس پایه دسترسی دارند. کلاس مشتق شده در پیاده سازی نسخه ی جدید خود از متد کلاس پایه آزاد است. این عمل، پنهان کردن متد کلاس پایه نامیده می شود و از طریق علامت گذاری متد با کلمه ی کلیدی new انجام می شود.

کلمه ی کلیدی new نشان می دهد که کلاس مشتق شده متدی از کلاس پایه را بطور عمدی پنهان و جایگزین کرده است. همان طور که در مثال ۶-۱ می بینید:

مثال ۶-۱

```
using System;
public class Window
{
    // constructor takes two integers to
    // fix location on the console
    public Window( int top, int left )
    {
        this.top = top;
        this.left = left;
    }
    // simulates drawing the window
    public void DrawWindow( )
    {
        Console.WriteLine( "Drawing Window at {0}, {1}",top, left );
    }
    // these members are private and thus invisible
    // to derived class methods; we'll examine this
    // later in the chapter
    private int top;
    private int left;
}
// ListBox derives from Window
public class ListBox : Window
{
    // constructor adds a parameter
    public ListBox( int top, int left, string theContents ) :
        base( top, left ) // call base constructor
    {
        mListBoxContents = theContents;
    }
    // a new version (note keyword) because in the
    // derived method we change the behavior
    public new void DrawWindow( )
    {
        base.DrawWindow( ); // invoke the base method
        Console.WriteLine( "Writing string to the listbox: {0}",mListBoxContents );
    }
    private string mListBoxContents; // new member variable
}
public class Tester
{
    public static void Main( )
```

```
{
// create a base instance
Window w = new Window( ۰, ۱۰ );
w.DrawWindow( );
// create a derived instance
ListBox lb = new ListBox( ۲۰, ۳۰, "Hello world" );
lb.DrawWindow( );
}
}
```

خروجی بصورت زیر است:

```
Drawing Window at ۰, ۱۰
Drawing Window at ۲۰, ۳۰
Writing string to the listBox: Hello world
```

مثال ۱-۶ با اعلان کلاس پایه‌ی Window شروع می‌شود. این کلاس یک سازنده و یک متد ساده به نام DrawWindow() پیاده‌سازی می‌کند. دو فیلد خصوصی top و left وجود دارند. این برنامه در بخش‌های بعدی بطور دقیق تحلیل می‌شود.

۶-۲-۲- فراخوانی سازنده‌های کلاس پایه

در مثال ۱-۶ کلاس جدید ListBox از Window مشتق می‌شود و سازنده‌ی خود را دارد که سه پارامتر می‌گیرد. سازنده‌ی ListBox، سازنده‌ی پدر خود را با قرار دادن کالن (:) بعد از لیست پارامترهای خود احضار می‌کند و سپس سازنده‌ی کلاس پایه‌ی خود را با کلمه‌ی کلیدی base احضار می‌کند.

```
public ListBox( int theTop, int theLeft, string theContents):
base(theTop, theLeft) // call base constructor
```

چون کلاس‌ها نمی‌توانند سازنده را به ارث ببرند، یک کلاس مشتق شده باید سازنده‌ی خود را پیاده‌سازی کند و فقط می‌تواند سازنده‌ی کلاس پایه‌ی خود را بطور صریح فراخوانی کند. اگر کلاس پایه، یک سازنده‌ی پیش فرض دارد، نیاز نیست سازنده‌ی کلاس مشتق شده آن را به طور صریح احضار کند. البته سازنده‌ی پیش فرض هنگام ایجاد شی بطور غیرصریح ساخته می‌شود. با این وجود، اگر کلاس پایه، سازنده‌ی پیش فرض نداشته باشد، هر سازنده‌ی کلاس مشتق شده باید به طور صریح یکی از سازنده‌های کلاس پایه‌ی خود را احضار کند. کلمه‌ی کلیدی base، کلاس پایه را برای شی جاری تعیین می‌کند.

۶-۲-۳- کنترل دسترسی

می‌توانید میدان دید یک کلاس و اعضای آن را از طریق کاربرد معرف‌های دسترسی همچون public و private و protected محدود کنید.

کلاس‌ها همانند اعضای خود می‌توانند با هر سطح دستیابی طرأحی شوند. اگر یک عضو کلاس معرف دسترسی مختلفی نسبت به خود کلاس داشته باشد، حداقل دسترسی به آن اعمال می‌شود. پس اگر کلاس MyClass را بصورت زیر تعریف کنید:

```
public class MyClass
{
// ...
protected int myValue;
}
```

حتی اگر خود کلاس public باشد، قابلیت دستیابی به myValue محافظت شده است. یک کلاس public به هر کلاس دیگر که می‌خواهد با آن تعامل داشته باشد نمایان است. اگر یک کلاس دیگر بنام MyOtherClass ایجاد کنید که از MyClass مشتق می‌شود.

```
public class MyClass : MyOtherClass
{
Console.WriteLine("myInt: {۰}", myInt);
}
```


کلاس MyOtherClass می تواند به myInt دسترسی داشته باشد، چون MyOtherClass از MyClass مشتق می شود و هر کلاسی که از MyClass مشتق نشده باشد، نمی تواند به myInt دسترسی داشته باشد.

توجه: ایجاد متدها و خصوصیات محافظت شده نسبت به ایجاد فیلدهای محافظت شده معمول تر است. تقریباً فیلدها همواره خصوصی هستند.

۳-۶- چند ریختی

دو جنبه‌ی قدرتمند برای وراثت وجود دارد: یکی قابلیت استفاده‌ی مجدد است. زمانی که کلاس ListBox را ایجاد کردید، قادر هستید بعضی از منطق کلاس پایه را مجدداً استفاده کنید. جنبه‌ی دوم وراثت، چندریختی است. ریخت به معنی شکل است. پس چندریختی به قابلیت استفاده از چندین شکل یک نوع داده بدون توجه به جزئیات آن اشاره دارد.

زمانی که شرکت مخابرات یک سیگنال زنگ به تلفن شما ارسال می کند، نوع تلفن شما در طرف دیگر خط را نمی شناسد. ممکن است یک تلفن سنتی قدیمی یا یک تلفن الکترونیکی جدید باشد. آن فقط نوع پایه تلفن را می شناسد و انتظار دارد هر نمونه‌ی مشتق شده از این نوع داده نحوه زنگ خوردن خود را بداند. زمانی که شرکت مخابرات به تلفن شما سیگنال زنگ ارسال می کند، فقط متد زنگ تلفن شما را فراخوانی می کند. در اصل با تلفن شما بصورت چندریختی رفتار می کند.

۳-۶-۱- ایجاد انواع داده‌ی چندریختی

چون ListBox یک Window است و Button نیز یک Window است، شما انتظار دارید در هر جایی که می خواهید پنجره را بکار ببرید، قادر باشید یکی از این دو را بکار ببرید. برای مثال احتمال دارد یک فرم یک کلکسیون از همه نمونه‌های مشتق شده از پنجره که تحت مدیریت خود است را نگه دارد. زمانی که پنجره باز می شود، آن می تواند از هر پنجره بخواهد که خودش را رسم کند. برای این عمل، فرم لازم ندارد که نوع داده‌ی اصلی مشتق شده را بداند. خلاصه اینکه فرم می خواهد با اشیاء پنجره بصورت چندریختی رفتار کند.

چندریختی را در دو مرحله پیاده سازی کنید:

۱- یک کلاس پایه با متدهای مجازی ایجاد کنید.

۲- کلاس‌های مشتق شده‌ای ایجاد کنید، که رفتار متدهای مجازی کلاس پایه را override می کنند.

برای ایجاد یک متد در کلاس پایه که چندریختی را پشتیبانی کند، آن متد را بصورت virtual علامت گذاری کنید. برای مثال، جهت نشان دادن اینکه متد DrawWindow() در کلاس پنجره‌ی مثال ۶-۱ چندریخت است. کلمه‌ی کلیدی virtual را بصورت زیر به اعلان آن متد اضافه کنید.

```
public virtual void DrawWindow( )
```

هر کلاس مشتق شده برای ارث‌بری و کاربرد متد DrawWindow() کلاس پایه، بوسیله پیاده‌سازی نسخه‌ی جدید خود از DrawWindow() آزاد است. اگر یک کلاس مشتق شده متد DrawWindow() را override کند. نسخه‌ی override کننده برای هر نمونه از کلاس مشتق شده احضار خواهد شد. متد مجازی کلاس پایه را با استفاده از کلمه‌ی کلیدی override در تعریف متد کلاس مشتق شده override کنید و سپس کد تغییر یافته را به متد override کننده اضافه کنید. مثال ۶-۲ نحوه‌ی override کردن متدهای مجازی را نشان می دهد.

مثال ۶-۲- متدهای مجازی

```
using System;
public class Window
{
// constructor takes two integers to
```

```

// fix location on the console
public Window( int top, int left )
{
    this.top = top;
    this.left = left;
}
// simulates drawing the window
public virtual void DrawWindow( )
{
    Console.WriteLine( "Window: drawing Window at {·}, {\\}",top, left );
}
// these members are protected and thus visible
// to derived class methods. We'll examine this
// later in the chapter. (Typically, these would be private
// and wrapped in protected properties, but the current approach
// keeps the example simpler.)
protected int top;
protected int left;
} // end Window
// ListBox derives from Window
public class ListBox : Window
{
    // constructor adds a parameter
    // and calls the base constructor
    public ListBox(int top,int left,string contents ) : base( top, left )
    {
        listBoxContents = contents;
    }
    // an overridden version (note keyword) because in the
    // derived method we change the behavior
    public override void DrawWindow( )
    {
        base.DrawWindow( ); // invoke the base method
        Console.WriteLine( "Writing string to the listbox: {·}",listBoxContents );
    }
    private string listBoxContents; // new member variable
} // end ListBox
public class Button : Window
{
    public Button(
        int top,
        int left ) : base( top, left )
    {}
    // an overridden version (note keyword) because in the
    // derived method we change the behavior
    public override void DrawWindow( )
    {
        Console.WriteLine( "Drawing a button at {·}, {\\}\\n",top, left );
    }
} // end Button
public class Tester
{
    static void Main( )
    {
        Window win = new Window( \\, ٢ );
        ListBox lb = new ListBox( ٣, ٤, "Stand alone list box" );
        Button b = new Button( ٥, ٦ );
        win.DrawWindow( );
        lb.DrawWindow( );
        b.DrawWindow( );
        Window[] winArray = new Window[٣];
        winArray[·] = new Window( \\, ٢ );
        winArray[\\] = new ListBox( ٣, ٤, "List box in array" );
        winArray[٢] = new Button( ٥, ٦ );
        for ( int i = ٠; i < ٣; i++ )

```

```
{
winArray[i].DrawWindow( );
} // end for
} // end Main
} // end Tester
```

خروجی بصورت زیر است:

```
Window: drawing Window at ۱, ۲
Window: drawing Window at ۳, ۴
Writing string to the listbox: Stand alone list box
Drawing a button at ۵, ۶
Window: drawing Window at ۱, ۲
Window: drawing Window at ۳, ۴
Writing string to the listbox: List box in array
Drawing a button at ۵, ۶
```

در مثال ۶-۲، ListBox از پنجره مشتق می‌شود و نسخه‌ی DrawWindow() خود را پیاده‌سازی می‌کند.

```
public override void DrawWindow( )
{
base.DrawWindow( ); // invoke the base method
Console.WriteLine ("Writing string to the listbox: {۰}", listBoxContents);
}
```

کلمه‌ی کلیدی override به کامپایلر می‌گوید این کلاس بطور عمدی نحوه‌ی کار DrawWindow را override کرده است. بطور مشابه، متد DrawWindow() را در کلاس دیگر که از Window مشتق می‌شود، override خواهید کرد(کلاس Button).

در بدنه‌ی مثال، سه شیء از انواع Window، ListBox و Button ایجاد کرده و سپس DrawWindow() را روی هر کدام فراخوانی کرده است.

```
Window win = new Window(۱,۲);
ListBox lb = new ListBox(۳,۴,"Stand alone list box");
Button b = new Button(۵,۶);
win.DrawWindow( );
lb.DrawWindow( );
b.DrawWindow( );
```

این کد همان طور که شما انتظار دارید کار می‌کند. برای هر کدام متد DrawWindow() به طور صحیح فراخوانی می‌شود. هنوز مفهوم واقعی چندریختی اعمال نشده است. سحر واقعی زمانی شروع می‌شود که یک آرایه از اشیاء Window ایجاد کنید.

چون ListBox یک Window است، پس برای قرار دادن یک ListBox در آرایه‌ای از Window آزاد هستید. بطور مشابه می‌توانید یک Button را نیز به این آرایه اضافه کنید.

```
Window[] winArray = new Window[۳];
winArray[۰] = new Window(۱,۲);
winArray[۱] = new ListBox(۳,۴,"List box in array");
winArray[۲] = new Button(۵,۶);
```

خط اول یک آرایه بنام winArray برای نگه‌داشتن سه شیء Window تعریف می‌کند. سه خط بعدی اشیاء جدید Window را به آرایه اضافه می‌کنند.

زمانی که متد DrawWindow() را روی هر کدام از این اشیاء فرا می‌خوانید، چه اتفاقی می‌افتد؟

```
for (int i = ۰; i < winArray.Length-۱; i++)
{
winArray[i].DrawWindow( );
}
```

این کد یک متغیر شمارنده بنام i بکار می‌برد. آن متد DrawWindow() را روی هر عنصر آرایه فراخوانی می‌کند. کامپایلر می‌داند که آن سه شیء Window دارد و شما متد DrawWindow() را روی هر کدام از آنها فراخوانی کرده‌اید.

اگر متد DrawWindow() را بصورت virtual علامت‌گذاری نکرده بودید، در هر بار اجرا، همان متد اصلی کلاس Window سه بار فراخوانی می‌شد. با این وجود، چون متد DrawWindow() را به صورت مجازی علامت‌گذاری کردید و کلاس مشتق شده آن متد را override کرده است، زمانی که متد DrawWindow() را روی عناصر آرایه فراخوانی می‌کنید، همه چیز مطابق انتظار شما پیش می‌رود. کامپایلر نوع داده‌ها را در زمان اجرای اشیاء واقعی تعیین می‌کند و متد صحیح هر کدام را فراخوانی می‌کند و این ماهیت چندریختی است.

با توجه به اینکه، طبق این مثال، متدهای override کننده با کلمه‌ی کلیدی override علامت‌گذاری می‌شوند.

```
(public override void DrawWindow
```

حال کامپایلر می‌داند که زمان برخورد با این اشیاء چند ریختی متد override شده را بکار برد. کامپایلر مسئول پی‌گیری نوع واقعی اشیاء و اداره‌ی دیر مقید کردن^۱ است. بنابراین زمانی که ارجاع پنجره واقعاً به یک شیء ListBox اشاره می‌کند، متد ListBox.DrawWindow() فراخوانی می‌شود.

۶-۳-۲- نسخه‌سازی^۲ با new و override

برنامه‌نویس زبان C#، override کردن یک متد مجازی را با کلمه‌ی کلیدی override بصورت واضح انجام می‌دهد. این عمل در تولید نسخه‌های جدید از کد کمک می‌کند. تغییرات در کلاس پایه، کد موجود در کلاس‌های مشتق شده را شکست نخواهد داد. نیاز به کلمه‌ی کلیدی override در جلوگیری از این مشکل کمک می‌کند.

فرض کنید شرکت A کلاس پایه Window مثال ۶-۲ را نوشته است و کلاس‌های ListBox و Button توسط برنامه‌نویسان شرکت B نوشته شده‌اند، که یک کپی از کلاس Window شرکت A را به عنوان کلاس پایه خریده‌اند. برنامه‌نویسان شرکت B، روی طراحی کلاس Window کمی کنترل دارند یا اصلاً ندارند.

حال فرض کنید، یکی از برنامه‌نویسان شرکت B تصمیم دارد، متد Sort() را به ListBox اضافه کند.

```
public class ListBox : Window
{
    public virtual void Sort( ) {...}
}
```

این کد تا زمانی که شرکت A، نسخه‌ی دوم کلاس Window را منتشر نکرده، مشکلی ندارد. برنامه‌نویس شرکت A یک متد Sort() به کلاس عمومی Window خود اضافه می‌کند.

```
public class Window
{
    // ...
    public virtual void Sort( ) {...}
}
```

در زبان‌های شی‌گرای دیگر (همچون ++C)، متد مجازی جدید Sort() در کلاس Window، به عنوان یک متد مجازی پایه برای متد Sort() در ListBox عمل می‌کند و این چیزی نیست که توسعه‌دهنده‌ی ListBox قصد داشت.

C# از این پریشانی جلوگیری می‌کند. در C#، یک تابع virtual همواره بعنوان ریشه‌ی گسیل^۳ مجازی بررسی می‌گردد. بدین صورت، زمانی که C# یک متد مجازی می‌یابد، به سلسله مراتبی وراثت قبلی نگاه نمی‌کند. اگر یک متد مجازی جدید Sort() به Window اضافه گردد، رفتار زمان اجرای ListBox تغییر نمی‌یابد.

زمانی که ListBox مجدداً کامپایل می‌شود، کامپایلر یک هشدار تولید می‌کند.

^۱ Late binding

^۲ Versioning

^۳ Dispatch

```
... \class\cs(۵۴,۲۴): warning CS۰۱۱۴: 'ListBox.Sort( )' hides
inherited member 'Window.Sort( )'.
To make the current member override that implementation,
add the override keyword. Otherwise add the new keyword.
```

توجه: هرگز از هشدارها صرف نظر نکنید، آنها را همانند خطاهایی در نظر گرفته و بررسی کنید.

برای حذف کردن هشدار، برنامه‌نویس باید قصد خود را معین کند. او می‌تواند متد Sort کلاس ListBox خود را با new علامت‌گذاری کند تا نشان دهد آن یک override از متد مجازی پنجره نیست.

```
public class ListBox : Window
{
public new virtual void Sort( ) {...}
```

با این عمل، پیام هشدار را حذف می‌کنید. از طرف دیگر، اگر برنامه‌نویس می‌خواهد متد پنجره را override کند، فقط لازم است که کلمه‌ی کلیدی override را بطور صریح بکار برد.

```
public class ListBox : Window
{
public override void Sort( ) {...}
```

توجه: برای پرهیز از این هشدار ممکن است بخواهید کلمه‌ی new را به همه‌ی متد های مجازی خود اضافه کنید. این یک ایده‌ی بدی است. زمانی که new در کد ظاهر می‌گردد، آن می‌خواهد نسخه‌سازی کد را مستندسازی کند.

اگر برنامه‌نویس کلاسی از ListBox مشتق کند، این کلاس‌های مشتق شده، متد Sort() را از ListBox() نه از Window به ارث می‌برند.

۶-۴- کلاس‌های انتزاعی^۱

هر نوع Window، شکل و ظاهر متفاوتی دارد. لیست‌های باز شو با دکمه‌ها خیلی تفاوت دارند. بطور واضح هر زیرکلاسی از Window باید متد DrawWindow() خود را پیاده‌سازی کند. براساس کلاس window ما مجبور به این کار نیستیم. برای اینکه یک کلاس را به پیاده‌سازی یک متد از کلاس پایه مجبور کنیم، لازم است متد را بصورت انتزاعی طراحی کنیم.

یک متد مجازی، هیچ پیاده‌سازی ندارد. آن یک نام و نشانه^۲ ایجاد می‌کند که باید در همه‌ی کلاس‌های مشتق شده پیاده‌سازی شود. بعلاوه، ایجاد حداقل یک متد انتزاعی در هر کلاس، آن کلاس را انتزاعی می‌کند.

کلاس‌های انتزاعی، یک پایه برای کلاس‌های مشتق شده بنا می‌نهند. تعریف یک شی از کلاس انتزاعی نامعقول است. زمانی که متدی از یک کلاس را انتزاعی اعلان می‌کنید، ایجاد هر نمونه از آن کلاس را منع می‌کنید. اگر متد DrawWindow() را به عنوان یک متد انتزاعی در کلاس Window طراحی کنید، پس کلاس Window نیز انتزاعی می‌شود. پس می‌توانید از Window مشتق بگیرید، ولی نمی‌توانید نمونه‌هایی از خود آن ایجاد کنید.

ایجاد Window.DrawWindow() انتزاعی بدین معنی است که هر کلاس مشتق شده از Window باید متد DrawWindow() منحصر به خود را پیاده‌سازی کند. اگر کلاس مشتق شده، متد انتزاعی را پیاده‌سازی نکند، کلاس مشتق شده نیز انتزاعی خواهد شد و هیچ نمونه‌ای از آن امکان‌پذیر نیست.

طراحی یک متد بعنوان انتزاعی با قرار دادن کلمه‌ی کلیدی abstract در ابتدای تعریف متد انجام می‌شود.

```
abstract public void DrawWindow( );
```

^۱ Abstract

^۲ Signature

(چون متد هیچ پیاده‌سازی ندارد، آکولادهای باز و بسته ندارد و فقط یک ؛ بعد از آن قرار دارد). اگر در تعریف یک کلاس یک یا چند متد انتزاعی باشد، تعریف کلاس باید `abstract` علامت‌گذاری شود.

```
abstract public class Window
```

مثال ۳-۶ ایجاد یک کلاس `Window` انتزاعی و یک متد انتزاعی `DrawWindow()` را ارائه می‌کند.

مثال ۳-۶

```
using System;
public abstract class Window
{
    // constructor takes two integers to
    // fix location on the console
    public Window( int top, int left )
    {
        this.top = top;
        this.left = left;
    }
    // simulates drawing the window
    // notice: no implementation
    public abstract void DrawWindow( );
    protected int top;
    protected int left;
} // end class Window
// ListBox derives from Window
public class ListBox : Window
{
    // constructor adds a parameter
    public ListBox(int top,int left,string contents ) : base( top, left )
    {
        // call base constructor
        listBoxContents = contents;
    }
    // an overridden version implementing the
    // abstract method
    public override void DrawWindow( )
    {
        Console.WriteLine( "Writing string to the listbox: {0}",listBoxContents );
    }
    private string listBoxContents; // new member variable
} // end class ListBox
public class Button : Window
{
    public Button(
        int top,
        int left ) : base( top, left ) { }
    // implement the abstract method
    public override void DrawWindow( )
    {
        Console.WriteLine( "Drawing a button at {0}, {1}\n",top, left );
    }
} // end class Button
public class Tester
{
    static void Main( )
    {
        Window[] winArray = new Window[۳];
        winArray[۰] = new ListBox( ۱, ۲, "First List Box" );
        winArray[۱] = new ListBox( ۳, ۴, "Second List Box" );
        winArray[۲] = new Button( ۵, ۶ );
        for ( int i = ۰; i < ۳; i++ )
        {
            winArray[i].DrawWindow( );
        }
    }
}
```

```
} // end for loop
} // end main
} // end class Tester
```

خروجی بصورت زیر است:

```
Writing string to the listBox: First List Box
Writing string to the listBox: Second List Box
Drawing a button at ۰, ۱
```

در مثال ۶-۳، کلاس Window بصورت انتزاعی اعلان شده است و نمی توان نمونه ای از آن تولید کرد.. اگر اولین عضو آرایه را به جای دستور شماره ۱ با دستور ۲ جایگزین کنید:

```
winArray[۰] = new ListBox(۱,۲,"First List Box");    ۱
```

```
winArray[۰] = new Window(۱,۲);                    ۲
```

در زمان کامپایل خطای زیر تولید می گردد:

```
Cannot create an instance of the abstract class or interface 'Window'
```

اشیاء ListBox و Button را می توان نمونه سازی کرد، چون این کلاس ها متد انتزاعی را override می کنند. پس کلاس ها را واقعی (نه انتزاعی) می کنند.

اغلب یک کلاس انتزاعی، متدهای غیرانتزاعی نیز دارد. بطور معمول این متدها نیز با virtual علامت گذاری خواهند شد، تا برنامه نویسانی که از کلاس انتزاعی مشتق می گیرند، آن کد پیاده سازی شده در کلاس انتزاعی را انتخاب کنند یا آنها را override کنند. با این وجود، به منظور ایجاد یک نمونه از کلاس مشتق شده، باید تمامی متدهای انتزاعی override شوند.

۶-۵- کلاس های مهرشده^۱

طرف مقابل سکه ی طراحی انتزاعی، طراحی مهرشده است. برخلاف یک کلاس انتزاعی که باید کلاس هایی از آن مشتق شوند، یک کلاس مهرشده اجازه نمی دهد کلاسی از آن مشتق شود. کلمه ی کلیدی sealed قبل از اعلان کلاس قرار می گیرد تا مانع مشتق گرفتن شود. اغلب برای جلوگیری از وراثت تصادفی، کلاس ها را بصورت مهرشده علامت گذاری می کنند.

اگر در مثال ۶-۳ اعلان Window از abstract به sealed تغییر یابد. کامپایل برنامه شکست می خورد. اگر سعی کنید این پروژه را بسازید، کامپایلر خطای زیر را تولید می کند.

```
'ListBox' cannot inherit from sealed type 'Window'
```

زمانی که می دانید تولید کلاس های مشتق شده از یک کلاس را نیاز نخواهیم داشت و زمانی که کلاس شما فقط خصوصیات و متدهای ایستا دارد، مایکروسافت توصیه می کند sealed را بکار ببرید.

۶-۶- ریشه ی همه کلاس ها (Object)

همه ی کلاس های C#، نهایتاً از کلاس Object مشتق می شوند. Object کلاس پایه ی همه ی کلاس های دیگر است. یک کلاس پایه، پدر بلاواسطه ی یک کلاس مشتق شده است. یک کلاس مشتق شده می تواند پایه ی کلاس های مشتق شده ی دیگر باشد. کلاس ریشه، بالاترین کلاس یک درخت وراثت است. در object، C# کلاس ریشه است.

^۱ Sealed

کلاس Object چندین متد دارد. زیرکلاس‌ها می‌توانند آنها را override کنند. به عنوان مثال، متدهای Equals() (تعیین مساوی بودن دو شی) و ToString() (یک رشته برای نمایش شی جاری بر می‌گرداند) را شامل است. متد ToString() یک رشته با نام کلاسی که شی به آن تعلق دارد برمی‌گرداند. جدول ۱-۶ متدهای Object را خلاصه می‌کند.

جدول ۱-۶

متد	کار مربوطه
Equals ()	معادل بودن دو شی را ارزیابی می‌کند.
GetHashCode ()	اجازه می‌دهد اشیاء تابع hash خود را برای استفاده در کلکسیون‌ها فراهم کنند.
GetType ()	دسترسی به نوع داده‌ی شی را فراهم می‌کند.
ToString ()	یک نمایش رشته‌ای از نام نوع شی را فراهم می‌کند.
Finalize ()	منابع خارج از حافظه را پاک می‌کند. بوسیله‌ی یک مخرب طراحی می‌شود.

در مثال ۴-۶ کلاس Dog متد ToString() ارث‌بری شده از Object را override می‌کند تا وزن Dog را برگرداند.

مثال ۴-۶

```
using System;
public class Dog
{
    private int weight;
    // constructor
    public Dog( int weight )
    {
        this.weight = weight;
    }
    // override Object.ToString
    public override string ToString( )
    {
        return weight.ToString( );
    }
}
public class Tester
{
    static void Main( )
    {
        int i = ۰;
        Console.WriteLine( "The value of i is: {۰}", i.ToString( ) );
        Dog milo = new Dog( ۱۲ );
        Console.WriteLine( "My dog Milo weighs {۰} pounds", milo);
    }
}
Output:
The value of i is: ۰
My dog Milo weighs ۱۲ pounds
```


متدهای بعضی از کلاس‌ها (همچون Console) رشته‌ها را به کار می‌برند. این متدها، متد ToString() را روی کلاس شما فراخوانی خواهند کرد، اگر شما ToString() را override نکرده باشید، کلمه‌ی Dog به Console.WriteLine برگردانده می‌شود. مستندات متد Object.ToString() نشانه‌ی آن را فاش می‌کند:

```
public virtual string ToString( );
```

آن یک متد مجازی public است که هیچ پارامتری نمی‌گیرد و یک رشته بر می‌گرداند. همه‌ی انواع داده‌ای درونی (همچون int) که از Object مشتق می‌شوند، می‌توانند متدهای Object را احضار کنند.

توجه: اعلان صریح مشتق شدن از Object لازم نیست، چون وراثت از نوع ضمنی است.

۶-۷- خلاصه

- اختصاصی کردن به عنوان رابطه‌ی is-a توصیف می‌شود. عکس اختصاصی کردن، تعمیم است.
- اختصاصی کردن و تعمیم، متقابل و سلسله‌مراتبی هستند. اختصاصی کردن عمل متقابل تعمیم است. هر کلاس می‌تواند چندین کلاس اختصاصی مشتق شده از آن داشته باشد. پس یک حالت انشعاب ایجاد می‌کند.
- #C اختصاصی کردن را از طریق وراثت پیاده‌سازی می‌کند.
- کلاس مشتق شده ویژگی‌ها و رفتار public و protected کلاس پایه را به ارث می‌برد و در اضافه کردن یا تغییر دادن ویژگی‌ها و رفتار خود آزاد هستند.
- برای پیاده‌سازی وراثت نام کلاس پایه را بعد از یک کالن به دنبال نام کلاس مشتق شده بنویسید.
- یک کلاس مشتق شده می‌تواند سازنده‌ی کلاس پایه را با قرار دادن یک کالن بعد از لیست پارامترهای سازنده‌ی خود با کلمه‌ی کلیدی base احضار کند.
- کلاس‌ها می‌توانند شبیه اعضای خود معرف‌های دسترسی public, private و protected را بکار ببرند.
- متدی که در کلاس پایه به صورت virtual غلامت‌گذاری شده باشد، می‌تواند بوسیله‌ی کلاس‌های مشتق شده override گردد. اگر کلاس‌های مشتق شده کلمه‌ی کلیدی override را در تعریف متد بکار ببرند، پیاده‌سازی چندریختی انجام داده‌اند. زمانی که متد مجازی را روی هر شی مشتق شده فراخوانی کنید، رفتار override شده احضار می‌شود.
- یک متد abstract هیچ پیاده‌سازی ندارد. آن فقط نام و نشانه‌ی متد مجازی را فراهم می‌کند. این متد باید توسط کلاس مشتق شده override گردد. هر کلاسی که یک متد انتزاعی دارد، خود نیز انتزاعی است و نمی‌توان شیئی از آن ایجاد کرد.
- نمی‌توان از کلاس sealed مشتق گرفت.
- در #C نهایتاً همه‌ی کلاس‌ها از Object مشتق می‌شوند و تعدادی متد مفید را به ارث می‌برند.

متدهای داخلی

آنچه که در این فصل یاد خواهید گرفت:

- روش overload کردن متدها و استفاده از آنها
- استفاده از خصوصیات برای کیسوله کردن اطلاعات یک شی
- کار با پارامترهای out و ref و نقش آنها در ارسال داده‌ها به متدها
- معاون‌های get و set برای دسترسی یا مقداردهی به خصوصیات کلاس‌ها

در فصل‌های قبل دیدیم که کلاس‌ها شامل فیلدها و متدهایی هستند. فیلدها حالت شی را نشان می‌دهند و متدها رفتار شی را تعریف می‌کنند.

شما در این فصل □ نحوه کار متدها را دقیق‌تر کاوش خواهید کرد. تا بحال نحوه‌ی ایجاد متد را دیده‌اید. شما overload کردن متد را یاد خواهید گرفت. تکنیکی که به شما اجازه می‌دهد چندین متد هم‌نام ایجاد کنید. این عمل سرویس‌گیرنده‌ها را برای احضار متد با انواع پارامتر مختلف قادر می‌سازد.

این فصل خصوصیات را معرفی می‌کند. خصوصیات شبیه متغیرهای عضو هستند، اما خصوصیات همچون متدها پیاده‌سازی می‌شوند. این عمل پنهان کردن داده‌ها^۱ را به نحو خوب ادامه می‌دهد.

۷-۱- overload کردن متدها

اغلب می‌خواهید بیش از یک متد هم‌نام داشته باشید. معمول‌ترین مثال این مطلب داشتن چندین سازنده‌ی هم‌نام است که به شما اجازه می‌دهد اشیاء را با انواع مختلف پارامترها ایجاد کنید. برای مثال، اگر شما یک شی Time ایجاد می‌کنید. احتمال دارد بخواهید شی Time را با ارسال تاریخ □ ساعت ایجاد کنید. مواقعی دیگر □ ممکن است بخواهید یک شی Time را با ارسال یک شی موجود ایجاد کنید. overload کردن سازنده، این گزینه‌های متعدد را فراهم می‌سازد.

^۱ Information hiding

شیء `DateTime` یک شیء درونی کتابخانه `System` است □ که تعداد زیادی از عضوهای داده‌ای کلاس `Time` را دارد. مناسب است که سرویس‌گیرنده مجاز باشد یک شیء `Time` جدید را با ارسال سال □ ماه □ روز □ ساعت □ دقیقه و ثانیه به آن ایجاد کند. بعضی سرویس‌گیرنده‌ها یک یا چند سازنده را ترجیح می‌دهند.

به منظور `overload` کردن سازنده □ باید مطمئن باشید که هر سازنده نشانه‌ی منحصر به فردی دارد. نشانه‌ی یک متد از نام و لیست پارامترهایش تشکیل شده است. اگر دو متد نام یا لیست پارامتر مختلف داشته باشند. نشانه‌های مختلفی دارند. لیست پارامترها می‌توانند در تعداد یا نوع پارامترها متفاوت باشند. چهار خط زیر نشان می‌دهد چگونه متدها را بوسیله نشانه آنها متمایز کنیم.

```
void MyMethod(int p۱);
void MyMethod(int p۱, int p۲); // different number
void MyMethod(int p۱, string s۱); // different types
void SomeMethod(int p۱); // different name
```

سه متد اول همه `overload` های متد `MyMethod` () هستند. دو مورد اولی از نظر تعداد پارامترها متفاوت هستند و متد دوم و سوم در نوع پارامتر دوم با هم متفاوت هستند. این تغییرات در نشانه متدها کافی است تا کامپایلر متدها را متمایز کند.

متد چهارم با بقیه از نظر نام فرق می‌کند، این `overload` نیست، فقط یک متد متفاوتی است. یک کلاس می‌تواند هر تعداد متد با نشانه‌های مختلف داشته باشد. مثال ۱-۷ کلاس `Time` را با دو سازنده نشان می‌دهد. یکی از آن متدها یک شیء `DateTime` را می‌گیرد و دیگری ۶ عدد صحیح می‌گیرد.

مثال ۱-۷

```
using System;
namespace MethodOverloading
{
    public class Time
    {
        // private member variables
        private int Year;
        private int Month;
        private int Date;
        private int Hour;
        private int Minute;
        private int Second;
        // public accessor methods
        public void DisplayCurrentTime( )
        {
            System.Console.WriteLine( "{۰}/{۱}/{۲} {۳}:{۴}:{۵}",Month, Date, Year, Hour,
                                      Minute, Second );
        }
        // constructors
        public Time( System.DateTime dt )
        {
            Year = dt.Year;
            Month = dt.Month;
            Date = dt.Day;
            Hour = dt.Hour;
            Minute = dt.Minute;
            Second = dt.Second;
        }
        public Time( int Hour, int Minute, int Second )
        {
            this.Year = Year;
            this.Month = Month;
            this.Date = Date;
            this.Hour = Hour;
            this.Minute = Minute;
            this.Second = Second;
        }
    }
}
```

```

}
}
class Tester
{
public void Run( )
{
System.DateTime currentTime = System.DateTime.Now;
Time time\ = new Time( currentTime );
time\ .DisplayCurrentTime( );
Time time۲ = new Time( ۲۰۰۰, ۱۱, ۱۸, ۱۱, ۰۳, ۳۰ );
time۲ .DisplayCurrentTime( );
}
static void Main( )
{
Tester t = new Tester( );
t.Run( );
}
}
}

```

خروجی شبیه زیر است:

```

۷/۱۰/۲۰۰۸ ۱۶:۱۷:۳۲
۱۱/۱۸/۲۰۰۰ ۱۱:۳:۳۰

```

اگر نشانه‌ی یک تابع فقط نام آن باشد □ کامپایلر نمی‌تواند سازنده‌ها را هنگام ایجاد اشیاء جدید Time بشناسد. با این وجود □ چون نشانه‌ی متد پارامترها و نوع داده‌های آنها را شامل است، کامپایلر قادر است هر سازنده را با نشانه مورد نظر تطابق دهد.

```

System.DateTime currentTime = System.DateTime.Now;
Time time\ = new Time(currentTime);
public Time(System.DateTime dt)

```

همچنین کامپایلر قادر است برای time۲ سازنده‌ای که نشانه‌ی آن ۶ آرگومان صحیح می‌گیرد را فراخوانی کند.

```

Time time۲ = new Time(۲۰۰۰,۱۱,۱۸,۱۱,۰۳,۳۰);
public Time(int Year, int Month, int Date, int Hour, int Minute, int Second)

```

زمانی که یک متد را overload می‌کنید، باید نشانه آن را تغییر دهید. در تغییر نوع داده بازگشتی آزاد هستید، چون تغییرات آن متد را overload نمی‌کند. دو متد که نشانه یکسان دارند و نوع بازگشتی آن متفاوت باشد یک خطای کامپایلر تولید می‌کنند.

۷-۲- کپسوله کردن داده‌ها با خصوصیات

در کل، طراحی متغیرهای عضو یک کلاس بصورت private پسندیده است. بدین معنی که فقط متدهای عضو کلاس می‌توانند به مقادیر آنها دستیابی کنند. زمانی که از دستیابی مستقیم به متغیرهای عضو یک کلاس جلوگیری می‌کنید، پنهان کردن داده‌ها را با اجبار اعمال می‌کنید که بخشی از کپسوله سازی یک کلاس است.

برنامه‌نویسان شی‌گرا می‌گویند متغیرهای عضو باید private باشند. این عمل خوب است، اما چگونه دسترسی به داده‌ها را برای سرویس‌گیرنده‌ها فراهم کنیم. جواب برنامه‌نویسان C# کاربرد خصوصیات است.

خصوصیات دسترسی به حالت کلاس را بدون دسترسی مستقیم به فیلدها اجازه می‌دهند و پیاده‌سازی خصوصیات شبیه متدها انجام می‌شود. این راه‌حل ایده‌آل است. سرویس‌گیرنده دسترسی مستقیم به حالت شی را می‌خواهد. طراح کلاس می‌خواهد حالت داخلی کلاس را در فیلدهای کلاس پنهان کند و دسترسی غیر مستقیم را از طریق یک متد فراهم می‌کند.

با جدا کردن حالت کلاس از متدهایی که به حالت کلاس دسترسی دارند، طراح کلاس برای تغییر حالت داخلی کلاس در حد نیاز آزاد است. زمانی که کلاس Time برای اولین بار ایجاد شد، احتمالاً مقدار Hour به عنوان یک متغیر عضو ذخیره می‌شد.

زمانی که کلاس مجدداً طراحی می‌شود، ممکن است مقدار `Hour` محاسبه یا از پایگاه داده بازیابی شود. اگر سرویس گیرنده دسترسی مستقیم به متغیر عضو `Hour` داشته باشد، تغییر دادن نحوه حل مسئله □ سرویس گیرنده را با شکست مواجه خواهد کرد.

بطور خلاصه □ ویژگی پنهان سازی داده مورد نیاز، طراحی شی گرای خوب را فراهم می‌کند. مثال ۷-۲ یک خصوصیت بنام `Hour` ایجاد می‌کند که در پاراگراف‌های زیر بحث می‌شود.

```
using System;
namespace Properties
{
    public class Time
    {
        // private member variables
        private int year;
        private int month;
        private int date;
        private int hour;
        private int minute;
        private int second;
        // create a property
        public int Hour
        {
            get
            {
                return hour;
            }
            set
            {
                hour = value;
            }
        }
        // public accessor methods
        public void DisplayCurrentTime( )
        {
            System.Console.WriteLine("Time: {0}/{1}/{2} {3}:{4}:{5}",month, date, year, hour,
                                     minute, second );
        }
        // constructors
        public Time( System.DateTime dt )
        {
            year = dt.Year;
            month = dt.Month;
            date = dt.Day;
            hour = dt.Hour;
            minute = dt.Minute;
            second = dt.Second;
        }
    }
    class Tester
    {
        public void Run( )
        {
            System.DateTime currentTime = System.DateTime.Now;
            Time t = new Time( currentTime );
            t.DisplayCurrentTime( );
            // access the hour to a local variable
            int theHour = t.Hour;
            // display it
            System.Console.WriteLine( "Retrieved the hour: {0}",theHour );
            // increment it
            theHour++;
            // reassign the incremented value back through
```

```
// the property
t.Hour = theHour;
// display the property
System.Console.WriteLine( "Updated the hour: {0}", t.Hour);
}
[STAThread]
static void Main( )
{
    Tester t = new Tester( );
    t.Run( );
}
}
```

خروجی باید چیزی شبیه این باشد :

```
Time : ۷/۱۰/۲۰۰۸ ۱۲:۷:۴۳
Retrieved the hour: ۱۲
Updated the hour: ۱۳
```

یک خصوصیت را با نوشتن نوع و نام آن که با { } دنبال می شود، تعریف کنید. در داخل آکولادها می توانید معاون های ^۱ set و get را اعلان کنید. این معاون ها خیلی شبیه متدها هستند، اما در واقع بخشی از خصوصیت هستند. هدف این معاون ها فراهم سازی یک راه ساده برای بازیابی یا تغییر مقدار عضو خصوصی کلاس است. هیچ کدام یک از این معاون ها پارامترهای صریح ندارند اگرچه معاون set یک پارامتر ضمنی بنام value دارد که برای تنظیم مقدار متغیر عضو استفاده می شود.

طبق قرارداد □ اسامی خصوصیت ها با علائم پاسکال نوشته می شوند.

در مثال ۷-۲ اعلان خصوصیت Hour دو معاون get و set را ایجاد می کند.

```
{
get
{
return hour;
}
set
{
hour = value;
}
}
```

هر معاون یک بدنه دارد که کار بازیابی یا تنظیم مقدار خصوصیت را انجام می دهد. ممکن است مقدار خصوصیت در یک پایگاه داده ذخیره شود یا اینکه در یک متغیر عضو خصوصی ذخیره شود.

```
private int hour;
```

۷-۲-۱- معاون get

بدنه ی معاون get شبیه متدی است که یک شی از نوع خصوصیت را برمی گرداند. در مثال ۷-۲ معاون خصوصیت Hour شبیه متدی است که یک مقدار int برمی گرداند. آن مقدار متغیر عضو خصوصی hour را برمی گرداند.

```
get
{
return hour;
}
```

^۱ Accessor

هر زمان به بازایی مقدار خصوصیت نیاز است، معاون `get` احضار می‌شود. به عنوان مثال در کد زیر مقدار خصوصیت `Hour` شی `Time` به یک متغیر محلی انتساب داده می‌شود. در سرویس گیرنده، متغیر محلی `theHour` به مقدار خصوصیت `Hour` شی `t` انتساب داده می‌شود.

```
Time t = new Time(currentTime);
int theHour = t.Hour;
```

۷-۲-۲- معاون `set`

معاون `set` یک خصوصیت را مقداردهی می‌کند. زمان تعریف یک معاون `set` باید کلمه‌ی کلیدی `value` را برای نمایش آرگومان ضمنی بکار برید. که این مقدار به خصوصیت انتساب داده می‌شود.

```
set
{
    hour = value;
}
```

مجدداً در اینجا یک متغیر عضو خصوصی برای ذخیره مقدار خصوصیت استفاده می‌شود. اما معاون `set` می‌تواند آنرا به یک پایگاه داده بنویسد یا متغیرهای عضو دیگر مورد نیاز را بروز کند. زمانیکه یک مقدار به خصوصیت انتساب می‌کنید، معاون `set` بطور اتوماتیک احضار می‌شود و مقدار پارامتر ضمنی با مقدار مورد نظر شما مقداردهی می‌شود.

```
theHour++;
t.Hour = theHour;
```

خط اول مقدار متغیر محلی `theHour` را افزایش می‌دهد. مقدار جدید به خصوصیت `Hour` شی `t` انتساب داده می‌شود. در واقع مقدار `theHour` به عنوان پارامتر ضمنی `value` به معاون `set` ارسال می‌شود تا آن را به متغیر عضو محلی `hour` منتسب کند.

می‌توانید خصوصیت فقط خواندنی را با عدم سازی بخش `set` خصوصیت ایجاد کنید. بطور مشابه `□` یک خصوصیت فقط نوشتنی را با عدم پیاده‌سازی بخش `get` ایجاد کنید.

۷-۳- برگرداندن چندین مقدار

متدها فقط می‌توانند یک مقدار برگردانند، اما این همواره مناسب نیست. به کلاس `Time` برگردید. شاید ایجاد متد `GetTime()` برای برگرداندن ساعت، دقیقه و ثانیه کار بزرگی باشد. شما نمی‌توانید هر سه مقدار را برگردانید، اما می‌توانید در سه پارامتر آنها را ارسال کنید. اجازه دهید متد `GetTime()` پارامترها را تغییر دهد و نتیجه‌ی آنها را بزرسی کنید. مثال ۷-۳ اولین تلاش است.

مثال ۷-۳

```
using System;
namespace PassByRef
{
    public class Time
    {
        // private member variables
        private int Year;
        private int Month;
        private int Date;
        private int Hour;
        private int Minute;
        private int Second;
        // public accessor methods
        public void DisplayCurrentTime()
```

```

{
System.Console.WriteLine( "{0}/{1}/{2} {3}:{4}:{5}",Month, Date, Year, Hour,
                           Minute, Second );
}
public void GetTime(int theHour,int theMinute,int theSecond )
{
theHour = Hour;
theMinute = Minute;
theSecond = Second;
}
// constructor
public Time( System.DateTime dt )
{
Year = dt.Year;
Month = dt.Month;
Date = dt.Day;
Hour = dt.Hour;
Minute = dt.Minute;
Second = dt.Second;
}
}
class Tester
{
public void Run()
{
System.DateTime currentTime = System.DateTime.Now;
Time t = new Time( currentTime );
t.DisplayCurrentTime();
int theHour = 0;
int theMinute = 0;
int theSecond = 0;
t.GetTime( theHour, theMinute, theSecond );
System.Console.WriteLine( "Current time: {0}:{1}:{2}",theHour, theMinute,
                           theSecond);
}
static void Main()
{
Tester t = new Tester();
t.Run();
}
}
}

```

خروجی چیزی شبیه این است :

۷/۱/۲۰۰۸ ۱۲:۲۲:۱۹

Current time: ۰:۰:۰

توجه کنید که زمان جاری در خروجی ۰:۰:۰ است. بطور واضح □ تلاش اول کارساز نیست. مشکل در پارامترها است. شما سه پارامتر صحیح را به GetTime() ارسال کردید و پارامترهای GetTime() را تغییر دادید و آنها در برگشت بدون تغییر بودند، چون این پارامترها از نوع داده‌ی مقداری هستند.

۷-۳-۱-ارسال انواع داده‌ی مقداری بوسیله ارجاع

زمانی که یک نوع داده‌ی مقداری به متد ارسال می‌کنید □ یک کپی از آن مقدار ایجاد می‌شود. زمانی که پارامتر را تغییر می‌دهید، در اصل کپی را تغییر داده‌اید. در متد Run() متغیرهای صحیح اصلی توسط تغییرات متد GetTime() تحت تأثیر قرار نگرفته‌اند.

آنچه شما نیاز دارید، ارسال پارامترهای صحیح بوسیله ارجاع است. تا تغییرات ایجاد شده در متد به شی اصلی فراخوانی متد اعمال گردد. زمانی که یک شی را با ارجاع ارسال می‌کنید، پارامتر به همان شی اشاره می‌کند. پس زمانی که در پارامترهای GetTime تغییراتی ایجاد می‌کنید، این تغییرات به متغیرهای اصلی در Run() اعمال می‌گردد.

دو تغییر کوچک در مثال ۳-۷ لازم است. ابتدا پارامترهای متد GetTime() را طوری تغییر دهید که پارامترها را بصورت پارامترهای ref نشان دهد.

```
public void GetTime(ref int theHour, ref int theMinute, ref int theSecond )
{
    theHour = Hour;
    theMinute = Minute;
    theSecond = Second;
}
```

تغییر دوم فراخوانی متد GetTime() برای ارسال آرگومان‌ها به صورت ارجاع ها است.

```
t.GetTime(ref theHour, ref theMinute, ref theSecond);
```

اگر شما مرحله دوم را انجام ندهید □ کامپایلر هشدار می‌دهد که نمی‌تواند int را به ref int تبدیل کند.

این تغییرات در مثال ۴-۷ نشان داده می‌شود.

مثال ۴-۷

```
using System;
namespace PassByRef
{
    public class Time
    {
        // private member variables
        private int Year;
        private int Month;
        private int Date;
        private int Hour;
        private int Minute;
        private int Second;
        // public accessor methods
        public void DisplayCurrentTime()
        {
            System.Console.WriteLine( "{0}/{1}/{2} {3}:{4}:{5}", Month, Date, Year, Hour,
                                      Minute, Second );
        }
        // takes references to ints
        public void GetTime(int theHour, int theMinute, int theSecond )
        {
            theHour = Hour;
            theMinute = Minute;
            theSecond = Second;
        }
        // constructor
        public Time( System.DateTime dt )
        {
            Year = dt.Year;
            Month = dt.Month;
            Date = dt.Day;
            Hour = dt.Hour;
            Minute = dt.Minute;
            Second = dt.Second;
        }
    }
    class Tester
    {
        public void Run()
```

```

{
System.DateTime currentTime = System.DateTime.Now;
Time t = new Time( currentTime );
t.DisplayCurrentTime();
int theHour = ۰;
int theMinute = ۰;
int theSecond = ۰;
// pass the ints by reference
t.GetTime( ref theHour, ref theMinute, ref theSecond );
System.Console.WriteLine( "Current time: {۰}:{۱}:{۲}",theHour, theMinute,
                                                                    theSecond );
}
static void Main()
{
Tester t = new Tester();
t.Run();
}
}
}

```

حال خروجی شبیه زیر است :

```

۷/۱/۲۰۰۸ ۱۲:۲۵:۴۱
Current time: ۱۲:۲۵:۴۱

```

حال نتایج حاصله زمان درست را نشان می‌دهند.

با اعلان این پارامترها بصورت `ref` کامپایلر را راهنمایی کردید تا آنها را با ارجاع ارسال کند. بخاطر داشته باشید که پارامترهای `ref` ارجاعاتی به مقدار اصلی خود هستند.

۷-۳-۲- پارامترهای `out` و انتساب روشن^۱

#C انتساب روشن را تحمیل می‌کند، یعنی لازم است همه متغیرها قبل از استفاده مقداردهی شوند. در مثال ۷-۴ □ قبل از ارسال پارامترها به `GetTime()` آنها را مقداردهی اولیه کردید.

```

int theHour = ۰;
int theMinute = ۰;
int theSecond = ۰;
t.GetTime( ref theHour, ref theMinute, ref theSecond);

```

مقداردهی اولیه این متغیرها نامعقول است، چون فوراً بوسیله ارجاع به `GetTime()` ارسال می‌شوند که در آنجا تغییر خواهند یافت. اما اگر این کار را انجام ندهید، خطاهای کامپایلر زیر گزارش می‌شوند.

```

Use of unassigned local variable 'theHour'
Use of unassigned local variable 'theMinute'
Use of unassigned local variable 'theSecond'

```

#C معرف `out` را برای این چنین موقعیت‌ها فراهم کرده است. این معرف #C ضرورت مقداردهی اولیه یک پارامتر ارجاعی را حذف می‌کند. پارامترهای متد `GetMethod()` هیچ اطلاعاتی برای متد آماده نمی‌کنند. بطور ساده آنها یک مکانیزم برای گرفتن اطلاعات هستند. پس علامت‌گذاری آنها با کلمه کلیدی `out`، ضرورت مقداردهی اولیه آنها را در خارج از متد حذف می‌کند. در متد فراخوانده شده قبل از بازگشت متد، باید به پارامترهای `out` یک مقدار انتساب داده شود. اعلان `GetTime()` بصورت زیر تغییر می‌یابد.

```

public void GetTime(out int theHour,out int theMinute,out int theSecond )
{
theHour = Hour;
theMinute = Minute;

```

^۱ Definite assignment

```
theSecond = Second;  
}
```

احضار جدید متد در Main() بصورت زیر است.

```
int theHour;  
int theMinute;  
int theSecond;  
t.GetTime( out theHour, out theMinute, out theSecond);
```

کلمه‌ی کلیدی `out` همان مفهوم `ref` را می‌رساند، به استثناء اینکه `out` اجازه می‌دهد پارامترهای موجود در فراخوانی متد بدون مقداردهی اولیه استفاده شوند.

۷-۴- خلاصه

- `overload` کردن همان عمل ایجاد دو یا چند متد هم نام است که تعداد و یا نوع پارامترهای آنها متفاوت است.
- برای سرویس‌گیرنده □ خصوصیات شبیه اعضا هستند □ اما برای طراح خصوصیات شبیه متدها هستند. طراح به کمک خصوصیات می‌تواند نحوه‌ی بازیابی مقدار خصوصیت را بدون شکستن مفهوم برنامه سرویس‌گیرنده تغییر دهد.
- خصوصیات معاون‌های `get` و `set` را برای بازیابی یا تغییر یک فیلد بکار می‌برند. معاون `set` یک پارامتر ضمنی بنام `value` دارد که مقدار جدید از طریق آن به خصوصیت انتساب داده می‌شود.
- زمانی که یک پارامتر را بوسیله ارجاع به متد ارسال می‌کنید □ تغییرات داخل متد به شی اصلی در فراخوانی متد اعمال می‌گردد. انواع داده‌ی مقداری را بوسیله `ref` و `out` می‌توانید بصورت ارجاعی ارسال کنید.
- پارامتر `out` ضرورت مقداردهی اولیه یک متغیر را قبل از ارسال آن به متد حذف می‌کند.

آرایه‌ها

آنچه که در این فصل یاد خواهید گرفت:

- طرز کار با آرایه‌های دوبعدی و سه‌بعدی
- آشنایی با نوع داده‌ی Array و توابع ایستای آن
- نحوه‌ی استفاده از دستور foreach
- تفاوت مابین آرایه‌های مستطیلی و ناهموار

در بیشتر برنامه‌ها می‌خواهیم در یک لحظه با کلکسیون^۱ی از اشیاء کار کنیم. آرایه، ساده‌ترین کلکسیون در C# است و تنها نوع داده کلکسیون^۱ی است که C# بصورت درونی پشتیبانی می‌کند. کلکسیون‌های دیگر همچون پشته، صف بخشی از زبان نیستند، آنها بخشی از FCL هستند.

در این فصل کار با سه نوع آرایه یک بعدی، چند بعدی و آرایه‌های ناهموار را یاد خواهید گرفت.

۸-۱- کاربرد آرایه‌ها

آرایه یک کلکسیون اندیس‌گذاری^۱ شده از اشیاء هم نوع است. C# گرامر اصیل را برای اعلان آرایه‌ها فراهم می‌کند. برای تفهیم یک آرایه‌ی یک بعدی، یک دنباله از صندوق‌های پستی را تصور کنید (شکل ۸-۱). هر صندوق پستی دقیقاً می‌تواند یک شی را نگه دارد. هر صندوق پستی یک شماره دارد. پس شما می‌توانید تشخیص دهید یک عنصر در کدام صندوق است.

شکل ۸-۱



نکته‌ی مهم درباره آرایه‌ها این است که می‌توان با کل آرایه بصورت موجودیت واحد برخورد کرد. همانطور که خواهید دید، با استفاده از حلقه‌ها به راحتی می‌توانید یک عمل روی هر عنصر یک آرایه انجام دهید.

^۱ Indexed

۸-۱-۱- اعلان آرایه‌ها

در #C یک آرایه را به صورت زیر اعلان کنید.

```
type[] array-name;
```

به عنوان مثال :

```
int[] myIntArray;
```

در واقع شما یک آرایه اعلان نکردید. شما یک متغیر اعلان کردید که ارجاع به یک آرایه از اعداد صحیح را نگه می‌دارد. علامت [] به کامپایلر می‌گوید، شما یک آرایه اعلان می‌کنید و نوع داده، نوع عناصر آرایه را مشخص می‌کند. در مثال قبلی myIntArray آرایه‌ای از اعداد صحیح است. نمونه‌ای از یک آرایه با استفاده از کلمه کلیدی new ایجاد می‌شود. به عنوان مثال :

```
myIntArray = new int[۵];
```

این دستور یک آرایه از ۵ عدد صحیح با مقدار اولیه صفر ایجاد می‌کند.

تمایز ما بین خود آرایه و عناصر نگه داشته شده در آرایه مهم است. myIntArray آرایه است و عناصر نگه داشته شده در آن ۵ عدد صحیح هستند.

آرایه‌های #C انواع داده‌ی ارجاعی هستند که روی Heap ایجاد می‌شوند. پس آرایه‌ای که متغیر myIntArray به آن اشاره می‌کند، روی Heap تخصیص داده می‌شود. عناصر آرایه بر اساس نوع آنها تخصیص داده می‌شوند. چون اعداد صحیح از انواع داده‌ی مقداری هستند، عناصر آرایه نیز انواع مقداری خواهند بود. پس همه‌ی عناصر آرایه در داخلی بلاکی از حافظه برای آرایه ایجاد خواهند شد. بلوک حافظه تخصیص یافته به یک آرایه از انواع داده ارجاعی، شامل ارجاعاتی به عناصر واقعی هستند که در قسمت مجزایی روی Heap ایجاد می‌شوند.

۸-۱-۲- فهم مقادیر پیش فرض

زمانی که یک آرایه از انواع داده‌ی مقداری ایجاد می‌کنید، هر عنصر مقدار پیش فرضی دارد که در آرایه ذخیره می‌شود. دستور :

```
myIntArray = new int[۵];
```

یک آرایه از ۵ عدد صحیح ایجاد می‌کند. هر کدام مقدار پیش فرض صفر دارند.

بر خلاف آرایه‌ای از انواع داده مقداری، داده‌های نوع ارجاعی در یک آرایه مقدار پیش فرض ندارند. در عوض، ارجاع‌ها با null مقداردهی اولیه می‌شوند. اگر سعی کنید یک عنصر از آرایه‌ای از نوع داده‌ی ارجاعی را قبل از مقداردهی اولیه آن دستیابی کنید، یک استثناء تولید خواهد شد.

فرض کنید کلاس Button را ایجاد کرده‌ایم. یک آرایه از اشیاء Button را با دستور زیر اعلان کنید.

```
Button[] myButtonArray;
```

و آرایه واقعی را به صورت زیر تعریف کنید.

```
myButtonArray = new Button[۳];
```

می‌توانید اعلان و تعریف آرایه را به طور خلاصه شده‌ی زیر بنویسید

```
Button[] myButtonArray = new Button[۳];
```

این دستور یک آرایه‌ی سه عنصری از ارجاعات به اشیاء Button ایجاد نمی‌کند، بلکه آرایه myButtonArray را با سه ارجاع null ایجاد می‌کند. برای استفاده از این آرایه باید به ازای هر عنصر در آرایه یک شیء Button انتساب دهید.

۸-۱-۳-دسترسی به عناصر آرایه

با استفاده از عملگر اندیس می‌توانید به عناصر یک آرایه دستیابی کنید. مقدار صفر، اندیس اولین عنصر آرایه است. خصوصیت Length آرایه، تعداد اشیاء نگه داشته شده را معین می‌کند. بنابراین اشیاء از ۰ تا Length-۱ اندیس‌گذاری می‌شوند.

مثال ۸-۱ مفاهیم آرایه را نشان می‌دهد. در این مثال، کلاس Tester آرایه‌ی Employees و آرایه اعداد صحیح را ایجاد می‌کند و سپس مقادیر هر دو را چاپ می‌کند.

مثال ۸-۱

```
using System;
namespace Learning_CSharp
{
    // a simple class to store in the array
    public class Employee
    {
        public Employee(int empID)
        {
            this.empID = empID;
        }
        public override string ToString( )
        {
            return empID.ToString( );
        }
        private int empID;
    }
    public class Tester
    {
        static void Main( )
        {
            int[] intArray;
            Employee[] empArray;
            intArray = new int[۵];
            empArray = new Employee[۲];
            // populate the arrays
            for (int i = ۰; i<intArray.Length; i++)
            {
                intArray[i] = i*۲;
            }
            for (int i = ۰; i<empArray.Length; i++)
            {
                empArray[i] = new Employee(i+۱۰۰۵);
            }
            // output array values
            Console.WriteLine("intArray values:");
            for (int i = ۰; i<intArray.Length; i++)
            {
                Console.WriteLine(intArray[i].ToString( ));
            }
            Console.WriteLine("\nemployee IDs:");
            for (int i = ۰; i<empArray.Length; i++)
            {
                Console.WriteLine(empArray[i].ToString( ));
            }
        }
    }
}
```

خروجی شبیه زیر است :

```
intArray values:
```

```

۰
۲
۴
۶
۸
employee IDs:
۱۰۰۵
۱۰۰۶
۱۰۰۷

```

این مثال با تعریف کلاس Employee آغاز می‌شود، که یک سازنده با پارامتری از نوع صحیح را پیاده‌سازی می‌کند. کلاس Employee متد ToString() را برای چاپ مقدار خصوصیت ID شی Employee پیاده‌سازی می‌کند.

متد Main() یک جفت آرایه را اعلان و تعریف می‌کند. ابتدا آرایه‌ای اعداد صحیح با مقادیر صفر پر می‌شود. سپس صریحاً با یک حلقه for مقداردهی می‌شود. شمارنده‌ی حلقه همه عناصر آرایه را با افزایش مقدار اندیس i طی می‌کند.^۱ مقدار intArray را با دستور زیر تنظیم کنید.

```
intArray[i] = i*۲;
```

مقدار i در اولین مرحله صفر است و [۰intArray] با ۲*۰ مقداردهی می‌شود و در مرحله‌ی بعدی مقدار i یک است و عنصر دوم به مقدار ۲*۱ مقداردهی می‌شود و ... این حلقه تا زمانی که i مساوی intArray.Length شود، اجرا می‌گردد. توجه کنید که مقدار i برای آخرین عنصر آرایه -intArray.Length است.

باید محتوای آرایه Employee به وسیله دست ساخته شود، چون مقادیر آن از نوع داده ارجاعی هستند، هنگام ایجاد آرایه مقداردهی اولیه نمی‌شوند و با null پر می‌شوند. در حال حاضر باید برای هر اندیس آرایه، شی Employee جدید ایجاد شود. در نهایت، محتوای آرایه‌ها نشان داده می‌شوند، تا مطمئن شوید همانطور که قصد داشتید پر شده‌اند.

۲-۸- دستور foreach

دستور foreach اجازه می‌دهد تمامی عناصر یک آرایه یا کلکسیون دیگر را طی کنید. گرامر دستور foreach بصورت زیر است.

```
foreach (type identifier in expression) statement
```

مثال ۸-۱ را برای جایگزینی دو دستور for نهایی با دستور foreach مرور کنید. همانطور که در مثال ۸-۲ نشان داده شده است، می‌توانید جایگزینی‌ها را انجام دهید.

مثال ۸-۲

```

foreach ( int i in intArray )
{
    Console.WriteLine( i.ToString( ) );
}
foreach ( Employee e in empArray )
{
    Console.WriteLine( e.ToString( ) );
}

```

خروجی یکسان خواهد بود.

^۱ Iterate

۸-۳- مقداردهی اولیه عناصر آرایه

می‌توانید مقداردهی اولیه محتوای یک آرایه را در زمان معرفی آن با یک لیست از مقادیر در داخل {} انجام دهید. #C یک گرامر کوتاه و بلند فراهم می‌کند.

```
int[] myIntArray = new int[۵] { ۲, ۴, ۶, ۸, ۱۰ }; // بلند
```

```
int[] myIntArray = { ۲, ۴, ۶, ۸, ۱۰ }; // گرامر کوتاه
```

در گرامر کوتاه، #C بطور اتوماتیک یک آرایه‌ی مناسب با اندازه تعداد عناصر {} ایجاد می‌کند. از نظر عملیاتی هر دو دستور یکسان هستند. بیشتر برنامه‌نویسان گرامر کوتاه را بکار می‌برند.

۸-۴- کلید کلیدی params

کلمه کلیدی params اجازه می‌دهد یک تعداد پویا از پارامترهای هم نوع را به یک متد ارسال کنید. آنچه که متد دریافت می‌کند، آرایه‌ای از آن نوع داده است.

در مثال بعدی، متد DisplayVals() را ایجاد می‌کنید که یک تعداد متغیر از آرگومان‌های صحیح را می‌گیرد.

```
public void DisplayVals(params int[] intVals)
```

شما برای طی کردن این آرایه همانند هر آرایه از اعداد صحیح آزاد هستید.

```
foreach (int i in intVals)
{
    Console.WriteLine("DisplayVals {۰}", i);
}
```

در فراخوانی متد به ایجاد صریح آرایه نیاز ندارد. خود کامپایلر پارامترها را در یک آرایه برای متد DisplayVals فراهم خواهد آورد.

```
t.DisplayVals(۵, ۶, ۷, ۸);
```

اگر خودتان ترجیح دهید، می‌توانید داده‌ها را از طریق یک آرایه ارسال کنید.

```
int [] explicitArray = new int[۵] { ۱, ۲, ۳, ۴, ۵ };
t.DisplayVals(explicitArray);
```

می‌توانید فقط یک آرگومان params در متد بکار ببرید و این آرگومان باید آخرین آرگومان متد باشد.

مثال ۸-۳ کاربرد کلمه‌ی کلیدی params را ارائه می‌کند.

مثال ۸-۳

```
using System;
namespace UsingParams
{
    public class Tester
    {
        static void Main( )
        {
            Tester t = new Tester( );
            t.DisplayVals(۵, ۶, ۷, ۸);
            int [] explicitArray = new int[] { ۱, ۲, ۳, ۴, ۵ };
            t.DisplayVals(explicitArray);
        }
        public void DisplayVals(params int[] intVals)
        {
```



```
foreach (int i in intVals)
{
    Console.WriteLine("DisplayVals {0}",i);
}
}
}
```

خروجی شبیه این است.

```
displayvals ۵
displayvals ۶
displayvals ۷
displayvals ۸
displayvals ۱
displayvals ۲
displayvals ۳
displayvals ۴
displayvals ۵
```

۸-۵- آرایه‌های چندبعدی

آرایه‌ها می‌توانند به صورت ردیف‌های بلندی از شکاف‌ها باشند که می‌توانند مقادیر را جا دهند. تصور کنید که ۱۰ ردیف از شکاف‌ها دارید که یکی بالای دیگری است. این یک آرایه‌ی دو بعدی کلاسیک از سطرها و ستون‌ها است. ردیف‌ها سرتاسر آرایه را می‌پیمایند و ستون‌ها آرایه را به بالا و پایین می‌پیمایند. همانطور که شکل ۸-۲ نشان می‌دهد.

شکل ۸-۲

	Col 0	Col 5	Col 10
Row 1			
Row 2			
Row 3			
Row 4			
Row 5			

تصور بعد سوم سخت‌تر است. بسیار خوب، حال ۴ بعدی را تصور کنید. آرایه‌های چندبعدی مفید هستند، حتی اگر شما نتوانید دقیقاً تصور کنید که آنها شبیه چه چیزی هستند؟

۸-۵-۱- آرایه‌های مستطیلی

آرایه مستطیلی، یک آرایه دوبعدی است. در آرایه دو بعدی کلاسیک، تعداد سطرها با بعد اول و تعداد ستون‌ها با بعد دوم مشخص می‌شود. برای اعلان یک آرایه دوبعدی، گرامر زیر را به کار برید.

```
type [,] array-name
```

به عنوان مثال: برای اعلان و معرفی یک آرایه مستطیلی دوبعدی به نام `myRectangularArray` که شامل دو سطر و سه ستون از اعداد صحیح است، دستور زیر را بنویسید:

```
int [,] myRectangularArray = new int[۲,۳];
```

مثال ۸-۴ آرایه دوبعدی را اعلان، تعریف و مقداردهی اولیه می‌کند. سپس محتوای آن را چاپ می‌کند. در این مثال یک حلقه‌ی `for` برای مقداردهی اولیه عناصر آرایه استفاده می‌شود.

مثال ۸-۴

```
using System;
namespace RectangularArray
{
public class Tester
{
static void Main( )
{
const int rows = ۴;
const int columns = ۳;
// declare a {x} integer array
int[,] rectangularArray = new int[rows, columns];
// populate the array
for ( int i = ۰; i < rows; i++ )
{
for ( int j = ۰; j < columns; j++ )
{
rectangularArray[i, j] = i + j;
}
}
// report the contents of the array
for ( int i = ۰; i < rows; i++ )
{
for ( int j = ۰; j < columns; j++ )
{
Console.WriteLine( "rectangularArray[{۰},{۱}] = {۲}",i, j,
rectangularArray[i, j] );
}
}
}
}
}
```

خروجی شبیه زیر است:

```
rectangularArray[۰,۰] = ۰
rectangularArray[۰,۱] = ۱
rectangularArray[۰,۲] = ۲
rectangularArray[۱,۰] = ۱
rectangularArray[۱,۱] = ۲
rectangularArray[۱,۲] = ۳
rectangularArray[۲,۰] = ۲
rectangularArray[۲,۱] = ۳
rectangularArray[۲,۲] = ۴
rectangularArray[۳,۰] = ۳
rectangularArray[۳,۱] = ۴
rectangularArray[۳,۲] = ۵
```

گروه‌ها در اعلان `int [,]` مشخص می‌کنند که نوع داده‌ی مورد نظر یک آرایه از اعداد صحیح است و کاما برای مشخص کردن آرایه دوبعدی است. تعریف `rectangularArray` با `new int [row,cols]`، اندازه هر بعد را مقداردهی می‌کند. در اینجا دستورات اعلان و تعریف آرایه ترکیب شده‌اند.

این برنامه آرایه‌ی مستطیلی را با یک جفت حلقه تودرتو پر می‌کند (تکرار هر ستون در هر سطر). در سطر اول `rectangularArray[۰,۰]`، اولین عنصر و `rectangularArray[۰,۱]`، اولین عنصر بعدی است. در سطر دوم `rectangularArray[۱,۰]`، اولین عنصر بوده و عنصر بعدی `rectangularArray[۱,۱]` می‌باشد.

همانطور که می‌توانید به آرایه یک بعدی با لیستی از مقادیر در داخل آکولادها مقدار اولیه دهید. می‌توان یک آرایه‌ی دو بعدی را نیز با گرامری مشابه مقداردهی اولیه کرد. مثال ۸-۵ این عمل را نشان می‌دهد.

مثال ۸-۵

```
using System;
namespace InitializingMultiDimensionalArray
{
public class Tester
{
static void Main( )
{
const int rows = ۴;
const int columns = ۳;
// imply a ۴x۳ array
int[,] rectangularArray =
{
{۰,۱,۲}, {۳,۴,۵}, {۶,۷,۸}, {۹,۱۰,۱۱}
};
for ( int i = ۰; i < rows; i++ )
{
for ( int j = ۰; j < columns; j++ )
{
Console.WriteLine( "rectangularArray[{۰},{۱}] = {۲}", i, j,
rectangularArray[i, j] );
}
}
}
}
}
```

خروجی شبیه این است :

```
rectangularArray[۰,۰] = ۰
rectangularArray[۰,۱] = ۱
rectangularArray[۰,۲] = ۲
rectangularArray[۱,۰] = ۳
rectangularArray[۱,۱] = ۴
rectangularArray[۱,۲] = ۵
rectangularArray[۲,۰] = ۶
rectangularArray[۲,۱] = ۷
rectangularArray[۲,۲] = ۸
rectangularArray[۳,۰] = ۹
rectangularArray[۳,۱] = ۱۰
rectangularArray[۳,۲] = ۱۱
```

مثال قبلی خیلی شبیه مثال ۸-۴ است اما در این مثال ابعاد دقیق آرایه را با مقداردهی اولیه آن مشخص می‌کنید.

```
int[,] rectangularArrayrectangularArray =
{
{۰,۱,۲}, {۳,۴,۵}, {۶,۷,۸}, {۹,۱۰,۱۱}
};
```

انتساب مقادیر در ۴ لیست در داخل {} که هر کدام شامل سه عنصر هستند، یک آرایه‌ی ۴ سطری و سه ستونی را مشخص می‌کند. می‌توانید ببینید کامپایلر #C مفهوم روش گروه بندی مقادیر ورودی را می‌فهمد. چون آن قادر است به اشیاء با آفست‌های مناسب دستیابی کند.

آرایه‌های #C هوشمند هستند و محدوده آنها را نگه می‌دارند. زمانی که یک آرایهٔ ۴*۳ تعریف می‌کنید، باید با آن همانطور رفتار کنید و نباید آن را به صورت یک آرایه‌ی ۴*۳ در نظر بگیرید.

۸-۵-۲-آرایه‌های ناهموار^۱

آرایه‌های ناهموار، آرایه‌ای از آرایه هستند. چون نیاز نیست، همه‌ی سطرهاى آن هم اندازه باشند. آن را ناهموار می‌گویند، چون نمایش گرافیکی آن چهارگوش نیست.

^۱ Jagged Arrays

زمانی که یک آرایه ناهموار ایجاد می‌کنید، تعدادی سطر در آرایه‌ی خود اعلان کنید. هر سطر یک آرایه نگه می‌دارد که هر طولی می‌توانند داشته باشند. این آرایه‌ها باید اعلان شده باشند. پس می‌توانید مقادیر عناصر را در این آرایه‌های درونی بریزید.

در یک آرایه ناهموار، هر بعد، یک آرایه یک بعدی است. برای اعلان یک آرایه‌ی ناهموار، گرامر زیر را بکار ببرید، که تعداد [ها تعداد ابعاد آرایه را نشان می‌دهد.

```
type [] []...
```

برای مثال، یک آرایه دوبعدی ناهموار از اعداد صحیح به نام `myJaggedArray` به صورت زیر اعلان کنید.

```
int [] [] myJaggedArray;
```

با نوشتن `[۴]myJaggedArray[۲]` به پنجمین عنصر آرایه‌ی سوم دستیابی کنید.

مثال ۸-۶ یک آرایه‌ی ناهموار به نام `myJaggedArray` ایجاد می‌کند و عناصر آن را مقداردهی اولیه کرده و سپس محتوای آنها را چاپ می‌کند. برای ذخیره فضای برنامه از مقداردهی اولیه اتوماتیک اعداد صحیح به مقدار صفر بهره می‌برد و فقط بعضی از عناصر را مقداردهی می‌کند.

مثال ۸-۶

```
using System;
namespace JaggedArray
{
public class Tester
{
static void Main( )
{
const int rows = ۴;
// declare the jagged array as ۴ rows high
int[][] jaggedArray = new int[rows][];
// the first row has ۰ elements
jaggedArray[۰] = new int[۰];
// a row with ۲ elements
jaggedArray[۱] = new int[۲];
// a row with ۳ elements
jaggedArray[۲] = new int[۳];
// the last row has ۰ elements
jaggedArray[۳] = new int[۰];
// Fill some (but not all) elements of the rows
jaggedArray[۰][۲] = ۱۰;
jaggedArray[۱][۱] = ۱۲;
jaggedArray[۲][۱] = ۹;
jaggedArray[۲][۲] = ۹۹;
jaggedArray[۳][۰] = ۱۰;
jaggedArray[۳][۱] = ۱۱;
jaggedArray[۳][۲] = ۱۲;
jaggedArray[۳][۳] = ۱۳;
jaggedArray[۳][۴] = ۱۴;
for ( int i = ۰; i < ۵; i++ )
{
Console.WriteLine( "jaggedArray[۰][{۰}] = {۱}",i, jaggedArray[۰][i] );
}
for ( int i = ۰; i < ۲; i++ )
{
Console.WriteLine( "jaggedArray[۱][{۰}] = {۱}",i, jaggedArray[۱][i] );
}
for ( int i = ۰; i < ۳; i++ )
{
Console.WriteLine( "jaggedArray[۲][{۰}] = {۱}",i, jaggedArray[۲][i] );
}
}
```

```
for ( int i = ۰; i < ۵; i++ )
{
    Console.WriteLine( "jaggedArray[۳][{۰}] = {۱}",i, jaggedArray[۳][i] );
}
}
}
```

خروجی شبیه زیر است :

```
jaggedArray[۰][۰] = ۰
jaggedArray[۰][۱] = ۰
jaggedArray[۰][۲] = ۰
jaggedArray[۰][۳] = ۱۵
jaggedArray[۰][۴] = ۰
jaggedArray[۱][۰] = ۰
jaggedArray[۱][۱] = ۱۲
jaggedArray[۲][۰] = ۰
jaggedArray[۲][۱] = ۹
jaggedArray[۲][۲] = ۹۹
jaggedArray[۳][۰] = ۱۰
jaggedArray[۳][۱] = ۱۱
jaggedArray[۳][۲] = ۱۲
jaggedArray[۳][۳] = ۱۳
jaggedArray[۳][۴] = ۱۴
```

در این مثال یک آرایه‌ی ناهموار با ۴ ردیف ایجاد می‌شود.

```
int[][] jaggedArray = new int[rows][];
```

توجه کنید که بعد دوم مشخص نشده است. این بعد با ایجاد یک آرایه برای هر سطر تنظیم می‌شود. هر کدام از این آرایه‌ها اندازه‌ی مختلف دارند.

```
// the first row has ۵ elements
jaggedArray[۰] = new int[۵];
// a row with ۲ elements
jaggedArray[۱] = new int[۲];
// a row with ۳ elements
jaggedArray[۲] = new int[۳];
// the last row has ۵ elements
jaggedArray[۳] = new int[۵];
```

زمانی که برای هر سطر یک آرایه معین شد. شما فقط نیاز دارید اعضای مختلف آنها را پر کنید و محتویات آن را برای اطمینان بیشتر چاپ کنید. روش دیگر چاپ مقادیر به خروجی، کاربرد دو حلقه‌ی for تودرتو است که از خصوصیت Length برای کنترل طول حلقه استفاده می‌کنند.

```
for (int i = ۰; i < jaggedArray.Length; i++ )
{
    for (int j = ۰; j < jaggedArray[i].Length; j++)
    {
        Console.WriteLine("jaggedArray[{۰}][{۱}] = {۲}",i, j, jaggedArray[i][j]);
    }
}
```

در این مورد حلقه‌ی بیرونی سطرها را طی می‌کند و حلقه‌ی درونی هر ستون از سطر معین شده را در خروجی چاپ می‌کند. چون از Length برای کنترل طول حلقه استفاده می‌شود. مهم نیست که هر سطر طول مختلفی داشته باشد.

نحوه‌ی نمایش آرایه‌های دوبعدی مستطیلی بصورت `rectangularArray[i,j]` و نمایش آرایه دوبعدی ناهموار به صورت `jaggedArray[i][j]` انجام می‌شود.

۸-۶- متدهای آرایه

اگرچه آرایه را به عنوان یک نوع داده‌ی درونی به کار بردید. در واقع آرایه یک شی از کلاس `System.Array` است. آرایه‌ها در #C چندین متد و خصوصیت را فراهم کرده‌اند، که در جدول ۸-۱ بعضی از متدها و خصوصیات مهم آمده است.

جدول ۸-۱

متد یا خصوصیت	هدف
<code>()BinarySearch</code>	متد ایستای عمومی <code>overload</code> شده که یک آرایه یک بعدی مرتب شده را جستجو می‌کند.
<code>()Clean</code>	یک محدوده از عناصر آرایه را به صفر یا <code>null</code> مقداردهی می‌کند
<code>()Copy</code>	بخشی از یک آرایه را به آرایه دیگر کپی می‌کند.
<code>()CreateInstance</code>	یک نمونه جدید از آرایه را معرفی می‌کند.
<code>()IndexOf</code>	اندیس اولین نمونه از یک مقدار در یک آرایه‌ی یک بعدی را بر می‌گرداند.
<code>()LastIndexOf</code>	اندیس آخرین نمونه از یک مقدار در یک آرایه‌ی یک بعدی را بر می‌گرداند
<code>()Reverse</code>	ترتیب عناصر در یک آرایه‌ی یک بعدی را معکوس می‌کند.
<code>()Sort</code>	مقادیر یک آرایه‌ی یک بعدی را مرتب می‌کند
<code>Length</code>	این خصوصیت طول آرایه را بر می‌گرداند
<code>()GetEnumerator</code>	متد عمومی که یک <code>IEnumerator</code> بر می‌گرداند

۸-۷- مرتب کردن آرایه‌ها

دو متد ایستای مفید در جدول ۸-۱ شایسته‌ی بررسی هستند: `()Sort` و `()Reverse`. این متدها همان کاری که شما فکر می‌کنید انجام می‌دهند. متد `()Reverse` ترتیب عناصر در آرایه را معکوس می‌کند و متد `()Sort` عناصر را با نظم مرتب می‌کند. این متدها برای آرایه‌هایی از انواع داده‌ی درونی #C همچون `string` و `int` و ... پشتیبانی می‌شوند. ولی برای کاربرد متد `()Sort` در همه کلاس‌هایی که خود تعریف کرده‌اید، پیاده‌سازی واسطه `Comparable` ضروری است. مثال ۸-۷ کاربرد این دو متد را برای دستکاری اشیاء `string` نشان می‌دهد.

مثال ۸-۷

```
using System
namespace ArraySortAndReverse
{
    public class Tester
    {
        public static void PrintMyArray( string[] theArray )
        {
            foreach ( string str in theArray )
            {
                Console.WriteLine( "Value: {0}", str );
            }
            Console.WriteLine( "\n" );
        }
        static void Main( )
```

```
{
String[] myArray =
    {
        "Proust", "Faulkner", "Mann", "Hugo"
    };

PrintMyArray( myArray );
Array.Reverse( myArray );
PrintMyArray( myArray );
String[] myOtherArray =
    {
        "We", "Hold", "These", "Truths",
        "To", "Be", "Self", "Evident",
    };
PrintMyArray( myOtherArray );
Array.Sort( myOtherArray );
PrintMyArray( myOtherArray );
}
}
}
```

خروجی آن شبیه زیر است :

Value: Proust	Value: Faulkner
Value: Mann	Value: Hugo
Value: Hugo	Value: Mann
Value: Faulkner	Value: Proust
Value: We	Value: Hold
Value: These	Value: Truths
Value: To	Value: Be
Value: Self	Value: Evident
Value: Be	Value: Evident
Value: Hold	Value: Self
Value: These	Value: To
Value: Truths	Value: We

(خروجی بصورت ستونی زیر هم چاپ می‌شود. در هر سطر دو مقدار نشان داده شده است)

این مثال با ایجاد یک آرایه از رشته‌ها بوسیله‌ی کلمات زیر آغاز می‌گردد.

```
"Proust", "Faulkner", "Mann", "Hugo"
```

این آرایه چاپ می‌شود و سپس به متد `Array.Reverse()` ارسال می‌شود. مجدداً آرایه چاپ می‌شود. بطور مشابه، این مثال دومین آرایه را با نام `myOtherArray` و با کلیدهای زیر ایجاد می‌کند.

```
"We", "Hold", "These", "Truths",
"To", "Be", "Self", "Evident",
```

این آرایه به متد `Array.Sort()` ارسال می‌شود، که آنها را بر اساس حروف الفبا مرتب می‌کند.

۸-۸- خلاصه

- آرایه یک کلکسیون اندیس‌گذاری شده از اشیاء هم نوع است.
- یک آرایه را با نوع داده عناصر آن که با [] دنبال می‌شود و سپس نام آرایه قرار می‌گیرد اعلان کنید. سپس آرایه را با کلمه‌ی کلیدی `new` و تعداد عناصر آرایه تعریف کنید.
- اندیس اولین عنصر آرایه همواره صفر و اندیس آخرین عنصر `-Length` است.
- یک حلقه `for` را برای طی کردن سرتاسر آرایه (بوسیله شماره‌دهنده حلقه به عنوان اندیس آرایه) بکار برید.

- دستور `foreach` طی کردن سرتاسر عناصر آرایه را بدون نیاز به یک شمارنده مجاز می‌دارد.
- زمانی که آرایه‌ای ایجاد می‌شود، می‌توان مقادیر اولیه آن را در داخل آکولادها `{ }` مشخص کرد.
- کلمه کلیدی `params` امکان ارسال تعداد دلخواه پارامترهای هم نوع به یک متد را فراهم می‌سازد. متد با این پارامترها همانند یک آرایه برخورد خواهد کرد.
- آرایه‌ها می‌توانند بیشتر از یک بعد داشته باشند. یک آرایه‌ی دو بعدی دو اندیس دارد که سطرها و ستون‌ها را مشخص می‌کنند.
- یک آرایه‌ی مستطیلی آرایه دو بعدی است که همه سطرها تعداد ستون‌های یکسانی دارند.
- آرایه ناهموار آرایه‌ای است، که لازم نیست همه‌ی سطرها تعداد ستون‌های یکسانی داشته باشند.
- خصوصیت `Length` یک آرایه، تعداد عناصر آرایه را بر می‌گرداند.
- کلاس آرایه یک مجموعه متد برای مرتب کردن، جستجو کردن و دستکاری عناصر دارد.

فصل نهم

ساختارها

آنچه که در این فصل یاد خواهید گرفت:

- با ساختار و ویژگی‌های آن
- نحوه‌ی ایجاد ساختارها و آشنایی با خصوصیات نوع مقداری بودن آن

ساختار^۱ یک نوع داده تعریف شده کاربری است. ساختارها شبیه کلاس‌ها هستند. آنها ممکن است سازنده‌ها، خصوصیات، متدها، فیلدها، عملگرها، انواع داده‌ی تودرتو و اندیس‌گذارها را شامل شوند.

تفاوت‌های مهمی مابین کلاس و ساختار وجود دارد. برای مثال، ساختارها وراثت یا مخرب‌ها را پشتیبانی نمی‌کنند. مهم‌تر اینکه، اگرچه کلاس یک نوع داده‌ی ارجاعی است، ولی ساختار یک نوع داده‌ی مقداری است. بنابراین، ساختارها برای نمایش اشیائی که مفهوم ارجاع را نیاز ندارد مفید هستند.

دید اجماع این است که شما باید ساختارها را فقط برای انواع داده‌ی که کوچک، ساده و مشابه انواع داخلی C# هستند بکار برید.

برنامه‌نویسان ++C توجه کنند: معنی ساختار C# خیلی متفاوت از ++C است. در ++C یک ساختار دقیقاً شبیه یک کلاس است، با استثناء میدان دید که به طور پیش فرض متفاوت هستند. در کلاس پیش فرض private و در ساختار public است. در C#، ساختارها انواع داده‌ی مقداری هستند. ساختارهای C# محدودیت‌هایی دارند که در این فصل بحث خواهند شد.

ساختارها در آرایه‌ها مؤثرتر عمل می‌کنند. با این وجود، در هنگام استفاده با کلکسیون‌ها کارایی کمتری دارند. کلکسیون‌ها اشیاء را به صورت ارجاعی انتظار می‌روند و ساختارها باید جعبه‌بندی شوند. جعبه‌بندی و از جعبه در آوردن عملیات سربرار هستند. پس کلاس‌ها در کلکسیون‌ها بهتر کار می‌کنند.

^۱ Struct

در این فصل نحوه‌ی تعریف، کار با ساختارها و نحوه‌ی مقداردهی اولیه مقادیر آنها با استفاده از سازنده‌ها را یاد خواهید گرفت.

۹-۱- تعریف ساختارها

نحوه‌ی اعلان یک ساختار تقریباً شبیه کلاس است.

```
[attributes] [access-modifiers] struct identifier [:interface-list] { struct-  
members }
```

مثال ۹-۱ تعریف یک ساختار را نشان می‌دهد. Location یک نقطه روی یک سطح دوبعدی را نشان می‌دهد. توجه کنید که ساختار Location دقیقاً شبیه یک کلاس اعلان می‌شود، به استثناء اینکه به جای کلمه‌ی `class` از `struct` استفاده شده است. همچنین توجه کنید که سازنده Location دو عدد صحیح را گرفته و مقدار آنها را به اعضای نمونه `xVal` و `yVal` انتساب می‌دهد. مختصات `x, y` در Location به صورت خصوصیات اعلان می‌شوند.

مثال ۹-۱

```
#region Using directives
using System;
using System.Collections.Generic;
using System.Text;
#endregion
namespace CreatingAStruct
{
    public struct Location
    {
        private int xVal;
        private int yVal;
        public Location( int xCoordinate, int yCoordinate )
        {
            xVal = xCoordinate;
            yVal = yCoordinate;
        }
        public int x
        {
            get
            {
                return xVal;
            }
            set
            {
                xVal = value;
            }
        }
        public int y
        {
            get
            {
                return yVal;
            }
            set
            {
                yVal = value;
            }
        }
        public override string ToString( )
        {
            return ( String.Format( "{0}, {1}", xVal, yVal ) );
        }
    }
    public class Tester
```

```

{
public void myFunc( Location loc )
{
loc.x = ۵۰;
loc.y = ۱۰۰;
Console.WriteLine( "In MyFunc loc: {۰}", loc );
}
static void Main( )
{
Location loc\ = new Location( ۲۰۰, ۳۰۰ );
Console.WriteLine( "Loc\ location: {۰}", loc\ );
Tester t = new Tester( );
t.myFunc( loc\ );
Console.WriteLine( "Loc\ location: {۰}", loc\ );
}
}
}
Output:
Loc\ location: ۲۰۰, ۳۰۰
In MyFunc loc: ۵۰, ۱۰۰
Loc\ location: ۲۰۰, ۳۰۰

```

برخلاف کلاس‌ها، ساختارها از ارث‌بری پشتیبانی نمی‌کنند. آنها قطعاً از Object مشتق می‌شوند، اما نمی‌توانند از کلاس یا ساختار دیگری ارث‌بری کنند. ساختارها قطعاً مهر شده هستند. شبیه کلاس‌ها، ساختارها می‌توانند چند واسط را پیاده‌سازی کنند. تفاوت‌های دیگر کلاس و ساختار به صورت زیر هستند.

- ساختار سازنده پیش‌فرض یا مخرب ندارد.

ساختارها مخرب ندارند. نمی‌توانند یک سازنده بدون پارامتر سفارشی داشته باشند. اگر ساختار سازنده نداشته باشد، CLR شی ساختار را مقداردهی اولیه خواهد کرد. اگر شما یک سازنده غیر پیش‌فرض فراهم کنید، مقداردهی اولیه توسط CLR اتفاق نخواهد افتاد و همه فیلدها به طور صریح مقداردهی اولیه می‌شوند.

- بدون مقداردهی اولیه

نمی‌توانید یک فیلد از یک ساختار را مقداردهی اولیه کنید. پس دستور زیر غیر مجاز هستند.

```

private int xVal = ۵۰;
private int yVal = ۱۰۰;

```

ساختارها طوری طراحی می‌شوند که ساده و سبک باشند. در حالیکه داده‌های عضو private پنهان کردن داده و کپسوله کردن را ترقی می‌دهند. بعضی برنامه‌نویسان احساس می‌کنند این ایده برای ساختارها ضایع کننده است. آنها اعضای داده را public کرده و پیاده‌سازی ساختار را ساده می‌کنند. برنامه‌نویسانی دیگر احساس می‌کنند که خصوصیات، واسط ساده و واضح فراهم می‌کنند و برنامه‌نویسی خوب، پنهان کردن داده را حتی با اشیاء سبک وزن پیشنهاد می‌کند. با توانایی‌های جدید VS ۲۰۰۵، می‌توان به کمک خصوصیات public، متغیرهای public را با متغیرهای private جایگزین کرد. کافی است روی متغیر کلیک راست کرده و Refactor Encapsulate Field را اجرا کنید. VS متغیر public شما را به private تبدیل کرده و معاون‌های get و set را ایجاد می‌کند.

۹-۲-۱ ایجاد ساختارها

یک نمونه از یک ساختار را با استفاده کلمه‌ی کلیدی new در یک دستور انتساب همانند یک نمونه از کلاس ایجاد کنید. در مثال ۹-۱، کلاس Tester یک نمونه از Location را به صورت زیر ایجاد می‌کند:

```
Location loc\ = new Location(۲۰۰,۳۰۰);
```

در اینجا نمونه جدید locl نامگذاری شده و دو مقدار ۲۰۰ و ۳۰۰ به آن رد می‌شوند.

۹-۲-۱- ساختارها به صورت انواع داده مقداری

تعریف کلاس Tester مثال ۸-۱ یک شی Location را شامل است. ساختار locl با مقادیر ۲۰۰ و ۳۰۰ را ایجاد کرده است. این خط از کد سازنده Location را فراخوانی می‌کند.

```
Location loc\ = new Location(۲۰۰, ۳۰۰);
```

سپس WriteLine() فراخوانی می‌شود.

```
Console.WriteLine("Loc\ location: {۰}", loc\);
```

متد WriteLine() یک شی را انتظار می‌رود. اما متأسفانه Location یک ساختار (نوع داده مقداری) است. کامپایلر به طور اتوماتیک ساختار را جعبه‌بندی می‌کند و آن شی جعبه‌بندی شده به WriteLine() رد می‌شود. متد ToString() روی شی جعبه‌بندی شده فراخوانی می‌شود و چون ساختار از Object ارث‌بری می‌کند. آن قادر است به چندریختی عکس‌العمل نشان دهد و ممکن است هر متد را override کند.

```
Loc\ location: ۲۰۰, ۳۰۰
```

می‌توانید این جعبه‌بندی را با تغییر تکه کد قبلی بصورت زیر اجتناب کنید: در صورتی که ToString را override کرده باشید.

```
Console.WriteLine("Loc\ location: {۰}", loc\.ToString());
```

ساختارها اشیاء مقداری هستند و زمان رد کردن به یک تابع، آنها بوسیله مقادیر رد می‌شوند. همانطور که در خط کد بعدی می‌بینید شی locl به متد myFunc() رد می‌شود.

```
t.myFunc(loc\);
```

در myFunc() مقادیر جدید به x, y انتساب داده می‌شوند و این مقادیر جدید در خروجی چاپ می‌شوند:

```
Loc\ location: ۵۰, ۱۰۰
```

زمانی که به تابع فراخوانی کننده (Main) برگردید و WriteLine() را فراخوانی کنید، مقادیر بدون تغییر هستند.

```
Locl Location: ۲۰۰, ۳۰۰
```

ساختار به صورت یک شی مقداری رد شده است و یک کپی از آن در myFunc() ایجاد می‌شود. اعلان را به class تغییر دهید:

```
public class Location
```

و آزمایش را مجدداً اجرا کنید. خروجی این است:

```
Loc\ location: ۲۰۰, ۳۰۰
```

```
In MyFunc loc: ۵۰, ۱۰۰
```

```
Loc\ location: ۵۰, ۱۰۰
```

حالا شی Location مفهوم ارجاع‌ها را دارد. پس زمانی که مقادیر آن در myFunc() تغییر یابد، هنگام برگشت به Main()، شی اصلی نیز تغییر داده می‌شود.

۹-۲-۲- ایجاد ساختارها بدون new

چون locl یک ساختار (نه یک کلاس) است، آن روی Stack ایجاد می‌شود. پس در مثال ۹-۱ زمانی که عملگر new فراخوانی می‌شود.

```
Location loc\ = new Location(۲۰۰،۳۰۰);
```

شی Location منتج شده، در روی پشته ایجاد می‌شود. عملگرد new سازنده Location را فراخوانی می‌کند. با این وجود، بر خلاف یک کلاس، ایجاد یک ساختار بدون استفاده از new امکان‌پذیر است. این عمل با متغیرهای درونی C# سازگار است. مثال ۸-۲ را مشاهده کنید.

مثال ۸-۲

```
#region Using directives
using System;
using System.Collections.Generic;
using System.Text;
#endregion
namespace StructWithoutNew
{
    public struct Location
    {
        public int xVal;
        public int yVal;
        public Location( int xCoordinate, int yCoordinate )
        {
            xVal = xCoordinate;
            yVal = yCoordinate;
        }
        public int x
        {
            get
            {
                return xVal;
            }
            set
            {
                xVal = value;
            }
        }
        public int y
        {
            get
            {
                return yVal;
            }
            set
            {
                yVal = value;
            }
        }
        public override string ToString( )
        {
            return ( String.Format( "{۰}, {۱}", xVal, yVal ) );
        }
    }
    public class Tester
    {
        static void Main( )
        {
            Location loc\; // no call to the constructor
            loc\.xVal = ۷۰; // initialize the members
            loc\.yVal = ۲۲۰;
            Console.WriteLine( loc\ );
        }
    }
}
```

در مثال ۸-۲ قبل از فراخوانی یک متد از loc1 و رد کردن این شی به WriteLine()، متغیرهای محلی را به طور مستقیم مقداردهی اولیه کردید.

```
loc1.xVal = ۷۵;
loc1.yVal = ۲۲۵;
```

اگر در برنامه‌ی خود یکی از دستورات را به توضیح تبدیل کرده و مجدداً کامپایل کنید.

```
static void Main()
{
    Location loc1;
    loc1.xVal = ۷۵;
    // loc1.yVal = ۲۲۵;
    Console.WriteLine(loc1);
}
```

یک خطای کامپایل به صورت زیر خواهید گرفت.

Use of unassigned local variable 'loc1'

زمانی که همه مقادیر را انتساب دهید، می‌توانید از طریق خصوصیات x, y به آنها دستیابی کنید.

```
static void Main()
{
    Location loc1;
    loc1.xVal = ۷۵; // assign member variable
    loc1.yVal = ۲۲۵; // assign member variable
    loc1.x = ۳۰۰; // use property
    loc1.y = ۴۰۰; // use property
    Console.WriteLine(loc1);
}
```

زمان استفاده از خصوصیات دقت کنید. اگرچه کپسوله کردن فیلدهای خصوصی از طریق خصوصیات امکان‌پذیر است. در واقع خصوصیات، متدهای عضو هستند و تا زمانی که همه متغیرهای عضو را مقداردهی اولیه نکرده باشید، نمی‌توانید یک متد عضو را فراخوانی کنید.

۹-۳- خلاصه

- ساختار یک نوع داده تعریف شده کاربری است
- ساختارها را فقط برای انواع داده‌ی که کوچک، ساده و مشابه انواع داخلی C# هستند بکار برید.
- ساختار نوع‌داده‌ی مقداری است، در حالیکه کلاس نوع داده‌ی ارجاعی است.
- ایجاد یک نمونه از یک ساختار، شبیه کلاس است، اما بدون کلمه‌ی کلیدی new نیز می‌توان عمل کرد.
- در ساختارها نیز می‌توان از پنهان‌سازی و کپسوله کردن داده‌ها استفاده کرد.
- به راحتی می‌توان فیلدهای public یک ساختار را به خصوصیات public تبدیل کرد. یعنی اعضای ساختار private باشند. (با کمک Refactor Encapsulate Field)
- نمی‌توان در تعریف ساختار از ارث‌بری استفاده کرد، ولی ساختار می‌تواند چندین واسط را پیاده‌سازی کند.
- هنگام ارسال یک ساختار به تابع، یک کپی از آن به ساختار ارسال می‌شود.
- ساختار می‌تواند متدهای کلاس Object را override کند.
- هنگام ارسال یک نمونه از ساختار به متدهای کنسولی، آنها جعبه‌بندی می‌شوند.
- در تعریف ساختار مقداردهی اولیه اعضای آن امکان‌پذیر نیست.

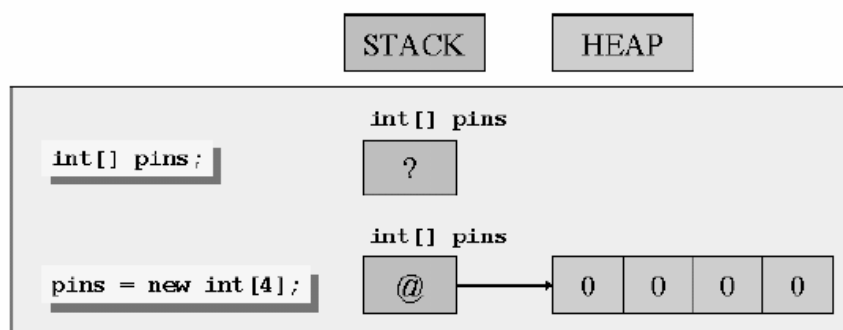
کلاس‌های کلکسیون

آنچه که در این فصل یاد خواهید گرفت:

- کلکسیون‌ها و نحوه‌ی استفاده از آنها برای نگهداشتن مجموعه‌ای از داده‌ها
- تفاوت مابین آرایه‌ها و کلکسیون‌ها
- انواع مختلفی از کلکسیون‌ها (پشته، صف،...)
- آشنایی با متدهای عمومی کلکسیون‌ها

آرایه‌ها مفید هستند، اما آنها محدودیت‌های خودشان را دارند. با این وجود، آرایه‌ها تنها راه جمع‌آوری داده‌های هم‌نوع هستند. چارچوب NET چندین کلاس برای جمع‌آوری عناصر با هم بوسیله‌ی روش‌های دیگری فراهم می‌سازد. اینها کلاس‌های کلکسیون^۱ هستند و در فضای نامی `System.Collections` قرار دارند.

شکل ۱۰-۱

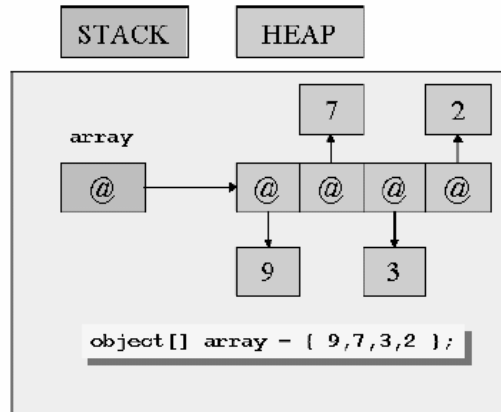


کلاس‌های پایه‌ی کلکسیون عناصر خود را بصورت `object` پذیرفته، نگه داشته و بر می‌گردانند. یعنی نوع داده‌ی عناصر داخل کلاس کلکسیون `object` است. برای درک این مفهوم، مقایسه‌ی یک آرایه از متغیرهای `int` (نوع مقداری) با یک آرایه از `object` (نوع داده‌ی ارجاعی) یاری دهنده است. چون `int` یک نوع داده‌ی مقداری است، این آرایه، مقادیر متغیرها را مستقیماً نگه می‌دارد (همانطور که در شکل ۱۰-۱ می‌بینید).

^۱ Collection

حال بررسی کنید اگر عناصر آرایه از نوع object باشد، چه تاثیری دارد. شما هنوز می‌توانید مقادیر صحیح را به این آرایه اضافه کنید (می‌توانید مقداری از هر نوع را اضافه کنید). زمانی که یک مقدار صحیح اضافه می‌کنید، آن مقدار بطور اتوماتیک جعبه‌بندی می‌شود و هر عنصر آرایه (یک ارجاع به شی) به کپی جعبه‌بندی شده‌ی مقدار صحیح اشاره می‌کند. این عمل در شکل ۱۰-۲ نمایش داده می‌شود.

شکل ۱۰-۲



به خاطر دارید که نوع داده‌ی عناصر یک کلکسیون object است. یعنی در زمان درج یک مقدار به یک کلکسیون، آن مقدار جعبه‌بندی می‌شود و زمانی که یک مقدار را از کلکسیون می‌خوانید، باید با استفاده از قالب بندی آن را از جعبه درآورد. بخش‌های بعدی یک مرور کوتاه روی ۴ کلاس بسیار مفید فراهم می‌کنند. برای جزئیات بیشتر با مستندات چارچوب NET. مراجعه کنید.

۱۰-۱- کلاس ArrayList

ArrayList یک کلاس مفید برای مخلوط کردن عناصر در یک آرایه است. این کلاس امکانات مناسبی دارد که آرایه‌ی معمولی از آنها محروم است.

- اگر بخواهید یک آرایه را تغییر اندازه دهید، باید آرایه‌ی جدیدی ایجاد کرده و عناصر آن را کپی کنید و سپس ارجاع‌های آرایه را بروز کنید.
- اگر بخواهید یک عنصر را از آرایه حذف کنید، باید یک کپی از آن عنصر ایجاد کنید و عناصر بعد از آن را به یک خانه قبلی خود جابجا کنید. نه تنها این عمل کند می‌باشد، بلکه از عنصر آخری دو کپی داریم.
- اگر بخواهید یک عنصر به آرایه درج کنید، باید همه عناصر را یک خانه به جلو جابجا کرده و عنصر جدید را در خانه میانی درج کنید. در این حالت آخرین عنصر را از دست می‌دهید.
- در اینجا با استفاده از کلاس ArrayList می‌توانید به این محدودیت‌ها چیره شوید.
- با استفاده از متد Remove می‌توانید یک عنصر از ArrayList حذف کنید. ArrayList بطور اتوماتیک عناصر خود را مجدداً مرتب می‌سازد.

توجه: نمی‌توانید متد Remove را در یک حلقه‌ی foreach برای طی کردن سراسر یک ArrayList بکار ببرید.

- با استفاده از متد Add می‌توانید یک عنصر به انتهای ArrayList اضافه کنید. شما عنصر مورد نظر جهت اضافه شدن را تهیه می‌کنید و ArrayList در صورت نیاز خودش را تغییر اندازه می‌دهد.

- با استفاده از متد Insert می‌توانید یک عنصر میان عناصر ArrayList درج کنید. در صورت نیاز، ArrayList اندازه‌ی خودش را تغییر می‌دهد.

مثال ۱۰-۱ نحوه‌ی ایجاد، دستکاری و طی کردن سراسر محتویات یک ArrayList را نشان می‌دهد.

مثال ۱۰-۱

```
using System;
using System.Collections;
...
ArrayList numbers = new ArrayList();
...
// fill the ArrayList
foreach (int number in new int[12]{10,9,8,7,7,6,5,10,4,3,2,1})
{
    numbers.Add(number);
}
...
// remove first element whose value is 7 (the 4th element, index 3)
numbers.Remove(7);
// remove the element that's now the 7th element, index 6 (10)
numbers.RemoveAt(6);
...
// iterate remaining 10 elements using a for statement
for (int i = 0; i != numbers.Count; i++)
{
    int number = (int)numbers[i]; // Notice the cast
    Console.WriteLine(number);
}
...
// iterate remaining 10 using a foreach statement
foreach (int number in numbers) // No cast needed
{
    Console.WriteLine(number);
}
```

کد را اجرا کرده و خروجی آن را مشاهده کنید.

توجه: می‌توان تعداد عناصر داخل یک کلکسیون را با خصوصیت Count بدست آورد. در آرایه از خصوصیت Length استفاده می‌شود.

۱۰-۲-کلاس Queue

کلاس Queue یک مکانیزم FIFO (اولین ورودی-اولین خروجی) را پیاده‌سازی می‌کند. یک عنصر جدید به انتهای صف درج می‌شود (عمل Enqueue) و از جلوی صف حذف می‌شود (Deque). مثال ۱۰-۲ از صف و عملیات آن است.

مثال ۱۰-۲

```
using System;
using System.Collections;
...
Queue numbers = new Queue();
...
// fill the queue
foreach (int number in new int[4]{9, 3, 7, 2})
{
    numbers.Enqueue(number);
    Console.WriteLine(number + " has joined the queue");
}
...
```

```
// iterate through the queue
foreach (int number in numbers)
{
    Console.WriteLine(number);
}

...
// empty the queue
while (numbers.Count != 0)
{
    int number = (int)numbers.Dequeue();
    Console.WriteLine(number + " has left the queue");
}
```

برنامه را اجرا کرده و خروجی آن را مشاهده کنید.

۱۰-۳-کلاس Stack

کلاس Stack یک مکانیزم LIFO (اولین ورودی-آخرین خروجی) را پیاده‌سازی می‌کند. یک عنصر به بالای پشته اضافه می‌شود (عمل Push) و یک عنصر از بالای پشته حذف می‌شود (عمل Pop). برای درک تصویری، یک پشته از سینی‌ها را در نظر بگیرید. سینی‌های جدید به بالا اضافه می‌شوند و سینی‌ها از بالا برداشته می‌شوند. پس آخرین سینی گذاشته شده، اولین سینی است که برداشته خواهد شد. مثال ۱۰-۳ کار پشته را نشان می‌دهد.

مثال ۱۰-۳

```
using System;
using System.Collections;
...
Stack numbers = new Stack();
...
// fill the stack
foreach (int number in new int[4]{9, 3, 7, 2})
{
    numbers.Push(number);
    Console.WriteLine(number + " has been pushed on the stack");
}

...
// iterate through the stack
foreach (int number in numbers)
{
    Console.WriteLine(number);
}

...
// empty the stack
while (numbers.Count != 0)
{
    int number = (int)numbers.Pop();
    Console.WriteLine(number + "has been popped off the stack");
}
```

برنامه را اجرا کرده و خروجی را مشاهده کنید.

۱۰-۴-کلاس Hashtable

انواع داده‌ی آرایه و ArrayList یک روش برای نگاشت یک اندیس به یک عنصر فراهم می‌کنند. شما یک عدد صحیح را به عنوان اندیس در داخل گروه باز و بسته قرار می‌دهید.

شاید در بعضی مواقع لازم باشد یک نوع داده‌ی غیر صحیح همچون رشته، تاریخ یا عدد اعشاری را به یک مقدار دیگر نگاشت کنید. در زبان‌های دیگر، این یک آرایه انجمنی^۱ خوانده می‌شود. کلاس `HashTable` نگهداری دو آرایه از نوع `object` برای این عمل فراهم می‌سازد. یکی از آرایه‌ها برای نگه داشتن کلیدهای نگاشت و دیگری آرایه‌ای برای نگه‌داری مقادیر نگاشت شده است. زمانی که یک زوج کلید/مقدار را به جدول `Hash` درج می‌کنید، آن بطور اتوماتیک، کلید را به مقدار مورد نظر مرتبط می‌سازد و شما را برای بازیابی مقدار تخصیص یافته به کلید قادر می‌سازد. چندین پیامد مهم از طراحی `HashTable` وجود دارد:

- یک `HashTable` نمی‌تواند کلید تکراری داشته باشد. اگر متد `Add` را برای اضافه کردن یک کلید بکار برید و این کلید در آرایه‌ی کلیدها موجود باشد، یک استثناء رخ می‌دهد. ولی اگر از علامت `[]` برای اضافه کردن زوج کلید/مقدار استفاده کنید، در صورت بودن کلید، فقط مقدار مربوط به آن کلید را بروز می‌کند. می‌توانید با متد `ContainKey` وجود یک کلید در `HashTable` را آزمایش کنید.
- زمانی که یک دستور `foreach` را برای طی کردن یک `HashTable` بکار می‌برید، یک مقدار از نوع `DictionaryEntry` برگردانده می‌شود. کلاس `DictionaryEntry` دسترسی به عناصر کلید و مقادیر دو آرایه را با خصوصیت‌های `Key` و `Value` فراهم می‌سازد.

مثال ۱۰-۴، سن اعضای خانواده را به اسامی آنها اختصاص می‌دهد و سپس آنها را در خروجی چاپ می‌کند.

مثال ۱۰-۴

```
using System;
using System.Collections;
...
Hashtable ages = new Hashtable();
...
// fill the SortedList
ages["John"] = 41;
ages["Diana"] = 42;
ages["James"] = 13;
ages["Francesca"] = 11;
...
// iterate using a foreach statement
// the iterator generates a DictionaryEntry object containing a key/value pair
foreach (DictionaryEntry element in ages)
{
    string name = (string)element.Key;
    int age = (int)element.Value;
    Console.WriteLine("Name: {0}, Age: {1}", name, age);
}
```

خروجی این برنامه به صورت زیر است:

```
Name: James, Age: 13
Name: John, Age: 41
Name: Francesca, Age: 11
Name: Diana, Age: 42
```

۱۰-۵- کلاس `SortedList`

کلاس `SortedList` بسیار شبیه کلاس `HashTable` است. آن اختصاص دادن کلیدها به مقادیر را مجاز می‌دارد. تفاوت این است که آرایه‌ی کلیدها همواره مرتب شده است. زمانی که یک زوج مقدار/کلید به `SortedList` درج می‌کنید، کلید در اندیس مناسبی از آرایه‌ی کلیدها درج می‌شود تا این آرایه همچنان مرتب شده بماند و مقدار مرتبط با آن کلید نیز در همان

^۱ Associative

اندیس از آرایه مقادیر قرار می‌گیرد. کلاس SortedList ترازبندی اتوماتیک کلیدها و مقادیر را تضمین می‌کند (حتی زمانی که عناصری را حذف یا درج کنید). یعنی، شما می‌توانید زوج مقدار/کلید را با هر ترتیب دلخواه خود به SortedList اضافه کنید، آنها بر اساس مقادیر کلیدها مرتب می‌شوند.

شبهه کلاس Hashtable، یک SortedList نمی‌تواند کلیدهای تکراری داشته باشد. زمانی که یک دستور foreach برای طی کردن سراسر یک SortedList استفاده می‌کنید، یک DictionaryEntry برگردانده می‌شود. با این وجود، اشیاء DictionaryEntry برگردانده شده بر اساس خصوصیت Key مرتب شده هستند.

مثال ۱۰-۵، سن اعضای یک خانواده را به اسامی آنها اختصاص می‌دهد و سپس آنها را چاپ می‌کند. البته به جای Hashtable یک SortedList بکار می‌برد.

مثال ۱۰-۵

```
using System;
using System.Collections;
...
SortedList ages = new SortedList();
...
// fill the SortedList
ages["John"] = 39;
ages["Diana"] = 40;
ages["James"] = 12;
ages["Francesca"] = 10;
...
// iterate using a foreach statement
// the iterator generates a DictionaryEntry object containing a key/value pair
foreach (DictionaryEntry element in ages)
{
    string name = (string)element.Key;
    int age = (int)element.Value;
    Console.WriteLine("Name: {0}, Age: {1}", name, age);
}
```

خروجی این برنامه بر اساس اسامی اعضای خانواده مرتب می‌شوند.

```
Name: Diana, Age: 40
Name: Francesca, Age: 10
Name: James, Age: 12
Name: John, Age: 39
```

۱۰-۶- کلاس BitArray

کلاس BitArray از مقادیر بیتی تشکیل شده است. مقدار بیت‌ها ۱ و ۰ هستند که ۱ به معنی true و ۰ به معنی false است. این کلاس مفهوم کاراتری از ذخیره و بازیابی مقادیر بیتی فراهم می‌کند. جدول ۱۰-۱ متدها و خصوصیات خاص BitArray را لیست می‌کند.

جدول ۱۰-۱ اعضای کلاس BitArray

نام عضو	گرامر
سازنده‌ی این کلاس، overload می‌شود. اینها چند تا از overLoadهای آن هستند.	<pre>(BitArray(bool[] bits) (BitArray(int[] bits) (BitArray(int count, bool default</pre>

<code>(BitArray And(BitArray value</code>	<code>And</code> : این متد یک عمل <code>And</code> بیتی روی شی جاری و پارامتر <code>BitArray</code> انجام می‌دهد و نتیجه در <code>BitArray</code> قرار داده می‌شود.
<code>(Bool Get(int index</code> <code>virtual bool IsReadOnly</code> <code>{;get}</code>	<code>Get</code> : این متد مقدار یک بیت خاص از <code>BitArray</code> را برمی‌گرداند. <code>IsReadOnly</code> : اگر کلکسیون فقط خواندنی باشد مقدار <code>true</code> برمی‌گرداند، در غیر اینصورت <code>false</code> بر می‌گرداند.
<code>[virtual object this [int index</code> <code>}</code> <code>;get; set</code> <code>{</code>	<code>Item</code> : این خصوصیت مقدار یک بیت از اندیس خاص را بدست آورده یا مقداردهی می‌کند.
<code>} public int length</code> <code>;get;set</code> <code>{</code>	<code>Length</code> : تعداد بیت های موجود در کلکسیون را بدست آورده یا مقداردهی می‌کند.
<code>()BitArray Not</code>	<code>Not</code> : این متد بیت‌های کلکسیون را مکمل ۱ می‌کند. این نتیجه در <code>BitArray</code> برگردانده می‌شود.
<code>(BitArray Or(BitArray value</code>	<code>Or</code> : این متد یک عمل <code>or</code> بیتی روی پارامتر <code>BitArray</code> و شی جاری انجام می‌دهد. نتیجه در <code>BitArray</code> برگردانده می‌شود.
<code>(void Set(int index, bool value</code>	<code>Set</code> : یک بیت خاصی را مقداردهی می‌کند.
<code>(Void SetAll(bool value</code>	<code>SetAll</code> : این متد همه بیت‌های کلکسیون را <code>true</code> یا <code>false</code> قرار می‌دهد.
<code>(BitArray Xor(BitArray value</code>	<code>Xor</code> : این متد روی شی جاری و پارامتر <code>BitArray</code> عمل <code>xor</code> بیتی انجام می‌دهد.
متد <code>GetEnumerator()</code> را پیاده‌سازی می‌کند.	اعضای <code>IEnumerable</code>
متد <code>Clone()</code> را پیاده‌سازی می‌کند.	اعضای <code>ICloneable</code>
<code>Count</code> , <code>CopyTo()</code> , <code>IsSynchronized</code> و <code>SyncRoot</code> را در بر دارد.	اعضای <code>ICollection</code>

مثال ۱۰-۶ یک کاربرد از کلاس `BitArray` را نشان می‌دهد. کلاس `Employee` یک کلاس `BitArray` دارد که ثبت نام کارمند در برنامه‌های مختلف را پی‌گیری می‌کند (همچون طرح سلامت و). چون ثبت نام فقط دو حالت `true` یا `false` دارد، پس می‌توان از این کلکسیون استفاده کرد. در کلاس `Employee`، خصوصیتی برای خواندن یا مقداردهی ثبت نام در برنامه‌های مختلف فراهم شده است.

مثال ۱۰-۶

```
using System;
```

```

using System.Collections;
namespace Arshia.CsharpBook{
public class Starter{
public static void Main(){
Employee ben=new Employee();
ben.InProfitSharing=false;
ben.InHealthPlan=false;
Employee Valerie=new Employee();
Valerie.InProfitSharing=false;
Participation("Ben",ben);
Participation("Valerie",Valerie);
}
public static void participation(string name, Employee person)
{
Console.WriteLine(name+":");
if (person.InProfitSharing)
{
Console.WriteLine("participating in "+ "Profit Sharing");
}
if (person.InHealthPlan)
{
Console.WriteLine("participating in "+"Health Plan");
If (person.InCreditUnion)
{
Console.WriteLine("participating in " + "Credit Union");
}
}
}

public class Employee
{
public Employee()
{
eFlags.SetAll(true);
}
private BitArray eFlags=new BitArra(۳);
public bool InProfitSharing
{
set
{
eFlags.Set(۰,value);
}
get
{
return eFlags.Get(۰);
}
}
public bool InHealthPlan
{
set
{
eFlags.Set(۱,value);
}
get{
return eFlags.Get(۱);
}
}
public bool InCreditUnion
{
set
{
eFlags.Set(۲,value);
}
get
{

```

```
return eFlags.Get(۲);
}
}
}
}
```

۱۰-۷-مقایسه‌ی آرایه‌ها و کلکسیون‌ها

موارد زیر خلاصه‌ای از تفاوت‌های مهم مابین آرایه‌ها و کلکسیون‌ها هستند.

- نوع داده‌ی عناصر نگه داشته شده در یک آرایه اعلان می‌شود، در حالیکه کلکسیون این کار را انجام نمی‌دهد. بدین دلیل که کلکسیون‌ها اعضاء خود را بصورت object ذخیره می‌کنند.
- یک آرایه اندازه‌ی ثابت دارد و نمی‌تواند بزرگ یا کوچک شود. یک کلکسیون می‌تواند بطور اتوماتیک اندازه‌ی خود را تغییر دهد.
- آرایه یک ساختار خواندنی/نوشتنی است. هیچ روشی برای ایجاد آرایه‌ی فقط خواندنی نیست. با این وجود، کاربرد کلکسیون‌ها در متد فقط خواندنی امکان پذیر است. این کلاس‌ها متد ReadOnly را فراهم می‌سازند.

۱۰-۸-کاربرد کلاس‌های کلکسیون برای بازی کارت‌ها

این بخش یک برنامه‌ی کاربردی را شرح می‌دهد که تخصیص یک بسته کارت به چهار بازیکن را شبیه‌سازی می‌کند. کارت‌ها در بسته هستند یا در اختیار یکی از چهار بازیکن هستند. بسته و کارت‌های موجود در دست بصورت اشیائی از نوع ArrayList پیاده‌سازی می‌شوند. شاید فکر کنید اینها باید بصورت یک آرایه پیاده‌سازی شوند. همواره ۵۲ کارت در بسته و ۱۳ کارت در دست وجود دارد. این درست است، اما از این حقیقت چشم‌پوشی می‌کند، زمانی که کارت‌ها به دست بازیکنان داده می‌شود، آنها در بسته نخواهند بود. اگر یک آرایه برای پیاده‌سازی یک بسته بکار برید، شما مجبور هستید تعداد کارت‌های نگه داشته شده در دست بازیکنان را ثبت کنید. بطور مشابه، زمانی که کارت‌ها از دست بازیکنان به بسته برگردانده می‌شوند، باید کارت‌های موجود در دست بازیکنان را نیز ثبت کنید.

۱۰-۹-خلاصه

- کلکسیون‌ها اشیائی از نوع object نگه می‌دارند.
- کلکسیون‌ها بسیاری از محدودیت‌های آرایه‌ها را حذف می‌کنند.
- برخی از ساختمان‌داده‌ها به صورت کلکسیون‌های آماده در FCL وجود دارند.
- هنگام اضافه کردن یک عنصر به کلکسیون، آن بطور اتوماتیک جعبه‌بندی می‌شود. در هنگام استفاده از عناصر داخل کلکسیون، باید آنها را صریحا از جعبه در آورد.
- اندازه‌ی کلکسیون‌ها برخلاف آرایه‌ها قابل تغییر است.

genericها

آنچه که در این فصل یاد خواهید گرفت:

- آشنایی با genericها برای پیاده‌سازی الگوریتم‌ها بدون وابستگی به نوع داده
- ایجاد genericها و استفاده از آنها در تعریف نوع داده‌ها
- آشنایی با محدودیت‌های تعریف genericها و غلبه بر آنها
- نحوه‌ی کار با genericهای موجود در FCL

^۱ genericها جزء قدرتمندترین و مفیدترین ویژگی‌های NET ۲.۰ هستند. genericها به شما اجازه می‌دهند، یک ساختار داده‌ی نوع امن، یک کلاس یا ^۲سودمند تعریف کنید، بدون اینکه نوع داده‌ی واقعی مورد استفاده آن را مشخص کنید. genericها به پیشرفت مهمی در کارایی و کیفیت کد منجر می‌شوند، چون می‌توانید بدون نیاز به پیاده‌سازی مجدد یک الگوریتم آن را با ساختارهای داده‌ای دیگر بکار ببرید. genericها شبیه الگوهای ++C هستند، اما از نظر پیاده‌سازی و توانایی‌ها باهم تفاوت دارند.

۱۱-۱- تعریف generic

یک کلاس generic شبیه یک کلاس عادی تعریف می‌شود، ولی بعد از نام کلاس نوع generic مشخص می‌گردد. نوع داده‌ی generic بوسیله کلاس، می‌توانند به عنوان نوع داده‌ی یک عضو از کلاس یا به عنوان نوع داده‌ی پارامترهای متدها استفاده شوند.

```
public class MyGeneric <T>
{ private T member;
Public void Method (T obj)
{
...
}
}
```

هر کلاسی می‌تواند از نوع داده‌ی object بصورت یک نوع generic استفاده کند. ولی به کمک genericها می‌توان نوع داده‌ی مورد نیاز در داخل کلاس را هنگام اعلان یک نمونه از آن کلاس مشخص کرد.

^۱ Generic

^۲ Helper

^۳ Template

شما سرویس دهنده را فقط یکبار به عنوان یک سرویس دهنده generic پیاده سازی می کنید و سپس می توانید آن را با هر نوع داده ای بکار ببرید. مثال زیر نحوه ی تعریف و کاربرد یک Stack generic را نشان می دهد.

```
public class Stack<T>
{
    T[] m_Items;
    public void Push(T item)
    {...}
    public T Pop( )
    {...}
}
Stack<int> stack = new Stack<int>( );
stack.Push(۱);
stack.Push(۲);
int number = stack.Pop( );
```

مثال ۱۱-۱ پیاده سازی کامل Stack generic را نشان می دهد.

مثال ۱۱-۱

```
public class Stack<T>
{
    const int DefaultSize = ۱۰۰;
    readonly int m_Size;
    int m_StackPointer = ۰;
    T[] m_Items;
    public Stack( ) : this(DefaultSize)
    {}
    public Stack(int size)
    {
        m_Size = size;
        m_Items = new T[m_Size];
    }
    public void Push(T item)
    {
        if(m_StackPointer >= m_Size)
        {
            throw new StackOverflowException( );
        }
        m_Items[m_StackPointer] = item;
        m_StackPointer++;
    }
    public T Pop( )
    {
        m_StackPointer--;
        if(m_StackPointer >= ۰)
        {
            return m_Items[m_StackPointer];
        }
        else
        {
            m_StackPointer = ۰;
            throw new InvalidOperationException("Cannot pop an empty stack");
        }
    }
}
```

کلاس Stack با استفاده از پارامتر نوع T generic تعریف می شود.

```
public class Stack<T>
{
    {...}
}
```

زمانی که Stack generic را بکار می برید، باید مشخص کنید کامپایلر به جای نوع داده ی T generic از چه نوع داده ای استفاده کند. در هر دو حالت اعلان یا تعریف یک نمونه از آن کلاس، تعیین نوع داده ی generic لازم است.

```
Stack<int> stack = new Stack<int>( );
```

کامپایلر و کنترل‌کننده‌ی زمان اجرا بقیه کارها را انجام می‌دهند. همه‌ی متدها (یا خصوصیات) که یک T را پذیرفته یا بر می‌گردانند، به جای T از نوع داده‌ی مشخص شده استفاده خواهند کرد.

توجه: T یک پارامتر نوع داده‌ی generic است. در حالی که Stack<T> یک نوع داده‌ی generic است. مزیت این نوع برنامه‌نویسی به عدم پیاده‌سازی مجدد یک الگوریتم برای انواع داده‌های دیگر است.

۱۱-۲- پیاده‌سازی generic ها

بطور سطحی generic های C# شبیه الگوهای ++C هستند. اما اختلافات مهمی در روش پیاده‌سازی و پشتیبانی توسط کامپایلر دارند. در مقایسه با الگوهای ++C، generic های C# ایمنی نوع داده را بهبود می‌دهند، اما توانایی‌های محدودتری دارند. در بعضی از کامپایلرهای ++C، زمانی که یک کلاس الگو با یک نوع داده‌ی خاص را بکار می‌برید، کامپایلر کد الگو را کامپایل نمی‌کند. زمانی که نوع داده را مشخص می‌کنید، کامپایلر کد را در داخل برنامه درج می‌کند (inline) و هر جا که به پارامتر نوع داده generic برخورد کند، با نوع داده مشخص شده جایگزین می‌کند. این عملیات بر عهده‌ی پیوند دهنده^۱ ++C است و انجام آن همیشه امکان‌پذیر نیست. این عمل زمان بارگذاری برنامه و حافظه مصرفی را افزایش می‌دهد.

در .NET ۲.۰، generic ها در CLR و IL بصورت محلی پشتیبانی می‌شوند. زمانی که کد C# طرف سرویس‌دهنده را کامپایل کنید، شبیه انواع داده‌ای دیگر به IL کامپایل می‌شود. با این وجود، IL فقط پارامترها یا محل‌هایی را برای انواع داده‌ی واقعی در بر می‌گیرد. علاوه بر این، فراداده‌های سرویس‌دهنده‌ی generic، اطلاعات generic را در بر دارند.

کامپایلر طرف سرویس‌گیرنده، این فراداده‌ها را برای پشتیبانی ایمنی نوع داده بکار می‌برد. زمانی که سرویس‌گیرنده یک نوع داده‌ی خاص را به جای نوع داده‌ی generic فراهم می‌کند، کامپایلر سرویس‌گیرنده، پارامتر نوع داده‌ی generic را در فراداده‌های سرویس‌دهنده با نوع داده‌ی خاص جایگزین می‌کند. این عمل یک تعریف از یک generic را با نوع داده‌ی خاص ایجاد می‌کند، همانند اینکه هیچ نوع داده generic وجود نداشته است. بدین روش کامپایلر می‌تواند صحت پارامترهای متد، بررسی ایمنی نوع داده و حتی هوشمندی IDE را برای نوع داده خاص به اجرا در آورد.

سوال جالب این است که چگونه .NET کد IL کلاس generic داخل سرویس‌دهنده را به کد ماشین ترجمه می‌کند. آن بر اساس اینکه نوع داده‌ی مشخص شده در ماشین واقعی از نوع مقداری یا ارجاعی است، تولید می‌گردد. اگر سرویس‌گیرنده یک نوع داده مقداری مشخص کند، کامپایلر JIT پارامترهای نوع داده generic را با نوع مقداری مشخص شده جایگزین می‌کند و کد IL را به کد ماشین محلی کامپایل می‌کند. چون در حالت عادی کد کلاس generic بر اساس نوع مقداری ترجمه شده است، پس کافی است یک ارجاع به کد سرویس‌دهنده برگردانده شود، چون کامپایلر JIT همان کد سرویس‌دهنده را با نوع داده‌ی مشخص شده مقداری بکار می‌برد.

اگر سرویس‌گیرنده یک نوع داده ارجاعی مشخص کند، کامپایلر JIT پارامترهای generic را با object جایگزین کرده و IL را به کد ماشین محلی کامپایل می‌کند. این کد می‌تواند برای همه کاربردهای یک نوع داده ارجاعی بکار رود. توجه کنید که در این روش، JIT فقط کد واقعی را مجدداً استفاده می‌کند. نمونه‌ها براساس اندازه‌ی آنها به Heap مدیریت شده اختصاص داده می‌شوند و هیچ عمل قالب‌بندی وجود ندارد.

^۱ Linker

۱۱-۳-۱۱ اعمال کردن generic ها

به دلیل پشتیبانی محلی generic ها در IL و CLR، همه زبان های مطیع CLR می توانند از مزایای انواع داده generic بهره ببرند. برای مثال کد VB ۲۰۰۵ زیر، کلاس generic مثال ۱۱-۱ را بکار می برد.

```
Dim stack As Stack(Of Integer)
stack = new Stack(Of Integer)
stack.Push(۳)
Dim number As Integer
number = stack.Pop( )
```

می توانید generic ها را در کلاس ها و ساختارها بکار ببرید. این یک ساختار generic مفید از Point است.

```
public struct Point<T>
{
    public T X;
    public T Y;
}
```

می توانید generic Point را با مختصات int بکار ببرید.

```
Point<int> point;
point.X = ۱;
point.Y = ۲;
```

و برای رسم نمودارهایی که دقت اعشار نیاز دارند:

```
Point<double> point;
point.X = ۱,۲;
point.Y = ۳,۴;
```

۱۱-۴-۱۱ انواع داده generic چندگانه

یک نوع داده ی واحد می تواند چندین پارامتر نوع داده ی generic تعریف کند. برای مثال، لیست پیوندی generic نشان داده شده در مثال ۱۱-۲ را ملاحظه کنید.

مثال ۱۱-۲

```
class Node<K,T>
{
    public K Key;
    public T Item;
    public Node<K,T> NextNode;
    public Node( )
    {
        Key = default(K);
        Item = default(T);
        NextNode = null;
    }
    public Node(K key,T item,Node<K,T> nextNode)
    {
        Key = key;
        Item = item;
        NextNode = nextNode;
    }
}
public class LinkedList<K,T>
{
    Node<K,T> m_Head;
    public LinkedList( )
```

```

{
m_Head = new Node<K,T>( );
}
public void AddHead(K key,T item)
{
Node<K,T> newNode = new Node<K,T>(key,item,m_Head.NextNode);
m_Head.NextNode = newNode;
}
}

```

لیست پیوندی گره‌ها را ذخیره می‌کند.

```

class Node<K,T>
{...}

```

هر گره یک کلید (پارامتر نوع K generic) و یک مقدار (پارامتر نوع T generic) را در بر دارد. هر گره‌ای یک ارجاع به گره بعدی لیست دارد. لیست پیوندی نیز با پارامترهای نوع K generic و T تعریف می‌شود.

```

public class LinkedList<K,T>
{...}

```

این عمل به لیست اجازه می‌دهد، متدهای generic همچون AddHead() را در اختیار قرار دهد.

```

public void AddHead(K key,T item);

```

زمانی که یک متغیر از نوع یک کلاس generic اعلان می‌کنید، باید انواع داده‌ی مورد استفاده را مشخص کنید. با این وجود، خود انواع داده‌ای مشخص شده می‌توانند انواع داده‌ای generic باشند. برای مثال، لیست پیوندی یک متغیر عضو بنام m_Head از نوع Node<K, T> دارد، که به اولین عنصر لیست پیوندی ارجاع می‌کند. m_Head با استفاده از پارامترهای نوع داده‌ی K generic و T اعلان می‌شود.

```

Node<K,T> m_Head;

```

لازم است در زمان تعریف یک گره، انواع داده‌ی مشخص را فراهم سازید. می‌توانید پارامترهای نوع generic مربوط به خود لیست پیوندی را بکار ببرید.

```

public void AddHead(K key,T item)
{
Node<K,T> newNode = new Node<K,T>(key,item,m_Head.NextNode);
m_Head.NextNode = newNode;
}

```

توجه کنید که برای بالابردن قابلیت خوانایی، اسامی پارامترهای نوع generic لیست پیوندی و گره یکسان هستند، ولی می‌توانند اسامی دیگری داشته باشند، همچون:

```

public class LinkedList<U,V>
{...}

```

یا

```

public class LinkedList<KeyType,DataType>
{...}

```

در این حالت، m_Head بصورت زیر اعلان می‌شود:

```

Node<KeyType,DataType> m_Head;

```

زمانی که سرویس‌گیرنده از لیست پیوندی استفاده می‌کند، آن باید انواع داده‌ی مشخصی را فراهم سازد. سرویس‌گیرنده می‌تواند int را برای کلید و string را برای عناصر انتخاب کند.

```

LinkedList<int,string> list = new LinkedList<int,string>( );
list.AddHead(۱۲۳,"AAA");

```

همچنین می‌توانید هر ترکیب از انواع داده دیگر را انتخاب کنید:

```
LinkedList<DateTime,string> list = new LinkedList<DateTime,string>( );
list.AddHead(DateTime.Now, "AAA");
```

۱۱-۵- محدودیت‌های generic

بوسیله generic های #C، کامپایلر کد generic را بطور مستقل از نوع داده‌ی انتخاب شده توسط سرویس گیرنده، به IL ترجمه می‌کند. در نتیجه، کد generic می‌تواند سعی بر کاربرد متدها، خصوصیات یا اعضای پارامترهای نوع generic داشته باشد، در حالیکه با نوع داده مشخص شده توسط کاربر ناسازگار است. این غیر قابل قبول است، چون عامل از بین بردن ایمنی نوع داده است. در #C، نیاز دارید کامپایلر را برای در نظر گرفتن محدودیت‌های مربوط به انواع داده استفاده شده در سرویس گیرنده به جای نوع generic راهنمایی کنید. سه نوع محدودیت وجود دارد که در زیر به آنها اشاره می‌شود:

- محدودیت‌های مشتق

- به کامپایلر نشان می‌دهد یک پارامتر نوع generic از کدام نوع پایه همچون واسط یا کلاس پایه‌ی خاص مشتق می‌شود.

- محدودیت سازنده‌ی پیش فرض

- به کامپایلر نشان می‌دهد که پارامتر نوع generic سازنده‌ی عمومی پیش فرض را دارد.

- محدودیت نوع داده مقداری یا ارجاعی

- پارامتر نوع generic را به یک نوع داده‌ی مقداری یا ارجاعی محدود می‌کند.

یک نوع generic می‌تواند چندین محدودیت را بکار گیرد و در زمان کاربرد می‌توانید سیستم هوشمند IDE را برای انعکاس محدودیت‌های پارامتر نوع generic استفاده کنید. (مانند پیشنهاد متدها یا خصوصیات از نوع داده‌ی پایه)

توجه: اگرچه محدودیت‌ها اختیاری هستند، اغلب در زمان توسعه یک نوع داده‌ی generic ضروری هستند. بدون آنها، کامپایلر یک روش محافظه کار و ایمن نوع داده بکار می‌برد و فقط دسترسی به عملکرد object در پارامترهای نوع داده‌ی generic را مجاز می‌دارد.

محدودیت‌ها بخشی از فراداده‌ی نوع داده‌ی generic هستند. بنابراین سرویس گیرنده می‌تواند مزایای آنها را به خوبی استفاده کند. کامپایلر طرف سرویس گیرنده به توسعه‌دهنده‌ی سرویس گیرنده اجازه می‌دهد فقط از انواع داده‌ای مطابق با محدودیت‌ها و ایمنی نوع داده استفاده کند.

یک مثال برای شرح نیاز به کاربرد محدودیت‌ها بصورت زیر آمده است. فرض کنید دوست دارید، جستجو کردن و اندیس گذاری بوسیله توانایی‌های کلید را به لیست پیوندی مثال ۱۱-۲ اضافه کنید.

```
public class LinkedList<K,T>
{
    T Find(K key)
    {...}
    public T this[K key]
    {
        get
        {
            return Find(key);
        }
    }
}
```

این مثال به سرویس گیرنده نوشتن کد زیر را اجازه می‌دهد:

```
LinkedList<int,string> list = new LinkedList<int,string>( );
list.AddHead(۲۲,"AAA");
list.AddHead(۴۵۶,"BBB");
string item = list[۴۵۶];
Debug.Assert(item == "BBB");
```

برای پیاده‌سازی عمل جستجو، لازم است لیست را پیمایش کرده و کلید هر گره را با کلید موجود مقایسه کنید و عنصر گره مرتبط با کلید را برگردانید. مشکل این است که پیاده‌سازی Find() بصورت زیر کامپایل نمی‌شود:

```
T Find(K key)
{
    Node<K,T> current = m_Head;
    while(current.NextNode != null)
    {
        if(current.Key == key) //Will not compile
        {
            break;
        }
        else
        {
            current = current.NextNode;
        }
    }
    return current.Item;
}
```

کامپایلر خط زیر را نمی‌پذیرد:

```
if(current.Key == key)
```

چون نمی‌داند آیا K عملگر == را پشتیبانی می‌کند. برای مثال، ساختارها این پیاده‌سازی را فراهم نمی‌کنند. می‌توانید بر محدودیت عملگر == با استفاده از واسط IComparable چیره شوید.

```
public interface IComparable
{
    int CompareTo(object obj);
}
```

در صورتی که شی موردنظر شما با شی پیاده‌سازی‌کننده یکسان باشد، صفر بر می‌گرداند. پس متد Find() می‌تواند آن را بصورت زیر بکار برد.

```
if(current.Key.CompareTo(key) == ۰)
```

متأسفانه این کد نیز کامپایل نمی‌شود، چون کامپایلر به هیچ روشی نمی‌داند که K واسط IComparable را پیاده‌سازی می‌کند.

با قالب‌بندی صریح می‌توانید کامپایلر را برای کامپایل کردن این خط مجبور سازید. اما ایمنی نوع داده را از دست می‌دهید.

```
if(((IComparable)(current.Key)).CompareTo(key) == ۰)
```

اگر نوع داده‌ی سرویس‌گیرنده از IComparable مشتق نشود، در زمان اجرا یک استثنا رخ می‌دهد. بعلاوه، زمانی که نوع داده مشخص شده، یک نوع داده‌ی مقداری باشد، یک جعبه‌بندی اجباری از کلید انجام می‌شود که برخی دلایل بهره‌وری را تحت تأثیر قرار می‌دهد.

۱۱-۵-۱- محدودیت‌های مشتق

در C#، کلمه کلیدی where برای تعریف یک محدودیت استفاده می‌شود. کلمه کلیدی where را روی پارامتر نوع داده‌ی generic بکار ببرید که به دنبال آن : و بعد از آن نام واسط مورد نظر جهت پیاده‌سازی مشخص می‌گردد. برای مثال، در کد زیر یک محدودیت مشتق برای پیاده‌سازی متد Find() کلاس LinkedList لازم است.

```
public class LinkedList<K,T> where K : IComparable
```

```

{
T Find(K key)
{
Node<K,T> current = m_Head;
while(current.NextNode != null)
{
if(current.Key.CompareTo(key) == 0)
{
break;
}
else
{
current = current.NextNode;
}
}
return current.Item;
}
//Rest of the implementation
}

```

توجه: اگرچه این محدودیت استفاده از *IComparable* را مجاز می‌دارد، اما هنوز عیب جعبه‌بندی نوع داده‌ی مقداری حذف نشده است. برای غلبه بر این مشکل، فضای نامی *System* واسط *generic IComparable<T>* را تعریف می‌کند.

```

public interface IComparable<T>
{
int CompareTo(T other);
}

```

می‌توانید پارامتر نوع داده‌ی کلید را برای پشتیبانی از *IComparable<T>* محدود کنید. با این عمل نه تنها ایمنی نوع داده را بدست می‌آورید، جعبه‌بندی انواع داده‌ی مقداری نیز حذف می‌گردد.

```

public class LinkedList<K,T> where K : IComparable<K>
{...}

```

در حقیقت، همه انواع داده‌ای که در .NET ۱،۱ واسط *IComparable* را پشتیبانی می‌کنند، در .NET ۲،۰ نیز *IComparable<T>* را پشتیبانی می‌کنند. این عمل شما را قادر می‌سازد، از انواع داده‌ی معمول (string, int, ...) برای کلیدها استفاده کنید. در حالیکه *IComparable<T>* برای مرتب‌سازی و منظم‌کردن طراحی می‌شود، .NET برای عمل مقایسه واسط *generic IEquatable<T>* را تعریف می‌کند.

```

public interface IEquatable<T>
{
bool Equals(T other);
}

```

در #C ۲،۰ همه محدودیت‌ها قبل از لیست مشتق کلاس *generic* ظاهر می‌شوند. برای مثال، اگر *LinkedList* از واسط *IComparable<T>* مشتق می‌شود، کلمه‌ی کلیدی *where* را قبل از آن قرار دهید.

```

public class LinkedList<K,T> : IEnumerable<T> where K : IComparable<K>
{...}

```

زمانی که سرویس‌گیرنده یک متغیر از نوع *LinkedList* اعلان می‌کند که کلید آن یک نوع داده‌ی گسسته^۱ باشد. کامپایلر طرف سرویس‌گیرنده اصرار دارد که نوع داده‌ی کلید آن از *IComparable<T>* مشتق شود و از ایجاد کد سرویس‌گیرنده سرباز می‌زند.

با استفاده از کاما می‌توانید چندین واسط را روی پارامتر نوع داده‌ی *generic* یکسانی محدود کنید. مثال:

```

public class LinkedList<K,T> where K : IComparable<K>, IConvertible
{...}

```

^۱ Concrete

می توانید برای هر پارامتر نوع داده‌ی generic مربوط به یک کلاس محدودیت‌هایی فراهم سازید.

```
public class LinkedList<K,T> where K : IComparable<K>
where T : ICloneable
{...}
```

۱۱-۵-۲- محدودیت سازنده

فرض کنید می‌خواهید یک نمونه از شی generic را در داخل یک کلاس generic تعریف کنید. مشکل اینجاست که کامپایلر نمی‌داند آیا نوع داده‌ی مشخص شده، سازنده‌ی مطابق با حالت مورد استفاده‌ی شما را دارد یا نه. پس کامپایلر عمل کامپایل این خط را نمی‌پذیرد. برای حل این مشکل، #C به شما اجازه می‌دهد یک پارامتر نوع داده‌ی generic را برای پشتیبانی یک سازنده‌ی پیش فرض public محدود کنید. این بوسیله محدودیت new() انجام می‌شود. مثال زیر یک روش متفاوت برای پیاده‌سازی سازنده‌ی پیش فرض نوع داده‌ی <generic Node<K,T> مثال ۱۱-۲ را نشان می‌دهد.

```
class Node<K,T> where K : new( )
where T : new( )
{
public K Key;
public T Item;
public Node<K,T> NextNode;
public Node( )
{
Key = new K( );
Item = new T( );
NextNode = null;
}
//Rest of the implementation
```

می‌توانید محدودیت سازنده را با محدودیت مشتق ترکیب کنید. محدودیت سازنده در آخر لیست محدودیت ظاهر می‌گردد.

```
public class LinkedList<K,T> where K : IComparable<K>,new( )where T : new( )
{...}
```

۱۱-۵-۳- محدودیت نوع مقداری / ارجاعی

می‌توانید یک پارامتر نوع داده‌ی generic را طوری محدود کنید که یک نوع داده‌ی مقداری باشد (همچون bool, int, ...)

```
public class MyClass<T> where T : struct
{...}
```

بطور مشابه می‌توانید یک پارامتر نوع داده generic را طوری محدود کنید که یک نوع داده‌ی ارجاعی باشد:

```
public class MyClass<T> where T : class
{...}
```

محدودیت struct/class نمی‌تواند بوسیله کلاس پایه استفاده شود، چون محدودیت کلاس پایه به یک کلاس اشاره می‌کند. بطور مشابه نمی‌توانید محدودیت struct و سازنده‌ی پیش فرض را باهم بکار ببرید، چون سازنده‌ی پیش فرض به یک کلاس اشاره می‌کند. اگرچه می‌توانید محدودیت کلاس و سازنده‌ی پیش فرض را باهم بکار ببرید، اما ارزشی ندارد. می‌توانید محدودیت struct/class را با محدودیت واسط ترکیب کنید، بطوریکه محدودیت struct/class در ابتدای لیست می‌باشد.

۱۱-۶- کلاس‌ها و کلکسیون‌های generic در FCL

در فضای نامی System.Collections.Generic تعداد زیادی کلاس کلکسیون و واسط generic تعریف می‌شوند. این کلاس‌های generic می‌توانند به جای کلاس‌های کلکسیون عادی بکار روند.

۱۱-۶-۱-مروری بر کلسیون های generic

این بخش مروری بر کلاس های کلسیون و واسط های NET. generic است. واسط های اصلی و عملکرد آنها در جدول زیر تعریف می شوند:

واسط	متدها و خصوصیات	توصیف
<code><ICollection<T></code>	<code>Add()</code> , <code>Clear()</code> , <code>Contains()</code> , <code>CopyTo()</code> , <code>Remove()</code> , <code>Count</code> , <code>IsReadOnly</code>	واسط <code><ICollection<T></code> بوسیله کلاس های کلسیون پیاده سازی می شود. متدهای این واسط برای اضافه کردن، حذف کردن عنصر از کلسیون استفاده می شوند. واسط <code><generic ICollection<T></code> از واسط غیر کل <code>IEnumerable</code> ارث بری می کند. پس می توانید اشیائی را که متدهای <code>IEnumerable</code> را لازم دارند به کلسیون <code><ICollection<T></code> رد کنید.
<code><IList<T></code>	<code>Insert()</code> , <code>RemoveAt()</code> , <code>IndexOf()</code> Item	واسط <code><IList<T></code> دسترسی به عناصر یک کلسیون را بوسیله یک اندیس گذار مجاز می دارد. درج یا حذف عناصر در هر موقعیتی از لیست امکان پذیر است. شبیه <code><ICollection<T></code> از <code>IEnumerable</code> ارث بری می کند.
<code><IEnumerable<T></code>	<code>GetEnumerator()</code>	در صورت استفاده از <code>foreach</code> در یک کلسیون، پیاده سازی این واسط لازم است. این واسط متد <code>GetEnumerator()</code> را پیاده سازی می کند که یک شمارنده از نوع <code><IEnumerator<T></code> بر می گرداند.
<code><IEnumerator<T></code>	<code>Current</code>	دستور <code>foreach</code> یک شمارنده برای دستیابی به همه عناصر یک کلسیون بکار می برد. این شمارنده واسط <code><IEnumerator<T></code> را پیاده سازی می کند. واسط <code><IEnumerator<T></code> از واسط غیر <code>generic IEnumerator</code> و <code>IDisposable</code> ارث بری می کند. متدهای <code>Reset</code> , <code>MoveNext()</code> را تعریف می کند. <code><IEnumerator<T></code> نسخه ایمن نوع داده ی خصوصیت <code>Current</code> را تعریف می کند.
<code>IDictionary<TKey, TValue></code>	<code>Add()</code> , <code>ContainsKey()</code> , <code>Remove()</code> , <code>TryGetValue()</code> , <code>Item</code> , <code>Keys</code> , <code>Values</code>	واسط <code><IDictionary<K,V></code> بوسیله کلسیون هایی که عناصر آن یک مقدار و یک کلید دارند پیاده سازی می شوند.
<code><IComparer<T></code>	<code>Compare()</code>	واسط <code><IComparer<T></code> برای مرتب سازی عناصر یک کلسیون بوسیله متد <code>Compare()</code> پیاده سازی می شود.

کلاس های کلسیون generic و عملکرد آنها در جدول زیر نشان داده می شوند.

کلاس	واسط های پیاده سازی شده	توصیف
<code><List<T></code>	<code><IList<T></code> <code>ICollection<T></code> <code><IEnumerator<T></code>	کلاس <code><generic List<T></code> جانشینی برای کلاس <code>ArrayList</code> است. بطور مشابه کلاس <code><List<T></code> می تواند بطور پویا بزرگ و کوچک شود. علاوه بر پیاده سازی سه واسط، عملیات اضافی همچون مرتب کردن و معکوس کردن اعضای لیست را پشتیبانی می کند.

این کلاس کلکسیون برای ذخیره زوج‌های کلید و مقدار استفاده می‌شود.	<pre> IDictionary<TKey,TValue> ICollection<KeyValuePair<TKey, TValue>> IEnumerable<KeyValuePair<TKey,TValue>>ISerializable IDeserializationCallback </pre>	Dictionary<TKey <, TValue>
این کلاس شبیه Dictionary <Tkey,Tvalue> است، با این تفاوت که در این کلکسیون کلیدها مرتب شده هستند.	<pre> IDictionary<TKey,TValue> ICollection<KeyValuePair<TKey, TValue>> IEnumerable<KeyValuePair<TKey,TValue>> IDeserializationCallback </pre>	SortedList<TKey <, TValue>
LinkedList<T> یک لیست دو پیوندی است، که یکی از اعضای آن به قبلی و دیگری به بعدی ارجاع می‌کند.	<pre> ICollection<T> IEnumerable<T> ISerializable IDeserializationCallback </pre>	<LinkedList<T>
Queue<T> یک کلکسیون FIFO است. عنصر جدید به انتهای صف اضافه می‌شود و عنصر حذفی از سر صف حذف می‌گردد. متد Enqueue() یک عنصر به ته صف اضافه می‌کند و متد Dequeue() عنصری را از سر صف حذف می‌کند. می‌توان با متد Peek() بدون حذف یک عنصر از عنصر سر صف باخبر شد.	<pre> ICollection<T> IEnumerable<T> </pre>	<Queue<T>
Stack<T> یک کلکسیون LIFO است. عنصری که اخیراً اضافه شده، اولین عنصری است که می‌توان به آن دسترسی پیدا کرد. عناصر جدید به بالای پشته اضافه می‌شوند و از بالای پشته نیز حذف می‌شوند. متد Push() یک عنصر جدید به بالای پشته می‌گذارد و متد Pop() عنصر بالای پشته را بر می‌دارد. متد Peek() بدون حذف عنصر بالای پشته آن را بدست می‌آورد.	<pre> ICollection<T> IEnumerable<T> </pre>	<Stack<T>

۱۱-۶-۲-مثالی از کاربرد واسط **IEnumerable**

برای اینکه یک کلاس دستور foreach را پشتیبانی کند، لازم است واسط **IEnumerable<T>** را پیاده‌سازی کند. واسط **IEnumerable<T>** فقط یک متد بنام **GetEnumerator()** دارد که یک پیاده‌سازی از **IEnumerator<T>** برمی‌گرداند. #C با استفاده از کلمه‌ی کلیدی **yield** در ایجاد شمارنده به توسعه‌دهنده‌ی کلاس یاری می‌رساند(همانطور که در مثال ۱۱-۳ مشاهده می‌کنید).

مثال ۱۱-۳

```

using System;
using System.Collections.Generic; // for the generic classes
namespace Enumerable

```

```

{
    public class ListBoxTest : IEnumerable<String>
    {
        private string[] strings;
        private int ctr = 0;
        // Enumerable classes can return an enumerator
        public IEnumerator<string> GetEnumerator( )
        {
            foreach ( string s in strings )
            {
                yield return s;
            }
        }
        // required to fulfill IEnumerable
        System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator( )
        {
            throw new NotImplementedException( );
        }
        // initialize the list box with strings
        public ListBoxTest( params string[] initialStrings )
        {
            // allocate space for the strings
            strings = new String[100];
            // copy the strings passed in to the constructor
            foreach ( string s in initialStrings )
            {
                strings[ctr++] = s;
            }
        }
        // add a single string to the end of the list box
        public void Add( string theString )
        {
            strings[ctr] = theString;
            ctr++;
        }
        // allow array-like access
        public string this[int index]
        {
            get
            {
                if ( index < 0 || index >= strings.Length )
                {
                    // handle bad index
                }
                return strings[index];
            }
            set
            {
                strings[index] = value;
            }
        }
        // publish how many strings you hold
        public int GetNumEntries( )
        {
            return ctr;
        }
    }
}
public class Tester
{
    static void Main( )
    {
        // create a new list box and initialize
        ListBoxTest lbt =
            new ListBoxTest( "Hello", "World" );
        // add a few strings
    }
}

```

```

lbt.Add( "Proust" );
lbt.Add( "Faulkner" );
lbt.Add( "Mann" );
lbt.Add( "Hugo" );
// test the access
string subst = "Universe";
lbt[\] = subst;
// access all the strings
foreach ( string s in lbt )
{
    if ( s == null )
    {
        break;
    }
    Console.WriteLine( "Value: {0}", s );
}
}
}

```

خروجی آن شبیه زیر است:

```

Value: Hello
Value: Universe
Value: Proust
Value: Faulkner
Value: Mann
Value: Hugo

```

برنامه در Main() آغاز می‌شود. یک `ListBoxTest` جدید ایجاد کرده و دو رشته به سازنده‌ی آن ارسال می‌کند. زمانی که شی ایجاد می‌شود، یک آرایه بنام `Strings` با ۲۵۶ درایه ایجاد می‌شود. حلقه‌ی `foreach` بطور اتوماتیک واسط `IEnumerable<T>` را بکار می‌برد و `GetEnumerator()` را احضار می‌کند.

برای برگرداندن یک `IEnumerator` از نوع `string` متد `GetEnumerator()` بصورت زیر پیاده سازی می‌شود.

```

public IEnumerator<string> GetEnumerator( )
{
    foreach ( string s in strings )
    {
        yield return s;
    }
}

```

کلمه‌ی کلیدی جدیدی بنام `yield` صریحا برای برگرداندن یک مقدار با شی شمارنده استفاده می‌شود. توجه کنید که پیاده‌سازی ما یک پیاده‌سازی از متد غیر `generic GetEnumerator()` را در بردارد. آن در تعریف `IEnumerable` `generic` لازم است و معمولا برای رها کردن یک استثناء تعریف می‌شود. چون ما انتظار نداریم آن را فراخوانی کنیم.

```

// required to fulfill IEnumerable
System.Collections.IEnumerator
System.Collections.IEnumerable.GetEnumerator( )
{
    throw new NotImplementedException( );
}

```

۱۱-۷-خلاصه

- یک کلاس `generic` شبیه یک کلاس عادی تعریف می‌شود، ولی بعد از نام کلاس نوع `generic` مشخص می‌گردد
- `generic`ها به پیشرفت مهمی در کارایی و کیفیت کد منجر می‌شوند، چون می‌توانید بدون نیاز به پیاده‌سازی مجدد یک الگوریتم آن را با ساختارهای داده‌ای دیگر بکار ببرید.

- بطور سطحی generic های C# شبیه الگوهای ++C هستند. اما اختلافات مهمی در روش پیاده سازی و پشتیبانی توسط کامپایلر دارند.
- بوسیله generic های C# کامپایلر کد generic را بطور مستقل از نوع داده ای انتخاب شده توسط سرویس گیرنده، به IL ترجمه می کند.
- محدودیتها بخشی از فراداده ای نوع داده ای generic هستند. بنابراین سرویس گیرنده می تواند مزایای آنها را به خوبی استفاده کند.
- می توانید یک پارامتر نوع داده ای generic را طوری محدود کنید که یک نوع داده ای مقداری باشد.
- در فضای نامی System.Collections.Generic تعداد زیادی کلاس کلکسیون و واسط generic تعریف می شوند.
- C# با استفاده از کلمه ی کلیدی yield در ایجاد شمارنده به توسعه دهنده ی کلاس یاری می رساند.

کاربرد اندیس گذارها

بعد از این فصل خواهید توانست:

- اعلان‌های آرایه‌گون را با استفاده از اندیس گذارها کپسوله کنید.
- دسترسی خواندن اندیس گذارها را بوسیله اعلان `get` کنترل کنید.
- دسترسی نوشتن به اندیس گذارها را بوسیله اعلان `set` کنترل کنید.
- واسطه‌هایی که اندیس گذارها را اعلان می‌کنند، ایجاد کنید.
- اندیس گذارها را در ساختارها و کلاس‌هایی که از واسطه‌ها ارث‌بری می‌کنند، پیاده‌سازی کنید.

۱۲-۱- اندیس گذار

همانند خصوصیت که یک فیلد هوشمند است، اندیس گذار یک آرایه‌ی هوشمند است. نحوه‌ی کاربرد یک اندیس گذار دقیقاً شبیه کاربرد یک آرایه هست. ابتدا یک مثال را بدون کمک از اندیس گذارها بررسی می‌کنیم. سپس همان مساله با یک راه حل بهتر به کمک اندیس گذارها بررسی می‌گردد.

۱۲-۱-۱- مثال بدون کاربرد اندیس گذار

به طور معمول یک نوع داده صحیح را برای نگه داشتن یک مقدار صحیح به کار می‌برید. یک نوع داده صحیح مقدارش را به صورت یک دنباله‌ی ۳۲ بیتی ذخیره می‌کند، که هر بیت می‌تواند صفر یا یک باشد. در بیشتر مواقع به این نمایش داخلی دقت نمی‌کنید. یک نوع داده صحیح را فقط به عنوان یک سطل برای نگه داشتن یک مقدار صحیح به کار می‌برید. با این وجود، بعضی مواقع نوع داده صحیح برای اهداف دیگر استفاده می‌شود. بعضی از برنامه‌ها تک به تک بیت‌های یک داده صحیح را دستکاری می‌کنند. به عبارت دیگر احتمال دارد یک برنامه عدد صحیح را برای نگهداری ۳۲ بیت به کار برد، نه به این دلیل که آن می‌تواند یک عدد صحیح را نشان دهد.

توجه: برخی از برنامه‌های قدیمی انواع داده صحیح را برای ذخیره حافظه به کار می‌برند. یک مقدار صحیح منفرد ۳۲ بیت نگه می‌دارد، که هر کدام می‌توانند ۰ یا ۱ باشند. در برخی موارد ۱ را برای مشخص کردن مقدار `true`، و ۰ را برای مشخص کردن مقدار `false` تخصیص می‌دهند و نوع داده صحیح را به صورت مجموعه‌ای از مقادیر بولین در نظر می‌گیرند. به عنوان مثال، عبارت زیر عملگرهای بیتی `>>` و `&` را برای مشخص کردن ۱ یا ۰ بودن بیت اندیس ۶ در متغیر `bits` بکار می‌رود.

```
(bits & (1 << 6)) != 0
```

اگر بیت اندیس ۶ صفر باشد، مقدار این عبارت با false ارزیابی می‌شود و اگر یک باشد، مقدار آن با true ارزیابی می‌شود. این یک عبارت پیچیده است، اما در مقایسه با عبارت زیر که مقدار بیت اندیس ۶ را ۰ قرار می‌دهد، ساده است.

```
bits &= ~(1 << 6)
```

عبارت زیر مقدار بیت اندیس ۶ را ۱ قرار می‌دهد.

```
bits |= (1 << 6)
```

مشکل این مثال‌ها این است:

اگرچه آنها کار می‌کنند، اما مشخص و واضح نیست که چرا و چگونه کار می‌کنند. آنها پیچیده هستند و راه حل آنها بسیار سطح پائین است و حالت انتزاعی مساله را از بین می‌برند.

۱۲-۱-۲- کاربرد اندیس‌گذارها در مثال قبلی

در اینجا می‌خواهیم یک نوع داده صحیح را نه به صورت یک مقدار صحیح بلکه به صورت یک آرایه‌ی ۳۲ بیتی به کار ببریم. بنابراین بهترین راه حل این مسئله کاربرد int به صورت یک آرایه‌ی ۳۲ بیتی است. به عبارت دیگر اگر bits یک نوع صحیح باشد، برای دسترسی به بیت اندیس ۶ خواهیم نوشت:

```
bits[6]
```

و به عنوان مثال برای نوشتن مقدار true در بیت اندیس ۶ آن خواهیم نوشت

```
bits[6] = true
```

مناسفانه نمی‌توانیم علامت گروه‌ها را روی یک عدد صحیح استفاده کنیم. آن فقط بر روی یک آرایه یا روی یک نوع داده شبیه آرایه کار می‌کند. اندیس‌گذار یک نمونه از آن است. بنابراین راه حل مسئله این است که یک نوع داده جدید شبیه آرایه ایجاد کنیم، تا شبیه یک آرایه از متغیرهای bool عمل کند. اما بوسیله یک نوع داده صحیح پیاده‌سازی می‌شود. این نوع داده جدید را IntBits نامگذاری کنید. IntBits یک مقدار صحیح را در بر خواهد گرفت، اما هدف ما استفاده از IntBits به صورت یک آرایه از متغیرهای bool خواهد بود.

نکته

چون IntBit کوچک و سبک وزن است ترجیح می‌دهیم آن را به جای کلاس یک ساختار تعریف کنیم.

```
struct IntBits
{
    public IntBits(int initialBitValue)
    {
        bits = initialBitValue;
    }
    // indexer to be written here
    private int bits;
}
```

برای تعریف یک اندیس‌گذار، یک علامتی بکار می‌برید که به یک خصوصیت آرایه‌ای شباهت دارد. اندیس‌گذار IntBits شبیه زیر است:

```
struct IntBits
{
    ...
    public bool this [ int index ]
    {
        get
        {
            return (bits & (1 << index)) != 0;
        }
        set
        {
            if (value) // Turn the bit on if value is true, otherwise turn it off
```

```
bits |= (1 << index);
else
bits &= ~(1 << index);
}
}
...
}
```

به موارد زیر توجه کنید:

- اندیس‌گذار یک متد نیست و به جای پرانتز از کروشه استفاده می‌کند.
- یک اندیس‌گذار همواره یک آرگومان منفرد می‌گیرد که مابین [] قرار می‌گیرد. این آرگومان عنصر مورد نظر جهت دستیابی را مشخص می‌کند.
- همه‌ی اندیس‌گذارها کلمه کلیدی `this` را بجای نام متد به کار می‌برند. یک کلاس یا ساختار فقط می‌تواند یک اندیس‌گذار تعریف کند و آن همیشه با `this` نام‌گذاری می‌شود. البته می‌توان از آن اندیس‌گذار `overload` های مختلفی تعریف کرد.
- اندیس‌گذارها معاون‌های `set` و `get` را شبیه خصوصیات در بر دارند. معاون‌های `set` و `get` عبارتهای بیتی مثال قبلی را پیاده‌سازی کرده‌اند.
- آرگومان مشخص شده در اعلان اندیس‌گذارها با مقدار اندیس مشخص شده در هنگام فراخوانی پر می‌شود. معاون‌های `set` و `get` می‌توانند این آرگومان را برای تعیین عنصر مورد نظر بخوانند.

توجه: برای جلوگیری از وقوع هر استثنای در داخل کد اندیس‌گذار، یک کنترل‌کننده‌ی محدوده برای مقدار اندیس در نظر بگیرید.

می‌توانیم بعد از اعلان اندیس‌گذار، یک متغیر از نوع `IntBits` را به جای یک عدد صحیح بکار ببریم و علامت [] را در صورت نیاز به آن اعمال کنیم.

```
int adapted = ۱۲;
IntBits bits = new IntBits(adapted);
bool peek = bits[۱]; // retrieve bool at index ۱
bits[۰] = true; // set the bit at index ۰ to true
bits[۳۱] = false; // set the bit at index ۳۱ to false
```

توجه: معاون‌های `set` و `get` در اندیس‌گذارها و خصوصیت‌ها بطور مشابه بکار می‌روند. اندیس‌گذارها شبیه یک خصوصیت چند مقداری هستند. اگرچه تعریف خصوصیات `static` مجاز است، اما کاربرد اندیس‌گذار `static` نامعقول است.

کاربرد اندیس‌گذارها در عبارات خواندن/نوشتن ترکیبی^۱ امکان‌پذیر است. در این مورد معاون‌های `set` و `get` استفاده می‌شوند. به عنوان مثال دستور زیر را ملاحظه فرمائید

```
bits[۱] ^= true;
```

این دستور بطور اتوماتیک بصورت زیر ترجمه می‌شود

```
bits[۱] = bits[۱] ^ true;
```

توجه: اعلان اندیس‌گذاری که فقط معاون `get` یا معاون `set` دارد، مجاز است.

۱۲-۲- مقایسه آرایه‌ها و اندیس‌گذارها

زمانی که یک اندیس‌گذار به کار می‌برید، از نظر نحوی شبیه آرایه است. با این وجود، تفاوت‌های مهمی مابین اندیس‌گذارها و آرایه‌ها وجود دارد.

^۱ Combined

- اندیس‌گذارها می‌توانند اندیس‌های غیر عددی بکار برند، در حالیکه آرایه‌ها فقط می‌توانند اندیس‌های صحیح بکار برند.

```
public int this [ string name ] { ... } // okay
```

نکته: کلاس Hashtable که از زوج مرتب‌های (مقدار / کلید) تشکیل شده است، برای اضافه کردن یک مقدار جدید، به جای استفاده از متد Add می‌تواند از روشی شبیه اندیس‌گذار استفاده کند.

```
Hashtable ages = new Hashtable();
ages.Add("John", ٤١);
```

می‌توانید کد زیر را بنویسید:

```
Hashtable ages = new Hashtable();
ages["John"] = ٤١;
```

- اندیس‌گذارها شبیه متدها می‌توانند OverLoad شوند، ولی آرایه‌ها نمی‌توانند.

```
public Name this [ PhoneNumber number ] { ... }
public PhoneNumber this [ Name name ] { ... }
```

- اندیس‌گذارها نمی‌توانند همانند پارامترهای ref یا out بکار روند، ولی عناصر آرایه می‌توانند.

```
IntBits bits; // bits contains an indexer
Method(ref bits[١]); // compile-time error
```

١٢-٣-خصوصیات آرایه‌ها و اندیس‌گذارها

برگرداندن یک آرایه برای یک خصوصیت ممکن است، اما به خاطر داشته باشید که آرایه‌ها از نوع داده‌های ارجاعی هستند. پس نمایش داده یک آرایه بصورت یک خصوصیت می‌تواند رونویسی حجم زیادی از داده‌ها را بطور تصادفی ممکن سازد. ساختار زیر یک خصوصیت آرایه‌ای بنام Data را نمایش می‌دهد.

```
struct Wrapper
{
    int[] data;
    ...
    public int[] Data
    {
        get { return this.data; }
        set { this.data = value; }
    }
}
```

کد زیر نحوه‌ی استفاده از این خصوصیت را نشان می‌دهد

```
Wrapper wrap = new Wrapper();
...
int[] myData = wrap.Data;
myData[٠]++;
myData[١]++;
```

ولی برای دسترسی به تک تک عناصر آرایه می‌توانیم اندیس‌گذار را بصورت زیر ایجاد کنیم:

```
struct Wrapper
{
    int[] data;
    ...
    public int this [int i]
    {
        get
        {
```

```

return this.data[i];
}
set
{
    this.data[i] = value;
}
}
}

```

کد زیر اندیس‌گذار را در روشی مشابه یک خصوصیت بکار می‌برد.

```

Wrapper wrap = new Wrapper();
...
int[] myData = new int[۲];
myData[۰] = wrap[۰];
myData[۱] = wrap[۱];
myData[۰]++;
myData[۱]++;

```

در این حالت تغییر مقدار آرایه myData هیچ تاثیری روی آرایه اصلی در شی Wrapper ندارد. اگر بخواهید واقعا مقدار داده در شی wrapper را تغییر دهید، دستوری شبیه زیر بنویسید:

```
wrap[۰]++;
```

۱۲-۴-اندیس‌گذارها در واسط‌ها

در یک واسط فقط می‌توانید اندیس‌گذارها را اعلان کنید. برای انجام این کار کلمات کلیدی `set`, `get` را مشخص کنید. اما بدنه معاون‌های `set`, `get` را با ; جایگزین کنید. هر کلاس یا ساختاری که واسط را پیاده‌سازی می‌کند، باید اندیس‌گذارهای اعلان شده در واسط را نیز پیاده‌سازی کند. به عنوان مثال:

```

interface IRawInt
{
    bool this [ int index ] { get; set; }
}
struct RawInt : IRawInt
{
    ...
    public bool this [ int index ]
    {
        get { ... }
        set { ... }
    }
    ...
}

```

اگر یک واسط، اندیس‌گذاری را در یک کلاس پیاده‌سازی می‌کند، می‌توانید پیاده‌سازی‌های اندیس‌گذار را بصورت مجازی اعلان کنید. این عمل به کلاس‌های مشتق شده اجازه می‌دهد، معاون‌های `set`, `get` را `Override` کنند. به عنوان مثال:

```

class RawInt : IRawInt
{
    ...
    public virtual bool this [ int index ]
    {
        get { ... }
        set { ... }
    }
    ...
}

```

همچنین می‌توانید یک اندیس‌گذار را بوسیله گرامر پیاده‌سازی صریح، پیاده‌سازی کنید. برای مثال:

```

struct RawInt : IRawInt
{
    ...
    bool IRawInt.this [ int index ]

```

```
{  
get { ... }  
set { ... }  
}  
...  
}
```

۱۲-۵-خلاصه

- ، اندیس‌گذار یک آرایه‌ی هوشمند است. نحوه‌ی کاربرد یک اندیس‌گذار دقیقاً شبیه کاربرد یک آرایه هست.
- یک نوع داده صحیح مقدارش را به صورت یک دنباله‌ی ۳۲ بیتی ذخیره می‌کند، که هر بیت می‌تواند صفر یا یک باشد.
- . بعضی از برنامه‌ها تک به تک بیت‌های یک داده صحیح را دستکاری می‌کنند.
- نمی‌توانیم علامت‌کروشه‌ها را روی یک عدد صحیح استفاده کنیم. آن فقط بر روی یک آرایه یا روی یک نوع داده شبیه آرایه کار می‌کند. اندیس‌گذار یک نمونه از آن است.
- اندیس‌گذار یک متد نیست و به جای پرانتز از کروشه استفاده می‌کند.
- یک اندیس‌گذار همواره یک آرگومان منفرد می‌گیرد که مابین [] قرار می‌گیرد.
- کاربرد اندیس‌گذارها در عبارات خواندن/نوشتن ترکیبی^۱ امکان‌پذیر است.
- اندیس‌گذارها می‌توانند اندیس‌های غیر عددی بکار برند، در حالیکه آرایه‌ها فقط می‌توانند اندیس‌های صحیح بکار برند.
- در یک واسط فقط می‌توانید اندیس‌گذارها را اعلان کنید. برای انجام این کار کلمات کلیدی `set`, `get` را مشخص کنید. اما بدنه معاون‌های `set`, `get` را با ؛ جایگزین کنید.

^۱ Combined

Overload کردن عملگرها

آنچه که در این فصل یاد خواهید گرفت:

- مفهوم overload کردن عملگرها
- استفاده از عملگرهای معمول زبان C# در انواع داده‌ای تعریف شده توسط کاربر
- نحوه‌ی استفاده از کلمه‌ی کلیدی operator
- Overload کردن عملگرهای دوتایی و یکتایی
- Overload کردن عملگرهای تبدیل صریح و ضمنی

یکی از اهداف طراحی در C# این است که کلاس‌های تعریف شده توسط کاربر می‌توانند عملکرد انواع داده‌ی داخلی C# را داشته باشند. برای مثال، فرض کنید که یک نوع داده برای نمایش کسرها تعریف کرده‌اید. این کلاس می‌تواند همه توانایی‌های انواع داده‌ی داخلی را داشته باشد، یعنی شما قادر هستید عملیات ریاضی را روی نمونه‌هایی از کسرها انجام دهید و کسرها را به انواع داده‌ی داخلی همچون `int` و ... تبدیل کنید. می‌توانید متدهایی برای هر عمل پیاده‌سازی کنید و آنها را با نوشتن دستوراتی بصورت زیر فراخوانی کنید.

```
Fraction theSum = firstFraction.Add(secondFraction);
```

اگر چه این دستور کار می‌کند، ولی جالب نیست و شبیه استفاده‌ی انواع داده‌ی داخلی نیست. بهتر است بصورت زیر نوشته شود.

```
Fraction theSum = firstFraction + secondFraction;
```

این دستور با نحوه‌ی کارکرد انواع داده‌ی داخلی سازگار و مشهود است و همانند متغیرهای `int` جمع می‌شوند.

در این فصل تکنیک‌هایی برای اضافه کردن عملگرها به انواع داده‌ی تعریف شده توسط کاربر یاد خواهید گرفت. همچنین نحوه‌ی اضافه کردن عملگرهای تبدیل صریح و ضمنی را یاد خواهید گرفت.

۱۳-۱- کاربرد کلمه‌ی کلیدی operator

در C# می‌توانید عملگرها را با متدهای ایستایی که مقادیر بازگشتی آنها نتیجه‌ی یک عمل را نشان می‌دهند پیاده‌سازی کنید، بطوریکه عملوندهای آنها همان پارامترها هستند. زمانی که یک عملگر برای کلاسی ایجاد می‌کنید، گفته می‌شود آن عملگر overload شده است. پس برای overload کردن عملگر + بصورت زیر خواهید نوشت:

```
public static Fraction operator+(Fraction lhs, Fraction rhs)
```

پارامتر اول عبارت سمت چپ عملگر می‌باشد و پارامتر دوم عبارت سمت راست عملگر می‌باشد.

گرامر C# برای overload کردن یک عملگر، نوشتن کلمه‌ی کلیدی operator و به دنبال آن نام عملگر است. کلمه‌ی کلیدی operator یک معرف متد است. پس برای overload کردن عملگر +، operator + را بنویسید.

زمانیکه عبارت زیر را می‌نویسید:

```
Fraction theSum = firstFraction + secondFraction;
```

عملگر overload شده‌ی + احضار می‌شود که firstFraction پارامتر اول آن و secondFraction پارامتر دوم آن خواهد شد. زمانی که کامپایلر عبارت زیر را می‌بیند:

```
firstFraction + secondFraction
```

آن را به عبارت زیر ترجمه می‌کند.

```
Fraction.operator+(firstFraction, secondFraction)
```

در نتیجه، یک Fraction جدید برگردانده می‌شود که در این نمونه مقدار آن به شیء Fraction بنام theSum انتساب داده می‌شود.

توجه: ایجاد عملگرهای غیر ایستا امکان‌پذیر نیست و عملگرهای دوتایی باید دو عملوند داشته باشند.

۱۳-۲- پشتیبانی دیگر زبان‌های NET.

C# در CLS توانایی overload کردن عملگرها را برای کلاس‌های شما فراهم می‌کند، اما بعضی از زبان‌های NET (همچون VB.NET) overload کردن عملگرها را پشتیبانی نمی‌کنند. پس مطمئن شوید کلاس‌های شما روش دیگری برای انجام همین اعمال را پشتیبانی می‌کنند. پس اگر عملگر جمع + را overload کردید، بهتر است متد Add() را برای انجام همان کار نیز اضافه کنید.

۱۳-۳- ایجاد عملگرهای مفید

Overload کردن عملگرها، کد شما را مشهودتر می‌سازد و کد شما بیشتر شبیه انواع داده‌ی داخلی C# عمل می‌کند. ولی اگر شیوه‌ی معمول کاربرد عملگرها را دستکاری کنید، کد شما پیچیده و غیرقابل مدیریت خواهد شد. از روش‌های جدید و مزاجی در کاربرد عملگرها دوری کنید.

برای مثال، اگرچه ممکن است عملگر ++ را برای افزایش حقوق ماهیانه‌ی کارمند به کلاس Employee اضافه کنید، ولی این عملگر برای سرویس‌گیرندگان این کلاس سردرگمی عجیبی ایجاد می‌کند. بهتر است زمانی از overload کردن عملگرها استفاده کنید که مفهوم آن کاملاً واضح باشد.

۱۳-۴- عملگرهای دوتایی منطقی

Overload کردن عملگر تساوی (==) برای تست مساوی بودن دو شیء کاملاً معمول است. C# اصرار دارد در صورت Overload کردن عملگر تساوی، بایستی عملگر نامساوی (!=) نیز overload شود. بطور مشابه، عملگرهای دوتایی (<)، بزرگتر از (>)، کوچکتر مساوی (<=) و بزرگتر مساوی (>=) overload می‌شوند.

۱۳-۵- عملگر تساوی

اگر عملگر تساوی را overload کنید، توصیه می‌شود متد مجازی Equals() فراهم شده توسط object را نیز override کنید و عملکرد آن را شبیه عملگر تساوی قرار دهید. این عمل چندریختی و سازگاری با زبان‌های دیگر NET را برای کلاس شما به ارمغان می‌آورد. کلاس‌های FCL عملگرهای overload شده را بکار نخواهند برد. اما انتظار دارید کلاس‌های شما متدهای اصلی را پیاده‌سازی کنند. کلاس object متد Equals() را با نشانه‌ی زیر پیاده‌سازی می‌کند.

```
public virtual bool Equals(object o)
```

با override کردن این متد، کلاس Fraction شما با اشیاء دیگر بصورت چندریختی عمل می‌کند. در داخل بدنه‌ی Equals()، باید مطمئن شویم دو شیء Fraction باهم مقایسه می‌شوند.

```
public override bool Equals(object o)
{
    if (! (o is Fraction) )
    {
        return false;
    }
    return this == (Fraction) o;
}
```

عملگر is برای بررسی نوع یک شیء در زمان اجرا استفاده می‌شود. پس عبارت o is Fraction در صورتی که o با یک نمونه از Fraction سازگار باشد به true، در غیر اینصورت به false ارزیابی می‌شود.

کامپایلر انتظار دارد متد GetHashCode() را نیز override کنید.

۱۳-۶- عملگرهای تبدیل

C# بطور ضمنی int را به long تبدیل می‌کند و به شما اجازه می‌دهد بطور صریح long را به int تبدیل کنید. تبدیل از int به long ضمنی است و آن ایمن است، چون هر مقدار داده‌ی int را می‌توان در مقدار داده‌ای از نوع long قرار داد. عمل تبدیل بر عکس (از long به int) باید صریح باشد، چون ممکن است مقداری از اطلاعات از دست برود.

```
int myInt = ۵;
myLong;
myLong = myInt; // implicit
myInt = (int) myLong; // explicit
```

باید همین عمل را برای کسرها نیز انجام دهید. تبدیل ضمنی یک عدد صحیح به کسر امکان پذیر است، چون هر عدد صحیح معادل کسر همان عدد روی یک است (۱۵==۱/۱۵). ولی تبدیل کسر به عدد صحیح باید بصورت صریح باشد. پس ممکن است مقدار کسری ۴/۹ به مقدار صحیح ۲ تبدیل گردد.

در هنگام پیاده‌سازی تبدیل‌های خود، در صورتی که می‌دانید عمل تبدیل هیچ داده‌ای را از بین نمی‌برد کلمه‌ی کلیدی implicit را بکار برید و در غیر اینصورت کلمه کلیدی explicit را بکار برید.

مثال ۱۳-۱ نحوه‌ی پیاده‌سازی تبدیل‌های صریح و ضمنی و بعضی از عملگرهای کلاس Fraction را نشان می‌دهد. زمانی که این مثال را کامپایل می‌کنید، تعدادی هشدار دریافت خواهید کرد، چون GetHashCode() پیاده‌سازی نشده است.

مثال ۱-۱۳

```

public class Fraction
{
    private int numerator;
    private int denominator;
    public Fraction(int numerator, int denominator)
    {
        Console.WriteLine("In Fraction Constructor(int, int)");
        this.numerator=numerator;
        this.denominator=denominator;
    }
    public Fraction(int wholeNumber)
    {
        Console.WriteLine("In Fraction Constructor(int)");
        numerator = wholeNumber;
        denominator = ۱;
    }
    public static implicit operator Fraction(int theInt)
    {
        Console.WriteLine("In implicit conversion to Fraction");
        return new Fraction(theInt);
    }
    public static explicit operator int(Fraction theFraction)
    {
        Console.WriteLine("In explicit conversion to int");
        return theFraction.numerator /theFraction.denominator;
    }
    public static bool operator==(Fraction lhs, Fraction rhs)
    {
        Console.WriteLine("In operator ==");
        if (lhs.denominator == rhs.denominator &&lhs.numerator == rhs.numerator)
        {
            return true;
        }
        // code here to handle unlike fractions
        return false;
    }
    public static bool operator !=(Fraction lhs, Fraction rhs)
    {
        Console.WriteLine("In operator !=");
        return !(lhs==rhs);
    }
    public override bool Equals(object o)
    {
        Console.WriteLine("In method Equals");
        if (! (o is Fraction) )
        {
            return false;
        }
        return this == (Fraction) o;
    }
    public static Fraction operator+(Fraction lhs, Fraction rhs)
    {
        Console.WriteLine("In operator+");
        if (lhs.denominator == rhs.denominator)
        {
            return new Fraction(lhs.numerator+rhs.numerator, lhs.denominator);
        }
        // simplistic solution for unlike fractions
        // ۱/۲ + ۲/۴ == (۱*۲) + (۲*۲) / (۲*۴) == ۱۰/۸
        int firstProduct = lhs.numerator * rhs.denominator;
        int secondProduct = rhs.numerator * lhs.denominator;
        return new Fraction(

```

```

firstProduct + secondProduct,
lhs.denominator * rhs.denominator
);
}
public override string ToString( )
{
String s = numerator.ToString( ) + "/" +denominator.ToString( );
return s;
}
}
public class Tester
{
static void Main( )
{
Fraction f1 = new Fraction(۳,۴);
Console.WriteLine("f1: {۰}", f1.ToString( ));
Fraction f2 = new Fraction(۲,۴);
Console.WriteLine("f2: {۰}", f2.ToString( ));
Fraction f3 = f1 + f2;
Console.WriteLine("f1 + f2 = f3: {۰}", f3.ToString( ));
Fraction f4 = f3 + ۵;
Console.WriteLine("f3 + ۵ = f4: {۰}", f4.ToString( ));
Fraction f5 = new Fraction(۲,۴);
if (f5 == f2)
{
Console.WriteLine("F5: {۰} == F2: {۱}",f5.ToString( ),f2.ToString( ));
}
}
}

```

کلاس Fraction با دو سازنده شروع می‌شود: یکی صورت و مخرج را می‌گیرد و دیگری کل عدد را می‌گیرد. سازنده‌ها با اعلان دو عملگر تبدیل دنبال می‌شوند. عملگر تبدیل اولی یک عدد صحیح را به یک کسر تبدیل می‌کند:

```

public static implicit operator Fraction(int theInt)
{
return new Fraction(theInt);
}

```

این تبدیل با implicit علامت‌گذاری می‌شود، چون هر عدد صحیح می‌تواند به یک Fraction تبدیل شود که صورت کسر همان عدد صحیح بوده و مخرج آن عدد یک می‌باشد. این مسئولیت را به سازنده‌ای که یک عدد صحیح می‌گیرد واگذار کنید.

عملگر تبدیل دوم، تبدیل صریح Fraction‌ها به عدد صحیح است:

```

public static explicit operator int(Fraction theFraction)
{
return theFraction.numerator /theFraction.denominator;
}

```

چون این مثال عمل تقسیم صحیح انجام می‌دهد، مقدار صحیح بر می‌گرداند. پس اگر ۱۶/۱۵ باشد، مقدار صفر بر می‌گرداند. یک عملگر تبدیل پیچیده ممکن است عمل گردکردن را انجام دهد.

عملگرهای تبدیل با عملگر تساوی (==) و عملگر نامساوی (!=) دنبال می‌شوند. بخاطر دارید که اگر یکی از این دو عملگر را پیاده‌سازی کردید، باید دیگری را نیز پیاده‌سازی کنید.

تساوی مقدار یک Fraction بدین صورت است که صورت‌ها و مخرج‌ها باهم مطابقت داشته باشند. به عنوان مثال: ۴/۳ و ۸/۶ مساوی نیستند. البته ممکن است در یک پیاده‌سازی این کسرهای ساده شده و مساوی به حساب آیند. کلاس

Fraction همه عملگرهای ریاضی (جمع، تفریق، ضرب و تقسیم) را پیاده‌سازی نمی‌کند. برای فهم ساده‌ی مطلب، فقط عمل جمع بصورت بسیار ساده پیاده‌سازی می‌شود. در صورت مساوی بودن مخرج‌ها، عمل جمع بصورت زیر انجام می‌شود:

```
public static Fraction operator+(Fraction lhs, Fraction rhs)
{
    if (lhs.denominator == rhs.denominator)
    {
        return new Fraction(lhs.numerator+rhs.numerator, lhs.denominator);
    }
}
```

اگر مخرج‌ها مساوی نباشند، با مخرج مشترک گرفتن (حاصلضرب هر دو مخرج) عمل جمع انجام می‌شود.

```
int firstProduct = lhs.numerator * rhs.denominator;
int secondProduct = rhs.numerator * lhs.denominator;
return new Fraction(firstProduct + secondProduct, lhs.denominator *
                    rhs.denominator);
```

این کد با یک مثال بهتر فهمیده می‌شود. اگر بخواهید $\frac{2}{1}$ و $\frac{4}{3}$ را باهم جمع کنید، صورت کسر اول (۱) در مخرج کسر دوم ضرب شده و در یک متغیر ذخیره می‌شود (۴). می‌توانید صورت کسر دوم (۳) را در مخرج کسر اول (۲) ضرب کرده و در متغیر دیگری ذخیره کنید (۶). می‌توانید مجموع دو متغیر را در صورت جواب قرار داده (۱۰) و حاصلضرب دو مخرج را در مخرج جواب قرار دهید (۸). کسر بدست آمده ($\frac{10}{8}$) جواب صحیحی است.

نهایتاً، متد ToString() را override کنید تا مقدار آن را در قالب رشته‌ای مخرج/صورت برگرداند.

```
public override string ToString()
{
    String s = numerator.ToString() + "/" + denominator.ToString();
    return s;
}
```

حال کلاس Fraction را در دست دارید و برای آزمایش آماده است. با دو کسر ساده $\frac{4}{3}$ و $\frac{4}{2}$ آن را تست کنید.

```
Fraction f1 = new Fraction(3,4);
Console.WriteLine("f1: {0}", f1.ToString());
Fraction f2 = new Fraction(2,4);
Console.WriteLine("f2: {0}", f2.ToString());
```

خروجی مورد انتظار ما در سازنده‌ها بوسیله دستور WriteLine() چاپ می‌شوند.

```
In Fraction Constructor(int, int)
f1: 3/4
In Fraction Constructor(int, int)
f2: 2/4
```

خط بعدی در متد Main()، عملگر ایستای + را احضار می‌کند. هدف این عملگر جمع دو کسر و برگرداندن مجموع آنها در یک کسر جدید است.

```
Fraction f3 = f1 + f2;
Console.WriteLine("f1 + f2 = f3: {0}", f3.ToString());
```

بررسی خروجی، نحوه‌ی کار operator + را نشان می‌دهد.

```
In operator+
In Fraction Constructor(int, int)
f1 + f2 = f3: 5/4
```

operator + احضار می‌شود و سپس سازنده ی f3 دو مقدار صحیح صورت و مخرج کسر جدید را می‌گیرد.

آزمایش بعدی در Main() یک عدد int را به یک Fraction بنام f3 اضافه می‌کند و مقدار نتیجه را به یک Fraction جدید بنام f4 انتساب می‌دهد.

```
Fraction f4 = f3 + 5;
```

```
Console.WriteLine("f۳ + ۵: {۰}", f۴.ToString());
```

خروجی این قطعه کد، مراحل تبدیل‌های مختلف را نشان می‌دهد.

```
In implicit conversion to Fraction
In Fraction Constructor(int)
In operator+
In Fraction Constructor(int, int)
f۳ + ۵ = f۴: ۲۵/۴
```

توجه کنید که عملگر تبدیل ضمنی برای تبدیل ۵ به یک Fraction احضار شده است. در دستور برگشت از عملگر تبدیل ضمنی، سازنده Fraction فراخوانی می‌شود و کسر ۵/۱ را ایجاد می‌کند. این کسر جدید به همراه ۳f به عملگر + رد می‌شوند و مجموع آنها به سازنده کسر ۴f رد می‌شود.

در آزمایش نهایی یک کسر جدید (۵f) ایجاد می‌شود. آزمایش روی تساوی آن با ۲f است. اگر مساوی باشند، مقادیر آنها چاپ می‌شود.

```
Fraction f۵ = new Fraction(۲,۴);
if (f۵ == f۲)
{
    Console.WriteLine("F۵: {۰} == F۲: {۱}", f۵.ToString(), f۲.ToString());
}
```

خروجی این قطعه کد، ایجاد ۵f را نشان می‌دهد. سپس کار عملگر تساوی overload شده را نشان می‌دهد.

```
In Fraction Constructor(int, int)
In operator ==
F۵: ۲/۴ == F۲: ۲/۴
```

۱۳-۷-خلاصه

- زمانی که یک عملگر برای کلاسی ایجاد می‌کنید، گفته می‌شود آن عملگر overload شده است.
- گرامر C# برای overload کردن یک عملگر، نوشتن کلمه‌ی کلیدی operator و به دنبال آن نام عملگر است.
- بعضی از زبان‌های NET (همچون VB.NET) overload کردن عملگرها را پشتیبانی نمی‌کنند. پس اگر عملگر جمع + را overload کردید، بهتر است متد Add() را برای انجام همان کار نیز اضافه کنید.
- Overload کردن عملگرها، کد شما را مشهودتر می‌سازد و کد شما بیشتر شبیه انواع داده‌ی داخلی C# عمل می‌کند.
- اگر عملگر تساوی را overload کنید، توصیه می‌شود متد مجازی Equals() فراهم شده توسط object را نیز override کنید و عملکرد آن را شبیه عملگر تساوی قرار دهید.
- در هنگام پیاده‌سازی تبدیل‌های خود، در صورتی که می‌دانید عمل تبدیل هیچ داده‌ای را از بین نمی‌برد کلمه‌ی کلیدی implicit را بکار ببرید و در غیر این‌صورت کلمه کلیدی explicit را بکار ببرید.

ایجاد برنامه‌های فرم ویندوز

آنچه که در این فصل یاد خواهید گرفت:

- مقدمه: با چند خط کد می‌توانید یک برنامه کاربردی ویندوزی بسازید که اداره کردن رویدادها و فرم‌های فرزند را نشان دهد.
- کاربرد کنترل‌های Form: همه کنترل‌ها از کلاس پایه Control ارث‌بری می‌کنند. اعضای این کلاس یک روش یکتا برای تغییر موقعیت، اندازه و ظاهر کنترل فراهم می‌کنند.
- کلاس Form: کلاس Form خصوصیات سفارشی دارد که ظاهر آن را تحت تاثیر قرار می‌دهد و آن را قادر می‌سازد تا با منوها کار کند و فرم‌های فرزند را مدیریت کند.
- فرم‌های MDI: یک واسط چند سندی (MDI)^۱، ظرفی برای نگهداشتن فرم‌های فرزند است. از طریق منوی اصلی می‌توان فرم‌ها را سازمان‌دهی، دستیابی و دستکاری کرد.
- کار با منوها: NET دو نوع منو را پشتیبانی می‌کند: منوی فرم اصلی^۲ و منوی محتوا^۳ که می‌تواند به کنترل‌های منفرد اختصاص داده شود.
- اضافه کردن کمک به فرم: دکمه‌های Help، ToolTip و HTML Help گزینه‌هایی برای اضافه کردن کمک به فرم هستند.
- وراثت فرم: وراثت بصری ایجاد سریع یک فرم را ممکن می‌سازد.

این فصل فرض می‌کند توسعه‌دهندگان مسئول ایجاد برنامه‌های GUI برای میزکار هستند (برخلاف برنامه‌های تحت اجرا روی سرور وب یا دستگاه موبایل). این تمایز بسیار مهم است، چون NET کتابخانه‌های کلاس مجزایی را برای هر نوع برنامه فراهم کرده و آنها را در فضاهای نامی متمایز گروه‌بندی می‌کند.

System.Windows.Forms

فرم‌های ویندوز

^۱ Multiple Document Interface

^۲ MainMenu

^۳ Context Menu

System.Web.UI.WebControls

فرم های وب

System.Web.UIMobileControls فرم‌های موبایل مخصوص دستگاه‌های جیبی و دستی

اگرچه این فصل روی فرم‌های ویندوز تمرکز دارد. مهم است بدانید که برنامه‌های مدرن مجبور هستند محیط‌های چندگانه را پشتیبانی کنند. NET. سعی می‌کند یک احساس و نظر یکنواختی برای هر نوع برنامه‌ی توسعه یافته ارائه دهد.

معمولاً توسعه‌دهندگان برای توسعه برنامه‌های GUI به IDE (همچون VS) استناد می‌کنند. به راحتی می‌توان روی این حقیقت مسلط شد که فرم کلاسی است که از کلاس‌های دیگر ارث‌بری می‌کند و خصوصیات و متدهای خود را دارد. این فصل برای فراهم کردن یک فهم درست از فرم‌ها در کنار VS.NET، اعضای کلاس، نحوه پیاده‌سازی آنها و تأثیر روی رفتار فرم را بررسی می‌کند. مواردی همچون نمایش فرم‌ها، تغییر اندازه آنها، قابل لغزیدن^۱ یا شفاف کردن^۲ آنها، ایجاد فرم‌های ارث‌بری شده و واکنش آنها به ماوس و صفحه کلید را بررسی می‌کنیم.

این فصل درباره اصول طراحی GUI نیست. اما نحوه اضافه کردن قابلیت‌هایی همچون فایل‌های Help، ترتیب Tab مابین کنترل‌ها و بهبود قابلیت استفاده فرم را ارائه می‌دهد. کنترل‌ها فقط بصورت کلی بحث می‌شوند.

۱۴-۱- برنامه‌نویسی یک فرم ویندوز

اجرای همه برنامه‌های ویندوزی در یک پنجره اصلی طراحی شده آغاز می‌شود. در واقع این پنجره یک شی Form است که از کلاس System.Windows.Forms.Form ارث‌بری می‌کند. پنجره اولی با ارسال یک نمونه از آن کلاس به متد ایستای Application.Run() نمایش داده می‌شود.

چالش یک توسعه‌دهنده ایجاد واسطی است که قانون پایه طراحی را در برداشته باشد. بدین معنی که طرح یک فرم عملیات آن را تا وسیع‌ترین حد ممکن پشتیبانی کند. برای رسیدن به این هدف، باید توسعه‌دهنده خصوصیات، متدها و رویدادهای کلاس Form را همانند کنترل‌های قرار گرفته روی آن بفهمد.

۱۴-۱-۱- ایجاد دستی یک برنامه کاربردی ویندوز

اجازه دهید یک برنامه ویندوزی ساده را با استفاده از ویرایشگر متن و کامپایلر C# از خط فرمان ایجاد کنیم. این برنامه در مثال ۱۴-۱ نشان داده شده است. این برنامه یک پنجره و یک دکمه روی خود دارد که زمان کلیک روی دکمه یک پیام ظاهر می‌گردد. این ساده‌ترین تمرین برای نمایش نحوه ایجاد یک فرم، اضافه کردن یک کنترل به آن، تنظیم یک اداره‌کننده برای اداره رویدادی از کنترل است.

مثال ۱۴-۱

```
using System;
using System.Windows.Forms;
using System.Drawing;
class MyWinApp
{
    static void Main()
    {
        // (۱) Create form and invoke it
        Form mainForm = new SimpleForm();
        Application.Run(mainForm);
    }
}
// User Form derived from base class Form
```

^۱ Scrollable

^۲ Transparent

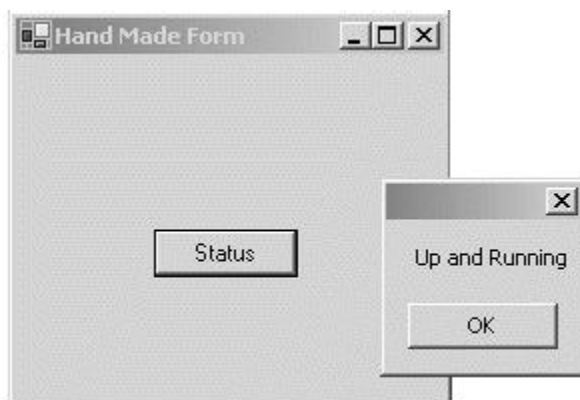
```
class SimpleForm:Form
{
private Button button\;
public SimpleForm()
{
this.Text = "Hand Made Form";
// (۲) Create a button control and set some attributes
button\ = new Button();
button\.Location = new Point(۹۶,۱۱۲);
button\.Size = new Size(۷۲,۲۴);
button\.Text= "Status";
this.Controls.Add(button\);
// (۳) Create delegate to call routine when click occurs
button\.Click += new EventHandler(button\_Click);
}
void button\_Click(object sender, EventArgs e)
{
MessageBox.Show("Up and Running");
}
}
```

این برنامه را در فایل winform.cs ذخیره کرده و با دستور زیر در خط فرمان کامپایل کنید .

```
csc /t:winform.exe /r:System.Windows.Forms.dll winform.cs
```

بعد از کامپایل برنامه، با تایپ کردن winform آن را اجرا کنید. صفحه نمایش مورد نظر در شکل ۱۴-۱ آمده است. خروجی شامل یک فرم پدر و یک فرم فرزند ایجاد شده با کلیک بر روی دکمه است. نکته مهم این است که قبل از بستن فرم فرزند نمی‌توان به فرم پدر دسترسی پیدا کرد. این مثالی از فرم modal (سبک‌دار) است که فقط آخرین فرم باز شده می‌تواند دستیابی شود. حالت دیگر فرم modeless (بدون سبک) است که می‌توانیم همزمان به فرم‌های پدر و فرزند دسترسی داشته باشیم.

شکل ۱۴-۱



از نظر منطقی، کد به سه بخش تقسیم می‌شود.

۱- ایجاد فرم

فرم پدر یک نمونه از کلاس SimpleForm است که از Form ارث‌بری می‌کند و ویژگی‌های سفارشی فرم را تعریف می‌کند. فرم با ارسال یک نمونه به متد Application.Run() احضار می‌شود.

۲- ایجاد کنترل دکمه

با ایجاد یک نمونه از کنترل و اضافه کردن آن روی فرم، یک کنترل روی فرم قرار می‌گیرد. هر فرم یک خصوصیت Controls دارد که یک نوع داده‌ی Control.Collection برمی‌گرداند و کلکسیون از اشیاء موجود روی فرم ارائه می‌کند. در این مثال متد Controls.Add برای اضافه کردن یک دکمه به فرم استفاده می‌شود. متد Remove برای حذف

پویای یک کنترل از فرم موردنظر است. در زمان طراحی، IDE زمان کشیدن یک کنترل به روی فرم، همان متد Add را بکار می‌برد. اگر در زمان اجرا بخواهید کنترل‌هایی حذف یا اضافه کنید، خود مسئول کنترل کدتان هستید.

کنترل‌ها تعدادی خصوصیت دارند که بر ظاهر آنها نظارت می‌کنند. دو مورد اساسی Size و Location هستند. آنها بصورت زیر پیاده‌سازی می‌شوند.

```
button\Size = new Size(۷۲،۲۴); // width, height
button\Location = new Point(۹۶،۱۱۲); //x,y
```

۳- اداره کردن رویداد کلیک دکمه

برای اداره کردن یک رویداد، باید متدی جهت پاسخ به رویداد فراهم کنیم و یک نماینده برای احضار آن متد در زمان وقوع رویداد، ایجاد کنیم. در این مثال button\Click- متدی است که رویداد را پردازش می‌کند. نماینده‌ی منتسب به رویداد Click با دستور زیر ایجاد می‌شود.

```
button\Click += new EventHandler(button\_Click);
```

این دستور یک نمونه از نماینده درونی (C) EventHandler ایجاد می‌کند و متد button\Click را با آن ثبت می‌کند.

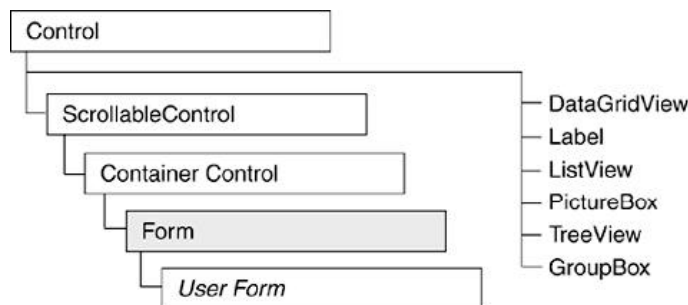
NET ۲ یک ویژگی با عنوان انواع داده‌ی جزئی اضافه می‌کند، که با آن می‌توان یک کلاس را در فایل‌های فیزیکی مجزا قرار داد (بخش‌های مختلف یک کلاس را در فایل‌های مختلف تعریف کرد). برای ایجاد یک کلاس جزئی، کلمه کلیدی *partial* را قبل از کلمه‌ی کلیدی *class* قرار دهید. توجه کنید که فقط یکی از اعلان‌ها باید وراثت را مشخص کنند. پروسه کامپایل همه‌ی این قطعه‌ها را در یک کلاس ترکیب می‌کند.

این تمرین روی این حقیقت تأکید دارد که کار با فرم‌ها شبیه کار با هر کلاس دیگر در NET است پس آشنایی با اعضای کلاس و کاربرد تکنیک‌های برنامه‌نویسی استاندارد C# جهت دسترسی به آنها لازم است.

۱۴-۲- کلاس‌های کنترل در Windows.Forms

مثال قبلی نحوه مشتق کردن یک فرم سفارشی از کلاس Windows.Forms.Form را نشان داد. حال به سلسله مراتب مشتق شدن از فرم و عملکرد پیشنهادی هر کلاس نگاهی بیاندازید.

شکل ۱۴-۲



۱۴-۲-۱- کلاس Control

احتمالاً یک امر ذاتی به نظر می‌رسد که کلاس Form در سلسله مراتب زیر کلاس Control است. چون معمولاً یک فرم کنترل‌هایی را دربر دارد. اما سلسله مراتب نشان‌دهنده‌ی وراثت است و ظرف را نشان می‌دهد. فرم یک کنترل ظرف است و اعضای کلی کلاس Control می‌توانند روی آن اعمال شوند.

System.Windows.Forms.DLL بیش از ۵۰ کنترل دارد که برای استفاده روی فرم ویندوز در دسترس هستند. تعدادی از آنها مستقیماً از Control ارث‌بری می‌کنند. همه کنترل‌ها یک مجموعه مرکزی^۱ از خصوصیات، متدها و رویدادهای ارث‌بری شده را به اشتراک می‌گذارند. این اعضا کارهایی همچون زیر را اجازه می‌دهد.

تغییر اندازه و موقعیت کنترل، آرایش آن با متن، رنگ، ویژگی‌های سبک، پاسخ به رویدادهای صفحه کلید و ماوس. این فصل خصوصیات مشترک مابین همه کنترل‌ها را بررسی می‌کند و در فصل‌های بعدی تک تک کنترل‌ها بصورت دقیق بررسی خواهند شد.

خصوصیات control

جدول ۱-۱۴ بعضی از خصوصیات عمومی که اکثر کنترل‌ها از کلاس Control ارث‌بری می‌کنند، را نشان می‌دهد.

جدول ۱-۱۴

توصیف	خصوصیت	طبقه
یک شی Size که با عرض و ارتفاع مشخص می‌شود.	Size	اندازه و موقعیت
یک شی Point که x و y مختصات کنترل را در روی فرم تعیین می‌کنند.	Location	
این مقادیر از کلاس های اندازه و موقعیت شی مشتق می‌شوند، که $Bottom=Top + Height$ و $Right=Left+Width$	Height, Width, Right, Left, Top	
یک مستطیل که اندازه و موقعیت یک کنترل را تعریف می‌کند. $=Button.Bounds$ (New Rectangle (۱۰،۱۰،۵۰،۶۰	Bounds	
ناحیه‌ای از کنترل که شامل نوار عنوان و لغزنده‌ها نیست.	ClientRectangle	
لبه‌هایی از ظرف را معین می‌کند که کنترل به آنها لنگر می‌اندازد. این عمل هنگام تغییر اندازه ظرف مفید است.	Anchor	
لبه‌هایی از ظرف که کنترل به آنها می‌چسبد.	Dock	
رنگ زمینه و پس زمینه کنترل را مشخص می‌کند. رنگ بصورت یک خصوصیت ایستا از ساختار Color معین می‌شود.	BackColor ForeColor	رنگ و ظاهر
عکسی مورد استفاده در زمینه کنترل را مشخص می‌کند.	BackGroundImage	
متن انتساب داده شده به کنترل	Text	متن
ویژگی‌های فونت متن را شرح می‌دهد. سبک حروف، اندازه، درشتی ^۲ و غیره.	Font	

^۱ Core

^۲ Bold

مقدار <code>int</code> که ترتیب <code>Tab</code> این کنترل روی ظرفیت را مشخص می‌کند.	<code>TabIndex</code>	کانون تمرکز ^۱
مقدار بولین که نشان می‌دهد آیا کنترل می‌تواند کانون کلید <code>Tab</code> باشد.	<code>TabStop</code>	
مقدار بولین که مشخص می‌کند آیا کانون روی کنترل است.	<code>Focused</code>	
مقدار بولین که مشخص می‌کند آیا کنترل نمایش داده شود؟	<code>Visible</code>	
حالت جاری دکمه‌های ماوس را برمی‌گرداند.	<code>MouseButton</code>	صفحه کلید
یک نوع داده <code>Point</code> که موقعیت جاری اشاره‌گر ماوس را برمی‌گرداند.	<code>MousePosition</code>	و ماوس
نشان می‌دهد کدام کلید کنترلی فشار داده شده است (<code>Shift</code> , <code>Alt</code> , <code>Ctrl</code>)	<code>ModifierKeys</code>	
شکل اشاره‌گر ماوس را مشخص می‌کند (زمان قرار گرفتن روی کنترل (. مقدار انتساب داده شده یک خصوصیت ایستا از کلاس <code>Cursors</code> است. <code>Hand</code> <code>Cross</code> <code>UpArrow</code> <code>Default</code> . <code>Beam</code> <code>Arrow</code> <code>WaitCursor</code> .	<code>Cursor</code>	
مقدار <code>int</code> که <code>Handle</code> کنترل ویندوز را نشان می‌دهد.	<code>Handle</code>	حالت زمان اجرا
مقدار بولین که نشان می‌دهد آیا کنترل کانون را در اختیار دارد.	<code>Focused</code>	

۱۴-۲-۲-کار با کنترل‌ها

VS.Net در زمان کشیدن یک کنترل روی فرم، تغییر اندازه و موقعیت آن، بطور اتوماتیک کد را تولید می‌کند و طراحی بصری را به مقدار خصوصیات متناظر ترجمه می‌کند. با این وجود، بعضی مواقع در حین اجرای یک برنامه، لازم است کنترل‌ها را تغییر دهید (پنهان کردن، جابجا کردن، تغییر اندازه). در حقیقت اندازه و موقعیت براساس اندازه صفحه نمایش کاربر است، که در زمان اجرا می‌تواند تشخیص داده شود. یک IDE نمی‌تواند این کار را انجام دهد. به همین دلیل ضروری است تا یک برنامه‌نویس نحوه استفاده از این خصوصیات را درک کند.

اندازه و موقعیت

همانطور که در مثال اخیر دیدیم، اندازه یک کنترل با شی `Size` که یک عضوی از فضای نامی `System.Drawing` است، تعیین می‌گردد.

```
button\Size = new Size(۸۰،۴۰); // (width, height)
button۲.Size = button\Size; //Assign size to second button
```

در زمان اجرا با انتساب یک شی `Size` به یک کنترل می‌توان آن را تغییر اندازه داد. این تکه کد نشان می‌دهد، زمانی که روی دکمه کلیک کنید، چگونه اداره‌کننده‌ی رویداد `Click` می‌تواند برای تغییر اندازه دکمه استفاده شود.

```
private void button\Click(object sender, EventArgs e)
```

^۱ Focus


```
{
MessageBox.Show("Up and Running");
Button button\ = (Button) sender; //Cast object to Button
button\ .Size = new Size(۹۰,۲۰); //Dynamically resize
```

شی `System.Drawing.Point` می‌تواند برای انتساب به `Location` کنترل استفاده شود. مجموعه آرگومانهای آن، مختصات `x` و `y` (در پیکسل) گوشه بالا - چپ یک کنترل هستند. مختصات `x` تعداد پیکسل‌ها از سمت چپ ظرف هستند و مختصات `y` تعداد پیکسل‌ها از بالای ظرف هستند.

```
button\ .Location = new Point(۲۰,۴۰); // (x,y) coordinates
```

تشخیص ارتباط این موقعیت با ظرف کنترل مهم است. اگر یک دکمه در داخل یک `GroupBox` قرار گیرد، دیگر ظرف آن `GroupBox` است نه `Form`. در حین اجرا با انتساب یک شی `Point` جدید می‌توان موقعیت یک کنترل را تغییر داد.

روش دیگر تنظیم اندازه و موقعیت استفاده از خصوصیت `Bounds` در یک دستور است.

```
button\ .Bounds = new Rectangle(۲۰,۴۰, ۱۰۰,۸۰);
```

شی `Rectangle` با مختصات گوشه بالا-چپ و گوشه پایین - راست ایجاد می‌شود.

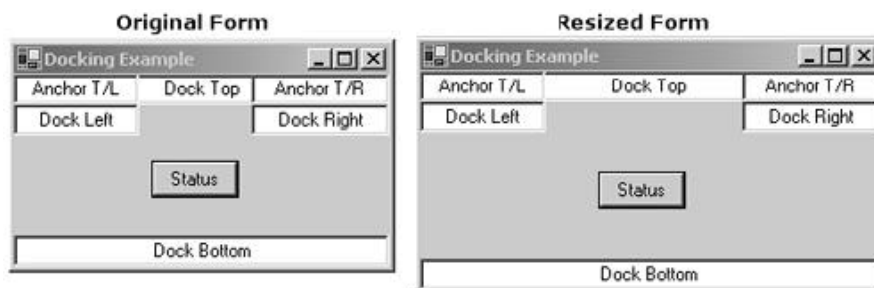
چگونه یک کنترل را لنگر ببندیم و بچسبانیم

خصوصیت `Dock` برای چسباندن یک کنترل به یکی از لبه‌های ظرف استفاده می‌شود. بیشتر مواقع مقدار پیش فرض این خصوصیت، `None` است. البته استثناء وجود دارد. کنترل‌های `StatusStrip` و `StatusBar` به `Button` و `ToolStrip` و `ToolBar` به `Up` تنظیم می‌شوند. گزینه‌های مربوطه اعضای نوع داده‌ی شمارشی شی `DockStyle` هستند که شامل `Top`، `Bottom`، `Left`، `Right` و `Fill` است. گزینه `Fill` کنترل را به هر چهار لبه می‌چسباند و زمانی که اندازه ظرف تغییر کند، اندازه آن کنترل نیز تغییر می‌یابد. برای چسباندن یک `TextBox` به بالای فرم دستور زیر را بکار برید.

```
TextBox\ .Dock = DockStyle.Top;
```

شکل ۱۴-۳ نحوه تأثیر چسباندن را روی اندازه و موقعیت کنترل هنگام تغییر اندازه فرم نشان می‌دهد.

شکل ۱۴-۳

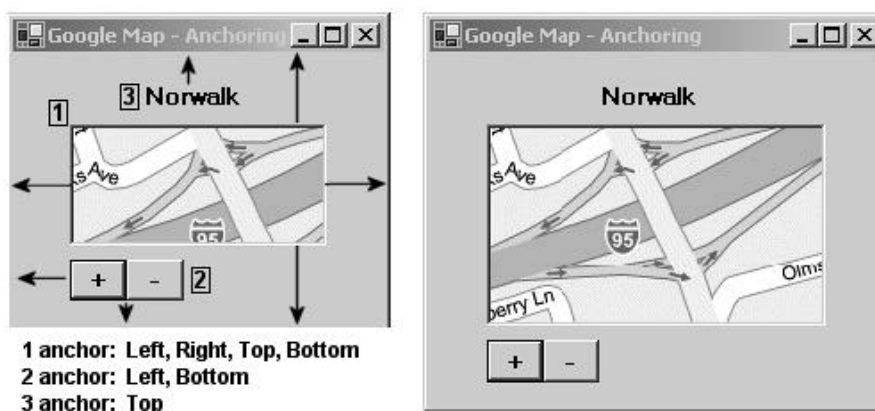


تغییر اندازه فرم، اندازه کنترل‌های چسبانده شده به راست یا چپ را تحت تأثیر قرار نمی‌دهد. با این وجود کنترل‌های چسبانده شده به بالا و پایین، بطور افقی منبسط یا منقبض می‌شوند تا کل فضای موجود را پر کنند.

کلاس `Form` و همه کنترل‌های ظرف دیگر، خصوصیت `DockPadding` را دارند که برای تنظیم مقدار فضای مابین لبه ظرف و کنترل چسبانده شده استفاده می‌شود.

خصوصیت `Anchor` اجازه می‌دهد یک کنترل در موقعیت ثابتی (مرتبط با یک ترکیبی از لبه‌ی بالا، چپ، راست یا پایین ظرف آن) قرار گیرد. شکل ۱۴-۴ تأثیرات لنگر انداختن را نمایش می‌دهد.

شکل ۱۴-۶



زمانی که فرم منبسط یا منقبض می‌شود، فاصله‌ی مابین لبه‌های لنگر انداخته شده‌ی کنترل بدون تغییری ماند. PictureBox از نظر افقی و عمودی منبسط می‌شود تا فاصله آن از همه لبه‌ها ثابت بماند. کنترل Panel یک فاصله ثابت از لبه‌ی چپ و پایین نگه می‌دارد و Label فقط به بالا لنگر انداخته است و فاصله آن از لبه‌های بالا، چپ و راست بدون تغییر می‌ماند.

کد تعریف یک موقعیت لنگر کنترل، خصوصیت Anchor را با مقادیر شمارشی AnchorStyles مقداردهی می‌کند. چندین مقدار با استفاده از عملگر OR(۱) ترکیب می‌شوند.

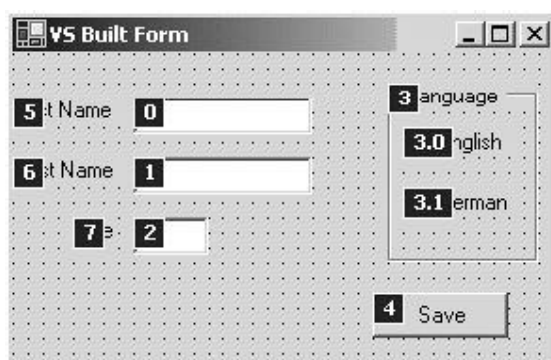
```
btnPanel.Anchor = (AnchorStyles.Bottom | AnchorStyles.Left);
```

ترتیب Tab و کانون

ترتیب Tab دنباله‌ای تعریف می‌کند، تا زمانی که کلید Tab فشرده می‌شود، کانون با همان ترتیب به کنترل‌ها وارد می‌شود. دنباله‌ی پیش فرض، ترتیبی است که کنترل‌ها به ظرف اضافه می‌شوند.

ترتیب Tab باید دنباله‌ی منطقی که کاربران برای ورود اطلاعات انتظار دارند، پیش‌بینی کند و آنها را از طریق پردازش راهنمایی کند. فرم شکل ۱۴-۵ همچون طرحی را نشان می‌دهد. کاربر می‌تواند کلید Tab را از اولین فیلد تا فیلدهای بعدی پایین نگه دارد، تا دکمه‌ای که عمل نهایی را احضار می‌کند.

شکل ۱۴-۵



دو چیز را در شکل مشاهده کنید. اول اینکه، اگرچه برچسب‌ها یک ترتیب Tab دارند، ولی آنها صرفنظر می‌شوند و هرگز کانون را بدست نمی‌آورند. دوم اینکه، کنترل‌های یک ظرف یک ترتیب Tab مرتبط با خود آن دارند نه با فرم.

ترتیب Tab کنترل با مقدار خصوصیت TabIndex تعیین می‌شود.

```
TextBox\Tabindex = ۰; //First item in tab sequence
```

در VS.NET می‌توانید مقدار این خصوصیت را مستقیماً با Property Manager یا با اجرای `View→TabOrder` و کلیک بر روی کادر هر کنترل تنظیم کنید. اگر می‌خواهید یک کنترل در ترتیب Tab قرار نگیرد. مقدار `Tabstop` را False قرار دهید. با این وجود، با کلیک ماوس می‌توان کانون را به آن کنترل داد.

زمانی که فرم بارگذاری می‌شود، کانون روی کنترلی قرار می‌گیرد که پایین‌ترین مقدار `TabIndex` را دارد. در حین اجرا، با استفاده از متد `Focus` می‌توان کانون را به یک کنترل خاص انتقال داد.

```
if(textBox1.CanFocus)
{
    textBox1.Focus();
}
textBox1.Focus();
```

طی کردن همه کنترل‌های روی یک فرم

همه کنترل‌های روی یک فرم در کلکسیون `Controls` قرار دارند. با شمارش سرتاسر این کلکسیون، بررسی هر کنترل، تعیین نام و نوع آن، تغییر خصوصیات مورد نیاز امکان‌پذیر است. یک کاربرد معمول، پاک کردن مقدار فیلدهای انتخاب شده روی فرم در یک عمل‌نوسازی^۱ است. این مثال کوتاه، هر کنترل در شکل ۱۴-۵ را بررسی می‌کند و نام و نوع آن را نمایش می‌دهد.

```
int ctrlCt = this.Controls.Count; // ۸
foreach (Control ct in this.Controls)
{
    object ob = ct.GetType();
    MessageBox.Show(ob.ToString()); //Displays type as string
    MessageBox.Show(ct.Name);
}
```

در زمان شمارش اشیاء کنترل، چندین مورد وجود دارد که باید آگاه باشید:

- نوع هر کنترل بصورت یک نام کامل برگردانده می‌شود. برای مثال `TextBox` به صورت `System.Forms.Form.TextBox` بیان می‌شود.
- فقط اشیاء موجود در ظرف سطح بالا لیست می‌شوند. در این مثال، مقدار `Controls.Count` به جای مقدار ۱۰، مقدار ۸ است. چون `GroupBox` به صورت یک کنترل شمارش می‌شود و کنترل‌های فرزند آن مستثنی می‌شود.
- می‌توانید یک خصوصیت `HasChildren` کنترل را برای تعیین ظرف بودن کنترل بکار ببرید.

مثال ۱۴-۲ برای لیست کردن همه کنترل‌های فرزند بصورت بازگشتی بکار می‌برد.

مثال ۱۴-۲

```
void IterateThroughControls(Control parent)
{
    foreach (Control c in parent.Controls)
    {
        MessageBox.Show(c.ToString());
        if(c.HasChildren)
        {
            IterateThroughControls(c);
        }
    }
}
```

اعمال این کد به شکل ۱۴-۵ همه‌ی کنترل‌های روی فرم اصلی را بصورت لیست سلسله‌مراتبی نتیجه می‌دهد. یک کنترل به همراه فرزندان خود لیست می‌شود.

۱۴-۲-۳- رویدادهای Control

زمانی که یک کلید صفحه کلید را فشار می‌دهید یا ماوس را کلیک می‌کنید، کنترل مرتبط یک رویداد برای نشان دادن عمل خاصی که رخ داده است را می‌کند. یک رویداد ثبت شده، روالی که به رویداد پاسخ می‌دهد را اداره می‌کند و آن عملی که باید انجام گیرد را قانونی می‌کند.

اولین گام اداره کردن یک رویداد، تعیین نماینده منتسب شده به رویداد است. سپس باید متد اداره کردن رویداد را با آن ثبت کنید و مطمئن شوید نشانه متد با پارامترهای مشخص شده نماینده تطابق دارد. جدول ۱۴-۲ اطلاعات مورد نیاز برای کار با رویدادهای رها شده ماوس و صفحه کلید را خلاصه می‌کند.

جدول ۱۴-۲

رویداد	پارامترها / نماینده درونی	توصیف
Click DoubleClick MouseEnter MouseLeave MouseOver MouseWheel	Event Handler (Object sender, EventArgs e)	رویدادهای رها شده با کلیک کردن، دابل کلیک کردن یا حرکت ماوس
MouseDown Mouse UP MouseMove	Mouse Event Hardler (Objectsender , MouseEventArgs e)	رویدادهای رها شده با ماوس و حرکات دکمه ماوس. توجه کنید اگر عمل ماوس روی یک کنترل در ظرف جاری رخ دهد، این رویداد رها می‌شود
KeyUp KeyDown	KeyEvent Hardler(Object sender, KeyEventArgs e)	رویدادهای رها شده با پایین یا بالا کردن کلیدهای صفحه کلید
KeyPress	KeyPressEventHardler Object sender (KeyPressEventArgs e)	رویداد رها شده با فشار دادن هر کلیدی

اداره کردن رویدادهای ماوس

علاوه بر رویدادهای آشنای Click و DoubleClick، همه کنترل‌های فرم ویندوز، رویدادهای MouseHover و MouseEnter و MouseLeave را به ارث می‌برند. دو رویداد آخری زمانی که ماوس به محدوده‌ی یک کنترل وارد شده یا آن را ترک می‌کند رها می‌شوند. آنها برای ایجاد یک اثر Mouseover مفید هستند که برای صفحات وب معمول هستند.

برای نشان دادن این مطلب، مثالی که زمان حرکت ماوس روی کادر متن، رنگ پیش زمینه آن را تغییر می‌دهد، ملاحظه کنید. کد زیر نماینده‌هایی برای فراخوانی OnMouseEnter و OnMouseLeave جهت انجام تغییر رنگ پیش زمینه تنظیم می‌کند.

```
TextBox userID = new TextBox();
userID.MouseEnter += new EventHandler(OnMouseEnter);
userID.MouseLeave += new EventHandler(OnMouseLeave);
```

متدهای اداره کننده رویداد، نشانه‌ی نماینده EventHandler را تطابق می‌دهند و پارامتر sender را به نوع Control برای دستیابی به خصوصیات آن قالب‌بندی می‌کند.

```
private void OnMouseEnter(object sender, EventArgs e)
{
```

```
Control ctrl = (Control) sender;
ctrl.BackColor= Color.Bisque;
}
private void OnMouseLeave(object sender, System.EventArgs e)
{
Control ctrl = (Control) sender;
ctrl.BackColor= Color.White;
}
```

اداره کردن رویدادها با *override* کردن اداره‌کننده‌های رویداد پیش‌فرض کلاس پایه (Control) امکان‌پذیر است. این متدها با الگوی *OnEventArgs* نامگذاری می‌شوند. در *NET* مکانیزم نماینده، تعیین چندین اداره‌کننده رویداد برای یک رویداد را مجاز می‌دارد و همچنین پردازش چندین رویداد توسط یک اداره‌کننده‌ی رویداد منفرد را مجاز می‌دارد.

نماینده‌های رویدادهای *MouseDown*، *MouseUp* و *MouseMove* به جای آرگومان کلی *EventArgs*، آرگومان دوم خود را از نوع *MouseEventArgs* می‌گیرند. این نوع داده، اطلاعات حالت اضافی درباره ماوس را از طریق خصوصیات نشان داده شده در جدول ۱۴-۳ در اختیار قرار می‌دهد.

جدول ۱۴-۳

توصیف	خصوصیت
مشخص می‌کند کدام کلید ماوس فشار داده شده است. مقدار خصوصیت با نوع شمارشی <i>MouseButtons</i> تعیین می‌شود.	Button
تعداد کلیک‌ها در زمان آخرین رویداد	Clicks
تعداد دورهایی که چرخ ماوس می‌چرخد. مقدار مثبت نشان‌دهنده حرکت به جلو و مقدار منفی نشان‌دهنده حرکت به عقب است.	Delta
مختصات ماوس مرتبط با گوشه چپ-بالای ظرف. این معادل خصوصیت <i>MousePosition</i> کنترل است.	X, Y

خصوصیات این جدول عملاً برای برنامه‌هایی که باید حرکات ماوس را روی یک فرم دنبال کنند، مفید هستند. مثال معمول برنامه‌های گرافیکی هستند که به موقعیت ماوس و دکمه برای کنترل ترسیم روی صفحه تکیه دارند. برای ارائه این مطلب، مثال ۱۴-۳ یک برنامه ترسیم ساده است که یک خط روی فرم با شروع از محلی که کلید ماوس فشار داده می‌شود تا جایی که آن رها می‌شود رسم می‌کند. برای اینکه آن برنامه کمی جالب شود، اگر دکمه چپ را بکشید خط سیاه رسم می‌کند و اگر دکمه راست کشیده شود خط قرمز رسم می‌کند.

مثال ۱۴-۳

```
using System;
using System.Windows.Forms;
using System.Drawing;
class MyWinApp
{
static void Main()
{
Form mainForm = new DrawingForm();
Application.Run(mainForm);
}
}
// User Form derived from base class Form
class DrawingForm:Form
{
Point lastPoint = Point.Empty; // Save coordinates
public Pen myPen; //Defines color of line
```

```

public DrawingForm()
{
    this.Text = "Drawing Pad";
    // reate delegates to call MouseUp and MouseDown
    this.MouseDown += new MouseEventHandler(OnMouseDown);
    this.MouseUp += new MouseEventHandler(OnMouseUp);
}
private void OnMouseDown(object sender, MouseEventArgs e)
{
    myPen = (e.Button==MouseButtons.Right)? Pens.Red:
    Pens.Black;
    lastPoint.X = e.X;
    lastPoint.Y = e.Y;
}
private void OnMouseUp(object sender, MouseEventArgs e)
{
    // Create graphics object
    Graphics g = this.CreateGraphics();
    if (lastPoint != Point.Empty)
    g.DrawLine(myPen, lastPoint.X,lastPoint.Y,e.X,e.Y);
    lastPoint.X = e.X;
    lastPoint.Y = e.Y;
    g.Dispose();
}
}

```

حتی بدون فهم از گرافیک NET، نقش کلاسهای گرافیکی مرتبط باید بدیهی باشد. یک شی Graphics از متد DrawLine برای انجام ترسیم واقعی استفاده می‌کند. پارامترهای این متد، شی Pen و مختصات خط مورد نظر هستند. زمانی که دکمه فشرده می‌شود، برنامه مختصات را ذخیره می‌کند و myPen را براساس نوع دکمه فشار داده شده تنظیم می‌کند (یک قلم قرمز برای دکمه راست و یک قلم سیاه برای دکمه چپ). زمانی که دکمه‌ی ماوس رها می‌شود، خط از مختصات قبلی به موقعیت جاری رسم می‌شود. شی MouseEventArgs همه اطلاعات مورد نیاز را برای انجام این کار فراهم می‌کند.

اداره کردن رویدادهای صفحه کلید

رویدادهای صفحه کلید با تعریف یک نماینده برای فراخوانی متد اداره‌کننده‌ی رویداد سفارشی اداره می‌شوند. دو آرگومان به اداره‌کننده‌ی رویداد ارسال می‌شود. آرگومان sender شی رها کننده‌ی رویداد را مشخص می‌کند و آرگومان دوم فیلدهایی که رویداد را توصیف می‌کنند در بردارد. برای رویداد KeyPress آرگومان دوم از نوع KeyPressEventArgs است. این نوع داده، یک فیلد Handled دارد، که با روال اداره‌کننده رویداد true قرار داده می‌شود تا تعیین کند آن رویداد را پردازش کرده است. خصوصیت دیگر KeyChar است که کلید فشار داده شده را معین می‌کند.

Keychar برای محدود کردن ورودی یک فیلد مفید است. این قطعه کد نشان می‌دهد، محدود کردن ورودی یک فیلد به ارقام چقدر ساده است. زمانی که یک کاراکتر غیر عددی وارد می‌شود، Handled به true تنظیم می‌شود که موتور فرم را از نمایش کاراکتر منع می‌کند. در غیر اینصورت روال اداره‌کننده‌ی رویداد، هیچ کاری انجام نمی‌دهد و کاراکتر نمایش داده می‌شود.

```

private void OnKeyPress(object sender, KeyPressEventArgs e)
{
    if (! char.IsDigit(e.KeyChar)) e.Handled = true;
}

```

رویداد KeyPress فقط برای کلیدهای کاراکتری قابل چاپ رها می‌شود. آن کلیدهای غیر کاراکتری همچون Alt و Shift را صرف نظر می‌کند. آن رویداد، همه ضربه‌های کلیدها را تشخیص می‌دهد و برای شروع رویدادهای KeyDown و KeyUp ضروری است. اداره‌کننده‌های رویدادهای مربوطه، یک پارامتر از نوع KeyEventArgs دریافت می‌کنند که ضربه کلید تکی یا ضربه ترکیب کلیدها را تشخیص می‌دهد. جدول ۱۴-۴ خصوصیات مهم فراهم شده بوسیله KeyEventArgs را لیست می‌کند.

جدول ۴-۱۴

عضو	توصیف
Alt-Control-Shift	مقدار بولین که فشار داده شدن کلیدهای Alt و Control و Shift مشخص می‌کند.
Handled	مقدار بولین که مشخص می‌کند آیا یک رویداد اداره شده است.
KeyCode	کد کلید را برمی‌گرداند. این کد از نوع شمارشی Keys است.
KeyData	داده کلید را برای رویداد برمی‌گرداند. این هم از نوع شمارشی Keys است با این تفاوت که کلیدهای چندگانه را تشخیص می‌دهد.
Modifiers	ترکیب کلیدهای کنترلی فشار داده شده را تشخیص می‌دهد. (Alt و ctrl و shift)

به چند مورد توجه کنید:

- خصوصیات Alt، Control و Shift، میان‌برهای ساده‌ای برای مقایسه مقدار KeyCode با اعضای Alt، Control یا Shift از نوع شمارشی Keys هستند.

- KeyCode مقدار یک کلید تک را نشان می‌دهد. KeyData یک مقدار برای کلید تکی یا ترکیبی از کلیدهای فشار داده شده را در بردارد.

- نوع شمارشی Keys، به تشخیص کلیدها مجرمانه است، چون اعضای آن همه کلیدها را نشان می‌دهد. اگر VS.NET را بکار می‌برید، زمانی که نوع شمارشی Keys در یک حالت متشابه استفاده شود، سیستم هوشمند همه اعضای آن را لیست می‌کند. برای مثال ارقام با ۱D و ۲D و غیره نمایش داده می‌شوند.

قطعه کد قبلی نشان می‌دهد، چگونه KeyPress را بکار ببریم تا مطمئن شویم یک کاربر فقط کلیدهای عددی را فشار می‌دهد. با این وجود از چسباندن (Paste) داده‌های غیرعددی به کمک کلید ترکیبی Ctrl+V جلوگیری نمی‌کند. با استفاده از رویداد KeyDown می‌توان این کلید ترکیبی را تشخیص داد، تا با تنظیم یک پرچم، اداره‌کننده رویداد KeyPress را راهنمایی کند تا تلاش برای چسباندن را صرف‌نظر کند.

در این مثال دو اداره‌کننده رویداد ثبت می‌شود تا در زمان وارد کردن داده‌ها از طریق صفحه کلید فراخوانی شوند. ابتدا KeyDown احضار می‌شود و اگر کاربر کلید ترکیبی Ctrl+V را فشار دهد، مقدار Paste را true قرار می‌دهد. اداره‌کننده‌ی رویداد KeyPress این پرچم را برای تعیین اینکه آیا ضربه‌های کلید را بپذیرد بکار می‌برد.

```
private bool paste;
//Register event handlers for TextBox t.
//They should be registered in this order,
//because the last registered is the
//first executed
t.KeyPress += new KeyPressEventHandler(OnKeyPress);
t.KeyDown += new KeyEventHandler(OnKeyDown);
private void OnKeyDown(object sender, KeyEventArgs e)
{
    if (e.Modifiers == Keys.Control && e.KeyCode == Keys.V)
    {
        paste=true; //Flag indicating paste attempt
        string msg = string.Format("Modifier: {0} \nKeyCode: {1} \nKeyData: {2}",
            e.Modifiers.ToString(), e.KeyCode.ToString(), e.KeyData.ToString());
        MessageBox.Show(msg); //Display property values
    }
}
```



```

}
private void OnKeyPress(object sender, KeyPressEventArgs e)
{
if (paste==true) e.Handled = true;
}

```

این برنامه مقادیر زیر را برای خصوصیات انتخاب شده از KeyEventArgs در حین فشار دادن Ctrl+V نمایش می‌دهد.

```

Modifier: Control
KeyCode: V
KeyData: Control, V

```

۱۴-۳-کلاس Form

کلاس Form تمامی اعضای کلاس Control را همانند ScrollableControl، (که خصوصیات فراهم می‌کند تا قادر به لغزاندن باشد) ارث‌بری می‌کند. بدین علت تعداد زیادی خصوصیات برای کنترل ظاهر فرم، کار با فرم‌های فرزند، ایجاد فرم‌های Modal، نمایش منوها و تعامل با میزکار از طریق نوارهای حالت و ابزار اضافه می‌کند. جدول ۱۴-۵ لیست انتخابی از این خصوصیات را نشان می‌دهد.

جدول ۱۴-۵

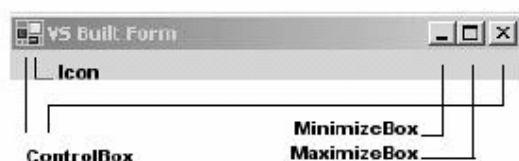
توصیف	خصوصیت	طبقه
سبک لبه‌ی فرم را بدست آورده یا مقدار دهی می‌کند. آن با نوع شمارشی FormBorderStyle تعریف می‌شود. Fixed*None FixedSingle Sizable FixedDialog	FormBorderStyle	ظاهر
مقدار بولین که تعیین می‌کند آیا آیکون منو در گوشه چپ فرم و دکمه close در گوشه راست دیده شوند.	ControlBox	
مقدار بولین که مشخص می‌کند آیا دکمه‌های کوچک‌کردن و بزرگ‌کردن فرم روی فرم نشان داده شوند.	MaximizeBox MinimizeBox	
کدری فرم و همه کنترل‌های روی آن را بدست آورده یا مقداردهی می‌کند. مقدار ماکزیمم آن ۱ است. در ویندوز ۹۵ و ۹۸ کار نمی‌کند.	Opacity	
یک رنگ که ناحیه شفاف روی فرم را نشان می‌دهد. هر کنترل یا بخشی از فرم که رنگ پیش‌زمینه یکسانی با آن دارند نمایش داده نمی‌شوند. کلیک روی این ناحیه شفاف، رویداد را به فرم زیرین آن ارسال می‌کند.	TransparencyKey	
نشان می‌دهد آیا فرم اندازه خود با اندازه فونت مورد استفاده تطبیق می‌دهد.	AutoScale	اندازه و موقعیت
اندازه فرم به استثناء لبه‌ها و نوار عنوان	ClientSize	
یک نوع داده Point که محل قرارگیری فرم روی پنجره میزکار را مشخص می‌کند.	DesktopLocation	
موقعیت اولیه یک فرم را مشخص می‌کند. آن یک مقدار از نوع	StartPosition	

شیء FormStartPosition می‌گیرد.		
CenterParent: در مرکز محدود فرم پدر قرار می‌گیرد.		
CenterScreen: در مرکز صفحه نمایش		
Manual: مقدار DesktopLocation را بکار می‌برد.		
WindowsDefaultLocation: ویندوز مقدار آن را تنظیم می‌کند.		
یک شیء Size که اندازه‌ی حداقل و حداکثر فرم را معین می‌کند.	MinimumSize	
مقدار (۰،۰) نشان می‌دهد هیچ محدودیتی وجود ندارد.	MaximumSize	
مقدار بولین که مشخص می‌کند آیا عنوان برنامه در نوار وظیفه ظاهر گردد. مقدار پیش‌فرض آن true است.	ShowInTaskBar	
مشخص می‌کند آیا فرم به صورت پنجره TopLevel یا پنجره TopMost ظاهر گردد. یک پنجره‌ی TopLevel پدر ندارد. فرم TopMost همواره در بالای همه فرم‌های غیر TopMost نمایش داده می‌شود.	TopLevel TopMost	
مشخص می‌کند که فرم در شروع کار، چگونه نمایش داده شود. آن مقدار خود را از نوع شمارشی FormWindowState می‌گیرد.	WindowState	
Normal, Minimized, Maximized		
فرم به عنوان مالک فرم معین می‌گردد.	Owner	فرم‌های مالک
یک آرایه از نوع Form که فرم‌های ملکی ^۱ بوسیله‌ی یک فرم را شامل است.	OwnedForms	

۱۴-۳-۱ تنظیم ظاهر یک فرم

چهار خصوصیت نشان داده شده در شکل ۱۴-۶ کنترل می‌کنند کدام دکمه و آیکون روی لبه‌ی بالای فرم نمایش داده شوند. خصوصیت Icon فایل ico ای را معین می‌کند که به عنوان آیکون گوشه چپ استفاده می‌شود. مقدار ControlBox تعیین می‌کند آیا آیکون و دکمه close نمایش داده شوند یا نمایش داده نشوند. بطور مشابه در مورد MaximizeBox و MinimizeBox نیز چنین است.

شکل ۱۴-۶



هدف این خصوصیات چیزی بالاتر از کنترل عملکرد ظاهر فرم است. به عنوان مثال، دکمه‌های کوچک و بزرگ کردن را از روی فرم‌های modal برای جلوگیری از بزرگ و کوچک کردن فرم پدر برمی‌دارند.

^۱ Owned

کد ری^۱ فرم

خصوصیت `opacity` سطح شفافیت فرم را تعیین می‌کند. محدوده‌ی مقادیر آن از ۰ تا ۱ است که هر مقدار کمتر از ۱ شفافیت جزئی ایجاد می‌کند تا اجازه دهد عناصر زیر فرم نمایش داده شوند. بیشتر فرم‌ها با مقدار ۱ بهتر کار می‌کنند، اما تنظیم کد ری می‌تواند یک راه مؤثر برای نمایش فرزند با فرم‌های `TopMost` باشد که یک فرم اصلی را پنهان می‌کند. یک روش معمول تنظیم کد ری یک فرم به مقدار ۱ در هنگام دریافت کانون و کاهش کد ری هنگام از دست دادن کانون است. اغلب این تکنیک با پنجره‌های جستجو استفاده می‌شود که روی بالاترین سند در حال جستجو شناور است.

حال می‌خواهیم نشان دهیم، چگونه کد ری یک فرم را هنگام فعال شدن به ۱ و هنگام غیرفعال شدن به ۰/۸ تنظیم کنیم. برای این کار از رویدادهای `DeActivate` و `Activated` بهره می‌گیریم، که زمان از دست دادن یا بدست آوردن کانون رها می‌شوند. ابتدا نماینده‌های فراخوانی روال‌های اداره کردن رویداد را مقداردهی می‌کنیم.

```
this.Deactivate += new System.EventHandler(Form_Deactivate);
this.Activated += new System.EventHandler(Form_Activate);
```

کد اداره‌کننده رویداد متناظر بسیار کوچک است

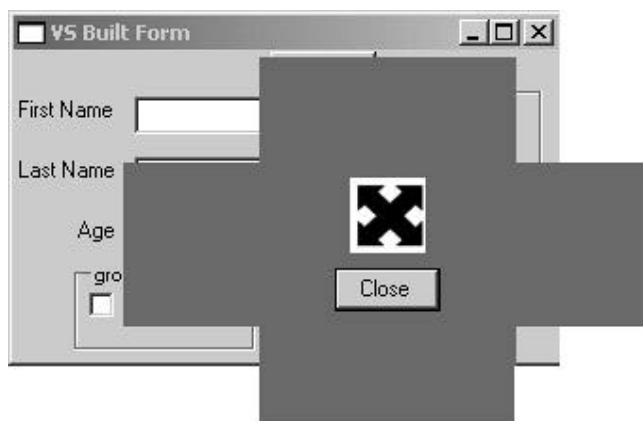
```
void Form_Deactivate(object sender, EventArgs e)
{
    this.Opacity= .۸;
}
void Form_Activate(object sender, EventArgs e)
{
    this.Opacity= ۱;
}
```

شفافیت فرم

شفافیت کل فرم بوسیله کد ری تحت تأثیر قرار می‌گیرد. خصوصیت دیگری بنام `TransparencyKey` وجود دارد که ناحیه‌ی انتخاب شده از کل فرم را شفاف می‌کند. این خصوصیت یک رنگ پیکسل را تعیین می‌کند که زمان ترسیم فرم بصورت شفاف درمی‌آید. تأثیر آن شبیه ایجاد یک چاله است که هر ناحیه‌ی زیر فرم را نمایان می‌کند. در حقیقت اگر روی یک ناحیه شفاف کلیک کنید، رویداد توسط فرم زیرین تشخیص داده می‌شود.

معمولی‌ترین کاربرد شفافیت به ایجاد فرم‌های غیر مستطیلی است. زمانی که سبک `FormBorderStyle.None` استفاده شود، نوار عنوان حذف می‌شود و فرمی با شکل هندسی می‌تواند ایجاد شود. مثال بعدی نحوه‌ی ایجاد یک فرم صلیب شکل را نشان می‌دهد.

شکل ۱۴-۷



آنچه که نیاز داریم این است که ناحیه‌های شفاف فرم را با همان رنگ TransparencyKey هم رنگ سازیم. مطمئن شوید که این رنگ در جای دیگر فرم استفاده نخواهد شد. روش استاندارد تنظیم رنگ پیش زمینه‌ی فرم به رنگ TransparencyKey و ترسیم یک تصویر با رنگ متفاوت است.

برای ایجاد فرم شکل ۷-۱۴، کنترل های Panel را در هر گوشه از فرم استاندارد قرار داده و خصوصیات BackColor آنها را Color.Red قرار دهید. با استفاده از کد زیر فرم ایجاد شده و نمایش داده می‌شود.

```
CustomForm myForm = new CustomForm();
myForm.TransparencyKey = Color.Red;
myForm.FormBorderStyle= FormBorderStyle.None;
myForm.Show();
```

این قطعه کد، تاثیر ایجاد ناحیه‌های پانل شفاف و حذف نوار عنوان را نشان می‌دهد. بعد از ناپدید شدن نوار عنوان، باید یک روش برای حرکت دادن فرم فراهم کنیم، نیاز به اداره کننده‌ی رویداد ماوس احساس می‌شود.

در مرکز فرم یک تصویر چهار جهته در یک PictureBox نشان داده می‌شود، که کاربر با کشیدن آن می‌تواند فرم را بکشد. مثال ۱۴-۴ رویداد های MouseMove، MouseUp، MouseDown را برای پیاده‌سازی حرکت فرم نشان می‌دهد.

مثال ۱۴-۴

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
public class CustomForm : Form
{
    private Point lastPoint = Point.Empty; //Save mousedown
    public CustomForm()
    {
        InitializeComponent(); // set up form
        //Associate mouse events with pictureBox
        pictureBox.MouseDown += new MouseEventHandler(OnMouseDown);
        pictureBox.MouseUp += new MouseEventHandler(OnMouseUp);
        pictureBox.MouseMove += new MouseEventHandler(OnMouseMove);
    }
    private void OnMouseDown(object sender, MouseEventArgs e)
    {
        lastPoint.X = e.X;
        lastPoint.Y = e.Y;
    }
    private void OnMouseUp(object sender, MouseEventArgs e)
    {
        lastPoint = Point.Empty;
    }
    //Move the form in response to the mouse being dragged
    private void OnMouseMove(object sender, MouseEventArgs e)
    {
        if (lastPoint != Point.Empty)
        {
            //Move form in response to mouse movement
            int xInc = e.X - lastPoint.X;
            int yInc = e.Y - lastPoint.Y;
            this.Location = new Point(this.Left + xInc,
            this.Top+yInc);
        }
    }
    // Close Window
    private void button1_Click(object sender, EventArgs e)
    {
        this.Close();
    }
}
```

```

}
}

```

منطق این مثال سراسر است. زمانی که کاربر روی PictureBox کلیک می‌کند، مختصات در lastPoint ذخیره می‌شود. زمانی که کاربر ماوس را حرکت می‌دهد، خصوصیت Location فرم را برای اعمال تفاوت مابین مختصات جدید و موقعیت ذخیره شده قبلی مقداردهی می‌کند. زمانی که ماوس رها می‌شود، lastPoint پاک می‌شود. توجه داشته باشید که پیاده‌سازی کامل به کد اداره کردن تغییر اندازه‌ی فرم را نیز نیاز دارد.

تنظیم اندازه و موقعیت فرم

موقعیت اولیه فرم بصورت مستقیم یا غیرمستقیم بوسیله خصوصیات StartPosition تعیین می‌شود. همانطور که جدول ۱۴-۶ تشریح کرده، مقادیر آن خصوصیات از نوع شمارشی FormStartPosition است. این مقادیر برای قرار گرفتن فرم در مرکز صفحه‌ی نمایش، مرکز فرم پدر، در موقعیت پیش‌فرض پنجره‌ها یا یک موقعیت انتخاب شده دلخواه استفاده می‌شوند. مقدار Manual انعطاف‌پذیری بالایی دارد، چون اجازه می‌دهد برنامه موقعیت را تنظیم کند.

جدول ۱۴-۶

عمل	رویدادهای رها شده	توصیف
ایجاد شی فرم		سازنده‌ی فرم فراخوانی می‌شود. در VS متد InitializeComponent برای مقداردهی اولیه فرم فراخوانی می‌شود
نمایش فرم	Form.Load Form.Activated	ابتدا رویداد Load و به دنبال آن رویداد Activated فراخوانی می‌شود.
فعال شدن فرم	Form.Activated	زمانی که کاربر فرم را انتخاب می‌کند رخ می‌دهد.
غیرفعال شدن فرم	Form.DeActivated	زمانی که فرم کانون را از دست می‌دهد غیر فعال می‌شود.
بستن فرم	Form.DeActivated Form.Closing Form.Closed	با اجرای Form.Close یا کلیک روی دکمه بستن، فرم بسته می‌شود.

موقعیت اولیه در اداره کننده رویداد Form.Load قرار داده می‌شود. این مثال فرم را در موقعیت (۲۰۰،۰) گوشه‌ی چپ-بالا بارگذاری می‌کند.

```

private void opaque_Load(object sender, System.EventArgs e)
{
    this.DesktopLocation = new Point(۲۰۰،۰);
}

```

موقعیت اولیه‌ی فرم را می‌توان در فرمی که شی فرم را ایجاد کرده و نمایش می‌دهد، تنظیم کرد.

```

opaque opForm = new opaque();
opForm.Opacity = ۱;
opForm.TopMost = true; //Always display form on top
opForm.StartPosition = FormStartPosition.Manual;
opForm.DesktopLocation = new Point(۱۰،۱۰);
opForm.Show();

```

اندازه‌ی فرم می‌تواند با استفاده از خصوصیات Size و ClientSize مقداردهی گردد. معمولاً خصوصیت دوم می‌توصیه می‌شود، چون ناحیه‌ی تعاملی فرم را معین می‌کند. این خصوصیت با یک نمونه از شی Size مقداردهی می‌شود.

```
this.ClientSize = new System.Drawing.Size(۲۰۸, ۱۳۳);
```

اغلب مطلوب است موقعیت و اندازه‌ی یک فرم را متناسب با اندازه‌ی صفحه‌ی نمایش مشخص کنیم. اندازه‌ی صفحه‌ی نمایش از طریق خصوصیت `Screen.PrimaryScreen.WorkingArea` در دسترس است. این خصوصیت یک مستطیل بر می‌گرداند، که اندازه‌ی صفحه‌ی نمایش را بدون نوارهای وظیفه، نوارهای ابزار لنگر انداخته و پنجره‌های لنگر انداخته نشان می‌دهد. مثال زیر اندازه‌ی صفحه‌ی نمایش را برای مقداردهی ارتفاع و عرض فرم را بکار می‌برد.

```
int w = Screen.PrimaryScreen.WorkingArea.Width;
int h = Screen.PrimaryScreen.WorkingArea.Height;
this.ClientSize = new Size(w/۴, h/۴);
```

بعد از این که فرم فعال شد، ممکن است بخواهید نحوه‌ی تغییر اندازه‌ی آن را کنترل کنید. خصوصیات `MinimumSize` و `MaximumSize` توجه بیشتری لازم دارند. در مثال زیر اندازه‌ی حداقل و حداکثر یک فرم تنظیم می‌گردد.

```
//w and h are the screen's width and height
this.MaximumSize = new Size(w/۲, h/۲);
this.MinimumSize = new Size(۲۰۰, ۱۵۰);
```

مقداردهی عرض و ارتفاع با صفر، هر نوع محدودیتی را حذف می‌کند.

۱۴-۳-۲-نمایش فرم‌ها

بعد از این که فرم اصلی اجرا شد و بالا آمد، آن می‌تواند نمونه‌هایی از فرم جدید را ایجاد کرده و به دو روش نمایش دهد: با استفاده از متد `Form.ShowDialog` یا متد `Form.ShowDialog` که از کلاس `Control` به ارث برده است. فرم را به صورت یک کادر دیالوگ `Modal` نشان می‌دهد. این نوع فرم بعد از فعال شدن تا زمانی که بسته نشود، کنترل را به فرم‌های دیگر رها نمی‌کند. کادرهای دیالوگ در فصل‌های بعدی بررسی می‌شوند.

متد `Form.Show` یک فرم `Modeless` را نشان می‌دهد. بدین معنی که هیچ رابطه‌ای با فرم ایجاد کننده ندارد و کاربر درانتخاب فرم جدید یا قدیمی آزاد است. اگر فرم ایجاد کننده، فرم اصلی نباشد، بستن آن هیچ تاثیری روی فرم جدید ندارد، در غیر این صورت با بستن فرم اصلی، همه فرم‌های برنامه‌ی کاربردی بسته می‌شوند.

چرخه‌ی زندگی یک فرم `modeless`

فرم تحت تسلط یک تعداد معین از فعالیت‌ها در عمر خود است: آن ایجاد می‌شود، نمایش داده می‌شود، کانون را بدست آورده و از دست می‌دهد و در نهایت بسته می‌شود. بیشتر این فعالیت‌ها با یک یا چند رویداد همراه می‌شود تا برنامه را قادر سازد، عمل‌های انتساب داده شده به این رویدادها را انجام دهند. جدول ۱۴-۶ این عمل‌ها و رویدادها را خلاصه می‌کند.

ایجاد و نمایش فرم

زمانی که یک فرم، فرم دیگری را اجرا می‌کند، آن کدهای موردنظر هر دو طرف را می‌نویسد. باید فرم ایجاد شده برای انجام مقداردهی اولیه و اضافه کردن کنترل‌ها راه‌اندازی شود. علاوه بر این، باید نماینده‌ها برای فراخوانی روتین‌های اداره کردن رویدادها تنظیم گردند. اگر `VS.Net` را بکار می‌برید، هر کد مقداردهی اولیه‌ی کاربر باید بعد از فراخوانی `InitializeComponent` قرار گیرد.

واضح‌ترین کار کلاسی که یک شی `Form` ایجاد می‌کند، ایجاد و نمایش شی است. کار دیگر این است که مطمئن شویم فقط یک نمونه از آن کلاس ایجاد می‌شود، چون ممکن نیست بخواهید به ازای هر کلیک روی یک دکمه شی جدیدی ایجاد گردد. یک روش مدیریت این عمل، بهره‌گیری از رویداد `Close` است که در حین بستن فرم ایجاد شده رخ می‌دهد. اگر فرم بسته نشده باشد، نمونه جدید ایجاد نمی‌شود. کد زیر این عمل را نشان می‌دهد.

یک نماینده‌ی EventHandler برای آگاه کردن یک متد در زمان بسته شدن فرم جدید (opform) تنظیم می‌شود. در زمان فشار دادن یک دکمه، یک پرچم کنترل می‌کند برای ایجاد یا نمایش فرم چه اتفاقی می‌افتد. اگر نمونه‌ای از فرم وجود نداشته باشد، آن ایجاد شده و نمایش داده می‌شود. متد FormActivate برای دادن کانون به فرم جدید استفاده می‌شود.

```
//Next statement is at beginning of form's code
public opaque opForm;
bool closed = true; //Flag to indicate if opForm exists
//Create new form or give focus to existing one
private void button\ _Click(object sender, System.EventArgs e)
{
    if (closed)
    {
        closed = false;
        opForm = new opaque();
        //Call OnOpClose when new form closes
        opForm.Closed += new EventHandler(OnOpClose);
        opForm.Show(); //Display new form object
    } else {
        opForm.Activate(); //Give focus to form
    }
}
//Event handler called when child form is closed
private void OnOpClose(object sender, System.EventArgs e)
{
    closed = true; //Flag indicating form is closed
}
```

۱۴-۳-۳-فعال سازی و غیرفعال سازی فرم

یک فرم در اولین بار نمایش یا زمانی که کاربر روی آن کلیک می‌کند یا از طریق کلید Alt+Tab به روی آن می‌رود فعال می‌شود و رویداد Activated فرم را رها می‌کند. برعکس، زمانی که فرم کانون را از دست می‌دهد (از طریق بستن یا خارج شدن از انتخاب)، رویداد DeActivate رخ می‌دهد. در قطعه کد بعدی، اداره کننده‌ی رویداد DeActivate، متن روی دکمه را به resume تغییر داده و آن را از کار می‌اندازد و کنترل کننده‌ی رویداد Activated دکمه را مجدداً به کار می‌اندازد.

```
this.Deactivate += new System.EventHandler(Form_Deactivate);
this.Activated += new System.EventHandler(Form_Activate);
//
void Form_Deactivate(object sender, EventArgs e)
{
    button\ .Enabled = false;
    button\ .Text = "Resume";
}
void Form_Activate(object sender, EventArgs e)
{
    button\ .Enabled = true;
}
```

بستن فرم

زمانی که یک فرم بسته می‌شود، رویداد Closing رخ می‌دهد و آخرین فرصت را برای انجام یک سری وظایف پاک‌سازی یا جلوگیری از بسته شدن فرم فراهم می‌سازد. این رویداد نماینده CancelEventHandler را برای احضار متدهای کنترل کننده‌ی رویداد به کار می‌برد. این نماینده یک پارامتر CancelEventArgs تعریف می‌کند که خصوصیت Cancel را در بر دارد. برای لغو کردن بسته شدن فرم، آن را true قرار دهید. در این مثال کاربر قبل از بستن فرم یک پیام اعلان می‌دارد.

```
this.Closing += new CancelEventHandler(Form_Closing);
void Form_Closing(object sender, CancelEventArgs e)
{
}
```

```

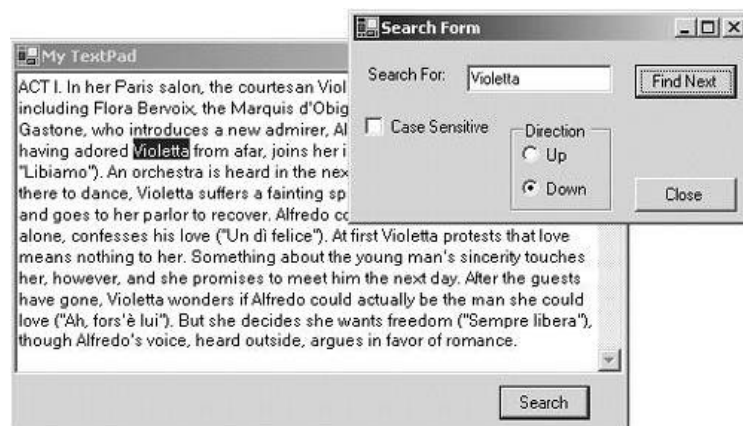
if(MessageBox.Show("Are you sure you want to Exit?", "", MessageBoxButtons.YesNo)
    == DialogResult.No)
{
    //Cancel the closing of the form
    e.Cancel = true;
}
}

```

۱۴-۳-۴ فعل و انفعال فرم‌ها – یک برنامه کاربردی نمونه

زمانی که چندین شی فرم ایجاد می‌شوند، باید یک روش برای دسترسی به حالت و محتوای کنترل‌های روی فرم دیگر وجود داشته باشد. آن اساساً به تنظیم معرف‌های دسترسی برای نمایش دادن فیلدها و خصوصیات روی هر فرم بستگی دارد. برای مثال، یک برنامه‌ی کاربردی شامل دو فرم `modeless` بسازید. (شکل ۱۴-۸). فرم اصلی دو کنترل دارد: یک `TextBox` که مستنداتی را نگه می‌دارد و یک دکمه `Search`، که هنگام کلیک روی آن فرم جستجو باز می‌شود. فرم جستجو یک `TextBox` دارد که متن مورد جستجو در سند فرم اصلی را می‌گیرد. بطور پیش‌فرض، عبارت مورد جستجو، متن روشن شده در سند فرم اصلی است. آن متن می‌تواند مستقیماً توسط کاربر وارد شود.

شکل ۱۴-۸



زمانی که دکمه‌ی `Next` را فشار دهید، برنامه رخدادهای بعدی رشته‌ی مورد جستجو را در سند اصلی جستجو می‌کند. اگر یک رخداد پیدا شود، آن روشن می‌شود. برای جالب‌تر شدن، روی فرم گزینه‌های برای جستجوی رو به جلو و رو به عقب و حساس به حروف نیز وجود دارد.

هدف اصلی توسعه‌ی این برنامه، نحوه در دسترس قرار دادن اطلاعات کنترل‌های روی یک فرم برای فرم‌های دیگر است. فرم اصلی باید محتوای `documentText` را برای فرم جستجو در اختیار قرار دهد تا آن بتواند یک رخداد رشته را روشن کند. فرم جستجو باید محتوای `txtSearch` را در اختیار فرم اصلی قرار دهد تا هر متن روشن شده را قبل از انتقال کنترل به فرم جستجو در آن قرار دهد.

`DocForm`، محتوای `documentText` را از طریق یک فیلد بنام `myText` به اشتراک می‌گذارد که زمان بارگذاری فرم به مقدار `documentText` انتساب داده می‌شود. تنظیم `myText` به صورت `public static` فرم جستجو را قادر می‌سازد به خصوصیات کادر متنی از طریق `DoForm.myText` دسترسی داشته باشد.

```

public static TextBox myText; //Declare public variable
private void docForm_Load(object sender, System.EventArgs e)
{
    myText = documentText;
}

```

`SearchForm` محتویات `txtSearch` را از طریق یک خصوصیت رشته‌ای فقط نوشتنی به اشیاء دیگر نمایان می‌کند.

```
public String SearchPhrase
{
    set { txtSearch.Text = value;} //Write Only
}
```

DocForm همانند هر شیئی که یک نمونه از SearchForm ایجاد می‌کند، می‌تواند این خصوصیت را مقداردهی می‌کند. حال به جزئیات باقیمانده در دو فرم توجه کنید.

کد فرم اصلی

زمانی که دکمه‌ی روی فرم DocForm کلیک می‌شود. برنامه کاربردی یک نمونه جدید از SearchForm را ایجاد می‌کند یا کنترل را به نمونه موجود رد می‌کند. در هر دو حالت، ابتدا کادر متنی را بررسی می‌کند و هر متن روشن شده (SelectedText) را به شی SearchForm یا به خصوصیت SearchPhrase رد می‌کند. (مثال ۱۴-۵) تکنیک‌های شرح داده شده در مثال‌های اخیر را برای ایجاد شی و تنظیم یک نماینده استفاده می‌کند تا شی DocForm را از بسته شدن فرم جستجو خبر دهد.

مثال ۱۴-۵

```
private void btnSearch_Click(object sender, System.EventArgs e)
{
    //Create instance of search form if it does not exist
    if (closed)
    {
        closed= false;
        searchForm = new SearchForm(); //Create instance
        searchForm.TopMost = true;
        searchForm.Closed += new EventHandler(onSearchClosed);
        searchForm.StartPosition = FormStartPosition.Manual;
        searchForm.DesktopLocation = new Point(this.Right-۲۰۰,this.Top-۲۰);
        searchForm.SearchPhrase = documentText.SelectedText;
        searchForm.Show();
    } else
    {
        searchForm.SearchPhrase = documentText.SelectedText;
        searchForm.Activate();
    }
}

private void onSearchClosed(object sender, System.EventArgs e)
{
    closed= true;
}
```

کد فرم جستجو

مثال ۱۴-۶ کد اجرا شده در زمان کلیک روی دکمه FindNext را نشان می‌دهد. جستجوی وقوع بعدی رشته مورد جستجو می‌تواند سند را با استفاده از متد LastIndexOf به بالا یا بوسیله متد IndexOf به پایین پیمایش کند. منطق برنامه می‌تواند حساسیت حروف را نادیده گیرد یا در نظر گیرد.

مثال ۱۴-۶

```
private void btnFind_Click(object sender, System.EventArgs e)
{
    int ib; //Index to indicate position of match
    string myBuff = DocForm.myText.Text; //Text box contents
    string searchText= this.txtSearch.Text; //Search string
    int ln = searchText.Length; //Length of search string
    if (ln>۰)
    {
        //Get current location of selected text
        int selStart = DocForm.myText.SelectionStart;
```



```

if (selStart >= DocForm.myText.Text.Length)
{
    ib = 0;
} else
{
    ib = selStart + 1;
}
if (!this.chkCase.Checked) //Case-insensitive search
{
    searchText = searchText.ToUpper();
    myBuff = myBuff.ToUpper();
}
if (this.radDown.Checked) ib = myBuff.IndexOf(searchText, ib);
if (this.radUp.Checked && ib > 1) ib = myBuff.LastIndexOf(searchText, ib - 1, ib - 1);
if (ib >= 0) //Highlight text on main form
{
    DocForm.myText.SelectionStart = ib;
    DocForm.myText.SelectionLength = txtSearch.Text.Length;
}
}
}
}

```

۱۴-۳-۵- فرم‌های مالک و ملک

زمانی که یک فرم نمونه‌ای از یک فرم `modeless` را نشان می‌دهد، بطور پیش‌فرض رابطه صریحی ما بین خود و فرم جدید ایجاد نمی‌کند، فرم‌ها بطور مستقل عمل می‌کنند. آنها می‌توانند بدون تأثیر روی فرم‌های دیگر بسته یا کوچک شوند. فرم سازنده، هیچ روشی برای تمایز مابین این فرم‌ها ندارد.

اغلب هر فرمی یک وابستگی روی فرم دیگر دارد. در مثال قبلی پنجره جستجوی شناور فقط به عنوان یک همراه برای سندی که آن جستجو می‌کند، وجود دارد. رابطه آن با فرم ایجادکننده‌اش رابطه مالک-ملک^۱ بیان می‌شود. در NET این عمل می‌تواند بالاتر از یک رابطه منطقی باشد. فرم خصوصیتی بنام `Owner` دارد که می‌تواند نمونه‌ای از فرم مالک آن قرار داده شود. بعد از برقراری این رابطه، رفتار فرم‌های مالک و ملک پیوند داده می‌شوند. برای مثال فرم ملک همواره در بالای فرم مالک نمایان است. این عمل نیاز به تنظیم `SearchForm` به عنوان یک فرم `TopMost` را در مثال قبلی حذف می‌کند.

یک رابطه مالک-ملک با تنظیم خصوصیت `Owner` یک فرم جدید ایجاد شده با فرم ایجاد کننده آن برقرار می‌گردد.

```

opaque opForm = new opaque();
opForm.Owner = this; //Current form now owns new form
opForm.Show();

```

این رابطه تعامل کاربر با فرم را به سه روش تحت تأثیر قرار می‌دهد. حتی اگر مالک فعال باشد، فرم ملک همواره بالای فرم مالک است. بستن فرم مالک، فرم ملک را می‌بندد و کوچک کردن فرم مالک، همه فرم‌های ملک را کوچک می‌کند و فقط یک آیکن روی نوار وظیفه می‌ماند.

مزیت دیگر اینکه یک فرم مالک، کلکسیونی بنام `OwnedForms` دارد که همه فرم‌های ملک ایجاد شده توسط آن را در برمی‌گیرد. مثال زیر نحوه ایجاد دو فرم ملک `opForm` و `opForm2` را برای یک مالک نشان می‌دهد و سپس قبل از نمایش آنها، کلکسیونی را برای تنظیم عنوان هر فرم طی می‌کند.

```

opaque opForm = new opaque();
opForm.Owner = this; //Set current form to owner form
opaque opForm2 = new opaque();
opForm2.Owner = this; //Set current form to owner form
for (int ndx=0; ndx<this.OwnedForms.Length; ndx++)
{
    myForms.Text = "Owner: Form\ - Form"+ndx.ToString();
}

```

^۱ Owner-owned

```
myForms.Show();
}
```

توجه داشته باشید، اگرچه فرم‌های Modal ویژگی‌های یک فرم ملک را ارائه می‌کنند، خصوصیت Owner بایستی صریحاً یک رابطه برقرار کند.

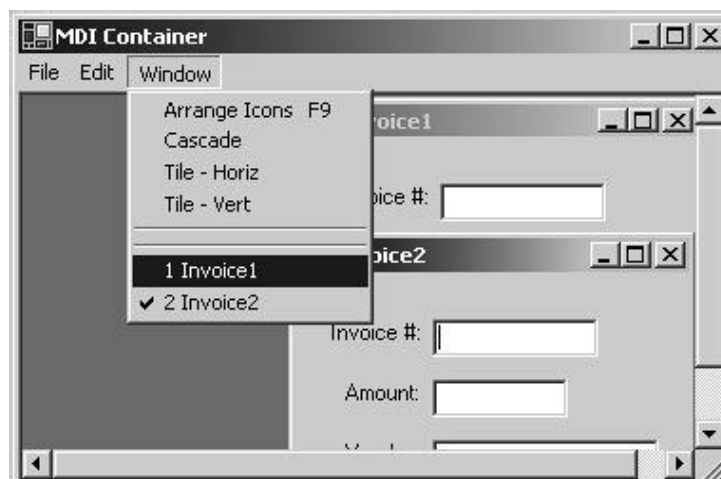
فرم‌های MDI

یک برنامه MDI با یک پنجره برنامه کاربردی و چند پنجره سند توصیف می‌گردد. از نظر ساختار، آن به عنوان یک ظرف واحد برای نگهداشتن چندین سند استفاده می‌شود. برای مدیریت کلکسیون سندها، برنامه MDI یک سیستم منو با گزینه‌های باز کردن، ذخیره کردن، بستن یک سند و سوئیچ مابین سندها، مرتب کردن سندها با چیدمان مختلف در بردارد.

هیچ کلاس خاصی برای ایجاد یک برنامه MDI نیاز نیست، فقط لازم است خصوصیت IsMdiContainer فرم true قرار داده شود. فرم‌های فرزند با تنظیم خصوصیت MdiParent به فرم اصلی معین می‌شوند.

فرم MDI شکل ۹-۱۴ سه عنصر نشان می‌دهد که یک فرم MDI در بردارد: ظرف پدر، فرم‌های فرزند و یک منو برای مدیریت ایجاد، انتخاب و مرتب کردن فرم‌های فرزند.

شکل ۹-۱۴



فرم ظرف MDI از طریق دستور زیر در سازنده‌ی آن ایجاد می‌شود.

```
this.IsMdiContainer = true;
```

از نظر سنتی، فرم‌های فرزند با انتخاب یک گزینه از منوی File- New همچون File-open یا File ایجاد می‌شوند. کد زیر یک نمونه از فرم فرزند ایجاد می‌کند و خصوصیت MdiParent آن را مقداردهی می‌کند.

```
invForm myForm = new invForm();
myForm.MdiParent = this;
mdiCt += mdiCt; //Count number of forms created
myForm.Text= "Invoice" + mdiCt.ToString();
myForm.Show();
```

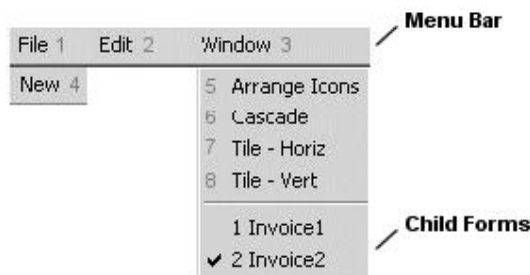
یک متغیر تعداد فرم‌های ایجاد شده را می‌شمارد و برای شناسایی منحصر فرد آن به خصوصیت Text هر فرم الحاق می‌کند.

ایجاد یک منو و فرم MDI

بحث فرم‌های MDI بدون بررسی منوهای مورد نیاز و مدیریت پنجره‌های ظرف ناتمام است. منوی فرم پدر MDI، حداقل یک بخش File برای ایجاد یا بازیابی فرم‌ها و یک بخش Windows برای لیست کردن همه فرم‌های فرزند و انتخاب یک فرم فرزند دارد.

منوی اصلی از دو کلاس ایجاد می‌شود: کلاس MainMenu به عنوان یک ظرف برای کل ساختار منو عمل می‌کند و کلاس MenuItem که اجزا منو را در منو نشان می‌دهد. هر دوی این کلاس‌ها، یک خصوصیت کلکسیون بنام MenuItem را نمایش می‌دهند که برای ایجاد سلسه مراتب منو بوسیله اضافه کردن اجزا منو استفاده می‌شود. بعد از اینکه اجزای منو مشخص شدند، مرحله بعدی ربط دادن آنها به روال‌های مناسب، اداره کردن رویداد با استفاده از روشی شبیه نماینده است. مثال مربوط به ایجاد کردن منوی نمایش داده شده در شکل ۱۴-۱۰ را دنبال کنید. بعداً ایجاد منو در VS.Net را بررسی خواهیم کرد. ایجاد منوها با استفاده از VS.Net سریعتر و ساده‌تر است، ولی انعطاف‌پذیری لازم در ایجاد منوها در مقایسه با زمان اجرا ندارد.

شکل ۱۴-۱۰



مرحله اول اعلان شی منوی اصلی و اجزا منو بصورت متغیرهای کلاس است.

```
private MainMenu mainMenu\;
private MenuItem menuItem\; //File
private MenuItem menuItem2; //Edit
private MenuItem menuItem3; //Window
private MenuItem menuItem4; //File - New
```

منوی اصلی و اجزا منو در داخل سازنده‌ی کلاس ایجاد می‌شوند.

```
this.mainMenu\ = new System.Windows.Forms.MainMenu();
this.menuItem\ = new System.Windows.Forms.MenuItem("File");
this.menuItem2 = new System.Windows.Forms.MenuItem("Edit");
this.menuItem3 = new System.Windows.Forms.MenuItem("Window");
this.menuItem4 = new System.Windows.Forms.MenuItem("New");
```

سپس سلسه مراتب منو با اضافه کردن اجزا منو به شی منوی اصلی برای ایجاد نوار منو بنا می‌شوند. اجزا نوار منو، اجزا اضافه شده به کلکسیون MenuItem هستند که منوی بازشو^۱ را ایجاد می‌کنند.

```
//Add menu items to main menu object
this.mainMenu\ .MenuItem.AddRange(new
System.Windows.Forms.MenuItem[] {
this.menuItem\,
this.menuItem2,
this.menuItem3});
//Add menu item below File
this.menuItem\ .MenuItem.Add(this.menuItem4);
//Add menu items to Window menu item
this.menuItem3.MdiList = true; //Causes child forms to display
this.menuItem3.MenuItem.AddRange(new System.Windows.Forms.MenuItem[]
{this.menuItem5, this.menuItem6, this.menuItem7, this.menuItem8});
//Set menu on form
this.Menu = this.mainMenu\;
```

نکات اصلی مورد توجه در این کد عبارتند از:

- متدهای Add و AddRange یک یا چندین منو را به کلکسیون MenuItem اضافه می‌کنند.

^۱ popup menu

- انتساب true به خصوصیت MdiList باعث می‌شود یک لیست از فرم‌های فرزند در زیر آن منو به عنوان یک زیر منو نمایش داده شوند. (Invoice و Invoice۱ در شکل ۱۴-۱۰ لیست می‌شوند).
- برای قرار دادن یک منو روی یک فرم، خصوصیت Menu می‌فرم را با شی MainMenu مقداردهی کنید.

گام نهایی، تنظیم کد اداره کردن رویدادها است که منطقی را برای پشتیبانی عملیات منوها فراهم می‌کند. این کد یک نماینده و یک متد برای پشتیبانی یک رویداد رها شده بوسیله‌ی کلیک روی File-New تعریف می‌کند. این کد هر زمان که روی این منو کلیک شود یک نمونه از invForm ایجاد می‌کند.

```
//Following is defined in constructor
MenuItem.Click += new System.EventHandler(menuItem.Click);
private void menuItem_Click(object sender, System.EventArgs e)
{
    invForm myForm = new invForm();
    myForm.MdiParent = this;
    mdiCt += mdiCt; //Count number of forms created
    myForm.Text= "Invoice" + mdiCt.ToString();
    myForm.Show();
}
```

گزینه‌ی Window روی نوار منو، زیر منویی دارد که به شما اجازه می‌دهد فرم‌های فرزند ظرف MDI را مجدداً مرتب کنید. متد LayoutMdi یک فرم ساده از این عمل را پیاده سازی می‌کند. بعد از تنظیم نماینده‌ها به روش معمول، روال‌های اداره کردن رویداد را ایجاد کنید.

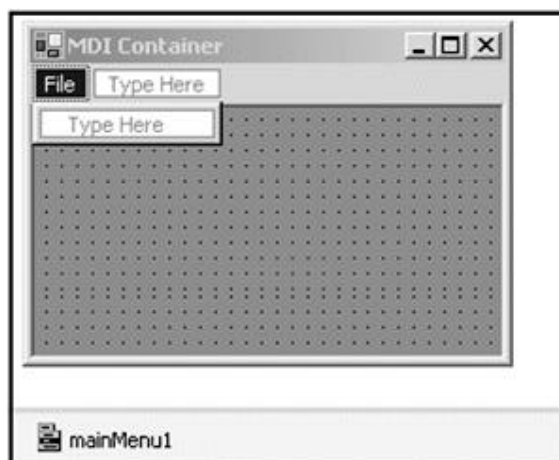
```
private void menuItem_Click(object sender, System.EventArgs e)
{
    this.LayoutMdi(MdiLayout.ArrangeIcons);
}
private void menuItem_Click(object sender, System.EventArgs e)
{
    this.LayoutMdi(MdiLayout.Cascade);
}
private void menuItemv_Click(object sender, System.EventArgs e)
{
    this.LayoutMdi(MdiLayout.TileHorizontal);
}
private void menuItem^_Click(object sender, System.EventArgs e)
{
    this.LayoutMdi(MdiLayout.TileVertical);
}
```

متدها با رد کردن مقدار شمارشی مناسب MdiLayout به متد LayoutMdi مجدداً فرم‌های فرزند را مرتب می‌کنند.

ایجاد یک منوی MDI با استفاده از VS.NET

روی آیکون MainMenu از پنجره Toolbox دابل کلیک کنید. دو چیز اتفاق می‌افتد: یک آیکون در قسمت قطعه‌ها ظاهر می‌گردد و یک الگوی منوی لنگر انداخته به بالای فرم ظاهر می‌گردد. عناوین منو را به سلولهای ظاهر شده تایپ کنید(شکل ۱۴-۱۱). ردیف افقی بالای سلولها، نوار منو را نمایش می‌دهد. با حرکت عمودی، می‌توانید منوهای باز شو در زیر منوی سطح بالا را ایجاد کنید. بعد از تایپ کردن نام منو، روی سلول دابل کلیک کنید تا یک رویداد Click برای منو ایجاد شود. VS.NET نماینده و بدنه‌ی متد را بطور اتوماتیک ایجاد می‌کند.

شکل ۱۴-۱۱



پنجره Properties را برای نمایش خصوصیات سلول فعال بکار برید، تا بتوانید اسامی پیش فرض انتساب داده شده به منوها را تغییر دهید.

۱۴-۴-کار با منوها

بخش قبلی، یک مقدمه‌ی قوی در مورد منوها فراهم ساخت. این بخش یک لیست از خصوصیات موثر بر ظاهر منو و نحوه استفاده از کلاس ContextMenu را شرح می‌دهد.

۱۴-۴-۱-خصوصیات MenuItem

سیستم منوی .NET با فلسفه سودمندگرایی طراحی شده است. منو برای قشنگی نیست، بلکه برای انجام کار است. حال چند تا از خصوصیات مفید آنرا می‌آوریم.

Enabled: مقداردهی آن با true، دکمه‌ی منو را کدر کرده و آن را غیر قابل دسترس می‌سازد.

Checked: یک علامت در کنار متن منو قرار می‌دهد.

RadioCheck: یک دکمه رادیویی کنار متن منو قرار می‌دهد. مقدار Checked باید true باشد.

BreakBar یا Break: مقداردهی آن با true باعث می‌شود منو در یک ستون جدید قرار گیرد.

Shortcut: یک کلید میان‌بر از اعضای نوع شمارشی Shortcut تعریف می‌کند. این اعضا باعث می‌شوند که زمان فشار دادن این کلیدهای ترکیبی، منوی مورد نظر انتخاب شود. با قرار دادن علامت & قبل از یک حرف در متن منو، آن حرف زیر خطدار شده و برای فعال کردن آن منو کافی است ترکیب کلید Alt با آن حرف را به کار برید.

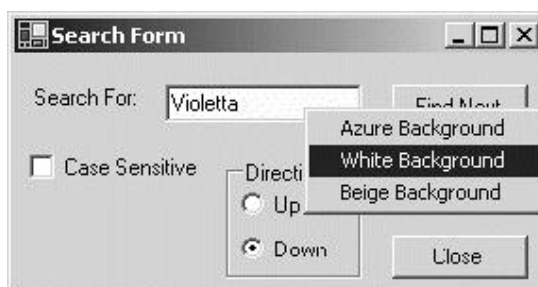
۱۴-۴-۲-منوهای زمینه

علاوه بر کلاس‌های MainMenu و MenuItem که بحث شده‌اند، یک کلاس بنام ContextMenu وجود دارد که از کلاس Menu ارث‌بری می‌کند. کلاس ContextMenu به تک تک کنترل‌های تخصیص داده می‌شود و اغلب زمان فشار دادن کلیک راست روی یک کنترل، یک منوی popup باز می‌شود.

دستورات ساخت یک منو بر اساس ContextMenu همانند MainMenu است. تنها تفاوت این است که از نظر بصری نوار منوی سطح بالا وجود ندارد و منو در نزدیک کنترلی که احضار کرده‌ایم ظاهر می‌شود.

یک منو می‌تواند به چندین کنترل مرتبط شود یا هر کنترل می‌تواند منوی مختص خود را داشته باشد. برای مثال، ممکن است یک منوی زمینه برای همه کنترل‌های TextBox و منوی دیگر برای همه دکمه‌ها داشته باشیم. برای فهم بیشتر، یک منو که پشت زمینه‌ی کنترل TextBox را رنگی می‌کند، ایجاد کنید. (شکل ۱۴-۱۲)

شکل ۱۴-۱۲



۱۴-۴-۳- ساختن یک منوی زمینه^۱

ایجاد یک منوی زمینه، شبیه ایجاد یک MainMenu است. اگر از VS.NET استفاده می‌کنید، کنترل ContextMenu را روی یک فرم کشیده و بطور بصری زیر منوها را اضافه کنید. اگر بصورت دستی کدنویسی می‌کنید، یک نمونه از کلاس ContextMenu ایجاد کرده و با استفاده از متد MenuItems.Add منوها را به آن اضافه کنید. کد زیر نمونه‌ای از کد کاربر برای ایجاد منو است. توجه کنید که فقط یک متد، همه رویدادهای کلیک روی هر منو را اداره می‌کند.

```
private ContextMenu contextMenu; //Context menu
private TextBox txtSearch; //Text box that will use menu
// Following is in constructor
contextMenu = new ContextMenu();
// Add menu items and event handler using Add method
contextMenu.Items.Add("Azure Background", new
    System.EventHandler(this.menuItem_Click));
contextMenu.Items.Add("White Background", new
    System.EventHandler(this.menuItem_Click));
contextMenu.Items.Add("Beige Background", new
    System.EventHandler(this.menuItem_Click));
```

منوی کامل شده، بوسیله‌ی تنظیم خصوصیت ContextMenu یک کنترل به آن کنترل نسبت داده می‌شود.

```
this.txtSearch.ContextMenu = this.contextMenu;
```

کلیک راست روی txtSearch باعث می‌شود، منو ظاهر گردد و با کلیک بر روی یکی از زیر منوها روال اداره کردن رویداد فراخوانی می‌شود:

```
private void menuItem_Click(object sender, System.EventArgs e)
{
    //Sender identifies specific menu item selected
    MenuItem conMi = (MenuItem) sender;
    string txt = conMi.Text;
    //SourceControl is control associated with this event
    if(txt == "Azure Background")
        this.contextMenu.SourceControl.BackColor = Color.Azure;
    if(txt == "White Background")
        this.contextMenu.SourceControl.BackColor = Color.White;
    if(txt == "Beige Background")
        this.contextMenu.SourceControl.BackColor = Color.Beige;
}
```

^۱ context menu

دو چیز مهم مورد توجه در این مثال، آرگومان `sender` که منوی انتخاب شده را معرفی می‌کند و خصوصیت `SourceControl` که منوی زمینه کنترل تخصیص یافته به این رویداد را معرفی می‌کند. توانایی معرفی کنترل و زیر منو، یک متد را قادر می‌سازد تا رویدادها را از هر کنترل روی فرم یا هر زیر منو در منوی زمینه اداره کند.

۱۴-۵- اضافه کردن کمک به یک فرم

اکثریت کاربران نرم افزار مستندات را نمی‌خوانند. کاربران انتظار دارند برنامه مبنی بر درک^۱ باشد و مستندسازی حساس به متن را در هر جا که نیاز است فراهم کند. علاوه بر گزینه‌ی `Help` روی نوار منو، یک برنامه آرایش شده، باید یک دستیار برای کنترل‌های روی فرم فراهم سازد. `NET` چندین راه برای پی‌کربندی یک سیستم کمکی مجتمع پیشنهاد می‌کند.

- کاربرد آسان `ToolTip` که زمان انتقال ماوس روی یک کنترل ظاهر می‌شود. اینها بصورت یک خصوصیت بوسیله کنترل `ToolTip` مشخص می‌شوند.

- کنترل `HelpProvider` یک بسط دهنده است که خصوصیتی را به کنترل‌های موجود اضافه می‌کند. این خصوصیات کنترل را قادر می‌سازند به فایل‌های `CHM` ویندوز ارجاع کنند.

- یک اداره کننده سفارشی رویداد می‌تواند برای پیاده‌سازی کدی استفاده شود که بطور صریح رویداد `HelpRequested` `Control` را با استفاده از دکمه `Help` را اداره می‌کند. رها شده با فشار دادن کلید `F1` یا با استفاده از دکمه `Help` را اداره می‌کند.

۱۴-۵-۱- `ToolTip` ها

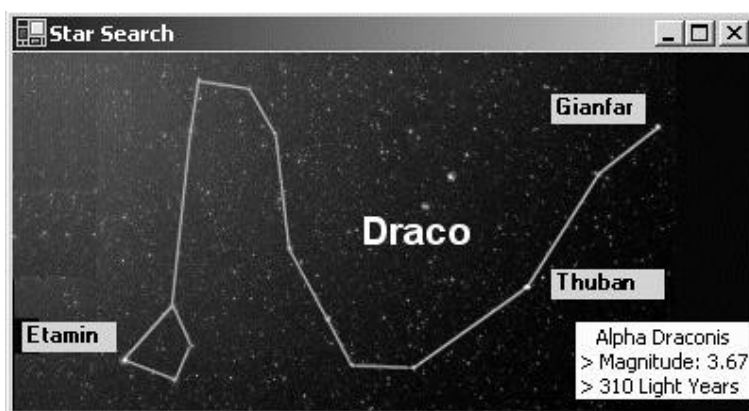
اگر `VS.NET` را بکار می‌برید، کنترل `ToolTip` را از کادر ابزار انتخاب کنید و آنرا به فرم‌تان اضافه کنید. اثر آن اضافه کردن یک خصوصیت `(string ToolTip on toolTip1)` روی هر کنترل است. در زمان قرار گرفتن ماوس روی هر کنترل مقدار آن نمایش داده می‌شود.

جالب‌تر از همه، کاربرد `ToolTip` در یک برنامه، حاشیه‌نویسی را بطور اتوماتیک برای اشیا روی صفحه نمایش یا نقشه‌ها فراهم می‌کند که بطور پویا در جواب به درخواست‌های کاربر ایجاد می‌شود. آنها نقاط جالب را بصورت برجسب‌ها یا کادرهای تصویری در بردارند. به عنوان یک مثال، تصویر شکل ۱۴-۱۳ را ملاحظه کنید. یک صورت فلکی و برجسب‌هایی برای مهمترین ستاره‌ها نشان می‌دهد. زمانی که یک کاربر مکان‌نما را روی برجسب می‌برد، متن `ToolTip`ی که ستاره را شرح می‌دهد، نمایش داده می‌شود.

مثال ۱۴-۷ یک بخش از کد را نشان می‌دهد که فرم نمایش داده شده در شکل ۱۴-۱۳ را ایجاد می‌کند. خصوصیت `BackgroundImage`، تصویری که صورت فلکی را نشان می‌دهد قرار داده می‌شود. متناسب با موقعیت سه ستاره (که فقط یک ستاره نشان داده می‌شود)، برجسب‌ها روی آن قرار داده می‌شود. خصوصیت `Tag` هر برجسب، با توصیف ستاره مقداردهی می‌شود و یک `ToolTip` این اطلاعات برجسب را با استفاده از متد `SetToolTip` بدست می‌آورد.

شکل ۱۴-۱۳

^۱ intuitive



مثال ۱۴-۷

```
public class StarMap:Form
{
public StarMap()
{
this.Text = "Star Search";
this.Width=۴۰۰;
this.Height=۲۲۰;
// Place image of constellation on form

this.BackgroundImage= new Bitmap(@"c:\dracoblack.gif");
// Add name of star on Label
Label star\ = new Label();
Star\.Location = new Point(۲۸۰,۱۱۰);
Star\.Size = new Size(۱۰,۱۱);
star\.Text = "Thuban";
star\.Tag = " Alpha Draconis\n> Magnitude: ۳,۶۷\n>"+ " ۳۱۰ Light Years";
star\.Font = new Font(star.Font, star.Font.Style |FontStyle.Bold);
this.Controls.Add(star\);
ToolTip tooltip\ = new ToolTip();
tooltip\.AutoPopDelay= ۱۰۰۰; // Tool tip displays
// for ۱,۰ secs.
// Tool tip text comes from Tag property of Label
tooltip\.SetToolTip(star\, star\.Tag.ToString());
// Add labels for other stars Etamin and Gianfar here ...
}
}
```

توجه: برای تغییر مقدار ToolTip یک کنترل، باید یک نمونه از ToolTip کنترل را گرفته و متد RemoveAll آن را اجرا کنید و سپس متد SetToolTip را برای مقداردهی مجدد رشته ToolTip بکار ببرید.

۱۴-۵-۲- پاسخ به ۱F و دکمه Help

کاربران زیادی به کلید ۱F به عنوان یک روش بالفعل برای احضار کمک اعتنا می کنند. NET. پشتیبانی داخلی ۱F را با رهاکردن رویداد Control.HelpRequested هنگام فشار دادن ۱F توسط کاربر فراهم می کند. همچنین هنگامی که کاربر روی دکمه Help در بالای فرم کلیک کرده و سپس با استفاده از مکان نمای Help روی یک کنترل کلیک می کند رها می سازد. شکل ۱۴-۱۴ را ببینید.

دکمه ی Help با تنظیم خصوصیات زیر نمایش داده می شود.

- MinimizeBox و MaximizeBox را false قرار دهید.

- HelpButton را true قرار دهید.

شکل ۱۴-۱۴



یک روش توصیه شده، ایجاد یک روال اداره کننده رویداد است و هر کنترلی آن را احضار می‌کند. به عنوان مثال، کد زیر نماینده‌هایی را برای دو کادر متنی تعریف می‌کند که زمان وقوع رویداد HelpRequested متد ShowHelp را آگاه می‌سازند. این متد، خصوصیت Tag تخصیص یافته به هر کنترل یا نام کنترل را جهت تعیین کمک مربوط به آن کنترل بکار می‌برد.

```
this.date.HelpRequested += new HelpEventHandler(ShowHelp);
this.ssn.HelpRequested += new HelpEventHandler(ShowHelp);
this.ssn.Tag = "Enter as: nnn-nn-nnnn";
this.date.Tag = "Enter as: mm/dd/yyyy";
private void ShowHelp(object sender, HelpEventArgs e)
{
    Control reqControl = (Control)sender;
    // Technique ۱: Use tag associated with control
    MessageBox.Show(reqControl.Tag.ToString());
    // Technique ۲: Link to specific text within a CHM file
    string anchor = "#" + reqControl.Name;
    // ShowHelp invokes a compiled Help file
    Help.ShowHelp(reqControl, @"c:\ctest.chm", HelpNavigator.Topic, "customers.htm"
        + anchor);
    e.Handled = true; // Always set this
}
```

اداره کننده‌ی رویداد، دو آرگومان دریافت می‌کند: sender که کنترل کانون‌دار را در زمان وقوع رویداد تعیین می‌کند و HelpEventArgs فقط دو خصوصیت Handled و MousePos را دارد. خصوصیت Handled می‌تواند برای نشان دادن رویدادی که اداره شده است استفاده شود. MousePos یک نوع داده‌ی Point است که موقعیت ماوس را روی فرم مشخص می‌کند. این متد با تشخیص کنترل فعال و استفاده از این دانش جهت انتخاب متن Help، اطلاعات کمکی را فراهم می‌سازد. در این مثال، اولین تکنیک، خصوصیت Tag یک کنترل را به عنوان پیام Help نمایش می‌دهد. دومین و جالبترین تکنیک متد Help.ShowHelp را برای نمایش بخشی از یک فایل HTML بکار می‌برد که نام کنترل را استفاده می‌کند. بویژه در داخل فایل Ctest.chm، صفحه Customers.htm را جستجو می‌کند. سپس آن صفحه را برای یک برچسب نام‌دار همچون a > <Name=sss جستجو می‌کند. اگر پیدا شود، HTML را در آن موقعیت نمایش می‌دهد.

متد ShowHelp مفیدترین متد کلاس Help است. آن چندین OverLoad را برای نمایش فایل‌های Help کامپایل شده یا فایل‌های HTML در یک فرمت HTML دارد.

```
// URL may be .chm file or html file
public static void ShowHelp(Control parent, string url);
// HelpNavigator defines the type of .chm file to be displayed
public static void ShowHelp(Control parent, string url, HelpNavigator navigator);
// Displays contents of Help file for a specified keyword
public static void ShowHelp(Control parent, string url, string keyword);
// param is used with HelpNavigator.Topic to refine selection
public static void ShowHelp(Control parent, string url, HelpNavigator navigator,
    object param);
```

نوع شمارشی HelpNavigator مشخص می‌کند، کدام بخش از یک فایل CHM نمایش داده شود. اعضای آن Index، Find، TableofContents و Topic هستند. اگر با این مفاهیم آشنا نیستند، فایل‌های Help کامپایل شده، چندین فایل HTML را با یک جدول اختیاری از محتویات و اندیس‌های کلمه کلیدی بسته‌بندی می‌کنند. نرم افزار Microsoft HTML Help Workshop ساده‌ترین روش برای یادگیری نحوه استفاده و ایجاد این فایل‌ها است.

۱۴-۵-۳- کنترل HelpProvider

این کنترل با VS.NET استفاده می‌شود. ارزش اصلی آن حذف نیاز به اداره کردن صریح رویداد HelpRequested است. آن یک بسط دهنده است که چندین خصوصیت به هر کنترل اضافه می‌کند. این خصوصیات با پارامترهای متد ShowHelp متناسب هستند که آنرا فراخوانی می‌کنند. با انتخاب HelpProvider از کادر ابزار، آنرا به فرم اضافه کنید. سپس خصوصیت HelpNameSpace آنرا نام فایل HTML یا CHM قرار دهید که متد به آن ارجاع خواهد کرد. ۴ خصوصیت بسط‌افته به هر کنترل روی فرم اضافه می‌ند.

۱- ShowHelp: برای فعال‌ردن ویژگی Help آن را true قرار دهید.

۲- HelpNavigator: مقدار شمارشی HelpNavigator را می‌گیرد.

۳- HelpKeyword: به پارامتر keyword یا params در متد ShowHelp مربوط می‌شود.

۴- HelpString: زمانی که دکمه Help برای کلیک روی یک کنترل استفاده می‌شود، متن این پیام نمایش داده می‌شود.

Help برای کنترلی که ShowHelp آن false است، فعال نمی‌شود. اگر آن true قرار داده شود، اما بقیه خصوصیات آن تنظیم نشوند، فایل ارجاع داده شده در HelpNameSpace نمایش داده می‌شود. پیکربندی عمومی Help فقط تنظیم مقدار HelpString است که دکمه Help یک پیام خاص کوتاه را نشان می‌دهد و F1 یک صفحه HTML را باز می‌کند.

۱۴-۶- وراثت فرم‌ها

همانند وراثت یک کلاس از کلاس پایه، یک فرم GUI می‌تواند تنظیمات، خصوصیات و چارچوب کنترل یک فرم موجود را به ارث برد. بدین معنی که می‌توانید فرم‌هایی را با ویژگی‌های استاندارد جهت بکار رفتن به عنوان الگوهایی برای فرم‌های مشتق شده ایجاد کنید. قبل از نگاه کردن به جزئیات وراثت فرم‌ها، ابتدا نحوه ذخیره یک مجموعه از فرم‌های پایه را در یک کتابخانه کد و سازمان‌دهی آنها بوسیله فضای نامی را بررسی می‌کنیم.

۱۴-۶-۱- ایجاد و استفاده یک کتابخانه از فرم‌ها

هر فرم یک فایل فیزیکی CS دارد. یک کتابخانه از چندین فرم، بوسیله کامپایل کردن هر فایل CS به داخل یک فایل مشترک dll ایجاد می‌شود. بعد از این کار، فرم‌ها می‌توانند بوسیله هر زبان مطیع دستیابی شوند.

به عنوان مثال: کامپایلر را از خط فرمان برای کامپایل کردن دو فرم به داخل یک فایل dll واحد بکار می‌بریم.

```
csc /t:library product.cs customer.cs /out:ADCFormLib.dll
```

یک فرم پایه باید یک فضای نامی برای فرم مشتق شده فراهم کند تا آن را ارجاع کند. کد زیر یک فضای اسمی Products برای مثال ما تعریف می‌کند.

```
namespace Products
{
    public class ProductForm : System.Windows.Forms.Form
    {
```

برای وراثت از این فرم، یک کلاس گرامر استاندارد وراثت را بکار می‌برد و کلاس پایه را بوسیله فضای نامی آن و نام کلاس معین می‌کند.

```
// User Form derived from base class Form
class NewProductForm: Products.ProductForm
{
```

در مرحله نهایی، کامپایلر باید یک ارجاع به اسمبلی خارجی ADCFormLib بدهد تا بتواند کلاس پایه را بیابد. اگر VS.NET را بکار می‌برید از منوی Project گزینه Add Reference را برای تعیین اسمبلی استفاده کنید و در خط فرمان پرچم reference استفاده می‌شود.

```
csc /t:winexe /r:ADCFormLib.dll myApp.cs
```

۱۴-۶-۲- کاربرد فرم ارث‌بری شده

اگر فرم مشتق شده هیچ کد اضافی فراهم نکند، در زمان اجرا یک فرم مشابه فرم اصلی تولید می‌کند. البته، فرم مشتق شده برای اضافه کردن کنترل‌ها و کد پشتیبان آزاد است. تنها محدودیت این است که به یک منوی موجود نمی‌توان چیزی اضافه کرد. با این وجود یک منوی کامل می‌تواند به فرم اضافه شود یا جایگزین منوی فرم اصلی شود.

خصوصیات کنترل‌های ارث‌بری شده می‌توانند تغییر یابند، اما معرف دسترسی پیش‌فرض آنها باید از private به protected تغییر یابد و فرم اصلی مجدداً کامپایل شود. فرم مشتق شده برای ایجاد تغییرات آزاد است. آن ممکن است موقعیت کنترل‌ها را تغییر دهد، حتی خصوصیت Visible کنترل‌ها را false قرار دهد تا نمایش داده نشوند.

۳۶-۶-۳- Override کردن رویدادها

فرض کنید فرم پایه یک دکمه دارد که به رویداد کلیک با فراخوانی کد اداره کننده رویداد جهت بستن فرم پاسخ می‌دهد. با این وجود، در فرم مشتق شده، می‌خواهید کنترل بازبینی داده‌ها را قبل از بستن فرم اضافه کنید. کار طبیعی اضافه کردن یک نماینده و متد اداره کننده رویداد برای پاسخ دادن به رویداد Click دکمه در فرم مشتق شده است. با این وجود، این عمل، اداره کننده رویداد اصلی در فرم پایه را Override نمی‌کند و هر دو روال اداره کردن رویداد فراخوانی می‌شوند.

تغییر ساختار اداره کننده اصلی رویداد برای فراخوانی یک متد مجازی راه‌حل مناسبی است که در فرم مشتق شده می‌تواند Override شود. این مثالی از کد کلاس پایه است.

```
private void btn\_Clicked(object sender, System.EventArgs e)
{
    ButtonClicks(); // Have virtual method do actual work
}
protected virtual void ButtonClicks()
{
    this.Close();
}
```

فرم مشتق شده، متد مجازی را Override می‌کند و کد خود را برای اداره کردن رویداد دارد.

```
protected override void ButtonClicks()
{
    // Code to perform any data validation
    this.Close();
}
```

۱۴-۶-۴- ایجاد فرم‌های ارث‌بری شده با VS .NET

برای ایجاد یک فرم ارث‌بری شده با استفاده از VS.NET، یک پروژه شامل فرم پایه خود را باز کرده و آن را کامپایل کنید. سپس منوی Project - Add Inherited Form را انتخاب کنید. یک نام جدید به آن داده و آن را باز کنید. سپس یک کادر محاوره‌ای Inheritance Picker ظاهر می‌گردد که فرم‌های پایه مطلوب را لیست می‌کند. Browse را برای نمایش فرم‌هایی که در کتابخانه‌های خارجی هستند بکار ببرید.

۱۴-۷- خلاصه

- برخلاف مهاجرت به سمت برنامه‌های کاربردی تحت اینترنت، هنوز نیاز ضروری به برنامه‌نویسی ویندوز داریم.
- فرم‌های ویندوز ویژگی‌ها و عملکردهای برتر از فرم‌های وب فراهم می‌کنند. اکثریت برنامه‌های دنیای واقعی روی شبکه‌های محلی اجرا می‌شوند. NET FCL. یک مجموعه‌ی گرانبها از کلاس‌ها برای پشتیبانی برنامه‌نویسی فرم‌ها فراهم می‌کند.
- کلاس Control در بالای سلسله مراتب، خصوصیات، متدها و رویدادهایی را فراهم می‌کند تا اجازه دهد کنترل‌ها روی فرم‌ها، موقعیت‌یابی و دستکاری شوند.
- رویدادهای صفحه کلید و ماوس یک برنامه را برای تشخیص هر کلیدی یا کلیک دکمه‌های ماوس قادر می‌سازند.
- کلاس Form همه اعضای کلاس Control را به ارث می‌برد و خصوصیت‌هایی به آن برای تعیین ظاهر، موقعیت و رابطه با فرم‌های دیگر اضافه می‌کند.
- یک فرم ایجاد شده با فرم دیگر، می‌تواند modal باشد، یعنی تا زمانی که باز است کانون را از دست نمی‌دهد یا می‌تواند modeless باشد که کانون می‌تواند به هر فرمی جابجا شود.
- در یک برنامه‌ی MDI، یک فرم به عنوان یک ظرف برای نگه‌داشتن فرم‌های فرزند به خدمت گرفته می‌شود. ظرف یک منو برای انتخاب کردن یا مرتب کردن مجدد فرم‌های فرزند فراهم می‌کند.
- زمانی که کاربر کلید F۱ را فشار می‌دهد، NET. یک رویداد HelpRequested را می‌سازد. این می‌تواند با متد Help ترکیب شود که فایل‌های (.chm) (HTML) کامپایل شده را پشتیبانی می‌کند تا یک توسعه دهنده را برای فراهم کردن اطلاعات کمکی روی فرم قادر سازد.

فصل پانزدهم

کنترل‌های فرم ویندوز

آنچه که در این فصل یاد خواهید گرفت:

- مقدمه : یک دیاگرام سلسله مراتب کلاس، روش مناسبی برای گروه‌بندی کنترل‌های فرم بر اساس عملکرد آن پیشنهاد می‌کند.
- کنترل‌های دکمه: کنترل‌های Button، CheckBox و RadioButton طراحی شده‌اند تا برای کاربران امکان یک یا چند انتخاب را فراهم سازند.
- کنترل‌های کادر متنی و کادر عکس: کنترل PictureBox برای نمایش و مقیاس‌بندی تصاویر استفاده می‌شود. کنترل TextBox برای نمایش و ویرایش یک یا چند خط از متن بکار برده می‌شود.
- کنترل لیست: کنترل‌های ListBox، ComboBox و CheckListBox واسط‌های مختلفی برای نمایش و دستکاری داده در یک لیست پیشنهاد می‌کنند.
- کنترل ListView و TreeView: کنترل ListView چندین نما برای نمایش اقلام داده و آیکن‌های انتساب شده به آنها ارائه می‌کند. کنترل TreeView اطلاعات سلسله مراتبی را در یک ساختار درختی قابل پی‌گیری ارائه می‌کند.
- کنترل Timer و ProgressBar: یک تایمر برای کنترل زمانی که باید یک رویداد احضار شود، استفاده می‌شود. یک ProgressBar پیشرفت یک عمل را بصورت بصری نمایش می‌دهد.
- ایجاد یک کنترل کاربری: زمانی که هیچ کنترلی نیاز کاربر را برآورده نمی‌کند، می‌توان با استفاده از ترکیب چند کنترل یا اضافه کردن ویژگی‌هایی به یک کنترل، کنترل سفارشی ایجاد کرد.
- انتقال داده‌ها مابین کنترل‌ها : یک روش ساده برای کپی کردن یا انتقال قلم داده‌ای از یک کنترل به یک کنترل دیگر، کشیدن و رها کردن می‌باشد. Net تعداد متنوعی از کلاس‌ها و رویدادها برای پیاده‌سازی این ویژگی فراهم می‌سازد.
- کاربرد منابع: منابع مورد نیاز یک برنامه همچون عنوان، برچسب‌های توصیفی و تصاویر می‌توانند در یک اسمبلی محلی یا سراسری تعبیه شوند. این ویژگی مخصوصاً برای توسعه برنامه‌های کاربردی بین‌المللی مفید هستند.

در فصل قبلی کلاس Control، متدها، خصوصیات و رویدادهای آن بررسی شد. این فصل ویژگی‌های تک به تک کنترل‌ها را بررسی می‌کند.

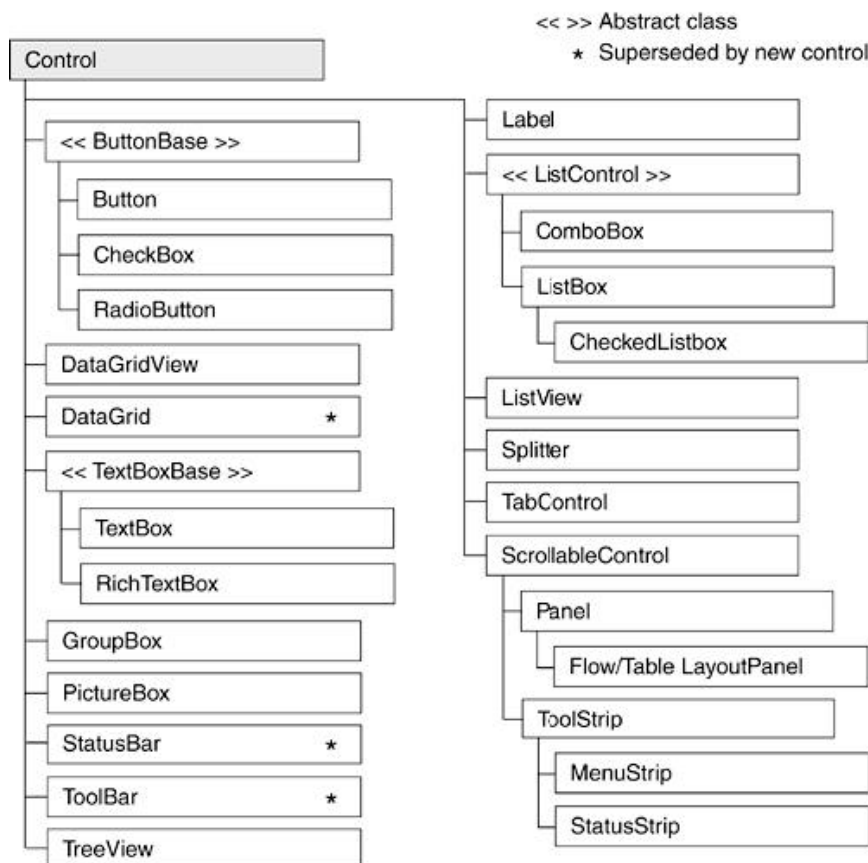
فرم‌های ویندوز فقط برای کاربرد کنترل‌های درونی استاندارد محدود نشده‌اند. می‌توان کنترل‌های GUI سفارشی برای توسعه کنترل‌های موجود ایجاد کرد. می‌توان یک کنترل کاملاً جدید یا کنترلی بر اساس کنترل‌های موجود ایجاد کرد. چند مثال این فصل، نحوه توسعه یک کنترل و ایجاد یک کنترل کاربری را ارائه می‌کنند. این فصل نگاهی بر فایل‌های منبع و نحوه استفاده از آنها در تولید برنامه‌های کاربردی GUI جهت پشتیبانی کاربران کشورها و فرهنگ‌های مختلف دارد.

۱۵-۱- مطالعه کنترل‌های NET. فرم‌های ویندوز

فضای نامی System.Windows.Forms یک خانواده‌ی بزرگ از کنترل‌ها در بر دارد که به واسطه کاربری مبتنی بر ویندوز، شکل و توانایی عمل می‌دهد. هر کنترلی یک مجموعه مشترک از اعضای کلاس Control را به ارث می‌برد. سپس متدها، خصوصیات و رویدادهایی به آن کنترل اضافه می‌کند تا ظاهر و رفتار متمایز خود را پیدا کند.

شکل ۱-۱۵ سلسله مراتب ارث‌بری کنترل‌های فرم‌های ویندوز را نشان می‌دهد. کنترل‌های علامت‌گذاری شده بوسیله * برای فراهم کردن سازگاری مابین .NET ۲.۰ و .NET ۱.۰ فراهم شده‌اند. مخصوصاً DataGridView با DataGrid، StatusBar با StatusStrip و Toolbar با ToolStrip جانشین شده است. جدول ۱-۱۵ خلاصه‌ای از کنترل‌های معمول این سلسله مراتب را فراهم می‌کند.

شکل ۱-۱۵



جدول ۱-۱۵

توصیف

کاربرد

کنترل

زمانی که یک کلیک ماوس رخ می‌دهد یا در زمان فشار دادن کلید Enter یا ESC یک رویداد آزاد می‌کند.	Button	یک دکمه روی فرم نشان می‌دهد. خصوصیت Text آن عنوان روی دکمه را تعیین می‌کند
به کاربر اجازه می‌دهد یک یا چند گزینه را انتخاب کند.	CheckBox	یک کادر انتخاب به همراه یک متن یا تصویر در کنار آن دارد. می‌توان کادر انتخاب را به صورت یک دکمه نشان داد. <code>CheckBox.Appearance = Appearance.Button</code>
لیستی از اقلام را نشان می‌دهد.	CheckedListBox	ListBox به همراه یک کادر انتخاب در قبل از هر قلم داده
عملکرد TextBox و ListBox را با هم فراهم می‌سازد.	ComboBox	کنترل ترکیبی که یک کادر متنی در یک لیست بازشو دربر دارد. خصوصیات دو کنترل TextBox و ListBox را ترکیب می‌کند.
داده‌ها را در یک قالب شبکه‌ای دستکاری می‌کند	DataGridView GridView	DataGridView بهترین کنترل برای نمایش داده‌های رابطه‌ای است. آن به پایگاه داده نیز مقید می‌شود. در ۲۰۰۷ Net آمده است و جانشین DataGridView است.
کنترل‌ها را گروه‌بندی می‌کند	GroupBox	اصولاً برای گروه‌بندی دکمه‌های رادیویی بکار می‌رود. آن یک حاشیه در اطراف کنترل‌های دربرگیرنده قرار می‌دهد.
یک کلکسیون از تصاویر را مدیریت می‌کند	ImageList	این کنترل ظرف یک کلکسیون از تصاویر را نگه می‌دارد. این تصاویر توسط کنترل‌هایی همچون ToolStrip, ListView و TreeView استفاده می‌شود.
اطلاعات توصیفی به فرم اضافه می‌کند	Label	متونی که محتویات یک کنترل یا نحوه‌ی استفاده از یک کنترل را شرح می‌دهند.
یک لیست از اقلام را نشان می‌دهد. ممکن است یک یا چند تا از آنها انتخاب شوند.	ListBox	ممکن است متون یا اشیاء ساده را در بر گیرد. متدها، خصوصیات و رویدادهای آن، عملیات انتخاب، تغییر، اضافه کردن و مرتب‌سازی اقلام را مجاز می‌دارند.
اقلام داده و زیر اقلام داده را نمایش می‌دهد.	ListView	آن می‌تواند یک قالب شبکه‌ای داشته باشد که هر سطری یک قلم داده مختلف و زیر قلم داده‌های آن را نمایش می‌دهد. امکان نمایش قلم داده‌ها بصورت آیکون وجود دارد.
یک منو به یک فرم اضافه می‌کند.	MenuStrip	یک سیستم منو و زیرمنو برای یک فرم فراهم می‌کند. آن جانشین کنترل MainMenu است.
کنترل‌ها را گروه‌بندی می‌کند	Panel	یک ظرف قابل مشاهده یا غیرقابل مشاهده برای گروه‌بندی کنترل‌هاست. می‌تواند نوارهای لغزنده داشته باشد. <code>FlowPanelLayout</code> بطور اتوماتیک کنترل‌ها را بصورت افقی یا عمودی ترازبندی می‌کند. <code>TablePanelLayout</code> کنترل‌ها را در یک شبکه ترازبندی می‌کند
یک تصویر را در بر می‌گیرد.	PictureBox	برای نگهداری تصاویر با قالب‌های استاندارد متنوع بکار می‌رود. بعضی از خصوصیات مربوط به نحوه قرار گرفتن عکس

و تغییر اندازه عکس را فراهم می‌سازد.		
پیشرفت یک عمل را نشان می‌دهد.	پیشرفت یک عمل را نشان می‌دهد که یک بازخورد از پیشرفت یک عمل همچون کپی فایل را به کاربر می‌دهد.	ProgressBar
این کنترل به کاربر اجازه می‌دهد از میان چند گزینه، یکی را انتخاب کند	یک دکمه رادیویی ویندوز را نشان می‌دهد.	RadioButton
یک مجموعه از قاب‌ها را برای نمایش حالت یک برنامه فراهم می‌کند.	یک نوار حالت برای نمایش اطلاعات فعالیت‌های جاری فرم را فراهم می‌سازد.	StatusStrip
ورودی کاربر را می‌پذیرد.	می‌تواند یک ورودی تک یا چند خطی را بپذیرد. با استفاده از خصوصیات آن می‌توان کلمات عبور، نوار لغزنده، تنظیم حالت حروف و محدودیت فقط خواندنی آن را مشخص کرد.	TextBox
داده‌ها را همانند گره‌های یک درخت نمایش می‌دهد.	عملیاتی همچون باز کردن و بستن گره‌ها، اضافه کردن، حذف کردن و کپی کردن گره‌ها در درخت را پشتیبانی می‌کند.	TreeView

در این بخش کنترل‌ها بطور دقیق بررسی نمی‌شوند، فقط برخی از خصوصیات بسیار معمول و کارآی کنترل‌ها عنوان می‌شوند.

۱۵-۲- کلاس‌های Button، GroupBox، Panel و Label

۱۵-۲-۱- کلاس Button

دکمه، معمول‌ترین روش قادر ساختن یک کاربر برای شروع یک فعالیت است. بطور معمول، دکمه به کلیک ماوس یا ضربه کلید با رها کردن یک رویداد کلیک پاسخ می‌دهد، که این رویداد بوسیله یک متد اداره کننده رویداد اداره می‌شود.

Constructor : `Public Button ()`

سازنده این کلاس یک نمونه بدون برچسب ایجاد می‌کند. خصوصیت `Text` دکمه، عنوان آن را تعیین می‌کند و می‌توان یک کلید دسترسی برای آن دکمه تعریف کرد. خصوصیت `Image` آن برای قراردادن یک تصویر روی پس زمینه دکمه استفاده می‌شود.

تنظیم ظاهر یک دکمه

سبک‌های دکمه به قراردادن متن و یک تصویر روی دکمه محدود می‌شوند. می‌توان آنرا صاف یا سه بعدی کرد و رنگ نوشته و پس زمینه‌ی آن را به هر رنگ موجود تنظیم کرد. خصوصیات زیر برای تعریف ظاهر دکمه‌ها، کادرهای انتخاب و دکمه‌های رادیویی استفاده می‌شوند.

`Flatstyle`: می‌تواند چهار مقدار بگیرد: `Flatstyle.Flat`، `Flatstyle.Popup`، `Flatstyle.Standard` و `Flatstyle.System.Standard`. گزینه `Flat` یک دکمه صاف ایجاد می‌کند. `Popup` یک دکمه صاف ایجاد می‌کند، ولی با رفتن ماوس بر روی آن سه بعدی می‌شود. `System` مناسب با سبک سیستم عامل یک دکمه درست می‌کند.

`Image`: تصویری برای قرار گرفتن روی دکمه تعیین می‌کند. متد `Image.FromFile` برای ایجاد یک شی تصویر از یک فایل خاص استفاده می‌شود.

```
button1.Image = Image.FromFile ("c:\\book.gif");
```


ImageAlign: محل تصویر روی دکمه را مشخص می‌کند. مقدار آن با نوع شمارشی **ContentAlignment** مشخص می‌گردد.

```
button\ .ImageAlign =ContentAlignment .MiddleRight;
```

TextAlign: محل متن روی عکس را با استفاده از مقادیر **ContentAlignment** مشخص می‌کند.

اداره کردن رویدادهای Button

یک رویداد **Click** دکمه می‌تواند به چندین روش رخ دهد: بوسیله کلیک دکمه ماوس، با فشار دادن کلید **Enter** یا **SpaceBar** یا با فشار دادن کلید ترکیبی **Alt** و کلید دسترسی.

یک کلید دسترسی با قراردادن **&** قبل از یکی از کاراکترهای مقدار خصوصیت **Text** کنترل ایجاد می‌شود. قطعه کد زیر یک دکمه اعلان می‌کند، کلید دسترسی آن را **C** قرار می‌دهد و یک اداره کننده رویداد برای رویداد **Click** ثبت می‌کند.

```
Button btnClose = new Button();
btnClose.Text= "&Close"; // Pushing ALT + C triggers event
btnClose.Click += new EventHandler(btnClose_Clicked);
// Handle Mouse Click, ENTER key, or Space Bar
private void btnClose_Clicked(object sender, System.EventArgs e)
{
    this.Close();
}
```

توجه داشته باشید که رویداد **Click** دکمه می‌تواند حتی در صورتی که فوکس روی آن نباشد نیز رخ دهد.

خصوصیات **AcceptButton** و **CancelButton** مشخص می‌کنند که رویداد **Click** دکمه با فشار دادن کلیدهای **Enter** یا **ESC** رخ شوند.

*توجه: خصوصیت **CancelButton** فرم را روی دکمه ای تنظیم کنید که رویداد **Click** آن فرم را می‌بندد. این یک روش ساده برای بستن یک پنجره بوسیله کلید **ESC** فراهم می‌کند.*

۱۵-۲-۲-کلاس CheckBox

با این کنترل می‌توان یک ترکیب از گزینه‌های روی یک فرم را انتخاب کرد. برخلاف **RadioButton** که می‌توان فقط یکی از اعضای گروه را انتخاب کرد.

```
Constructor : Public CheckBox()
```

یک کادر انتخاب انتخاب نشده و بدون برچسب توسط سازنده ایجاد می‌شود. خصوصیات **Text** و **Image** قراردادن یک متن یا تصویر اختیاری را در کنار کادر انتخاب ممکن ساخته‌اند.

تنظیم ظاهر CheckBox

کادرهای انتخاب در دو سبک نمایش داده می‌شوند: بصورت یک کادر انتخاب سنتی دنبال شده با متن (یا یک تصویر) یا بصورت یک دکمه دو وضعیت (اگر صاف باشد، انتخاب شده و اگر برآمده باشد، انتخاب نشده است). ظاهر کنترل با تنظیم خصوصیت **Appearance** به یکی از دو مقدار **Appearance.normal** یا **Appearance.Button** انجام می‌شود. کد زیر دو کادر انتخاب نمایش داده شده در شکل ۱۵-۲ را ایجاد می‌کند.

```
// Create traditional check box
this.checkBox1 = new CheckBox();

this.checkBox1.Location =new System.Drawing.Point(۱۰,۱۲۰);
```

```

this.checkBox۱.Text = "La Traviata";

this.checkBox۱.Checked = true;

// Create Button style check box
this.checkBox۲ = new CheckBox();

this.checkBox۲.Location = new System.Drawing.Point(۱۰,۱۵۰);

this.checkBox۲.Text = "Parsifal";

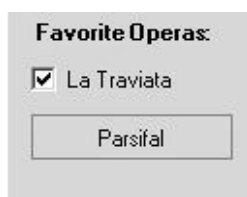
this.checkBox۲.Appearance = Appearance.Button;

this.checkBox۲.Checked = true;

this.checkBox۲.TextAlign = ContentAlignment.MiddleCenter;

```

شکل ۱۵-۲



۱۵-۲-۳-کلاس RadioButton

RadioButton یک کنترل انتخابی است که شبیه کادر انتخاب عمل می‌کند، به استثناء اینکه در هر گروه فقط می‌توان یک مورد را انتخاب کرد. یک گروه شامل چندین کنترل موجود در یک ظرف یکسانی است.

Constructor : Public RadioButton()

یک **RadioButton** انتخاب نشده و بدون هیچ متنی توسط سازنده ایجاد می‌شود. خصوصیات **Text** و **Image** می‌دهند یک متن اختیاری یا یک تصویر در کنار کادر قرار گیرند. ظاهر دکمه رادیویی بوسیله خصوصیات مشابه کادر انتخاب یا دکمه تعریف می‌شوند: **Appearance** و **Flatstyle**.

قرار دادن دکمه‌های رادیویی در یک گروه

دکمه‌های رادیویی قرار گرفته در یک گروه اجازه می‌دهند، فقط یکی از آنها انتخاب شوند. برای مثال، ده سؤال چند گزینه‌ای، ده گروه دکمه رادیویی نیاز دارند. در کنار نیاز به عملکرد گروه‌ها، آنها یک فرصت برای ایجاد یک طرح زیبا فراهم می‌سازند.

اغلب کنترل‌های ظرف **Panel** و **GroupBox** تصاویر پشت زمینه و سبک‌های مختلف را پشتیبانی می‌کنند که می‌توانند ظاهر یک فرم را بهبود دهند. شکل ۱۵-۳ قرار دادن دکمه‌های رادیویی روی یک **GroupBox** که یک تصویر پشت زمینه دارد را نشان می‌دهد.

شکل ۱۵-۳



مثال ۱۵-۱ قطعه کدی ارائه می‌دهد که برای قراردادن دکمه‌های رادیویی روی کنترل `GroupBox` استفاده می‌شود و برای اینکه تصویر پشت دکمه‌ها دیده شود، آنها را شفاف می‌سازد.

مثال ۷-۱

```
using System.Drawing;
using System.Windows.Forms;
public class OperaForm : Form
{
    private RadioButton radioButton۱;
    private RadioButton radioButton۲;
    private RadioButton radioButton۳;
    private GroupBox groupBox۱;
    public OperaForm()
    {
        this.groupBox۱ = new GroupBox();
        this.radioButton۲ = new RadioButton();
        this.radioButton۲ = new RadioButton();
        this.radioButton۱ = new RadioButton();
        // All three radio buttons are created like this
        // For brevity only code for one button is included
        this.radioButton۲.BackColor = Color.Transparent;
        this.radioButton۳.Font = new Font("Microsoft Sans Serif", ۸.۲۵F,
                                           FontStyle.Bold);
        this.radioButton۳.ForeColor = SystemColors.ActiveCaptionText;
        this.radioButton۳.Location = new Point(۱۶, ۸۰);
        this.radioButton۳.Name = "radioButton۳";
        this.radioButton۲.Text = "Parsifal";
        // Group Box
        this.groupBox۱ = new GroupBox();
        this.groupBox۱.BackgroundImage = Image.FromFile("C:\\opera.jpg");
        this.groupBox۱.Size = new Size(۱۲۰, ۱۱۲);
        // Add radio buttons to groupbox
        groupBox۱.Add( new Control[] { radioButton۱, radioButton۲, radioButton۳ } );
    }
}
```

توجه کنید که خصوصیت `BackColor` دکمه رادیویی، `Color.Transparent` قرار داده می‌شود. `BackColor` یک خصوصیت محدود شده است، بدین معنی که آن رنگ کنترل پدر خود را می‌گیرد. اگر هیچ رنگی به دکمه رادیویی انتساب داده نشود، آن `BackColor` کنترل `GroupBox` را گرفته و تصویر را پنهان می‌کند.

۱۵-۲-۴-کلاس GroupBox

GroupBox یک کنترل ظرف است که یک حاشیه در اطراف کلکسیون کنترل‌های خود قرار می‌دهد. همانطور که در مثال قبلی نشان داده شد، آن اغلب برای گروه‌بندی دکمه‌های رادیویی استفاده می‌شود. اما آن یک روش مناسب و راحت برای سازمان‌دهی و مدیریت کنترل‌های مرتبط به هم روی یک فرم است. برای مثال، قرار دادن خصوصیت Enabled آن به مقدار False، همه کنترل‌های کادر گروه غیرفعال می‌گردند.

Constructor : public GroupBox()

بطور پیش‌فرض، یک GroupBox بدون عنوان با عرض ۲۰۰ پیکسل و ارتفاع ۱۰۰ پیکسل توسط سازنده آن ایجاد می‌گردد.

۱۵-۲-۵-کلاس Panel

این کلاس به عنوان یک ظرف بزرگ برای گروه‌بندی کلکسیونی از کنترل‌ها استفاده می‌شود. آن بسیار شبیه کنترل GroupBox است. اما یک فرزند از کلاس ScrollableClass است و توانایی لغزاندن را دارد.

Constructor : public Panel()

سازنده‌ی این کلاس، یک ناحیه ظرف بدون حاشیه ایجاد می‌کند که توانایی لغزاندن آن نیز غیرفعال است. بطور پیش‌فرض، Panel رنگ پس زمینه‌ی ظرف خود را می‌گیرد، بطوریکه روی فرم غیرقابل مشاهده می‌گردد.

چون GroupBox و Panel هدف یکسانی را مدنظر دارند، اغلب برنامه‌نویس‌ها با سؤالی در مورد انتخاب یکی از آنها روبرو هستند؟

- GroupBox می‌تواند یک عنوان قابل مشاهده داشته باشد، در حالی که کلاس Panel ندارد.
- GroupBox همیشه یک حاشیه نشان می‌دهد. حاشیه Panel با خصوصیت BorderStyle معین می‌گردد. که مقدار آن می‌تواند Borderstyle.Fixed3D, Borderstyle.Single, Borderstyle.None باشد.
- GroupBox امکان لغزاندن ندارد، در حالیکه با True قرار دادن خصوصیت Autoscroll کنترل Panel بصورت اتوماتیک لغزان می‌شود.
- Panel برای قراردادن یا تنظیم کنترل‌های روی خود، هیچ ویژگی پیشنهاد نمی‌کند. به همین دلیل، زمانی که شمای کنترل‌ها در زمان طراحی معین باشند استفاده می‌شود، اما این همیشه ممکن نیست. بیشتر برنامه‌های کاربردی یک فرم بوسیله کنترل‌های مبتنی بر حالت زمان اجرا تولید می‌کنند. برای پشتیبانی از ایجاد پویایی کنترل‌ها، NET دو ظرف پیشنهاد می‌کند که از Panel ارث‌بری می‌کنند و محل کنترل‌های خود را بصورت اتوماتیک تنظیم می‌کنند: FlowLayoutPanel و TableLayoutPanel.

۱۵-۲-۶-کنترل FlowLayoutPanel

شکل ۱۵-۴ یک طرح‌بندی از کنترل‌ها را با استفاده از FlowLayoutPanel نشان می‌دهد.

۴-۱۵



این کنترل ساده یک سازنده بدون پارامتر و دو خصوصیت با ارزش دارد. خصوصیت `FlowDirection` که جهت اضافه شدن کنترل‌ها به ظرف را معین می‌کند و خصوصیت `WrapControls` معین می‌کند که آیا کنترل‌های فرزند به سطر دیگری منتقل شوند یا طول کنترل‌ها کوتاه شوند.

قطعه کد زیر یک `FlowDirection` ایجاد می‌کند و کنترل‌هایی به کلکسیون آن اضافه می‌کند.

```
FlowLayoutPanel flp = new FlowLayoutPanel();
flp.FlowDirection = FlowDirection.LeftToRight;
// Controls are automatically positioned left to right
flp.Controls.Add(Button1);
flp.Controls.Add(Button2);
flp.Controls.Add(TextBox1);
flp.Controls.Add(Button3);
this.Controls.Add(flps); // Add container to form
```

اعضای نوع شمارشی `FlowDirection`، موارد `LeftToRight`، `RightToLeft`، `BottomUp` و `TopDown` هستند و مقدار پیش‌فرض آن `LeftToRight` است.

۱۵-۲-۷-کنترل `TableLayoutPanel`

شکل ۱۵-۵ طرح‌بندی شبکه‌ای را نشان می‌دهد که حاصل استفاده یک ظرف `TableLayoutPanel` است.

شکل ۱۵-۵



این قطعه کد یک `TableLayoutPanel` ایجاد می‌کند و همان چهار کنترل مثال قبلی را به آن اضافه می‌کند. در خصوصیات ظرف تعیین می‌شود که طرح‌بندی شبکه، دو سطر و دو ستون دارد و یک سبک حاشیه `Inset` دور هر سلول بکار می‌برد. کنترل‌ها همواره از چپ به راست و بالا به پایین اضافه می‌شوند.

```
TableLayoutPanel tlp = new TableLayoutPanel();
// Causes the inset around each cell
tlp.CellBorderStyle = TableLayoutPanelCellBorderStyle.Inset;
tlp.ColumnCount = 2; // Grid has two columns
tlp.RowCount = 2; // Grid has two rows
// If grid is full add extra cells by adding column
tlp.GrowStyle = TableLayoutPanelGrowStyle.AddColumns;
// Padding (pixels) within each cell (left, top, right, bottom)
tlp.Padding = new Padding(10, 4, 5);
tlp.Controls.Add(Button1);
tlp.Controls.Add(Button2);
// Other controls added here
```

خصوصیت `GrowStyle` بسیار با ارزش است. آن مشخص می‌کند در صورت پر شدن سطرها و ستون‌ها، چگونه کنترل‌های جدید به آن اضافه شوند. در این مثال، متد `AddColumns` یک ستون برای افزودن کنترل‌های جدید اضافه می‌کند. گزینه‌های دیگر `AddRows` و `None` هستند. در صورتی که `None` انتخاب شود، در صورتی که شبکه پر شده باشد، با اضافه شدن کنترل جدید، استثنایی رخ می‌دهد.

۱۵-۲-۸-کلاس Label

این کلاس برای اضافه کردن اطلاعات توصیفی روی یک فرم استفاده می‌شود.

constructor: `public Label()`

یک نمونه از برچسب بدون عنوان توسط سازنده ایجاد می‌شود. برای انتساب یک مقدار به برچسب خصوصیت `Text` را بکار برید. خصوصیات `Image`، `BorderStyle` و `TextAlign` برای تعریف و زیباسازی ظاهر برچسب استفاده می‌شوند.

کد زیر برچسب نشان داده شده در شکل ۱۵-۶ را ایجاد می‌کند.

```
Label imgLabel = new Label();
imgLabel.BackColor= Color.White;
Image img = Image.FromFile("c:\\rembrandt.jpg");
imgLabel.Image= img;
imgLabel.ImageAlign= ContentAlignment.TopCenter;
imgLabel.Text="Rembrandt";
imgLabel.TextAlign= ContentAlignment.BottomCenter;
imgLabel.BorderStyle= BorderStyle.Fixed3D;
imgLabel.Size = new Size(img.Width+۱۰, img.Height+۲۵);
```

شکل ۱۵-۶



`UseMnemonic` یکی از خصوصیات ناآشنای آن است. با مقداردهی آن به `True` و قرار دادن علامت `&` قبل از یک کاراکتر در خصوصیت `Text` برچسب، می‌توانید یک کلید دسترسی ایجاد کنید. برای مثال، اگر یک برچسب مقدار `&Sum` دارد با فشار دادن کلیدهای `Alt-S` کانون به آن منتقل می‌گردد.

۱۵-۳-کنترل‌های PictureBox و TextBox

۱۵-۳-۱-کلاس PictureBox

کنترل `PictureBox` برای نمایش تصاویری همچون `bitmap`، `Icon`، `metafile`، `JPEG`، `GIF` یا `PNG` استفاده می‌شود. آن یک کنترل پویاست و انتخاب عکس‌ها در زمان اجرا و طراحی، تغییر اندازه و تغییر موقعیت کنترل را در زمان اجرا و طراحی مجاز می‌دارد.

constructor: `public PictureBox()`

سازنده‌ی این کلاس یک کادر عکس خالی ایجاد می‌کند (`Image = null`)، که خصوصیت `SizeMode` آن تنظیم می‌شود تا هر تصویری در گوشه بالا سمت چپ کادر نمایش داده شود.

دو خصوصیت آشنای آن `Image` و `SizeMode` است. `Image` تصویری که در کادر عکس نمایش داده خواهد شد را مشخص می‌کند. آن می‌تواند به یکی از اعضای نوع `PictureBoxSizeMode` انتساب داده شود:

۱- `PictureBox.AutoSize`: با تصویر هم اندازه می‌شود.

۲- `CenterImage`: تصویر در وسط کادر نمایش داده می‌شود و در صورت نیاز برش داده می‌شود.

۳- Normal: تصویر در گوشه بالا سمت چپ قرار می‌گیرد و در صورت نیاز برش داده می‌شود.

۴- StretchImage: تصویر کوچک شده یا کشیده می‌شود تا با کادر متناسب گردد.

شکل ۷-۱۵ برخی از ویژگی‌های کنترل PictureBox را ارائه می‌کند. آن روی فرم سه کادر عکس کوچک برای نگه داشتن تصاویر کوچک و یک کادر عکس بزرگ برای نمایش تصویر کامل دارد. زمانی که کاربر روی تصویر کوچک دابل کلیک می‌کند، تصویر بزرگ نمایش داده می‌شود.

شکل ۷-۱۵



کد موجود در مثال ۷-۲ واضح است. اداره کننده رویداد ShowPic به هر رویداد DoubleClick با تنظیم مقدار خصوصیت Image کنترل PictureBox بزرگ به تصویر موجود در عکس کوچک عکس‌العمل نشان می‌دهد. توجه داشته باشید که تصاویر اصلی اندازه bigPicture دارند و برای متناسب شدن با کادرهای عکس کوچک، بطور اتوماتیک کوچکتر شده‌اند (با تنظیم خصوصیت SizeMode).

مثال ۷-۲

```
using System;
using System.Drawing;
using System.Windows.Forms;
public class ArtForm : Form
{
    private PictureBox bigPicture;
    private PictureBox tn1;
    private PictureBox tn2;
    private PictureBox tn3;
    private Button btnClear;
    public ArtForm()
    {
        bigPicture = new PictureBox();
        tn1 = new PictureBox();
        tn2 = new PictureBox();
        tn3 = new PictureBox();
        btnClear = new Button();
        bigPicture.Location = new Point(۹۰, ۳۰);
        bigPicture.Name = "bigPicture";
        bigPicture.Size = new Size(۱۶۰, ۱۶۰);
        this.Controls.Add(bigPicture);
        // Define picturebox to hold first thumbnail image
        tn1.BorderStyle = BorderStyle.FixedSingle;
        tn1.Cursor = Cursors.Hand;
```

```

tn1.Image = Image.FromFile("C:\\\\schiele1.jpg");
tn1.Location = new Point(۸, ۱۶);
tn1.Name = "tn1";
tn1.Size = new Size(۵۶, ۵۶);
tn1.SizeMode = PictureBoxSizeMode.StretchImage;
this.Controls.Add(tn1);
// Code for other thumbnails would go here
// Button to clear picture box
btnClear.Location = new Point(۱۳۶, ۱۹۲);
btnClear.Name = "btnClear";
btnClear.Size = new Size(۸۸, ۲۴);
btnClear.Text = "Clear Image";
this.Controls.Add(btnClear);
btnClear.Click += new EventHandler(this.btnClear_Click);
// Set up event handlers for double click events
tn1.DoubleClick += new EventHandler(ShowPic);
tn2.DoubleClick += new EventHandler(ShowPic);
tn3.DoubleClick += new EventHandler(ShowPic);
}
static void Main()
{
    Application.Run(new ArtForm());
}
private void btnClear_Click(object sender, EventArgs e)
{
    bigPicture.Image = null; // Clear image
}
private void ShowPic (object sender, EventArgs e)
{
    // Sender is thumbnail image that is double clicked
    bigPicture.Image = ((PictureBox) sender).Image;
}
}

```

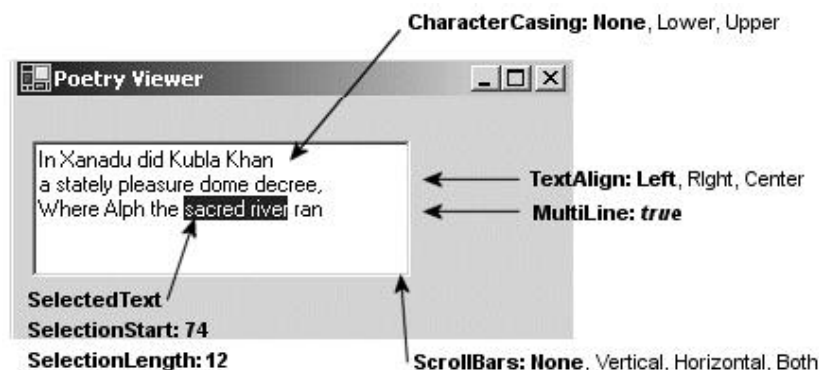
۱۵-۳-۲- کلاس TextBox

کنترل آشنای TextBox، یک کنترل آسان است که چند خصوصیت مؤثر روی ظاهرش دارد و تعدادی از خصوصیات، محتوای آن را کنترل می‌کنند. توسعه دهنده بوسیله تنظیم اداره کننده‌های رویداد و روتین‌های بازبینی، داده‌های وارد شده در کادر متنی را کنترل می‌کنند.

constructor: public TextBox()

سازنده این کلاس، یک TextBox ایجاد می‌کند که یک خط متن می‌پذیرد و رنگ و فونت ظرف به آن انتساب داده می‌شود. به آسانی می‌توان کنترل را به یک کادر اداره کننده متن چند خطی تبدیل کرد که تعداد خاصی کاراکتر می‌پذیرد و آنها را به چپ، راست یا وسط تنظیم می‌کند. شکل ۱۵-۸، برخی از خصوصیات آن را ارائه می‌کند.

شکل ۱۵-۸



با استفاده از خصوصیت `Text` و متد `AppendText` متنی در کادر متنی قرار می‌گیرد.

```
tPoetry.Text =
    "In Xanadu did Kubla Khan\r\na stately pleasure dome decree,";
txtPoetry.AppendText("\r\nWhere Alph the sacred river ran");
```

خصوصیت مهمی از `TextBox`، `ReadOnly` است که از تغییر محتوای کادر متنی جلوگیری می‌کند. خصوصیت `PasswordChar` برای ماسک کردن کاراکترهای داخل کادر به کاراکترهای خاص (معمولاً یک کلمه عبور) تنظیم می‌شود.

کادرهای متنی و کاراکترهای بازگشت به سر سطر

زمانی که داده‌های یک `TextBox` را به یک پایگاه داده ذخیره می‌کنید، می‌خواهید مطمئن شوید هیچ کاراکتر خاص در آن تعبیه نشده باشد (همانند کاراکتر برگشت به سر سطر). اگر به خصوصیات `TextBox` نظری بیفکنید، `AcceptsReturn` را خواهید یافت. اگر مقدار آن را `False` قرار دهید، `TextBox` فشار دادن کلید `Enter` توسط کاربر را نادیده می‌گیرد. با این وجود، نام این خصوصیت بعضی مواقع گمراه‌کننده است. فقط زمانی که خصوصیت `AcceptsReturn` فرم به یک دکمه روی فرم تنظیم شده باشد، کار می‌کند. اگر `AcceptsReturn` تنظیم نشده باشد (و خصوصیت `MultiLine` کادر متنی `True` باشد)، زمانی که کلید `Enter` فشار داده شود (`TextBox\r\n` یک خط جدید دریافت می‌کند).

این عمل توسعه‌دهنده را از کار اداره کردن بازگشت به سر سطر ناخواسته رها می‌کند. دو روش در دسترس است: به محض وارد کردن داده، ضربه‌های کلید را گرفته یا قبل از ذخیره کاراکترها در متن، آنها را استخراج کنید. روش اول یک اداره کننده رویداد صفحه کلید بکار می‌برد.

```
// Set up event handler in constructor for TextBox txtPoetry
txtPoetry.KeyPress += new KeyPressEventHandler(onKeyPress);
private void onKeyPress( object sender, KeyPressEventArgs e)
{
    if(e.KeyChar == (char)۱۳) e.Handled = true;
}
```

با دادن مقدار `True` به `Handled` از اضافه شدن `\r\n` به کادر متنی جلوگیری می‌شود. این عمل برای ورود از طریق صفحه کلید خوب کار می‌کند، اما در عملیات `Cut` و `Paste` تأثیری ندارد. برای برطرف کردن این مورد، از `Paste` کردن جلوگیری کرده یا اینکه در یک مرحله بازرسی نهایی همه کاراکترهای بازگشت به سر سطر را با یک فضای خالی یا هر کاراکتر انتخاب جایگزین کنید.

```
txtPoetry.Text = txtPoetry.Text.Replace(Environment.NewLine, " ");
```

نکته: دو روش معمول برای وارد کردن یک کاراکتر برگشت به سر سطر وجود دارد:

```
txtPoetry.Text = "Line ۱\r\nLine ۲";
txtPoetry.Text = "Line ۱"+Environment.NewLine+"Line ۲";
```

۱۵-۴-کلاس‌های ComboBox و CheckedListBox، ListBox

۱۵-۴-۱-کلاس ListBox

کنترل ListBox لیستی از اقلام فراهم می‌سازد، که ممکن است کاربر یک یا چند تا از آنها را انتخاب کند. معمولاً اطلاعات این لیست از نوع متنی است. اما می‌تواند تصاویر و اشیایی را شامل شود. ویژگی‌های دیگر آن مدهایی برای جستجوی متنی، مرتب‌سازی، نمایش چند ستونی، نوارهای لغزنده افقی و عمودی و یک روش ساده برای override کردن ظاهر پیش‌فرض و ایجاد عناصر لیست توسط خود کاربر را دربر دارد.

constructor: public ListBox()

سازنده یک ListBox خالی ایجاد می‌کند. کد ایجاد ListBox در سازنده‌ی فرم ظرف یا اداره کننده رویداد Form.Load قرار می‌گیرد. اگر خصوصیت ListBox.Sorted مقدار True قرار داده شود، اقلام ListBox بر اساس ترتیب صعودی الفبایی مرتب می‌شوند. همچنین اگر اندازه کنترل برای نمایش همه اقلام کافی نباشد، بطور اتوماتیک نوارهای لغزنده عمودی اضافه می‌شوند.

اضافه کردن اقلام به یک ListBox

ListBox یک کلکسیون بنام Items دارد که همه اقلام لیست را در بر دارد. با مقید کردن ListBox به یک منبع داده (ADO.NET) یا با استفاده از متد Add، می‌توان عناصری به لیست اضافه کرد. اگر خصوصیت Sorted مقدار False باشد، اقلام به همان ترتیب که وارد می‌شوند لیست می‌شوند. همچنین یک متد بنام Insert برای قراردادن یک قلم داده در موقعیت خاص وجود دارد.

```
lstArtists.Items.Add("Monet");
lstArtists.Items.Add("Rembrandt");
lstArtists.Items.Add("Manet");
lstArtists.Items.Insert(0, "Botticelli"); //Place at top
```

توجه: برای جلوگیری از ترسیم مجدد ListBox در زمان اضافه کردن یک قلم داده، متد ListBox.BeginUpdate را قبل از اضافه کردن و ListBox.EndUpdate را بعد از اضافه کردن آخرین قلم داده اجرا کنید.

همچنین امکان دارد کادرهای لیست اشیایی را شامل شوند، چون یک شی اعضای زیادی دارد، سؤال این است که چه چیز در لیست TextBox نمایش داده می‌شود. چون بطور پیش‌فرض ListBox نتایج حاصل از متد ToString یک قلم داده را نشان می‌دهد، ضروری است این متد System.Object برای هر قلم داده override شده باشد. کلاس زیر برای ایجاد اقلام ListBox استفاده می‌شود.

```
// Instances of this class will be placed in a ListBox
public class Artist
{
    public string BDate, DDate, Country;
    private string firstname;
    private string lastname;
    public Artist(string birth, string death, string fname,
                  string lname, string ctry)
    {
        BDate = birth;
        DDate = death;
        Country = ctry;
        firstname = fname;
        lastname = lname;
    }
}
```

```

public override string ToString()
{
    return (lastname+" , "+firstname);
}
public string GetLName
{
    get
    {
        return lastname;
    }
}
public string GetFName
{
    get
    {
        return firstname;
    }
}
}

```

متد ToString برای برگرداندن نام و فامیل هنرمند override شده است، که در ListBox نمایش داده می‌شود. ListBox شکل ۹-۱۵ با استفاده از این دستورات تولید شده است.

```

lstArtists.Items.Add (new Artist("۱۸۳۲", "۱۸۸۳", "Edouard", "Manet", "Fr" ));
lstArtists.Items.Add (new Artist("۱۸۴۰", "۱۹۲۶", "Claude", "Monet", "Fr"));
lstArtists.Items.Add (new Artist("۱۶۰۶", "۱۶۶۹", "Von Rijn", "Rembrandt", "Ne"));
lstArtists.Items.Add (new Artist("۱۴۴۵", "۱۵۱۰", "Sandre", "Botticelli", "It"));

```

شکل ۹-۱۵



انتخاب و جستجوی اقلام در یک ListBox

خصوصیت SelectionMode معین می‌کند چه تعداد قلم داده ListBox در یک لحظه قابل انتخاب هستند. آن چهار مقدار از نوع شمارشی SelectionMode را می‌گیرد: None، Single، MultiSingle و MultiExtended.

مقدار MultiSingle بوسیله کلیک روی قلم داده یا فشار دادن SpaceBar انتخاب می‌کند. MultiExtended کاربرد کلیدهای Shift و Ctrl را مجاز می‌دارد.

رویداد SelectedIndexChanged یک راه ساده برای تشخیص انتخاب یک قلم داده از ListBox است. زمانی که کاربر روی یک قلم داده کلیک می‌کند یا کلیدهای جهت‌نما را حرکت می‌دهد، این رویداد اتفاق می‌افتد. کاربرد معمول آن نمایش اطلاعات دقیق در مورد عناصر موجود در کنترل‌های دیگر فرم است. در کد زیر زمانی که یک هنرمند از ListBox انتخاب می‌شود، تاریخ تولد و مرگ آن را نمایش می‌دهد. شکل ۹-۱۵ را ببینید.

```

// Set up event handler in constructor
lstArtists.SelectedIndexChanged += new EventHandler(ShowArtist);
//
private void ShowArtist(object sender, EventArgs e)
{
    // Cast to artist object in order to access properties
    Artist myArtist = lstArtists.SelectedItem as Artist;
}

```

```

if (myArtist != null) {
    txtBirth.Text = myArtist.Dob; // Place dates in text boxes
    txtDeath.Text = myArtist.Dod;
}
}

```

خصوصیت `SelectedItem` قلم داده‌ی انتخاب شده در `ListBox` را بر می‌گردانند. این شی از طریق عملگر `as` به `myArtist` انتساب داده می‌شود تا مطمئن شود شی یک نوع `Artist` است. می‌توان خصوصیت `SelectedIndex` را برای ارجاع به قلم داده انتخاب شده بکار برد.

```
myArtist = lstArtists.Items[lstArtists.SelectedIndex] as Artist;
```

برای کار با یک `ListBox` چند انتخابی یک روش مختلفی لازم است. معمولاً تا زمانی که همه اقلام داده انتخاب نشده‌اند، نمی‌خواهید به رویداد انتخاب پاسخ دهید. یک روش کلیک روی یک دکمه توسط کاربر است تا خبر دهد همه انتخاب‌ها انجام شده و عمل بعدی نیاز است. همه انتخاب‌ها در بخشی از `SelectedItem` کلکسیون قرار می‌گیرند، پس یافتن آنها توسط یک شمارنده راحت است.

```
foreach (Artist a in lstArtists.SelectedItems)
```

```
    MessageBox.Show(a.GetLName);
```

متد `SetSelected` برای انتخاب یک یا چند قلم داده از طریق برنامه‌نویسی بکار می‌رود. آن متد قلم داده‌ها را انتخاب کرده و رویداد `SelectedIndexChanged` را رها می‌سازد. در این مثال، برای انتخاب همه هنرمندان متولد فرانسه استفاده می‌شود.

```

for (int ndx = 0; ndx < lstArtists.Items.Count-1; ndx++)
{
    Artist a = lstArtists.Items[ndx] as Artist;
    if (a.country == "Fr") lstArtists.SetSelected(ndx, true);
}

```

سفارشی کردن ظاهر یک `ListBox`

`ListBox` به همراه کنترل‌های `ComboBox`، `MenuItem` و `TabControl` شکل ترسیم خاص خود را دارد. بدین معنی که با مقداردی خصوصیات یک کنترل، می‌توانید از آن بخواهید، زمانی که لازم است محتویات کنترل مجدداً رسم شود، یک رویداد آزاد سازد. یک اداره کننده سفارشی رویداد، ترسیم واقعی را فراهم می‌سازد.

برای فعال کردن ترسیم خودکار `ListBox`، خصوصیت `DrawMode` آن باید یکی از مقادیر نوع شمارشی `DrawMode` قرار داده شود: `OwnerDrawFixed` یا `OwnerDrawVariable`. اولی هر قلم داده را با یک اندازه ثابت رسم می‌کند. دومی قلم داده‌هایی با اندازه متغیر را مجاز می‌دارد. هر دو مورد باعث می‌شوند، رویداد `DrawItem` رها شود و به عمل ترسیم توسط اداره کننده رویداد تکیه دارد.

با استفاده از `ListBox` مثال قبلی، در سازنده‌ی فرم، خصوصیت `DrawMode` را مقداردهی کرده و یک اداره کننده رویداد برای رویداد `DrawItem` ثبت می‌کنیم.

```

lstArtists.DrawMode = DrawMode.OwnerDrawFixed;
lstArtists.ItemHeight = ۱۶; // Height (pixels) of item
lstArtists.DrawItem += new DrawItemEventHandler(DrawList);

```

نماینده `DrawItemEventHandler` دو پارامتر دارد: شی `Sender` و شی `DrawItemEventArgs`. پارامتر دومی جالب‌تر است و خصوصیاتی در ارتباط با ظاهر و حالت کنترل دارد. جدول ۱۵-۲ خلاصه‌ای از آنهاست.

جدول ۱۵-۲

عضو	توصیف
BackColor	رنگ پس زمینه انتساب داده شده به کنترل است.
Bounds	مختصات قلم داده را تعریف می‌کند تا بصورت یک شی Rectangle رسم شود
Font	فونت انتساب داده شده به کنترل است.
ForeColor	رنگ متن نمایش داده شده روی کنترل
Graphics	سطح کنترل را به صورت یک شی Graphics ارائه می‌کند، تا ترسیم روی آن رخ دهد.
Index	اندیس قلم داده که رسم می‌شود.
State	حالت قلم داده‌ای که رسم می‌شود. مقدار آن از نوع شمارشی DrawItemState است. در ListBox مقادیر (Selected) یا (0) را دارد.
DrawBackground()	پس زمینه پیش‌فرض را رسم می‌کند.
DrawFocusRectangle()	مستطیل کانون را دور قلم داده‌ای که کانون را دارد، رسم می‌کند.

Index برای یافتن موقعیت یک قلم داده استفاده می‌شود. Font، BackColor و ForeColor تنظیمات هر کدام را بر می‌گردانند. Bounds یک ناحیه مستطیلی را تعریف می‌کند که عمل ترسیم قلم داده‌ی جدید در آنجا رخ دهد. State برای مشخص کردن حالت انتخاب قلم داده در زمان ترسیم مفید است، مخصوصاً زمانی که ListBox انتخاب چندگانه را پشتیبانی می‌کند.

اداره کننده رویداد برای ترسیم اقلام داده‌ای ListBox در مثال ۱۵-۳ نشان داده می‌شود. رفتار آن بوسیله عملی که انجام می‌شود، تعیین می‌گردد. اگر قلم داده‌ای انتخاب نشده باشد، یک حاشیه سیاه در پشت زمینه برای انتخاب آن رسم می‌شود. اگر یک قلم داده‌ای اضافه شود، پس زمینه با رنگی متناسب با کشور هنرمند پر می‌شود و نام و فامیل هنرمند نمایش داده می‌شود.

در این پروسه دانشی درباره مفاهیم GDI+ لازم است. با این وجود، هدف متدها از نام و محتوای آنها واضح است. FillRectangle یک ناحیه مستطیلی تعریف شده بوسیله شی Rectangle را پر می‌کند و DrawString با استفاده از رنگ فونت تعریف شده توسط شی Brush، متنی را روی شی Graphics رسم می‌کند. شکل ۱۵-۹ خروجی را نشان می‌دهد.

مثال ۱۵-۳

```
private void DrawList(object sender, DrawItemEventArgs e)
{
    // Draw ListBox Items
    string ctry;
    Rectangle rect = e.Bounds;
    Artist a = lstArtists.Items[e.Index] as Artist;
    string artistName = a.ToString();
    if ( (e.State & DrawItemState.Selected) == DrawItemState.Selected )
    {
        // Draw Black border around the selected item
```

```

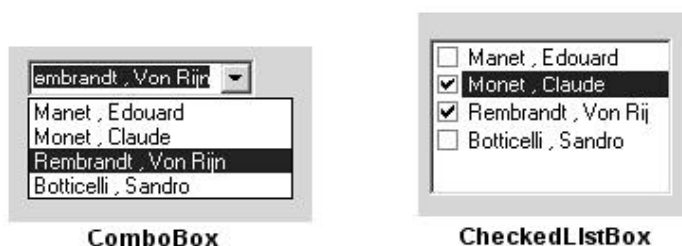
        e.Graphics.DrawRectangle(Pens.Black, rect);
    } else {
        ctry = a.Country;
        Brush b; // Object used to define backcolor
        // Each country will have a different backcolor
        b = Brushes.LightYellow; // Netherlands
        if (ctry == "Fr") b = Brushes.LightGreen;
        if (ctry == "It") b = Brushes.Yellow;
        e.Graphics.FillRectangle(b, rect);
        e.Graphics.DrawString(artistName, e.Font, Brushes.Black, rect);
    }
}

```

۱۵-۴-۲- کنترل‌های دیگری از لیست: ComboBox و CheckedListBox

کنترل ComboBox یک کنترل ترکیبی از ListBox و TextBox است. (شکل ۱۵-۱۰ را ببینید). شبیه listBox از ListControl مشتق می‌شود و بیشتر خصوصیات آن را متصرف می‌شود.

شکل ۱۵-۱۰



از نظر ظاهری، کنترل ComboBox یک کادر متنی دارد که محتوای آن از طریق خصوصیت Text در دسترس است و یک لیست بازشو که قلم داده‌ی انتخابی از طریق خصوصیت SelectedItem در دسترس است. زمانی که یک قلم داده انتخاب می‌شود، متن آن در کادر متنی نمایش داده می‌شود. ComboBox در ایجاد سؤالاتی که کاربر یکی از اقلام داده را انتخاب می‌کند یا جواب را تایپ می‌کند مفید است. ساختار آن شبیه ListBox است.

```

ComboBox cbArtists = new ComboBox();
cbArtists.Size = new System.Drawing.Size(۱۲۰, ۲۱);
cbArtists.MaxDropDownItems= ۴; // Max number of items to display
cbArtists.DropDownWidth = ۱۴۰; // Width of drop-down portion
cbArtists.Items.Add(new Artist("۱۸۳۲", "۱۸۸۲", "Edouard", "Manet", "Fr" ));
// Add other items here...

```

CheckedListBox حالت تغییر یافته‌ای از کنترل ListBox است که یک کادر انتخاب برای هر قلم داده در لیست اضافه می‌کند. رفتار پیش‌فرض این کنترل، انتخاب یک قلم داده با اولین کلیک است و با کلیک دوم قلم داده‌ی انتخاب شده از انتخاب خارج می‌شود.

برای اینکه با یک کلیک، قلم داده انتخاب شده یا از انتخاب خارج شود، خصوصیت CheckOnClick را True قرار دهید. اگرچه آن انتخاب چندگانه را پشتیبانی نمی‌کند، CheckedListBox اجازه می‌دهد چندین قلم داده انتخاب شوند و آنها در یک کلکسیون CheckedItems قرار می‌گیرند.

قطعه کد زیر، کلکسیون اشیاء Artist را بررسی می‌کند که قلم داده‌های انتخاب شده روی کنترل را در بر دارد.

```

// List all items with checked box.
foreach (Artist a in clBox.CheckedItems)
    MessageBox.Show(a.ToString()); // -> Monet, Claude

```

می‌توانید سراسر کلکسیون را طی کرده و حالت انتخاب قلم داده‌ها را صریحاً معین کنید.

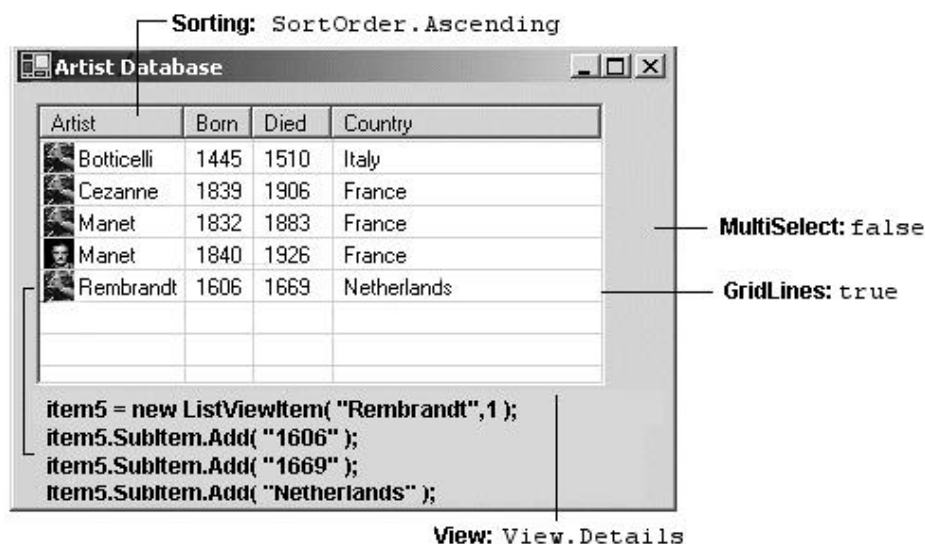
```
for (int i=0; i< clBox.Items.Count; i++)
{
    if(clBox.GetItemCheckState(i) == CheckState.Checked)
    {
        Do something
    } else
    {
        do something if not checked
    }
}
```

۱۵-۵-کلاس‌های ListView و TreeView

۱۵-۵-۱-کلاس ListView

ListView کنترل دیگری است که لیستی از اطلاعات را نمایش می‌دهد. این کنترل داده‌ها را بصورت رابطه‌ای همچون قلم داده‌ها و زیر قلم داده‌ها ارائه می‌کند. به روش‌های متعددی می‌توان داده‌ها را نمایش داد. شبکه‌ی چند ستونی با آیکن‌های کوچک یا بزرگ برای نمایش قلم داده‌ها استفاده می‌شوند. همچنین تصاویر و کادرهای انتخاب کنترل را زینت می‌دهد. شکل ۱۱-۱۵ خصوصیات و متدهای اساسی استفاده شده برای طرح‌بندی یک نمای دقیق کنترل را ارائه می‌کند. ستون اول به همراه یک عکس، متن یک قلم داده را در بر دارد. مابقی ستون‌ها، زیر قلم داده‌های قلم داده‌ی پدر را در بر دارند.

شکل ۱۱-۱۵



اجازه دهید نحوه ایجاد این سبک از ListView را بررسی کنیم.

ایجاد یک شیء ListView

ListView بوسیله‌ی یک سازنده بدون پارامتر ایجاد می‌شود.

```
ListView listView1 = new ListView();
```

تعریف ظاهر شیء ListView

```
// Set the view to show details
listView1.View = View.Details;
```

خصوصیت View یکی از ۵ طرح‌بندی را برای کنترل مشخص می‌کند.

Details: در ستون اول یک آیکون و یک متن نمایش داده می‌شود. زیر قلم داده‌ها در ستون‌های مابقی نمایش داده می‌شوند.

LargeIcon: برای هر قلم داده یک آیکون بزرگ و یک برچسب در زیر آن نشان می‌دهد.

List: هر قلم داده بصورت یک آیکون کوچک و برچسبی در سمت راست آن نمایش داده می‌شود. آیکون‌ها در میان ستون‌های کنترل مرتب می‌شوند.

SmallIcon: هر قلم داده در یک ستون بصورت یک آیکون کوچک برچسبی در سمت راست آن ظاهر می‌گردد.

Tile: هر قلم داده بصورت یک آیکون با اندازه طبیعی ظاهر می‌گردد که برچسب و جزئیات آن در سمت راست ظاهر می‌گردد. فقط در WinXP و ۲۰۰۳ موجود است.

توجه: خصوصیت ListView.View در زمان اجرا قابل تغییر می‌باشد و می‌توانید مابین نماهای ممکن سوئیچ کنید.

بعد از اینکه نمای Details انتخاب می‌شود، خصوصیات دیگری که ظاهر و رفتار کنترل را تعریف می‌کنند، مقداردهی می‌شوند.

```
// Allow the user to rearrange columns
listView1.AllowColumnReorder = true;
// Select the entire row when selection is made
listView1.FullRowSelect = true;
// Display grid lines
listView1.GridLines = true;
// Sort the items in the list in ascending order
listView1.Sorting = SortOrder.Ascending;
```

این خصوصیات بطور اتوماتیک قلم داده‌ها را مرتب می‌سازند، به کاربر اجازه می‌دهند ستون‌ها را بکشند و زمانی که یک قلم داده انتخاب می‌شود، کل سطر انتخاب می‌گردد.

تنظیم سرآیندهای ستون

در نمای Details، تا زمانی که حداقل یک ستون به کنترل اضافه نشود، داده‌های نمایش داده نمی‌شوند. بوسیله متد Columns.Add ستون‌ها را اضافه کنید. ساده ترین شکل بصورت زیر است:

```
ListView.Columns.Add(caption, width, textAlign)
```

Caption متنی است که نمایش داده می‌شود. Width تعداد پیکسل‌های عرض ستون را مشخص می‌کند. در صورتی که ۱- باشد، اندازه ستون بطور اتوماتیک متناسب با بزرگترین قلم داده تغییر می‌یابد و اگر ۲- باشد، با عرض سرآیند هم اندازه می‌شود.

```
// Create column headers for the items and subitems
listView1.Columns.Add("Artist", -۲, HorizontalAlignment.Left);
listView1.Columns.Add("Born", -۲, HorizontalAlignment.Left);
listView1.Columns.Add("Died", -۲, HorizontalAlignment.Left);
listView1.Columns.Add("Country", -۲, HorizontalAlignment.Left);
```

متد Add یک نوع داده columnHeader ایجاد کرده و به کلکسیون Column کنترل ListView اضافه می‌کند. این متد یک overload دارد که شی columnHeader را مستقیماً به عنوان پارامتر می‌گیرد.

```
ColumnHeader cHeader:
cHeader.Text = "Artist";
cHeader.Width = -۲;
```



```
cHeader.TextAlign = HorizontalAlignment.Left;
ListView.Columns.Add(ColumnHeader cHeader);
```

ایجاد قلم داده‌های ListView

سازنده‌ی کلاس ListView چندین overload دارد. آنها می‌توانند برای ایجاد یک قلم داده واحد یا یک قلم داده‌ی واحد و زیرقلم داده‌های آن استفاده شوند. آنها گزینه‌هایی برای تعیین آیکون قلم داده و رنگ نوشته و پس زمینه دارند.

سازنده‌ها

```
public ListViewItem(string text);
public ListViewItem(string[] items);
public ListViewItem(string text,int imageIndex);
public ListViewItem(string[] items,int imageIndex);
public ListViewItem(string[] items,int imageIndex,
Color foreColor,Color backColor,Font font);
```

قطعه کد زیر، نحوه استفاده از overloadهای مختلف را در ایجاد قلم داده‌ها و زیر قلم داده‌های شکل ۱۵-۸ ارائه می‌کند.

```
// Create item and three subitems
ListViewItem item۱ = new ListViewItem("Manet",۲);
item۱.SubItems.Add("۱۸۳۲");
item۱.SubItems.Add("۱۸۸۳");
item۱.SubItems.Add("France");
// Create item and subitems using a constructor only
ListViewItem item۲ = new ListViewItem (new string[]
{ "Monet", "۱۸۴۰", "۱۹۲۶", "France"}, ۳);
// Create item and subitems with blue background color
ListViewItem item۳ = new ListViewItem
(new string[] { "Cezanne", "۱۸۳۹", "۱۹۰۶", "France"}, ۱,Color.Empty,
Color.LightBlue, null);
```

برای نمایش قلم داده‌ها، آنها را به کلکسیون Items کنترل ListView اضافه کنید.

```
// Add the items to the ListView
listView۱.Items.AddRange(new ListViewItem[] {item۱,item۲,item۳,item۴,item۵});
```

تعیین آیکون‌ها

می‌توان دو کلکسیون از تصاویر به ListView اختصاص داد: LargeImageList، تصاویری را برای استفاده در نمای LargeIcon در بر دارد و SmallImageList، تصاویری برای استفاده در نماهای دیگر دارد. تصور کنید آنها آرایه‌هایی از تصاویر هستند که بوسیله پارامتر imageIndex در سازنده ListViewItem یک قلم داده اختصاص داده می‌شود. اگرچه آنها بصورت آیکون‌هایی بیان می‌شوند، ولی ممکن است تصاویری از هر قالب گرافیکی استاندارد باشند.

قطعه کد زیر دو شی ImageList ایجاد می‌کند، تصاویری به آنها اضافه می‌کند، و آنها را به خصوصیات LargeImageList و SmallImageList انتساب می‌دهد.

```
// Create two ImageList objects
ImageList imageListSmall = new ImageList();
ImageList imageListLarge = new ImageList();
imageListLarge.ImageSize = new Size(۵۰,۵۰); // Set image size
// Initialize the ImageList objects
// Can use same images in both collections since they're resized
imageListSmall.Images.Add(Bitmap.FromFile("C:\\botti.gif"));
imageListSmall.Images.Add(Bitmap.FromFile("C:\\cezanne.gif"));
imageListLarge.Images.Add(Bitmap.FromFile("C:\\botti.gif"));
imageListLarge.Images.Add(Bitmap.FromFile("C:\\cezanne.gif"));
// Add other images here
```

```
// Assign the ImageList objects to the ListView.
listView1.LargeImageList = imageListLarge;
listView1.SmallImageList = imageListSmall;
ListViewItem lvItem1 = new ListViewItem("Cezanne", 1);
```

اندیس ۱ تصاویر cezanne.gif را به عنوان آیکون‌های کوچک و بزرگ انتخاب می‌کند. تعیین اندیسی که در ImageList نباشد، آیکون اندیس صفر را در نظر می‌گیرد. اگر ImageList تعریف نشده باشد، هیچ آیکونی نمایش داده نمی‌شود. شکل ۱۲-۱۵، ListView شکل ۱۱-۱۵ را با نمای View.LargeIcon نشان می‌دهد.

```
listView1.View = View.LargeIcon;
```

شکل ۱۲-۱۵



کار با کنترل ListView

کارهای معمول اختصاص داده شده به کنترل ListView، طی کردن همه محتویات کنترل، طی کردن فقط قلم داده‌های انتخاب شده، تشخیص قلم داده‌ای که کانون را دارد، مرتب‌سازی قلم داده‌ها بوسیله هر ستونی است. در زیر قطعه کدهایی برای انجام این کارها آمده است.

طی کردن همه قلم داده‌ها یا قلم داده‌های انتخاب شده

می‌توانید foreach را برای ایجاد حلقه‌های تو در تو بکار ببرید، که یک قلم داده را انتخاب کند و سپس همه زیر قلم داده‌های قلم داده‌ی حلقه بیرونی را طی می‌کند.

```
foreach (ListViewItem lvi in listView1.Items)
{
    string row = "";
    foreach (ListViewItem.ListViewSubItem sub in lvi.SubItems)
    {
        row += " " + sub.Text;
    }
    MessageBox.Show(row); // List concatenated subitems
}
```

در هنگام کار با این کلکسیون، باید از چند چیز با خبر باشید. اولاً، زیر قلم داده اول (اندیس صفر) در واقع متن قلم داده را در بر دارد و زیر قلم داده نیست. ثانیاً، زیر قلم داده‌ها با مرتب کردن ستون‌های کنترل ListView تحت تأثیر قرار نمی‌گیرد. این عمل ظاهر را تغییر می‌دهد، اما ترتیب اصلی زیر قلم داده‌ها را تحت تأثیر قرار نمی‌دهد.

همان منطق برای لیست کردن فقط قلم داده‌های انتخاب شده استفاده می‌شود. تنها تفاوت این است که طی کردن لیست روی کلکسیون listView1.SelectedItems رخ می‌دهد.

```
foreach (ListViewItem lvisel in listView1.SelectedItems)
```

تشخیص قلم داده انتخاب شده جاری

علاوه بر رویدادهای اصلی همچون Click و DoubleClick، کنترل ListView رویداد دیگری بنام SelectedIndexChanged دارد که زمان انتقال کانون از یک قلم داده به دیگری رخ می‌دهد. قطعه کد زیر در یک اداره کننده رویداد، خصوصیت FocusedItem را برای تعیین قلم داده‌ی جاری بکار می‌برد.

```
// Set this in the constructor
listView1.SelectedIndexChanged += new EventHandler(lv_IndexChanged);
// Handle SelectedIndexChanged Event
private void lv_IndexChanged(object sender, System.EventArgs e)
{
    string ItemText = listView1.FocusedItem.Text;
}
```

توجه کنید که می‌توان این کد را در رویداد Click نیز بکار برد، چون آنها نیز نماینده EventHandler را بکار می‌برند. رویدادهای MouseDown و MouseUp برای تشخیص قلم داده جاری استفاده می‌شوند. این نمونه‌ای از یک اداره کننده رویداد MouseDown است.

```
private void listView1_MouseDown(object sender, MouseEventArgs e)
{
    ListViewItem selection = listView1.GetItemAt(e.X, e.Y);
    if (selection != null)
    {
        MessageBox.Show("Item Selected: "+selection.Text);
    }
}
```

متد ListViewItem.GetItemAt قلم داده‌ی موجود در محلی که دکمه ماوس فشار داده می‌شود، را بر می‌گرداند. اگر ماوس روی قلم داده‌ای نباشد، null برگردانده می‌شود.

مرتب‌سازی قلم داده‌های یک کنترل ListView

مرتب‌سازی قلم داده‌های یک ListView بوسیله مقادیر ستون‌ها، یک ویژگی ساده آن جهت پیاده‌سازی است. راز سادگی آن خصوصیت ListViewItemSorter است، که یک شی را برای مرتب‌کردن قلم داده‌ها در زمان فراخوانی متد ListView.Sort مشخص می‌کند. پیاده‌سازی در سه مرحله انجام می‌شود.

۱- یک نماینده برای متصل کردن رویداد ColumnClick به یک اداره کننده رویداد برقرار کنید.

۲- یک متد اداره کننده‌ی رویداد ایجاد کنید که خصوصیت ListViewItemSorter را یک نمونه از کلاس قرار دهید که عملیات مقایسه را در مرتب‌سازی انجام می‌دهد.

۳- یک کلاس برای مقایسه کردن مقادیر ستون‌ها ایجاد کنید. آن باید واسط IComparer را وراثت کند و متد IComparer.Compare را پیاده‌سازی کند.

کد زیر منطق را پیاده‌سازی می‌کند. زمانی که یک ستون کلیک می‌شود، اداره کننده رویداد یک نمونه از کلاس ListViewItemComparer را بوسیله ارسال ستون کلیک شده ایجاد می‌کند. این شی به خصوصیت ListViewItemSorter انتساب داده می‌شود که منجر به مرتب‌سازی می‌شود.

```
// Connect the ColumnClick event to its event handler
listView1.ColumnClick += new ColumnClickEventHandler(ColumnClick);
// ColumnClick event handler
private void ColumnClick(object o, ColumnClickEventArgs e)
{
    // Setting this property immediately sorts the
    // ListView using the ListViewItemComparer object
```

```

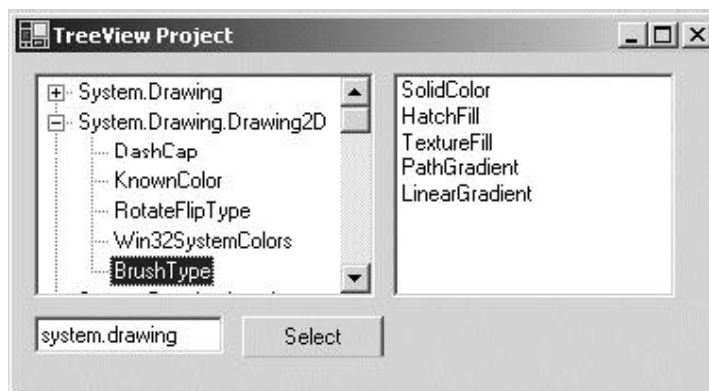
        this.listView1.ListViewItemSorter = new ListViewItemComparer(e.Column);
    }
    // Class to implement the sorting of items by columns
    class ListViewItemComparer : IComparer
    {
        private int col;
        public ListViewItemComparer()
        {
            col = 0; // Use as default column
        }
        public ListViewItemComparer(int column)
        {
            col = column;
        }
        // Implement IComparer.Compare method
        public int Compare(object x, object y)
        {
            string xText = ((ListViewItem)x).SubItems[col].Text;
            string yText = ((ListViewItem)y).SubItems[col].Text;
            return String.Compare(xText, yText);
        }
    }
}

```

۱۵-۵-۲-کلاس TreeView

همانطور که از اسمش پیداست، کنترل TreeView یک نمای درخت‌گونه از داده‌های سلسله‌مراتبی به عنوان واسطه کاربر فراهم می‌سازد. مدل برنامه‌نویسی آن مبتنی بر ساختار درخت شامل گره‌های پدر و فرزند است. هر گره بصورت یک شیء TreeNode پیاده‌سازی می‌شود و می‌تواند کلکسیون Nodes خود را داشته باشد. شکل ۱۵-۱۳ یک کنترل TreeView را نشان می‌دهد، که در ارتباط با یک ListView جهت نمایش اعضای شمارشی یک اسمبلی انتخاب شده بکار می‌رود.

شکل ۱۵-۱۳



کلاس TreeNode

هر قلم داده در یک درخت بوسیله یک نمونه از کلاس TreeNode نمایش داده می‌شود. با استفاده از خصوصیات Tag، Text، یا ImageIndex، داده‌ها به هر گروه اختصاص داده می‌شود. خصوصیت Text برچسب گره را نگه می‌دارد، که در کنترل TreeView نشان داده نمی‌شود. Tag یک نوع داده object است، که می‌توانیم هر شیء سفارشی را به آن انتساب دهیم. ImageIndex اندیسی از یک ImageList انتساب داده شده به کنترل TreeView است. آن تصویر یک گره را مشخص می‌کند.

علاوه بر این خصوصیات اصلی، کلاس TreeNode اعضای متعدد دیگری فراهم می‌سازد، که برای اضافه کردن گره‌ها، تغییر ظاهر یک گره، هدایت گره‌ها در یک گره بکار می‌روند (جدول ۱۵-۳ را ببینید).

جدول ۱۵-۳

کاربرد	عضو	توصیف
ظاهر	BackColor, ForeColor	رنگ متن و پس زمینه گره را قرار می‌دهد.
	Expand(), Collapse()	گره را برای نمایش فرزندان باز می‌کند یا برای پنهان کردن فرزندان گره را می‌بندد.
هدایت	FirstNode, LastNode, NextNode, PrevNode	اولین یا آخرین گره کلکسیون را بر می‌گرداند. گره قبلی یا بعدی مرتبط با گره جاری را بر می‌گرداند.
	Index	اندیس گره جاری در کلکسیون است.
	Parent	پدر گره جاری را بر می‌گرداند.
دستکاری گره	Nodes.Add(), Nodes.Remove(), Nodes.Insert(), Nodes.Clear()	یک گره به کلکسیون Nodes اضافه یا حذف می‌کند. Insert یک گره به یک موقعیت با اندیس مشخص اضافه می‌کند و متد Clear همه گره های درخت را از کلکسیون پاک می‌کند.
	Clone()	یک گره و کل زیر درخت آن را کپی می‌کند.

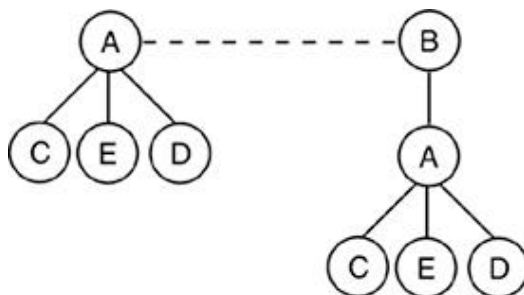
حال بررسی می‌کنیم چگونه اعضای TreeView و TreeNode برای انجام عملیات پایه‌ای TreeView استفاده می‌شوند.

اضافه کردن و حذف کردن گره‌ها

کد زیر با استفاده از ترکیب متدهای Add، Insert و Clone درخت شکل ۱۵-۱۴ را ایجاد می‌کند. متدها روی کنترل TreeView۱ موجود انجام می‌شوند.

```
TreeNode tNode;
// Add parent node to treeView1 control
tNode = treeView1.Nodes.Add("A");
// Add child node: two overloads available
tNode.Nodes.Add(new TreeNode("C"));
tNode.Nodes.Add("D");
// Insert node after C
tNode.Nodes.Insert(1, new TreeNode("E"));
// Add parent node to treeView1 control
tNode = treeView1.Nodes.Add("B");
```

شکل ۱۵-۱۴



در این نقطه، هنوز نیاز داریم یک کپی از گره A و زیر درختش را به گره پدر B اضافه کنیم. این عمل با کپی کامل زیر درخت A و اضافه کردن آن به گره B انجام می‌شود. گره A با treeView1.Nodes[۰] ارجاع داده می‌شود، چون آن اولین گره

کلکسیون کنترل است. توجه کنید که متد Add گره‌ها را به یک کلکسیون الحاق می‌کند. آنها می‌توانند بوسیله اندیس موقعیت در کلکسیون ارجاع داده شوند.

```
// Clone first parent node and add to node B
TreeNode clNode = (TreeNode) treeView1.Nodes[0].Clone();
tNode.Nodes.Add(clNode);
// Add and remove node for demonstration purposes
tNode.Nodes.Add("G");
tNode.Nodes.Remove(tNode.LastNode);
```

طی کردن همه گره‌ها در یک TreeView

همانند هر کلکسیون، دستور foreach ساده‌ترین راه برای طی کردن اعضای کلکسیون فراهم می‌کند. دستورات زیر همه گره‌های سطح بالای یک کنترل را نشان می‌دهند.

```
foreach (TreeNode tn in treeView1.Nodes)
{
    MessageBox.Show(tn.Text);
    // If (tn.IsVisible) true if node is visible
    // If (tn.IsSelected) true if node is currently selected
}
```

روش دیگر طی کردن کلکسیون از طریق خصوصیت treeNode.NextNode است.

```
tNode = treeView1.Nodes[0];
while (tNode != null) {
    MessageBox.Show(tNode.Text);
    tNode = tNode.NextNode;
}
```

تشخیص گره انتخاب شده

زمان انتخاب یک گره، کنترل TreeView یک رویداد AfterSelect رها می‌سازد که یک پارامتر TTreeViewEventArgs به کد اداره کننده رویداد ارسال می‌کند. این پارامتر عملی که باعث انتخاب گره شده و گره انتخاب شده را معین می‌کند. مثال TreeView زیر نحوه اداره این رویداد را نشان می‌دهد. می‌توانید رویداد MouseDown را اداره کرده و با استفاده از متد GetNodeAt گره را تشخیص دهید که گره موجود در مختصات جاری ماوس را بر می‌گرداند.

```
private void treeView1_MouseDown(object sender, MouseEventArgs e)
{
    TreeNode tn = treeView1.GetNodeAt(e.X, e.Y);
    // You might want to remove the node: tn.Remove()
}
```

یک مثال TreeView با کاربرد انعکاس

این مثال نحوه ایجاد یک کاوشگر ساده را نشان می‌دهد (شکل ۱۵-۱۳) که یک TreeView برای نمایش انواع شمارشی یک اسمبلی خاص را بکار می‌برد. زمانی که یک گره روی درخت کلیک می‌شود، اعضای نوع شمارشی انتخاب شده در یک کنترل ListView نمایش داده می‌شود.

اطلاعات یک اسمبلی در فرا داده‌ی آن ذخیره می‌شود و NET در فضای نامی System.Reflection کلاس‌هایی برای بدست آوردن این فراداده‌ها فراهم می‌سازد. مثال ۱۵-۴ انواع داده‌ای یک اسمبلی را طی می‌کند تا یک TreeView ایجاد کند. گره‌های پدر اسامی فضاهای نامی منحصر به فرد را دارند و گره‌های فرزند انواع داده‌ای داخل فضای یک اسمبلی را طی می‌کند تا یک TreeView ایجاد کند. گره‌های پدر اسامی فضاهای نامی منحصر به فرد را دارند و گره‌های فرزند انواع داده‌ای داخل فضاهای نامی را در بردارند.

برای اینکه فقط انواع enum را در بر گیرد، یک بررسی انجام می‌شود تا مطمئن گردد نوع داده از system.Enum ارث‌بری می‌کند.

```
using System.Reflection;
//
private void GetEnums()
{
    TreeNode tNode=null;
    Assembly refAssembly ;
    Hashtable ht= new Hashtable(); // Keep track of namespaces
    string assem = AssemName.Text; // Textbox with assembly name
    tvEnum.Nodes.Clear(); // Remove all nodes from tree
    // Load assembly to be probed
    refAssembly = Assembly.Load(assem);
    foreach (Type t in refAssembly.GetTypes())
    {
        // Get only types that inherit from System.Enum
        if(t.BaseType!=null && t.BaseType.FullName=="System.Enum")
        {
            string myEnum = t.FullName;
            string nSpace =
            myEnum.Substring(0,myEnum.LastIndexOf("."));
            myEnum= myEnum.Substring(myEnum.LastIndexOf(".")+1) ;
            // Determine if namespace in hashtable
            if( ht.Contains(nSpace))
            {
                // Find parent node representing this namespace
                foreach (TreeNode tp in tvEnum.Nodes)
                {
                    if(tp.Text == myEnum) { tNode=tp; break;}
                }
            }
            else
            {
                // Add parent node to display namespace
                tNode = tvEnum.Nodes.Add(nSpace);
                ht.Add(nSpace,nSpace);
            }
            // Add Child - name of enumeration
            TreeNode cNode = new TreeNode();
            cNode.Text= myEnum;
            cNode.Tag = t; // Contains specific enumeration
            tNode.Nodes.Add(cNode);
        }
    }
}
```

توجه کنید انعکاس چگونه استفاده می‌شود. متد ایستای Assembly.Load برای ایجاد یک نوع داده Assembly استفاده می‌شود. متد Assembly.GetTypes یک آرایه Type شامل همه انواع داده‌ای طراحی شده در اسمبلی برمی‌گرداند.

```
refAssembly = Assembly.Load(assem);
foreach (Type t in refAssembly.GetTypes())
```

خصوصیت Type.FullName نام نوع داده را بر می‌گرداند که فضای نامی را نیز شامل است. این برای استخراج نام enum و نام فضای نامی استفاده می‌شود. Type در فیلد Tag از گره‌های فرزند ذخیره می‌شود و بعداً برای بازیابی اعضای enum استفاده می‌شود.

بعد از اینکه TreeView ساخته شد، نمایش اعضای یک نوع شمارشی هنگام کلیک بر روی گره کار نهایی آن می‌باشد. لازم است یک اداره کننده رویداد ثبت شود تا در زمان رخ دادن رویداد AfterSelect اعلام گردد.

```
tvEnum.AfterSelect += new
TreeViewEventHandler(tvEnum_AfterSelect);
```

اداره کننده رویداد از طریق خصوصیت `treeViewEventArgs.Node`، گره انتخاب شده را معین می‌کند. آن فیلد `Tag` گره را به یک کلاس `Type` قالب‌بندی می‌کند و متد `GetMembers` را برای بازیابی اعضای نوع داده بصورت انواع داده `MemberInfo` بکار می‌برد. نام هر عضو با خصوصیت `MemberInfo.Name` در اختیار قرار می‌گیرد که در `ListView` نمایش داده می‌شود.

```
// ListView lView;
// lView.View = View.List;
private void tvEnum_AfterSelect(Object sender, TreeViewEventArgs e)
{
    TreeNode tn = e.Node; // Node selected
    ListViewItem lvItem;
    if(tn.Parent != null) // Exclude parent nodes
    {
        lView.Items.Clear(); // Clear ListView before adding items
        Type cNode = (Type) tn.Tag;
        // Use Reflection to iterate members in a Type
        foreach (MemberInfo mi in cNode.GetMembers())
        {
            if(mi.MemberType==MemberTypes.Field &&mi.Name != "value__" )
                // skip this

            {
                lView.Items.Add(mi.Name);
            }
        }
    }
}
```

۱۵-۶- کلاس‌های `StatusStrip` و `Timer`، `ProgressBar`

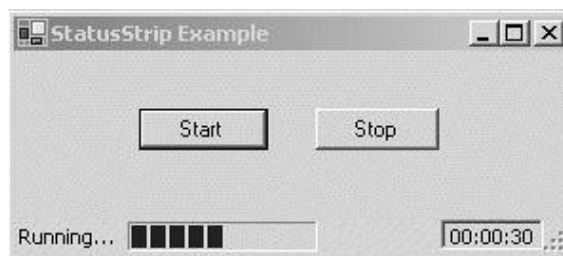
`ProgressBar` و `Timer` کنترل‌های سبک وزنی هستند که نقش‌های مکمل در یک برنامه کاربردی دارند. `Timer` یک عملی را آغاز می‌کند و `ProgressBar` حالت یک عمل یا عملکرد آن را منعکس می‌کند. در حقیقت `Timer` یک کنترل نیست، اما یک قطعه‌ای است که از کلاس `ComponentModel.Component` ارث‌بری می‌کند. آن اغلب اوقات در پروسه‌ها برای تنظیم برخی فعالیت پشت‌زمینه استفاده می‌شود. آن فعالیت ممکن است بروزآوری دوره‌ای یک فایل یا یک پشتیبان‌گیری زمان‌بندی شده از داده‌ها باشد. از طرف دیگر، `ProgressBar` یک بازخورد بصری از پیشرفت یک عمل را فراهم می‌کند (همچون کپی کردن فایل یا مراحل یک نصب).

سومین کلاس بحث شده در این بخش `StatusStrip` است که اغلب در ارتباط با یک `Timer` و `ProgressBar` استفاده می‌شود. آن کنترل در روی فرم شبیه نواری است که به یک یا چند بخش تقسیم می‌شود یا تکه‌هایی که اطلاعات حالت را فراهم می‌سازند. هر بخش بصورت یک کنترل پیاده‌سازی می‌شود، که در یک ظرف `StatusStrip` اضافه می‌شوند. برای اینکه یک کنترل در `StatusStrip` قرار گیرد، باید از کلاس `ToolStripItem` ارث‌بری کرده باشد.

ایجاد یک StatusStrip

حال اجازه دهید یک فرم شامل یک StatusStrip چند تکه ایجاد کنیم. همانطور که در شکل ۱۵-۱۵ نشان داده شده است، نوار پایین فرم یک برچسب، میله پیشرفت و کنترل‌های پانل را شامل است. برچسب (ToolStripLabel) اطلاعات متنی برای تشریح حالت کلی برنامه کاربردی فراهم می‌سازد. میله پیشرفت بصورت یک شی ToolStripProgressBar پیاده‌سازی می‌شود. آن از نظر عمل شبیه ProgressBar است، اما از ToolStripItem ارث‌بری می‌کند. یک StatusStripPanel وقت سپری شده از زمان نمایش فرم را نشان می‌دهد. یک اداره کننده رویداد بوسیله زمان‌سنج کنترل می‌شود تا در هر ۵ ثانیه میله پیشرفت و پانل ساعت را به‌نگام سازد.

شکل ۱۵-۱۵



مثال ۱۵-۵ کد تولید StatusStrip را در بر دارد. سرهای چپ و راست میله پیشرفت به ترتیب مقادیر ۰ و ۱۲۰ را نمایش می‌دهند. زمانی که متد PerformStep اجرا می‌شود، در هر مرحله به اندازه ۱۰ واحد میله رشد می‌کند. در هر دقیقه تکرار می‌شود.

Timer کنترل می‌کند چه زمانی میله رشد کند و وقت سپری شده بروز شود. خصوصیت Interval آن مقداری قرار داده می‌شود، که فرکانس رهایی رویداد Tick را کنترل می‌کند. در این مثال، در هر ۵ ثانیه رویداد رخ می‌شود، که میله‌ی پیشرفت ۱۰ واحد رشد می‌کند و وقت به اندازه ۵ ثانیه سپری می‌شود.

مثال ۱۵-۵

```
// These variables have class scope
Timer currTimer;
StatusStrip statusStrip1;
StatusStripPanel panel1;
ToolStripProgressBar pb;
DateTime startDate = DateTime.Now;
private void BuildStrip()
{
    currTimer = new Timer();
    currTimer.Enabled = true;
    currTimer.Interval = ۵۰۰۰; // Fire tick event every ۵ seconds
    currTimer.Tick += new EventHandler(timer_Tick);
    // Panel to contain elapsed time
    panel1 = new StatusStripPanel();
    panel1.BorderStyle = Border3DStyle.Sunken;
    panel1.Text = "...:..:..";
    panel1.Padding = new Padding(۲);
    panel1.Name = "clock";
    panel1.Alignment = ToolStripItemAlignment.Right; //Right align
    // Label to display application status
    ToolStripLabel ts = new ToolStripLabel();
    ts.Text = "Running...";
    // ProgressBar to show time elapsing
    pb = new ToolStripProgressBar();
```

```

pb.Step = ۱۰; // Size of each step or increment
pb.Minimum = ۰;
pb.Maximum = ۱۲۰; // Allow ۱۲ steps
// Status strip to contain components
statusStrip۱ = new StatusStrip();
statusStrip۱.Height = ۲۰;
statusStrip۱.AutoSize = true;
// Add components to strip
statusStrip۱.Items.AddRange(new ToolStripItem[] {
ts, pb, panel۱ } );
this.Controls.Add(statusStrip۱);
}
private void timer_Tick(object sender, EventArgs e)
{
    // Get difference between current datetime
    // and form startup time
    TimeSpan ts = DateTime.Now.Subtract(startDate);
    string elapsed = ts.Hours.ToString("00") + ":" + ts.Minutes.ToString("00") +
        ":" + ts.Seconds.ToString("00");

    ((StatusStripPanel)statusStrip۱.Items["clock"]).Text= elapsed;
    // Advance progress bar
    if (pb.Value == pb.Maximum) pb.Value = ۰;
    pb.PerformStep(); // Increment progress bar
}

```

StatusStripPanel که وقت سپری شده را نشان می‌دهد، چندین خصوصیت دارد که ظاهر و موقعیت آن را کنترل می‌کند. علاوه بر موارد نشان داده شده در اینجا، آن یک خصوصیت Image برای نمایش یک تصویر دارد. کلاس StatusStripPanel از کلاس ToolStripLabel ارث‌بری می‌کند. هر دو می‌توانند برای نمایش متن بکار روند، اما پانل یک خصوصیت BorderStyle دارد که ToolStripLabel فاقد آن است.

۱۵-۷- ایجاد کنترل‌های سفارشی

در بعضی مواقع، با یک کار برنامه‌نویسی روبرو خواهید شد که کنترل‌های استاندارد موجود نیاز شما را برآورده نمی‌کند. ممکن است بخواهید یک کنترل TextBox را بسط دهید تا متناسب با محتوای آن رنگ پس زمینه تغییر کند. یک مجموعه مکرراً استفاده شده از دکمه‌های رادیویی را در یک کنترل واحد گروه‌بندی کنید یا کنترل جدید ایجاد کنید که یک ساعت دیجیتال با تاریخی در زیر آن نشان می‌دهد. این نیازها با سه نوع اساسی از کنترل‌های سفارشی مرتبط هستند:

- ۱- کنترلی که از یک کنترل موجود مشتق می‌شود و عملیات آن را بسط می‌دهد.
- ۲- یک کنترل که به عنوان یک ظرف بکار می‌رود تا اجازه دهد چندین کنترل باهم تعامل داشته باشند. این نوع کنترل به عنوان یک کنترل کاربری بیان می‌شوند. آن به جای Control مستقیماً از System.Windows.Forms.UserControl مشتق می‌شود و کنترل‌های استاندارد را انجام می‌دهد.
- ۳- کنترلی که مستقیماً از کلاس Control مشتق می‌شود. این نوع کنترل از ابتدا ساخته می‌شود و توسعه‌دهنده مسئولیت ترسیم واسطه GUI آن، پیاده‌سازی متدها و خصوصیات آن را بر عهده دارد و باید طوری باشد که بوسیله کد قابل دستکاری باشد. حال نحوه بسط یک کنترل موجود و ایجاد یک کنترل کاربری را بررسی می‌کنیم.

۱۵-۷-۱- بسط یک کنترل

ساده‌ترین راه ایجاد یک کنترل سفارشی، بسط یک کنترل موجود است. برای شرح دادن آن، یک کلاس از TextBox مشتق کنید که فقط ارقام را می‌پذیرد که کنترل کاملاً ساده‌ای است. کلاس جدید NumericTextBox را با TextBox به عنوان کلاسی پایه آن ایجاد کنید. تنها کد مورد نیاز یک اداره کننده رویداد برای پردازش رویداد KeyPress است و فقط یک رقم می‌پذیرد.

```
class NumericTextBox: TextBox
{
    public NumericTextBox()
    {
        this.KeyPress += new KeyPressEventHandler(TextBoxKeyPress);
    }
    protected void TextBoxKeyPress(object sender,
        KeyPressEventArgs e)
    {
        if (! char.IsDigit(e.KeyChar)) e.Handled = true;
    }
}
```

بعد از اینکه کنترل بسط یافته به یک فایل DLL کامپایل شود، می‌تواند روی هر فرمی اضافه شود.

۱۵-۷-۲- ایجاد یک UserControl سفارشی

یک کنترل کاربری را همانند یک فرم تصور کنید. شبیه یک فرم، آن ظرفی را فراهم می‌سازد که چیزهای مرتبط روی آن جای می‌گیرند. بعد از کامپایل، کل مجموعه کنترل‌های روی آن بصورت یک کنترل کاربری واحد رفتار می‌کنند. البته کاربران هنوز می‌خواهند با هر کدام از کنترل‌های عضو مستقیماً تعامل داشته باشند. دسترسی به اعضای کنترل در زمان طراحی و برنامه‌نویسی از طریق متدها و خصوصیات تعریف شده روی کنترل کاربری در دسترس است.

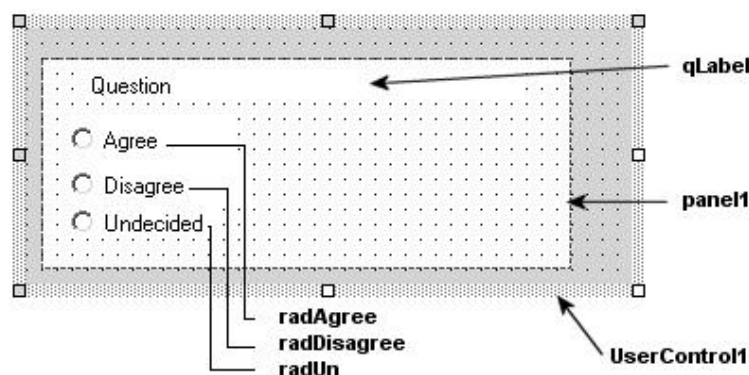
ساده‌ترین راه طراحی یک کنترل بوسیله یک IDE همچون VS.Net است که تغییر اندازه و محل کنترل‌ها را ساده می‌سازد. روش معمول ایجاد یک کنترل کاربری در VS.NET بار کردن یک پروژه از نوع Windows Control Library است. این عمل پنجره طراحی کنترل را سریعاً بار می‌کند. پنجره طراحی در Windows Application با انتخاب Add - Add → User Control و مسیرهای دیگر قابل اضافه کردن است. اگرچه VS.NET پروسه ایجاد یک کنترل را سرعت می‌دهد، آن هیچ کد اختصاصی تولید نمی‌کند.

یک مثال از User Control

به عنوان یک مثال، اجازه دهید یک کنترلی ایجاد کنیم که می‌تواند برای ایجاد یک پرسشنامه بکار رود. کنترل یک برچسب برای نمایش سؤال و سپس دکمه رادیویی بر روی یک کنترل پانل برای نمایش گزینه‌های جواب را در بر دارد. کنترل سه خصوصیت دارد: یکی برای انتساب سؤال به برچسب، یکی برای تنظیم رنگ پس زمینه کنترل Panel و دیگری دکمه رادیویی انتخاب شده توسط کاربر به عنوان جواب را تعیین می‌کند.

شکل ۱۵-۱۶ طرح‌بندی کنترل کاربر و اسامی انتساب شده به هر کنترل را نشان می‌دهد.

شکل ۱۵-۱۶



در اینجا نحوه نمایش اعضا بصورت فیلدهایی از کلاس UserControl1 نشان داده می‌شود.

```
public class UserControl1 : System.Windows.Forms.UserControl
{
    private Panel panel1;
    private RadioButton radAgree;
    private RadioButton radDisagree;
    private RadioButton radUn;
    private Label qLabel;
```

مثال ۱۵-۶ که سه خصوصیت را شامل است. SetQ خصوصیت Text برچسب را با سؤال مقداردهی می‌کند. PanelColor رنگ پانل را قرار می‌دهد و Choice جواب انتخاب شده بوسیله کاربر را بصورت یک نوع داده شمارشی Choices برمی‌گرداند.

```
public enum Choices
{
    Agree = ۱,
    DisAgree = ۲,
    Undecided = ۳,
}

public string SetQ
{
    set { qLabel.Text = value; }
    get { return (qLabel.Text); }
}

public Color PanelColor
{
    set { panel1.BackColor= value; }
    get { return (panel1.BackColor); }
}

public Choices Choice
{
    get
    {
        Choices usel;
        usel = Choices.Undecided;
        if (radDisagree.Checked) usel= Choices.DisAgree;
        if (radAgree.Checked) usel = Choices.Agree;
        return (usel); }
}
}
```

استفاده از UserControl سفارشی

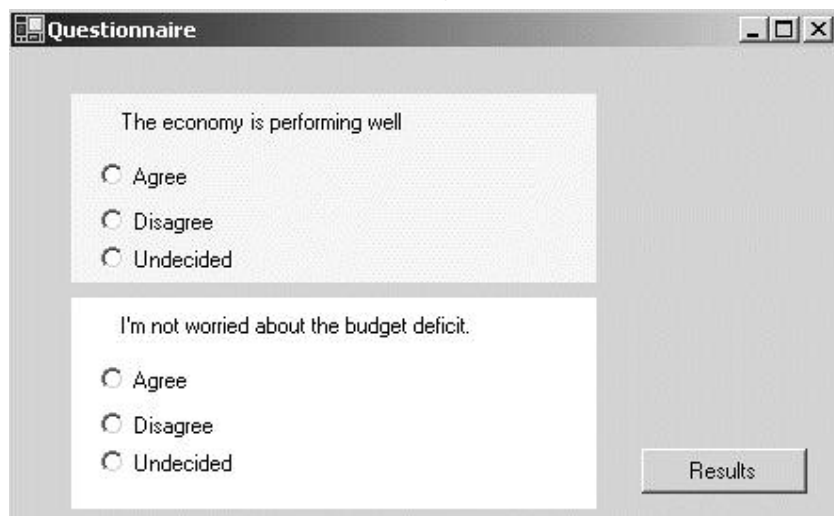
اگر کنترل کاربری به عنوان بخشی از پروژه برنامه کاربردی ویندوزی VS.NET توسعه داده شود، آن بطور اتوماتیک به جعبه ابزار در زیر برگه Windows Forms اضافه می‌شود. بطور ساده آن را انتخاب کرده و روی فرم بکشید. در غیر اینصورت باید

روی برگه جعبه ابزار کلیک راست کرده و Customize ToolBox را انتخاب کنید، کنترل را جستجو کرده و آن را به جعبه ابزار اضافه کنید.

شکل ۱۵-۱۷ یک مثال از کاربرد کنترل جدید را فراهم می‌سازد. در این مثال، دو نمونه از کنترل بنام Q۱ و Q۲ روی فرم قرار دارند.

```
private usercontrol.UserControl۱ Q۱;
private usercontrol.UserControl۱ Q۲;
```

شکل ۱۵-۱۷



می‌توان در زمان اجرا، در اداره کننده رویداد Form.Load یا در سازنده خصوصیات را مقداردهی کرد. اگر VS.NET را بکار می‌برید، می‌توان در زمان طراحی بوسیله Property Browser خصوصیات را مقداردهی کرد.

```
Q۱.SetQ = "The economy is performing well";
Q۲.SetQ = "I'm not worried about the budget deficit.";
Q۱.PanelColor = Color.Beige;
```

مرحله نهایی در برنامه کاربردی، انجام دادن کارهایی است که بعد از کامل شدن پرسشنامه لازم هستند. زمانی که روی دکمه کلیک می‌شود، کد زیر همه کنترل‌های روی فرم را طی می‌کند. اگر نوع کنترل UserControl باشد، خصوصیت Choice آن برای برگرداندن جواب کاربر استفاده می‌شود.

```
private void button۱_Click(object sender, System.EventArgs e)
{
    foreach (Control ct in this.Controls)
    {
        if (ct is usercontrol.UserControl۱)
        {
            UserControl۱ uc = (UserControl۱)ct;
            // Display control name and user's answer
            MessageBox.Show(ct.Name+" "+uc.Choice.ToString());
        }
    }
}
```

کار با UserControl در زمان طراحی

اگر یک برنامه کاربردی بوسیله VS.NET توسعه می‌دهید که کنترل سفارشی بکار می‌برد، شما خواهید دید که Property Browser همه خصوصیات خواندنی / نوشتنی را لیست می‌کند. بطور پیش فرض، آنها در یک طبقه Misc قرار داده می‌شوند

و هیچ توصیفی به آنها اختصاص نیافته است. برای حرفه‌ای کردن کنترل‌های خودتان، باید یک طبقه^۱ برای رویدادها و خصوصیات کنترل ایجاد کنید و یک توصیف متنی برای هر عضو طبقه اضافه کنید.

طبقه‌ها و توصیف‌ها در Property Browser موجود هستند که از فراداده‌های مبتنی بر صفات الحاق شده به اعضای یک نوع داده می‌آیند. این یک مثال از صفات اضافه شده به خصوصیت PanelColor است.

```
[Browsable(true), Category("QControl"), Description("Color of panel behind
question block")]
public Color PanelColor
{
    set {panel.BackColor = value;}
    get {return (panel.BackColor);}
}
```

صفت Browsable مشخص می‌کند آیا خصوصیت در کاوشگر نمایش داده شود. بطور پیش‌فرض true است. دو صفت دیگر، طبقه‌ای که خصوصیت در زیر آن نمایش داده می‌شود و متنی که در هنگام انتخاب خصوصیت در زیر Property Browser ظاهر می‌گردد را مشخص می‌کنند.

همواره به خاطر داشته باشید که انگیزه ایجاد کنترل‌های کاربری سفارشی قابلیت استفاده مجدد است. اگر کنترلی فقط یک بار استفاده می‌شود، نیازی نیست زمانی برای آن صرف شود.

۱۵-۸- استفاده از کشیدن و انداختن بوسیله کنترل‌ها

توانایی کشیدن داده‌ها از یک کنترل و انداختن به کنترل دیگر یک ویژگی آشنای برنامه‌نویسی GUI بوده است. NET. بوسیله چندین کلاس و نوع شمارشی این ویژگی را پشتیبانی می‌کند، تا یک کنترل قادر باشد مقصد یا مبدأ عمل کشیدن و انداختن باشد.

مروری بر کشیدن و انداختن

عمل مورد نظر، یک کنترل مبدأ که داده‌هایی برای انتقال یا کپی دارد و یک کنترل مقصد که داده‌های کشیده شده را دریافت می‌کند لازم دارد. مبدأ عمل مورد نظر را در پاسخ به یک رویداد (معمولاً رویداد MouseDown) آغاز می‌کند. اداره کننده رویداد کنترل مبدأ عمل واقعی را بوسیله احضار متد DoDragDrop شروع می‌کند. این متد دو پارامتر دارد: داده‌ای که کشیده می‌شود و یک پارامتر نوع شمارشی DragDropEffects که اثرات و عملکردهایی که کنترل مبدأ پشتیبانی می‌کند را مشخص می‌کند. (جدول ۱۵-۴ را ببینید).

جدول ۱۵-۴

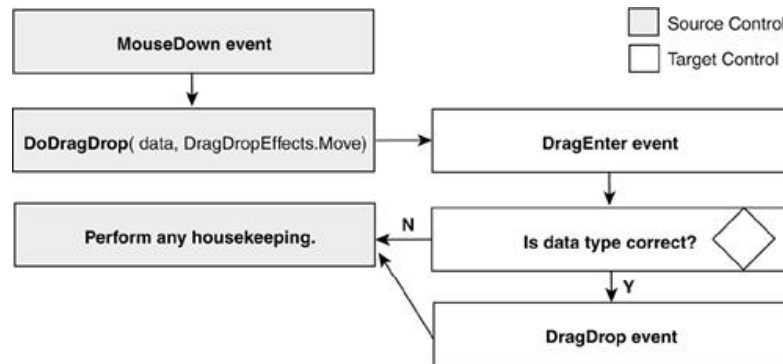
عضو	توصیف
All	داده به کنترل مقصد منتقل می‌شود و در کنترل مقصد برای نمایش محل داده‌های جدید عمل لغزاندن رخ می‌دهد.
Copy	داده‌ها از مبدأ به مقصد کپی می‌شوند.
Link	داده‌ها از مبدأ به مقصد پیوند داده می‌شوند.
Move	داده‌ها از مبدأ به مقصد منتقل می‌شوند.

^۱ Category

سطح کنترل را به صورت یک شی Graphics ارائه می‌کند تا ترسیم روی آن رخ دهد. کنترل مقصد از پذیرفتن داده سر باز می‌زند.	None
لغزاندن رخ می‌دهد یا روی کنترل مقصد رخ خواهد داد.	Scroll

هنگامی که ماوس در سرتاسر فرم حرکت می‌کند، متد `DoDragDrop` کنترل زیر موقعیت جاری اشاره‌گر را تعیین می‌کند. اگر خصوصیت `AllowDrop` آن `true` باشد، آن کنترل به عنوان هدف انداختن معتبر است و رویداد `DragEnter` آن رخ می‌شود. اداره کننده رویداد `DragEnter` دو کار دارد: بررسی اینکه داده کشیده شده یک نوع داده قابل قبول است و مطمئن شود عمل درخواست شده نیز قابل پذیرش است. زمانی که عمل انداختن واقعی رخ می‌دهد، کنترل مقصد رویداد `DragDrop` رخ می‌سازد. این اداره کننده رویداد، مسئول قرار دادن داده در کنترل هدف است (شکل ۱۵-۱۸ را ببینید).

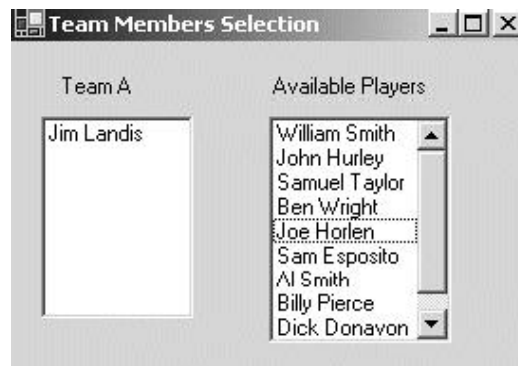
شکل ۱۵-۱۸



بعد از پایان یافتن اداره کننده رویداد `DragDrop`، کنترل مبدأ عملیات پاکسازی را انجام می‌دهد. برای مثال، اگر عملیات شامل انتقال داده باشد، بایستی داده از کنترل مبدأ حذف شود.

برای شرح دادن این ایده‌ها، یک برنامه کاربردی برای انتساب بازیکنان از یک فهرست بازیکنان به تیم ایجاد کنید (شکل ۱۵-۱۹ را ببینید). `TeamA` با کشیدن اسامی از `Available` به لیست `TeamA` ایجاد می‌شود. هر دو لیست بوسیله کادرهای لیست پیاده‌سازی می‌شوند و لیست `Available` طوری تنظیم شده که می‌توان فقط یک قلم داده انتخاب کرد.

شکل ۱۵-۱۹



یک نام با فشار دادن دکمه سمت راست ماوس و کشیدن آن به لیست هدف انتخاب می‌شود. برای جالب کردن عمل، نگه داشتن کلید `Ctrl` به جای انتقال نام یک کپی از آن ایجاد می‌کند. بعد از ایجاد فرم و کنترل‌های آن، گام اول تنظیم کنترل مبدأ (`lstPlayers`) برای پاسخ دادن به رویداد `MouseDown` و کنترل هدف (`lstTeamA`) برای اداره کردن رویدادهای `DragEnter` و `DragDrop` می‌باشد:

```
lstPlayers.MouseDown +=
new MouseEventHandler(Players_MouseDown);
```

```
lstTeamA.DragEnter += new DragEventHandler(TeamA_DragEnter);
lstTeamA.DragDrop += new DragEventHandler(TeamA_Drop);
```

گام بعدی کدنویسی اداره کننده رویداد روی کنترل‌های مبدأ و هدف است، که عملیات کشیدن و انداختن را پیاده‌سازی می‌کنند.

مسئولیت‌های کنترل مبدأ

اداره کننده رویداد MouseDown ابتدا جهت اطمینان بررسی می‌کند یک قلم داده از ListBox انتخاب شده باشد. سپس متد DoDragDrop را فراخوانی می‌کند و مقدار عنصر انتخاب شده را به همراه تأثیرات قابل پذیرش (Copy و Move) به آن ارسال می‌کند. نوع شمارشی DragDropEffects یک صفت FlagsAttribute دارد که ترکیب بیتی مقادیر داده شده به آن را ممکن می‌سازد. مقدار برگردانده شده از این متد، همان تأثیری است که توسط کنترل هدف استفاده می‌شود. اداره کننده رویداد از این اطلاعات برای انجام دادن هر عمل مورد نیاز جهت پیاده‌سازی تأثیر استفاده می‌کند. در این مثال، یک عمل انتقال بدین معنی است که باید مقدار کشیده شده از کنترل مبدأ حذف شود.

```
private void Players_MouseDown(object sender, MouseEventArgs e)
{
    if ( lstPlayers.SelectedIndex >=0)
    {
        string players;
        int ndx = lstPlayers.SelectedIndex;
        DragDropEffects effect;
        players = lstPlayers.Items[ndx].ToString();
        if(players != "")
        {
            // Permit target to move or copy data
            effect = lstPlayers.DoDragDrop(players,
            DragDropEffects.Move | DragDropEffects.Copy);
            // Remove item from ListBox since move occurred
            if (effect == DragDropEffects.Move)
                lstPlayers.Items.RemoveAt (ndx);
        }
    }
}
```

مسئولیت‌های کنترل هدف

کنترل مقصد باید اداره کننده‌های رویدادهای DragEnter و DragDrop را پیاده‌سازی کند. هر دو رویداد یک پارامتر از نوع MouseEventArgs می‌گیرند که اطلاعات مورد نیاز جهت پردازش رویداد کشیدن و انداختن را در بر دارند. (جدول ۱۵-۵ را ببینید)

جدول ۱۵-۵

کاربرد	توصیف
AllowedEffect	تأثیراتی که بوسیله کنترل مبدأ پشتیبانی می‌شوند. مثلاً اگر Move پشتیبانی شود. <pre>== (if ((e.AllowedEffect & DragDropEffects.Move (DragDropEffects.Move</pre>
Data	یک IDataObject بر می‌گرداند که داده تخصیص یافته به این عمل را در بر دارد. این شی متدهایی برای برگرداندن اطلاعاتی درباره داده‌ها پیاده‌سازی می‌کند. متد GetData داده‌ها را واکنشی می‌کند و GetDataPresent نوع داده را بررسی می‌کند.

تأثیر انداختن کنترل هدف را مقداردهی کرده یا به دست می‌آورد.	Effect
حالت کلیدهای Alt، Ctrl، Shift و دکمه‌های ماوس را با یک عدد صحیح بر می‌گرداند.	KeyState
۱- دکمه سمت چپ ماوس ۲- دکمه سمت راست ماوس	
۴- کلید Shift ۸- کلید Ctrl	
۱۶- دکمه وسط ماوس ۳۲- کلید Alt	
مختصات اشاره‌گر ماوس با x و y مشخص می‌شوند.	X, Y

اعضای Data، Effect و KeyState بصورت زیر استفاده می‌شوند:

- `Data.GetDataPresent` بوسیله اداره کننده رویداد `DragEnter` جهت اطمینان از قابل پردازش بودن نوع داده توسط کنترل هدف استفاده می‌شود.
 - اداره کننده رویداد `DragDrop` برای دسترسی به داده کشیده شده روی آن، از متد `Data.GetData` استفاده می‌کند. پارامتر این متد معمولاً یک فیلد ایستا از کلاس `DataFormats` است که فرمت داده بازگشتی را مشخص می‌کند.
 - اداره کننده رویداد
 - `Effect` توسط اداره کننده رویداد `DragEnter` مقداردهی می‌شود، تا کنترل مبدأ را از نحوه پردازش داده‌های آن آگاه سازد. مقدار `DragDropEffects.None` از رها شدن رویداد `DragDrop` جلوگیری می‌کند.
- مثال ۸-۱۵ کد دو اداره کننده رویداد مورد نظر را نشان می‌دهد.

مثال ۸-۱۵

```
enum KeyPushed
{
    // Corresponds to DragEventArgs.KeyState values
    LeftMouse = ۱,
    RightMouse = ۲,
    ShiftKey = ۴,
    CtrlKey = ۸,
    MiddleMouse = ۱۶,
    AltKey = ۳۲,
}

private void TeamA_DragEnter(object sender, DragEventArgs e)
{
    KeyPushed kp = (KeyPushed) e.KeyState;
    // Make sure data type is string
    if (e.Data.GetDataPresent(typeof(string)))
    {
        // Only accept drag with left mouse key
        if ((kp & KeyPushed.LeftMouse) == KeyPushed.LeftMouse)
        {
            if ((kp & KeyPushed.CtrlKey) == KeyPushed.CtrlKey)
            {
                e.Effect = DragDropEffects.Copy; // Copy
            }
        }
    }
}
```

```

else
{
    e.Effect = DragDropEffects.Move; // Move
}
else // Is not left mouse key
{
    e.Effect = DragDropEffects.None;
}
}
else // Is not a string
{
    e.Effect = DragDropEffects.None;
}
}
// Handle DragDrop event
private void TeamA_Drop(object sender, DragEventArgs e)
{
    // Add dropped data to TextBox
    lstTeamA.Items.Add(
        (string) e.Data.GetData(DataFormats.Text));
}

```

بوسیله صفت `FlagsAttribute` یک `enum` ایجاد می‌شود، تا بررسی مقادیر `KeyState` را راحت‌تر و خواناتر سازد. عمل `and` منطقی `KeyState` با مقدار `(A CtrlKey)`، یک مقدار مساوی مقدار `CtrlKey` مربوط به کلید `Ctrl` بر می‌گرداند. می‌توان در یک برنامه یک کنترل را به عنوان مبدأ و هدف کشیدن و انداختن بکار گرفت. می‌توانید به هر دو لیست مثال قبلی هر دو نقش مبدأ و هدف بدهید تا انعطاف بالاتری داشته باشد. این عمل به شما اجازه می‌دهد یک بازیکن را از لیست `lstTeamA` به `lstPlayers` `ListBox` برگردانید. بدین منظور لازم است اداره کننده‌های مناسب رویداد اضافه شوند. توجه: کشیدن و رها کردن فقط برای متن نیست. کلاس `DataFormats` فرمت‌های از قبل تعریف شده‌ای دارد که می‌تواند به عنوان فیلدهای `static` استفاده شوند. اینها شامل موارد `Wave Audio`، `PenData`، `Bitmap` و مقادیر متعدد دیگر هستند.

۱۵-۹- کاربرد منابع

شکل ۱۵-۷ این فصل، کاربرد کنترل‌های `PictureBox` را برای بزرگ کردن و نمایش یک تصویر کوچک انتخاب شده نشان داد. هر تصویر کوچک از یک فایل محلی به برنامه کاربردی بارگذاری می‌شود.

```

tn1 = new PictureBox();
tn1.Image = Image.FromFile("c:\\schiele.jpg");

```

در صورتی که فایل `schiele.jpg` در مسیر فهرست ریشه کامپیوتر کاربر موجود باشد، این کد خوب کار می‌کند. با این وجود، تکیه بر مسیر فهرست برای یافتن این فایل دو عیب واضح دارد: فایل می‌تواند توسط کاربر حذف یا تغییر نام داده شود و چون منبع خارجی است، باید در حین نصب مجزا از کد اداره شود. این مشکل را می‌توان از طریق تعبیه کردن تصویر در یک اسمبلی حل کرد.

یک برنامه کاربردی `GUI` را در نظر بگیرید که در چندین کشور با زبان‌های مختلف استفاده می‌شود. تلاش ما برای وفق دادن صفحات نمایش با هر کشوری است. حداقل چیزی که نیاز است، نمایش یک متن با زبان اصلی و تغییر تصاویر و موقعیت کنترل‌های روی فرم است. راه حل مناسب، منطق برنامه را از واسط کاربر جدا می‌کند، در این راه حل به ازای هر کشور منابع قابل تعویض در نظر گرفته می‌شود که بر اساس تنظیمات فرهنگ کامپیوتر بارگذاری می‌شوند.

ایده مشترک این دو مثال، نیاز برای مقید کردن یک منبع خارجی به یک برنامه کاربردی است. `NET` چند فایل منبع خاص را فراهم می‌کند، که می‌توانند برای نگه داشتن هر داده‌ی غیراجرایی همچون تصویر، رشته‌ها و داده‌های دائمی استفاده شوند.

این فایل‌های منبع می‌توانند در یک اسمبلی قرار گیرند یا در داخل اسمبلی‌های پیرو^۱ کامپایل شوند تا بتوانند بطور دلخواه بوسیله یک اسمبلی اصلی برنامه کاربری دستیابی شوند.

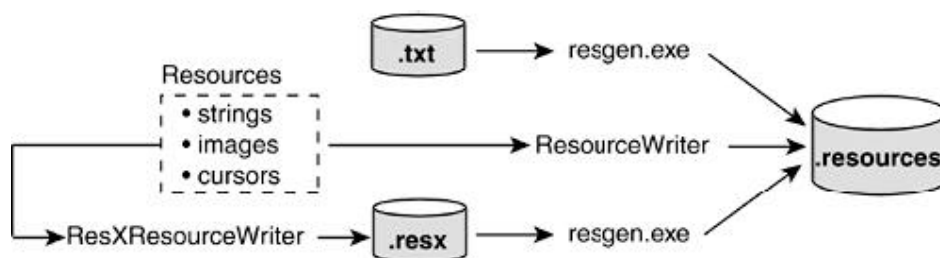
حال اصول کار با فایل‌های منبع و نحوه تعبیه آنها در اسمبلی را بررسی می‌کنیم و به نقش اسمبلی‌های پیرو در برنامه‌های کاربردی محلی نگاهی می‌کنیم.

۱۵-۹-۱- کار با فایل‌های منبع

فایل‌های منبع سه فرمت دارند: فایل‌های *.txt با فرمت مقدار / نام و فایل‌های *.resx با فرمت XML و فایل‌های *.resources با فرمت دودویی. چراسه تا ؟ فرمت متنی یک روش ساده برای اضافه کردن منابع رشته‌ای فراهم می‌سازند. نسخه XML، رشته‌ها و اشیاء دیگر همچون تصاویر را پشتیبانی می‌کند و نسخه دودویی، هم ارز دودویی فایل XML است. آنها تنها فرمتی هستند که می‌توانند در یک اسمبلی تعبیه شود. فرمت‌های دیگر قبل از اتصال به یک اسمبلی باید به فایل resources تبدیل شوند. شکل ۱۵-۲۰ روش‌های ایجاد یک فایل resources را شرح می‌دهد.

فضای نامی System.Resources انواع داده‌ای مورد نیاز برای دستکاری فایل‌های منبع را در بر دارد. آن کلاس‌هایی برای خواندن و نوشتن فرمت‌های فایل منبع، بارگذاری منابع از یک اسمبلی به یک برنامه را شامل است.

شکل ۱۵-۲۰



ایجاد رشته‌های منبع از روی یک فایل متنی

فایل‌های متنی که مقدارهای رشته‌ای دارند، در صورتی که یک برنامه لازم دارد یک واسط سفارشی در محیط اجرایی خود فراهم سازد، مفید هستند. یک فایل منبع نیاز برای کدنویسی نسخه‌های چندگانه از یک برنامه کاربردی را حذف می‌کند. به جای آن، توسعه‌دهنده یک برنامه کاربردی و چندین فایل منبع ایجاد می‌کند که برچسب‌های واسط، متن، پیام‌ها و عنوان‌ها را در بر دارند. برای مثال، یک نسخه English یک برنامه فایل منبع English تعبیه شده در اسمبلی را دارد. یک نسخه German فایل منبع German را تعبیه خواهد کرد. ایجاد رشته‌های منبع و دسترسی به آنها در یک برنامه کاربردی چهار گام نیاز دارد.

۱- یک فایل متنی بوسیله رشته‌های مقدار / نام ایجاد کنید، که در برنامه کاربردی استفاده خواهند شد. فایل فرمت زیر را دارد.

```
;German version (this is a comment)
```

^۱ Sattelite

```
Language=German
Select=Wählen Sie aus
Page=Seite
Previous=Vorherig
Next=Nächst
```

۲- با استفاده از نرم افزار سودمند Resource File Generator فایل متنی را به فایل Resources تبدیل کنید.

```
resgen.exe:
> resgen german.txt german.resources
```

توجه کنید که فایل متنی با ویرایشگر متن با کد گذاری UTF-۸ ذخیره شود، همان چیزی که resgen انتظار دارد.

۳- کلاس System.Resources.ResourceManager را برای خواندن رشته‌ها از فایل منبع بکار برید. همانطور که نشان داده شده، کلاس ResourceManager دو آرگومان می‌پذیرد: نام فایل منبع و اسمبلی که آن را در بردارد. کلاس Assembly بخشی از فضای نامی System.Reflection است و در این مثال برای برگرداندن اسمبلی جاری استفاده می‌شود. بعد از ایجاد مدیر منبع، متد GetString آن بوسیله برنامه کاربردی جهت بازیابی رشته‌ها بر اساس نام رشته‌ها از فایل منبع استفاده می‌شود.

```
// new ResourceManager(resource file, assembly)
ResourceManager rm = new
    ResourceManager("german", Assembly.GetExecutingAssembly());
nxtButton.Text= rm.GetString("Next");
```

۴- برای اینکه کد قبلی اجرا شود، فایل منبع باید بخشی از برنامه کاربردی باشد. آن در زمان کامپایل به اسمبلی تعبیه می‌گردد.

```
csc /t:exe /resource:german.resources myApp.cs
```

کاربرد کلاس ResourceWriter برای ایجاد یک فایل Resources.

راه حل قبلی برای اضافه کردن رشته‌ها به یک فایل منبع خوب کار می‌کند. با این وجود، یک فایل منبع می‌تواند اشیاء دیگری همچون تصاویر و اشکال مکان نما را در بر دارد.

برای قرار دادن اینها در یک فایل Resources، کلاس System.Resources.ResourceWriter را پیشنهاد می‌کند. کد زیر در یک فایل سودمند یا کمکی جا می‌گیرد. نحوه ایجاد یک شی ResourceWriter را نشان می‌دهد و متد AddResource را برای ذخیره یک رشته و تصویر در یک فایل منبع بکار می‌برد.

```
IResourceWriter writer = new ResourceWriter("myResources.resources");
    // .Resources output file
Image img = Image.FromFile(@"c:\schiele1.jpg");
rw.AddResource("Page","Seite"); // Add string
rw.AddResource("artistwife",img); // Add image
rw.Close(); // Flush resources to the file
```

کاربرد کلاس ResourceManager برای دستیابی به منابع

همانطور که بوسیله منابع رشته‌ای انجام دادید، ما کلاس ResourceManager را برای دستیابی به منابع بوسیله برنامه کاربردی بکار می‌بریم. برای شرح مطلب، به کد نمایش داده شده در ابتدای این بخش توجه کنید.

```
tn1.Image = Image.FromFile("C:\\schiele1.jpg");
```

ResourceManager به ما اجازه می‌دهد، ارجاع یک منبع خارجی را جایگزین کنیم. در این حالت، تصویر بخشی از اسمبلی خواهد بود. متد GetString مثال اخیر بوسیله متد GetObject جایگزین می‌شود.

```
ResourceManager rm = new ResourceManager("myresources",
    Assembly.GetExecutingAssembly());
// Extract image from resources in assembly
tn1.Image = (Bitmap) rm.GetObject("artistwife");
```

کاربرد کلاس ResXResourceWriter برای ایجاد یک فایل .resx

کلاس ResXResourceWriter شبیه کلاس ResourceWriter است، با استثناء اینکه منابع به یک فایل .resx اضافه می‌شوند که منابع را در یک فرمت XML میانی نمایش می‌دهد. این فرمت در زمان ایجاد برنامه‌های سودمند برای خواندن، مدیریت و ویرایش منابع مفید است. ویرایش منابع در فایل‌های منبع Resource بسیار مشکل است.

```
ResXResourceWriter rwx = new ResXResourceWriter(@"c:\myresources.resx");
Image img = Image.FromFile(@"c:\schiele1.jpg");
rwx.AddResource("artistwife",img); // Add image
rwx.Generate(); // Flush all added resources to the file
```

فایل منتج شده، اطلاعات سرآیند XML را به همراه علامت‌های مقدار / نام برای هر ورودی منبع در بر دارد. داده واقعی (تصویر) مابین علامت‌های مقدار ذخیره می‌شود. این یک بخش از فایل myresources.resx است که در ویرایشگر متنی دیده می‌شود.

```
<data name="face" type="System.Drawing.Bitmap, System.Drawing,
Version=۱,۰,۳۳۰۰,۰,Culture=neutral,
PublicKeyToken=b۰۳f۵f۷f۱۱d۵۰a۳a" mimetype="application/xmicrosoft.
net.object.bytearray.base۶۴">
<value> ---- Actual Image bytes go here ----
</value>
```

توجه کنید: اگرچه این مثال فقط یک تصویر را در فایل ذخیره می‌کند، یک فایل .resx می‌تواند چندین نوع منبع را در بر گیرد.

کاربرد کلاس ResXResourceReader برای خواندن یک فایل .resx

کلاس ResXResourceReader یک IDictionaryEnumerator (شمارنده) برای طی کردن همه علامت‌های یک فایل .resx بکار می‌برد. این قطعه کد محتوای یک فایل منبع را لیست می‌کند.

```
ResXResourceReader rrx = new ResXResourceReader("c:\myresources.resx");
// Enumerate the collection of tags
foreach (DictionaryEntry de in rrx)
{
    MessageBox.Show("Name: "+de.Key.ToString()+"\nValue: " +
                    de.Value.ToString());

    // Output --> Name: artistwife
    // --> Value: System.Drawing.Bitmap
}
rrx.Close();
```

تبدیل یک فایل .resx به یک فایل .resources

با استفاده از برنامه resgen.exe یک فایل .resx به فایل .resources تبدیل می‌شود.

```
resgen myresources.resx myresources.resources
```

اگر پارامتر دوم نباشد، فایل خروجی همان نام پارامتر اول است. همچنین این برنامه سودمند می‌تواند یک فایل .resources به یک فایل .resx تبدیل کند. گرامر همانند مثال قبلی است.

۱۵-۹-۲ VS.NET و منابع

VS.NET بطور اتوماتیک برای هر فرم یک فایل .resx ایجاد می‌کند و آنها را همانند منابع اضافه شده به پروژه بروز می‌کند. با انتخاب گزینه Show All File در Solution Explorer می‌توانید فایل‌های منبع را ببینید.

در زمان اجرای فرمان ساخت پروژه، فایل‌های resources از فایل‌های resx ایجاد می‌شوند. در خود کد، یک شی ResourceManager برای فراهم کردن دسترسی به منابع در زمان اجرا ایجاد می‌شود.

```
ResourceManager resources = new ResourceManager(typeof(Form1));
```

کاربرد فایل‌های منبع برای ایجاد فرم‌های محلی

در زبان مادری .NET، یک برنامه کاربردی محلی برنامه‌ای است که چندین زبان را پشتیبانی می‌کند. معمولاً این بدین معنی است که واسطه‌های کاربری مختلف برای نمایش متون و عکس‌های سفارشی شده با فرهنگ‌ها یا کشورهای فردی فراهم می‌سازد. فایل‌های منبع .NET برای پشتیبانی از این برنامه‌ها طراحی می‌شوند.

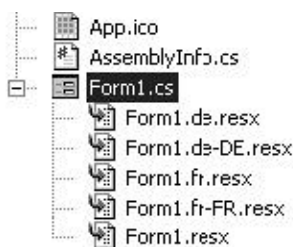
خلاصه اینکه، فایل‌های منبع می‌توانند برای هر فرهنگ پشتیبانی شده تنظیم شوند. برای مثال، ممکن است متون و برچسب کنترل‌های واسطه German در یک فایل قرار گیرند و متون و برچسب کنترل‌های واسطه French در فایل دیگری قرار گیرد. در زمان اجرای برنامه کاربردی، آن تنظیمات فرهنگ جاری را بدست می‌آورد و منابع مناسبی را بکار می‌برد. این عمل با تشخیص فایل‌های منبع به کلاس CultureInfo انجام می‌شود. فایل‌های منبع بصورت اسمبلی‌های پیرو بسته‌بندی می‌شوند و در فایل‌های DLL ذخیره می‌شوند.

محلی کردن منابع با استفاده از VS .NET

برای محلی کردن یک فرم، باید خصوصیت Localizable آن را true قرار دهید. این عمل هر کنترل روی فرم را با یک منبع دگرگون می‌کند، که خصوصیات آن کنترل در فایل resx فرم ذخیره شده باشد. این عمل باعث می‌شود، برای هر فرهنگی که یک فرم پشتیبانی می‌کند فایل‌های resx مجزایی ایجاد شود.

همانطور که می‌دانید، هر فرهنگی با دو کاراکتر از زبان و دو کاراکتر اختیاری از کشور مشخص می‌شود (en-us). اصطلاحات فرهنگ بی‌اثر و فرهنگ خاص برای شرح دادن یک فرهنگ هستند. یک فرهنگ خاص، زبان و کشور خاصی دارد. یک فرهنگ بی‌اثر فقط زبان دارد. در مستندات MSDN در کلاس CultureInfo می‌توانید لیست کاملی از اسامی فرهنگ‌ها را بیابید.

شکل ۱۵-۲۱

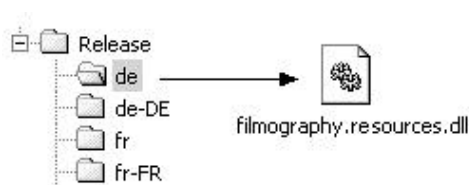


برای اختصاص دادن فرهنگ‌های دیگر به فرم، در پنجره properties مقدار خصوصیت Language فرم را به زبان محلی دیگر تنظیم کنید. این عمل منجر می‌شود یک فایل resx فقط برای آن فرهنگ ذخیره شود و بقیه را تحت تأثیر قرار نمی‌دهد.

فایل‌های منبع در پوشه‌هایی ذخیره می‌شوند. همانطور که در شکل ۱۵-۲۱ می‌بینید، در زمان ساخت پروژه، یک اسمبلی پیرو ایجاد می‌شود و منابع هر فرهنگی را در بر می‌گیرد (همانطور که در شکل ۱۵-۲۲ می‌بینند). این فایل DLL با آن پوشه

هم نام است

شکل ۱۵-۲۲



تعیین منابع محلی در زمان اجرا

بطور پیش‌فرض یک ریسمان برنامه کاربردی خصوصیت `CurrentThread.CurrentUICulture` دارد که تنظیمات فرهنگ ماشین محلی را دارد. نمونه‌های `ResourceManager` این مقدار را برای تعیین منابعی که باید بارگذاری شوند، بکار می‌برند. آنها اسمبلی پیرو را برای تنظیمات این فرهنگ جستجو می‌کنند (به همین دلیل نام‌گذاری و موقعیت پوشه‌ها و فایل‌های منبع مهم هستند). اگر منابع فرهنگ خاص پیدا نشود، منابع اسمبلی اصلی استفاده می‌شوند.

توجه: ساده‌ترین روش تست یک برنامه کاربردی با فرهنگ‌های دیگر، تغییر مقدار خصوصیت `CurrentUICulture` به فرهنگ دلخواه خود است. برای مثال، دستور زیر قبل از `InitializeComponent()` فرهنگ خاص `German` را تعیین می‌کند.

```
System.Threading.Thread.CurrentThread.CurrentUICulture =
    new System.Globalization.CultureInfo("de-DE");
```

ایجاد یک اسمبلی پیرو بدون VS.NET

یکی از مزایای کاربرد اسمبلی‌های پیرو، امکان اضافه کردن آنها به یک برنامه کاربردی و تغییر دادن آنها بدون کامپایل مجدد برنامه کاربردی است. تنها نیاز این است که یک پوشه در مسیر مناسب قرار داده شود و اسمبلی پیرو و پوشه اسامی مناسب داشته باشند.

فرض کنید یک فایل `resx` دارید که بوسیله‌ی مترجم به `French Canadian` تبدیل شده است. می‌توانید بصورت دستی در سه گام یک اسمبلی پیرو ایجاد کرده و به برنامه کاربردی اضافه کنید:

۱- فایل `resx` را به فایل `resources` تبدیل کنید.

```
filmography.Form\fr-CA.resources
```

۲- با استفاده از برنامه `Soudمند (Al.exe)` (Assembly Linker) فایل `resources` را به یک اسمبلی پیرو تبدیل کنید.

```
Al.exe
/t:lib
/embed:filmography.Form\fr-CA.resources
/culture:fr-CA
/out:filmography.resources.dll
```

پوشه `fr-CA` را در زیر پوشه `Release` ایجاد کرده و فایل اسمبلی جدید را به آن کپی کنید.

با قرار دادن اسمبلی پیرو در یک پوشه مناسب، آن فوراً برای اجرا در دسترس است و لازم نیست برنامه کاربردی مجدداً کامپایل شود.

۱۵-۱۰- خلاصه

- بیش از ۵۰ کنترل GUI در NET FCL موجود هستند.

- این فصل تعدادی از کنترل‌های مهم را بررسی کرده است. همه آنها از کلاس `System.Windows.Forms.Control` مشتق شده‌اند که خصوصیات و متدهای کلاس `Control` را به ارث برده‌اند.
- اگرچه هر کنترلی عملکرد منحصر به فرد خود را دارد، طبقه بندی آنها بر اساس رفتار و مشخصات مشابه امکان پذیر است.
- انواع دکمه برای شروع یک عمل یا یک انتخاب بکار می‌روند و شامل `Button`، `CheckBox` و `RadioButton` هستند. اغلب بوسیله یک کنترل `Panel` یا `GroupBox` گروه بندی می‌شوند.
- `TextBox` می‌تواند برای نگهداری یک خط متن یا کل یک سند بکار رود. متدهای متعددی برای جستجوی متن کادر و تعیین متن انتخاب شده در دسترس است.
- `PictureBox` برای نگه داشتن تصاویر است و یک خصوصیت `SizeMode` برای تعیین محل و اندازه تصویر در کادر عکس وجود دارد.
- چندین کنترل برای نمایش لیستی از داده‌ها وجود دارند. `ListBox` و `ComboBox` داده‌ها را در یک فرمت متنی ساده نمایش می‌دهند. با این وجود، ممکن است داده اصلی یک شی باشد که چندین خصوصیت دارد.
- `TreeView` و `ListView` برای نمایش داده‌ها بوسیله یک رابطه سلسله مراتبی مفید هستند.
- `ListView` می‌تواند داده‌ها را با چندین نما همچون شبکه‌ای و آیکونی نمایش دهند.
- `TreeView` یک تشبیه درخت را نشان می‌دهد که داده‌ها بصورت گره‌های پدر و فرزند نمایش داده می‌شوند.
- بیشتر این کنترل‌ها عملیات کشیدن و انداختن را پشتیبانی می‌کنند که عمل کپی یا انتقال داده‌ها از یک کنترل به کنترل دیگر را ساده می‌سازد. کنترل مبداء با فراخوانی متد `DoDragDrop` عمل را شروع می‌کند و داده‌ها و تأثیرات ممکن را به کنترل هدف رد می‌کند.
- برای برنامه‌هایی که کنترل‌های غیراستاندارد نیاز دارند، `NET` ایجاد کنترل‌های سفارشی را مجاز داشته است. آنها می‌توانند از ابتدا ایجاد شوند یا از یک کنترل موجود مشتق شوند یا ترکیبی از کنترل‌ها را بصورت یک کنترل کاربری ایجاد کنند.

فصل شانزدهم

نمایش کادرهای محاوره‌ای

آنچه که در این فصل یاد خواهید گرفت:

- با روشهای مختلف ایجاد یک کادر پیغام، با آیکون‌ها و دکمه‌های گوناگون آشنا خواهید شد.
- با نحوه‌ی ایجاد یک کادر Open برای دسترسی به فایل‌ها آشنا خواهید شد.
- با چگونگی ایجاد یک کادر Save جهت استفاده در ذخیره اطلاعات برنامه آشنا خواهید شد.
- با استفاده از کادر Font به کاربر اجازه خواهید دهید، فونت مورد نظر خود را انتخاب کند.
- با کادر Color و موارد استفاده از آن در برنامه آشنا خواهید شد.
- با استفاده از کادر Print قابلیت‌های مربوط به امور چاپ را به برنامه اضافه خواهیم کرد.

VC++ 2005 دارای چندین کادر محاوره‌ای درونی است که می‌تواند به طراحی ظاهر برنامه کمک زیادی کند. این کادرها در حقیقت همان پنجره‌های عمومی هستند که در بیشتر برنامه‌های تحت ویندوز مشاهده کرده‌اید. به علاوه، این کادرها دارای خصوصیت‌ها و متدهای فراوانی هستند که به وسیله‌ی آنها می‌توانید این کادرها را با قسمت‌های مختلف برنامه‌ی خود هماهنگ کنید.

در این فصل، کادرهای محاوره‌ای را به تفصیل مورد بررسی قرار خواهیم داد و مشاهده خواهیم کرد چگونه به وسیله آنها می‌توانیم برنامه‌هایی که دارای ظاهری حرفه‌ای‌تر هستند، طراحی کنیم.

۱۶-۱- کادر محاوره ای MessageBox

همانطور که تاکنون متوجه شده‌اید، کادر MessageBox یکی از کادرهایی است که در اغلب برنامه‌ها مورد استفاده قرار می‌گیرد. از این کادر عموماً "برای نمایش یک پیغام به کاربر و دریافت جواب کاربر استفاده می‌شود. باوجود اینکه در برنامه‌های قبلی به صورت یکنواخت از این کادر استفاده می‌کردیم، اما این کادر می‌تواند براساس موقعیت برنامه، دارای ظاهری متفاوت باشد. برای مثال، می‌توانید علاوه بر نمایش متن در آن، آیکون خاصی را نیز برای آن مشخص کنیم و یا دکمه‌های دیگری به جز دکمه OK در آن قرار دهیم.

در استفاده‌ی روزمره از برنامه‌های کامپیوتری، کادرهای پیغام گوناگونی را مشاهده می‌کنید که دارای آیکون‌هایی مانند آیکون‌های شکل ۱۶-۱ هستند. در این بخش مشاهده خواهید کرد که چگونه می‌توان از این آیکون‌ها در کادرهای محاوره‌ای استفاده کرد.



شکل ۱-۱۶

هنگام ایجاد یک برنامه ویندوزی، در مواقعی نیاز دارید که موردی را به کاربر اطلاع دهید و یا به کاربر هشدار دهید که یک پیشامد غیرمنتظره رخ داده است. برای مثال، فرض کنید کاربر اطلاعاتی از برنامه را تغییر داده است و بدون ذخیره کردن تغییرات سعی در بستن برنامه دارد. در این حالت، می‌توانید کادر پیغامی حاوی آیکن هشدار (سومین آیکن از چپ) و یا آیکن اطلاعات (اولین آیکن از چپ) و یک پیغام مناسب را به کاربر نمایش دهید و بگویید که در صورت بسته شدن برنامه، تمام اطلاعات ذخیره نشده از بین می‌روند. همچنین می‌توانید دکمه‌های OK و Cancel را در کادر پیغام قرار دهید تا کاربر بتواند به بستن برنامه ادامه دهد و یا این عمل را لغو کند.

در مواردی مشابه مورد بالا، استفاده از کادر پیغام می‌تواند به تسریع طراحی برنامه کمک کند. زیرا به وسیله آن می‌توانید با نمایش یک پیغام مناسب شامل آیکن و دکمه‌های موردنظر، به کاربر اجازه دهید در مورد یک مسئله خاص تصمیم‌گیری کند. همچنین با استفاده از کادر پیغام در بخش مدیریت خطا، می‌توانید خطاهای اتفاق افتاده در برنامه را با آیکن و دکمه‌های مناسب به کاربر اطلاع دهید.

قبل از اینکه استفاده از کادرهای پیغام گوناگون را در کد بررسی کنیم، ابتدا بهتر است با کلاس MessageBox آشنا شویم. همانطور که می‌دانید این کلاس دارای متدی به نام Show است که برای نمایش یک کادر پیغام به کار می‌رود. عنوان کادر پیغام، متن نمایش داده شده در آن، آیکن‌ها و نیز دکمه‌های فرمان کادر پیغام، همه به وسیله پارامترهای این متد مشخص می‌شوند. در ابتدا ممکن است مقداری پیچیده به نظر برسد، اما مشاهده خواهید کرد که استفاده از آن بسیار ساده است.

آیکن‌های قابل استفاده در یک کادر پیغام را در شکل ۱-۱۶ مشاهده کردید. در جدول ۱-۱۶ چهار آیکن قابل استفاده در کادر پیغام آورده شده است. در حقیقت آیکن مورد استفاده در این قسمت از سیستم عامل دریافت می‌شود و فعلاً "چهار آیکن برای این مورد در نظر گرفته شده است که برای هماهنگی بعضی از آنها دارای چند نام هستند:

جدول ۱-۱۶

نام عضو	توضیح
Information و Asterisk	مشخص می‌کند که آیکن اطلاعات در کادر پیغام نمایش داده می‌شود.
Stop و Hand و Error	مشخص می‌کند که یک آیکن خطا در کادر پیغام نمایش داده شود.
Warning و Exclamation	مشخص می‌کند که یک آیکن هشدار در کادر پیغام نمایش داده شود.
Question	مشخص می‌کند که یک علامت سوال در کادر پیغام نمایش داده شود.
None	مشخص می‌کند که آیکونی در کادر پیغام نمایش داده نشود.

دکمه‌های موجود برای کادر پیغام:

در هر کادر پیغام می‌توانید یکی از چندین گروه دکمه‌ی موجود را نمایش دهید. در جدول ۲-۱۶ گزینه‌های قابل انتخاب برای این مورد شرح داده شده‌اند:

جدول ۲-۱۶

نام عضو	شرح
AbortRetryIgnore	مشخص می‌کند که کادر شامل دکمه‌های Abort، Retry و Ignore باشد.
Ok	مشخص می‌کند که کادر شامل دکمه Ok باشد.

مشخص می‌کند که کادر شامل دکمه‌های Ok و Cancel باشد.	OkCancel
مشخص می‌کند که کادر شامل دکمه‌های Retry و Cancel باشد.	RetryCancel
مشخص می‌کند که کادر شامل دکمه‌های Yes و No باشد.	YesNo
مشخص می‌کند که کادر شامل دکمه‌های Yes و No و Cancel باشد.	YesNoCancel

تنظیم دکمه‌ی پیش فرض:

هنگام تنظیم ویژگی‌های مختلف یک کادر پیغام برای نمایش، علاوه بر مشخص کردن دکمه‌های آن، می‌توانید مشخص کنید که کدام دکمه به عنوان پیش‌فرض در نظر گرفته شود. به عبارت دیگر، با استفاده از این ویژگی مشخص می‌کنید که در بین دکمه‌های موجود در کادر، کدام دکمه باید دارای کانون باشد. با تنظیم این مورد می‌توانید به کاربر اجازه دهید، که بعد از خواندن متن کادر پیغام، با فشار دادن کلید Enter و بدون حرکت ماوس، دکمه‌ی پیش‌فرض را انتخاب کند. برای تنظیم این مورد، باید از شمارنده `MessageBoxDefaultButton` استفاده کنید که شرح گزینه‌های آن در جدول ۱۶-۳ آمده است:

جدول ۱۶-۳

نام عضو	شرح
Button۱	دکمه اول کادر پیغام به عنوان دکمه پیش‌فرض در نظر گرفته می‌شود.
Button۲	دکمه دوم کادر پیغام به عنوان دکمه پیش‌فرض در نظر گرفته می‌شود.
Button۳	دکمه سوم کادر پیغام به عنوان دکمه پیش‌فرض در نظر گرفته می‌شود.

ترتیب این دکمه‌ها از سمت چپ در نظر گرفته می‌شود. برای مثال، اگر در کادر پیغام سه دکمه Yes و No و Cancel داشته باشید و دکمه سوم را به عنوان دکمه پیش‌فرض مشخص کنید، دکمه Cancel پیش‌فرض خواهد بود. همچنین اگر در کادر پیغام دو دکمه Yes و No داشته باشید و دکمه سوم را به عنوان پیش‌فرض مشخص کنید، دکمه Yes پیش‌فرض خواهد بود.

گزینه‌های مختلف کادر پیغام

هنگام کار با کادر پیغام علاوه بر گزینه‌های بالا، موارد دیگری نیز قابل تنظیم است که در شمارنده `MessageBoxOptions` قرار دارد، بعضی از موارد پرکاربرد که در این قسمت قابل تنظیم هستند، در جدول ۱۶-۴ توضیح داده شده‌اند:

نام عضو	شرح
RightAlign	مشخص می‌کند که متن داخل کادر پیغام باید از سمت راست نوشته شود، این حالت برعکس حالت پیش‌فرض است که متن از سمت چپ نوشته می‌شود.
RTLReading	کادر پیغام باید برای نمایش متن راست به چپ تنظیم شود. این حالت برای نمایش متن به زبان‌هایی مناسب است که از راست به چپ نوشته می‌شوند (مانند فارسی). برای مثال، در این حالت آیکون کادر پیغام در سمت راست متن قرار می‌گیرد.

حالت‌های مختلف استفاده از متد Show

همانطور که می‌دانید، برای نمایش یک کادر پیغام از متد `Show` کلاس `MessageBox` استفاده می‌کنیم. کدی که در زیر مشاهده می‌کنید، متد `Show` را به گونه‌ای فراخوانی می‌کند که یک کادر پیغام مشابه شکل ۱۶-۲ را نمایش دهد. در این کد

متنی که باید در کادر نمایش داده شود، به عنوان پارامتر اول به متد فرستاده می‌شود و به دنبال آن نیز متنی که باید در نوار عنوان کادر قرار بگیرد، وارد می‌شود. سپس دکمه‌هایی روی کادر و نیز آیکون آن مشخص شده است. در انتها نیز دکمه پیش‌فرض معین شده است.

```
MessageBox.Show("My Text", "My Caption", MessageBoxButtons.OKCancel,
    MessageBoxIcon.Information, MessageBoxDefaultButton.Button1);
```



شکل ۱۶-۲

حال که با آیکون‌ها و دکمه‌های قابل استفاده در کادر پیغام آشنا شدید، بهتر است به بررسی متد Show از کلاس MessageBox بپردازیم. این متد به چندین روش قابل استفاده است و برای فراخوانی آن، می‌توانید پارامترهای گوناگونی را به آن ارسال کنید. برای مثال، در برنامه‌هایی که از ابتدای کتاب تاکنون نوشته‌ایم، تنها متنی که باید نمایش داده می‌شد را به این متد می‌فرستادیم، اما در مثال قبل بیشتر جزئیات کادر پیغام را برای متد مشخص کردیم. پرکاربردترین نوع‌های فراخوانی این متد در لیست زیر آمده است:

```
MessageBox.Show(Text)
MessageBox.Show(Text, Caption)
MessageBox.Show(Text, Caption, Button)
MessageBox.Show(Text, Caption, Button, Icon)
MessageBox.Show(Text, Caption, Button, Icon, DefaultButton)
```

در این لیست، پارامتر Text که یک پارامتر اجباری است، مشخص‌کننده متنی است که باید توسط کادر نمایش داده شود و می‌تواند یک مقدار ثابت و یا یک متغیر رشته‌ای باشد. بقیه پارامترهای این تابع به صورت اختیاری هستند:

Caption: مشخص‌کننده متنی است که باید در نوار عنوان کادر نمایش داده شود. اگر این پارامتر به تابع فرستاده نشود، متنی در نوار عنوان نمایش داده نمی‌شود.

Button: مقداری از نوع شمارشی MessageBoxButtons است. این پارامتر نوع دکمه‌های روی کادر را نمایش می‌دهد. اگر این پارامتر حذف شود، فقط دکمه Ok در کادر نمایش داده می‌شود.

Icon: مقداری از نوع شمارشی MessageBoxIcon است و برای تعیین آیکون کادر استفاده می‌شود. اگر این پارامتر حذف شود، آیکونی در کادر پیغام نمایش داده نخواهد شد.

DefaultButton: مقداری از نوع شمارشی MessageBoxDefaultButton است و برای تعیین دکمه فرمان پیش‌فرض در فرم به کار می‌رود. اگر این مقدار در فراخوانی تابع تعیین نشود، دکمه اول به عنوان دکمه فرمان پیش‌فرض در نظر گرفته می‌شود.

تمام حالت‌های متد Show که در بالا ذکر شدند، مقداری از نوع شمارشی DialogResult برمی‌گردانند. این مقدار مشخص می‌کند کدامیک از دکمه‌های کادر پیغام توسط کاربر انتخاب شده است. در جدول ۱۶-۵ تمام گزینه‌های نوع شمارشی DialogResult مورد بررسی قرار گرفته است.

جدول ۱۶-۵

Abort	مقدار بازگشتی Abort است و مشخص می‌کند که کاربر بر روی دکمه Abort کلیک کرده است.
Cancel	مقدار بازگشتی Cancel است و مشخص می‌کند که کاربر بر روی دکمه Cancel کلیک کرده است.
Ignore	مقدار بازگشتی Ignore است و مشخص می‌کند که کاربر بر روی دکمه Ignore کلیک کرده است.
No	مقدار بازگشتی No است و مشخص می‌کند که کاربر بر روی دکمه No کلیک کرده است.
None	مقدار بازگشتی None است. به عبارت دیگر، هنوز گزینه‌ای از کادر پیغام توسط کاربر انتخاب نشده است.
OK	مقدار بازگشتی Ok است و مشخص می‌کند که کاربر بر روی دکمه Ok کلیک کرده است.
Retry	مقدار بازگشتی Retry است و مشخص می‌کند که کاربر بر روی دکمه Retry کلیک کرده است.
Yes	مقدار بازگشتی Yes است و مشخص می‌کند که کاربر بر روی دکمه Yes کلیک کرده است.

نمونه‌هایی از کادر پیغام

در مواردی که فقط یک دکمه در کادر پیغام به کار می‌رود. نیازی به بررسی نتیجه کادر پیغام نداریم، اما اگر در کادر پیغام بیش از یک دکمه استفاده کنیم، بعد از نمایش باید نتیجه را نیز بررسی کنیم. در مثال زیر مشاهده خواهیم کرد که چگونه می‌توان یک کادر پیغام با بیش از یک دکمه نمایش داده و سپس مشخص کرد که کاربر کدام دکمه را انتخاب کرده است.

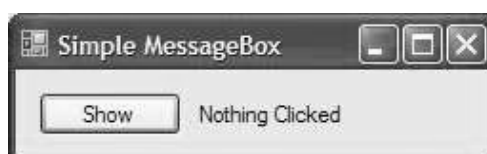
مثال ۱۶-۱- ایجاد کادر پیغام با دو دکمه

(۱) VS ۲۰۰۵ را اجرا کرده و پروژه جدید از نوع Windows Application ایجاد کرده و با SimpleMessageBox نامگذاری کنید.

(۲) در قسمت طراحی بر روی فرم برنامه کلیک کرده و سپس خصوصیت Text آن را به SimpleMessageBox تغییر دهید.

(۳) با استفاده از جعبه ابزار، یک کنترل Button به فرم اضافه کرده و خصوصیت Name آن را برابر با btnShow و خصوصیت Text آن را برابر با Show قرار دهید.

(۴) سپس یک کنترل Label در فرم قرار دهید. این کنترل برای نمایش گزینه‌ای به کار می‌رود که کاربر از کادر پیغام انتخاب کرده است. خصوصیت Name این گزینه را به lblResult و خصوصیت Text آن را به NothingClicked تغییر دهید. سپس اندازه فرم را به گونه‌ای تنظیم کنید که فرم شما مشابه شکل ۱۶-۳ شود.



شکل ۱۶-۳

۵) بر روی دکمه‌ی Show دو بار کلیک کنید تا متد مربوط به رویداد Click آن ایجاد شود. سپس کد مشخص شده در زیر را در آن وارد کنید.

```
private void btnShow_Click(object sender, EventArgs e)
{
    if ( MessageBox.Show("Your Internet Connection will be closed now!", "Dial-Up
        Networking Notification", MessageBoxButtons.OKCancel, MessageBoxIcon.None,
        MessageBoxDefaultButton.Button1) == DialogResult.OK)
    {
        lblResult.Text = "OK Clicked!";
        // Call some method here...
    }
    else
    {
        lblResult.Text = "Cancel Clicked!";
        // Call some method here...
    }
}
```

۶) برنامه را اجرا کرده و بر روی دکمه‌ی Show کلیک کنید. کادر پیغامی مشابه شکل ۴-۱۶ مشاهده خواهید کرد.



شکل ۴-۱۶

۷) بر روی دکمه‌ی Ok و یا دکمه Cancel کلیک کنید. مشاهده می‌کنید که نتیجه‌ی انتخاب شما در برجسب نمایش داده می‌شود.

همانطور که در فصل‌های قبل گفتیم، برای استفاده از کارکترهای کنترلی در یک رشته از علامت \ استفاده می‌کنند. یکی از کاراکترهای کنترلی n است که باعث می‌شود ادامه متن در یک خط جدید نمایش داده شود.

نکته : همواره دقت کنید که از کادر پیغام بیش از اندازه استفاده نکنید و سعی کنید برای استفاده از آن دلیل مناسبی داشته باشید، زیرا استفاده بیش از اندازه از آن باعث ناراحتی کاربر می‌شود. در مواقعی از کادر پیغام استفاده کنید که بخواهید کاربر را از رخ دادن خطایی در برنامه آگاه کنید و یا به کاربر در مورد یک مسئله مهم که ممکن است باعث ایجاد خطا و یا از دست رفتن اطلاعات شود هشدار دهید. یک مثال برای زمانی است که بدون عمل ذخیره، تغییرات انجام شده توسط کاربر از بین برود.

۱۶-۲- کنترل OpenFileDialog

تقریباً در نوشتن بیشتر برنامه‌های ویندوزی مواقعی وجود دارد که بخواهید داده‌هایی را در فایل نوشته و یا از آن بخوانید، پس به کنترلی نیاز دارید تا به وسیله‌ی آن بتوانید فایلی را باز کنید و یا داده‌هایی را در یک فایل ذخیره کنید.

در چارچوب NET، دو کنترل برای این موارد در نظر گرفته شده است: OpenFileDialog و SaveFileDialog. در این بخش به بررسی کنترل OpenFileDialog می‌پردازیم و در بخش بعد نیز کنترل SaveFileDialog را بررسی خواهیم کرد.

هنگامی که با برنامه‌های ویندوز مانند Word و Paint کار می‌کنید، معمولاً برای باز کردن فایل و یا ذخیره آن و یا... با محیطی یکسان روبرو می‌شوید. این نوع کادرها به صورت مجموعه‌ای استاندارد در ویندوز وجود دارند و برنامه‌نویسان می‌توانند از آنها در برنامه‌های خود استفاده کنند.

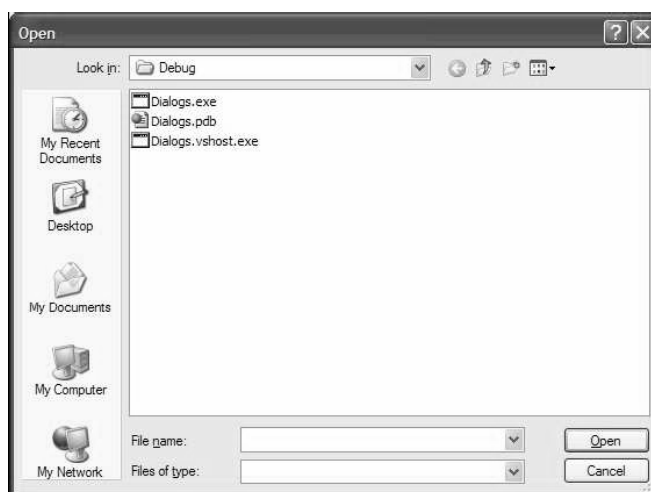
در NET، برای دسترسی به پنجره Open از این مجموعه، باید از کلاس OpenFileDialog استفاده کرد. برای استفاده از این کلاس در NET، مانند هر کلاس دیگر باید یک متغیر از آن ایجاد و سپس خصوصیت‌های آن را به وسیله کد تنظیم کرد و یا هنگام طراحی فرم می‌توان با استفاده از جعبه ابزار این کنترل را در برنامه قرار داده و از آن استفاده کرد. در هر دو حالت شی ایجاد شده دارای متدها، رویدادها و خصوصیت‌های یکسان خواهد بود.

برای دسترسی به این کنترل در جعبه ابزار، باید به بخش Dialogs آن مراجعه کنید و با استفاده از ماوس این کنترل را بر روی فرم قرار دهید. سپس تنها کاری که باید انجام دهید، این است که خصوصیت‌های مورد نظر آن را به وسیله پنجره Properties تنظیم کرده و سپس متد مربوط به نمایش آن را فراخوانی کنید.

برای استفاده از کنترل OpenFileDialog به صورت کلاس، ابتدا باید شیئی از نوع این کلاس ایجاد کنید. سپس در مواقعی که به این کنترل نیاز داشتید، به این شی مقدار بدهید و از آن استفاده کنید. پس از پایان استفاده نیز می‌توانید آن را نابود کنید تا منابع اشغال شده به وسیله آن آزاد شوند.

در این فصل با OpenFileDialog به صورت یک کنترل برخورد می‌کنیم. اما هنگامی که کاملاً مفهوم آن را درک کردید و توانستید به راحتی در برنامه از آن استفاده کنید، می‌توانید از این کنترل به صورت یک کلاس نیز استفاده کنید.

برای نمایش پنجره‌ی Open کافی است متد ShowDialog کلاس OpenFileDialog را فراخوانی کنید، به این ترتیب پنجره‌ای مشابه شکل ۱۶-۶ نمایش داده خواهد شد.



شکل ۱۶-۶

خصوصیت‌های کنترل OpenFileDialog

اگرچه کادر محاوره‌ای نمایش داده شده در شکل ۱۶-۶ یک صفحه Open استاندارد در ویندوز است. معمولاً هنگامی که این کادر را در برنامه‌ای مشاهده می‌کنید، فقط فایل‌های خاصی در آن نمایش داده شده است (برای مثال، این کادر در برنامه‌ی NotePad فقط فایل‌های متنی را نمایش می‌دهد)، اما در این کادر هیچ محدودیتی در نوع فایل‌های قابل نمایش دیده

نمی‌شود. در این پنجره تمام فایل‌های موجود در فهرست جاری را مشاهده می‌کنید و نمی‌توانید مشخص کنید که چه نوع فایل‌هایی را نمایش دهد. برای فیلتر کردن نوع فایل‌های نمایش داده شده، در این کادر باید از خصوصیت‌های این کنترل استفاده کرده و آنها را به نحوی تنظیم کنید که کادر فقط فایل‌های مورد نظر شما را نمایش دهد.

البته این یکی از مواردی است که با استفاده از خصوصیت‌های این کنترل قابل تنظیم است. در جدول ۱۶-۶ لیستی از خصوصیت‌های پرکاربرد این کنترل را بررسی می‌کنیم:

جدول ۱۶-۶

شرح	خصوصیت
این خصوصیت مشخص می‌کند که اگر کاربر پسوندی را برای فایل مشخص نکرد، برنامه به طور اتوماتیک پسوند را به آن اضافه کند یا نه؟ این مورد بیشتر در کادر <code>SaveFileDialog</code> که در بخش بعد توضیح داده خواهد شد، استفاده می‌شود.	<code>AddExtension</code>
مشخص می‌کند اگر کاربر آدرس فایلی را وارد کرد که وجود نداشت، برنامه پیغام خطایی را نمایش بدهد یا نه؟	<code>CheckFileExist</code>
مشخص می‌کند اگر کاربر آدرس مسیری را وارد کرد که وجود نداشت، برنامه پیغام خطایی را نمایش بدهد یا نه؟	<code>CheckPathExist</code>
پسوند پیش‌فرض را برای فایل انتخاب شده مشخص می‌کند.	<code>DefaultExt</code>
با میانبرها استفاده می‌شود و مشخص می‌کند که اگر کاربر یک میانبر را انتخاب کرد، مسیر فایل اصلی برگشت داده شود (<code>true</code>) و یا مسیر خود فایل میانبر به برنامه برگردد (<code>false</code>).	<code>DereferenceLinks</code>
مشخص کننده نام فایلی است که در این کادر انتخاب شده است.	<code>FileName</code>
مشخص کننده نام فایل‌هایی است که در این پنجره انتخاب شده است. این خصوصیت از نوع فقط-خواندنی است.	<code>FileNames</code>
این خصوصیت حاوی رشته‌ای است که برای فیلتر کردن فایل‌هایی که باید در پنجره <code>Open</code> نمایش داده شوند به کار می‌رود. به وسیله این رشته می‌توانید، چندین گروه فیلتر را برای این پنجره نمایش ایجاد کنید و کاربر بتواند یکی از آنها را انتخاب کند.	<code>Filter</code>
مشخص کننده آدرس مسیری است که باید در ابتدا، در پنجره <code>Open</code> نمایش داده شود.	<code>InitialDirectory</code>
مشخص می‌کند آیا کاربر می‌تواند چندین فایل را در این پنجره انتخاب کند و یا نه؟	<code>MultiSelect</code>
مشخص می‌کند آیا قسمت <code>ReadOnly</code> در پنجره <code>Open</code> انتخاب شده است و یا نه؟	<code>ReadOnlyChecked</code>
تعیین می‌کند آیا کادر <code>Open</code> باید آدرس مسیری که قبل از بسته شدن در آن قرار داشت را برگرداند یا نه؟	<code>RestoreDirectory</code>
مشخص می‌کند آیا دکمه <code>Help</code> نیز در پنجره <code>Open</code> نمایش داده شود یا نه؟	<code>ShowHelp</code>
مشخص می‌کند آیا امکان تعیین این که فایل به صورت فقط-	<code>ShowReadOnly</code>

خواندنی باز شود، برای کاربر وجود داشته باشد یا نه؟

مشخص کننده متنی است که در نوار عنوان پنجره Open نمایش داده می‌شود.	Title
مشخص می‌کند که آیا پنجره فقط باید نام فایل‌های معتبر ویندوز را قبول کند و یا هر نامی را می‌تواند دریافت کند؟	ValidateNames

متدهای OpenFileDialog

اگرچه متدهای زیادی در کنترل OpenFileDialog وجود دارند، اما در مثال‌های این بخش بیشتر بر روی متد ShowDialog تمرکز خواهیم کرد. در لیست زیر، نام و شرح استفاده‌ی بعضی از توابع پر کاربرد این کنترل آمده است:

- **Dispose**: حافظه اشغال شده توسط این کنترل را آزاد می‌کند.
- **OpenFile**: فایلی را که به وسیله کاربر در پنجره Open انتخاب شده است به صورت فقط -خواندنی باز می‌کند. نام فایل به وسیله خصوصیت FileName مشخص می‌شود.
- **Reset**: مقدار تمام خصوصیت‌های کنترل OpenFileDialog را به حالت اولیه بر می‌گرداند.
- **ShowDialog**: کادر محاوره‌ای پنجره Open را نمایش می‌دهد.

استفاده از تابع ShowDialog بسیار واضح است، زیرا این متد هیچ پارامتری را به عنوان ورودی دریافت نمی‌کند. بنابراین قبل از اینکه این تابع را فراخوانی کنید، باید تمام خصوصیت‌های موردنظر را در کنترل تنظیم کنید. بعد از اینکه پنجره Open نمایش داده شد، می‌توانید با بررسی مقدار خصوصیت‌های کنترل مشخص کنید که چه فایل و یا چه فایل‌هایی، در چه مسیرهایی انتخاب شده‌اند. یک نمونه از فراخوانی این متد در قطعه کد زیر نمایش داده شده است:

```
openFileDialog1.ShowDialog();
```

این متد مقداری از نوع شمارشی DialogResult به صورت OK و یا Cancel بر می‌گرداند. مقدار OK به معنی کلیک کردن کاربر بر روی دکمه‌ی Open و مقدار Cancel برابر با کلیک کردن روی Cancel است. توجه داشته باشید که این کنترل هیچ فایلی را برای برنامه باز نمی‌کند و یا محتویات آن را نمی‌خواند. این کنترل فقط رابطی است که به کاربر اجازه می‌دهد یک یا چند فایل را برای باز شدن به وسیله‌ی برنامه مشخص کند. بعد از این مرحله، شما باید در برنامه، نام و آدرس فایل‌های مشخص شده توسط کاربر را به وسیله خصوصیت‌های کنترل OpenFileDialog بدست آورده و سپس آنها را باز کنید.

استفاده از کنترل OpenFileDialog

حال که خصوصیت‌ها و متدهای مهم کنترل OpenFileDialog را بررسی کردیم، بهتر است از این کنترل در یک برنامه استفاد کنیم تا با نحوه استفاده از آن در برنامه آشنا شویم.

در مثال بعدی، با استفاده از کنترل OpenFileDialog برنامه‌ای خواهید نوشت که کاربر بتواند در آن یک فایل متنی را مشخص کند و برنامه محتویات آن فایل را در یک TextBox نمایش دهد.

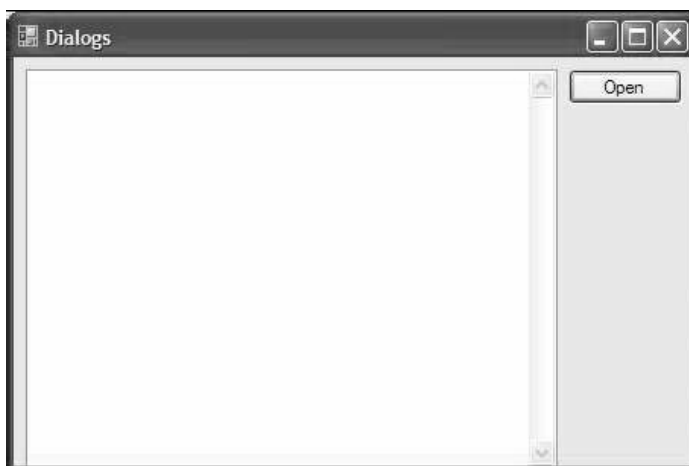
مثال ۱۶-۲- کار با کنترل OpenFileDialog

(۱) در محیط VS ۲۰۰۵ یک پروژه ویندوز جدید به نام Dialog ایجاد کنید.

(۲) سپس نام فرم را به Dialogs.cs تغییر دهید.

(۳) با استفاده از پنجره Properties خصوصیت‌های فرم را به صورت زیر تغییر دهید:

- خصوصیت Size آن را برابر با ۴۵۶-۳۰۴ قرار دهید.
 - خصوصیت StartPosition را برابر با CenterScreen قرار دهید.
 - خصوصیت Text را برابر با Dialogs قرار دهید.
- (۴) برای این که فایل مشخص شده توسط کاربر را در برنامه نمایش دهید، به یک کنترل TextBox نیاز دارید. همچنین باید یک کنترل Button نیز بر روی فرم قرار دهید تا کاربر به وسیله آن بتواند پنجره Open را نمایش دهد. بنابراین یک کنترل TextBox و یک کنترل Button به فرم خود اضافه کرده و خصوصیت‌های آنها را بر اساس لیست زیر تنظیم کنید:
- خصوصیت Name کنترل TextBox را برابر با txtFile، خصوصیت Anchor را برابر با Left, Bottom, Top, Right، خصوصیت Location را برابر با ۸-۸، خصوصیت MultiLine را برابر با true، خصوصیت ScrollBars را برابر با Vertical و خصوصیت Size را برابر با ۳۵۲-۲۶۴ قرار دهید.
 - خصوصیت Name دکمه را برابر با btnOpen، خصوصیت Text آن را برابر با Open، خصوصیت Anchor را برابر با Top-Right و خصوصیت Location را برابر با ۸-۳۶۷ قرار دهید.
- (۵) بعد از اینکه کنترل‌ها را در فرم قرار دادید و خصوصیت آنها را طبق لیست قبلی تنظیم کردید، فرم برنامه شما باید مشابه شکل ۱۶-۷ شده باشد.



شکل ۱۶-۷

- نکته: هنگامی که خصوصیت Anchor را برای کنترل‌های این فرم تنظیم کنیم، با تغییر اندازه فرم به وسیله کاربر اندازه کنترل‌ها نیز به صورت متناسب تغییر خواهد کرد.
- (۶) در نوار ابزار به قسمت Dialogs بروید، کنترل OpenFileDialog را انتخاب کرده و بر روی آن دوبار کلیک کنید. مشاهده می‌کنید که این کنترل به قسمت پایین محیط طراحی VS اضافه خواهد شد. حال می‌توانید این کنترل را در این قسمت انتخاب کرده و خصوصیت‌های مختلف آن را به وسیله پنجره Properties تنظیم کنید. با این وجود، فعلاً نام و خصوصیت‌های پیش‌فرض این کنترل را قبول کنید. در قسمت‌های بعدی با استفاده از کد، خصوصیت‌های مورد نظر را در این کنترل تغییر خواهیم داد.
- (۷) به قسمت ویرایشگر کد بروید و در ابتدای کلاس مربوط به فرم، یک متغیر رشته‌ای تعریف کنید تا نام فایل در آن ذخیره شود. در قسمت‌های بعدی برنامه، نام فایلی که به وسیله کادر Open برگردانده می‌شود را در این متغیر ذخیره خواهیم کرد:

```
public partial class Dialogs : Form
{
    // Declare variables
    private string strFileName;
```

۸) حال باید کد مربوط به باز کردن کادر Open را در رویداد کلیک کنترل btnOpen قرار دهیم. برای این کار به قسمت طراحی فرم بروید و بر روی این کنترل دو بار کلیک کنید تا متد مربوط به رویداد کلیک آن ایجاد شود. سپس کدهای مشخص شده در زیر را به آن اضافه کنید.

```
private void btnOpen_Click(object sender, EventArgs e)
{
    // Set the OpenFileDialog properties
    openFileDialog1.Filter = "Text files (*.txt) |*.txt|" + " All files (*.*) |*.*";
    openFileDialog1.FilterIndex = 1;
    openFileDialog1.Title = "Demo Open File Dialog";
    // Show the OpenFileDialog and if the user clicks the
    // Open button, load the file
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        // Save the file name
        strFileName = openFileDialog1.FileName;
        // Read the contents of the file
        txtFile.Text = System.IO.File.ReadAllText(strFileName);
    }
}
```

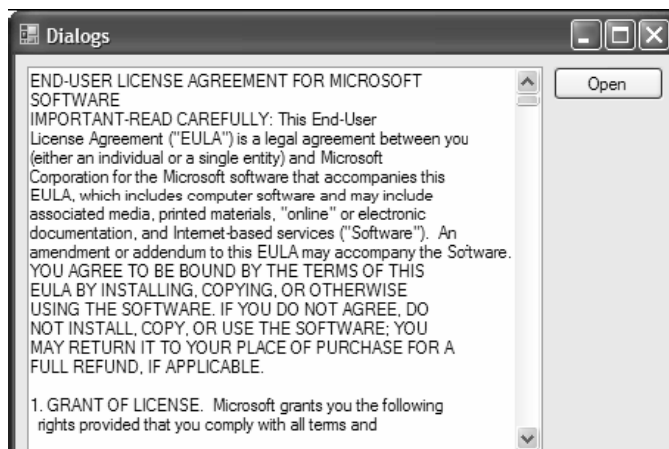
۹) حال برنامه را اجرا کرده و هنگامی که فرم اصلی برنامه نمایش داده شد، بر روی دکمه‌ی Open کلیک کنید تا کادر محاوره‌ای Open نمایش داده شود. توجه کنید که عنوان این کادر همانطور که در کد مشخص کرده بودید تغییر کرده است. در جعبه ترکیبی Files Of Type در پایین کادر کلیک کنید. مشاهده می‌کنید که دو نوع فیلتر برای نمایش فایل‌ها در نظر گرفته شده است.

۱۰) یکی از فیلترهای نمایش داده شده در این قسمت را انتخاب کنید. سپس یک فایل متنی را در دیسک مشخص کرده و بر روی دکمه Open کلیک کنید، مشاهده خواهید کرد که همانند شکل ۱۶-۸ محتویات فایل در فرم نمایش داده خواهد شد.

۱۱) برای امتحان، از برنامه خارج شوید و مجدداً آن را اجرا کنید. بر روی دکمه‌ی Open کلیک کنید. مشاهده خواهید کرد که کادر Open مسیری را نمایش می‌دهد که در اجرای قبلی برنامه فایل را از آن انتخاب کردید.

بررسی نکات مهم برنامه

اولین خصوصیتی که باید تنظیم شود، خصوصیت Filter است. این خصوصیت به شما اجازه می‌دهد فیلترهایی که در جعبه ترکیبی Files Of Type در کادر نمایش داده می‌شوند را مشخص کنید. هنگامی که بخواهید یک فیلتر برای پسوندی خاص ایجاد کنید، باید ابتدا توضیح آن فیلتر را وارد کرده، سپس یک خط عمود (|) قرار دهید و سپس پسوند فایل را وارد کنید. اگر بخواهید چندین فیلتر در این کادر وجود داشته باشند، باید هر یک از آنها به وسیله یک خط عمودی از یکدیگر جدا کنید.



شکل ۱۶-۸

```
// Set the OpenFileDialog properties
openFileDialog1.Filter = "Text files (*.txt) |*.txt| All files (*.*) |*.*";
```

دومین خصوصیتی که باید تنظیم شود، خصوصیت `FilterIndex` است که مشخص می‌کند کدام فیلتر باید به صورت پیش‌فرض در فرم در نظر گرفته شود. مقدار ۱ برای این خصوصیت مشخص می‌کند که فیلتر اول به عنوان فیلتر پیش‌فرض گرفته می‌شود.

```
openFileDialog1.FilterIndex = 1;
```

در انتها نیز با استفاده از خصوصیت `Title` عنوان پنجره `Open` را تغییر می‌دهیم:

```
openFileDialog1.Title = "Demo Open File Dialog";
```

برای نمایش کادر `Open` باید از تابع `Show` استفاده کنیم. همانطور که گفتیم این تابع مقداری از نوع `DialogResult` را بر می‌گرداند که می‌تواند مقدار `DialogResult.Ok` یا `DialogResult.Cancel` داشته باشد. اگر کاربر در پنجره `Open` روی دکمه `Open` کلیک کند، مقدار `DialogResult.Ok` توسط تابع برگردانده می‌شود. در صورتی که کاربر دکمه `Cancel` را انتخاب کند، مقدار بازگشتی برابر با `DialogResult.Cancel` خواهد بود.

```
// Show the OpenFileDialog and if the user clicks the
// Open button, load the file
if (openFileDialog1.ShowDialog() == DialogResult.OK)
```

برای آشنایی با دستورات کار با فایل‌ها به فصل فایل‌ها مراجعه کنید.

۱۶-۳- کنترل `SaveFileDialog`

حال که می‌توانید با استفاده از کنترل `OpenFileDialog` یک فایل را باز کرده و از اطلاعات آن در برنامه استفاده کنید، بهتر است به بررسی کنترل `SaveFileDialog` بپردازیم، تا مشاهده کنید که چگونه می‌توان به وسیله آن اطلاعاتی را در دیسک ذخیره کرد. همانند کنترل `OpenFileDialog`، این کنترل نیز می‌تواند هم به صورت یک کنترل و هم به صورت یک کلاس مورد استفاده قرار گیرد. در این قسمت از `SaveFileDialog` به عنوان یک کنترل استفاده می‌کنیم، اما بعد از اینکه با این کنترل بیشتر آشنا شدید، می‌توانید از آن به عنوان یک کلاس نیز استفاده کنید. بعد از اینکه فایلی را در برنامه باز کردید، ممکن است بخواهید تغییراتی در آن ایجاد کرده و نتیجه را در دیسک ذخیره کنید. در این شرایط است که کنترل `SaveFileDialog` می‌تواند موثر واقع شود. کنترل `SaveFileDialog` نیز کارکردی مشابه کنترل `OpenFileDialog` دارد، البته در جهت عکس. این کنترل به شما اجازه می‌دهد یک نام و آدرس را برای ذخیره‌ی یک فایل در دیسک از کاربر دریافت کنید. مجدداً باید ذکر کنم که همانند کنترل `OpenFileDialog` این کنترل نیز فایلی را در دیسک ذخیره نمی‌کند، بلکه فقط یک رابط استاندارد را برای برنامه به وجود می‌آورد تا کاربر به وسیله آن بتواند محلی را برای ذخیره اطلاعات مشخص کند.

خصوصیت‌های کنترل `SaveFileDialog`

در جدول ۱۶-۷ لیستی از خصوصیت‌های پرکاربرد کنترل `SaveFileDialog` به همراه کاربرد آنها آورده شده است. همانطور که مشاهده می‌کنید این کنترل (و یا کلاس، بسته به نوعی که از آن استفاده می‌کنید)، خصوصیت‌های زیادی دارد که می‌توان به وسیله آنها، رفتار کنترل را در برنامه تغییر داد.

جدول ۱۶-۷

شرح	خصوصیت
مشخص می‌کند اگر کاربر فایلی را مشخص کرد که وجود نداشت، برای ایجاد آن فایل از کاربر سوال شود یا نه؟	CreatePrompt
مشخص می‌کند اگر کاربر خواست فایل را بر روی فایل دیگری ذخیره کند، پیغام هشدار به کاربر نمایش داده شود یا نه؟	OverWritePrompt
تعیین می‌کند آیا کادر Save باید آدرس فهرستی را که قبل از بسته شدن در آن قرار داشت، برگرداند یا نه؟	ResotreDirectory

بقیه خصوصیات شبیه کنترل OpenFileDialog است.

متدهای کنترل SaveFileDialog

متدهای کنترل SaveFileDialog همانند متدهای OpenFileDialog هستند. برای مطالعه متدهای کنترل OpenFileDialog می‌توانید به بخش قبلی مراجعه کنید. در تمام مثال‌های بعدی نیز همانند کنترل OpenFileDialog از تابع ShowDialog برای نمایش کادر Save استفاده می‌کنیم.

استفاده از کنترل SaveFileDialog

برای بررسی نحوه کارکرد کنترل SaveFileDialog، از مثال قسمت قبلی استفاده می‌کنیم. در این قسمت می‌خواهیم برنامه را به صورتی تغییر دهیم که متن داخل TextBox را در فایلی ذخیره کند. در این قسمت، با استفاده از کنترل SaveFileDialog پنجره Save را به کاربر نمایش داده و به او اجازه می‌دهیم تا مکانی را برای ذخیره محتویات TextBox مشخص کند. سپس محتویات داخل آن را در فایلی در مسیر مشخص شده توسط کاربر ذخیره می‌کنیم.

مثال ۱۶-۳ کار با کنترل SaveFileDialog

(۱) برنامه‌ی مثال قبلی را مجدداً باز کنید.

(۲) در فرم اصلی برنامه کنترل Button دیگری اضافه کرده و خصوصیت‌های آن برابر با لیست زیر قرار دهید.

• خصوصیت Name را برابر با btnSave قرار دهید.

• خصوصیت Text را برابر با Save قرار دهید.

• خصوصیت Anchor را برابر با Top-Right قرار دهید.

• خصوصیت Location را برابر با ۳۶۷-۳۸ قرار دهید.

(۳) در جعبه ابزار به قسمت Dialogs بروید و بر روی کنترل SaveFileDialog دو بار کلیک کنید.

(۴) بر روی دکمه‌ی btnSave دو بار کلیک کنید تا متد مربوط به رویداد کلیک آن ایجاد شود. سپس کد زیر را در آن متد وارد کنید:

```
private void btnSave_Click(object sender, EventArgs e)
{
    // Set the save dialog properties
    saveFileDialog1.DefaultExt = "txt";
    saveFileDialog1.FileName = strFileName;
    saveFileDialog1.Filter = "Text files (*.txt)|*.txt|All files (*.*)|*.*";
    saveFileDialog1.FilterIndex = 1;
    saveFileDialog1.OverwritePrompt = true;
```

```

saveFileDialog1.Title = "Demo Save File Dialog";
// Show the Save file dialog and if the user clicks
the // Save button, save the file
if (saveFileDialog1.ShowDialog() == DialogResult.OK)
{
    // Save the file name
    strFileName = saveFileDialog1.FileName;
    // Write the contents of the text box in file
    System.IO.File.WriteAllText(strFileName, txtFile.Text);
}
}

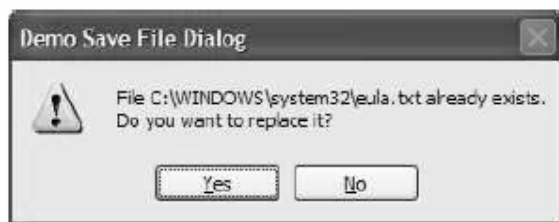
```

۵) در این مرحله می‌توانید برنامه خود را تست کنید، بنابراین پروژه را اجرا کرده و متن ساده‌ای را داخل آن وارد کنید. سپس بر روی دکمه‌ی Save کلیک کنید. مشاهده خواهید کرد که کادر محاوره‌ای Save نمایش خواهد شد.

۶) نامی را برای فایل انتخاب کرده و بر روی دکمه‌ی ok کلیک کنید. به این ترتیب متن داخل TextBox در فایلی با نام و مسیری که مشخص کرده بودید ذخیره می‌شود. برای امتحان این مورد می‌توانید با کلیک کردن بر روی دکمه‌ی Open فایل ایجاد شده را مجدداً در برنامه باز کرده و مشاهده کنید.

۷) برای تست عملکرد خصوصیت OverwritePrompt در کنترل SaveFileDialog متن دیگری را در TextBox وارد کرده و بر روی دکمه‌ی Save کلیک کنید. مجدداً مسیر و نام فایل قبلی را برای ذخیره فایل جدید وارد کنید. مشاهده خواهید کرد که پیغامی همانند شکل ۱۶-۹ نمایش داده می‌شود و می‌گویید که فایلی با این نام موجود است. آیا می‌خواهید آن را با این فایل تعویض کند؟ در صورتی که بر روی گزینه Yes کلیک کنید، فایل قبلی پاک می‌شود و فایل جدید به جای آن قرار می‌گیرد. اگر بر روی گزینه NO کلیک کنید، به کادر Save بر می‌گردید تا نام دیگری را برای فایل انتخاب کنید.

شکل ۱۶-۹



نکته: هنگامی که صفحه Save و یا Open نمایش داده می‌شود، منویی که به وسیله کلیک راست نمایش داده، اجازه می‌دهد کارهایی را از قبیل انتقال فایل به محلی دیگر، حذف فایل و یا تغییر نام آن را انجام دهید. همچنین بر حسب اینکه چه نرم‌افزاری بر روی سیستم شما نصب شده باشند، گزینه‌های دیگری نیز در این منو نمایش داده می‌شوند. برای مثال اگر WinZip یا WinRar بر روی سیستم شما نصب شده باشد، در این پنجره می‌توانید فایل‌ها را فشرده کنید.

بررسی نکات مهم برنامه

در این برنامه ابتدا خصوصیت DefaultExt را تنظیم کرده‌ایم. اگر کاربر هنگام مشخص کردن نام فایل پسوندی برای آن مشخص نکرده باشد، پسوندی که در این خصوصیت مشخص شده است، به طور اتوماتیک به انتهای فایل اضافه می‌شود

```
"saveFileDialog1.DefaultExt= " txt
```

خصوصیت OverwritePrompt مقداری را از نوع Boolean قبول می‌کند. اگر مقدار این خصوصیت را برابر با true قرار دهید، در صورتی که کاربر بخواهد فایلی را بر روی فایل دیگری ذخیره کند، به او هشدار داده می‌شود. اگر مقدار این خصوصیت برابر با false باشد، در صورت رخ دادن چنین مشکلی، بدون اینکه موردی به کاربر اطلاع داده شود، فایل قبلی پاک می‌شود و فایل جدید بر روی آن ذخیره می‌شود.

```
saveFileDialog1.OverwritePrompt = true;
```

بقیه موارد در مثال قبلی بوده است.

۱۶-۴- کنترل FontDialog

شاید بعضی مواقع بخواهید به کاربر اجازه دهید که فونت خاصی را انتخاب کند، تا اطلاعات او با آن فونت نمایش داده شوند، و یا ممکن است بخواهید لیستی از تمام فونت‌هایی که در سیستم کاربر نصب شده است را در برنامه استفاده کنید. در این موقع می‌توانید از کنترل FontDialog استفاده کنید. این کنترل تمام فونت‌های نصب شده در سیستم کاربر را در یک کادر استاندارد نمایش می‌دهد و به کاربر اجازه می‌دهد تا فونت خاصی را از بین آنها انتخاب کند.

همانند کادرهای OpenFileDialog و SaveFileDialog، کادر محاوره‌ای FontDialog هم می‌تواند به صورت یک کنترل و هم می‌تواند به صورت یک کلاس مورد استفاده قرار بگیرد. استفاده از خصوصیت‌های کادر مشخص کنید که کدام فونت توسط کاربر انتخاب شده است.

خصوصیت‌های کنترل FontDialog

جدول ۱۶-۸ لیستی از خصوصیت‌های پرکاربرد FontDialog را نمایش می‌دهد.

جدول ۱۶-۸

شرح	خصوصیت
مشخص می‌کند آیا کاربر می‌تواند با استفاده از قسمت Script کادر، مجموعه کاراکترهایی جدای از آنچه که در قسمت Script مشخص شده است را انتخاب کند یا نه؟ در صورت اینکه مقدار این خصوصیت برابر با true باشد، تمام مجموعه کاراکترهای موجود در قسمت Script نمایش داده می‌شود.	AllowScriptChange
رنگ فونت انتخاب شده را مشخص می‌کند.	Color
نام فونت انتخاب شده را مشخص می‌کند	Font
مشخص می‌کند اگر کاربر نام فونتی را انتخاب کرد که وجود نداشت، کادر پیغامی برای خطا نمایش داده شود یا نه؟	FontMustExist
حداکثر اندازه‌ای که کاربر می‌تواند برای فونت انتخاب کند را مشخص می‌کند.	MaxSize
حداقل اندازه‌ای که کاربر می‌تواند برای فونت انتخاب کند را مشخص می‌کند.	MinSize
مشخص می‌کند کادر محاوره‌ای که نمایش داده می‌شود باید دارای دکمه‌ی Apply نیز باشد یا نه؟	ShowApply
مشخص می‌کند در کادر فونت، امکان انتخاب رنگ نیز وجود داشته باشد یا نه؟	ShowColor
مشخص می‌کند آیا کادر فونت باید قسمتی برای تعیین زیر خط داربودن و یا انتخاب رنگ متن توسط کاربر را داشته باشد یا نه؟	ShowEffects
مشخص می‌کند آیا کادر فونت دارای دکمه فرمان Help باشد یا نه؟	ShowHelp

متدهای کنترل FontDialog

در مثال‌های بعدی فقط از یکی از متدهای کنترل FontDialog استفاده خواهیم کرد که آن نیز متد ShowDialog برای نمایش کادر محاوره‌ای خواهد بود، علاوه بر این متدهای زیادی برای این کنترل وجود دارند، مانند متد Reset که باعث می‌شود مقدار تمام خصوصیت‌های کنترل به حالت اول برگردد.

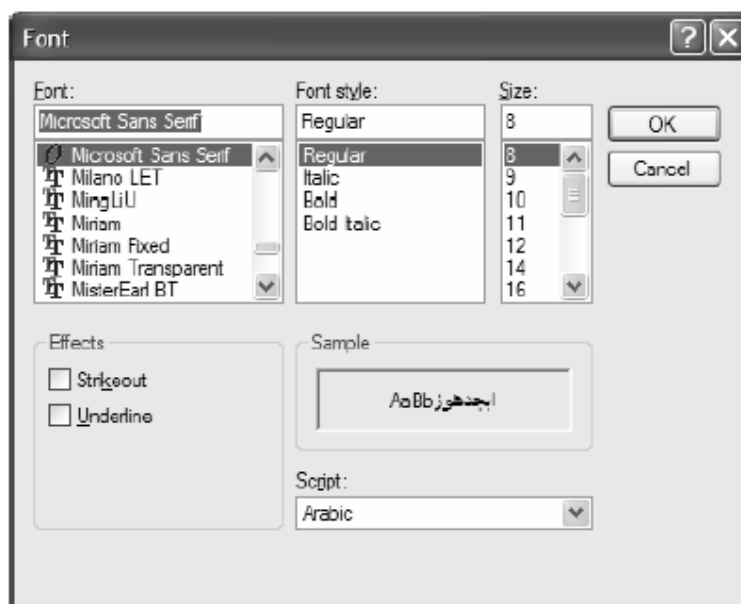
استفاده از کنترل FontDialog

برای نمایش کنترل FontDialog نیاز به تنظیم هیچ مقداری نیست، بلکه می‌توان به طور مستقیم متد ShowDialog این کنترل را فراخوانی کرد تا کادر محاوره‌ای نمایش داده شود.

```
fontDialog1.ShowDialog();
```

در این صورت کادر محاوره‌ای همانند شکل ۱۶-۱۰ نمایش داده می‌شود.

شکل ۱۶-۱۰



مشاهده می‌کنید که کادر فونت دارای یک بخش Effects است که به کاربر اجازه می‌دهد تعیین کند یک فونت دارای خط و یا زیرخط نیز باشد. نمایش این بخش به این علت است که خصوصیت ShowEffects بطور پیش‌فرض دارای مقدار true است. در این کادر قسمت Color برای انتخاب رنگ نمایش داده نشده است. زیرا مقدار پیش‌فرض ShowColor برابر با false است. برای نمایش قسمت رنگ، بایستی قبل از فراخوانی تابع ShowDialog مقدار این خصوصیت را برابر با true قرار داد.

```
fontDialog1.ShowColor = true;
fontDialog1.ShowDialog();
```

متد ShowDialog از این کادر محاوره‌ای نیز، همانند تمام متدهای ShowDialog مقداری را از نوع DialogResult بر می‌گرداند. این مقدار می‌تواند برابر با DialogResult.Ok و یا DialogResult.Cancel باشد.

هنگامی که کادر فونت نمایش داده شد و کاربر بر روی گزینه OK کلیک کرد، می‌توانید با استفاده از خصوصیت‌های Color و Font کنترل FontDialog بررسی کنید که کاربر چه نوع فونت و چه رنگی را انتخاب کرده است و سپس آن را در برنامه استفاده کنید و یا در متغیری قرار داده و در بخش‌های بعدی استفاده کنید.

حال که با این کادر و نحوه کارکرد آن آشنا شدید، در مثال بعدی از آن استفاده خواهیم کرد. در بخش بعد از برنامه‌ای که در دو مثال قبلی ایجاد کرده بودیم استفاده می‌کنیم و آن را مقداری گسترش می‌دهیم. در قسمت‌های قبلی کاربر می‌توانست در

برنامه فایلی را باز کرده، تغییراتی را در آن انجام دهد و سپس فایل را ذخیره کند، در این قسمت بخشی را به برنامه اضافه می‌کنیم که کاربر به وسیله آن بتواند فونت متن درون TextBox را تغییر دهد.

مثال ۱۶-۴- کار با کنترل FontDialog

(۱) مجدداً پروژهی قبلی را باز کنید.

(۲) در قسمت طراحی فرم، کنترل Button دیگری به فرم اضافه کرده و خصوصیت‌های آن را مطابق لیست زیر تعیین کنید:

- خصوصیت Name را برابر با btnFont قرار دهید.
- خصوصیت Anchor را برابر با Top-Right قرار دهید.
- خصوصیت Anchor را برابر با ۶۸-۳۶۷ قرار دهید.
- خصوصیت Text را برابر با Font قرار دهید.

(۳) برای نمایش کادر فونت، باید یک کنترل FontDialog بر روی فرم قرار دهید. برای این کار در جعبه ابزار به قسمت Dialog بروید و در آنجا بر روی کنترل FontDialog دو بار کلیک کنید. به این صورت یک کنترل FontDialog در قسمت پایین محیط طراحی اضافه خواهد شد. تمام تنظیمات پیش‌فرض این کنترل را قبول کنید و خصوصیت‌های آن را تغییر ندهید.

(۴) بر روی دکمه‌ی btnFont دو بار کلیک کنید تا متد مربوط به رویداد کلیک آن ایجاد شود. سپس کد زیر را به آن متد اضافه کنید:

```
private void btnFont_Click(object sender, EventArgs e)
{
    // Set the FontDialog control properties
    fontDialog1.ShowColor = true;
    // Show the Font dialog
    if (fontDialog1.ShowDialog() == DialogResult.OK)
    {
        // If the OK button was clicked set the font
        // in the text box on the form
        txtFile.Font = fontDialog1.Font;
        // Set the color of the font in the text box
        // on the form
        txtFile.ForeColor = fontDialog1.Color;
    }
}
```

(۵) برنامه را اجرا کنید. هنگامی که فرم برنامه نمایش داده شد، بر روی دکمه‌ی Font کلیک کنید تا کادر محاوره‌ای Font همانند شکل ۱۱-۱۶ نمایش داده شود. فونت و رنگ جدیدی را برای TextBox انتخاب کرده و بر روی دکمه OK کلیک کنید.

(۶) حال چندین خط متن را در فرم وارد کنید. مشاهده خواهید کرد که متن با فونت و رنگ جدید نوشته خواهد شد.

(۷) همچنین اگر فایلی را با استفاده از دکمه‌ی Open باز کنید، رنگ و فونت جدید در آن اعمال می‌شود. برای تست این مورد، روی دکمه‌ی Open کلیک کنید تا کادر Open نمایش داده شود. سپس یک فایل متنی را انتخاب کرده و آن را باز کنید. مشاهده می‌کنید که محتویات فایل با رنگ و فونت جدید نمایش داده می‌شود.

شکل ۱۱-۱۶

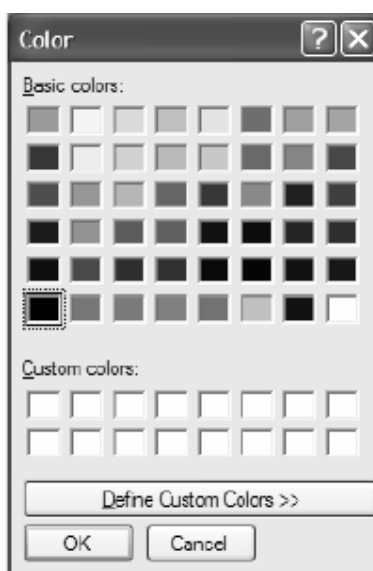


۱۶-۵- کنترل ColorDialog

شاید لازم باشید به کاربر اجازه دهید رنگی را در برنامه انتخاب کند. برای مثال، ممکن است بخواهید از این رنگ در تنظیم رنگ پس زمینه‌ی فرم، در تنظیم رنگ یک کنترل و یا برای تنظیم رنگ متن داخل TextBox استفاده کنید. ۲۰۰۵VS همانند کادر font، یک کادر استاندارد نیز برای انتخاب رنگ در اختیار برنامه‌نویس قرار می‌دهد که ColorDialog نام دارد. همانند قسمت‌های قبلی، کادر ColorDialog می‌تواند به عنوان یک کنترل و هم به عنوان یک کلاس مورد استفاده قرار گیرد.

کنترل ColorDialog که در شکل ۱۶-۱۲ نشان داده شده است، به کاربر اجازه می‌دهد بین ۴۸ رنگ ابتدایی رنگی را انتخاب کند.

شکل ۱۶-۱۲



دقت کنید که علاوه بر این رنگ‌های ابتدایی کاربر می‌تواند بر روی دکمه‌ی Define Custom Colors کلیک کرده و با ترکیب رنگ‌ها، رنگ مورد نظر خود را ایجاد کند. این مورد باعث انعطاف‌پذیری بیشتر این کادر می‌شود و به کاربر اجازه می‌دهد رنگ مورد نظر خود را ایجاد کرده و در برنامه از آن استفاده کند (شکل ۱۶-۱۳).

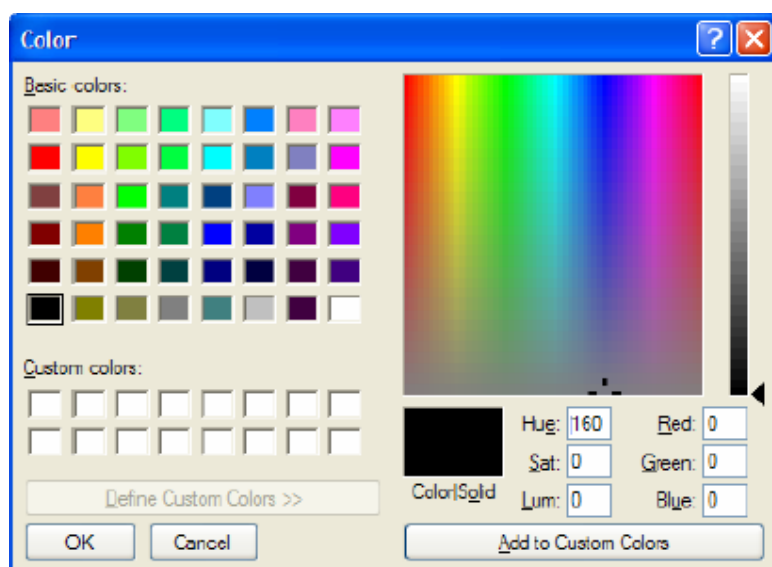
خصوصیت‌های کنترل ColorDialog

قبل از اینکه از این کنترل استفاده کنیم، بهتر است بعضی از خصوصیت‌های پرکاربرد آن را بررسی کنیم. در جدول ۹-۱۶ نام تعدادی از آن خصوصیت‌ها و کاربرد آنها شرح داده شده است:

جدول ۹-۱۶

شرح	خصوصیت
مشخص می‌کند که آیا کاربر می‌تواند از قسمت Custom Color نیز برای تعریف رنگ جدید استفاده کند یا نه. در صورتی که مقدار این گزینه برابر با false باشد، دکمه فرمان Define Custom Colors غیر فعال خواهد بود.	AllowFullOpen
مشخص می‌کند که آیا کادر محاوره‌ای تمام رنگ‌های موجود را به عنوان رنگ‌های ابتدایی نمایش دهد یا نه؟	AnyColor
رنگی که در کادر به وسیله کاربر انتخاب شده است، را مشخص می‌کند.	Color
مجموعه رنگ‌هایی را در بخش Custom Color کادر نمایش داده می‌شود را مشخص می‌کند.	CustomColors
مشخص می‌کند که هنگام نمایش داده شدن کادر Color قسمت Custom Color هم به صورت پیش‌فرض دیده شود یا نه؟	FullOpen
مشخص می‌کند که دکمه فرمان Help در کادر Color نمایش داده شود یا نه؟	ShowHelp

شکل ۱۶-۱۳



همانطور که مشاهده می‌کنید، خصوصیت‌های این کنترل نسبت به کنترل‌های قبلی کمتر است. همین مورد باعث می‌شود که استفاده از این کنترل حتی از کنترل‌های قبلی نیز راحت‌تر باشد.

همانند کادرهای قبلی کنترل، ColorDialog نیز دارای تابع ShowDialog است که باعث نمایش آن می‌شود. نحوه کارکرد این تابع نیز همانند قسمت‌های قبلی است. بنابراین در این قسمت از توضیح مجدد آن صرف‌نظر می‌کنیم.

استفاده از کنترل ColorDialog

برای نمایش کادر Color تنها کافی است که متد ShowDialog آن را فراخوانی کنید:

```
colorDialog1.ShowDialog();
```

این تابع مقداری را از نوع DialogResult بر می‌گرداند، که مشخص می‌کند کاربر در کادر بر روی دکمه OK کلیک کرده است و یا بر روی دکمه Cancel.

برای دسترسی به مقدار رنگی که توسط کاربر در این کادر انتخاب شده است، باید از خصوصیت Color این کنترل استفاده کنید. سپس می‌توانید این رنگ را به کنترل‌هایی که می‌توانید رنگ آنها را تعیین کنید نسبت دهید. برای مثال، می‌توانید رنگ متن یک TextBox را برابر با رنگ انتخاب شده در این کادر قرار دهید:

```
txtFile.ForeColor = colorDialog1.Color;
```

در مثال بعدی به پروژه‌ی قبلی امکانی را اضافه می‌کنیم که کاربر بتواند به وسیله آن رنگ زمینه فرم را تغییر دهد.

مثال ۱۶-۵-کار با کنترل ColorDialog

(۱) پروژه Dialogs را باز کرده و به قسمت طراحی فرم مربوط به form1 بروید.

(۲) با استفاده از جعبه ابزار یک کنترل Button بر روی فرم قرار داده و خصوصیت‌های آن را مطابق با مقادیر زیر تنظیم کنید:

- خصوصیت Name آن را برابر با btnColor قرار دهید.
- خصوصیت Anchor آن را برابر با Top-Right قرار دهید.
- خصوصیت Location آن را برابر با ۳۶۷-۹۸ قرار دهید.
- خصوصیت Text آن را برابر با color قرار دهید.

(۳) سپس با استفاده از قسمت Dialogs جعبه ابزار، یک کنترل ColorDialog به برنامه اضافه کنید. این کنترل به قسمت پایین طراحی فرم اضافه خواهد شد.

(۴) بر روی دکمه‌ی btnColor دوبار کلیک کرده تا متد مربوط به رویداد click آن ایجاد شود. سپس کد زیر را به آن اضافه کنید.

```
private void btnColoe_Click(object sender, EventArgs e)
{
    // Show the Color dialog
    if (colorDialog1.ShowDialog() == DialogResult.OK)
    {
        // Set the BackColor property of the form
        this.BackColor = colorDialog1.Color;
    }
}
```

(۵) تمام کدی که باید وارد می‌کردید همین بود. برای امتحان برنامه را اجرا کنید.

(۶) هنگامی که فرم برنامه نمایش داده شد. بر روی دکمه‌ی Color کلیک کنید تا کادر محاوره‌ای color نمایش داده شود. در این کادر یکی از رنگ‌های ابتدایی را انتخاب کرده و یا روی دکمه Define Custom Colors کلیک کنید و رنگی را از آن قسمت انتخاب کنید. سپس روی دکمه OK کلیک کنید تا کادر بسته شود.

- ۷) با کلیک روی دکمه Ok در کادر Color رنگ زمینه فرم با رنگی که در کادر انتخاب کرده بودید تعویض می‌شود.
- ۸) همانند کادر Font نیازی نیست که قبل از نمایش فرم خصوصیت Color را برابر رنگ انتخاب شده در مرحله قبلی قرار دهید. زیرا کنترل ColorDialog خود مقدار رنگی که آخرین بار توسط کاربر انتخاب شده است را نگهداری می‌کند. به این ترتیب بعد از اینکه کاربر مجدداً وارد این کادر شد، مشاهده می‌کند رنگی که در مرحله قبل انتخاب کرده بود همچنان به صورت انتخاب شده است.

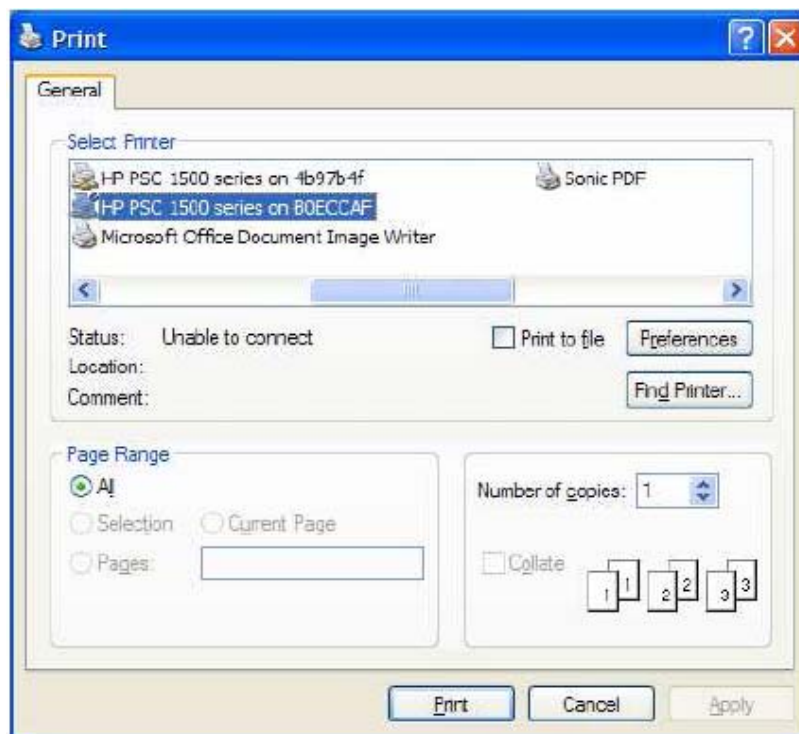
۱۶-۶- کنترل PrintDialog

به احتمال زیاد هر برنامه‌ای معمولاً نیاز به امکان چاپ دارد. این نیاز می‌تواند به صورت نیاز به چاپ ساده‌ی یک متن و یا موارد پیشرفته‌تری مانند چاپ قسمتی از متن و یا صفحات مشخصی از آن باشد. در قسمت بعد به بررسی چگونگی چاپ یک متن ساده خواهیم پرداخت و نحوه استفاده از کلاس‌های مربوط به چاپ در .NET را مشاهده خواهیم کرد.

یکی از کنترل‌هایی که در VC#۲۰۰۵ برای چاپ به کار می‌رود، کنترل PrintDialog است. این کنترل کار چاپ را انجام نمی‌دهد، بلکه به کاربر اجازه می‌دهد که چاپگری را برای چاپ انتخاب کرده و تنظیمات قبل از چاپ را در آن چاپگر انجام دهد. برای مثال، کاربر می‌تواند در این کادر جهت صفحه، کیفیت چاپ و یا محدوده موردنظر برای چاپ را تعیین کند. شما از این ویژگی‌ها در مثال بعدی استفاده نخواهید کرد، اما همانطور که در شکل ۱۶-۱۴ مشاهده می‌کنید، تمام این قابلیت‌ها به وسیله کادر PrintDialog قابل دسترسی است.

همانند تمام کادرهایی که در بخش‌های قبلی مشاهده کردید، کادر Print نیز دارای دو دکمه Ok و Cancel است. بنابراین تابع ShowDialog مربوط به این کادر هم مقدار DialogResult.Ok و یا DialogResult.Cancel را بر می‌گرداند و می‌توانید از دستور if برای بررسی نتیجه برگشت داده شده توسط کادر استفاده کنید.

شکل ۱۶-۱۴



خصوصیت‌های کنترل PrintDialog

در جدول ۱۰-۱۶ لیستی از خصوصیت‌های پرکاربرد کنترل PrintDialog و نیز توضیح آنها آمده است :

جدول ۱۰-۱۶

شرح	خصوصیت
مشخص می‌کند آیا گزینه Print To File در کادر فعال باشد یا نه ؟	AllowPrintToFile
مشخص می‌کند در کادر، دکمه رادیویی Selectin فعال باشد یا نه ؟	AllowSelection
مشخص می‌کند در کادر، دکمه رادیویی Pages فعال باشد یا نه ؟	AllowSomePages
سندی که برای چاپ استفاده می‌شود را مشخص می‌کند.	Document
تنظیماتی که در کادر برای چاپگر انتخابی اعمال می‌شود را نگهداری می‌کند.	PrinterSettings
مشخص می‌کند آیا گزینه Print To File انتخاب شده است یا نه ؟	PrintToFile
مشخص می‌کند آیا دکمه فرمان Help در کادر نمایش داده شود یا نه ؟	ShowHelp
مشخص می‌کند دکمه فرمان Network در کادر print نمایش داده می‌شود یا نه ؟	ShowNetwork

استفاده از کنترل PrintDialog

برای نمایش کادر Print، کافی است که متد ShowDialog آن را فراخوانی کنید. به این صورت کادر print همانند شکل ۱۴-۱۶ نشان داده خواهد شد. همانطور که پیشتر نیز گفتیم کنترل PrintDialog فقط کادری را برای تنظیمات چاپ نمایش می‌دهد و نمی‌تواند هیچ متنی را چاپ کند. قطعه کد زیر می‌تواند برای نمایش کادر print مورد استفاده قرار بگیرید:

```
PrintDialog.ShowDialog();
```

۱۶-۶-۱- کلاس PrintDocument

قبل از اینکه متد ShowDialog در کنترل PrintDialog را فراخوانی کنید باید خصوصیت Document کلاس PrintDialog را تنظیم کنید این خصوصیت مقداری را از نوع کلاس PrintDocument دریافت می‌کند. کلاس PrintDocument می‌تواند تنظیمات چاپگر را دریافت کرده و سپس با توجه به آن تنظیمات، خروجی خود (که در حقیقت همان اطلاعات موردنظر ما است) را برای چاپ به چاپگر فرستد. این کلاس در فضای نامی System.Drawing.Printing قرار دارد. پس بهتر است که قبل از استفاده از آن، برای اینکه هر بار نام کامل این فضای نام را وارد نکنیم، با استفاده از راهنمای using آن را به برنامه اضافه کنیم.

خصوصیات کلاس PrintDocument

قبل از ادامه، بهتر است نگاهی به بعضی از خصوصیات مهم کلاس PrintDocument که در جدول ۱۱-۱۶ آمده‌اند داشته باشیم.

جدول ۱۱-۱۶

شرح	خصوصیت
مشخص کننده‌ی تنظیمات پیش‌فرض چاپگر برای چاپ سند (اطلاعات) مورد نظر	DefaultPageSettings

است.	
مشخص کننده نامی است که هنگام چاپ سند نمایش داده می‌شود. همچنین این نام در کادر Print Status و در لیست اسناد موجود در صف چاپ برای مشخص کردن سند می‌شود.	DocumentName
محتوای شیئی از کلاس PrintController است که پروسه را مدیریت می‌کند.	PrintController
مشخص کننده چاپگری است که برای چاپ این سند می‌شود.	PrinterSettings

چاپ یک سند

متد Print از کلاس PrintDocument، سندی را به وسیله چاپگر مشخص شده در خصوصیت PrinterSettings چاپ می‌کند. هنگامی که این متد را در برنامه فراخوانی کنید، هر بار که صفحه‌ای بخواهد به وسیله این متد چاپ شود، متد مربوط به رویداد PrintPage از کلاس PrintDocument نیز فراخوانی می‌شود. متد Print به وسیله این متد مشخص می‌کند که کدام بخش از فایل باید در صفحه جاری چاپ شود. بنابراین قبل از اینکه بتوانید متنی را چاپ کنید، باید متدی را برای این رویداد ایجاد کنید. سپس در این متد باید یک صفحه از متن را به وسیله کلاس StreamReader از فایل خوانده و آن را به چاپگر بفرستید تا چاپ شود.

در مثال ۱۶-۶ مشاهده خواهیم کرد که چگونه می‌توان محتویات یک فایل متنی را به وسیله کلاس PrintDocument چاپ کرد.

مثال ۱۶-۶- کار با کنترل PrintDialog

(۱) در محیط VS.NET، پروژه Dialogs را باز کنید.

(۲) با استفاده از جعبه ابزار کنترل Button دیگری را بر روی فرم قرار داده و خصوصیت‌های آن را مطابق لیست زیر تنظیم کنید:

- خصوصیت Name را برابر با btnPrint قرار دهید.
- خصوصیت Anchor را برابر با Top-Right قرار دهید.
- خصوصیت Location را برابر با ۱۲۸-۳۶۷ قرار دهید.
- خصوصیت Text را برابر با Print قرار دهید.

(۳) در جعبه ابزار به قسمت Printing بروید و بر روی کنترل PrintDialog دوبار کلیک کنید تا بر روی فرم قرار بگیرد. مشاهده خواهید کرد که این کنترل نیز همانند کادرهای قبلی، در پایین قسمت طراحی فرم قرار می‌گیرد.

(۴) به قسمت ویرایشگر کد بروید و با استفاده از راهنمای using فضاهای نامی زیر را به برنامه اضافه کنید:

```
using System.IO;
using System.Drawing.Printing;
```

(۵) حال متغیرهای زیر را به صورت عمومی در ابتدای کلاس مربوط به فرم برنامه تعریف کنید.

```
// Declare variables
private string strFileName;
private StreamReader objStreamToPrint;
private Font objPrintFont;
```

۶) به قسمت طراحی فرم برگردید و بر روی دکمه‌ی btnPrint دو بار کلیک کنید تا متد مربوط به رویداد کلیک آن ایجاد شود. سپس کد زیر را در این متد وارد کنید.

```
private void btnPrint_Click(object sender, EventArgs e)
{
    // Declare an object for the PrintDocument class
    PrintDocument objPrintDocument = new PrintDocument();
    // Set the DocumentName property
    objPrintDocument.DocumentName = "Text File Print
    Demo";
    // Set the PrintDialog properties
    printDialog1.AllowPrintToFile = false;
    printDialog1.AllowSelection = false;
    printDialog1.AllowSomePages = false;
    // Set the Document property for
    // the objPrintDocument object
    printDialog1.Document = objPrintDocument;
    // Show the Print dialog
    if (printDialog1.ShowDialog() == DialogResult.OK)
    {
        // If the user clicked on the OK button
        // If the user clicked on the OK button
        // then set the StreamReader object to
        // the file name in the strFileName variable
        objStreamToPrint = new StreamReader(strFileName);
        // Set the printer font
        objPrintFont = new Font("Arial", ۱۰);
        // Set the PrinterSettings property of the
        // objPrintDocument Object to the
        // PrinterSettings property returned from the
        // PrintDialog control
        objPrintDocument.PrinterSettings =
        printDialog1.PrinterSettings;
        // Add an event handler for the PrintPage event
        of
        // the objPrintDocument object
        objPrintDocument.PrintPage +=
        new PrintPageEventHandler(prtPage);
        // Print the text file
        objPrintDocument.Print();
        // Clean up
        objStreamToPrint.Close();
        objStreamToPrint = null;
    }
}
```

۷) سپس متد زیر را در قسمت ویرایشگر کد وارد کنید.

```
private void prtPage(object sender, PrintPageEventArgs e)
{
    // Declare variables
    float sngLinesPerpage = ۰;
    float sngVerticalPosition = ۰;
    int intLineCount = ۰;
    float sngLeftMargin = e.MarginBounds.Left;
    float sngTopMargin = e.MarginBounds.Top;
    string strLine;
    // Work out the number of lines per page.
    // Use the MarginBounds on the event to do this
    sngLinesPerpage = e.MarginBounds.Height /
    objPrintFont.GetHeight(e.Graphics);
```



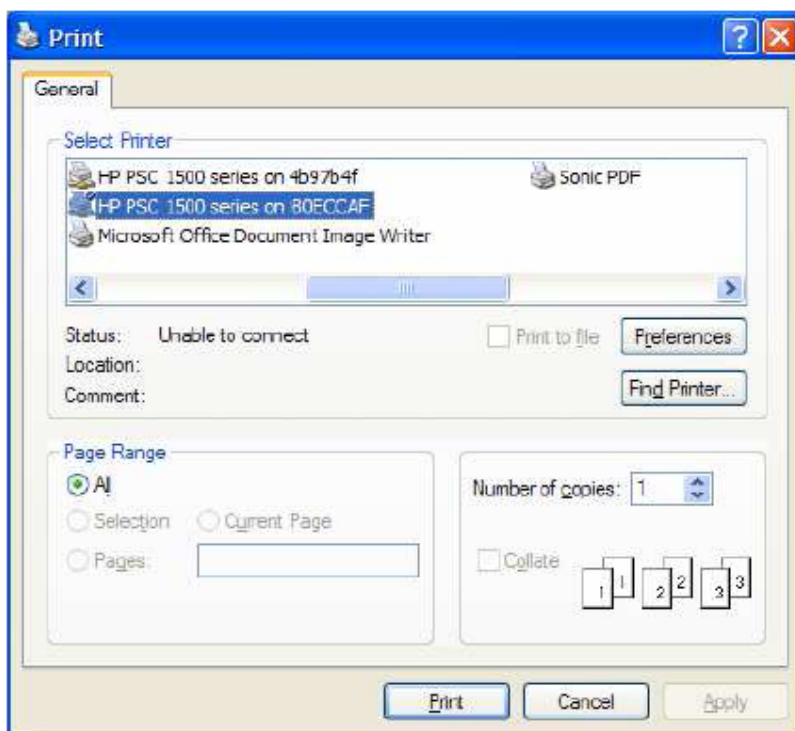
```
// Now iterate through the file printing out each
line.
// This assumes that a single line is not wider than
// the page width. Check intLineCount first so that we
// don't read a line that we won't print
strLine = objStreamToPrint.ReadLine();
while((intLineCount < sngLinesPerpage) &&
(strLine != null))
{
// Calculate the vertical position on the page
sngVerticalPosition = sngTopMargin +
(intLineCount * objPrintFont.GetHeight(e.Graphics));
// Pass a StringFormat to DrawString for the
// Print Preview control
e.Graphics.DrawString(strLine, objPrintFont,
Brushes.Black, sngLeftMargin,
sngVerticalPosition,
new StringFormat());
// Increment the line count
intLineCount = intLineCount + 1;
// If the line count is less than the lines per
// page then read another line of text
if (intLineCount < sngLinesPerpage)
{
strLine = objStreamToPrint.ReadLine();
}
}
// If we have more lines then print another page
if (strLine != null)
{
e.HasMorePages = true;
}
else
{
e.HasMorePages = false;
}
}
```

۸) حال می‌توانید عملکرد کدهای این قسمت را مشاهده کنید. بنابراین روی دکمه start در نوار ابزار کلیک کنید تا برنامه اجرا شود.

۹) در فرم اصلی برنامه، روی دکمه‌ی Open کلیک کنید و فایلی را باز کنید تا محتویات آن در فرم نمایش داده شود. سپس بر روی دکمه‌ی print کلیک کنید تا کادر محاوره‌ای Print همانند شکل ۱۶-۱۵ نمایش داده شود. توجه کنید که در این کادر گزینه Print To File و همچنین قسمت‌های selection و pages غیرفعال هستند. دلیل غیرفعال بودن این قسمت‌ها به خاطر این است که قبل از فرخوانی متد ShowDialog خصوصیت‌های AllowSelection و AllowPrintToFile را برابر با false قرار دادیم. اگر در سیستم خود بیش از یک چاپگر داشته باشید، همانند شکل ۱۶-۱۵ می‌توانید تعیین کنید که فایل باز شده به وسیله کدام چاپگر، چاپ شود.

۱۰) روی دکمه Print کادر کلیک کنید تا محتویات فایل چاپ شوند.

شکل ۱۶-۱۵



بررسی مثال چاپ

این مثال را با تعریف یک شی از کلاس `PrintDocument` آغاز کردیم. عمل اصلی چاپ به وسیله این شی صورت می‌گیرد:

```
PrintDocument objPrintDocument = new PrintDocument();
```

سپس خصوصیت `DocumentName` مربوط به این شی را تنظیم کردیم. اگر همزمان چند برنامه بخواهند از چاپگر استفاده کنند، سند‌های آنها در یک صف چاپ قرار می‌گیرد. نامی که در این قسمت وارد می‌کنیم، برای مشخص کردن سند مربوط به برنامه‌ی ما در صف چاپ به کار می‌رود.

```
objPrintDocument.DocumentName = "Text File Print Demo";
```

در قسمت بعد به تنظیم بعضی از خصوصیت‌های کنترل `PrintDialog` می‌پردازیم. در این بخش فقط می‌خواهیم یک عمل چاپ ساده را انجام دهیم، به همین دلیل بهتر است قسمت‌های `Print To File` و همچنین `Selection` و `Pages` را در کادر `Print` غیرفعال کنیم. برای این کار کافی است خصوصیت‌های مربوط به آنها را برابر با `false` قرار دهیم:

```
printDialog.AllowPrintToFile = false;
printDialog.AllowSelection = false;
printDialog.AllowSomePages = false;
```

قبل از نمایش کادر `Print` باید مشخص کنید تنظیماتی که کاربر در این کادر مشخص می‌کند برای چاپ چه سندی به کار می‌روند. برای این کار باید خصوصیت `Document` کنترل `PrintDialog` را برابر با شی `PrintDocument` ای قرار دهید که نشان دهنده‌ی سند موردنظر است.

```
printDialog.Document = objPrintDocument;
```

حال می‌توانیم کادر `Print` را نمایش دهیم. همانند کادرهای قبلی برای این کار کافی است متد `ShowDialog` مربوط به این کنترل را فراخوانی کنیم. این متد نیز مقداری را از نوع `DialogResult` بر می‌گرداند. اگر کاربر در کادر روی دکمه‌ی `Print` کلیک کند، مقدار `DialogResult.Cancel` را بر می‌گرداند. همانند قسمت‌های قبلی با استفاده از دستور `if` نتیجه را بررسی می‌کنیم.

اگر کاربر در کادر روی دکمه `Print` کلیک کند، باید محتویات فایلی که آدرس آن در متغیر `strFileName` قرار دارد را چاپ کنیم. بنابراین ابتدا یک شی از نوع `StreamReader` تعریف می‌کنیم. این شی برای دسترسی به محتویات یک فایل

مورد استفاده قرار می‌گیرد و هنگام تعریف آن باید آدرس فایل موردنظر را به آن بفرستیم. پس متغیر `strFileName` که حاوی آدرس فایل است را به عنوان پارامتر به این شی ارسال می‌کنیم.

```
objStreamToPrint = new StreamReader(strFileName);
```

سپس باید فونت و اندازه متن را برای چاپ مشخص کنیم. به همین علت شیئی را از نوع `Font` تعریف کرده و فونت `Arial` و اندازه ۱۰ را برای آن تعریف می‌کنیم.

```
objPrintFont = new Font("Arial", ۱۰);
```

همانطور که در قسمت‌های قبلی نیز گفتیم، هنگامی که یک رویداد به وسیله یک کلاس رخ می‌دهد، تعدادی از متدها برای پاسخ دادن به آن رویداد اجرا می‌شوند. برای مثال، در قسمت‌های قبل مشاهده کردید، در زمان طراحی با دوبار کلیک بر روی کنترل `Button`، متدی ایجاد می‌شد و این متد در طی اجرای برنامه هنگامی که کاربر روی آن کنترل کلیک می‌کرد، توسط برنامه فراخوانی می‌شد. برای بررسی دقیق‌تر این مورد باید بگوییم که هر رویداد شامل لیستی از متدها است. هنگامی که رویداد رخ می‌دهد، کلاس مربوطه تمام متدهای موجود در لیست مربوط به آن رویداد را فراخوانی می‌کند. برای مثال می‌توانید چندین متد تعریف کنید و آنها را به رویداد کلیک یک `Button` اضافه کنید. به این ترتیب هنگامی که بر روی آن دکمه کلیک شود، تمام متدهایی که به آن اضافه کرده‌اید اجرا خواهند شد.

در قسمت‌های قبلی گفتیم که کلاس `PrintDocument` برای اینکه تشخیص دهد چه متنی را باید چاپ کند، در هر صفحه رویداد `PrintPage` را فراخوانی می‌کند. به عبارت دقیق‌تر، باید بگوییم که این کلاس در هر مرحله تمام توابعی که در لیست رویداد `PrintPage` هستند را اجرا می‌کند. پس باید متدی را ایجاد کنیم و آن را به لیست متدهای رویداد `PrintPage` اضافه کنیم. برای این کار متد `prtPage` را ایجاد کرده و آن را به وسیله دستور زیر به رویداد `PrintPage` اضافه می‌کنیم:

```
objPrintDocument.PrintPage += new PrintPageEventHandler(prtPage);
```

حال باید چاپگر مورد استفاده برای چاپ و همچنین تنظیم‌های آن را، برای شی `PrintDocument` مشخص کنیم. برای اینکار، کافی است تنظیم‌هایی که کاربر در کادر `Print` مشخص کرده است را به این شی بفرستیم. تنظیم‌های کادر `Print` در خصوصیت `PrinterSettings` ذخیره می‌شوند. پس کافی است، خصوصیت `objPrintDocument.PrinterSettings` را برابر با آن قرار دهیم.

```
objPrintDocument.PrinterSettings = printDialog.PrinterSettings;
```

حال باید متد `Print` را فراخوانی کنیم. این متد رویداد `PrintPage` را احضار می‌کند و احضار این رویداد نیز باعث می‌شود که کد درون متد `prtPage` اجرا شود.

```
objPrintDocument.Print();
```

نکته دیگری که در اضافه کردن یک متد به یک رویداد باید در نظر داشته باشید، این است که متدهایی که می‌توانند به رویداد `PrintPage` اضافه شوند، باید دارای ساختار خاصی باشند. این متدها نباید مقداری را برگردانند (باید مقدار برگشتی آنها به صورت `void` تعریف شود). همچنین باید دو پارامتر را از ورودی دریافت کنند. اولین پارامتر، مشخص کننده شیئی است که این رویداد را فراخوانی کرده است. نام این پارامتر `sender` و نوع آن کلاس `Object` خواهد بود. پارامتر باید شیئی باشد. بنابراین متد `prtPage` که باید به وسیله رویداد `PrintPage` فراخوانی شود مشابه زیر خواهد بود

```
private void prtPage(object sender, PrintPageEventArgs e)
```

حال به بررسی کدهایی می‌پردازیم که در داخل این متد باید اجرا شوند. ابتدا باید تعدادی متغیر تعریف کنیم و مقادیر آنها را تنظیم کنیم. توجه کنید که مقادیر متغیرهای `sngLeftMargin` و `sngTopMargin` به وسیله مقادیر موجود در پارامتر `printPageEventArgs` که به متد ارسال می‌شود تنظیم خواهد شد.

```
float sngLinesPerpage = ۰;
float sngVerticalPosition = ۰;
int intLineCount = ۰;
float sngLeftMargin = e.MarginBounds.Left;
```

```
float sngTopMargin = e.MarginBounds.Top;
string strLine;
```

حال باید مشخص کنیم که در هر لحظه چند خط می تواند چاپ شود. برای اینکار، باید ارتفاع قابل چاپ در صفحه را بر ارتفاع فونت (ارتفاع هر خط) تقسیم کنیم. برای دسترسی به ارتفاع قابل چاپ در صفحه می توانیم از خصوصیت `MarginBounds.Height` در شی `e` از کلاس `printPageEventArgs` استفاده کنیم (این شی به عنوان پارامتر به متد فرستاده شده است).

ارتفاع قابل چاپ در صفحه در کادر `PrintDialog` تنظیم شده و در خصوصیت `PrinterSettings` قرار می گیرد. همانطور که مشاهده کردید در کدهای قبلی این خصوصیت را در خصوصیت `PrinterSettings` مربوط به شی `objPrintDocument` قرار دادیم. شی `objPrintDocument` هم هنگامی که بخواهد رویداد `PrintPage` فراخوانی کند، این مقدار را به وسیله شی از کلاس `PrintPageEventArgs` به متدهای فراخوانی شده می فرستد.

```
sngLinesPerpage = e.MarginBounds.Height /
objPrintFont.GetHeight(e.Graphics);
```

پس به این ترتیب، متغیر `sngLinesPerPage` حاوی تعداد خطوطی خواهد بود که در هر صفحه قرار می گیرد. حال باید محتویات فایل را خط به خط خوانده و در صفحه برای چاپ قرار دهیم. برای این کار با استفاده از یک حلقه، متن داخل فایل را در خط به خط در صفحه وارد می کنیم. اجرای این حلقه تا زمانی ادامه پیدا می کند که یا متن داخل فایل تمام شود و به انتهای فایل برسیم و یا تعداد خطهایی که در صفحه قرار داده ایم برابر با حداکثر تعداد خطهایی که در صفحه قرار داده ایم شود، به عبارت دیگر صفحه پر شود. بنابراین ابتدا خط را خوانده و در متغیر `strLine` قرار می دهیم و سپس حلقه را اجرا می کنیم:

```
strLine = objStreamToPrint.ReadLine();
while((intLineCount < sngLinesPerpage) && (strLine !=
null))
{
```

قبل از اینکه متنی را در صفحه قرار دهیم، باید مشخص کنیم که موقعیت عمودی متن در صفحه چقدر است. به عبارت دیگر، باید فاصله متن را از بالای صفحه مشخص کنیم. برای تعیین فاصله باید اندازه قسمت سفید بالای صفحه را با ارتفاع تعداد خطهایی که تاکنون چاپ شده اند، در ارتفاع هر خط را بدست آورید:

```
sngVerticalPosition = sngTopMargin +
(intLineCount * objPrintFont.GetHeight(e.Graphics));
```

برای اینکه واقعا متن را به چاپگر بفرستیم، باید از متد `DrawString` در کلاس `Graphics` استفاده کنیم. کلاس `Graphics` بصورت یکی از خصوصیت های کلاس `printPageEventArgs` به این متد فرستاده می شود. پارامترهایی که متد `DrawString` دریافت می کند، عبارتند از: متنی که باید چاپ شود، فونت متنی که باید چاپ شود، رنگ متنی که باید چاپ شود (این رنگ باید از نوع شمارشی `Brushes` انتخاب شود). فاصله متن از سمت چپ صفحه، فاصله متن از بالای صفحه، و قالب متن برای چاپ. در این قسمت قالبی برای متن مشخص نمی کنیم بلکه یک شی جدید از کلاس `StringFormat` ایجاد کرده و آن را به متد می فرستیم.

```
e.Graphics.DrawString(strLine, objPrintFont,
Brushes.Black, sngLeftMargin,
sngVerticalPosition,
new StringFormat());
```

به این ترتیب یک خط از متن را چاپ کرده ایم، پس یک واحد به تعداد خطها اضافه می کنیم:

```
intLineCount = intLineCount + 1;
```

حال بررسی می کنیم که صفحه پر شده است یا نه؟ اگر صفحه پر نشده بود، خط دیگری را از فایل خوانده و در متغیر `strLine` قرار می دهیم، تا حلقه با خط جدید ادامه پیدا کند:

```
if (intLineCount < sngLinesPerpage)
{
strLine = objStreamToPrint.ReadLine();
```

بعد از اینکه یک صفحه کاملاً پر شد، برنامه از حلقه خارج می‌شود. حال باید مشخص کنیم، که صفحه دیگری هم باید چاپ شود و یا اینکه متن داخل فایل تمام شده است. اگر متن داخل فایل تمام شده بود، باید خصوصیت HasMorePages را برابر با false قرار دهیم، که متد Print بار دیگر باعث فراخوانی شدن رویداد PrintPage نشود. اما اگر متن تمام نشده بود، کافی است که برای چاپ ادامه‌ی متن خصوصیت HasMorePages را برابر با true قرار دهیم. به این ترتیب، متد print بار دیگر رویداد PrintPage را فراخوانی می‌کند تا متد prtPage بتواند صفحه بعد را چاپ کند.

```
if (strLine != null)
{
    e.HasMorePages = true;
}
else
{
    e.HasMorePages = false;
}
```

هنگامی که تمام متن داخل فایل به چاپگر فرستاده شد، وظیفه متد Print تمام شده است و برنامه به ادامه کدهای موجود در متد btnPrint-Click بر می‌گردد. تنها کاری که باید در ادامه انجام دهیم این است که فضای اشغال شده به وسیله اشیای مربوط به چاپ و نیز اشیای مربوط به خواندن از فایل را آزاد کنیم.

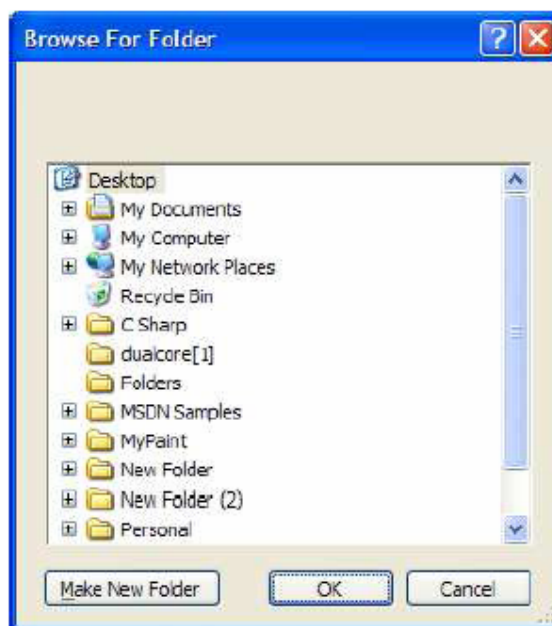
```
objStreamToPrint.Close();
objStreamToPrint = null;
```

۱۶-۷- کنترل FolderBrowserDialog

شاید در برنامه نیاز داشته باشید به کاربر اجازه دهید که به جای انتخاب یک فایل یک پوشه را مشخص کند. برای مثال، ممکن است بخواهید کاربر پوشه‌ای را برای ذخیره فایل‌های پشتیبان و یا پوشه‌ای را برای ذخیره فایل‌های موقتی برنامه مشخص کند. در این مواقع می‌توانید با استفاده از کنترل FolderBrowserDialog، کادر استاندارد Browse For Folder را در برنامه نمایش دهید. همانطور که ممکن است در دیگر برنامه‌های ویندوز نیز مشاهده کرده باشید، این کادر فقط پوشه‌های موجود در کامپیوتر را نمایش می‌دهد و به واسطه‌ی آن، کاربر می‌تواند پوشه‌ای را در برنامه مشخص کند.

همانند تمام کادرهای دیگر، کادر FolderBrowser می‌تواند به صورت کنترل مورد استفاده قرار گیرد و هم به صورت یک کلاس. شکل ۱۶-۱۶ یک کادر FolderBrowser را بدون تنظیم خصوصیت‌های آن (با مقادیر پیش فرض خصوصیت‌ها) نمایش می‌دهد. توجه کنید که در قسمت پایین این فرم یک دکمه فرمان Make New Folder وجود دارد که به کاربر اجازه می‌دهد پوشه جدیدی را ایجاد کند.

شکل ۱۶-۱۶



خصوصیت‌های کنترل FolderBrowser

قبل از اینکه نحوه استفاده از این کنترل را در کد مشاهده کنیم، بهتر است به بررسی خصوصیت‌های مهم آن بپردازیم. در جدول ۱۶-۱۲ لیستی از نام و نحوه استفاده از خصوصیت‌های مهم این کنترل آورده شده است.

جدول ۱۶-۱۲

نام خصوصیت	شرح
Description	مشخص کننده متنی است که به عنوان توضیح در کادر نمایش داده می‌شود.
RootFolder	مشخص کننده آدرس پوشه‌ای است که به صورت پیش فرض باید در کادر نمایش داده شود.
SelectedPath	مشخص کننده آدرس مسیری که به وسیله کاربر انتخاب شده است.
ShowNewFolderButton	مشخص می‌کند آیا دکمه‌ی Make New Folder در کادر نمایش داده شود یا نه؟

کادر محاوره‌ای Folder Browser اولین کادری است که تقریباً از تمام خصوصیت‌های آن استفاده خواهیم کرد. همانند تمام کادرهای دیگر، این کنترل نیز دارای متدی به نام ShowDialog است، که باعث نمایش داده شدن کادر در برنامه می‌شود. نحوه استفاده از این متد در این کنترل همانند کادرهای دیگر است، بنابراین نیازی به توضیح مجدد آن نیست.

استفاده از کنترل FolderBrowser

همانند تمام کادرهای محاوره‌ای دیگر، قبل از نمایش کادر Browse For Folder، باید بعضی از خصوصیت‌های آن را تغییر دهیم. سه خصوصیتی که عموماً قبل از نمایش این کادر تنظیم می‌شوند، در قطعه کد زیر نشان داده شده‌اند. اولین خصوصیت Description است که یک توضیح و یا دستورالعمل را برای کاربر در صفحه نمایش می‌دهد. متنی که در این خصوصیت قرار داده شود، هنگام فراخوانی تابع ShowDialog در بالای کادر نوشته خواهد شد.

خصوصیت بعدی خصوصیت RootFolder است. این خصوصیت مشخص می‌کند که هنگام نمایش کادر، چه پوشه‌ای به صورت پیش فرض نمایش داده شود. این خصوصیت مقداری را از نوع شمارشی Environment.SpecialFolder دریافت

می‌کند و این نوع شمارشی نیز خود حاوی آدرس پوشه‌های مخصوص سیستم‌عامل ویندوز مانند پوشه My Documents است. خصوصیت دیگری که قبل از نمایش کادر تنظیم می‌شود، خصوصیت ShowNewFolderButton است. اگر مقدار این خصوصیت برابر با true باشد، دکمه‌ی Make New Folder در کادر نمایش داده می‌شود تا به کاربر اجازه داده شود پوشه‌ی جدیدی را ایجاد کند، در غیر این صورت، این دکمه نمایش داده نخواهد شد.

```
folderBrowserDialog\Description =
"Select a folder for your backups:";
folderBrowserDialog\RootFolder =
Environment.SpecialFolder.MyComputer;
folderBrowserDialog\ShowNewFolderButton = false;
```

بعد از تنظیم خصوصیت‌های لازم، می‌توانید با فراخوانی تابع ShowDialog کادر Browse For Folder را نمایش دهید:

```
folderBrowserDialog.ShowDialog();
```

این تابع نیز همانند کادرهای قبلی مقداری را از نوع DialogResult بر می‌گرداند. می‌توانید با استفاده از یک دستور if به بررسی نتیجه آن بپردازید. برای دسترسی به آدرس پوشه‌ای که کاربر انتخاب کرده است، می‌توانید از مقدار خصوصیت SelectedPath استفاده کرده و آن را در متغیری ذخیره کنید. این خصوصیت آدرس پوشه انتخاب شده توسط کاربر را بر می‌گرداند. برای مثال، اگر کاربر پوشه temp را درون درایو c انتخاب کند، مقدار این خصوصیت به صورت c:\temp خواهد بود.

```
strFileName = folderBrowserDialog.SelectedPath;
```

در مثال بعدی، مجدداً از پروژه Dialogs استفاده کرده و کادر Browse For Folder را نمایش می‌دهیم. اگر کاربر پوشه‌ای را در این کادر انتخاب کرد، آدرس آن را در TextBox درون فرم نمایش خواهیم داد.

مثال ۱۶-۷ کار با کنترل FolderBrowser

(۱) به قسمت طراحی فرم در پروژه Dialogs بروید.

(۲) با استفاده از جعبه ابزار، کنترل Button دیگری را به فرم برنامه اضافه کرده و خصوصیت‌های آن را بر طبق لیست زیر تنظیم کنید:

- خصوصیت Name را برابر با btnBrowse قرار دهید.
- خصوصیت Text را برابر با Browse قرار دهید.
- خصوصیت Location را برابر با ۳۶۷-۱۵۸ قرار دهید.
- خصوصیت Anchor را برابر با Top'Right قرار دهید.

(۳) حال باید یک کنترل FolderBrowserDialog را به برنامه اضافه کنید. برای این کار، در جعبه ابزار به قسمت Dialogs بروید و بر روی کنترل FolderBrowserDialog دو بار کلیک کنید. مشاهده خواهید کرد که این کنترل نیز همانند کنترل‌های قبلی به قسمت پایین طراحی فرم اضافه خواهد شد.

(۴) بر روی دکمه‌ی btnBrowse دو بار کلیک کنید تا متد مربوط به رویداد Click آن ایجاد شود. سپس کد زیر را در آن متد وارد کنید:

```
private void btnBrowse_Click(object sender, EventArgs e)
{
// Set the FolderBrowserDialog control properties
folderBrowserDialog\Description =
"Select a folder for your backups:";
folderBrowserDialog\RootFolder =
Environment.SpecialFolder.MyComputer;
folderBrowserDialog\ShowNewFolderButton = false;
// Show the Browse For Folder dialog
```



```
if (folderBrowserDialog.ShowDialog() ==
DialogResult.OK)
{
// Display the selected folder
txtFile.Text = folderBrowserDialog.SelectedPath;
}
}
```

۵) تمام کد مورد نیاز برای این برنامه همین بود. برای امتحان عملکرد برنامه، درنوار ابزار روی دکمه Start کلیک کنید.

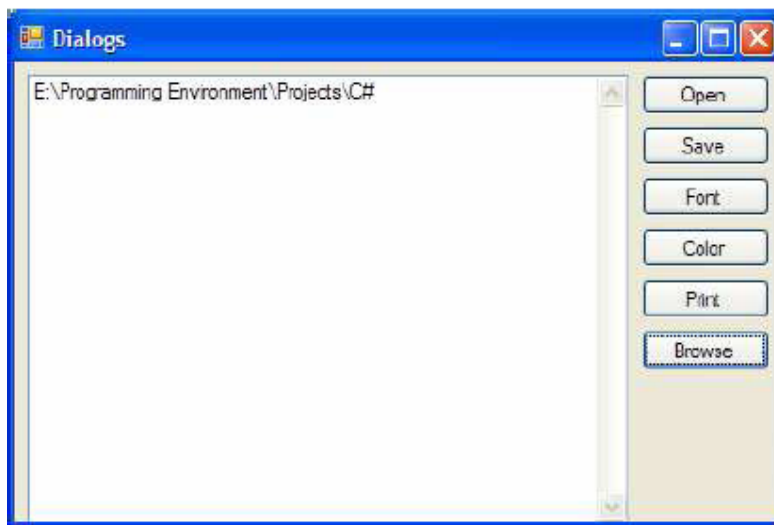
۶) هنگامی که فرم برنامه نمایش داده شد، روی دکمه‌ی Browse کلیک کنید. کادر Browse For Folder همانند شکل ۱۶-۱۷ نمایش داده خواهد شد.

شکل ۱۶-۱۷



۷) پوشه‌ای را در کامپیوتر خود مشخص کرده و روی دکمه فرمان Ok کلیک کنید. مشاهده خواهید کرد که آدرس کامل پوشه مشخص شده، همانند شکل ۱۶-۱۸ در فرم نمایش داده خواهد شد.

شکل ۱۶-۱۸



۱۶-۸- خلاصه

- در این فصل بعضی از کادرها را که در برنامه‌های C# ۲۰۰۵ قابل استفاده است را بررسی کردیم.

- این کادرها عبارتند از `MessageBox`، `OpenFileDialog`، `SaveFileDialog`، `FontDialog`، `FolderBrowserDialog`، `PrintDialog`، `ColorDialog`.
- همانطور که مشاهده کردید، این کادرها رابط‌های کاربری استاندارد را برای برنامه فراهم می‌کنند و به واسطه آنها می‌توانید برنامه‌ای با ظاهر حرفه‌ای‌تر و مشابه دیگر برنامه‌های ویندوز طراحی کنید.
- اگرچه برای استفاده از این کادرها از کنترل‌های متناظر آنها در جعبه ابزار استفاده کردید، اما به خاطر داشته باشید که تمام این کادرها می‌توانند همانند یک کلاس مورد استفاده قرار بگیرند. به عبارت دیگر کلاس متناظر با این کنترل‌ها نیز همین خصوصیت‌ها و متدها را ارائه می‌دهند و تفاوتی ندارد که در برنامه از آنها به عنوان کلاس استفاده کنید و یا به عنوان کنترل.
- برای استفاده از این کادرها به صورت کنترل، می‌توانید متغیری را از نوع کلاس مرتبط با کادر مورد نظرتان تعریف کنید و در هر قسمتی از برنامه که خواستید از آن کادر استفاده کنید با استفاده از دستور `new` کادر را ایجاد کنید.
- بعد از استفاده هم می‌توانید متغیر را از بین ببرید تا حافظه گرفته شده به وسیله آن آزاد شود. به این ترتیب حافظه کمتری در برنامه استفاده خواهید کرد و برنامه کارایی بیشتری خواهد داشت.

واسط‌ها

آنچه که در این فصل یاد خواهید گرفت:

- مفهوم واسط و تاثیر آن در کدنویسی
- نحوه‌ی تعریف واسط و پیاده‌سازی آن در کلاس‌ها
- استفاده از واسط‌ها برای ارث‌بری چندگانه
- توسعه و ترکیب واسط‌های موجود
- استفاده از واسط برای تشخیص قابلیت‌های یک کلاس
- کاربرد عملگرهای `is` و `as`

یک واسط^۱، قراردادی است که برای یک سرویس‌گیرنده نحوه‌ی رفتار یک کلاس یا ساختار را تضمین خواهد کرد. زمانی که یک کلاس، واسطی را پیاده‌سازی می‌کند، آن می‌گوید من تضمین می‌کنم که متدها، خصوصیات، رویدادها و اندیس‌گذارهای آن واسط را پشتیبانی خواهیم کرد.

واسط برای یک کلاس انتزاعی جهت ایجاد قراردادهایی مابین کلاس‌ها و سرویس‌گیرنده‌هایش یک چاره پیشنهاد می‌کند. این قراردادهای با استفاده از کلمه کلیدی `interface` اظهارنامه را ایجاد می‌کنند، که یک نوع داده‌ی ارجاعی برای کپسوله کردن قرارداد اعلان می‌کنند.

زمانی که یک واسط را تعریف می‌کنید، ممکن است متدها، خصوصیات، اندیس‌گذارها یا رویدادهایی تعریف کنید که بوسیله‌ی کلاس پیاده‌سازی کننده‌ی واسط، پیاده‌سازی خواهند شد. واسط‌ها اغلب با کلاس‌های انتزاعی مقایسه می‌شوند. یک کلاس انتزاعی به عنوان یک کلاس پایه برای یک خانواده از کلاس‌های مشتق شده بکار گرفته می‌شود. در حالیکه واسط‌ها برای ترکیب شدن با درخت‌های وراثت درگیر هستند.

زمانی که یک کلاس، واسطی را پیاده‌سازی می‌کند، آن باید تمامی بخش‌های واسط را پیاده‌سازی کند. به عبارت دیگر، کلاس با قرار داد کامل تعریف شده بوسیله این واسط موافق است.

نکته: برخلاف جاوا، `#C` کاربرد فیلدهای ثابت در واسط‌ها را پشتیبانی نمی‌کند، ولی می‌توان از ثابت‌های شمارشی استفاده کرد.

^۱ Interface

در فصل‌های قبلی دیدیم که وراثت از یک کلاس انتزاعی، رابطه‌ی is-a را پیاده‌سازی می‌کند. پیاده‌سازی یک واسط، یک رابطه‌ی متفاوتی تعریف می‌کند که ما تا به حال ندیده‌ایم و رابطه‌ی پیاده‌سازی نامیده می‌شود. این دو رابطه کاملاً متفاوت هستند. ماشین یک وسیله نقلیه است، اما آن ممکن است قابلیت CanBeBoughtWithABigLoan را پیاده‌سازی کند.

در این فصل نحوه‌ی ایجاد، پیاده‌سازی و کاربرد واسط‌ها را یاد خواهید گرفت. نحوه‌ی پیاده‌سازی چندین واسط و نحوه‌ی ترکیب و توسعه واسط‌ها را یاد می‌گیرید.

۱۷-۱-تعریف و پیاده‌سازی یک واسط

گرامر تعریف یک واسط بصورت زیر است.

```
[attributes] [access-modifier] interface interface-name[:base-  
list]{interface-body}
```

در حال حاضر درباره صفات فکر نکنید. در فصل‌های آتی بررسی خواهند شد. معرف‌های دسترسی شامل internal, protected, private, public, و protected internal هستند. کلمه‌ی کلیدی interface با نام واسط دنبال می‌شود. معمولاً، نام واسط با حرف بزرگ I شروع می‌شود.

Base-list، واسط‌هایی که این واسط بسط می‌دهد را لیست می‌کند.

Interface-body: متدها، خصوصیات و آنچه که باید توسط کلاس پیاده‌سازی کننده، پیاده‌سازی شود را شرح می‌دهد.

فرض کنید می‌خواهید یک واسط ایجاد کنید که متدها و خصوصیات مورد نیاز جهت ذخیره و بازیابی اطلاعات از یک پایگاه داده یا هر ساختار دیگر همچون فایل را توصیف کند و تصمیم دارید آن را IStorable بنامید.

احتمالاً در این واسط دو متد Read(), Write() را مشخص می‌کنید که در Interface-body ظاهر می‌گردد.

```
interface IStorable  
{  
void Read();  
void Write(object);  
}
```

هدف یک واسط، تعریف توانایی‌هایی است که می‌خواهد در یک کلاس موجود باشد.

مثال: ممکن است بخواهید یک کلاس بنام Document ایجاد کنید و نوع داده‌ی Document بتواند در یک پایگاه داده ذخیره شود. سپس تصمیم دارید واسط IStorable را در Document پیاده‌سازی کنید.

برای انجام این کار، همان گرامر ارث‌بری را بکار برید که کلاس جدید Document از واسط IStorable ارث‌بری می‌کند.

```
public class Document : IStorable  
{  
public void Read() {...}  
public void Write(object obj) {...}  
// ...  
}
```

حال شما به عنوان نویسنده کلاس Document مسئول هستید یک پیاده‌سازی معنی‌دار از متدهای IStorable فراهم کنید. با معین کردن Document به عنوان پیاده‌سازی کننده‌ی IStorable، شما باید همه‌ی متدهای IStorable را پیاده‌سازی کنید وگرنه در زمان کامپایل با خطا مواجه خواهید شد. در مثال ۱۷-۱، کلاس Document واسط IStorable را پیاده‌سازی می‌کند.

مثال ۱۷-۱

```
#region Using directives  
using System;  
using System.Collections.Generic;
```

```

using System.Text;
#endregion
namespace SimpleInterface
{
    // declare the interface
    interface IStorable
    {
        // no access modifiers, methods are public
        // no implementation
        void Read( );
        void Write( object obj );
        int Status { get; set; }
    }
    // create a class which implements the IStorable interface
    public class Document : IStorable
    {
        // store the value for the property
        private int status = ۰;
        public Document( string s )
        {
            Console.WriteLine( "Creating document with: {۰}", s );
        }
        // implement the Read method
        public void Read( )
        {
            Console.WriteLine(
                "Implementing the Read Method for IStorable" );
        }
        // implement the Write method
        public void Write( object o )
        {
            Console.WriteLine(
                "Implementing the Write Method for IStorable" );
        }
        // implement the property
        public int Status
        {
            get
            {
                return status;
            }
            set
            {
                status = value;
            }
        }
    }
    // Take our interface out for a spin
    public class Tester
    {
        static void Main( )
        {
            // access the methods in the Document object
            Document doc = new Document( "Test Document" );
            doc.Status = -۱;
            doc.Read( );
            Console.WriteLine( "Document Status: {۰}", doc.Status );
        }
    }
}

```

Output:

```

Creating document with: Test Document
Implementing the Read Method for IStorable
Document Status: -۱

```

مثال ۱۷-۱ یک واسط ساده بنام `IStorable` با دو متد `Read()` و `Write()` و یک خصوصیت از نوع `integer` بنام `Status` را تعریف می‌کند. توجه داشته باشید که اعلان خصوصیت، یک پیاده‌سازی برای `get()` و `set()` فراهم نمی‌کند، اما بطور ساده مشخص می‌کند که این متدها وجود دارند.

```
{; int Status {get ; set
```

توجه داشته باشید که معرف‌های دسترسی در اعلان متد وجود ندارند. در حقیقت، تعیین یک معرف دسترسی یک خطای کامپایل تولید می‌کند. متدهای واسط مطلقاً `public` هستند، چون واسط قراردادی است که بوسیله‌ی کلاس‌های دیگر استفاده می‌شود. نمی‌توانید یک نمونه از یک واسط ایجاد کنید. به جای آن از کلاسی که آن را پیاده‌سازی می‌کند، نمونه‌ای ایجاد کنید.

کلاسی که واسط را پیاده‌سازی می‌کند، باید قرارداد را بطور کامل و دقیق پیاده‌سازی کند. `Document` باید هر دو متد `Read()` و `Write()` و خصوصیت `Status` را فراهم کند. اینکه کلاس `Document` چگونه این نیازمندی‌ها را تکمیل می‌کند کاملاً به خود کلاس مرتبط است. اگرچه واسط `IStorable`، خصوصیت `Status` را دارد. اما آن نمی‌داند که کلاس `Document` چگونه آن را ذخیره می‌کند (به صورت یک متغیر عضو یا یک فیلد در پایگاه داده). جزئیات بر روی پیاده‌سازی کلاس است.

۱۷-۱-۱ پیاده‌سازی بیش از یک واسط

کلاس‌ها می‌توانند بیش از یک واسط را پیاده‌سازی کنند. برای مثال، اگر کلاس `Document` می‌تواند ذخیره شود و همچنین می‌تواند فشرده‌سازی شود، ممکن است دو واسط `IStorable` و `ICompressible` را برای پیاده‌سازی انتخاب کنید. برای انجام این کار، اعلان خود را طوری تغییر دهید که هر دو واسط را پیاده‌سازی می‌کند. اسامی واسط‌ها را با کاما از هم جدا کنید.

```
public class Document : IStorable, ICompressible
```

در این صورت کلاس `Document` باید متدهای مشخص شده بوسیله‌ی واسط `ICompressible` را پیاده‌سازی کند.

```
public void Compress()
{
    Console.WriteLine("Implementing the Compress Method");
}
public void Decompress( )
{
    Console.WriteLine("Implementing the Decompress Method");
}
```

۱۷-۱-۲ بسط‌دادن واسط‌ها

بسط دادن یک واسط موجود برای اضافه کردن متدها یا اعضای جدید یا تغییر نحوه‌ی کار اعضای موجود امکان‌پذیر است. برای مثال، ممکن است `ICompressible` را با واسط جدید `ILoggedCompressible` بسط دهید که واسط اصلی را با متدهایی جهت پی‌گیری بایت‌های ذخیره شده بسط می‌دهد.

```
interface ILoggedCompressible : ICompressible
{
    void LogSavedBytes();
}
```

کلاس‌ها متناسب با نیاز خود برای پیاده‌سازی یکی از دو واسط `ICompressible` یا `ILoggedCompressible` آزاد هستند. اگر کلاسی واسط `ILoggedCompressible` را پیاده‌سازی کند، آن باید تمامی متدهای هر دو واسط را پیاده‌سازی

کند. اما اشیاء این نوع داده می‌توانند به یکی از واسط‌های ILoggedCompressible یا ICompressible قالب‌بندی شوند.

۱۷-۱-۳- ترکیب واسط‌ها

بطور مشابه، می‌توانید واسط‌های جدیدی با ترکیب واسط‌های موجود و اضافه کردن خصوصیات و متدهای جدید ایجاد کنید. برای مثال، ممکن است ایجاد IStorableCompressible را تصمیم بگیرید. این واسط متدهای هر دو واسط را همراه یک متد جدید برای ذخیره‌کردن اندازه‌ی اصلی عنصر فشرده نشده ترکیب خواهد کرد.

```
interface IStorableCompressible : IStorable, ILoggedCompressible
{
    void LogOriginalSize();
}
```

مثال ۱۷-۲: بسط‌دادن و ترکیب واسط‌ها را نشان می‌دهد.

مثال ۱۷-۲

```
#region Using directives
using System;
using System.Collections.Generic;
using System.Text;
#endregion
namespace ExtendAndCombineInterface
{
    interface IStorable
    {
        void Read( );
        void Write( object obj );
        int Status { get; set; }
    }
    // here's the new interface
    interface ICompressible
    {
        void Compress( );
        void Decompress( );
    }
    // Extend the interface
    interface ILoggedCompressible : ICompressible
    {
        void LogSavedBytes( );
    }
    // Combine Interfaces
    interface IStorableCompressible : IStorable, ILoggedCompressible
    {
        void LogOriginalSize( );
    }
    // yet another interface
    interface IEncryptable
    {
        void Encrypt( );
        void Decrypt( );
    }
    public class Document : IStorableCompressible, IEncryptable
    {
        // hold the data for IStorable's Status property
        private int status = 0;
        // the document constructor
        public Document( string s )
        {
            Console.WriteLine( "Creating document with: {0}", s );
        }
        // implement IStorable
```

```

public void Read( )
{
    Console.WriteLine(
        "Implementing the Read Method for IStorable" );
}
public void Write( object o )
{
    Console.WriteLine(
        "Implementing the Write Method for IStorable" );
}
public int Status
{
    get
    {
        return status;
    }
    set
    {
        status = value;
    }
}
// implement ICompressible
public void Compress( )
{
    Console.WriteLine( "Implementing Compress" );
}
public void Decompress( )
{
    Console.WriteLine( "Implementing Decompress" );
}
// implement ILoggedCompressible
public void LogSavedBytes( )
{
    Console.WriteLine( "Implementing LogSavedBytes" );
}
// implement IStorableCompressible
public void LogOriginalSize( )
{
    Console.WriteLine( "Implementing LogOriginalSize" );
}
// implement IEncryptable
public void Encrypt( )
{
    Console.WriteLine( "Implementing Encrypt" );
}
public void Decrypt( )
{
    Console.WriteLine( "Implementing Decrypt" );
}
}
public class Tester
{
    static void Main( )
    {
        // create a document object
        Document doc = new Document( "Test Document" );
        // cast the document to the various interfaces
        IStorable isDoc = doc as IStorable;
        if ( isDoc != null )
        {
            isDoc.Read( );
        }
        else
        {
            Console.WriteLine( "IStorable not supported" );
            ICompressible icDoc = doc as ICompressible;

```

```

if ( icDoc != null )
{
    icDoc.Compress ( );
}
else
    Console.WriteLine( "Compressible not supported" );
ILoggedCompressible ilcDoc = doc as ILoggedCompressible;
if ( ilcDoc != null )
{
    ilcDoc.LogSavedBytes( );
    ilcDoc.Compress( );
    // ilcDoc.Read( );
}
else
    Console.WriteLine( "LoggedCompressible not supported" );
IStorableCompressible isc = doc as IStorableCompressible;
if ( isc != null )
{
    isc.LogOriginalSize( ); // IStorableCompressible
    isc.LogSavedBytes( ); // ILoggedCompressible
    isc.Compress( ); // ICompressible
    isc.Read( ); // IStorable
}
else
{
    Console.WriteLine( "StorableCompressible not supported" );
}
IEncryptable ie = doc as IEncryptable;
if ( ie != null )
{
    ie.Encrypt( );
}
else
    Console.WriteLine( "Encryptable not supported" );
}
}
}
Output:
Creating document with: Test Document
Implementing the Read Method for IStorable
Implementing Compress
Implementing LogSavedBytes
Implementing Compress
Implementing LogOriginalSize
Implementing LogSavedBytes
Implementing Compress
Implementing the Read Method for IStorable
Implementing Encrypt

```

مثال ۱۷-۲ با پیاده‌سازی واسط IStorable و Icompressible آغاز می‌شود. سپس واسط دومی به واسط ILoggedCompressible بسط داده می‌شود و سپس هر دو در IStorablecompressible ترکیب می‌شوند و در نهایت، یک واسط جدید بنام IEncryptable اضافه می‌کند.

برنامه‌ی Tester شی Document جدید ایجاد می‌کند و سپس آن را بصورت یک نمونه از واسط‌های متنوع به کار می‌برد. شما برای قالب‌بندی آزاد هستید.

```
ICompressible icDoc = doc as ICompressible
```

البته این عمل غیر ضروری است، چون کامپایلر می‌داند که doc واسط Icompressible را پیاده‌سازی می‌کند. پس می‌توان از قالب‌بندی ضمنی استفاده کرد:

```
ICompressible icDoc = doc;
```


از طرف دیگر، اگر یقین ندارید که آیا کلاس‌تان یک واسط معینی را پیاده‌سازی می‌کند، می‌توانید با استفاده از عملگر `as` قالب‌بندی کنید و سپس تست کنید آیا شی قالب‌بندی شده `null` است یا نه.

```
ICompressible icDoc = doc as ICompressible;
if ( icDoc != null )
{
    icDoc.Compress( );
}
else
    Console.WriteLine( "Compressible not supported" );
```

در هنگام قالب‌بندی می‌توانید به یک واسط بسط‌یافته نیز قالب‌بندی کنید.

```
;ILoggedCompressible icDoc=doc as ILoggedCompressible
```

۱۷-۲-دستیابی به متدهای واسط

می‌توانید به اعضای واسط `IStorable` دستیابی کنید، همانطور که اگر آنها اعضای کلاس `Document` باشند:

```
Document doc = new Document("Test Document");
doc.status = -1;
doc.Read();
```

می‌توانید یک نمونه از واسط را بوسیله قالب‌بندی `Document` به نوع داده‌ی واسط ایجاد کنید و سپس آن واسط را برای دستیابی به متدها بکار ببرید.

```
IStorable isDoc = doc;
isDoc.status = 0;
isDoc.Read();
```

در این حالت، در واقع می‌دانید که `Document` یک `IStorable` است. همانطور که قبلاً بیان کردیم، نمی‌توانید مستقیماً یک نمونه از یک واسط ایجاد کنید. بدین دلیل، نمی‌توانید بنویسید:

```
IStorable isDoc = new IStorable();
```

با این وجود، می‌توانید یک نمونه از کلاس پیاده‌سازی کننده را به صورت زیر ایجاد کنید:

```
Document doc = new Document("Test Document");
```

حال می‌توانید یک نمونه از واسط را با قالب‌بندی شی پیاده‌سازی کننده به نوع داده‌ی واسط ایجاد کنید.

```
IStorable isDoc = doc;
```

می‌توانید این دو مرحله را بصورت زیر ترکیب کنید.

```
IStorable isDoc = new Document("Test Document");
```

دستیابی به یک واسط، رفتار چندریختی با واسط را مجاز می‌دارد. به عبارت دیگر، می‌توانید دو یا چند کلاس را با یک واسط پیاده‌سازی کنید و سپس به این کلاس فقط از طریق واسط دستیابی کنید. می‌توانید در زمان اجرا از نوع داده‌ی واقعی آن چشم‌پوشی کرده و به جای همدیگر بکار ببرید.

۱۷-۲-۱-قالب‌بندی به یک واسط

در بیشتر موارد نمی‌دانید، آیا یک شی، واسط خاصی را پشتیبانی می‌کند. یک کلکسیون از اشیاء داده شده است و ممکن است ندانید که آیا یک شی خاص، واسط `IStorable` یا `ICompressible` یا هر دو را پشتیبانی می‌کند؟ فقط می‌توانید آنها را به واسط‌ها قالب‌بندی کنید.

```
Document doc = myCollection[0];
IStorable isDoc = (IStorable) doc;
isDoc.Read();
```

```
ICompressible icDoc = (ICompressible) doc;
icDoc.Compress( );
```

اگر آن بفهمد که Document فقط واسط IStorable را پیاده‌سازی می‌کند:

```
public class Document : IStorable
```

قالب‌بندی به ICompressible درست کامپایل می‌شود، چون ICompressible یک واسط معتبر است. با این وجود، به دلیل قالب‌بندی نادرست، زمانی که برنامه اجرا می‌شود یک استثناء رها می‌شود.

```
An exception of type System.InvalidCastException was thrown.
```

۱۷-۲-۲-عملگر is

برای احضار متدهای مناسب، دوست داریم قادر باشیم واسط پشتیبانی شده توسط کلاس را بشناسیم. C# دو روش برای انجام این کار دارد. روش اول، کاربرد عملگر is است. گرامر عملگر is بصورت زیر است.

```
expression is type
```

در صورتی که قالب‌بندی به type بدون رها کردن استثناء انجام شود، عملگر is به مقدار true ارزیابی می‌شود. مثال ۱۷-۳ کاربرد عملگر is را برای تست پیاده‌سازی واسط‌های IStorable و ICompressible توسط کلاس Document ارائه می‌کند.

مثال ۱۷-۳

```
#region Using directives
using System;
using System.Collections.Generic;
using System.Text;
#endregion
namespace IsOperator
{
    interface IStorable
    {
        void Read( );
        void Write( object obj );
        int Status { get; set; }
    }
    // here's the new interface
    interface ICompressible
    {
        void Compress( );
        void Decompress( );
    }
    // Document implements IStorable
    public class Document : IStorable
    {
        private int status = 0;
        public Document( string s )
        {
            Console.WriteLine(
                "Creating document with: {0}", s );
        }
        // IStorable.Read
        public void Read( )
        {
            Console.WriteLine( "Reading..." );
        }
        // IStorable.Write
        public void Write( object o )
        {
            Console.WriteLine( "Writing..." );
        }
    }
}
```

```
// IStorable.Status
public int Status
{
    get
    {
        return status;
    }
    set
    {
        status = value;
    }
}
// derives from Document and implements ICompressible
public class CompressibleDocument : Document, ICompressible
{
    public CompressibleDocument(String s) :
    base(s)
    { }
    public void Compress( )
    {
        Console.WriteLine("Compressing...");
    }
    public void Decompress( )
    {
        Console.WriteLine("Decompressing...");
    }
}
public class Tester
{
    static void Main( )
    {
        // A collection of Documents
        Document[] docArray = new Document[۲];
        // First entry is a Document
        docArray[۰] = new Document( "Test Document" );
        // Second entry is a CompressibleDocument (ok because
        // CompressibleDocument is a Document)
        docArray[۱] =
        new CompressibleDocument("Test compressibleDocument");
        // don't know what we'll pull out of this hat
        foreach (Document doc in docArray)
        {
            // report your name
            Console.WriteLine("Got: {۰}", doc);
            // Both pass this test
            if (doc is IStorable)
            {
                IStorable isDoc = (IStorable)doc;
                isDoc.Read( );
            }
            // fails for Document
            // passes for CompressibleDocument
            if (doc is ICompressible)
            {
                ICompressible icDoc = (ICompressible)doc;
                icDoc.Compress( );
            }
        }
    }
}
Output:
Creating document with: Test Document
Creating document with: Test compressibleDocument
```

```
Got: IsOperator.Document
Reading...
Got: IsOperator.CompressibleDocument
Reading...
Compressing...
```

متد Main() با ارزیابی کردن عبارت if زیر، قانونی بودن هر قالب‌بندی را بررسی می‌کند.

```
if (doc is IStorable)
```

این عبارت واضح و روشن است. فقط اگر شی از نوع واسط قانونی باشد، دستور if می‌گوید قالب‌بندی اتفاق افتاده است. کلاس Document این تست را رد می‌کند، ولی بعدی شکست می‌خورد.

```
if (doc is ICompressible)
```

اما کلاس CompressibleDocument هر دو تست را رد می‌کند.

ما هر دو نوع سند را در یک آرایه گذاشتیم. قبل از تلاش برای فراخوانی متدهای ICompressible، باید مطمئن شوید که نوع داده‌ی Document، واسط ICompressible را پیاده‌سازی می‌کند. عملگر is این تست را برای شما انجام می‌دهد.

۱۷-۲-۳-عملگر as

عملگر as عملیات is و قالب‌بندی را باهم ترکیب می‌کند. ابتدا بررسی می‌کند آیا عمل قالب‌بندی معتبر است، اگر معتبر باشد، عمل قالب‌بندی را انجام می‌دهد. اگر عمل قالب‌بندی معتبر نباشد، عملگر as مقدار null برمی‌گرداند.

کاربرد عملگر as، نیاز به اداره کردن استثنای قالب‌بندی را حذف می‌کند. و از سربار مربوط به دو عمل قالب‌بندی دوری می‌کنیم. بدین دلایل، قالب‌بندی واسط‌ها با استفاده از as بهینه است.

گرامر عملگر as بصورت زیر است.

```
expression as type
```

کد زیر، کد مثال ۱۷-۳ را با استفاده از عملگر as و تست مقدار null وفق می‌دهد.

```
static void Main()
{
    // A collection of Documents
    Document[] docArray = new Document[۲];
    // First entry is a Document
    docArray[۰] = new Document( "Test Document" );
    // Second entry is a CompressibleDocument (ok because
    // CompressibleDocument is a Document)
    docArray[۱] = new CompressibleDocument("Test compressibleDocument");
    // don't know what we'll pull out of this hat
    foreach (Document doc in docArray)
    {
        // report your name
        Console.WriteLine("Got: {۰}", doc);
        // Both pass this test
        IStorable isDoc = doc as IStorable;
        if (isDoc != null)
        {
            isDoc.Read( );
        }
        // fails for Document
        // passes for CompressibleDocument
        ICompressible icDoc = doc as ICompressible;
        if (icDoc != null)
        {
            icDoc.Compress( );
        }
    }
}
```

}

۱۷-۲-۴-مقایسه عملگرهای as و is

اگر الگوی طراحی شما برای تست یک شی این باشد که آیا شی از همان نوع داده‌ی مورد نیاز است و عمل قالب‌بندی را نیز فوراً انجام دهید، عملگر as کاراتر است. در بعضی مواقع فقط می‌خواهید نوع داده را تست کنید و عمل قالب‌بندی لازم نیست، در این حالت عملگر is بهتر است.

۱۷-۲-۵-مقایسه کلاس انتزاعی و واسط

واسط‌ها بسیار شبیه کلاس‌های انتزاعی هستند. در حقیقت می‌توانید اعلان یک `IStorable` را تغییر دهید تا یک کلاس انتزاعی باشد.

```
abstract class Storable
{
    abstract public void Read();
    abstract public void Write( );
}
```

`Document` می‌تواند از `Storable` ارث‌بری کند و تفاوتی با کاربرد واسط‌ها ندارد. فرض کنید یک کلاس `List` از یک شرکت خریداری کرده‌اید که توانایی ترکیب با `Storable` را دارد. در ++C می‌توانید کلاس `StorableList` را ایجاد کنید که از هر دو کلاس `List` و `Storable` ارث‌بری کند، اما در C# امکان‌پذیر نیست، چون C# وراثت چندگانه را مجاز نمی‌دارد. با این وجود، C# پیاده‌سازی هر تعداد واسط و مشتق گرفتن یک کلاس پایه را مجاز می‌دارد. پس با ایجاد `Storable` به عنوان یک واسط، می‌توانید از کلاس `List` و واسط `IStorable` ارث‌بری کنید. مثال `StorableList` بصورت زیر خواهد شد.

```
public class StorableList : List, IStorable
{
    // List methods here ...
    public void Read( ) {...}
    public void Write(object obj) {...}
    // ...
}
```

۱۷-۳-override کردن پیاده‌سازی‌های واسط

کلاس پیاده‌سازی کننده می‌تواند هر تعداد از متدهای پیاده‌سازی شده‌ی واسط را به عنوان مجازی علامت‌گذاری کند. کلاس‌های مشتق شده، برای دست‌یافتن به چندریختی می‌توانند این متدها را `override` کنند. برای مثال، ممکن است یک کلاس `Document`، واسط `IStorable` را پیاده‌سازی کند و متدهای `Read()` و `Write()` را بصورت `virtual` علامت‌گذاری کنید. ممکن است کلاس `Document` متدهای `Read()` و `Write()` را برای نوشتن در یک فایل پیاده‌سازی کند و توسعه دهندگان بخواهند انواع داده جدید مشتق شده از `Document` را برای خواندن و نوشتن در پایگاه داده بکار ببرند.

مثال ۱۷-۴ پیچیدگی مثال ۱۷-۳ را از بین می‌برد و `override` کردن یک پیاده‌سازی واسط را ارائه می‌کند. متد `Read()` بصورت `virtual` علامت‌گذاری می‌شود و بوسیله‌ی `Document.Read()` پیاده‌سازی می‌شود و در کلاس `Note` مشتق شده از `Document`، `override` می‌شوند.

مثال ۱۷-۴

```
#region Using directives
using System;
using System.Collections.Generic;
using System.Text;
```

```

#endregion
namespace overridingInterface
{
    interface IStorable
    {
        void Read( );
        void Write( );
    }
    // Simplify Document to implement only IStorable
    public class Document : IStorable
    {
        // the document constructor
        public Document( string s )
        {
            Console.WriteLine(
                "Creating document with: {·}", s );
        }
        // Make read virtual
        public virtual void Read( )
        {
            Console.WriteLine(
                "Document Read Method for IStorable" );
        }
        // NB: Not virtual!
        public void Write( )
        {
            Console.WriteLine(
                "Document Write Method for IStorable" );
        }
    }
    // Derive from Document
    public class Note : Document
    {
        public Note( string s ):
            base(s)
        {
            Console.WriteLine(
                "Creating note with: {·}", s );
        }
        // override the Read method
        public override void Read( )
        {
            Console.WriteLine(
                "Overriding the Read method for Note!" );
        }
        // implement my own Write method
        public new void Write( )
        {
            Console.WriteLine(
                "Implementing the Write method for Note!" );
        }
    }
    public class Tester
    {
        static void Main( )
        {
            // create a document reference to a Note object
            Document theNote = new Note( "Test Note" );
            IStorable isNote = theNote as IStorable;
            if ( isNote != null )
            {
                isNote.Read( );
                isNote.Write( );
            }
            Console.WriteLine( "\n" );
        }
    }
}

```

```
// direct call to the methods
theNote.Read( );
theNote.Write( );
Console.WriteLine( "\n" );
// create a note object
Note note2 = new Note( "Second Test" );
IStorable isNote2 = note2 as IStorable;
if ( isNote2 != null )
{
    isNote2.Read( );
    isNote2.Write( );
}
Console.WriteLine( "\n" );
// directly call the methods
note2.Read( );
note2.Write( );
}
}
}
Output:
Creating document with: Test Note
Creating note with: Test Note
Overriding the Read method for Note!
Document Write Method for IStorable
Overriding the Read method for Note!
Document Write Method for IStorable
Creating document with: Second Test
Creating note with: Second Test
Overriding the Read method for Note!
Document Write Method for IStorable
Overriding the Read method for Note!
Implementing the Write method for Note!
```

در این مثال، کلاس Document یک واسط ساده شده IStorable را پیاده‌سازی می‌کند. طراح Document فقط متد Read() را بصورت virtual انتخاب کرده است. در دنیای واقعی، یا همه یا هیچکدام بصورت virtual علامت‌گذاری می‌شوند. ضروری نیست کلاس Note متد Read() را override کند و آزادی عمل دارد.

در کلاس Tester، متدهای Read و Write به ۴ روش فراخوانی می‌شوند:

- + از طریق ارجاع کلاس پایه به شی مشتق شده
- + از طریق یک واسط ایجاد شده از روی ارجاع کلاس پایه به شی مشتق شده
- + از طریق یک شی مشتق شده
- + از طریق یک واسط ایجاد شده از روی شی مشتق شده

برای بنا کردن دو فراخوانی اول، یک ارجاع Document ایجاد می‌شود و آدرس یک شی Note جدید ایجاد شده روی Heap به ارجاع Document انتساب داده می‌شود.

```
; ("Document theNote = new Note("Test Note
```

یک ارجاع واسط ایجاد می‌شود و عملگر as برای قالب‌بندی Document به ارجاع IStorable استفاده می‌شود:

```
IStorable isNote = theNote as IStorable;
```

حال می‌توانید متدهای Read() و Write() را از طریق واسط احضار کنید. البته می‌توان متدهای Read() و Write() را از طریق خود شی بطور مستقیم فراخوانی کرد. در هر دو حالت، متد Read() از Note و متد Write() از Document فراخوانی می‌شوند(به دلیل چند ریختی).

برای فهم بیشتر مطلب، یک شی از کلاس Note تعریف کنید و مستقیماً متدهای Read () و Write () آن را فراخوانی کنید. نتیجه را با قبلی مقایسه کنید.

```
Note note2 = new Note("Second Test");
```

۱۷-۴- پیاده‌سازی صریح واسط

در پیاده‌سازی‌هایی که تا بحال نشان داده شده‌اند، کلاس پیاده‌سازی کننده، یک متد عضو با نام و نشانه‌ای همانند واسط ایجاد می‌کند. ضرورتی ندارد بگوییم این متد، پیاده‌سازی متد واسط است. بلکه کامپایلر بصورت ضمنی آنرا درک می‌کند.

اگر یک کلاس دو واسط را پیاده‌سازی کند که متدهای هم نام دارند، چه اتفاقی می‌افتد. مثال ۱۷-۵ دو واسط IStorable و ITalk را ایجاد می‌کند. متد Read () در ITalk برای خواندن کتاب با صدای بلند است. متأسفانه، این متد با متد Read () در IStorable تداخل دارد.

چون هر دو واسط، متد Read () را دارند، باید کلاس پیاده‌سازی کننده (Document)، حداقل پیاده‌سازی صریح را برای یکی از این دو متد بکار برد. در پیاده‌سازی صریح، کلاس پیاده‌سازی کننده، صریحاً واسط متد را معین می‌کند.

```
void ITalk.Read()
```

این عمل مشکل تداخل را رفع می‌کند، اما یک سری اثرات جالب ایجاد می‌کند. اول اینکه، در مورد متدهای دیگر نیازی به پیاده‌سازی صریح نیست.

```
public void Talk( )
```

چون تداخلی وجود ندارد، می‌تواند بصورت معمول تعریف شود.

مهمتر اینکه، متد پیاده‌سازی صریح، نمی‌تواند یک معرف دسترسی داشته باشد.

```
void ITalk.Read( )
```

این متد بطور ضمنی public است.

در حقیقت متد اعلان شده از طریق پیاده‌سازی صریح نمی‌تواند با معرف‌های abstract، virtual، override و new اعلان شود. مهمتر اینکه، دسترسی به متد پیاده‌سازی شده صریح از طریق خود شی امکان‌پذیر نیست. اگر دستور زیر را بنویسید، کامپایلر فرض می‌کند هدف شما واسط پیاده‌سازی شده ضمنی برای IStorable است.

```
theDoc.Read( );
```

تنها روش دسترسی به واسط پیاده‌سازی شده صریح، از طریق قالب‌بندی به یک واسط است.

```
ITalk itDoc = theDoc;
```

```
itDoc.Read();
```

پیاده‌سازی صریح در مثال ۱۷-۵ نشان داده می‌شود.

مثال ۱۷-۵

```
#region Using directives
using System;
using System.Collections.Generic;
using System.Text;
#endregion
namespace ExplicitImplementation
{
    interface IStorable
    {
        void Read( );
        void Write( );
    }
}
```



```

}
interface ITalk
{
void Talk( );
void Read( );
}
// Modify Document to implement IStorable and ITalk
public class Document : IStorable, ITalk
{
// the document constructor
public Document( string s )
{
Console.WriteLine( "Creating document with: {0}", s );
}
// Make read virtual
public virtual void Read( )
{
Console.WriteLine( "Implementing IStorable.Read" );
}
public void Write( )
{
Console.WriteLine( "Implementing IStorable.Write" );
}
void ITalk.Read( )
{
Console.WriteLine( "Implementing ITalk.Read" );
}
public void Talk( )
{
Console.WriteLine( "Implementing ITalk.Talk" );
}
}
public class Tester
{
static void Main( )
{
// create a document object
Document theDoc = new Document( "Test Document" );
IStorable isDoc = theDoc;
isDoc.Read( );
ITalk itDoc = theDoc;
itDoc.Read( );
theDoc.Read( );
theDoc.Talk( );
}
}
}
Output:
Creating document with: Test Document
Implementing IStorable.Read
Implementing ITalk.Read
Implementing IStorable.Read
Implementing ITalk.Talk

```

۱۷-۴-۱-در اختیار قراردادن متدهای انتخابی از واسط

یک طراح کلاس می‌تواند از مزایای پیاده‌سازی صریح یک واسط استفاده کند. واسط برای سرویس‌گیرندگان کلاس پیاده‌سازی کننده فقط از طریق قالب‌بندی نمایان است. فرض کنید مفهوم شی `Document` بیان می‌کند که آن واسط `IStorable` را پیاده‌سازی می‌کند. اما نمی‌خواهد متدهای `Write`، `Read()` بخشی از واسط عمومی `Document` باشند. با استفاده از پیاده‌سازی صریح می‌توان به این امر رسید که این متدها فقط از طریق قالب‌بندی در دسترس خواهند بود. این عمل API های عمومی `Document` را حفظ می‌کند، در حالیکه آن هنوز هم `IStorable` را پیاده‌سازی می‌کند. اگر سرویس‌گیرنده، شیئی را بخواهد که واسط `IStorable` را پیاده‌سازی می‌کند، می‌تواند عمل قالب‌بندی انجام دهد، اما

زمانی که شی Document را بکار می‌برد، Read() و Write() در آن وجود ندارند. در مثال ۱۷-۵، شی Document متد Talk() را به عنوان یک متد از Document در اختیار قرار می‌دهد، اما ITalk.Read() فقط از طریق عمل قالب‌بندی می‌تواند در دسترس قرار گیرد.

توجه: چون پیاده‌سازی صریح واسط، از کاربرد virtual جلوگیری می‌کند، یک کلاس مشتق شده مجبور است متد را مجدداً پیاده‌سازی کند. پس، اگر Note از Document مشتق شود، آن باید ITalk.Read() را مجدداً پیاده‌سازی کند.

۱۷-۴-۲-پنهان کردن اعضا

پنهان کردن عضوی از یک واسط امکان‌پذیر است. برای مثال، فرض کنید یک واسط بنام IBase داریم که خصوصیتی بنام P دارد.

```
interface IBase
{
    int P { get; set; }
}
```

فرض کنید یک واسط جدید بنام IDerived مشتق می‌کنیم که خصوصیت P را با یک متد P() پنهان می‌کند.

```
interface IDerived : IBase
{
    new int P();
}
```

پیاده‌سازی این واسط مشتق شده، حداقل یک عضو صریح لازم دارد. باید پیاده‌سازی صریح را حداقل برای خصوصیت واسط پایه یا متد مشتق شده بکار برید. پس، هر سه نسخه‌ی زیر قانونی هستند.

```
class myClass : IDerived
{
    // explicit implementation for the base property
    int IBase.P { get { ... } }
    // implicit implementation of the derived method
    public int P( ) { ... }
}

class myClass : IDerived
{
    // implicit implementation for the base property
    public int P { get { ... } }
    // explicit implementation of the derived method
    int IDerived.P( ) { ... }
}

class myClass : IDerived
{
    // explicit implementation for the base property
    int IBase.P { get { ... } }
    // explicit implementation of the derived method
    int IDerived.P( ) { ... }
}
```

۱۷-۴-۳-دستیابی به کلاس‌های مهرشده و انواع داده‌ی مقداری

در کل، دستیابی به متدهای یک واسط از طریق یک قالب‌بندی واسط ارجح‌تر است. کلاس‌های مهر شده و انواع داده‌ی مقداری از این قضیه مستثنی هستند. در این حالت، احضار متد واسط از طریق شی ارجح‌تر است.

زمانی که یک واسط را در یک ساختار پیاده‌سازی می‌کنید، شما آن را بصورت یک نوع مقداری پیاده‌سازی می‌کنید. زمانی که آن را به یک ارجاع واسط قالب‌بندی می‌کنید، یک جعبه‌بندی ضمنی شی انجام می‌شود. متأسفانه، زمانی که این واسط را برای تغییر شی بکار می‌برید، شی جعبه‌بندی شده نه شی مقداری اصلی تغییر می‌یابد. پس اگر مقدار ساختار را از متد داخلی

تغییر دهید، نوع داده‌ی جعبه‌بندی شده بدون تغییر می‌ماند. مثال ۶-۱۷ یک ساختار برای پیاده‌سازی IStorable ایجاد می‌کند و تأثیر جعبه‌بندی ضمنی در زمان قالب‌بندی ساختار به یک ارجاع واسط را نشان می‌دهد.

مثال ۶-۱۷

```
using System;
#region Using directives
using System;
using System.Collections.Generic;
using System.Text;
#endregion
namespace ReferencesOnValueTypes
{
    // declare a simple interface
    interface IStorable
    {
        void Read( );
        int Status { get;set;}
    }
    // Implement through a struct
    public struct myStruct : IStorable
    {
        public void Read( )
        {
            Console.WriteLine(
                "Implementing IStorable.Read" );
        }
        public int Status
        {
            get
            {
                return status;
            }
            set
            {
                status = value;
            }
        }
        private int status;
    }
    public class Tester
    {
        static void Main( )
        {
            // create a myStruct object
            myStruct theStruct = new myStruct( );
            theStruct.Status = -۱; // initialize
            Console.WriteLine(
                "theStruct.Status: {۰}", theStruct.Status );
            // Change the value
            theStruct.Status = ۲;
            Console.WriteLine( "Changed object." );
            Console.WriteLine(
                "theStruct.Status: {۰}", theStruct.Status );
            // cast to an IStorable
            // implicit box to a reference type
            IStorable isTemp = ( IStorable ) theStruct;
            // set the value through the interface reference
            isTemp.Status = ۴;
            Console.WriteLine( "Changed interface." );
            Console.WriteLine( "theStruct.Status: {۰}, isTemp: {۱}",
                theStruct.Status, isTemp.Status );
            // Change the value again
            theStruct.Status = ۱;
        }
    }
}
```

```

Console.WriteLine( "Changed object." );
Console.WriteLine( "theStruct.Status: {۰}, isTemp: {۱}",
theStruct.Status, isTemp.Status );
}
}
}
Output:
theStruct.Status: -۱
Changed object.
theStruct.Status: ۲
Changed interface.
theStruct.Status: ۲, isTemp: ۴
Changed object.
theStruct.Status: ۶, isTemp: ۴

```

در مثال ۱۷-۶ واسط `IStorable` یک متد `Read()` و یک خصوصیت `Status` دارد. این واسط با یک ساختاری بنام `myStruct` پیاده‌سازی می‌شود.

```
public struct myStruct : IStorable
```

کد جالب در داخل `Tester` است. با ایجاد یک نمونه از ساختار و مقداردهی خصوصیت با ۱- شروع می‌شود. سپس مقدار `Status` چاپ می‌شود.

خروجی آن بصورت زیر است:

```
theStruct.Status: -۱
```

سپس مقدار خصوصیت `Status` از طریق خود شی تغییر می‌دهد. خروجی بصورت زیر تغییر می‌یابد.

```
Changed object.
```

```
theStruct.Status: ۲
```

تا اینجا، چیز جالبی وجود نداشت. در این نقطه، یک ارجاع به واسط `IStorable` ایجاد کنید. این یک جعبه‌بندی ضمنی از شی مقداری `theStruct` ایجاد می‌کند. سپس این واسط را برای تغییر دادن مقدار `Status` به ۴ بکار برید. خروجی می‌تواند کمی جالب به نظر رسد:

```
Changed interface.
```

```
theStruct.Status: ۲, isTemp: ۴
```

وه، شیئی که به ارجاع واسط اشاره می‌کند تغییر یافته است، اما شی مقداری ساختار تغییر نیافته است. جالب‌تر زمانی است که متد را از طریق خود شی دستیابی می‌کنید. خروجی نشان می‌دهد که شی مقداری تغییر یافته است، اما مقدار ارجاع جعبه‌بندی شده برای ارجاع واسط تغییر نیافته است.

```
Changed object.
```

```
theStruct.Status: ۶, isTemp: ۴
```

۱۷-۵- خلاصه

- یک واسط، قراردادی است که برای یک سرویس‌گیرنده نحوه‌ی رفتار یک کلاس یا ساختار را تضمین خواهد کرد
- زمانی که یک کلاس، واسطی را پیاده‌سازی می‌کند، آن باید تمامی بخش‌های واسط را پیاده‌سازی کند.
- هدف یک واسط، تعریف توانایی‌هایی است که می‌خواهد در یک کلاس موجود باشد.
- معرف‌های دسترسی در اعلان متد وجود ندارند. در حقیقت، تعیین یک معرف دسترسی یک خطای کامپایل تولید می‌کند. متدهای واسط مطلقاً `public` هستند.

- بسط دادن یک واسط موجود برای اضافه کردن متدها یا اعضای جدید یا تغییر نحوه‌ی کار اعضای موجود امکان‌پذیر است.
- کلاس‌ها می‌توانند بیش از یک واسط را پیاده‌سازی کنند.
- می‌توانید واسط‌های جدیدی با ترکیب واسط‌های موجود و اضافه کردن خصوصیات و متدهای جدید ایجاد کنید.
- عملگر `as` عملیات `is` و قالب‌بندی را باهم ترکیب می‌کند. ابتدا بررسی می‌کند آیا عمل قالب‌بندی معتبر است، اگر معتبر باشد، عمل قالب‌بندی را انجام می‌دهد. اگر عمل قالب‌بندی معتبر نباشد، عملگر `as` مقدار `null` برمی‌گرداند.
- واسط‌ها بسیار شبیه کلاس‌های انتزاعی هستند. در حقیقت می‌توانید اعلان یک `IStorable` را تغییر دهید تا یک کلاس انتزاعی باشد.
- کلاس پیاده‌سازی کننده می‌تواند هر تعداد از متدهای پیاده‌سازی شده‌ی واسط را به عنوان مجازی علامت‌گذاری کند.
- پیاده‌سازی صریح برای از بین بردن مشکل تداخل اسمی متدهای هم‌نام در واسط‌ها است.
- تنها روش دسترسی به واسط پیاده‌سازی شده صریح، از طریق قالب‌بندی به یک واسط است.
- پنهان کردن عضوی از یک واسط امکان‌پذیر است
- زمانی که یک واسط را در یک ساختار پیاده‌سازی می‌کنید، شما آن را بصورت یک نوع مقداری پیاده‌سازی می‌کنید.

نماینده‌ها و رویدادها

بعد از خواندن این فصل قادر به انجام کارهای زیر خواهید بود:

- اعلان یک نوع داده‌ی نماینده برای ایجاد یک انتزاع از یک نشانه متد
- ایجاد یک نمونه از نوع داده‌ی نماینده برای ارجاع به یک متد بی‌نام
- فراخوانی یک متد از طریق یک نماینده
- اعلان یک فیلد رویداد
- اداره کردن یک رویداد با استفاده از نماینده
- رها کردن یک رویداد

بیشتر کدهایی که تا به حال نوشته‌اید به طور ترتیبی اجرا می‌شوند. در بعضی مواقع لازم است برنامه جریان جاری اجرا را متوقف کرده و کار مهمتر دیگری را انجام دهد. زمانی که کار کامل شود، برنامه می‌تواند از جایی که ترک شده بود ادامه یابد. مثال کلاسیک این نوع برنامه‌ها، برنامه‌های ویندوزی است. یک فرم کنترل‌هایی همچون دکمه و کادر متنی‌ها را نمایش می‌دهد. زمانی که روی یک دکمه کلیک می‌کنید یا متنی در کادر متنی تایپ می‌کنید، انتظار دارید فرم فوراً به آن عکس‌العمل نشان دهد. برنامه باید کار جاری را موقتاً متوقف کند و ورودی شما را اداره کند. این نوع عملیات فقط به واسطه‌های گرافیکی کاربر اعمال نمی‌شود، در هر نوع برنامه می‌توانیم از این تکنولوژی استفاده کنیم.

به منظور اداره‌ی این نوع برنامه، در زمان اجرا دو چیز باید فراهم گردد: وسیله‌ای برای تشخیص مورد اضطراری که رخ داده است و روشی برای تعیین متدی که در حین وقوع این رویداد باید اجرا شود. این عملیات هدف رویدادها و نماینده‌ها است.

۱۸-۱- اعلان و کاربرد نماینده‌ها

یک نماینده^۱، اشاره‌گری به یک متد است. یک نماینده در هنگام فراخوانی شبیه یک متد عادی بوده و شبیه یک متد رفتار می‌کند. زمانی که یک نماینده را فراخوانی می‌کنید، برنامه متدی که نماینده به آن ارجاع می‌کند را اجرا می‌کند. می‌توانید به طور پویا متدی که یک نماینده به آن ارجاع می‌کند را تغییر دهید. بنابراین کدی که یک نماینده را فراخوانی می‌کند، ممکن است در هر بار اجرا متد مختلفی را اجرا کند. بهترین راه فهم نماینده‌ها، دیدن آنها در عمل است. یک مثال بررسی می‌کنیم.

^۱ Delegate

۱۸-۱-۱- سناریوی کارخانه اتوماتیک

فرض کنید سیستم‌های کنترل را برای یک کارخانه‌ی اتوماتیک می‌نویسید. کارخانه تعداد زیادی ماشین مختلف دارد که هر کدام کارهای مجزایی در تولید اقلام ساخته شده به وسیله کارخانه انجام می‌دهند. شکل دادن و تا کردن ورقه‌های فلزی، جوشکاری ورقه‌ها به هم، رنگ کاری ورقه‌ها و غیره. هر ماشین توسط فروشنده‌ی خاصی ساخته و نصب شده است. ماشین‌ها، همه کنترل شده به وسیله کامپیوتر هستند و هر فروشنده مجموعه‌ای از API ها فراهم کرده‌اند تا بتوانید برای کنترل آنها به کار برید. کار شما جمع کردن سیستم‌های مختلف این ماشین‌ها در یک برنامه کنترلی واحد است. یکی از جوانبی که تصمیم دارید روی آن تمرکز کنید، فراهم کردن یک وسیله برای خاموش کردن همه ماشین‌ها در مواقع اضطراری است.

توجه: عبارت API به معنی واسط برنامه‌نویسی کاربردی است. آن یک متد یا مجموعه‌ای از متدها است که به شما اجازه می‌دهد یک نرم افزار را کنترل کنید. می‌توانید چارچوب NET را به صورت مجموعه‌ای از توابع API در نظر بگیرید.

هر ماشین پروسه‌ی کنترلی منحصر به فرد خود را برای خاموش کردن امن دارد. اینها به صورت زیر خلاصه می‌شوند:

```
StopFolding(); // Folding and shaping machine
FinishWelding(); // Welding machine
PaintOff(); // Painting machine
```

۱۸-۱-۲- پیاده‌سازی کارخانه بدون کاربرد نماینده‌ها

یک روش ساده برای پیاده‌سازی عمل خاموش کردن سیستم در یک برنامه‌ی کنترلی بصورت زیر نمایش داده می‌شود:

```
class Controller
{
    ...
    public void ShutDown()
    {
        folder.StopFolding();
        welder.FinishWelding();
        painter.PaintOff();
    }
    ...
    // Fields representing the different machines
    private FoldingMachine folder;
    private WeldingMachine welder;
    private PaintingMachine painter;
}
```

اگرچه این روش کار می‌کند، ولی آن خیلی قابل توسعه و انعطاف‌پذیر نیست. اگر کارخانه یک ماشین جدید بخرد، شما باید این کد را تغییر دهید. کلاس Controller و کلاس‌های ماشین خیلی قوی به هم پیوسته‌اند.

۱۸-۱-۳- پیاده‌سازی کارخانه با استفاده یک نماینده

اگرچه اسامی هر متد متفاوت است، ولی همه‌ی آنها صورت یکسانی دارند. آنها هیچ پارامتری نمی‌گیرند و هیچ مقداری بر نمی‌گردانند. قالب کلی هر متد بصورت زیر است.

```
void methodName();
```

در اینجا یک نماینده مفید است. یک نماینده که با این صورت متد مطابقت داشته باشد، می‌تواند برای ارجاع به هر متد خاموش کردن ماشین استفاده شود. یک نماینده را شبیه زیر اعلان کنید:

```
delegate void stopMachineryDelegate();
```

به نکات زیر توجه کنید:

• زمان اعلان یک نماینده کلمه‌ی کلیدی delegate را بکار برید.

- یک نماینده صورت متدهایی که می‌تواند به آنها ارجاع کند را تعریف می‌کند. یک نوع داده مقدار بازگشتی، نام نماینده و پارامترهای آن را مشخص کنید.

بعد از تعریف نماینده، می‌توانید یک نمونه از آن ایجاد کنید و با استفاده از عملگر += آنرا به متد منطبق ارجاع دهید. می‌توانید این کار را در سازنده‌ی کلاس Controller انجام دهید.

```
class Controller
{
    delegate void stopMachineryDelegate();
    ...
    public Controller()
    {
        this.stopMachinery += folder.StopFolding;
    }
    ...
    private stopMachineryDelegate stopMachinery; // Create an instance of the delegate
}
```

شما یک متد به نماینده اضافه کردید. اما در واقع، متد را در این نقطه فراخوانی نکرده‌اید. عملگر + overload می‌شود تا در زمان استفاده در نماینده‌ها، معنی جدیدی داشته باشد. توجه کنید که شما فقط نام متد را مشخص کردید و نباید هیچ پارامتر یا پارامتری را در بر داشته باشد. کاربرد عملگر += روی یک نماینده‌ی مقداردهی اولیه نشده صحیح است. آن بطور اتوماتیک مقداردهی اولیه خواهد شد. شما می‌توانید کلمه‌ی کلیدی new را برای مقداردهی اولیه یک نماینده به متد خاص بکار ببرید، شبیه این:

```
this.stopMachinery = new stopMachineryDelegate(folder.stopFolding);
```

می‌توانید با احضار نماینده، متد را فراخوانی کنید.

```
public void ShutDown()
{
    this.stopMachinery();
    ...
}
```

احضار یک نماینده دقیقاً شبیه فراخوانی یک متد است و اگر متدی که نماینده به آن ارجاع می‌کند پارامتری دارد، همین جا باید آنها را مشخص کنید.

نکته: اگر شما برای احضار نماینده‌ی مقداردهی اولیه نشده تلاش کنید، یک استثناء NullReferenceException خواهید گرفت.

مزیت اصلی کاربرد یک نماینده توانایی ارجاع آن به بیش از یک متد است. به سادگی عملگر += را برای اضافه کردن متدها به نماینده به کار ببرید.

```
public Controller()
{
    this.stopMachinery += folder.StopFolding;
    this.stopMachinery += welder.FinishWelding;
    this.stopMachinery += painter.PaintOff;
}
```

احضار this.stopMachinery () در متد ShutDown از کلاس Controller، به طور اتوماتیک هر متد را به ترتیب فراخوانی خواهد کرد. متد ShutDown () نیازی نیست بداند چند تا ماشین وجود دارد یا اسامی متد چیست. با استفاده از عملگر -= می‌توانید یک متد را از یک نماینده حذف کنید.

```
this.stopMachinery -= folder.StopFolding;
```


شمای جاری، متدهای ماشین را در سازنده به نماینده اضافه می‌کند. برای اینکه کلاس Controller را کاملاً مستقل از ماشین‌های متنوع نگه دارید. لازم است یک روشی فراهم کنید که از خارج کلاس بتوان متدهایی را به نماینده اضافه کرد. چندین انتخاب وجود دارد:

- متغیر نماینده را public اعلان کنید.

```
public stopMachineryDelegate stopMachinery;
```

- متغیر نماینده را private تعریف کنید، ولی یک خصوصیت برای خواندن و نوشتن به آن اضافه کنید. البته باید تعریف نماینده را public مشخص کنید.

```
public delegate void stopMachineryDelegate();
...
public stopMachineryDelegate StopMachinery
{
    get
    {
        return this.stopMachinery;
    }
    set
    {
        this.stopMachinery = value;
    }
}
```

پیاده‌سازی متدهای Add و Remove، کپسوله‌سازی کامل را فراهم می‌آورد. متد Add یک متد را به عنوان یک پارامتر گرفته و آن را به نماینده اضافه می‌کند و با متد Remove یک متد را از نماینده حذف می‌کند.

```
public void Add(stopMachineryDelegate stopMethod)
{
    this.stopMachinery += stopMethod;
}
public void Remove(stopMachineryDelegate stopMethod)
{
    this.stopMachinery -= stopMethod;
}
```

اگر برنامه‌نویس شی‌گرای وسواسی هستید، احتمالاً روش Remove/ Add را انتخاب می‌کنید. هر تکنیکی که انتخاب می‌کنید، باید کد مربوط به اضافه کردن متد به نماینده را از سازنده‌ی Controller حذف کنید. سپس اشیای مورد نیاز از کلاس Control و ماشین‌های دیگر را به صورت زیر معرفی کنید.

```
Controller control = new Controller();
FoldingMachine folder = new FoldingMachine();
WeldingMachine welder = new WeldingMachine();
PaintingMachine painter = new PaintingMachine();
...
control.Add(folder.StopFolding);
control.Add(welder.FinishWelding);
control.Add(painter.PaintOff);
...
control.ShutDown();
...
```

۱۸-۱-۴- متدها و نماینده‌های بی‌نام

همه مثال‌های اضافه کردن یک متد به نماینده که شما دیده‌اید، همگی نام متد را به کار می‌برند. برای مثال، به سناریوی کارخانه‌ی اتوماتیک که متد StopFolding شی Folder را به نماینده‌ی stopMachinery اضافه می‌کنید برگردید. ما این کار را کردیم:

```
this.stopMachinery += folder.StopFolding;
```

این روش در صورتی که یک متد مناسب منطبق با نشانه‌ی نماینده وجود داشته باشد، مفید است. اما اگر نباشد، چه اتفاقی می‌افتد؟ فرض کنید، متد StopFolding نشانه زیر را داشته باشد.

```
void StopFolding(int shutDownTime); // Shut down within the
specified number of seconds
```

این با متدهای FinishWelding و PaintOff فرق می‌کند. پس نمی‌توانیم نماینده‌ی یکسانی برای اداره کردن هر سه متد به کار ببریم.

ایجاد یک مبدل متد

روش پیرامون این مسأله، ایجاد متد دیگری است که StopFolding را فراخوانی می‌کند و هیچ پارامتری نمی‌گیرد. شبیه این:

```
void FinishFolding()
{
    folder.StopFolding(0); // Shutdown immediately
}
```

توجه: متد FinishFolding یک مثال کلاسیک از مبدل است. متدی که یک متد را به متد با نشانه‌ی مختلف تبدیل می‌کند.

در بیشتر موارد، متدهای مبدل همانند این کوچک هستند و گم کردن آنها در دریایی از متدها ساده‌تر است. جدا از کاربرد آن برای تبدیل متد StopFolding جهت استفاده توسط نماینده، آن در جای دیگر فراخوانی می‌شود. #C متدهای بی‌نام را برای این مواقع فراهم می‌کند.

کاربرد یک متد بی‌نام به عنوان یک مبدل

متد بی‌نام، متدی است که نام ندارد. این بسیار عجیب به نظر می‌رسد. اما متدهای بی‌نام کاملاً مفید هستند. در صورتی که یک قطعه کد داریم که هرگز مستقیماً فراخوانی نمی‌شود، اما دوست دارید از طریق یک نماینده قادر به فراخوانی آن باشید، اینها فرصت خوبی هستند. متد FinishFolding که تا به حال بحث شده، یک مثال است. هدف منحصر به فرد آن، فراهم کردن مبدل برای StopFolding است. هر جایی که یک نماینده را به کار می‌برید، می‌توانید متد بدون نام استفاده کنید. کد مورد نظر را داخل {} بعد از کلمه‌ی کلیدی delegate قرار دهید. برای استفاده از متد بدون نام به عنوان یک مبدل برای متد StopFolding و اضافه کردن آن به نماینده‌ی StopMachinery، می‌توانید این را بنویسید.

```
this.stopMachinery += delegate { folder.StopFolding(0); };
```

دیگر نیازی به ایجاد متد FinishFolding نیست.

می‌توانید یک متد بدون نام را به عنوان یک پارامتر به جای نماینده به یک متد ارسال کنید. شبیه این:

```
control.Add(delegate { folder.StopFolding(0); } );
```

ویژگی‌های متدهای بی‌نام

- هر پارامتر مورد نیاز در {} بعد از کلمه‌ی کلیدی delegate مشخص می‌شوند. برای مثال:

```
control.Add(delegate(int param1, string param2) { /* code that uses param1
and param2 ... */ });
```

- متدهای بدون نام می‌توانند مقادیری را برگردانند. اما نوع مقدار بازگشتی باید با نماینده‌ای که به آن اضافه می‌شود تطابق داشته باشد.

- شما می‌توانید عملگر = - را برای حذف یک متد از یک متغیر نماینده به کار برید. اما آن چیزی که حذف خواهد کرد را دقیقاً حس نمی‌کند. متدهای بی‌نام که کد یکسانی را دربردارند، در واقع نمونه‌های مختلفی از کد هستند.
- کد یک متد بی‌نام، کد عادی #C است. آن می‌تواند از چندین دستور، فراخوانی متدها، تعریف متغیرها و غیر تشکیل شود.
- زمانی که متد بی‌نام خاتمه می‌یابد، متغیرهای تعریف شده در یک متد بی‌نام، از دامنه خارج می‌شوند.
- یک متد بی‌نام می‌تواند به متغیرهای دامنه‌ی خود دستیابی کند و تغییرات انجام دهد. متغیرهای دستکاری شده با این روش را متغیرهای خارجی تصرف شده^۱ گویند. در مورد این ویژگی خیلی مواظب باشید.

فعال کردن هشدارها بوسیله‌ی رویدادها

در بخش قبلی نحوه‌ی اعلان یک نماینده، فراخوانی آن و ایجاد نمونه‌هایی از نماینده را فرا گرفتیم. اگرچه نماینده‌ها احضار چندین متد را به صورت غیرمستقیم مجاز می‌دارند، شما هنوز مجبور هستید نماینده را صریحاً احضار کنید. اگر یک نماینده هنگام وقوع یک چیز مهم بطور اتوماتیک اجرا گردد، این عمل مفید خواهد بود

مثال: در سناریوی کارخانه‌ی اتوماتیک، توانایی احضار نماینده StopMachinery برای متوقف کردن وسیله (زمانی که بیش از حد داغ باشد)، حیاتی است. در چارچوب NET، رویدادها به شما اجازه می‌دهند عملیات مهم را تعریف کرده و محبوس^۲ نمایید و آنها را برای یک نماینده که جهت اداره کردن این موقعیت فراخوانی می‌شود، نظم دهید. بیشتر کلاس‌ها در چارچوب NET، رویدادها را آشکار می‌سازند.

۱۸-۲-اعلان یک رویداد

یک رویداد را در یک کلاس نامزد برای عمل کردن به صورت یک منبع رویداد اعلان کنید. یک منبع رویداد معمولاً کلاسی است که بر محیط نظارت می‌کند و در صورت وقوع یک چیز مهم، یک رویداد آزاد می‌کند. در کارخانه‌ی اتوماتیک، یک منبع رویداد می‌تواند کلاس مربوط به نظارت بر دمای لحظه‌ای ماشین باشد. کلاس ناظر دماسنج اگر تشخیص دهد که یک ماشین بیش از حد گرم شده است، رویداد زیاد گرم شدن را آزاد می‌کنند. یک رویداد لیستی از متدها را نگه می‌دارد که هنگام آزاد شدن رویداد باید فراخوانی شوند. بعضی اوقات این متدها را متعهدها^۳ می‌گویند. این متد باید برای اداره کردن رویداد زیاد گرم شدن فراهم شود و عمل اصلاحی ضروری انجام گیرد (برای مثال، ماشین خاموش شود).

یک رویداد را شبیه اعلان یک فیلد، اعلان کنید. چون رویدادها قصد دارند با نماینده‌ها به کار برده شوند. نوع یک رویداد باید نماینده باشد و در خط اعلان کلمه‌ی کلیدی event را قرار دهید.

مثال: نماینده‌ی StopMachineryDelegate از کارخانه اتوماتیک را در نظر بگیرید. من آن را به یک کلاس جدید به نام TemperatureMonitor انتقال داده‌ام، که این کلاس یک واسط برای نظارت الکترونیکی دمای دستگاه‌ها فراهم می‌کند.

```
class TemperatureMonitor
{
    public delegate void StopMachineryDelegate();
    ...
}
```

^۱ Captured

^۲ Trap

^۳ Subscriber

حال می‌توانید رویداد MachineOverheating را تعریف کنید که StopMachineryDelegate را احضار خواهد کرد.

```
class TemperatureMonitor
{
public delegate void StopMachineryDelegate();
public event StopMachineryDelegate MachineOverheating;
...
}
```

منطق کلاس TemperatureMonitor به طور اتوماتیک رویداد MachineOverheating را در صورت نیاز رها می‌کند. یک رویداد کلکسیونی داخلی از نماینده‌های وابسته را نگه می‌دارد. نیازی نیست متغیر نماینده‌تان را به صورت دستی نگه دارید.

۱۸-۲-۱- متعهد شدن به یک رویداد

شبیه نماینده‌ها، رویدادها نیز با عملگر += می‌آیند. با استفاده از عملگر += به یک رویداد متعهد می‌شوید. در کارخانه‌ی اتوماتیک، زمانی که رویداد MachineOverheating رها می‌شود، نرم افزار کنترل هر ماشین می‌تواند برای فراخوانی متدهای خاموش سازمانده‌ی شوند.

```
TemperatureMonitor tempMonitor = new TemperatureMonitor();
...
tempMonitor.MachineOverheating += delegate { folder.StopFolding(); };
tempMonitor.MachineOverheating += welder.FinishWelding;
tempMonitor.MachineOverheating += painter.PaintOff;
```

توجه کنید که گرامر با اضافه کردن یک متد به یک نماینده یکسان است. حتی می‌توانید با استفاده از متد بی‌نام متعهد شوید. زمانی که رویداد tempMonitor.MachineOverheating اجرا می‌گردد، آن همه‌ی متدهای متعهد شده را فراخوانی می‌کند و ماشین‌ها را خاموش می‌کند.

۱۸-۲-۲- غیر متعهد شدن از یک رویداد

به طوری که می‌دانید عمل گر -= برای وابسته کردن یک نماینده به یک رویداد استفاده می‌شود. می‌توانید حدس بزنید که عملگر -= برای جدا کردن یک نماینده از یک رویداد استفاده می‌گردد. فراخوانی عملگر -= متد را از کلکسیون نماینده داخلی رویداد حذف می‌کند. اغلب این عمل با عنوان غیرمتعهد شدن از یک رویداد بیان می‌کنند.

۱۸-۲-۳- رها کردن یک رویداد

با فراخوانی یک رویداد همچون یک متد یا نماینده، می‌توانید آن رویداد را رها کنید. زمانی که رویداد رها می‌شود، همه‌ی نماینده‌های وابسته شده به ترتیب فراخوانی می‌شوند. برای مثال، در اینجا کلاس tempretuerMonitor یک متد خصوصی Notify دارد که رویداد MachineryOverheating را رها می‌کند.

```
class TemperatureMonitor
{
public delegate void StopMachinerDelegate;
public event StopMachineryDelegate MachineOverheating;
...
private void Notify()
{
if (this.MachineOverheating != null)
{
this.MachineOverheating();
}
}
...
}
```

این زبان ویژه است. بررسی `null` ضروری است، چون یک فیلد رویداد قطعاً `null` است و فقط زمانی که یک متد با استفاده از عملگر `==` به آن متعهد گردد، غیر `null` می شود. اگر سعی کنید یک رویداد `null` را رها کنید. یک استثناء `NullReferenceException` دریافت خواهید کرد. اگر نماینده رویدادی تعریف کند که پارامترهایی را انتظار می رود، زمانی که رویداد رها می شود، باید آرگومانهای مناسب فراهم شوند.

۱۸-۳- رویدادهای GUI

همان طور که قبلاً بیان شد، کنترل ها و کلاس های چارچوب .NET برای ساختن GUI ها، رویداد را بطور وسیع به خدمت می گیرند. برای مثال، کلاس `Button` از کلاس `Control` مشتق می شود. یک رویداد `Click` از نوع `EventHandler` را به ارث می برد. کد زیر را ببینید. نماینده `EventHandler` دو پارامتر دارد. یکی ارجاع به شیئی است که رویداد را رها کرده است و شی `EventArgs` که اطلاعات اضافی درباره رویداد را دربردارد.

```
namespace System
{
    public delegate void EventHandler(object sender, EventArgs args) ;
    public class EventArgs
    {
        ...
    }
}
namespace System.Windows.Forms
{
    public class Control :
    {
        public event EventHandler Click;
        ...
    }
    public class Button : Control
    {
        ...
    }
}
```

زمانی که روی دکمه کلیک کنید، کلاس `Button` بطور اتوماتیک رویداد `Click` را رها می کند. این نظم، ایجاد یک نماینده برای یک متد انتخابی و وابسته کردن نماینده به رویداد موردنظر را ساده می کند. مثال زیر یک فرم ویندوز را نشان می دهد که یک دکمه به نام `okay`، متدی به نام `okay_Click` و کد اتصال رویداد `Click` دکمه `okay` به متد `okay_Click` را دارد.

```
class Example : System.Windows.Forms.Form
{
    private System.Windows.Forms.Button okay;
    ...
    public Example()
    {
        this.okay = new System.Windows.Forms.Button();
        this.okay.Click += new System.EventHandler(this.okay_Click);
        ...
    }
    private void okay_Click(object sender, System.EventArgs args)
    {
        // Your code to handle the Click event
    }
}
```

در هنگام استفاده از IDE مربوط به `VS.NET`، IDE کدی تولید می کند که متدها را به طور اتوماتیک به رویدادها متعهد می کند.

توجه: اضافه کردن یک متد به یک رویداد بدون ایجاد یک نمونه از یک نماینده امکان پذیر است. شما می‌توانید دستور زیر را با دستور بعدی جایگزین کنید.

```
this.okay.Click += new System.EventHandler (this.okay_Click);
```

با

```
this.okay.Click += this.okay_Click;
```

با این وجود، محیط طراحی فرم‌های ویندوز در VS ۲۰۰۵ همیشه اولین نسخه را تولید می‌کند. رویدادهایی که کلاس‌های GUI تولید می‌کنند، همواره الگوی یکسانی دارند. رویدادها از نوع نماینده‌ای هستند که دو پارامتر دارد و مقدار void بر می‌گرداند. اولین آرگومان آن شی رها کننده‌ی رویداد است و آرگومان دوم یک EventArgs است.

آرگومان sender اجازه می‌دهد یک متد برای چندین رویداد استفاده شود. متد مربوطه به نماینده می‌تواند آرگومان sender را بررسی کرده و به طور مناسب جواب دهد. برای مثال، می‌توانید متد یکسانی را برای متعهد کردن دو دکمه به رویداد Click بکار ببرید. زمانی که رویداد رها می‌شود، کد متد می‌تواند آرگومان sender را برای معین کردن دکمه‌ی کلیک شده استفاده شود.

خلاصه

- برای اعلان یک نوع داده نماینده، کلمه کلیدی delegate را نوشته و به دنبال آن نوع مقدار بازگشتی، نام یک نوع داده نماینده و پارامترهای آن را بنویسید.

```
delegate void stopMachineryDelegate();
```

- برای احضار یک نماینده به صورت زیر عمل کنید:

```
;private stopMachineryDelegate stopMachinery  
;() stopMachinery
```

- در ایجاد یک نمونه از یک نماینده، آرگومان نماینده باید متدی باشد که نشانه آن با نماینده منطبق باشد.

```
this.stopMachinery = new stopMachineryDelegate(folder.stopFolding);
```

- برای اعلان یک رویداد، کلمه کلیدی event را نوشته و به دنبال آن نوع مقدار بازگشتی و نام رویداد را بنویسید.

```
public delegate void StopMachineryDelegate();  
public event StopMachineryDelegate MachineOverheating;
```

- برای متعهد کردن یک متد، یک نمونه نماینده ایجاد کرده و نمونه نماینده را از طریق عملگر += به رویداد وابسته کنید.

```
tempMonitor.MachineOverheating += welder.FinishWelding;
```

- غیرمتعهد کردن متد از یک رویداد، همانند متعهد کردن است، فقط به جای عملگر += از عملگر -= استفاده کنید.
- قبل از اجرای یک متد مرتبط با event حتماً مطمئن شوید که رویداد موردنظر null نباشد.

فصل نوزدهم

اداره کردن استثناء

آنچه که در این فصل یاد خواهید گرفت:

- با مفهوم استثناء و نحوه‌ی مدیریت آن توسط سیستم عامل آشنا خواهید شد.
- با نحوه‌ی کنترل استثناء‌های برنامه توسط برنامه‌نویس آشنا خواهید شد.
- ساختار `try/catch/finally` را برای اداره کردن استثناء/ی به کار خواهید برد.
- استثناء‌های سفارشی جهت تولید پیام‌های مناسبی که در .NET پیش بینی نشده است ایجاد خواهید کرد.

۱۹-۱-مقدمه

یکی از مهمترین جنبه‌های مدیریت یک شی، اطمینان از رفتار و تعامل آن با سیستم‌های دیگر است که با یک خطا، برنامه را خاتمه ندهد. بدین معنی که باید یک برنامه با خطاهای زمان اجرا بصورت مطلوب رفتار نماید. اینکه آنها از خطای کد برنامه (FCL) یا خطاهای سخت‌افزاری نشأت می‌گیرند.

.NET یک تکنیک بنام اداره کردن ساخت‌یافته‌ی استثناء (SEH)^۱ به منظور برخورد با شرایط خطا برای توسعه‌دهندگان فراهم می‌سازد. هدف اصلی این است که، زمانی که یک استثناء رخ می‌دهد، یک شی استثناء ایجاد شده و کنترل برنامه به بخش خاصی از کد رد می‌شود. در اصطلاحات .NET، شی استثناء از یک بخش به بخش دیگر که آن را درک می‌کند، منتقل می‌شود.

در مقایسه با تکنیک‌های اداره کردن خطا، که به کدهای خطا استناد می‌کنند و مقادیر بیتی را تنظیم می‌کنند، SEH مزایای مهمی را پیشنهاد می‌کند:

❖ استثناء همانند یک شی به برنامه رد می‌شود. خصوصیات آن، یک توصیف از استثناء، نام اسمبلی که استثناء را رد کرده است، مقداری از پشته که نشان دهنده دنباله‌ی فراخوانی‌های منجر شده به این استثناء هستند، را شامل می‌شوند.

❖ اگر یک استثناء به برنامه رد شود و برنامه نتواند آن را درک کند، CLR برنامه را خاتمه می‌دهد. این عمل توسعه دهنده را به اداره کردن خطا مجبور می‌سازد.

❖ لازم نیست کد تشخیص و اداره‌ی استثناء در محل رخ دادن خطا قرار گیرد. بدین معنی که، کد اداره کردن استثناء می‌تواند در یک کلاس خاص مختص یافته به آن خطا قرار گیرد.

^۱ Structured exception handling

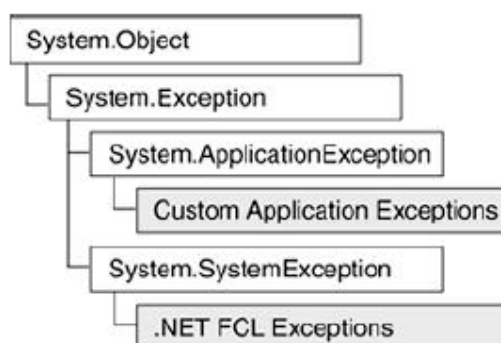
❖ استثناها در هر دو سطح برنامه و سیستم بطور سازگار و انحصاری استفاده می‌شوند. همه متدهای چارچوب .NET، زمانی که یک خطا رخ دهد استثناء رد می‌کنند.

قبل از مشاهده مکانیزم‌های واقعی پیاده‌سازی اداره کردن استثناء، اجازه دهید خود استثناء را بررسی کنیم. همانطور که قبلاً شرح دادیم، استثناء یک کلاس است. همه استثنای .NET از کلاس System.Exception مشتق می‌شوند. پس فهم این کلاس برای کار با استثناء ضروری است.

۱۹-۲- کلاس System.Exception

همانطور که در شکل ۱۹-۱ نمایش داده شده است، System.Exception کلاس پایه برای دو زیر کلاس کلی SystemException و ApplicationException است که همه اشیاء استثناء از آنها ارث‌بری می‌کنند. استثنای چارچوب .NET (همچون IOException و ArithmeticException) مستقیماً از SystemException مشتق می‌شوند، در حالی که استثنای سفارشی برنامه از ApplicationException ارث‌بری می‌کنند. تنها هدف این کلاس‌ها گروه‌بندی استنهااست. چون آنها هیچ خصوصیت یا متدی به کلاس پایه System.Exception اضافه نمی‌کنند.

شکل ۱۹-۱



کلاس System.Exception چند تا عضو دارد. جدول ۱۹-۱ اعضای این کلاس را خلاصه می‌کند.

جدول ۱۹-۱

توصیف	نوع	خصوصیت
یک URL که به مستندات کمکی اشاره می‌کند.	String	HelpLink
مقدار null دارد، مگر اینکه استثناء در هنگام اداره کردن استثناء دیگر رخ دهد. با متد GetBaseException می‌توان زنجیره‌ی استثنای تودرتو را یافت.	Exception	InnerException
متنی که استثناء را شرح می‌دهد.	String	Message
نام اسمبلی که استثناء را تولید کرده است.	String	Source
دنباله‌ای از اسامی و نشانه‌ی متدها که قبل از استثناء فراخوانی شده‌اند. این اطلاعات در خطازدایی ارزشمند هستند.	String	StackTrace
جریبانی در مورد متدی که استثناء را رد کرده است فراهم می‌کند. نام متدی که استثناء در آن اتفاق افتاده است را بر می‌گرداند. آن	MethodBase	TargetSite

یک خصوصیت بنام DeclaringType دارد که نام کلاس مربوط به متد را بر می گرداند.		
این خصوصیت محافظت شده در زمان کار با کد Com استفاده می شود. زمانی که یک استثنا به یک سرویس گیرنده ی Com رد می شود، این مقدار به یک HRESULT در دنیای کد مدیریت نشده تبدیل می گردد.	Int32	HRESULT

۱۹-۳- کدنویسی برای اداره کردن استثناها

#C یک ساختار try/catch/finally برای پیاده سازی اداره کردن استثناء (شکل ۱۹-۲) به کار می برد. زمانی که یک استثناء رخ می دهد، سیستم بلوک catch ی را جستجو می کند که می تواند استثناء جاری را اداره کند. آن جستجو را از متد جاری شروع می کند و اگر پیدا نشود لیست بلوک های catch موجود در متد فراخوانده ی این متد را جستجو می کند، اگر عمل جستجو بلوک catch منطبق را پیدا نکند یک استثناء اداره نشده رخ می دهد. همانطور که بعداً بررسی خواهیم کرد، برنامه کاربردی مسئول تعریف یک سیاست برای برخورد با این استثناها است. به جزئیات کاربرد این سه بلوک توجه کنید.

شکل ۱۹-۲

```
try {
    // Code that may cause an exception.
    // It may consist of multiple lines of code.
}
// May contain any number of catch blocks.
catch(exception name) {
    // Place code here that handles the exception.
    // Catch block may contain a throw statement.
}
catch(exception name) {
    // Place code here that handles the exception.
}

finally {
    // This code is always executed whether or not an
    // exception occurs.
}
```

بلوک try

کد داخل این بلوک را ناحیه ی محافظ می گویند، چون آن ناحیه بلوک های Catch یا finally را برای اداره کردن استثناهای ممکن یا حذف عوارض جانبی دارند. هر بلوک try باید حداقل یک بلوک catch یا finally مرتبط با آن داشته باشد.

بلوک catch

یک بلوک `Catch`، کلمه‌ی کلیدی `catch` و به دنبال آن در داخل پرانتزها یک عبارت به نام فیلتر استثناء را شامل می‌شود. فیلتر استثناء، نوع استثنای که آن جوابگر است را مشخص می‌کند. کد داخل بلوک عکس‌العمل را پیاده‌سازی می‌کند.

فیلتر استثناء، استثنائی که آن اداره می‌کند را تعریف می‌کند و زمانی که یک استثناء به آن پاس داده می‌شود آنرا بصورت یک پارامتر به کار می‌گیرد. دستور زیر را ملاحظه کنید.

```
Catch (Divide By zero      ex) {.....} Exception
```

اگر یک `System.DivideByZeroException` رخ دهد، این فیلتر احضار می‌شود. متغیر `ex` به استثناء ارجاع می‌کند و دسترسی به خصوصیات آنرا فراهم می‌سازد. همچون `ex.Message` و `ex.StackTrace`

زمانی که چندین بلاک `catch` بکار می‌برید، ترتیب مهم است. آنها باید بصورت سلسله‌مراتبی لیست شوند. که از خاص‌ترین استثناء شروع شده و به استثناء کلی‌تر خاتمه می‌یابد. در واقع اگر ترتیب آنها را درست نچینید، کامپایلر یک خطا تولید می‌کند.

```
{.....} (Catch(Divide By zero Exception      ex
```

```
{.....} (Catch ( Index out of Range Exception      ex
```

```
{.....} (Catch (Exception      ex
```

این کد ابتدا استثناء تقسیم بر صفر یا اندیس خارج از محدوده را جستجو می‌کند. فیلتر استثناء آخری (`Exception`) هر استثناء مشتق شده از `System.Exception` را پاس می‌کند.

زمانی که یک استثناء رخ دهد، کد این بلوک اجرا می‌شود و از بقیه بلوک‌های `catch` پرش می‌شود. اگر بلوک `finally` وجود داشته باشد کنترل برنامه به آن جریان می‌یابد.

توجه: ممکن است بلوک `catch` یک دستور `throw` برای پاس کردن استثناء به متد فراخوانی کننده در بالای پشته داشته باشد. دستور `throw` یک پارامتر اختیاری در داخل پرانتزها دارد که نوع استثناء پاس شده را مشخص می‌کند اگر `throw` بدون پارامتر استفاده شود، استثناء در همین متد پاس می‌شود. زمانی که متد فراخوانی کننده برای اداره کردن استثناء بهتر باشد، یک استثناء را به آن پاس می‌کنند.

بلوک `finally`

به عنوان یک بلوک تصفیه در نظر گرفته می‌شود. بلوک `finally` اجرا می‌شود. خواه استثناء رخ دهد، خواه رخ ندهد. آن محل مناسبی برای انجام عملیات تصفیه همچون بستن فایل و اتصالات پایگاه داده است. اگر بلوک `catch` نداشته باشیم، این بلوک ضروری است در غیر این صورت اختیاری است.

مثال اداره کردن استثناهای عمومی `System.Exception`

مثال ۱۹-۱ بلوک‌های `try/catch/finally` را در برخورد با یک استثناء تولید شده توسط CLR نشان می‌دهد.

مثال ۱۹-۱

Listing ۴-۲ Using System;

```
// Class to illustrate results of division by zero
public class TestExcep
{
```

```

public static int Calc(int j)
{
    return (100 / j);
}
}
class MyApp
{
    public static void Main()
    {
        TestExcep exTest = new TestExcep();
        try
        {
            // Create divide by zero in called method
            int dZero = TestExcep.Calc(0);
            // This statement is not executed
            Console.WriteLine("Result: {0}", dZero);
        }
        catch (DivideByZeroException ex)
        {
            Console.WriteLine("{0}\n{1}\n", ex.Message, ex.Source);
            Console.WriteLine(ex.TargetSite.ToString());
            Console.WriteLine(ex.StackTrace);
        }
        catch (Exception ex)
        {
            Console.WriteLine("General "+ex.Message);
        }
        finally
        {
            Console.WriteLine("Cleanup occurs here.");
        }
    }
}

```

در این مثال `TestExcep.Calc` یک استثناء تقسیم بر صفر پاس می‌کند، زمانی که آن را با مقدار صفر فراخوانی می‌کنیم. چون `Calc` هیچ کدی برای اداره کردن استثناء ندارد، به طور اتوماتیک به نقطه فراخوانی `Calc` در `MyApp` پاس داده می‌شود و در آنجا، کنترل برنامه به بلوک اداره کردن استثناء `DivideByZeroException` جریان می‌یابد. برای نمونه، دستورات بلوک `catch` اطلاعات زیر را نمایش می‌دهد.

مقدار داخل آن	خصوصیت
Attempted to divide By Zero	Ex.Message
Zeroexcept	Ex.Source
(void Main)	ex.TargetSite
(at MyApp .Main)	ex.StackTrace

۱۹-۴- چگونه یک کلاس استثناء سفارشی ایجاد کنیم؟

زمانی که شما نیاز دارید خطاهای منتشر شده توسط کلاس‌ها را شرح دهید، کلاس‌های استثناء سفارشی مفید هستند. برای مثال، ممکن است بخواهید استثناء مربوط به رفتار خاص خطاساز را شرح دهید یا مشکل یک

پارامتر را مشخص کنید که شرایطی را رعایت نکرده است. در کل، استثناءهای سیستمی خاص در دسترس هستند. اگر کافی نیستند، می‌توانید کلاس‌های خاص خودتان را ایجاد کنید.

در مثال ۱۹-۲ اگر شی هر دو واسط مورد نیاز را پیاده‌سازی نکند، متد یک استثناء سفارشی تولید می‌کند. استثناء NoDescException یک پیام بر می‌گرداند که خطا و نام شی خطاساز را شرح می‌دهد.

مثال ۱۹-۲

```
// Custom Exception Class
[Serializable]
public class NoDescException : ApplicationException
{ // Three constructors should be implemented
public NoDescException(){}
public NoDescException(string message):base(message){}
public NoDescException(string message, Exception innerEx)
:base(message, innerEx){ }
}
// Interfaces that shape objects are to implement
public interface IShapeFunction
{ double GetArea(); }
public interface IShapeDescription
{ string ShowMe();}
// Circle and Rectangle classes are defined
class Circle : IShapeFunction
{
private double radius;
public Circle (double rad)
{
radius= rad;
}
// Methods to implement both interfaces
public double GetArea()
{ return (۳,۱۴*radius*radius); }
}
class Rectangle : IShapeFunction, IShapeDescription
{
private int width, height;
public Rectangle(int w, int h)
{
width= w;
height=h;
}
// Methods to implement both interfaces
public double GetArea()
{ return (height*width); }
public string ShowMe()
{ return("rectangle"); }
}
public class ObjAreas
{
public static void ShowAreas(object ObjShape)
{
// Check to see if interfaces are implemented
if (!(ObjShape is IShapeDescription &&
ObjShape is IShapeFunction) )
{
// Throw custom exception
string objName = ObjShape.ToString();
throw new NoDescException
("Interface not implemented for "+objName);
}
// Continue processing since interfaces exist
IShapeFunction myShape = (IShapeFunction)ObjShape;
```

```
IShapeDescription myDesc = (IShapeDescription) ObjShape;
string desc = myDesc.ShowMe();
Console.WriteLine(desc+" Area= "+
myShape.GetArea().ToString());
}
}
```

برای دیدن عملی استثناء سفارشی، دو شی shape ایجاد کنید و آنها را به عنوان پارامتر به متد ایستای `ObjAreas.ShowAreas` ارسال کنید.

```
static ObjAreas.ShowAreas method.
Circle myCircle = new Circle(۴,۰);
Rectangle myRect = new Rectangle(۵,۲);
```

```
try
{
ObjAreas.ShowAreas(myRect);
ObjAreas.ShowAreas(myCircle);
}
catch (NoDescException ex)
{
Console.WriteLine(ex.Message);
}
```

متد `ShowAreas` بررسی می‌کند تا مطمئن شود شی مربوط به آن، هر دو واسط را پیاده‌سازی می‌کند. در غیر اینصورت، آن یک نمونه از استثناء `NoDescException` پاس می‌دهد و کنترل را به کد فراخوانی کننده رد می‌کند. در این مثال، شی `Circle` فقط یک واسط پیاده‌سازی می‌کند، در نتیجه یک استثناء رخ می‌دهد.

به طراحی `NoDescException` توجه کنید. آن یک مدل مفید است که قوانین پیاده‌سازی یک نوع استثناء سفارشی را نشان می‌دهد:

- باید کلاس استثناء از `ApplicationException` مشتق شود.
- بر اساس قرارداد، در انتهای نام استثناء کلمه‌ی `Exception` باشد. کلاس پایه `Exception` سه سازنده عمومی تعریف می‌کند که باید در کلاس شی باشد.

۱- یک سازنده با پراپرتی خالی که به عنوان پیش فرض است.

۲- سازنده‌ای با یک پارامتر، رشته‌ای که معمولاً نام `Message` دارد.

۳- سازنده‌ای با یک پارامتر رشته‌ای و یک پارامتر از نوع `Exception`، زمانی که یک استثناء در حین اداره کردن استثناء دیگر رخ دهد بکار می‌رود.

+ مقدار دهنده اولیه `base` را در ایجاد شی بکار برید. اگر می‌خواهید خصوصیات یا فیلدهایی را اضافه کنید، یک سازنده جدید برای این مقادیر ایجاد کنید.

+ صفت `serializable` برای تعیین ترتیبی بودن استثناء است. یعنی آن می‌تواند بصورت `xml` برای اهداف ذخیره و انتقال نمایش داده شود. می‌توان از آن صرف‌نظر کرد، چون فقط زمانی لازم است که یک استثناء از کد مربوط به دامنه کاربردی دیگر پاس داده شود.

۱۹-۵-استثنای‌های اداره نشده

زمانی که CLR نتواند بلوک catch مرتبط با یک استثناء را پیدا کند، استثنای‌های اداره نشده رخ می‌دهد. نتیجه پیش فرض اینکه، CLR آن را با متدهای خود اداره می‌کند.

اگرچه آن یک هشدار برای کاربر یا توسعه دهنده فراهم می‌کند، ولی هیچ راهی برای برخورد با آن توصیه نمی‌کند. راه حل مسئله این است که از مزیت اداره کننده‌های رویداد استثناء اداره نشده NET، برای جمع کردن همه آنها به کلاس اداره کننده استثناء سفارشی خود استفاده کنید.

کلاس سفارشی یک روش مطلوب برای بنا کردن یک سیاست برخورد با استثنای‌های اداره نشده فراهم می‌کند.

دستکاری متن در C#

آنچه که در این فصل یاد خواهید گرفت:

- کاراکترها و یونیکد^۱: به طور پیش فرض، NET یک کاراکتر را به صورت یک کاراکتر یونیکد ۱۶ بیتی ذخیره می کند. این عمل برنامه های کاربردی را برای پشتیبانی مجموعه کاراکترهای بین المللی پشتیبانی می کند. این تکنیک با عنوان محلی کردن^۲ بیان می شود.
- مروری بر String: در NET رشته ها تغییرناپذیر هستند. برای کاربرد مؤثر رشته ها، فهم معنی آنها و اینکه چگونه تغییرناپذیری، عملیات رشته ها را تحت تأثیر قرار می دهد، ضروری است.
- عملیات String: علاوه بر عملیات پایه ای رشته، NET تکنیک های فرمت دهی اعداد و تاریخ را پشتیبانی می کند.
- StringBuilder: این کلاس یک روش مؤثر برای الحاق رشته ها فراهم می سازد.
- عبارات منظم: کلاس Regex در NET، یک موتور برای پارس کردن، تطابق و استخراج مقادیر از یک رشته بکار می برد.

این فصل توانایی های اداره کردن رشته بوسیله کلاس های NET را معرفی می کند. موضوعات این فصل شامل کاربرد متدهای پایه String جهت استخراج و دستکاری محتوای رشته است. کاربرد متد String.Format اعداد و تاریخ ها را در فرمت های خاص نشان می دهد و کاربرد عبارتهای منظم جهت انجام تطابق الگوی پیشرفته است. این فصل نحوه ی بهینه سازی کاربرد رشته های حرفی در کامپایلر JIT را در بر دارد و غیره.

۲۰-۱- کاراکترها و یونیکد

یکی از رویدادهای مهم در محاسبات، معرفی مجموعه کاراکتر هفت بیتی اسکی به عنوان شمای کدگذاری استاندارد جهت تعیین منحصر به فرد کاراکترهای الفبایی و سمبل های مشترک دیگر است. آن بر اساس الفبای لاتین بوده و ۱۲۸ کاراکتر را شامل است. استاندارد ANSI تعداد کاراکترها را دو برابر کرده و سمبل های واحد پول و الفبای اروپایی را به آن اضافه کرد. چون آن بر اساس کاراکترهای لاتین است، تعدادی شمای کدگذاری برای نمایش کاراکترهای غیر لاتین همچون عربی، فارسی و یونانی ایجاد شده است.

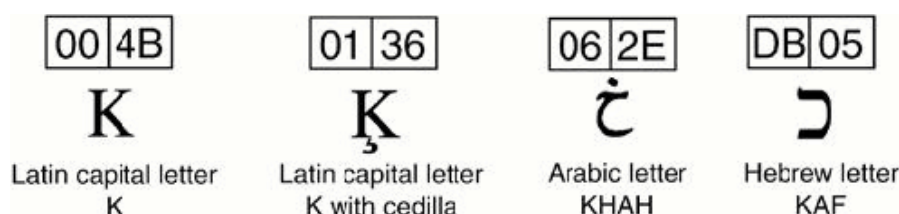
^۱ Unicode

^۲ Localization

ائتلاف بین‌المللی به دلیل نیاز به یک شمای کدگذاری جهانی، مشخصه‌ی یونیکد را پیشنهاد دادند. در حال حاضر آن یک استاندارد جهانی است که برای هر کاراکتر یک عدد منحصر به فرد تعریف می‌کند. نوع PlatForm، نوع برنامه و نوع زبان مهم نیست.

۲۰-۱-۱-یونیکد

NET. به طور کامل استاندارد یونیکد را پشتیبانی می‌کند. نمایش داخلی یک کاراکتر، یک عدد صحیح بدون علامت ۱۶ بیتی است، که شمای کدگذاری یونیکد را پیروی می‌کند. یک کاراکتر دو بیتی می‌تواند ۶۵۵۳۵ مقدار مختلف ارائه دهد. شکل ۲۰-۱ نشان می‌دهد چرا دو بایت نیاز است.



شکل ۲۰-۱

کاراکتر حرف بزرگ سمت چپ، یک عضو از مجموعه کاراکتر پایه لاتین است که ۱۲۸ کاراکتر اسکی اصلی را در بر می‌گیرد. مقدار دهدهی ۷۵ می‌تواند در یک بایت قرار گیرد. بیت‌های بی‌نیاز صفر قرار داده می‌شوند. با این وجود سه کاراکتر بعدی مقادیری در محدوده ۳۱۰ تا ۵۶۶۰۹ دارند که حداقل به دو بایت حافظه نیاز دارند.

کاراکترهای یونیکد یک مشخصه‌ی منحصر به فرد متشکل از یک نام و مقدار دارند. نسخه جاری یونیکد ۹۶۳۸۲ کاراکتر تعریف می‌کند. این کاراکترها در بیشتر از ۱۳۰ مجموعه کاراکتر گروه‌بندی شده‌اند، که حروف الفبای زبان‌ها، سمبل‌های ریاضی، موزیک، OCR، اشکال جغرافیایی، القاء نابینایان و کاربردهای دیگر را شامل هستند.

چون ۱۶ بیت نمی‌تواند ۱۰۰۰۰۰ کاراکتر جهانی را پشتیبانی کند، برای بعضی از مجموعه کاراکترهای بایتهای بیشتری مورد نیاز است. راه حل یونیکد، مکانیزم تعریف یک کاراکتر با دو مجموعه ۱۶ بیتی است. این زوج کد، یک زوج جانشین^۱ نامیده می‌شوند. جانشین بالا و پایین یک کاراکتر انتزاعی ۳۲ بیتی را به یک ۱۶ بیتی نگاشت می‌کنند. این روش بیش از ۱۰۰۰۰۰ کاراکتر را پشتیبانی می‌کند. جانشین‌ها از مقادیری که در فضای بالای یونیکد هستند، ساخته می‌شوند و با کاراکترهای واقعی اشتباه نمی‌شوند.

به عنوان یک توسعه‌دهنده، می‌توانید جزئیات کاراکترهای ۱۶ یا ۳۲ بیتی را در نظر نگیرند، چون API و کلاس‌های NET. جزئیات اصلی نمایش کاراکترهای یونیکد را اداره می‌کنند. اما زمانیکه می‌خواهید یک رشته را به صورت بایت به بایت در یک جریان بنویسید. تشخیص جانشین‌ها ضروری است. بدین منظور، NET. یک شی خاص برای طی کردن بایت‌ها فراهم می‌کند.

نکته: اگر کامپیوتر شما یک فونت برای پشتیبانی کاراکترهای یونیکد را داشته باشد، می‌تواند آنها را نمایش دهید. در سیستم عامل ویندوز می‌توانید با استفاده از `ttfext.exe` یک فونت مربوط به یونیکد را نصب کنید. برنامه‌های کنسولی نمی‌توانند کاراکترهای یونیکد را چاپ کنند، چون خروجی کنسول همواره در یک سبک حروف نامتناسب نمایش داده می‌شود.

^۱ Surrogate

۲۰-۱-۲ کار با کاراکترها

در .NET یک کاراکتر واحد به صورت یک ساختار char یا کلاس Char نمایش داده می‌شود. ساختار char تعداد کمی عضو برای تبدیل کردن و بازرسی مقدار آنها تعریف می‌کند. اکنون مروری گذرا بر بعضی از عملیات استاندارد کاراکتر داریم.

انتساب یک مقدار به یک نوع داده‌ی char

واضح‌ترین روش انتساب یک مقدار به یک متغیر char، با یک مقدار حرفی است. با این وجود، چون یک مقدار char در درون به صورت یک عدد نمایش داده می‌شود، می‌توانید یک مقدار عددی نیز به آن انتساب دهید. اینها مثال‌هایی هستند.

```
string klm = "KLM";
byte b = 75;
char k;
// Different ways to assign 'K' to variable K
k = 'K';
k = klm[0]; // Assign "K" from first value in klm
k = (char) 75; // Cast decimal
k = (char) b; // cast byte
k = Convert.ToChar(75); // Converts value to a char
```

تبدیل یک مقدار Char به یک مقدار عددی

زمانی که یک کاراکتر به یک عدد تبدیل می‌شود، نتیجه‌ی آن مقدار یونیکد کاراکتر است. قالب‌بندی، مؤثرترین روش انجام این کار است، اگرچه متد Convert می‌تواند به کار برده شود. در حالت خاصی که char یک رقم می‌باشد و شما می‌خواهید یک مقدار وابسته به زبان‌شناسی^۱ انتساب دهید، می‌توانید از متد ایستایی GetNumericValue استفاده کنید.

```
// '7' has Unicode value of 55
char k = '7';
int n = (int) k; // n = 55
n = (int) char.GetNumericValue(k); // n = 7
```

۲۰-۱-۳ کاراکترها و محلی کردن

یکی از مهمترین ویژگیهای .NET، توانایی تشخیص اتوماتیک و اعمال قوانین فرهنگ خاص یک زبان یا یک کشور به یک برنامه‌ی کاربردی است. این پروسه به محلی کردن معروف است، که نحوه‌ی فرمت‌دهی اعداد، تاریخ، نمایش سمبل‌های واحد پول یا انجام عملیات مقایسه‌ی رشته‌ها را مشخص می‌کند. در مواقع عملی، محلی کردن بدین معنی است که یک برنامه به یک کاربر در پاریس تاریخ را به صورت ۲۰۰۴ و ۹May نشان می‌دهد و در تگزاس به صورت ۲۰۰۴/۹/۵ نشان می‌دهد. CLR به طور اتوماتیک فرهنگ کامپیوتر محلی را تشخیص داده و تنظیم‌ها را انجام می‌دهد.

چارچوب .NET بیش از یکصد نام و مشخصه فرهنگ فراهم می‌کند که کلاس CultureInfo جهت معین کردن زبان / کشور استفاده می‌شود تا در عملیات حساس به فرهنگ در یک برنامه بکار برده شود. اگرچه در زمان کار با رشته‌ها، محلی کردن اثر بیشتری دارد، در این مثال متد Char.ToUpper یک روش مفید برای نمایش این مفهوم است.

```
// Include the System.Globalization namespace
// Using CultureInfo - Azerbaijan
char i = 'i';
// Second parameter is false to use default culture settings
// associated with selected culture
CultureInfo myCI = new CultureInfo("az", false );
i = Char.ToUpper(i, myCI);
```

^۱ Linguistic

یک OverLoad از متد ToUpper یک شی CultureInfo برای تعیین فرهنگ می‌گیرد، که در اجرای متد استفاده می‌شود. در این مثال، از اختصار زبان آذری کشور آذربایجان است. زمانی که CLR پارامتر CultureInfo را می‌بیند، آن هر جنبه از فرهنگ که ممکن است عملیات را تحت تأثیر قرار دهد، در نظر می‌گیرد. زمانی که هیچ پارامتری فراهم نشود، CLR فرهنگ پیش فرض سیستم را در نظر می‌گیرد.

در سیستم عامل ویندوز، چارچوب .NET اطلاعات فرهنگ پیش‌فرض را از تنظیمات کشور و فرهنگ سیستم بدست می‌آورد. آن این مقادیر را به خصوصیت Thread.CurrentThread.CurrentCulture انتساب می‌دهد. می‌توانید این گزینه را با استفاده از Regional Options در Control Panel انتخاب کنید.

چرا آذربایجان یک کشور کوچک بالای دریای خزر را برای ارائه محلی کردن انتخاب کردیم؟ بین همه کشورهای دنیا که مجموعه کاراکتر لاتین را به کار می‌برند، فقط آذربایجان و ترکیه حرف بزرگ i را با U+۰۰۴۹I نشان نمی‌دهند. بلکه با یک I که در بالای آن یک نقطه است (U+۰۱۳۰I) نشان می‌دهند. برای اطمینان از اینکه متد ToUpper() این عمل را درست انجام می‌دهد، ما باید یک نمونه از کلاس CultureInfo با نام و فرهنگ آذری (az) ایجاد کرده و به متد مورد نظر رد کنیم. این عمل کاراکتر یونیکد را درست نتیجه می‌دهد و ۸,۳ میلیون آذربایجانی را راضی نگه می‌دارد.

۲۰-۱-۴- کاراکترها و دسته‌های یونیکد آنها

استاندارد یونیکد، کاراکترها را به یکی از ۳۰ دسته تقسیم‌بندی می‌کند. NET یک نوع شمارشی UnicodeCategory را برای نمایش این دسته‌ها فراهم می‌کند و متد Char.GetUnicodeCategory() دسته‌ی کاراکتر را بر می‌گرداند. در اینجا یک مثال آمده است:

```
Char k = 'K';
int iCat = (int) char.GetUnicodeCategory(k); // 0
Console.WriteLine(char.GetUnicodeCategory(k)); // UppercaseLetter
char cr = (Char)13;
iCat = (int) char.GetUnicodeCategory(cr); // 14
Console.WriteLine(char.GetUnicodeCategory(cr)); // Control
```

این متد K را به طور صحیح به عنوان یک UppercaseLetter معین می‌کند و کاراکتر برگشت را به صورت یک Control بر می‌گرداند. char یک مجموعه از متدهای ایستا را به صورت یک میانبر برای تعیین دسته‌ی یونیکد کاراکتر دارد. آنها بر اساس فراخوانی GetUnicodeCategory یک مقدار true یا false بر می‌گردانند. جدول ۱-۲۰ این متدها را لیست می‌کند.

جدول ۱-۵

متد	دسته یونیکد	توصیف
IsControl	۴	کد این کاراکترها U+۰۰۷F یا در محدوده‌ی U+۰۰۰۰ تا F۰۰۱U یا U+۰۰۰۸ تا U+۰۰۹F است.
IsDigit	۸	در محدوده‌ی ۰ تا ۹
IsLetter	۰, ۱, ۲, ۴	حروف
IsLetterorDigit	۰, ۱, ۸	اجتماع حروف و ارقام
IsLower	۱	حروف کوچک

حروف بزرگ	IsUpper	۰
سمبل تأکید مثال	IsPunctuation	۱۸, ۱۹, ۲۰, ۲۱, ۲۲
DashPunctuation (19)		۲۳, ۲۴
OpenPunctuation (20),		
OtherPunctuation (24).		
جدا کننده فاصله، جداکننده خط،	IsSeparator	۱۱, ۱۲, ۱۳
جدا کننده پاراگراف		
مقدار آن یک جانشین بالا یا پایین است.	IsSurrogate	۱۶
سمبل	IsSymbol	۲۵, ۲۶, ۲۸
فضاهای خالی می‌توانند هر کدام یک از این کاراکترها باشند. ۲۰x۰	IsWhiteSpace	۱۱
(فضای خالی) D۰x۰ (کاراکتر برگشت)		
(خط جدید) A۰x۰ (افقی) ۰۹x۰ (tab)		
(فرم جدید) C۰x۰ (عمودی) B۰x۰ (tab)		

کاربرد این متدها سر راست است. نکته‌ی جالب اینکه، آنها دو تا OverLoad دارند که یکی از آنها یک کاراکتر واحد را به عنوان پارامتر می‌گیرد و دیگری دو پارامتر یکی رشته و دیگری اندیس کاراکتر را می‌گیرد.

```
Console.WriteLine(Char.IsSymbol('+')); // true
Console.WriteLine(Char.IsPunctuation('+')): // false
string str = "black magic";
Console.WriteLine(Char.IsWhiteSpace(str, 5)); // true
char p = '.';
Console.WriteLine(Char.IsPunctuation(p)); // true
Int iCat = (int) char.GetUnicodeCategory(p); // 24
Char p = '(';
Console.WriteLine(Char.IsPunctuation(p)); // true
int iCat = (int) char.GetUnicodeCategory(p); // 20
```

۲۰-۲-کلاس رشته

کلاس رشته قبلاً استفاده شده است. این فصل نگاهی دقیق روی ایجاد، مقایسه و فرمت‌دهی رشته‌ها دارد.

۲۰-۲-۱-ایجاد رشته‌ها

یک رشته با اعلان یک متغیر از نوع string و انتساب یک مقدار به آن ایجاد می‌شود. مقدار موردنظر ممکن است یک رشته‌ی حرفی یا رشته‌ی ایجاد شده با عمل الحاق باشد. این یک پروسه‌ی سرسری است و اغلب برنامه‌نویسان در زمان بهبود کارایی کد به آن توجهی ندارند. در .NET یک فهم از نحوه‌ی اداره کردن رشته‌های حرفی می‌تواند در بهبود کارایی آن به توسعه‌دهنده کمک کند.

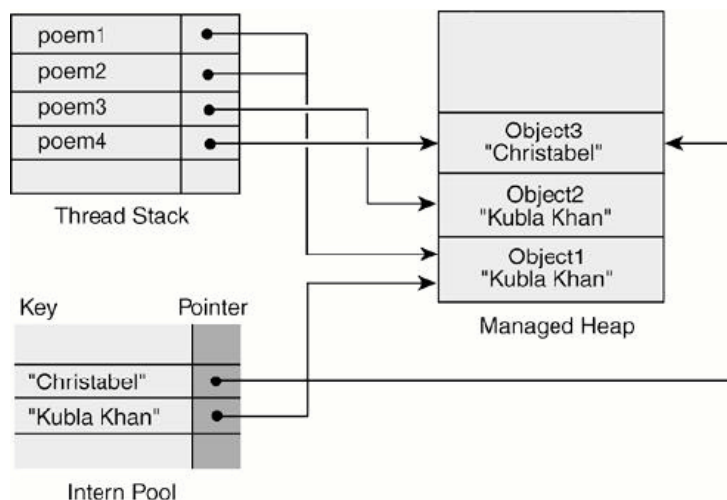
۲۰-۲-۲-داخل کردن^۱ رشته‌ها

قبلاً در مورد نحوه ذخیره‌ی انواع داده مقداری و ارجاعی در حافظه بحث شده است. بیاد دارید که انواع داده‌ی مقداری روی یک Stack ذخیره می‌شوند. در حالیکه انواع داده‌ی ارجاعی روی یک Heap مدیریت شده جای می‌گیرند. همچنین CLR یک ناحیه‌ی سومی در حافظه به نام استخر داخلی^۲ کنار می‌گذارد که در طول کامپایل، همه رشته‌های حرفی در آن ذخیره می‌شوند. با استفاده از این استخر تکثیر کپی مقادیر رشته‌ای ذخیره شده حذف می‌گردد. کد زیر را بررسی کنید.

```
string poem1 = "Kubla Khan";
string poem2 = "Kubla Khan";
string poem3 = String.Copy(poem2); // Create new string object
string poem4 = "Christabel";
```

شکل ۲۰-۲ یک دید ساده از نحوه‌ی ذخیره رشته‌ها و مقادیر آنها در حافظه را نشان می‌دهد.

شکل ۲۰-۲



استخر داخلی به صورت یک جدول hash پیاده‌سازی می‌شود. کلید جدول hash رشته‌ی مورد نظر است و اشاره‌گر آن به شی رشته‌ی انتساب یافته در روی heap مدیریت شده ارجاع می‌کند. زمانی که کامپایلر JIT، کد قبلی را کامپایل می‌کند، آن اولین نمونه از "Kubla Khan" را در استخر قرار می‌دهد و یک ارجاع به شی رشته در روی heap مدیریت شده ایجاد می‌کند. زمانی که آن با ارجاع دوم به "Kubla Khan" (poem^۲) روبرو می‌شود، CLR می‌بیند این رشته در حافظه وجود دارد، بدین دلیل به جای ایجاد یک رشته‌ی جدید، poem^۲ را به همان شی (poem^۱) ارجاع می‌دهد. این پروسه به داخل کردن رشته‌ها معروف است. در ادامه‌ی مثال، متد String.Copy یک رشته‌ی جدید به نام Poem^۳ ایجاد می‌کند. در روی heap مدیریت شده یک شی برای آن ایجاد می‌کند. سرانجام، رشته‌ی حرفی انتساب داده شده به poem^۴ به استخر اضافه می‌شود.

برای بررسی اثرات عملی داخل کردن رشته، اجازه دهید مثال قبلی را توسعه دهیم. کدی اضافه می‌کنیم که عملگر تساوی (==) را برای مقایسه مقادیر رشته‌ای به کار می‌برد و متد Object.ReferenceEqual را برای مقایسه آدرس آنها اضافه می‌کند.

```
Console.WriteLine(poem1 == poem2); // true
Console.WriteLine(poem1 == poem3); // true
Console.WriteLine(ReferenceEquals(poem1, poem3)); // false
Console.WriteLine(ReferenceEquals(poem1,
    "Kubla Khan")); // true
```

^۱ Interning

^۲ Intern Pool

دو دستور اول مقدار متغیرها را مقایسه می‌کنند و یک مقدار true بر می‌گردانند. دستور سوم موقعیت حافظه‌ی متغیرهای poem3 و poem2 را مقایسه می‌کند. چون آنها به اشیاء مختلفی در روی heap ارجاع دارند مقدار false برگردانده می‌شود. طراحان NET. تصمیم گرفتند مقادیر پویای ایجاد شده را از استخر خارج کنند. چون بررسی استخر داخلی در زمان ایجاد هر رشته، روی کارایی اثر منفی دارد. با این وجود، متد String.Intern برای اضافه کردن یک رشته‌ی پویا به استخر حرفی استفاده می‌شود.

```
string khan = "Khan";
string poem5 = "Kubla" + khan;
Console.WriteLine(ReferenceEquals(poem5, poem1)); // false
// Place the contents of poem5 in the intern pool-if not there
poem5 = String.Intern(poem5);
Console.WriteLine(ReferenceEquals(poem5, poem1)); // true
```

متد String.Intern مقدار "Kubla Khan" را در استخر داخلی جستجو می‌کند. چون آن رشته در استخر وجود دارد، نیازی نیست اضافه گردد. متد یک ارجاع به شیء موجود بر می‌گرداند و آن را به poem5 انتساب می‌دهد. چون poem5 و poem1 به شیء یکسانی اشاره می‌کنند، عمل مقایسه در دستور آخری مقدار true است. توجه داشته باشید که شیء ایجاد شده برای poem5 آزاد می‌شود و در حین GC بعدی جاروب می‌شود.

برای استفاده از مزیت مقایسه بوسیله ارجاع، فقط در صورتی که رشته در چندین عمل مقایسه حضور داشته باشد، متد String.Intern را بکار ببرید.

۲۰-۲-۳-مروری بر عملیات رشته‌ها

کلاس System.String تعداد زیادی متد ایستا و نمونه فراهم می‌کند، که بیشتر آنها چندین شکل OverLoad دارند. این متدها بر اساس عملکرد اصلی آنها به ۴ دسته اصلی گروه‌بندی می‌شوند.

مقایسه رشته: متدهای String.Compare، String.Equals و String.CompareOrdinal روش‌های مختلفی برای مقایسه‌ی مقادیر رشته‌ای پیشنهاد می‌کنند. انتخاب یک روش به نوع مقایسه (ترتیبی یا حرفی) وابسته است و اینکه آیا حالت و فرهنگ عمل را در نظر بگیرد یا نه.

اندیس‌گذاری و جستجو: یک رشته آرایه‌ای از کاراکترهای یونیکد است که با طی کردن سراسر آرایه یا با استفاده از متدهای اندیس خاص می‌توان مقادیر رشته‌ای را تعیین محل کرد.

تبدیل رشته‌ها: این دسته از عملیات شامل درج، پیمایش، حذف، جایگذاری، حذف فضاهای خالی و جدا کردن رشته‌های کاراکتری است.

بیشتر متدهای رشته به فرهنگ وابسته هستند. هر جا که لازم باشد، نحوه تأثیر فرهنگ روی رفتار یک متد را نشان خواهیم داد.

۲۰-۳-مقایسه‌ی رشته‌ها

ساده‌ترین راه تعیین مساوی بودن دو رشته این است که ببینیم آیا آنها به آدرس حافظه یکسانی اشاره می‌کنند. ما این کار را با استفاده از متد ReferenceEquals انجام می‌دهیم. اگر دو رشته آدرس یکسانی را به اشتراک نگذارند، مقایسه کاراکتر به کاراکتر برای تعیین مساوی بودن آنها لازم است. این عمل بیشتر از قبلی طول می‌کشد، ولی نمی‌توان از آن چشم‌پوشی کرد.

NET. برای بهینه‌کردن پروسه‌ی مقایسه، متد Equals را فراهم کرد که هر دو نوع عمل مقایسه را به طور اتوماتیک انجام می‌دهد. می‌توانیم عمل آن را در شبه کد زیر شرح دهیم.

```
If string1 and string2 reference the same memory location
Then strings must be equal
Else
Compare strings character by character to determine equality
```

این قطعه کد شکل‌های ایستا و ارجاع متد Equals را نشان می‌دهد.

```
string poem1 = "Kubla Khan";
string poem2 = "Kubla Khan";
string poem3 = String.Copy(poem2);
string poem4 = "kubla khan";
//
Console.WriteLine(String.Equals(poem1,poem2)); // true
Console.WriteLine(poem1.Equals(poem3)); // true
Console.WriteLine(poem1 == poem3); // equivalent to Equals
Console.WriteLine(poem1 == poem4); // false - case differs
```

توجه کنید که عملگر ==، (که متد Equals را فراخوانی می‌کند) روش مطلوب‌تر بیان عمل مقایسه است. اگرچه متد Equals بیشتر نیازهای مقایسه را برآورده می‌سازد، آن هیچ OverLoadی برای گرفتن فرهنگ حساب کاربری و حساسیت حروف را ندارد. متد Compare کلاس String این محدودیت‌ها را رفع می‌کند.

۲۰-۳-۱- کاربرد String.Compare

String.Compare یک متد انعطاف‌پذیر در عمل مقایسه است که می‌تواند با در نظر گرفتن فرهنگ و حالت حروف بکار برده شود. آن OverLoad های زیادی برای گرفتن پارامترهای فرهنگ و حالت حروف دارد.

گرامر:

```
int Compare (string str1, string str2)
Compare (string str1, string str2, bool IgnoreCase)
Compare (string str1, string str2, bool IgnoreCase,
CultureInfo ci)
Compare (string str1, int index1, string str2, int index2,
int len)
```

پارامترها

Str۱، Str۲: رشته‌های مورد نظر جهت عمل مقایسه.

IgnoreCase: برای در نظر نگرفتن حساسیت حروف باید true باشد. به طور پیش‌فرض false است.

index۱، index۲: محل شروع در Str۱ و Str۲

C۲: یک شیء CultureInfo که فرهنگ مورد استفاده را نشان می‌دهد.

متد Compare () یک مقدار صحیح بر می‌گرداند که نتیجه‌ی مقایسه را نشان می‌دهد. اگر دو رشته مساوی باشند، یک مقدار صفر برگردانده می‌شود. اگر رشته‌ی اول کوچکتر از رشته‌ی دوم باشد، یک مقدار منفی برگردانده می‌شود. اگر رشته اول بزرگتر از رشته‌ی دوم باشد، یک مقدار مثبت برگردانده می‌شود.

قطعه کد زیر نحوه‌ی استفاده از دستور Compare را برای در نظر گرفتن فرهنگ و حساسیت حروف نشان می‌دهد.

```
int result;
string stringUpper = "AUTUMN";
string stringLower = "autumn";
// (1) Lexical comparison: "A" is greater than "a"
result = string.Compare(stringUpper,stringLower); // 1
// (2) IgnoreCase set to false
result = string.Compare(stringUpper,stringLower,false); // 1
// (3) Perform case-insensitive comparison
result = string.Compare(stringUpper,stringLower,true); // 0
```

شاید اثر نهایی اطلاعات فرهنگ روی عمل مقایسه مهمتر از حالت حروف باشد. NET لیستی از قوانین مقایسه برای هر فرهنگ دارد. زمانی که متد Compare اجرا می‌شود، CLR فرهنگ اختصاص داده شده را بررسی می‌کند و قوانین آن را اعمال می‌کند. نتیجه اینکه، ممکن است دو رشته روی یک کامپیوتر با فرهنگ US یا یک فرهنگ ژاپنی به طور مختلف مقایسه شوند. این حالت‌ها زمانی مهم است که بخواهید فرهنگ جاری، بقیه فرهنگ‌ها را باطل کند تا همه کاربران از رفتار یکسان برنامه مطمئن شوند. برای مثال، بسیار مهم است که یک عمل مرتب‌سازی در هر جایی که اجرا می‌شود، درست و دقیق اجرا گردد.

به طور پیش‌فرض متد Compare اطلاعات فرهنگ را بر اساس مقدار خصوصیت Thread.CurrentThread.CurrentCulture بکار می‌برد. برای باطل کردن مقدار پیش‌فرض، یک شی CultureInfo به عنوان پارامتر متد تهیه کنید. دستور زیر نحوه‌ی ایجاد یک شی برای ارائه زبان و کشور Germany نشان می‌دهد.

```
CultureInfo ci = new CultureInfo("de-DE"); // German culture
```

برای تعیین صریح یک فرهنگ پیش‌فرض یا هیچ فرهنگی، کلاس CultureInfo دو خصوصیت دارد که به عنوان پارامتر می‌تواند بگیرد: CurrentCulture برای استفاده کردن فرهنگ ریسمان جاری استفاده می‌شود و InvariantCulture که از هر فرهنگی چشم‌پوشی می‌کند.

مثل زیر نحوه تأثیر فرهنگ‌های مختلف روی نتیجه‌ی عمل Compare () را نشان می‌دهد.

```
using System.Globalization; // Required for CultureInfo
// Perform case-sensitive comparison for Czech culture
string s1 = "circle";
string s2 = "chair";
result = string.Compare(s1, s2,
    true, CultureInfo.CurrentCulture); // 1
result = string.Compare(s1, s2,
    true, CultureInfo.InvariantCulture); // 1
// Use the Czech culture
result = string.Compare(s1, s2,
    true, new CultureInfo("cs-CZ")); // -1
```

دو مقدار رشته‌ای "Circle" ، "Chair" با استفاده از سه حالت فرهنگ US، هیچ فرهنگی و فرهنگ چکوسلواکی مقایسه می‌شوند. دو مقایسه‌ی اول مقدار مثبت بر می‌گردانند، در حالی که در فرهنگ چکوسلواکی مقدار منفی بر می‌گرداند. چون در قوانین زبان چکوسلواکی "Ch" به صورت یک کاراکتر تکی در نظر گرفته می‌شود که بعد از کاراکتر C ظاهر می‌گردد.

در صورتی که می‌خواهید یک برنامه‌ای بنویسید که اطلاعات فرهنگ کاربر جاری را بکار برد، یک پارامتر CultureInfo صریح بکار برید تا در عملیات رشته‌ای مشکلی پیش نیاید.

۲۰-۳-۲ کاربرد String.CompareOrdinal

اگر می‌خواهید یک مقایسه براساس مقدار ترتیبی کاراکترها انجام دهید. String.CompareOrdinal را بکار برید. آن یک الگوریتم ساده برای مقایسه مقدار یونیکد در رشته بکار می‌برد. اگر دو رشته مساوی باشند، مقدار صفر بر می‌گرداند. در صورتی که رشته‌ی اول بزرگتر از رشته دوم باشد، مقدار کوچکتر از صفر بر می‌گرداند، در غیر اینصورت مقدار بزرگتر از صفر بر می‌گرداند. کد زیر تفاوت مابین Compare و CompareOrdinal را نشان می‌دهد.

```
string stringUpper = "AUTUMN";
string stringLower = "autumn";
//
result = string.Compare(stringUpper, stringLower,
    false, CultureInfo.InvariantCulture); // 1
```

```
result = string.CompareOrdinal(stringUpper, stringLower); // -32
```

متد Compare یک مقایسه‌ی حرفی انجام می‌دهد که رشته حروف بزرگ از رشته حروف کوچک، بزرگتر است. CompareOrdinal مقادیر یونیکد را بررسی می‌کند. چون (U+۰۰۴۱A) از (U+۰۰۰۶۱a) کمتر است، رشته اول کوچکتر از دومی است.

۲۰-۴- جستجو، تغییر و کدگذاری محتوای یک رشته

این بخش متدهایی از String را شرح می‌دهد که نسبت به کارهای آشنا همچون محل‌یابی یک ریز رشته در رشته، تغییر حالت حروف یک رشته و جایگذاری و حذف زیر رشته و غیره متفاوت عمل می‌کنند.

۲۰-۴-۱- جستجوی محتویات یک رشته

رشته یک آرایه‌ای از کاراکترها است و با استفاده از گرامر آرایه [String[n]] می‌تواند جستجو شود، که n موقعیت یک کاراکتر در رشته است. برای محل‌یابی یک زیررشته‌ی یک یا چند کاراکتری در یک رشته، کلاس String متدهای IndexOf و IndexOfAny را پیشنهاد می‌کند. جدول ۲۰-۲ این متدها را خلاصه می‌کند.

جدول ۲۰-۲

عضو	توصیف
[n]	یک کاراکتر ۱۶ بیتی قرار گرفته در موقعیت n از رشته را فهرست می‌کند.
IndexOf / LastIndexOf String, (int start,) ([int count	اندیس اولین یا آخرین وقوع یک زیررشته در رشته را بر می‌گرداند. اگر تطابقی رخ ندهد، -۱ بر می‌گرداند.
Cont تعداد کاراکترهای مورد بررسی است.	<pre>String Poem="kubla khan Int n=Poem.Index of ("la");//۳ n=Poem.Index of ("h");// ۶);//۴n=Poem Index of ('k',</pre>
IndexOfAny/LastIndex OfAny	اندیس اولین / آخرین کاراکتر را در یک آرایه از کاراکترهای یونیکد بر می‌گرداند.
	<pre>String Poem="kubla khan [Char[] Vowels = new char [۵ {'', 'u', '۲a', 'e', '۴} N=Poem. Index of on (Vowels); //۱ N=Poem. Last Index of Any (Vowels);//۸ N=Poem. Index of Any (Vowels, ۲);//۴</pre>

۲۰-۴-۲- جستجوی رشته‌ی جانشین‌دار

همه این تکنیک‌ها فرض می‌کنند هر رشته یک دنباله از کاراکترهای ۱۶ بیتی را در بر دارد. فرض کنید برنامه کاربردی شما با مجموعه کاراکتر ۳۲ بیتی خاور دور کار می‌کند. اینها در حافظه به صورت یک زوج جانشین شامل یک مقدار ۱۶ بیتی بالا و پایین نمایش داده می‌شوند. برای عبارتی همچون [Poem[ndx یک مشکل پیش می‌آید که فقط نصف یک زوج جانشین را بر می‌گرداند.

در برنامه‌هایی که باید از این جانشین‌ها استفاده کنند، کلاس String در .NET با همه کاراکترها به صورت عناصر متنی برخورد می‌کند و می‌تواند به طور اتوماتیک یک کاراکتر ۱۶ بیتی را از یک کاراکتر جانشین تشخیص دهد. مهمترین عضو آن متد GetEnumerator است که یک نوع شمارشی بر می‌گرداند. این نوع شمارشی می‌تواند برای طی کردن سراسر عناصر متنی در یک رشته استفاده شود.

```
TextElementEnumerator tEnum =
StringInfo.GetTextElementEnumerator(poem) ;
while (tEnum.MoveNext()) // Step through the string
{
Console.WriteLine(tEnum.Current); // Print current char
}
```

به خاطر دارید که در انواع داده شمارشی، متدهای MoveNext() و Current پیاده‌سازی می‌شوند.

۲۰-۴-۳- تبدیل رشته‌ها

جدول ۲۰-۳ مهمترین متدهای کلاس String را برای تغییر دادن یک رشته خلاصه می‌کند. چون رشته‌ی اصلی غیر قابل تغییر است، این متدها رشته‌های جدیدی ایجاد می‌کنند که حافظه خاص خود را دارند.

جدول ۲۰-۳

متد	توصیف
(Insert (int, string)	یک رشته را در محل مشخص شده درج می‌کند.
PadLeft/PadRight	در سمت چپ یا راست یک رشته کاراکتر خاصی را تکرار می‌کند تا طول رشته به مقدار معینی برسد. اگر کاراکتری مشخص نشود، کاراکتر خالی اضافه می‌گردد.
(Remove (p, n	تعداد n کاراکتر را با شروع از محل p حذف می‌کند.
(Replace (A, B	همه‌ی زیررشته‌های A را با B جایگزین می‌کند. A و B کاراکتر یا رشته هستند.
	String astring = "nap ace sap. Path";

```
String istring=astring.Replace ('a','۱');
//istring--> "nip ice sip pith"
```

در آرایه‌ی Char لیست جدا کننده‌ها وجود دارد که یک رشته را به چندین زیررشته تقسیم می‌کنند و نتیجه در یک آرایه‌ی رشته‌ای بر گردانده می‌شود.

([])Split(Char

```
string words = "red,blue orange ";
string [] split = words.Split(new Char []
{' ', ','});
Console.WriteLine(split[2]); // orange
```

یک کپی از رشته با حروف بزرگ یا کوچک بر می‌گردانند.

()Toupper

```
String Poem۲= "kubla khan";
```

(Toupper(CultureInfo

```
Poem۲=Poem۲. Toupper (Culture Info. Invarian+Culture);
```

()Tolower

(Tolower(CultureInfo

فضاهای خالی ابتدا و انتهای رشته را حذف می‌کند. اگر آرایه‌ی Char خالی نباشد، همه کاراکترهای این آرایه در ابتدا و انتهای رشته حذف می‌شوند.

([])Trim (params Char

()Trim

```
String name=" Samuel Coleridge";
Nam=name.Trim( );
```

همه کاراکترهای مشخص شده آرایه‌ی Char را از ابتدا یا انتهای رشته حذف می‌کنند. اگر مقدار آرایه null باشد، فضاهای خالی حذف خواهند شد.

([])TrimEnd (params Char
TrimStart(params Char
([]

```
String name="Samuel Coleridge";
Trino Name=nam. Trimstart(null);
Shortname=name. Trim END('e','g','I');
//Short Name → "Samuel Colerid";
```

یک زیر رشته به طور یک را با شروع از موقعیت n استخراج می‌کند.

(SubString (n

اگر L مشخص شده باشد طول این زیر رشته L خواهد بود.

(Substring (n,l

```
String title = "kubla khan";
Console. Write line (title. Substring (۲,۳)); //bla
```

کاراکترهای یک رشته را استخراج کرده و در یک آرایه از کاراکترهای یونیکد قرار می‌دهد.

()ToCharArray

(ToCharArray(n,l

```
String my Vowels= "aeiou";
Char{ } Vowel Arr;
Vowel Arr=myVowels. Tochar Array( );
Console write line (Vowel Arr [۱]);
```

بیشتر این متدها شبیه زبانه‌های دیگر هستند و همانند انتظار شما رفتار می‌کنند. مورد جالب این است که بیشتر این متدها در کلاس StringBuilder نیستند. فقط Insert و Remove, Replace وجود دارد.

۲۰-۴-۴-کدگذاری رشته

زمانی که لازم باشد رشته‌ها و بایت‌ها برای عملیاتی همچون نوشتن در یک فایل یا جاری شدن روی شبکه تبدیل گردند، از کدگذاری رشته استفاده می‌کنند. کدگذاری و کدگشایی کاراکترها دو فایده‌ی اصلی پیشنهاد می‌کنند: کارایی و قابلیت ارتباط بین پروسه‌ها. بیشتر کاراکترهای متون انگلیسی می‌توانند با ۸ بیت نمایش داده شوند. با استفاده از کدگذاری می‌توان بایت اضافی را در انتقال و ذخیره‌سازی حذف کرد. انعطاف‌پذیری و رمزگذاری، ارتباط یک برنامه با داده‌های موروثی^۱ یا داده‌های ثالث^۲ با فرمت‌های مختلف را ممکن می‌سازد.

چارچوب .NET شکل‌های زیادی از کدگذاری و کدگشایی کاراکترها را پشتیبانی می‌کند. معمول‌ترین آنها به صورت زیر هستند.

UTF-۸

هر کاراکتر بر اساس مقدار اصلی آن با یک تا ۴ بایت کدگذاری می‌شود. کاراکترهای سازگار با اسکی در یک بایت ذخیره می‌شوند. کاراکترهای مابین ۰۸۰۰X تا FF۰۷X در دو بایت ذخیره می‌شوند و کاراکترهایی که یک مقدار بزرگتر یا مساوی ۰۸۰۰X دارند به سه بایت تبدیل می‌شوند و جانشین‌ها به صورت ۴ بیتی نوشته می‌شوند. زمانی که کدگذاری مد نظر نباشد، کلاس‌های .NET به طور پیش‌فرض UTF۸ را بکار می‌برند.

UTF۱۶

هر کاراکتر با ۲ بایت کدگذاری می‌شوند (به استثناء جانشین‌ها). این کدگذاری را کدگذاری یونیکد نیز گویند.

ASCII

هر کاراکتر به صورت یک کاراکتر ۸ بیتی اسکی کدگذاری می‌شود. زمانی که همه کاراکترها مابین ۰۰OX تا f۷OX هستند از این کد استفاده کنید.

کدگذاری و کدگشایی در فضای نامی System.Text با کلاس Encoding انجام می‌شوند. این کلاس انتزاعی چندین خصوصیت ایستا دارد که برای پیاده‌سازی تکنیک کدگذاری خاص، یک شی بر می‌گردانند. این خصوصیات ASCII، UTF۸ و Unicode هستند. آخری برای کدگذاری UTF۱۶ است.

یک شی کدگذاری چندین متد با overloadهای مختلف جهت تبدیل کاراکترها و بایت‌ها پیشنهاد می‌کند. در این مثال دو متد بسیار مفید ارائه می‌گردد: GetBytes یک رشته متنی را به بایت‌ها تبدیل می‌کند و GetString عکس این عمل را انجام می‌دهد.

```
string text= "In Xanadu did Kubla Khan";
Encoding UTF8Encoder = Encoding.UTF8;
byte[] textChars = UTF8Encoder.GetBytes(text);
Console.WriteLine(textChars.Length); // 24
// Store using UTF-16
textChars = Encoding.Unicode.GetBytes(text);
Console.WriteLine(textChars.Length); // 48
// Treat characters as two bytes
string decodedText = Encoding.Unicode.GetString(textChars);
Console.WriteLine(decodedText); // "In Xanadu did ... "
```

^۱ Legacy

^۲ Third Party

همچنین می‌توانید اشیاء کدگذاری را به طور مستقیم تعریف کنید. در این مثال، شیء UTF-۸ می‌تواند به صورت زیر ایجاد شود.

```
UTF8Encoding UTF8Encoder = new UTF8Encoding();
```

سازنده‌ی این کلاس‌ها به استثناء ASCIIEncoding، پارامترهایی را برای کنترل بیشتر روی پروسه‌ی کدگذاری تعریف می‌کنند. به عنوان مثال، مشخص می‌کنند در صورت مواجه شدن با یک کاراکتر نامعتبر، یک استثناء رها شود.

۲۰-۵-کلاس StringBuilder

عیب اصلی رشته‌ها این است که در هر زمان با تغییر محتویات یک متغیر رشته‌ای، باید به آن حافظه تخصیص یابد. فرض کنید یک حلقه ایجاد کردیم که ۱۰۰ بار تکرار می‌شود و یک کاراکتر را به یک رشته الحاق می‌کند. ما در حافظه ۱۰۰ تا رشته داریم که هر کدام با رشته‌ی قبلی فقط در یک کاراکتر اختلاف دارند.

کلاس StringBuilder با تخصیص یک ناحیه‌ی کاری (بافر) که متدهای آن می‌توانند روی رشته اعمال شوند، این مشکل را حل می‌نماید. این متدها راه‌هایی را برای الحاق کردن، درج کردن، پاک کردن، حذف کردن و جایگزین کردن کاراکترها دارند. بعد از اتمام عملیات، متد ToString() برای تبدیل بافر به یک رشته در متغیر رشته‌ای استفاده می‌شود.

قطعه کد ۲۰-۱ بعضی از متدهای StringBuilder را برای ایجاد یک لیست تفکیک شده با کاما بیان می‌کند.

قطعه کد ۲۰-۱

```
using System;
using System.Text;
public class MyApp
static void Main()
{
// Create comma delimited string with quotes around names
string namesF = "Jan Donna Kim ";
string namesM = "Rob James";
StringBuilder sbCSV = new StringBuilder();
sbCSV.Append(namesF).Append(namesM);
sbCSV.Replace(" ", "','');
// Insert quote at beginning and end of string
sbCSV.Insert(0,"'").Append("'");
string csv = sbCSV.ToString();
// csv = 'Jan','Donna','Kim','Rob','James'
}
}
```

همه عملیات در یک بافر واحد رخ می‌دهند و تا انتساب نهایی هیچ عمل تخصیص حافظه نیاز نیست. نگاه رسمی به این کلاس و اعضای آن می‌افکنیم.

۲۰-۵-۱-مروری بر کلاس StringBuilder

سازنده‌های کلاس StringBuilder یک مقدار رشته‌ای اولیه را همانند مقادیر صحیح می‌پذیرند تا مقدار فضای اولیه و حداکثر فضای تخصیص یافته به بافر را معین کنند.

```
// StringBuilder(initial value)
StringBuilder sb1 = new StringBuilder("abc");
// StringBuilder(initial value, initial capacity)
StringBuilder sb2 = new StringBuilder("abc", 16);
// StringBuiler(Initial Capacity, maximum capacity)
StringBuilder sb3 = new StringBuilder(32,128);
```

ایده‌ی StringBuilder امکان استفاده از آن به عنوان یک بافر برای انجام عملیات رشته‌ای است. در این مثال نحوه‌ی کار متدهای Append، Insert، Replace و Remove را می‌بینید.

```
int i = 4;
char[] ch = {'w','h','i','t','e'};
string myColor = " orange";
StringBuilder sb = new StringBuilder("red blue green");
sb.Insert(0, ch); // whitered blue green
sb.Insert(5, " "); // white red blue green
sb.Insert(0,i); // 4white red blue green
sb.Remove(11R); // 4 red blue green
sb.Append(myColor); // 4 red blue green orange
sb.Replace("blue","violet"); // 4 red violet green orange
string colors = sb.ToString();
```

۲۰-۵-۲- مقایسه‌ی StringBuilder و الحاق رشته

قطعه کد ۲۰-۲- بهره‌وری StringBuilder را در برابر عملگر الحاق آزمایش می‌کند. بخش اول این برنامه عملگر + را برای الحاق حرف 'a' به یک رشته در حلقه‌ای با ۵۰۰۰۰ تکرار بکار می‌برد. بخش دوم نیز همان کار را انجام می‌دهد. اما متد StringBuilder.Append را بکار می‌برد. Environment.TickCount، زمان شروع و پایان را در واحد میلی ثانیه تهیه می‌کند.

قطعه کد ۲۰-۲

```
using System.Text;
public class MyApp
static void Main()
{
    Console.WriteLine("String routine");
    string a = "a";
    string str = string.Empty;
    int istart, istop;
    istart = Environment.TickCount;
    Console.WriteLine("Start: "+istart);
    // Use regular C# concatenation operator
    for(int i=0; i<50000; i++)
    {
        str += a;
    }
    istop = Environment.TickCount;
    Console.WriteLine("Stop: "+istop);
    Console.WriteLine("Difference: " + (istop-istart));
    // Perform concatenation with StringBuilder
    Console.WriteLine("StringBuilder routine");
    StringBuilder builder = new StringBuilder();
    istart = Environment.TickCount;
    Console.WriteLine("Start: "+istart);
    for(int i=0; i<50000; i++)
    {
        builder.Append(a);
    }
    istop = Environment.TickCount;
    str = builder.ToString();
    Console.WriteLine("Stop: "+Environment.TickCount);
    Console.WriteLine("Difference: "+ (istop-istart));
}
}
```

اجرای این برنامه خروجی زیر را نتیجه می‌دهد.

```
String routine
Start: 1422091687
Stop: 1422100046
Difference: 9359
```

```
StringBuilder routine
Start: 1QOONMMMQS
Stop: 1422100062
Difference: 16
```

الحاق استاندارد مدت زمان ۹,۳۵۹ میلی ثانیه در برابر ۱۶ میلی ثانیه `StringBuilder` طول می کشد. `StringBuilder` هیچ مزیت مهمی ندارد، مگر اینکه در دستکاری پیشرفته متن به جای الحاق استاندارد استفاده شود.

۲۰-۶-فرمت دهی مقادیر عددی، تاریخ و زمان

متد `String.Format` وسیله‌ی اصلی فرمت دهی تاریخ و داده‌های عددی جهت نمایش است. آن یک رشته مرکب از متن و عناصر فرمت تعبیه شده در آن را به همراه یک یا چند آرگومان داده‌ای می پذیرد. هر عضو فرمت به یک آرگومان داده‌ای ارجاع می کند و نحوه‌ی فرمت دهی آن را مشخص می کند. `CLR` رشته‌ی خروجی را با تبدیل هر مقدار داده‌ای به یک رشته، فرمت دهی آن با توجه به عنصر فرمت دهی متناسب و جایگزینی آن در مقدار فرمت دهی شده، ایجاد می کند. این یک مثال ساده است.

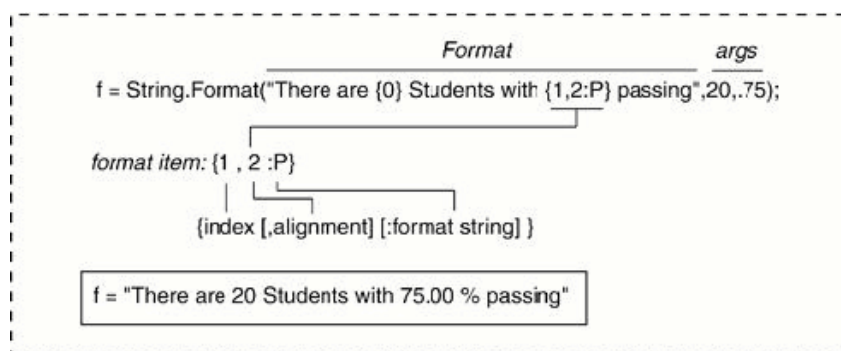
```
String s = String.Format("The square root of {0} is {1}.", 64, 8);
// output: The square root of 64 is 8.
```

این متد چندین `Overload` دارد. اما این معمول ترین است و دو ویژگی مشترک همه را ارائه می کند: یک رشته‌ی فرمت و یک لیست از آرگومان‌های داده‌ای. توجه کنید که متد `Console.WriteLine` همان پارامترها را می گیرد و می تواند به جای متد `String.Format` در خروجی کنسول استفاده شود.

۲۰-۶-۱-ساختن یک عنصر فرمت

شکل ۲۰-۳ مثال `String.Format` را به اجزاء پایه‌ای آن می شکند. جالب ترین آنها عنصر فرمت است که روش نمایش داده را تعریف می کند.

شکل ۲۰-۳



همانطور که می توانید ببینید، هر عنصر فرمت یک اندیس و یک آرگومان اختیاری ترازبندی و رشته‌ی فرمت را شامل است. همه‌ی آنها در بین کاراکترهای `{}` قرار می گیرند.

۱. اندیس یک مقدار صحیح با پایه صفر است که آرگومان داده‌ای مرتبط با آن را نشان می دهد. اندیس می تواند چند بار در یک رشته تکرار شود.

۲. ترازبندی اختیاری، یک عدد صحیح است که حداقل طول ناحیه‌ی مربوط به مقدار فرمت دهی شده را مشخص می کند. اگر مقدار ترازبندی مثبت باشد، مقدار آرگومان، راست چین است. اگر مقدار آن منفی باشد، آن چپ چین است.

۳. رشته فرمت اختیاری، کدهای فرمت‌دهی اعمال شده به مقدار آرگومان را در بر دارد. اگر آن مشخص نشده باشد، خروجی متد ToString مربوط به آرگومان استفاده می‌شود. NET. چندین کد فرمت استاندارد برای ایجاد رشته‌هایی با فرمت سفارشی فراهم کرده است.

۲۰-۶-۲-فرمت‌دهی مقادیر عددی

۹ کاراکتر یا مشخصه‌های فرمت برای فرمت‌دهی اعداد به پول رایج، علمی، هگزادسیمال و نمایش‌های دیگر در دسترس هستند. برای مشخص کردن دقت می‌توانید به هر کاراکتر یک عدد الحاق کنید. معمولاً این عدد تعداد ارقام اعشار را مشخص می‌کند. #C مشخصه‌های فرمت استاندارد جدول ۲۰-۴ را تشخیص می‌دهد.

جدول ۲۰-۴

علامت فرمت	توصیف	الگو	خروجی
c یا C	عدد به صورت یک مقدار پولی نمایش می‌گردد. دقت، تعداد ارقام اعشار را نشان می‌دهد.	{٠٠٠:٠}, ١٤٥٨,٧٥}	١.٤٥٨,٧٥ \$
d یا D	اعداد دهدی. به اعداد صحیح اعمال می‌گردد. دقت – تعداد کل فضای اشغالی را مشخص می‌کند. فضاهای خالی با صفر پر می‌شوند.	{٠٠٠:٠}, ٤٥٥}, ٥D:٠} {٠٠٠:٠}, -٥D:٠}	٠٠٤٥٥ ٠٠٤٥٥-
e یا E	علمی. عدد به نمایش علمی تبدیل می‌شود. دقت ddddE+nnm تعداد ارقام اعشار را مشخص می‌کند.	{٠٠٠:٤E٢}, ٣٢٩٨,٧٨ {٠٠٠:٤E٤}, -٥٤٧٨٣,٤	٠٠٣E+٣,٣٠ - 5.4783+E004
f یا F	نقطه اعشار ثابت. عدد به فرمت dddd.ddd تبدیل می‌شود. دقت، تعداد ارقام اعشار را مشخص می‌کند.	{٠٠٠:٠F٠}, ١٦٢,٥٧ {٠٠٠:٠F٢}, ٨١٦٢,٥٧	١٦٢ ٨١٦٢,٥٧
g یا G	کلی. عدد بر اساس دقت و نوع عدد به اعشار ثابت یا نمایش علمی تبدیل می‌گردد. در صورتی که توان بزرگتر مساوی دقت یا کمتر از -٤ باشد، از نمایش علمی استفاده می‌شود.	{٠٠٠:G}, ٠.٠٠٠٠٠٩٩ {٠٠٠:G٢}, ٤٥٥,٨٩ {٠٠٠:G٣}, ٤٥٥,٨٩ {٠٠٠:G}, ٧٨٣٢٢٩,٣٤	9.9E-06 4.6E+02 456 783229.34
n یا N	عدد. عدد را به یک رشته تبدیل می‌کند که هر ١٠٠٠ با کاما جدا می‌شود. تعداد ارقام اعشار را دقت مشخص می‌کند.	{٠٠٠:٠N}, ١٠٤٥,٧٨ {٠٠٠:٠N١}, ٤٥,٩٨	1,045.78 4RKV
p یا P	درصد. عدد به صد تقسیم می‌شود و به صورت درصد نمایش داده می‌شود. تعداد ارقام اعشار با دقت مشخص می‌گردد.	{٠٠٠:٠P}, ٠,٧٨ {٠٠٠:٠P٣}, ٠,٧٨٦٥	78.00 % 78.650 %
r یا R	گرد کردن. عدد را به یک رشته تبدیل می‌کند که همه ارقام اعشار به درستی حفظ می‌شوند. باید عدد از نوع اعشاری شناور باشد.	{٠٠٠:٠R}, ١,٦٢٧٣٦	1.62736
x یا X	هگزا دسیمال. عدد را به نمایش هگزا تبدیل می‌کند.	{٠٠٠:٠X}, ٢٥ {٠٠٠:٠X٤}, ٢٥ {٠٠٠:٠X٤}, ٢١	19 001V 001f

دقت عدد حداقل تعداد ارقام قابل نمایش را نشان می‌دهد. فضاهای اضافی با صفر پر می‌شوند.

الگوهای این جدول مستقیماً در دستور Console.WriteLine و Console.Write قابل استفاده هستند.

```
Console.WriteLine("The Hex value of {0} is {0:X} ",31); //1F
```

علامت‌های فرمت می‌توانند برای بهبود خروجی متد ToString بکار روند.

```
decimal pct = .758M;
Console.Write("The percent is "+pct.ToString("P2")); // 75.80 %
```

.NET کاراکترهای فرمت‌دهی خاصی را برای ایجاد فرمت‌های عددی سفارشی فراهم می‌کند. معمول‌ترین آنها عبارتند از: # (علامت عدد)، ۰ (صفر)، (کاما)، . (نقطه)، % (درصد) و (سمی کالن). کد زیر کاربرد آنها را نشان می‌دهد.

```
decimal dVal = 2145.88M; // decimal values require M suffix
string myFormat;
myFormat = dVal.ToString("#####"); // 2146
myFormat = dVal.ToString("#,###.00"); // 2,145.88
myFormat = String.Format("Value is {0:#,###.00};
(#{,###.00})", -4567);
// semicolon specifies alternate formats. (4,567.00)
myFormat = String.Format("Value is {0:$#,###.00}", 4567);
// $4,567.00
Console.WriteLine("{0:##.00%}", .18); // 18.00 %
```

همه این کاراکترها به استثناء خود توصیف هستند. آن فرمت را به دو گروه تقسیم می‌کند. گروه اول به مقادیر مثبت و گروه دوم به مقادیر منفی اعمال می‌شوند. دو سمی کالن برای ایجاد سه گروه برای اعداد مثبت، منفی و مقادیر صفر استفاده می‌شوند.

۲۰-۶-۳-فرمت‌دهی تاریخ و زمان

در فرمت‌دهی تاریخ به یک شی DateTime نیاز داریم. همانند اعداد، این شی نیز مجموعه علائم استاندارد فرمت خود را دارد. در جدول ۲۰-۵ خلاصه‌ی آنها آمده است.

جدول ۲۰-۵

علامت فرمت	توصیف	مثال انگلیسی	مثال آلمانی
d	تاریخ کوتاه	۲۰۰۴/۱۹/۱	۱۹,۱,۲۰۰۴
D	تاریخ بلند	Monday, January ۱۹, ۲۰۰۴	Montag, ۱۹ Januar, ۲۰۰۴
f	زمان/تاریخ کامل (زمان کوتاه)	Monday, January ۱۹, ۲۰۰۴ ۴:۰۵ PM	Montag, ۱۹ Januar, ۲۰۰۴ ۱۶:۰۵
F	زمان / تاریخ کامل (زمان کامل)	Monday, January ۱۹, ۲۰۰۴ ۴:۰۵:۲۰ PM	Montag, ۱۹ Januar, ۲۰۰۴ ۱۶:۰۵:۲۰
g	زمان / تاریخ کلی (زمان کوتاه)	PM ۴:۰۵ ۲۰۰۴/۱۹/۱	۶:۰۵ ۲۰۰۴/۱/۱۹
G	زمان / تاریخ کلی (زمان بلند)	PM ۴:۰۵:۲۰ ۲۰۰۴/۱۹/۱	۱۶:۰۵:۲۰ ۲۰۰۴/۱/۱۹

Januar ۱۹	January ۱۹	روز / ماه	m, M
Januar, ۲۰۰۴	January, ۲۰۰۴	ماه / سال	y, Y
۱۶:۰۵	PM ۴:۰۵	زمان کوتاه	t
۱۶:۰۵:۲۰	PM ۴:۰۵:۲۰	زمان بلند	T
۱۹-۰۱-۲۰۰۴	۱۶:۰۵:۲۰T ۹-۰۱-۲۰۰۴	با زمان / تاریخ جهانی ۸۶۰۱Iso مطابقت می‌کند. زمان محلی را بکار می‌برد.	S
۱۶:۰۵:۲۰T			
۱۹-۰۱-۲۰۰۴	Z۱۶:۰۵:۲۰ ۱۹-۰۱-۲۰۰۴	زمان - تاریخ جهانی	u
Z۱۶:۰۵:۲۰			
Montag, ۱۹. Januar	Monday, January ۱۹, ۲۰۰۴ ۲۱:۰۵:۲۰ PM	زمان تاریخ جهانی زمان جهانی را بکار می‌برد.	U
۲۱:۰۵:۲۰ ۲۰۰۴			

دستورات زیر تعدادی مثال فرمت‌دهی تاریخ را نشان می‌دهند. در هر کدام، یک شی از نوع DateTime به عنوان آرگومان به یک رشته‌ی فرمت رد می‌شود.

```
DateTime curDate = DateTime.Now; // Get Current Date
Console.WriteLine("Date: {0:d} ", curDate); // 1/19/2004
// f: --> Monday, January 19, 2004 5:05 PM
Console.WriteLine("Date: {0:f} ", curDate);
// g: --> 1/19/2004 5:05 PM
Console.WriteLine("Date: {0:g} ", curDate);
```

اگر هیچ کدام از علائم استاندارد فرمت نیاز شما را برآورده نسازد، می‌توانید یک فرمت سفارشی را با یک دنباله از کاراکترهای زیر ایجاد کنید. جدول ۲۰-۶، چند نمونه مفید در فرمت‌دهی تاریخ‌ها را لیست می‌کند.

جدول ۲۰-۶

فرمت	توصیف
d	روز ماه. صفر اضافی وجود ندارد.
dd	روز ماه. همیشه دو رقم وجود دارد.
ddd	روز هفته. با سه کاراکتر اختصار
dddd	نام کامل روز هفته
M	عدد ماه. بدون صفر اضافی
MM	عدد ماه. همیشه دو رقم دارد
MMM	نام ماه با سه حرف اختصار
MMMM	نام کامل ماه
y	یک یا دو رقم آخر سال
yy	یک یا دو رقم آخر سال. در صورت نیاز صفر اضافی قرار می‌گیرد.

چهار رقم سال	YYYY
ساعت در فرمت ۲۴ ساعته	HH
دقیقه. در صورت نیاز با صفرهای اضافی	mm

اینها مثال‌هایی از فرمت‌های سفارشی تاریخ هستند.

```
DateTime curDate = DateTime.Now;
f = String.Format("{0:dddd} {0:MMM} {0:dd}", curDate);
// output: Monday Jan 19
f = currDate.ToString("dd MMM yyyy")
// output: 19 Jan 2004
// The standard short date format (d) is equivalent to this:
Console.WriteLine(currDate.ToString("M/d/yyyy")); // 1/19/2004
Console.WriteLine(currDate.ToString("d")); // 1/19/2004
CultureInfo ci = new CultureInfo("de-DE"); // German
f = currDate.ToString("dd-MMMM-yyyy HH:mm", ci)
// output: 19-Januar-20MQ 23:07
```

در فرمت‌دهی سفارشی تاریخ، متد ToString به String.Format ترجیح داده می‌شود. آن گرامر مناسبی برای تعبیه فضاهای خالی و جداکننده‌های خاص مابین عناصر تاریخ دارد. به علاوه، پارامتر دوم آن یک مشخص‌کننده فرهنگ است.

۲۰-۶-۴- تاریخ‌ها و فرهنگ

در سراسر جهان، تاریخ‌ها به روش‌های مختلفی نمایش داده می‌شوند. توانایی اضافه کردن فرهنگ در فرمت‌دهی توسط NET، بارکاری توسعه‌دهنده را کاهش داده است. برای مثال، اگر فرهنگ سیستم شما German است، به طور اتوماتیک تاریخ‌ها فرمت‌دهی می‌شوند تا فرمت European را منعکس کنند. روز قبل از ماه قرار می‌گیرد و روز و ماه و سال با نقطه از هم جدا می‌شوند (به جای /) و عبارت January ۱۹Monday، به صورت Montag، ۱۹Januar. مشخص می‌شود. این مثال متد ToString را با پارامتر CultureInfo با فرهنگ German به کار می‌برد.

```
CultureInfo ci = new CultureInfo("de-DE"); // German
Console.WriteLine(curDate.ToString("D", ci));
// output ---> Montag, 19. Januar 2004
Console.WriteLine(curDate.ToString("dddd", ci)); // --> Montag
```

آخرین دستور فرمت سفارشی "dddd" را برای چاپ کردن نام کامل روز هفته بکار می‌رود. این فرمت در برابر خصوصیت شمارشی DateTime.DayOfWeek است که فقط یک مقدار انگلیسی بر می‌گرداند.

کلاس‌های DateTimeFormatInfo و NumberFormatInfo

این دو کلاس نحوه اعمال فرمت‌های بیان شده قبلی روی تاریخ‌ها و اعداد را مدیریت می‌کنند. برای مثال، کلاس NumberFormatInfo خصوصیتی دارد که کاراکتر سمبل پول، کاراکتر جدا کننده دهدهی و تعداد ارقام دهدهی برای نمایش یک مقدار پول را مشخص می‌کند. به طور مشابه، کلاس DateTimeFormatInfo خصوصیتی تعریف می‌کند که به طور مجازی با همه علائم استاندارد فرمت تاریخ‌ها یکسان هستند. به عنوان مثال، خصوصیت FullDateTimePattern می‌باشد که معادل علامت فرمت F برای فرمت‌دهی تاریخ می‌باشد.

کلاس‌های NumberFormatInfo و DateTimeFormatInfo با فرهنگ‌های خاص مرتبط هستند و خصوصیات آنها وسیله‌ای برای ایجاد فرمت‌های منحصر به فرد مورد نیاز فرهنگ‌های مختلف هستند. NET یک مجموعه‌ای از پیش تعریف شده از مقادیر خصوصیت برای هر فرهنگی فراهم می‌کند، که نمی‌توان آنها را override کرد.

بر اساس این که آیا فرهنگ جاری یا غیره ارجاع می‌شوند، این خصوصیات می‌توانند به روش‌های مختلفی دستیابی شوند. دستورات زیر به فرهنگ جاری ارجاع دارند.

```
NumberFormatInfo.CurrentInfo.<property>
CultureInfo.CurrentCulture.NumberFormat.<property>
```

دستور اول خصوصیت ایستای CurrentInfo را بکار می‌برد و قطعاً فرهنگ جاری را بکار می‌برد. دستور دوم یک فرهنگ را بطور صریح مشخص می‌کند و برای دسترسی به خصوصیات فرهنگ دیگر یک نمونه از CultureInfo مناسب است.

```
CultureInfo ci = new CultureInfo("de-DE");
string f = ci.NumberFormat.CurrencySymbol;
```

خصوصیاتی از کلاس‌های NumberFormatInfo و DateTimeFormatInfo که در ارتباط با فرهنگ غیر جاری هستند، می‌توانند تغییر یابند. قطعه کد ۲۰-۳ یک نمونه از نحوه‌ی کار با این کلاس‌ها را پیشنهاد می‌کند

قطعه کد ۲۰-۳

```
using System
using System.Globalization
Class MyApp
// NumberFormatInfo
string curSym = NumberFormatInfo.CurrentInfo.CurrencySymbol;
int dd = NumberFormatInfo.CurrentInfo.CurrencyDecimalDigits;
int pdd = NumberFormatInfo.CurrentInfo.PercentDecimalDigits;
// --> curSym = "$" dd = 2 pdd = 2
// DateTimeFormatInfo
string ldp= DateTimeFormatInfo.CurrentInfo.LongDatePattern;
// --> ldp = "dddd, MMMM, dd, yyyy"
string enDay = DateTimeFormatInfo.CurrentInfo.DayNames[1];
string month = DateTimeFormatInfo.CurrentInfo.MonthNames[1];
CultureInfo ci = new CultureInfo("de-DE");
string deDay = ci.DateTimeFormat.DayNames[1];
// --> enDay = "Monday" month = February deDay = "Montag"
// Change the default number of decimal places
// in a percentage
decimal passRate = .840M;
Console.Write(passRate.ToString("p",ci)); // 84,00%
ci.NumberFormat.PercentDecimalDigits = 1;
Console.Write(passRate.ToString("p",ci)); // 84,0%
}
```

خلاصه اینکه، NET برای برآورده کردن نیاز شما به فرمت‌دهی تاریخ‌ها و اعداد، الگوهای متنوعی را پیشنهاد می‌کند. برای این منظور دو کلاس NumberFormatInfo و DateTimeFormatInfo وجود دارند که سبیل‌ها و قوانین مورد استفاده در فرمت‌دهی را تعریف می‌کنند. NET به ازای هر فرهنگ خصوصیات فرمت خاص آن را فراهم کرده است.

۲۰-۷- عبارات منظم

کاربرد رشته‌ها و عبارت منظم برای انجام تطابق الگو از زمان اولین زبان‌های برنامه‌نویسی بوده است. در اواسط ۱۹۶۰، SNOBOL برای بیان هدف دستکاری رشته و متن طراحی شد. آن بر روی ابزار Grep در محیط Unix تأثیر گذاشت. Grep کاربرد عبارات منظم را گسترده ساخت. همه‌ی آنهایی که با Grep یا Perl یا هر زبان اسکریپتی دیگر کار کرده‌اند، شباهت پیاده‌سازی عبارات منظم در NET را تشخیص خواهند داد.

تطابق الگو بر اساس مفهوم ساده‌ی اعمال یک الگوی رشته‌ای خاص به متون، برای تطابق یک نمونه یا نمونه‌هایی از آن الگو در متن است. بر خلاف متن، الگوی اعمال شده یک عبارت منظم یا به اختصار Regex بیان می‌گردد.

۲۰-۷-۱-کلاس Regex

فرض کنید کلاس Regex به عنوان موتور ارزیابی عبارات منظم است و الگوها را به رشته‌های هدف اعمال می‌کند. این کلاس متدهای ایستا و نمونه فراهم می‌کند که عبارات منظم را برای جستجوی متن، استخراج و جایگذاری بکار می‌برند. کلاس Regex و همه‌ی کلاس‌های مرتبط با آن، در فضای نامی System.Text.RegularExpressions یافته می‌شوند.

گرامر

```
Regex( string pattern )
Regex( string pattern, RegexOptions)
```

پارامترها:

Pattern: عبارت منظم برای تطابق الگو

Regex Options: یک نوع شمارشی که مقادیر آن نحوه‌ی اعمال عبارت منظم را کنترل می‌کند. مقادیر آن عبارتند از:

CultureInvariant- از فرهنگ صرفنظر می‌کند.

IgnoreCase- از حالت حروف صرفنظر می‌کند.

RightToLeft- رشته را از چپ به راست پردازش می‌کند.

مثال:

```
Regex r1 = new Regex(" "); // Regular expression is a blank
String words[] = r1.Split("red blue orange yellow");
// Regular expression matches upper- or lowercase "at"
Regex r2 = new Regex("at", RegexOptions.IgnoreCase);
```

همانطور که مثال نشان می‌دهد، ایجاد یک شی Regex کاملاً ساده است. اولین پارامتر سازنده یک عبارت منظم است. پارامتر اختیاری دوم، یک یا چند مقدار شمارشی RegexOptions است که نحوه‌ی اعمال عبارت منظم را کنترل می‌کند.

متدهای Regex

کلاس Regex تعدادی متد برای تطابق الگو و دستکاری متن دارد. آنها Match، Split، Replace، IsMatch و Matches را شامل هستند. آنها Overloadهای ایستا و نمونه‌ای دارند که شبیه هستند، اما یکسان نیستند.

اگر بخواهید یک عبارت منظم را مکرراً بکار ببرید، ایجاد یک شی Regex موثرتر است. زمانی که شی ایجاد می‌شود، آن عبارت منظم را به یک فرم خاصی کامپایل می‌کند و تا زمان وجود شی می‌توان از آن استفاده کرد. بر خلاف آن، متدهای ایستا در زمان استفاده‌ی عبارت، آن را مجدداً کامپایل می‌کنند.

حال اجازه دهید برخی از متدهای Regex را بررسی کنیم. در این مثال‌ها، هدف معرفی متدهای عبارت‌های منظم است. به همین دلیل از مثال‌های ساده استفاده می‌شود.

IsMatch()

این متد عبارت منظم را با یک رشته ورودی مطابقت می‌دهد و یک مقدار بر می‌گرداند. این مقدار مشخص می‌کند، آیا یک تطابق پیدا شده است.

```
string searchStr = "He went that a way";
Regex myRegex = new Regex("at");
// instance methods
bool match = myRegex.IsMatch(searchStr); // true
// Begin search at position 12 in the string
match = myRegex.IsMatch(searchStr, 12); // false
```

```
// Static Methods - both return true
match = Regex.IsMatch(searchStr, "at");
match = Regex.IsMatch(searchStr, "AT", RegexOptions.IgnoreCase);
```

(Replace)

این متد یک الگوی مطابقت شده را با یک رشته‌ی خاص جایگزین می‌کند و نتیجه را بر می‌گرداند. این متد چندین overload برای مشخص کردن محل شروع جستجو و تعداد جایگزینی‌ها دارد.

گرامر

```
static Replace (string input, string pattern, string replacement
[,RegexOptions])
Replace(string input, string replacement)
Replace(string input, string replacement, int count)
Replace(string input, string replacement, int count, int startat)
```

پارامتر Count حداکثر تعداد تطبیق‌ها را مشخص می‌کند. StartAt محل شروع عمل جستجوی تطابق‌ها را نشان می‌دهد. این متد نسخه‌های مختلفی دارد که یک پارامتر نماینده MatchEvaluator را می‌پذیرد. هر وقت یک تطابق پیدا شود، این نماینده فراخوانی می‌شود و می‌تواند برای سفارشی کردن پروسه‌ی جایگزینی استفاده شود.

قطعه کد زیر شکل‌های ایستا و نمونه متد را ارائه می‌کند.

```
string newStr;
newStr = Regex.Replace("soft rose","o","i"); // sift rise
// instance method
Regex myRegex = new Regex("o"); // regex = "o"
// Now specify that only one replacement may occur
newStr = myRegex.Replace("soft rose","i",1); // sift rose
```

(Split)

این عمل رشته را با توجه به محل‌های تطابق یافته شده به یک آرایه تفکیک می‌کند. آن شبیه متد (String.Split) است، با این استثناء که تطابق به جای یک کاراکتر با رشته‌ی کاراکتری و بر اساس یک عبارت منظم انجام می‌شود.

گرامر

```
String[] Split(string input)
String[] Split(string input, int count)
String[] Split(string input, int count, int startat)
Static String[] Split(string input, string pattern)
```

پارامترها

Input: رشته مورد نظر جهت تفکیک

Count: حداکثر تعداد عناصر آرایه، مقدار صفر هر تعداد عنصر را ممکن می‌سازد. اگر تعداد تطابق‌ها از حداکثر بیشتر باشد، آخرین بخش مابقی رشته را در بر دارد.

Startat: محل شروع جستجو در رشته‌ی ورودی را مشخص می‌کند.

Pattern: الگوی عبارت منظم برای تطبیق با رشته ورودی است.

مثال کوتاه زیر یک رشته شامل لیستی از اسامی هنرمندان را تجزیه کرده و در یک آرایه قرار می‌دهد. یک کاما و به دنبال آن صفر یا چند فضای خالی اسامی را از هم جدا می‌کند. عبارت منظم برای تطبیق این جداساز "[*]" است. نحوه‌ی ایجاد این عبارت را بعداً خواهید دید.

```
// Regex to match a comma followed by 0 or more spaces
string patt = @",[ ]*";
// Static method
string[] artists = Regex.Split(impressionists, patt);
// Instance method is used to accept maximum of four matches
Regex myRegex = new Regex(patt);
string[] artists4 = myRegex.Split(impressionists, 4);
foreach (string master in artists4)
    Console.Write(master);
// Output --> "Manet" "Monet" "Degas" "Pissarro,Sisley"
```

(Matches() , Match

این متدها یک رشته ورودی را برای تطابق با عبارت منظم جستجو می‌کنند. متد Match() یک شی Match واحد بر می‌گرداند و متد Matches() شی MatchCollection (یک کلکسیون از همه‌ی تطابق‌ها) را بر می‌گرداند.

گرامر

```
Match Match(string input)
Match Match(string input, int startat)
Match Match(string input, int startat, int numchars)
static Match(string input, string pattern, [RegexOptions])
```

متد Matches() overloadهای مشابهی دارد، اما یک شی MatchCollection بر می‌گرداند.

Match و Matches مفیدترین متدهای Regex هستند. شی Match بسیار غنی بوده و خصوصیاتِ همچون رشته تطبیق شده، طول آن و محل آن در رشته ورودی را دارد. آن یک خصوصیت Groups دارد که شکستن یک الگوی تطبیق به چندین زیررشته تطبیق‌شده را مجاز می‌دارد. جدول ۲۰-۷ اعضای انتخابی از کلاس Match را نشان می‌دهد.

جدول ۲۰-۷

عضو	توصیف
Index	این خصوصیت محل اولین کاراکتر تطابق یافته در رشته را بر می‌گرداند.
Groups	یک کلکسیون از گروه‌های کلاس. گروه‌ها بوسیله‌ی قرار دادن بخش‌هایی از عبارت منظم در پارامتر ایجاد می‌شوند. متنی که با الگوی داخل پرانتزها تطابق دارد در کلکسیون Groups قرار می‌گیرند.
Length	طول رشته‌ی تطبیق شده
Success	وابسته به یافتن یک تطبیق، true یا false می‌باشد.
Value	زیر رشته تطبیق شده را بر می‌گرداند.
NextMatch	یک Match جدید بر اساس کاراکترهای بعد از تطبیق قبلی بر می‌گرداند.

کد زیر کاربرد اعضای این کلاس را نشان می‌دهد. توجه کنید که نقطه (.) در عبارت منظم به صورت یک کاراکتر عام (جایگزینی) عمل می‌کند که با هر کاراکتری مطابقت دارد.

```
string verse = "In Xanadu did Kubla Khan";
string patt = ".an..."; // "." matches any character
Match verseMatch = Regex.Match(verse, patt);
```

```

Console.WriteLine(verseMatch.Value); // Xanadu
Console.WriteLine(verseMatch.Index); // 3
//
string newPatt = "K(..)"; //contains group(..)
Match kMatch = Regex.Match(verse, newPatt);
while (kMatch.Success) {
    Console.WriteLine(kMatch.Value); // -->Kub -->Kha
    Console.WriteLine(kMatch.Groups[1]); // -->ub -->ha
    kMatch = kMatch.NextMatch();
}

```

این مثال متد `NextMatch` را برای طی کردن سراسر رشته‌ی هدف بکار می‌برد و هر تطبیق را به `kMatch` انتساب می‌دهد. پرائتهای دو طرف دو نقطه در `newPatt` بدون هیچ اثری روی الگوی اصلی آن را به گروه‌هایی تقسیم می‌کند. در این مثال، دو کاراکتر بعد از `K` به اشیاء `Groups` انتساب داده می‌شوند.

بعضی اوقات ممکن است یک برنامه نیاز داشته باشد قبل از پردازش تطبیق‌ها آنها را در یک کلکسیون جمع کند. این کار هدف کلاس `MatchCollection` است. این کلاس فقط یک ظرف برای نگه داشتن اشیاء `Match` است و با استفاده از متد `Regex.Matches` ایجاد می‌شود. `Count` مفیدترین خصوصیت آن است و تعداد تطبیق‌ها را بر می‌گرداند و `Item` یک عضو واحد از کلکسیون را بر می‌گرداند. در این جا روش دیگر نوشتن حلقه‌ی `NextMatch` مثال قبلی آمده است.

```

string verse = "In Xanadu did Kubla Khan";
String newpatt = "K(..)";
foreach (Match kMatch in Regex.Matches(verse, newpatt))
    Console.WriteLine(kMatch.Value); // -->Kub -->Kha
// Could also create explicit collection and work with it.
MatchCollection mc = Regex.Matches(verse, newpatt);
Console.WriteLine(mc.Count); // 2

```

۲۰-۷-۲- ایجاد عبارات منظم

مثال‌هایی که تا به حال برای ارائه متدهای `Regex` استفاده شده‌اند، فقط عبارتهای منظم ابتدایی را بکار گرفتند. حال، کاوش می‌کنیم چگونه عبارت منظم واقعی و مفید ایجاد کنیم. اگر این موضوع برای شما تازه است، شما طراحی عبارات منظم را با پروسه آزمایش و خطا کشف خواهید کرد. البته تقریباً همه الگوهای عمومی مورد استفاده روی یک سایت وب یافته می‌شود. این سایت یک کتابخانه قابل جستجو از الگوهای `Regex` را نگه می‌دارد (www.regexlib.com).

یک عبارت منظم به چهار نوع مختلف از فراکاراکترها تقسیم می‌شود که هر کدام نقش خاصی از پروسه تطبیق دارند.

کاراکترهای تطبیق: اینها یک نوع خاصی از کاراکتر را تطبیق می‌دهند. برای مثال `d\` هر رقم از ۰ تا ۹ را تطبیق می‌دهد.

کاراکترهای تکرار: از تکرار یک کاراکتر یا عنصر تطبیقی جلوگیری می‌کند. برای مثال: `{۳d\}` می‌تواند به جای `d\d\d\` برای سه رقم تطبیقی استفاده شود.

کاراکترهای موقعیتی: محلی از رشته‌ی مقصد که یک تطبیق در آنجا باید رخ دهد. برای مثال: `{۳d\^}` لازم است تطبیق در ابتدای رشته رخ دهد.

کاراکترهای `Escape`: کاربرد `\` قبل از یک کاراکتر مفهوم خاصی می‌رساند. برای مثال `\{` اجازه می‌دهد گروه‌ی بسته در رشته تطبیق باشد.

جدول ۲۰-۸ الگوهای معروف را خلاصه می‌کند.

جدول ۲۰-۸

الگو	معیار تطبیق	مثال
------	-------------	------

[illegible]

می توان استفاده کرد.

[^۰۰۰] همه کاراکترها به استثناء کاراکترهای داخل [^aeiou]، x را تطبیق می دهد.
کروشه را تطبیق می دهد.

یک مثال تطابق الگو

این الگوهای کاراکتری را برای ایجاد یک عبارت منظم جهت تطبیق یک شماره تأمین اجتماعی اعمال کنید.

```
bool iMatch = Regex.IsMatch("245-09-8444", @"^\d\d\d-\d\d-\d\d\d\d$");
```

این یک روش بسیار سر راست است، که هر کاراکتر شماره تأمین اجتماعی با یک کاراکتر در عبارت منظم تطابق دارد. آن ساده به نظر می رسد، ولی در رشته های بلند تکرار این کاراکترها مزاحمت ایجاد می کند. کاراکترهای تکرار این عمل را بهبود می دهند.

```
bool iMatch = Regex.IsMatch("245-09-8444", @"^\d{3}-\d{2}-\d{4}$");
```

می توانیم محدودیت های دیگری روی شماره تأمین اجتماعی بررسی کنیم. ممکن است بخواهید مطمئن شوید، آن روی یک خط بوده و در ابتدا یا انتهای خط است. این نیاز دارد کاراکترهای موقعیت را در ابتدا یا انتهای دنباله تطبیق بکار برید.

فرض کنید، می خواهیم مطمئن شویم شماره تأمین اجتماعی فقط روی یک خط باشد. برای انجام این کار دو کاراکتر لازم داریم. یکی برای اطمینان از اینکه تطبیق در ابتدای خط است و دیگری برای اطمینان از اینکه تطبیق در انتهای خط است. با توجه به جدول ۹-۲۰ کاراکترهای ^ و \$ در عبارت منظم برای برآوردن این شرط می توانند استفاده شوند. رشته جدید به صورت زیر خواهد بود.

```
@'^^\d{3}-\d{2}-\d{4}$'
```

جدول ۹-۲۰

کاراکتر موقعیت	توصیف
^	الگوی بعد از آن باید در ابتدای رشته یا خط باشد.
\$	الگوی قبل از آن باید در انتهای خط یا انتهای رشته باشد.
A\	الگوی قبل از آن باید در ابتدای یک رشته باشد.
b \B\	تا مرز یک کلمه حرکت می کند. که می تواند یک کاراکتر کلمه یا کاراکتر غیر کلمه باشد.
z \Z\	الگو باید در انتهای رشته (Z\) یا در انتهای رشته قبل از یک خط جدید باشد.

این کاراکترهای موقعیت، هیچ فضایی در عبارت نمی گیرند، آنها فقط محلی که باید تطابق رخ دهد را مشخص می کنند.

به عنوان یک بهبود نهایی روی الگوی SSN، کاراکترهای مابین خط فاصله را به سه گروه تقسیم کنید. برای ایجاد گروه، بخش های مورد نظر را در داخل پرانتزها قرار دهید. در این حالت می توانید هر بخش را به طور مستقل بررسی کنید. کد ساده زیر این الگوی پیشنهادی را بکار می برد.

```
string ssn = "245-09-8444";
string ssnPatt = @"^(\d{3})-(\d{2})-(\d{4})$";
Match ssnMatch = Regex.Match(ssn, ssnPatt);
```

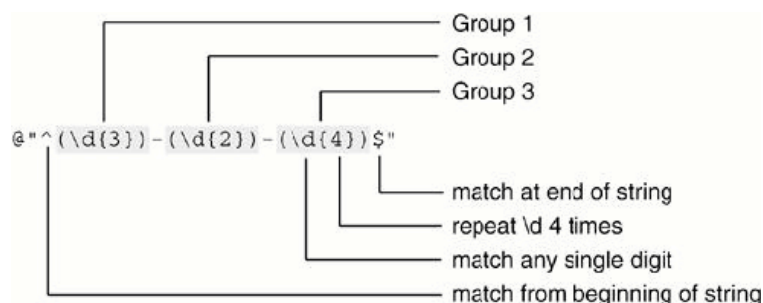
```

if (ssnMatch.Success) {
    Console.WriteLine(ssnMatch.Value); // 245-09-8444
    Console.WriteLine(ssnMatch.Groups.Count); // 4
    // Count is 4 since Groups[0] is set to entire SSN
    Console.WriteLine(ssnMatch.Groups[1]); // 245
    Console.WriteLine(ssnMatch.Groups[2]); // 09
    Console.WriteLine(ssnMatch.Groups[3]); // 8444
}

```

ما حالا یک الگوی مفیدی داریم که کاراکترهای موقعیت، تکرار و گروه را استفاده کرده است. ابتدا یک الگوی ساده ایجاد کردیم، سپس در چندین مرحله آن را بهبود دادیم. این روش معمول ایجاد عبارات منظم پیچیده است. (شکل ۲۰-۴ را ببینید).

شکل ۲۰-۴



کار با گروه‌ها

همانطور که در مثال قبلی دیدیم، متن منتج شده از یک تطابق، با قرار دادن بخش‌های یک عبارت در داخل پرانتزها، به طور اتوماتیک می‌تواند به زیر رشته‌ها یا گروه‌هایی تقسیم شود. متنی که با الگوی داخل پرانتزها تطبیق است، یک عضوی از کلکسیون `Match.Groups` می‌شود. این کلکسیون با یک آرایه از خانه‌ی صفر اندیس‌گذاری می‌شود. عنصر صفر برای کل تطابق، عنصر ۱ برای گروه اول، عنصر ۲ برای گروه دوم و غیره هستند.

برای بکارگیری آسان گروه‌ها می‌توانید آنها را نام‌گذاری کنید. نام مورد نظر در داخل پرانتز و بعد از پرانتز باز قرار می‌گیرد، که گرامر آن `<name>?` است. برای مشاهده‌ی کاربرد این گروه‌ها، فرض کنید در یک رشته اسامی هفته و دماهای حداکثر و حداقل وجود دارد و می‌خواهیم آنها را تجزیه کنیم.

```
string txt = "Monday Hi:88 Lo:56 Tuesday Hi:91 Lo:61";
```

عبارت منظم این تطبیق دو گروه دارد. روز و دماها. کد زیر یک کلکسیون از تطبیق‌ها ایجاد می‌کند و سراسر کلکسیون را طی می‌کند و محتوای هر گروه را چاپ می‌کند.

```

string rgPatt = @"(?<day>[a-zA-Z]+)\s*(?<temps>Hi:\d+\s*Lo:\d+)";
MatchCollection mc = Regex.Matches(txt, rgPatt); //Get matches
foreach (Match m in mc)
    Console.WriteLine("{0} {1}",
        m.Groups["day"], m.Groups["temps"]);
//Output: Monday Hi:88 Lo:56
// Tuesday Hi:91 Lo:61

```

توجه: بعضی مواقع می‌خواهید از پرانتزها برای اهداف دیگری استفاده کنید. برای اینکه اطلاعات داخل پرانتزها به عنوان گروه‌بندی در نظر گرفته شود، در داخل پرانتز بعد از پرانتز باز علامت `?` را قرار دهید. در `(an|in|on)` هدف OR منطقی است و گروه‌بندی نیست، پس می‌نویسیم: `(an|in|on:?)`

ارجاع به عقب یک گروه

اغلب معمول است یک عبارت منظم ایجاد کنیم که یک تطابق بر اساس نتیجه‌ی تطابق قبلی را شامل است. برای مثال، در حین کنترل گرامر در پردازشگرهای کلمه، هر کلمه‌ای را که یک تکرار از کلمه‌های قبلی باشند علامت می‌زنند. می‌توانیم یک عبارت منظم برای انجام همین عمل ایجاد کنیم. راز آن تعریف یک گروه برای تطابق یک کلمه و استفاده از کلمه تطبیق شده به عنوان بخشی از الگو است. برای فهم بیشتر کد زیر را ملاحظه کنید.

```
string speech = "Four score and and seven years";
patt = @"(\b[a-zA-Z]+\b)\s\1"; // Match repeated words
MatchCollection mc = Regex.Matches(speech, patt);
foreach(Match m in mc) {
    Console.WriteLine(m.Groups[1]); // --> and
}
```

این کد فقط کلمات تکراری را تطبیق می‌دهد. عبارت منظم را بررسی کنید.

متن/الگو	توصیف
and and @"(\b[a-zA-Z]+\b)\s"	هر کلمه که با مرز یک کلمه (\b) و فضای خالی دنبال می‌شود را تطبیق می‌دهد.
and and \1	علامت ارجاع به عقب با یک \ که به دنبال آن شماره گروه قرار دارد مشخص می‌شود و می‌توان به یک گروه ارجاع کرد. اثر آن درج مقدار یک گروه تطبیق شده به داخل عبارت است.

به جای شماره گروه می‌توانید اسم آن را بکار ببرید. گرامر این نوع ارجاع بصورت <نام گروه>k است که به دنبال عبارت منظم نوشته می‌شود:

```
patt = @"(?<word>\b[a-zA-Z]+\b)\s\k<word>";
```

۲۰-۷-۳- مثال‌هایی از کاربرد عبارات منظم

این بخش یک نگاه سریعی روی بعضی از الگوهای معمول دارد که برای اداره کردن تطبیق الگوهای معمول استفاده می‌شوند. باید دو چیز از این مثال‌ها بفهمید. اینها روش‌های نامحدودی برای ایجاد عبارات‌ها جهت حل یک مسئله واحد هستند و بیشتر مسائل تطبیق الگو نکات ظریفی دارند که فوراً آشکار نمی‌شوند.

کاربرد Replace برای معکوس کردن کلمات

```
string userName = "Claudel, Camille";
userName = Regex.Replace( userName, @"(\w+),\s*(\w+)", "$2 $1" );
Console.WriteLine(userName); // Camille Claudel
```

عبارت منظم نام اول و آخر را به گروه‌های ۱ و ۲ انتساب می‌دهد. پارامتر سوم متد Replace ارجاع به این گروه‌ها را با قرار دادن \$ قبل از شماره گروه مجاز می‌دارد. در این مثال، نام کامل تطبیق شده با کلمات گروه ۲ و به دنبال آن گروه ۱ جایگزین می‌گردد.

پارس کردن اعداد

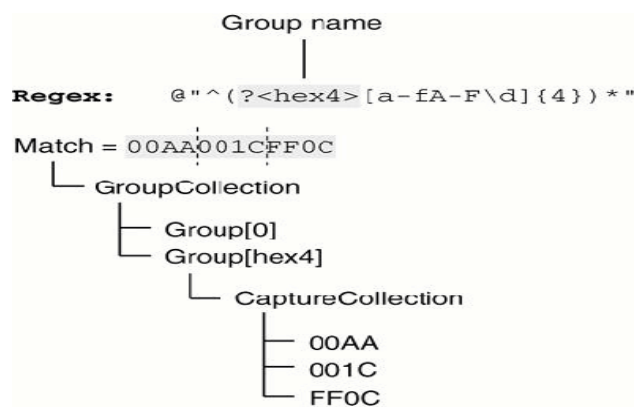
```
String myText = "98, 98.0, +98.0, +98";
string numPatt = @"^d+"; // Integer
numPatt = @"^d+\.?\d*" | @"^\.d+"; // Allow decimal
numPatt = @"^([+-]?\d+\.?\d*" | @"^([+-]?\.\d+)"); // Allow + or -
```

توجه کنید که کاربرد سمبل OR (|) در خط سوم کد، برای پیشنهاد کردن چندین الگو است. در این مثال، وجود یک عدد اختیاری قبل از نقطه اعشار را مجاز می‌دارد.

کد زیر کاراکتر ^۸ را برای لنگر انداختن الگو به ابتدای خط بکار می‌برد. عبارت منظم یک گروه چهار بایتی برای تطابق دارد. کاراکتر * باعث می‌شود تا زمانیکه به null نرسیده، گروه تکرار شود. زمانی که این گروه اعمال گردد، آن یک عدد هگزا دسیمال ۴ رقمی را بر می‌دارد که در شی CaptureCollection قرار می‌گیرد.

```
string hex = "00AA001CFF0C";
string hexPatt = @"^(?<hex4>[a-fA-F\d]{4})*";
Match hexMatch = Regex.Match(hex, hexPatt);
Console.WriteLine(hexMatch.Value); // --> 00AA001CFF0C
CaptureCollection cc = hexMatch.Groups["hex4"].Captures;
foreach (Capture c in cc)
    Console.WriteLine(c.Value); // --> 00AA 001C FF0C
```

شکل ۵-۵ رابطه‌ی سلسله‌مراتبی مابین Match، GroupCollection و CaptureCollection را نشان می‌دهد



شکل ۵-۲۰

۲۰-۸- خلاصه

- از زمانیکه کاراکترهای ۷ بیتی ASCII یا ANSI معرفی شدند، تقاضا برای کار با کاراکترها افزایش یافت.
- امروزه یونیکد استاندارد نمایش بیش از ۹۰۰۰۰ کاراکتر را تعریف می‌کند. که این استاندارد را با کاراکترهای ۱۶ بیتی می‌پوشاند.
- NET مفهوم محلی کردن را پشتیبانی می‌کند، یعنی رعایت اطلاعات فرهنگ ماشین محلی را تضمین می‌کند.
- اداره کردن رشته‌ها با یک مجموعه‌ی غنی از متدهای کلاس‌های String و StringBuilder در NET فراهم شده است.
- متدهای متنوعی برای مقایسه‌ی رشته‌ها موجود است که حالت حروف و فرهنگ را در صورت نیاز در نظر می‌گیرند.
- متد String.Format برای فرمت‌دهی نمایش اعداد، رشته‌ها، تاریخ و واحد پول و غیره استفاده می‌شود.
- دستکاری رشته‌ها حافظه را زیاد مصرف می‌کنند، برای کاربرد کارای حافظه از کلاس StringBuilder استفاده می‌کنند.
- برای تطابق الگوها و پارس کردن رشته‌ها می‌توان از کلاس Regex استفاده کرد.

کار با فایل‌ها

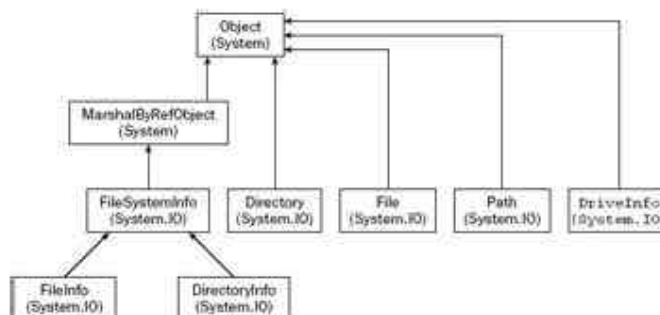
آنچه که در این فصل یاد خواهید گرفت:

- کاوش ساختار فهرست، پیدا کردن فایل‌ها و پوشه‌های موجود و بررسی خصوصیات آنها
- انتقال، کپی و حذف فایل‌ها و پوشه‌ها
- خواندن و نوشتن متن در فایل‌ها
- کنترل دسترسی به فایل‌ها

مایکروسافت مدل‌های شی زیادی برای پوشاندن این حوزه فراهم کرده است. در این فصل نحوه‌ی کاربرد کلاس‌های پایه NET برای انجام کارهای شرح شده در لیست قبلی را یاد می‌گیرید. کلاس‌های مرتبط با عملیات سیستم فایل تقریباً در فضای نامی System.IO هستند.

۱-۲۱- مدیریت سیستم فایل

کلاس‌هایی که برای جستجوی سیستم فایل و انجام عملیاتی همچون انتقال، کپی کردن و حذف کردن فایل‌ها هستند، در شکل ۱-۲۱ نشان داده می‌شوند. فضای نامی هر کلاس در داخل [] در کنار نام کلاس نشان داده می‌شود.



شکل ۱-۲۱

لیست زیر کار این کلاس‌ها را شرح می‌دهد

System.MarshalByRefObject: کلاس پایه‌ی کلاس‌هایی از NET است که از دور قابل کنترل هستند و مرتب‌کردن^۱ داده‌های مابین دامنه‌های کاربردی را مجاز می‌دارد.

FileInfo: کلاس پایه که هر شی سیستم فایل را نشان می‌دهد.

File و FileInfo: این کلاس‌ها یک فایل را روی سیستم فایل نشان می‌دهند.

DirectoryInfo, Directry: این کلاس‌ها یک پوشه روی سیستم فایل را نشان می‌دهند.

Path: این کلاس اعضای ایستایی دارد که می‌توانند برای دستکاری اسامی مسیرها بکار روند.

DriveInfo: این کلاس متدها و خصوصیات فراهم می‌کند که اطلاعاتی درباره یک درایو منتخب فراهم می‌کند

۲۱-۱-۱- کلاس‌های مربوط به پوشه‌ها و فایل‌ها در NET.

در لیست قبلی دو کلاس برای نشان دادن یک پوشه و دو کلاس برای یک فایل استفاده می‌شدند. متناسب با تعداد دفعات نیاز به دسترسی یک فایل یا پوشه، یکی از این دو نوع کلاس را بکار می‌برند.

File و DirectoryInfo: فقط متدهای ایستا را در بر دارند و هرگز نمونه‌ای از آن ایجاد نمی‌شود. هر زمانی که یک متد عضو این کلاس‌ها را فراخوانی می‌کنید، این کلاس‌ها را با تهیه‌ی مسیر شی مناسب سیستم فایل به کار برید. زمانیکه بخواهید فقط یک عمل روی یک پوشه یا یک فایل انجام دهید، کاربرد این کلاس‌ها موثرتر است، چون آن سربار مربوط به ایجاد نمونه از یک کلاس را صرفه‌جویی می‌کند.

FileInfo, DirectoryInfo: تقریباً همه متدهای عمومی File و DirectoryInfo را به همراه برخی از خصوصیات سازنده‌های public پیاده‌سازی می‌کنند. اما آنها حالت شی را نشان می‌دهند و اعضای این کلاس ایستا نیستند. قبل از تخصیص یک فایل یا پوشه به این کلاس‌ها، باید نمونه‌ای از این کلاس‌ها ایجاد کنید. پس اگر می‌خواهید چندین عمل روی یک شی انجام دهید این کلاس‌ها موثر هستند. و چون آنها اطلاعات تصدیق را در هنگام ایجاد شی سیستم فایل می‌خوانند، نیازی نیست مجدداً این اطلاعات خوانده شده و بررسی گردند. در مقایسه با کلاس‌های بدون حالت، همه جزئیات باید قبل از فراخوانی متدها کنترل گردد.

در این بخش اکثراً با کلاس FileInfo و DirectoryInfo کار می‌کنید، اما برخی از متدهای فراخوانی شده با کلاس‌های DirectoryInfo و File پیاده‌سازی می‌شوند (اگرچه این متدها پارامتر اضافی نیاز دارند). مثال:

```
FileInfo myFile = new FileInfo(@"C:\Program Files\My Program\ReadMe.txt");
myFile.CopyTo(@"D:\Copies\ReadMe.txt");
```

کد زیر نیز همان تاثیر را دارد:

```
File.Copy(@"C:\Program Files\My Program\ReadMe.txt", @"D:\Copies\ReadMe.txt");
```

تکه کد اول زمان بیشتری برای اجرا می‌گیرد. چون نیاز دارد یک نمونه از شی FileInfo ایجاد کند و آن را برای انجام اعمال بعدی روی همان فایل آماده می‌سازد. اما تکه کد دوم نیازی نیست نمونه‌ای از آن شی ایجاد شود.

می‌توانید با رد کردن یک رشته شامل سیستم فایل مورد نظر به سازنده، نمونه‌ای از کلاس FileInfo یا DirectoryInfo ایجاد کنید. کد مربوط به ایجاد پوشه به صورت زیر است:

```
DirectoryInfo myFolder = new DirectoryInfo(@"C:\Program Files");
```

اگر مسیر مورد نظر وجود نداشته باشد، یک استثنای زمان اجرا تولید نخواهد شد، مگر اینکه بخواهید یکی از متدهای این شی را فراخوانی کنید. با استفاده از خصوصیت Exists می‌توان بود یا نبود یک پوشه و فایل را فهمید.

```
FileInfo test = new FileInfo(@"C:\Windows");
```

^۱ Marshaling

```
Console.WriteLine(test.Exists.ToString());
```

خروجی کد قبلی false است، چون c:\windows فایل نیست و اگر بخواهید متد FileInfo.Open() را اجرا کنید یک استثناء رخ می‌دهد.

بعد از اینکه تعیین کردید شی سیستم فایل موجود است، می‌توانید اطلاعاتی درباره آن با استفاده از خصوصیات جدول زیر بیابید.

نام خصوصیت	توصیف
CreateTime	زمان ایجاد فایل یا پوشه
DirectoryName (فقط در FileInfo)	مسیر کامل پوشه‌ی فایل
Parent (فقط در DirectoryInfo)	فهرست پدر یک زیر فهرست مشخص
Exists	آیا فایل یا پوشه مورد نظر وجود دارد
Extension	پسوند فایل را بر می‌گرداند، برای پوشه‌ها رشته خالی بر می‌گردد.
FullName	نام مسیر کامل فایل یا پوشه
LastAccessTime	زمان آخرین دسترسی به فایل یا پوشه
LastWriteTime	زمان آخرین تغییر فایل یا پوشه
Name	نام فایل یا پوشه
Root (فقط در DirectoryInfo)	بخش ریشه مسیر
Length (فقط در FileInfo)	اندازه فایل بر حسب بایت

می‌توانید عملیاتی روی فایل با استفاده از متدهای جدول زیر انجام دهید.

نام	هدف
Create()	یک پوشه یا فایل خالی با نام داده شده ایجاد می‌کند. در FileInfo یک شی Stream بر می‌گرداند که به شما اجازه می‌دهد در فایل بنویسید.
Delete()	فایل یا پوشه را حذف می‌کند. در پوشه‌ها یک گزینه برای حذف بازگشتی وجود دارد.
MoveTo()	یک فایل یا پوشه را تغییر نام یا انتقال می‌دهد.
CopyTo (فقط در FileInfo)	فایل را کپی می‌کند. هیچ متدی برای کپی کردن پوشه‌ها وجود ندارد. باید پوشه‌ها را ایجاد کرده و فایل‌ها را یکی یکی کپی کنید.
GetDirectories (فقط در DirectoryInfo)	یک آرایه از اشیاء DirectoryInfo بر می‌گرداند که همه پوشه‌های داخل این پوشه را نمایش می‌دهد.
GetFiles (فقط در FileInfo)	یک آرایه از اشیاء FileInfo بر می‌گرداند که همه فایل‌های داخل پوشه را

DirectoryInfo	نشان می‌دهد.
(GetFileSystemInfos) (فقط در DirectoryInfo)	اشیاء FileInfo , DirectoryInfo بر می‌گرداند که همه فایل‌ها و پوشه‌های داخل این پوشه را نشان می‌دهد.

توجه: این جدول خصوصیات و متدهای اصلی را لیست می‌کند.

توجه داشته باشید که جداول قبلی بیشتر خصوصیات و متدهای مربوط به خواندن و نوشتن در فایل‌ها را لیست نکرده‌اند. کلاس FileInfo تعدادی متد با عناوین Open(), OpenRead(), OpenText(), OpenWrite(), Create() و CreateText() را پیاده‌سازی می‌کند که اشیاء Stream را بر می‌گرداند. جالب اینکه زمان ایجاد، آخرین زمان دسترسی و آخرین زمان تغییر قابل تغییر هستند.

```
// displays the creation time of a file,
// then changes it and displays it again
FileInfo test = new FileInfo(@"C:\MyFile.txt");
Console.WriteLine(test.Exists.ToString());
Console.WriteLine(test.CreationTime.ToString());
test.CreationTime = new DateTime(۲۰۰۱, ۱, ۱, ۷, ۳۰, ۰);
Console.WriteLine(test.CreationTime.ToString());
```

اجرای این برنامه نتیجه‌ای شبیه زیر تولید می‌کند.

```
True
۶/۵/۲۰۰۵ ۲:۵۹:۳۲ PM
۱/۱/۲۰۰۱ ۷:۳۰:۰۰ AM
```

توانایی تغییر دستی این خصوصیات ابتدا خطرناک به نظر می‌رسد، اما آن می‌تواند خیلی مفید باشد. اگر برنامه‌ای دارید که یک فایل را با خواندن آن و حذف کردن و ایجاد یک فایل جدید با محتویات جدید تغییر می‌دهد، شاید بخواهید تاریخ ایجاد فایل را برای تطابق با تاریخ ایجاد فایل قدیمی اصلی تغییر دهید.

۲۱-۱-۲-کلاس Path

کلاس Path، کلاسی نیست که بخواهید نمونه‌ای از آن تعریف کنید. در عوض، متدهای ایستایی دارد که عملیات روی اسامی مسیرها را ساده‌تر می‌سازد. مثال، فرض کنید می‌خواهید نام مسیر کامل یک فایل را نمایش دهید. فایل README.txt در پوشه c:\MyDocuments است. می‌توانید مسیر آن را با استفاده از کد زیر بدست آورید:

```
Console.WriteLine(Path.Combine(@"C:\My Documents", "ReadMe.txt"));
```

کاربرد کلاس Path، بسیار ساده‌تر از بررسی سمبل‌های جداسازی بصورت دستی است، چون کلاس Path از قالب‌های مختلف مسیرها در سیستم‌عامل‌های مختلف آگاه است. در حال حاضر، ویندوز تنها سیستم‌عاملی است که توسط .Net پشتیبانی می‌شود. اگر .Net روی uinx جابجا شود، Path قادر است خود را با مسیرهای uinx وفق دهد.

بخش بعدی یک مثال ارائه می‌دهد که نشان می‌دهد چگونه فهرست‌ها را browse کنید و خصوصیات فایل‌ها را ببینید.

۲۱-۱-۳-مثال File Browser

این بخش یک برنامه کاربردی ساده بنام FileProperties ارائه می‌دهد که یک واسط کاربردی ساده دارد و می‌تواند سیستم فایل را Browse کرده و زمان ایجاد، زمان آخرین دسترسی، زمان آخرین تغییر و اندازه فایل را ببینید.

روش کار برنامه بصورت زیر است: نام فایل یا پوشه را در کادر متنی بالای پنجره تایپ کرده و روی دکمه Display کلیک می‌شود. اگر مسیر یک فایل را تایپ کنید، جرئیات آن در کادرهای متنی پایین فرم نمایش داده می‌شود و محتویات پوشه در ListBox نمایش داده می‌شود. شکل ۲-۳۴ برنامه کاربردی FileProperties را نشان می‌دهد.

شکل ۲-۳۴



کاربر با کلیک بر روی هر پوشه در لیست سمت راست، می‌تواند به پوشه‌های داخلی حرکت کند و با کلیک بر روی دکمه Up به پوشه پدر حرکت کند. شکل ۲-۳۴ محتویات پوشه My Documents را نشان می‌دهد. کاربر می‌تواند با کلیک بر روی نام فایل در ListBox، آن را انتخاب کند. این عمل خصوصیات فایل را در کادرهای متنی پایین برنامه نشان می‌دهد. شکل ۳-۳۴ را ببینید.

شکل ۳-۳۴



توجه: به دلخواه می‌توانید زمان ایجاد، زمان آخرین دسترسی، زمان آخرین تغییر پوشه‌ها را با استفاده از خصوصیات DirectoryInfo نشان دهید.

یک پروژه استاندارد Application Windows در VS ۲۰۰۵ ایجاد کنید و کادرهای متنی و کادر لیست را از ناحیه کادر ابزار Windows Forms اضافه کنید و کنترل‌ها را به اسامی زیر تغییر نام دهید:

```
listBoxFolders, listBoxFiles, buttonUp, buttonDisplay, textBoxFolder, textBoxInput,
textBoxFileName, textBoxLastWriteTime, textBoxLastAccessTime, textBoxCreationTime,
textBoxFileSize
```

لازم است نشان دهید از فضای نامی System.IO استفاده خواهید کرد.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO;
```

یک فیلد عضو به فرم اصلی اضافه کنید.

```
partial class Form1 : Form
{
private string currentFolderPath;
```

لازم است اداره کننده‌های رویداد را برای رویدادهای تولید شده توسط کاربر اضافه کنید. ورودی‌های ممکن کاربر عبارتند از:

- کاربر روی دکمه Display کلیک می‌کند.
- کاربر روی نام فایل در فایل‌های کادر لیست Files کلیک می‌کند
- کاربر روی نام یک پوشه در کادر لیست Folders کلیک می‌کند.
- کاربر روی دکمه Up کلیک می‌کند .

متد ClearAllFields() محتویات همه کنترل‌های روی فرم را پاک می‌کند. چون خیلی از رویدادها باید قبل از انجام کار خود، همه کنترل‌ها را پاک کنند.

```
protected void ClearAllFields()
{
listBoxFolders.Items.Clear();
listBoxFiles.Items.Clear();
textBoxFolder.Text = "";
textBoxFileName.Text = "";
textBoxCreationTime.Text = "";
textBoxLastAccessTime.Text = "";
textBoxLastWriteTime.Text = "";
textBoxFileSize.Text = "";
}
```

متد DisplayFileInfo() پروسه‌ی نمایش اطلاعات یک فایل داده شده را در کادرهای متنی اداره می‌کند. این متد یک پارامتر می‌گیرد (مسیر کامل نام یک فایل را بصورت رشته می‌گیرد).

```
protected void DisplayFileInfo(string fileFullName)
{
FileInfo theFile = new FileInfo(fileFullName);
if (!theFile.Exists)
throw new FileNotFoundException("File not found: " + fileFullName);
textBoxFileName.Text = theFile.Name;
textBoxCreationTime.Text = theFile.CreationTime.ToLongTimeString();
textBoxLastAccessTime.Text = theFile.LastAccessTime.ToLongDateString();
textBoxLastWriteTime.Text = theFile.LastWriteTime.ToLongDateString();
textBoxFileSize.Text = theFile.Length.ToString() + " bytes";
}
```

اگر مشکلی در پیدا کردن یک فایل باشد، یک استثناء رخ می‌دهد که روال فراخوانی‌کننده این متد، مسؤل کنترل این استثناء است. در نهایت، متد `DisplayFolderList()`، محتویات یک پوشه را در دو کادر لیست نشان می‌دهد. مسیر کامل پوشه به عنوان پارامتر به این متد رد می‌شود.

```
protected void DisplayFolderList(string folderFullName)
{
    DirectoryInfo theFolder = new DirectoryInfo(folderFullName);
    if (!theFolder.Exists)
        throw new DirectoryNotFoundException("Folder not found: " + folderFullName);
    ClearAllFields();
    textBoxFolder.Text = theFolder.FullName;
    currentFolderPath = theFolder.FullName;
    // list all subfolders in folder
    foreach(DirectoryInfo nextFolder in theFolder.GetDirectories())
        listBoxFolders.Items.Add(nextFolder.Name);
    // list all files in folder
    foreach(FileInfo nextFile in theFolder.GetFiles())
        listBoxFiles.Items.Add(nextFile.Name);
}
```

حال اداره کننده‌های رویدادها را بررسی کنید. رویداد مربوط به کلیک روی دکمه `Display` پیچیده‌ترین آنها است. چون لازم است سه حالت ممکن برای متنی که کاربر در کادر متنی وارد کرده است را اداره کند. برای مثال، ممکن است نام یک فایل، مسیر کامل پوشه یا هیچکدام باشد.

```
protected void OnDisplayButtonClick(object sender, EventArgs e)
{
    try
    {
        string folderPath = textBoxInput.Text;
        DirectoryInfo theFolder = new DirectoryInfo(folderPath);
        if (theFolder.Exists)
        {
            DisplayFolderList(theFolder.FullName);
            return;
        }
        FileInfo theFile = new FileInfo(folderPath);
        if (theFile.Exists)
        {
            DisplayFolderList(theFile.Directory.FullName);
            int index = listBoxFiles.Items.IndexOf(theFile.Name);
            listBoxFiles.SetSelected(index, true);
            return;
        }
        throw new FileNotFoundException("There is no file or folder with "
            + "this name: " + textBoxInput.Text);
    }
    catch(Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

در این کد، ابتدا به ازای متن تایپ شده توسط کاربر، به کمک نمونه‌هایی از `DirectoryInfo` و `FileInfo` و خصوصیت `Exist`، فایل یا پوشه بودن آن را بررسی می‌کند. اگر هیچکدام نبود، یک استثناء بروز می‌دهد. اگر فایل یا پوشه باشد، رویدادهایی را به ترتیب فراخوانی می‌کند.

کد زیر به اداره کننده‌ی رویداد انتخاب یک عضو در کادر لیست `Files` مربوط است (توسط کاربر یا از طریق برنامه نویسی). آن نام کامل فایل انتخابی را ایجاد کرده و به متد `DisplayFileInfo()` رد می‌کند.

```
protected void OnListBoxFilesSelected(object sender, EventArgs e)
{

```

```
try
{
string selectedString = listBoxFiles.SelectedItem.ToString();
string fullFileName = Path.Combine(currentFolderPath, selectedString);
DisplayFileInfo(fullFileName);
}
catch(Exception ex)
{
MessageBox.Show(ex.Message);
}
}
```

اداره کننده رویداد مربوط به انتخاب یک پوشه در کادر لیست Folders شبیه این است. با این استثناء که متد DisplayFolderList() را برای بهنگام سازی محتوای کادرهای لیست فراخوانی می‌کند.

```
protected void OnListBoxFoldersSelected(object sender, EventArgs e)
{
try
{
string selectedString = listBoxFolders.SelectedItem.ToString();
string fullPathName = Path.Combine(currentFolderPath, selectedString);
DisplayFolderList(fullPathName);
}
catch(Exception ex)
{
MessageBox.Show(ex.Message);
}
}
```

در نهایت، زمانی که روی دکمه up کلیک شود، باید متد DisplayFolderList() فراخوانی شود. به استثناء اینکه باید مسیر کامل پوشه پدر را جهت نمایش بدست آورید. این عمل با خصوصیت FileInfo.DirectoryName انجام می‌شود که مسیر کامل پوشه پدر را بر می‌گرداند.

```
protected void OnUpButtonClick(object sender, EventArgs e)
{
try
{
string folderPath = new FileInfo(currentFolderPath).DirectoryName;
DisplayFolderList(folderPath);
}
catch(Exception ex)
{
MessageBox.Show(ex.Message);
}
}
```

۲۱-۲-انتقال، کپی و حذف فایل‌ها

همانطور که قبلاً شرح داده شده، انتقال و حذف فایل‌ها و پوشه‌ها بوسیله متدهای MoveTo() و Delete() از کلاس‌های FileInfo و DirectoryInfo انجام می‌شود. متدهای معادل آنها در کلاسهای File و DirectoryInfo متدهای Move() و Delete() هستند. در مورد کاربرد این متدها کاملاً هوشیار باشید. این بخش کاربرد آنها را در موارد خاص با فراخوانی متدهای ایستای Move()، Copy() و Delete() از کلاس File ارائه می‌دهد. برای این کار مثال قبلی FileProperties را ایجاد کرده و آن را FilePropertiesAndMovement بنامید. این مثال ویژگی اضافی دارد. هنگام نمایش خصوصیات یک فایل گزینه‌هایی برای حذف، انتقال یا کپی فایل به موقعیت دیگر می‌دهد.

۲۱-۲-۱-مثال FilePropertiesAndMovement

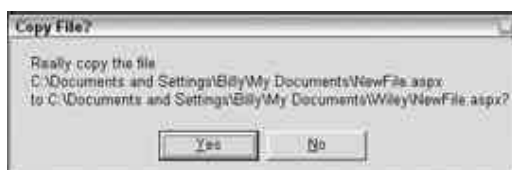
شکل ۴-۲۴ واسط کاربری برنامه جدید را نشان می‌دهد.

شکل ۴-۳۴



همانطور که می‌توانید ببینید، این برنامه در ظاهر شبیه FileProperties است. با استثناء اینکه سه دکمه و یک کادر متنی در پایین پنجره اضافه شده است. این کنترل‌ها فقط زمانی فعال هستند که خصوصیات یک فایل نمایش داده می‌شوند، در بقیه حالات، غیر فعال هستند. زمانی که خصوصیات یک فایل منتخب نمایش داده می‌شوند، FilePropertiesAndMovement بطور اتوماتیک مسیر کامل فایل را در کادر متنی پایین جهت ویرایش قرار می‌دهد. کاربران برای انجام عملیات مناسب می‌توانند روی هر کدام از دکمه‌ها کلیک کنند، که یک کادر پیام برای تایید عمل کاربر باز می‌شود. (شکل ۵-۳۴ را ببینید).

شکل ۵-۳۴



زمانیکه که کاربر روی Yes کلیک کند، عمل شروع خواهد شد. در بعضی مواقع، عمل انجام شده باعث ایجاد یک حالت نادرستی می‌شود (مثلاً تغییر نام یک پوشه یا یک فایل، حذف یک پوشه). برای حل این مشکل، همه کنترل‌ها بعد از انجام هر کاری دوباره مقداردهی می‌شوند.

۲۱-۲-۲- بررسی کد برنامه FilePropertiesAndMovement

برای کدنویسی این پروسه، لازم است کنترل‌های مرتبط و اداره کننده‌های رویداد آنها را به کد FileProperties اضافه کنید. به کنترل‌های جدید اسامی buttonMoveTo، buttonCopyTo، buttonDelete، textBoxNewPath، داده شده است.

ابتدا اداره کننده‌ی رویدادی که زمان کلیک کاربر روی دکمه Delete فراخوانی می‌شود را ببینید:

```
protected void OnDeleteButtonClick(object sender, EventArgs e)
{
    try
```

```

{
    string filePath = Path.Combine(currentFolderPath,
    textBoxFileName.Text);
    string query = "Really delete the file\n" + filePath + "?";
    if (MessageBox.Show(query,
    "Delete File?", MessageBoxButtons.YesNo) == DialogResult.Yes)
    {
        File.Delete(filePath);
        DisplayFolderList(currentFolderPath);
    }
}
catch(Exception ex)
{
    MessageBox.Show("Unable to delete file. The following exception"
    + " occurred:\n" + ex.Message, "Failed");
}
}

```

در کد این متد بلوک try به دلیل ریسک وقوع یک استثناء قرار داده می‌شود. برای مثال، شما جواز حذف فایل را نداشته باشید یا فایل توسط پروسه‌ی دیگری منتقل شده است، ولی هنوز برنامه شما آن را نشان می‌دهد. مسیر فایل مورد نظر جهت حذف را از فیلد CumentParentPath بسازید که شامل پوشه‌ی پدر و متن داخل کادر متنی textBoxFileName است. متدهای انتقال و کپی در روشی مشابه ساختار بندی می‌شوند.

```

protected void OnMoveButtonClick(object sender, EventArgs e)
{
    try
    {
        string filePath = Path.Combine(currentFolderPath,
        textBoxFileName.Text);
        string query = "Really move the file\n" + filePath + "\nto "
        + textBoxNewPath.Text + "?";
        if (MessageBox.Show(query,
        "Move File?", MessageBoxButtons.YesNo) == DialogResult.Yes)
        {
            File.Move(filePath, textBoxNewPath.Text);
            DisplayFolderList(currentFolderPath);
        }
    }
    catch(Exception ex)
    {
        MessageBox.Show("Unable to move file. The following exception"
        + " occurred:\n" + ex.Message, "Failed");
    }
}

protected void OnCopyButtonClick(object sender, EventArgs e)
{
    try
    {
        string filePath = Path.Combine(currentFolderPath,
        textBoxFileName.Text);
        string query = "Really copy the file\n" + filePath + "\nto "
        + textBoxNewPath.Text + "?";
        if (MessageBox.Show(query,
        "Copy File?", MessageBoxButtons.YesNo) == DialogResult.Yes)
        {
            File.Copy(filePath, textBoxNewPath.Text);
            DisplayFolderList(currentFolderPath);
        }
    }
    catch(Exception ex)
    {
        MessageBox.Show("Unable to copy file. The following exception"
        + " occurred:\n" + ex.Message, "Failed");
    }
}

```

}

هنوز کارتان کامل نشده است و لازم است مطمئن شوید کادر متنی و دکمه‌های جدید در زمان‌های مناسب فعال و غیرفعال می‌شوند. برای فعال کردن آنها در هنگام نمایش محتویات یک فایل، کد زیر را به DisplayFileInfo() اضافه کنید.

```
protected void DisplayFileInfo(string fileFullName)
{
    FileInfo theFile = new FileInfo(fileFullName);
    if (!theFile.Exists)
        throw new FileNotFoundException("File not found: " + fileFullName);
    textBoxFileName.Text = theFile.Name;
    textBoxCreationTime.Text = theFile.CreationTime.ToString();
    textBoxLastAccessTime.Text = theFile.LastAccessTime.ToString();
    textBoxLastWriteTime.Text = theFile.LastWriteTime.ToString();
    textBoxFileSize.Text = theFile.Length.ToString() + " bytes";
    // enable move, copy, delete buttons
    textBoxNewPath.Text = theFile.FullName;
    textBoxNewPath.Enabled = true;
    buttonCopyTo.Enabled = true;
    buttonDelete.Enabled = true;
    buttonMoveTo.Enabled = true;
}
```

لازم است یک تغییری در DisplayFolderList داده شود.

```
protected void DisplayFolderList(string folderFullName)
{
    DirectoryInfo theFolder = new DirectoryInfo(folderFullName);
    if (!theFolder.Exists)
        throw new DirectoryNotFoundException("Folder not found: " + folderFullName);
    ClearAllFields();
    DisableMoveFeatures();
    textBoxFolder.Text = theFolder.FullName;
    currentFolderPath = theFolder.FullName;
    // list all subfolders in folder
    foreach(DirectoryInfo nextFolder in theFolder.GetDirectories())
        listBoxFolders.Items.Add(nextFolder.Name);
    // list all files in folder
    foreach(FileInfo nextFile in theFolder.GetFiles())
        listBoxFiles.Items.Add(nextFile.Name);
}
```

DisableMoveFeatures یک تابع سودمند کوچک است که کنترل‌های جدید را غیرفعال می‌کند.

```
void DisableMoveFeatures()
{
    textBoxNewPath.Text = "";
    textBoxNewPath.Enabled = false;
    buttonCopyTo.Enabled = false;
    buttonDelete.Enabled = false;
    buttonMoveTo.Enabled = false;
}
```

لازم است کدی به متد ClearAllFields() اضافه کنید تا محتوای کادرهای متنی جدید را پاک کند.

```
protected void ClearAllFields()
{
    listBoxFolders.Items.Clear();
    listBoxFiles.Items.Clear();
    textBoxFolder.Text = "";
    textBoxFileName.Text = "";
    textBoxCreationTime.Text = "";
    textBoxLastAccessTime.Text = "";
    textBoxLastWriteTime.Text = "";
    textBoxFileSize.Text = "";
    textBoxNewPath.Text = "";
}
```

در حال حاضر کد کامل است.

۲۱-۳- خواندن و نوشتن در فایل‌ها

در اصل، خواندن و نوشتن در فایل‌ها بسیار ساده است. با این وجود، از طریق اشیاء `FileInfo` و `DirectoryInfo` انجام نمی‌شود. در عوض با استفاده از چارچوب `.Net ۲.۰` از طریق کلاس `File` می‌توانید آن کار را انجام دهید. بعداً خواهیم دید که از طریق شی `Stream` نیز می‌توان این کار را انجام داد.

۲۱-۳-۱- خواندن یک فایل

برای مثال خواندن یک فایل، یک برنامه کاربردی `Windows Form` ایجاد کنید که یک کادر متنی و یک دکمه و یک کادر متنی چند خطی داشته باشد. فرم شما باید شبیه شکل ۳۴-۶ ظاهر گردد

شکل ۳۴-۶



مقصود این فرم آن است که ابتدا کاربر مسیر فایل مشخص را در کادر متنی وارد کرده و روی دکمه کلیک کند. سپس برنامه کاربردی محتوی فایل را خوانده و در کادر متنی چند خطی نمایش دهد. کد مثال بصورت زیر می‌باشد:

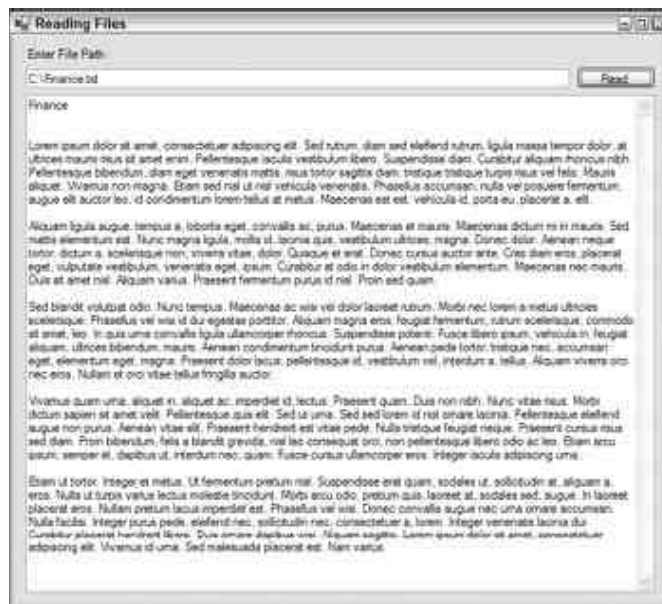
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO;
namespace ReadingFiles
{
    partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
```



```
{
textBox2.Text = File.ReadAll(textBox1.Text);
}
}
```

در این مثال فضای نامی `System.IO` در ابتدای برنامه اضافه می‌گردد و رویداد `button1_Click` برای خواندن محتویات فایل است و متد `File.ReadAll()` محتویات فایل را می‌خواند. این متد ابتدا فایل را باز کرده و محتویات آن را خوانده و سپس فایل را می‌بندد. مقدار بازگشتی متد، رشته‌ای است که همه محتویات فایل را در بر دارد. شکل ۷-۳۴ را ببینید.

شکل ۷-۳۴



در مثال قبلی نشانه‌ی متد `File.ReadAll` بصورت زیر است.

```
File.ReadAll(filePath);
```

حالت دیگر این نشانه، نحوه‌ی کدگذاری فایل را مشخص می‌کند.

```
File.ReadAll(filePath, Encoding);
```

با استفاده از این گزینه، می‌توان در حین خواندن فایل، نحوه‌ی کدگذاری آن را مشخص کرد. پس می‌توانید چیزی شبیه این انجام دهید:

```
File.ReadAll(textBox1.Text, Encoding.ASCII);
```

متدهای دیگری نیز وجود دارد. متد `ReadAllBytes` یک فایل باینری را باز کرده و محتویات آن را در یک آرایه‌ی بایتی می‌خواند. این متدها زیاد جالب نیستند، چون بیشتر هدف این است که فایل‌ها خط به خط خوانده شوند. در این حالت، می‌توانید از متد `ReadAllLines` استفاده کنید که این نوع عمل را مجاز می‌دارد.

۲۱-۳-۲-نوشتن به یک فایل

همانند خواندن از فایل‌ها که توسط چارچوب `Net ۲.۰` بسیار ساده شده است، نوشتن در یک فایل نیز بسیار آسان است. همانطور که FCL متدهای `ReadAll`، `ReadAllLines` و `ReadAllBytes` را برای خواندن داده فراهم ساخته است، برای نوشتن به یک فایل نیز متدهای `WriteAll`، `WriteAllBytes` و `WriteAllLines` را فراهم کرده است.

به عنوان مثالی برای نوشتن به یک فایل، همان برنامه Windows Forms را بکار برید. اما کادر متنی چند خطی را برای وارد کردن اطلاعات به یک فایل بکار برید. کد رویداد Click\button بصورت زیر ظاهر خواهد شد.

```
private void button1_Click(object sender, EventArgs e)
{
    File.WriteAllText(textBox1.Text, textBox2.Text);
}
```

برنامه را ایجاد کرده و اجرا کنید. در کادر متنی اولی c:\Testing.txt را تایپ کرده و در کادر متنی دوم محتوای تصادفی تایپ کنید. سپس روی دکمه کلیک کنید. هیچ چیز بصری اتفاق نخواهد افتاد. اما اگر به درایو ریشه C:\ نگاه کنید، فایل Testing.txt را با محتوای مشخص شده خواهید دید.

ابتدا متد WriteAll در موقعیت مشخص شده، فایل متنی جدیدی ایجاد می‌کند و محتوای مورد نظر را در آن نوشته و ذخیره می‌کند و سپس آن را می‌بندد.

اگر برنامه را مجدداً اجرا کنید و فایلی همانم با قبلی ولی با محتوای جدید ایجاد کنید، هیچ اتفاق خاصی نمی‌افتد و فایل قبلی رونویسی می‌شود. مهم است توجه کنید که محتوای جدید به فایل اضافه نمی‌شود، بلکه محتوای جدید بطور کامل محتوای قبلی را رونویسی می‌کند. در حقیقت، همه متدهای WriteAll، WriteAllBytes و WriteAllLines فایل‌ها را موجود را رونویسی می‌کنند. پس هنگام استفاده از این متدها مواظب باشید.

در مثال قبلی متد WriteAll نشانه زیر را بکار می‌برد.

```
File.WriteAllText(filePath, Content)
```

می‌توان کدگذاری فایل جدید را مشخص کرد.

```
File.WriteAllText(filePath, Content, Encoding)
```

متد WriteAllBytes به شما اجازه می‌دهد محتوای یک آرایه‌ی بایتی را در یک فایل بنویسید و متد WriteAllLines به شما اجازه می‌دهد یک آرایه از رشته‌ها را به یک فایل بنویسید. برای مثال، کد زیر را در اداره کننده رویداد وارد کنید.

```
private void button1_Click(object sender, EventArgs e)
{
    string[] movies = {"Grease", "Close Encounters of the Third Kind", "The Day After Tomorrow"};
    File.WriteAllLines("C:\Testing.txt", movies);
}
```

حال روی دکمه کلیک کنید. برنامه یک فایل Testing.txt با محتوای زیر به شما خواهد داد.

```
Grease
Close Encounters of the Third Kind
The Day After Tomorrow
```

متد WriteAllLines هر عنصر از آرایه‌ی رشته‌ای را در یک سطر فایل متنی می‌نویسد.

۲۱-۳-۳- جریان‌ها

هدف جریان^۱ از زمان‌های گذشته بوده است. شی Stream برای انتقال داده استفاده می‌شود. داده می‌تواند در یکی از دو جهت انتقال داده شود.

- اگر داده‌ها از یک منبع بیرونی به برنامه شما انتقال داده شود، آن را خواندن از جریان می‌گویند.
- اگر داده از برنامه شما به یک منبع بیرونی منتقل شود، آن را نوشتن به یک جریان می‌گویند.

اغلب اوقات منبع بیرونی یک فایل است، اما ضرورتاً اینطور نیست. حالت‌های ممکن دیگر عبارتند از:

^۱ Stream

- خواندن و نوشتن روی شبکه با استفاده از پروتکل شبکه
- خواندن و نوشتن به یک لوله نام‌گذاری شده (pipe)
- خواندن و نوشتن فضایی از حافظه

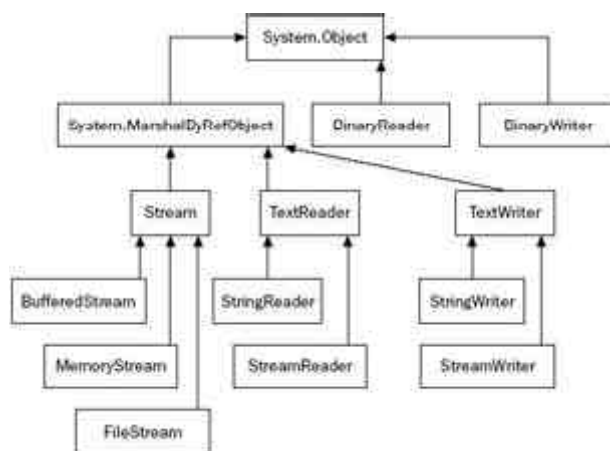
مایکروسافت یک کلاس پایه‌ی Net برای نوشتن و خواندن از حافظه بنام `System.IO.MemoryStream` تهیه کرده است. کلاس `System.Net.Sockets.NetworkStream` داده شبکه را اداره می‌کند. کلاس پایه جریان برای خواندن و نوشتن در لوله‌ها وجود ندارد، ولی می‌توانید با ارث‌بری از کلاس کلی `Stream` یک کلاس برای آن ایجاد کنید. جریان هیچ فرضی درباره طبیعت منبع خارجی در نظر نمی‌گیرد.

منبع خارجی می‌تواند یک متغیر در داخل کد خودتان باشد. این پارادوکس است، اما تکنیک کاربرد جریان‌ها برای انتقال داده مابین متغیرها، می‌تواند در هنگام تبدیل انواع داده‌ای مفید باشد. زبان C# شبیه این را برای تبدیل انواع داده‌ای صحیح و رشته‌ها یا قالب‌بندی رشته‌ها بکار می‌برد.

مزیت وجود یک شی مجزا برای انتقال داده به جای استفاده از کلاس‌های `FileInfo` یا `DirectoryInfo` این است که با متمایز کردن انتقال داده از منبع داده خاص، جابجا کردن منابع داده بسیار آسان است. کلاس‌های `StringWriter`، `StreamReader` بخشی از همان درخت وارث هستند که بعداً برای خواندن یا نوشتن در فایل‌های متنی با عنوان کلاس‌های `StreamWriter`، `StreamReader` استفاده خواهند شد.

شکل ۸-۳۴ سلسه مراتب واقعی کلاس‌های مرتبط با جریان را در فضای نامی `System.IO` نشان می‌دهد.

شکل ۸-۳۴



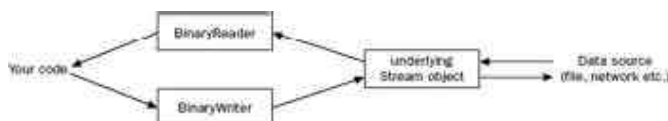
تا آنجا که به خواندن و نوشتن فایل‌ها اهمیت داده می‌شود، کلاس‌های که بیشتر برای ما اهمیت دارند، عبارتند از:

`FileStream`: این کلاس قصد دارد داده‌های دودویی را در یک فایل دودویی خوانده یا بنویسد. با این وجود می‌توانید آن را برای خواندن یا نوشتن هر فایل‌ی بکار ببرید.

`StreamReader`، `StreamWriter`: این کلاس‌ها مخصوصاً برای خواندن و نوشتن فایل‌های متنی طراحی شده‌اند.

اگرچه کلاس‌های `BinaryReader`، `BinaryWriter` در مثال‌های ما استفاده نمی‌شوند، آنها نیز می‌توانند مفید باشید. در واقع این کلاس‌ها جریان‌ها را پیاده‌سازی نمی‌کنند، اما قادر هستند بسته‌بندی اشیاء دیگر جریان را فراهم کنند. کلاس‌های `BinaryReader`، `BinaryWriter` دستورات قالب‌بندی اضافی برای داده‌های دودویی را فراهم می‌کنند، که به شما اجازه می‌دهد محتوی متغیرهای C# را به جریان مرتبط نوشته یا از آن بخوانید. شکل ۹-۳۴ را ببینید.

شکل ۹-۳۴



تفاوت مابین کاربرد این کلاس‌ها یا کاربرد اشیاء جریان اصلی این است که اشیاء جریان پایه با بایت‌ها کار می‌کنند. برای مثال: فرض کنید می‌خواهید یک متغیر از نوع Long را در یک فایل باینری بنویسید. اگر بخواهید از کلاس جریان معمولی استفاده کنید، باید هر هشت بایت را بطور صریح در آن بنویسید، ولی با استفاده از متد Write از کلاس BinaryWriter می‌توان این متغیر را یکجا به عنوان پارامتر به آن ارسال کرد تا در فایل باینری نوشته شود و بطور مشابه متد BinaryReader.Read()، ۸ بایت از جریان استخراج کرده و در متغیر از نوع Long می‌ریزد.

۲۱-۳-۴- جریان‌های بافر شده

به دلیل بهره‌روی، خروجی در زمان خواندن و یا نوشتن فایل، بافر می‌شود. بدین معنی که اگر برنامه‌ی شما بخواهد دوبایت بعدی یک جریان فایل را بخواند، جریان مربوطه درخواست خود را به سیستم‌عامل رد می‌کند. سپس سیستم‌عامل به جای اینکه محل فایل را پیدا کرده و آن را جهت خواندن باز کند. بلوک بزرگی از داده‌ها را از روی دیسک به ناحیه خاصی از حافظه بنام بافر بازبایی می‌کند. در دستورهای خواندن و نوشتن بعدی، به جای ارجاع مستقیم به فایل روی دیسک از این بافر استفاده خواهد کرد.

۲۱-۳-۵- خواندن و نوشتن در فایل‌های دودویی

خواندن و نوشتن در فایل‌های دودویی می‌تواند از طریق کلاس FileStream انجام شود.

کلاس FileStream

یک نمونه از کلاس FileStream برای خواندن یا نوشتن در فایل‌ها استفاده می‌شود. به منظور ایجاد یک شیء FileStream، چهار تکه اطلاعات لازم دارید.

- File: فایلی که می‌خواهید به آن دسترسی کنید.
- Mode: شیوه‌ی بازکردن فایل را مشخص می‌کند. برای مثال، آیا شما قصد دارید یک فایل جدید ایجاد کنید یا فایل موجود را باز کنید؟ و اگر فایل موجود را باز کردید قصد دارید اطلاعات آن رونویسی شوند یا اطلاعات جدید به آن اضافه گردند؟
- Access: نحوه‌ی دسترسی شما به فایل را مشخص می‌کند. برای مثال می‌خواهید یک فایل را بخوانید یا بنویسید یا هر دو کار را انجام دهید.
- Share: مشخص می‌کند دسترسی به این فایل منحصر است یا اجازه می‌دهید بطور همزمان جریان‌های دیگر نیز به این فایل دسترسی داشته باشند و اینکه دسترسی جریان‌های دیگر خواندن و نوشتن یا هر دو می‌تواند باشد.

بخش اول اطلاعات رشته‌ای است که نام و مسیر کامل فایل را شامل است. سه بخش باقیمانده بوسیله انواع شمارشی Net. بنام های FileAccess, FileShare, FileMode نمایش داده می‌شوند. مقادیر این انواع شمارشی در جدول زیر لیست می‌شوند. آنها بیانگر کار خود هستند.

نوع شمارشی	مقادیر
FileMode	Append-Create-CreateNew-Open-OpenOrCreate-Truncate
FileAccess	Read-ReadWrite-Write

Inheritable-None-Read-ReadWrite-Write	FileShare
---------------------------------------	-----------

اگر مد یک فایل با حالت آن ناسازگار باشد، استثنائ‌هایی روی می‌دهد. اگر فایل وجود نداشته باشد، Append، Open و Truncate استثناء تولید می‌کنند و اگر فایل موجود باشد، CreateNew به یک استثناء منجر می‌شود. انواع شمارشی FileShare، FileAccess بیت‌ی هستند و می‌توانید مقادیر آنها را با عملگر بیت‌ی or (|) ترکیب کنید.

FileStream چندین سازنده دارد. سه سازنده‌ی ساده آن بصورت زیر کار می‌کنند:

```
// creates file with read-write access and allows other streams read access
FileStream fs = new FileStream(@"C:\C# Projects\Project.doc",
    FileMode.Create);
// as above, but we only get write access to the file
FileStream fs2 = new FileStream(@"C:\C# Projects\Project2.doc",
    FileMode.Create, FileAccess.Write);
// as above but other streams don't get
// fs2 is open
FileStream fs3 = new FileStream(@"C:\C# Projects\Project3.doc",
    FileMode.Create, FileAccess.Write, FileShare.None);
```

همانطور که این کد نشان می‌دهد، overloadهای این سازنده‌ها روی مقادیر پیش‌فرض پارامتر سوم و چهارم تأثیر می‌گذارند (FileShare.Read, FileAccess.ReadWrite). ایجاد یک جریان فایل از یک نمونه FileInfo امکان‌پذیر است.

```
FileInfo myFile1 = new FileInfo(@"C:\C# Projects\Project1.doc");
FileStream fs1 = myFile1.OpenRead();
FileInfo myFile2 = new FileInfo(@"C:\C# Projects\Project2.doc");
FileStream fs2 = myFile2.OpenWrite();
FileInfo myFile3 = new FileInfo(@"C:\C# Projects\Project3.doc");
FileStream fs3 = myFile3.Open(FileMode.Append, FileAccess.Write,
    FileShare.None);
FileInfo myFile4 = new FileInfo(@"C:\C# Projects\Project4.doc");
FileStream fs4 = myFile4.Create();
```

متد FileInfo.OpenRead() یک جریان با دسترسی فقط خواندنی به یک فایل موجود تهیه می‌کند. در حالی که متد FileInfo.OpenWrite() دسترسی خواندن و نوشتن به شما می‌دهد. متد FileInfo.Open() پارامترهای مد، حالت و اشتراک را بطور صریح می‌خواهد. البته بعد از استفاده یک جریان، باید آن را ببندید.

```
fs.Close();
```

بستن یک جریان، منابع تخصیص یافته به آن را آزاد می‌کند و به برنامه‌های دیگر اجازه می‌دهد تا جریان‌های دیگر را روی همان فایل تنظیم کنند. در بین باز کردن تا بستن جریان، شما داده‌هایی را روی فایل نوشته یا از آن خواهید خواند. FileStream چند متد برای انجام اینکار پیاده‌سازی می‌کند.

متد ReadByte ساده‌ترین راه خواندن داده است. آن یک بایت از جریان را می‌گیرد و نتیجه را به یک مقدار صحیح مابین ۰ تا ۲۵۵ قالب‌بندی می‌کند. اگر به انتهای جریان رسیده باشد، مقدار ۱- بر می‌گرداند.

```
int NextByte = fs.ReadByte();
```

اگر قصد دارید در یک لحظه چندین بایت را بخوانید، می‌توانید متد Read را فراخوانی کنید. تعداد معینی بایت را خوانده و در یک آرایه قرار می‌دهد. متد Read() تعداد بایت‌های واقعا خوانده شده را بر می‌گرداند. اگر مقدار صفر برگرداند، در انتهای جریان هستید. دستور زیر داده‌ها را خوانده و در یک آرایه‌ی بایتی به نام ByteArray قرار می‌دهد.

```
int nBytesRead = fs.Read(ByteArray, 0, nBytes);
```

پارامتر دوم یک آفست است که محل شروع پرکردن خانه‌های آرایه را مشخص می‌کند و پارامتر سوم تعداد بایت‌ها جهت خواندن است.

اگر بخواهید داده‌ها را در یک فایل بنویسید، دو متد وجود دارد: `WriteByte()` و `Write()`. متد `WriteByte()` یک بایت را به جریان می‌نویسد.

```
byte NextByte = \00;
fs.WriteByte(NextByte);
```

از طرفی دیگر، متد `Write` یک آرایه از بایت‌ها را می‌نویسد. برای مثال، اگر آرایه `ByteArray` با مقادیر خاص مقداردهی اولیه شده باشد، می‌توانیم کد زیر را برای نوشتن `nByte` اول آن به فایل بکار ببریم:

```
fs.Write(ByteArray, 0, nBytes);
```

۲۱-۳-۶- خواندن و نوشتن در فایل‌های متنی

کلاس `StreamReader`

`StreamReader` برای خواندن فایل‌های متنی استفاده می‌شود. ایجاد یک نمونه `StreamReader` خیلی ساده‌تر از ایجاد یک نمونه `FileStream` است، چون بعضی از گزینه‌های `FileStream` لازم نیست. نیازی نیست نوع دسترسی و مد فایل مشخص گردد، چون به طور پیش‌فرض مشخص است.

- لازم است متدهای کدگذاری مختلف را بدانید تا در آغاز کار با فایل نوع کدگذاری را مشخص کنید.
- می‌توانید به جای تهیه نام فایل، ارجاعی به یک جریان دیگر تهیه کنید.

`StreamReader` می‌تواند برای خواندن و پردازش داده از هر نوع منبع بکار برد. نتیجه اینکه، تعداد زیادی سازنده دارد و مجموعه‌ای از متدهای `FileInfo` همچون `CreateText()` و `OpenText()` یک ارجاع `StreamReader` برمی‌گرداند. چند تا از سازنده‌های `StreamReader` به صورت زیر هستند:

```
StreamReader sr = new StreamReader(@"C:\My Documents\ReadMe.txt");
```

```
StreamReader sr = new StreamReader(@"C:\My Documents\ReadMe.txt", Encoding.UTF8);
```

با فرض اینکه `fs` یک نمونه از `FileStream` است

```
FileStream fs = new FileStream(@"C:\My Documents\ReadMe.txt",
    FileMode.Open, FileAccess.Read, FileShare.None);
StreamReader sr = new StreamReader(fs);
```

در صورتی که `myfile` یک نمونه از `FileInfo` باشد

```
FileInfo myFile = new FileInfo(@"C:\My Documents\ReadMe.txt");
StreamReader sr = myFile.OpenText();
```

مانند `FileStream` بعد از استفاده `StreamReader` آن را ببندید.

متدهای کلاس `StreamReader`

`ReadLine()`: خط جاری را تا انتها خوانده و در یک رشته برمی‌گردد.

```
string nextLine = sr.ReadLine();
```

`ReadToEnd()`: از محل جاری تا انتهای فایل را خوانده و در یک رشته برمی‌گرداند.

```
string restOfStream = sr.ReadToEnd();
```

`Read()`: می‌تواند برای خواندن یک یا چند کاراکتر استفاده شود.

```
int nextChar = sr.Read();
// to read \00 characters in.
int nChars = \00;
char [] charArray = new char[nChars];
int nCharsRead = sr.Read(charArray, 0, nChars);
```

nChars تعداد کارکترهای درخواستی است و nCharRead تعداد کارکترهای واقعا خوانده شده است.

کلاس StreamWriter

اساسا همانند StreamReader کار می‌کند. به استثناء اینکه، فقط می‌توانید برای نوشتن یک فایل بکار برید. حالت‌های ممکن سازندهی StreamWriter به صورت زیر هستند.

```
StreamWriter sw = new StreamWriter(@"C:\My Documents\ReadMe.txt");
```

نحوه‌ی کدگذاری بطور پیش‌فرض UTF-8 است. اگر بخواهید نحوه‌ی کدگذاری را مشخص کنید، می‌توانید از سازنده‌ی زیر استفاده کنید. پارامتر دوم می‌گوید: آیا فایل برای اضافه کردن داده‌ها باز شود یا نه.

```
StreamWriter sw = new StreamWriter(@"C:\My Documents\ReadMe.txt", true,
Encoding.ASCII);
```

در صورتیکه fs یک نمونه از FileStream باشد:

```
FileStream fs = new FileStream(@"C:\My Documents\ReadMe.txt", FileMode.CreateNew,
; (FileAccess.Write, FileShare.Read
; (StreamWriter sw = new StreamWriter(fs
```

اگر myFile نمونه‌ای از FileInfo باشد:

```
FileInfo myFile = new FileInfo(@"C:\My Documents\NewFile.txt");
StreamWriter sw = myFile.CreateText();
```

همانند دیگر کلاس‌های جریان، بعد از استفاده StreamWriter، باید آن را ببندید.

متدهای کلاس StreamWriter

متد Write(): با استفاده از این متد می‌توانید یک کارکتر یا یک رشته یا یک آرایه کارکتری را به جریان بنویسید.

```
string nextLine = "Groovy Line";
sw.Write(nextLine);
char nextChar = 'a';
sw.Write(nextChar);
char [] charArray = new char[100];
// initialize these characters
sw.Write(charArray);
```

می‌توانید بخشی از یک آرایه کاراکتری را در جریان بنویسید.

```
int nCharsToWrite = 50;
int startAtLocation = 20;
char [] charArray = new char[100];
// initialize these characters
sw.Write(charArray, startAtLocation, nCharsToWrite);
```

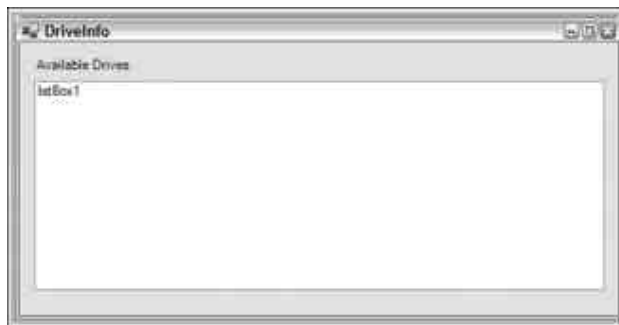
۲۱-۳-۷ رمزنگاری با کلاس CryptoStream

۲۱-۴ خواندن اطلاعات درایو

علاوه بر کار با فایل‌ها و فهرست‌ها، چارچوب .Net ۲.۰ توانایی خواندن اطلاعات یک درایو خاص را معرفی می‌کند. این با استفاده از کلاس DriveInfo انجام می‌شود. کلاس DriveInfo می‌تواند سیستم را جهت بدست آوردن درایوهای آن پویش کند و سپس اطلاعات دقیق در مورد هر درایو را در اختیار شما قرار دهد.

مثال: یک برنامه Windows Form ساده ایجاد کنید که یک ListBox ساده دارد (همانند شکل ۲۱-۱۰).

شکل ۲۱-۱۰



بعد از تنظیم فرم، کد دو رویداد (رویداد بارگذاری فرم و رویداد انتخاب از لیست) را اضافه کنید. کد شما بصورت زیر نمایش داده خواهد شد.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO;
namespace DriveInfo
{
    partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            DriveInfo[] di = DriveInfo.GetDrives();
            foreach (DriveInfo itemDrive in di)
            {
                listBox1.Items.Add(itemDrive.Name);
            }
        }
        private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
            DriveInfo di = new DriveInfo(listBox1.SelectedItem.ToString());
            MessageBox.Show("Available Free Space: " + di.AvailableFreeSpace + "\n" +
                "Drive Format: " + di.DriveFormat + "\n" +
                "Drive Type: " + di.DriveType + "\n" +
                "Is Ready: " + di.IsReady.ToString() + "\n" +
                "Name: " + di.Name + "\n" +
                "Root Directory: " + di.RootDirectory + "\n" +
                "ToString() Value: " + di.ToString() + "\n" +
                "Total Free Space: " + di.TotalFreeSpace + "\n" +
                "Total Size: " + di.TotalSize + "\n" +
                "Volume Label: " + di.VolumeLabel.ToString(), di.Name +
                " DRIVE INFO");
        }
    }
}
```

اولین مرحله معرفی فضای نامی System.IO است. در رویداد Form1_Load با استفاده از کلاس DriveInfo یک لیست از تمامی درایوهای موجود روی سیستم تهیه کنید. این عمل با استفاده از متد DriveInfo.GetDrives() انجام می‌شود.

که یک آرایه از اشیاء DriveInfo بر می‌گرداند و با استفاده از حلقه‌ی foreach می‌توانید آنها را یافته و به ListBox اضافه کنید. این کار چیزی شبیه شکل ۲۱-۱۱ تولید می‌کند.

شکل ۲۱-۱۱



این فرم به کاربر اجازه می‌دهد یکی از درایوهای لیست را انتخاب کند. زمانی که انتخاب انجام شد، یک کادر پیام که شامل جزئیات درایو انتخاب شده است نمایش داده می‌شود.

۲۱-۵-امنیت فایل

زمانیکه چارچوب .Net ۱.۱/۱.۰ اولین بار معرفی شد. هیچ روش ساده برای کار با ACLهای فایل‌ها، فهرست‌ها نبود. بنابراین لازم بود با کدنویسی پیچیده‌ی COMها به این عمل دسترسی داشت. در حال حاضر نسخه .Net ۲.۰ کار با ACLها را با یک فضای نامی جدید System.Security.AccessControl ساده‌تر کرده است. با این فضای نامی جدید، دستکاری تنظیمات امنیت فایل‌ها، فهرست‌ها، کلیدهای رجستری، اشتراک شبکه، اشیاء Active Directory و غیره امکان‌پذیر است.

۲۱-۵-۱-خواندن ACLهای یک فایل

با یک مثال ساده این مطلب را ارائه می‌کنیم. برنامه‌ی زیر یک برنامه کنسولی است که اطلاعات ACL یک فایل خاص را نشان می‌دهد.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;
using System.Security.AccessControl;
namespace ConsoleApplication\
{
    class Program
    {
        static string myFilePath;
        static void Main(string[] args)
        {
            Console.WriteLine("Provide full file path: ");
            myFilePath = Console.ReadLine();
            try
            {
                using (FileStream myFile = new FileStream(myFilePath,
                    FileMode.Open, FileAccess.Read))
                {
                    FileSecurity fileSec = myFile.GetAccessControl();
                    foreach (FileSystemAccessRule fileRule in
                        fileSec.GetAccessRules(true, true,
                            typeof(System.Security.Principal.NTAccount)))
                    {

```

```

Console.WriteLine("{0} {1} {2} access for {3}", myFilePath,
fileRule.AccessControlType == AccessControlType.Allow ?
"provides" : "denies",
fileRule.FileSystemRights,
fileRule.IdentityReference.ToString());
}
}
}
catch
{
Console.WriteLine("Incorrect file path given!");
}
Console.ReadLine();
}
}
}
}

```

اولین مرحله ارجاع به فضای نامی System.Security.AccessControl است. این عمل دسترسی به کلاسهای FileSecurity و FileSystemAccessRule را به شما خواهد داد. بعد از اینکه فایل خاص در یک شی FileStream قرار گرفت. ACLهای فایل از طریق متد GetAccessControl() روی شی File بدست می‌آید. این اطلاعات در یک کلاس FileSecurity قرار می‌گیرند. این کلاس قوانین دسترسی را در بر دارد. هر قانون دسترسی با یک شی FileSystemAccessRule نمایش داده می‌شود. این مثال را اجرا کرده و خروجی آن را مشاهده کنید.

خواندن ACLهای یک فهرست شبیه فایل است. فقط به جای شی FileStream از DirectoryInfo استفاده کنید

۲۱-۵-۲- اضافه کردن و حذف ACLهای یک فایل

دستکاری ACLهای یک منبع با استفاده از اشیاء استفاده شده در مثال‌های قبلی امکان‌پذیر است. کد زیر مثال قبلی خواندن ACLهای یک فایل را تغییر می‌دهد. در این مثال ACLهای یک فایل خاص خوانده شده، تغییر یافته و مجدداً خوانده می‌شود.

```

try
{
using (FileStream myFile = new FileStream(myFilePath,
FileMode.Open, FileAccess.ReadWrite))
{
FileSecurity fileSec = myFile.GetAccessControl();
Console.WriteLine("ACL list before modification:");
foreach (FileSystemAccessRule fileRule in
fileSec.GetAccessRules(true, true,
typeof(System.Security.Principal.NTAccount)))
{
Console.WriteLine("{0} {1} {2} access for {3}", myFilePath,
fileRule.AccessControlType == AccessControlType.Allow ?
"provides" : "denies",
fileRule.FileSystemRights,
fileRule.IdentityReference.ToString());
}
Console.WriteLine();
Console.WriteLine("ACL list after modification:");
FileSystemAccessRule newRule = new FileSystemAccessRule(
new System.Security.Principal.NTAccount(@"PUSHKIN\Tuija"),
FileSystemRights.FullControl,
AccessControlType.Allow);
fileSec.AddAccessRule(newRule);
File.SetAccessControl(myFilePath, fileSec);
foreach (FileSystemAccessRule fileRule in
fileSec.GetAccessRules(true, true,
typeof(System.Security.Principal.NTAccount)))
{

```

```
Console.WriteLine("{0} {1} {2} access for {3}", myFilePath,
fileRule.AccessControlType == AccessControlType.Allow ?
"provides" : "denies",
fileRule.FileSystemRights,
fileRule.IdentityReference.ToString());
}
}
}
```

در این مثال یک قانون دسترسی جدید به ACL فایل اضافه می‌شود. این عمل با استفاده از شی `FileSystemAccessRule` انجام می‌شود. در اینجا یک نمونه از شی `NTAccount` جدید ایجاد شده و دسترسی `FullControl` این فایل به آن داده می‌شود. سپس متد `AddAccessRule` کلاس `FileSecurity` برای تخصیص قانون جدید استفاده می‌شود. متد `SetAccessControl` از شی `File` یک نمونه `FileSecurity` را به یک فایل تخصیص می‌دهد.

خروجی برنامه‌ی بالا را بررسی کنید.

برای حذف یک قانون از لیست ACL، کافی است در کد قبلی تغییر زیر را انجام دهید. به جای دستور

```
fileSec.AddAccessRule(newRule);
```

کد زیر را بنویسید:

```
fileSec.RemoveAccessRule(newRule);
```

۲۱-۶- خلاصه

- کلاس‌های مرتبط با عملیات سیستم فایل تقریباً در فضای نامی `System.IO` هستند.
- کلاس‌های `File` و `Directory` فقط متدهای ایستا را در بر دارند و هرگز نمونه‌ای از آن ایجاد نمی‌شود
- کلاس `FileInfo` تعدادی متد با عناوین `Open()`، `OpenRead()`، `OpenText()`، `OpenWrite()`، `Create()` و `CreateText()` را پیاده‌سازی می‌کند که اشیاء `Stream` را بر می‌گردانند.
- کلاس `Path` عملیات روی اسامی مسیرها را ساده‌تر می‌سازد.
- انتقال و حذف فایل‌ها و پوشه‌ها بوسیله متدهای `MoveTo()`، `Delete()` از کلاس‌های `FileInfo` و `DirectoryInfo` انجام می‌شود.
- و متد `File.ReadAll()` محتویات فایل را می‌خواند.
- ابتدا متد `WriteAll` در موقعیت مشخص شده، فایل متنی جدیدی ایجاد می‌کند و محتوای مورد نظر را در آن نوشته و ذخیره می‌کند و سپس آن را می‌بندد.
- شی `Stream` برای انتقال داده استفاده می‌شود. داده می‌تواند در یکی از دو جهت انتقال داده شود.
- خواندن و نوشتن در فایل‌های دودویی می‌تواند از طریق کلاس `FileStream` انجام شود.
- `StreamReader` برای خواندن فایل‌های متنی استفاده می‌شود
- کلاس `DriveInfo` می‌تواند سیستم را جهت بدست آوردن درایوهای آن پویش کند و سپس اطلاعات دقیق در مورد هر درایو را در اختیار شما قرار دهد.

- در حال حاضر نسخه .Net ۲,۰ کار با ACLها را با یک فضای نامی جدید
System.Security.AccessControl ساده‌تر کرده است.

فصل بیست و دوم

استفاده از بانک اطلاعاتی

آنچه که در این فصل یاد خواهید گرفت:

- با مفهوم بانک‌های اطلاعاتی آشنا خواهید شد.
- با دستور SELECT در زبان SQL آشنا خواهید شد و از آن استفاده خواهید کرد.
- کنترل‌های دسترسی به داده‌های درون بانک اطلاعاتی را بررسی خواهید کرد.
- با نحوه‌ی استفاده از داده‌ها در برنامه‌های ویندوز آشنا خواهید شد.
- از ویژگی‌های دسترسی به اطلاعات در ۲۰۰۵VS استفاده خواهید کرد.

۲۲-۱-مقدمه

اغلب برنامه‌های کامپیوتری که امروزه نوشته می‌شوند به نحوی با داده‌ها و اطلاعات مختلف کار می‌کنند. در ۲۰۰۵VS بیشتر این برنامه‌ها، داده‌های مورد نیاز خود را در بانک‌های اطلاعاتی رابطه‌ای نگهداری می‌کنند. بنابراین در هنگام نوشتن این نوع برنامه‌ها نیاز دارید که بتوانید در برنامه‌ی خود با نرم‌افزارهای مربوط به این بانک‌های اطلاعاتی، مانند SQL server، Access، oracle و یا Sybase کار کنید.

در ۲۰۰۵VS ابزارهای و ویژگی‌های زیادی برای متصل شدن به انواع بانک‌های اطلاعاتی وجود دارد. به وسیله‌ی این ابزارها می‌توانید اطلاعات خود را در درون این بانک‌های اطلاعاتی قرار دهید و یا آن را از بانک‌های اطلاعاتی دریافت کرده و تغییرات مورد نظر خود را روی آنها انجام دهید. در طی این فصل سعی کنید که با این ابزارها و نحوه‌ی کارکرد آنها در برنامه بیشتر آشنا شوید.

در فصل بعدی تمرکز خود را روی استفاده از بانک‌های اطلاعاتی از طریق کدنویسی قرار خواهیم داد و مشاهده خواهیم کرد که چگونه می‌توان از طریق برنامه‌نویسی به صورت مستقیم به این بانک‌های اطلاعاتی دسترسی پیدا کرد. بعد از اینکه مقداری در کدنویسی بانک‌های اطلاعاتی تمرین کردید، خواهید دید که استفاده از کد نسبت به استفاده از ویژگی‌ها و ابزارها زمان بسیار کمتری را اشغال می‌کند.

نکته: برای انجام تمرینات و مثال‌های این فصل لازم است که نسخه‌ی ۲۰۰۰ (و یا بالاتر) برنامه Microsoft Access که جزئی از برنامه‌ی Microsoft office به شمار می‌رود را نصب کنید.



بانک اطلاعاتی چیست؟

اصولاً "هر بانک اطلاعاتی شامل یک و یا چند فایل بزرگ و پیچیده است که داده‌ها در آن در یک قالب و فرمت ساخت‌یافته ذخیره می‌شوند. موتور بانک اطلاعاتی معمولاً به برنامه‌ای اطلاق می‌شود که این فایل و یا فایل‌هایی و نیز داده‌های درون آنها را مدیریت می‌کند. در طی این فصل از برنامه‌ی Microsoft Access به عنوان موتور بانک اطلاعاتی استفاده خواهیم کرد.

اشیای موجود در Access

یک فایل بانک اطلاعاتی مربوط به برنامه‌ی Access (که پسوند آن نیز .mdb است) معمولاً از قسمت‌های مختلفی مانند جدول‌ها، پرس‌وجوها، فرم‌ها، گزارشات، ماکروها و ماژول‌ها تشکیل شده است. به این قسمت‌های تشکیل دهنده‌ی یک بانک اطلاعاتی اشیای بانک اطلاعاتی گفته می‌شود. در یک فایل مربوط به بانک اطلاعاتی عموماً "داده‌های زیادی وجود دارند و به همین دلیل، موتورهای بانک اطلاعاتی مانند Access سعی می‌کنند با ارائه دادن امکانات اضافی، به کاربران اجازه دهند با این اطلاعات کار کنند، در بین اشیایی که در یک بانک اطلاعاتی Access وجود دارند، جدول‌ها و پرس‌وجوها برای نگهداری داده‌ها و یا دسترسی به آنها به کار می‌روند. دیگر اشیای یک بانک اطلاعاتی مانند فرم‌ها و یا گزارشات برای این است که کاربران بتوانند به سادگی با داده‌های موجود در جدول کار کنند.

اما به هر حال به علت پیچیده بودن ساختار موتورهای بانک اطلاعاتی، کاربران معمولی حتی با استفاده از این قسمت‌ها نیز نمی‌توانند به درستی از اطلاعات درون بانک اطلاعاتی استفاده کنند. هدف ما از نوشتن یک برنامه‌ی بانک اطلاعاتی با استفاده از VS ۲۰۰۵ و یا هر زبان برنامه‌نویسی دیگر این است که به کاربر اجازه دهیم به سادگی از اطلاعات درون بانک‌ها استفاده کند، پس در این برنامه‌ها فقط به اطلاعات درون یک بانک اطلاعاتی نیاز خواهیم داشت نه به قسمتهایی مانند فرم‌ها و یا گزارشات. بنابراین در طی این فصل بیشتر تمرکز خود را روی دو قسمت اصلی بانک‌های اطلاعاتی یعنی جدول‌ها و پرس‌وجوها قرار می‌دهیم.

جدول‌ها

یک جدول شامل یک مجموعه از اطلاعات است که معمولاً "حاوی یک و یا چند ستون و نیز یک و یا چند ردیف از داده‌ها است. در Access (و نیز بیشتر بانک‌ها یا اطلاعاتی) به هر یک از این ستون‌ها یک فیلد گفته می‌شود. همچنین هر ردیف از این اطلاعات نیز یک رکورد نامیده می‌شوند. هر فیلد در یک جدول از بانک اطلاعاتی، یکی از مشخصه‌های داده‌ای که در آن جدول ذخیره شده است را نگهداری می‌کند. برای مثال، فیلدی به نام firstName در یک جدول، مشخص کننده‌ی نام مشترک و یا کارمندی است که اطلاعات او در آن جدول ذخیره شده است. بنابراین این فیلد یکی از مشخصه‌های آن کارمند و یا مشترک را نمایش می‌دهد. در هر جدول یک رکورد شامل یک مجموعه از فیلدها است که اطلاعات و مشخصه‌های مربوط به یک نمونه از داده‌هایی که در آن جدول ذخیره شده است را نشان می‌دهد. برای مثال، جدولی را در نظر بگیرید که دارای دو فیلد (دو ستون اطلاعات) به نامهای firstName و LastName است و برای نگهداری اسامی کارمندان استفاده می‌شود. به مجموعه‌ی نام و نام خانوادگی هر کارمندی که اطلاعات او در این جدول وجود داشته باشد یک رکورد گفته می‌شود. برای مثال، در شکل ۱-۲۲، EmployeeID، firstName و فیلدهای این جدول و هر ردیف از اطلاعات نیز رکوردهای آن را مشخص می‌کند.

شکل ۱-۲۲

Employees : Table				
	Employee ID	Last Name	First Name	Title
▶ +	1	Davolio	Nancy	Sales Representative
▶ +	2	Fuller	Andrew	Vice President, Sales
▶ +	3	Leverling	Janet	Sales Representative
▶ +	4	Peacock	Margaret	Sales Representative
▶ +	5	Buchanan	Steven	Sales Manager
▶ +	6	Suyama	Michael	Sales Representative
▶ +	7	King	Robert	Sales Representative
▶ +	8	Callahan	Laura	Inside Sales Coordinator
▶ +	9	Dodsworth	Anne	Sales Representative
*	(AutoNumber)			
Record: 1 of 9				

پرس وجوها

در هر بانک اطلاعاتی عموماً^۱ به یک سری از دستورات که زبان SQL نوشته شده است و برای دریافت اطلاعات از بانک اطلاعاتی و یا ایجاد تغییراتی در اطلاعات موجود در بانک به کار می‌رود، یک پرس‌وجو گفته می‌شود. با استفاده از پرس‌وجوها می‌توانیم داده‌هایی را در جدول‌های بانک اطلاعاتی وارد کنیم، آنها را از یک و یا چند جدول بدست آورده و یا تغییراتی را در آنها ایجاد کنیم.

در یک موتور بانک اطلاعاتی به دو روش می‌توانیم از پرس‌وجوها استفاده کنیم. اول این است که یک دستور SQL را بنویسیم و سپس آن را اجرا کرده و نتیجه‌ی آن را مشاهده کنیم. روش دوم این است که همانند دیگر زبان‌های برنامه‌نویسی یک زیربرنامه با استفاده از دستورات SQL ایجاد کنیم و سپس با فراخوانی آن زیربرنامه اطلاعاتی بدست آوریم. در برنامه‌هایی که به زبان #VC می‌نویسیم نیز می‌توانیم از یک زیربرنامه برای دسترسی به اطلاعات مورد نیاز استفاده کنیم و هم می‌توانیم دستور SQL مورد نیاز را با استفاده از برنامه به موتور بانک اطلاعاتی بفرستیم و نتایج حاصل را دریافت کرده و نمایش دهیم.

البته استفاده از زیربرنامه‌های موجود در یک موتور بانک اطلاعاتی نسبت به دستورات معمولی از سرعت بیشتر برخوردار است. زیرا استفاده از زیربرنامه‌های موجود در یک موتور بانک اطلاعاتی نسبت به دستورات معمولی از سرعت بیشتری برخوردار است. زیرا موتور بانک اطلاعاتی می‌تواند دستورات درون آن زیربرنامه را تحلیل کرده و یک روش کلی برای سریع‌تر اجرا کردن آن ایجاد کند (به عبارت دیگر می‌تواند آنها را کامپایل کند). اما دستوراتی که به صورت عادی به موتور بانک اطلاعاتی می‌دهیم تا آنها را اجرا کند و داده‌های مربوط را برگرداند، هر مرتبه لازم است که تفسیر شده و سپس اجرا شوند و این مورد باعث می‌شود که سرعت اجرای کمتری نسبت به زیربرنامه‌ها داشته باشد.

برای درک بهتر مفهوم پرس‌وجوها، بهتر است ابتدا مقداری با زبان SQL و دستورات آن آشنا شویم. خوشبختانه، زبان SQL نسبت به زبانهای برنامه‌نویسی دیگر بسیار ساده‌تر است و به سرعت می‌توان نحوه‌ی استفاده از آن را یاد گرفت.

به زیربرنامه‌هایی که برای پرس‌وجو از یک بانک اطلاعاتی نوشته می‌شود. زیربرنامه‌های ذخیره شده^۱ گفته می‌شود.

دستور SELECT در زبان SQL

زبان SQL برخلاف چیزی که ممکن است تصور کنید، زیاد مشابه زبانهای برنامه‌نویسی که تاکنون دیده‌اید نیست. دستورات این زبان به وسیله موسسه استاندارد ملی آمریکا (ANSI) به صورت استاندارد درآمده است. نسخه‌ی استاندارد این زبان که ANSI SQL نیز نامیده می‌شود، به وسیله‌ی تمام موتورهای بانک اطلاعاتی پشتیبانی می‌شود. اما هر یک از این موتورهای بانک اطلاعاتی امکانات مخصوص بیشتری را نیز به این زبان اضافه کرده‌اند که معمولاً فقط در همان موتور بانک اطلاعاتی قابل استفاده است.

^۱ Stored procedure

مزایای یادگیری ANSI SQL در این است که به این وسیله، هنگامی که اصول دستورات زبان SQL را آموختید می‌توانید از آنها برای برنامه‌نویسی SQL در تمام موتورهای بانک اطلاعاتی استفاده کنید. به این ترتیب برای این که بتوانید موتور بانک اطلاعاتی خود را تغییر دهید، فقط کافی است نحوه‌ی کارکرد با رابط گرافیکی آن را یاد بگیرید و سپس می‌توانید از دستورات SQL استاندارد در آن محیط نیز استفاده کنید. البته همانطور که گفتیم هر موتور بانک اطلاعاتی دارای دستورات خاص خود است که باعث افزایش کارایی و بهینه ساختن اجرای دستورات می‌شود. اما تا حد ممکن بهتر است از این دستورات در برنامه‌ی خود استفاده نکنید و دستورات استاندارد SQL را به کار ببرید. به این ترتیب، می‌توانید هر زمان که لازم باشد به سادگی موتور بانک اطلاعاتی خود را تغییر دهید.

زبان SQL از تعداد کمی دستور تشکیل شده است که هر یک کار خاصی را انجام می‌دهند. یکی از پرکاربردترین و مهمترین این دستورات SELECT است که به وسیله‌ی آن می‌توانید یک یا چند فیلد اطلاعات مربوط به یک یا چند رکورد در جدول بانک اطلاعاتی خود را بدست آورید. البته دقت داشته باشید که به وسیله‌ی دستور SELECT فقط می‌توانید داده‌ها را از جداول بدست آورید، اما نمی‌توانید هیچ تغییری در آنها ایجاد کنید.

ساده‌ترین دستور SELECT در زبان SQL مشابه دستور زیر است:

```
SELECT * FROM Employees;
```

این دستور همانطور که مفهوم کلمات آن نیز مشخص می‌کنند، به این معنی است که اطلاعات موجود در تمام فیلدهای مربوط به همه‌ی رکوردهای جدول Employees را انتخاب کن. علامت * در دستور SELECT به معنی تمام فیلدها است. کلمه‌ی Employees نیز نام جدولی در بانک اطلاعاتی است که این دستور باید بر روی آن اجرا شود.

اگر بخواهید فقط فیلدهای مربوط به نام و نام خانوادگی افرادی که اطلاعات آنها در جدول Employees وارد شده است را بدست آورید، کافی است که علامت * را با نام فیلدهای مورد نظر خود به صورت زیر عوض کنید:

```
SELECT [First Name], [Last Name] FROM Employees;
```

دقت کنید که هنگام وارد کردن این دستور حتماً باید از علامت [] در ابتدای نام فیلدها استفاده کنید. زیرا نام این فیلدها حاوی فضای خالی (space) است و باعث می‌شود که برنامه در تفسیر نام First Name با مشکل مواجه شود. استفاده از [] به موتور بانک اطلاعاتی می‌گوید که کلمات داخل [] را به عنوان یک نام در نظر بگیرد. البته اگر نام این فیلد حاوی کاراکتر فضای خالی نبود، می‌توانستید از [] استفاده نکنید.

همانطور که مشاهده می‌کنید دستورات SQL همانند زبان انگلیسی عادی و روزمره هستند و حتی فردی که برنامه‌نویس نیست نیز می‌تواند آن را خوانده و مفهوم آن را درک کند. برای مثال، اگر بخواهیم فقط داده‌هایی که دارای شرط خاصی هستند از جدول انتخاب شده و نمایش داده شوند، کافی است از عبارت WHERE در پایان دستور SELECT استفاده کنیم. مثلاً "اگر بخواهیم در دستور قبل فقط افرادی که نام خانوادگی آنها با حروف D شروع می‌شوند انتخاب شوند، باید از دستور زیر استفاده کنیم:

```
SELECT [First Name], [Last Name] FROM Employees
WHERE [Last Name] LIKE 'D*';
```

عبارت WHERE باعث می‌شود فقط داده‌هایی از جدول انتخاب شوند که در شرط مقابل عبارت WHERE صدق می‌کنند. بنابراین دستور SELECT قبلی باعث می‌شود که موتور بانک اطلاعاتی به داخل جدول Employees برود و فیلد Last Name و First Name تمام رکوردهایی که Last Name آنها با حرف D شروع می‌شود را انتخاب کند. عبارت 'D*' نیز به این معنی است که هر عبارتی که با حروف D شروع شده است. برای مثال، عبارت 'D*' به این معنی است که "هر عبارتی که در آن حرف D وجود داشته باشد."

در آخر نیز بعد از اینکه داده‌های مورد نظر خود را انتخاب کردید، می‌توانید آنها را به نحوی که تمایل دارید به صورت صعودی و یا نزولی مرتب کنید، برای مثال، براساس فیلد First Name برای کار باید در انتهای دستور SELECT از عبارت ORDER BY استفاده کنید.


```
SELECT [First Name], [Last Name] FROM Employees  
WHERE [Last Name] LIKE 'D*' ORDER BY [First Name];
```

اجرای این دستور باعث می شود اطلاعاتی از جدول انتخاب شوند. برای مثال، خروجی این دستور می تواند مانند زیر باشد:

```
Angela Dunn  
David Dunstan  
Zebedee Dean
```

همانطور که مشاهده می کنید در این قسمت از یک دستور تقریباً "کامل استفاده کردیم، اما درک آن نیز بسیار ساده بود و تقریباً" بسیار مشابه چیزی بود که در زبان انگلیسی برای منظور خود باید عنوان کنید. معمولاً هنگامی که اطلاعات را براساس فیلدهای رشته ای مرتب می کنید، داده ها به صورت صعودی مرتب می شوند. به این صورت که اطلاعات با حرف A ابتدا و اطلاعات با حرف Z در انتها نمایش داده می شوند. اما هنگامی که بخواهید اطلاعات را براساس یک فیلد عددی مرتب کنید، ممکن است تمایل داشته باشید که داده های بزرگتر ابتدا نمایش داده شوند. برای مثال ممکن است بخواهید اطلاعاتی که انتخاب می شوند، بر اساس قیمت کالا مرتب شده و کالای های گرانتر نیز در بالای جدول قرار بگیرند. بنابراین لازم است که اطلاعات را به صورت نزولی مرتب کنید. برای این کار کافی است در پایان دستور ORDER BY عبارت DESC استفاده کنید. به این ترتیب داده ها به صورت نزولی مرتب خواهند شد.

```
SELECT [First Name], [Last Name] FROM Employees  
WHERE [Last Name] LIKE 'D*' ORDER BY [First Name] DESC;
```

اجرای دستور بالا نتایجی را مشابه زیر برمی گرداند:

```
Zebedee Dean  
David Dunstan  
Angela Dunn
```

نکته: اگر می خواهید در دستور خود مشخصاً "قید کنید که اطلاعات باید براساس صعودی مرتب شوند، می توانید در انتهای دستور ORDER BY عبارت ASC استفاده کنید. البته استفاده از این عبارت الزامی نیست، زیرا به صورت پیش فرض اطلاعات به صورت صعودی مرتب می شوند.

بطور خلاصه، می توان گفت که دستور SELECT می تواند با ساختاری مشابه زیر مورد استفاده قرار بگیرد:

```
SELECT select-list  
FROM table-name  
[WHERE search-condition]  
[ORDER BY order-by-expression [ASC | DESC]]
```

این عبارت به این معنی است که در قسمت select-list حتماً باید لیستی از نام فیلدهای مورد نظر و یا علامت * برای انتخاب تمام فیلدها را ذکر کنید. همچنین در قسمت table-list نیز باید نام جدول مورد نظر را بیاورد. می توانید از عبارت WHERE در دستور SELECT خود استفاده کنید. به این ترتیب فقط داده هایی که در شرط search-condition صدق می کنند، انتخاب خواهند شد. با استفاده از قسمت ORDER BY نیز می توانید داده ها را مرتب کنید. برای این کار باید در قسمت order-by-expression فیلدی که می خواهید داده ها براساس آن مرتب شوند را ذکر کنید. برای صعودی و یا نزولی بدن مرتب سازی نیز می توانید از عبارت ASC و یا DESC در انتهای دستور استفاده کنید.

البته اگر بخواهید داده ها را از چندین جدول یک بانک اطلاعاتی استخراج کنید و یا براساس رابطه ی خاصی به داده ها دسترسی پیدا کنید، دستورات SQL به مقدار قابل ملاحظه ای پیچیده خواهند شد که توضیح این گونه دستورات از اهداف این کتاب خارج است و در برنامه های این فصل و فصل بعد نیز به آنها نیازی نخواهیم داشت.

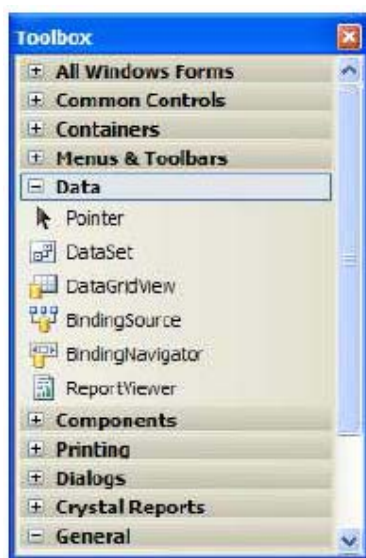
در هر حال بهترین روش برای یادگیری نحوه ی استفاده از دستورات SQL، تمرین و کار کردن با این دستورات است. قبل از این که به ادامه ی فصل پردازیم بهتر است به سوالات زیر به صورت ذهنی پاسخ دهید.

- چگونه می‌توانید یک دستور SELECT بنویسیم که داده‌های موجود در فیلدهای Name، Description و Price را از یک جدول به نام Products استخراج کند؟
- چگونه می‌توان دستور بالا را به گونه‌ای تغییر داد تا فقط داده‌هایی را برگرداند که در فیلد Description آنها عبارت DVD وجود داشته باشد؟
- چگونه می‌توان اطلاعات بالا را براساس قیمت به گونه‌ای مرتب کرد که اجناس گرانتر در ابتدای جدول قرار بگیرد؟

۲-۲۲- کنترل‌های دسترسی اطلاعات

در ویژوال #C ۲۰۰۵ برای دسترسی به اطلاعات و نمایش آنها سه کنترل مهم و اصلی وجود دارند که عبارتند از: BindingSource، TableAdapter و DataSet. دو کنترل BindingSource و DataSet همانطور که در شکل ۲-۲۲ مشاهده می‌کنید در قسمت Data در جعبه ابزار وجود دارند. کنترل TableAdapter نیز براساس مسیری که برای دسترسی به اطلاعات درون بانک اطلاعاتی و نمایش آنها طی می‌کنید به صورت اتوماتیک ایجاد خواهد شد.

شکل ۲-۲۲



نکته: این کنترل‌ها که عموماً به عنوان کنترل‌های داده‌ای شناخته می‌شوند، خود فقط چندین کلاس هستند، مانند تمام کلاس‌های دیگر NET که در قسمتهای قبلی از آنها استفاده کردیم. در این فصل فقط با نحوه‌ی استفاده از این کلاس‌ها در برنامه‌های ویندوزی آشنا می‌شویم. در فصل بعد، سعی می‌کنیم که کلاس‌های مربوط به این کنترل‌ها را با جزئیات بیشتری بررسی کنیم.

DataSet

کنترل DataSet در حقیقت همانند یک مخزن است که داده‌های مورد نیاز را در حافظه‌ی کامپیوتر نگهداری می‌کند. این کنترل همانند یک موتور بانک اطلاعاتی کوچک عمل می‌کند که داده‌های مورد نیاز خود را در حافظه نگهداری می‌کنند. با استفاده از این کنترل می‌توانید داده‌ها را درون جدول‌هایی نگهداری کرده و سپس از کنترل DataView که در فصل بعدی توضیح داده خواهد شد. به چندین روش پرس‌وجوهای را روی این داده‌ها اجرا کنید.

کنترل DataSet از قدرت و امکانات زیادی برخوردار است. این کنترل علاوه‌براین، توانایی ذخیره‌ی داده‌ها در جدول، حجم زیادی از فراداده (اطلاعاتی درباره‌ی داده‌ها) را نیز نگهداری می‌کند. این اطلاعات شامل مواردی مانند نام جدول‌ها و یا فیلدها، نوع داده‌های موجود، اطلاعات مورد نیاز برای مدیریت داده‌ها و یا اطلاعاتی در رابطه با لغو کردن تغییرات اعمال شده در داده‌ها می‌باشد.

تمام این اطلاعات در قالب XML در حافظه ذخیره می‌شوند. به علاوه، یک کنترل DataSet می‌تواند به سادگی در قالب XML در دیسک ذخیره شده و یا از قالب XML از دیسک در حافظه قرار داده شود. همچنین این کنترل می‌تواند به صورت XML از طریق شبکه‌های مختلف مانند اینترنت به برنامه‌های دیگر فرستاده شود و مورد استفاده قرار گیرد.

به علت اینکه داده‌های یک کنترل DataSet در حافظه قرار دارند، بنابراین می‌توانید به سادگی در بین آنها به جلو و یا عقب حرکت کنید و یا در آنها تغییری را ایجاد کنید. البته این تغییرات در داده‌های موجود در حافظه اعمال می‌شوند و تا زمانی که مشخص نکنید به داده‌ای موجود در بانک اطلاعاتی منعکس نخواهند شد. در مورد این کنترل در فصل بعد بیشتر صحبت خواهیم کرد. اما در این فصل فقط داده‌هایی را در آن قرار داده و سپس به وسیله‌ی کنترل‌های دیگری آن داده‌ها را در برنامه نمایش خواهیم داد.

DataGridView

این کنترل برای نمایش داده‌های موجود در یک بانک اطلاعاتی در فرم برنامه به کار می‌رود. برای کار با آن کافی است آن را به منبع داده‌های خود، برای مثال یکی از جدول‌های موجود در بانک اطلاعاتی، متصل کرده و سپس این کنترل را تنظیم کنید تا آن داده‌ها را همانند یک جدول، یعنی ستون‌ها را به صورت عمودی و ردیف‌ها را به صورت افقی نمایش دهد.

همچنین این کنترل دارای خاصیت‌های زیادی است که به وسیله‌ی آنها می‌توانید ظاهر آن را تنظیم کنید تا به شکلی که مد نظر شماست تبدیل شود. علاوه‌براین، به وسیله‌ی این کنترل می‌توانید عنوان ستون‌های داده‌ها و یا روش نمایش آنها نیز تعیین کنید.

BindingSource

این کنترل همانند پلی برای ایجاد ارتباط بین داده‌های موجود در منبع داده‌ای شما (DataSet) و نیز کنترل‌هایی که برای نمایش داده‌ها مورد استفاده قرار می‌گیرند به کار می‌رود. بنابراین، هنگامی که بخواهید به وسیله‌ی کنترل‌هایی داده‌های موجود در برنامه‌ی خود را نمایش دهید، و یا به هر دلیل دیگری بخواهید به آنها در منبع اطلاعاتی دسترسی داشته باشید، این ارتباط باید از طریق این کنترل صورت بگیرد.

برای مثال تصور کنید که داده‌های موجود در یک DataSet را به وسیله‌ی یک کنترل DataGridView در فرم برنامه نمایش داده‌اید و حال می‌خواهید این داده‌ها براساس یکی از ستونها مرتب شده و سپس نمایش داده شوند. برای این کار کنترل DataGridView این تقاضا را به کنترل BindingSource می‌فرستد و سپس این کنترل آن را به کنترل DataSet اعلام می‌کند. در ادامه‌ی این فصل با نحوه‌ی استفاده از این کنترل بیشتر آشنا خواهیم شد.

BindingNavigator

کنترل BindingNavigator یک رابط گرافیکی استاندارد را برای حرکت بین رکوردهای موجود در یک بانک اطلاعاتی ایجاد می‌کند. این کنترل بسیار مشابه کنترلی است که در پایین جدول در برنامه Access نمایش داده می‌شود. این کنترل نیز همانند کنترل DataGridView می‌تواند به کنترل BindingSource متصل شده و از طریق آن به داده‌های موجود در برنامه دسترسی داشته باشد. به این ترتیب، برای مثال هنگامی که روی کلید Next در این کنترل کلیک کردید تا به رکورد بعدی اطلاعات بروید، درخواست شما به وسیله‌ی BindingNavigator به کنترل BindingSource فرستاده شده و سپس از کنترل BindingSource به کنترل DataSet (و یا منبع اطلاعاتی دیگر که در برنامه از آن استفاده می‌کنید) اعلام می‌شود.

TableAdapter

تنها یک کنترل داده‌ای دیگر مانده است که باید در مورد آن صحبت کنیم: کنترل DataAdapter. این کنترل در جعبه ابزار وجود ندارد که بتوانید آن را همانند کنترل‌های قبلی بر روی فرم قرار دهید. بلکه بسته به روشی که کنترل‌های داده‌ای دیگر را در برنامه قرار داده و آنها را تنظیم می‌کنید، این کنترل به صورت اتوماتیک ایجاد می‌شود.

این کنترل حاوی پرس‌وجوهایی برای انتخاب داده‌های موجود در بانک اطلاعاتی و نیز اطلاعاتی در مورد نحوه‌ی اتصال برنامه به بانک است. همچنین این کنترل حاوی متدهایی است که به وسیله‌ی آنها می‌توان داده‌ها را از جداول بانک اطلاعاتی بدست آورد و در کنترل‌هایی مانند DataSet قرار داد و سپس در برنامه از آن داده‌ها استفاده کرد. این کنترل این قابلیت را دارد که براساس دستور SELECT ای که برای انتخاب داده‌ها از بانک اطلاعاتی برای آن وارد می‌کنید دستورات UPDATE، INSERT و نیز DELETE مناسب برای داده‌های انتخاب شده در بانک اطلاعاتی ایجاد کند.

در فصل بعد بیشتر با این کنترل آشنا خواهیم شد.

۲۲-۳- مقید کردن داده‌ها

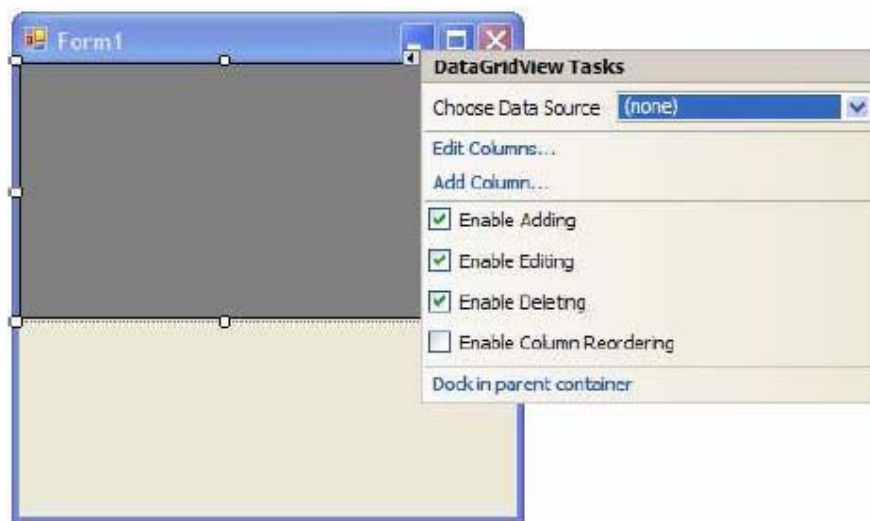
مقید کردن داده‌ها به این معنی است که داده‌هایی که به وسیله‌ی کنترل DataSource به آنها دسترسی دارید را به یک کنترل خاص نسبت دهید. به عبارت دیگر کنترل را بتوانید به نحوی تنظیم کنید که داده‌های مورد نیاز خود را به وسیله‌ی کنترل‌های دسترسی داده‌ها در برنامه دریافت کند و سپس آنها را به صورت اتوماتیک به کاربر نمایش دهد. به این ترتیب کاربر می‌تواند آنها را مشاهده کرده و یا تغییرات موردنظر خود را در آنها اعمال کند، در VC# تقریباً تمام کنترل‌ها تا حدی اتصال به داده‌ها را پشتیبانی می‌کنند، اما بعضی از کنترل‌ها نیز وجود دارند که مخصوص این کار طراحی شده‌اند، مانند کنترل DataGridView و یا TextBox. در مثال بعدی اطلاعاتی که به وسیله‌ی کنترل DataSource به آنها دسترسی داریم را به کنترل DataGridView متصل کرده و به وسیله‌ی این کنترل نمایش می‌دهیم. در بخش بعد نیز این اطلاعات را به وسیله‌ی کنترل TextBox در برنامه نمایش خواهیم داد.

مثال ۲۲-۱- مقید کردن داده‌ها به کنترل DataGridView

(۱) با استفاده از VS۲۰۰۵ یک برنامه‌ی ویندوزی جدید به نام Northwind Customers DataGridView ایجاد کنید.

(۲) با استفاده از جعبه ابزار به قسمت Data بروید و سپس روی کنترل DataGridView دو بار کلیک کرده تا یک نمونه از این کنترل روی فرم برنامه قرار بگیرد. به این ترتیب کادر DataGridView Tasks به صورت اتوماتیک همانند شکل ۲۲-۳ نمایش داده خواهند شد.

شکل ۲۲-۳

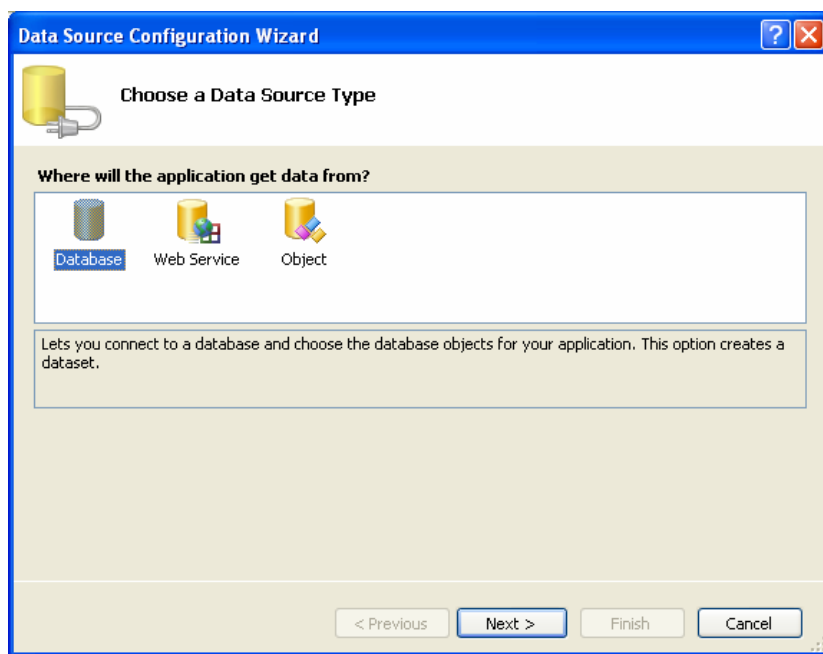


۳) در این کادر، در لیست روبروی عبارت Choose DataSource کلیک کرده و سپس در این لیست روی لینک DataSource Configuration Wizard کلیک کنید. به این ترتیب ویزارد DataSource Configuration Wizard نمایش داده خواهد شد.

۴) در صفحه‌ی اول این ویزارد، یعنی پنجره‌ی Choose a DataSource Type می‌توانید منبع داده‌ای مورد نظر خودتان را انتخاب کنید. همانطور که در شکل ۲۲-۴ نیز مشاهده می‌کنید، در این قسمت می‌توانید انواع مختلفی از منبع‌های داده‌ای را مشخص کرده و به آنها متصل شوید. برای مثال، اگر می‌خواهید به یک بانک اطلاعاتی که توسط نرم افزارهای مختلفی مانند Access، Oracle، SQL Server و یا ایجاد می‌شود دسترسی داشته باشید، روی آیکن Database کلیک کنید. اگر می‌خواهید از طریق یک وب سرویس به بانک اطلاعاتی خود متصل شوید، روی آیکن Web Service کلیک کنید. آیکن objects نیز برای دسترسی به کنترل‌های داده‌ای در لایه‌ی منطق تجاری به کار می‌رود.

در این قسمت آیکن Database را انتخاب کرده و سپس روی دکمه‌ی Next کلیک کنید.

شکل ۲۲-۴



۵) در پنجره‌ی Choose Your Data Connection روی دکمه‌ی New Connection کلیک کنید.

۶) به این ترتیب پنجره‌ی Choose DataSource نمایش داده خواهد شد. در این پنجره گزینه‌ی Microsoft Access Database file را از لیست DataSource انتخاب کرده و روی دکمه‌ی Continue کلیک کنید.

۷) در کادر Add Connection روی دکمه‌ی Browse کلیک کرده و سپس به پوشه‌ی samples در مکان نصب برنامه‌ی office بروید. این پوشه به صورت پیش‌فرض برای office ۲۰۰۳ در آدرس C:\Program Files\Microsoft Office\Office11\Samples قرار دارد. در این آدرس فایل Northwind.mdb را انتخاب کرده و سپس روی دکمه‌ی OK کلیک کنید تا نام و مسیر فایل انتخابی به کادر متنی موجود در پنجره‌ی Add Connection اضافه شوند. سپس در این پنجره نیز روی دکمه‌ی OK کلیک کنید تا کادر Add Connection بسته شود و به پنجره‌ی Choose Your Data Connection برگردید. در این پنجره نیز روی دکمه‌ی Next کلیک کنید.

به این ترتیب کارد پیغام نمایش داده خواهد شد و از شما می‌پرسد که فایل بانک اطلاعاتی که انتخاب کرده‌اید جزئی از پروژه نیست. آیا می‌خواهید این فایل به پوشه‌ی پروژه کپی شده و از نسخه‌ی کپی آن استفاده شود؟ در این کادر روی دکمه‌ی YES کلیک کنید.

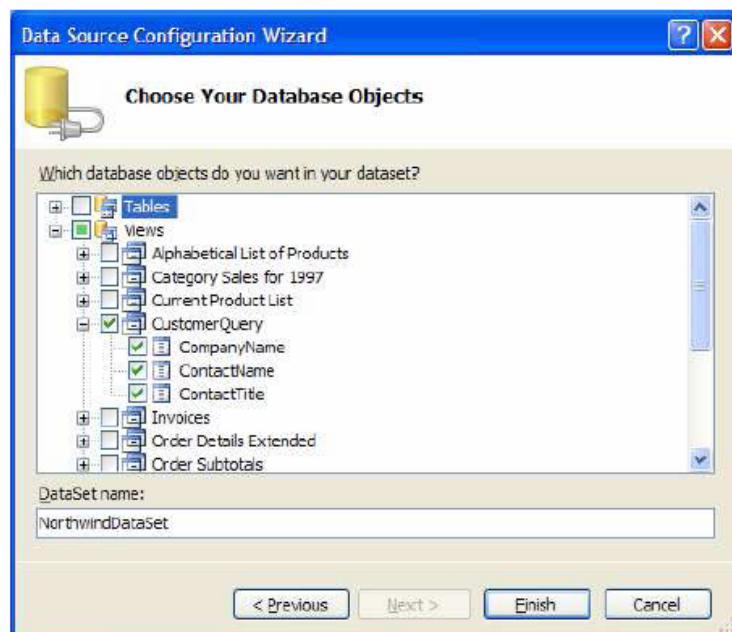
۸) به این ترتیب پنجره‌ی Save the ConnectionString on the Application Configuration File نمایش داده می‌شود. در این پنجره نیز روی دکمه Next کلیک کنید.

۹) بعد از طی این مراحل پنجره‌ی Choose Your Data Objects نمایش داده می‌شود و به شما اجازه می‌دهد تا داده‌های مورد نیاز در برنامه را انتخاب کنید. در این قسمت می‌توانید انتخاب کنید که داده‌های مورد نیاز شما از یک جدول درون بانک اطلاعاتی وارد برنامه شوند، با اجرای پروسیجرهای ذخیره شده در بانک اطلاعاتی ایجاد شده و در اختیار برنامه قرار بگیرند و یا از روشهای دیگر موجود برای گردآوری داده‌های مورد نیاز استفاده شود.

بنابراین در لیست نمایش دهنده‌ی اشیای موجود در بانک اطلاعاتی روی علامت مثبت کنار Views کلیک کرده و سپس از لیست باز شده همانند شکل ۲۲-۴ گزینه‌ی مورد نظر را انتخاب کنید. اگر روی علامت مثبت کنار آن کلیک کنید، لیست تمام فیلدهایی که به وسیله‌ی این پرس‌وجو برگشته می‌شود نمایش داده خواهند شد. بعد از مشاهده‌ی این صفحه روی دکمه‌ی finish کلیک کنید تا کار در این قسمت به اتمام برسد. (فرض کنید نام گزینه‌ی انتخاب شده CustomerQuery است).

در این لحظه، ویزارد یک شی از نوع DataSet به نام NorthWindDataSet، یک شی از نوع BindingSource به نام CustomerQueryBindingSource و نیز یک شی از نوع TableAdapter به نام CustomerQueryTableAdapter ایجاد می‌کند.

شکل ۲۲-۵



۱۰- در فرم اصلی برنامه روی مثلث کوچک کنار DataGridView کلیک کرده تا کادر DataGridView Tasks نمایش داده شود. به این علت که در این قسمت نمی‌خواهیم داده‌های موجود را حذف کرده، اضافه کنیم و یا تغییر دهیم، با کلیک روی گزینه‌های Enable Adding، Enable Editing و Enable Deleting علامت تیک کنار آنها را حذف کنید. اما می‌خواهیم که بتوانیم داده‌ها را براساس ستون‌های مورد نظر مرتب کنیم. پس روی گزینه‌ی Enable Column Reordering کلیک کنید تا انتخاب شود. سپس در قسمتی از نوار عنوان فرم برنامه کلیک کنید تا این پنجره پنهان شود.

۱۱- روی کنترل DataGridView کلیک کرده و سپس با استفاده از پنجره‌ی Properties خاصیت Dock آن را به Fill تغییر دهید.

۱۲- حال برنامه را اجرا کنید. مشاهده خواهید کرد که کنترل DataGridView به وسیله‌ی اطلاعات موجود در بانک اطلاعاتی پر خواهد شد.

با کلیک روی نام هر یک از ستون‌های موجود در جدول، می‌توانید اطلاعات را بر اساس آن ستون به صورت صعودی مرتب کنید. کلیک مجدد روی هر ستون باعث می‌شود که اطلاعات بر اساس آن ستون به صورت نزولی مرتب شوند. برای تشخیص نحوه‌ی مرتب شدن اطلاعات نیز می‌توانید از جهت مثلث کوچکی که در کنار نام ستون نمایش داده می‌شود استفاده کنید.

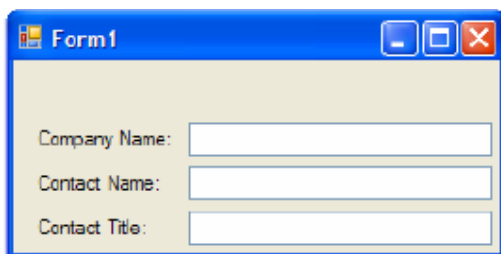
همانطور که مشاهده می‌کنید در این قسمت توانستید بدون اینکه حتی یک خط کد در برنامه وارد کنید داده‌هایی را از یک بانک اطلاعاتی بدست آورده و آنها را نمایش دهید. تمام کدهای مورد نیاز برای این موارد به صورت اتوماتیک توسط این ویزارد نوشته شده‌اند.

مثال ۲۲-۲- مقید کردن داده‌ها به کنترل TextBox

۱- با استفاده از VS یک برنامه‌ی ویندوزی جدید به نام Northwind Customers BindingNavigator ایجاد کنید.

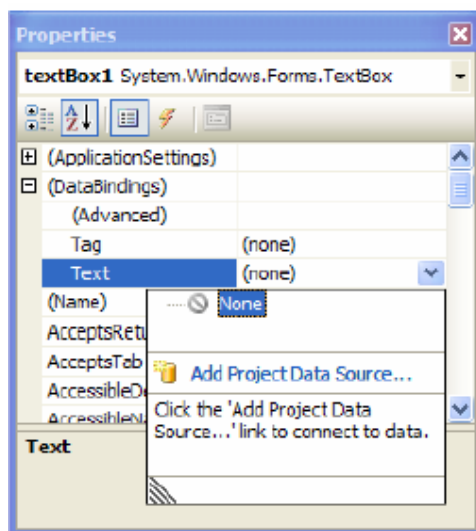
۲- با استفاده از کادر ابزار سه کنترل Label و سه کنترل TextBox به برنامه‌ی خود اضافه کنید. سپس خاصیت‌ها و مکان این کنترل‌ها را به گونه‌ای تغییر داده و تنظیم کنید که فرم برنامه مشابه شکل ۲۲-۵ شود.

شکل ۲۲-۱۲



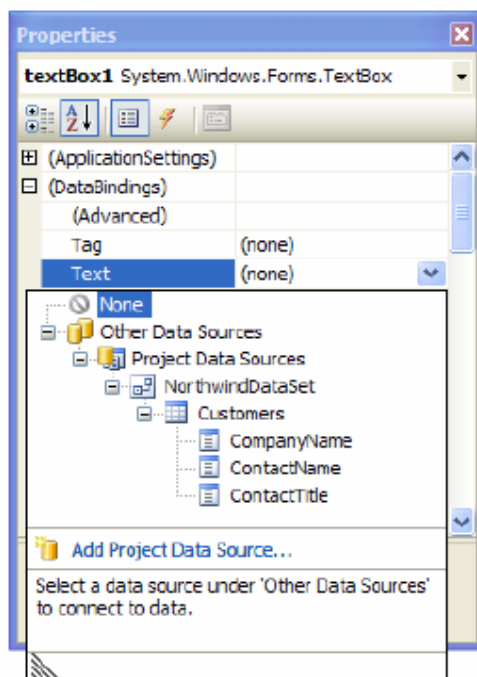
۳- در فرم برنامه روی TextBox اول کلیک کنید تا انتخاب شود. سپس با استفاده از پنجره‌ی Properties روی علامت مثبت کنار خاصیت (DataBindings) این کنترل کلیک کنید. در لیستی که نمایش داده می‌شود، خاصیت Text را انتخاب کرده و روی علامت مثلث کوچک مقابل آن کلیک کنید. به این ترتیب پنجره‌ی DataSource همانند شکل ۲۲-۶ نمایش داده می‌شود. در این پنجره روی لینک Add Project DataSource ... کلیک کنید. تا ویزارد Configuration Wizard، همانند آنچه در مثال ۲۲-۱ مشاهده کرده بودید نمایش داده شود.

شکل ۲۲-۶



- ۴- در پنجره‌ی Choose a DataSource Type آیکن DataBase را انتخاب کرده و روی کلید Next کلیک کنید.
- ۵- در پنجره‌ی Choose Your Data Connection روی دکمه‌ی New Connection کلیک کنید تا کادر Add Connection نمایش داده شود.
- ۶- در کادر Add Connection روی دکمه‌ی Browse کلیک کرده و سپس پایگاه داده‌ی NorthWind را انتخاب کنید و روی Next کلیک کنید.
- به این ترتیب کارد پیغامی نمایش داده خواهد شد و به شما می‌گوید که فایل بانک اطلاعاتی جزئی از پروژه نیست. آیا می‌خواهید این فایل به پوشه پروژه کپی شود و از نسخه‌ی کپی آن استفاده شود. شما Yes را انتخاب کنید.
- ۷- در پنجره‌ی Save the Connection String to the Application Configuration File روی دکمه‌ی Next کلیک کنید.
- ۸- در پنجره‌ی Choose Your DataBase Objects، روی علامت مثبت که در سمت چپ Tables قرار دارد، در لیست Database Objects کلیک کنید. سپس در لیستی که برای Tables نمایش داده می‌شود روی گزینه‌ی Customers کلیک کرده تا فیلدهای این جدول نیز نمایش داده شوند. در پایان نیز با کلیک کردن روی فیلدهای CompanyName، ContactName و ContactTitle آنها را انتخاب کرده و سپس روی کلید Finish کلیک کنید.
- ۹- مجدداً در پنجره‌ی Properties روی علامت مثلث کوچک که در مقابل خاصیت Text قرار دارد کلیک کنید. این بار پنجره‌ی DataSource همانند شکل ۷-۲۲ نمایش داده می‌شود. به ترتیب روی علامت مثبت کنار گزینه‌های Other Data Sources، Project Data Sources، NorthwindDataSet و در آخر نیز Customers کلیک کنید.
- حال روی فیلد CompanyName در این قسمت کلیک کنید. به این ترتیب کادر بسته شده و خاصیت Text در این TextBox به فیلد CompanyName در کنترل DataSet متصل خواهد شد.

شکل ۷-۲۲



۱۰- کنترل TextBox دوم را از فرم برنامه انتخاب کرده و سپس با استفاده از قسمت DataBindings در پنجره‌ی Properties روی خاصیت Text کلیک کنید و مشابه قسمت قبل، فیلد ContactName را به این کادر اختصاص دهید.

۱۱- مراحل قبل را برای TextBox سوم نیز انتخاب کرده و فیلد سوم را به آن متصل کنید.

۱۲- حال در کادر ابزار به قسمت Data بروید و روی کنترل BindingNavigator دوبار کلیک کرده تا یک نمونه از آن در فرم قرار بگیرد. این کنترل بطور اتوماتیک به بالای فرم برنامه متصل خواهد شد.

۱۳- با استفاده از پنجره‌ی Properties خاصیت DataSource کنترل BindingNavigator را برابر با CustomersBindingSource قرار دهید.

۱۴- حال برنامه را اجرا کنید. مشاهده خواهید کرد که فرم برنامه همانند شکل ۲۲-۸ نمایش داده خواهد شد. به وسیله‌ی این فرم می‌توانید بین رکوردهای موجود در بانک اطلاعاتی جا به جا شوید. برای مثال به رکورد بعدی و یا رکورد قبلی بروید. همچنین کلیدهایی نیز وجود دارند که به وسیله‌ی آنها می‌توانید به اولین و یا آخرین رکورد موجود منتقل شوید.

با کلیک کردن روی دکمه‌ی Delete یکی از رکوردهای موجود در DataSet حذف خواهد شد، اما توجه کنید که این رکورد از DataSet حذف می‌شود نه از بانک اطلاعاتی. همچنین کلیک کردن روی دکمه‌ی New نیز باعث می‌شود یک رکورد جدید در DataSet نه در بانک اطلاعاتی ایجاد شود. برای اینکه تغییرات ایجاد شده در بانک اطلاعاتی اعمال شود، لازم است مقداری کد در برنامه وارد کنید.

شکل ۲۲-۸



خلاصه

در این فصل با مفهوم بانک اطلاعاتی آشنا شدید و تعدادی از دستوره‌های SQL برای شما معرفی شد. که مهمترین آنها SELECT می‌باشد.

سپس به بررسی نحوه‌ی استفاده از داده‌های موجود در یک بانک اطلاعاتی در یک برنامه‌ی ویندوزی پرداختیم و مشاهده کردیم که چگونه می‌توان اطلاعات را به کنترل‌هایی مانند DataGridView و یا TextBox متصل کرد. با تعدادی از کنترل‌های مهم و ضروری جهت دسترسی به داده‌های بانک اطلاعاتی آشنا شدید.

در این فصل مشاهده کردید که چگونه می‌توان با استفاده از ویزارد‌های موجود در VS ۲۰۰۵ داده‌های موجود در یک بانک اطلاعاتی را به سرعت به کنترل‌های موجود در فرم متصل کرد. اما در شرایطی ممکن است نیاز به کنترل بیشتر روی نحوه‌ی ارتباط با بانک اطلاعاتی و یا نحوه‌ی متصل شدن کنترل‌ها به داده‌ها داشته باشید. در فصل بعد سعی خواهیم کرد که خط مشی متفاوتی را نسبت به این فصل پیش گرفته و سعی کنیم با استفاده از برنامه‌نویسی، کنترل‌های یک فرم را به داده‌های موجود در یک بانک اطلاعاتی متصل کنیم. همچنین با کنترل‌های دسترسی به داده‌ها بهتر آشنا خواهیم شد و مشاهده خواهیم کرد که چگونه می‌وان از خاصیت‌ها و یا متدهای آنها در برنامه استفاده کرد.

فصل بیست و سوم

برنامه‌نویسی بانک اطلاعاتی با

ADO.NET و SQL SERVER

آنچه که در این فصل یاد خواهید گرفت:

● آشنایی با اشیای ADO.NET

● نحوه‌ی اتصال کنترل‌ها به داده‌ها

● بررسی روشهای جستجو و یا مرتب‌سازی داده‌های درون حافظه با استفاده از اشیای DataView در ADO.NET

● با نحوه انتخاب، درج، ویرایش و یا حذف داده‌های درون یک بانک اطلاعاتی به وسیله‌ی ADO.NET آشنا خواهیم شد.

● برنامه‌نویسی پایگاه‌داده‌های تحت SQL Server

۲۳-۱-مقدمه

در فصل قبلی با بانک اطلاعاتی و برنامه‌نویسی بانک اطلاعاتی به صورت مقدماتی آشنا شدیم. توانستیم اطلاعات داخل یک جدول از بانک اطلاعاتی را در یک برنامه تحت ویندوز بدست آورده و آنها را در جدولی در فرم نمایش دهیم. همچنین بدون اینکه لازم باشد کدی وارد کنیم، توانستیم امکاناتی مانند مرتب کردن داده‌ها را نیز به برنامه اضافه کنیم.

در تمام این موارد از یک ویزارد استفاده کردیم و آن ویزارد کدهای زیادی را برای انجام موارد مورد نظر ما نوشت، کدهایی مانند ایجاد اتصال به بانک اطلاعاتی، تنظیم کردن DataAdapter و نیز ایجاد یک DataSet مخصوص برای جداول مورد نظر ما. استفاده از این روش برای دسترسی ساده به بانک اطلاعاتی و انجام کارهای معمولی مانند دریافت و مشاهده اطلاعات از یک یا چند جدول روش مناسبی است، اما برای نوشتن برنامه‌های بزرگتر، لازم است که کنترل بیشتری بر داده‌ها و یا کنترل‌های موجود در برنامه داشته باشیم و این کار نیز فقط از طریق کدنویسی میسر است.

در این فصل سعی خواهیم کرد که نگاه عمیق‌تری به مبحث دسترسی به بانک اطلاعاتی داشته باشیم. تکنولوژی‌هایی که در فصل قبل برای دسترسی به داده‌ها و یا تغییر در آنها استفاده کردیم، از قبیل کنترل‌هایی برای دریافت اطلاعات از بانک اطلاعاتی، کنترل‌هایی برای ذخیره آنها در حافظه و نیز کنترل‌هایی برای متصل کردن این داده‌ها به کنترل‌های موجود در فرم،

همه مجموعاً به نام ADO.NET شناخته می‌شوند. در این فصل سعی خواهیم کرد با توانایی‌ها و قابلیت‌های درونی ADO.NET برای دسترسی به داده‌های درون یک بانک اطلاعاتی و نیز ایجاد تغییرات در آنها آشنا شویم. همچنین مشاهده خواهیم کرد که چگونه می‌توان داده‌هایی که به وسیله یک DataSet درون حافظه ذخیره شده است را تغییر دهیم، فیلتر کرده و یا ویرایش کنیم.

داده‌هایی که از یک بانک اطلاعاتی استخراج می‌شوند، برای نمایش داده شدن باید به یکی از کنترل‌های موجود در فرم متصل شوند. بنابراین لازم است که اتصال داده‌ها به کنترل‌ها را نیز دقیق‌تر بررسی کنیم. به عبارت دیگر در این فصل مشاهده خواهیم کرد که چگونه می‌توان کنترل‌های موجود در فرم را به گونه‌ای تنظیم کرد که در هر لحظه فقط داده‌های مربوط به یک رکورد را نمایش دهند (برای مثال، مانند TextBox‌ها) و یا چگونه می‌توان با استفاده از اشیایی مانند CurrencyManager بین رکوردها حرکت کرد.

در طی این فصل با نحوه‌ی استفاده از بانک‌های اطلاعاتی ایجاد شده به وسیله‌ی موتور بانک اطلاعاتی SQL Server نیز آشنا خواهیم شد و خواهیم دید چگونه می‌توان به وسیله‌ی سرویس دهنده‌ی اطلاعاتی SqlConnection به آنها دسترسی پیدا کرد. سرویس دهنده‌ی اطلاعاتی SqlConnection، نسبت به سرویس دهنده‌ی اطلاعاتی OleDb (که برای کار با بانک اطلاعاتی ایجاد شده با Access استفاده می‌شود) از سرعت بیشتری برخوردار است، اما فقط می‌تواند با بانک‌های اطلاعاتی تحت SQL Server کار کند.

برای انجام تمرینات این فصل لازم است که به یکی از نرم افزارهای SQL Server، VS SQL Server، یا SQL Server ۲۰۰۵ دسترسی داشته باشید. زیرا در برنامه‌های این فصل از بانک اطلاعاتی نمونه‌ای که در این نرم افزارها وجود دارد (بانک اطلاعاتی Pubs) استفاده شده است.

۲۳-۲-ADO.NET

همانطور که در ابتدای فصل نیز ذکر شد، به مجموعه کنترل‌هایی که برای دسترسی به داده‌های یک بانک اطلاعاتی در NET استفاده می‌شود ADO.NET گفته می‌شود. ADO.NET برای دسترسی به داده‌ها از معماری غیرمتصل استفاده می‌کند. معماری غیرمتصل به این معنی است که ابتدا برنامه به موتور بانک اطلاعاتی مورد نظر متصل شده، داده‌های مورد نیاز خود را از بانک اطلاعاتی دریافت کرده و آنها را در حافظه کامپیوتر ذخیره می‌کند. سپس اتصال برنامه از بانک اطلاعاتی قطع می‌شود و تغییرات مورد نظر خود را در داده‌های موجود در حافظه انجام می‌دهد. هر زمان که لازم باشد، تغییرات ایجاد شده در بانک اطلاعاتی ذخیره شوند، برنامه یک اتصال جدید را به بانک اطلاعاتی ایجاد کرده و از طریق این اتصال تغییراتی را که در داده‌ها اعمال کرده بود را در جدول اصلی ایجاد می‌کند. کنترل‌های اصلی که داده‌های دریافتی از بانک اطلاعاتی را در حافظه نگهداری می‌کند، کنترل DataSet است. این کنترل خود از چند کنترل دیگر مانند اشیایی از نوع DataTable تشکیل شده است. بعد از اینکه داده‌ها در حافظه قرار گرفتند می‌توانید بین آنها جستجو کنید، دستورات SELECT مورد نظر خود را روی آنها اجرا کرده و آنها را به این وسیله فیلتر کنید و یا تغییراتی را در این داده‌ها ایجاد کنید که در طی این فصل با نحوه‌ی انجام این موارد آشنا خواهیم شد.

استفاده از معماری غیرمتصل مزایای زیادی دارد که مهمترین آن افزایش توانایی برنامه در سرویس دادن به چندین کاربر به صورت همزمان است. به عبارت دیگر، می‌توانیم بگوییم با استفاده از این روش می‌توانیم تعداد افرادی که می‌توانند به صورت همزمان از برنامه استفاده کنند را از ده‌ها نفر به صدها نفر افزایش دهیم. دلیل این مورد در این است که در این روش برنامه‌ها فقط در مواقع مورد نیاز به بانک اطلاعاتی متصل می‌شوند و بعد از اجرای وظایف لازم، اتصال خود را قطع می‌کنند، به این ترتیب منابع استفاده شده برای اتصال آنها به بانک اطلاعاتی نیز آزاد شده و در اختیار کاربران دیگر قرار می‌گیرد.

۲۳-۲-۱- فضای نامی Data

کلاس‌های اصلی ADO.NET در فضای نام System.Data قرار می‌گیرند، این فضای نامی خود نیز شامل چند فضای نامی دیگر است که مهمترین آنها عبارتند از System.Data.OleDb و System.Data.SqlClient. فضای نامی System.Data.SqlClient شامل کلاس‌هایی است که برای دسترسی به بانک‌های اطلاعاتی ایجاد شده به وسیله SQL Server به کار می‌رود. فضای نامی System.Data.OleDb نیز حاوی کلاس‌هایی است که برای دسترسی به بانک‌های اطلاعاتی از نوع OLE (مانند بانک‌های اطلاعاتی Access) مورد استفاده قرار می‌گیرد.

در فضای نامی System.Data دو فضای نام دیگر وجود دارند که عبارتند از System.Data.OracleClient و System.Data.Odbc. فضای نامی OracleClient برای دسترسی به بانک اطلاعاتی ایجاد شده به وسیله موتور بانک اطلاعاتی Oracle مورد استفاده قرار می‌گیرد.

کلاس‌های موجود در این فضای نام نیز، همانند کلاس‌های موجود در فضای نام SqlConnection برای دسترسی به بانک‌های اطلاعاتی از نوع Oracle بهینه‌سازی شده‌اند. فضای نام Odbc نیز حاوی کلاس‌هایی است که برای دسترسی به بانک‌های اطلاعاتی قدیمی از نوع ODBC که تکنولوژی OleDb را پشتیبانی نمی‌کنند ایجاد شده است.

فضای نام‌های SqlConnection، OleDb، OracleClient و نیز Odbc در ADO.NET به عنوان سرویس دهنده‌های اطلاعاتی شناخته می‌شوند. در .NET سرویس دهنده‌های اطلاعاتی دیگری نیز وجود دارند، اما در طی این کتاب فقط بر روی دو سرویس دهنده‌ی اول تمرکز خواهیم کرد.

در طی این فصل با استفاده از فضای نام SqlConnection به بانک‌های اطلاعاتی از نوع SQL Server دسترسی خواهیم داشت. البته در ADO.NET استفاده از دیگر سرویس دهنده‌های اطلاعاتی نیز بسیار مشابه استفاده از این سرویس دهنده است. بنابراین به راحتی می‌توانید از تکنیک‌هایی که با استفاده از کلاس‌های موجود در این فضای نامی خواهید آموخت در سرویس دهنده‌های دیگر نیز از قبیل OleDb استفاده کنید و یا تکنیک‌های سرویس دهنده‌هایی مانند OleDb را در این قسمت به کار ببرید. در ADO.NET بر اساس نوع موتور بانک اطلاعاتی که داده‌های شما به وسیله‌ی آن ایجاد شده‌اند، یکی از سرویس دهنده‌های موجود را انتخاب کرده و از آن استفاده می‌کنید. اما لازم نیست که مجدداً نحوه‌ی استفاده از آن سرویس دهنده را مطالعه کنید، زیرا تمامی این سرویس دهنده‌ها بسیار مشابه یکدیگر کار می‌کنند و اگر نحوه‌ی استفاده از یکی از آنها را بیاموزید می‌توانید به راحتی از دیگر سرویس دهنده‌ها نیز استفاده کنید.

کلاس‌های موجود در این سرویس دهنده‌های اطلاعاتی به حدی زیاد هستند که نمی‌توانیم تمام آنها را در این قسمت معرفی کنیم. با این وجود در این قسمت ابتدا با تعدادی از مهمترین آنها که در طی مثال‌های این فصل نیز به کار رفته‌اند آشنا می‌شویم.

این کلاس‌ها عبارتند از:

SqlConnection
SqlCommand
SqlDataAdapter
SqlParameter

به خاطر داشته باشید که این کلاس‌ها فقط برای ارتباط با بانک‌های اطلاعاتی SQL Server مورد استفاده قرار می‌گیرند. برای استفاده از بانک‌های اطلاعاتی OLEDB می‌توانید از کلاس‌های متناظر اینها در فضای نامی System.Data.SqlClient را با استفاده از راهنمای using به برنامه اضافه کرد تا لازم نباشد هر بار با نام کامل آنها را وارد کنیم. بنابراین به خاطر داشته باشید که در ابتدای برنامه‌های این قسمت دستور زیر را نیز اضافه کنید:

```
using System.Data.SqlClient;
```

همچنین برای استفاده از کلاس‌های پایه‌ای ADO.NET مانند DataSet و یا DataView باید فضای نامی System.Data را نیز به برنامه اضافه کنیم. بنابر این در ابتدای برنامه‌های خود دستور زیر را نیز وارد کنید.

```
using System.Data;
```

خوب بهتر است که ابتدا نگاهی به کلاس‌های اصلی موجود در فضای نام SqlConnection داشته باشیم و نحوه‌ی کاربرد آنها را بررسی کنیم.

۲-۲-۲۳- کلاس SqlConnection:

تقریباً می‌توانیم بگوییم که کلاس SqlConnection در قلب کلاس‌هایی قرار دارد که در این قسمت مورد استفاده قرار می‌دهیم، زیرا این کلاس وظیفه برقراری ارتباط بین برنامه و بانک اطلاعاتی برنامه را بر عهده دارد. هنگامی که بخواهید یک نمونه از این کلاس را ایجاد کنید، باید پارامتری را به نام `ConnectionString` به آن ارسال کنید. این پارامتر متغیری از نوع رشته‌ای است که تمام داده‌های مورد نیاز برای اتصال به یک بانک اطلاعاتی را شامل است. البته بعد از ایجاد شیئی از این کلاس، نیز می‌توانیم با استفاده از خاصیت `ConnectionString` در این کلاس مقدار آن را تغییر داده و رشته‌ی جدیدی را برای این پارامتر مشخص کنیم. در برنامه‌هایی که در فصل قبل ایجاد کردیم، VS با استفاده از اطلاعاتی که در کادر `Add Connction` دریافت می‌کرد، چنین متنی را ایجاد کرده و در اختیار `SqlConnection` قرار می‌داد. اما اغلب بهتر است که متن لازم برای `ConnectionString` را خودمان بنویسیم. برای این کار ابتدا باید بدانیم که ساختار این متن‌ها باید چگونه باشد.

ایجاد بخش‌های مختلف `ConnectionString`

ساختار متنی که برای `ConnectionString` باید مورد استفاده قرار گیرد، بستگی به سرویس دهنده‌ی اطلاعاتی دارد که مورد استفاده قرار می‌دهیم. برای مثال اگر بخواهیم از `SQL Server` به عنوان موتور بانک اطلاعاتی برنامه خود استفاده کنیم (به این ترتیب لازم است که از سرویس دهنده‌ی `SqlConnection` در برنامه استفاده کنیم)، باید مقادیر پارامترهای `Server` و `Database` را طبق جدول زیر مشخص کنیم.

پارامتر	توضیح
Server	نام سرور بانک اطلاعاتی که می‌خواهید از آن استفاده کنید. این پارامتر معمولاً حاوی نام کامپیوتری است که موتور بانک اطلاعاتی <code>SQL Server</code> در آن نصب شده است. اگر <code>SQL Server</code> بر روی همان کامپیوتری که برنامه را اجرا می‌کند، نصب شده است، می‌توانید از مقداری مانند <code>local</code> و یا <code>localhost</code> ، برای این پارامتر استفاده کنید. اما اگر <code>SQL Server</code> ای که در کامپیوتر دیگری در شبکه نصب شده است استفاده می‌کنید، لازم است که مقدار این پارامتر را برابر با نام آن کامپیوتر در شبکه قرار دهید. همچنین اگر در آن کامپیوتر بیش از یک <code>SQL Server</code> قرار داشته باشد، باید بعد از نام کامپیوتر یک علامت \ قرار داده و سپس نام <code>SQL Server</code> ای که می‌خواهید مورد استفاده قرار دهید را ذکر کنید.
Database	نام بانک اطلاعاتی که می‌خواهید مورد استفاده قرار دهید، در این پارامتر قرار می‌گیرد (برای مثال بانک اطلاعاتی <code>Pubs</code>).

برای ایجاد امنیت در بانک‌های اطلاعاتی ایجاد شده به وسیله‌ی `SQL Server`، باید هنگام دسترسی به آنها ابتدا هویت استفاده کننده توسط `SQL Server` مشخص شود. بنابراین اگر بخواهیم توسط یک برنامه به داده‌های موجود در یک بانک اطلاعاتی دسترسی داشته باشیم، باید اطلاعات لازم برای این تعیین هویت را همراه با دیگر اطلاعات در متن `ConnectionString` مشخص کنیم. این تعیین هویت به دو روش می‌تواند توسط `SQL Server` انجام شود.

روش اول: استفاده از نام کاربری و کلمه‌ی عبور لازم برای دسترسی به بانک اطلاعاتی است که در این صورت باید این دو پارامتر را به صورت مستقیم در متن `ConnectionString` قرار دهیم.

روش دوم: استفاده از حساب کاربری است که در ویندوز از آن استفاده می‌کنیم.

برای اینکه بتوانیم با استفاده از روش اول توسط SQL Server تعیین هویت شویم، باید پارامترهای نام کاربری و کلمه‌ی عبور را همانطور که در جدول زیر شرح داده شده است، در متن `ConnectionString` قرار دهیم.

پارامتر	توضیح
User ID	این پارامتر باید حاوی نام کاربری باشد که برای اتصال به بانک اطلاعاتی می‌خواهیم از آن استفاده کنیم. برای اینکه بتوانیم با استفاده از این روش از بانک اطلاعاتی استفاده کنیم، باید یک حساب کاربری به این نام در SQL Server ایجاد شده و اجازه‌ی دسترسی به داده‌های مورد نیاز نیز به آن داده شود.
Password	کلمه‌ی عبوری که برای این نام کاربری مورد استفاده قرار می‌گیرد.

علاوه بر این SQL Server می‌تواند به گونه‌ای تنظیم شود که برای تعیین هویت کاربرانی که به آن دسترسی پیدا می‌کنند از حساب ویندوزی که با آن وارد کامپیوتر شده‌اند استفاده کند. در این صورت دیگر نیازی نیست که در متن `ConnectionString` مقادیر نام کاربری و کلمه‌ی عبور را وارد کنید، بلکه فقط باید مشخص کنید که از `Integrated Security` استفاده می‌کنید (این سیستم به این علت `Integrated Security` نامیده می‌شود که با استفاده از آن ویندوز و SQL Server با یکدیگر سعی خواهند که به حداکثر امنیت ممکن در ایجاد یک ارتباط دسترسی پیدا کنند و دیگر نیازی نباشد که نام کاربری و کلمه‌ی عبور را به صورت مستقیم در متن `ConnectionString` قرار دهیم). برای استفاده از این سیستم باید مقدار پارامتر `Integrated Security` را در متن `ConnectionString` برابر با `true` قرار دهید.

البته توجه داشته باشید که در این روش نیز باید در محیط SQL Server به حساب کاربری که برای این فرد در ویندوز ایجاد شده است، اجازه‌ی دسترسی به اطلاعات موجود داده شود، در غیر این صورت کاربر نمی‌تواند به بانک اطلاعاتی متصل شود. برای اینکه بهتر متوجه شوید که چگونه پارامترهای لازم در `ConnectionString` تنظیم می‌شوند، به قطعه کد زیر نگاه کنید. این کد برای ایجاد یک شیء جدید از نوع `SqlConnection` به کار می‌رود.

```
SqlConnection objConnection = new
SqlConnection("Server=localhost;Database=Pubs;User=" + ID=sa;Password=csdotnet;");
```

همانطور که مشاهده می‌کنید این `ConnectionString` برای استفاده از یک SQL Server به کار می‌رود. مقدار `localhost` در پارامتر مشخص می‌کند که سرور SQL که می‌خواهیم از آن استفاده کنیم در کامپیوتری قرار دارد که برنامه در آن اجرا شده است. مقدار پارامتر `Database` نیز نام بانک اطلاعاتی که می‌خواهیم به آن متصل شویم را تعیین می‌کند. در این جا این `ConnectionString` برای دسترسی به بانک اطلاعاتی `Pubs` به کار می‌رود. در آخر نیز مقدار پارامترهای `User ID` و `Password` مشخص می‌شوند که برای تعیین نام کاربری و کلمه‌ی عبور لازم برای دسترسی به اطلاعات به کار می‌روند. دقت کنید که برای تعیین مقدار هر پارامتر از علامت `=` و برای جدا کردن پارامترهای مختلف از یکدیگر از علامت `;` استفاده شده است.

متصل شدن و قطع کردن اتصال از یک بانک اطلاعاتی

بعد از اینکه با ایجاد `ConnctionString` نحوه‌ی برقراری ارتباط با بانک اطلاعاتی را مشخص کردیم، می‌توانیم با استفاده از متدهای `Open` و `Close` در کلاس `SqlConnection` به بانک اطلاعاتی متصل شده و یا اتصال خود را قطع کنیم. یک نمونه از این کار در قطعه کد زیر نشان داده شده است:


```
// Open the database connection
objConnection.Open();
// ... Use the connection
objConnection.Close();
// Close the database connection
```

البته متدها و خاصیت‌های فراوان دیگری در کلاس `SqlConnection` وجود دارند که می‌توانیم در برنامه از آنها استفاده کنیم، اما مواردی که در اینجا با آنها آشنا شدیم، پرکاربردترین آنها به شمار می‌روند و فکر می‌کنم که برای شروع فقط آشنایی با این موارد کافی باشد.

۲۳-۲-۳ کلاس `SqlCommand`

کلاس `SqlCommand` حاوی یک دستور `SQL` برای اجرا روی داده‌های دریافت شده از بانک اطلاعاتی است. این دستور می‌تواند یک دستور `SELECT` برای انتخاب داده‌هایی خاص، یک دستور `INSERT` برای درج داده‌های جدید در بانک اطلاعاتی، یک دستور `DELETE` برای حذف داده‌ها از بانک اطلاعات و یا حتی فراخوانی یک پروسیجر ذخیره شده در بانک اطلاعاتی باشد. دستور `SQL`‌ای که در این کلاس نگهداری می‌شود می‌تواند شامل پارامترها نیز باشد.

از متد سازنده‌ی کلاس `SqlCommand` چندین نسخه `Overload` شده است، اما ساده‌ترین آنها برای ایجاد یک شیء از کلاس `SqlCommand` هیچ پارامتری را دریافت نمی‌کند. بنابراین می‌توانید بعد از ایجاد شیء با استفاده از خاصیت‌ها و یا متدهای موجود، آن شیء را تنظیم کنید. قطعه کد زیر نحوه‌ی ایجاد یک شیء از نوع `SqlCommand` را نمایش می‌دهد:

```
SqlCommand objCommand = new SqlCommand();
```

در برنامه‌های بانک اطلاعاتی معمولاً از اشیای ایجاد شده از کلاس `SqlCommand` به تنهایی استفاده نمی‌کنند، بلکه آنها را با `DataSet` ها و `DataAdapter` ها به کار می‌برند. به این ترتیب می‌توانند از دستور `INSERT, SELECT` و یا ... که در آن نگهداری می‌شود برای مقاصد مورد نیاز استفاده کنند. همچنین اشیای `SqlCommand` می‌توانند به همراه اشیای ایجاد شده از کلاس `DataReader` مورد استفاده قرار گیرند. کلاس `DataReader` کاربردی همانند `Dataset` دارد، اما منابع سیستم (مانند حافظه و ...) را کمتر مصرف می‌کند و انعطاف‌پذیری کمتری نیز دارد. در ادامه‌ی این فصل بیشتر تمرکز خود را روی نحوه‌ی استفاده از این اشیاء با کلاس `DataSet` قرار خواهیم داد.

خاصیت `Connection`

قبل از اینکه بتوانیم از یک شیء از کلاس `SqlCommand` استفاده کنیم باید بعضی از خاصیت‌های آن را تنظیم کنیم. اولین خاصیتی که باید تنظیم شود، خاصیت `Connection` است. این خاصیت همانطور که در قطعه کد زیر نمایش داده شده است، می‌تواند یک مقدار از نوع `SqlConnection` را دریافت کند:

```
objCommand.Connection = objConnection;
```

برای اینکه بتوانیم دستور `SQL`‌ای که در این شیء نگهداری می‌شود را با موفقیت اجرا کرده و نتیجه‌ی آن را دریافت کنیم، ابتدا با استفاده از متد `Open` در `SqlConnection` اتصال با بانک اطلاعاتی را برقرار کرده و سپس دستور را اجرا کنیم.

خاصیت `CommandText`

خاصیت بعدی که باید تنظیم کنیم، خاصیت `CommandText` است. این خاصیت متنی را دریافت می‌کند که می‌تواند حاوی یک دستور `SQL` و یا فراخوانی یک پروسیجر ذخیره شده در بانک اطلاعاتی باشد که باید روی داده‌ها اجرا شود. برای مثال، قطعه کد زیر یک نمونه از دستور `SQL` که در این خاصیت قرار داده شده است را نمایش می‌دهد:

```
SqlConnection objConnection = new
SqlConnection("Server=localhost;Database=Pubs;User " +
"ID=sa;Password=csdotnet;");
SqlCommand objCommand = new SqlCommand();
objCommand.Connection = objConnection;
objCommand.CommandText = "INSERT INTO authors " +
"(au_id, au_lname, au_fname, contract) " +
"VALUES('123-45-6789', 'Barnes', 'David', 1)";
```


دستور INSERT یکی از دستورات ساده SQL است که برای درج یک رکورد از اطلاعات در یک جدول به کار می‌رود. این دستور در این قسمت بیان می‌کند که "یک رکورد جدید از اطلاعات در جدول authors ایجاد کن. سپس فیلد au_id این رکورد را برابر با '۶۷۸۹-۴۵-۱۲۳' قرار بده، فیلد au_lname را برابر با Barnes قرار بده، فیلد au_fname را برابر با 'David' قرار بده و فیلد Contract را نیز برابر با ۱ قرار بده."

روش استفاده از دستور INSERT برای درج یک ردیف از اطلاعات در یک جدول به این صورت است که بعد از دستور INSERT INTO نام جدولی که می‌خواهیم اطلاعات در آن قرار بگیرد را ذکر می‌کنیم. سپس نام فیلدهایی را که باید کامل کنیم، را در داخل یک پرانتز می‌آوریم و هر یک را نیز با یک ویرگول از هم جدا می‌کنیم. سپس عبارت VALUES نوشته و در یک پرانتز دیگر، مقدار مورد نظر برای آن فیلدها را به ترتیب وارد می‌کنیم.

در اینجا فرض کردیم که هنگام نوشتن برنامه، مقداری که باید در هر یک از فیلدها قرار گیرد مشخص است. اما همانطور که می‌دانید در اغلب موارد چنین شرایطی رخ نمی‌دهد و مقدار هر یک از این فیلدها در طول اجرای برنامه توسط کاربر تعیین می‌شوند. خوشبختانه می‌توانیم دستورات SQL را به گونه‌ای ایجاد کنیم که همانند یک متد، پارامتر دریافت کنند. سپس هنگام اجرای برنامه این پارامترها را از کاربر دریافت کرده و آنها را در دستور قرار می‌دهیم و دستور را اجرا می‌کنیم. بهتر است مقداری هم با پارامترهایی که می‌توانند در شیء SqlCommand قرار گیرند آشنا شویم.

خاصیت Parameters

قبل از اینکه بتوانیم نحوه‌ی استفاده از پارامترها در یک دستور SQL را ارائه کنیم، باید با مفهوم جاگیرنده‌ها آشنا شوید که متغیرهایی هستند که در یک دستور SQL قرار می‌گیرند و می‌توانند در زمان اجرای برنامه جای خود را با عباراتی خاص عوض کنند. این متغیرها با علامت @ در یک دستور مشخص می‌شوند و هنگامی که از آنها در یک دستور SQL استفاده کنیم، قبل از اجرای دستور باید تمامی آنها را با مقادیر مناسب تعویض کنیم. برای مثال، اگر بخواهیم در دستور قبل مقادیر لازم برای قسمت VALUES از دستور INSERT را در زمان اجرای برنامه مشخص کنیم، باید جای آنها را با چهار جاگیرنده به صورت زیر عمل کنیم:

همانطور که مشاهده می‌کنید در اینجا به جای اینکه از چند مقدار ثابت در دستور استفاده کنیم، از چند جاگیرنده استفاده کرده‌ایم. همچنین جاگیرنده‌ها در دستورها با استفاده از @ مشخص شده‌اند. البته هیچ ضرورتی ندارد که نام یک جاگیرنده همان فیلدی باشد که قرار است مقدار جاگیرنده در آن قرار بگیرد. اما این کار باعث می‌شود که برنامه خوانا تر شده و درک آن ساده‌تر باشد.

```
SqlConnection objConnection = new
SqlConnection("Server=localhost;Database=Pubs;User " +
"ID=sa;Password=csdotnet;");
SqlCommand objCommand = new SqlCommand();
objCommand.Connection = objConnection;
objCommand.CommandText = "INSERT INTO authors " +
"(au_id, au_lname, au_fname, contract) " +
"VALUES(@au_id, @au_lname, @au_fname, @au_contract)";
```

خوب، بعد از اینکه با استفاده از این روش پارامترهایی را در دستور ایجاد کردیم، باید قبل از دستور SQL این جاگیرنده‌ها را با مقادیر مناسب تعویض کنیم. این کار به صورت اتوماتیک توسط برنامه در زمان اجرای دستور انجام می‌شود. اما ابتدا باید پارامترهایی را ایجاد کرده و آن را در لیست Parameters در شیء ایجاد شده از کلاس SqlCommand قرار می‌دهیم تا برنامه بداند هنگام اجرای دستور هر جاگیرنده را باید با مقدار چه تغییری در برنامه عوض کند. توجه کنید که اصطلاح پارامتر

در این قسمت به پارامترهایی اشاره می‌کند که برای اجرای یک دستور SQL و یا پروسیجر ذخیره شده در بانک اطلاعاتی لازم است، نه به پارامترهایی که در ویژوال #C به متدها فرستاده می‌شود.

برای دسترسی به لیست پارامترهایی که در یک شی از کلاس SqlCommand وجود دارد می‌توانیم از خاصیت Parameters در این کلاس استفاده کنیم. این خاصیت حاوی لیستی از جاگیرنده‌ها به همراه متغیرهای وابسته به آنها است. بنابراین در برنامه قبل از اجرای دستور، باید به وسیله‌ی این لیست مشخص کنیم که هر جاگیرنده با مقدار چه متغیری باید تعویض شود. ساده‌ترین روش انجام این کار در کد زیر نشان داده شده است.

```
SqlConnection objConnection = new
SqlConnection("Server=localhost;Database=Pubs;User
ID=sa;Password=csdotnet;");
SqlCommand objCommand = new SqlCommand();
objCommand.Connection = objConnection;
objCommand.CommandText = "INSERT INTO authors " +
"(au_id, au_lname, au_fname, contract) " +
"VALUES(@au_id, @au_lname, @au_fname, @au_contract)";
objCommand.Parameters.AddWithValue("@au_id",
txtAuId.Text);
objCommand.Parameters.AddWithValue("@au_lname",
txtLastName.Text);
objCommand.Parameters.AddWithValue("@au_fname",
txtFirstName.Text);
objCommand.Parameters.AddWithValue("@au_contract",
chkContract.Checked);
```

متد `AddWithValue` نام یک جاگیرنده و متغیری که مقدار مربوط به آن را در زمان اجرای برنامه نگهداری می‌کند را به عنوان پارامتر دریافت کرده و آن را به لیست `Parameters` اضافه می‌کند. برای مثال، در این قسمت مشخص کرده‌ایم که هنگام اجرای دستور، مکان جاگیرنده با نام `@au_id` باید با مقدار خاصیت `Text` کنترل `txtAuId` عوض شود. همچنین مکان جاگیرنده با نام `@au_lname` نیز با مقدار خاصیت `Text` مربوط به کنترل `txtLastName` عوض شود و ...

متد `ExecuteNonQuery`

بعد از انجام تمام این مراحل می‌توانید دستور موجود در این شی را روی داده‌های بانک اطلاعاتی اجرا کنید. برای این کار ابتدا باید اتصال خود را به بانک اطلاعاتی برقرار کنید. سپس با فراخوانی متد `ExecuteNonQuery` دستور موجود در شی `SqlCommand` را اجرا کنید. البته این متد همانطور که نام آن نیز مشخص می‌کند فقط زمانی کاربرد دارد که بخواهیم دستوری را روی بانک اطلاعاتی اجرا کنیم که داده‌ای را بر نمی‌گرداند. برای مثال، اگر دستور موجود در شی `SqlCommand` یک دستور `SELECT` باشد که اطلاعاتی را از جدول استخراج کرده و آنها را به برنامه دهد، نمی‌توانیم برای اجرای آن از این متد استفاده کنیم. اما اگر دستور مورد استفاده فقط تغییراتی را در داده‌های بانک اطلاعاتی ایجاد کند (برای مثال، یک رکورد از اطلاعات را به جدول اضافه کند) می‌توانیم با فراخوانی آن متد دستور را در بانک اطلاعاتی اجرا کنیم.

این متد بعد از اجرا، عددی را به عنوان خروجی بر می‌گرداند که مشخص کننده‌ی تعداد رکوردهایی است که با اجرای این دستور SQL تغییر کرده‌اند. این عدد معمولاً برای بررسی صحت اجرای دستور مورد استفاده قرار می‌گیرد. قطعه کد زیر نحوه‌ی استفاده از دستور `ExecuteNonQuery` را در برنامه نشان می‌دهد.

```
SqlConnection objConnection = new
SqlConnection("Server=localhost;Database=Pubs;User
ID=sa;Password=csdotnet;");
SqlCommand objCommand = new SqlCommand();
objCommand.Connection = objConnection;
objCommand.CommandText = "INSERT INTO authors " +
"(au_id, au_lname, au_fname, contract) " +
"VALUES(@au_id, @au_lname, @au_fname, @au_contract)";
objCommand.Parameters.AddWithValue("@au_id",
txtAuId.Text);
objCommand.Parameters.AddWithValue("@au_lname",
```

```
txtLastName.Text);
objCommand.Parameters.AddWithValue("@au_fname",
txtFirstName.Text);
objCommand.Parameters.AddWithValue("@au_contract",
chkContract.Checked);
objConnection.Open();
objCommand.ExecuteNonQuery();
objConnection.Close();
```

۲۳-۲-۴-کلاس SqlDataAdapter

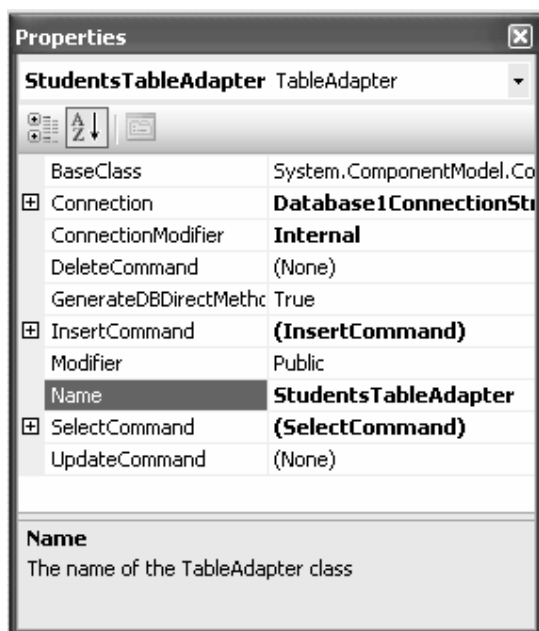
کلاس SqlDataAdapter در برنامه‌های بانک اطلاعاتی، همانند پلی بین جداول بانک اطلاعاتی و نیز داده‌های موجود در حافظه که به وسیله‌ی DataSet نگهداری می‌شوند، عمل می‌کنند. این کلاس برای دسترسی به بانک اطلاعاتی از شی ایجاد شده از کلاس SqlCommand ی که به آن نسبت داده می‌شود استفاده می‌کند و همانطور که می‌دانید، هر شی از کلاس SqlCommand حاوی شیئی از کلاس SqlConnection است که ارتباط آن را با بانک اطلاعاتی برقرار می‌کند. بنابراین می‌توانیم بگوئیم که کلاس DatAdapter برای دسترسی به بانک اطلاعاتی از کلاس SqlCommand و SqlConnection استفاده می‌کند.

کلاس SqlDataAdapter دارای خاصیتی به نام SelectCommand است که از دستور موجود در آن شیئی برای دریافت داده‌های مورد نیاز برنامه از بانک اطلاعاتی به کار می‌رود. به عبارت دیگر، SqlDataAdapter دستوری که در این خاصیت نگهداری می‌شود را روی بانک اطلاعاتی اجرا کرده و نتایج آن را در کلاس‌هایی مانند DataSet و یا DataTable قرار می‌دهد تا در برنامه‌ها مورد استفاده قرار گیرند. علاوه بر این کلاس SqlDataAdapter دارای خاصیت‌هایی به نام InsertCommand، DeleteCommand و UpdateCommand است که هر یک شیئی از نوع SqlCommand را قبول می‌کنند و SqlDataAdapter از دستور ذخیره شده در هر یک از آنها به ترتیب برای حذف، درج و یا ویرایش داده‌ها در بانک اطلاعاتی استفاده می‌کند. در حقیقت هنگامی که ما در طی برنامه تغییراتی را در درون داده‌های موجود در حافظه نگهداری می‌کنیم، SqlDataAdapter با استفاده از دستورات موجود در این خاصیت‌ها تغییرات ما را از داده‌های حافظه به داده‌های موجود در بانک اطلاعاتی منتقل می‌کند. ممکن است ابتدا این موارد کمی پیچیده به نظر برسند، اما استفاده از آنها مانند تمام قسمت‌های دیگری که تاکنون مشاهده کرده‌ایم ساده هستند. در قسمت‌های قبل مشاهده کردید که چگونه می‌توان دستورات SELECT مورد نیاز را برای انتخاب داده‌ها از بانک اطلاعاتی ایجاد کرد. برای تکمیل دستورات مورد نیاز برای کلاس SqlDataAdapter نیز فقط کافی است که دستور SELECT مورد نظر خود را وارد کنید. در VS کلاسی به نام CommandBuilder وجود دارد که می‌تواند بر اساس دستور SELECT وارد شده، دستورات UPDATE، INSERT و یا DELETE مناسب تولید کند. بنابراین بهتر است که ابتدا با هم خاصیت SelectCommand و نحوه‌ی استفاده از آن را بررسی کنیم. سپس مشاهده خواهیم کرد که چگونه می‌توان با استفاده از CommandBuilder دستورات دیگر را نیز تولید کرد.

خاصیت SelectCommand

همانطور که در شکل ۲۳-۱ نیز نشان داده شده است، خاصیت SelectCommand در کلاس SqlDataAdapter برای دریافت داده‌های مورد نیاز در برنامه از بانک اطلاعاتی و قرار دادن آنها در DataSet به کار می‌رود.

شکل ۲۳-۱



هنگامی که بخواهید با استفاده از کلاس `DataAdapter` اطلاعات مورد نیاز را از یک بانک اطلاعاتی دریافت کنید، ابتدا باید خاصیت `SelectCommand` را در `DataAdapter` تنظیم کنید. این خاصیت شیئی از نوع `SqlCommand` دریافت کرده که این شی مشخص می‌کند داده‌ها چگونه باید از بانک اطلاعاتی انتخاب و نیز چه داده‌هایی باید انتخاب شوند. اشیایی که از کلاس `SqlCommand` ایجاد می‌شوند، همانطور که در قسمت قبلی مشاهده کردید خود خاصیت‌هایی دارند که قبل از استفاده باید آنها را تنظیم کرد. این خاصیت عبارتند از:

- `Connection`: یک شیئی از کلاس `SqlConnection` در این قسمت قرار گرفته و نحوه‌ی اتصال به بانک اطلاعاتی را مشخص می‌کند.
- `CommandText`: دستور `SQL` و یا نام پروسیجر ذخیره شده در بانک اطلاعاتی که باید توسط شی اجرا شود، در این قسمت ذخیره می‌شود.

در قسمت قبل از یک دستور `SQL` مشخص در خاصیت `CommandText` از کلاس `SqlCommand` استفاده کردیم. اما اگر بخواهیم نام یک پروسیجر ذخیره شده در بانک اطلاعاتی را در این خاصیت قرار دهیم تا اجرا شود، باید خاصیت دیگری به نام `CommandType` را نیز در کلاس `SqlCommand` تنظیم کرده و مقدار آن را برابر با `StoredProcedure` قرار دهیم تا مشخص شود که متن درون `CommandText` نام یک پروسیجر ذخیره شده است، نه یک دستور `SQL`. البته در این فصل فقط از دستورات `SQL` در خاصیت `CommandText` استفاده می‌کنیم، بنابراین نیازی نیست که خاصیت `CommandType` را تغییر داده و برابر با مقدار خاصی قرار دهیم.

تنظیم خاصیت `SelectCommand` با استفاده از دستور `SQL`

قطعه کدی که در زیر آورده شده است نحوه‌ی تنظیم خاصیت‌های مورد نیاز برای اجرای یک دستور `SQL` به وسیله‌ی کلاس `DataAdapter` را نمایش می‌دهد:

```
// Declare a SqlDataAdapter object...
SqlDataAdapter objDataAdapter = new SqlDataAdapter();
// Assign a new SqlCommand to the SelectCommand property
objDataAdapter.SelectCommand = new SqlCommand();
// Set the SelectCommand properties...
objDataAdapter.SelectCommand.Connection = objConnection;
objDataAdapter.SelectCommand.CommandText =
"SELECT au_lname, au_fname FROM authors " +
"ORDER BY au_lname, au_fname";
```

اولین کاری که باید در این قسمت انجام دهیم این است که شیئی را از نوع SqlDataAdapter ایجاد کنیم. سپس باید خاصیت SelectCommand آن را تنظیم کنیم. برای تنظیم این خاصیت باید یک شیئی از کلاس SqlCommand را به آن نسبت دهیم، بنابراین شیئی را از این کلاس ایجاد کرده و تنظیمات مربوط به Connection آن را انجام می‌دهیم تا بتواند به یک بانک اطلاعاتی متصل شود. در آخر نیز خاصیت CommandText آن را نیز برابر با یک دستور SQL قرار می‌دهیم تا آن را روی بانک اطلاعاتی اجرا کرده و نتیجه را دریافت کند.

تنظیم خاصیت SelectCommand با استفاده از پروسیجر ذخیره شده

در این قسمت مشاهده خواهیم کرد که برای استفاده از یک پروسیجر ذخیره شده در برنامه، خاصیت‌های لازم را باید چگونه تنظیم کرد. همانطور که در قسمت قبلی نیز گفته شد، یک پروسیجر ذخیره شده، یک مجموعه از دستورات SQL است که تحت نام مشخص و به صورت یک واحد در بانک اطلاعاتی ایجاد شده و نگهداری می‌شود. در این قسمت فرض می‌کنیم پروسیجر به نام usp_select در بانک اطلاعاتی وجود دارد که می‌خواهیم به جای استفاده از دستور SQL، آن را فراخوانی کرده و نتایج اجرای آن را دریافت کنیم. قطعه کد زیر نحوه‌ی انجام این کار را نمایش می‌دهد:

```
// Declare a SqlDataAdapter object...
SqlDataAdapter objDataAdapter = new SqlDataAdapter();
// Assign a new SqlCommand to the SelectCommand property
objDataAdapter.SelectCommand = new SqlCommand();
// Set the SelectCommand properties...
objDataAdapter.SelectCommand.Connection = objConnection;
objDataAdapter.SelectCommand.CommandText = "usp_select";
objDataAdapter.SelectCommand.CommandType =
CommandType.StoredProcedure;
```

همانطور که مشاهده می‌کنید در این قطعه برنامه، برخلاف قسمت قبل که یک دستور SQL را در خاصیت CommandText قرار می‌دادیم، از نام یک پروسیجر ذخیره شده استفاده کردیم. پس باید به نحوی مشخص کنیم که متن داخل CommandText نام یک پروسیجر ذخیره شده است، نه یک دستور SQL. برای تعیین نوع متن موجود در این خاصیت، باید از خاصیت CommandType استفاده کنیم. مقدار پیش فرض این خاصیت برابر با CommandType.Text است که مشخص می‌کند متن موجود یک دستور SQL است. در این قطعه کد، این مقدار را تغییر داده و برابر با CommandType.StoredProcedure قرار می‌دهیم تا مشخص شود مقدار موجود در خاصیت CommandText نام یک پروسیجر ذخیره شده است.

استفاده از CommandBuilder برای ایجاد دستورات SQL دیگر

با استفاده از خاصیت SelectCommand موجود در کلاس DataAdapter می‌توانیم داده‌های مورد نیاز در برنامه را از بانک اطلاعاتی استخراج کرده و در یک DataSet در حافظه قرار دهیم. سپس در طول برنامه می‌توانیم به کاربر اجازه دهیم تا تغییرات مورد نظر خود را در داده‌های موجود در حافظه ایجاد کرده و بعد از اتمام آنها، این تغییرات را به داده‌های موجود در بانک منعکس کنیم. برای این کار لازم است که دستورات SQL مورد نیاز برای درج، حذف و یا ویرایش داده‌های دریافتی را به کلاس DataAdapter اضافه کنیم تا این کلاس بتواند با استفاده از این دستورات، تغییرات ایجاد شده را در بانک اطلاعاتی وارد کند.

اما برای ایجاد این دستورات لازم است که به زبان SQL تسلط بیشتری داشته باشیم. خوشبختانه روش ساده‌تری هم برای انجام این کار وجود دارد و آن استفاده از کلاس CommandBuilder است که می‌تواند با توجه به دستور SELECT که برای DataAdapter وارد کرده‌ایم، دستورات INSERT، UPDATE و نیز DELETE مناسب تولید کند. قطعه کد زیر نحوه‌ی استفاده از این دستورات را نمایش می‌دهد.

```
// Declare a SqlDataAdapter object...
SqlDataAdapter objDataAdapter = new SqlDataAdapter();
// Assign a new SqlCommand to the SelectCommand property
objDataAdapter.SelectCommand = new SqlCommand();
```

```
// Set the SelectCommand properties...
objDataAdapter.SelectCommand.Connection = objConnection;
objDataAdapter.SelectCommand.CommandText = "usp_select";
objDataAdapter.SelectCommand.CommandType =
CommandType.StoredProcedure;
// automatically create update/delete/insert commands
SqlCommandBuilder objCommandBuilder =
new SqlCommandBuilder(objDataAdapter);
```

با استفاده از این کلاس، دستورات لازم برای منعکس کردن تغییرات ایجاد شده از DataSet به بانک اطلاعاتی به صورت اتوماتیک نوشته می‌شود. در ادامه‌ی فصل با نحوه‌ی انجام این کار بیشتر آشنا خواهیم شد، اما فعلاً بهتر است ببینیم که چگونه می‌توان داده‌ها را از یک بانک اطلاعاتی استخراج کرده و در یک DataSet در حافظه قرار داد.

متد Fill

با استفاده از متد Fill در کلاس DataAdapter می‌توانید دستور SQL موجود در خاصیت SelectCommand را در بانک اطلاعاتی اجرا کرده، و سپس داده‌های برگشتی از اجرای این دستور را درون یک DataSet در حافظه قرار دهید. البته قبل از استفاده از این متد، باید شیئی از نوع DataSet ایجاد کنیم.

```
// Declare a SqlDataAdapter object...
SqlDataAdapter objDataAdapter = new SqlDataAdapter();
// Assign a new SqlCommand to the SelectCommand property
objDataAdapter.SelectCommand = new SqlCommand();
// Set the SelectCommand properties...
objDataAdapter.SelectCommand.Connection = objConnection;
objDataAdapter.SelectCommand.CommandText = "usp_select";
objDataAdapter.SelectCommand.CommandType =
CommandType.StoredProcedure;
DataSet objDataSet = new DataSet();
```

حال که یک شی DataSet و نیز DataAdapter مورد نیاز را ایجاد کردیم، می‌توانیم با استفاده از متد Fill داده‌های بانک اطلاعاتی را در DataSet قرار دهیم. متد Fill نیز همانند بسیاری از متدهای دیگر دارای نسخه‌های گوناگونی است، اما یکی از پرکاربردترین آنها به صورت زیر مورد استفاده قرار می‌گیرد:

```
SqlDataAdapter.Fill(DataSet, String);
```

پارامتر DataSet در این متد مشخص‌کننده‌ی نام DataSet ای است که باید داده‌ها در آن قرار بگیرند. پارامتر String نیز نام جدولی را مشخص می‌کند که داده‌ها در آن جدول قرار می‌گیرند. DataSet ها نیز می‌توانند همانند بانک‌های اطلاعاتی شامل چندین جدول مختلف از اطلاعات باشند. بنابراین هنگامی که می‌خواهیم داده‌ای را در آن قرار دهیم باید مشخص کنیم که نام جدولی که داده‌ها در آن قرار می‌گیرند چه باید باشد؟ در اینجا می‌توانیم هر نام که تمایل داشته باشیم برای جدول انتخاب کنیم، اما بهتر است همواره از اسامی جدولی استفاده کنیم که داده‌ها از آن گرفته شده‌اند. به این ترتیب درک برنامه بسیار راحت‌تر خواهد بود.

قطعه کد زیر یک پروسیجر ذخیره شده در بانک اطلاعاتی را اجرا کرده و نتایج برگشتی از آن را به وسیله‌ی متد Fill در جدولی به نام authors در DataSet قرار می‌دهد:

```
// Declare a SqlDataAdapter object...
SqlDataAdapter objDataAdapter = new SqlDataAdapter();
// Create an instance of a new select command object
objDataAdapter.SelectCommand = new SqlCommand();
// Set the SelectCommand properties...
objDataAdapter.SelectCommand.Connection = objConnection;
objDataAdapter.SelectCommand.CommandText = "usp_select";
objDataAdapter.SelectCommand.CommandType =
CommandType.StoredProcedure;
DataSet objDataSet = new DataSet();
// Fill the DataSet object with data...
objDataAdapter.Fill(objDataSet, "authors");
```


متد Fill برای اتصال به بانک اطلاعاتی از شی Connection ای که در خاصیت SelectCommand قرار دارد استفاده می‌کند. این متد ابتدا بررسی می‌کند که این اتصال Connection به بانک اطلاعاتی برقرار است یا نه. در صورتی که اتصال برقرار باشد، متد Fill داده‌های مورد نیاز را از بانک اطلاعاتی بدست آورده، اما اتصال Connection با بانک اطلاعاتی را قطع نمی‌کند. اگر هم ارتباط شی Connection با بانک اطلاعاتی قطع باشد، متد Fill با فراخوانی متد Open ارتباط را برقرار کرده و پس از بدست آوردن اطلاعات مورد نیاز، متد Close را فراخوانی می‌کند تا اتصال به بانک اطلاعاتی مجدداً قطع شود.

به این ترتیب داده‌ها از بانک اطلاعاتی درون حافظه قرار می‌گیرند و می‌توانید به صورت مستقل آنها را تغییر دهید. دقت کنید که ابتدای کلاس DataSet کلمه‌ی Sql وجود ندارد. دلیل این مورد هم این است که این کلاس متعلق به فضای نام System.Data.SqlClient نیست بلکه در فضای نام System.Data قرار دارد. به عبارت دیگر، کلاس DataSet به سرویس دهنده‌ی اطلاعاتی خاصی از قبیل SqlConnection یا OleDb تعلق ندارد و وظیفه‌ی آن نگهداری اطلاعات به دست آمده (به هر نحوی) در حافظه است. هنگامی که اطلاعات را در حافظه قرار دادیم دیگر نیازی نیست بدانیم که این اطلاعات از کجا به دست آمده‌اند (تا زمانی که بخواهیم آنها را دوباره در بانک اطلاعاتی قرار دهیم).

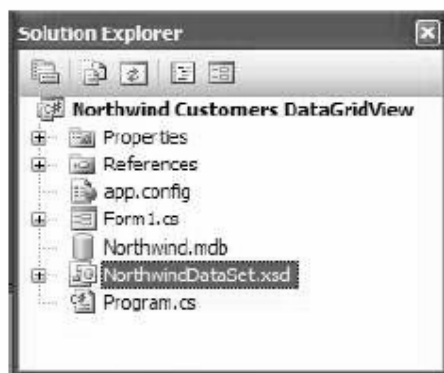
۲۳-۲-۵-کلاس DataSet

همانطور که گفتیم کلاس DataSet برای نگهداری اطلاعات به دست آمده از بانک اطلاعاتی در حافظه به کار می‌رود. این کلاس شامل مجموعه‌ای از جداول، رابطه‌ها، قید و شرط‌ها و دیگر مواردی است که از بانک اطلاعاتی خوانده شده است. این کلاس خود همانند یک موتور بانک اطلاعاتی کوچک عمل می‌کند که می‌تواند داده‌ها را درون خود در جدولی مجزا نگهداری کرده و به کاربر اجازه دهد که آنها را ویرایش کند. همچنین با استفاده از کلاس DataView پرس‌وجوهایی را روی داده‌های موجود در آن اجرا کرد.

داده‌هایی که در این کنترل قرار دارند از بانک اطلاعاتی قطع هستند، ارتباطی با بانک ندارند. در طول برنامه می‌توانیم داده‌های موجود در آن را حذف کرده، ویرایش یا اضافه کنیم و بعد از اتمام تغییرات مورد نظر، مجدداً با استفاده از DataAdapter به بانک اطلاعاتی متصل شده و تغییرات را در بانک اطلاعاتی ذخیره کنیم.

کلاس DataSet از ساختار XML برای ذخیره داده‌ها استفاده می‌کند. به این ترتیب می‌توانید داده‌های موجود در یک شیء از کلاس DataSet را به سادگی در یک فایل ذخیره کرده و یا آن را با استفاده از شبکه به کامپیوتر دیگری منتقل کنید. البته هنگام برنامه‌نویسی و کار با DataSet لازم نیست با آنها در قالب XML رفتار کنید. بلکه کافی است تمام کارهای مورد نظر خود را با استفاده از خاصیت‌ها و یا متدهای موجود در DataSet انجام دهید، بقیه‌ی امور را کلاس DataSet کنترل خواهد کرد.

مانند هر سند XML، یک DataSet نیز دارای یک الگو است (فایلی که ساختار داده‌های درون یک یا چند فایل XML را شرح می‌دهد). در فصل قبل هنگامی که با استفاده از ویزارد یک DataSet را به برنامه اضافه کردیم، فایلی با پسوند XSD ایجاد شد و الگوی DataSet در آن قرار گرفت (شکل ۲۳-۲).



شکل ۲۳-۲

این فایل حاوی الگوی XML اطلاعاتی بود که به وسیله‌ی CustomerDataSet نگهداری می‌شد. VS به وسیله‌ی این فایل، کلاسی را از کلاس DataSet مشتق می‌کند تا بتواند داده‌های دریافت شده از بانک اطلاعاتی را در شیئی از آن کلاس نگهداری کند. البته تمام این موارد نیز از دید برنامه‌نویس دور می‌ماند و به صورت درونی توسط DataSet انجام می‌شود.

همچنین می‌توانیم فیلدهای درون یک جدول از DataSet را به کنترل‌های درون فرم متصل کنیم، تا آن کنترل‌ها داده‌های خود را به وسیله‌ی آن فیلد به دست آورند. در فصل قبل مقداری با انجام این کار آشنا شدید، در ادامه‌ی فصل نیز بیشتر در این مورد صحبت خواهیم کرد.

۲۳-۲-۶-کلاس DataView

کلاس DataView عموماً برای جستجو، مرتب کردن، فیلتر کردن، ویرایش کردن و یا حرکت کردن در بین داده‌های درون یک DataSet مورد استفاده قرار می‌گیرد. کنترل DataView یک کنترل قابل اتصال است، یعنی همانطور که می‌توان کنترل‌ها را به یک DataSet متصل کرد، می‌توان آنها را به یک DataView نیز متصل کرد. در این مورد در ادامه فصل بیشتر صحبت خواهیم کرد.

همانطور که گفتیم یک کنترل DataSet می‌تواند شامل چندین جدول باشد که هر یک از آنها به وسیله‌ی یک کنترل DataTable مشخص می‌شود. در حقیقت هنگامی که با استفاده از DataAdapter داده‌هایی را درون یک DataSet قرار می‌دهید، ابتدا یک جدول جدید ایجاد کرده (یک شیئی جدید از نوع DataTable) و سپس داده‌ها را درون آن قرار می‌دهید و به DataSet اضافه می‌کنید. کاری که کنترل DataView انجام می‌دهد این است که به شما اجازه می‌دهد به صورتی که تمایل دارید به داده‌های یکی از جداول DataSet نگاه کنید. برای مثال، آنها را به صورت مرتب شده مشاهده می‌کنید و یا همانند یک بانک اطلاعاتی، دستورات SQL خاصی را روی جداول اجرا کرده و نتایج را ببینید. می‌توانید یک شی از کلاس DataView را به گونه‌ای ایجاد کنید که تمامی داده‌های موجود در یک جدول از DataSet باشد و فقط نحوه‌ی نمایش آنها را تغییر دهید. برای مثال، اگر جدولی به نام authors در DataSet وجود داشته باشد که براساس LastName مرتب شده است، می‌توانید یک DataView را به گونه‌ای ایجاد کنید که حاوی همان اطلاعات باشد، اما آنها را ابتدا براساس FirstName و سپس LastName مرتب کند و یا حتی می‌توانید یک DataView ایجاد کنید که فقط فیلد LastName از جدول authors را نمایش دهد و یا فقط فیلد FirstName را نمایش دهد و ...

البته با وجود اینکه می‌توانید به وسیله‌ی کلاس DataView اطلاعات درون یک DataTable را به گونه‌ای دیگر مشاهده کنید، باید دقت داشته باشید که اطلاعات درون DataTable نیز منعکس خواهد شد و بر عکس.

برای ایجاد یک شی از کلاس DataView، باید نام جدولی که می‌خواهیم به آن متصل شود را در متد سازنده‌ی آن مشخص کنیم. در قطعه کد زیر، یک شی از کلاس DataView ایجاد شده و به جدول authors از ObjDataSet متصل می‌شود.

دقت کنید که برای دسترسی به یک جدول خاص از خاصیت Tables در کلاس DataSet به همراه نام جدول مورد نظر استفاده کرده‌ایم.

```
// Set the DataView object to the DataSet object...
DataView objDataView = new
DataView(objDataSet.Tables("authors"));
```

خاصیت Sort

هنگامی که یک شی از نوع DataView ایجاد کرده و آن را به یک جدول درون DataSet متصل کردید، می‌توانید نحوه‌ی نمایش داده‌ها را تغییر دهید. برای مثال تصور کنید که می‌خواهید داده‌های درون جدول را به گونه‌ای متفاوت مرتب کنید. برای این کار می‌توانید از خاصیت Sort در کلاس DataView استفاده کرده و مقدار آن را برابر با نام ستون و یا ستون‌هایی قرار دهید که می‌خواهید داده‌ها براساس آنها مرتب شوند. قطعه کد زیر جدول authors را به وسیله‌ی DataView‌ای که در قسمت قبل ایجاد کردیم، براساس FirstName و LastName مرتب می‌کند:

```
objDataView.Sort = "au_fname, au_lname";
```

همانطور که مشاهده می‌کنید عبارتی که به این خاصیت نسبت داده می‌شود، همانند عبارتی است که در مقابل ORDER BY دستور SELECT زبان SQL وارد می‌کردیم. در این قسمت نیز همانند دستور SELECT، تمام مرتب‌سازی‌ها به طور پیش‌فرض به صورت صعودی انجام می‌شوند و برای اینکه بتوانیم ترتیب مرتب شدن آنها را به نزولی تغییر دهیم، باید در مقابل نام ستون از عبارت DESC استفاده کنیم. برای مثال، قطعه کد زیر، داده‌های موجود در جدول authors را براساس فیلد FirstName به صورت صعودی و فیلد LastName به صورت نزولی مرتب می‌کند:

```
objDataView.Sort = "au_fname, au_lname DESC";
```

خاصیت RowFilter

علاوه بر مرتب کردن داده‌ها، با استفاده از DataView می‌توانید داده‌های موجود در یک جدول را فیلتر کنید، به گونه‌ای که فقط داده‌هایی که دارای شرایط خاصی هستند نمایش داده شوند. این امکان همانند قسمت WHERE از دستور SELECT در زبان SQL است که شرط خاصی را برای نمایش داده شدن داده‌ها ایجاد می‌کند. برای فیلتر کردن اطلاعات نیز می‌توانید از خاصیت RowFilter استفاده کرده و شرط موردنظر خود را در آن قرار دهید. نحوه‌ی وارد کردن دستورات در این قسمت نیز همانند وارد کردن شرط‌ها در قسمت WHERE از دستور SELECT است. فقط توجه داشته باشید، به علت اینکه کل عبارت شرط باید درون علامت " قرار بگیرند، پس اگر بخواهید رشته‌ای را در شرط مشخص کنید باید آن را درون علامت ' قرار دهید. برای مثال، قطعه کد زیر در جدول authors فقط افرادی را که LastName آنها برابر با Green است نمایش می‌دهد:

```
// Set the DataView object to the DataSet object...
DataView objDataView = new
DataView(objDataSet.Tables("authors"));
```

و یا قطعه کد زیر در جدول authors افرادی که LastName آنها مخالف Green است را بر می‌گرداند:

```
// Set the DataView object to the DataSet object...
DataView objDataView = new
DataView(objDataSet.Tables("authors"));
```

به علاوه در شزطی که در این قسمت وارد می‌کنید می‌توانید با استفاده از عبارات AND یا OR چندین شرط را با یکدیگر ترکیب کرده و سپس داده‌ها را بر اساس شرط نهایی نمایش دهید. برای مثال، قطعه کد زیر در جدول authors افرادی را نمایش می‌دهد که FirstName آنها با حرف D شروع شده و LastName آنها نیز برابر با Green باشد:

```
objDataView.RowFilter =
"au_lname <> 'Green' AND au_fname LIKE 'D*';
```

متد Find

برای پیدا کردن یک رکورد خاص از اطلاعات در بانک اطلاعاتی، می‌توانید از متد Find در کلاس DataView استفاده کنید. البته قبل از فراخوانی این متد، باید داده‌های جدول را بر حسب فیلدی که می‌خواهید جستجو را براساس آن انجام دهید مرتب کنید. به عبارت دیگر قبل از فراخوانی متد Find، باید داده‌های موجود در جدول را براساس ستونی که حاوی کلید مورد نظر شماست مرتب کنید.

برای مثال، تصور کنید که می‌خواهید با استفاده از ObjDataView که در قسمت قبل ایجاد کردیم، در جدول authors به دنبال رکوردی بگردید که FirstName آن برابر با Ann باشد. برای این کار، ابتدا باید جدول را براساس فیلد au_fname مرتب کنید، سپس با استفاده از متد Find به دنبال Ann بگردید. قطعه کد زیر روش انجام این کار را نمایش می‌دهد:

```
int intPosition;
objDataView.Sort = "au_fname";
intPosition = objDataView.Find("Ann");
```

به این ترتیب DataView در جدول به نام فردی می‌گردد که FirstName آن برابر با Ann باشد و شماره‌ی مکان آن را بر می‌گرداند. اگر چنین فردی در جدول پیدا نشد، این مقدار تهی را بر می‌گرداند. دقت کنید به محض اینکه متد Find اولین گزینه را پیدا کرد، مکان آن را برگردانده و از جستجوی ادامه‌ی جدول صرف‌نظر می‌کند، بنابراین اگر می‌دانید که بیش از یک فرد با این نام در جدول وجود دارد برای مشاهده تمام آنها می‌توانید از روش فیلتر کردن که توضیح داده شد استفاده کنید.

همچنین این متد به کوچکی و یا بزرگی حروف حساس نیست و هر فردی که نام او Ann و یا ANN و یا ... باشد را برمی‌گرداند. البته دقت کنید که این متد دقیقاً به دنبال متنی که وارد شده است برمی‌گردد. بنابراین باید تمام کلمه و یا کلماتی که می‌خواهید جستجو براساس آن انجام گیرد را به صورت دقیق در این قسمت وارد کنید. برای مثال، اگر می‌خواهید در جدول به دنبال فردی با نام خانوادگی Del Castillo بگردید، نمی‌توانید Del را به عنوان پارامتر به متد Find بفرستید و انتظار داشته باشید که این نام را برای شما برگرداند. بلکه باید نام کامل او را در این قسمت وارد کنید، همانند دستور زیر:

```
objDataView.Sort = "au_lname";
intPosition = objDataView.Find("del castillo");
```

در قسمت قبل مشاهده کردید که با استفاده از DataView می‌توان یک جدول را براساس چند فیلد مرتب کرد. همین مورد برای جستجو کردن نیز صادق است. به عبارت دیگر، می‌توانید براساس چند فیلد به جستجوی داده‌ها بپردازید. برای این کار، بعد از مرتب کردن جدول، آرایه‌ای از نوع Object ایجاد می‌کنید و سپس مقدار مورد نظر برای هر ستون را در آن قرار می‌دهید. سپس این آرایه را به عنوان پارامتر به متد Find می‌فرستید. برای مثال، اگر بخواهیم ببینیم که آیا فردی با نام Simon و نام خانوادگی Watts در جدول وجود دارد یا نه. می‌توانیم از قطعه کد زیر استفاده کنیم:

```
int intPosition;
Object[] arrValues = new Object[1];
objDataView.Sort = "au_fname, au_lname";
// Find the author named "Simon Watts".
arrValues[0] = "Simon";
arrValues[1] = "Watts";
intPosition = objDataView.Find(arrValues);
```

نکته: دقت کنید که در این قسمت حتماً باید آرایه‌ای از نوع object به متد Find فرستاده شود. دلیل این امر هم در این است که در NET تمام نوع‌های داده‌ای از کلاس object مشتق می‌شوند. بنابراین اگر بخواهیم آرایه‌ای داشته باشیم که هر متغیری را بتوانیم در آن قرار دهیم، باید آن را از نوع object تعریف کنیم. در اینجا نیز لازم است آرایه‌ای داشته باشیم که بتوانیم متغیری از هر نوع داده‌ای را در آن قرار دهیم. برای مثال، فرض کنید بخواهید جستجو در جدول authors را به گونه‌ای تغییر دهید که سن آنها بالاتر از ۲۵ و نیز نام آنها برابر Ann است را پیدا کند. در این صورت، باید یک متغیر از نوع عددی و یک متغیر از نوع رشته‌ای را در آرایه قرار دهید.

۲۳-۳- استفاده از کلاس‌های ADO.NET در عمل

تاکنون با اصول کار کلاس‌های موجود در ADO.NET آشنا شدیم و مشاهده کردیم که چگونه می‌توان داده‌ها را به وسیله این کلاس‌ها از بانک اطلاعاتی SQL SERVER بدست آورده و یا در آنها وارد کرد. اما تا این قسمت از فصل فقط ذهن خود را با یک سری مطالب تئوری درگیر کرده بودیم، و برای اینکه مطمئن شویم نحوه استفاده از این کلاس‌ها، متدها، خاصیت‌ها و ... را درست درک کرده‌ایم، بهترین راه این است که از آنها در یک مثال عملی استفاده کنیم. در دو مثال بعدی با استفاده از قدرت DataSet داده‌ها را از بانک اطلاعاتی استخراج کرده و به کاربر نمایش خواهیم داد.

در مثال اولی از کلاس‌های SqlDataAdapter، SqlConnection، SqlCommand و نیز DataSet استفاده کرده و بوسیله آنها یک برنامه ساده ایجاد می‌کنیم که داده‌ها را از یک بانک اطلاعاتی بدست آورد و در یک DataGrid نمایش دهد. در واقع برنامه‌ای که در این قسمت خواهیم نوشت، عملکردی بسیار مشابه برنامه فصل قبل خواهد داشت. البته با این تفاوت عمده که در این قسمت به جای استفاده از ویزارد، از کد نویسی استفاده خواهیم کرد.

نکته: هنگام نوشتن برنامه‌های واقعی، معمولاً از ویزارد و کدنویسی به صورت همزمان استفاده می‌کنند تا بتوانند به سرعت و به راحتی برنامه‌هایی با انعطاف‌پذیری بالا ایجاد کنند. کنترل‌هایی که در قسمت قبل با استفاده از جعبه ابزار به فرم اضافه کردیم را در این قسمت با استفاده از کد ایجاد خواهیم کرد. البته نحوه استفاده آنها در هر دو روش یکسان خواهد بود. همچنین در فصل قبل اغلب از ویزاردها استفاده می‌کردیم، در صورتی که در این فصل بیشتر بر کدنویسی تمرکز خواهیم کرد.

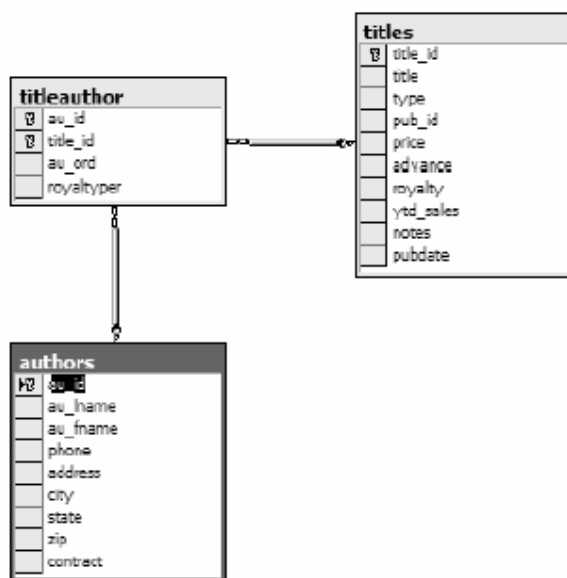
۲۳-۳-۱- کاربرد DataSet در برنامه

قبل از اینکه نوشتن برنامه‌ی این قسمت را شروع کنیم، بهتر است که به بررسی داده‌هایی که می‌خواهیم در این برنامه نمایش دهیم و نیز رابطه‌ی بین آنها بپردازیم. اطلاعات این برنامه از بانک اطلاعاتی pubs در SQL Server ۲۰۰۰ استخراج می‌شود. البته اگر از نسخه‌های ۲۰۰۵، ۷ و یا msde به جای SQL Server ۲۰۰۰ استفاده می‌کنید نیز باید همین اطلاعات را در بانک اطلاعاتی pubs مشاهده کنید.

این بانک اطلاعاتی مربوط به یک انتشارات فرضی است. در این برنامه می‌خواهیم لیستی از نویسندگان، کتابهایی که تاکنون چاپ کرده‌اند و قیمت هر کدام را نمایش دهیم. در شکل ۲۳-۳ این جدول‌ها را به همراه فیلدهای موجود در هر کدام و نیز رابطه‌های بین آنها را نمایش می‌دهد.

همانطور که در شکل مشاهده می‌کنید در این برنامه می‌خواهیم نام و نام خانوادگی نویسنده را از جدول authors بدست آورده و به همراه عنوان و قیمت کتاب او که در جدول titles قرار دارد، در برنامه نمایش دهیم. به علت اینکه یک کتاب می‌تواند بیش از یک نویسنده داشته باشد و نیز یک نویسنده نیز می‌تواند بیش از یک کتاب نوشته باشد، اطلاعات این دو جدول در جدول دیگری به نام titleauthor به یکدیگر متصل شده‌اند.

شکل ۲۳-۳



با توجه به رابطه‌ی بین جدول و نیز اطلاعاتی که می‌خواهیم از آنها استخراج کنیم، دستور SELECT ای که باید در این مورد استفاده کنیم مشابه زیر خواهد بود:

```

SELECT au_lname, au_fname, title, price
FROM authors
JOIN titleauthor ON authors.au_id = titleauthor.au_id
JOIN titles ON titleauthor.title_id = titles.title_id
ORDER BY au_lname, au_fname
  
```

خط اول این دستور نام فیلدهایی را که می‌خواهیم از جدول استخراج کنیم را نمایش می‌دهد و خط دوم هم مشخص کننده‌ی نام جدول اصلی است که داده‌ها از آن استخراج می‌شوند. در این قسمت داده‌ها از دو جدول authors و نیز titles استخراج می‌شوند، اما جدول authors را به عنوان جدول اصلی در نظر می‌گیریم.

خط سوم بین ستون au_id در جدول authors و نیز همین ستون در جدول titleauthor رابطه برقرار می‌کند. به این ترتیب هر زمان که یک رکورد از جدول authors انتخاب شود، تمام رکوردهای موجود در جدول titleauthor که مقدار ستون au_id آنها برابر با مقدار این ستون در رکورد انتخاب شده از جدول authors باشد نیز انتخاب خواهند شد.

خط چهارم نیز مانند خط سوم، بین جدول titles و جدول titleauthor از طریق ستون title_id رابطه برقرار می‌کند. به این ترتیب هر زمان که یک رکورد از جدول titleauthor انتخاب شود، رکوردهای متناظر آن در جدول titles نیز انتخاب خواهند شد. خط آخر نیز اطلاعات را براساس نام خانوادگی و سپس نام، به صورت صعودی مرتب می‌کند.

نکته: ممکن است که توضیحات این دستور SELECT برای درک آن کافی نباشد، اما در هر صورت برای اتمام این مثال کافی است. مسلماً هنگامی که بخواهید برنامه‌های بانک اطلاعاتی واقعی بنویسید، نیاز خواهید داشت که مفهوم بانک‌های اطلاعاتی رابطه‌ای را درک کرده باشید و نیز بتوانید دستورات SELECT پیچیده‌ای برای انتخاب داده‌ها از چندین جدول بنویسید. بنابراین در این صورت بهتر است که کتاب‌هایی را در این زمینه نیز مطالعه کنید.

مثال ۲۳-۱

(۱) با استفاده از VS یک برنامه‌ی ویندوزی جدید به نام DATASETEXAMPLE ایجاد کنید.

(۲) با استفاده از پنجره‌ی Properties، خاصیت‌های فرم را به صورت زیر تغییر دهید:

خاصیت Size را برابر با ۲۳۰/۶۰۰ قرار دهید.

خاصیت StartPosition را برابر با CenterScreen قرار دهید.

خاصیت Text را برابر با Bound DataSet قرار دهید.

(۳) با استفاده از قسمت Data در جعبه ابزار، یک کنترل DataGridView به فرم برنامه اضافه کرده و خاصیت‌های آن را به صورت زیر تغییر دهید:

خاصیت Name را برابر با grdAuthorTitles قرار دهید.

خاصیت Anchor را برابر با Top/Left/Right/Bottom قرار دهید.

خاصیت Location را برابر با ۰/۰ قرار دهید.

خاصیت Size را برابر با ۵۹۲/۲۰۳ قرار دهید.

(۴) ویرایشگر کد مربوط به کلاس Form1 را باز کرده و ابتدا فضای نام‌هایی که در طول برنامه به آنها نیاز خواهیم داشت را به برنامه اضافه کنید. برای این کار دستور زیر را به بالای تعریف کلاس Form1 اضافه کنید:

```
// Using Data and SqlClient namespaces...
using System.Data;
using System.Data.SqlClient;
public partial class Form1 : Form
{
}
```

(۵) در مرحله‌ی بعد لازم است که اشیای لازم برای دسترسی به بانک اطلاعاتی و دریافت داده‌ها را ایجاد کنیم. بنابراین کدهای مشخص شده در زیر را به برنامه‌ی خود اضافه کنید. مطمئن شوید که اطلاعات مربوط به نام کاربری و نیز کلمه‌ی عبور در ConnectionString به درستی وارد شده است.

```
public partial class Form1 : Form
{
    SqlConnection objConnection = new SqlConnection(
        "server=localhost;database=pubs;" +
        "user id=sa;password=");
    SqlDataAdapter objDataAdapter = new SqlDataAdapter();
    DataSet objDataSet = new DataSet();
    public Form1()
    {

```

نکته: دقت کنید که اگر سرور بانک اطلاعاتی که از استفاده می‌کنید در کامپیوتر دیگری به غیر از کامپیوتری که در حال استفاده از آن هستید قرار دارد، باید مقدار پارامتر Server را به نام کامپیوتر حاوی SQL Server تغییر دهید. همچنین باید مقدار پارامتر Password, User ID را نیز به گونه‌ای تنظیم کنید که به یک نام کاربری و کلمه‌ی عبور مناسب در سرور اشاره کنند. در غیر اینصورت برنامه نخواهد توانست به داده‌های لازم در بانک اطلاعاتی دسترسی پیدا کند. اگر نام کاربری که در سرور تعریف کرده‌اید کلمه عبور ندارد، باید قسمت Password را ConnectionString ذکر کنید اما در مقابل آن چیزی ننویسید. برای مثال Password=.

(۶) به قسمت طراحی فرم Form1 برگردید و روی نوار عنوان آن دو بار کلیک کنید تا متد مربوط به رویداد load فرم به صورت اتوماتیک ایجاد شود. سپس کد مشخص شده در زیر را به این متد اضافه کنید:

```
private void Form1_Load(object sender, EventArgs e)
{
    // Set the SelectCommand properties...
    objDataAdapter.SelectCommand = new SqlCommand();
    objDataAdapter.SelectCommand.Connection =
    objConnection;
    objDataAdapter.SelectCommand.CommandText =
    "SELECT au_name, au_fname, title, price " +
    "FROM authors " +
    "JOIN titleauthor ON authors.au_id = " +
    "titleauthor.au_id " +
    "JOIN titles ON titleauthor.title_id = " +

```

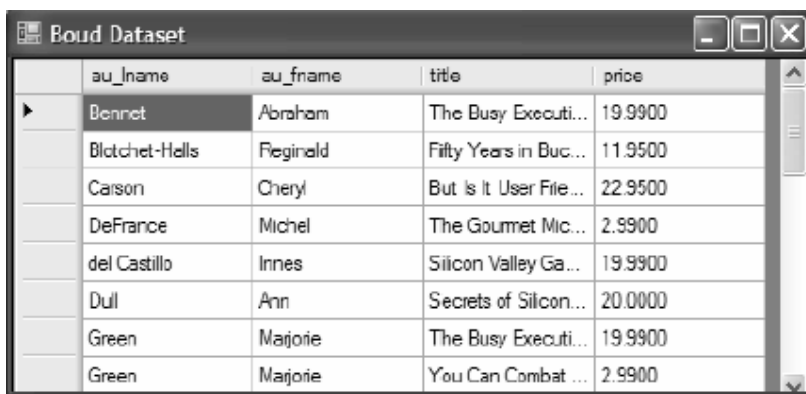
```

"titles.title_id " +
"ORDER BY au_lname, au_fname";
objDataAdapter.SelectCommand.CommandType =
CommandType.Text;
// Open the database connection...
objConnection.Open();
// Fill the DataSet object with data...
objDataAdapter.Fill(objDataSet, "authors");
// Close the database connection...
objConnection.Close();
// Set the DataGridView properties
// to bind it to our data...
grdAuthorTitles.AutoGenerateColumns = true;
grdAuthorTitles.DataSource = objDataSet;
grdAuthorTitles.DataMember = "authors";
// Clean up
objDataAdapter = null;
objConnection = null;
}

```

۷) با اجرای برنامه، نتیجه‌ای مشابه شکل ۴-۲۳ مشاهده خواهید کرد.

شکل ۴-۲۳



	au_lname	au_fname	title	price
▶	Bennet	Abraham	The Busy Executi...	19.9900
	Blotch-Hells	Reginald	Fifty Years in Buc...	11.9500
	Carson	Cheryl	But is it User File...	22.9500
	DeFrance	Michel	The Gourmet Mic...	2.9900
	del Castillo	Innes	Silicon Valley Ga...	19.9900
	Dull	Ann	Secrets of Silicon...	20.0000
	Green	Marjorie	The Busy Executi...	19.9900
	Green	Marjorie	You Can Combat ...	2.9900

۸) دقت کنید که کنترل DataGridView دارای خاصیت درونی مرتب کردن داده‌ها است. بنابراین اگر روی یکی از نام‌های ستون‌ها کلیک کنید، داده‌های موجود در فرم براساس آن ستون مرتب می‌شوند. همچنین کلیک دوباره بر روی نام یک ستون باعث می‌شود که داده‌ها برحسب آن ستون به صورت نزولی مرتب شوند.

نکته: در این برنامه به علت کمبود جا، کدهای مربوط به مدیریت خطاها و استثناهای احتمالی حذف شده است، اما بهتر است در برنامه‌ای که می‌نویسید این کدها را نیز اضافه کنید.

مثال ۴-۲۳

۱- در زیر از تغییراتی که می‌توانید در یک DataGridView انجام دهید تا داده‌ها بهتر نمایش داده شوند، آورده شده است:

- عنوان ستون‌ها را برابر با نام مناسبی قرار دهید.
- اندازه‌ی هر ستون را به گونه‌ای تغییر دهید تا بتوان به راحتی داده‌های آن را مطالعه کرد.
- رنگ هر ردیف از اطلاعات را به گونه‌ای تغییر دهید که به صورت متمایز نمایش داده شوند.
- داده‌ها را در ستون‌ها به صورت راست - چپ قرار دهید (برای نمایش داده‌های عددی).

۲- برای انجام این موارد، در متد form_Load تغییرات مشخص شده در زیر را اعمال کنید:

```

private void Form1_Load(object sender, EventArgs e)
{
    // Set the SelectCommand properties...
    objDataAdapter.SelectCommand = new SqlCommand();
    objDataAdapter.SelectCommand.Connection =
    objConnection;
    objDataAdapter.SelectCommand.CommandText =

```

```

"SELECT au_lname, au_fname, title, price " +
"FROM authors " +
"JOIN titleauthor ON authors.au_id = " +
"titleauthor.au_id " +
"JOIN titles ON titleauthor.title_id = " +
"titles.title_id " +
"ORDER BY au_lname, au_fname";
objDataAdapter.SelectCommand.CommandType =
CommandType.Text;
// Open the database connection...
objConnection.Open();
// Fill the DataSet object with data...
objDataAdapter.Fill(objDataSet, "authors");
// Close the database connection...
objConnection.Close();
// Set the DataGridView properties
// to bind it to our data...
grdAuthorTitles.AutoGenerateColumns = true;
grdAuthorTitles.DataSource = objDataSet;
grdAuthorTitles.DataMember = "authors";
// Declare and set
// the currency header alignment property...
DataGridViewCellStyle objAlignRightCellStyle = new
DataGridViewCellStyle();
objAlignRightCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
// Declare and set the alternating rows style...
DataGridViewCellStyle objAlternatingCellStyle = new
DataGridViewCellStyle();
objAlternatingCellStyle.BackColor = Color.WhiteSmoke;
grdAuthorTitles.AlternatingRowsDefaultCellStyle =
objAlternatingCellStyle;
// Declare and set the style for currency cells ...
DataGridViewCellStyle objCurrencyCellStyle = new
DataGridViewCellStyle();
objCurrencyCellStyle.Format = "c";
objCurrencyCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
// Change column names
// and styles using the column name
grdAuthorTitles.Columns["price"].HeaderCell.Value =
"Retail Price";
grdAuthorTitles.Columns["price"].HeaderCell.Style =
objAlignRightCellStyle;
grdAuthorTitles.Columns["price"].DefaultCellStyle =
objCurrencyCellStyle;
// Change column names
// and styles using the column index
grdAuthorTitles.Columns[0].HeaderText = "Last Name";
grdAuthorTitles.Columns[1].HeaderText = "First Name";
grdAuthorTitles.Columns[2].HeaderText = "Book Title";
grdAuthorTitles.Columns[2].Width = 225;
// Clean up
objDataAdapter = null;
objConnection = null;
objCurrencyCellStyle = null;
objAlternatingCellStyle = null;
objAlignRightCellStyle = null;
}

```

به کمک کلاس DataGridViewCellStyle می‌توانید تنظیمات نمایشی سطر و ستون‌های DataGrid را قالب‌بندی کنید.

۳- مجدداً برنامه را اجرا کنید. مشاهده خواهید کرد که داده‌ها در جدولی مشابه شکل ۲۳-۵ نمایش داده می‌شوند. با مقایسه‌ی این شکل با شکل ۲۳-۴ متوجه تفاوت‌های ایجاد شده در برنامه خواهید شد

شکل ۲۳-۵

Boud Dataset				
	Last Name	First Name	Book Title	Retail Price
▶	Bennet	Abraham	The Busy Executive's Database Guide	۱۹/۹۹ ریال
	Blotchet-Hals	Reginald	Fifty Years in Buckingham Palace Kitchens	۱۱/۹۵ ریال
	Carson	Cheryl	But Is It User Friendly?	۲۲/۹۵ ریال
	DeFrance	Michel	The Gourmet Microwave	۲/۹۹ ریال
	del Castillo	Innes	Silicon Valley Gastronomic Treats	۱۹/۹۹ ریال
	Dull	Ann	Secrets of Silicon Valley	۲۰/۰۰ ریال
	Green	Marjorie	The Busy Executive's Database Guide	۱۹/۹۹ ریال
	Green	Marjorie	You Can Combat Computer Stress!	۲/۹۹ ریال

۲۳-۴-اتصال داده‌ها

کنترل `DataGridView` بهترین کنترل برای نمایش تمام داده‌ها در فرم برنامه است. این کنترل علاوه بر این قابلیت می‌تواند به راحتی به کاربر اجازه دهد که داده‌ها را حذف و یا ویرایش کند و یا داده‌های جدید را در جدول وارد کند. اما با این وجود ممکن است در شرایطی بخواهید در هر لحظه فقط یک سطر از داده‌ها را در برنامه نمایش دهید. در این مواقع تنها راه این است که تعدادی کنترل ساده مانند `TextBox` بر روی فرم قرار داده و هر یک از آنها را به یکی از فیلدهای جدول در برنامه متصل کنیم، سپس در هر لحظه اطلاعات مربوط به یک سطر از اطلاعات را در این کنترل‌ها قرار دهیم با استفاده از این روش می‌توانید کنترل بیشتری روی داده‌ها داشته باشید. اما کدی که برای این کار باید در برنامه وارد کنید نیز مشکل‌تر خواهد بود، زیرا باید برنامه‌ای بنویسید که هر یک از کنترل‌های روی فرم را به فیلد مربوط متصل کند. سپس قسمتی را در فرم برنامه طراحی کنید که به وسیله آن بتوان در بین سطرهای اطلاعات حرکت کرد. بنابراین برای انجام این کار لازم است که درگیر اموری مانند اتصال کنترل‌های ساده به داده‌ها و نیز مدیریت اتصال شویم.

در بحث راجع به اتصال داده‌ها، منظور از کنترل‌های ساده کنترل‌هایی هستند که در هر لحظه فقط می‌توانند مقدار یک داده را در خود نگهداری کنند، برای مثال مانند `TextBox`، `CheckBox`، `RadioButton` و یا کنترل‌هایی از این قبیل. کنترل‌هایی مانند `ListBox`، `ComboBox` و یا `DataGridView` که در هر لحظه می‌توانند بیش از یک آیتم از داده‌های موجود در برنامه را نمایش دهند به عنوان کنترل‌های ساده در نظر گرفته نمی‌شوند.

۲۳-۴-۱- `CurrencyManager`, `BindingContext`

هر فرم دارای شیئی از نوع `BindingContext` است که اتصالات کنترل‌های درون فرم را مدیریت می‌کند. بنابراین به علت اینکه فرم برنامه شما به صورت درونی دارای چنین شیئی است، نیازی نیست که آن را در یک کد ایجاد کنید.

شی `BindingContext` در حقیقت یک مجموعه از اشیاء از نوع `CurrencyManager` را مدیریت می‌کند. وظیفه‌ی `CurrencyManager` نیز این است که بین کنترل‌هایی که به منبع داده‌ای (مثلاً `DataSet`) متصل هستند و منبع داده‌ای، و نیز این کنترل‌ها با دیگر کنترل‌هایی که در فرم به همان منبع داده‌ای متصل هستند هماهنگی برقرار کند. به این ترتیب می‌توان مطمئن شد که تمام این کنترل‌ها در فرم در حال نمایش داده‌های موجود در یک سطر هستند. شی `CurrencyManager` می‌تواند این هماهنگی را بین کنترل‌ها و منابع داده‌ای مختلفی مانند `DataSet`، `DataTable`، `DataGridView`، `DataSetView` ایجاد کند. هر زمان که منبع داده‌های جدید به فرم برنامه اضافه کنید، یک شی `CurrencyManager` جدید نیز به صورت اتوماتیک ایجاد می‌شود. به این ترتیب کار با کنترل‌های متصل به یک منبع داده‌ای در فرم برنامه بسیار ساده خواهد شد.

اگر در برنامه‌ی خود از چندین منبع داده‌ای مختلف استفاده می‌کنید، می‌توانید یک متغیر از نوع `CurrencyManager` ایجاد کرده و آن را به `CurrencyManager` مربوط به منبع داده‌ای مورد نظر خود در `BindingContext` ارجاع دهید. به

این ترتیب به وسیله‌ی این متغیر می‌توانید به CurrencyManager مورد نظر خود BindingContext دسترسی داشته باشید و به وسیله‌ی آن نمایش داده‌ها را بر روی فرم کنترل کنید.

قطعه کد زیر با استفاده از شی DataSet ای که در برنامه‌ی قبل ایجاد کرده بودیم، یک ارجاع به شی CurrencyManager ای که منبع داده‌ای مربوط به جدول Authors را کنترل می‌کرد ایجاد می‌کند. برای این کار، ابتدا یک متغیر از کلاس CurrencyManager ایجاد می‌کنیم. سپس مقدار این متغیر را برابر با CurrencyManager مربوط به منبع داده‌ای objDataSet در BindingContext قرار می‌دهیم. البته دقت کنید شیئی که در BindingContext ذخیره می‌شود از CurrencyManager نیست و باید با استفاده از عملکرد () آن را به صورت صریح به CurrencyManager تبدیل کنیم:

```
CurrencyManager objCurrencyManager;
objCurrencyManager =
(CurrencyManager) (this.BindingContext[objDataSet]);
```

بعد از اینکه با استفاده از این کد ارجاعی به این شی ایجاد کردیم، می‌توانیم با استفاده از خاصیت Position موقعیت رکورد جاری را کنترل کنیم. برای مثال، کد زیر موقعیت رکورد جاری را یک واحد افزایش می‌دهد:

منظور از رکورد جاری در CurrencyManager یک منبع داده‌ای، رکوردی است که تمام کنترل‌های ساده‌ی فرم که به این منبع داده‌ای متصل شده‌اند باید اطلاعات آن رکورد را نمایش دهند.

```
objCurrencyManager.Position += 1;
```

و یا دستور زیر باعث می‌شود که تمام کنترل‌های ساده‌ای که به objDataSet متصل شده‌اند، اطلاعات رکورد قبلی را نمایش دهند:

```
objCurrencyManager.Position -= 1;
```

همچنین برای نمایش اطلاعات مربوط به اولین رکورد در objDataSet می‌توانیم از کد زیر استفاده کنیم:

```
objCurrencyManager.Position = 0;
```

خاصیت Count در کلاس CurrencyManager حاوی تعداد رکوردهایی است که در منبع داده‌ای که به وسیله‌ی CurrencyManager مدیریت می‌شود وجود دارد. بنابراین برای نمایش اطلاعات مربوط به رکورد آخر در فرم برنامه می‌توانید از کد زیر استفاده کنید:

```
objCurrencyManager.Position = objCurrencyManager.Count - 1;
```

دقت کنید که در این کد از تعداد رکوردهای موجود منهای یک برای رفتن به آخرین رکورد استفاده کرده‌ایم. دلیل این کار این است که اندیس رکوردها در این شی از صفر شروع می‌شود. بنابراین اندیس رکورد آخر برابر با تعداد کل رکوردها منهای یک خواهد بود.

۲۳-۴-۲- اتصال کنترل‌ها

برای اتصال یک کنترل به یک منبع داده‌ای، باید از خاصیت DataBindings آن کنترل استفاده کنیم. این خاصیت از کلاس DataBindingsCollection است و خود نیز دارای چندین خاصیت و متد مختلف است. اما در این قسمت از متد Add آن استفاده خواهیم کرد. این متد سه پارامتر دریافت کرده و به صورت زیر فراخوانی می‌شود:

```
object.DataBindings.Add(propertyName,
dataSource, dataMember);
```

این پارامترها برای موارد زیر مورد استفاده قرار می‌گیرند:

Object: نام کنترلی است که می‌خواهیم یک اتصال جدید برای آن ایجاد کنیم.

propertyName: حاوی نام خاصیتی است که می‌خواهیم در طول برنامه مقدار خود را از منبع داده‌ای دریافت کند.

dataSource: نام منبع داده‌ای است که می‌خواهیم اطلاعات مورد نیاز برای این کنترل را از آن دریافت کنیم و می‌توان شامل یک `DataTable`، `DataSet`، `DataRow` یا هر منبع داده‌ای دیگری باشد.

dataMember: مشخص کننده‌ی نام فیلدی از منبع داده‌ای است که می‌خواهیم آن را به خاصیت `propertyName` کنترل متصل کنیم.

مثالی از نحوه استفاده از متد `Add` در قطعه کد زیر آورده شده است. کد زیر خاصیت `Text` در کنترل `txtFirstName` را به فیلد `au-fname` از شی `objDataView` متصل می‌کند:

```
txtFirstName.DataBindings.Add("Text",
objDataView, "au_fname");
```

در مواقعی ممکن است بعد از ایجاد اتصال در یک کنترل بخواهید تمام اتصالات آن با منابع داده‌ای را حذف کنید. برای این کار می‌توانید از متد `Clear` در کلاس `ControlBindingsCollection` استفاده کنید. این متد تمام اتصالاتی که برای خاصیت‌های مختلف یک کنترل تعریف شده بود را حذف می‌کند. نحوه‌ی استفاده از این متد در کد زیر نشان داده شده است:

```
txtFirstName.DataBindings.Clear();
```

حال که با اشیای `BindingContext`، `ControlBindingsCollection` و نیز `CurrencyManager` آشنا شدیم. بهتر است نحوه‌ی استفاده از آنها در یک برنامه را نیز مشاهده کنیم.

مثال ۲۳-۳- اتصال کنترل‌های ساده به منبع داده‌ای

۱- با استفاده از `VS` یک پروژه با نام `BindingExample` ایجاد کنید.

۲- یک کنترل `ToolTip` به فرم اضافه کنید. این کنترل همانند کنترل‌های دیگر به قسمت پایین بخش طراحی فرم مربوط به `Form` اضافه خواهد شد.

۳- بر روی فرم برنامه کلیک کنید تا انتخاب شود. سپس با استفاده از پنجره `Properties` خاصیت‌های آن را به صورت زیر تغییر دهید :

خاصیت `FormBorderStyle` را برابر `FixedDialog` قرار دهید.

خاصیت `maximizeBox` را برابر با `False` قرار دهید.

خاصیت `MinimizeBox` را برابر با `False` قرار دهید.

خاصیت `Size` را برابر با `۳۶۰"۴۳۰` قرار دهید.

خاصیت `StartPosition` را برابر با `CenterScreen` قرار دهید.

خاصیت `Text` را برابر با `Binding Controls` قرار دهید.

۴) در این قسمت باید کنترل‌هایی را به فرم برنامه اضافه کرده و سپس خاصیت‌های مختلف آنها را تنظیم کنید تا فرم برنامه مشابه شکل ۲۳-۶ شود.

این مراحل به این دلیل طی می‌شوند تا ظاهر برنامه همانند یک برنامه‌ی واقعی شود. با این وجود، این مراحل اهمیت زیادی ندارد و در صورت لزوم می‌توانید از آنها صرف‌نظر کرده و خودتان فرمی را مشابه شکل ۲۳-۶ ایجاد کنید. البته در این صورت دقت کنید که اسامی کنترل‌ها باید مشابه آنچه باشد که در این قسمت عنوان شده است. در غیر اینصورت ممکن است در اجرای برنامه با مشکل مواجه شوید.

شکل ۲۳-۶

۵) یک کنترل GroupBox به فرم برنامه اضافه کرده و خاصیت‌های آن را به صورت زیر دهید:

- خاصیت Size را برابر با ۱۲۸؛۴۰۸ قرار دهید.
- خاصیت Location را برابر با ۸؛۸ قرار دهید.
- خاصیت Text را برابر با Authors & Titles قرار دهید.

نکته: برای نمایش علامت & در عنوان یک کنترل GroupBox باید از علامت && استفاده کنید. استفاده از یک & در عنوان باعث می‌شود که کاراکتر بعد از آن با زیر خط نمایش داده شود.

۶) (چهار کنترل Label با خاصیت‌هایی که در جدول زیر عنوان شده است را به کنترل GroupBox1 اضافه کنید:

Name	Location	Size	Text	AutoSize
Label۱	۲۶؛۸	۱۶؛۶۴	Last Name	False
Label۲	۵۰؛۸	۱۶؛۶۴	First Name	False
Label۳	۷۴؛۸	۱۶؛۵۶	Book Title	False
Label۴	۹۸؛۸	۱۶؛۶۴	Price	False

۷) با استفاده از جعبه ابزار چهار کنترل TextBox نیز به GroupBox1 در برنامه اضافه کرده و خاصیت‌های آن را بر اساس جدول زیر تنظیم کنید:

Name	Location	Size	ReadOnly
txtLastName	۲۴؛۷۲	۲۰؛۸۸	True
txtFirstName	۴۸؛۷۲	۲۰؛۸۸	True
txtBookTitle	۷۲؛۷۲	۲۰؛۳۲۸	False
txtPrice	۹۶؛۷۲	۲۰؛۴۸	False

۸) با استفاده از جعبه ابزار کنترل GroupBox دیگری به فرم اضافه کرده و خاصیت‌های آن را طبق لیست زیر تنظیم کنید:

خاصیت Location را برابر با ۱۴۴;۸ قرار دهید.

خاصیت Size را برابر با ۱۶۸;۴۰۸ قرار دهید.

خاصیت Text را برابر با Navigation قرار دهید.

۹) دو کنترل Label به GroupBox ۲ اضافه کرده و بر اساس جدول زیر آنها را تنظیم کنید:

Name	Location	Size	Text	AutoSize
Label۵	۲۳;۸	۱۶;۶۴	Field	False
Label۶	۴۸;۸	۱۶;۸۰	SearchCriteria	False

۱۰) با استفاده از جعبه ابزار یک کنترل ComboBox به GroupBox ۲ اضافه کنید. خاصیت Name آن را برابر با CboField، خاصیت Location را برابر با ۲۱;۸۸، خاصیت Size را برابر ۲۱;۸۸ و خاصیت DropDownStyle را برابر با DropDownList قرار دهید.

۱۱) دو کنترل TextBox به GroupBox ۲ اضافه کرده و خاصیت‌های آن را براساس جدول زیر تغییر دهید:

Name	Location	Size	TabStop	TextAlign
txtSearchCriteria	۴۸;۸۸	۲۰;۲۰۰	-	-
txtRecordPosition	۱۳۰;۱۵۲	۲۰;۸۵	False	Center

۱۲) ده کنترل Button به GroupBox ۲ اضافه کرده و خاصیت‌های آنها را به صورت زیر تغییر دهید:

Name	Location	Size	Text	ToolTip On ToolTip۱
btnPerformSort	۱۶;۳۰۴	۲۴;۹۶	Perform Sort	-
btnPerformSearch	۴۸;۳۰۴	۲۴;۹۶	Perform Search	-
btnNew	۸۸;۴۰	۲۴;۷۲	New	-
btnAdd	۸۸;۱۲۰	۲۴;۷۲	Add	-
btnUpdate	۸۸;۲۰۰	۲۴;۷۲	Update	-
btnDelete	۸۸;۲۸۰	۲۴;۷۲	Delete	-
btnMoveFirst	۱۲۸;۸۸	۲۴;۲۹	>	Move First
btnMovePrevious	۱۲۸;۱۲۰	۲۴;۲۹	>	Move Previous
btnMoveNext	۱۲۸;۲۰۰	۲۴;۲۹	<	Move Next
btnMoveLast	۱۲۸;۲۷۲	۲۴;۲۹	<	Move Last

۱۳) در آخر نیز یک کنترل StatusStrip به برنامه اضافه کنید. نیازی به تغییر خاصیت‌های Name، Location و یا Size این کنترل نیست. بعد از انتخاب این کنترل، با استفاده از منوی کنار آن یک کنترل StatusLabel را به آن اضافه کنید.

۱۴) بعد از اتمام این مراحل، فرم کامل شده‌ی برنامه باید مشابه شکل ۲۳-۶ باشد.

۱۵) حال قسمت کدنویسی برنامه را شروع می‌کنیم. به قسمت ویرایشگر کد مربوط به کلاس Form1 رفته و با قرار دادن کد زیر در بالای کدها، فضای نام System.Data و System.Data.SqlClient را به برنامه اضافه کنید:

```
// Import Data and SqlClient namespaces
using System.Data;
using System.Data.SqlClient;
```

۱۶) سپس اشیایی که باید به صورت سراسری در برنامه وجود داشته باشند را در ابتدای کلاس تعریف کنیم. همچنین یک رشته‌ی ثابت تعریف کرده و دستور SQL ای که می‌خواهیم در طول برنامه به کار ببریم را در آن قرار می‌دهیم. بنابراین کد زیر را به ابتدای کلاس Form1 اضافه کنید:

```
public partial class Form1 : Form
{
    // Constant strings
    private const string _CommandText =
        "SELECT authors.au_id, au_lname, au_fname, " +
        "titles.title_id, title, price " +
        "FROM authors " +
        "JOIN titleauthor ON authors.au_id = " +
        "titleauthor.au_id " +
        "JOIN titles ON titleauthor.title_id = " +
        "titles.title_id " +
        "ORDER BY au_lname, au_fname";
    private const string _ConnectionString =
        "server=localhost;database=pubs;" +
        "user id=sa;password=";
    // Declare global objects...
    SqlConnection objConnection;
    SqlDataAdapter objDataAdapter;
    DataSet objDataSet;
    DataView objDataView;
    CurrencyManager objCurrencyManager;
```

نکته: قبل از وارد کردن قطعه کد بالا در برنامه، ConnectionString را براساس تنظیمات سرور بانک اطلاعاتی خود تغییر دهید. ID-User Password مربوط به حساب کاربری خود را وارد کرده و همچنین اگر سرور روی کامپیوتر دیگری قرار دارد، به جای استفاده از localhost نام کامپیوتر سرور در شبکه را وارد کنید.

۱۷) کد درون متد سازنده‌ی فرم را به صورت زیر تغییر دهید:

```
public Form1()
{
    objConnection = new SqlConnection(_ConnectionString);
    objDataAdapter = new SqlDataAdapter(_CommandText, objConnection);
    InitializeComponent();
}
```

۱۸) اولین زیربرنامه‌ای که باید ایجاد کنیم، زیربرنامه‌ای به نام FillDataSetAndView است. این زیربرنامه به همراه چند زیربرنامه‌ی دیگر در ابتدای برنامه فراخوانی می‌شوند. کد زیر را به Form1، بعد از تعریف متغیرها اضافه کنید:

```
private void FillDataSetAndView()
{
    // Initialize a new instance of the DataSet object...
    objDataSet = new DataSet();
    // Fill the DataSet object with data...
    objDataAdapter.Fill(objDataSet, "authors");
    // Set the DataView object to the DataSet object...
    objDataView = new DataView(
        objDataSet.Tables["authors"]);
    // Set our CurrencyManager object
    // to the DataView object...
    objCurrencyManager = (CurrencyManager)(this.BindingContext[objDataView]);
}
```

۱۹) در این قسمت باید زیر برنامه‌ای به فرم اضافه کنیم تا کنترل‌های موجود در فرم را به فیلدهای مربوط به آنها در DataView اضافه کند:

```
private void BindFields()
{
    // Clear any previous bindings...
    txtLastName.DataBindings.Clear();
    txtFirstName.DataBindings.Clear();
    txtBookTitle.DataBindings.Clear();
    txtPrice.DataBindings.Clear();
    // Add new bindings to the DataView object...
    txtLastName.DataBindings.Add("Text",
    objDataView, "au_lname");
    txtFirstName.DataBindings.Add("Text",
    objDataView, "au_fname");
    txtBookTitle.DataBindings.Add("Text",
    objDataView, "title");
    txtPrice.DataBindings.Add("Text",
    objDataView, "price");
    // Display a ready status...
    ToolStripStatusLabel1.Text = "Ready";
}
```

۲۰) سپس زیر برنامه‌ای به کلاس اضافه می‌کنیم که موقعیت رکورد جاری را در فرم برنامه نمایش دهد:

```
private void ShowPosition()
{
    // Always format the number
    // in the txtPrice field to include cents
    try
    {
        txtPrice.Text =
        Decimal.Parse(txtPrice.Text).ToString("##0.00");
    }
    catch (System.Exception e)
    {
        txtPrice.Text = "0";
        txtPrice.Text =
        Decimal.Parse(txtPrice.Text).ToString("##0.00");
    }
    // Display the current position
    // and the number of records
    txtRecordPosition.Text =
    (objCurrencyManager.Position + 1) +
    " of " + objCurrencyManager.Count;
}
```

۲۱) تا اینجا زیر برنامه‌های لازم را به برنامه اضافه کرده‌ایم، اما در هیچ قسمت از کد از این زیر برنامه‌های ایجاد شده استفاده نکرده‌ایم. این زیر برنامه‌ها لازم است که قبل از نمایش داده شدن فرم و هنگام بارگذاری آن فراخوانی شوند. بنابراین به قسمت طراحی فرم بروید و روی قسمت خالی از فرم دو بار کلیک کنید تا متد مربوط به رویداد Load فرم ایجاد شود (دقت کنید که باید در یک قسمت خالی از فرم دو بار کلیک کنید، نه در قسمتی خالی از کنترل GroupBox). سپس کد مشخص شده در زیر را به این متد اضافه کنید:

```
private void Form1_Load(object sender, EventArgs e)
{
    // Add items to the combo box...
    cboField.Items.Add("Last Name");
    cboField.Items.Add("First Name");
    cboField.Items.Add("Book Title");
    cboField.Items.Add("Price");
    // Make the first item selected...
    cboField.SelectedIndex = 0;
    // Fill the DataSet and bind the fields...
    FillDataSetAndView();
    BindFields();
    // Show the current record position...
    ShowPosition();
}
```

۲۲) حال باید کد کلیدهای مربوط به حرکت بین رکوردها را در برنامه وارد کنیم. برای این کار لازم است که چهار بار به قسمت طراحی فرم بروید و روی هر کدام از دکمه های `btnMoveLast` `btnMovePrevious` `btnMoveNext` `btnMoveFirst` کلیک کنید تا متد مربوط به رویداد کلیک هر یک از آنها ایجاد شود. کد مشخص شده در زیر را به متد مربوط به رویداد `Click` کنترل `btnMoveFirst` اضافه کنید:

```
private void btnMoveFirst_Click(object sender, EventArgs e)
{
    // Set the record position to the first record...
    objCurrencyManager.Position = 0;
    // Show the current record position...
    ShowPosition();
}
```

۲۳) کد زیر را به متد مربوط به رویداد `Click` کنترل `btnMovePrevious` اضافه کنید:

```
private void btnMovePrevious_Click(object sender,
    EventArgs e)
{
    // Move to the previous record...
    objCurrencyManager.Position -= 1;
    // Show the current record position...
    ShowPosition();
}
```

۲۴) کد زیر را به متد `btnMoveNext_Click` اضافه کنید:

```
private void btnMoveNext_Click(object sender, EventArgs e)
{
    // Move to the next record...
    objCurrencyManager.Position += 1;
    // Show the current record position...
    ShowPosition();
}
```

۲۵) در آخر نیز برای تکمیل این قسمت لازم است که کد زیر را در متد `btnMoveLast_Click` قرار دهید:

```
private void btnMoveLast_Click(object sender, EventArgs e)
{
    // Set the record position to the last record...
    objCurrencyManager.Position =
    objCurrencyManager.Count - 1;
    // Show the current record position...
    ShowPosition();
}
```

۲۶) تا این قسمت کد زیادی را در برنامه وارد کرده ایم و احتمالاً مشتاق هستید که نتیجه ی آن را مشاهده کنید. برنامه را اجرا کنید. مشاهده خواهید کرد که کنترل های فرم هر یک به فیلد مربوط به خود در `DataView` متصل شده اند. روی کلیدهای مربوط به ابتدا و یا انتهای رکوردها کلیک کنید تا نحوه ی عملکرد کلاس `CurrencyManager` برای ایجاد هماهنگی بین رکوردی که کنترل ها نمایش می دهند را مشاهده کنید.

با اجرای برنامه باید فرمی را مشابه شکل ۲۳-۷ مشاهده کنید. تا اینجا در فرم برنامه فقط کلیدهای مربوط به جابه جا شدن بین رکوردها عمل می کنند. با کلیک کردن روی هر یک از کلیدهای بعدی و قبلی و یا روی کلیدهای مربوط به ابتدا و انتها بین رکوردهای موجود در فرم جابه جا شوید. مشاهده خواهید کرد هر بار که با استفاده از این کلیدها رکورد جاری را تغییر دهید، عدد نمایش داده شده در کادر در فرم برنامه نیز تغییر کرده و شماره رکورد جاری را نمایش می دهد.

شکل ۲۳-۷

اگر در رکورد اول باشید، می‌توانید روی کلید مربوط به رکورد قبلی کلیک کنید، اما هیچ اتفاقی رخ نخواهد داد، زیرا هم اکنون در رکورد قبلی هستید. همچنین می‌توانید به آخرین رکورد بروید و روی کلید مربوط به رکورد بعدی کلیک کنید، اما باز هم هیچ تغییری را مشاهده نخواهیم کرد، زیرا در آخرین رکورد هستید.

همچنین اگر ماوس را روی هر یک از این دکمه‌ها ببرید، توضیحی را مشاهده خواهیم کرد که عملکرد کلید را توضیح می‌دهد. این مورد فقط برای ایجاد رابط کاربری بهتر اضافه شده است.

نکته: قسمت‌های مربوط به مدیریت خطاها و استثنای احتمالی از کد این قسمت حذف شده‌اند تا جای کمتری گرفته شود. در هنگام وارد کردن این کد بهتر است که این قسمت را نیز اضافه کنید.

مثال ۲۳-۴- اضافه کردن قابلیت مرتب‌سازی به برنامه

۱) به قسمت طراحی فرم مربوط به Form1 بروید و روی دکمه‌ی Perform Sort دو بار کلیک کنید تا متد مربوط به رویداد Click این کنترل ایجاد شود. سپس کد مشخص شده در زیر را به این متد اضافه کنید:

```
private void btnPerformSort_Click(object sender,
EventArgs e)
{
    // Determine the appropriate item selected and set the
    // Sort property of the DataView object...
    switch(cboField.SelectedIndex)
    {
        case 0: // Last Name
            objDataView.Sort = "au_lname";
            break;
        case 1: // First Name
            objDataView.Sort = "au_fname";
            break;
        case 2: // Book Title
            objDataView.Sort = "title";
            break;
        case 3: // Price
            objDataView.Sort = "price";
            break;
    }
    // Call the click event for the MoveFirst button...
    btnMoveFirst_Click(null, null);
    // Display a message
    // that the records have been sorted...
    ToolStripStatusLabel1.Text = "Records Sorted";
}
```


۲) برنامه را اجرا کنید تا قابلیت را که در این قسمت به برنامه اضافه کردیم، ببینید. در کنترل ComboBox موجود در فرم یک ستون را انتخاب کرده و سپس روی دکمه Perform Sort کلیک کنید تا داده‌ها براساس آن ستون مرتب شود. شکل ۲۳-۸ فرم برنامه را در حالتی نمایش می‌دهد که داده‌های موجود در آن براساس ستون Price مرتب شده‌اند

شکل ۲۳-۸

مثال ۲۳-۵- اضافه کردن قابلیت جستجو به برنامه

۱) به قسمت طراحی فرم بروید و روی دکمه‌ی Perform Search دو بار کلیک کنید تا متد مربوط به رویداد Click آن ایجاد شود. سپس کد مشخص شده در زیر را در این متد وارد کنید:

```
private void btnPerformSearch_Click(object sender,
EventArgs e)
{
    // Declare local variables...
    int intPosition;
    // Determine the appropriate item selected and set the
    // Sort property of the DataView object...
    switch(cboField.SelectedIndex)
    {
        case 0: // Last Name
            objDataView.Sort = "au_lname";
            break;
        case 1: // First Name
            objDataView.Sort = "au_fname";
            break;
        case 2: // Book Title
            objDataView.Sort = "title";
            break;
        case 3: // Price
            objDataView.Sort = "price";
            break;
    }
    // If the search field is not price then...
    if (cboField.SelectedIndex < 3)
    {
        // Find the last name, first name, or title...
        intPosition = objDataView.Find(txtSearchCriteria.Text);
    }
    else
    {
        // otherwise find the price...
        intPosition = objDataView.Find(
```

```

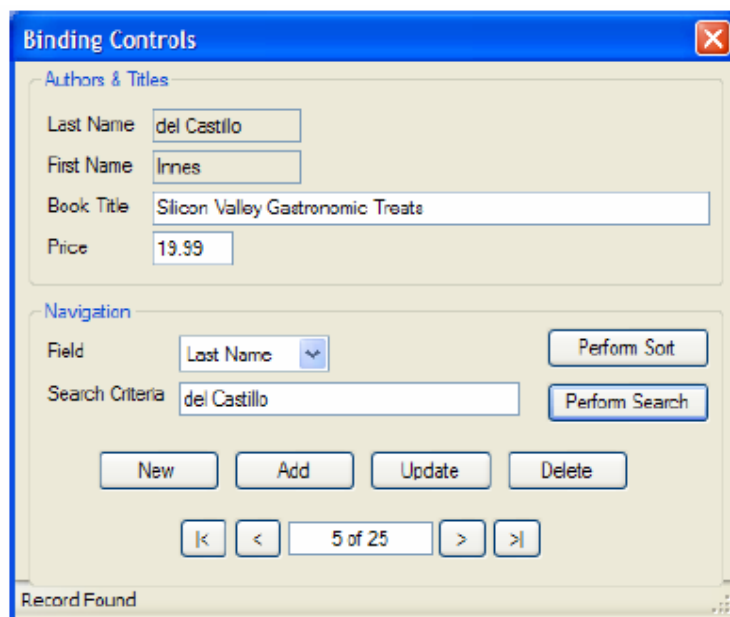
Decimal.Parse(txtSearchCriteria.Text));
}
if (intPosition == -1)
{
    // Display a message
    // that the record was not found...
    ToolStripStatusLabel1.Text = "Record Not Found";
}
else
{
    // Otherwise display a message that the record
    // was found and reposition the CurrencyManager
    // to that record...
    ToolStripStatusLabel1.Text = "Record Found";
    objCurrencyManager.Position = intPosition;
}
// Show the current record position...
ShowPosition();
}

```

۲) برنامه را اجرا کنید تا قابلیت جدید آن را نیز امتحان کنیم. فیلدی که می‌خواهید جستجو براساس آن صورت گیرد را از داخل ComboBox انتخاب کرده و سپس عبارت مورد جستجو را در داخل فیلد Search Criteria وارد کنید. در آخر نیز روی دکمه Perform Search کلیک کنید.

اگر رکورد مورد نظر شما در بین داده‌ها پیدا شود مشاهده خواهید کرد که اطلاعات آن رکورد در فرم نمایش داده می‌شود و موقعیت رکورد جاری به رکورد پیدا شده تغییر می‌کند. همچنین پیغامی در نوار وضعیت نوشته می‌شود و مشخص می‌کند که رکورد مورد نظر پیدا شده است (شکل ۲۳-۹). همچنین اگر هیچ رکوردی پیدا نشود، متنی در نوار وضعیت نوشته می‌شود و مشخص می‌کند که داده‌ی مورد نظر پیدا نشده است.

شکل ۲۳-۹



مثال ۲۳-۶- اضافه کردن رکورد جدید

۱) ابتدا به قسمت طراحی فرم Form1 بروید و روی دکمه‌ی btnNew دو بار کلیک کنید تا متد مربوط به رویداد click آن ایجاد شود. سپس کد مشخص شده در زیر را در این متد وارد کنید:

```

private void btnNew_Click(object sender, EventArgs e)
{
    // Clear the book title and price fields...
    txtBookTitle.Text = "";
    txtPrice.Text = "";
}

```

۲) حال باید کد مربوط به متد `btnAdd_Click` را وارد کنیم. این متد مسئول اضافه کردن یک رکورد داده‌ای جدید به برنامه است. این زیربرنامه، طولانی‌ترین زیربرنامه‌ای است که در این پروژه وجود دارد و کد زیادی را شامل می‌شود دلیل آن نیز رابطه‌ی بین عنوان کتاب‌ها و نیز نویسندگان آنها و نیز کلید اصلی که برای عنوان کتاب‌ها استفاده می‌شود است. به قسمت طراحی فرم بروید و روی دکمه `Add` دو بار کلیک کنید تا متد مربوط به رویداد `Click` این کنترل ایجاد شود. سپس کد زیر را در این متد وارد کنید:

```
private void btnAdd_Click(object sender, EventArgs e)
{
    // Declare local variables and objects...
    int intPosition, intMaxID;
    String strID;
    SqlCommand objCommand = new SqlCommand();
    // Save the current record position...
    intPosition = objCurrencyManager.Position;
    // Create a new SqlCommand object...
    SqlCommand maxIdCommand = new SqlCommand(
        "SELECT MAX(title_id)" +
        "FROM titles WHERE title_id LIKE 'DM%'",
        objConnection);
    // Open the connection, execute the command
    objConnection.Open();
    Object maxId = maxIdCommand.ExecuteScalar();
    // If the MaxID column is null...
    if (maxId == DBNull.Value)
    {
        // Set a default value of 1000...
        intMaxID = 1000;
    }
    else
    {
        // otherwise set the strID variable
        // to the value in MaxID...
        strID = (String)maxId;
        // Get the integer part of the string...
        intMaxID = int.Parse(strID.Remove(0, 2));
        // Increment the value...
        intMaxID += 1;
    }
    // Finally, set the new ID...
    strID = "DM" + intMaxID.ToString();
    // Set the SqlCommand object properties...
    objCommand.Connection = objConnection;
    objCommand.CommandText = "INSERT INTO titles " +
        "(title_id, title, type, price, pubdate) " +
        "VALUES(@title_id,@title,@type,@price,@pubdate);" +
        "INSERT INTO titleauthor (au_id, title_id) " +
        "VALUES(@au_id,@title_id)";
    // Add parameters for the placeholders in the SQL in
    // the CommandText property...
    // Parameter for the title_id column...
    objCommand.Parameters.AddWithValue("@title_id",
        strID);
    // Parameter for the title column...
    objCommand.Parameters.AddWithValue("@title",
        txtBookTitle.Text);
    // Parameter for the type column
    objCommand.Parameters.AddWithValue("@type", "Demo");
    // Parameter for the price column...
    objCommand.Parameters.AddWithValue("@price",
        txtPrice.Text).DbType = DbType.Currency;
    // Parameter for the pubdate column
    objCommand.Parameters.AddWithValue("@pubdate",
        DateTime.Now);
    // Parameter for the au_id column...
    objCommand.Parameters.AddWithValue("@au_id",
        this.BindingContext[objDataView, "au_id"].Current);
    // Execute the SqlCommand object
    // to insert the new data...
```

```

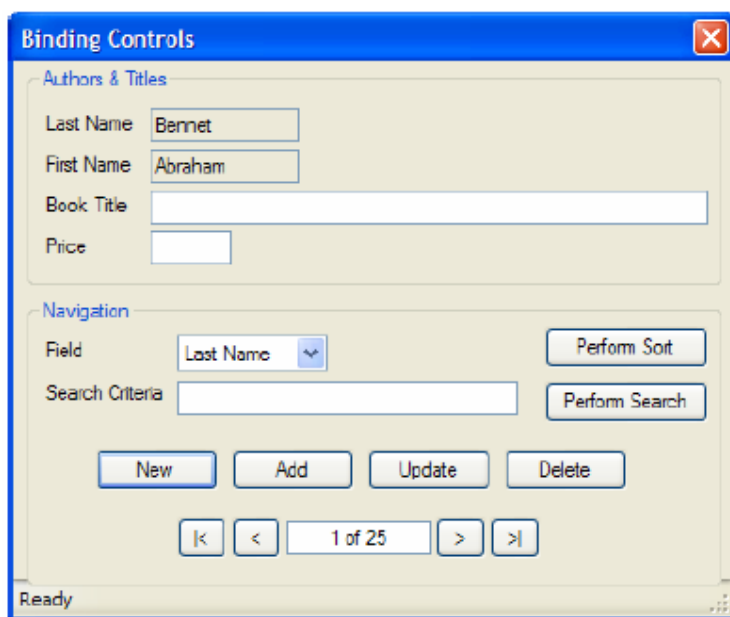
try
{
objCommand.ExecuteNonQuery();
}
catch(SqlException SqlExceptionErr)
{
MessageBox.Show(SqlExceptionErr.Message);
}
// Close the connection...
objConnection.Close();
// Fill the dataset and bind the fields...
FillDataSetAndView();
BindFields();
// Set the record position
// to the one that you saved...
objCurrencyManager.Position = intPosition;
// Show the current record position...
ShowPosition();
// Display a message that the record was added...
ToolStripStatusLabel1.Text = "Record Added";
}

```

متد ExecuteScalar دستورات SQL ای را اجرا می‌کند که فقط یک مقدار اسکالر برمی‌گردانند.

۳) برنامه را اجرا کرده و کاربری را که می‌خواهید عنوان کتاب جدیدی را برای او ثبت کنید، انتخاب کنید، سپس روی دکمه‌ی Add کلیک کنید. به این ترتیب کادرهای Price, BookTitle خالی خواهند شد و می‌توان داده‌های مربوط به کتاب جدید را همانند شکل ۲۳-۱۰ وارد کنید. در برنامه به تعداد رکوردهایی که هم اکنون وجود دارند توجه کنید.

شکل ۲۳-۱۰



۴) حال نام کتاب و قسمت آن را فیلدهای مربوطه وارد کرده و روی دکمه‌ی Add کلیک کنید. به این ترتیب پیغامی در نوار وضعیت نمایش داده می‌شود و بیان می‌کند که رکورد جدید با موفقیت اضافه شده است. همچنین همانطور که در شکل مشخص است تعداد رکوردها در برنامه یک واحد افزایش پیدا می‌کند.

شکل ۲۳-۱۱

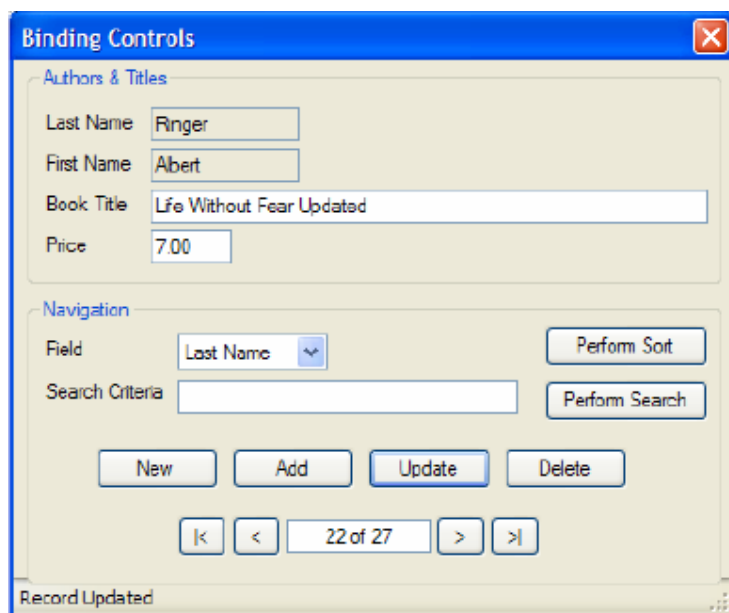
مثال ۲۳-۷- ویرایش داده‌ها

(۱) به قسمت طراحی فرم و روی دکمه‌ی `btnUpdate` دو بار کلیک کنید تا متد مربوط به رویداد `click` آن ایجاد شود. سپس کد مشخص شده در زیر را در این متد وارد کنید:

```
private void btnUpdate_Click(object sender, EventArgs e)
{
    // Declare local variables and objects...
    int intPosition;
    SqlCommand objCommand = new SqlCommand();
    // Save the current record position...
    intPosition = objCurrencyManager.Position;
    // Set the SqlCommand object properties...
    objCommand.Connection = objConnection;
    objCommand.CommandText = "UPDATE titles " +
        "SET title = @title, price = @price " +
        "WHERE title_id = @title_id";
    objCommand.CommandType = CommandType.Text;
    // Add parameters for the placeholders in the SQL in
    // the CommandText property...
    // Parameter for the title field...
    objCommand.Parameters.AddWithValue("@title",
    txtBookTitle.Text);
    // Parameter for the price field...
    objCommand.Parameters.AddWithValue("@price",
    txtPrice.Text).DbType = DbType.Currency;
    // Parameter for the title_id field...
    objCommand.Parameters.AddWithValue("@title_id",
    this.BindingContext[objDataView, "title_id"].Current);
    // Open the connection...
    objConnection.Open();
    // Execute the SqlCommand object to update the data...
    objCommand.ExecuteNonQuery();
    // Close the connection...
    objConnection.Close();
    // Fill the DataSet and bind the fields...
    FillDataSetAndView();
    BindFields();
    // Set the record position
    // to the one that you saved...
    objCurrencyManager.Position = intPosition;
    // Show the current record position...
    ShowPosition();
    // Display a message that the record was updated...
    ToolStripStatusLabel1.Text = "Record Updated";
}
```

۲) برنامه را اجرا کنید. حال می‌توانید اطلاعات مربوط به کتابی که اضافه کرده بودید را تغییر دهید و یا تغییراتی را در اطلاعات مربوط به دیگر کتاب‌ها ایجاد کنید. یک کتاب را انتخاب کرده و با استفاده از کادر Price قیمت آن را تغییر دهید. سپس روی دکمه‌ی Update کلیک کنید. به این ترتیب تغییرات مورد نظر شما در بانک اطلاعاتی ذخیره می‌شود و پیغامی نیز در نوار وضعیت نمایش داده می‌شود و ثبت تغییرات را اعلام می‌کند.

شکل ۲۳-۱۲



مثال ۲۳-۸- حذف کردن یک رکورد

۱) به قسمت طراحی فرم رفته و روی دکمه‌ی btnDelete دو بار کلیک کنید تا متد مربوط به رویداد click آن ایجاد شود. سپس کد مشخص شده در زیر را در این متد وارد کنید:

```
private void btnDelete_Click(object sender, EventArgs e)
{
    // Declare local variables and objects...
    int intPosition;
    SqlCommand objCommand = new SqlCommand();
    // Save the current record position - 1 for the one to
    // be deleted...
    intPosition = this.BindingContext[objDataView].Position - 1;
    // If the position is less than 0 set it to 0...
    if (intPosition < 0)
        intPosition = 0;
    // Set the Command object properties...
    objCommand.Connection = objConnection;
    objCommand.CommandText = "DELETE FROM titleauthor " +
        "WHERE title_id = @title_id;" +
        "DELETE FROM titles WHERE title_id = @title_id";
    // Parameter for the title_id field...
    objCommand.Parameters.AddWithValue("@title_id",
        this.BindingContext[objDataView, "title_id"].Current);
    // Open the database connection...
    objConnection.Open();
    // Execute the SqlCommand object to update the data...
    objCommand.ExecuteNonQuery();
    // Close the connection...
    objConnection.Close();
    // Fill the DataSet and bind the fields...
    FillDataSetAndView();
    BindFields();
    // Set the record position
    // to the one that you saved...
```

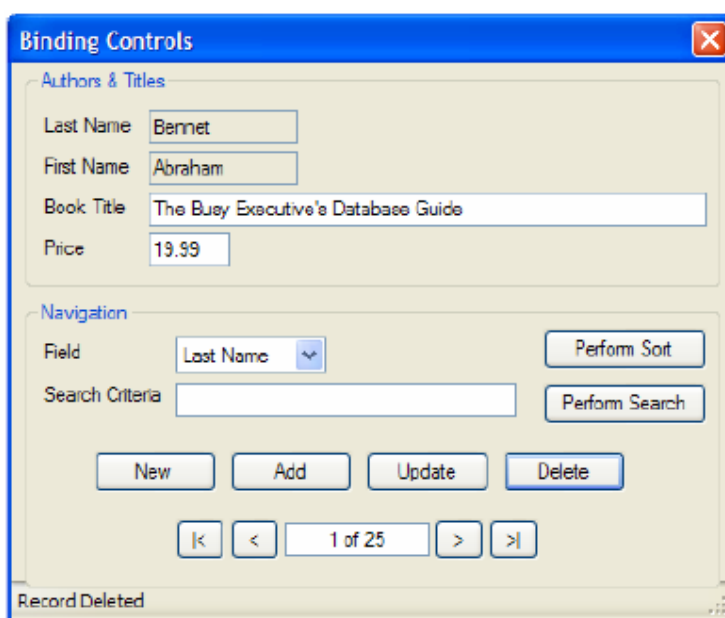
```

this.BindingContext[objDataView].Position =
intPosition;
// Show the current record position...
ShowPosition();
// Display a message that the record was deleted...
ToolStripStatusLabel1.Text = "Record Deleted";
}

```

۲) خوب به این ترتیب این پروژه نیز به پایان رسید. اما بهتر است قبل از اینکه از تمام شدن آن خوشحال شویم ابتدا قابلیت جدیدی که اضافه کرده‌ایم را امتحان کنیم. برنامه را اجرا کرده و هر کتابی که می‌خواهید حذف کنید را انتخاب کنید، سپس روی دکمه‌ی Delete کلیک کنید. به خاطر داشته باشید که بانک اطلاعاتی pubs که در این برنامه از آن استفاده کرده‌ایم یک بانک اطلاعاتی نمونه است و ممکن است افراد دیگری نیز به این SQL Server متصل شوند و بخواهند از آن برای تمرینات خود استفاده کنند. بنابراین بهتر است داده‌هایی را حذف کنیم که در قسمت قبل ایجاد کرده بودیم. قبل از اینکه یک کتاب را حذف کنید، به شماره‌ی رکوردها در برنامه توجه کنید، سپس کتاب مورد نظر خود را حذف کنید. البته در این قسمت ممکن است با خطا مواجه شوید، زیرا کتابی که برای حذف انتخاب کرده‌اید ممکن است با داده‌های جدول sales در بانک اطلاعاتی رابطه داشته باشند. بنابراین باید کتاب دیگری را انتخاب کرده و حذف کنید.

شکل ۲۳-۱۳



۲۳-۵- خلاصه

در این فصل با بعضی از کلاس‌های مهم ADO.NET از قبیل SqlConnection, SqlCommand, SqlDataAdapter و SqlParameter آشنا شدید و مشاهده کردیم که این کلاس‌ها چگونه می‌توانند هنگام دریافت اطلاعات، وارد کردن اطلاعات جدید، حذف اطلاعات جاری و یا ویرایش آنها کمک کنند. البته تمام این کلاس‌ها برای بانک‌های اطلاعاتی مورد استفاده قرار می‌گیرند که بوسیله‌ی موتور بانک اطلاعاتی SQL Server ایجاد شده باشند. این کلاس‌ها دارای کلاس‌های متنطری هستند که با پیشوند OleDb شروع می‌شوند و در فضای نامی System.Data.OleDb قرار دارند. همچنین در این فصل کلاس‌های DataSet و DataView از فضای نامی System.Data را بررسی کرده و نحوه‌ی استفاده از آنها در یک برنامه را مشاهده کردیم و دیدیم که چگونه می‌توان با استفاده از این کلاس‌ها اشیایی را ایجاد کرد و داده‌های موجود در آنها را به کنترل‌های ساده‌ی موجود در فرم متصل کرد.

نحوه‌ی استفاده از CurrencyManager را برای جابه‌جایی مابین رکوردها دیدید. در این فصل برای دسترسی به داده‌ها، ایجاد داده‌های جدید، حذف داده‌های موجود و یا ویرایش آنها از روش‌های دستی استفاده کرده و کد مربوط به تمام این موارد را خودمان در برنامه وارد کردیم.

فصل بیست و چهارم

ریسمان‌ها و همگام‌سازی

آنچه که در این فصل یاد خواهید گرفت:

- آشنایی با برنامه‌های چندوظیفه‌گی و نحوه‌ی برنامه‌نویسی آنها
- آشنایی با برنامه‌نویسی چندریسمانی
- همگام‌سازی ریسمان‌ها
- قفل کردن منبع مختلف برای جلوگیری از دسترسی مخرب ریسمان‌های همگام
- کار با مانیتورها

ریسمان‌ها مسئول چند وظیفه‌گی یک برنامه‌ی کاربردی واحد هستند. فضای نامی `System.Threading` یک مجموعه از کلاس‌ها و واسط‌ها را برای مدیریت برنامه‌نویسی چند ریسمانی فراهم می‌کند. احتمالاً اکثر برنامه‌نویس‌ها هرگز نیازی به مدیریت صریح ریسمان‌ها نداشتند، چون CLR پشتیبانی ریسمان‌بندی را در داخل کلاس‌ها بصورت انتزاعی درمی‌آورد تا چندریسمانی را ساده سازد.

بخش اول این فصل نحوه‌ی ایجاد، مدیریت و از بین بردن^۱ ریسمان‌ها را نشان می‌دهد. حتی اگر ریسمان‌های خود را صریحاً ایجاد نکرده باشند، می‌خواهید مطمئن باشید که کد شما می‌تواند چند ریسمانی را اداره کند. اگر در یک محیط چندریسمانی^۲ اجرا شود یا در قطعاتی که ممکن است توسط برنامه‌نویسان دیگر در برنامه‌های چندریسمانی استفاده شوند، باید این ایده را در نظر گرفت. این عمل برای توسعه‌دهندگان سرویس‌های وب و کنترل از راه دور مهم است. اگرچه برنامه‌های تحت وب اکثر ویژگی‌های برنامه‌های رومیزی را دارند، آنها روی یک سرور اجرا می‌شوند و بایستی ایده‌های بهره‌وری و چندریسمانی برای آنها در نظر گرفته شود.

بخش دوم فصل روی همگام‌سازی تمرکز دارد. زمانی که یک منبع محدود (مانند یک اتصال پایگاه داده) دارید، ممکن است محدود کردن دسترسی به آن منبع در یک لحظه توسط یک ریسمان لازم باشد.

یک شباهت کلاسیک، توالیت هواپیما است. می‌خواهید دسترسی به توالیت را در هر لحظه فقط برای یک نفر مجاز دارید. این عمل با یک قفل روی در انجام می‌شود. زمانی که مسافر می‌خواهد توالیت را استفاده کند، آنها سعی می‌کنند در را کنترل

^۱ Kill

^۲ MultiThread

کنند. اگر در قفل شده باشد، آنها دنبال کار دیگری می‌روند یا در یک صف منتظر می‌مانند. زمانی که منبع آزاد می‌شود، یک فرد دیگر از صف خارج شده و منبع را می‌گیرد و آن را قفل می‌کند.

گاهی اوقات، ممکن است ریسمان‌های متعددی بخواهند به یک منبع در برنامه‌ی شما دسترسی داشته باشند (همچون یک فایل). شاید مهم باشد مطمئن شوید که در هر لحظه فقط یک ریسمان به منبع دسترسی دارد و منبع خود را قفل کرده و بعد از دستیابی یک ریسمان به آن، منبع را آزاد کنید. قفل‌های برنامه‌نویسی با توجه به توزیع منابع می‌توانند بسیار پیچیده باشند.

۲۴-۱-ریسمان‌ها

در صورتی که یک برنامه بخواهد در یک لحظه دو کار بطور همزمان انجام دهد، ریسمان‌ها استفاده می‌شوند. برای مثال، می‌خواهید π را با ۱۰ بلیون رقم اعشار حساب کنید. بعد از شروع این محاسبه، تا زمانی که آن کار می‌کند، برنامه هیچ خروجی به واسط کاربر نمی‌نویسد. چون این محاسبه چند میلیون سال طول می‌کشد. حال ممکن است بخواهید مقدار جدید حاصل را پردازنده در اختیار شما قرار دهد. علاوه بر این، ممکن است بخواهید یک دکمه توقف قرار دهید تا کاربر در هر لحظه بتواند آن را متوقف سازد. در صورتی که برنامه شما اجازه دارد کلیک روی دکمه توقف را اداره کند، شما ریسمان اجرایی دیگری لازم دارید.

کاربرد معمول دیگر از ریسمان‌بندی، در صورتی است که باید منتظر یک رویداد بمانید: همچون ورودی کاربر، خواندن از فایل یا دریافت داده از شبکه. در حالیکه پروسه‌ی شما منتظر است، آزاد کردن پردازنده برای شروع اجرای عمل دیگر، یک ایده خوب است و آن اجرای برنامه شما را سریع‌تر می‌سازد.

از طرف دیگر، به بعضی از شرایط توجه کنید. ریسمان‌بندی می‌تواند سرعت اجرای برنامه را پایین بیاورد. فرض کنید می‌خواهید علاوه بر محاسبه π ، سری فیبوناچی را نیز بدست آورید. اگر یک چند پردازنده داشته باشید، حتی اگر هر کدام ریسمان خود را داشته باشند، سرعت بالاست. ولی اگر از یک پردازنده استفاده کنیم، محاسبه‌ی این مقادیر در چند ریسمان، قطعاً با سرعت کمتری انجام خواهد شد. چون عمل سوییچ مابین ریسمان‌ها انجام می‌شود که یک سربار است.

۲۴-۱-۱-شروع ریسمان‌ها

ساده‌ترین راه ایجاد یک ریسمان، ایجاد یک نمونه جدید از کلاس Thread است. سازنده‌ی Thread یک آرگومان از نوع نماینده می‌گیرد. CLR کلاس نماینده ThreadStart را برای این منظور تهیه کرده است، که به متد مورد نظر شما اشاره می‌کند. این کلاس به شما اجازه می‌دهد، یک ریسمان ایجاد کنید و به آن بگویید "زمانی که شروع شدی، این متد را اجرا کن". اعلان نماینده‌ی ThreadStart بصورت زیر است.

```
public delegate void ThreadStart();
```

همانطور که می‌بینید متدی که شما به این نماینده الحاق می‌کنید، نباید پارامتری داشته باشد و مقدار void برمی‌گرداند. پس یک ریسمان جدید را بصورت زیر ایجاد می‌کنید.

```
Thread myThread = new Thread( new ThreadStart(myFunc) );
```

برای مثال: می‌خواهیم دو ریسمان ایجاد کنیم که یکی از صفر به بالا می‌شمارد و دیگری از ۱۰۰۰ به پایین می‌شمارد.

```
public void Incremter()
{
    for (int i = 0; i < 1000; i++)
    {
        Console.WriteLine("Incremter: {0}", i);
    }
}
```

```
}  
public void Decrementer()  
{  
for (int i = ۱۰۰۰; i >= ۰; i--)  
{  
Console.WriteLine("Decrementer: {۰}", i);  
}  
}
```

برای اجرای این متدها در ریسمانها، دو ریسمان جدید ایجاد کنید و هرکدام را با یک نماینده‌ی ThreadStart مقداردهی اولیه کنید. این نماینده‌ها در ابتدا با توابع عضو مربوطه مقداردهی اولیه می‌شوند.

```
Thread t۱ = new Thread( new ThreadStart(Incrementer) );  
Thread t۲ = new Thread( new ThreadStart(Decrementer) );
```

با ایجاد نمونه‌هایی از ریسمان‌ها، اجرای آنها شروع نمی‌شود. برای انجام این کار، باید متد Start مربوطه به شی Thread فراخوانی شود.

```
t۱.Start();  
t۲.Start();
```

توجه: اگر کار دیگری انجام ندهد، بعد از اینکه تابع تمام شود، ریسمان‌ها متوقف می‌گردند.

مثال ۱-۲۴ یک برنامه‌ی کامل و خروجی آن را نشان می‌دهد. لازم است با یک دستور Using System.Threading کامپایلر را از وجود کلاس Thread با خبر سازید. به خروجی توجه کنید، می‌توانید ببینید که پردازنده ما بین t۲ و t۱ سوئیچ می‌کند.

مثال ۱-۲۴

```
#region Using directives  
using System;  
using System.Collections.Generic;  
using System.Text;  
using System.Threading;  
#endregion  
namespace UsingThreads  
{  
class Tester  
{  
static void Main( )  
{  
// make an instance of this class  
Tester t = new Tester( );  
Console.WriteLine( "Hello" );  
// run outside static Main  
t.DoTest( );  
}  
public void DoTest( )  
{  
// create a thread for the Incrementer  
// pass in a ThreadStart delegate  
// with the address of Incrementer  
Thread t۱ =  
new Thread(  
new ThreadStart( Incrementer ) );  
// create a thread for the Decrementer  
// pass in a ThreadStart delegate  
// with the address of Decrementer  
Thread t۲ =  
new Thread(  
new ThreadStart( Decrementer ) );
```

```
// start the threads
t1.Start( );
t2.Start( );
}
// demo function, counts up to ۱K
public void Incrementer( )
{
    for ( int i = ۰; i < ۱۰۰۰; i++ )
    {
        System.Console.WriteLine(
            "Incrementer: {۰}", i );
    }
}
// demo function, counts down from ۱k
public void Decrementer( )
{
    for ( int i = ۱۰۰۰; i >= ۰; i-- )
    {
        System.Console.WriteLine(
            "Decrementer: {۰}", i );
    }
}
} }
```

Output (excerpt):

```
Incrementer: ۱۰۲
Incrementer: ۱۰۳
Incrementer: ۱۰۴
Incrementer: ۱۰۵
Incrementer: ۱۰۶
Decrementer: ۱۰۰۰
Decrementer: ۹۹۹
Decrementer: ۹۹۸
Decrementer: ۹۹۷
```

ابتدا پردازنده به اولین ریسمان اجازه می‌دهد تا شمردن ۱۰۶ اجرا گردد. سپس ریسمان دوم مشارکت کرده و شمردن از ۱۰۰۰ به پایین را آغاز می‌کند. سپس به اولین ریسمان اجازه‌ی اجرا داده می‌شود. مدت زمان واقعی اختصاص داده شده به هر ریسمان بوسیله زمان‌بند ریسمان اداره می‌شود و به فاکتورهای زیادی بستگی است: همچون سرعت پردازنده، درخواست پردازنده از طرف برنامه‌های دیگر.

۲۰-۱-۲- پیوند زدن ریسمان‌ها

زمانی که می‌خواهید به یک ریسمان بگویید تا اجرای کامل ریسمان دیگر منتظر بماند، ریسمان اول را به ریسمان دوم پیوند زنید، یعنی سر ریسمان اول را به ته ریسمان دوم پیوند زدید.

برای پیوند زدن ریسمان ۱ (t1) به ریسمان ۲ (t2) بنویسید.

```
t2.Join( );
```

اگر این دستور در یک متد در t1 اجرا شود، t1 مکث خواهد کرد و برای تکمیل و خروج t2 منتظر خواهد ماند. برای مثال، ممکن است در بدنه‌ی متد Main()، از ریسمان بخواهد تا اتمام همه ریسمان‌های دیگر منتظر بماند قبل از اینکه آن پیام مندرج شده را بنویسد. در تکه کد بعدی، فرض کنید یک کلکسیون از ریسمان‌ها بنام MyThreads ایجاد کرده‌اید. کلکسیون را طی می‌کند و ریسمان جاری را به همه ریسمان‌های کلکسیون پیوند می‌زند.

```
foreach (Thread myThread in myThreads)
```

```
{
    myThread.Join();
}
Console.WriteLine("All my threads are done.");
```

پیام آخری تا زمانیکه همه ریسمان‌ها پایان نیافته‌اند، اجرا نخواهد شد. در یک محیط تولید، ممکن است یک دنباله از ریسمان‌ها را برای بنا کردن بعضی کارها شروع کنید و ادامه اجرای ریسمان اصلی را تا کامل شدن همه ریسمان‌ها متوقف سازید.

۲۰-۱-۳-بلوکه کردن ریسمان‌ها با Sleep

گاهی اوقات، می‌خواهید ریسمان‌تان را برای یک مدت کوتاهی معلق سازید. برای مثال ممکن است جهت تست زمان سیستم، ساعت را به مدت یک ثانیه معلق سازید. این به شما اجازه می‌دهد، زمان جدید را در حدود یک ثانیه بدون اختصاص هزاران میلیون سیکل ماشین نمایش دهید.

کلاس Thread یک متد ایستای عمومی برای این منظور پیشنهاد می‌کند. این متد overload می‌شود. یک نسخه از آن یک مقدار int و نسخه‌ی دیگر آن یک شی timespan را به عنوان ورودی می‌گیرد. هر کدام مقدار زمان معلق کردن ریسمان را با واحد میلیونم ثانیه نمایش می‌دهند. (مقدار صحیح ۲۰۰۰ یعنی ۲ ثانیه)

اگرچه اشیاء timespan می‌توانند تیک‌ها (۱۰۰ نانوثانیه) را اندازه‌گیری کنند، واحد داده‌ها در متد Sleep() میلی ثانیه است. برای اینکه ریسمان‌تان را به یک ثانیه خواب وادار کنید، می‌توانید متد ایستای Thread.Sleep() را احضار کنید که ریسمان احضار شده را معلق می‌سازد.

```
Thread.Sleep(۱۰۰۰);
```

گاهی اوقات، متد Sleep را با مقدار صفر فراخوانی می‌کنند. بدین منظور که به زمان‌بند ریسمان القاء کنند، نوبت اجرا را به ریسمانی دیگر بدهد، حتی اگر زمان‌بند مقدار بیشتری زمان به ریسمان شما داده باشد.

اگر مثال ۲۴-۱ را با اضافه کردن یک دستور (Thread.Sleep(۱)) بعد از هر دستور WriteLine تغییر دهید، خروجی تغییر می‌یابد.

```
for (int i = 0; i < ۱۰۰۰; i++)
{
    Console.WriteLine("Incrementer: {۰}", i);
    Thread.Sleep(۱);
}
```

این تغییر کوچک کافی است تا هر ریسمان بعد از چاپ یک مقدار توسط ریسمان دیگر، فرصت اجرا پیدا کند. خروجی این تغییر را منعکس می‌کند.

```
Incrementer: ۰
Incrementer: ۱
Decrementer: ۱۰۰۰
Incrementer: ۲
Decrementer: ۹۹۹
Incrementer: ۳
Decrementer: ۹۹۸
Incrementer: ۴
Decrementer: ۹۹۷
Incrementer: ۵
Decrementer: ۹۹۶
```

Incrementer: ۶
Decrementer: ۹۹۵

۲۴-۱-۴- از بین بردن ریسمان‌ها

معمولاً، ریسمان‌ها بعد از اجرای دوره‌ی خود از بین می‌روند. می‌توانید از یک ریسمان بخواهید خود را از بین ببرد. واضح‌ترین روش تنظیم فلگ بولین `KeepAlive` است که ریسمان بصورت دوره‌ای آن را بررسی می‌کند. زمانی که حالت فلگ تغییر یابد، ریسمان می‌تواند خود را متوقف سازد.

روش دیگر فراخوانی `Thread.Interrupt()` است، که از ریسمان می‌خواهد خود را از بین ببرد. نهایتاً در لاعلاجی و اگر برنامه بخواهد خود را متوقف سازد، ممکن است `Thread.Abort` را فراخوانی کنید. این عمل یک استثنای `ThreadAbortException` رها می‌کند، که ریسمان می‌تواند تشخیص دهد. ریسمان با استثنای `ThreadAbortException` بصورت یک سیگنال برخورد می‌کند تا فوراً ریسمان را از بین برد. در هر حال، شما شبیه یک سیاست خودکشی، ریسمان را نمی‌کشید.

ممکن است بخواهید در واکنش به یک رویداد همچون کلیک کاربر روی دکمه `Cancel`، ریسمان را از بین ببرید. ممکن است اداره‌کننده‌ی رویداد دکمه `Cancel` در ریسمان `t` باشد و رویدادی که لغو می‌شود در ریسمان `t` باشد. در اداره‌کننده‌ی رویدادتان می‌توانید `Abort` را روی `t` فراخوانی کنید.

```
t.Abort();
```

یک استثناء در متد جاری `t` رها می‌شود که `t` می‌تواند آنرا تشخیص دهد.

در مثال ۲۴-۲ سه ریسمان ایجاد می‌شوند و در یک آرایه از اشیاء `Thread` ذخیره می‌شوند. قبل از شروع ریسمان‌ها خصوصیات `IsBackground` آنها را `True` قرار دهید (ریسمان‌های زمینه دقیقاً شبیه ریسمان‌های پیش‌زمینه اجرا می‌شوند به استثناء اینکه آنها نمی‌توانند مانع خاتمه یافتن یک پردازش شوند). هر ریسمان نامگذاری شده و شروع می‌شود (همچون `Thread` و `Thread`). یک پیام برای نشان دادن شروع ریسمان نمایش داده می‌شود و سپس ریسمان اصلی قبل از شروع ریسمان بعدی ۵۰ میلی ثانیه خواب می‌رود.

بعد از شروع سه ریسمان، ۵۰ میلی ثانیه دیگر می‌گذرد و اولین ریسمان با فراخوانی `Abort()`، کنار گذاشته می‌شود. سپس ریسمان اصلی هر سه ریسمان در حال اجرا را بهم پیوند می‌زند. اثر عمل این است که تا زمانیکه همه ریسمان‌های دیگر کامل نشده‌اند، ریسمان اصلی ادامه نخواهد یافت. زمانی که آنها کامل شوند، ریسمان اصلی یک پیام `All My Thread Are Done` را چاپ می‌کند. کد منبع کامل در مثال ۲۴-۲ نمایش داده می‌شود.

مثال ۲۴-۲

```
#region Using directives
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;
#endregion
namespace InterruptingThreads
{
    class Tester
    {
        static void Main( )
        {
            // make an instance of this class
            Tester t = new Tester( );
            // run outside static Main
            t.DoTest( );
        }
    }
}
```

```

}
public void DoTest( )
{
// create an array of unnamed threads
Thread[] myThreads =
{
new Thread( new ThreadStart(Decrementer) ),
new Thread( new ThreadStart(Incrementer) ),
new Thread( new ThreadStart(Decrementer) ),
new Thread( new ThreadStart(Incrementer) )
};
// start each thread
int ctr = 1;
foreach (Thread myThread in myThreads)
{
myThread.IsBackground = true;
myThread.Start( );
myThread.Name = "Thread" + ctr.ToString( );
ctr++;
Console.WriteLine("Started thread {0}",
myThread.Name);
Thread.Sleep(50);
}
// ask the first thread to stop
myThreads[0].Interrupt( );
// tell the second thread to abort immediately
myThreads[1].Abort( );
// wait for all threads to end before continuing
foreach (Thread myThread in myThreads)
{
myThread.Join( );
}
// after all threads end, print a message
Console.WriteLine("All my threads are done.");
}
// demo function, counts down from 100
public void Decrementer( )
{
try
{
for (int i = 100; i >= 0; i--)
{
Console.WriteLine(
"Thread {0}. Decrementer: {1}",
Thread.CurrentThread.Name,
i);
Thread.Sleep(1);
}
}
catch (ThreadAbortException)
{
Console.WriteLine(
"Thread {0} aborted! Cleaning up...",
Thread.CurrentThread.Name);
}
catch (System.Exception e)
{
Console.
WriteLine("Thread has been interrupted ");
}
finally

```

```

{
    Console.WriteLine(
        "Thread {0} Exiting. ",
        Thread.CurrentThread.Name);
    } }
    // demo function, counts up to 10
public void Incrementer( )
{
    try
    {
        for (int i = 0; i < 10; i++)
        {
            Console.WriteLine(
                "Thread {0}. Incrementer: {1}",
                Thread.CurrentThread.Name,
                i);
            Thread.Sleep(1);
        }
    }
    catch (ThreadAbortException)
    {
        Console.WriteLine(
            "Thread {0} aborted!",
            Thread.CurrentThread.Name);
    }
    catch (System.Exception e)
    {
        Console.
            WriteLine("Thread has been interrupted");
    }
    finally
    {
        Console.WriteLine(
            "Thread {0} Exiting. ",
            Thread.CurrentThread.Name);
    } } }
Output (excerpt):
Started thread Thread\
Thread Thread\. Decrementer: 10
Thread Thread\. Decrementer: 99
Started thread ThreadY
Thread ThreadY. Incrementer: 0
Thread Thread\. Decrementer: 98
Started thread ThreadZ
Thread ThreadZ. Decrementer: 10
Thread Thread\. Decrementer: 97
Thread ThreadY. Incrementer: 1
Started thread ThreadF
Thread ThreadF. Incrementer: 0
Thread ThreadY aborted!
Thread ThreadZ. Decrementer: 99
Thread ThreadY Exiting.
Thread has been interrupted
Thread ThreadZ. Decrementer: 98
Thread ThreadF. Incrementer: 1

```



```
Thread Thread۱ Exiting.
Thread Thread۳. Decrementer: ۹۷
Thread Thread۳. Decrementer: ۱
Thread Thread۴. Incrementer: ۹۸
Thread Thread۳. Decrementer: ۰
Thread Thread۴. Incrementer: ۹۹
Thread Thread۳ Exiting.
Thread Thread۴ Exiting.
All my threads are done.
```

دیدید که ابتدا ریسمان اول شروع می‌شود و از ۱۰۰ تا ۹۹ می‌شمارد. سپس ریسمان دوم شروع می‌شود تا زمانیکه ریسمان‌های سوم و چهارم آغاز شوند، این دو ریسمان با هم اجرا می‌شوند. بعد از یک مدت زمان کوتاه، thread۲ گزارش می‌دهد که کنار گذاشته شده است و سپس گزارش می‌دهد در حال خروج است. بعد از زمان کمتری thread۱ گزارش می‌دهد دچار وقفه شده است. چون وقفه، ریسمان را در یک حالت انتظار نگه می‌دارد، اجرای آن به اندازه‌ی Abort فی‌الفور نیست.

دو ریسمان باقیمانده تا زمان انجام کار خود، ادامه می‌یابند. آنها بطور طبیعی خارج می‌گردند و ریسمان اصلی که به همه ریسمان‌ها پیوند خورده بود، با چاپ پیام خروج خود ادامه می‌یابد.

۲۴-۲-همگام‌سازی

گاهی اوقات، ممکن است بخواهید دسترسی به یک منبع همچون خصوصیات و متدهای یک شی را کنترل کنید، تا اینکه در هر لحظه فقط یک متد بتواند آن منبع را استفاده یا تغییر دهد. شی شما شبیه توالی هواپیما است و ریسمان‌های مرتبط شبیه افراد منتظر در صف هستند. همگام‌سازی با یک قفل روی شی فراهم می‌شود. این عمل به توسعه‌دهنده کمک می‌کند تا زمانیکه کار ریسمان اول روی آن شی تمام نشده است، شی دیگری سرزده به آن دسترسی نداشته باشد.

این بخش سه مکانیزم همگام‌سازی را بررسی می‌کند: کلاس InterLock، دستور lock در C# و کلاس Monitor. در ابتدا شما به ایجاد یک منبع اشتراکی نیاز دارید (اغلب یک فایل یا چاپگر یا در حالت ساده یک متغیر صحیح بنام Counter). شما مقدار Counter را در هر دو ریسمان افزایش خواهید داد. برای شروع، متغیر عضو را اعلان کرده و آنرا با صفر مقداردهی کنید.

```
int counter = ۰;
```

متد Incrementer را برای افزایش متغیر عضو Counter تغییر دهید.

```
public void Incrementer()
{
    try
    {
        while (counter < ۱۰۰۰)
        {
            int temp = counter;
            temp++; // increment
            // simulate some work in this method
            Thread.Sleep(۱);
            // assign the Incremented value
            // to the counter variable
            // and display the results
            counter = temp;
            Console.WriteLine(
                "Thread {۰}. Incrementer: {۱}",
```

```
Thread.CurrentThread.Name,
counter);
}}
```

هدف ما در اینجا شبیه‌سازی کاری است که ممکن است با یک منبع کنترل شده انجام می‌شود. عیناً همانطور که ممکن است یک فایل را باز کنید، محتوای آن را تغییر داده و آن را ببندید.

در اینجا مقدار Counter را به یک متغیر موقتی خوانده و متغیر موقتی را افزایش دهید و برای شبیه‌سازی این کار یک میلی ثانیه آنرا بخوابانید و سپس مقدار افزایش یافته را به Counter برگردانید.

مشکل این است که ریسمان اول مقدار (Counter) را می‌خواند و آن را به یک متغیر موقت انتساب می‌دهد. سپس مقدار متغیر موقت را افزایش می‌دهد. زمانی که آن ریسمان کارش را انجام می‌دهد، ریسمان دوم مقدار (Counter) را می‌خواند و مقدار آن را به یک متغیر موقت انتساب می‌دهد. ریسمان اول کار خود را خاتمه داده و مقدار متغیر موقت (۱) را به Counter انتساب داده و آنرا نمایش می‌دهد. ریسمان دوم نیز همان کار را انجام می‌دهد. او چاپ می‌شود در دور بعدی، همان چیز اتفاق می‌افتد. به جای این که دو ریسمان مقادیر ۲ و ۳ و ۴ و ... را بشمارند، مقادیر ۱ و ۲ و ۳ و ۴ و ... را چاپ می‌کند.

مثال ۲۴-۳ خروجی و کد منبع کامل را برای این مثال نشان می‌دهد.

مثال ۲۴-۳

```
#region Using directives
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;
#endregion
namespace SharedResource
{
    class Tester
    {
        private int counter = ۰;
        static void Main( )
        {
            // make an instance of this class
            Tester t = new Tester( );
            // run outside static Main
            t.DoTest( );
        }
        public void DoTest( )
        {
            Thread t۱ = new Thread( new ThreadStart( Incrementer ) );
            t۱.IsBackground = true;
            t۱.Name = "ThreadOne";
            t۱.Start( );
            Console.WriteLine( "Started thread {۰}",
            t۱.Name );
            Thread t۲ = new Thread( new ThreadStart( Incrementer ) );
            t۲.IsBackground = true;
            t۲.Name = "ThreadTwo";
            t۲.Start( );
            Console.WriteLine( "Started thread {۰}",
            t۲.Name );
            t۱.Join( );
        }
    }
}
```

```
t2.Join( );
// after all threads end, print a message
Console.WriteLine( "All my threads are done." );
}
// demo function, counts up to ۱K
public void Incrementer( )
{
    try
    {
        while ( counter < ۱۰۰۰ )
        {
            int temp = counter;
            temp++; // increment
            // simulate some work in this method
            Thread.Sleep( ۱ );
            // assign the decremented value
            // and display the results
            counter = temp;
            Console.WriteLine(
                "Thread {۰}. Incrementer: {۱}",
                Thread.CurrentThread.Name,
                counter );
        }
        catch ( ThreadInterruptedException )
        {
            Console.WriteLine(
                "Thread {۰} interrupted! Cleaning up...",
                Thread.CurrentThread.Name );
        }
        finally
        {
            Console.WriteLine(
                "Thread {۰} Exiting. ",
                Thread.CurrentThread.Name );
        }
    }
}
Output:
Started thread ThreadOne
Started thread ThreadTwo
Thread ThreadOne. Incrementer: ۱
Thread ThreadOne. Incrementer: ۲
Thread ThreadOne. Incrementer: ۳
Thread ThreadTwo. Incrementer: ۳
Thread ThreadTwo. Incrementer: ۴
Thread ThreadOne. Incrementer: ۴
Thread ThreadTwo. Incrementer: ۵
Thread ThreadOne. Incrementer: ۵
Thread ThreadTwo. Incrementer: ۶
Thread ThreadOne. Incrementer: ۶
```

۲۰-۲-۱-کاربرد Interlocked

CLR یک تعداد مکانیزم همگام‌سازی فراهم می‌کند. آنها شامل ابزار همگام‌سازی عمومی همچون بخش‌های بحرانی (با نام قفل‌ها در NET) و کلاس `Monitor` هستند. هر کدام در ادامه‌ی این بخش بحث می‌شوند.

افزایش و کاهش یک مقدار، یک الگوی برنامه‌نویسی عمومی است و آن اغلب به حفاظت از طریق همگام‌سازی نیاز دارد، که CLR یک کلاس خاص بنام `Interlocked` فقط برای این منظور پیشنهاد می‌کند. `Interlocked` فقط دو متد با نام‌های `Increment` و `Decrement` دارد که علاوه بر افزایش و کاهش یک مقدار، کنترل همگام‌سازی را نیز در نظر می‌گیرند. متد `Incrementer` مثال ۲۴-۳ را بصورت زیر تغییر دهید.

```
public void Incrementer( )
{
    try
    {
        while (counter < ۱۰۰۰)
        {
            int temp = Interlocked.Increment(ref counter);
            // simulate some work in this method
            Thread.Sleep(۰);
            // display the incremented value
            Console.WriteLine(
                "Thread {۰}. Incrementer: {۱}",
                Thread.CurrentThread.Name,
                temp);
        }
    }
}
```

بلوک‌های `Catch` و `Finally` و مابقی برنامه عیناً مانند مثال قبلی است. متد `InterLocked.Increment()` یک پارامتر واحد از نوع `int` را می‌پذیرد. چون مقادیر `int` بصورت مقداری ارسال می‌شوند، کلمه‌ی کلیدی `ref` را به همراه آن بکار برید.

متد `Overload Increment()` می‌شود و می‌تواند به جای یک نوع `int`، یک نوع `long` را بگیرد.

زمانی که این تغییر انجام شود، دسترسی به عضو `Counter` همگام می‌شود و خروجی آن همان چیزی است که انتظار می‌رفت.

```
Output (excerpts):
Started thread ThreadOne
Started thread ThreadTwo
Thread ThreadOne. Incrementer: ۱
Thread ThreadTwo. Incrementer: ۲
Thread ThreadOne. Incrementer: ۳
Thread ThreadTwo. Incrementer: ۴
Thread ThreadOne. Incrementer: ۵
Thread ThreadTwo. Incrementer: ۶
Thread ThreadOne. Incrementer: ۷
Thread ThreadTwo. Incrementer: ۸
Thread ThreadOne. Incrementer: ۹
Thread ThreadTwo. Incrementer: ۱۰
Thread ThreadOne. Incrementer: ۱۱
Thread ThreadTwo. Incrementer: ۱۲
Thread ThreadOne. Incrementer: ۱۳
Thread ThreadTwo. Incrementer: ۱۴
Thread ThreadOne. Incrementer: ۱۵
Thread ThreadTwo. Incrementer: ۱۶
Thread ThreadOne. Incrementer: ۱۷
Thread ThreadTwo. Incrementer: ۱۸
Thread ThreadOne. Incrementer: ۱۹
```

Thread ThreadTwo. Incrementer: ۲۰

۲۴-۲-۲-کاربرد قفل‌ها

اگر بخواهید یک مقدار را افزایش یا کاهش دهید، اگر چه شی `InterLocked` خوب است، در بعضی مواقع می‌خواهید دسترسی به منابع دیگر را نیز کنترل کنید. چیزی که لازم است، یک مکانیزم همگام‌سازی کلی می‌باشد. این مکانیزم با ویژگی `lock` در `C#` فراهم می‌شود.

`lock`، یک بخش بحرانی از کد را علامت‌گذاری می‌کند. همگام‌سازی روی شی برگزیده‌ی شما را با یک قفل فراهم می‌سازد. گرامر کاربرد `lock`، درخواست یک قفل روی یک شی و اجرای یک دستور یا بلوکی از دستورات است. قفل در انتهای بلوک دستورات آزاد می‌شود.

`C#` پشتیبانی مستقیم قفل‌ها را از طریق کلمه کلیدی `lock` فراهم می‌کند. یک ارجاع به یک شی را به آن ارسال کرده و به دنبال این کلمه کلیدی بلوک دستورات نوشته می‌شود.

```
lock(expression) statement-block
```

مثال: می‌توانید متد `Incrementer` را با استفاده یک دستور `lock` بصورت زیر تغییر دهید.

```
public void Incrementer()
{
    try
    {
        while (counter < ۱۰۰۰)
        {
            int temp;
            lock (this)
            {
                temp = counter;
                temp ++;
                Thread.Sleep(۱);
                counter = temp;
            }
            // assign the decremented value
            // and display the results
            Console.WriteLine(
                "Thread {۰}. Incrementer: {۱}",
                Thread.CurrentThread.Name,
                temp);
        }
    }
}
```

بلوک‌های `Catch` و `Finally` و بقیه برنامه همانند مثال قبلی هستند. خروجی این کد، مساوی خروجی تولید شده با استفاده از `InterLocked` است.

۲۰-۲-۳-کاربرد مانیتورها

اشیاء استفاده شده تاکنون، برای بیشتر نیازها کافی هستند. برای کنترل پیشرفته روی منابع، ممکن است بخواهید یک مانیتور بکار ببرید. یک مانیتور به شما اجازه می‌دهد تصمیم بگیرید یک همگام‌سازی کی وارد و کی خارج شود و به شما اجازه می‌دهد برای آزاد شدن فضای دیگری از کدتان منتظر بمانید.

زمانی که می‌خواهید همگام‌سازی را شروع کنید، متد `Monitor.Enter()` را با ارسال شی موردنظر جهت قفل کردن، به آن فراخوانی کنید.

```
Monitor.Enter(this);
```

اگر مانیتور در دسترس نباشد، فرض می‌شود شی محافظت شده بوسیله مانیتور در حال استفاده است، می‌توانید کار دیگری انجام دهید. در حالی که منتظر هستید مانیتور در دسترس قرار گیرد، مجدداً سعی کنید. می‌توانید صریحاً (Wait) را فراخوانی کنید، که ریسمان شما را تا زمانی که مانیتور مشغول است معلق می‌سازد و توسعه‌دهنده (Pulse) را برای بیدار کردن ریسمان معلق فراخوانی می‌کند. (Wait) در کنترل نظم و ترتیب ریسمان‌ها کمک می‌کند.

مثال: فرض کنید می‌خواهید یک مقاله را از وب Download کرده و چاپ کنید. جهت بالا بردن کارایی، دوست دارید عمل چاپ در زمینه انجام شود. اما می‌خواهید مطمئن شوید قبل از شروع عمل چاپ حداقل ۱۰ صفحه Download شده است.

ریسمان چاپ شما منتظر خواهد ماند تا زمانی که ریسمان گرفتن فایل، خوانده شدن اندازه کافی از فایل را سیگنال دهد. نمی‌خواهید ریسمان گرفتن فایل را Join کنید، چون احتمال دارد فایل صدها صفحه باشد و نمی‌خواهید تا پایان یافتن عمل Download منتظر بمانید. اما می‌خواهید مطمئن شوید، قبل از چاپ فایل حداقل ۱۰ صفحه از آن Download شده است. (Wait) عین یک بلیط است.

برای شبیه‌سازی این مورد Tester را مجدداً بنویسید و متد Decrementer را به آن اضافه کنید. Incrementer تا ۱۰ به بالا می‌شمارد و متد Decrementer به پایین تا صفر می‌شمارد. شما کاهش شمارنده را شروع نمی‌کنید، مگر اینکه مقدار counter حداقل ۵ باشد.

در Enter, Decrementer را روی مانیتور فراخوانی کنید. سپس مقدار counter را بررسی کنید، اگر کمتر از ۵ باشد، متد (Wait) را روی مانیتور فراخوانی کنید.

```
if (counter < ۵)
{
    Monitor.Wait(this);
}
```

فراخوانی (Wait)، مانیتور را آزاد می‌کند. اما به CLR سیگنال می‌دهد زمانی که مانیتور آزاد شود، مجدداً برگردد. اگر ریسمان فعال Pulse را فراخوانی کند، ریسمان‌های منتظر یک شانس برای اجرای مجدد دریافت می‌کنند.

```
Monitor.Pulse(this);
```

متد (Pulse) به CLR سیگنال می‌دهد که تغییری در حالت رخ داده است، که ممکن است یک ریسمان منتظر را آزاد سازد. زمانی که یک ریسمان مربوط به مانیتور پایان می‌یابد، آن باید انتهای ناحیه‌ی کد کنترل شده را با فراخوانی (Exit) علامت‌گذاری کند.

```
Monitor.Exit(this);
```

مثال ۲۴-۴ شبیه‌سازی را ادامه می‌دهد. یک دسترسی همگام‌سازی شده بر یک متغیر counter را با استفاده از Monitor فراهم می‌سازد.

مثال ۲۴-۴

```
#region Using directives
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;
#endregion
namespace UsingAMonitor
{
    class Tester
    {
        private long counter = ۰;
        static void Main( )
        {
            // make an instance of this class
```

```

Tester t = new Tester( );
// run outside static Main
t.DoTest( );
}
public void DoTest( )
{
// create an array of unnamed threads
Thread[] myThreads =
{
new Thread( new ThreadStart(Decrementer) ),
new Thread( new ThreadStart(Incrementer) )
};
// start each thread
int ctr = ۱;
foreach ( Thread myThread in myThreads )
{
myThread.IsBackground = true;
myThread.Start( );
myThread.Name = "Thread" + ctr.ToString( );
ctr++;
Console.WriteLine( "Started thread {۰}", myThread.Name );
Thread.Sleep( ۵۰ );
}
// wait for all threads to end before continuing
foreach ( Thread myThread in myThreads )
{
myThread.Join( );
}
// after all threads end, print a message
Console.WriteLine( "All my threads are done." );
}
void Decrementer( )
{
try
{
// synchronize this area of code
Monitor.Enter( this );
// if counter is not yet ۱۰
// then free the monitor to other waiting
// threads, but wait in line for your turn
if ( counter < ۱۰ )
{
Console.WriteLine(
"[{۰}] In Decrementer. Counter: {۱}. Gotta Wait!",
Thread.CurrentThread.Name, counter );
Monitor.Wait( this );
}
while ( counter > ۰ )
{
long temp = counter;
temp--;
Thread.Sleep( ۱ );
counter = temp;
Console.WriteLine(
"[{۰}] In Decrementer. Counter: {۱}. ",
Thread.CurrentThread.Name, counter );
}
}
finally
{
Monitor.Exit( this );
}
}

```

```

void Incrementer( )
{
try
{
Monitor.Enter( this );
while ( counter < ۱۰ )
{
long temp = counter;
temp++;
Thread.Sleep( ۱ );
counter = temp;
Console.WriteLine(
"[{۰}] In Incrementer. Counter: {۱}",
Thread.CurrentThread.Name, counter );
}
// I'm done incrementing for now, let another
// thread have the Monitor
Monitor.Pulse( this );
}
finally
{
Console.WriteLine( "[{۰}] Exiting...",
Thread.CurrentThread.Name );
Monitor.Exit( this );
}}}}
Output:
Started thread Thread۱
[Thread۱] In Decrementer. Counter: ۰. Gotta Wait!
Started thread Thread۲
[Thread۲] In Incrementer. Counter: ۱
[Thread۲] In Incrementer. Counter: ۲
[Thread۲] In Incrementer. Counter: ۳
[Thread۲] In Incrementer. Counter: ۴
[Thread۲] In Incrementer. Counter: ۵
[Thread۲] In Incrementer. Counter: ۶
[Thread۲] In Incrementer. Counter: ۷
[Thread۲] In Incrementer. Counter: ۸
[Thread۲] In Incrementer. Counter: ۹
[Thread۲] In Incrementer. Counter: ۱۰
[Thread۲] Exiting...
[Thread۱] In Decrementer. Counter: ۹.
[Thread۱] In Decrementer. Counter: ۸.
[Thread۱] In Decrementer. Counter: ۷.
[Thread۱] In Decrementer. Counter: ۶.
[Thread۱] In Decrementer. Counter: ۵.
[Thread۱] In Decrementer. Counter: ۴.
[Thread۱] In Decrementer. Counter: ۳.
[Thread۱] In Decrementer. Counter: ۲.
[Thread۱] In Decrementer. Counter: ۱.
[Thread۱] In Decrementer. Counter: ۰.
All my threads are done.

```


در این مثال ابتدا Decrementer شروع می‌شود. در خروجی می‌بینید که (Thread) Decrementer آغاز شده و سپس منتظر می‌ماند. سپس Thread ۲ آغاز می‌شود. فقط زمانی که Thread ۲، Thread ۱ را بیدار می‌کند، Thread ۱ کارش را آغاز می‌کند.

سعی کنید آزمایشاتی روی این کد انجام دهید. ابتدا به متد (Pulse) توضیحات اضافه کنید. در می‌یابید که Thread ۱ هرگز ادامه نمی‌یابد. بدون (Pulse)، هیچ سیگنالی به ریسمان‌های منتظر وجود ندارد.

به عنوان آزمایش دوم، Incrementer را مجدداً طوری بنویسید که بعد از هر عمل افزایش مانیتور را بیدار کرده و از آن خارج شود.

```
void Incrementer()
{
    try
    {
        while (counter < ۱۰)
        {
            Monitor.Enter(this);
            long temp = counter;
            temp++;
            Thread.Sleep(۱);
            counter = temp;
            Console.WriteLine(
                "[{۰}] In Incrementer. Counter: {۱}",
                Thread.CurrentThread.Name, counter);
            Monitor.Pulse(this);
            Monitor.Exit(this);
        }
    }
}
```

متد Decrementer را با تغییر دستور if به یک دستور While بصورت زیر مجدداً بنویسید.

```
//if (counter < ۱۰)
while (counter < ۵)
```

این تغییرات باعث می‌شوند Thread ۲ بعد از هر عمل افزایش، متد Decrementer را بیدار کند. زمانی که مقدار Counter کوچکتر از ۵ باشد، Decrementer باید منتظر بماند. زمانی که مقدار Counter از ۵ بالا رود، Decrementer بطور کامل اجرا می‌شود. زمانی که آن اجرا شد، ریسمان Incrementer می‌تواند مجدداً اجرا شود. خروجی بصورت زیر نمایش داده می‌شود.

```
[Thread۲] In Incrementer. Counter: ۲
[Thread۱] In Decrementer. Counter: ۲. Gotta Wait!
[Thread۲] In Incrementer. Counter: ۳
[Thread۱] In Decrementer. Counter: ۳. Gotta Wait!
[Thread۲] In Incrementer. Counter: ۴
[Thread۱] In Decrementer. Counter: ۴. Gotta Wait!
[Thread۲] In Incrementer. Counter: ۵
[Thread۱] In Decrementer. Counter: ۴.
[Thread۱] In Decrementer. Counter: ۳.
[Thread۱] In Decrementer. Counter: ۲.
[Thread۱] In Decrementer. Counter: ۱.
[Thread۱] In Decrementer. Counter: ۰.
[Thread۲] In Incrementer. Counter: ۱
[Thread۲] In Incrementer. Counter: ۲
[Thread۲] In Incrementer. Counter: ۳
```

```
[Thread۲] In Incrementer. Counter: ۴
[Thread۲] In Incrementer. Counter: ۵
[Thread۲] In Incrementer. Counter: ۶
[Thread۲] In Incrementer. Counter: ۷
[Thread۲] In Incrementer. Counter: ۸
[Thread۲] In Incrementer. Counter: ۹
[Thread۲] In Incrementer. Counter: ۱۰
```

توجه: در هنگام برنامه‌نویسی برای برنامه‌های پیچیده بایستی مسائل بن‌بست را در هنگام کاربرد قفل‌ها در نظر گرفت. اگر همه‌ی ریسمانها در حالت Wait باشند، برنامه در بن‌بست قرار می‌گیرد.

۲۰-۳- خلاصه

- ریسمان‌ها مسئول چند وظیفه‌گی یک برنامه‌ی کاربردی واحد هستند.
- ریسمان‌بندی می‌تواند سرعت اجرای برنامه را پایین بیاورد.
- ساده‌ترین راه ایجاد یک ریسمان، ایجاد یک نمونه جدید از کلاس Thread است.
- زمانی که می‌خواهید به یک ریسمان بگویید تا اجرای کامل ریسمان دیگر منتظر بماند، ریسمان اول را به ریسمان دوم پیوند زنید.
- می‌توانید متد ایستای Thread.Sleep() را احضار کنید که ریسمان احضار شده را معلق می‌سازد.
- همگام‌سازی با یک قفل روی شی فراهم می‌شود.
- افزایش و کاهش یک مقدار، یک الگوی برنامه‌نویسی عمومی است و آن اغلب به حفاظت از طریق همگام‌سازی نیاز دارد، که CLR یک کلاس خاص بنام Interlocked فقط برای این منظور پیشنهاد می‌کند.
- lock، یک بخش بحرانی از کد را علامت‌گذاری می‌کند.
- یک مانیتور به شما اجازه می‌دهد تصمیم بگیرید یک همگام‌سازی کی وارد و کی خارج شود.
- فراخوانی Wait()، مانیتور را آزاد می‌کند.

فصل بیست و پنجم

دستیابی به اینترنت

آنچه که در این فصل یاد خواهید گرفت:

- برداشتن فایل‌ها از Web و گذاشتن فایل‌ها روی Web
- کار با سرویس‌های اینترنتی
- آشنایی با کلاس WebClient
- کاربرد کنترل جدید Web Browser در برنامه‌های ویندوز
- دستکاری آدرس‌های IP و انجام مراجعه‌های DNS

در بیشتر مواقع سرویس گیرندگانی که به صفحات ASP.NET دستیابی می‌کنند، کاربرانی خواهند بود که Internet Explorer یا کاوشگرهای وب دیگر همچون Opera یا Firefox را اجرا می‌کنند. با این وجود، ممکن است بخواهید ویژگی‌های کاوش WEB را به برنامه‌ی کاربردی خود اضافه کنید یا نیاز دارید برنامه‌هایی بنویسید که اطلاعات را از یک سایت وب بدست بیاورید. شاید فکر کنید پیاده‌سازی یک سرویس وب، بهترین راه حل است. با این وجود، اگر به سایت‌های عمومی اینترنت دستیابی می‌کنید، ممکن است هیچ کنترلی روی نحوه‌ی پیاده‌سازی سایت نداشته باشید. این فصل امکانات تهیه شده از طریق کلاس‌های پایه .NET را برای بکارگیری پروتکل‌های مختلف شبکه مخصوصاً HTTP و TCP جهت دستیابی به شبکه‌ها و اینترنت را می‌پوشاند.

جالب‌ترین فضاها نامی برای برنامه‌نویسی شبکه، فضاها نامی System.Net و System.Net.Socket هستند. فضای نامی System.Net کلاس‌ها و عملیات سطح بالا را در بر دارد. برای مثال، گرفتن^۱ یا گذاشتن^۲ فایل‌ها روی وب، انجام درخواست‌های وب با استفاده از HTTP و پروتکل‌های دیگر. در حالیکه System.Net.Socket کلاس‌هایی را برای انجام عملیات سطح پایین در بردارد. در صورتی که بخواهید مستقیماً با سوکت‌ها و پروتکل‌هایی نظیر TCP/IP کار کنید،

^۱ Download

^۲ Upload

کلاس‌های سطح پایین را مفید خواهید یافت. متدهای این کلاس‌ها توابع API سوکت ویندوز (Winsock) را تقلید می‌کنند که از واسط سوکت‌های برکلی مشتق می‌شوند.

این فصل یک راهنما برای شبکه‌بندی کامپیوتر نیست، اما مقدمه‌ای بر کاربرد چارچوب .NET در شبکه را شامل می‌شود. نگاهی کوتاه به کاربرد کنترل WebBrowser جدید در محیط ویندوز دارد و اینکه چگونه بعضی از عملیات خاص دستیابی به اینترنت را ساده‌تر انجام می‌دهد.

با این وجود این فصل با ساده‌ترین حالت ارسال یک تقاضا به یک سرور و ذخیره اطلاعات گرفته شده در جواب تقاضا شروع می‌کند.

۲۵-۱-کلاس WebClient

اگر می‌خواهید فقط یک فایل از یک URL خاص را درخواست کنید، System.NET.WebClient ساده‌ترین کلاس .NET برای این کار است. این کلاس سطح بالا برای انجام عملیات بوسیله یک یا دو فرمان طراحی شده است. در حال حاضر چارچوب .NET، URI‌هایی که با پروتکل‌های HTTP، HTTPS، File: شروع می‌شوند، پشتیبانی می‌کند.

توجه: URI تقریباً همان معنی URL را دارد، اما URI کمی کلی‌تر است و شما را به کاربرد پروتکل‌هایی همچون HTTP و FTP مجبور نمی‌سازد.

۲۵-۱-۱-گرفتن فایل‌ها

برای گرفتن فایل‌ها دو متد در کلاس WebClient وجود دارد. متد مورد نظر شما به نحوه‌ی پردازش محتویات فایل وابسته است. اگر می‌خواهید بطور ساده آن فایل را روی دیسک ذخیره کنید، متد DownloadFile() را بکار ببرید. این متد دو پارامتر می‌گیرد: URI فایل و یک محل (مسیر و نام فایل) برای ذخیره داده‌های درخواست شده.

```
WebClient Client = new WebClient();
Client.DownloadFile("http://www.reuters.com/", "ReutersHomepage.htm");
```

معمولاً برنامه کاربردی شما می‌خواهد داده‌های بازبازی کرده از وب سایت را پردازش کند. برای انجام این کار متد OpenRead() را بکار ببرید. متد OpenRead() یک ارجاع از نوع Stream بر می‌گرداند. سپس می‌توانید از آن ارجاع جهت بازبازی اطلاعات به حافظه استفاده کنید.

```
WebClient Client = new WebClient();
Stream strm = Client.OpenRead("http://www.reuters.com/");
```

۲۵-۱-۲-مثالی از WebClient

مثال اول کار متد WebClient.OpenRead() را نشان می‌دهد. محتوای صفحه‌ی گرفته شده را در یک کنترل ListBox نمایش خواهد داد. برای شروع کار ابتدا یک پروژه‌ی جدید c# با استاندارد Windows Forms ایجاد کرده و یک کنترل ListBox بنام listBox1 اضافه کنید که خصوصیت لنگرگاه آن DockStyle.Fill قرار داده شده باشد.

در ابتدای فایل فضا‌های نامی system.Net و System.IO را اضافه کنید و تغییرات زیر را روی سازنده‌ی فرم اصلی ایجاد کنید.

```
public Form1()
{
```

```
InitializeComponent();
System.Net.WebClient Client = new WebClient();
Stream strm = Client.OpenRead("http://www.reuters.com");
StreamReader sr = new StreamReader(strm);
string line;
while ( (line=sr.ReadLine()) != null )
{
    listBox1.Items.Add(line);
}
strm.Close();
}
```

در این مثال کلاس StreamReader را از فضای نامی System.IO به جریان شبکه وصل می‌کنید. با این عمل می‌توانید محتویات فایل را با استفاده از متدهای سطح بالا یی همچون ReadLine بدست آورید. شکل ۲۵-۱ نتایج اجرای این کد را نشان می‌دهد.



شکل ۲۵-۱

کلاس WebClient یک متد OpenWrite() نیز دارد. این متد یک جریان قابل نوشتن برمی‌گرداند که می‌توانید داده‌ها را به یک URI ارسال کنید. همچنین می‌توانید روش ارسال داده به منبع را نیز مشخص کنید. روش پیش‌فرض post است. تکه کد زیر فرض می‌کند یک فهرست قابل نوشتن بنام accept وجود دارد. این کد یک فایل بنام newfile.txt با محتویات hello world در فهرست accept ایجاد خواهد کرد.

```
WebClient webClient = new WebClient();
Stream stream = webClient.OpenWrite("ftp://localhost/accept/newfile.txt");
StreamWriter streamWriter = new StreamWriter(stream);
streamWriter.WriteLine("Hello World");
streamWriter.Close();
```

۲۵-۱-۳- گذاشتن فایل‌ها

کلاس WebClient همچنین متدهای UploadFile() و UploadData() را در اختیار قرار می‌دهد. متد UploadFile یک فایل را از مسیر محلی به یک موقعیت مشخص شده می‌گذارد، در حالی که متد UploadData داده‌های دودویی به شکل آرایه‌ای از بایت‌ها را به URI مشخص شده می‌گذارد.

```
WebClient client = new WebClient();WebClient client = new WebClient();
client.UploadFile("ftp://www.ourwebsite.com/NewFile.htm",
"C:\\WebSiteFiles\\NewFile.htm");
byte[] image;
// code to initialise image so it contains all the binary data for
// some jpg file
client.UploadData("ftp://www.ourwebsite.com/NewFile.jpg", image);
```

۲۵-۱-۴- کلاس‌های WebRequest و WebResponse

اگرچه کاربرد کلاس WebClient بسیار ساده است. اما ویژگی‌های محدود زیادی دارد. مخصوصاً اینکه نمی‌توان آنرا برای تهیه‌ی تصدیق‌گواهی بکار برد. یک مشکل گذاشتن داده‌ها عدم پذیرش بسیاری از سایت‌ها بدون تصدیق‌گواهی است. اضافه کردن اطلاعات سرآیند به درخواست‌ها و بررسی سرآیندها در جواب امکان‌پذیر است، ولی این عمل در WebClient پشتیبانی نمی‌شود، چون WebClient یک کلاس معمول است که برای ارسال یک تقاضا و دریافت پاسخ با هر پروتکلی طراحی شده است (همانند HTTP و WebClient). (FTP نمی‌تواند همه ویژگی‌های خاص هر پروتکل را اداره کند. اگر بخواهید از مزایای این ویژگی‌ها بهره‌مند شوید، شما نیاز دارید دو کلاس دیگر WebRequest و WebResponse را از فضای نامی System.Net بکار ببرید.

ابتدا نحوه‌ی گرفتن یک صفحه وب با استفاده از این کلاس‌ها را ببینید. این مثال همانند قبلی است، اما WebRequest و WebResponse را استفاده می‌کند. کد زیر تغییرات اساسی مورد نیاز روی مثال قبلی WebClient را برای کاربرد کلاس‌های WebRequest و WebResponse نشان می‌دهد.

```
(public Form)
{
;()InitializeComponent

;("WebRequest wrq = WebRequest.Create("http://www.reuters.com
;()WebResponse wrs = wrq.GetResponse
;()Stream strm = wrs.GetResponseStream
;(StreamReader sr = new StreamReader(strm
;string line
(while ( (line = sr.ReadLine()) != null
}
;(.Items.Add(line)\listBox
{
;()strm.Close
{
```

کد مثال با تعریف یک نمونه از WebRequest آغاز گشته است. نمونه از طریق یک سازنده ایجاد نشده است و به جای سازنده با فراخوانی متد ایستای WebRequest.Create() ایجاد می‌شود. همانطور که در ادامه فصل یاد خواهید گرفت، کلاس WebRequest بخشی از یک سلسله مراتب کلاس‌ها است که پروتکل‌های مختلف شبکه را پشتیبانی می‌کنند. متد WebRequest.Create() یک شیئی مناسب پروتکل داده شده ایجاد می‌کند.

کلاس WebRequest تقاضای اطلاعات را برای ارسال به یک URI خاص ارائه می‌دهد. URI به عنوان یک پارامتر به متد Create رد می‌شود. WebResponse داده‌های بازپایی شده از سرور را نشان می‌دهد. با فراخوانی متد WebRequest.GetResponse() یک تقاضای واقعی به سرور وب ارسال کرده و یک شی WebResponse برای بررسی داده‌ی بازگشتی ایجاد می‌شود. همانند شی WebClient می‌توانید یک جریان برای نمایش داده‌ها بکار ببرید. اما در این مثال، متد WebResponse.GetResponseStream() را بکار می‌برید.

۲۵-۱-۵- ویژگی‌های دیگر WebRequest و WebResponse

این بخش بطور گذرا یک مجموعه از توانایی‌های پشتیبانی شده بوسیله‌ی WebRequest و WebResponse و کلاس‌های مرتبط دیگر را بحث می‌کند.

اطلاعات سرآیند HTTP

یک بخش مهم از پروتکل HTTP، توانایی ارسال اطلاعات سرآیند به همراه جریان‌های تقاضا و جواب است. این اطلاعات می‌توانند جزئیات کاوشگر ارسال کننده‌ی تقاضا را در بر داشته باشند. همان طور که انتظار دارید، چارچوب NET پشتیبانی کامل دستیابی به داده‌ها را فراهم می‌سازد. کلاس‌های WebRequest و WebResponse خواندن اطلاعات سرآیند را پشتیبانی می‌کنند. با این وجود دو کلاس مشتق شده اطلاعات خاص HTTP را فراهم می‌سازند: `HttpWebRequest`، `HttpWebResponse`. همانطور که بعداً بطور دقیق خواهید دید، ایجاد یک `WebRequest` با یک `URI` HTTP، یک نمونه شی `HttpWebRequest` نتیجه می‌دهد. چون `HttpWebRequest` از `WebRequest` مشتق می‌شود، می‌توانید نمونه‌ی جدید را به جای `WebRequest` بکار ببرید. علاوه بر این می‌توانید نمونه‌ی ایجاد شده را به یک ارجاع `HttpWebRequest` قالب‌بندی کرده و به خصوصیات خاص پروتکل HTTP دسترسی داشته باشید. به همین شکل متد `GetResponse` یک نمونه از شی `HttpWebResponse` را بصورت یک ارجاع `WebResponse` بر می‌گرداند. مجدداً با یک قالب‌بندی ساده می‌توانید به ویژگی‌های خاص HTTP دستیابی کنید.

با اضافه کردن کد زیر قبل از فراخوانی متد `GetResponse()`، می‌توانید مجموعه‌ای از خصوصیات سرآیند را بررسی کنید.

```
listBox1.Items.Add("Request Timeout (ms) = " + wrq.Timeout);
listBox1.Items.Add("Request Keep Alive = " + hwrq.KeepAlive);
listBox1.Items.Add("Request AllowAutoRedirect = " + hwrq.AllowAutoRedirect);
```

خصوصیت `Timeout` با واحد میلی‌ثانیه است و مقدار پیش‌فرض آن ۱۰۰۰۰۰ می‌باشد. می‌توانید این خصوصیت را برای مقداردهی حداکثر زمان انتظار برای دریافت پاسخ (قبل از رها شدن یک استثنای `WebException`) بکار ببرید. با بررسی خصوصیت `Status` کلاس `WebException`، می‌توانید دلیل استثناء را بیابید. مقدار این خصوصیت مقادیر شمارشی هستند که هر کدام نوع خاصی از خطا را مشخص می‌کنند.

خصوصیت `KeepAlive` یک خصوصیت الحاقی خاص به پروتکل HTTP است. می‌توانید از طریق یک ارجاع `HttpWebRequest` به آن دستیابی کنید. خصوصیت `KeepAlive` استفاده‌ی یک اتصال در چندین تقاضا را مجاز می‌دارد و در زمان‌های بستن و باز کردن مجدد اتصال صرفه جویی می‌کند. مقدار پیش‌فرض آن `true` است.

خصوصیت `AllowAutoRedirect` نیز مختص کلاس `HttpWebRequest` است. این خصوصیت کنترل می‌کند آیا تقاضای وب، جواب‌های تغییر جهت از سرور وب را بطور اتوماتیک دنبال کند؟ مقدار پیش‌فرض آن `true` است. اگر می‌خواهید تعداد تغییر جهت‌ها را محدود کنید، خصوصیت `MaximumAutomaticRedirections` از کلاس `HttpWebRequest` را مقدار دلخواه قرار دهید.

اگرچه کلاس‌های تقاضا و جواب، بیشتر اطلاعات سرآیند را بصورت خصوصیات آشکار می‌سازند، ولی می‌توانید از خود خصوصیت `Headers` برای دیدن کل کلکسیون سرآیندها استفاده کنید. کد زیر را بعد از فراخوانی متد `GetResponse` قرار دهید تا همه‌ی سرآیندها را در کنترل `ListBox` قرار دهد.

```
;"WebRequest wrq = WebRequest.Create("http://www.reuters.com
;"WebResponse wrs = wrq.GetResponse
;WebHeaderCollection whc = wrs.Headers
(++for(int I = 0; I < whc.Count; i
}
```

```
;(listBox).Items.Add("Header " + whc.GetKey(i) + " : " + whc[i]
{
```

این مثال لیست سرآیندها را بصورت شکل ۲-۲۵ تولید می‌کند.

شکل ۲-۲۵



تصدیق

خصوصیت Credentials یکی دیگر از خصوصیات کلاس WebRequest است. اگر نیاز دارید تصدیق گواهی با تقاضای شما همراه گردد، می‌توانید یک نمونه از کلاس NetworkCredentials (از فضای نامی System.Net) با یک نام کاربر و رمز آن ایجاد کنید. باید قطعه کد بعدی را قبل از فراخوانی GetResponse قرار دهید.

```
;"NetworkCredential myCred = new NetworkCredential("myusername", "mypassword"
;wrq.Credentials = myCred
```

۲۵-۱-۶- تقاضاهای ناهمگام

یک ویژگی اضافی کلاس WebRequest توانایی تقاضای صفحات بصورت ناهمگام است. این ویژگی بسیار مهم است، چون تاخیر ما بین ارسال یک تقاضا به یک میزبان و دریافت جواب آن خیلی طولانی می‌باشد. متدهایی همچون WebClient.DownloadData و WebRequest.GetResponse تا زمان کامل شدن سرور بر نخواهند گشت (این دستورات بلوک‌های هستند). ممکن است نخواهید برنامه کاربردی‌تان تا مدت طولانی بی حرکت بماند. در این موارد بهتر است متدهای BeginGetResponse() و EndGetResponse() را بکار ببرید. متد BeginGetResponse بطور ناهمگام کار می‌کند و فوراً اجرای برنامه دنبال می‌شود. در زمان اجرا یک ریسمان پس زمینه برای بازیابی جواب از سرور ایجاد شده و مدیریت می‌شود. به جای برگرداندن یک شی WebResponse، یک شی پیاده‌سازی کننده واسط IAsyncResult بر می‌گرداند. با این واسط می‌توانید نظر سنجی کرده یا منتظر پاسخ بمانید تا جواب در دسترس قرار گیرد و سپس EndGetResponse را برای جمع کردن نتیجه احضار کنید.

می‌توانید یک نماینده به متد BeginGetResponse رد کنید. نماینده‌ی Callback یک متد با پارامتر ورودی از نوع ارجاع به IAsyncResult و مقدار بازگشتی void می‌پذیرد. زمانی که ریسمان جمع‌آوری جواب کارش تمام شود، در زمان اجرا نماینده احضار می‌شود تا شما را از تکمیل کار با خبر سازد. همانطور که در کد بعدی نشان داده شده است، فراخوانی EndGetResponse در متد Callback بازیابی شی WebResponse را برای شما ممکن می‌سازد.

```
(public Form)
{
;()InitializeComponent

;"WebRequest wrq = WebRequest.Create("http://www.reuters.com
;(wrq.BeginGetResponse(new AsyncCallback(OnResponse), wrq
{

(protected void OnResponse(IAsyncResult ar
```



```

}
;WebRequest wrq = (WebRequest)ar.AsyncState
;WebResponse wrs = wrq.EndGetResponse(ar)

```

```
... read the response //
```

```
{
```

توجه کنید که بازیابی شی `WebRequest` اصلی از طریق رد کردن شی به عنوان پارامتر دوم `BeginGetResponse` امکان پذیر است. پارامتر سوم یک شی ارجاع با عنوان پارامتر حالت است. در طول اجرای متد `CallBack` می توانید با استفاده از خصوصیت `IAsyncState` از `IAAsyncResult` حالت شی را بازیابی کنید.

۲۵-۲-نمایش خروجی بصورت یک صفحه HTML

مثال‌ها نشان دادند چگونه کلاس‌های پایه‌ی `NET` گرفتن و پردازش داده‌های اینترنت را آسان می‌سازند. با این وجود، تا بحال شما فایل‌های گرفته شده را بصورت متن خالی دیدید. اغلب اوقات می‌خواهید یک فایل `HTML` را در `Internet Explorer` ببینید. متأسفانه در این نسخه‌ی `NET` مایکروسافت `IE` وجود ندارد. نه اینکه شما نمی‌توانید اینکار را انجام دهید. قبل از نسخه‌ی `NET ۲.۰` می‌توانستید ارجاعی از یک شی `COM` از `IE` ایجاد کنید و توانایی‌های ارتباط داخلی `NET` را برای استفاده از امکانات یک کاوشگر بکار گیرید. حال در `NET ۲.۰` می‌توانید از کنترل داخلی `WebBrowser` استفاده کنید. می‌توانید `IE` را به عنوان یک پردازش شروع کرده و با استفاده از یک کلاس `Process` در فضای نامی `System.Diagnostics` یک صفحه‌ی وب را به آن ارسال کنید.

```

;()Process myProcess = new Process
;"myProcess.StartInfo.FileName = "iexplore.exe
;"myProcess.StartInfo.Arguments = "http://www.wrox.com
;()myProcess.Start

```

با این وجود کد قبلی `IE` را بعنوان پنجره‌ی جداگانه‌ای شروع می‌کند. برنامه شما هیچ ارتباطی با پنجره‌ی جدید ندارد و بنابراین نمی‌تواند آن را کنترل کند.

از طرف دیگر، ایجاد یک کنترل `WebBrowser` جدید، نمایش و کنترل کاوشگر را به عنوان بخشی از برنامه کاربردی شما ممکن می‌سازد. کنترل `WebBrowser` جدید کاملاً پیچیده است و تعداد زیادی متد، خصوصیت و رویداد دارد.

۲۵-۲-۱-کاوش کردن ساده وب از طریق برنامه کاربردی

برای سادگی بیشتر، یک برنامه کاربردی `Windows From` ایجاد کرده و روی آن یک کنترل `TextBox` و یک کنترل `WebBrowser` قرار دهید. سپس این برنامه را طوری می‌سازید که یک `URL` را در کادر متنی وارد کرده و با فشار دادن کلید `Enter` کار واکشی صفحه وب و نمایش سند خروجی را انجام می‌دهد.

در محیط طراحی `VS ۲۰۰۵`، برنامه کاربردی شما بصورت شکل ۲۵-۳ ظاهر می‌گردد.



شکل ۳-۳۵

در این برنامه زمانی که کاربر یک URL را تایپ کرده و کلید Enter را فشار می‌دهد (این کلید در برنامه کاربردی ثبت شده است)، کنترل WebBrowser صفحه مورد نظر را باز می‌کند و در خودش نمایش خواهد داد.

در این مثال هر کلید فشار داده شده در کادر متنی بوسیله رویداد `KeyPress\textBox` گرفته می‌شود. اگر کارکتر وارده کلید `Enter` باشد، کنترل `WebBrowser` صفحه‌ی مورد نظر را باز بای، کرده و در خودش نمایش خواهد داد.

کد این برنامه بصورت زیر ارائه می‌شود.

```
;using System
;using System.Collections.Generic
;using System.ComponentModel
;using System.Data
;using System.Drawing
;using System.Text
;using System.Windows.Forms

namespace CSharpInternet
{
    partial class Form1 : Form
    {
        ()public Form1
        {}

        ;()InitializeComponent
        {

            (private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
            {}

            (if (e.KeyChar == (char)'\r'
            {}

            ;(webBrowser1.Navigate(textBox1.Text
            {
                {
                    {
                        {
```

در متد `Navigate` کنترل `WebBrowser` خصوصیت `Text\textBox` را به عنوان URL مشخص کنید. نتیجه‌ی نهایی در شکل ۲۵-۴ نمایش داده می‌شود.



شکل ۲۵-۴

۲۵-۲-۲-شروع نمونه‌های IE

شاید برای شما جالب نباشد یک کاوشگر در برنامه‌ی کاربردی خود قرار دهید. اما برای شما جالب است، به کاربر اجازه دهید وب سایت شما در یک کاوشگر متداول بیاید. برای مثال یک برنامه کاربردی `Windows From` ایجاد کنید که یک کنترل `LinkLabel` روی آن وجود دارد. روی آن کنترل عبارت `"Visit our company web side"` نوشته شده است. به محض اینکه کنترل را روی فرم قرار دادید، کد زیر را برای باز کردن سایت وب شرکت خودتان در یک کاوشگر مستقل بکار برید.

```
private void linkLabel1_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
}
```

```
;()WebBrowser wb = new WebBrowser
;(wb.Navigate("http://www.wrox.com", true
{
```

در این مثال زمانی که کاربر روی کنترل `LinkLabel` کلیک کند، یک نمونه‌ی جدید از کلاس `WebBrowser` ایجاد می‌شود. سپس با استفاده از متد `WebBrowser.Navigate` صفحه‌ی مورد نظر را باز می‌کند. در صورتی که پارامتر دوم این متد `true` باشد، صفحه‌ی وب در پنجره مجزایی باز می‌شود. بطور پیش فرض این پارامتر `false` است.

۲۵-۲-۳-اعمال کردن بیشتر ویژگی‌های IE روی برنامه کاربردی

توجه کنید که در هنگام کار با مثال قبلی (مثالی که کنترل `WebBrowser` مستقیماً روی فرم قرار داشت)، با کلیک بر روی یک پیوند در صفحه نمایش، URL کادر متنی عوض نمی‌شود. با استفاده از رویدادهای کنترل `WebBrowser` می‌توانیم این تغییر را در برنامه ایجاد کنیم. بطوریکه با کلیک روی هر پیوند، آدرس آن در کادر متنی نمایش داده شود.

بروز کردن عنوان فرم به عنوان صفحه‌ی HTML بسیار ساده است. کافی است رویداد `DocumentTitleChanged` را ایجاد کرده و خصوصیت `Text` فرم را تغییر دهید.

```
(private void webBrowser1_DocumentTitleChanged(object sender, EventArgs e)
{
;()this.Text = webBrowser1.DocumentTitle.ToString
{
```

در این مثال زمانی که عنوان صفحه در کنترل WebBrowser تغییر یابد، رویداد DocumentTitleChanged رخ می‌دهد و شما کادر متنی روی فرم را بر اساس محتوای URL کامل صفحه تغییر خواهید داد. بدین منظور می‌توانید رویداد Navigated کنترل WebBrowser را بکار ببرید.

```
(private void webBrowser1_Navigated(object sender, WebBrowserNavigatedEventArgs e)
{
;()textBox1.Text = webBrowser1.Url.ToString
{
```

در این مثال زمانی که صفحه‌ی تقاضا شده بطور کامل در کنترل WebBrowser باز می‌شود، رویداد Navigated رخ می‌دهد. در این مثال مقدار Text کنترل textBox با URL صفحه بروز می‌شود. یعنی به محض اینکه یک صفحه در کنترل WebBrowser بارگذاری می‌شود، اگر URL آن تغییر یابد، URL جدید در کادر متنی نمایش داده خواهد شد. پس اگر کاربر بر روی یک پیوند در صفحه نمایش داده شده کلیک کند، URL صفحه‌ی جدید در کادر متنی نمایش داده خواهد شد.

حال اگر برنامه کاربردی را با تغییرات قبلی اجرا کنید. شما نحوه‌ی تغییر عنوان فرم و نوار آدرس را همانند نحوه‌ی کار IE خواهید دید. شکل ۲۵-۵ را ملاحظه کنید.



شکل ۵-۳۵

مرحله بعدی، ایجاد یک نوار ابزار شبیه IE است، که به کاربر اجازه می‌دهد کنترل WebBrowser را کمی بیشتر کنترل کند. یعنی شما دکمه‌هایی همچون Back, Forward, Stop, Home.Refresh را بکار خواهید گرفت.

به جای استفاده از کنترل ToolBar قبل از نوار آدرس، یک مجموعه از کنترل‌های Button در بالای فرم قرار خواهید داد. ۵ دکمه را در بالای کنترل TextBox همانند شکل ۲۵-۶ اضافه کنید.



شکل ۲۵-۶

در این مثال متن روی دکمه برای نشان دادن کار دکمه تغییر می‌یابد. البته می‌توانید عکس دکمه‌های روی IE ویندوز را گرفته و روی دکمه‌های خود قرار دهید. دکمه‌ها را با اسامی `buttonBack`، `buttonForward`، `buttonStop`، `buttonRefresh`، `buttonHome` نامگذاری کنید. برای اینکه تغییر اندازه‌ی فرم بطور مناسب کار کند، خصوصیت `Anchor` این دکمه‌ها را `Right` و `Top` قرار دهید.

باید در شروع برنامه دکمه‌های `buttonBack`، `buttonForward`، `buttonStop` غیرفعال باشند. اگر صفحه اولیه روی کنترل `WebBrowser` بارگذاری نشود، هیچ کاربردی ندارند. بعداً متناسب با محلی که کاربر در پشته صفحه‌ها قرار دارد، این دکمه‌ها فعال و غیرفعال خواهد شد. بنابراین زمانی که بارگذاری یک صفحه شروع می‌شود، لازم است دکمه `Stop` را فعال کنید و زمانی که بارگذاری تمام شد، این دکمه غیرفعال شود. کلاس `WebBrowser` خودش همه‌ی این متدها را دارد. پس مثال زیر کاملاً سراسر است.

```
;using System
;using System.Collections.Generic
;using System.ComponentModel
;using System.Data
;using System.Drawing
;using System.Windows.Forms

namespace CSharpInternet
{
    : Form \partial class Form
    {
        ()\public Form
        {
            ;()InitializeComponent
            {

                (_DocumentTitleChanged(object sender, EventArgs e)\private void webBrowser
                {

                    ;().DocumentTitle.ToString\this.Text = webBrowser
                    {
```

```
(private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    if (e.KeyChar == (char)
    {
        (.Text1.Navigate(textBox1webBrowser
        {
        {

private void webBrowser1_Navigated(object sender
(WebBrowserNavigatedEventArgs e
{
;().Url.ToString1.Text = webBrowser1textBox
{

(private void Form1_Load(object sender, EventArgs e)
{
;buttonBack.Enabled = false
;buttonForward.Enabled = false
;buttonStop.Enabled = false
{

(private void buttonBack_Click(object sender, EventArgs e)
{
;().webBrowser1.GoBack
;().textBox1.Text = webBrowser1.Url.ToString
{

(private void buttonForward_Click(object sender, EventArgs e)
{
;().webBrowser1.GoForward
;().textBox1.Text = webBrowser1.Url.ToString
{

(private void buttonStop_Click(object sender, EventArgs e)
{
;().webBrowser1.Stop

{

(private void buttonHome_Click(object sender, EventArgs e)
{
```

```
;()webBrowser1.GoHome
;()textBox1.Text = webBrowser1.Url.ToString
{

(private void buttonRefresh_Click(object sender, EventArgs e
}
;()webBrowser1.Refresh
{

(private void buttonSubmit_Click(object sender, EventArgs e
}
;webBrowser1.Navigate(textBox1.Text
{

(private void webBrowser1_CanGoBackChanged(object sender, EventArgs e
}
(if (webBrowser1.CanGoBack == true
}
;buttonBack.Enabled = true
{
else
}
;buttonBack.Enabled = false
{
{

(private void webBrowser1_CanGoForwardChanged(object sender, EventArgs e
}
(if (webBrowser1.CanGoForward == true
}
;buttonForward.Enabled = true
{
else
}
;buttonForward.Enabled = false
{
{

private void webBrowser1_Navigating(object sender
(WebBrowserNavigatingEventArgs e
}
;buttonStop.Enabled = true
```

```

{

private void webBrowser1_DocumentCompleted(object sender
(WebBrowserDocumentCompletedEventArgs e
}
;buttonStop.Enabled = false
{
{
{
{

```

فعالیت‌های مختلف زیادی در این مثال وجود دارد. چون زمانی که کاربر این برنامه را استفاده می‌کند، گزینه‌های زیادی وجود دارد. برای هر عمل، یک متد خاص از کلاس WebBrowser وجود دارد. برای مثال، برای دکمه‌ی Back روی فرم، متد GoBack() کنترل WebBrowser را بکار برید و برای دکمه‌های دیگر نیز متدهای متناظر همچون Stop، GoForward، Refresh و GoHome را استفاده کنید. بهتر است یک نوار ابزار شبیه IE مایکروسافت ایجاد کنید تا کار با برنامه راحت‌تر باشد.

زمانی که فرم برای اولین بار بارگذاری می‌شود، رویداد Load\Form دکمه‌های مناسب را غیرفعال می‌کند. از اینجا به بعد، کاربر می‌تواند URL را به کادر متنی وارد کرده و روی دکمه submit کلیک کند تا برنامه‌ی کاربردی صفحه دلخواه شما را بازبایی کند.

برای مدیریت فعال و غیرفعال کردن دکمه‌ها، باید یک مجموعه از رویدادها را داشته باشید. همانطور که قبلاً شرح داده شده است، با آغاز عمل گرفتن صفحه، دکمه‌ی Stop فعال می‌شود. بدین منظور یک اداره کننده رویداد برای رویداد Navigating اضافه کنید تا دکمه Stop فعال را کند.

```

private void webBrowser1_Navigating(object sender
(WebBrowserNavigatingEventArgs e
}
;buttonStop.Enabled = true
{

```

زمانی که بارگذاری سند تمام شود، باید دکمه‌ی Stop غیرفعال گردد.

```

private void webBrowser1_DocumentCompleted(object sender
(WebBrowserDocumentCompletedEventArgs e
}
;buttonStop.Enabled = false
{

```

برای فعال و غیرفعال کردن مناسب دکمه‌های Back و Forward، این کار متناسب با توانایی حرکت به جلو و عقب در پشته صفحه‌ها انجام می‌شود. این عملیات از طریق رویدادهای CanGoForwardChanged و CanGoBackChanged حاصل می‌شود.

```

(private void webBrowser1_CanGoBackChanged(object sender, EventArgs e
}

(.CanGoBack == true\if (webBrowser

```



```

}

;buttonBack.Enabled = true
{
else
}

;buttonBack.Enabled = false
{
}

(private void webBrowser1_CanGoForwardChanged(object sender, EventArgs e)
{
}

(.CanGoForward == true)if (webBrowser
{
;buttonForward.Enabled = true
{
else
}

;buttonForward.Enabled = false
{
}
}

```

حالا پروژه را اجرا کرده و یک صفحه وب را بازدید کنید و روی تعدادی پیوند کلیک کنید. شما باید قادر باشید نوار ابزار را جهت توسعه تجربه کاوش بکار ببرید. محصول نهایی در شکل ۲۵-۷ نمایش داده می‌شود.



شکل ۲۵-۷

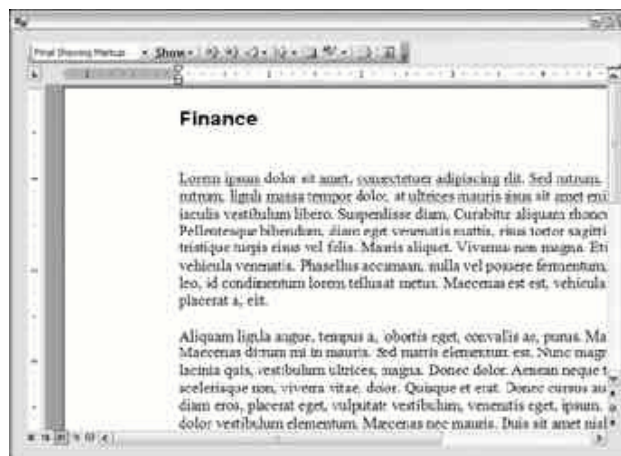
۲۵-۲-۴-نمایش مستندات با استفاده از کنترل WebBrowser

کنترل WebBrowser فقط برای نمایش صفحات وب محدود نشده است. در حقیقت می‌توانید به کاربر اجازه دهید، انواع مختلفی از مستندات را ببیند. تا بحال نحوه استفاده از کنترل WebBrowser را برای دستیابی به مستنداتی که از طریق URL قابل دسترسی هستند دیده‌اید. با این وجود، کنترل WebBrowser به شما اجازه می‌دهد یک مسیر مطلق از فایل‌هایی همچون Word، Excel، PDF و غیره را بکار ببرید.

برای مثال فرض کنید قطعه کد زیر را بکار می‌برید.

```
webBrowser1.Navigate("C:\\Financial Report.doc");
```

این دستور سند Word را در برنامه‌ی کاربردی شما باز خواهد کرد. نه تنها سند در کنترل WebBrowser ظاهر می‌گردد، بلکه نوار ابزار Word نیز ظاهر خواهد گشت. در شکل ۲۵-۸ ملاحظه کنید.



شکل ۲۵-۸

کاوشگر در شکل ۲۵-۹ یک فایل Adobe PDF را نشان می‌دهد.



شکل ۲۵-۹

علاوه بر سادگی باز کردن اسناد خاص در کنترل WebBrowser، کاربران می‌توانند اسناد را به روی کنترل WebBrowser کشیده و رها کنند، سند بطور اتوماتیک در کنترل WebBrowser باز می‌شود. برای غیرفعال کردن این قابلیت، خصوصیت AllowWebBrowser کنترل WebBrowser را false قرار دهید.

۲۵-۲-۵- چاپ کردن بوسیله کنترل WebBrowser

نه تنها کاربران می‌توانند کنترل WebBrowser را برای نمایش صفحات و مستندات بکار برند، بلکه می‌توانند آن را برای چاپ کردن مستندات نیز بکار برند. برای چاپ کردن صفحه یا سند مشاهده شده در کنترل دستور زیر را بکار برید.

```
webBrowser1.Print();
```

همانند قبل برای چاپ کردن یک صفحه یا سند نیازی نیست آن را ببینید. برای مثال، می‌توانید کلاس WebBrowser را برای بارگذاری یک سند HTML و چاپ کردن بکار برید، بدون اینکه سند بارگذاری شده را ببینید. همانند زیر:

```
WebBrowser wb = new WebBrowser();
wb.Navigate("http://www.wrox.com");
wb.Print();
```

۲۵-۲-۶-نمایش کد یک صفحه درخواست شده

در شروع این فصل کلاس‌های WebRequest و Stream را برای گرفتن یک صفحه‌ی دور جهت نمایش کد صفحه‌ی تقاضا شده بکار بردیم. می‌توانید این کد را برای همین کار نیز استفاده کنید.

```
(public Form)
{
;()InitializeComponent

;()System.Net.WebClient Client = new WebClient
;"Stream strm = Client.OpenRead("http://www.reuters.com
;(StreamReader sr = new StreamReader(strm
;string line
(while ( (line=sr.ReadLine()) != null
}
;(.Items.Add(line)\listBox
{

;()strm.Close
```

حال با توجه به امکانات کنترل WebBrowser، بدست آوردن نتایج یکسان ساده است. برای مشاهده‌ی کد صفحه‌ی موردنظر، می‌توانید تغییرات زیر را روی رویداد Document_Completed انجام دهید.

```
,private void webBrowser1_DocumentCompleted(object sender
(WebBrowserDocumentCompletedEventArgs e
}
;buttonStop.Enabled = false
;()textBox2.Text = webBrowser1.DocumentText.ToString
{
```

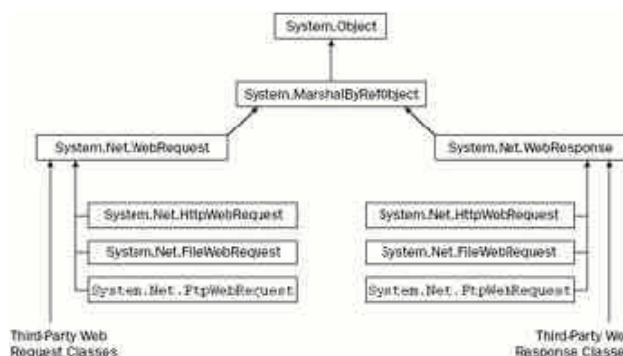
در داخل برنامه یک کنترل TextBox دیگر در زیر کنترل WebBrowser قرار دهید. هدف این است زمانی که کاربر یک صفحه را تقاضا می‌کند، نه تنها جنبه بصری صفحه به نمایش در آید، بلکه کد صفحه را نیز در داخل TextBox ببینیم. می‌توان کد صفحه را از طریق خصوصیت DocumentText کنترل WebBrowser بدست آورد که کل محتوای صفحه را بصورت یک رشته بر می‌گرداند. گزینه دیگر، استفاده از خصوصیت DocumentStream برای گرفتن محتویات صفحه بصورت یک Stream است. نتیجه نهایی اضافه کردن TextBox دوم برای نمایش کد محتویات صفحه را در شکل ۲۵-۱۰ ببینید.



شکل ۱۰-۳۵

۲۵-۳- سلسله مراتب کلاس های Web

در این بخش یک نگاه دقیق تر به معماری اصلی کلاس های `WebRequest` و `WebResponse` می اندازیم. شکل ۲۵-۱۱



شکل ۲۵-۱۱

این سلسله مراتب تعداد کلاس های بیشتری نسبت به آن دو کلاس استفاده شده در کدهای این فصل دارد. همچنین باید بدانید کلاس های `WebRequest` و `WebResponse` هر دو انتزاعی هستند و نمی توان نمونه هایی از آنها تعریف کرد. این کلاس های پایه عملیات کلی رفتار با درخواست ها و جواب های وب را مستقل از هر نوع پروتکل مورد استفاده فراهم می کنند. تقاضاها بوسیله ی پروتکل های خاصی (http, FTP, SMTP و غیره) ایجاد می شوند و یک کلاس مشتق شده برای اداره کردن این تقاضاها نوشته شده اند. به خاطر دارید که در مثال های قبلی متغیرها بصورت ارجاعاتی به کلاس های پایه تعریف می شوند. با این وجود، متد `WebRequest.Create()` یک شی `HttpWebRequest` به شما می داد و متد `GetResponse` در واقع یک شی `HttpWebResponse` بر می گرداند.

این مکانیزم مجازی بیشتر جزئیات را از کد سرویس گیرنده پنهان می سازد و پشتیبانی از انواع مختلف پروتکل ها را از طریق کد پایه یکسانی ممکن می سازد.

با این معماری باید قادر باشید، با استفاده از هر پروتکل تقاضاهایی ارسال کنید. با این وجود، در حال حاضر میکروسافت فقط کلاس های مشتق شده را برای پوشاندن پروتکل های HTTP, HTTPS, FTP فراهم می سازد. در چارچوب NET ۲.۰ آمده است. اگر بخواهید این پروتکل ها را بهینه تر سازید، لازم است از API های ویندوز استفاده کنید یا اینکه در انتظار کلاس های بهتر بمانید.

۲۵-۳-۱- کلاس‌های سودمند

این بخش یک مجموعه از کلاس‌های سودمند را برای ساده‌تر کردن برنامه‌نویسی وب در زمان برخورد با آدرس‌های IP و URI می‌پوشاند.

URI ها

URI و UriBuider دو کلاس از فضای نامی System هستند و هر دو برای نمایش یک URI هستند. UriBuider ایجاد یک URI را از طریق رشته‌هایی برای بخش‌های مختلف آن مجاز می‌دارد و کلاس URI تجزیه و ترکیب و مقایسه URI ها را ممکن می‌کند. سازندهی کلاس URI یک رشتهی URI کامل لازم دارد.

```
Uri MSPage = new
Uri("http://www.Microsoft.com/SomeFolder/SomeFile.htm?Order=true");
```

این کلاس تعدادی زیادی خصوصیت فقط خواندنی را در اختیار قرار می‌دهد. یک شی URI به محض اینکه ایجاد شد، دیگر تغییر داده نمی‌شود.

```
;string Query = MSPage.Query; // Order=true
string AbsolutePath = MSPage.AbsolutePath; // SomeFolder/SomeFile.htm
string Scheme = MSPage.Scheme; // http
(int Port = MSPage.Port; // ۸۰ (the default for http
string Host = MSPage.Host; // www.Microsoft.com
bool IsDefaultPort = MSPage.IsDefaultPort; // true since ۸۰ is default
```

از طرف دیگر UriBuilder تعدادی خصوصیت پیاده‌سازی می‌کند. این خصوصیت‌ها برای ایجاد یک URI کامل کافی هستند. این خصوصیت‌ها خواندنی-نوشتنی هستند. می‌توانید بخش‌های URI را برای سازنده فراهم کنید.

```
Uri MSPage = new
UriBuilder("http", "www.Microsoft.com", 80, "SomeFolder/SomeFile.htm");
```

یا می‌توانید مقادیر خصوصیات URI را مشخص کنید.

```
;UriBuilder MSPage = new UriBuilder
;"MSPage.Scheme = "http
;"MSPage.Host = "www.Microsoft.com
;MSPage.Port = ۸۰
;"MSPage.Path = "SomeFolder/SomeFile.htm
```

به محض اینکه مقداردهی اولیه UriBuilder کامل شد، می‌توانید شی URI متناسب را از طریق خصوصیت URI بدست آورید.

```
Uri CompletedUri = MSPage.Uri;
```

۲۵-۳-۲- آدرس‌های IP و اسامی DNS

در روی اینترنت سرورها از طریق آدرس IP یا نام میزبان (نام DNS گفته می‌شود) تشخیص داده می‌شوند. در کل، نام میزبان یک نام بشر دوستانه است که در پنجره‌ی کاوشگر وب تایپ می‌کنید. از طرف دیگر، یک آدرس IP شناسه‌ی کامپیوترها است که برای شناسایی همدیگر بکار می‌برند. آدرس‌های IP شناسه‌هایی هستند که برای تخمین رسیدن درخواست‌ها و پاسخ‌های وب به ماشین‌های مناسب استفاده می‌شوند. امکان دارد یک کامپیوتر بیش از یک IP داشته باشد.

برای کار با اسامی میزبان، باید یک تقاضای شبکه جهت ترجمه‌ی نام میزبان به یک آدرس IP ارسال گردد. کاری که بوسیله‌ی یک یا چند سرور DNS انجام می‌شود. یک سرور DNS یک جدول نگاشت اسامی میزبان به آدرس‌های IP را برای همه

کامپیوترهایی که می‌شناسد ذخیره می‌کند. همچنین آدرس IP سرورهای DNS دیگر را نیز نگه می‌دارد. کامپیوتر محلی شما حداقل باید یک سرور DNS را بشناسد. مدیر شبکه در زمان راه‌اندازی یک سیستم این اطلاعات را پیکربندی می‌کند.

قبل از ارسال تقاضا به خارج، ابتدا کامپیوتر شما نام سرور DNS را می‌پرسد و آدرس IP میزبان مورد نظر را از آن می‌گیرد. زمانی که آدرس IP مورد نظر بدست آمد، کامپیوتر می‌تواند تقاضای خود را آدرس‌دهی کرده و آن را روی شبکه ارسال کند. همه این کارها در پشت پرده اتفاق می‌افتد، در حالیکه کاربر فقط وب را کاوش می‌کند.

۲۵-۳-۳-کلاس‌های NET برای آدرس‌های IP

چارچوب NET تعدادی کلاس برای جستجوی آدرس‌های IP و یافتن اطلاعاتی در مورد کامپیوترهای میزبان فراهم می‌کند.

IPAddress

این کلاس یک آدرس IP نشان می‌دهد. خود آدرس از طریق متد `GetAddressBytes()` در دسترس است و شاید از طریق متد `ToString()` به حالت دهدهی نقطه‌ای تبدیل گردد. `IPAddress` یک متد ایستا بنام `Parse` نیز پیاده‌سازی می‌کند که عکس عمل `ToString` را انجام می‌دهد. یک رشته‌ی دهدهی نقطه‌دار را به یک `IPAddress` تبدیل می‌کند.

```
IPAddress ipAddress = IPAddress.Parse("234.56.78.9");
byte[] address = ipAddress.GetAddressBytes();
string ipString = ipAddress.ToString();
```

در این مثال بایت‌های آدرس به یک متغیر آرایه بایتی `address` انتساب داده شده است و به رشته `ipString` مقدار "۲۳۴،۵۶،۷۸،۹" انتساب داده شده است.

کلاس `IPAddress` تعدادی فیلد ایستا برای برگرداندن آدرس‌های خاص فراهم می‌سازد. برای مثال، آدرس‌های `LoopBack` به یک کامپیوتر اجازه می‌دهند به خودش پیام ارسال کند و آدرس `Broadcast` که اجازه می‌دهد یک پیام به شبکه‌ی محلی بخش شود.

```
// The following line will set loopback to "۱۲۷.۰.۰.۱"
// the loopback address indicates the local host
string loopback = IPAddress.Loopback.ToString();

// The following line will set broadcast address to "۲۵۵.۲۵۵.۲۵۵.۲۵۵"
// the broadcast address is used to send a message to all machines on//
// the local network
string broadcast = IPAddress.Broadcast.ToString();
```

IPHostEntry

کلاس `IPHostEntry` اطلاعات مرتبط با یک کامپیوتر میزبان خاص را کپسوله می‌کند. این کلاس نام میزبان را از طریق خصوصیت `HostName` در دسترس قرار می‌دهد و خصوصیت `AddressList` یک آرایه از اشیاء `IPAddress` بر می‌گرداند. در مثال `DNSLookupResolver` کلاس `IPHostEntry` استفاده خواهد شد.

کلاس `DNS` جهت بازیابی آدرس‌های IP، قادر است با سرور `DNS` پیش‌فرض ارتباط برقرار کند. دو متد ایستای مهم `Resolve`، `GetHostByAddress()` هستند که متد `Resolve()` برای دستیابی به جزئیات یک میزبان از طریق نام میزبان به کار می‌رود و متد `GetHostByAddress` جزئیات میزبان را از طریق IP آن بر می‌گرداند. هر دو متد یک شیء `IPHostEntry` بر می‌گردانند.

```
IPHostEntry wroxHost = Dns.Resolve("www.wrox.com");
IPHostEntry wroxHostCopy = Dns.GetHostByAddress("208.215.179.178");
```

در این کد هر دو شیء `IPHostEntry` جزئیاتی در مورد سرورهای `wrox.com` در بر خواهند داشت.

کلاس DNS با IPAddress و IPHostEntry فرق می‌کند، چون آن توانایی ارتباط با سرورها جهت بدست آوردن اطلاعات را دارد. برخلاف IPHostEntry و IPAddress، تعدادی ساختار داده‌ی ساده به همراه خصوصیات مناسب، دسترسی به داده‌های اصلی را ممکن می‌سازند.

۲۵-۳-۴-مثال DNSLookup

در این مثال کلاس های DNS و مرتبط با IP برای مراجعه به اسامی DNS ارائه می‌شوند. شکل ۲۵-۱۲ را ببینید.



شکل ۲۵-۱۲

در این برنامه‌ی کاربردی کاربر می‌تواند نام DNS را در یک کادر متنی وارد کند. زمانی که کاربر روی دکمه‌ی Resolve کلیک کند، برنامه‌ی متد Dns.Resolve() را برای بازیابی یک ارجاع به شی‌ی IPHostEntry بکار می‌برد و نام میزبان و آدرس‌های IP آن را نمایش می‌دهد. توجه کنید که ممکن است نام میزبان با آنچه که شما تایپ کرده‌اید متفاوت باشد، این زمانی رخ می‌دهد که نام DNS به عنوان مستعار برای نام DNS دیگر عمل کند.

برنامه‌ی کاربردی DNSLookup یک برنامه‌ی کاربردی تحت ویندوز #C است. کنترل‌هایی که بر روی فرم هستند، به ترتیب با اسامی txtBoxInput، btnResolve و txtBoxHostName و listBoxIPs نامگذاری می‌شوند. سپس متد زیر را به عنوان اداره کننده رویداد کلیک btnResolve به کلاس Form اضافه کنید.

```
(void btnResolve_Click (object sender, EventArgs e
{
try
{
;IPHostEntry iphost = Dns.Resolve(txtBoxInput.Text
foreach (IPAddress ip in iphost.AddressList
{
;()string ipaddress = ip.AddressFamily.ToString
;(listBoxIPs.Items.Add(ipaddress
;()listBoxIPs.Items.Add(" " + ip.ToString
{
;txtBoxHostName.Text = iphost.HostName
{
(catch (Exception ex
```



```

}
+ "MessageBox.Show("Unable to process the request because
+ "the following problem occurred:\n"
;("ex.Message, "Exception occurred
{
{

```

توجه کنید که در این کد هر نوع استثنایی کنترل شده است. یک استثناء زمانی است که کاربر یک نام DNS نامعتبر تایپ کند یا شبکه از کار افتاده باشد.

بعد از بازیابی نمونه‌ی `IPHostEntry`، خصوصیت `AddressList` آن را برای بدست آوردن یک آرایه از آدرس‌های IP بکار برید و سپس آن را در یک حلقه `foreach` طی کنید. هر آدرس IP را بصورت یک عدد صحیح و یک رشته نمایش می‌دهد که از متد `IPAddress.AddressFamily.ToString()` کمک می‌گیرد.

۲۵-۴- خلاصه

- جالب‌ترین فضاها برای برنامه‌نویسی شبکه، فضاها نامی `System.Net` و `System.Net.Socket` هستند. فضای نامی `System.Net` کلاس‌ها و عملیات سطح بالا را در بر دارد.
- در صورتی که بخواهید مستقیماً با سوکت‌ها و پروتکل‌هایی نظیر TCP/IP کار کنید، کلاس‌های سطح پایین را مفید خواهید یافت.
- برای گرفتن فایل‌ها دو متد در کلاس `WebClient` وجود دارد.
- متد `UploadFile` یک فایل را از مسیر محلی به یک موقعیت مشخص شده می‌گذارد.
- کلاس `WebRequest` تقاضای اطلاعات را برای ارسال به یک `URI` خاص ارائه می‌دهد.
- `WebResponse` داده‌های بازیابی شده از سرور را نشان می‌دهد.
- کلاس‌های `WebRequest` و `WebResponse` خواندن اطلاعات سرآیند را پشتیبانی می‌کنند.
- اگر نیاز دارید تصدیق گواهی با تقاضای شما همراه گردد، می‌توانید یک نمونه از کلاس `NetworkCredentials` ایجاد کنید
- ایجاد یک کنترل `WebBrowser` جدید، نمایش و کنترل کاوشگر را به عنوان بخشی از برنامه کاربردی شما ممکن می‌سازد.
- کنترل `WebBrowser` فقط برای نمایش صفحات وب محدود نشده است.
- می‌توانید کلاس `WebBrowser` را برای بارگذاری یک سند HTML و چاپ کردن بکار برید.
- `UriBuider` ایجاد یک `URI` را از طریق رشته‌هایی برای بخش‌های مختلف آن مجاز می‌دارد.
- کلاس `IPAddress` تعدادی فیلد ایستا برای برگرداندن آدرس‌های خاص فراهم می‌سازد.
- کلاس `IPHostEntry` اطلاعات مرتبط با یک کامپیوتر میزبان خاص را کپسوله می‌کند.

فصل بیست و ششم

برنامه نویسی شبکه

کاربرد پروتکل های TCP , UDP

آنچه که در این فصل عنوان می گردد:

- مقدمه ای بر شبکه بندی و سوکت ها
- مثال پردازش و انتقال فرمان با TCP
- مثال پردازش و انتقال فرمان با UDP
- ایجاد یک NewsTicker به وسیله چندپخشی UDP

۲۶-۱-مقدمه

می توان شبکه بندی را به عنوان ارتباط داخل پردازش^۱ تعریف کرد. دو یا چند پردازش با همدیگر ارتباط برقرار می کنند. پردازش ها روی کامپیوترهای یکسان یا مختلف یا روی دستگاه های فیزیکی دیگر اجرا می گردند. اتصال مابین گره های شبکه اکثراً^۲ بوسیله یک سیم (LAN- WAN و اینترنت) یا بوسیله فرکانس های رادیویی بی سیم (cell Phone- WireLess LAN- Bluetooth) یا بوسیله نور ماورای بنفش (infrared) برقرار می گردد.

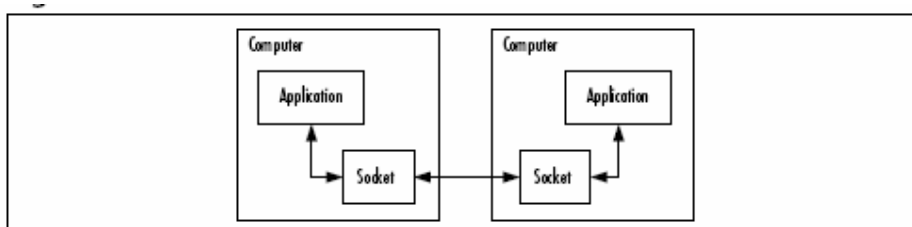
در این فصل مفاهیم پایه ای شبکه بندی و نحوه ی بنا کردن آن در C# را می پوشانیم. با مروری بر تاریخچه شبکه بندی و اینترنت و سوکت ها شروع می کنیم. سپس پروتکل های معمول TCP, UDP را بررسی می کنیم. همچنین روی پورت ها و کاربردهای آنها نگاهی می اندازیم. در نهایت کلاس های موجود NET را شرح می دهیم.

^۱ Inter process communication

داشتن یک روش استاندارد برای ارتباط انواع مختلف شبکه‌ها و انواع مختلف کامپیوترها ضروری است. بنابراین پروتکل TCP/IP بوسیله ARPA توسعه داده شده و یک استاندارد جهانی شد. TCP/IP یک خانواده پروتکل است که ارتباط کامپیوترهای متصل به هم و اشتراک منابع ما بین یک شبکه را مجاز می‌دارد (TCP و IP فقط دو پروتکل از این خانواده هستند). برای دسترسی به همه پروتکل‌های NET، به مستندات کلاس `Sytem.Net.Sockets.Socket` مراجعه کنید.

برای دستیابی به شبکه‌های مبنی بر IP از طریق یک برنامه به سوکت‌ها نیاز داریم. سوکت یک واسط برنامه‌نویسی و نقطه انتهایی ارتباط است که می‌تواند برای اتصال به کامپیوترهای دیگر، ارسال و دریافت داده به آنها استفاده شود. سوکت‌ها در سیستم عامل یونیکس برکلی عنوان شدند. به همین دلیل آنها Berkeley Sockets خوانده شدند. شکل ۱-۲۶ معماری کلی ارتباط مبتنی بر TCP/IP را نشان می‌دهد.

شکل ۱-۲۶



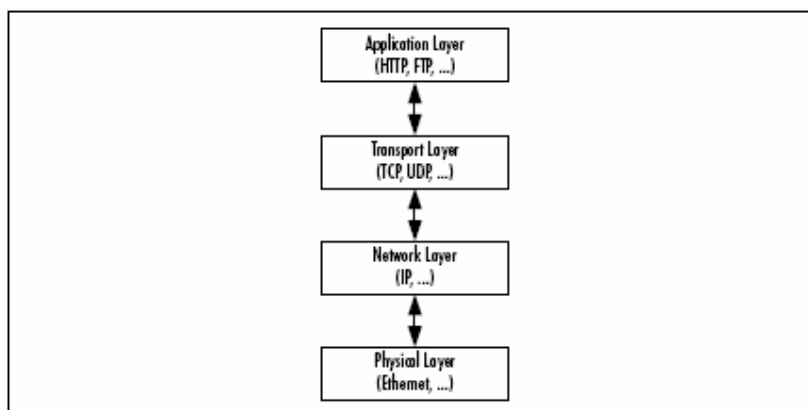
در کل سه نوع سوکت وجود دارد:

- سوکت‌های خام^۱: این سوکت‌ها روی لایه شبکه ایجاد می‌شوند. IP مثالی از سوکت‌های خام است.
- سوکت‌های داده‌گرام^۲: داده‌گرام‌ها بسته‌هایی از داده هستند. این نوع سوکت‌ها روی لایه انتقال پیاده‌سازی می‌شوند (شکل ۲-۲۶ را ببینید). با این وجود، فقط به یک لایه انتساب داده شده است، چون به عنوان مثال، IP مبتنی بر داده‌گرام است.

- سوکت‌های جریان^۳: برخلاف سوکت‌های داده‌گرام، این سوکت‌ها یک جریان داده فراهم می‌کنند.

در ادامه فصل، سوکت‌های داده‌گرام و جریان بصورت دقیق بررسی خواهند شد.

شکل ۲-۲۶



^۱ Raw

^۲ Datagram

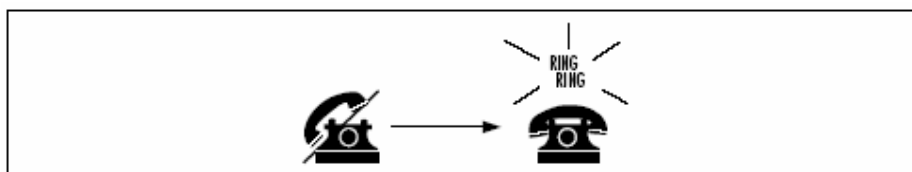
^۳ Stream

معماری‌های ارتباط مدرن، یک پشته از پروتکل‌های مختلف لایه‌ها را بکار می‌برند، که داده‌ها را به لایه بالایی تحویل می‌دهند. هر لایه اطلاعات مرتبط به لایه‌ی خود را به داده‌ها اضافه می‌کند و سپس به لایه بعدی تحویل می‌دهد. داده‌ها توسط پایین‌ترین لایه به کامپیوتر طرف مقابل تحویل داده می‌شوند. در طرف مقابل هر لایه برعکس لایه متناظر خود در ارسال کننده عمل می‌کند. شکل ۲۶-۲ یک پشته پروتکل نشان می‌دهد.

۲۶-۱-۱- مقدمه‌ای بر TCP

TCP یک ارتباط اتصال‌گرا یا جریان‌گرا و مطمئن است. ارتباط TCP شبیه تلفن است. ممکن است بخواهید با عموی خود صحبت کنید. بوسیله شماره‌گیری یک اتصال (نقطه به نقطه) برقرار می‌کنیم. در شکل ۲۶-۳ نمایش داده می‌شود. اگر عموی شما در خانه باشد، گوشی را برداشته با شما صحبت می‌کند (شکل ۲۶-۴). شرکت مخابرات تضمین می‌کند صدای شما را عیناً^۱ به سمت مقابل ارسال کند (قابلیت اطمینان). تا زمانی که ارتباط برقرار است، شما بطور پیوسته صحبت می‌کنید (جریان‌گرا).

شکل ۲۶-۳

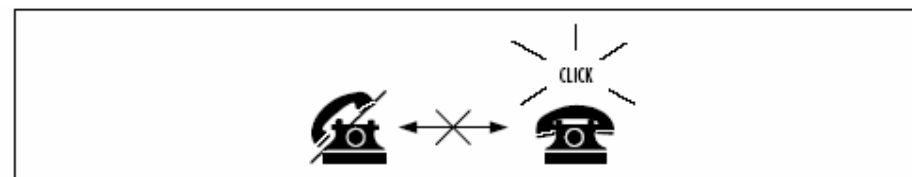


شکل ۲۶-۴



این اتصال تا زمان پایان مکالمه شما ادامه دارد (اتصال‌گرا). شکل ۲۶-۵ را برای مثال قطع اتصال مشاهده کنید.

شکل ۲۶-۵



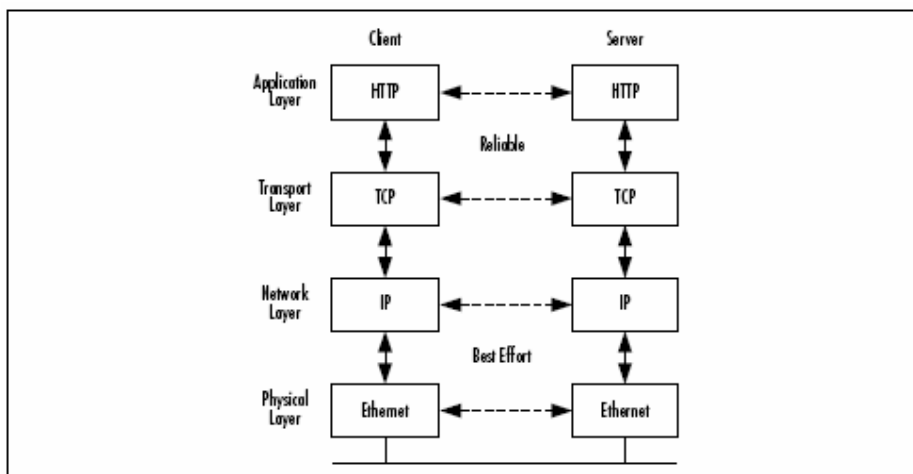
TCP پروتکل IP را به عنوان پروتکل شبکه خود بکار می‌برد. IP داده‌گرام-گرا و پروتکل Best-effort است. همانطور که قبلاً^۲ شرح داده شده، داده‌گرام‌ها بسته‌هایی از داده هستند. Best-effort به این معنی است که داده‌گرام‌ها بدون تضمین تحویل و صحت ترتیب ارسال می‌شوند.

همانطور که دیدید، TCP جریان‌گرا است. باید TCP جریان یافتن داده را شبیه‌سازی کند. بنابراین لازم است ترتیب و صحت داده‌گرام‌ها را کنترل کند. اگر داده‌گرام‌ها را کنترل کند، در صورت خراب شدن یا گم شدن، باید آن را مجدداً^۳ ارسال کند. اگر این کار انجام نشود، یک خطا گزارش می‌شود. TCP تعدادی زمان‌سنج پروتکل برای تضمین ارتباط همگام شده پیاده‌سازی می‌کند. در صورت نیاز، می‌توانیم از این زمان‌سنج‌ها در تولید اتمام مهلت زمانی^۴ استفاده کرد.

^۱ Timeout

مزیت TCP به قابلیت اطمینان آن است. TCP پایه پروتکل‌های قابل اطمینان همچون Http, Ftp, Telnet است. این پروتکل‌ها در صورتی لازم هستند که تحویل و ترتیب بسته‌ها مهم باشد. برای مثال، اگر یک پست الکترونیکی به عمومی خود با عبارت آغازی "Hello Aunt" ارسال کنید. آن به صورت 'Hlnt Aelvo' تحویل داده نمی‌شود. عیب Tcp کاهش کارایی آن به دلیل سربار مدیریت قابلیت اطمینان است. شکل ۲۶-۶ یک پشته از لایه‌های ارتباط بوسیله http را نشان می‌دهد.

شکل ۲۶-۶



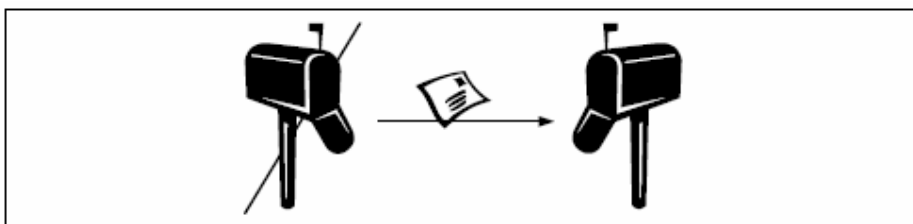
اگر قابلیت اطمینان لازم نباشد، می‌توانیم پروتکل UDP را انتخاب کنیم. این پروتکل در بخش بعدی بحث می‌شود.

۲۶-۱-۲- مقدمه‌ای بر UDP

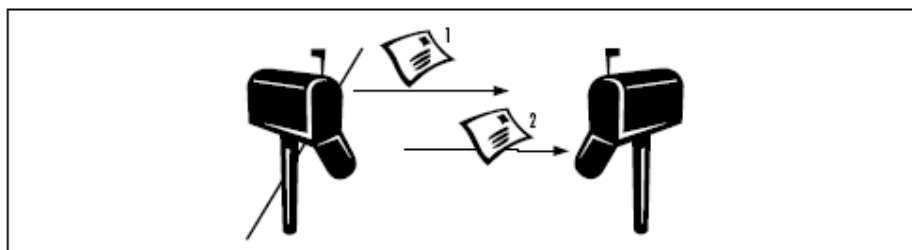
UDP یک پروتکل بی‌اتصال و پروتکل Best-effort داده‌گرام-گرا است. ارتباط UDP شبیه ارسال یک نامه است. ممکن است بخواهید یک نامه به عمومی خود ارسال کنید و نمی‌خواهید نامه را به صورت دستی به عمومی خود تحویل دهید. اداره پست، نامه را از شما تحویل گرفته و یکجا به عمومی شما تحویل می‌دهد (شکل ۲۶-۷ را ببینید). ارسال یک نامه اکثر اوقات نه همیشه، قابل اطمینان است. اداره پست، یک سرویس Best-effort پیشنهاد می‌کند. آنها ترتیب ارسال نامه را تضمین نمی‌کنند. اگر شما نامه ۱ را امروز و نامه ۲ را فردا ارسال کنید، عمومی شما ممکن است نامه شماره ۲ را قبل از نامه ۱ دریافت کند (شکل ۲۶-۸ را ببینید).

از طرف دیگر، ممکن است یکی از نامه‌ها گم شود. اداره پست تضمین نمی‌کند یک نامه حتماً تحویل خواهد شد (شکل ۲۶-۹ را ببینید).

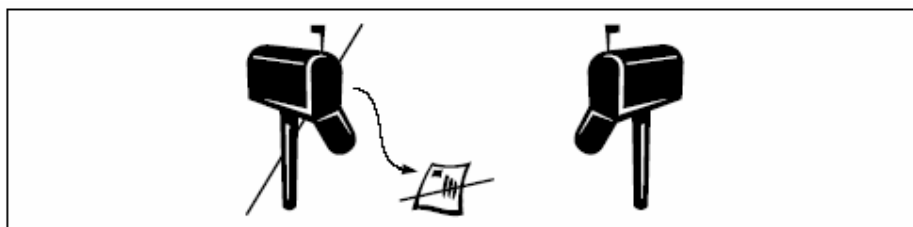
شکل ۲۶-۷



شکل ۲۶-۸



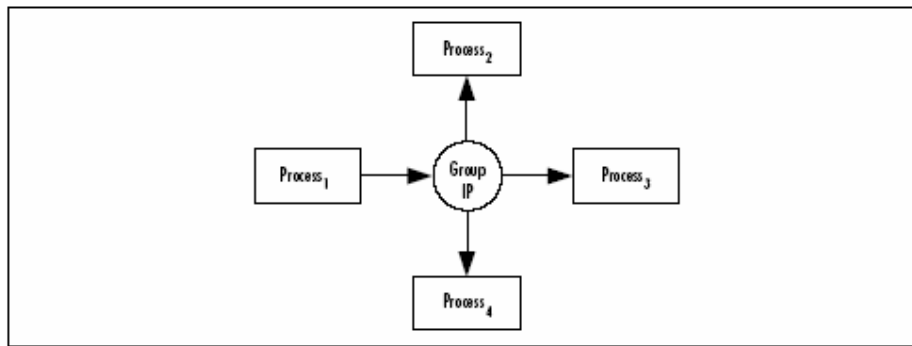
شکل ۹-۲۶



پس چرا این پروتکل استفاده می‌شود. به دلایل زیر:

- کارایی: UDP سریع‌تر از TCP است. چون هیچ بررسی سربرار روی بسته‌ها ندارد. بنابراین از آن در برنامه‌های کاربردی زمان بحرانی همچون جریان ویدیو و جریان صدا استفاده می‌کنند.
 - اگر برنامه شما در مورد گم شدن بسته‌ها دقت نمی‌کند. به عنوان مثال، یک سرور زمان را در نظر بگیرید. اگر سرور یک بسته ارسال کند و آن بسته گم شود، آن اسرار نمی‌کند که زمان را باید مجدداً ارسال کند. اگر سرویس گیرنده در تلاش بعدی آن را دریافت کند، بسته نادرست است.
 - UDP ترافیک کمتری در شبکه ایجاد می‌کند. ۸ بایت برای اطلاعات سرایند پروتکل نیاز دارد. در حالیکه TCP حداقل ۲۰ بایت نیاز دارد. زمانی که صحبت از هاردهای گیگا بایتی است، ۱۶ بایت یک مشکل به حساب نمی‌آید، اما مجموع همه بسته‌های ارسالی در ارتباطات جهانی را تصور کنید که ۱۶ بایت رقم بالایی ایجاد خواهد کرد.
 - اگر برنامه‌ی شما یک پروتکل Best-effort برای تحلیل شبکه نیاز دارد. برای نمونه دستور ping برای آزمایش ارتباط ما بین دو کامپیوتر یا پردازش استفاده می‌شود. این دستور میزان بسته‌های خراب شده یا گم شده را می‌خواهد تا کیفیت اتصال را تعیین کند. پس لازم نیست یک پروتکل قابل اطمینان برای برنامه‌هایی همچون Ping استفاده شود.
- عموماً " UDP برای DNS ، SNMP ، تلفن اینترنتی یا جریان یافتن اطلاعات چند رسانه‌ای استفاده می‌شود. مزیت دیگر UDP در چندپخشی آن است. یعنی تعدادی از پردازش‌ها بوسیله یک IP خاص در یک گروه قرار می‌گیرند (شکل ۱۰-۲۶ را ببینید). آدرس IP باید در محدوده ۲۲۴,۰۰,۱ تا ۲۳۹,۲۵۵,۲۵۵,۲۵۵ باشد. هر پروسه موجود در گروه می‌تواند بسته‌ای را به همه پروسه‌های دیگر گروه ارسال کند.

شکل ۱۰-۲۶



هیچ پروسه‌ای در مورد تعداد پروسه‌های موجود در گروه اطلاعی ندارد. اگر یک برنامه کاربردی بخواهد داده‌ای به دیگری ارسال کند، باید داده را به آدرس IP گروه ارسال کند. روی پروتکل لایه هیچ پروسه‌ای سرور نیست. کار شما تعریف سرورها و سرویس گیرنده‌ها است.

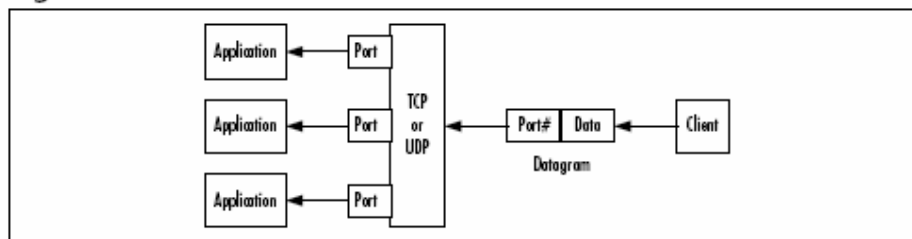
بخش بعدی پورت‌ها را توضیح می‌دهد. پورت‌ها برای تعیین برنامه‌های در حال اجرای روی یک کامپیوتر مهم هستند.

۲۶-۱-۳-مقدمه‌ای بر پورت‌ها

در کل هر کامپیوتری فقط یک اتصال واحد به شبکه دارد. اگر همه داده‌ها از طریق یک اتصال برسند، چگونه مشخص می‌شود آن داده‌ها توسط کدام برنامه‌ی در حال اجرا دریافت می‌شود؟ جواب همان پورت‌ها هستند. پورت یک عدد ۱۶ بیتی در محدوده ۰ تا ۶۵۵۳۵ است. شماره‌های پورت ۰ تا ۱۰۲۳ برای سرویس‌های خاص همچون (۸۰) HTTP، (۲۵) MAIL و (۲۳) Telnet رزرو شده‌اند.

یک برنامه متصل باید حداقل به یک پورت مقید شود. مقید کردن بدین معنی است که یک پورت به سوکت مورد استفاده‌ی برنامه کاربردی اختصاص داده می‌شود. برنامه کاربردی در سیستم عامل ثبت می‌شود. همه بسته‌های ورودی که در سرآیند آنها شماره پورت برنامه مورد نظر قرار دارد، به این برنامه تحویل داده می‌شود (شکل ۲۶-۱۱ را ببینید).

شکل ۲۶-۱۱



توجه کنید که شکل ۲۶-۱۱ به معنای مقید کردن فقط یک سوکت به یک پورت نیست. اگر یک سوکت از طریق یک پورت برای یک اتصال ورودی منتظر بماند، بطور عادی پورت برای برنامه‌های دیگری بلوکه می‌شود. انتظار سوکت برای یک اتصال در سمت سرور است. اگر یک اتصال بوسیله سوکت سرور پذیرفته شود، آن یک سوکت جدید برای این اتصال ایجاد می‌کند. سپس سوکت سرور می‌تواند برای یک تقاضای اتصال جدید منتظر بماند. پس چندین سرویس گیرنده می‌توانند از طریق یک پورت بطور همزمان ارتباط برقرار کنند.

یک مثال برای این کار در سرور WEB است. در حالیکه یک صفحه وب درخواست شده از یک سرور بارگذاری می‌شود، می‌توانید صفحه دیگری از آن سرور را با کاوشگر دیگر بارگذاری کنید. بخش بعدی کلاس‌های مهم Net را بیان می‌کند که ما در مثال‌های خود از آنها استفاده می‌کنیم.

۲۶-۱-۴- فضای نامی System.Net

در حالیکه فضای نامی System.Net.Sockets کلاس‌های پایه برای عملکرد شبکه را دربردارد. فضای نامی System.Net کلاس‌هایی دارد که کلاس‌های پایه شبکه را کپسوله کرده و دستیابی به آنها را راحت‌تر می‌سازند. کلاس‌های System.Net یک واسط برنامه‌نویسی ساده برای بعضی از پروتکل‌های مورد استفاده شبکه دارند.

در مرکز این فضای نامی کلاس‌های WebRequest و WebResponse هستند. این کلاس‌های انتزاعی پایه پیاده‌سازی پروتکل‌ها هستند. دو پروتکل از پیش تعریف شده‌اند: HTTP با کلاس HttpWebRequest (به همراه HttpWebResponse) و پروتکل file://: بوسیله کلاس FileWebRequest (به همراه FileWebResponse).

کلاس‌های کمکی دیگری همچون آدرس‌های IP، کلاس‌های تصدیق و جواز، استثناءها و گواهی‌ها وجود دارند. کلاس‌هایی که در این فصل استفاده می‌کنیم در جدول ۲۶-۱ نشان داده می‌شود.

جدول ۱-۵

توصیف	کلاس
یک آدرس IP را نشان می‌دهد.	IPAddress
یک نقطه انتهایی شبکه را تعیین می‌کند. نقطه انتهایی در شبکه شامل یک IP و یک پورت است.	IPEndPoint

۲۶-۱-۵- فضای نامی System.Net.Sockets

همانطور که قبلاً" شرح داده شده، این فضای نامی کلاس‌های پایه برای فراهم کردن عملیات شبکه را در بر دارد. کلاس مرکزی آن Socket است. همانطور که می‌دانید سوکت اساسی‌ترین واسط برنامه‌نویسی شبکه است. ما بیشتر کلاس‌های این فضای نامی را بکار می‌بریم. جدول ۲۶-۲ این کلاس‌ها را نشان می‌دهد.

جدول ۲-۲۶

توصیف	کلاس
واسط برنامه‌نویسی سوکت‌های برکلی را پیاده‌سازی می‌کند	Socket
دسترسی آسان به داده‌های سوکت‌های جریان را مجاز می‌دارد.	NetworkStream
یک سرویس گیرنده TCP برای وصل به یک سوکت سرور فراهم می‌کند.	TcpClient
یک سوکت TCP سرور برای گوش دادن به تقاضاهای اتصال ورودی پیاده‌سازی می‌کند.	TcpListener
یک سرویس گیرنده UDP با امکان چندپخشی فراهم می‌کند.	UdpClient

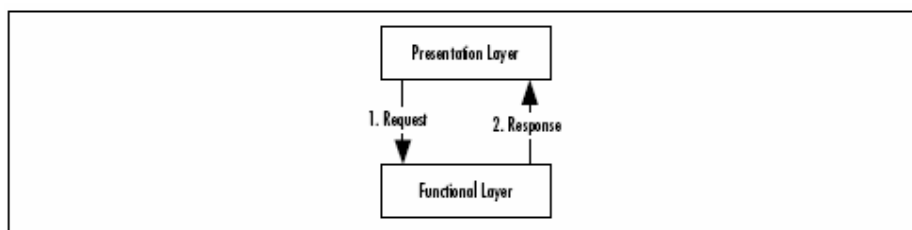
بحث تئوری کافی است. در بخش بعدی با مثال‌های عملی کاربرد TCP و غیره را نشان می‌دهیم.

۲۶-۲- مثال انتقال و پردازش دستورات در TCP

این مثال یک جدا سازی صریح مابین لایه نمایش و لایه عملیات است. لایه نمایش واسط کاربر است، چون باید شما ابتدا روی ارتباط تمرکز کنید. این مثال یک برنامه کنسولی بکار می‌برد و بعداً می‌توانید به ایجاد یک واسط کاربری خوب پردازش. لایه

عملیاتی بخشی از برنامه کاربردی است که همه کار را انجام می‌دهد (برای مثال، یک شی تجاری برای محاسبه بعضی چیزها). شکل ۲۶-۱۲ یک معماری ساده از اولین مثال را نشان می‌دهد

شکل ۲۶-۱۲

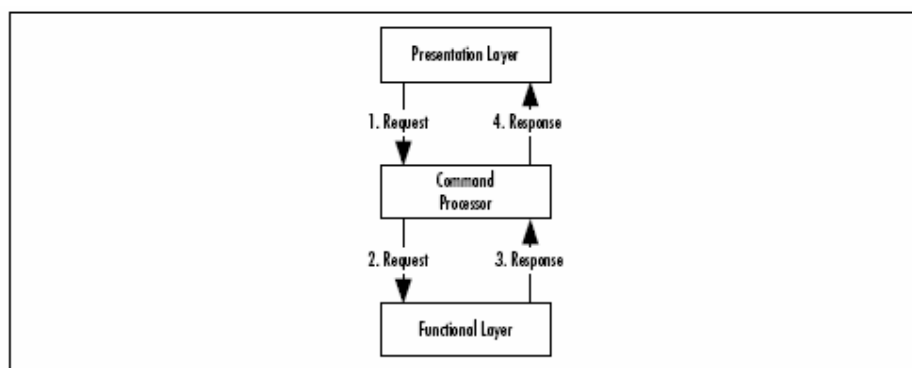


برای لایه‌ی نمایش محل اجرای عملیات مهم نیست. می‌توانید لایه عملیات را در همان برنامه کاربردی، در پروسه دیگر روی همان کامپیوتر یا روی یک کامپیوتر دیگر در LAN یا اینترنت پیاده‌سازی کنید. برای انعطاف‌پذیر کردن این معماری، یک پردازشگر دستور ما بین لایه نمایش و لایه عملیاتی اضافه خواهید کرد. پردازشگر دستور یک واسط استاندارد برای لایه عملیاتی است. لایه‌ی نمایش، تقاضاها را به صورت دستوراتی به پردازنده می‌دهد. پردازنده متدهای لایه عملیاتی را بر اساس دستورات اجرا می‌کند. در نهایت، پردازشگر دستور، نتایج را گرفته و آن را به لایه نمایش بر می‌گرداند. شکل ۲۶-۱۳ معماری توسعه یافته را نشان می‌دهد.

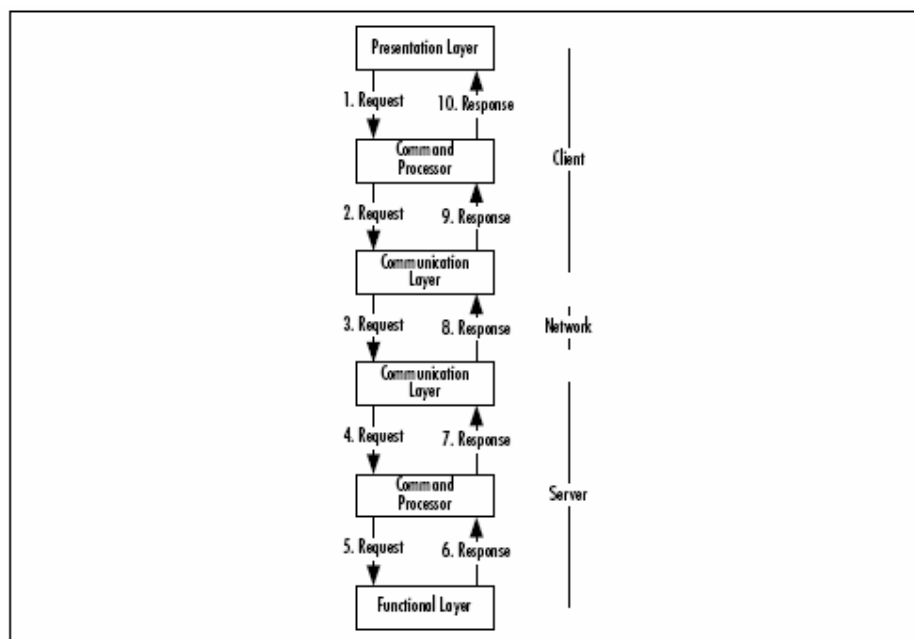
پردازشگر دستور دسترسی به لایه عملیاتی را با روش‌های مختلفی ساده می‌سازد (از طریق همان برنامه کاربردی یا ارتباط شبکه‌ای به کامپیوتر دیگر). شکل ۲۶-۱۴ یک مثال با یک لایه عملیاتی دور را نشان می‌دهد. مزیت این مدل در این است که لایه نمایشی نمی‌داند لایه عملیاتی کجا اجرا می‌گردد. آن فقط دستورات را به پردازشگر دستور داده و نتیجه را دریافت می‌کند.

یک مثال معمول از این نوع معماری، ارتباط کاوشگر وب با سرور وب است. شما در فیلد آدرس کاوشگر خود یک URL تایپ می‌کنید. کاوشگر این URL را به یک تقاضای GET از سرور وب تبدیل کرده و تقاضا را به آن ارسال می‌کند. سرور وب درخواست شما را تحلیل می‌کند و یک صفحه HTML به کاوشگر بر می‌گرداند.

شکل ۲۶-۱۳



شکل ۲۶-۱۴



این مثال همان کار را بصورت بسیار ساده انجام می‌دهد. یک سرویس گیرنده کنسولی، یک تقاضا به سرور ارسال می‌کند و سرور عبارت "Hello World" را به سرویس گیرنده بر می‌گرداند. این مثال یک پروتکل ارتباطی ساده را با دو دستور GET, EXIT پیاده‌سازی می‌کند. نمونه‌ای از ارتباط بصورت زیر است:

```
c: (establish tcp connection to the server)
s: (accept connection)
c: GET<CRLF>
s: "Hello World !"<CRLF>
c: EXIT<CRLF>
s: BYE<CRLF>
c: (close connection)
s: (close connection)
```

C همان سرویس گیرنده و S همان سرویس دهنده را مشخص می‌کند. <CRLF> علامت کلید ENTER است. معمولاً این روش بوسیله پروتکل‌های ارتباطی همچون HTTP یا SMTP استفاده می‌شود.

۲۶-۲-۱- کاربرد کلاس‌های معمول Net.

دو کلاس اصلی شبکه را برای این مثال نیاز دارید. در سمت سرویس گیرنده System.Net.Sockets.TcpClient و در سمت سرویس دهنده System.Net.Sockets.TcpListener. در کل در طرف سرویس گیرنده یک TcpClient به سرویس دهنده متصل می‌گردد. سپس با کمک یک جریان داده، بوسیله سرویس گیرنده روی اتصال کار می‌کنید. بعد از اتمام کار سرویس گیرنده را می‌بندد.

```
// connect client to the server 127.0.0.1:8080
TcpClient client = new TcpClient ( "127.0.0.1", 8080 );
// get the network stream for reading and writing something
// to the network
NetworkStream ns = client.GetStream ();
// read/write something from/to the stream
// disconnect from server
client.Close ();
```

طرف سرویس دهنده یک مرحله بیشتر دارد. اما در کل کد آنها شبیه هم است. یک TcpListener را روی یک پورت محلی مقید کنید. حال اگر یک سرویس گیرنده به گوش کننده وصل شود، یک سوکت بدست می‌آید و بوسیله این سوکت، یک جریان ایجاد می‌کند. از این نقطه کد شبیه طرف سرویس گیرنده است.

```
// create a listener for incoming TCP connections on port 8080
TcpListener listener = new TcpListener ( 8080 );
listener.Start ();
// wait for and accept an incoming connection
```

```
Socket server = listener.AcceptSocket ();
// create a network stream for easier use
NetworkStream ns = new NetworkStream ( server );
// read/write something from/to the stream
// disconnect from client
server.Close ();
```

بعد از یک نگاه به کاربرد کلی کلاس‌های شبکه‌بندی، اولین مثال خود را بررسی می‌کنیم.

۲۶-۲-۲- سرویس دهنده

با سرویس دهنده شروع می‌کنیم. کلاس آن بنام TcpHelloWorldServer است. برای سادگی سرویس گیرنده فقط یک پردازشگر دستور دارد. بعداً" مثال‌هایی نشان می‌دهیم که سرویس دهنده نیز یک پردازشگر دستور دارد. برای سرویس دهنده (قطعه کدهای ۱۵-۲۶ تا ۱۸-۲۶) فضاهای نامی زیر را لازم دارید. کلاس فقط متد Main () را دارد.

قطعه کد ۱۵-۲۶

```
using System;
using System.IO;
using System.Net.Sockets;
```

قطعه کد ۱۵-۲۶ یک قطعه از متد Main () در TcpHelloWorldServer است. آن مقداردهی اولیه سرویس دهنده را نشان می‌دهد. برای اینکه روی پورت ۸۰۸۰ منتظر اتصالات ورودی باشد، یک نمونه از TcpListener را بکار برید.

قطعه کد ۱۶-۲۶

```
Console.WriteLine ( "initializing server..." );
TcpListener listener = new TcpListener ( 8080 );
listener.Start ();
Console.WriteLine ( "server initialized, waiting for " +
"incoming connections..." );
Socket s = listener.AcceptSocket ();
// create a NetworkStream for easier access
NetworkStream ns = new NetworkStream ( s );
// use a stream reader because of ReadLine() method
StreamReader r = new StreamReader ( ns );
```

گوش‌کننده‌ی پورت ۸۰۸۰ به اتصالات ورودی گوش می‌دهد. متد AcceptSocket یک سوکت برای نمایش اتصال سرویس دهنده بر می‌گرداند. این متد برنامه را بلوک می‌کند، تا زمانیکه یک سرویس گیرنده با گوش‌کننده اتصالی باز کند.

بعد از اینکه یک اتصال برپا شد. سوکت برگردانده شده توسط AcceptSocket، برای انتقال داده بوسیله سرویس گیرنده متصل استفاده می‌شود. راحت‌ترین روش انجام این کار، استفاده از یک NetworkStream است. این کلاس در فضای نامی System.Net.Sockets قرار گرفته، که متدهای خواندن و نوشتن در شبکه را کپسوله می‌کند. پس می‌توانید این کلاس را فقط برای کار روی جریان‌ها بکار برید. مرحله بعدی ایجاد یک StreamReader است. این کلاس بخشی از فضای نامی System.IO است. این کلاس دسترسی به یک جریان را ساده می‌کند. در اینجا به خاطر متد ReadLine () از این کلاس استفاده می‌شود. این متد یک خط واحد از کاراکترها را می‌خواند. مجموعه‌ای از کاراکترها که در انتهای آنها n\r قرار می‌گیرد، را خط گویند.

بعد از اینکه سرویس گیرنده یک اتصال برپا می‌کند. آن یک دستور به سرویس دهنده‌ی متصل ارسال می‌کند. حال، دستور ورودی توسط سرویس دهنده تحلیل شده و اجرا می‌شود که آن در قطعه کد ۱۷-۲۶ نمایش داده می‌شود.

قطعه کد ۱۷-۲۶

```
bool loop = true;
while ( loop )
{
// read a line until CRLF
string command = r.ReadLine ();
string result;
Console.WriteLine ( "executing remote command: " +
command );
switch ( command )
{
case "GET":
```

```

result = "Hello World !";
break;
// finish communication
case "EXIT":
result = "BYE";
loop = false;
break;
// invalid command
default:
result = "ERROR";
break;
}
if ( result != null )
{
Console.WriteLine ( "sending result: " + result );
// add a CRLF to the result
result += "\r\n";
// convert data string to byte array
Byte[] res = System.Text.Encoding.ASCII.GetBytes (
result.ToCharArray () );
// send result to the client
s.Send ( res, res.Length, 0 );
}
}

```

اگر دستور GET دریافت شود، سرویس دهنده رشته "Hello World" برمی‌گرداند و حلقه ادامه پیدا می‌کند. در صورتی که دستور مجهولی برسد، دوباره حلقه ادامه پیدا می‌کند. در این حالت، رشته "ERROR" برگردانده می‌شود. با دستور EXIT، سرویس دهنده حلقه را متوقف می‌کند. بعد از این کار، باید اتصال بسته شود (قطعه کد ۱۸-۲۶) را ببینید). برای بستن اتصال متد Close () کلاس Socket را فراخوانی کنید. در نهایت، سرویس دهنده برای فشار دادن کلید Enter منتظر می‌ماند.

قطعه کد ۱۸-۲۶

```

Console.WriteLine ( "clearing up server..." );
s.Close ();
Console.Write ( "press return to exit" );
Console.ReadLine ();

```

تمام این کد برای سرویس دهنده بود، حال سرویس گیرنده را بررسی می‌کنیم.

۲۶-۲-۳- سرویس گیرنده

سرویس گیرنده کمی پیچیده‌تر از سرویس دهنده است. آن دو بخش دارد: برنامه کاربردی کنسولی (UI و پردازشگر دستور که قطعات ارتباط را در بر دارد).

ابتدا پردازشگر دستور بنام TcpRemoteCommandProcessor را بررسی می‌کنیم. قطعه کدهای ۱۹-۲۶ تا ۲۵-۲۶ در فایل Base.cs هستند که Base.cs یک کتابخانه بنام Base.dll تولید خواهد کرد. فضاهای نامی مورد استفاده‌ی پردازشگر دستور در قطعه ۱۹-۲۶ نمایش داده می‌شود.

قطعه کد ۱۹-۲۶

```

using System;
using System.IO;
using System.Net.Sockets;

```

ابتدا یک واسط خواهیم نوشت. پیاده‌سازی بیش از یک پردازشگر دستور با پروتکل‌های اصلی شبکه توسط این واسط امکان‌پذیر است. پس سرویس گیرنده نمونه‌ای از این واسط را بکار می‌برد. این واسط سرویس گیرنده را از پروتکل مورد استفاده شبکه مستقل می‌سازد (قطعه کد ۲۰-۲۶) را ببینید).

قطعه کد ۲۰-۲۶

```

public interface CommandProcessor
{
// execute a command and return the result
// if the return value is false the command processing loop
// should stop
bool Execute ( string command, ref string result );
}

```

حال کلاس TcpRemoteCommandProcessor را برای پیاده‌سازی واسط پردازشگر دستور ایجاد کنید. این کلاس سه متد دارد: یک سازنده، یک متد Close() و پیاده‌سازی متد Execute(). پردازشگر دستور در دو مد مختلف اجرا می‌گردد. در مد اتصال Hold، سازنده در داخل خود اتصال به سرویس دهنده را مستقیماً برقرار می‌کند. قطع اتصال در زمان فراخوانی متد Close() صورت می‌گیرد.

در مد اتصال Release در صورت درخواست ارسال یک دستور به سرویس دهنده از طرف پردازشگر، اتصال برقرار می‌گردد. بعد از بازیابی نتیجه، اتصال بسته می‌شود. مد اول برای ارتباط کوتاه یا بهره‌وری بالا است. مد دوم برای ارتباط طولانی است و می‌تواند برای صرفه‌جویی پول روی اینترنت یا کاهش استفاده از منابع شبکه بکار برده شود.

با فیلدهای کلاس شروع می‌کنیم. قطعه کد ۲۶-۲۱ همه اطلاعات و اشیاء مورد نیاز جهت اجرای پروسه ارتباط را نشان می‌دهد.

قطعه کد ۲۶-۲۱

```
// remote host
private string host = null;
// remote port
private int port = -1;
// connection mode
private bool releaseConnection = false;
// communication interface
private TcpClient client = null;
// outgoing data stream
private NetworkStream outputStream = null;
// ingoing data stream
private StreamReader inputStream = null;
```

حال سازنده کلاس را در قطعه کد ۲۶-۲۲ ببینید. آن سه پارامتر دارد: نام و شماره پورت میزبان و یک پرچم بولین برای تعیین مد اتصال. اگر پرچم True باشد، پردازشگر دستور در مد اتصال Release کار می‌کند، در غیر اینصورت، مد اتصال Hold فعال است. اگر پردازشگر دستور در مد Hold استفاده شود، سازنده به وسیله نام و شماره پورت میزبان فوراً به سرویس دهنده وصل می‌گردد. نهایتاً فیلدهای جریان ورودی و جریان خروجی را مقداردهی اولیه می‌کند.

قطعه کد ۲۶-۲۲

```
public TCPRemoteCommandProcessor ( string host, int port,
bool releaseConnection )
{
// add parameter checking here
this.host = host;
this.port = port;
this.releaseConnection = releaseConnection;
if ( !this.releaseConnection )
{
Console.WriteLine ( "connecting to " + this.host + ":" +
this.port + "..." );
this.client = new TcpClient ( this.host, this.port );
this.outStream = this.client.GetStream ();
this.inStream = new StreamReader ( this.outStream );
Console.WriteLine ( "connected to " + this.host + ":" +
this.port );
}
}
```

متد Close() کاملاً ساده است. آن فقط اتصال را می‌بندد (قطعه کد ۲۶-۲۳ را ببینید). این فقط در مد اتصال Release انجام خواهد شد. اگر پردازشگر فرمان در مد اتصال Hold باشد، این متد هیچ کاری انجام نمی‌دهد و همه فیلدها را null قرار می‌دهد.

قطعه کد ۲۶-۲۳

```
public void Close ()
{
if ( this.client != null )
{
this.client.Close ();
Console.WriteLine ( "connection closed: " + this.host +
":" + this.port );
}
```

```
}
}
```

متد Execute() پیچیده‌تر است. اگر پردازشگر دستور در مد اتصال Release باشد، آن باید ابتدا به سرویس دهنده وصل گردد و بعد از ارسال دستور آن را ببندد (قطعه کد ۲۴-۲۶ را ببینید). در هنگام ارسال در انتهای دستور کاراکترهای کلید Enter الحاق می‌شوند و سپس به یک آرایه بایتی تبدیل می‌شوند. این آرایه به جریان خروجی داده می‌شود. پردازشگر دستور جواب را از جریان ورودی می‌خواند و در نهایت بررسی می‌کند، آیا جواب رشته "BYE" است؟ اگر باشد True در غیر این صورت False برمی‌گرداند.

قطعه کد ۲۴-۲۶

```
public bool Execute ( string command, ref string result )
{
    // add parameter checking here
    bool ret = true;
    if ( this.releaseConnection )
    {
        Console.WriteLine ( "connecting to " + this.host + ":" +
            this.port + "..." );
        // open connection to the server
        this.client = new TcpClient ( this.host, this.port );
        this.outStream = this.client.GetStream ();
        this.inStream = new StreamReader ( this.outStream );
        Console.WriteLine ( "connected to " + this.host + ":" +
            this.port );
    }
    // add a CRLF to command to indicate end
    command += "\r\n";
    // convert command string to byte array
    Byte[] cmd = System.Text.Encoding.ASCII.GetBytes (
        command.ToCharArray () );
    // send request
    this.outStream.Write ( cmd, 0, cmd.Length );
    // get response
    result = this.inStream.ReadLine ();
    if ( this.releaseConnection )
    {
        // close connection
        this.client.Close ();
        Console.WriteLine ( "connection closed: " + host + ":" +
            port );
    }
    ret = !result.Equals ( "BYE" );
    return ret;
}
```

در پایان کار، شما یک سرویس گیرنده برای استفاده پردازشگر فرمان نیاز دارید. آن را TcpHelloWorldClient بنامید. کد منبع این سرویس گیرنده در فایل TcpHelloWorldClient قرار خواهد گرفت. آن برای ارتباط با سرویس دهنده یک نمونه از TcpCommandProcessor ایجاد می‌کند. سپس دستور GET را ارسال می‌کند و نتیجه را روی کنسول نمایش می‌دهد. سرانجام دستور EXIT را ارسال می‌کند و اتصال را می‌بندد.

قطعه کد ۲۵-۲۶

```
using System;
using System.IO;
using System.Net.Sockets;
public class TCPHelloWorldClient
{
    public static void Main ()
    {
        Console.WriteLine ( "initializing client..." );
        TCPRemoteCommandProcessor proc = new
            TCPRemoteCommandProcessor ( "127.0.0.1", 8080, false );
        string result;
        Console.WriteLine ( "requesting..." );
        proc.Execute ( "GET", ref result );
        Console.WriteLine ( "result: " + result );
        Console.WriteLine ( "closing connection..." );
        proc.Execute ( "EXIT", ref result );
        proc.Close ();
        Console.Write ( "press return to exit" );
        Console.ReadLine ();
    }
}
```

}

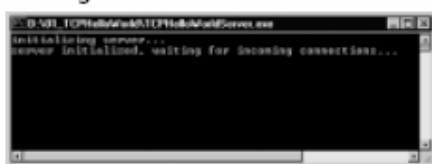
حال می‌توانید مثال را کامپایل کرده و اجرا کنید.

۲۶-۲-۴- کامپایل کردن و اجرای برنامه

فایل‌های منبع آنها را در ۲۰۰۵Vs ایجاد کرده و با اسامی مشخص شده ذخیره کنید و سپس آنها را کامپایل کنید تا فایل‌های اجرایی تولید شود. با دابل کلیک روی برنامه TcpHelloWorldServer، سرویس دهنده را اجرا کنید. یک پنجره کنسول شبیه شکل ۲۶-۲۶ ظاهر می‌گردد.

حال می‌توانید با دابل کلیک کردن روی TcpHelloWorldClient.exe، سرویس گیرنده را شروع کنید. پنجره کنسول دیگر شبیه شکل ۲۶-۲۷ ظاهر خواهد شد.

شکل ۲۶-۲۶

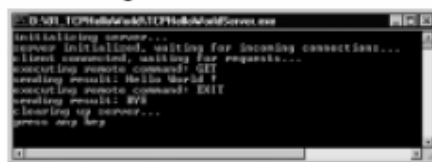


شکل ۲۶-۲۷



پنجره سرور شبیه شکل ۲۶-۲۸ می‌شود. حال می‌توانید با فشار دادن هر کلیدی برنامه‌ها را متوقف کنید. بخش بعدی همان مثال را بوسیله UDP ایجاد می‌کند.

شکل ۲۶-۲۸



۲۶-۳- مثال انتقال و پردازش دستور با UDP

در این بخش مثال بخش قبلی را با پروتکل UDP مجدداً می‌نویسید. مقدمه مربوط به معماری و پیاده‌سازی پروتکل ارتباطی را به خاطر بیاورید.

۲۶-۳-۱- کاربرد کلی کلاس‌های ضروری NET

بر خلاف TCP در کاربرد UDP فقط یک کلاس اصلی شبکه مورد نیاز است. چون اداره کردن ارتباط نظیر به نظیر (P2P) است. در هر دو طرف ارتباط، کلاس System.Net.Sockets.UdpClient را بکار می‌بریم. می‌توان گفت یک سرویس گیرنده UDP به یک پورت محلی مقید می‌شود، تا داده‌ها را از آن دریافت کند. داده‌ها مستقیماً و بدون برقراری ارتباط صریح به سرویس گیرنده UDP دیگر ارسال می‌شوند و آن همان ارتباط بدون اتصال است. در مجموع، کد هر دو طرف یکسان به نظر می‌رسد. یک UdpClient به پورت محلی مقید می‌شود. حال برای ارسال و دریافت داده آماده هستند. چون سرویس

گیرنده فقط به یک پورت محلی مقید شده، پس در متد Send() اطلاعات اتصال به میزبان دور لازم است. این اطلاعات برای ارسال داده به سرویس گیرنده UDP طرف مقابل لازم هستند. چون UdpClient را به پورت محلی مقید کردید، داده‌ها را از این پورت دریافت می‌کنید و برای متد Receive() محل دریافت خاصی را معین نمی‌کنید. بدین دلیل است که متغیر dummy مورد استفاده null قرار دادیم.

```
// bind client to local port where it receives data
UdpClient client = new UdpClient ( 8081 );
// create a byte array containing the characters of
// the string "a request"
Byte[] request = System.Text.Encoding.ASCII.GetBytes (
"a request".ToCharArray () );
// send request to the server
client.Send ( request, request.Length, "127.0.0.1", 8080 );
// create a dummy endpoint
IPEndPoint dummy = null;
// receive something from the server
byte[] response = client.Receive ( ref dummy );
// do something with the response
// unbind the client
client.Close ();
```

با کلاس‌های شبکه در چارچوب NET آشنا شدید، حال مثال دوم را بررسی می‌کنیم.

۲۶-۳-۲-سرور

ابتدا سرور را بررسی می‌کنیم. کلاس فقط یک متد بنام Main() دارد. مقداردهی اولیه سرور بسیار ساده است. فقط باید یک UdpClient را به یک پورت محلی مقید کنید. قطعه کد ۲۶-۲۹ ابتدای متد Main() را نشان می‌دهد.

قطعه کد ۲۶-۲۹

```
Console.WriteLine ( "initializing server" );
UdpClient server = new UdpClient ( 8080 );
```

چون UDP یک پروتکل بدون اتصال است. شما بدون دریافت یک تقاضا نمی‌توانید پاسخی ارسال کنید.

سرآیند داده‌گرام UDP اطلاعات سوکت ارسال کننده پیام را در بر دارد. در روی لایه IP، می‌توانید بگویید که داده‌گرام UDP در یک داده‌گرام IP تعبیه است. سرآیند داده‌گرام IP آدرس IP فرستنده را در بر دارد، اما در #C نمی‌توانید به این اطلاعات از طریق API دسترسی داشته باشید (حداقل با نسخه بتای NET) پس ساده‌ترین راه، اضافه کردن اطلاعات فرستنده به داده‌گرام است (اگر بخواهید گیرنده داده‌هایی را برگرداند). گرامر دستور ارسال به سرور بصورت زیر خواهد بود:

```
IP ADDRESS " : " PORT " : " COMMAND
```

که IP ADDRESS و PORT آدرس IP و پورت فرستنده بوده و COMMAND دستور مورد نظر جهت اجرا است. کد سرور برای دریافت یک دستور در قطعه کد ۲۶-۳۰ نشان داده شده است. بعد از دریافت دستور آن را به بخش‌های مختلف تفکیک می‌کند.

قطعه کد ۲۶-۳۰

```
// an endpoint is not needed the data will be sent
// to the port where the server is bound to
IPEndPoint dummy = null;
bool loop = true;
while ( loop )
{
Console.WriteLine ( "waiting for request..." );
byte[] tmp = server.Receive ( ref dummy );
// split request string into parts, part1=client IP
// address or DNS name, part2=client port, part3=command
string dg =
new System.Text.ASCIIEncoding ().GetString (
datagram );
string[] cmd = dg.Split ( new Char[] { ':' } );
string remoteClientHost = cmd[0];
int remoteClientPort = Int32.Parse ( cmd[1] );
string command = cmd[2];
string result = null;
```

```
// command execution
```

کد اجرای دستور همانند کلاس TcpHelloWorldServer است. بنابراین کد ارسال نتیجه بصورت قطعه کد ۲۶-۳۱ می‌باشد.

قطعه کد ۲۶-۳۱

```
// convert data string to byte array
Byte[] d = System.Text.Encoding.ASCII.GetBytes (
result.ToCharArray () );
// send result to the client
server.Send ( d, d.Length, remoteClientHost,
remoteClientPort );
```

کد متوقف کردن ارتباط شبیه کلاس TcpHelloWorldServer است. حال به بررسی سرویس گیرنده می‌پردازیم.

۲۶-۳-۳-سرویس گیرنده

سرویس گیرنده بنام UdpHelloWorldClient است و نام فایل آن UdpHelloWorldClient.cs است. کد آن با UdpHelloWorldClient فقط یک اختلاف دارد (پردازشگر دستور و ایجاد یک نمونه از آن). پردازشگر دستور بنام UdpCommandProcessor بوده و در فایل BASE.cs ذخیره می‌شود. قطعه کد ۲۶-۳۲ فقط خط متفاوت کد را نشان می‌دهد.

قطعه کد ۲۶-۳۲

```
UDPSRemoteCommandProcessor proc = new UDPSRemoteCommandProcessor ( 8081, "127.0.0.1", 8080 );
```

پارامتر ۸۰۸۱ پورت محلی بوده که پردازشگر دستور به آن مقید می‌شود. دو پارامتر دیگر سازنده، آدرس IP و پورت سرور می‌باشد که پردازشگر دستور به آن متصل می‌گردد.

حال پردازشگر دستور که بنام UdpCommandProcessor است، را بررسی می‌کنیم. شبیه TcpCommandProcessor سه متد دارد: یک سازنده، متد Close() و متد Executed(). ابتدا به فیلدهای کلاس نظری می‌افکنیم (قطعه کد ۲۶-۳۳ را ببینید).

قطعه کد ۲۶-۳۳

```
// the local port where the processor is bound to
private int localPort = -1;
// the remote host
private string remoteHost = null;
// the remote port
private int remotePort = -1;
// communication interface
private UdpClient client = null;
```

در سازنده، همه فیلدها مقداردهی می‌شوند و سرویس گیرنده UDP به یک پورت خاص مقید می‌شود.

قطعه کد ۲۶-۳۴

```
public UDPSRemoteCommandProcessor ( int localPort,
string remoteHost, int remotePort )
{
// add parameter checking here
this.localPort = localPort;
this.remoteHost = remoteHost;
this.remotePort = remotePort;
this.client = new UdpClient ( localPort );
}
```

متد Close() بسیار ساده است. آن متد Close() سرویس گیرنده UDP را فراخوانی می‌کند.

قطعه کد ۲۶-۳۵

```
public void Close ()
{
this.client.Close ();
}
```


متد Execute() بسیار شبیه به متد TcpCommandProcessor است. فقط نحوه اداره کردن ارتباط به دلیل نوع پروتکل (UDP) متفاوت است. کدی برای اضافه کردن آدرس IP سیستم محلی و پورت به دستور لازم است. همچنین روش ارسال و دریافت داده‌ها نیز متفاوت است. قطعه کد ۲۶-۳۶ را برای کد UDP ببینید.

قطعه کد ۲۶-۳۶

```
public bool Execute ( string command, ref string result )
{
    // add parameter checking here
    bool ret = true;
    Console.WriteLine ( "executing command: " + command );
    // build the request string
    string request = "127.0.0.1:" + this.localPort.ToString ()
    + ":" + command;
    Byte[] req = System.Text.Encoding.ASCII.GetBytes (
    request.ToCharArray () );
    client.Send ( req, req.Length, this.remoteHost,
    this.remotePort );
    // we don't need an endpoint
    IPEndPoint dummy = null;
    // receive datagram from server
    byte[] res = client.Receive ( ref dummy );
    result = System.Text.Encoding.ASCII.GetString ( res );
    ret = !result.Equals ( "BYE" );
    return ret;
}
```

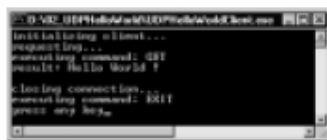
۲۶-۳-۴- کامپایل کردن و اجرای مثال

برنامه UdpHelloWorldClient.CS و UdpHelloWorldServer.CS را کامپایل کنید. ابتدا سرور و سپس سرویس گیرنده را اجرا کنید. پنجره‌های شکل‌های ۲۶-۳۷ و ۲۶-۳۸ و ۲۶-۳۹ را خواهید دید. حال با فشار دادن هر کلیدی برنامه‌های مورد نظر را ببندید.

شکل ۲۶-۳۷



شکل ۲۶-۳۸



شکل ۲۶-۳۹



بخش بعدی نحوه نوشتن یک برنامه کاربردی چندبخشی UDP را بررسی می‌کند.

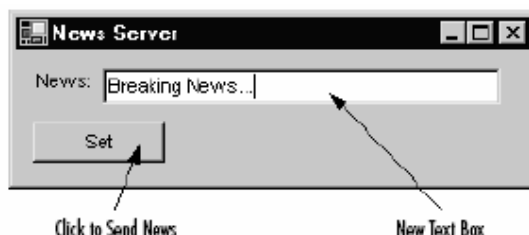
۲۶-۴-۱ ایجاد یک تلگراف اخبار بوسیله چندپخشی UDP

تلگراف اخبار^۱ برنامه‌ای است که یک سرور اخبار، پیام‌ها را به تعدادی سرویس گیرنده ارسال می‌کند. یک سرویس گیرنده خود را در سرور ثبت می‌کند. از آن لحظه به بعد سرویس گیرنده مجاز است پیام‌های جدید سرور را دریافت کند.

می‌توانید این معماری را به چندین روش پیاده‌سازی کنید. اما ساده‌ترین روش کاربرد چندپخشی UDP است. همانطور که در مقدمه شرح داده شد، می‌توانید برنامه‌های کاربردی را هم گروه کنید. یک آدرس IP و یک پورت، نام مستعار یک گروه است. بدین معنی که با ارسال یک داده از طرف یک نظیر به آن IP و پورت، بقیه نظیرهای گروه، آن داده را دریافت خواهند کرد.

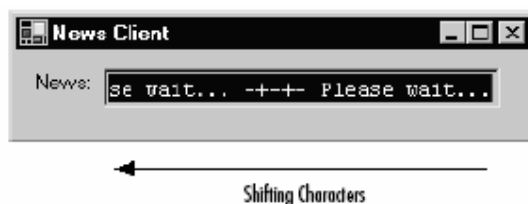
در این بخش، نحوه توسعه یک سرور و سرویس گیرنده تلگراف اخبار ساده را خواهیم دید. سرور یک برنامه کاربردی ویندوز با یک کادر متنی و یک دکمه است. کاربر اخبار را در کادر متنی تایپ می‌کند. با کلیک روی دکمه، سرور خبرها را به گروه ارسال می‌کند (شکل ۲۶-۴۰ را ببینید).

شکل ۲۶-۴۰



سرویس گیرنده یک برنامه کاربردی ویندوز با یک کادر متنی است. اگر اخبار برسند، خبرهای جدید روی این کادر نمایش داده خواهند شد و متن داخل کادر بصورت یک marquee به سمت چپ حرکت خواهد کرد (شکل ۲۶-۴۱ را ببینید).

شکل ۲۶-۴۱



۲۶-۴-۱ کاربرد کلی کلاس‌های مورد نیاز NET

همانطور که در UDP دیدید، فقط یک کلاس بنام `System.Net.Sockets.UdpClient` نیاز دارید. علاوه بر متدهای شرح داده شده در بخش قبلی، می‌توانید متد `UdpClient.JoinMulticastGroup()` را بکار ببرید. این متد یک نظیر UDP را به گروه چندپخشی ثبت می‌کند. مقداردهی اولیه سرور اخبار و سرور گیرنده با همان کد انجام می‌شود. ابتدا یک `UdpClient` به یک پورت محلی مقید کنید. سپس این سرویس گیرنده را با فراخوانی متد `JoinMulticastGroup()` به گروه چندپخشی ثبت کنید. این متد آدرس IP گروه را می‌گیرد. نهایتاً یک `IPEndPoint` برای دریافت داده از آن ایجاد می‌کنید. همانطور که در مقدمه بیان شده، `IPEndPoint` ترکیبی از یک آدرس IP و یک پورت است.

```
// create a peer bound to a local port
UdpClient peer = new UdpClient ( LOCAL_PORT );
// create the group IP address
IPAddress groupAddress = IPAddress.Parse ( GROUP_IP );
// add the peer to the group
peer.JoinMulticastGroup ( groupAddress );
```

```
// create an end point for sending data to the group
IPEndPoint groupEP = new IPEndPoint ( groupAddress,
GROUP_PORT );
```

کد ارسال و دریافت داده شبیه بخش مثال UDP است.

```
// send data to the group, d is a byte array
peer.Send ( d, d.Length, groupEP );
// receiving data from the group
IPEndPoint dummy = null;
byte[] d = peer.Receive ( ref dummy );
```

در بخش بعدی، یک کلاس که بوسیله سرور و سرویس گیرنده‌ی اخبار استفاده می‌شود را شرح می‌دهیم. این کلاس بنام UDPPeer است و در فایل Base.cs قرار دارد. آن یک واسط ساده برای کلاس UdpClient ایجاد می‌کند.

سرور می‌تواند در مد چندپخش و تک‌پخش هدایت شود. اگر آن با پورت محلی ایجاد شود، فقط در مد تک‌پخش فعال است. اگر با یک آدرس IP و پورت گروه چندپخش UDP تعریف شود، مد چندپخش فعال است. متد Close() سرور را متوقف می‌کند و متدهای Receive() و Send() برای دریافت و ارسال داده‌ها استفاده می‌شوند. حال به جزئیات وارد شده و با فیلدهای کلاس شروع می‌کنیم (قطعه کد ۲۶-۴۲ را ببینید).

قطعه کد ۲۶-۴۲

```
// udp peer
private UdpClient server = null;
// multicast group IP address
private IPAddress groupAddress = null;
// multicast group endpoint (IP address and port)
private IPEndPoint group = null;
```

فیلد Server به عنوان یک واسط ارتباطی برای تک‌پخش یا چندپخش لازم است. فیلدهای group و groupAddress فقط در مد چندپخش لازم هستند. فیلد groupAddress آدرس IP گروه چندپخش UDP است و group نقطه انتهایی است که داده به آن ارسال می‌شود.

سازنده‌ی تک‌پخش را در قطعه کد ۲۶-۴۳ می‌بینید. آن بسیار ساده است و فقط یک نظیر UDP را به یک پورت محلی مقید می‌کند.

قطعه کد ۲۶-۴۳

```
public UDPPeer ( int localPort )
{
// add parameter checking here
Console.WriteLine ( "initializing UDP server, port=" +
localPort + "..." );
this.server = new UdpClient ( localPort );
Console.WriteLine ( "UDP server initialized" );
}
```

سازنده‌ی چندپخش، سازنده‌ی تک‌پخش را برای مقید کردن نظیر UDP به پورت محلی فراخوانی می‌کند. علاوه بر این، نظیر را به گروه چندپخش ثبت می‌کند (قطعه کد ۲۶-۴۴ را ببینید). برای عمل ثبت، ایجاد یک نمونه از IPAddress با مقداری اولیه آدرس IP گروه لازم است. این آدرس با فیلد groupAddress ارائه می‌شود. فیلد group نمونه‌ای از کلاس IPEndPoint است و بعداً "برای دریافت داده لازم است".

قطعه کد ۲۶-۴۴

```
public UDPPeer ( int localPort, string groupIP,
int groupPort ) : this ( localPort )
{
// add parameter checking here
Console.WriteLine ( "adding UDP server to multicast " +
"group, IP=" + groupIP + ", port=" + groupPort + "..." );
this.groupAddress = IPAddress.Parse ( groupIP );
this.group = new IPEndPoint ( this.groupAddress,
groupPort );
this.server.JoinMulticastGroup ( this.groupAddress );
Console.WriteLine ( "UDP server added to group" );
}
```

متد Close() بسیار ساده است. در حالت چندپخشی نظیر را از گروه چندپخشی حذف می‌کند. در نهایت، آن متد UdpClient.Close() را فراخوانی می‌کند (قطعه کد ۲۶-۴۵ را ببینید).

قطعه کد ۲۶-۴۵

```
public void Close ()
{
    if ( this.groupAddress != null )
        this.server.DropMulticastGroup ( this.groupAddress );
    this.server.Close ();
}
```

متد ساده‌ی Receive()، اداره کردن آرایه بایتی را کپسوله می‌کند (قطعه کد ۲۶-۴۶ را ببینید). رشته بایتی دریافتی به یک رشته تبدیل می‌گردد و به فراخواننده‌ی این متد برگردانده می‌شود.

قطعه کد ۲۶-۴۶

```
public String Receive ()
{
    IPEndPoint dummy = null;
    // receive datagram
    byte[] data = this.peer.Receive ( ref dummy );
    return new System.Text.ASCIIEncoding ().GetString (
        data );
}
```

متد Send() نیز ساده است. بعد از تبدیل رشته داده شده به یک آرایه بایتی، متد Send() نظیر UDP را فراخوانی می‌کند (قطعه کد ۲۶-۴۷ را ببینید).

قطعه کد ۲۶-۴۷

```
public void Send ( string message )
{
    // add parameter checking here
    Console.WriteLine ( "sending " + message + "..." );
    // convert news string to a byte array
    Byte[] d = System.Text.Encoding.ASCII.GetBytes (
        message.ToCharArray () );
    this.server.Send ( d, d.Length, this.group );
    Console.WriteLine ( "message sent" );
}
```

بخش بعدی واسط کاربری سرور را شرح می‌دهد.

۲۶-۴-۲-سرور

کلاس UDPpeer توسعه یک کلاس واسط برای سرور اخبار را ساده می‌کند. کلاس سرور UDPNewsServer نامگذاری می‌شود و در فایل UDPNewsServer.CS قرار می‌گیرد.

این کلاس یک سازنده و سه متد دارد: یک اداره کننده رویداد برای رویداد بستن پنجره، یک اداره کننده رویداد برای دکمه و یک متد که توسط ریسمان برای ارسال اخبار استفاده می‌شود.

کلاس سرور اخبار از System.Windows.Forms.Form مشتق می‌شود. ابتدا فیلدهای این کلاس را بررسی می‌کنیم (قطعه کد ۵-۴۸ را مشاهده کنید).

قطعه کد ۲۶-۴۸

```
// local port where the UDP server is bound to
private const int LOCAL_PORT = 8080;
// multicast group IP address
private const string GROUP_IP = "225.0.0.1";
// multicast group port
private const int GROUP_PORT = 8081;
// UDP server
private UDPpeer server = null;
// a thread for sending new continuously
private Thread serverThread = null;
// a data field for typing in a new message
```

```
private TextBox text = null;
// a button for setting the new message
private Button setButton = null;
// the news message
private string news = "";
```

قطعه کد ۲۶-۴۹ کد سازنده را نشان می‌دهد که قبل از نمایش UI مقداردهی اولیه را انجام می‌دهد. اگر دکمه Send کلیک شود، باید سرور اخبار، اخبار ارسال شده به گروه چندپخش را بروز کند. به منظور دریافت یک اخطار بوسیله دکمه، متد OnSet () را به عنوان اداره کننده رویداد کلیک ثبت کنید. متد OnClose () برای رویداد بستن پنجره ثبت می‌شود. در نهایت یک ریسمان بوسیله متد Run () بطور پیوسته اخبار تایپ شده در کادر متنی را ارسال می‌کند.

قطعه کد ۲۶-۴۹

```
public UDPNewsServer ()
{
    // UI components initialization
    // add an event listener for click-event
    this.setButton.Click += new System.EventHandler ( OnSet );
    // add an event listener for close-event
    this.Closed += new System.EventHandler ( OnClosed );
    // create communication components
    this.server = new UDPPeer ( LOCAL_PORT, GROUP_IP,
    GROUP_PORT );
    // start communication thread
    this.serverThread = new Thread (
    new ThreadStart ( Run ) );
    this.serverThread.Start ();
    Console.WriteLine ( "initialization complete" );
}
```

چون باید ارسال پیام‌ها پیوسته باشد، استفاده از ریسمان لازم است. حال ریسمان را بررسی می‌کنیم (قطعه کد ۲۶-۵۰ را ببینید). هر ثانیه محتوای فیلد news کلاس را به گروه چندپخش ارسال می‌کند و یک پیام روی کنترل می‌نویسد تا نشان دهد در حال ارسال داده است. بعد از ارسال داده، این متد با فراخوانی متد Sleep () ریسمان را به مدت یک ثانیه مسکوت نگه می‌دارد.

قطعه کد ۲۶-۵۰

```
// sending thread
public void Run ()
{
    while ( true )
    {
        if ( !this.news.Equals ( "" ) )
        {
            Console.WriteLine ( "sending " + this.news );
            this.server.Send ( this.news );
        }
        // wait one second
        Thread.Sleep ( 1000 );
    }
}
```

فیلد news در رویداد کلیک دکمه set مقداردهی می‌شود (قطعه کد ۲۶-۵۱ را ببینید).

قطعه کد ۲۶-۵۱

```
// button click event handler
public void OnSet ( Object sender, EventArgs e )
{
    this.news = this.text.Text;
}
```

نهایتاً کد توقف ارتباط را ملاحظه کنید. آن در رویداد Closed فرم قرار می‌گیرد. این متد در ریسمان ارسال کننده داده با اجرای متد Abort () کار را متوقف می‌کند، تا زمانی که پروسه‌ی پدر از بین رود (این عمل با فراخوانی متد Join () ریسمان امکان پذیر است). بعد از آن، متد Close () شی UDPPeer فراخوانی می‌شود (قطعه کد ۲۶-۵۲ را مشاهده کنید).

قطعه کد ۲۶-۵۲

```
public void OnClosed ( Object sender, EventArgs e )
{
    Console.WriteLine ( "server shut down..." );
}
```

```
// stop thread
this.serverThread.Abort ();
// wait until it's stopped
this.serverThread.Join ();
this.server.Close ();
Application.Exit ();
}
```

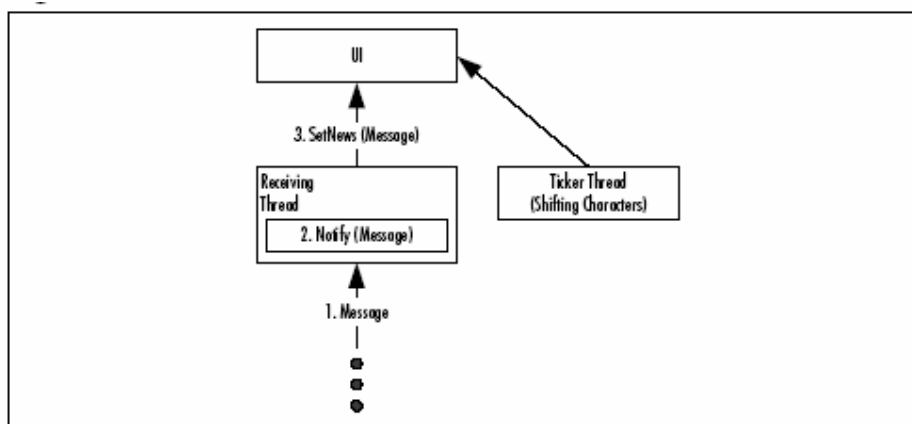
۲۶-۴-۳- سرویس گیرنده

سرویس گیرنده به دو بخش تقسیم می‌شود: یک کلاس سرویس گیرنده چندپخش UDP و یک واسط کاربر. کلاس سرویس گیرنده بنام `UDPMultiCastClient` است که در فایل `base.cs` قرار دارد.

در این مثال یک ارتباط ناهمگام را توسعه می‌دهیم. یک مثال از ارتباط ناهمگام صحبت کردن با دوست از طریق پست الکترونیکی یا chat است. شما یک پیام به دوست خود ارسال می‌کنید و می‌توانید کار دیگری انجام دهید، نه اینکه منتظر جواب بمانید. بعد از اینکه جواب دوست شما دریافت شود، به شما اطلاع داده می‌شود. این یک مثال ناهمگام است، یعنی اینکه واسط کاربری سرویس گیرنده می‌تواند استفاده شود، در حالیکه یک ریسمان در پس زمینه منتظر داده‌های ورودی است. اما زمانی که ریسمان یک پیام بگیرد، باید واسط کاربری را مطلع سازد. این عمل بوسیله ریسمان با فراخوانی یک نماینده که در فرم واسط کاربری پیاده‌سازی شده انجام می‌شود. شکل ۲۶-۵۳ معماری سرویس گیرنده را نشان می‌دهد. سرویس گیرنده از سه قطعه اصلی ساخته می‌شود: واسط کاربری، ریسمان `ticker` و ریسمان دریافت کننده. واسط کاربری یک فرم ساده با یک کادر متنی است. ریسمان `ticker` کارکترهای کادر متنی را کارکتر به کارکتر به چپ انتقال می‌دهد.

ریسمان دریافت کننده در `UDPMultiCastClient` پیاده‌سازی می‌شود و بطور مدام به پیام‌های ورودی گوش می‌کند. اگر یک پیام برسد، آن نماینده `Notify()` را که با متد `SetNews()` پیاده‌سازی شده است، فراخوانی می‌کند. نماینده `Notify()` در قطعه کد ۲۶-۵۴ نشان داده می‌شود. آن در فایل `Base.CS` قرار دارد. آن تا حدی شبیه یک اداره کننده رویداد است. اگر ریسمان یک پیام جدید دریافت کند، نماینده را با ارسال پیام به آن فراخوانی می‌کند.

شکل ۲۶-۵۳



قطعه کد ۲۶-۵۴

```
public delegate void Notify ( string text );
```

نکته: کلاس `System.Net.Sockets.Socket` یک واسط برای سوکت‌های ویندوز ایجاد می‌کند. علاوه بر این، این کلاس متدهایی برای ارتباط ناهمگام در این DLL و در کلاس `Socket` فراهم می‌کند. برای همه متدها همچون `Accept()` یا `Receive()` متدهای ناهمگام شبیه `BeginAccept()` و `EndAccept()` یا `BeginRecieve()` و `EndRecive()` وجود

دارد. برای مثال، `BeginAccept` انتظار ناهمگامی برای یک اتصال ورودی معرفی می‌کند. زمانی که یک اتصال پذیرفته شود، یک نماینده بنام `AsyncCallback` فراخوانی شود.

حال کد `UDPMultiCastClient` بررسی می‌گردد. آن یک سازنده و دو متد دارد. سازنده، سرویس گیرنده‌ی UDP را مقداردهی اولیه می‌کند، که از سرور اخبار پیام‌ها را دریافت می‌کند. متد `Run` یک ریسمان جهت گوش دادن به اخبار استفاده می‌کند و متد `Close` سرویس گیرنده را متوقف می‌سازد. ما حداقل سه فیلد کلاس نیاز داریم. یک نماینده‌ی اخطار، قطعات ارتباط و یک ریسمان برای دریافت ناهمگام داده (قطعه کد ۲۶-۵۵ را ببینید).

قطعه کد ۲۶-۵۵

```
// notification delegate
private Notify notify = null;
// communication interface
private UDPPeer peer = null;
// receiving thread
private Thread clientThread = null;
```

نماینده `Notify` توسط سازنده ذخیره می‌شود و نظیر UDP را با آدرس IP گروه و پورت مقداردهی اولیه می‌کند. در نهایت، ریسمان دریافت کننده‌ی اخبار را راه اندازی می‌کند (قطعه کد ۲۶-۵۶).

قطعه کد ۲۶-۵۶

```
public UDPMulticastClient ( string groupIP, int groupPort,
Notify notify )
{
    // add parameter validation here
    Console.WriteLine ( "initializing UDP multicast " +
        "client, group=" + groupIP + ", port=" + groupPort +
        "...");
    this.notify = notify;
    // create communication components
    this.client = new UDPPeer ( groupPort, groupIP,
        groupPort );
    // start listener thread
    this.clientThread = new Thread (
        new ThreadStart ( Run ) );
    this.clientThread.Start ();
    Console.WriteLine ( "UDP multicast client initialized" );
}
```

ریسمان دریافت کننده بوسیله متد `Run` پیاده‌سازی می‌شود. آن یک حلقه بی‌نهایت است که داده‌های موجود را دریافت می‌کند و آن را به نماینده `Notify` می‌دهد (قطعه کد ۲۶-۵۷).

قطعه کد ۲۶-۵۷

```
public void Run ()
{
    while ( true )
        this.notify ( this.peer.Receive () );
}
```

متد `Close` سرویس گیرنده را متوقف می‌سازد. آن ریسمان دریافت کننده را متوقف می‌سازد و متد `Close` مربوط به نظیر UDP آن را فراخوانی می‌کند (قطعه کد ۲۶-۵۸).

قطعه کد ۲۶-۵۸

```
public void Close ()
{
    this.clientThread.Abort ();
    this.clientThread.Join ();
    this.peer.Close ();
}
```

تا اینجا، سرویس گیرنده چندبخشی UDP ارائه شده است. حال، واسط کاربری سرویس گیرنده را بررسی می‌کنیم. واسط کاربری از کلاس `System.Windows.Forms.Form` مشتق می‌شود و `UDPNewsClient` نامیده می‌شود و در فایل `UDPNewsClient.cs` قرار دارد. آن یک کادر متنی ساده دارد. این کلاس یک سازنده و چهار متد دارد. در سازنده

مقداردهی اولیه برنامه کاربردی انجام می‌شود. علاوه بر این، یک متد اداره کننده رویداد بنام OnClosed() برای رویداد Closed() ثبت می‌شود. نهایتاً متدهای RunTicker() برای انتقال دادن کاراکترهای کادر متنی به چپ و متد SetNews() نماینده Notify() را پیاده‌سازی می‌کند و ریسمان گوش کننده UDPMultiCastClient برای بروزکردن اخبار کادر متنی استفاده می‌شود. ابتدا فیلدهای کلاس را در قطعه کد ۲۶-۵۹ ملاحظه کنید.

قطعه کد ۲۶-۵۹

```
// multicast group IP address
private const string GROUP_IP = "225.0.0.1";
// multicast group port
private const int GROUP_PORT = 8081;
// communication interface
private UDPMulticastClient client = null;
// ticker thread
private Thread tickerThread = null;
// new messages
private TextBox text = null;
// default news displayed at the beginning
private string news = "Please wait...";
```

سازنده کادر متنی، اداره کننده رویداد، نظیر UDP و ریسمان ticker را مقداردهی می‌کند. قطعه کد ۲۶-۶۰ سازنده را بدون مقداردهی اولیه کادر متنی نشان می‌دهد.

قطعه کد ۲۶-۶۰

```
public UDPNewsClient ()
{
    // initialize UI
    // add an event listener for close-event
    this.Closed += new System.EventHandler ( OnClosed );
    // start communication thread
    this.client = new UDPMulticastClient ( GROUP_IP,
    GROUP_PORT, new Notify ( SetNews ) );
    // start ticker thread
    this.tickerThread = new Thread (
    new ThreadStart ( RunTicker ) );
    this.tickerThread.Start ();
    Console.WriteLine ( "initialization complete" );
}
```

متد توقف سرویس گیرنده اخبار بوسیله رویداد Closed فراخوانی می‌شود (قطعه کد ۲۶-۶۱). آن سرویس گیرنده را بسته و ریسمان ticker را متوقف می‌سازد.

قطعه کد ۲۶-۶۱

```
public void OnClosed ( Object sender, EventArgs e )
{
    Console.WriteLine ( "client shut down" );
    this.client.Close ();
    this.tickerThread.Abort ();
    this.tickerThread.Join ();
    Application.Exit ();
}
```

ریسمان ticker هر ۵۰۰ میلی‌ثانیه یک کارکتر را به سمت چپ انتقال می‌دهد و کارکتر سمت چپ را حذف می‌کند. پیاده‌سازی آن زیاد هوشمند نیست، اما برای شبیه‌سازی کافی است. متد Notify() را نشان می‌دهد. آن پیام دریافتی بوسیله سرویس گیرنده چندپخشی را در متغیر اخبار قرار می‌دهد.

قطعه کد ۲۶-۶۲

```
public void RunTicker ()
{
    // initialize the textbox with the default text
    this.text.Text = " -+--+ " + this.news + " -+--+ " +
    this.news + " -+--+ ";
    while ( true )
    {
        string data = this.news + " -+--+ ";
        // repeat as long as there are characters in the data string
        while ( !data.Equals ( " " ) )
        {
```



```
// wait 500 milliseconds
Thread.Sleep ( 500 );
// remove the first character from the text field and add the
// first character of the data string
this.text.Text = this.text.Text.Substring ( 1 ) +
data[0];
// remove the first character from the data string
data = data.Substring ( 1 );
}
}

// notification method, used by multicast client
public void SetNews ( string news )
{
this.news = news;
}
}
```

۲۶-۴-۴-کامپایل کردن و اجرای مثال

ابتدا فایل‌های مورد نظر را کامپایل کرده و UDPNewsServer.exe و UDPNewsClient.exe را ایجاد کنید. سپس آنها را اجرا کرده و تست کنید.

ابتدا شکل ۲۶-۶۳، سپس شکل ۲۶-۶۴ ظاهر می‌گردد و با تایپ یک پیام و کلیک روی set بعد از مدتی شکل ۲۶-۶۵ ظاهر خواهد شد.

شکل ۲۶-۶۳



شکل ۲۶-۶۴



شکل ۲۶-۶۵



برنامه نویسی با سوکت در C#

آنچه که در این فصل یاد خواهید گرفت:

- آشنایی با کلاس Socket
- برنامه نویسی سطح پایین تحت شبکه با سوکت های TCP و UDP
- همگام سازی ورودی و خروجی در شبکه
- برنامه نویسی انتقال داده ی غیر همگام در شبکه
- انتخاب سرویس گیرنده از استخر آنها

۲۷-۱- کلاس Socket چارچوب .NET

محیط کاری .NET دارای یک کلاس Socket است که WinSock را پیاده سازی می کند. چون TcpListener، TcpClient و UdpClient برای پیاده سازی خود از کلاس Socket استفاده می کنند، کلاس Socket حاوی تمام عملکردهای این کلاس ها و عملکردهای دیگر است. واسط Socket یک API کلی است که امکانات زیادی را فراهم کرده است و بعضی از آنها را مورد بررسی قرار می دهیم. در این بخش دو برنامه به نام های سرویس گیرنده TCP با سوکت، و سرویس دهنده TCP با سوکت را خواهیم نوشت. پس از بررسی این دو برنامه، کلاس Socket و چند نوع شمارشی را مورد بحث قرار خواهیم داد.

۲۷-۱-۱- سرویس گیرنده ی TCP با کلاس Socket

برای اینکه سرویس گیرنده TCP از کلاس Socket استفاده کند، مراحل زیر را انجام می دهد:

۱. فراخوانی سازنده ی Socket. سازنده ی سوکت، نوع آدرس، نوع سوکت و نوع پروتکل را مشخص می کند.
 ۲. فراخوانی متد Connect () کلاس Socket. این متد یک پارامتر IPAddress را دریافت می کند که سرویس دهنده ی مورد نظر را مشخص می کند.
 ۳. ارسال و دریافت داده ها. با استفاده از متدهای Send() و Receive() کلاس Socket.
 ۴. بستن سوکت. با استفاده از متد Close() کلاس Socket.
- پس از بررسی کلاس Socket و چند نوع شمارشی، برنامه سرویس گیرنده TCP و سپس برنامه سرویس دهنده آن را با سوکت خواهید دید.

کلاس Socket

کلاس Socket با توجه به API سوکت های WinSock ساخته شده است. استفاده از کلاس Socket به مراحل زیر نیاز دارد:

۱. ایجاد نمونه ای از Socket با سازنده سوکت.

۲. اگر Socket یک سرویس‌دهنده است، متد Bind() را فراخوانی کنید تا نقطه پایانی^۱ محلی را تعیین کند.
۳. اگر Socket یک سرویس‌گیرنده است، Connect() را فراخوانی کنید تا به نقطه پایانی راه دور متصل شوید.
۴. اگر Socket یک سرویس‌دهنده است، Listen() را فراخوانی کنید تا در انتظار اتصال‌های درخواستی بماند.
۵. با استفاده از متدهای Send() و Receive() داده‌ها را روی TCP ارسال می‌کند، یا با استفاده از متدهای SendTo() و ReceiveFrom() داده‌ها را برای UDP ارسال و دریافت می‌کند.
۶. فراخوانی ShutDown() برای غیرفعال کردن سوکت.
۷. فراخوانی Close() برای بستن سوکت.

سازنده

```
public Socket(AddressFamily, SocketType, ProtocolType);
```

نمونه جدیدی از کلاس Socket ایجاد می‌کند. هر پارامتر با کلاس شمارشی خاص خودش، یعنی AddressFamily، SocketType و ProtocolType مشخص می‌شود. برای هدفی که در این کتاب به دنبال آن هستیم، AddressFamily برابر با AddressFamily.InterNetwork، SocketType برابر با SocketType.Stream و برای UDP برابر با Dgram و ProtocolType برابر با TCP و برای UDP برابر با UDP خواهد بود.

متدها

```
public void Bind(EndPoint localEP);
```

نقطه پایانی محلی را به Socket مقید می‌کند. استثناهای SocketException، ArgumentException و ObjectDisposedException را صادر می‌کند.

```
public void Close();
```

اتصال سوکت را می‌بندد.

```
public void Connect(EndPoint remoteEP);
```

اتصال را به سرویس‌دهنده راه دور برقرار می‌کند.

```
public object GetSocket(SocketOptionLevel, SocketOptionName);  
public void GetSocketOption(SocketOptionLevel, SocketOptionName, byte[]);  
public byte[] GetSocketOption(SocketOptionLevel, SocketOptionName, int);
```

گزینه خاصی از Socket را در یک شی یا آرایه‌ای از بایت‌ها بر می‌گرداند. لیست کاملی از خواص SocketOptionLevel و SocketOptionName در ادامه آمده است. این متد استثناهای SocketException و ObjectDisposedException را صادر می‌کند.

```
public void Listen(int backlog);
```

حالت Socket را برای اداره کردن درخواست اتصال‌های TCP تغییر می‌دهد و آن را در صف قرار می‌دهد تا توسط برنامه پذیرفته شود. backlog حداکثر تعداد درخواست‌های اتصال را مشخص می‌کند که می‌توانند در صف قرار گیرند. مقدار معمولی آن ۵ است، ولی بر حسب سیستم تغییر می‌کند. استثناهای SocketException و ObjectDisposedException را صادر می‌کند.

```
public bool Poll(int microSeconds, SelectMode mode);
```

^۱ End point

وضعیت سوکت را بررسی می‌کند. پارامتر اول مدت زمان انتظار برای پاسخ را به میکروثانیه مشخص می‌کند. مقدار منفی، انسداد نامحدود را نشان می‌دهد. وضعیت بررسی شده به پارامتر شمارشی `SelectMode` بستگی دارد. `SelectMode.SelectRead` قابلیت خواندن و `SelectMode.SelectWrite` قابلیت نوشتن، و `SelectMode.SelectError` وجود خطاها را بررسی می‌کند.

```
public int Receive (byte[] buffer);
public int Receive (byte[] buffer, SocketFlags flags);
public int Receive (byte[] buffer,int length, SocketFlags flags);
public int Receive (byte[] buffer, int length, SocketFlags flags);
public int Receive(byte[] buffer,int offset, int length, Socket, SocketFlags flag);
```

داده‌ها را از سوکت گرفته و به پارامتر `buffer` وارد می‌کند. پارامترهای اختیاری آن شامل `SocketFlags`، یک مقدار صحیح برای تعیین تعداد بایت‌ها جهت دریافت و آفستی از بافر است. این متد، تعداد بایت‌های دریافتی را بر می‌گرداند. استثنای `ArgumentOutOfRangeException`، `ArgumentNullException` و `ObjectDisposedException` را صادر می‌کند.

```
public int ReceiveFrom (byte[] buffer, ref EndPoint remoteEp);
public int ReceiveFrom (byte[] buffer, SocketFlags flags, ref EndPoint remoteEp);
public int ReceiveFrom (byte[] buffer, int length, SocketFlags flags, ref EndPoint remoteEp);
public int ReceiveFrom (byte[] buffer, int offset, int length, SocketFlags flags, ref EndPoint localEp);
```

داده‌گرام UDP را گرفته، در پارامتر `buffer` قرار می‌دهد و مرجع `EndPoint` را با اطلاعات نقطه پایانی فرستنده ترکیب می‌کند. پارامترهای اختیاری شامل `SocketFlags`، مقدار صحیح برای تعیین تعداد بایت‌ها جهت دریافت و آفستی از بافر است. تعداد بایت‌های دریافت شده را بر می‌گرداند. توجه کنید که تفاوت مهمی بین بافر بایتی برای دریافت داده‌گرام با `Socket` و `UdpClient` وجود دارد. `UdpClient` مرجعی به بافر از پیش تخصیص یافته را بر می‌گرداند، در حالی که در کلاس `Socket` لازم است پارامتر `buffer` از قبل با اندازه مناسبی تخصیص یابد. اگر سعی شود تعداد بایت‌هایی بیش از اندازه تخصیص یافته برای بافر دریافت گردد، استثنای `SocketException` صادر می‌شود.

```
public static void Select(IList readableList,IList WriteableList,IList errorList, int microseconds);
```

برای تعیین وضعیت یک یا چند نمونه از `Socket` به کار می‌رود. این متد یک تا سه نوع ظرف `IList`^۱ را دریافت می‌کند که نمونه‌های `Socket` را نگهداری می‌کنند (لیست‌ها نباید تهی باشند). نوع بررسی‌هایی که باید انجام شوند، به موقعیت `IList` در لیست پارامترها بستگی دارد. `Socket`‌ها در `IList` اول برای قابلیت خواندن بررسی می‌شوند. `Socket`‌ها در `IList` دوم برای قابلیت نوشتن بررسی می‌شوند. `Socket`‌ها در `IList` سوم برای خطاها بررسی می‌شوند. پارامتر آخر، مدت زمان انتظار پاسخ را بر حسب میکروثانیه مشخص می‌کند. استثنای `ArgumentOutOfRangeException` و `SocketException` را صادر می‌کند.

```
public int Send(byte[] buffer);
public int Send(byte[] buffer,socketFlags flags);
public int Send(byte[] buffer,int length.socketFlags flags);
public int Send(byte[] buffer,int offset,int length, SocketFlags flags);
```

داده‌ها را از پارامتر `buffer` به سوکت می‌فرستد. پارامترهای اختیاری شامل `SocketFlags`، یک مقدار صحیح برای تعیین تعداد بایت‌ها جهت ارسال و آفستی از بافر است. تعداد بایت‌های ارسال شده را بر می‌گرداند.

```
public int SendTo (byte[] buffer, EndPoint remoteEp);
public int Sendto (byte[] buffer, SocketFlags flags,EndPoint remotEP);
public int SendTo (byte[] buffer,int length,SocketFlags flags,EndPoint remoteEP);
public int SendTo (byte[] buffer, int offset, int length,SocketFlags flags, EndPoint remotEP);
```

^۱ Container

بسته‌ی داده‌گرام UDP را که در پارامتر `buffer` مشخص شده است به نقطه پایانی خاصی ارسال می‌کند. پارامترهای اختیاری شامل `SocketFlags`، یک مقدار صحیح برای تعیین تعداد بایت‌ها جهت ارسال و آفست بافر است. تعداد بایت‌های ارسال شده را بر می‌گرداند.

```
public void SetSocketOption(SocketOptionLevel optionLevel
,SocketOptionName optionName, byte[] optionValue);
public void SetSocketOption(SocketOptionLevel optionLevel
,SocketOptionName optionName, int optionValue);
public void SetSocketOption(SocketOptionLevel optionLevel
,SocketOptionName optionName, object optionValue);
```

مقادیری را برای گزینه‌های `Socket` تعیین می‌کند

```
Public void Shutdown(SocketShutdown how);
```

ارسال و دریافت در سوکت را غیرفعال می‌کند. پارامتر آن، یک نوع شمارشی `SocketShutdown` است که نشان می‌دهد چه کاری باید غیرفعال شود (ارسال، دریافت، یا هر دو).

خصوصیات

```
public bool Connected{get;}
```

مقداری را بر می‌گرداند که مشخص می‌کند آیا `Socket` برای یک عمل I/O، به منبع راه دور متصل است یا خیر.

```
Public EndPoint LocalEndPoint{get;}
```

یک نقطه پایانی محلی را برمی‌گرداند که `Socket` برای ارسال داده به آن مقید شده است.

```
Public EndPoint RemoteEndPoint{get;}
```

نقطه پایانی راه دور را می‌گیرد که سوکت برای ارتباط از آن استفاده می‌کند.

کلاس شمارشی `SocketOptionLevel`

شرح:

کلاس شمارشی `SocketOptionLevel`، سطحی را تعریف می‌کند که گزینه سوکت باید به آن اعمال شود. `SocketOptionLevel` ورودی متدهای `Socket.SetSocketOption()` و `Socket.GetSocketOption()` است.

اعضا:

IP: گزینه‌های `Socket` که به سوکت های IP اعمال می‌شود.

Socket: گزینه‌های `Socket` که به خود سوکت اعمال می‌شود.

Tcp: گزینه‌های `Socket` که به سوکت های TCP اعمال می‌شود.

Udp: گزینه‌های `Socket` که به سوکت های UDP اعمال می‌شود.

کلاس شمارشی `SocketOptionName`

شرح:

کلاس شمارشی `SocketOptionLevel` اسامی گزینه‌های سوکت را برای کلاس `Socket` تعریف می‌کند و به عنوان ورودی به متدهای `Socket.SetSocketOption()` و `Socket.GetSocketOption()` ارسال می‌گردد.

اعضا:

لیستی از گزینه‌های سوکت .NET، در جدول ۲۷-۱ آمده است.

کلاس شمارشی SocketFlags

شرح:

کلاس شمارشی SocketFlags مقادیر معتبری را برای پرچم‌های پیشرفته سوکت فراهم می‌کند و یک ورودی اختیاری به متدهای انتقال داده در کلاس Socket است. اگر لازم باشد، از متدی از کلاس Socket استفاده کنید که به پارامتر SocketFlags نیاز دارد، ولی اگر به هیچ مجموعه‌ای از پرچم‌ها نیاز نداشته باشد، از SocketFlags.None استفاده کنید.

اعضا:

DontRoute: ارسال بدون استفاده از جدول‌های مسیریابی.

MaxIOVectorLength: یک مقدار استاندارد را برای تعداد ساختمان‌های WSABUF فراهم می‌کند که برای ارسال و دریافت داده به کار می‌رود.

None: برای این تماس از هیچ پرچمی استفاده نمی‌کند.

OutOfBand: داده‌های out-of-band را پردازش می‌کند.

Partial: ارسال و دریافت جزئی پیام

Peek: پیام ورودی را دریافت می‌کند.

جدول ۱-۳ گزینه های سوکت.

SocketOptionName	نوع	مقدار	شرح
SocketOptionLevel			
AcceptConnection	Boolean	۰، ۱	این سوکت متد Listen() را فراخوانی کرده است. فقط خواندنی.
Broadcast	Boolean	۰، ۱	پیام‌های همه بخشی مجاز است یا نه.
ExclusiveAddressUse	Boolean	۰، ۱	سوکت را برای دستیابی انحصاری مقید قادر می‌سازد.
KeepAlive	Boolean	۰، ۱	پیام keep alive فعال می‌شود.
MaxConnections	Int۳۲	حداکثر اندازه	حداکثر طول صف قابل پذیرش توسط Socket.Listen() را مشخص می‌کند.
OutOfBandInline	Boolean	۰، ۱	داده‌های out-of-band را در استریم داده عادی می‌گیرد.
ReceiveBuffer	Int۳۲	بایت‌ها	تعداد بایت‌ها در بافر دریافت.
ReceiveLowWater	Int۳۲	بایت‌ها	حداقل بایت‌هایی که منجر به خاتمه Receive می‌شوند.
ReceiveTimeOut	Int۳۲	میلی ثانیه	مهلت دریافت
SendLowWater	Int۳۲	بایت‌ها	حداقل بایت‌هایی که باید ارسال شوند.

SendTimeout	Int۳۲	میلی ثانیه	مهلت ارسال.
Type	Int۳۲	SocketType	گرفتن نوع سوکت.
SocketOptionLevel.Tcp			
BsdUrgent	Boolean	۰،۱	داده های اضطراری.
Expedited	Boolean	۰،۱	داده های تسریع شده.
NoDelay	Boolean	۰،۱	عدم اجازه تاخیر در ادغام داده‌ها.
SocketOptionName	نوع	مقدار	شرح
SocketOptionLevel.Udp			
ChecksumCoverage	Boolean	۰،۱	تعیین پوشش جمع کنترلی.
NoChecksum	Boolean	۰،۱	داده‌گرام‌های UDP با جمع کنترلی صفر ارسال شدند.
SocketOptionLevel.IP			
AddMembership	MulticastOption	Group address, interface	اضافه کردن عضو گروه چند پخش.
AddSourceMembership	IPAddress	Group address	الحاق کردن گروه مبدا چند پخش.
BlockSource	Boolean	۰،۱	مسدود کردن داده‌ها از مبدا چند پخش.
DropMembership	MulticastOption	Group address, interface	حذف عنصر گروه چند پخش.
DropSourceMembership	IPAddress	Group address	حذف گروه مبدا چند پخش.
IpTimeToLive	Int۳۲	۰-۲۵۵	تعیین فیلد طول عمر در سرآیند IP.
UnblockSource	Boolean	۰،۱	خارج کردن چند پخش مسدود قبلی از انسداد.
UseLoopBack	Boolean	۰،۱	در صورت لزوم عبور از سخت افزار.

کلاس شمارشی SocketException

شرح:

این کلاس، زیر کلاس Exception است که در صورت بروز خطای سوکت، استثنایی را صادر می‌کند.

خصوصیات:

```
public override int ErrorCode{get;}
```

خاصیت ErrorCode حاوی شماره خطایی است که اتفاق افتاده است. چون استثنای SocketException به دلایل زیادی رخ می‌دهد، این خاصیت می‌تواند مفید باشد. زیرا با استفاده از کد خطا می‌توان تشخیص داد که چه وضعیتی رخ داده است تا آن وضعیت را اداره کرد. شماره خطا متناظر با کدهای خطا در WinSock ۲ (پیاده‌سازی سوکت در ویندوز) است.

```
Public virtual string Message {get;}
```

حاوی توصیف متنی از خطایی است که رخ داده است.

مثال برنامه سرویس‌گیرنده‌ی TCP

کد برنامه‌ی TcpEchoClientSockets.cs

```

using System;
using System.Text;
using System.IO;
using System.Net.Sockets;
using System.Net;

class TcpEchoClientSocket{
    static void Main(string[] args){
        if((args.Length<۲)|| (args.Length>۳)){//Test for cortrect # of args
            throw new ArgumentException("Parameters: <server>  <Word>  [<Port>]");
        }
        String server=args[۰]; //Server name or IP address
        //Convert input String to bytes
        byte[] byteBuffer=Encoding.ASCII.GetBytes(args[۱]);
        //Use port argument if supplied, otherwise default to ۷
        int servPort=(args.Length==۳)? Int۳۲.Parse(args[۲]):۷;
        Socket sock=null;
        try{
            //Create a TCPsocket instance
            sock=new Socket(AddressFamily.InterNetwork,SocketType.Stream,ProtocolType.Tcp);
            //Creates server IPEndPoint instance. We assume Resolve returns
            //at least one address
            IPEndPoint serverEndPoint=new IPEndPoint(Dns.Resolve(server).AddressList[۰],servPort);
            //Connect the socket to server on specified port
            sock.Connect(serverEndPoint);
            Console.WriteLine("Connected to server... sending echo string");
            //Send the encode string to server
            sock.Send(byteBuffer,۰,byteBuffer.Length,SocketFlags.None);
            Console.WriteLine("Sent {۰} bytes to server..",byteBuffer.Length);
            int totalBytesRcvd=۰; //Total bytes received so far
            int bytesRcvd=۰; //Bytes received in last read
            //Receiev the same string back from the server
            while(totalBytesRcvd<byteBuffer.Length){
                if((bytesRcvd=sock.Receive(byteBuffer,totalBytesRcvd,byteBuffer.Length-
                    totalBytesRcvd,SocketFlags.None))==۰){
                    Console.WriteLine("Connection closed prematurely");
                    Break;
                }
                totalBytesRcvd+=bytesRcvd;
            }
            Console.WriteLine("Received {۰} bytes from server:
            {۱}",totalBytesRcvd,Encoding.ASCII.GetString(byteBuffer,۰,totalBytesRcvd));
        } catch(Exception e){
            Console.WriteLine(e.Message);
        }finally{
            sock.Close();
        }
    }
}

```

۲۷-۱-۲- سرویس‌دهنده با کلاس Socket

برای سرویس‌دهنده‌ی TCP که از کلاس Socket استفاده می‌کند، مراحل زیر باید انجام گیرد.

۱. فراخوانی سازنده‌ی کلاس Socket: سازنده، نوع آدرس، نوع سوکت، و نوع پروتکل را مشخص می‌کند.

۲. فراخوانی متد Bind() کلاس Socket: متد Bind() سوکت را به آدرس محلی و پورت خاصی مقید می‌کند.

۳. فراخوانی متد Listen() کلاس Socket: متد Listen() یک پارامتر نوع صحیح را دریافت می‌کند که مشخص می‌کند چند اتصال می‌توانند در صف قرار گیرند، و برای درخواست اتصال‌ها (اتصال‌های ورودی) گوش فرا می‌دهد.

۴. به طور تکراری مراحل زیر را انجام می‌دهد:

- فراخوانی متد Accept() مربوط به کلاس Socket برای پذیرش درخواست‌های اتصال: متد Accept() هیچ پارامتری ندارد و یک نمونه از Socket را بر می‌گرداند که نشان‌دهنده سوکت سرویس‌گیرنده راه دور است.
- دریافت و ارسال داده‌ها: با استفاده از متدهای Receive() و Send() داده‌ها را انتقال می‌دهد.
- بستن سوکت سرویس‌گیرنده: با استفاده از متد Close() کلاس Socket.

۵. بستن سوکت سرویس‌دهنده: با استفاده از متد Close() کلاس Socket.

مثال برنامه سرویس‌دهنده TCP

کد برنامه‌ی TcpEchoServerSocket.cs

```
using System; //For Console, Int32, ArgumentException, Environment
using System.Net; //For IPAddress
using System.Net.Sockets; //For TcpListener, TcpClient

class TcpEchoServerSocket{

private const int BUFSIZE=۳۲; //Size of receive buffer
private const int BACKLOG=۵; //Outstanding connection queue max size

static void Main(string[] args){
if(args.Length>۱)//Test for correct # of args
throw new ArgumentException("Parameters:[<Port>]");

int servPort=(args.Length==۱)? Int32.Parse(args[۰]):۷;
Socket server=null;
try{
//Create a socket to accept client connections
server=new Socket(AddressFamily.InterNetwork,SocketType.Stream,ProtocolType.Tcp);
server.Bind(new IPEndPoint(IPAddress.Any,servPort));
server.Listen(BACKLOG);
}catch (SocketException se){
Console.WriteLine(se.ErrorCode+": "+se.Message);
Environment.Exit(se.ErrorCode);
}

byte[] rcvBuffer=new byte[BUFSIZE];//Receive buffer
int bytesRcvd; //Received byte count

for(;;){//run forever, accepting and servicing connections
Socket client=null;
try{
client=server.Accept(); //Get client connection
Console.Write("Handling client at"+ client.RemoteEndPoint +"-");
//Receive until client closes connection, indicated by • return value
int totalBytesEchoed=۰;
while( (bytesRcvd=client.Receive(rcvBuffer,۰,rcvBuffer.Length,SocketFlags.None)) >۰){
client.Send(rcvBuffer,۰,bytesRcvd,SocketFlags.None);
totalBytesEchoed+=bytesRcvd;
```

```

}
Console.WriteLine("echoed {0} bytes.",totalBytesEchoed);
Client.Close();// Close the socket. We are done with this client

} catch(Exception e){
Console.WriteLine(e.Message);
Client.Close();
}
}
}
}
}
}

```

۲۷-۱-۳-گزینه‌های سوکت

کسانی که پروتکل TCP/IP را ایجاد کردند، وقت زیادی را صرف رفتارهای پیش‌فرض نمودند، به طوریکه اغلب کاربردها را ارضا می‌کنند. برای اطلاعات بیشتر به RFC های ۱۱۲۲ و ۱۱۲۳ مراجعه کنید. طراحان برای اغلب کاربردها، کار خوبی انجام داده‌اند. اما معمولاً برای همه کاربردها مناسب نیست. برای این وضعیت‌ها، سوکت‌ها اجازه می‌دهند بسیاری از رفتارهای پیش‌فرض آنها تغییر کند، که در گزینه‌های سوکت انجام می‌شود. در نمونه‌هایی از TcpListener و UdpClient، با رفتارهای پیش‌فرض سر و کار دارید. کلاس TcpClient دارای زیر مجموعه‌ای از گزینه‌ها است که از طریق خواص عمومی قابل دستیابی‌اند. (جدول ۲-۳)

جدول ۲-۳-گزینه‌های سوکت که از طریق خواص کلاس TcpClient قابل دستیابی‌اند

شرح	خاصیت
اطلاعاتی را در مورد زمان ماندن سوکت مشخص می‌کند.	LingerState
مقداری را مشخص می‌کند که وقتی بافرهای ارسال یا دریافت پر نیستند، تاخیر را غیرفعال می‌کند.	Nodelay
اندازه بافر دریافتی را مشخص می‌کند.	ReceiveBufferSize
مدت زمانی را مشخص می‌کند که TcpClient باید منتظر باشد تا داده‌ها را درخواست کند(وقتی عمل خواندن آغاز شده باشد).	ReceiveTimeout
اندازه بافر ارسالی را مشخص می‌کند.	SendBufferSize
مدت زمانی را مشخص می‌کند که TcpClient باید منتظر بماند تا عمل ارسال با موفقیت کامل شود.	SendTimeout

برای دستیابی به تمام گزینه‌های سوکت، باید از کلاس Socket استفاده کنید. متدهای GetSocketOption() و SetSocketOption() مربوط به کلاس Socket قابلیت‌های تعیین گزینه‌ها را فراهم می‌کنند. این متدها overload شده‌اند، تا انواع داده گزینه‌های مختلف را پوشش دهند، اما در همه موارد، نام گزینه سوکت و سطح گزینه سوکت را دریافت می‌کنند. نام گزینه سوکت، نامی است که باید مقدار آن تعیین شود. و مقادیر معتبر آن در کلاس SocketOptionName آمده است. لیست کاملی از مقادیر SocketOptionName در جدول ۱-۳ آمده است. بحث در مورد تمام این گزینه‌ها در این کتاب نمی‌گنجد. خوانندگان می‌توانند برای اطلاعات تکمیلی به سایت www.msdn.microsoft.com مراجعه کنند. SocketOptionLevel، حوزه‌ای از گزینه سوکت است که باید تعیین شود، مثل سطح سوکت، سطح TCP، یا سطح IP. مقادیر معتبر آن در کلاس SocketOptionLevel در جدول ۱-۳ آمده است.

تنها راهکار تعیین گزینه‌های سوکت برای کلاس‌های سطح بالاتر (غیر از آنچه که در خواص TcpClient آمده است)، دستیابی به Socket موردنظر با استفاده از یک خاصیت Protected است. چون این خاصیت Protected است، فقط توسط کلاس‌هایی قابل دستیابی است که از کلاس Socket مشتق می‌شوند.

در مورد نیاز به تعیین مهلت زمانی در فراخوانی متد Receive() جهت جلوگیری از اجرای نامتناهی در صورت عدم پاسخ سرویس دهنده UDP یا مفقود شدن بسته‌ها، گزینه SocketOptionName.ReceiveTimeout را مقداردهی کنید.

مثال ۳-۸ مراحل زیر را انجام می‌دهد:

۱. رشته echo را به سرویس‌دهنده ارسال می‌کند.

۲. روی متد Receive() تا ۳ ثانیه مسدود می‌شود. اگر پاسخی دریافت نشود و مهلت زمانی به اتمام برسد، حداکثر ۵ بار دوباره شروع به ارسال می‌کند.

۳. سرویس‌گیرنده را خاتمه می‌دهد.

چون حد مهلت زمانی فقط با کلاس Socket وجود دارد، دو انتخاب را در پیش رو داریم: کد برنامه را با استفاده از کلاس Socket بازنویسی کنیم، یا از کلاس UdpClient استفاده کنیم و هر وقت نیاز به تعیین مهلت زمانی بود، نمونه‌ای از کلاس Socket را بازیابی کنیم. چون خاصیت UdpClient.Client که اجازه دستیابی به نمونه‌ای از کلاس Socket را می‌دهد، یک خاصیت protected است، مستقیماً قابل دستیابی نیست، مگر این که کلاس مشتقی از UdpClient ایجاد شود. برای تشریح کاربرد کلاس Socket برای UDP، انتخاب اول را بر می‌گزینیم.

مثال سرویس‌گیرنده‌ی UDP

کد برنامه UdpEchoClientTimeoutSocket

```
using System; //For String, Int۳۲, Boolean, Console
using System.Text; //For Encoding
using System.Net; //For EndPoint, IPEndPoint
using System.Net.Sockets; //For Socket, SocketOptionName, SocketOptionLevel;

class UdpEchoClientTimeOut{
private const int TIMEOUT=۳۰۰۰; //Resend timeout (milliseconds)
private const int MAXTRIES=۵; //Maximum retransmissions

static void Main(string[] args){
if((args.Length<۲)|| (args.Length>۳)){//Test for cortrect # of args
throw new ArgumentException("Parameters: <server> <Word> [<Port>]");
}
String server=args[۰]; //Server name or IP address
//Use port argument if supplied, otherwise default to ۷
int servPort=(args.Length==۳)? Int۳۲.Parse(args[۲]):۷;

//Create socket that is connected to server on specified port
Socket sock=new
Socket(AddressFamily.InterNetwork,SocketType.Dgram,ProtocolType.Udp);

//Set the receive timeout for this socket
sock.SetSocketOption(SocketOptionLevel.Socket,SocketOptionName.ReceiveTimeOut,TIME
OUT);

IPEndPoint remoteIPEndPoint =new
IPEndPoint(Dns.Resolve(server).AddressList[۰],servPort);
```

```

EndPoint remoteEndPoint=(EndPoint) remoteIPEndPoint;
//Convert input string to a packet of bytes
byte[] sendPacket=Encoding.ASCII.GetBytes(args[1]);
byte[] rcvPacket=new byte[sendPacket.Length];

int tries=0; //Packets may be lost, so we have to keep trying
Boolean receiveResponse=false;
Sock.SendTo(sendPacket, remoteEndPoint);

Console.WriteLine("Sent {0} bytes to server ...",sendPacket.Length);

Try{
//Attempt echo reply receive
sock.ReceiveFrom(rcvPacket,ref remoteEndPoint);
receiveResponse=true;
}catch(SocketException se){
tries++;
if (se.ErrorCode==10060)//WSAETIMEDOUT: connection timed out
Console.WriteLine("Time out,{0} more tries...",(MAXTRIES-tries));
else //We encountered an error other than a timeout, output error
// message
Console.WriteLine(se.ErrorCode+": "+se.Message);
}
}while ((!receivedResponse) && (tries<MAXTRIES));

if(receivedResponse)
Console.WriteLine(Received {0} bytes from {1}:{2}",rcvPacket.Length,
remoteEndPoint, Encoding.ASCII.GetString(rcvdPacket,0,rcvPacket.Length));
else
Console.WriteLine("No response-giving up.");
Sock.Close();
}
}

```

۲۷-۴-۱- پرچم‌های سوکت

کلاس شمارشی `SocketFlags` راه‌های دیگری را برای تغییر رفتار فراخوانی‌های `Send()` و `SendTo()` فراهم می‌کند. برای استفاده از پرچم‌های سوکت، پرچم مناسبی به متدهای `Send()` و `Receive()` ارسال می‌شود. گرچه پرداختن به این پرچم‌ها خارج از اهداف این کتاب است. مثالی را در مورد `SocketFlags.Peek` ارائه می‌کنیم.

`peek` به شما اجازه می‌دهد محتویات `Receive()` یا `ReceiveFrom()` را بدون خارج کردن نتایج بافر شبکه یا سیستم از صف، مشاهده کنید. معنایش این است که می‌توانید یک کپی از محتویات خواندن بعدی را ایجاد کنید، اما عمل خواندن بعدی همان بایت‌ها را مجدداً بر می‌گردانند. از نظر تئوری، این کار می‌تواند برای بررسی محتویات خواندن بعدی و تصمیم‌گیری کاربرد براساس دانش قبلی مفید باشد. در عمل، این وضعیت ناکارآمد است و همواره قابل اعتماد نیست. بنابراین بهتر است، ابتدا محتویات خوانده شود و سپس تصمیم گرفته شود که با آنها چه باید کرد. به هر حال، کد زیر می‌تواند چگونگی کار با `SocketFlags` را نشان می‌دهد:

```

,Socket s = new Socket (AddressFamily. InterNetwork, SocketType.Stream
;(ProtocolType.Tcp
Bind and/or Connect, create buffer//
.
.

```

```

.

Peek at the data without dequeuing it from the network buffer//

;(int len = s.Receive(buf, *, buf.Length, SocketFlags.Peek

This Receive will return (at least) the same data as the prior//

Receive, but this time it will be bequeued from the network buffer//

;(Len = s.Receive(buf, *, buf.Length, SocketFlags.None

```

۲۷-۱-۵ I/O بدون وقفه

فراخوانی‌های I/O ی سوکت ممکن است به چند دلیل مسدود شود. اگر داده‌ها نباشند، متدهای ورود داده‌ها مثل Read()، Receive() و ReceiveFrom() مسدود می‌شوند. اگر فضای کافی برای ذخیره (بافر) کردن داده‌های انتقالی نباشد، متدهای چاپ داده‌ها مثل Write()، Send() و SendTo() ممکن است مسدود شوند. متدهای Accept()، AcceptSocket() و AcceptTcpClient() مربوط به کلاس‌های Socket و TcpListener مسدود می‌شوند تا اتصال برقرار شود. به هر حال، زمان‌های رفت و برگشت طولانی، اتصال‌هایی با نرخ خطای زیاد، و سرویس‌دهنده‌های کند ممکن است موجب شوند بر قراری اتصال طول بکشد. در تمام این موارد، متد وقتی خاتمه می‌یابد که درخواست انجام شده باشد. البته، فراخوانی متد مسدود کننده، اجرای برنامه کاربردی را متوقف می‌کند. هنوز، کاربردهای معیوب را در طرف دیگر اتصال در نظر نگرفتیم.

اگر برنامه در حالی که منتظر کامل شدن فراخوانی متد است، کار دیگری برای انجام دادن داشته باشد، چه باید بکند؟ ممکن است این برنامه وقت کافی نداشته باشد که منتظر فراخوانی متد مسدود باشد. در مورد داده‌گرام‌های UDP مفقود چطور؟ خوشبختانه، راهکارهای مختلفی برای پرهیز از رفتارهای مسدود کننده‌ی ناخواسته وجود دارد. در این جا دو راهکار را بررسی می‌کنیم:

۱. بررسی وضعیت I/O

۲. فراخوانی‌های مسدود کننده با مهلت زمانی

جدول ۱-۵ این تکنیک‌ها را بر اساس نوع سوکتی که به کار گرفته می‌شود، نشان می‌دهد. در ادامه، روش سوم، به نام I/O ناهمگام را خواهید دید که در آن، فراخوانی I/O به جای مسدود شدن، فوراً خاتمه می‌یابد و توافق می‌کند که بعداً وقتی کامل شد، به شما خبر دهد.

جدول ۱-۵ راهکار اجتناب از انسداد.

عمل I/O	نوع سوکت	گزینه‌های اجتناب از مسدود شدن
پذیرش اتصال جدید	Socket	۱. قبل از فراخوانی Accept() سوکت را در وضعیت بدون انسداد قرار دهید. ۲. قبل از فراخوانی Accept متدهای Poll() یا Select() را فراخوانی کنید.
	TcpListener	۱. اگر Pending() مقدار true را بر می‌گرداند، AcceptSocket() یا AcceptTcpClient() را فراخوانی کنید.
ایجاد اتصال جدید	Socket	۱. قبل از فراخوانی Connect() سوکت را در حالت بدون انسداد قرار دهید. ۲. قبل از فراخوانی Connect متدهای Poll() یا Select() را فراخوانی کنید.
ارسال	Socket	۱. قبل از فراخوانی Send() یا SendTo() سوکت را در حالت بدون انسداد قرار دهید.

۲. قبل از فراخوانی Send() یا SendTo() متدهای Poll() یا Select() را فراخوانی کنید.	TcpClient	
۳. قبل از فراخوانی Send() یا SendTo() گزینه SendTimeout سوکت را مقداردهی کنید.		
۱. قبل از فراخوانی Write() روی استریم شبکه، خاصیت SendTimeout را مقدار دهید.		
۱. قبل از فراخوانی Receive() یا ReceiveFrom() سوکت را در حالت غیر انسداد قرار دهید.	Socket	دریافت
۲. قبل از فراخوانی Receive() یا ReceiveFrom() متدهای Poll() یا Select() را فراخوانی کنید.	TcpClient	
۳. قبل از فراخوانی Receive() یا ReceiveFrom() گزینه ReceiveTimeout سوکت را مقدار دهید.		
۴. اگر خاصیت >Available است، فقط Receive() یا ReceiveFrom() را مقدار دهید.		
۱. قبل از فراخوانی Read() روی استریم شبکه، خاصیت ReceiveTimeout را مقدار دهید.		
۲. اگر خاصیت DataAvailable برابر با True است، فقط Read() را روی استریم شبکه TcpClient فراخوانی کنید (خاصیت Length برای NetworkStream پشتیبانی شده است).		

۲۷-۱-۶- بررسی وضعیت I/O

یک روش اجتناب از رفتار مسدود شدن، عدم انجام فراخوانی است که منجر به انسداد می‌شود. این کار چگونه انجام می‌شود؟ برای بعضی از فراخوانی‌های I/O که می‌توانند مسدود شوند، ابتدا وضعیت I/O را بررسی می‌کنیم تا مشخص شود آیا I/O مسدود خواهد شد یا خیر. اگر این بررسی نشان دهد که فراخوانی مسدود نخواهد شد، می‌توانیم فراخوانی I/O را انجام دهیم و انتظار داریم که عمل فوراً کامل شود. اگر این بررسی نشان دهد که فراخوانی مسدود خواهد شد، می‌توان پردازش‌های دیگری را انجام داد و بعداً دوباره وضعیت I/O بررسی شود.

هنگام خواندن داده‌ها با TcpClient، این کار با بررسی خاصیت DataAvailable مربوط به NetworkStream وابسته به آن انجام می‌شود. اگر داده‌ای برای خواندن آماده باشد، مقدار true وگرنه مقدار false برگردانده خواهد شد:

```

; (TcpClient client = new TcpClient (server, port
; ) NetworkStream netstream = client. GetStream
:
} (If (netstream.DataAvailable
; (int len = netstream.Read(buf, 0, buf.Length
} else {
No data available, do other processing//

```

```
{  
    برای بررسی TcpListener، کافی است قبل از فراخوانی متدهای AcceptTcpClient() یا AcceptSocket()، متد  
    Pending() فراخوانی شود تا مشخص گردد آیا اتصال‌هایی معوق هستند یا خیر. اگر اتصال‌های معوقی وجود داشته باشند،  
    Pending() مقدار true وگرنه مقدار false را بر می‌گرداند:
```

```
;(TcpListener listener = new TcpListener(ipaddress , port  
;())Listener.Start  
.  
.  
}((If (listener.pending  
connections are pending,process them//  
;())TcpClient=listener.AcceptTcpClient  
.  
.  
}else{  
;("Console.WriteLine("no connections pending at this time  
{
```

با کلاس Socket، برای بررسی وضعیت I/O می‌توان از خاصیت Available استفاده کرد که از نوع int است. این خاصیت همیشه حاوی تعداد بایت‌هایی است که از شبکه دریافت شده، ولی هنوز خوانده نشدند. لذا، اگر Available بزرگ‌تر از صفر باشد، عمل خواندن مسدود نخواهد شد

```
,Socket sock=new Socket(AddressFamily.InterNetwork,SocketType.Stream  
;(ProtocolType.Tcp  
;(sock.Connect(serverEndPoint  
:  
{If(sock.Available>0  
we have data to read//  
;(sock.Receive(buf,buf.Length,0  
:  
}else{  
;("Console.WriteLine("no data available to read at this time  
{
```

متد Poll() کلاس Socket نیز امکان بررسی وضعیت I/O را فراهم می‌کند و در بخش بعد بحث می‌شود.

۲۷-۱-۷-فراخوانی‌های مسدود کننده با مهلت زمانی

در بخش قبل، چگونگی بررسی وضعیت I/O را قبل از انجام عمل I/O مطرح کردیم. اما گاهی ممکن است لازم باشد بدانیم که بعضی از رویدادهای I/O در مدت زمانی خاصی رخ نمی‌دهند. به عنوان مثال، در برنامه UdpEchoClientTimeOutSocket.cs دیدیم که در آن، سرویس گیرنده داده‌گرمی را به سرویس دهنده می‌فرستد و منتظر دریافت پاسخ می‌ماند. اگر داده‌گرام قبل از انقضای تایمر دریافت شود، ReceiveFrom از حالت انسداد خارج می‌شود تا سرویس گیرنده بتواند داده‌گرام مفقود را اداره کند. با بهره‌گیری از گزینه‌های سوکت، کلاس Socket می‌تواند حدی را برای

حداکثر زمان مسدود شدن روی ارسال و دریافت داده‌ها تعیین کند. این کار با خواص `SocketOption.SendTimeout` و `SocketOption.ReceiveTimeout` انجام می‌شود:

```
Socket sock=new Socket(AddressFamily.InterNetwork,SocketType.Stream,
ProtocolType.Tcp);
.
.
sock.SetSocketOption(SocketOptionLevel.Socket,
SocketOptionName.SendTimeout,۲۰۰۰);//set a ۲ second timeout on
send()/sendto()
```

اگر از کلاس `TcpClient` استفاده می‌کنید، این کلاس حاوی خواص `SendTimeout` و `ReceiveTimeout` است که می‌توانند تغییر کنند و بازایی شوند:

```
TcpClient client = new TcpClient(server,port);
.
.
Client.ReceiveTimeout = ۵۰۰۰;//set a ۵ second timeout on Read()
```

در هر مورد، اگر قبل از خاتمه فراخوانی متد، زمان مشخص شده به اتمام برسد، یک استثنای `SocketException` رخ می‌دهد که خاصیت `ErrorCode` آن برابر با ۱۰۰۶۰ (پایان مهلت اتصال) است.

متد `Poll()` کلاس `Socket` قابلیت‌های بیشتری دارد. `Poll()` دو گزینه را دریافت می‌کند: یک مقدار صحیح برحسب میکروثانیه که مدت انتظار برای دریافت پاسخ را مشخص می‌کند، و `mode` که مشخص می‌کند منتظر چه عملیاتی هستیم. زمان انتظار می‌تواند منفی باشد که در این صورت زمان انتظار نامتناهی را نشان می‌دهد. زمان انتظار می‌تواند صفر باشد، که اجازه می‌دهد `Poll()` برای بررسی پیشاپیش مورد استفاده قرار گیرد. پارامتر `mode` برابر با یکی از اعضای شمارشی `SelectMode` می‌شود که برحسب این که چه چیزی را بررسی می‌کنیم، می‌تواند `SelectRead`، `SelectWrite` یا `SelectError` باشد. اگر سوکت عملیات معوقی برای حالت درخواستی داشته باشد، `Poll()` مقدار `true` وگرنه مقدار `false` را بر می‌گرداند:

```
//Block for ۱ second waiting for data to read or incoming connections
If(sock.Poll(۱۰۰۰۰۰۰,SelectMode.SelectRead)){
//Socket has data to read or an incoming connection
}else{
//no data to read or incoming connections
}
```

به طور کلی، نظرسنجی بسیار ناکارآمد است، زیرا برای بررسی وضعیت I/O به طور مکرر فراخوانی می‌شود. این عمل را گاهی انتظار مشغولی می‌گویند، زیرا دائماً وضعیت را بررسی می‌کند. راه‌های اجتناب از نظر سنجی در ادامه بررسی می‌شوند، مثل استفاده از متد `Select()` کلاس `Socket` که همزمان انسداد را روی چند سوکت امکان پذیر می‌سازد و I/O ناهمگام.

فراخوانی `Write()` و `Send()` مسدود می‌شود تا زمانی که آخرین بایت نوشته شده، در بافر محلی پیاده‌سازی TCP ذخیره گردد. اگر فضای خالی بافر کوچک‌تر از اندازه نوشته‌ها باشد، قبل از خاتمه فراخوانی، بخشی از داده‌ها باید با موفقیت به طرف دیگر اتصال منتقل شده باشند. لذا، هر پروتکلی که بخش زیادی از داده‌ها را روی نمونه سوکت می‌فرستد، می‌تواند به مدت نامتناهی مسدود شود.

ایجاد اتصال `Socket` در میزبان و پورت مشخص، مسدود می‌شود تا اینکه اتصال برقرار گردد، اتصال رد شود، یا مهلت زمانی اعمال شده توسط سیستم، فرا رسد. مهلت زمانی سیستم طولانی است (بر حسب دقیقه)، و #C ابزاری برای کم کردن آن ندارد.

فرض کنید می‌خواهید یک سرویس‌دهنده echo بنویسیم که برای سرویس‌دادن به هر سرویس‌گیرنده دارای یک محدودیت زمانی باشد. یعنی، با یک مهلت زمانی TIMELIMIT تعریف شده، سرویس‌دهنده را طوری پیاده‌سازی می‌کنیم که پس از TIMELIMIT میلی‌ثانیه، نمونه سرویس‌دهنده خاتمه می‌یابد.

یک روش این است که نمونه سرویس‌دهنده، زمان باقیمانده را نگهداری کند، و با استفاده از تنظیم‌های مهلت زمانی ارسال و دریافت که شرح آنها گذشت، اطمینان حاصل شود که خواندن‌ها و نوشتن‌ها، بیش از آن زمان مسدود نمی‌شوند.

مثال سرویس‌دهنده‌ی Echo با مهلت زمانی معین

لیست برنامه‌ی TcpEchoServerTimeout.cs

```
using System; //For Console, Int32, ArgumentException , Environment
using System.Net; //For IPAddress
using System.Net.Sockets; //For TcpListener, TcpClient
class TcpEchoServerTimeOut{
private const int BUFSIZE=۳۲; //Size of receive buffer
private const int BACKLOG=۵; //Outstanding connection queue max size
private const int TIMELIMIT=۱۰۰۰۰; //Default time limit (ms)

static void Main(string[] args){
if(args.Length>۱)//Test for correct # of args
throw new ArgumentException("Parameters:<Port>");

int servPort=(args.Length==۱)? Int32.Parse(args[۰]):۷;
Socket server=null;
try{
//Create a socket to accept client connections
server=new Socket(AddressFamily.InterNetwork,SocketType.Stream,ProtocolType.Tcp);
server.Bind(new IPEndPoint(IPAddress.Any,servPort));
server.Listen(BACKLOG);
}catch (SocketException se){
Console.WriteLine(se.ErrorCode+": "+se.Message);
Environment.Exit(se.ErrorCode);
}

byte[] rcvBuffer=new byte[BUFSIZE];//Receive buffer
int bytesRcvd; //Received byte count
int totalBytesEchoed=۰;
for(;;){//run forever, accepting and servicing connections
Socket client=null;
try{
client=server.Accept(); //Get client connection
DateTime starttime=DateTime.Now;

//Set the ReceiveTimeout
client.SetSocketOption(SocketOptionLevel.Socket,SocketOptionName.ReceiveTimeout,TIMELIMIT);

Console.WriteLine("Handling client at"+ client.RemoteEndPoint+"-");
//Receive until closes connection, indicated by ۰ return value
totalBytesEchoed=۰;
while ((bytesRcvd=client.Receive(rcvBuffer,۰,rcvBuffer.Length,SocketFlags.None))>۰) {
client.Send(rcvBuffer, ۰, SocketFlags.None);
totalBytesEchoed+=bytesRcvd;

//Check elapsed time
```

```

    TimeSpan elapsed=DateTime.Now-starttime;
    if (TIMELIMIT-elapsed.TotalMilliseconds<0){
    Console.WriteLine("Abortig client, timelimit"+ TIMELIMIT+ "ms exceeded, echoed "+
    totalBytesEchoed+ " bytes");
    client.Close();
    throw new SocketException(۱۰۰۶۰);
    }
    //Set the ReceiveTimeout
    client.SetSocketOption(SocketOptionLevel.Socket,
    SocketOptionName.ReceiveTimeout,int (TIMELIMIT-
    elapsed.TotalMilliseconds));
    }
    Console.WriteLine("echoed {۰} bytes.", totalBytesEchoed);
    client.Close();
    }catch(SocketException se){
    if(se.ErrorCode==۱۰۰۶۰){//WSAETIMEDOUT: Connection timed out
    Console.WriteLine("Aborting client, timelimit"+ TIMELIMIT+ "ms exceeded, echoed" +
    totalBytesEchoed+ "bytes");
    } else{
    Console.WriteLine(se.ErrorCode+ ":" +se.Message);
    }
    client.Close();
    }
    }
    }
    }
    }

```

۲۷-۱-۸-تسهیم‌سازی

متد Select() کلاس Socket

برنامه‌هایی که تاکنون نوشتیم با I/O روی کانال یکتا سروکار داشتند. هر یک از نسخه‌های مربوط به سرویس‌دهنده echo، در هر زمان فقط با یک اتصال سرویس‌گیرنده سروکار داشتند. اما، گاهی لازم است که یک برنامه کاربردی قادر باشد I/O را همزمان روی چندین کانال انجام دهد. به عنوان مثال، ممکن است بخواهیم یک سرویس echo را همزمان روی چندین پورت تدارک ببینیم. مشکل این کار وقتی مشخص می‌شود که ببینید پس از اینکه سرویس‌دهنده سوکتی را ایجاد و به هر پورت مقید می‌کند، چه اتفاقی خواهد افتاد. سرویس‌دهنده آماده پذیرش اتصال است (متد Accept())، اما کدام سوکت باید انتخاب شود؟ فراخوانی Accept() یا Receive() روی یک سوکت ممکن است مسدود شود، و به این ترتیب، اتصالاتی که با سوکت‌های دیگر برقرار شده است، به طور غیرضروری منتظر می‌مانند. این مسئله نمی‌تواند با استفاده از سوکت‌های بدون انسداد حل شود، اما در این مورد سرویس‌دهنده به طور تکراری به نظرسنجی از سوکت‌ها می‌پردازد که اتلاف وقت است. علاقه‌مند هستیم به سرویس‌دهنده اجازه داده شود که مسدود گردد تا سوکتی آماده I/O شود.

خوشبختانه API سوکت راهی برای این کار تدارک دیده است. با استفاده از متد ایستای Select() مربوط به کلاس Socket، برنامه می‌تواند لیستی از سوکت‌ها را برای I/O‌های معوق بررسی کند. Select() برنامه را معوق می‌کند تا یک یا چند سوکت موجود در لیست، آماده‌ی انجام I/O شوند. لیست اصلاح می‌شود تا فقط نمونه‌هایی از Socket را در برگیرد که آماده انجام I/O هستند.

Select() چهار پارامتر دارد: سه پارامتر اول لیست‌هایی از نمونه کلاس Socket هستند و پارامتر چهارم، زمان بر حسب میکروثانیه است که مشخص می‌کند چه مدتی باید منتظر بماند. مقدار منفی این زمان، انتظار بینهایت را مشخص می‌کند. لیست‌های سوکت می‌توانند هر کلاسی باشند که واسط IList را پیاده‌سازی می‌کنند. لیست‌ها نشان می‌دهند که منتظر چه رویدادهایی هستیم. آنها به ترتیب، آمادگی خواندن، آمادگی نوشتن، و وجود خطا را بررسی می‌کنند. قبل از فراخوانی، لیست‌ها باید همراه با ارجاع‌هایی به نمونه‌های Socket باشند. وقتی فراخوانی کامل شد، لیست فقط شامل ارجاع‌های

Socket است که معیارهای آن لیست را برآورده می‌کند (قابلیت خواندن، قابلیت نوشتن، یا وجود خطا). اگر نمی‌خواهید تمام این وضعیت‌ها را در یک Select() بررسی کنید، می‌توانید دو لیست را تهی ارسال کنید.

اکنون مسئله اجرای سرویس echo را روی چندین پورت در نظر می‌گیریم. اگر برای هر پورت یک سوکت ایجاد کنیم، می‌توانیم آن سوکت را در ArrayList قرار دهیم. فراخوانی Select() با چنین لیستی، برنامه را به تعویق می‌اندازد تا یک درخواست echo برای حداقل یکی از سوکت‌ها فرا رسد. از آن پس می‌توانیم اتصال و echo را برای آن سوکت خاص تنظیم کنیم. مثال بعدی این مدل را پیاده‌سازی می‌کند.

مثال سرویس‌دهنده‌ی چند پورته‌ی

لیست برنامه‌ی TcpEchoServerSelectSocket

```
using System; //For Console, Int32, ArgumentException, Environment
using System.Net; //For IPAddress
using System.Collections; //For ArrayList
using System.Net.Sockets; //For Socket, SocketException

class TcpEchoServerSelectSocket{
private const int BUFSIZE=۳۲; //Size of receive buffer
private const int BACKLOG=۵; //Outstanding Connection queue max size
private const int SERVER۱_PORT=۸۰۸۰; //Port for first server
private const int SERVER۲_PORT=۸۰۸۱; //Port for second server
private const int SERVER۳_PORT=۸۰۸۲; //Port for third server
private const int SELECT_WAIT_TIME=۱۰۰۰; //Microsecond for Select() to wait

static void Main(string[] args){

Socket server۱=null;
Socket server۲=null;
Socket server۳=null;

Try{
//Create a socket to accept client connections
server۱=new Socket(AddressFamily.InterNetwork,SocketType.Stream, ProtocolType.Tcp);
server۲=new Socket(AddressFamily.InterNetwork,SocketType.Stream,
ProtocolType.Tcp);
server۳=new Socket(AddressFamily.InterNetwork,SocketType.Stream,
ProtocolType.Tcp);

server۱.Bind(new IPEndPoint(IPAddress.Any,SERVER۱_PORT));
server۲.Bind(new IPEndPoint(IPAddress.Any,SERVER۲_PORT));
server۳.Bind(new IPEndPoint(IPAddress.Any,SERVER۳_PORT));

server۱.Listen(BACKLOG);
server۲.Listen(BACKLOG);
server۳.Listen(BACKLOG);
}catch(SocketException se){
Console.WriteLine(se.ErrorCode+ ":"+ se.Message);
Environment.Exit(se.ErrorCode);
}
byte[] rcvBuffer=new byte[BUFSIZE]; // Receive buffer
int bytesRcvd; //Received byte count
```

```

for(;;){ //Run forever, accepting and servicing connections

Socket client=null;
//Create an array list of all three sockets
ArrayList acceptList= new ArrayList();
acceptList.Add(server۱);
acceptList.Add(server۲);
acceptList.Add(server۳);
try{
//The Select call will check readable status of each socket
//in the list
Socket.Select(acceptList, null, null, SELECT_WAIT_TIME);

//The acceptList will now contain Only the server sockets with
//pending connections
for(int i=۰;i<acceptList.Count;i++)
client=((Socket) acceptList[i]).Accept(); //Get client connection

IPEndPoint localEP=(IPEndPoint)((Socket) acceptList[i]).LocalEndPoint;
Console.WriteLine("Server port" +localEP.Port);
Console.WriteLine("- handling client at" + client.RemoteEndPoint+ "-");

//Receive until client closes connection, indicated by * return value
int totalBytesEchoed=۰;
while((bytesRcvd=client.Receive(rcvBuffer, *, rcvBuffer.Length,
    SocketFlags.None))>۰){
client.Send(rcvBuffer, *, bytesRcvd, SocketFlags.None);
totalBytesEchoed+=bytesRcvd;
}
Console.WriteLine("Echoed {۰} bytes.", totalBytesEchoed);
client.Close(); //Close the socket.
}
}catch(Exception e){
Console.WriteLine(e.Message);
Client.Close();
}
}
}
}

```

۲۷-۱-۹-۱ I/O ناهمگام

چارچوب .NET تعداد زیادی از متدها را برای برنامه‌نویسی شبکه فراهم کرده است که به طور ناهمگام اجرا می‌شوند. به این ترتیب، وقتی متد I/O منتظر است تا از حالت انسداد خارج شود، اجرای کد فراخوان ادامه می‌یابد. آنچه که اتفاق می‌افتد این است که متد ناهمگام در نخ خود اجرا می‌شود، با این تفاوت که جزئیات تنظیم نخ، ارسال داده‌ها و شروع نخ برای شما انجام می‌گیرد. کد فراخوان با سه گزینه می‌تواند زمان کامل شدن I/O را تعیین کند: ۱. می‌تواند یک متد callback را مشخص کند که پس از کامل شدن I/O فراخوانی شود، ۲. می‌تواند به طور دوره‌ای نظرسنجی کند تا ببیند آیا متد کامل شده است یا خیر، و ۳. پس از اینکه کارهای ناهمگام خود را تمام کرد، می‌تواند مسدود شود و منتظر کامل شدن بماند.

فراخوانی I/O ناهمگام دو قسمت دارد: فراخوانی شروع که برای آغاز عملیات به کار می‌رود، و فراخوانی پایانی که برای بازبایی نتایج فراخوانی پس از کامل شدن آن استفاده می‌شود. فراخوانی شروع، از متد همانام با متد مسدود شده استفاده می‌کند که واژه Begin قبل از نام آن قرار دارد. به همین ترتیب، فراخوانی پایان، از همان متد مسدود شده استفاده می‌کند که واژه End قبل از نام آن قرار دارد. عمل شروع و پایان، متقارن هستند، و هر فراخوانی به متد شروع، باید با یک فراخوانی

متد پایان تطبیق کند. عدم انجام این کار در برنامه‌هایی با زمان اجرای طولانی، مستلزم نگهداری متغیرهای حالت زیادی است و در نتیجه حافظه مصرف می‌شود.

مثال‌های دقیق‌تری را در نظر می‌گیریم. کلاس `NetworkStream` حاوی نسخه‌های ناهمگام متدهای `Read()` و `Write()` است که به صورت `BeginWrite()`، `BeginRead()`، `EndWrite()` و `EndRead()` پیاده‌سازی می‌شوند. جزئیات آنها را بررسی می‌کنیم.

متدهای `BeginWrite()` و `BeginRead()` دارای دو پارامتر دیگر و نوع برگشتی متفاوتی‌اند:

```
public override IAsyncResult BeginRead(byte[] buffer, int offset, int size, AsyncCallback
callback, object state);
public override IAsyncResult BeginWrite(byte[] buffer, int offset, int count, AsyncCallback
callback, object state);
```

دو پارامتر اضافی عبارت‌اند از نمونه‌ای از کلاس `AsyncCallback` و نمونه‌ای از `object` که می‌تواند هر نمونه از کلاس `C#` باشد. کلاس `AsyncCallback` یک `Delegate` است که یک متد `callback` را مشخص می‌کند که پس از کامل شدن گزینه ناهمگام، فراخوانی می‌شود. برای نمونه‌سازی این کلاس، نام متد `callback` به آن ارسال می‌شود:

```
AsyncCallback ac = new AsyncCallback(my methodToVCall);
:
public static void myMothodToCall(IAsyncResult result){
// callback code goes here
}
```

اگر نیاز به متد `callback` نباشد، این پارامتر می‌تواند تهی باشد (به یاد داشته باشید که متد پایانی باید در جایی فراخوانی شود). خود متد `callback` باید با نشانه‌ی زیر باشد:

```
public static void <callbackMethodName>(IAsyncResult)
```

کلاس `IAsyncResult` در ادامه بحث خواهد شد.

پارامتر `object` راهی برای حمل اطلاعات تعریف شده توسط کاربر، از فراخوان به `callback` است. این اطلاعات می‌تواند خود نمونه‌ی کلاس سوکت یا `NetworkStream` باشد، یا می‌تواند کلاس تعریف شده توسط کاربر باشد که شامل `NetworkStream`، بافر مورد استفاده، و هر چیزی است که متد `callback` برنامه کاربردی باید به آنها دسترسی داشته باشد.

انواع برگشتی متدهای `BeginRead()` و `BeginWrite()` نمونه‌ای از `IAsyncResult` است. وضعیت عملیات ناهمگام را نشان می‌دهد و می‌تواند برای نظرسنجی یا مسدود شدن روی برگشت آن عملیات به کار رود. اگر تصمیم به انسداد گرفتید، منتظر تکمیل شدن عملیات باشید، خاصیت `AsyncWaitHandle` مربوط به `IAsyncResult` حاوی متدی به نام `WaitOne()` است. با فراخوانی این متد، منجر به انسداد می‌شود تا زمانی که متد پایانی متناظر آن فراخوانی شود.

پس از کامل شدن عملیات ناهمگام، متد `callback` فراخوانی می‌شود. متد `callback` نمونه‌ای از `IAsyncResult` را به عنوان پارامتر دریافت می‌کند که دارای خاصیتی به نام `AsyncState` است که حاوی یک شی خواهد بود. این شی همان شیئی است که به متد شروع ارسال شده است، و قبل از به کارگیری باید به نوع اصلی خود تبدیل شود. نمونه‌ی `IAsyncResult` نیز به عنوان پارامتری در فراخوانی پایانی به کار می‌رود. فراخوانی پایان، تقارن فراخوانی را کامل می‌کند و نتایج فراخوانی را بر می‌گرداند. نتیجه‌ی آن دقیقاً همان مقداری است که نسخه‌ی همگام آن فراخوانی، بر می‌گرداند:

```
public override int EndRead(IAsyncResult asyncResult);
public override void EndWrite(IAsyncResult asyncResult);
```

به عنوان مثال، فرض کنید `BeginRead()` روی نمونه `NetworkStream` فراخوانی می‌شود، و علاوه بر پارامترهای معمولی، یک متد `(new AsyncCallback(my callback)) (callback)` و بافر خواندن به عنوان حالت به آن ارسال می‌شوند. فراخوانی `EndRead()` تعداد بایت‌های خوانده شده از `NetworkStream` را بر می‌گرداند که مشابه فراخوانی همگام متد `Read()` است:

```
public static void myCallback (IAsyncResult result){
```

```
byte[] buffer = (byte[]) result.AsyncState;
int bytesRead = EndRead(result);
Console.WriteLine ("Got {0} bytes of: {1}" ,bytesRead, buffer);
}
```

وقتی به تفاوت بین نسخه‌های همگام و ناهمگام متدها پی بردید، آماده بررسی کل API ناهمگام در چارچوب .NET هستید. به طور خلاصه، شامل موارد زیر است:

۱. متد شروع: فراخوانی شروع، علاوه بر متدهای نسخه همگام، نمونه‌ای از AsyncCallback را می‌گیرد که متد callback را مشخص می‌کند، و نمونه‌ای از یک شی را می‌گیرد که حاوی حالت‌های تعریف شده توسط کاربر است. فراخوانی شروع، یک نمونه از AsyncState را بر می‌گرداند که می‌تواند برای نظرسنجی یا انسداد روی برگشت فراخوانی به کار رود.

۲. حالت callback: حالت (پارامتر شی فراخوانی شروع)، که در خاصیت AsyncState مربوط به نمونه IAsyncResult ذخیره شده است، و به متد callback ارسال می‌شود.

۳. متد پایان: فراخوانی متد پایانی، نمونه IAsyncResult را که با فراخوانی متد callback برگردانده شده است، به عنوان پارامتر می‌پذیرد و مقداری را بر می‌گرداند که نسخه همگام آن متد بر می‌گرداند.

جدول ۲-۵ کلاس‌هایی از .NET، را نشان می‌دهد که در این کتاب به کار گرفته شدند و حاوی متدهای ناهمگام هستند.

جدول ۲-۵ چند متد ناهمگام در .NET.

متد ناهمگام	کلاس
BeginGetHostByName() / EndGetHostByName ()BeginResolve() / EndResolve	Dns
()BeginRead() / EndRead ()BeginWrite() / EndWrite	FileStream
()BeginRead() / EndRead ()BeginWrite() / EndWrite	NetworkStream
()BeginAccept() / EndAccept BeginConnect() / EndConnect ()BeginReceive() / EndReceive ()BeginReceiveFrom() / EndReciveFrom BeginSend() / EndSend BeginSendTi() / EndSendTo	Socket
()BeginRead() / EndRead ()BeginWrite() / EndWrite	Stream

اکنون زمان آن رسیده است که مثال‌هایی را ارائه کنیم. در ادامه، نسخه‌هایی از TcpEchoServer و TcpEchoClient را با استفاده از API ناهمگام پیاده‌سازی می‌کنیم. در هر یک از دو برنامه سرویس‌دهنده و سرویس‌گیرنده، فرض می‌کنیم این دو برنامه عملیات دیگری دارند که در هنگام انسداد در فراخوانی‌های مختلف شبکه، باید انجام شوند. برای شبیه‌سازی این وضعیت، متد doOtherStuff() را اضافه کردیم که حلقه‌ای را ۵ بار اجرا می‌کند، خروجی را چاپ کرده و به خواب می‌رود.

همچنین مشاهده خواهید کرد که تعداد متدهای ناهمگام تعریف شده برای کلاس Socket خیلی بیشتر از چیزی است که برای NetworkStream تعریف شده‌اند. برای تشریح مقایسه آنها، سرویس‌گیرنده echo از کلاس TcpClient با یک NetworkStream، و سرویس‌دهنده echo از کلاس Socket استفاده می‌کند.

مثال سرویس‌گیرنده‌ی ناهمگام

لیست برنامه‌ی TcpEchoClientAsync

```
using System; //For String, IAsyncResult, ArgumentException
using System.Text; //For Encoding
using System.Net.Sockets; //For TcpClient, NetworkStream
using System.Threading; //For ManualResetEvent
class ClientState{
//Object to contain client state, including network stream
//and the send/recv buffer
private byte[] byteBuffer;
private NetworkStream netStream;
private StringBuilder echoResponse;
private int totalBytesRcvd=0; //Total bytes received so far
public ClientState(NetworkStream netStream, byte[] byteBuffer){
this.netStream=netStream;
this.byteBuffer=byteBuffer;
echoResponse=new StringBuilder();
}
public NetworkStream NetStream{
get{
return netStream;
}
}
public byte[] ByteBuffer{
set{
byteBuffer=value;
}
get{
return byteBuffer;
}
}
public void AppendResponse(String Response){
echoResponse.Append(response);
}
public String EchoResponse{
get{
return echoResponse.ToString();
}
}
public void AddToTotalBytes(int count){
totalBytesRcvd+=count;
}
public int TotalBytes{
get{
return totalBytesRcvd;
}
}
}
class TcpEchoClientAsync{
//A manual event signal we will trigger when all reads are complete:
public static ManualResetEvent ReadDone=new ManualResetEvent(false);
static void Main(string[] args){
if((args.Length<2)|| (args.Length>2)){//Test for correct # of args
throw new ArgumentException("Parameters: <server> <Word> [<Port>]");
}
String server=args[0]; // server name or IP address
```

```

//Use port argument if supplied, otherwise default to ۷
int servPort=(args.Length==۲)? Int۲۲.Parse(args[۲]):۷;
Console.WriteLine("Thread{۰}({۱})-Main()",
Thread.CurrentThread.GetHashCode(),Thread.CurrentThread.ThreadState);
//Create TcpClient that is connected to server on specified port
TcpClient client=new TcpClient();
client.Connect(server,servPort);
Console.WriteLine("Thread{۰}({۱})-Main()",
Thread.CurrentThread.GetHashCode(),Thread.CurrentThread.ThreadState);
NetworkStream netStream=client.GetStream();
ClientState cs=new ClientState(netStream, Encoding.ASCII.GetBytes(args[۱]));
//send the encoded string to server
IAsyncResult result=netStream.BeginWrite(cs.ByteBuffer, ۰, cs.ByteBuffer.Length,
new AsyncCallback(WriteCallback), cs);
doOtherStuff();
result.AsyncWaitHandle.WaitOne(); //block until EndWrite is called
//Receive the same string back the server
result =netStream.BeginRead(cs.ByteBuffer, cs.TotalBytes, cs.ByteBuffer.Length-
cs.TotalBytes, new AsyncCallback(ReadCallback), cs);
doOtherStuff();
ReadDone.WaitOne(); //Block until ReadDone is manually set
netStream.Close(); //close the stream
client.Close(); // close the socket
}
public static void doOtherStuff(){
for(int x=۱; x<=۵; x++){
Console.WriteLine("Thread{۰}({۱})-doOtherStuff():{۲}...",
Thread.CurrentThread.GetHashCode(),Thread.CurrentThread.ThreadState,x);
Thread.Sleep(۱۰۰۰);
}
}
public static void WriteCallback(IAsyncResult asyncResult){
ClientState cs=(ClientState) asyncResult.AsyncState;
Cs.NetStream.EndWrite(asyncResult);
Console.WriteLine("Thread{۰}({۱})-WriteCallback():{۲} bytes...",
Thread.CurrentThread.GetHashCode(),Thread.CurrentThread.ThreadState,
cs.ByteBuffer.Length);
}
public static void ReadCallBack(IAsyncResult asyncResult){
ClientState cs=(ClientState) asyncResult.AsyncState;
int bytesRcvd=cs.NetStream.EndRead(asyncResult);
cs.AddToTotalBytes(bytesRcvd);
cs.AppendResponse(Encoding.ASCII.GetString(cs.ByteBuffer, ۰, bytesRcvd));
if(cs.TotalBytes < cs.ByteBuffer.Length){
Console.WriteLine("Thread{۰}({۱})-ReadCallback():{۲} bytes...",
Thread.CurrentThread.GetHashCode(),Thread.CurrentThread.ThreadState, bytesRcvd);
Cs.NetStream.BeginRead(cs.ByteBuffer,cs.TotalBytes, cs.ByteBuffer.Length -
cs.TotalBytes, new AsyncCallback(ReadCallback), cs.NetStream);
}else{
Console.WriteLine("Thread{۰}({۱})-ReadCallback():{۲} total...",
Thread.CurrentThread.GetHashCode(),Thread.CurrentThread.ThreadState,
cs.TotalBytes,cs.EchoResponse);
ReadDone.Set(); //Signal read Complete event
}
}
}
}

```



```

using System; //For Console, IAsyncResult, ArgumentException
using System.Net; //For IPEndPoint
using System.Net.Sockets; //For Socket
using System.Threading; //For ManualResetEvent
class ClientState{
//Object to contain client state, including the client socket
//and the receive buffer
private const int BUFSIZE=۲۲; //size of receive buffer
private byte[] rcvBuffer;
private Socket clntSock;
public ClientState(Socket clntSock){
this.clntSock=clntSock;
rcvBuffer=new byte[BUFSIZE]; //Receive buffer
}
public byte[] RcvBuffer{
get{
return rcvBuffer;
}
}
public Socket ClntSock{
get{
return clntSock;
}
}
}
class TcpEchoServerAsync{
private const int BACKLOG=۵; //outstanding connection queue max size
static void Main(string[] args){
if(args.Length!=۱) //Test for correct number of args
throw new ArgumentException("Parameters: <Port>");
int servPort=Int32.Parse(args[0]);
//Create a socket to accept client connections
Socket servSock=new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
servSock.Bind(new IPEndPoint(IPAddress.Any, servPort));
servSock.Listen(BACKLOG);
for(;;){ //Run server, accepting and servicing connections
Console.WriteLine("Thread{0}({1})-Main():calling BeginAccept",
Thread.CurrentThread.GetHashCode(),Thread.CurrentThread.ThreadState);
IAsyncResult result=servSock.BeginAccept(new AsyncCallback(AcceptCallback), servSock);
doOtherStuff();
//Wait for the EndAccept before issuing a new Accept
result.AsyncWaitHandle.WaitOne();
}
}
public static void doOtherStuff(){
for(int x=۱; x<=۵; x++){
Console.WriteLine("Thread{0}({1})-doOtherStuff():{2}...",
Thread.CurrentThread.GetHashCode(),Thread.CurrentThread.ThreadState,x);
Thread.Sleep(۱۰۰۰);
}
}
public static void AcceptCallback(IAsyncResult asyncResult){
Socket servSock=(Socket) asyncResult.AsyncState;
Socket clntSock=null;
try{
clntSock=servSock.EndAccept(asyncResult);
Console.WriteLine("Thread{0}({1})- AcceptCallback():handling client at {2}",
Thread.CurrentThread.GetHashCode(), Thread.CurrentThread.ThreadState,
clntSock.RemoteEndPoint);
ClientState cs=new ClientState(clntSock);

```

```

clntSock.BeginReceive(cs.RcvBuffer, ۰, cs.RcvBuffer.Length, SocketFlags.None, new
AsyncCallback(ReceiveCallback), cs);
}catch( SocketException se){
Console.WriteLine( se.ErrorCode + ":" + se.Message);
clntSock.Close();
}
}
public static void ReceiveCallback(AsyncResult asyncResult){
ClientState cs=(ClientState) asyncResult.AsyncState;
try{
int recvMsgSize= cs.ClntSock.EndReceive(asyncResult);
if (recvMsgSize>۰){
Console.WriteLine("Thread{۰}({۱})- ReciveCallback():received {۲} bytes",
Thread.CurrentThread.GetHashCode(), Thread.CurrentThread.ThreadState,
recvMsgSize);
}else{
cs.clntSock.Close();
}
}catch( SocketException se) {
Console.WriteLine(se.ErrorCode+ ":" + se.Message);
cs.ClntSock.Close();
}
}
public static void SendCallback(IAsyncResult asyncResult){
ClientState cs=(ClientState) asyncResult.AsyncState;
try{
int bytesSent=cs.ClntSock.EndSend(asyncResult);
Console.WriteLine("Thread{۰}({۱})- SendCallback():send {۲} bytes",
Thread.CurrentThread.GetHashCode(), Thread.CurrentThread.ThreadState, bytesSent);
cs.ClntSock.BeginReceive(cs.RcvBuffer, ۰, cs.RcvBuffer.Length, SocketFlags.None,
new AsyncCallback(ReceiveCallback), cs);
}catch( SocketException se){
Console.WriteLine(se.ErrorCode+ ":" + se.Message);
Cs.ClntSock.Close();
}
}
}
}

```

فصل بیست و هشتم

صف‌بندی پیام

این فصل عناوین زیر را بررسی می‌کند:

- مروری بر ¹MQ
- معماری MQ
- ابزار مدیریت صف‌بندی پیام
- برنامه‌نویسی صف‌بندی پیام
- برنامه کاربردی تکلیف درس²

۲۸-۱-مقدمه

System.Messaging یک فضای نامی است که کلاس‌هایی برای خواندن و نوشتن پیغام با قابلیت MQ سیستم‌عامل ویندوز را در بر دارد. پیام‌رسانی می‌تواند در یک سناریوی ارتباط منفصل² بکار رود، جایی که لازم نیست سرویس‌دهنده و سرویس‌گیرنده همزمان در حال اجرا باشند.

قبل از وارد شدن به برنامه‌نویسی صف‌بندی پیام، این بخش مفاهیم اساسی پیام‌رسانی را مورد بحث قرار می‌دهد و آن را با برنامه‌نویسی همگام و غیرهمگام مقایسه می‌کند. در برنامه‌نویسی همگام زمانی که یک متد احضار می‌شود، فراخواننده باید منتظر بماند تا زمانی که متد فراخوانی شده خاتمه یابد. در برنامه‌نویسی غیرهمگام، ریسمان فراخوانی کننده بطور همزمان با متد فراخوانی شده اجرا می‌شود. برنامه‌نویسی غیرهمگام به وسیله‌ی نماینده‌ها، کتابخانه‌های کلاسی که متدهای غیرهمگام را پشتیبانی می‌کنند (System.IO , System.Net) یا با استفاده از ریسمان‌های سفارشی انجام‌پذیر است. در هر دو مورد برنامه‌نویسی همگام و غیرهمگام، باید سرویس‌گیرنده و سرویس‌دهنده به طور همزمان در حال اجرا باشند.

اگرچه صف‌بندی پیام بطور ناهمگام کار می‌کند، اما چون سرویس‌گیرنده جهت خواندن داده‌های ارسال شده منتظر سرویس‌دهنده نمی‌ماند، یک تفاوت اساسی بین صف‌بندی پیام و برنامه‌نویسی غیرهمگام وجود دارد. صف‌بندی پیام می‌تواند در یک محیط ارتباط منفصل نیز انجام شود. زمانی که داده ارسال می‌شود، گیرنده می‌تواند روی خط نباشد. سپس زمانی که گیرنده روی خط می‌رود، بدون فرستادن درخواست، داده‌ها را دریافت می‌کند.

شما می‌توانید برنامه‌نویسی ارتباط متصل و منفصل را با صحبت کردن یک فرد روی تلفن و ارسال یک پست الکترونیکی مقایسه کنید. زمان صحبت کردن فردی روی تلفن، هر دو نفر باید در یک زمان متصل باشند. این ارتباط به صورت همگام است. در پست الکترونیکی فرستنده مطمئن نیست که چه زمانی به پست الکترونیکی فرستاده شده رسیدگی می‌شود.

¹ Message queuing

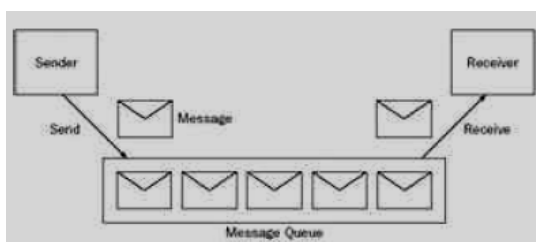
² Course order

³ Disconnected

مطمئناً این امکان وجود دارد که به پست الکترونیکی فرستاده شده، هیچ وقت رسیدگی نشود و ممکن است چشم‌پوشی شود. این طبیعت ارتباط منفصل است. برای پرهیز از این مشکل، تقاضای یک جواب جهت تأیید خوانده شدن پست الکترونیکی امکان‌پذیر می‌باشد. اگر در یک محدوده‌ی زمانی جواب داده نشود، ممکن است با این استثنا روبرو شده باشید. این عمل با صف‌بندی پیام امکان‌پذیر است.

در بعضی موارد صف‌بندی پیام همانند پست الکترونیکی برای ارتباط برنامه با برنامه، به جای ارتباط شخص با شخص است. با این وجود، صف‌بندی پیام ویژگی‌هایی که سرویس‌های پست الکترونیکی ندارد: همچون تحویل تضمینی، تراکنش‌ها، تأییدها، مد ویژه با استفاده‌ی حافظه و غیره را در بر می‌گیرد. همانطور که در بخش بعدی خواهید دید، صف‌بندی پیام تعداد زیادی ویژگی مفید برای ارتباط بین برنامه‌های کاربردی دارد.

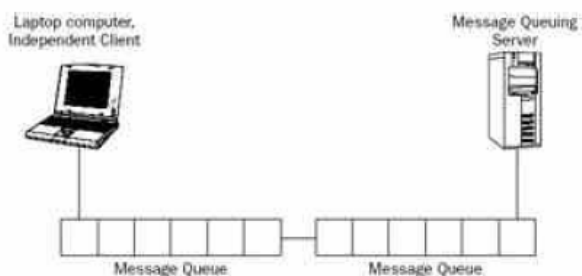
با صف‌بندی پیام می‌توانید در یک محیط منفصل یا متصل، پیام‌هایی را ارسال، دریافت و مسیریابی کنید. شکل ۱-۲۸ یک روش ساده از کاربرد پیام‌ها را نشان می‌دهد. فرستنده پیام‌ها را به صف پیام ارسال می‌کند و گیرنده پیام‌ها را از صف دریافت می‌کند.



شکل ۱-۲۸

۱-۲۸-۱-چه زمانی صف‌بندی پیام را به کار ببریم؟

یکی از مواردی که صف‌بندی پیام توصیه می‌شود، زمانی است که بیشتر مواقع برنامه‌ی کاربردی سرویس‌گیرنده از شبکه منفصل است (برای مثال: بررسی مشتریان روی سایت توسط کارشناس فروش). کارشناس فروش می‌تواند داده‌های سفارش را مستقیماً در سایت مشتری وارد کند. برنامه کاربردی برای هر سفارش یک پیام به صف پیام ارسال می‌کند، که روی سیستم سرویس‌گیرنده قرار می‌گیرد. به محض اینکه کارشناس فروش به اداره بر می‌گردد، سفارش جهت مدیریت به طور اتوماتیک از صف پیام سیستم سرویس‌گیرنده به سیستم مقصد منتقل می‌شود. سیستم مقصد جایی است که پیام پردازش می‌شود.

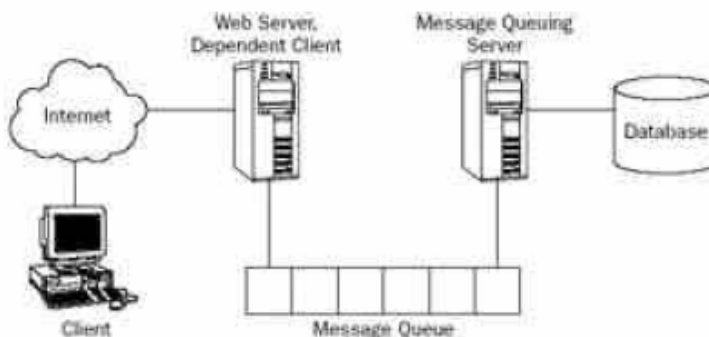


شکل ۲-۲۸

علاوه بر استفاده از یک کامپیوتر کیفی، کارشناس فروش می‌تواند یک دستگاه Pocket Windows جهت دسترسی به صف‌بندی پیام به کار برد.

صف‌بندی پیام در محیط‌های متصل نیز می‌تواند مفید واقع شود. یک سایت تجارت الکترونیکی را تصور کنید (شکل ۲۸-۳). جایی که سرور در زمان‌های قطعی مانند عصر هر روز یا آخر هفته، تراکنش‌ها را به طور کامل بارگذاری می‌کند، اما سرعت بارگذاری در شب پایین است. یک راه‌حل، خرید یک سرور با سرعت بالاتر یا اضافه کردن سرورهای اضافی به سیستم برای

اداره کردن ترافیک بالا است. اما یک راه‌حل ارزانتر نیز وجود دارد: بارگذاری‌های لحظاتی پرترافیک را با انتقال تراکنش‌ها از زمان‌های شلوغ به زمان‌های خلوت، یکنواخت کنید. در این طرح، سفارشات به صف پیام ارسال می‌شوند و طرف گیرنده سفارشات را با نرخ کاراتر از سیستم پایگاه داده می‌خواند. اکنون بار سیستم در کل زمان پخش می‌شود: بنابراین کار کردن سرور با تراکنش‌ها، می‌تواند از ارتقا دادن سرورهای پایگاه داده ارزان‌تر باشد.



شکل ۲۸-۳

۲۸-۱-۲- ویژگی‌های صف‌بندی پیام

صف‌بندی پیام بخشی از سیستم‌عامل ویندوز است. ویژگی‌های اصلی این سرویس به صورت زیر است:

- پیام‌ها می‌توانند در یک محیط منفصل ارسال شوند. لازم نیست برنامه‌های کاربردی ارسال و دریافت به طور همزمان در حال اجرا باشند.
- در مد ویژه، پیام‌ها می‌توانند بسیار سریع ارسال شوند. پیام‌های مد ویژه فقط در حافظه ذخیره می‌شوند.
- در یک مکانیزم قابل ترمیم، ارسال پیام‌ها می‌تواند با استفاده از تحویل تضمینی انجام شود. پیام‌های قابل ترمیم در فایل‌ها ذخیره می‌شوند. حتی در مواردی که سیستم راه‌اندازی مجدد می‌شود، پیام‌ها تحویل می‌گردند.
- صف‌های پیام می‌توانند از طریق ACLها امنیت پیام‌ها را کنترل کنند. به وسیله‌ی ACLها می‌توان مشخص کرد که کدام کاربر می‌تواند پیام‌ها را به یک صف ارسال یا از آن دریافت کند. همچنین جهت جلوگیری از استراق سمع در شبکه می‌توان پیام‌ها را رمزنگاری کرد.
- جهت ارسال سریع تر بخش‌های خاص با اولویت بالاتر، می‌توان از اولویت‌ها در اداره کردن پیام‌ها استفاده کرد.
- MQ۳.۰ ارسال پیام‌های چندبخشی را کنترل می‌کند.

در ادامه‌ی این فصل نحوه‌ی استفاده از این ویژگی‌ها بررسی می‌شود.

۲۸-۱-۳- محصولات صف بندی پیام

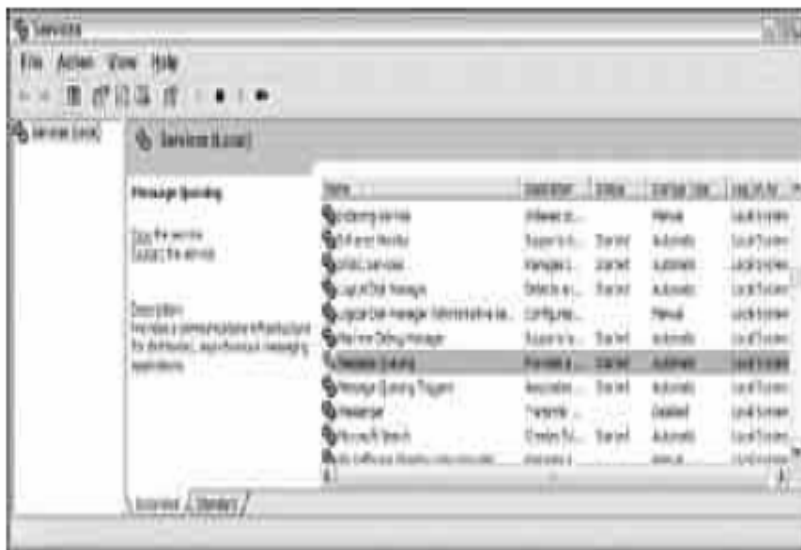
MQ۳.۰ بخشی از WinXP و Win۲۰۰۳ Server می‌باشد. در Win۲۰۰۰ نسخه MQ۲.۰ وجود دارد که پروتکل HTTP و پیام‌های چندبخشی را پشتیبانی نمی‌کند. می‌توان MQ۳.۰ را به عنوان بخشی از سیستم‌عامل نصب کرد. در WinXP از طریق Add-Remove Program و قسمت Windows Componets گزینه‌ی Message Queuing را انتخاب کنید. قطعات زیر می‌توانند در تنظیمات Message Queuing انتخاب شوند:

Common: زیر قطعه‌ی Common برای عملکرد پایه‌ای MQ لازم است.

Active Directory Integration: در این روش اسامی پیام‌ها به Active Directory نوشته می‌شوند. با این گزینه یافتن صف‌ها در مجتمع Active Directory امکان پذیر است و می‌توان صف‌ها را در برابر گروه‌ها و کاربران ویندوز ایمن کرد.

MsMQHTTP Support: ارسال و دریافت پیام‌ها را از طریق پروتکل HTTP ممکن می‌سازد.

Triggers: برنامه‌های کاربردی Triggers می‌توانند با دریافت یک پیام جدید اجرا گردند. بعد از نصب MQ باید سرویس آن راه‌اندازی گردد (شکل ۲۸-۴). این سرویس پیام‌ها را خوانده و می‌نویسد و برای مسیریابی پیام‌ها در طول شبکه با سرورهای MQ دیگر ارتباط برقرار می‌کند.



شکل ۲۸-۴

۲۸-۲- معماری MQ

بوسیله‌ی MQ پیام‌ها از صف پیام خوانده شده یا به صف پیام نوشته می‌شوند. پیام‌ها و صف‌های پیام خصوصیات متعددی دارند که در ادامه به دقت شرح داده خواهند شد.

۲۸-۲-۱- پیام‌ها

یک پیام به یک صف پیام ارسال می‌شود. هر پیام یک بدنه برای دربرگرفتن داده‌ی اصلی و یک برچسب برای عنوان پیام دارد. هر نوع اطلاعاتی می‌تواند در بدنه‌ی پیام قرار گیرد. بوسیله‌ی کلاس‌های NET، چندین فرمت‌دهنده، داده‌ها را برای گذاشتن در داخل بدنه پیام تبدیل می‌کنند. علاوه بر بدنه و برچسب، پیام اطلاعات بیشتری در مورد فرستنده، پیکربندی مهلت پایان، شناسه‌ی تراکنش یا اولویت را در بردارد.

صف‌های پیام چندین نوع پیام دارند:

- پیام عادی: به وسیله‌ی یک برنامه کاربردی ارسال می‌شود.
- پیام اعلام وصول: حالت یک پیام عادی را گزارش می‌دهد. پیام‌های اعلام وصول جهت گزارش موفقیت یا عدم موفقیت ارسال پیام‌های عادی به صف‌های مدیریت ارسال می‌شوند.
- پیام‌های جواب: در صورتیکه فرستنده‌ی اصلی به پاسخ خاصی نیاز داشته باشد، از طریق گیرنده ارسال می‌شود.

• پیام گزارش: بوسیله‌ی سیستم MQ تولید می‌شود. پیام‌های آزمایش و پیام‌های پیگرد به این دسته تعلق دارند.

یک پیام می‌تواند اولویت داشته باشد، تا ترتیب خواندن پیام‌ها از صف را مشخص کند. پیام‌ها براساس اولویت در صف مرتب می‌شوند. بنابراین پیام بعدی که از صف خوانده می‌شود، یکی از بالاترین اولویت‌هاست.

پیام‌ها دو مد تحویل دارند: ویژه و قابل ترمیم. پیام‌های ویژه خیلی سریع تحویل داده می‌شوند، چون حافظه فقط برای ذخیره کردن آنها استفاده می‌شود. پیام‌های قابل ترمیم در هر مرحله از مسیریابی در فایل‌هایی ذخیره می‌شوند، تا زمانی که تحویل داده شوند. این روش تحویل قابل اعتماد است، حتی در صورت خرابی شبکه یا راه‌اندازی مجدد سیستم مشکل‌ساز نیست.

پیام‌های تراکنشی، نسخه‌ی خاصی از پیام‌های قابل ترمیم هستند. با استفاده از پیام‌رسانی تراکنشی تضمین می‌شود که پیام‌ها فقط یک بار و به همان ترتیبی که ارسال شده‌اند به مقصد می‌رسند.

۲۸-۲-۲-صف پیام

صف پیام یک انباره‌ی پیام است. پیام‌های ذخیره شده روی دیسک را می‌توان در فهرست `>\system\windir\storage\mq\msmq` یافت. برای ارسال پیام‌ها، معمولاً از صف‌های عمومی یا خصوصی استفاده می‌شود، اما انواع دیگری از صف‌ها نیز وجود دارند:

• یک صف عمومی در Active Directory منتشر می‌گردد. اطلاعات راجع به این صف‌ها در میان دامنه‌های Active Directory کپی می‌گردند. می‌توانید از ویژگی‌های جستجو و کاوش جهت گرفتن اطلاعاتی درباره این صف‌ها استفاده کنید. بدون داشتن نام کامپیوتر مربوط به یک صف عمومی، می‌توانید به آن دستیابی کنید. همچنین انتقال یک صف از سیستمی به سیستم دیگر بدون آگاهی سرویس‌گیرنده نیز امکان‌پذیر می‌باشد. ایجاد صف‌های عمومی در یک محیط WorkGroup امکان‌پذیر نیست، چون به Active Directory نیاز است.

• صف‌های خصوصی در Active Directory منتشر نمی‌شوند. این صف‌ها فقط در صورت داشتن نام کامل مسیر صف قابل دستیابی هستند. صف‌های خصوصی در محیط WorkGroup نیز قابل استفاده هستند.

• صف‌های روزنامه (journal): برای نگهداری کپی پیام‌ها بعد از ارسال یا دریافت آنها استفاده می‌شوند. با فعال کردن روزنامه برای یک صف عمومی یا خصوصی، بطور اتوماتیک یک صف روزنامه ایجاد می‌شود. به وسیله صف‌های روزنامه دو نوع صف مختلف امکان‌پذیر هستند: روزنامه نگاری مبدأ و روزنامه نگاری مقصد. روزنامه‌نگار مبدأ بوسیله خصوصیات یک پیام به جریان می‌افتد. روزنامه‌نگاری مقصد بوسیله‌ی خصوصیات یک صف به جریان می‌افتد؛ این پیام‌ها در صف روزنامه سیستم مقصد ذخیره می‌شوند.

• برخلاف برنامه‌نویسی همگام که خطاها فوراً تشخیص داده می‌شوند، در صف‌بندی پیام با روش متفاوتی به خطاها رسیدگی می‌شود. اگر یک پیام در مهلت زمانی خاصی به سیستم مقصد نرسد، پیام در صف نامهی غیرقابل توزیع ذخیره می‌شود. می‌توان صف نامه‌های غیر قابل توزیع برای بررسی پیام‌هایی که نرسیده‌اند، بررسی کرد.

• صف‌های مدیریت شامل تصدیق‌هایی برای پیام‌های ارسال شده می‌باشند. فرستنده می‌تواند این صف را جهت دریافت تاییدیه از ارسال موفق پیام‌ها تعیین کند.

• اگر به بیش از یک پیام تصدیق ساده به عنوان پاسخی از طرف دریافت کننده نیاز باشد، می‌توان از یک صف جواب استفاده کرد. برنامه‌ی کاربردی دریافت کننده می‌تواند پیام‌های جواب را به فرستنده‌ی اصلی ارسال کند.

• یک صف گزارش برای آزمایش پیام‌ها استفاده می‌شود. صف‌های گزارش را می‌توان از طریق تغییر نوع یک صف خصوصی یا عمومی به شناسه‌ی از پیش تعریف شده‌ی `{CEq6AFD0020-108CFB11-CCEq- ۳۳F8EE55}` ایجاد کرد.

صف‌های گزارش می‌توانند به عنوان یک ابزار مفید برای پیگیری پیام‌ها روی مسیرهایشان بکار برده شوند.

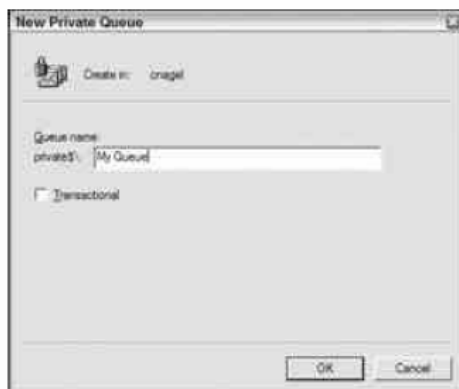
- صف‌های سیستم خصوصی هستند و به وسیله سیستم صف‌بندی پیام بکار برده می‌شوند. این صف‌ها برای پیام‌های مدیریت، ذخیره‌ی پیام‌های اطلاع‌رسانی و تضمین ترتیب صحیح پیام‌های تراکنشی استفاده می‌شوند.

۲۸-۲-۳- ابزارهای مدیریت صف‌بندی پیام

قبل از اینکه صف‌بندی پیام را با برنامه‌نویسی بررسی کنیم. این بخش ابزار مدیریت آن را جهت ایجاد و مدیریت صف‌ها و پیام‌ها، که بخشی از سیستم‌عامل هستند بررسی می‌کند. ابزارهایی که نشان داده شده‌اند، تنها بوسیله‌ی صف‌بندی پیام استفاده نمی‌شوند. فقط در صورتیکه MQ نصب شده باشد، ویژگی‌های صف‌بندی پیام و این ابزارها در دسترس هستند.

ایجاد صف‌های پیام

در پنجره‌ی MMC مربوط به Computer Management می‌توان صف‌های پیام را ایجاد کرد. در WinXP این MMC را با شروع از Control Panel و قسمت Administrative Tools اجرا کنید یا در Run دستور compmgmt.msc را اجرا کنید. در نمایش درختی Message Queuing در زیر شاخه‌ی Services and Application قرار می‌گیرد. بعد از انتخاب Private Queues یا Public Queues، می‌توانید از طریق منوی Action، صف‌های جدید را ایجاد کنید (شکل ۲۸-۵ را ببینید). فقط در صورتی که Message Queuing در مد Active Directory پیکربندی شده باشد، صف‌های عمومی قابل دسترس هستند.



شکل ۲۸-۵

خصوصیات صف پیام

بعد از ایجاد صف، در کنسول Computer Management می‌توانید از طریق منوی Action خصوصیات صف را تغییر دهید (شکل ۲۸-۶).



شکل ۲۸-۶

چندین گزینه را می‌توان پیکربندی کرد:

- برچسب که نام صف بوده و می‌تواند برای جستجوی صف به کار برده شود.
- نوع شناسه که به طور پیش‌فرض در {} قرار دارد و برای نگاشت چندین صف به یک دسته یا نوع واحد به کار می‌رود. صف‌های گزارش، یک شناسه نوع خاص استفاده می‌کنند. شناسه‌ی نوع، یک شناسه‌ی منحصر به فرد جهانی (uuid) یا GUID است.
- توجه: می‌توانید شناسه‌های منحصر به فرد را به کمک نرم‌افزارهای سودمند `uuidgen.exe` یا `guidgen.exe` ایجاد کنید. `uuidgen.exe` یک ابزار از نوع خط فرمان و `guidgen.exe` یک ابزار گرافیکی است.
- حداکثر اندازه‌ی همه‌ی پیام‌های یک صف می‌تواند محدود شود، تا دیسک به طور کامل پر نشود. $2,0\text{MQ}$ یک محدودیت GB۲ برای ذخیره پیام داشت، ولی $3,0\text{MQ}$ هیچ محدودیتی ندارد.
- گزینه‌ی `Authenticated` را برای کنترل مجاز بودن کاربران جهت خواندن یا نوشتن پیام علامت بزنید.
- بوسیله گزینه‌ی `privacy level` می‌توانید محتوای پیام را رمزنگاری کنید. مقادیر آن می‌توانند `None`، `Optional` یا `Body` انتخاب شوند. `None` به معنی پذیرش فقط پیام‌های رمزنگاری نشده است. `Body` فقط پیام‌های رمز شده را می‌پذیرد و مقدار پیش‌فرض `Optional` هر دو را می‌پذیرد.
- با تنظیمات `Journal` می‌توان روزنامه‌نگاری مقصد را پیکربندی کرد. بوسیله‌ی این گزینه، کپی پیام‌های دریافت شده در روزنامه ذخیره می‌شوند. حداکثر اندازه از فضای دیسک تخصیص یافته به پیام‌های روزنامه‌ای یک صف را می‌توان پیکربندی کرد. زمانی که اندازه به حداکثر برسد، روزنامه‌نگاری مقصد متوقف می‌گردد.
- $3,0\text{MQ}$ یک گزینه‌ی پیکربندی جدید به نام `Multicast` دارد که یک آدرس IP چندپخشی برای صف تعریف می‌کند. این آدرس IP چندپخشی می‌تواند در چندین گره از شبکه استفاده شود. پس زمانی که یک پیام به یک آدرس ارسال شود، توسط چندین صف دریافت می‌شود.

۲۸-۳- برنامه‌نویسی صف‌بندی پیام

اکنون که شما معماری صف‌بندی پیام را درک کردید، می‌توانید به برنامه‌نویسی آن پردازید. در بخش‌های بعدی نحوه‌ی ایجاد، کنترل صف‌ها و چگونگی ارسال و دریافت پیام‌ها را می‌بینید. همچنین یک برنامه‌ی کاربردی کوچک تکلیف درس می‌سازید که یک بخش ارسال و دریافت پیام دارد.

۲۸-۳-۱-ایجاد یک صف پیام

تا بحال نحوه‌ی ایجاد صف‌های پیام بوسیله‌ی Computer Management را دیده‌اید. بوسیله متد Create () از کلاس MessageQueue می‌توانید یک صف پیام ایجاد کنید. مسیر صف جدید به متد Create () رد می‌شود. مسیر شامل نام میزبان مربوط به محل صف و نام صف است. در مثال، صف MyNewPublicQueue روی میزبان محلی ایجاد شده است. برای ایجاد یک مسیر خصوصی، باید کلمه‌ی \$private در اول مسیر باشد. برای مثال، private\$\MyNewPrivateQueue\ بعد از احضار متد Create () می‌توانید خصوصیات صف را تغییر دهید. برای مثال، با استفاده از خصوصیت Label برچسب صف را DemoQueue قرار دهید. برنامه‌ی مثال، مسیر صف و نام فرمت را روی کنسول می‌نویسد. نام فرمت به طور اتوماتیک بوسیله‌ی یک UUID جهت دسترسی به صف بدون نام سرور ایجاد می‌شود.

```
using System;
using System.Messaging;
namespace Wrox.ProCSharp.Messaging
{
    class Program
    {
        static void Main(string[] args)
        {
            using (MessageQueue queue = MessageQueue.Create(@".\MyNewPublicQueue"))
            {
                queue.Label = "Demo Queue";
                Console.WriteLine("Queue created:");
                Console.WriteLine("Path: {0}", queue.Path);
                Console.WriteLine("FormatName: {0}", queue.FormatName);
            }
        }
    }
}
```

۲۸-۳-۲-برنامه کاربردی CourseOrder

برای نشان دادن نحوه‌ی کار صف‌بندی پیام، در این بخش یک راه‌حل ساده جهت تکلیف درس ایجاد می‌کنید. این راه‌حل از سه اسمبلی تشکیل شده است:

- یک کتابخانه ((CourseOrder کلاس‌های موجودیت برای پیام‌های انتقالی را در بر دارد.
- یک برنامه کاربردی (Windows Forms) CourseOrderSender که پیام‌ها را از صف پیام دریافت می‌کند.

کتابخانه‌ی کلاس CourseOrder

هر دو برنامه‌ی ارسال و دریافت پیام، اطلاعات تکلیف را نیاز دارند. به همین دلیل کلاس‌های موجودیت در اسمبلی مجزایی قرار داده می‌شوند. اسمبلی CourseOrder سه کلاس موجودیت دارد: CourseOrder ، Course ، Customer. در برنامه‌ی کاربردی نمونه، همه‌ی خصوصیات به صورت دنیای واقعی پیاده‌سازی نمی‌شوند، فقط مشخصات کافی جهت رساندن مفهوم این کار پیاده‌سازی شده است. در فایل Course.CS کلاس Course تعریف شده است. این کلاس فقط یک خصوصیت برای عنوان درس دارد:

```
using System;
namespace Wrox.ProCSharp.Messaging
{
    public class Course
    {
        public Course()
        {
        }
        public Course(string title)
        {
        }
    }
}
```

```
{
this.title = title;
}
private string title;
public string Title
{
get
{
return title;
}
set
{
title = value;
}
}
}
```

فایل Customer.cs کلاس Customer را در بردارد که شامل اسامی شرکت و شماره تماس می باشد:

```
using System;
namespace Wrox.ProCSharp.Messaging
{
public class Customer
{
public Customer()
{
}
public Customer(string company, string contact)
{
this.company = company;
this.contact = contact;
}
private string company;
public string Company
{
get
{
return company;
}
set
{
company = value;
}
}
private string contact;
public string Contact
{
get
{
return contact;
}
set
{
contact = value;
}
}
}
```

در فایل CourseOrder.cs کلاس Customer یک مشتری و یک درس را در یک تکلیف نگاشت می کند:

```
using System;
namespace Wrox.ProCSharp.Messaging
{
public class CourseOrder
```

```

{
public CourseOrder()
{
}
private Customer customer;
public Customer Customer
{
get
{
return customer;
}
set
{
customer = value;
}
}
private Course course;
public Course Course
{
get
{
return course;
}
set
{
course = value;
}
}
}
}

```

ارسال کننده‌ی پیام تکلیف درس

بخش دوم راه‌حل، یک برنامه کاربردی ویندوز به نام CourseOrderSender است. بوسیله‌ی این برنامه، تکلیف‌های درس به صف پیام ارسال می‌شوند. باید اسمبلی‌های CourseOrder و System.Messaging ارجاع داده شوند.

واسط کاربردی این برنامه در شکل ۲۸-۷ نمایش داده شده است. عناصر کادر بازشوی ComboBoxCourses چندین درس همچون "Web Services, Advance .NET Programming" و "ADO.NET" را در بر دارد.

شکل ۲۸-۷

زمانی که روی دکمه Submit the Order کلیک می‌شود، متد OnSubmitCourseOrder() احضار می‌شود. بوسیله‌ی این متد یک شی CourseOrder ایجاد می‌شود و خصوصیات آن با محتوای کنترل‌های TextBox و ComboBox پر می‌شود. سپس یک نمونه از MessageQueue برای بازکردن یک صف عمومی با یک نام فرمت ایجاد می‌شود. نام فرمت برای ارسال پیام استفاده می‌شود، حتی در حالتی که صف قابل دسترس نباشد، می‌توانید نام فرمت را با استفاده از کنسول Computer Management و خواندن شناسه‌ی صف پیام بدست آورید. متد Send() شی از CourseOrder را جهت سریال کردن آن بوسیله‌ی فرمت دهنده‌ی XMLMessage و نوشتن آن به صف فراخوانی می‌کند.

```
private void OnSubmitCourseOrder(object sender, EventArgs e)
{
    CourseOrder order = new CourseOrder();
    order.Course = new Course(comboBoxCourses.SelectedItem.ToString());
    order.Customer = new Customer(textCompany.Text, textContact.Text);
    using (MessageQueue queue = new MessageQueue(
        "FormatName:Public="))
    {
        queue.Send(order, "Course Order {" + order.Customer.Company + "}");
    }
    MessageBox.Show("Course Order submitted");
}
```

ارسال پیام‌های قابل ترمیم و اولویت‌دار

با تنظیم خصوصیت `Priority` کلاس `Message` می‌توان پیام‌ها را اولویت‌بندی کرد. اگر پیام‌ها به صورت خاص پیکربندی شده باشند، باید یک شی `Message` در همان جایی که بدنه‌ی پیام به سازنده‌ی آن رد می‌شود، ایجاد گردد. در این مثال، اولویت در `Message Priority` تنظیم می‌گردد. در صورتیکه کادر انتخاب `CheckBoxPriority` تیک داشته باشد، صفت اولویت پیام `MessagePriority.High` قرار داده می‌شود. `MessagePriority` یک نوع شمارشی است که به شما این امکان را می‌دهد تا از پایین‌ترین اولویت (۰) تا بالاترین اولویت (۷) به آن مقداردهی کنید. مقدار پیش‌فرض `Normal` مقدار اولویت ۳ را دارد.

برای قابل ترمیم کردن پیام، خصوصیت `Recoverable` پیام را `true` قرار دهید.

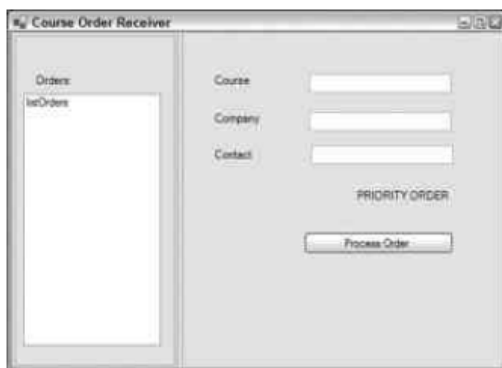
```
private void OnSubmitCourseOrder(object sender, EventArgs e)
{
    CourseOrder order = new CourseOrder();
    order.Course = new Course(comboBoxCourses.SelectedItem.ToString());
    order.Customer = new Customer(textCompany.Text, textContact.Text);
    using (MessageQueue queue = new MessageQueue(
        "FormatName:Public="))
    using (System.Messaging.Message message =
        new System.Messaging.Message(order))
    {
        if (checkBoxPriority.Checked)
            message.Priority = MessagePriority.High;
        message.Recoverable = true;
        queue.Send(message, "Course Order {" + order.Customer.Company + "}");
    }
    MessageBox.Show("Course Order submitted");
}
```

با اجرای برنامه می‌توانید تکلیف‌های درس را به صف پیام اضافه کنید (شکل ۲۸-۸).

شکل ۲۸-۸

دریافت کننده‌ی پیام تکلیف درس

نمای طراحی برنامه کاربردی دریافت تکلیف درس در شکل ۲۸-۹ نمایش داده شده است. این برنامه پیام‌ها را از صف پیام می‌خواند. این برنامه کاربردی برچسب هر تکلیفی را در کادر لیست listOrders نمایش می‌دهد. زمانیکه یک تکلیف انتخاب می‌شود، محتوای تکلیف بوسیله کنترل‌های سمت راست فرم نمایش داده می‌شوند.



شکل ۲۸-۹

در سازنده‌ی کلاس فرم CourseOrderRecieveForm شی MessageQueue ایجاد می‌شود و این شی به همان صف ایجاد شده در برنامه کاربردی ارسال کننده ارجاع می‌کند. XMLMessageFormater بوسیله‌ی انواع پیام‌ها که در خصوصیت Formater صف قرار دارند، پیام‌ها را می‌خواند.

برای نمایش پیام‌های موجود در لیست، یک ریسمان جدید ایجاد می‌شود که به صورت پنهان از دید، پیام‌ها را بررسی می‌کند. ریسمان PeekMessage متد اصلی است. اطلاعات بیشتر در مورد ریسمان‌ها را از فصل ریسمان‌ها و همگام‌سازی بخوانید.

```
using System;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;
using System.Threading;
using System.Messaging;
using Wrox.ProCSharp.Messaging;
namespace CourseOrderReceiver
{
    public partial class CourseOrderReceiverForm : Form
    {
        private MessageQueue orderQueue;
        public CourseOrderReceiverForm()
        {
            InitializeComponent();
            orderQueue = new MessageQueue(
                "FormatName:Public=");
            System.Type[] types = new Type[۲];
            types[۰] = typeof(CourseOrder);
            types[۱] = typeof(Customer);
            types[۲] = typeof(Course);
            orderQueue.Formatter = new XmlMessageFormatter(types);
            // start the thread that fills the ListBox with orders
            Thread t۱ = new Thread(new ThreadStart(PeekMessages));
            t۱.IsBackground = true;
            t۱.Start();
        }
    }
}
```

متد اصلی ریسمان (یعنی PeekMessage) شمارنده‌ی صف پیام را برای نمایش همه‌ی پیام‌ها به کار می‌برد. تا زمانی که پیام جدیدی در صف باشد، حلقه‌ی while ادامه پیدا می‌کند. اگر پیام دیگری در صف نباشد، ریسمان به مدت سه ساعت منتظر می‌ماند تا قبل از خروج آن متد پیام دیگری برسد.

برای نمایش پیام‌های صف در کادر لیست، باید ریسمان مربوط به عمل نوشتن به کادر لیست را به ریسمان ایجاد کننده کادر لیست واگذار کند. چون کنترل‌های Windows Form به یک ریسمان منفردی محدود می‌گردند. فقط سازنده‌ی کنترل می‌تواند به خصوصیات و متدهای آن دسترسی داشته باشد و متد Invoke() تقاضا را به ریسمان ایجاد کننده می‌فرستد.

```
private delegate void MethodInvoker(LabelIdMapping labelIdMapping);
private void PeekMessages()
{
    using (MessageEnumerator messageEnum = orderQueue.GetMessageEnumerator۲())
    {
        while (messageEnum.MoveNext(TimeSpan.FromHours(۳)))
        {
            Invoke(new MethodInvoker(AddListItem),
                new LabelIdMapping(messageEnum.Current.Label,
                    messageEnum.Current.Id));
        }
    }
    MessageBox.Show("No orders in the last ۳ hours. Exiting thread");
}
private void AddListItem(LabelIdMapping labelIdMapping)
{
    listOrders.Items.Add(labelIdMapping);
}
```

کنترل ListBox عناصری از کلاس LabelIdMapping را در بردارد. این کلاس برای نمایش برچسب پیام‌ها در کادر لیست استفاده می‌شود. اما شناسه‌ی پیام‌ها را پنهان نگه می‌دارد. شناسه‌ی پیام می‌تواند برای خواندن پیام در زمان دیگری استفاده شود.

```
private class LabelIdMapping
{
    private string label;
    private string id;
    public LabelIdMapping(string label, string id)
    {
        this.label = label;
        this.id = id;
    }
    public override string ToString()
    {
        return label;
    }
    public string Id
    {
        get
        {
            return id;
        }
    }
}
```

کنترل ListBox یک رویداد SelectedIndexChanged اختصاص داده شده به متد OnOrderSelectionChanged دارد. این متد شی LabelIdMapping را از انتخاب جاری می‌گیرد و شناسه‌ی آن را برای بررسی پنهان پیام توسط متد peekById() بکار می‌برد. سپس محتوای پیام در کنترل‌های TextBox نمایش داده می‌شود.

```
private void OnOrderSelectionChanged(object sender, EventArgs e)
{
    LabelIdMapping labelId = (LabelIdMapping)listOrders.SelectedItem;
    if (labelId == null)
```

```

return;
System.Messaging.Message message = orderQueue.PeekById(labelId.Id);
CourseOrder order = message.Body as CourseOrder;
if (order != null)
{
    textCourse.Text = order.Course.Title;
    textCompany.Text = order.Customer.Company;
    textContact.Text = order.Customer.Contact;
    buttonProcessOrder.Enabled = true;
    if (message.Priority > MessagePriority.Normal)
    {
        labelPriority.Visible = true;
    }
}
else
{
    labelPriority.Visible = false;
}
}
else
{
    MessageBox.Show("The selected item is not a course order");
}
}

```

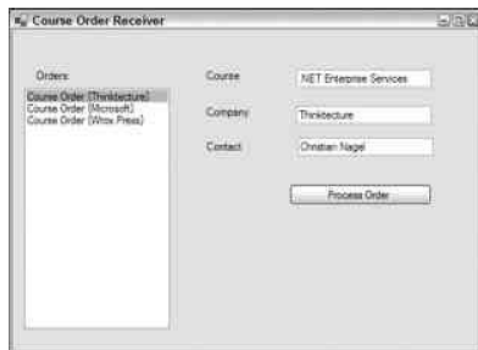
زمانی که روی دکمه‌ی Process Order کلیک شود، متد OnProcessOrder احضار می‌شود. در اینجا مجدداً، پیام جاری انتخاب شده از کادر لیست ارجاع داده می‌شود و با فراخوانی متد ReceiveById (پیام از صف حذف می‌گردد).

```

private void OnProcessOrder(object sender, EventArgs e)
{
    LabelIdMapping labelId = (LabelIdMapping)listOrders.SelectedItem;
    System.Messaging.Message message =
    orderQueue.ReceiveById(labelId.Id);
    listOrders.Items.Remove(labelId);
    listOrders.SelectedIndex = -1;
    buttonProcessOrder.Enabled = false;
    textCompany.Text = "";
    textContact.Text = "";
    textCourse.Text = "";
    MessageBox.Show("Course order processed");
}
}
}

```

شکل ۲۸-۱۰ اجرای برنامه‌ی کاربردی دریافت کردن پیام را نشان می‌دهد که سه تکلیف را لیست می‌کند و در حال جاری یک تکلیف انتخاب می‌شود.



شکل ۲۸-۱۰

دریافت کردن نتایج

با این نسخه از برنامه کاربردی، برنامه‌ی ارسال کننده هرگز نمی‌داند که آیا پیام مورد نظر تا بحال توزیع شده است. برای گرفتن نتایج از گیرنده، می‌توان صف‌های تصدیق یا جواب را به کار برد.

صف‌های تصدیق

بوسیله‌ی یک صف تصدیق، برنامه‌ی ارسال کننده می‌تواند اطلاعاتی درباره‌ی حالت پیام بگیرد. بوسیله‌ی این تصدیق‌ها می‌توانید تعیین کنید آیا دوست دارید یک پاسخ دریافت کنید؟ یعنی اینکه جواب ارسال درست یا اشتباه پیام به فرستنده برگردانده شود. به عنوان مثال، تصدیق‌ها می‌توانند در زمان رسیدن پیام‌ها به صف مقصد یا زمان خوانده شدن پیام‌ها یا در صورت نرسیدن پیام ارسال شوند.

در این مثال شیء AdministrationQueue از نوع کلاس Message با صف CourseOrderAck مقداردهی می‌شود. این صف باید شبیه صف عادی ایجاد شود. این صف به روش دیگری استفاده می‌شود. فرستنده‌ی اصلی تصدیق‌ها را دریافت می‌کند. برای گرفتن یک تصدیق در زمان خوانده شدن یک پیام، خصوصیت AcknowledgementType صف را با AcknowledgementType.FullReceive مقداردهی کنید.

```
Message message = new Message(order);
message.AdministrationQueue =
new MessageQueue(@".\CourseOrderAck");
message.AcknowledgementType = AcknowledgementTypes.FullReceive;
queue.Send(message, "Course Order {" +
order.Customer.Company + "}");
string id = message.Id;
```

Correlation id برای تعیین هر پیام تصدیق متعلق به پیام ارسال استفاده می‌شود. هر پیامی که ارسال می‌شود، یک شناسه دارد و پیام تصدیق مربوط به هر پیامی، شناسه‌ی پیام اصلی را به عنوان Correlation id نگه می‌دارد. پیام‌های صف تصدیق می‌توانند به وسیله‌ی متد MessageQueue.ReceiveByCorrelationId() برای دریافت تصدیق اختصاص داده شده به آنها اقدام کنند. برای پیدا کردن پیام‌هایی که اصلاً به مقصد نرسیده‌اند، به جای صف تصدیق از صف روزنامه‌ی غیرقابل توزیع استفاده کنید. اگر مقدار صفت UseDeadLetterQueue از کلاس Message را true قرار دهید، در صورتیکه پیام قبل از اتمام مهلت زمانی به آن مقصد نرسد، پیام‌ها به صف روزنامه‌ی غیرقابل توزیع کپی می‌شوند.

می‌توان مهلت‌های زمانی را به وسیله‌ی خصوصیات TimeToReachQueue و TimeToBeReceived تنظیم کرد.

صف‌های جواب

اگر اطلاعاتی بیش از تصدیق از کامپیوتر گیرنده نیاز دارید، از یک صف جواب می‌توان استفاده کرد. یک صف جواب شبیه یک صف عادی است، اما فرستنده‌ی اصلی آن صف را به عنوان یک گیرنده به کار می‌برد و گیرنده‌ی اصلی نیز آن صف را به عنوان یک فرستنده به کار می‌برد.

فرستنده باید مقدار خصوصیت ResponseQueue کلاس Message را به صف جواب انتساب دهد. کد نمونه‌ی زیر نشان می‌دهد که گیرنده چگونه صف جواب را برای برگرداندن یک پیام جواب به کار می‌برد. خصوصیت CorrelationId پیام جواب با شناسه‌ی پیام اصلی مقداردهی می‌شود. با این روش سرویس گیرنده تشخیص می‌دهد که هر جوابی متعلق به کدام پیام ارسالی است. این شبیه صف‌های تصدیق است. پیام جواب به وسیله‌ی متد Send() شیء MessageQueue از خصوصیت ResponseQueue ارسال می‌شود.

```
public void ReceiveMessage(Message message)
{
Message responseMessage = new Message("response");
responseMessage.CorrelationId = message.Id;
message.ReesponseQueue.Send(responseMessage);
}
```

صف‌های تراکنشی

بوسیله‌ی پیام‌های قابل ترمیم، رسیدن منظم پیام‌ها و یک بار رسیدن هر پیامی تضمین نمی‌شود. خطاهای شبکه منجر می‌شوند پیام‌ها چندین بار برسند، همچنین اگر فرستنده و گیرنده هر دو چندین پروتکل شبکه داشته باشند و صف‌بندی پیام نیز این پروتکل‌ها را به کار برد، این امر اتفاق می‌افتد. در جاهایی که نیازمند تضمین‌های زیر هستیم، می‌توان صف‌های تراکنشی را به کار برد:

• پیام‌ها به همان ترتیب ارسال، دریافت شوند.

• پیام‌ها فقط یک بار برسند.

بوسیله‌ی صف‌های تراکنشی، یک تراکنش واحد، ارسال و دریافت پیام‌ها را در اختیار نمی‌گیرد و طبیعت صف‌بندی پیام این است که زمان مابین ارسال و دریافت پیام می‌تواند کاملاً طولانی باشد. در مقابل آن، تراکنش‌ها باید کوتاه باشند. بوسیله‌ی صف‌بندی پیام، تراکنش اول برای ارسال پیام به صف، تراکنش دوم برای فرستادن پیام روی شبکه و تراکنش سوم برای دریافت پیام‌ها استفاده می‌شود.

مثال بعدی نحوه‌ی ایجاد صف پیام تراکنشی و چگونگی ارسال پیام‌ها با استفاده از تراکنش را نشان می‌دهد.

یک صف پیام تراکنشی با رد کردن مقدار `true` در پارامتر دوم متد `MessageQueue.Create` ایجاد می‌شود. اگر دوست دارید توسط یک تراکنش واحد چندین پیام را به صف بنویسید، باید یک نمونه از شی `MessageQueueTransaction` معرفی کرده و متد `Begin()` آن را احضار کنید. زمانی که ارسال همه پیام‌های متعلق به یک تراکنش پایان یابد، باید متد `Commit()` شی `MessageQueueTransaction` فراخوانی شود. برای لغو کردن یک تراکنش (هیچ پیامی به صف نوشته نشود)، باید متد `Abort()` فراخوانی شود. این فراخوانی در قسمت `Catch` انجام می‌گیرد.

```
using System;
using System.Messaging;
namespace Wrox.ProCSharp.Messaging
{
    class Program
    {
        static void Main(string[] args)
        {
            if (!MessageQueue.Exists(@".\MyTransactionalQueue"))
            {
                MessageQueue.Create(@".\MyTransactionalQueue", true);
            }
            MessageQueue queue = new MessageQueue(@".\MyTransactionalQueue");
            MessageQueueTransaction transaction =
                new MessageQueueTransaction();
            try
            {
                transaction.Begin();
                queue.Send("a", transaction);
                queue.Send("b", transaction);
                queue.Send("c", transaction);
                transaction.Commit();
            }
            catch
            {
                transaction.Abort();
            }
        }
    }
}
```

۲۸-۴-نصب MessageQueue

صف‌های پیام می‌توانند بوسیله متد `MessageQueue.Create` ایجاد شوند. با این وجود، کاربری که برنامه را اجرا می‌کند، معمولاً امتیازهای مدیریتی جهت ایجاد صف‌های پیام را ندارد. معمولاً صف‌های پیام با نصب یک برنامه ایجاد می‌شوند. برای نصب برنامه‌ها نصب، کلاس `MessageQueueInstaller` می‌تواند استفاده شود. اگر یک کلاس نصب کننده، بخشی از یک برنامه‌ی کاربردی باشد، برنامه‌ی سودمند `installuti.exe` از خط فرمان متد `Install()` نصب کننده را احضار می‌کند.

VS۲۰۰۵ یک پشتیبانی خاصی برای کاربرد `MessageQueueInstaller` در برنامه‌های کاربردی `Windows Form` دارد. اگر یک قطعه‌ی `MessageQueue` از کادر ابزار روی فرم گذاشته شود، برچسب هوشمند قطعه به شما اجازه می‌دهد یک نصب کننده از طریق منوی `Add Installer` اضافه کنید. حال می‌توانید شی `MessageQueueInstaller` را برای تعریف صف‌های تراکنشی، روزنامه، نوع فرمت دهنده، اولویت پایه و غیره را پیکربندی کنید.

۲۸-۵-خلاصه

در این فصل چگونگی استفاده از `Message Queuing` را دیدید. صفبندی پیام یک تکنولوژی مهم است که نه تنها ارتباط ناهمگام بلکه ارتباط منفصل را نیز فراهم می‌کند. فرستنده و گیرنده در زمان‌های مختلف می‌توانند در حال اجرا باشند، که این ویژگی بار کاری سرور را روی کل زمان توزیع می‌کند.

مهم‌ترین کلاس‌های صفبندی پیام `MessageQueue` و `Message` هستند. کلاس `MessageQueue` ارسال، دریافت و بررسی پنهان پیام‌ها را امکان‌پذیر می‌کند و کلاس `Message` برای تعریف محتوای پیام ارسالی است.

فصل بیست و نهم

برنامه‌نویسی NET و Com

اگر ما همه کدهای نوشته شده تا به حال را رها کرده و از ابتدا شروع کنیم، خوب به نظر می‌رسد. اما این عمل، یک گزینه‌ی قابل قبول برای بیشتر شرکت‌ها نیست. در گذشته بیشتر سازمان‌های توسعه‌دهنده، یک سرمایه‌گذاری قابل توجهی در توسعه و خرید قطعات Com و کنترل‌های ActiveX انجام داده بودند. مایکروسافت یک کمیته تضمین قابلیت استفاده‌ی مجدد این قطعات در NET را ایجاد کرده است و قطعات NET، از طریق Com به آسانی قابل فراخوانی هستند.

این فصل پشتیبانی NET برای وارد کردن کنترل‌های ActiveX و قطعات Com به برنامه، آشکار کردن^۱ کلاس‌های NET به برنامه‌های کاربردی بر پایه Com و فراخوانی مستقیم Win ۳۲ API را تشریح می‌کند.

۲۹-۱- وارد کردن^۲ کنترل‌های ActiveX

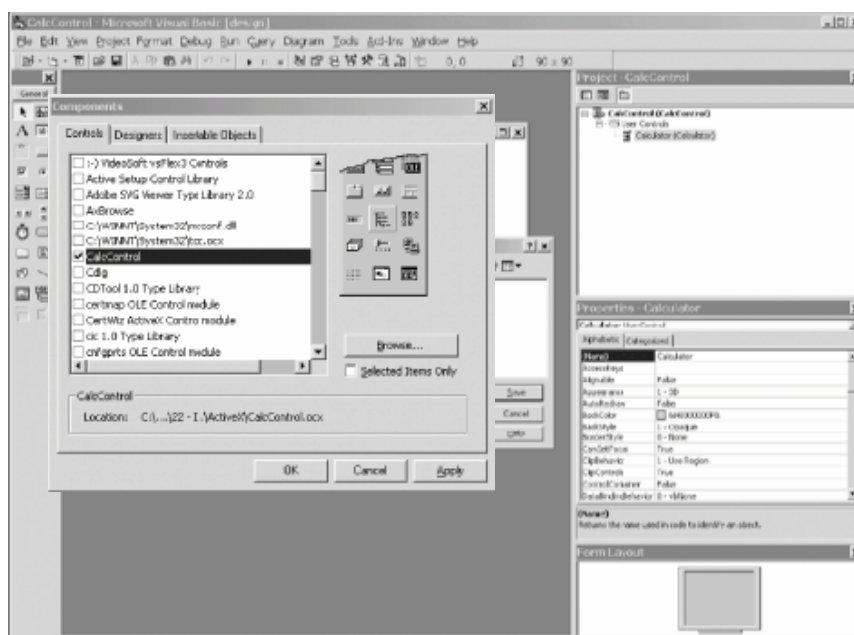
کنترل‌های ActiveX قطعات Com هستند، که روی فرم قرار داده می‌شوند و ممکن است یک واسط کاربر داشته باشند یا نداشته باشند. زمانی که مایکروسافت استاندارد OCX را توسعه داد، توسعه‌دهندگان اجازه یافتند کنترل‌های ActiveX را در VB ایجاد کرده و در ++C آنها را بکار گیرند. از این زمان بود که انقلاب کنترل ActiveX شروع شد. در طول چند سال گذشته، هزاران کنترل ActiveX توسعه داده شده‌اند و مورد استفاده قرار گرفته‌اند. ActiveXها، قطعه کدهای کوچک هستند که کار کردن با آنها راحت است. یک مثال کارا از ActiveX، استفاده‌ی مجدد کدهای باینری می‌باشند.

وارد کردن کنترل‌های ActiveX در NET، بسیار آسان است. VS ۲۰۰۵ قادر است کنترل‌های ActiveX را به طور خودکار وارد کند. در خط فرمان VS ۲۰۰۵ یک نرم‌افزار کمکی به نام AxImp توسعه داده شده است که اسمبلی‌های ضروری کنترل مورد نظر را در NET ایجاد می‌کند.

زمانی که کنترلی در محیط ویندوز استاندارد کار می‌کند، شما آن را به برنامه‌ی کاربردی Form Windows خود وارد خواهید کرد. برای ایجاد کنترل جدید، VB ۶ را باز کرده و در نوع پروژه جدید ActiveX Control را انتخاب کنید. فرم پروژه را تا حد امکان کوچک کنید، چون این کنترل واسط کاربر نخواهد داشت. روی User Control کلیک راست کرده و Properties را انتخاب کنید. نام آن را به Calculator تغییر دهید. در Project Explorer روی Project کلیک کرده و در پنجره Properties، آن را به CalcControl تغییر نام دهید. فوراً پروژه را ذخیره کرده و فایل و پروژه را CalcControl نامگذاری کنید. همانطور که در شکل ۲۹-۱ نمایش داده شده است:

^۱ Exposing

^۲ Import



شکل ۲۹-۱

حال می‌توانید ۴ تابع محاسباتی را، در پنجره کد از طریق انتخاب Code View اضافه کنید. کد مورد نظر VB در مثال ۲۹-۱ نشان داده شده است:

مثال ۲۹-۱

```
Public Function _
Add(left As Double, right As Double) _
As Double
Add = left + right
End Function
Public Function _
Subtract(left As Double, right As Double) _
As Double
Subtract = left - right
End Function
Public Function _
Multiply(left As Double, right As Double) _
As Double
Multiply = left * right
End Function
Public Function _
Divide(left As Double, right As Double) _
As Double
Divide = left / right
End Function
```

این قطعه کد، کد کامل کنترل است. از منوی File، گزینه‌ی Make CalcControl.ocx را اجرا کنید، تا این کد کامپایل شود. در قدم بعدی یک پروژه استاندارد اجرایی در VB ۶ بسازید. نام فرم را TestForm و نام پروژه را CalcTest تعیین و آنها را ذخیره کنید.

کنترل ActiveX را به عنوان یک قطعه به وسیله فشردن Ctrl+T و انتخاب CalcControl از فرم ظاهر شده، به برنامه‌ی خود اضافه کنید (چنانچه در شکل ۲۹-۲ نشان داده شده است).

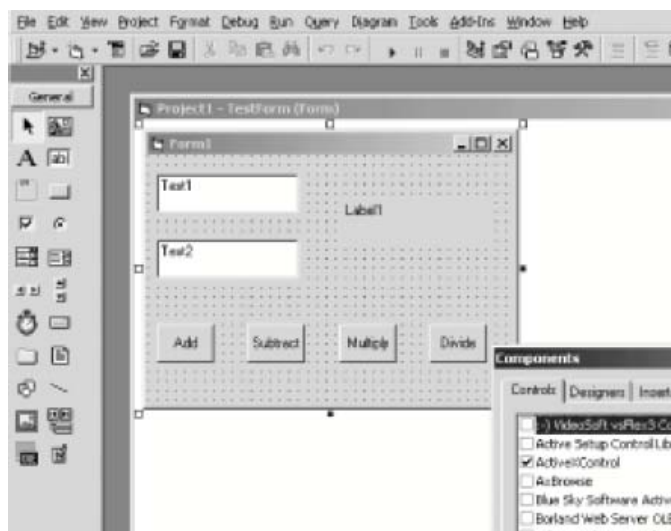
شکل ۲-۲۹

این عمل باعث می‌شود تا یک کنترل روی جعبه ابزار قرار گیرد. چنان که در شکل ۳-۲۹ می‌بینید



شکل ۳-۲۹

کنترل جدید را بر روی فرم TestForm اضافه کنید و آن را CalcControl بنامید. توجه داشته باشید که کنترل جدید قابل رویت نیست. این کنترل هیچ رابط کاربری ندارد. چهار دکمه و دو کادر متنی و یک برچسب به فرم اضافه کنید. چنانچه در شکل ۴-۲۹ نشان داده شده است:



شکل ۴-۲۹

اسامی دکمه‌ها را چنین انتخاب کنید: btnAdd, btnSubtract, btnMultiply, btnDivide. مثال ۲-۲۹ کد کامل این برنامه را نشان می‌دهد.

هرگاه که روی یکی از این دکمه‌ها کلیک کنید، از شما دو عدد خواسته می‌شود، که باید در کادرهای متنی وارد کنید و نتیجه در برچسب‌ها نشان داده می‌شود.

مثال ۲-۲۹

```
Private Sub btnAdd_Click()  
    Label1.Caption = _  
        calcControl.Add(CDbl(Text1.Text), _  
            CDbl(Text2.Text))  
End Sub  
Private Sub btnDivide_Click()  
    Label1.Caption = _  
        calcControl.Divide(CDbl(Text1.Text), _  
            CDbl(Text2.Text))  
End Sub  
Private Sub btnMultiply_Click()  
    Label1.Caption = _
```

```

calcControl.Multiply(CDbl(Text\Text), _
CDbl(Text۲.Text))
End Sub
Private Sub btnSubtract_Click( )
Label\Caption = _
calcControl.Subtract(CDbl(Text\Text), _
CDbl(Text۲.Text))
End Sub

```

۲۹-۲-وارد کردن یک کنترل به .NET

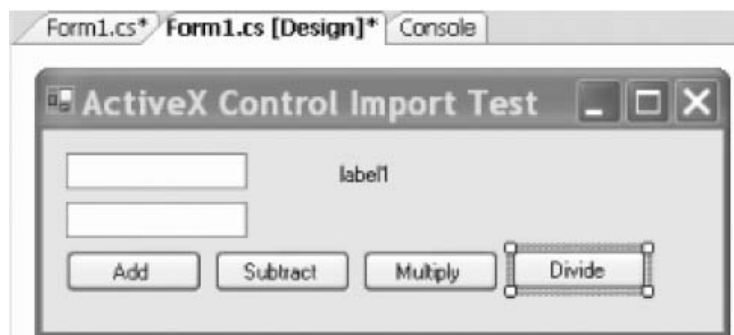
توجه داشته باشید که کنترل CalcControl باید درست کار کند. می‌توانید این کنترل را در VB استفاده کرده تا مطمئن شوید. می‌توانید فایل CalcControl.ocx را به محیط توسعه‌ی .NET کپی کنید. بعد از کپی کردن، لازم است فایل CalcControl.ocx را از طریق برنامه‌ی Regsvr ۳۲ ثبت کنیم. حال می‌توانید از کنترل CalcControl در برنامه‌ی .NET خود استفاده کنید:

```

regsrvr۳۲ CalcControl.ocx

```

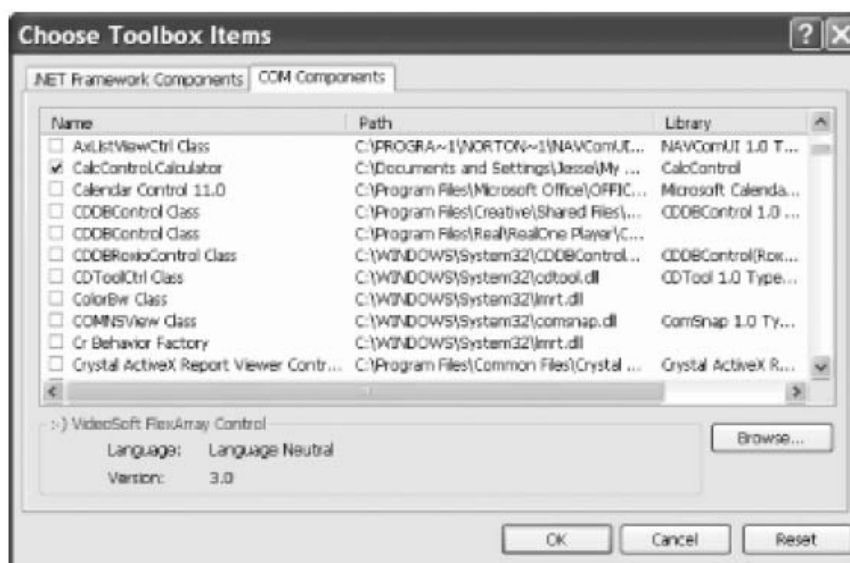
برای شروع کار، یک برنامه ویندوز #VC در VS ۲۰۰۵ ایجاد کنید. آن برنامه را InteropTest نامگذاری کنید، و فرم خود را با کشیدن کنترل‌ها روی آن به صورت شکل ۲۹-۵ طراحی کنید. فرم خود را TestForm نامگذاری کنید.



شکل ۲۹-۵

۲۹-۲-۱-وارد کردن یک کنترل

دو روش برای وارد کردن کنترل ActiveX به محیط توسعه VS ۲۰۰۵ وجود دارد: می‌توانید ابزار محیط VS ۲۰۰۵ را به کار ببرید، یا اینکه با استفاده از نرم‌افزار aximp در .NET SDK به صورت دستی کنترل را وارد کنید. برای استفاده‌ی VS ۲۰۰۵ از منوی Tools، Choose Toolbox Items را انتخاب کنید. در برگه‌ی Com Components، CalcControl.Calculator را پیدا کنید. همانطور که در شکل ۲۹-۶ نمایش داده شده است.

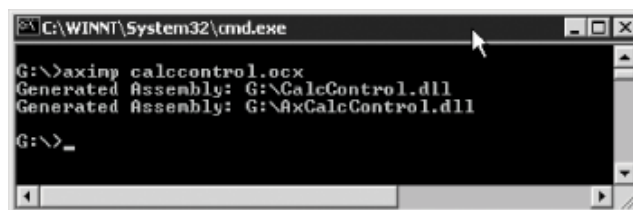


شکل ۲۹-۶

چون CalcControl روی سیستم NET. شما ثبت هست، در VS ۲۰۰۵ از منوی Choose ToolBox Tools را اجرا کرده و کنترل خود را پیدا کنید. زمانی که کنترل از کادر محاوره‌ای انتخاب می‌شود، به برنامه کاربردی شما وارد می‌شود. VS ۲۰۰۵ جزئیات کار، همچون اضافه کردن آن کنترل به نوار ابزار را انجام می‌دهد.

۲۹-۲-۲-وارد کردن کنترل به صورت دستی

می‌توانید یک کادر فرمان باز کرده و با استفاده از نرم‌افزار کمکی aximp.exe، کنترل را به صورت دستی وارد کنید. همانطور که در شکل ۲۹-۷ می‌بینید.

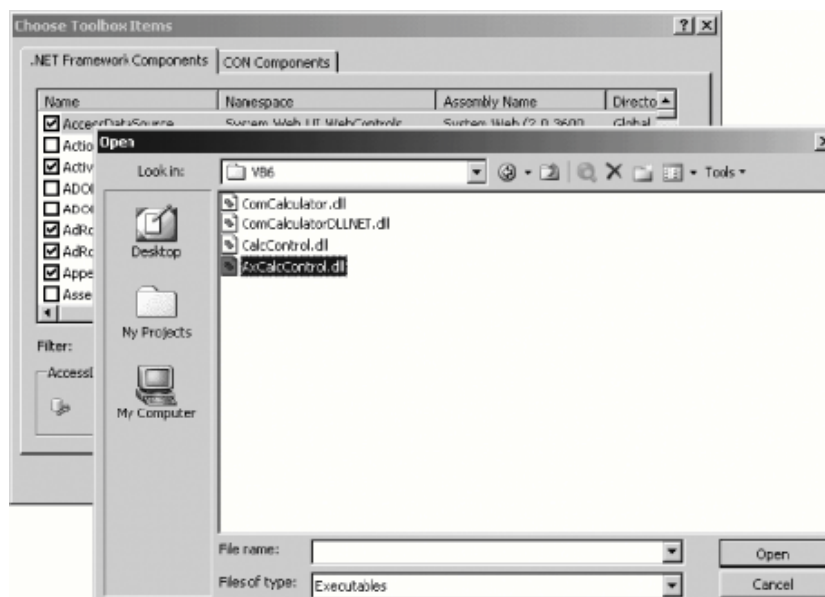


شکل ۲۹-۷

aximp.exe، یک کنترل ActiveX مورد نظر را به عنوان آرگومان خود گرفته و سه فایل زیر را تولید می‌کند:

- AxCalcControl.dll: یک کنترل ویندوز تحت NET.
- CalcControl.dll: یک کتابخانه کلاس NET.
- AxCalcControl.pdb: یک فایل اشکال زدایی

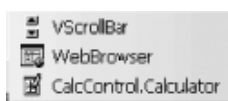
بعد از انجام این کار می‌توانید به پنجره‌ی Choose Toolbox Items برگشته و NET Framework Components را انتخاب کنید. حال می‌توانید مسیری را که کنترل Axcalccontrol.dll تحت NET قرار گرفته را جستجو کرده و فایل را به داخل کادر ابزار وارد کنید. همانطور که در شکل ۲۹-۸ می‌بینید.



شکل ۲۹-۸

۲۹-۲-۳- اضافه کردن کنترل به فرم

زمانی که کنترل وارد می‌شود، کنترل روی منوی کادر ابزار، همانند شکل ۲۹-۹ ظاهر می‌گردد. توجه داشته باشید که ممکن است کنترل در پایین کادر ابزار، نیز ظاهر گردد.



شکل ۲۹-۹

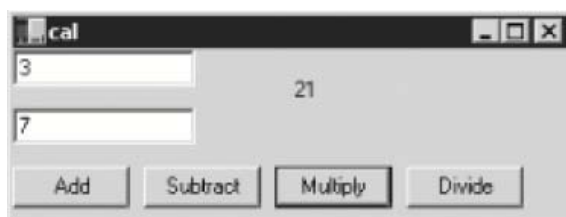
حال می‌توانید این کنترل را روی فرم کشیده و توابع آن را بکار ببرید، همانطور که در مثال ۶VB این کار را انجام دادید. اداره‌کننده‌های رویداد را برای هر دکمه بیافزایید. کد منبع برای اداره‌کننده‌های رویداد در مثال ۳-۲۹ نمایش داده شده است.

مثال ۲۹-۳

```
private void btnAdd_Click(object sender, System.EventArgs e)
{
    double left = double.Parse(textBox1.Text);
    double right = double.Parse(textBox2.Text);
    label1.Text = axCalculator.Add( ref left, ref right).ToString( );
}
private void btnDivide_Click(object sender, System.EventArgs e)
{
    double left = double.Parse(textBox1.Text);
    double right = double.Parse(textBox2.Text);
    label1.Text = axCalculator.Divide(ref left, ref right).ToString( );
}
private void btnMultiply_Click(object sender, System.EventArgs e)
{
    double left = double.Parse(textBox1.Text);
    double right = double.Parse(textBox2.Text);
    label1.Text = axCalculator.Multiply(ref left, ref right).ToString( );
}
```

```
private void btnSubtract_Click(object sender, System.EventArgs e)
{
    double left = double.Parse(textBox1.Text);
    double right = double.Parse(textBox2.Text);
    label1.Text = axCalculator.Subtract(ref left, ref right).ToString( );
}
```

هر متغیر، مقادیر مورد نیازش را از فیلدهای متنی^۱ بدست می‌آورد، و با استفاده از متد ایستای Double.Parse به نوع مضاعف^۲ تبدیل می‌کند و آن مقادیر را به متدهای ماشین حساب رد می‌کند. نتایج بازگشتی به رشته تبدیل شده و در برچسب^۳ درج می‌شوند. همانطور که در شکل ۱۰-۲۹ می‌بینید.



شکل ۱۰-۲۹

۲۹-۳-وارد کردن قطعات Com

وارد کردن کنترل‌های ActiveX می‌توانند نسبتاً سر راست باشند. بیشتر قطعات Com تولید شده توسط شرکت‌ها ActiveX نیستند، هر چند آنها فایل‌های استاندارد COM DLL هستند. برای اینکه بفهمید چگونه از این فایل‌ها به همراه NET استفاده کنید، به ۶VB برگشته و یک شی تجاری COM ایجاد کنید که دقیقاً شبیه قطعه قبلی کار کند.

گام اول ایجاد یک پروژه‌ی جدید ActiveX DLL است. این مراحل ساخت یک COM DLL استاندارد در ۶VB است. نام کلاس را ComCalc و نام پروژه را ComCalculater قرار دهید. فایل و پروژه را ذخیره کنید. متدها را از مثال ۲۹-۴، در پنجره کد کپی کنید.

مثال ۲۲-۴

```
Public Function
Add(left As Double, right As Double) _
As Double
Add = left + right
End Function
Public Function _
Subtract(left As Double, right As Double) _
As Double
Subtract = left - right
End Function
Public Function _
Multiply(left As Double, right As Double) _
As Double
Multiply = left * right
End Function
Public Function _
Divide(left As Double, right As Double) _
As Double
```

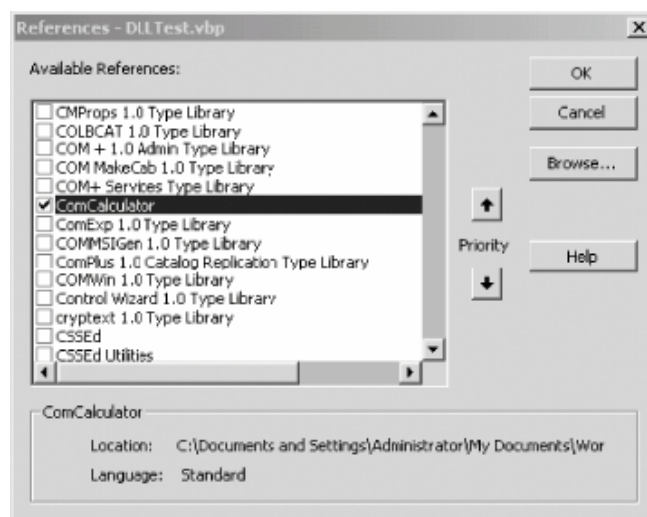
^۱ Textbox

^۲ Double

^۳ Label

```
Divide = left / right
End Function
```

با استفاده از منوی File گزینه‌ی DLL Make ComCalculator.dll را اجرا کنید. برای تست این، به برنامه‌ی اخیر برگشته و Calculator را از فرم کنترل حذف کنید. DLL جدید را، به وسیله‌ی باز کردن پنجره‌ی Project References، و یافتن ComCalculator، به برنامه اضافه کنید. چنانچه در شکل ۲۹-۱۱ نشان داده شده است.



شکل ۲۹-۱۱

۲۹-۳-۱- کدنویسی برنامه‌ی ComTestForm

کد تمرین قطعه Com خیلی شبیه اولین مثال است. در اینجا، یک نمونه از شی Comcalc ایجاد کرده و متدهای آن را فراخوانی کنید. مثال ۲۹-۵ را مشاهده کنید.

مثال ۲۹-۵

```
Private Sub btnAdd_Click()
Dim theCalc As New ComCalc
Label1.Caption = _
theCalc.Add(CDbl(Text1.Text), _
CDbl(Text2.Text))
End Sub
Private Sub btnDivide_Click( )
Dim theCalc As New ComCalc
Label1.Caption = _
theCalc.Divide(CDbl(Text1.Text), _
CDbl(Text2.Text))
End Sub
Private Sub btnMultiply_Click( )
Dim theCalc As New ComCalc
Label1.Caption = _
theCalc.Multiply(CDbl(Text1.Text), _
CDbl(Text2.Text))
End Sub
Private Sub btnSubtract_Click( )
Dim theCalc As New ComCalc
Label1.Caption = _
theCalc.Subtract(CDbl(Text1.Text), _
CDbl(Text2.Text))
End Sub
```

۲۹-۳-۲-وارد کردن COM DLL به NET.

تا به حال شما با ComCalc DLL کار می‌کردید که قادر بودید آن را به NET وارد کنید. قبل از وارد کردن آن، باید یکی از موارد مقید کردن دیرهنگام و زودهنگام را انتخاب کنید. زمانی که سرویس‌گیرنده یک متد را از سرویس‌دهنده فراخوانی می‌کند، باید مسئله آدرس متد روی حافظه‌ی سرویس‌دهنده حل شود. این پروسه، مقید کردن خواننده می‌شود. در مقید کردن زودهنگام، تحلیل آدرس یک متد روی سرویس‌دهنده، زمان کامپایل پروژه در سرویس‌گیرنده اتفاق می‌افتد و فراداده‌ها به ماژول NET سرویس‌گیرنده اضافه می‌شود. در مقید کردن دیرهنگام، تحلیل در زمان ارجاع رخ می‌دهد، زمانی که COM سرور را برای آن متد کاوش می‌کند.

مقید کردن زودهنگام مزایای زیادی دارد. مهمترین آنها بهره‌وری است. متدهای مقید شده زودهنگام نسبت به متدهای مقید شده دیرهنگام، سریع احضار می‌شوند. برای انجام مقید کردن زود هنگام، کامپایلر باید شی Com را بازرسی کند. در هنگام بازرسی باید ابتدا کتابخانه مورد نظر به NET وارد شود.

۲۹-۳-۳-وارد کردن کتابخانه نوع داده

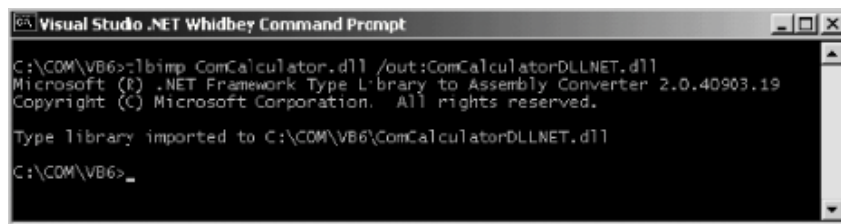
Com Dll تولید شده با VB۶، یک کتابخانه‌ی نوع داده همراه خود دارد. اما فرمت یک کتابخانه نوع Com با NET قابل استفاده نیست. برای حل این مشکل باید کتابخانه نوع Com را به یک اسمبلی وارد کنید. با دو روش می‌توان این کار را انجام داد. در IDE با رجیستر کردن قطعه می‌توانید آن را به اسمبلی وارد کنید. همانطور که در بخش زیر نشان داده می‌شود یا با استفاده از دستور Tlbimp.exe، کتابخانه‌ی نوع داده را به صورت دستی وارد کنید. Tlbimp.exe یک اسمبلی interop تولید خواهد کرد. شی NET که شی Com را بسته‌بندی می‌کند،^۱ Rcw خوانده می‌شود. سرویس‌گیرنده NET، Rcw را برای مقید کردن متدهای شی Com استفاده خواهد کرد که در بخش بعدی خواهید دید.

۲۹-۳-۴-وارد کردن به صورت دستی

فایل Comcalculator.Dll را به محیط NET خود کپی کرده و آن را با Regsvr ۳۲ ثبت کنید. سپس شما برای وارد کردن شی Com به NET با اجرای Tlbimp.exe آماده هستید. نحوه‌ی کار به صورت زیر است:

نام فایل اسمبلی: out\ نام فایل Tlbimp.exe dll

شکل ۲۹-۱۲ را مشاهده کنید.



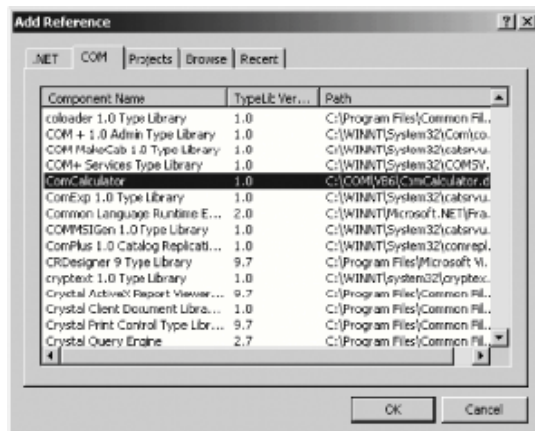
شکل ۲۹-۱۲

۲۲-۳-۵-ایجاد یک برنامه آزمایشی

حال می‌توانید یک برنامه‌ی آزمایشی جهت شی Com ایجاد کنید و آن را ComDllTest نامگذاری کنید. اگر تصمیم دارید کتابخانه را به صورت دستی وارد نکنید و آن را از طریق IDE وارد کنید. برای انجام این کار در کادر محاوره‌ای Add Reference Com را انتخاب کرده و شی Com ثبت شده را مشخص کنید. شکل ۲۹-۱۳ را ببینید.

^۱ Runtime callable wrapper

شکل ۲۹-۱۳

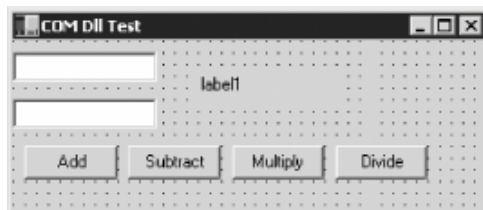


این عمل نرم افزار Tlbimp را احضار خواهد کرد و RCW منتج شد را به مسیر

C:\Documents and Settings\Administrator\Application
Data\Microsoft\VisualStudio\RCW.

کپی خواهد کرد.

باید توجه داشته باشید، چون Dll ای که آن تولید می کند هم نام ComDll است، اگر Tlbimp.exe را بکار برید، می توانید ارجاع را از طریق برگه Projects اضافه کنید. فهرستی که ComcalculaordllNETDll را جستجو کرده و آن را به ارجاع ها اضافه کنید. حال می توانید واسط برنامه را شبیه برنامه های قبلی ایجاد کنید. (همانند شکل ۲۹-۱۴)



شکل ۲۹-۱۴

کد مربوط به اداره کننده های رویداد چهار دکمه را همانند مثال ۲۹-۶ بنویسید.

مثال ۲۹-۶

```
#region Using directives
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Windows.Forms;
#endregion
namespace ComDLLTest
{
    partial class Form\ : Form
    {
        public Form\ ( )
        {
            InitializeComponent ( );
        }
        private void btnAdd_Click(
            object sender, System.EventArgs e )
        {
            Double left, right, result;
            left = Double.Parse( textBox\ .Text );
            right = Double.Parse( textBox\ .Text );
            ComCalculatorDLLNET.ComCalc theCalc =
            new ComCalculatorDLLNET.ComCalc ( );
```

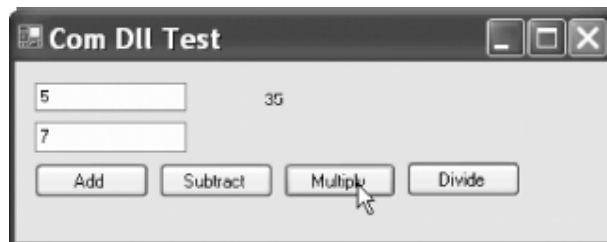
```

result = theCalc.Add( ref left, ref right );
label1.Text = result.ToString();
}
private void btnSubtract_Click(
object sender, System.EventArgs e )
{
Double left, right, result;
left = Double.Parse(textBox1.Text);
right = Double.Parse(textBox2.Text);
ComCalculatorDLLNET.ComCalc theCalc =
new ComCalculatorDLLNET.ComCalc();
result = theCalc.Subtract(ref left, ref right);
label1.Text = result.ToString();
}
private void btnMultiply_Click(
object sender, System.EventArgs e )
{
Double left, right, result;
left = Double.Parse( textBox1.Text );
right = Double.Parse( textBox2.Text );
ComCalculatorDLLNET.ComCalc theCalc =
new ComCalculatorDLLNET.ComCalc();
result = theCalc.Multiply( ref left, ref right );
label1.Text = result.ToString();
}
private void btnDivide_Click(
object sender, System.EventArgs e )
{
Double left, right, result;
left = Double.Parse( textBox1.Text );
right = Double.Parse( textBox2.Text );
ComCalculatorDLLNET.ComCalc theCalc =
new ComCalculatorDLLNET.ComCalc();
result = theCalc.Divide( ref left, ref right );
label1.Text = result.ToString();
}
}
}
}

```

به جای ارجاع به یک کنترل Activex که روی فرم قرار داد، باید نمونه‌ای از Comcalculator.Comcalc ایجاد کنید. شی Com همانند شیئی که در اسمبلی NET ایجاد شده باشد در دسترس است. و اجرای برنامه همانند انتظار ما کار می‌کند. شکل ۲۹-۱۵ را ببینید.

شکل ۲۹-۱۵



۲۹-۳-۶- کاربرد مفید کردن دیرنگام و انعکاس

اگر برای یک شی Com، فایل کتابخانه‌ی نوع داده نباشد، باید مفید کردن دیرنگام با انعکاس را بکار ببرید. برای دیدن نحوه‌ی انجام کار، با مثال ۲۹-۶ شروع کنید، اما ارجاع کتابخانه‌ی وارد شده را حذف کنید. باید ادراک کننده‌های رویداد چهار دکمه مجدداً نوشته شوند. شما نمی‌توانید یک نمونه از شی Comcalculator.ComCalc ایجاد کنید، بلکه باید متدهای آن را به طور پویا احضار کنید.

ابتدا باید یک شی Type جهت نگه داشتن اطلاعاتی در مورد نوع داده‌ی Comcalc تعریف کنید.

```
Type comCalcType;
comCalcType = Type.GetTypeFromProgID("ComCalculator.ComCalc");
```

متد `GetTypeFromProgID` چارچوب NET را برای باز کردن `ComDll` ثبت شده و بازیابی اطلاعات ضروری نوع داده شی خاص راهنمایی می‌شود. این عمل معادل فراخوانی `GetType` است.

```
Type theMathType = Type.GetType("System.Math");
```

حال می‌توانید همانند احضار یک متد روی یک کلاس شرح داده شده در یک اسمبلی NET اقدام کنید. با فراخوانی `CreateInstance` جهت برگرداندن یک نمونه از شی `ComCalc` شروع کنید.

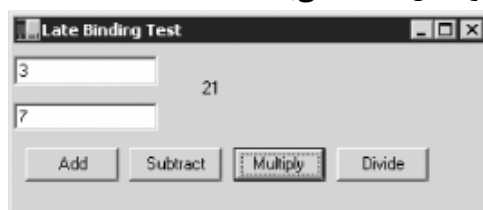
```
object comCalcObject = Activator.CreateInstance(comCalcType);
```

سپس یک آرایه برای نگهداشتن آرگومان‌های آن ایجاد کنید و سپس متد را با استفاده از `InvokeMember` احضار کنید.

نام متد را به صورت رشته، یک فیلد مقید کننده، شی منتج شده از `CreateInstance` و آرایه آرگومان‌های ورودی را به `InvokeMember` رد کنید.

```
object[] inputArguments = {left, right };
result = (Double) comCalcType.InvokeMember(
"Subtract", // the method to invoke
BindingFlags.InvokeMethod, // how to bind
null, // binder
comCalcObject, // the COM object
inputArguments); // the method arguments
```

نتایج این فراخوانی به `Double` قالب‌بندی شده و در متغیر محلی `Result` ذخیره می‌شود. سپس می‌توانید این نتیجه را در واسط کاربر نمایش دهید. همانطور که در شکل ۲۹-۱۶ می‌بینید.



شکل ۲۹-۱۶

همه اداره کننده‌های رویداد باید این کار را تکرار کنند، که فقط نام متد فراخوانده شده متفاوت است. کد مشترک را به یک متد کمکی خصوصی به نام `Inroke` منتقل کنید. مثال ۲۹-۷ باید دستور `using` را برای `System.Reflection` به کد منبع اضافه کند.

مثال ۲۹-۷

```
#region Using directives
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Reflection;
using System.Windows.Forms;
#endregion
namespace LateBinding
{
    partial class Form\ : Form
    {
        public Form\ ( )
        {
            InitializeComponent ( );
        }
        private void btnAdd_Click(
            object sender, System.EventArgs e )
        {
            Invoke( "Add" );
        }
    }
}
```

```

private void btnSubtract_Click(
object sender, System.EventArgs e )
{
    Invoke( "Subtract" );
}
private void btnMultiply_Click(
object sender, System.EventArgs e )
{
    Invoke( "Multiply" );
}
private void btnDivide_Click(
object sender, System.EventArgs e )
{
    Invoke( "Divide" );
}
private void Invoke( string whichMethod )
{
    Double left, right, result;
    left = Double.Parse( textBox\ .Text );
    right = Double.Parse( textBox\ .Text );
    // create a Type object to hold type information
    Type comCalcType;
    // an array for the arguments
    object[] inputArguments =
    { left, right };
    // get the type info from the COM object
    comCalcType =
    Type.GetTypeFromProgID(
    "ComCalculator.ComCalc" );
    // create an instance
    object comCalcObject =
    Activator.CreateInstance( comCalcType );
    // invoke the method dynamically and
    // cast the result to Double
    result = ( Double ) comCalcType.InvokeMember(
    whichMethod, // the method to invoke
    BindingFlags.InvokeMethod, // how to bind
    null, // binder
    comCalcObject, // the COM object
    inputArguments ); // the method arguments
    label\ .Text = result.ToString( );
}
}
}

```

۲۹-۴- صادر کردن قطعات NET

می‌توانید کلاس NET خود را جهت استفاده در قطعات Com موجود صادر کنید. ابزار Regasm، فراداده‌ی قطعه شما را در رجیستری سیستم ثبت خواهد کرد.

Regasm را با نام فایل اسمبلی Dll احضار کنید. این فایل Dll باید در GAC نصب شده باشد. برای مثال:

```
Regasm MyAssembly.dll
```

این عمل فرا داده‌ی قطعه شما را به رجیستری صادر خواهد کرد. مثال، می‌توانید یک پروژه جدید C# Dll ایجاد کنید و ۴ عمل ماشین حساب را مجدداً ایجاد کنید. مثال ۲۹-۸ را نشان می‌دهد.

مثال ۲۹-۸

```

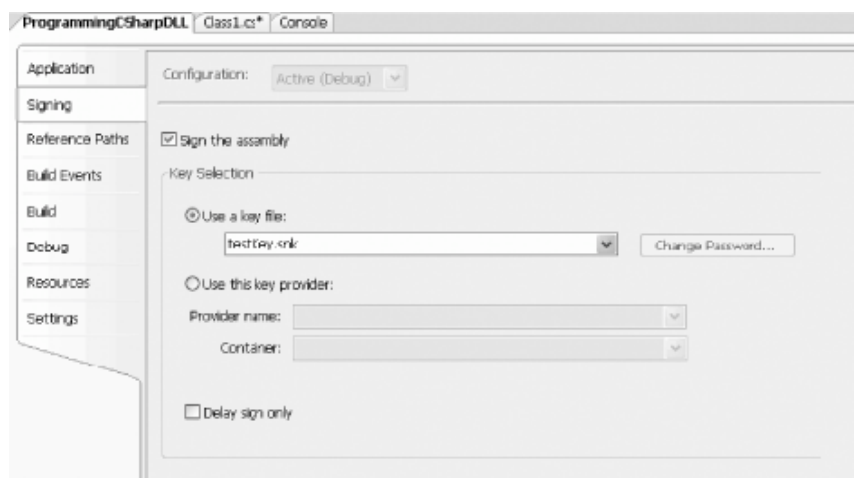
using System;
using System.Reflection;
[assembly: AssemblyKeyFile("test.key")]
namespace Programming_CSharp
{

```



```
public class Calculator
{
    public Calculator( )
    {
    }
    public Double Add (Double left, Double right)
    {
        return left + right;
    }
    public Double Subtract (Double left, Double right)
    {
        return left - right;
    }
    public Double Multiply (Double left, Double right)
    {
        return left * right;
    }
    public Double Divide (Double left, Double right)
    {
        return left / right;
    }
}
```

این فایل را با نام Calculator.CS در یک پروژه به نام ProgrammingCsharpDll ذخیره کنید. برای ایجاد یک نام Strong خصوصیات پروژه را باز کنید و برگه Signing را انتخاب کرده و اسمبلی را امضاء کنید. همانطور که در شکل ۲۹-۱۷ می بینید.



شکل ۲۹-۱۷

این عمل کادر محاوره‌ای Create Key را همانند شکل ۲۹-۱۸ باز خواهد کرد.

شکل ۲۲-۸



برنامه‌ی خود را به GAC اضافه کرده و آن را ثبت کنید.

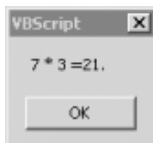
```
gacutil /i ProgrammingCSharpDLL.dll
Regasm ProgrammingCSharpDLL.dll
```

حال می‌توانید با استفاده از VBScript چهار تابع ماشین حساب را به عنوان یک شی Com احضار کنید. برای مثال، یک فایل اسکریپتی کوچک ویندوز همانند مثال ۲۹-۱۹ ایجاد کنید.

مثال ۲۹-۹

```
dim calc
dim msg
dim result
set calc = CreateObject("Programming_CSharp.Calculator")
result = calc.Multiply(۷,۳)
msg = "۷ * ۳ =" & result & "."
Call MsgBox(msg)
```

زمانی که آن اجرا شود، جهت تأیید نحوه کار شی، یک کادر محاوره‌ای باز می‌شود (شکل ۲۹-۱۹ را ببینید).



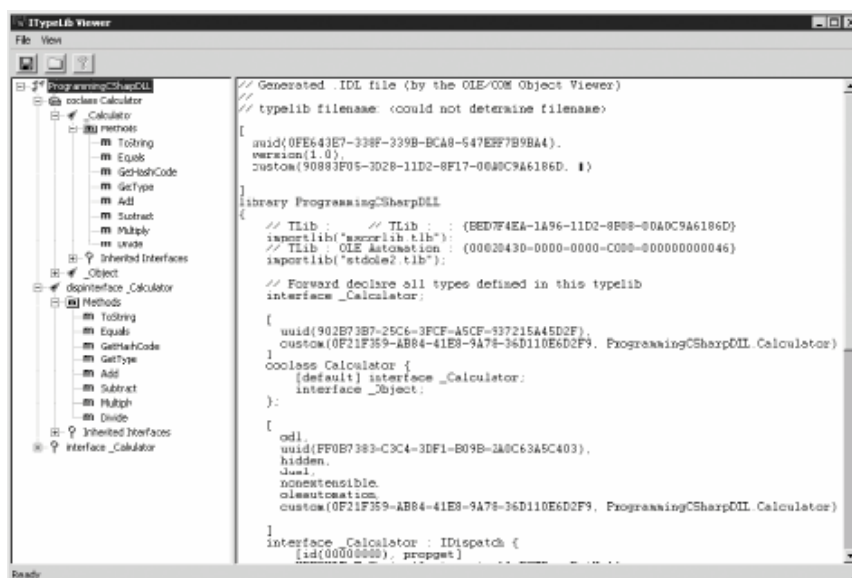
شکل ۲۹-۱۹

۲۲-۴-۱- ایجاد یک کتابخانه‌ی نوع داده

اگر بخواهید مقید کردن زودهنگام را با Dll NET بکار ببرید، در حالت معمولی، یک کتابخانه‌ی نوع داده ایجاد خواهید کرد. می‌توانید این کار را با استفاده نرم‌افزار کمکی TlbExp انجام دهید.

```
TlbExp ProgrammingCSharpDLL.dll /out:Calc.tlb
```

نتیجه‌ی حاصله یک کتابخانه‌ی نوع داده است که در برنامه OLE/Com Object Viewer می‌توانید آن را یافته و مشاهده کنید. همانطور که در شکل ۲۹-۲۰ می‌بینید.



شکل ۲۹-۲۰

با این کتابخانه‌ی نوع داده، می‌توانید کلاس ماشین حساب را به هر قطعه Com وارد کنید.

۲۹-۵-P/Invoke

احضار کد مدیریت نشده از طریق #C امکان‌پذیر است. در صورتی که نتوان یک کار را با استفاده از FCL انجام داد ممکن است کد مدیریت نشده استفاده شود. در نسخه ۲.۰ از .NET، کاربرد P/Invoke را به ندرت خواهید دید. NET Platform از P/Invoke فقط برای فراهم کردن دسترسی به API ویندوز استفاده می‌کند. اما می‌توانید برای فراخوانی توابع هر DLL از آن استفاده کنید.

برای دیدن نحوه‌ی کار آن، متد MoveTo از کلاس FileInfo را برای تغییر نام یک فایل به صورت زیر احضار می‌کنیم.

```
file.MoveTo(fullName + ".bak");
```

می‌توانید همین کار را با استفاده از Kernel۳۲.dll. از ویندوز و احضار متد MoveFile بنا کنید. برای انجام این کار به اعلان یک متد به صورت Static Extern و کاربرد صفت DllImport نیاز دارید.

```
[DllImport("kernel۳۲.dll", EntryPoint="MoveFile",
ExactSpelling=false, CharSet=CharSet.Unicode,
SetLastError=true)]
static extern bool MoveFile(
string sourceFile, string destinationFile);
```

صفت DllImport کلاس برای مشخص کردن یک متد مدیریت نشده استفاده می‌شود که از طریق P/Invoke فراخوانی خواهد شد. پارامترهای آن به صورت زیر هستند.

DllName: نام Dll ای که احضار می‌کنید.

EntryPoint: نام متدی از Dll که فراخوانی می‌کنید را مشخص می‌کند.

ExactSpelling: به CLR اجازه می‌دهد اسامی مختلف متدها را با قراردادهای نامگذاری CLR تطابق دهد.

CharSet: نحوه‌ی مارشال شدن آرگومان‌های رشته‌ی به متد را نشان می‌دهد.

SetLastError: با تنظیم این خصوصیت به مقدار true، به شما اجازه داده می‌شود Marshal.GetLastWin۳۲Error را فراخوانی کنید و آخرین خطایی که زمان احضار این متد رخ داده است بررسی کنید.

حال می‌توانید با فراخوانی متد MoveFile به صورت زیر تغییر نام را انجام دهید.

```
Tester.MoveFile(file.FullName, file.FullName + ".bak");
```

مثال ۲۹-۱۰

```

#region Using directives
using System;
using System.Collections.Generic;
using System.IO;
using System.Runtime.InteropServices;
using System.Text;
#endregion
namespace UsingPInvoke
{
    class Tester
    {
        // declare the WinAPI method you wish to P/Invoke
        [DllImport( "kernel۳۲.dll", EntryPoint = "MoveFile",
        ExactSpelling = false, CharSet = CharSet.Unicode,
        SetLastError = true )]
        static extern bool MoveFile(
        string sourceFile, string destinationFile );
        public static void Main( )
        {
            // make an instance and run it
            Tester t = new Tester( );
            string theDirectory = @"c:\test\media";
            DirectoryInfo dir =
            new DirectoryInfo( theDirectory );
            t.ExploreDirectory( dir );
        }
        // Set it running with a directory name
        private void ExploreDirectory( DirectoryInfo dir )
        {
            // make a new subdirectory
            string newDirectory = "newTest";
            DirectoryInfo newSubDir =
            dir.CreateSubdirectory( newDirectory );
            // get all the files in the directory and
            // copy them to the new directory
            FileInfo[] filesInDir = dir.GetFiles( );
            foreach ( FileInfo file in filesInDir )
            {
                string fullName = newSubDir.FullName +
                "\\\" + file.Name;
                file.CopyTo( fullName );
                Console.WriteLine( "{۰} copied to newTest",
                file.FullName );
            }
            // get a collection of the files copied in
            filesInDir = newSubDir.GetFiles( );
            // delete some and rename others
            int counter = ۰;
            foreach ( FileInfo file in filesInDir )
            {
                string fullName = file.FullName;
                if ( counter++ % ۲ == ۰ )
                {
                    // P/Invoke the Win API
                    Tester.MoveFile( fullName, fullName + ".bak" );
                    Console.WriteLine( "{۰} renamed to {۱}",
                    fullName, file.FullName );
                }
                else
                {
                    file.Delete( );
                    Console.WriteLine( "{۰} deleted.",

```

```
fullName );  
}  
}  
// delete the subdirectory  
newSubDir.Delete( true );  
}  
}  
}  
Output (excerpt):  
c:\test\media\newTest\recycle.wav renamed to  
c:\test\media\newTest\recycle.wav  
c:\test\media\newTest\ringin.wav renamed to  
c:\test\media\newTest\ringin.wav
```

اسمبلی‌ها و نسخه‌سازی

آنچه که در این فصل یاد خواهید گرفت:

- استفاده از اسمبلی‌ها به جای DLL‌ها
- انواع اسمبلی‌ها و نحوه‌ی ایجاد و کاربرد آنها
- کاربرد اسمبلی‌های خصوصی و اشتراکی
- اجزای یک اسمبلی
- استفاده از اسمبلی‌ها در نسخه‌سازی
- استفاده از اسامی قوی

اسمبلی، واحد اصلی توسعه در .NET است. اسمبلی یک کلکسیون از فایل‌ها است که بصورت یک فایل منفرد DLL یا اجرایی EXE ظاهر می‌گردد. همانطور که می‌دانید DLL‌ها کلکسیون‌هایی از کلاس‌ها و متدها هستند که فقط در صورت نیاز، به برنامه‌ی در حال اجرا پیوند داده می‌شوند.

اسمبلی‌ها، واحد .NET در استفاده‌ی مجدد، نسخه‌سازی، امنیت و توسعه هستند. این فصل، اسمبلی‌ها را بطور دقیق بررسی می‌کند که مباحث معماری و محتوای اسمبلی‌ها، اسمبلی‌های خصوصی و اسمبلی‌های اشتراکی را شامل می‌شود. اسمبلی‌ها علاوه بر کدهای برنامه، می‌توانند منابعی همچون فایل‌های GIF، تعریف نوع داده‌ی هر کلاس و فراداده‌های دیگری برای کد و داده را شامل شوند.

۳۰-۱-۱ فایل‌های PE^۱

در روی دیسک، اسمبلی‌ها فایل‌های اجرایی قابل حمل هستند. فایل‌های PE مطلب جدیدی نیستند. قالب یک فایل PE در .NET، دقیقاً شبیه فایل PE عادی ویندوز است. فایل‌های PE بصورت DLL‌ها و EXE‌ها پیاده‌سازی می‌شوند. از دید فیزیکی، اسمبلی‌ها یک یا چند ماژول را در بردارند. ماژول‌ها بخش‌های اصلی اسمبلی‌ها هستند. ماژول‌ها نمی‌توانند به تنهایی اجرا شوند. آنها باید در داخل اسمبلی‌ها ترکیب شوند تا قابل اجرا گردند. شما محتوای یک اسمبلی را بصورت یک واحد قابل استفاده‌ی مجدد یا قابل توسعه خواهید دید. اسمبلی‌ها بر اساس درخواست شما بارگذاری می‌شوند و در صورت عدم نیاز بارگذاری نخواهند شد.

۳۰-۱-۱-۱ فراداده

فراداده‌ها اطلاعاتی هستند که در اسمبلی ذخیره می‌شوند و انواع داده‌ای و متدهای اسمبلی را شرح می‌دهند و اطلاعات مفید دیگری درباره‌ی اسمبلی فراهم می‌سازند. اسمبلی‌ها خود - شرح^۱ خوانده می‌شوند، چون فراداده‌ها بطور کامل محتویات ماژول را شرح می‌دهند.

^۱ Portable Executable

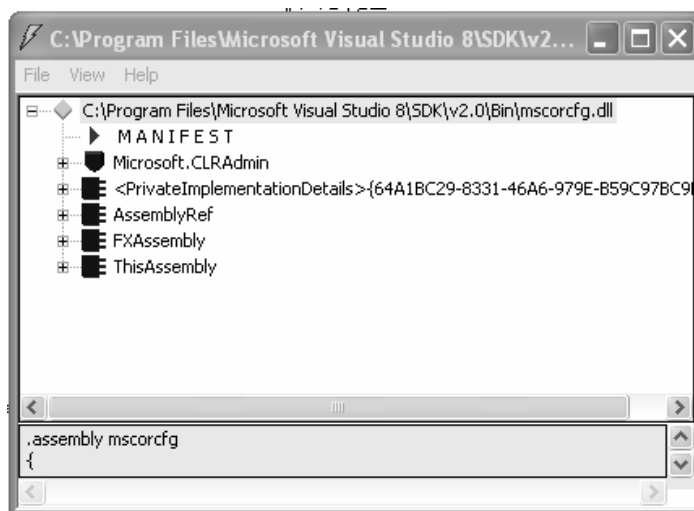
۳۰-۱-۲- محدوددهای امنیت

اسمبلی‌ها محدودیت‌های امنیت را به خوبی محدودیت‌های نوع داده شکل می‌دهند. یک اسمبلی محدوددهی دامنه‌ای برای انواع داده‌ای داخل خود است و تعریف نوع داده نمی‌تواند خلاف اسمبلی‌ها رفتار کند. می‌توانید با اضافه کردن یک ارجاع به اسمبلی مورد نیاز، می‌توانید محدودیت‌های انواع داده‌ای سراسر اسمبلی بیان کنید. این عمل در IDE یا روی خط فرمان در زمان کامپایل امکان‌پذیر است. نمی‌توانید یک نوع داده‌ای تعریف کنید که در دو اسمبلی باشد. معرف دستیابی `internal`، دسترسی را به داخل اسمبلی محدود می‌کند.

۳۰-۱-۳- اظهارنامه^۲ها

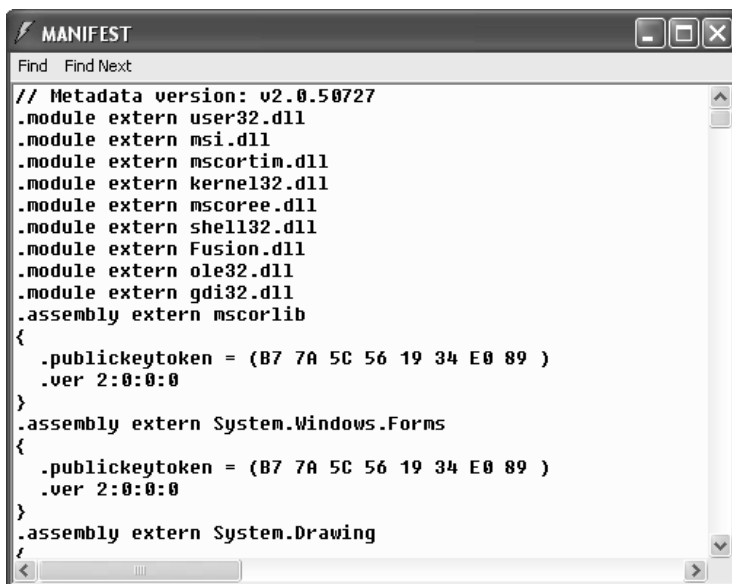
هر اسمبلی یک اظهارنامه را به عنوان بخشی از فراداده خود دارد. چیزهایی که در یک اسمبلی وجود دارند: اطلاعات شناسایی (نام، نسخه، غیره)، یک لیست از انواع داده‌ای و منابع داخلی اسمبلی، یک لیست از ماژول‌ها، یک نقشه برای اتصال نوع داده‌های `public` به کد پیاده‌سازی آنها و یک لیست از اسمبلی‌هایی که این اسمبلی به آنها ارجاع دارد. حتی ساده‌ترین برنامه نیز یک اظهارنامه دارد. با استفاده از نرم‌افزار سودمند `ILDASM`، می‌توانید اظهارنامه را بررسی کنید. زمانی که اظهارنامه را در `ILDASM` باز کنید، اظهارنامه یک برنامه اجرایی می‌تواند شبیه شکل ۳۰-۱ ظاهر گردد.

شکل ۳۰-۱



به اظهارنامه توجه کنید (خط دوم از بالا). با دابل کلیک روی اظهارنامه یک پنجره `Manifest` همانند شکل ۳۰-۲ باز می‌شود.

^۱ self describing^۲ manifest



شکل ۳۰-۲

این فایل بصورت یک نقشه از محتویات اسمبلی بکار گرفته می‌شود. در خط اول می‌توانید ارجاع به اسمبلی mscorlib را ببینید که توسط هر اسمبلی دیگر نیز ارجاع داده می‌شود. اسمبلی mscorlib، اسمبلی کتابخانه هسته NET است و روی هر NET Platform در دسترس است.

خط بعدی اسمبلی، یک ارجاع به اسمبلی یک برنامه است. می‌توانید ببینید که این اسمبلی فقط یک ماژول را در بردارد. در حال حاضر می‌توانید مابقی فراداده را نادیده بگیرید.

۳۰-۲-اسمبلی‌های چند ماژولی

اسمبلی‌ها می‌توانند بیش از یک ماژول را شامل شوند. البته این عمل به وسیلهٔ VS ۲۰۰۵ پشتیبانی نمی‌شود. یک اسمبلی تک ماژولی، فقط یک فایل دارد که می‌تواند یک فایل EXE یا DLL باشد. این تک ماژول، همهٔ انواع داده‌ای و پیاده‌سازی برنامه را در بردارد. اظهارنامه اسمبلی به وسیله‌ی این ماژول تعبیه می‌شود.

هر ماژول اظهارنامهٔ خودش را دارد که از اظهارنامهٔ اسمبلی مجزا است. اظهارنامهٔ اسمبلی، اسمبلی‌های ارجاع شده توسط این اسمبلی خاص را لیست می‌کند. به علاوه، اگر ماژول انواع داده‌ای را اعلان کند، آنها به همراه کد پیاده‌سازی کنندهٔ ماژول در اظهار نامه لیست می‌شوند. یک ماژول می‌تواند منابعی همچون تصاویر مورد نیاز ماژول را در بگیرد.

یک اسمبلی چندماژولی، چندین فایل (صفر یا بیشتر از صفر فایل EXE یا DLL) را در بردارد. در این حالت، اظهارنامه‌ی اسمبلی می‌تواند در یک فایل مستقل قرار گیرد یا در یکی از ماژول‌ها تعبیه گردد. زمانی که اسمبلی ارجاع داده می‌شود، در زمان اجرا، فایل در بردارنده‌ی اظهارنامه بارگذاری می‌شود و سپس ماژول‌های مورد نیاز را بارگذاری می‌کند.

۳۰-۲-۱-ایجاد یک اسمبلی چند ماژولی

برای نشان دادن اسمبلی‌های چندماژولی، مثال زیر یک جفت ماژول بسیار ساده ایجاد می‌کند که آنها در یک اسمبلی واحد ترکیب می‌گردند. ماژول اول یک کلاس Fraction است. این کلاس ساده ایجاد و دستکاری کسرهای عمومی را برای شما ممکن می‌سازد. مثال ۳۰-۱ را ملاحظه کنید.

مثال ۳۰-۱

#region Using directives


```
using System;
using System.Collections.Generic;
using System.Text;
#endregion
namespace ProgCS
{
    public class Fraction
    {
        private int numerator;
        private int denominator;
        public Fraction( int numerator, int denominator )
        {
            this.numerator = numerator;
            this.denominator = denominator;
        }
        public Fraction Add( Fraction rhs )
        {
            if ( rhs.denominator != this.denominator )
            {
                return new Fraction(
                    rhs.denominator * numerator +
                    rhs.numerator * denominator,
                    denominator * rhs.denominator );
            }
            return new Fraction(
                this.numerator + rhs.numerator,
                this.denominator );
        }
        public override string ToString( )
        {
            return numerator + "/" + denominator;
        }
    }
}
```

توجه کنید که کلاس Fraction در فضای نامی ProgCS است. نام کامل کلاس ProgCS.Fraction است. کلاس Fraction در سازه خود دو مقدار numerator و denominator را می‌گیرد. متد دیگر آن Add() است که یک Fraction دیگر گرفته و با قبلی جمع می‌کند. فرض بر این است که هر دو کسر، denominator مشترکی دارند. این کلاس ساده است، اما عملکرد ضروری این مثال را نشان خواهد داد.

کلاس دوم کلاس MyCalc است که مثال ۳۰-۲ آن را نشان می‌دهد.

مثال ۳۰-۲

```
#region Using directives
using System;
using System.Collections.Generic;
using System.Text;
#endregion
namespace ProgCS
{
    public class MyCalc
    {
        public int Add( int val1, int val2 )
        {
            return val1 + val2;
        }
        public int Mult( int val1, int val2 )
        {
            return val1 * val2;
        }
    }
}
```

از طرف دیگر، MyCalc یک کلاس بسیار محروم برای ساده نگه داشتن مطلب است، توجه کنید که کلاس MyCalc نیز در فضای نامی ProgCS است.

این موارد برای ایجاد یک اسمبلی کافی نیستند. یک فایل AssemblyInfo.CS برای اضافه کردن مقداری فراداده به اسمبلی بکار برید. می‌توانید فایل AssemblyInfo.CS را خودتان بنویسید. اما روش ساده‌تر استفاده از VS برای ایجاد اتوماتیک آن است.

VS فقط اسمبلی‌های تک ماژولی ایجاد می‌کند. می‌توانید یک منبع چند ماژولی را با استفاده از گزینه addModule/ در خط فرمان ایجاد کنید. ساده‌ترین راه، کامپایل و ایجاد یک اسمبلی چند ماژولی، با استفاده از makefile است. که می‌توانید با NotePad یا هر ویرایشگر دیگر ایجاد کنید. اگر با MAKEFILE آشنا نیستید، نگران نباشید. این تنها مثالی است که makefile نیاز دارد.

مثال ۳-۳۰ یک makefile حاصل را نشان می‌دهد. برای اجرای این مثال، یک کپی از makefile را با کپی فایل‌های Fraction.CS و Calc.CS و AssemblyInfo.CS در یک فهرست قرار دهید. یک پنجره فرمان NET را باز کرده و با فرمان CD به فهرست مورد نظر وارد شوید. برنامه‌ی nmake را بدون هیچ سوئیچ در خط فرمان احضار کنید. MySharedAssembly.DLL را در زیر فهرست Bin\ خواهید یافت.

مثال ۳-۳۰

```
ASSEMBLY= MySharedAssembly.dll
BIN=. \bin
SRC=.
DEST=. \bin
CSC=csc /nologo /debug+ /d:DEBUG /d:TRACE
MODULETARGET=/t:module
LIBTARGET=/t:library
EXETARGET=/t:exe
REFERENCES=System.dll
MODULES=$(DEST)\Fraction.dll $(DEST)\Calc.dll
METADATA=$(SRC)\AssemblyInfo.cs
all: $(DEST)\MySharedAssembly.dll
# Assembly metadata placed in same module as manifest
$(DEST)\$(ASSEMBLY): $(METADATA) $(MODULES) $(DEST)
$(CSC) $(LIBTARGET) /addmodule:$(MODULES: =;) /out:$@ %s
# Add Calc.dll module to this dependency list
$(DEST)\Calc.dll: Calc.cs $(DEST)
$(CSC) $(MODULETARGET) /r:$(REFERENCES: =;) /out:$@ %s
# Add Fraction
$(DEST)\Fraction.dll: Fraction.cs $(DEST)
$(CSC) $(MODULETARGET) /r:$(REFERENCES: =;) /out:$@ %s
$(DEST):
!if !EXISTS($(DEST))
mkdir $(DEST)
!endif
```

makefile با تعریف اسمبلی مورد نظر جهت ساخت آغاز می‌گردد.

```
ASSEMBLY= MySharedAssembly.dll
```

سپس فهرست‌های مورد استفاده شما را تعریف می‌کند. خروجی را در فهرست BIN در زیرفهرست جاری قرار می‌دهد و کد منبع را از فهرست جاری بازیابی می‌کند.

```
SRC=.
DEST=. \bin
```

اسمبلی را بصورت زیر ایجاد کنید.

```
$(DEST)\$(ASSEMBLY): $(METADATA) $(MODULES) $(DEST)
$(CSC) $(LIBTARGET) /addmodule:$(MODULES: =;) /out:$@ %s
```

این دستور اسمبلی را در فهرست مقصد bin قرار می‌دهد. این دستور به nmake می‌گوید، مقصد ساخت به مقصدهای سه ساخت دیگر لیست شده وابسته هستند. آن دستورات مورد نیاز خط فرمان را برای ایجاد اسمبلی فراهم می‌سازد.

فراداده بصورت زیر تعریف می‌شود.

```
METADATA=$(SRC)\AssemblyInfo.cs
```

ماژول‌ها بصورت دو تا DLL تعریف می‌شوند.

```
MODULES=$(DEST)\Fraction.dll $(DEST)\Calc.dll
```

خط کامپایل، کتابخانه را ایجاد می‌کند و ماژول‌ها را به آن اضافه می‌کند. سپس خروجی را در یک فایل اسمبلی MySharedAssembly.DLL قرار می‌دهد.

```
$(DEST)\$(ASSEMBLY): $(METADATA) $(MODULES) $(DEST)
$(CSC) $(LIBTARGET) /addmodule:$(MODULES: =;) /out:$@ %s
```

برای انجام این کار، لازم است nmake نحوه‌ی ایجاد ماژول‌ها را بداند. با گفتن نحوه‌ی ایجاد Calc.DLL به nmake شروع کنید. شما فایل منبع Calc.CS را لازم دارید. خط فرمان زیر را برای ایجاد DLL مورد نظر به nmake بگویید.

```
(REFERENCES:=;) $(DEST)\CALC.DLL:CALC.CS $(DEST) $(CSC) $(MODULETARGET) /R:)$
/OUT:$@%S
```

سپس همان کار را برای Fraction.DLL انجام دهید.

```
$(DEST)\FRACTION.DLL:FRACTION.CS $(DEST) $(CSC) $(MODULETARGET) )$
```

```
(REFERENCES:=;) /OUT:$@%S$R/R:
```

با اجرای nmake روی این، سه DLL ایجاد می‌شود: Calc.DLL، Fraction.DLL و MySharedAssembly.DLL. اگر MySharedAssembly.DLL را با ILDASM باز کنید، خواهید دید که آن فقط یک اظهار نامه دارد (شکل ۳-۳۰).

شکل ۳-۱۷



اگر اظهارنامه را بررسی کنید، فراداده‌های مربوط به کتابخانه‌های ایجاد شده را خواهید دید (همانطور که در شکل ۴-۳۰ می‌بینید).

شکل ۴-۳۰



ابتدا یک اسمبلی خارجی برای کتابخانه هسته (mscorlib) خواهید دید که با دو مازول ProgCS.Calc و ProgCS.Fraction دنبال می‌شوند.

حال شما یک اسمبلی دارید که سه فایل DLL را در بر دارد: MySharedAssembly.DLL به همراه اظهارنامه و Calc.DLL و Fraction.DLL به همراه انواع داده‌ای و پیاده سازی آنها.

۳-۲-۳۰ آزمون اسمبلی

برای کاربرد این مازول‌ها، یک برنامه‌ی راه انداز ایجاد خواهید کرد. مثال ۳۰-۴ را مشاهده کنید. این برنامه را در فایل Test.CS همانند مازول‌های دیگر در همان فهرست ذخیره کنید.

مثال ۳۰-۴

```
namespace Programming_CSharp
{
    using System;
    public class Test
    {
        // main will not load the shared assembly
        static void Main( )
        {
            Test t = new Test( );
            t.UseCS( );
            t.UseFraction( );
        }
        // calling this loads the myCalc assembly
        // and the mySharedAssembly as well
        public void UseCS( )
        {
            ProgCS.myCalc calc = new ProgCS.myCalc( );
            Console.WriteLine("3+5 = {0}\n3*5 = {1}",
                calc.Add(3,5), calc.Mult(3,5));
        }
        // calling this adds the Fraction assembly
        public void UseFraction( )
        {
            ProgCS.Fraction frac1 = new ProgCS.Fraction(3,5);
            ProgCS.Fraction frac2 = new ProgCS.Fraction(1,5);
            ProgCS.Fraction frac3 = frac1.Add(frac2);
            Console.WriteLine("{0} + {1} = {2}",
                frac1, frac2, frac3);
        }
    }
}
```

خروجی:

3+5 = 8

3*5 = 15

3/5 + 1/5 = 4/5

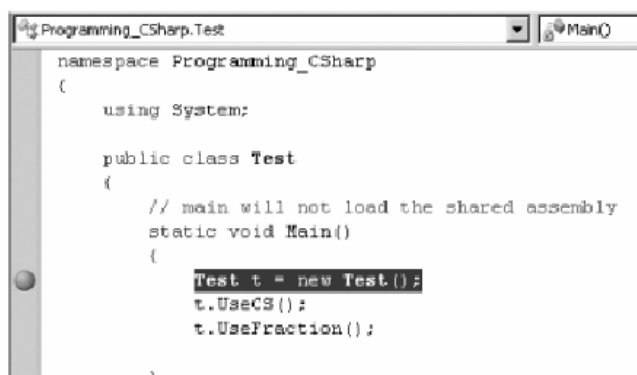
برای درک اهداف این ارائه، مهم است هیچ کد وابسته به ماژول‌هایتان در Main() قرار ندهید. چون نمی‌خواهید ماژول‌هایتان در زمان بارگذاری Main()، بارگذاری شوند. بنابراین هیچ شیء از Calc و Fraction در Main() قرار داده نمی‌شوند. زمانی که UseCalc و UseFraction را فراخوانی می‌کنید، قادر هستید بارگذاری تک‌تک ماژول‌ها را ببینید.

بارگذاری اسمبلی

یک اسمبلی بوسیله AssemblyResolver به داخل برنامه کاربردی بارگذاری می‌شود. AssemblyResolver بطور اتوماتیک بوسیله چارچوب .NET فراخوانی می‌شود. نمی‌توانید آن را صریحاً فراخوانی کنید. کار آن بارگذاری برنامه شما است.

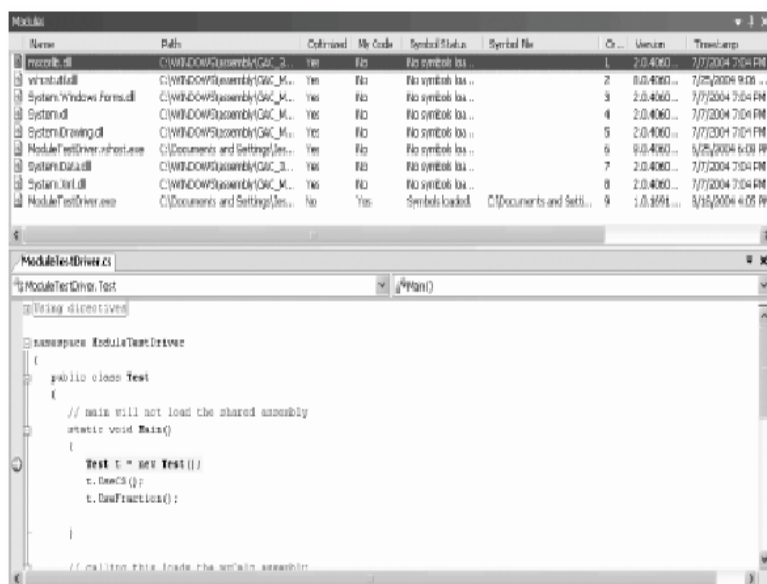
سه DLL ایجاد شده باید در همان فهرست مثال ۳۰-۴ ایجاد و در زیرفهرست bin قرار گیرند. همانند شکل ۵-۱۷ یک نقطه توقف در خط دوم Main() قرار دهید.

شکل ۳۰-۵



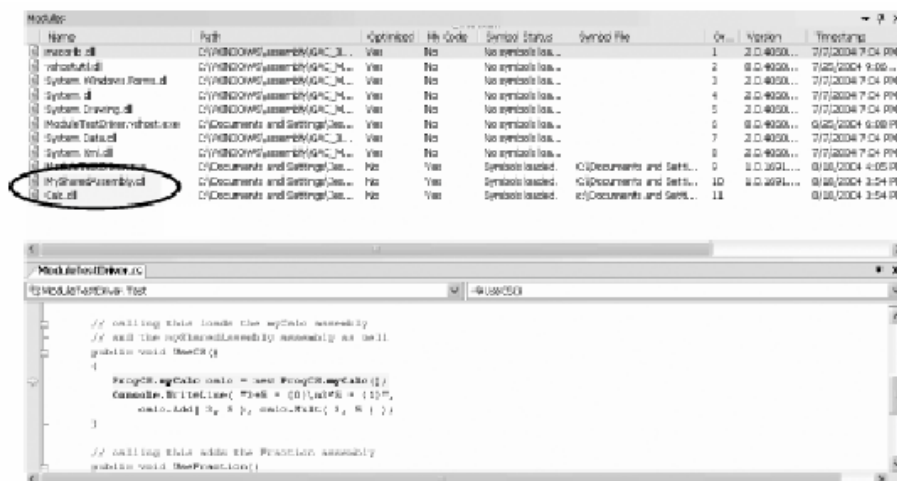
تا نقطه شکست اجرا کرده و پنجره ماژول‌ها را باز کنید. فقط دو ماژول ما بارگذاری می‌شود (شکل ۳۰-۶ را ببینید).

شکل ۳۰-۶



اگر در محیط VS.NET کار نمی‌کنید، دستور Launch .System.Diagnostics.Debugger () را قبل از خط دوم (Main) قرار دهید. این دستور استفاده از اشکال‌یاب را ممکن می‌سازد. (برنامه Test.CS را با گزینه های debug/ و r:MySharedAssembly.DLL کامپایل کنید).

تا فراخوانی اولین متد پیش رفته و پنجره ماژول را ببینید. به محض اینکه، به دستور UseCS وارد شوید، AssemblyLoader تشخیص می‌دهد یک ماژول از MySharedAssembly.DLL مورد نیاز است پس DLL بارگذاری می‌شود و از اظهارنامه اسمبلی در می‌یابد که ماژول Calc.DLL را لازم دارد. این ماژول همانند شکل ۳۰-۷ بارگذاری می‌شود.



زمانی که به دستور UseFraction وارد شوید، DLL نهایی نیز بارگذاری می‌شود.

۳۰-۲-۳- اسمبلی‌های خصوصی

اسمبلی‌ها به دو صورت استفاده می‌شوند: خصوصی و اشتراکی. اسمبلی‌های خصوصی فقط در یک برنامه کاربردی استفاده می‌شوند. اسمبلی‌های اشتراکی مابین چندین برنامه به اشتراک گذاشته می‌شوند.

همه اسمبلی‌هایی که تا بحال ساخته‌اید، خصوصی هستند. بطور پیش فرض، کامپایلر C# هنگام کامپایل، اسمبلی خصوصی ایجاد می‌کند. همه فایل‌های اسمبلی در پوشه یکسانی (یا در ساختار درختی از زیر پوشه‌ها) ذخیره می‌شوند. این درخت پوشه‌ها، از مابقی سیستم مجزا است و می‌توانید با کپی کردن پوشه‌ها به سیستم دیگر، این برنامه را به آن سیستم نیز منتقل کنید.

برای اسمبلی خصوصی می‌توانید هر نامی را انتخاب کنید. تداخل اسمی با اسمبلی‌های برنامه دیگر مهم نیست. چون این اسمبلی‌ها برای برنامه بصورت محلی استفاده می‌شوند.

در گذشته، DLL‌ها روی سیستم نصب می‌شدند و در رجستری ویندوز یک مشخصه ایجاد می‌شد و در بعضی موارد نصب مجدد برنامه روی ماشین دیگر قابل توجه بود. اما در صورت استفاده از اسمبلی‌ها، نصب یک برنامه به سادگی عمل کپی فایل‌ها به یک فهرست مناسب است.

۳۰-۲-۴- اسمبلی‌های اشتراکی

می‌توانید اسمبلی‌هایی ایجاد کنید و با برنامه‌های دیگر به اشتراک گذارید. در صورتی که یک کلاس یا کنترل کلی نوشتید که توسط توسعه‌دهندگان دیگر قابل استفاده باشد، اسمبلی‌های اشتراکی مفید هستند. اگر بخواهید یک اسمبلی را به اشتراک گذارید، باید نیازمندی‌های دقیقی را برآورده سازید:

اولاً: باید اسمبلی شما یک نام قوی داشته باشد. اسامی قوی، اسامی منحصر بفرد سراسری هستند.

- هیچ فرد دیگری نمی‌تواند نام قوی یکسانی تولید کند. چون اسمبلی تولید شده با کلید خصوصی تضمین می‌کند، نام متفاوتی با اسمبلی‌های تولید شده توسط کلیدهای خصوصی دیگر دارد.

ثانیاً: اسمبلی اشتراکی باید در مقابل پایمال شدن توسط نسخه‌های جدید محافظت گردد و هر نسخه جدید منتشر شده، باید شماره نسخه جدیدی داشته باشد.

نهایتاً: برای اشتراک گذاری اسمبلی، آن را در GAC قرار دهید. این بخشی از سیستم فایل است که به وسیله CLR برای نگه داشتن اسمبلی‌های اشتراکی کنار گذاشته می‌شود.

۳۰-۲-۵- پایان جهنم DLL

اسمبلی‌ها پایانی برای جهنم DLL‌ها هستند. این سناریو را به خاطر بسپارید. شما برنامه A را روی ماشین خود نصب کردید و آن یک تعداد از DLL‌ها را به فهرست Windows شما بارگذاری می‌کند. چند ماه با آن کار می‌کنید، سپس برنامه B را روی ماشین خود نصب می‌کنید. بصورت غیرقابل انتظار برنامه A اجرا نمی‌شود (برنامه A و B بهم مرتبط نیستند). پس چه اتفاقی افتاده است؟ برنامه B یکی از DLL‌های برنامه A را جایگزین کرده است و برنامه A غیرقابل اجرا شده است.

زمانی که DLL‌ها اختراع شدند، بدلیل صرفه‌جویی فضای دیسک، ایده بسیار خوبی به نظر می‌رسیدند. در فرضیه، DLL‌ها با نسخه‌های قبلی سازگار هستند و بطور اتوماتیک به نسخه‌های جدید DLL به روز می‌شوند. اما Pat Johnson می‌گوید در فرضیه، فرضیه و عمل یکسان هستند، اما در عمل هرگز اینطور نیست. زمانی که DLL جدید به کامپیوتر اضافه می‌شود، برنامه قدیمی که از DLL موجود در سیستم استفاده می‌کرده است، بطور ناگهانی به DLL جدیدی متصل می‌گردد، که با آن ناسازگار است و این برنامه برخلاف انتظار، متوقف می‌گردد. این پدیده منجر شده است، مشتریان از نصب نرم‌افزارهای جدید یا حتی به روز کردن برنامه‌های موجود دوری کنند. اما با حضور اسمبلی‌ها این کابوس از بین رفت.

۳۰-۳- نسخه‌ها

در NET. اسمبلی‌های اشتراکی بطور منحصر بفرد به وسیله نام و نسخه آنها معین می‌شوند. GAC اجازه می‌دهد نسخه‌های جدید و قدیمی یک اسمبلی در کنار همدیگر مورد استفاده قرار گیرند. این عمل فقط برای اسمبلی‌های اشتراکی اعمال می‌گردد و در مورد اسمبلی‌های خصوصی نیاز نیست.

ممکن است شماره نسخه یک اسمبلی شبیه این باشد: ۱:۰:۲۲۰۴:۲۱ (۴ عدد که با کالن از هم جدا شده‌اند). دو عدد اول (۱:۰) نسخه‌های اصلی و فرعی هستند. عدد سوم (۲۲۰۴) شماره ساخت و عدد چهارم (۲۱) شماره تجدید نظر است.

زمانی که دو اسمبلی شماره‌های اصلی و فرعی متفاوتی داشته باشند، آنها قانوناً باهم ناسازگار هستند. زمانی که شماره‌های ساخت متفاوتی دارند، ممکن است آنها سازگار باشند یا نباشند. زمانی که فقط شماره‌های تجدید نظر متفاوت باشند، آنها باهم سازگار می‌باشند. این یک تئوری است، اما AssemblyResolver در CLR این قانون را کنار گذاشته و فقط آنها را برای یادآوری توسعه دهنده بکار می‌گیرند.

۳۰-۳-۱- اسامی قوی

برای استفاده یک اسمبلی اشتراکی، باید دو نیاز را برآورده سازید:

- باید بتوانید نام دقیق اسمبلی مورد نظر خود را مشخص کنید.

- لازم است مطمئن شوید، در اسمبلی مورد استفاده فضولی نشده است و توسط سازنده واقعی آن تأیید شده باشد. بدین دلیل در زمان ایجاد اسمبلی شما یک امضاء دیجیتالی لازم دارید.

این دو نیازمندی بوسیله اسمی قوی برآورده می‌شوند. اسمی قوی بصورت سراسری و منحصر بفرد هستند و رمزنگاری کلید عمومی را بکار می‌برند. نام قوی یک رشته از ارقام هگزا است و توسط انسان قابل خواندن نیست.

برای ایجاد یک نام قوی، یک جفت کلید عمومی - خصوصی برای یک یا چند اسمبلی ایجاد می‌شود. یک درهم سازی (Hash) از نام‌ها و محتویات فایل‌های موجود در اسمبلی گرفته می‌شود. سپس Hash با کلید خصوصی اسمبلی رمزگذاری می‌شود و نشانه کلید عمومی (یک Hash هشت بیتی از کلید کامل) در قسمت اظهارنامه به همراه کلید عمومی قرار می‌گیرد. این عمل به امضاء کردن اسمبلی معروف است. زمانی که یک برنامه، اسمبلی را بارگذاری می‌کند. CLR برای کدگشایی Hash فایل‌های اسمبلی، از کلید عمومی استفاده می‌کند، تا مطمئن شود این فایل‌ها دستکاری نشده‌اند. این عمل از تداخل اسمی نیز جلوگیری می‌کند.

با استفاده از نرم‌افزار کمکی sn می‌توان یک نام قوی ایجاد کرد.

```
sn -k c:\myStrongName.snk
```

پرچم k- برای مشخص کردن ایجاد یک جفت کلید جدید است که به فایل مشخص شده نوشته می‌شود. می‌توانید این فایل را هر کجا که دوست دارید بکار ببرید. به خاطر دارید که یک نام قوی، رشته‌ای از بایت‌ها است که توسط انسان قابل خواندن نیست.

با استفاده از یک صفت می‌توانید این نام قوی را به اسمبلی خودتان اختصاص دهید.

```
using System.Runtime.CompilerServices;
[assembly: AssemblyKeyFile("c:\myStrongName.key")]
```

در حال حاضر، می‌توانید این کد را در بالای فایل خود قرار دهید تا نام قوی را به اسمبلی شما اختصاص دهد.

۳-۳-۳۰ GAC

زمانی که نام قوی را ایجاد کردید و آن را به اسمبلی خود اختصاص دادید، کاری که مانده است قرار دادن اسمبلی در GAC می‌باشد. با استفاده از نرم‌افزار کمکی gacutil، می‌توانید اسمبلی را به GAC اضافه کنید.

```
gacutil /i MySharedAssembly.dll
```

یا می‌توانید File Explorer را باز کرده و اسمبلی مورد نظر را به GAC بکشید. GAC در مسیر SystemRoot%\assembly% قرار دارد.

۳-۳-۳۰ ایجاد یک اسمبلی اشتراکی

بهترین روش فهم اسمبلی اشتراکی، ایجاد یک نمونه است. به پروژه چندمازولی اخیر برگردید (مثالهای ۳۰-۱ تا ۳۰-۴ را ببینید) و به فهرست مربوط به فایل‌های Calc.CS و Fraction.CS بروید.

سعی کنید این آزمایش را انجام دهید: فهرست bin را برای برنامه مورد نظر پیدا کرده و مطمئن شوید کپی محلی از فایل‌های MySharedAssembly.DLL را ندارید.

در اسمبلی ارجاع شده (MySharedAssembly) خصوصیت CopyLocal را false مقداردهی کنید. برنامه را اجرا کنید. آن برنامه با یک استثناء موفق نمی‌شود، چون آن نمی‌تواند اسمبلی را بارگذاری کند.

```
Unhandled Exception: System.IO.FileNotFoundException: File or assembly name
MySharedAssembly, or one of its dependencies, was not found.
File name: "MySharedAssembly"
at Programming_CSharp.Test.UseCS( )
at Programming_CSharp.Test.Main( )
```


حال DLLها را به فهرست برنامه مورد نظر کپی کنید و آن را دوباره اجرا کنید. حال برنامه شما بطور موفق اجرا می‌شود. حال اجازه دهید اسمبلی MySharedAssembly را یک اسمبلی اشتراکی کنیم. دو مرحله دارد: اولاً یک نام قوی برای اسمبلی ایجاد کنید و سپس اسمبلی را در GAC قرار دهید.

مرحله ۱: ایجاد یک نام قوی

در خط فرمان با استفاده از دستور زیر یک جفت کلید ایجاد کنید.

```
sn -k keyFile.snk
```

حال فایل AssemblyInfo.CS مربوط به MySharedAssembly.DLL را باز کرده و این خط را تغییر دهید.

```
[assembly: AssemblyKeyFile("")]
```

را بصورت زیر تغییر دهید.

```
[assembly: AssemblyKeyFile("keyFile.snk")]
```

این عمل فایل کلید را برای اسمبلی تنظیم می‌کند. فایل اسمبلی را مجدداً ایجاد کنید و سپس DLL منتج را در ILDASM باز کرده و اظهارنامه را باز کنید. شما باید یک کلید عمومی ببینید (همانطور که در شکل ۳۰-۸ می‌بینید).

شکل ۳۰-۸



بوسیله اضافه کردن نام قوی، این اسمبلی را امضاء کردید. برای نمایش اینکه اسامی GAC و ارجاع در اظهارنامه سرویس گیرنده تطابق دارند، نام قوی را از DLL خواهید گرفت. برای انجام این کار، به فهرست مربوط به DLL رفته و دستور زیر را در خط فرمان وارد کنید.

```
sn -T MySharedAssembly.dll
```

توجه کنید که SN به حالت حروف حساس است. جواب این دستور باید چیزی شبیه این باشد:

```
Public key token is 01fad8e0f0941a4d
```

مرحله دوم: قرار دادن اسمبلی اشتراکی در GAC.

مرحله بعدی کشیدن اسمبلی کتابخانه به داخل GAC است. بدین منظور، پنجره Explorer را باز کنید و به فهرست %SystemRoot%\بروید، زمانی که روی زیر فهرست Assembly دابل کلیک کنید، برنامه نمایش GAC توسط Explorer شروع می‌شود.

می‌توانید اسمبلی را به داخل GAC بکشید یا برنامه کمکی زیر را در خط فرمان احضار کنید.

```
Gacutil /i mySharedAssembly.dll
```

حال می‌توانید بررسی کنید که نشانه کلید عمومی در اینجا با نشانه بدست آمده از دستور sn مطابقت دارد؟ شکل ۳۰-۹ را ملاحظه کنید.

شکل ۳۰-۹

Assembly Name	Version	Culture	Public Key Token
Microsoft.StdFormat	7.0.3300.0		b03f5f7f11d50a3a
Microsoft.VisualBasic	8.0.1200.0		b03f5f7f11d50a3a
Microsoft.VisualBasic.Compatibility	8.0.1200.0		b03f5f7f11d50a3a
Microsoft.VisualBasic.Compatibility.Data	8.0.1200.0		b03f5f7f11d50a3a
Microsoft.VisualBasic.Vsa	8.0.1200.0		b03f5f7f11d50a3a
Microsoft.VisualBasic	8.0.1200.0		b03f5f7f11d50a3a
Microsoft.VisualBasic.ApplicationVerifier	1.0.0.0		b03f5f7f11d50a3a
Microsoft.VisualBasic.VSCodeParser	8.0.1200.0		b03f5f7f11d50a3a
Microsoft.VisualBasic.VSCodeProvider	8.0.1200.0		b03f5f7f11d50a3a
Microsoft.VisualStudio	2.0.3600.0		b03f5f7f11d50a3a
Microsoft.VisualStudio.CommandBars	8.0.0.0		b03f5f7f11d50a3a
Microsoft.VisualStudio.Configuration	2.0.3600.0		b03f5f7f11d50a3a

به محض اینکه این کار انجام شد. شما یک اسمبلی اشتراکی دارید که می‌تواند توسط هر سرویس گیرنده دستیابی شود. برنامه مورد نظر را نوسازی کرده و مجدداً آن را ایجاد کنید و اظهارنامه آنرا ببینید(همانطور که در شکل ۳۰-۱۰ نشان داده شده است).

شکل ۳۰-۱۰

```
.assembly extern MySharedAssembly
{
    .publickeytoken = (A5 92 9F 01 02 E0 C4 73 )
    .ver 1:0:535:29377
}
```

این MySharedAssembly است که بصورت یک اسمبلی خارجی لیست می‌شود و کلید عمومی آن با مقدار نشان داده شده در GAC مطابقت دارد.

ILDASM را بسته و کد خود را اجرا کنید. آن باید درست کار کند حتی اگر DLL های این کتابخانه در مسیر جاری نباشند. شما یک اسمبلی اشتراکی ایجاد و استفاده کردید.

۳۰-۳-۴-اسمبلی‌های مورد نیاز دیگر

اظهارنامه اسمبلی، ارجاع‌هایی به اسمبلی‌های دیگر دارد. هر ارجاعی، نام اسمبلی دیگر، شماره‌ی نسخه، فرهنگ مورد نیاز و نشانه‌ی کلید عمومی اسمبلی دیگر (یک امضاء دیجیتالی) را در بردارد.

فرهنگ، یک رشته است که مشخصه‌های ملی و زبان را برای کاربری که برنامه‌ی شما بکار می‌برد نمایش می‌دهد. در فرهنگ، فرمت تاریخ معین می‌شود. به عنوان مثال تاریخ با فرمت (سال / روز / ماه) یا (سال / ماه / روز) نمایش داده شود.

نصب ویژوال C# ۲۰۰۵

قبل از اینکه بتوانیم استفاده از VS و VC# را شروع کنیم، باید آن را در کامپیوتر خود نصب کنیم. VC# به گونه‌های مختلفی توسط مایکروسافت ارائه شده است. نسخه‌ی VC#ی که شما از آن استفاده می‌کنید، ممکن است به یکی از حالت‌های زیر باشد:

- به عنوان بخشی از VS ۲۰۰۵، که یک مجموعه از ابزارها و زبان‌های برنامه‌نویسی است. این مجموعه شامل VB، J# و نیز VC++ می‌شود. VS در نسخه‌های Professional، Standard، Tools For Office و VS Team System منتشر شده است. هر کدام یک از این نسخه‌ها نسبت به نسخه‌ی قبلی از امکانات و ابزارهای بیشتری برای برنامه‌نویسی بهره‌مند هستند.
- به عنوان نگارش Express: این نگارش نسخه‌ای از VC# است که شامل یک سری از امکانات و ویژگی‌های محدود در VS می‌شود.

شکل ۱



این دو نسخه از VC# با شما این امکان را می‌دهد تا برای ویندوز برنامه بنویسید. مراحل نصب هر دوی آنها کاملاً واضح است. در حقیقت، باید گفت VS آنقدر باهوش است که بفهمد برای اجرا شدن روی کامپیوتر شما به چه چیزهایی نیاز دارد.

مراحل نصب VC# ۲۰۰۵ با نسخه‌ی Team System

۱- با قرار دادن CD مربوط به VS ۲۰۰۵، برنامه‌ی نصب بطور اتوماتیک اجرا می‌شود. اگر اجرا نشد در درایو مربوط به CD برنامه‌ی setup.exe را اجرا کنید. بعد از اجرای برنامه باید صفحه‌ای مشابه شکل ۱ ببینید.

۲- این پنجره مراحل نصب را نشان می‌دهد. برای اجرای درست فرایند نصب، VS نیاز دارد که یک سری از برنامه‌های روی سیستم عامل را به روز کند. برنامه‌ی نصب لیستی از موارد مورد نیاز جهت نصب را به شما نشان می‌دهد و شما باید قبل از

نصب VS، این برنامه‌ها را نصب کنید. بعد از اینکه VS تغییرات لازم در سیستم را انجام داد، وارد نصب خود برنامه می‌شویم. برای این مرحله روی Install Visual Studio کلیک کنید.

۳- بعد از پذیرفتن قرارداد شرکت، روی Continue کلیک کنید تا به مرحله بعد بروید.

۴- در این مرحله روش‌های مختلف نصب VS در اختیار شما قرار می‌گیرد که عبارتند از:

- Default: این گزینه باعث می‌شود VS با ابزارهایی که بصورت پیش فرض انتخاب شده‌اند در سیستم نصب شود.
 - Full: با انتخاب این گزینه، VS و تمام ابزارهای جانبی آن به صورت کامل در سیستم شما نصب می‌شوند. اگر از نظر فضایی که با انتخاب این گزینه در سیستم شما اشغال می‌شود مشکلی ندارید، بهتر است که هنگام نصب، این گزینه را انتخاب کنید تا VS بصورت کامل نصب شود.
 - Custom: با انتخاب این گزینه، لیستی از تمام قسمت‌های موجود در VS نمایش داده می‌شوند و می‌توانید انتخاب کنید که کدام قسمت‌ها باید نصب شوند و کدامیک نباید نصب شوند.
- در این مرحله برای اینکه با قسمت‌های موجود در VS نیز آشنا شویم، گزینه‌ی Custom را انتخاب کرده و دکمه‌ی Next را فشار دهید.

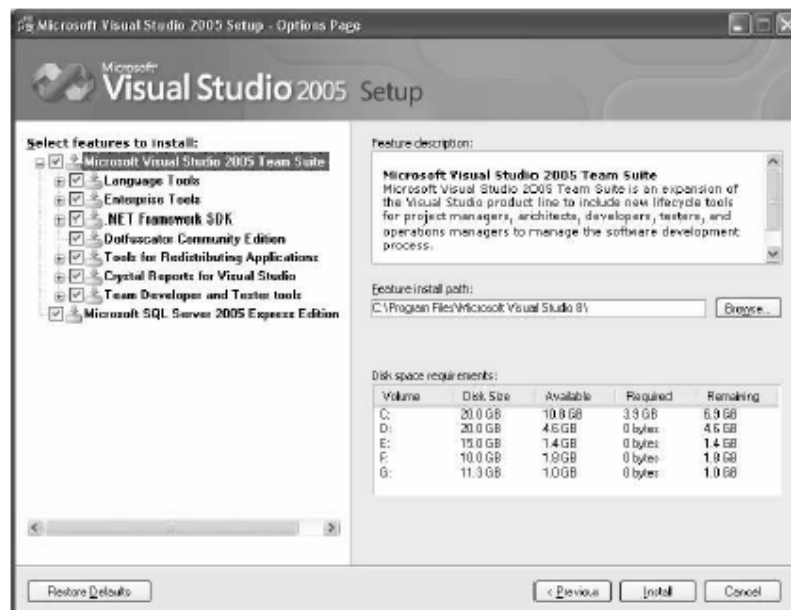
۵- با وارد شدن به این قسمت، لیست اجزای قابل نصب VS را می‌توانید ببینید. بدین ترتیب می‌توانید فقط قسمت‌هایی را که به آنها نیاز دارید نصب کنید. برای مثال، اگر فضای دیسک شما کم است و از VC++ ۲۰۰۵ استفاده نمی‌کنید، می‌توانید آن را نصب نکنید. در این قسمت همچنین می‌توانید مکان نصب برنامه را نیز تعیین کنید. در این قسمت باید SQL Server Express ۲۰۰۵ را انتخاب کنید.

برای هر گزینه از لیست سه قسمت وجود دارد که اطلاعات آن را نمایش می‌دهند:

- قسمت Feature Description یک طرح کلی و کارا از قسمت انتخاب شده را شرح می‌دهد.
- قسمت Feature Install Path مکانی که فایل‌های بخش انتخاب شده در آن نصب می‌شوند را نشان می‌دهد.
- در نهایت، قسمت Disk Space Requirements به شما نشان می‌دهد که با نصب بخش انتخاب شده، فضای دیسک شما چگونه تغییر می‌کند.

بعد از اینکه نصب VS به پایان رسید، در زمان اجرای VC# ۲۰۰۵ اطلاعات زیادی از دیسک به حافظه و برعکس منتقل می‌شوند. تعیین مقدار دقیق حافظه‌ی مورد نیاز ممکن نیست. برای برنامه‌نویسی بهتر، حداقل به ۱۰۰ MB فضای خالی نیاز است.

شکل ۲



۶- بعد از انتخاب قسمت‌های مورد نظر، روی گزینه Install کلیک کنید. حالا شما می‌توانید مقداری استراحت کنید، تا برنامه بر روی سیستم نصب شود. زمان مورد نیاز برای نصب بسته به قسمت‌های انتخاب شده بستگی دارد.

۷- هنگامی که نصب برنامه تمام شد، صفحه‌ای را مشاهده می‌کنید که پایان نصب را اطلاع می‌دهد. در این مرحله، برنامه‌ی نصب هر شمی که با آن روبرو شده باشد را گزارش می‌دهد. همچنین در این مرحله می‌توانید گزارش عملکرد را نیز بررسی کنید. روی گزینه‌ی Done کلیک کنید تا وارد بخش نصب مستندات یا راهنمای VS شویم.

محیط توسعه‌ی VC# ۲۰۰۵

در ابتدا جالب است بدانید که برای برنامه‌نویسی به زبان C#، به برنامه‌ی VC# ۲۰۰۵ نیازی ندارید. می‌توانید برنامه‌ی خود را با یک ویرایشگر متنی مانند NotePad نیز بنویسید. اما برنامه‌های VC# معمولاً طولانی هستند و نوشتن آنها با NotePad زمان زیادی را صرف می‌کند. انتخاب بهتر برای انجام این کار، استفاده از محیط توسعه‌ی VS است که به IDE معروف است. IDE مربوط به VS امکانات بسیار زیادی را در اختیار شما قرار می‌دهد که بطور یقین با استفاده از ویرایشگرهای متنی به آنها دسترسی نخواهید داشت.

شکل ۳



صفحه‌ی Profile Setup

یک IDE، محیطی است شامل یک سری ابزار که موجب سهولت کار توسعه و طراحی نرم‌افزار می‌شود. VS ۲۰۰۵ را اجرا کنید تا ببینید با چه چیزی روبرو می‌شوید. اگر شما مراحل پیش فرض نصب را انتخاب کرده‌اید، به منوی Start بروید و در شاخه‌ی All Programs منوی Microsoft Visual Studio ۲۰۰۵ گزینه‌ی Microsoft Visual Studio را اجرا کنید. صفحه‌ی شروع VS به سرعت نمایش داده می‌شود و بعد از آن پنجره‌ی Choose Default Environment Settings را خواهید دید. از لیست ظاهر شده گزینه‌ی Visual C# Development Settings را انتخاب کرده و روی Start Visual Studio کلیک کنید. محیط توسعه مایکروسافت همانند شکل ۳ نمایش داده خواهد شد.

منو

احتمالا اشتیاق زیادی برای شروع کد نویسی دارید. اما در ابتدا بهتر است کمی IDE را بررسی کنیم. گردش خودمان را در IDE از منوها و نوارهای ابزار شروع می‌کنیم. همانطور که می‌بینید منوها و نوار ابزارها در این برنامه نیز تفاوت چندانی با برنامه‌های دیگر مایکروسافت از قبیل Word و یا Excel ندارد.

نوار منوی VS ۲۰۰۵ به صورت دینامیک است یعنی بر حسب کاری که می‌خواهید انجام دهید یک سری از گزینه‌ها به منو اضافه شده و یا از آن حذف می‌شوند. وقتی فقط محیط IDE خالی را در مقابل خود دارید، منوی VS شامل گزینه‌های File, Edit, View, Data, Tools, Test, Window, Community و منوی Help است. اما هنگامی که کاربر یک پروژه را شروع کند، منوی کامل VS همانند شکل ۴ نمایش داده خواهد شد.

شکل ۴

File Edit View Project Build Debug Data Format Tools Test Window Community Help

در اینجا به توضیح کامل در مورد همه‌ی منوها نیازی نداریم. در طول این کتاب به مرور با تمامی آنها آشنا خواهید شد. اما در زیر برای آشنایی اولیه شرح مختصری از عملکرد هر یک از منوها آورده شده است:

- **File**: به نظر می‌رسد که همه برنامه‌های ویندوزی یک منوی فایل دارند. در این منو حداقل چیزی که پیدا می‌شود راهی برای خارج شدن از برنامه است. البته در منوی File این برنامه گزینه‌های بیشتری مثل بازکردن - بستن یا ذخیره کردن یک فایل خاص و یا تمام پروژه هم وجود دارد.
- **Edit**: این منو هم مثل برنامه‌های دیگر شامل گزینه‌هایی است که انتظار آن را دارید: Undo, Redo, Cut, Copy, Paste و Delete.
- **View**: منوی View به شما اجازه می‌دهد تا به سرعت به پنجره‌های موجود در IDE مثل Explorer Solution و پنجره‌ی Properties, پنجره‌ی Toolbar, Output ها و... دسترسی داشته باشید.
- **Project**: این منو به شما اجازه می‌دهد تا فایل‌های مختلف از قبیل فرم‌های جدید و یا کلاس‌ها را به برنامه‌ی خود اضافه کنید.
- **Build**: این منو زمانی مفید خواهد بود که برنامه‌ی خود را تمام کنید و بخواهید که آن را بدون استفاده از محیط VC# اجرا کنید (احتمالا از طریق منوی Start، مثل همه‌ی برنامه‌های ویندوزی دیگر از قبیل Word و یا Excel).
- **Debug**: این منو به شما اجازه می‌دهد تا برنامه خودتان را در داخل محیط VS خط به خط اجرا کنید. همچنین از طریق این منو شما به Debugger از VS ۲۰۰۵ نیز دسترسی خواهید داشت. به وسیله Debugger می‌توانید عملکرد کد خود را در هنگام اجرای برنامه خط به خط بررسی کرده و مشکلات آن را متوجه شوید.
- **Data**: آن به شما کمک می‌کند تا از اطلاعات به دست آمده از یک بانک اطلاعاتی استفاده کنید. البته این منو زمانی نمایش داده می‌شود که در حال کار بر روی قسمت‌های بصری برنامه خود باشید (در پنجره اصلی VS، قسمت Design) فعال باشد، نه زمانی که در حال نوشتن کد هستید.
- **Format**: این منو نیز فقط زمانی که در حال کار با قسمت‌های بصری برنامه باشید، نمایش داده می‌شود. به وسیله گزینه‌های این منو می‌توانید طریقه قرار گرفتن اشیای موجود در فرم برنامه (از قبیل TextBox ها، دکمه‌ها و...) را کنترل کنید.
- **Tools**: در این قسمت می‌توانید محیط VS ۲۰۰۵ IDE را کنترل و یا تنظیم کنید. همچنین پیوندی به برنامه‌های اضافی نصب شده در کنار VS نیز، در این قسمت وجود دارد.
- **Test**: منوی Test به شما اجازه می‌دهد برنامه‌هایی ایجاد کنید تا به وسیله آن بتوانید بعد از اتمام یک برنامه، قسمت‌های مختلف آن را از نظر کارایی و یا عملکرد بررسی کنید.
- **Window**: این منو در همه‌ی برنامه‌هایی که امکان باز کردن بیش از یک پنجره در هر لحظه را به کاربر می‌دهند، مثل Word و یا Excel، نیز وجود دارد. گزینه‌های موجود در این منو به شما اجازه می‌دهند که در بین پنجره‌های

موجود در IDE جابه‌جا شوید. نام پنجره‌هایی که در هر لحظه در محیط VS باز هستند، در پایین نوار ابزار نمایش داده می‌شوند که با کلیک کردن روی هر کدام از آنها پنجره‌ی مربوطه نمایش داده می‌شود.

- **Community:** این منو دسترسی به منابع برنامه‌نویسی، مکان‌هایی برای پرسیدن سوالات و نیز جستجو بین نمونه‌ی کدها را در اینترنت فراهم می‌کند.
- **Help:** منوی Help به شما اجازه‌ی دسترسی به مستندات VS2005 را می‌دهد. راه‌های زیادی برای دسترسی به این اطلاعات وجود دارند (برای مثال، از طریق محتویات، اندیس و یا جستجو). این منو همچنین دارای گزینه‌هایی برای وصل شدن به وب سایت میکروسافت، دریافت آخرین نسخه‌های به روزرسانی و همچنین گزارش دادن مشکلات برنامه است.

نوار ابزارها

نوار ابزارهای زیادی در IDE VS وجود دارند، مانند Formatting، Image Editor و Text Editor. برای حذف یا اضافه کردن این نوار ابزارها، می‌توانید از گزینه‌ی Toolbars در منوی View استفاده کنید. هر کدام از این نوار ابزارها، دسترسی سریع شما را به یک دستور پرکاربرد فراهم می‌کند. بدین صورت مجبور نخواهید بود که هر بار برای اجرای آن دستور منوها را جستجو کنید. برای مثال، گزینه‌ی **File→New→Project**.. از نوار منو، به وسیله‌ی نوار ابزار استاندارد (شکل ۵) قابل دسترسی است.

شکل ۵



نوار ابزار استاندارد به چند بخش که شامل گزینه‌های مرتبط به هم هستند تقسیم شده است. هر بخش به وسیله‌ی یک خط عمودی از بخش‌های دیگر تفکیک شده است. پنج آیکون اول، شامل کارهای عمومی بر روی فایل و یا پروژه هستند. که از طریق منوی **File** و یا منوی **Project** قابل دسترسی هستند، مانند باز کردن و یا ذخیره کردن فایل‌ها. گروه بعدی آیکون‌ها، برای ویرایش استفاده می‌شوند (**Cut**، **Copy** و **Paste**). گروه بعدی نیز برای لغو کردن آخرین عمل انجام شده، برگرداندن عمل لغو شده و ... است.

گروه چهارم از آیکون‌ها به شما اجازه می‌دهد اجرای برنامه‌ی خود را شروع کنید. (بوسیله‌ی مثلث سبز رنگ). در این قسمت همچنین می‌توانید پیگیربندی برنامه‌ی خود را مشخص کرده و یا نحوه‌ی اجرای آن را تعیین کنید.

گروه آخر از آیکون‌ها دسترسی سریع شما را به قسمت‌های مختلف VS مانند **Solution Explorer**، پنجره‌ی **Properties**، **Toolbox**، **Object Browser**، **Start Page** و یا صفحات دیگر فراهم می‌کند. اگر هر کدام یک از این پنجره‌ها بسته شده باشد، با کلیک بر روی آیکون آن در این قسمت، پنجره‌ی مورد نظر نمایش داده خواهد شد. نکته: اگر فراموش کردید که هر آیکون چه کاری انجام می‌دهد، اشاره‌گر ماوس خود را برای چند لحظه بر روی آن نگه دارید. بدین ترتیب کادری ظاهر می‌شود که نام آیکون مورد نظر را نمایش می‌دهد.