


شابان انتشارات گهر،  
تهران و مشهد و ...

کارنامه  
 هیدر ویلیامسون

XML

مرجع کامل XML

هیدر ویلیامسون



**پیمان آسمانی گهر**

در سال ۱۳۵۵ در تهران متولد شد. تحصیلات لیسانس و فوق لیسانس خود را در تهران گذراند. وی کارشناسی خود را در تهران در رشته کامپیوتر با گرایش برنامه‌نویسی پایان برد. آسمانی گهر علاوه بر تدریس در مراکز آموزشی عالی و فنی انقلاب اسلامی، دانش‌گهی شهید خدایی پور و ولی‌فقیه، مدیریت پروژه‌های نرم‌افزاری کلان را نیز عهده‌دار بوده‌است.

از آثار چاپ شده وی علاوه بر ترجمه کتاب مرجع کامل XML، مرجع کامل JSP ۲.۰ را نیز ترجمه و منتشر ساخته‌است.

JSP تکنولوژی طراحی صفحات وب برای برنامه‌نویسان جاوا است. این کتاب برای برنامه‌نویسان و دانش‌جویان کامپیوتر کاربرد فراوان داشته‌است.

**مرجع کامل XML**

کامل‌ترین مرجع موجود XML

ایجاد استاندارد ساده و پیچیده XML و پیاده‌سازی برنامه‌های کاربردی مبتنی بر XML از نکات نهمی هر آنچه جهت کار با این زبان نشانه‌گذاری نیاز دارید. نمونه استفاده از CSS و XSLT را خواهید آموخت. کاربرد Schema را خواهید دید. نمونه استفاده از XLink و XPath جهت انتخاب بخشی از سند را مشاهده خواهید کرد.

بررسی استانداردهای مربوط به نیازمندی‌های استفاده از XML

ایجاد استاندارد XML خوش ساخت

تعیین خصایص و داده‌های استاندارد XML

کاربرد شمای XML در برنامه‌های کاربردی جهت محدود کردن مقادیر


زمان و نحوه استفاده از شیوه‌نامه‌های CSS و XSL

نمونه‌های برنامه‌های کاربردی XML با استفاده از XPointer، XLink و ...

بررسی جزئیات شیوه‌نامه‌ها، DTD ها و روش‌های XML	بررسی راه کارهای استفاده از XLink و XPath جهت توسعه برنامه‌های کاربردی مبتنی بر XML	بررسی جزئیات کامل CSS و XSLT
----------------------------------------------------	-------------------------------------------------------------------------------------------	---------------------------------

پیمان آسمانی گهر

مدرس واحد آموزش عالی و فنی انقلاب اسلامی



به نام خدا

- پایان ترم ۱۲ نمره
- کار کلاسی ۲ نمره
- پروژه عملی ۶ نمره

## مقدمه

از زمان ایجاد کامپیوتر تا امروز، برنامه‌نویسی پیشرفت بسیاری داشته و زبان‌های برنامه‌نویسی جدید به وجود آمده‌اند. زبان‌های برنامه‌نویسی جدیدتر ویژگی‌های زبان‌های برنامه‌نویسی قبلی را پوشش داده و ویژگی‌های جدیدی را به آن می‌افزاید. C# نیز از این قاعده مستثنی نیست. C# نیز مانند جاوا از زبان‌های C و C++ نشأت گرفته‌است. ایجاد زبان C به منزله‌ی شروع عصر برنامه‌نویسی نوین می‌باشد. زبان C توسط Denis Richie در دهه ۷۰ میلادی برای سیستم عامل یونیکس به وجود آمد. اگرچه با توجه به فضای امروزی درک این مسئله دشوار به نظر می‌رسد ولی ایجاد زبان C به منزله‌ی هوای تازه‌ای بود که برنامه‌نویسان مدت‌ها در انتظار آن بودند. زبان برنامه‌نویسی C از منطق ساخت‌یافته که در دهه ۶۰ میلادی معرفی شده بود، استفاده می‌کرد.

در سال‌های پایانی دهه ۷۰ میلادی گستردگی برنامه‌های کاربردی به حدی رسید که برنامه‌نویسی ساخت‌یافته و زبان C توانایی مدیریت آن را نداشته یا به سختی از عهده آن برمی‌آمدند. به همین دلیل نیاز به متدولوژی جدیدی احساس شد. این نیاز باعث به وجود آمدن برنامه‌نویسی شیء‌گرا (OOP) گردید. با استفاده از برنامه‌نویسی شیء‌گرا مدیریت پروژه‌های بزرگتر امکان پذیر می‌شد ولی اشکال این بود که زبان C برنامه‌نویسی شیء‌گرا را پشتیبانی نمی‌کرد. بنابر این نسخه شیء‌گرا C یا همان C++ به وجود آمد.

C++ توسط Bjarne Stroustrup در سال ۱۹۷۹ به وجود آمد. نام این زبان ابتدا C with classes نامیده شد، ولی در سال ۱۹۸۳ به C++ تغییر نام یافت.

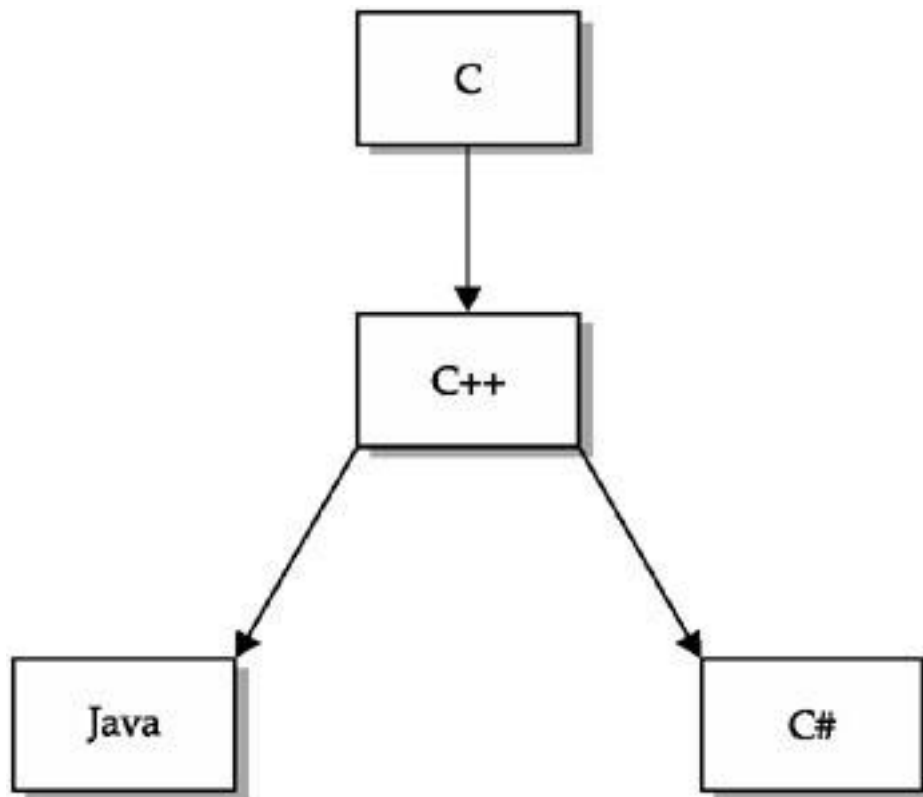
پیشرفت بعدی در زبان‌های برنامه‌نویسی معرفی زبان برنامه‌نویسی جاوا بود. توسعه‌ی زبان برنامه‌نویسی جاوا که در ابتدا Oak نامیده می‌شد، در سال ۱۹۹۱ در شرکت Sun آغاز شد. جاوا زبان برنامه‌نویسی می‌باشد که از C++ نشأت گرفته‌است. زبان برنامه‌نویسی جاوا تغییر عمده‌ای در هنر برنامه‌نویسی به وجود نیاورد. تفاوتی که زبان برنامه‌نویسی جاوا به وجود آورد ایجاد امکان نوشتن برنامه در یک محیط و اجرای آن در تمام محیط‌ها بود. در واقع هنگامی که برنامه‌های جاوا کامپایل می‌شوند، فایل اجرایی (exe) آن‌ها ایجاد نمی‌شود و کد معادلی که آن را بایت‌کد می‌نامیم، ایجاد خواهد شد. بایت‌کد ایجاد شده در زمان اجرا توسط ماشین مجازی جاوا تبدیل به فایل اجرایی خواهد شد. با توجه به این قابلیت زبان برنامه‌نویسی جاوا زبان مناسبی برای نوشتن برنامه‌های کاربردی در اینترنت است، زیرا اینترنت شبکه‌ای بسیار گسترده و دارای کامپیوترهای با ویژگی‌های متنوع و غیر قابل کنترل می‌باشد. بنابر این امکان نوشتن برنامه‌های کاربردی که در هر محیطی نوشته شوند، تنها راه حل ممکن است.

زبان C# نیز تلفیقی از زبان‌های برنامه‌نویسی C++ و جاوا می‌باشد. مزیتی که C# نسبت به جاوا دارد، امکان نوشتن برنامه با استفاده از mixed-language programming می‌باشد. این قابلیت امکان تعامل بین برنامه‌های کاربردی زبان‌های متفاوت را ایجاد می‌نماید.

زبان C# باید به موازات .Net Framework، مورد توجه قرار گیرد و نباید آن را به صورت مقوله‌ای جداگانه بررسی کرد. با استفاده از C# قادر به ایجاد یک صفحه‌ی وب پویا<sup>۱</sup>، یک Web Service، یک مولفه‌ی توزیع شده<sup>۲</sup>، یک

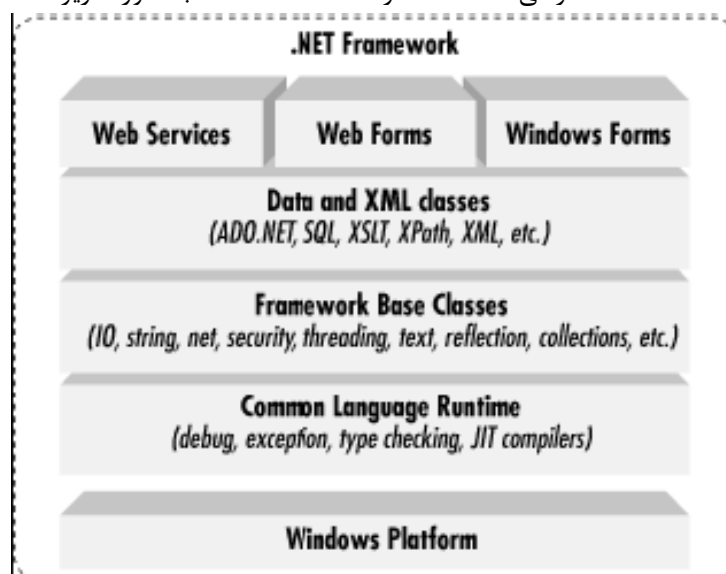
1- Dynamic  
2- Distributed Component

مولفه جهت دسترسی به بانک اطلاعاتی، یک برنامه‌ی Desktop یا حتی یک برنامه‌ی برای ابزارهای نظیر تلفن همراه یا دستیار دیجیتالی<sup>۱</sup> خواهید بود.



### .Net Framework

جهت اجرای برنامه‌های کاربردی که با استفاده از C# (یا هر کدام از زبان‌های Net) نوشته می‌شوند، وجود محیط اجرایی آن‌ها - Net Framework - الزامی است. ساختار Net Framework به صورت زیر است.



## CLR

یکی از مهم‌ترین عناصر Net است. <sup>۱</sup> CLR مدیریت اجرای برنامه‌های کاربردی Net را به عهده دارد. علاوه بر اجرای کد وظایف دیگری را نیز انجام می‌دهد. این وظایف عبارتند از: مدیریت حافظه و Threadها، امنیت و Interoperability کدهای زبان‌های مختلف.

## MSIL

هنگامی که یک برنامه C# را کامپایل می‌کنید، کد قابل اجرا تولید نخواهد شد، بلکه فایلی تولید می‌شود که محتوای آن شبه‌کدی به نام MSIL (Microsoft Intermediate Language) می‌باشد. MSIL مجموعه‌ای از دستورات را که به معماری CPU وابسته نیستند و حالت portable دارند، را تعریف می‌نماید. به عبارت دیگر MSIL یک زبان اسمبلی portable تعریف می‌کند.

تبدیل MSIL به کد اجرایی به عهده CLR است. عملیات تبدیل MSIL به کد اجرایی هنگام اجرای برنامه انجام می‌شود. برای این منظور CLR ابتدا JIT Compiler را فعال کرده و JIT Compiler تبدیل کد میانی به کد اجرایی را انجام می‌دهد.

در صورتی که کد برنامه توسط برنامه‌های که با زبان‌های دیگری نوشته شده‌است، استفاده شود لازم است که با استاندارد CLS هم‌خوانی داشته‌باشد. خصوصیات IL عبارتند از:

- شیء‌گرایی و استفاده از واسط
- استفاده از صفت‌ها
- امکان تشخیص تمایز بین نوع مقدار و نوع ارجاع
- نوع داده‌ای قوی
- رفع خطا با استفاده از Exception

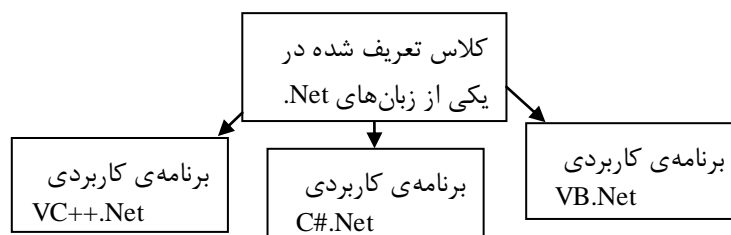
## نوع داده‌ای قوی

نوع داده‌ای همه‌ی متغیرها در IL به طور صریح مشخص می‌شود. IL اجازه انجام هر عملی را روی انواع داده‌ای مبهم نمی‌دهد.

## CTS

جهت پشتیبانی از ویژگی چندزبانی (Interoperability) از CTS<sup>۲</sup> استفاده می‌شود. CTS شامل مجموعه‌ای از انواع داده‌ها و استانداردهای برای ایجاد کلاس‌های سفارشی کاربر است. مزایای CTS به شرح زیر می‌باشد.

۱. ارث‌بری یک کلاس از کلاس دیگری که توسط زبان دیگری نوشته شده‌است.
۲. ایجاد شیء از کلاسی که توسط زبان دیگری نوشته شده‌است.
۳. استفاده از شیء یا اشاره‌گری به شیء به عنوان پارامتر متد کلاسی که توسط زبان دیگری نوشته شده‌است.
۴. اشکال‌زدایی برنامه که شامل اشیاء و کلاس‌هایی است که در زبان دیگری نوشته شده‌است.



1- Common Language Runtime

2- Common Type System

## CLS

CLS<sup>۱</sup> مجموعه‌ای از حداقل استانداردها است که توسط همه‌ی کامپایلرهای .NET پشتیبانی شده و ویژگی‌های عمومی را که بین زبان‌های مختلف مشترک است، توصیف می‌نماید. علاوه بر CTS جهت interoperability باید از قواعد CLS پیروی نمود. در غیر این صورت امکان استفاده از کلاس توسط زبان‌های دیگر وجود نخواهد داشت. چون IL زبان بسیار توانمندی است، نویسنده‌ی اکثر کامپایلرها ترجیح می‌دهند که قابلیت‌های کامپایلر را تنها به پشتیبانی از مجموعه‌ای از امکانات پیشنهاد شده توسط IL و CTS محدود کنند. این حالت تا زمانی که کامپایلر تمام موارد تعریف شده در CLS را پشتیبانی می‌کند، خوب است.

به طور مثال IL به بزرگ و کوچک بودن کاراکتر حساس است. برنامه‌نویسانی که با زبان‌های حساس به کاراکتر کار می‌کنند انعطاف بیشتری برای انتخاب نام دارند. VB 2008 حساس به کاراکتر نیست. بنابراین کامپایلرهای شخصی قدرت کافی برای پشتیبانی تمامی خصوصیات .NET را ندارد و اگر کلاس‌هایتان را محدود به ارائه تنها خصوصیات سازگار با CLS کنید، کد نوشته‌شده به هر زبان دیگر می‌تواند از کلاس‌های شما استفاده کند.

## ویژگی‌های CLS

۱. امکان استفاده از متدها و متغیرهای Global وجود ندارد.
۲. اسامی باید منحصر به فرد باشند.
۳. کلاس Exception شما باید زیر کلاس Exception باشد.
۴. CLR از اشاره گرهای پشتیبانی می‌کند.

## Assembly

یک ساختار منطقی که شامل کدهای کامپایل شده برای Net Framework می‌باشد. یک Assembly می‌تواند روی یک فایل یا چندین فایل به صورت DLL یا EXE ذخیره شود. امکان استفاده از اشیاء COM، فایل‌ها و شبه‌داده‌ها در Assembly وجود دارد.

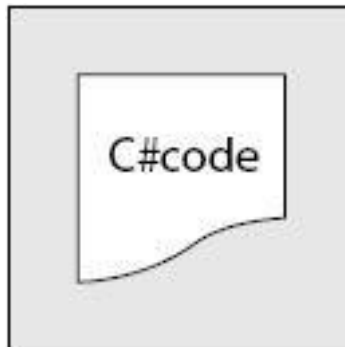
## ویژگی‌های Assembly

۱. خود توصیف (Self Describing): یک Assembly خود توصیف می‌باشد، زیرا شبه‌داده<sup>۲</sup> در مورد متغیرها و متدهای مورد استفاده Assembly می‌باشد.
۲. Side By Side: این ویژگی موجب می‌شود بتوانیم چند Assembly را در یک برنامه کاربردی نصب نماییم.
۳. Manifest: Version Dependency یک Assembly نسخه منابع مورد استفاده Assembly را در خود نگه می‌دارد. Manifest بخشی از Assembly می‌باشد که شامل شبه‌داده‌ها می‌باشد. هنگامی که به یک Assembly در برنامه کاربردی رجوع کنید نسخه‌ی Assembly در Manifest برنامه‌ی کاربردی ذخیره می‌شود.
۴. دامن‌های برنامه‌ی کاربردی: این ویژگی باعث می‌شود بتوانید چندین برنامه کاربردی را به صورت مستقل از یکدیگر اجرا نمایید. این برنامه‌های کاربردی قسمتی از یک Process هستند. به خاطر اینکه هر برنامه‌ی کاربردی مستقل از دیگر برنامه‌ها است، بروز خطا در یک برنامه تاثیری در دیگر برنامه‌ها نخواهد داشت.
۵. Zero-Impact Installation: همان‌طور که قبلاً هم ذکر کردیم، برای نصب یک Assembly نیازی به register کردن آن برای سیستم عامل وجود ندارد، فقط کافی است از دستورات copy، xcopy یا ... استفاده نمایید. این ویژگی را Zero-Impact Installation می‌نامیم.

1- Common Language Specification  
2- Meta Data

### مراحل تبدیل برنامه کاربردی Net به کد اجرایی

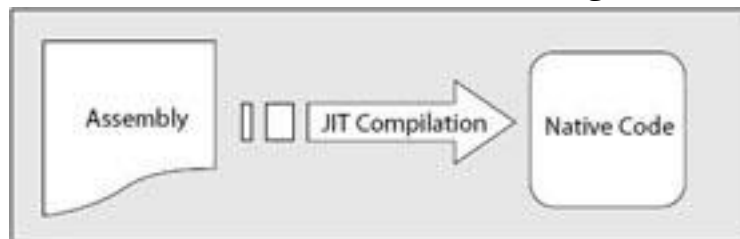
۱. نوشتن برنامه کاربردی با یکی از زبان‌های Net. برای مثال زبان C#



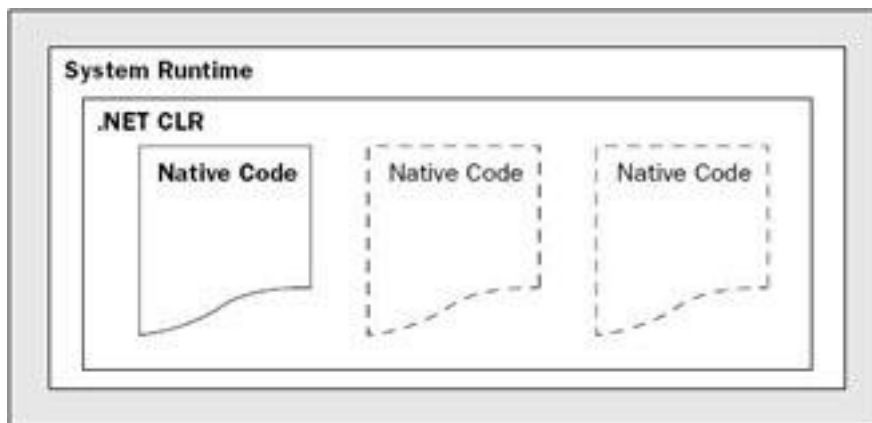
۲. تبدیل کدهای برنامه به MSIL و ذخیره کدهای MSIL داخل Assembly



۳. تبدیل کد MSIL به کد اجرایی با استفاده از JIT Compiler در زمان اجرا



۴. اجرای کد با استفاده از CLR



### برنامه‌نویسی شیء‌گرا

از زمان ایجاد کامپیوتر تاکنون برنامه‌نویسی تغییرات عمده‌ای کرده‌است. در صورتی که بخواهیم تاریخچه برنامه‌نویسی را مرور نماییم، مراحل زیر قابل ذکر می‌باشد.

۱. برنامه‌نویسی به زبان ماشین: در ابتدای ایجاد کامپیوتر برنامه‌ها از کدهای باینری ماشین تشکیل می‌شدند.
۲. برنامه‌نویسی به زبان اسمبلی: با پیچیده‌تر شدن برنامه‌ها زبان اسمبلی به وجود آمد. برنامه‌هایی که به زبان اسمبلی نوشته می‌شدند، به جای کدهای باینری از سمبل‌های معادل آنها تشکیل می‌شدند.

۳. برنامه‌نویسی رویه‌ای<sup>۱</sup>: پیچیده‌شدن برنامه‌های کاربردی ادامه داشت و کار به جایی رسید که زبان اسمبلی هم دیگر جوابگوی نیاز برنامه‌نویسان نبود، بنابراین زبان‌هایی رویه‌ای مانند Fortran و Cobol به وجود آمدند.
۴. برنامه‌نویسی ساخت‌یافته: پیچیده‌شدن و گسترش دامنه‌ی کاربرد برنامه‌ی کاربردی باز هم ادامه یافت. بنابراین نیاز به روش‌های جدیدتری احساس شد و زبان‌های ساخت‌یافته مانند C به وجود آمد.
۵. برنامه‌نویسی شیء‌گرا: امروزه برنامه‌های کاربردی آن قدر پیچیده شده‌اند، که برنامه‌نویسی ساخت‌یافته نیز جوابگوی آنها نیست و برای توسعه آن‌ها از زبان‌های شیء‌گرا نظیر جاوا و C# استفاده می‌شود.

### پشتیبانی از مفهوم شیء‌گرایی و واسط

- کلاسی که به یک زبان نوشته شده می‌تواند از کلاس نوشته شده به زبان دیگر مشتق شود.
- یک کلاس می‌تواند شامل نمونه‌ای از یک کلاس دیگر باشد، بدون توجه به اینکه زبان نوشتن هر کدام از این دو کلاس چه بوده است.
- یک کلاس می‌تواند به طور مستقیم توابع شیء دیگری را که به زبان دیگری نوشته شده است را فراخوانی کند.
- اشیاء یا اشاره‌گر به اشیاء می‌توانند بین توابع انتقال داده شوند.

### زباله‌روبی

NET. در زمان اجرا زباله‌روب<sup>۲</sup> حافظه را فعال می‌کند. زباله‌روب برنامه‌ای است که هدف آن پاک‌سازی حافظه است. تمام حافظه‌ای که به صورت پویا درخواست می‌شود در Heap نگهداری می‌شود.

زمانی که NET تشخیص می‌دهد که Heap مربوط به یک فرآیند در حال پر شدن است عملیات زباله‌روبی<sup>۳</sup> را انجام می‌دهد. جمع‌آوری زباله‌ها معمولاً در بین متغیرهایی که در حوزه کاری برنامه هستند، اجرا می‌شود. برای این کار به اشیاء ذخیره شده در Heap مراجعه می‌کند تا مشخص شود امکان دسترسی به کدامیک از آن‌ها وجود ندارد. هر شیئی که ارجاعی به آن وجود نداشته باشد، احتمالاً بلا استفاده بوده و حذف می‌شود.

یک نکته مهم در مورد زباله‌روبی این است که قطعی نیست. به عبارت دیگر نمی‌توانید ضمانت کنید که چه زمانی زباله‌روبی فراخوانی می‌شود و تنها زمانی که CLR تشخیص دهد فراخوانی می‌شود. البته امکان فراخوانی آن به طور صریح نیز وجود دارد. برای این کار باید این فرآیند را بازنویسی کرده و زباله‌روب را در کد برنامه فراخوانی کنید.

### فضای نامی

فضای نامی<sup>۴</sup> روشی برای سازمان‌دهی کلاس‌های مرتبط با یکدیگر و انواع دیگر است. برخلاف یک فایل یا یک مولفه فضای نامی ماهیت منطقی دارد نه فیزیکی. زمانی که یک کلاس را در یک فایل C# تعریف می‌کنید، می‌توانید آن را در تعریف یک فضای نامی قرار دهید. اگر کلاس دیگری تعریف می‌کنید که توابع و عملیات آن را در فایل دیگری قرار می‌دهید، می‌توانید آن را در همان فضای نامی کلاس قبلی تعریف کنید با این کار یک گروه بندی منطقی ایجاد خواهید کرد که به دیگر برنامه‌نویسان استفاده از کلاس‌ها و چگونگی ارتباط و استفاده از آن‌ها را نشان می‌دهد.

```
namespace CustomerPhoneApp
{
    using system;
    public struct Subscriber
    {
        //Code
    }
}
```

1- Procedural  
2- Garbage Collector  
3- Garbage Collection  
4- Namespace

نام کامل ساختار فوق CustomerPhoneApp.Subscriber است. این ویژگی باعث می‌شود تا کلاس‌هایی که اسامی کوتاه و مشابه دارند، ابهامی ایجاد نکنند. هم‌چنین می‌توانید با ایجاد ساختارهای سلسله‌مراتبی برای انواع داده‌ای خود فضای نامی تودرتو ایجاد کنید.

```
namespace Wrox
{
    namespace ProCSharp
    {
        namespace Basics
        {
            public Class NamespaceExample
            {
                //Code
            }
        }
    }
}
```

قطعه‌کد فوق را به صورت زیر هم می‌توان نوشت.

```
namespace Wrox.ProCSharp.Basics
{
    public Class NamespaceExample
    {
        //Code
    }
}
```

### دستور using

نام فضای نامی از نظر طول ممکن است، طولانی باشد و تایپ آن در کد برنامه خسته‌کننده شود. برای جلوگیری از تکرار نام فضای نامی می‌توان از دستور using استفاده کرد.

```
using Wrox.ProCSharp;
```

**نکته:** کاربرد دیگر کلمه‌ی کلیدی using این است که می‌توان به کلاس و فضای نامی نام مستعار نسبت داد. در صورتی که نام کلاس یا فضای نامی طولانی باشد و بخواهید چندین بار در کدتان به آن اشاره کنید، می‌توانید به آن نام مستعار نسبت دهید.

```
using alias = Namespace;
```

### قواعد نام‌گذاری

چگونگی انتخاب نام‌ها برای قابل‌فهم ساختن برنامه‌ها اهمیت دارد. نام‌هایی که انتخاب می‌کنید باید بازتاب هدف از تعریف آن‌ها باشد و به گونه‌ای طراحی شود که تداخلی با نام‌های دیگر نداشته باشد. برای نام‌گذاری در NET. از قاعده‌ی Pascal Caing استفاده می‌شود. در این قانون پیشنهاد می‌شود حروف اول هر کلمه با حرف بزرگ نوشته‌شود، مانند EmployeeSalary. توجه داشته باشید که برای تعریف ثوابت نیز از همین قاعده پیروی کنید.

همچنین باید سعی کنید که نام‌ها سبک ثابتی داشته باشند. به طور مثال اگر نام یکی از تابع‌ها ShowConfirmationDialog( ) است. تابع دیگر نباید ShowDialogWarning( ) یا WarningShowDialog باشد. نام این تابع باید ShowWarningDialog( ) باشد.



## تابع ( ) Main

نقطه‌ی آغاز برنامه‌های C# تابع ( ) Main است. این تابع static بوده و دارای مقدار داده بازگشتی void یا int است. تابع ( ) Main می‌تواند دارای آرگومان هم باشد. اگر چه وجود آرگومان اجباری نبوده و فقط زمانی ذکر می‌شود که بخواهیم مقادیری را از طریق خط فرمان به کلاس ارسال نماییم.

## ورودی/خروجی Console

توابعی که برای خواندن و نوشتن داده استفاده می‌شوند، عبارتند از:

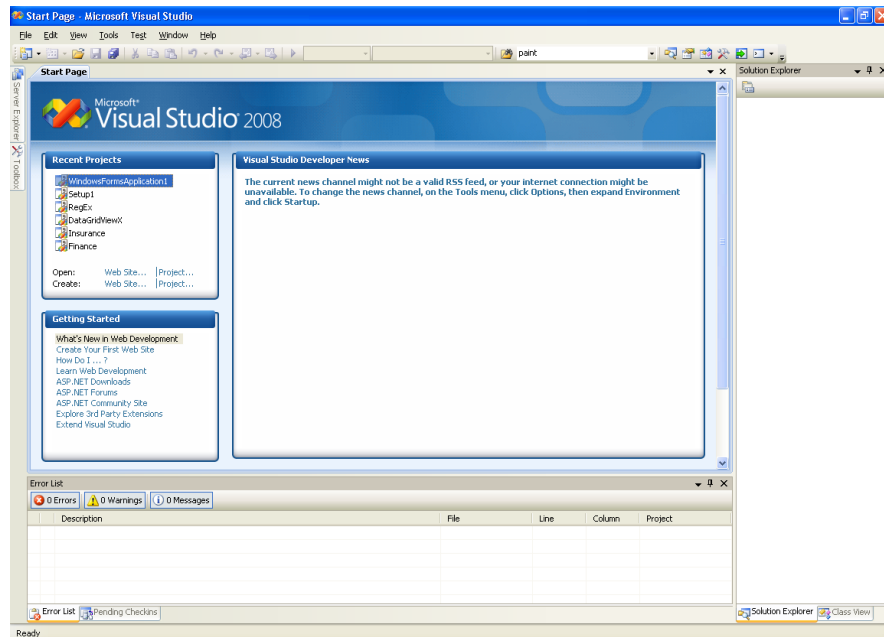
ReadLine() برای خواندن یک خط متن از خط فرمان. خط متن زمانی خاتمه می‌یابد که کلید Enter کلیک شود.

Write() مقدار خاصی را در خط فرمان می‌نویسد.

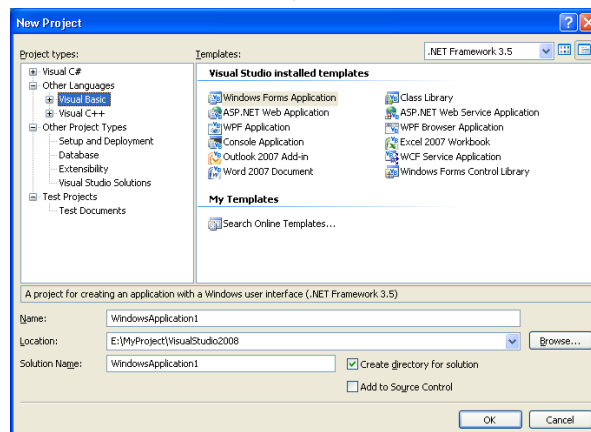
WriteLine() مشابه دستور Write() است. با این تفاوت که بعد از چاپ خروجی به ابتدای خط بعد می‌رود.

## آشنایی با محیط VS.NET 2008

پنجره‌ی اصلی VS.NET به صورت زیر است.

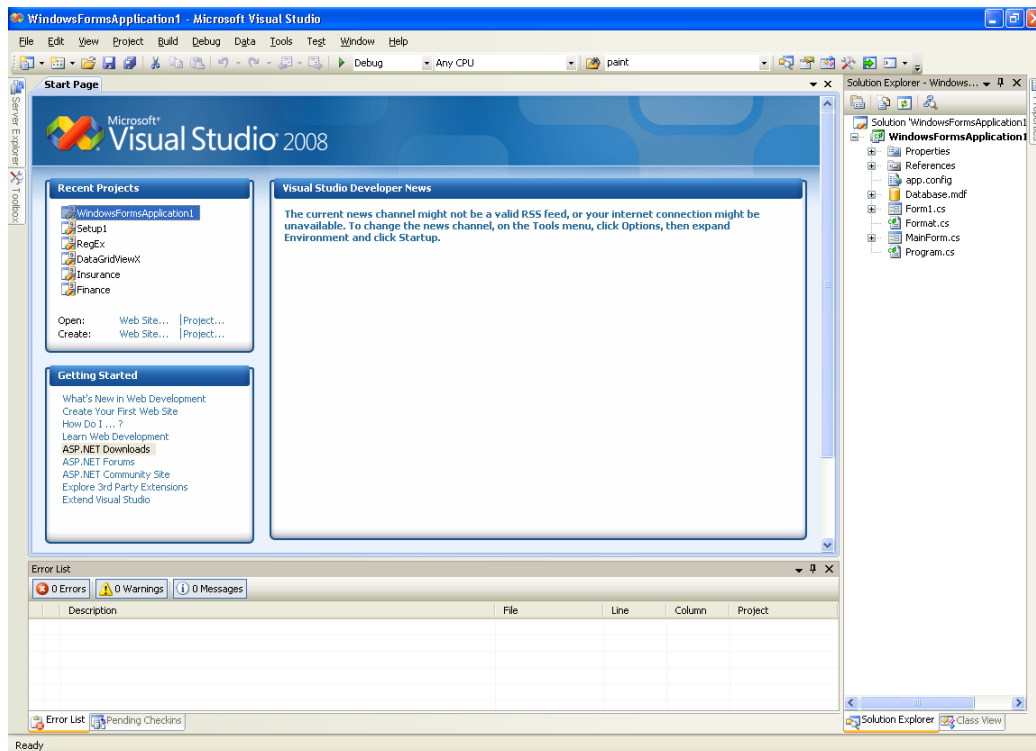


برای تعریف یک پروژه‌ی جدید از منوی File گزینه New و سپس گزینه Project را انتخاب کرده تا پنجره زیر ظاهر شود.



با توجه به نوع پروژه و زبان برنامه‌نویسی موردنظر گزینه‌ی مناسب را انتخاب می‌کنیم. به عنوان مثال اگر برنامه‌ی کاربردی ویندوز و زبان C# مدنظر باشد، ابتدا گزینه C# را در منوی چپ و گزینه Window Form Application را در منوی سمت راست انتخاب می‌نماییم.

فیلد Name نام پروژه و فیلد Location محل ذخیره فایل‌های پروژه است. پس از کلیک دکمه OK پنجره زیر نمایش داده خواهد شد.



همانطور که در این پنجره ملاحظه می‌کنید در قسمت سمت راست (Solution Explorer) لیست فایل‌های پروژه مشاهده می‌شود. برنامه‌ی کاربردی ویندوز شما شامل یک فرم به نام Form1 می‌باشد.

برنامه‌ی کاربردی شما علاوه بر Form1 شامل فایل‌هایی به شرح زیر است.

۱. AssemblyInfo.cs : این فایل شامل اطلاعات مربوط به Assembly نظیر نسخه‌ی آن است.

۲. Form.cs : این فایل شامل کد کلاس والد Form1 می‌باشد.

۳. References : پوشه References شامل تعدادی فایل است. این فایل‌ها ارجاع به فضاها می‌نمایند.

مورد استفاده در برنامه‌ی کاربردی را در خود جای داده‌اند. به عنوان مثال می‌توان از فایل‌های System،

System.Drawing، System.Data و System.Windows.Forms نام برد.

هنگامی که یک برنامه‌ی کاربردی ویندوز ایجاد می‌نمایید، یک فضای نامی پیش‌فرض که هم نام پروژه است، ایجاد می‌شود. در این مثال فضای نامی ایجاد شده SampleWindowsApplication نام دارد. هرگاه یک شیء جدید به فرم اضافه شود، قطعه کد مربوط به آن به کلاس Form1 اضافه می‌شود.

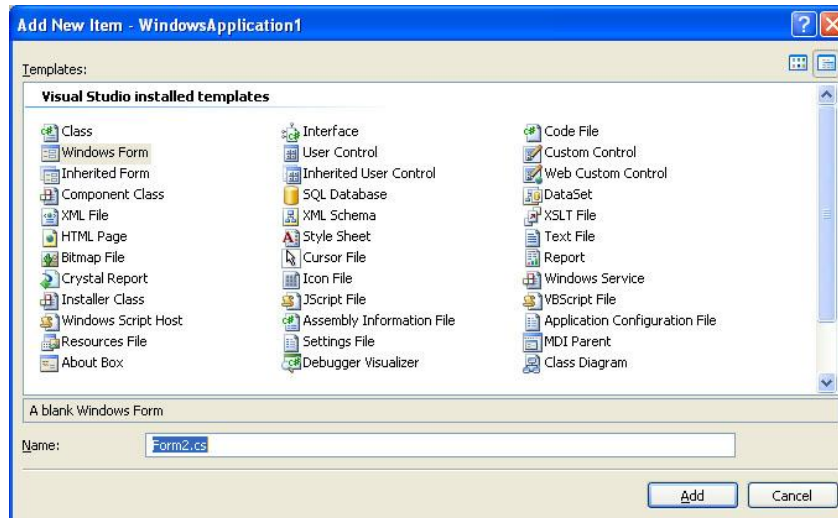
علاوه بر این کلاس فرم دارای متدی به نام Dispose() است. این متد فضای حافظه اختصاص داده شده به مولفه‌هایی را که توسط برنامه‌ی کاربردی استفاده نمی‌شوند، را آزاد می‌کند.

### افزودن فرم به برنامه‌ی کاربردی

هر برنامه‌ی کاربردی ویندوز حداقل دارای یک فرم است که هنگام ایجاد پروژه به وجود می‌آید. برای ایجاد فرم‌های بعدی مراحل زیر را انجام دهید.

۱. روی نام پروژه در پنجره Solution Explorer کلیک راست نمایید.

۲. در لیست نمایش داده شده گزینه Add و سپس گزینه Add Windows Form را انتخاب نمایید، تا پنجره زیر نمایش داده شود.



۳. نام فرم را در قسمت Name درج کرده و دکمه Add را کلیک نمایید.

### افزودن اشیاء به فرم

پس از ایجاد برنامه‌ی کاربردی نوبت افزودن اشیاء مورد نیاز به فرم است. برای این منظور از Button شروع می‌کنیم. یک Button به فرم اضافه کنید و ویژگی‌های آن را به صورت زیر تغییر دهید. مقدار فیلد Text را به welcome، مقدار فیلد Name را به sampleButton و مقدار Font را به Arial تغییر دهید. هنگامی که یک شیء جدید به فرم اضافه می‌کنید قطعه کد مربوط به آن به کلاس Form1 اضافه می‌شود. برای مثال در مورد Button فوق قطعه کد زیر اضافه می‌شود.

```
private System.Windows.Forms.Button sampleButton;
```

به منظور افزودن عملیات مورد نظر به دکمه ایجاد شده روی آن Double Click نمایید تا متدی به صورت زیر نمایش داده شود.

```
private void sampleButton_Click(object sender, System.EventArgs e) {  
  
}
```

عملیات مورد نظر Button را داخل این متد قرار می‌دهیم. برای مثال عبارت زیر را داخل بدنه متد درج کنید.  
 MessageBox.Show("This is a sample Windows Application");

### انواع کنترل‌های فرم

کنترل‌های فرم برای تعامل با کاربر مورد استفاده قرار می‌گیرند. لازم به ذکر است که امکان ایجاد کنترل‌های سفارشی کاربر علاوه بر کنترل‌های موجود در محیط Visual Studio.net وجود دارد.

#### Button

این کنترل امکان انجام عملیات مورد نظر کاربر را از طریق کلیک فراهم می‌کند. برای این منظور باید عملیات مورد نظر کاربر را در رخداد Click آن قرار دهید. برای ایجاد یک Button مراحل زیر را انجام می‌دهیم.

۱. افزودن Button به فرم

۲. تغییر مقدار خصایص Name و Text. در این مثال مقدار خصیصه ی Name را برابر btnShow و مقدار خصیصه Text را برابر Show قرار می‌دهیم.

۳. Double Click روی Button و افزودن قطعه کد زیر به رخداد کلیک آن.

```
Form2 newForm = new Form2();
newForm.Show();
this.Hide();
```

### Label

از این کنترل به منظور افزودن متن ثابت یا تصویر به فرم استفاده می‌شود. برای ایجاد یک Label مراحل زیر را انجام می‌دهیم.

۱. افزودن Label به فرم
۲. تغییر مقدار خصایص آن. تنها خصیصه که تغییر مقدار آن در مورد Label ضروری است خصیصه Text می‌باشد. در این مثال مقدار خصیصه‌ی Name را برابر lblDescription و مقدار خصیصه‌ی Text را برابر Click on the show button to display Form2 قرار می‌دهیم.

### TextBox

این کنترل امکان دریافت ورودی کاربر را فراهم می‌کند. علاوه بر این امکان نمایش داده‌های پویا و قابل تغییر توسط TextBox وجود دارد. به عبارت دیگر امکان تغییر مقدار خصیصه‌ی Text آن در زمان اجرا وجود دارد.

### MenuStrip

- از این کنترل به منظور افزودن منو به فرم استفاده می‌شود. برای ایجاد یک MenuStrip مراحل زیر را انجام می‌دهیم.
۱. افزودن MenuStrip به فرم
  ۲. کلیک روی عبارت Type Here و درج مقدار &File.
  ۳. درج عبارت &New در TextArea زیر گزینه فایل. برای افزودن گزینه‌های Open، Save و Save as نیز به همین صورت عمل می‌کنیم.
  ۴. برای افزودن Edit و گزینه‌های Cut، Copy و Paste نیز به مانند File و زیر مجموعه‌هایش عمل می‌کنیم.
  ۵. پس از انجام مراحل فوق نوبت افزودن عملیات موردنظر به گزینه‌های منو می‌باشد. برای این منظور ابتدا روی گزینه New کلیک دابل می‌کنیم تا کد مربوط به آن نمایش داده شود. رخداد مربوط به New به صورت زیر است.
- ```
private void newToolStripMenuItem_Click(object sender,
    System.EventArgs e) {
}
```

عبارت زیر را در بدنه متد درج کنید.

```
Form2 newForm = new Form2();
newForm.Show();
this.Hide();
```

### GroupBox

از این کنترل به منظور گروه‌بندی تعدادی از کنترل‌های فرم نظیر TextBox، RadioButton، CheckBox و .. استفاده می‌شود.

### RadioButton

از این کنترل به منظور ایجاد امکان انتخاب یک گزینه از بین گروهی از گزینه‌ها استفاده می‌شود. برای گروه‌بندی تعدادی از RadioButton‌ها از GroupBox استفاده می‌شود. در مثال قبل برای مشاهده Form2 از یک کنترل button استفاده کردیم. در صورتی که تعداد فرم‌های موردنظر زیاد باشد می‌توان به جای دکمه از RadioButton استفاده کرد. برای این منظور به صورت زیر عمل می‌کنیم.

۱. افزودن GroupBox به فرم

۲. مقدار خصیصه `GroupBox Text` را برابر `Forms` قرار دهید.
۳. سه عدد `RadioButton` داخل `GroupBox` اضافه کنید.
۴. مقدار خصایص `Name` و `Text` دکمه‌های رادیویی را به ترتیب برابر `btnForm1` و `Form1`، `btnForm2` و `Form2` و `btnForm3` و `Form3` قرار دهید.
۵. پس از انجام مراحل فوق نوبت افزودن عملیات موردنظر به دکمه‌های رادیویی می‌باشد. برای این منظور ابتدا روی دکمه‌های رادیویی `Double Click` می‌کنیم تا کد مربوط به آن نمایش داده شود.

```
private void btnForm1_CheckedChanged(object sender,
    System.EventArgs e) {
}
}
```

عبارت زیر را در بدنه متد درج نمایید.

```
Form1 newForm = new Form1();
newForm.Show();
this.Hide();
```

۶. برای دکمه‌های رادیویی `btnForm2` و `btnForm3` نیز به همین صورت عمل می‌کنیم.

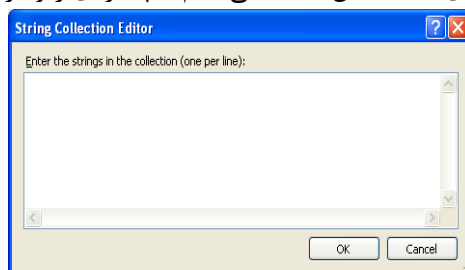
### CheckBox

این کنترل امکان انتخاب وضعیت را برای کاربر فراهم می‌کند. وضعیت یکی از مقادیر `true` یا `false` می‌باشد. برای تعیین اینکه آیا یک `CheckBox` انتخاب شده‌است یا نه از خصیصه `Checked` آن استفاده می‌شود. مقدار بازگشتی این خصیصه یکی از مقادیر `true` یا `false` می‌باشد. برای افزودن عملیات به این کنترل نیز مانند دیگر کنترل‌های باید روی آن `Double Click` کرده و عملیات موردنظر را در رخداد مربوطه درج کنید. برای این منظور می‌توانید از خصیصه `CheckBox` کنترل `CheckBox` جهت تشخیص انتخاب شدن آن استفاده نمایید. مقدار بازگشتی خصیصه `CheckBox` یکی از مقادیر `Checked` یا `Unchecked` می‌باشد.

### ListBox

این کنترل امکان انتخاب چند گزینه را در یک لیست برای کاربر فراهم می‌کند. تک/چند انتخابی بودن این کنترل با توجه به مقدار خصیصه `SelectionMode` انجام می‌شود. در صورتی که مقدار خصیصه `SelectionMode` برابر `one` باشد، کاربر فقط امکان انتخاب یک گزینه را خواهد داشت و در صورتی که مقدار خصیصه `SelectionMode` برابر `MultiSimple` یا `MultiExtended` باشد، کاربر فقط امکان انتخاب چند گزینه را خواهد داشت. برای ایجاد `ListBox` که فقط امکان انتخاب یک گزینه در آن وجود دارد، مراحل زیر را انجام می‌دهیم.

۱. افزودن `ListBox` به فرم
۲. تغییر مقدار خصایص `Name` و `SelectionMode`. در این مثال مقدار خصیصه `Name` را برابر `listBox1` و مقدار خصیصه `SelectionMode` را برابر `one` قرار می‌دهیم.
۳. تا این مرحله `ListBox` خالی است. برای افزودن تعدادی گزینه به `ListBox` در پنجره `Properties` خصیصه `Items` را انتخاب کرده و روی نماد ... آن کلیک می‌کنیم تا پنجره زیر باز شود.



۴. هر یک از گزینه‌های ListBox را در پنجره‌ی String Collection Editor در یک سطر درج نمایید.
۵. دکمه OK را کلیک نمایید تا پنجره‌ی String Collection Editor بسته شود.

### ComboBox

این کنترل امکان انتخاب یک گزینه را از بین تعدادی گزینه برای کاربر فراهم می‌کند. برای تعیین گزینه‌ای که کاربر انتخاب کرده‌است، از خاصیتی SelectedIndex استفاده می‌شود. مقدار بازگشتی خاصیتی SelectedIndex اندیس کنترل انتخاب شده‌است. در صورتی که هیچ کنترلی انتخاب نشده‌باشد، مقدار بازگشتی این کنترل 1- خواهد بود.

**نکته:** هر دو کنترل ComboBox و ListBox امکان انتخاب یک گزینه را برای کاربر فراهم می‌آورد. تفاوت بین ComboBox و ListBox در این است که در صورت استفاده از ComboBox کاربر علاوه بر امکان انتخاب یک گزینه امکان درج رشته‌های موردنظر خود را نیز دارد، در حالی که در صورت استفاده از ListBox امکان درج وجود ندارد و کاربر الزاماً باید یکی از گزینه‌ها را انتخاب نماید.

### MonthCalendar

این کنترل امکان انتخاب تاریخ مورد نظر را برای کاربر فراهم می‌کند. به صورت پیش‌فرض تاریخ انتخاب‌شده تاریخ جاری می‌باشد. برای تغییر ماه از علامت پیکان در قسمت بالای کنترل MonthCalendar استفاده می‌شود. علاوه بر این امکان انتخاب چند تاریخ یا یک بازه‌ی تاریخ نیز در این کنترل وجود دارد.

### DateTimePicker

این کنترل امکان انتخاب یک تاریخ را برای کاربر فراهم می‌کند. به صورت پیش‌فرض تاریخ انتخاب‌شده تاریخ جاری می‌باشد.

### متغیرها

برای تعریف متغیرها به صورت زیر عمل می‌کنیم.

<Data Type> <Variable1, Variable2, ...>;

مثال:

int x = 10;

### محدوده‌ی متغیرها

۱. Block: فقط داخل بلوکی که تعریف شده‌است، قابلیت دسترسی به آن وجود دارد.
  ۲. Procedure: فقط داخل روبه‌ای که تعریف شده‌است، قابلیت دسترسی به آن وجود دارد.
  ۳. Namespace: فقط داخل فضای‌نامی که تعریف شده‌است، قابلیت دسترسی به آن وجود دارد.
- نکته: امکان تعریف دو متغیر با یک نام در یک بازه وجود ندارد.

### انواع متغیر

۱. متغیر نمونه<sup>۱</sup>: این نوع متغیرها هنگام ایجاد یک شی از کلاس مقدار می‌گیرند، اشیاء متفاوتی که از روی یک کلاس ساخته می‌شوند، می‌توانند دارای مقادیر متفاوتی برای این نوع متغیر باشند.
۲. متغیر کلاس<sup>۲</sup>: متغیرهای کلاس در هنگام تعریف کلاس مقدار می‌گیرند، و مقدارشان برای تمام اشیاء کلاس یکسان است.

1- Instance Variable

2- Class Variable

۳. متغیر محلی<sup>۱</sup>: متغیرهایی که داخل بدنه یک متد تعریف می‌شوند از این دسته هستند.

### انواع مقدار و انواع مرجع

قبل از بررسی انواع داده‌ای بهتر است بدانید C# بین دو دسته از انواع داده‌ای تفاوت قائل می‌شود.

■ انواع مقدار<sup>۲</sup>

■ انواع مرجع<sup>۳</sup>

تفاوت این دو نوع داده‌ای در نحوه ذخیره‌سازی آن‌ها است. در واقع انواع داده‌ای مقدار، مقدار را ذخیره می‌کند در حالی که انواع داده‌ای ارجاع اشاره‌گری به مقدار را ذخیره می‌کنند. علاوه بر این انواع مقدار در پشته و انواع مرجع در heap مدیریت شده ذخیره می‌شوند. انواع داده‌ای پایه نظیر float، int و ... و همچنین structها از نوع مقدار و کلاس‌ها از نوع ارجاع هستند.

### انواع داده‌ای

| نوع داده‌ای | حجم حافظه | نوع CTS | مقادیر مجاز                                      |
|-------------|-----------|---------|--------------------------------------------------|
| bool        | 8         | Boolean | false و True                                     |
| char        | 16        | Char    | '\u0000'...'\uFFFF'                              |
| byte        | 8         | Byte    | 0...255                                          |
| sbyte       | 8         | SByte   | -128...127                                       |
| short       | 16        | Int16   | -32768...32767                                   |
| ushort      | 16        | UInt16  | 0...65535                                        |
| int         | 32        | Int32   | -2147483648...2147483647                         |
| uint        | 32        | UInt32  | 0...4294967295                                   |
| long        | 64        | Int64   | -9223372036854775808...9223372036854775807       |
| ulong       | 64        | UInt64  | 0...18446744073709551615                         |
| decimal     | 128       | Decimal | $1.0 \times 10^{-28}$ ... $7.9 \times 10^{28}$   |
| float       | 32        |         | $1.5 \times 10^{-45}$ ... $3.4 \times 10^{38}$   |
| double      | 64        |         | $5.0 \times 10^{-324}$ ... $1.7 \times 10^{308}$ |
| object      |           | Object  | نوع داده‌ای ریشه تمام انواع داده‌ای CTS.         |
| string      |           | String  | رشته‌های کاراکتری Unicode.                       |

### انواع تبدیل

۱. تبدیل صریح<sup>۴</sup>:

```
int x;
long y = 100;
x = (int) y;
```

۲. تبدیل ضمنی<sup>۵</sup>:

```
int x = 100;
long y;
y = x;
```

نکته: تبدیلاتی که به صورت ضمنی انجام می‌شود، به شرح زیر است.

| نوع داده‌ای اولیه | انواع داده‌ای مجاز جهت تبدیل ضمنی |
|-------------------|-----------------------------------|
|-------------------|-----------------------------------|

- 1- Local Variable
- 2- Value Types
- 3- Reference Types
- 4- Explicit
- 5- Implicit

|                                                               |        |
|---------------------------------------------------------------|--------|
| decimal, long, double, float                                  | int    |
| decimal, double, float                                        | long   |
| int, decimal, long, double, float                             | short  |
| short, int, decimal, long, double, float                      | sbyte  |
| int, uint, long, ulong, short, ushort, decimal, double, float | byte   |
| int, uint, long, ulong, decimal, double, float                | ushort |
| long, ulong, decimal, double, float                           | uint   |
| decimal, double, float                                        | ulong  |
| double                                                        | float  |
| int, uint, long, ulong, ushort, decimal, double, float        | char   |

نکته: تبدیلاتی که باید به صورت صریح انجام می‌شود، به شرح زیر است.

| نوع داده‌ای اولیه | انواع داده‌ای مجاز جهت تبدیل صریح                                       |
|-------------------|-------------------------------------------------------------------------|
| int               | uint, byte, sbyte, short, ushort, char, ulong                           |
| long              | int, uint, byte, sbyte, short, ushort, char, ulong                      |
| short             | uint, byte, sbyte, ushort, char, ulong                                  |
| sbyte             | uint, byte, ushort, char, ulong                                         |
| byte              | sbyte, char                                                             |
| ushort            | byte, sbyte, short, char                                                |
| uint              | int, byte, sbyte, short, ushort, char                                   |
| ulong             | int, uint, byte, sbyte, short, ushort, char, long                       |
| float             | int, uint, long, ulong, byte, sbyte, short, ushort, char, decimal       |
| double            | int, uint, long, ulong, byte, sbyte, short, ushort, char, float         |
| decimal           | int, uint, long, ulong, byte, sbyte, short, ushort, char, float, double |
| char              | byte, sbyte, short                                                      |

### ثوابت

برای تعریف ثوابت به صورت زیر عمل می‌کنیم.

```
const <Data Type> <Variable> = value;
```

مثال:

```
const int x = 10;
```

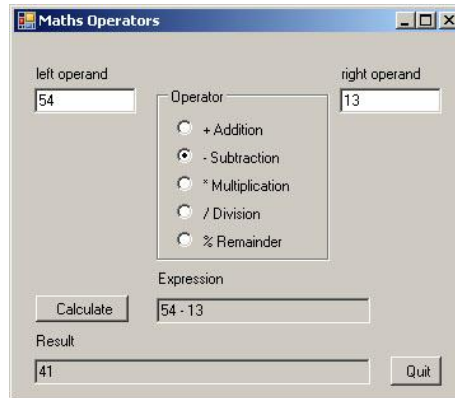
### عملگر

| عملگر | عملکرد           |
|-------|------------------|
| +     | جمع              |
| -     | تفریق            |
| *     | ضرب              |
| /     | تقسیم            |
| %     | باقیمانده        |
| --    | یک واحد کاهش     |
| ++    | یک واحد افزایش   |
| ==    | کنترل برابری     |
| !=    | کنترل عدم برابری |
| >     | بزرگتر           |
| <     | کوچکتر           |
| >=    | بزرگتر مساوی     |



|              |        |
|--------------|--------|
| کوچکتر مساوی | <=     |
| AND منطقی    | & و && |
| OR منطقی     | و      |
| XOR          | ^      |
| NOT منطقی    | !      |

تمرین: برنامه‌ای بنویسید که دارای صفحه‌ای به صورت زیر باشد و چهار عمل اصلی را انجام دهد.



### دستور if

ساختار دستور if به یکی از صورت‌های زیر است.

1. if (Boolean-expression) statement1
2. if (Boolean-expression) statement1  
else statement2
3. if (Boolean-expression) statement1  
else if (Boolean-expression) statement2

مثال:

```
int x;
if (x >= 0) {
    System.Console.WriteLine("x is a positive number.");
}
else {
    System.Console.WriteLine("x is a negative number.");
}
```

در این مثال در صورتی که x بزرگتر یا مساوی باشد، پیغام x is a positive number و در غیر این صورت پیغام x is a negative number نمایش داده می‌شود.

در صورت لزوم می‌توانید دستور if را به صورت تودرتو<sup>۱</sup> استفاده نمایید.

### دستور switch

ساختار دستور switch به صورت زیر است.

```
switch (expression) {
    case constant-expression:
        statement;
        break;
    [default:
        Statement;
        Break;]
}
```

لازم به ذکر است که وجود قسمت default اختیاری می‌باشد. قسمت default هنگامی اجرا می‌شود که هیچ کدام از case ها اجرا نشود.

مثال:

```
int x;
switch (x) {
case 1:
    System.Console.WriteLine("x is a positive number.");
    break;
case 2:
    System.Console.WriteLine("x is a negative number.");
    break;
default :
    System.Console.WriteLine("x is equal to 0.");
    break;
}
```

**حلقه for**

ساختار حلقه for به صورت زیر است.

```
for (initializer; condition; iterator) {
    Statement;
}
```

مثال:

```
for (int i = 0; i < 100; i++) {
    System.Console.WriteLine("{0} ", i);
    if (i % 10 == 0) {
        System.Console.WriteLine("\t{0} ", i);
    }
}
```

**حلقه while**

ساختار حلقه while به صورت زیر است.

```
while (condition) {
    Statement;
}
```

مثال:

```
int x = 2;
while (x < 10) {
    Console.WriteLine (x);
    x ++;
}
```

**حلقه do while**

ساختار حلقه do while به صورت زیر است.

```
do {
    Statement;
} while (condition);
```

مثال:

```
int x = 2;
do {
    Console.WriteLine (x);
    x ++;
} while (x < 10);
```

**نکته:** تفاوت دستور `while` و `do while` در این است که اگر شرط دستور `do while` برقرار نباشد، بدنه‌ی حلقه یک بار اجرا می‌شود درحالی‌که در این صورت بدنه‌ی حلقه `while` اجرا نخواهد شد.

### دستور `break`

جهت خاتمه‌ی حلقه یا اجرای دستور `switch` استفاده می‌شود. به مثال مربوط به دستور `switch` توجه نمایید.

### دستور `continue`

جهت بازگشتن به ابتدای حلقه استفاده می‌شود.

مثال:

```
for (int i = 0; i < 100; i++) {
    if (i % 10 == 0) {
        continue;
    }
    System.Console.WriteLine("{0} ", i);
}
```

در این مثال اگر باقیمانده تقسیم `i` بر ۱۰ برابر صفر باشد، هیچ خروجی چاپ نخواهد شد، شمارنده‌ی حلقه افزایش یافته و بدنه‌ی حلقه مجدداً اجرا می‌شود.

## کلاس‌ها و ساختارها

کلاس و ساختار<sup>۱</sup> قالب‌هایی برای ایجاد اشیاء هستند. هر شیء شامل داده‌ها و متدهایی برای تغییر و دسترسی به آن‌ها است. کلاس تعیین می‌کند که هر شیء خاص از کلاس چه داده‌ها و توابعی در اختیار دارد. فرق ساختار با کلاس در نحوه‌ی ذخیره‌سازی آن‌ها در حافظه و طریقه دسترسی به آن‌ها است. کلاس‌ها انواع ارجاعی هستند و در `Heap` نگهداری می‌شوند، در حالی که ساختارها در پشته ذخیره می‌گردند. علاوه بر این ساختارها ارث‌بری ندارند. گاهی ممکن است به دلایل کارایی برای ایجاد انواع داده‌ای کوچک از ساختار استفاده کنید.

## کلاس

مجموعه‌ای از اشیاء که دارای ویژگی‌ها و عملیات مشترکی می‌باشند، را یک کلاس می‌نامیم. به عبارت دیگر یک کلاس مفهومی ذهنی است و شیء نمونه مجسم آن. به عنوان مثال هر کدام از شما نمونه‌ای از کلاس دانشجوی هستید. نمونه‌های یک کلاس را شیء (object) یا instance می‌نامیم. در زبان برنامه‌نویسی C# برای تعریف نوع داده‌ای جدید از یک کلاس جدید تعریف می‌کنیم. برای تعریف کلاس به صورت زیر عمل می‌کنیم.

```
<modifiers> class <class name> {
    <Modifier> <Data Type> <Property1, Property2, ...>;
    ...
    <Return Type>| void <Method Name> {
        ...
    }
    ...
}
```

همان‌طور که ملاحظه می‌کنید یک کلاس از دو بخش تشکیل شده‌است. بخش اول فیلد (Field)‌ها و خصایص (Property‌های) کلاس و بخش دوم متدهای کلاس می‌باشد. متدهای یک کلاس ویژگیهای رفتاری کلاس را تعریف می‌نمایند. مثال:

```
class Building {
    public int floors; // number of floors
```

```
public int area; // total square footage of building
public int occupants; // number of occupants
}
```

**نکته:** برای ایجاد نمونه از کلاس و ساختار از کلمه‌ی کلیدی new استفاده می‌شود.

```
Building building = new Building();
```

### اعضای کلاس

داده‌ها و توابع کلاس را اعضای کلاس می‌نامیم.

### اعضای داده

اعضای داده اعضایی هستند که شامل داده‌هایی برای کلاس هستند این اعضاء عبارتند از: فیلدها، ثوابت و رخدادها. به هر متغیری که در رابطه با کلاس تعریف شود، فیلد گویند. فیلدهای کلاس Building را در مثال فوق ملاحظه کردید. پس از تعریف شیء به صورت زیر به فیلدهای آن دسترسی پیدا می‌کنیم.

```
building.floors = 4;
```

**نکته:** رخدادها اعضایی از کلاس هستند که به اشیاء اجازه می‌دهند تا آن‌ها را در زمان وقوع بعضی وقایع فراخوانی کنند. به عنوان مثال می‌توان رخداد کلیک را در نظر گرفت.

Modifierهای مجاز هنگام تعریف فیلدها به شرح زیر می‌باشد.

۱. internal: فقط داخل Assembly قابل استفاده است.
۲. private: فقط داخل کلاس قابل استفاده است.
۳. protected: فقط داخل کلاس و زیرکلاس‌های آن قابل استفاده است.
۴. protected internal: فقط داخل کلاس، زیرکلاس‌های آن و کلاس‌های داخل Assembly قابل استفاده است.
۵. public: همه‌جا قابل استفاده است.
۶. readonly: متغیر به صورت فقط خواندنی تعریف می‌شود. در این صورت مقداردهی متغیر هنگام تعریف متغیر یا در متد سازنده انجام‌پذیر است.

### اعضای تابع

اعضای تابع<sup>۱</sup> اعضایی هستند که توابعی برای دست‌کاری داده‌ها در کلاس ایجاد می‌کنند. انواع توابع عبارتند از: توابع خصوصیات، سازنده‌ها، مخرب‌ها و غیره. در واقع یک متد ساختار منطقی است که جهت انجام عملیات خاصی مورد استفاده قرار می‌گیرد. امکان فراخوانی متدها توسط اشیاء برای انجام عملیات متد وجود دارد. برای تعریف یک متد به صورت زیر عمل می‌کنیم.

```
<Modifier> <Data Type> <Method Name> (param1, param2, ...) {
    Statement(s);
}
```

فراخوانی یک متد به صورت زیر انجام می‌شود.

```
object1.method1(parameter1, parameter2,...);
```

مثال:

```
using System;
namespace Wrox.ProCSharp.MathTestSample {
    class MainEntryPoint {
        static void Main() {
            // Try calling some static functions
        }
    }
}
```

۱ - اعضای تابع را متد نیز می‌نامند

```

    Console.WriteLine("Pi is " + MathTest.GetPi());
    int x = MathTest.GetSquareOf(5);
    Console.WriteLine("Square of 5 is " + x);
    // Instantiate at MathTest object
    MathTest math = new MathTest(); // this is C#'s way of
    // instantiating a reference type call non-static methods

    math.value = 30;
    Console.WriteLine("Value field of math variable contains " + math.value);
    Console.WriteLine("Square of 30 is " + math.GetSquare());
}
}
// Define a class named MathTest on which we will call a method
class MathTest {
    public int value;
    public int GetSquare() {
        return value*value;
    }
    public static int GetSquareOf(int x) {
        return x*x;
    }
    public static double GetPi() {
        return 3.14159;
    }
}
}

```

### فیلدهای کلاس، خصایص کلاس و متدهای accessor

به ازای هر یک از خصایص کلاس می‌توان یک متد get برای خواندن مقدار و یک متد set برای مقداردهی آن وجود داشته باشد. برای تعریف متدهای get و set به صورت زیر عمل می‌کنیم.

```

<Modifier> <Data Type> <Field Name>;
<Modifier> <Data Type> <Property Name> {
    get {
        return Field Name;
    }
    set {
        Field Name = value;
    }
}

```

برای درک بهتر مطلب به مثال زیر توجه کنید.

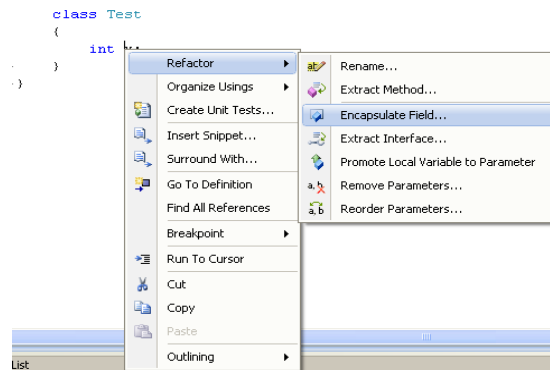
```

private int x;
public int X {
    get {
        return x;
    }
    set {
        x = value;
    }
}
static void Main() {
    Tester t = new Tester();
    t.x = 10;
    System.Console.WriteLine("X is : {0}\n",t.x);
    System.Console.ReadKey();
}

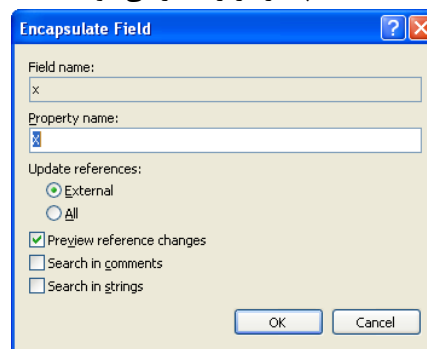
```

**نکته:** با توجه به نوع متدهای accessor یک خصیصه ممکن است سه حالت متفاوت داشته باشد. حالات یک خصیصه عبارتند از:

۱. فقط خواندنی: در صورتی که خصیصه فقط متد get داشته‌باشد.
  ۲. فقط نوشتنی: در صورتی که خصیصه فقط متد set داشته‌باشد.
  ۳. خواندنی/نوشتنی: در صورتی که خصیصه هم متد get و هم متد set داشته‌باشد.
- نکته: برای ایجاد متدهای accessor در VS.NET مراحل زیر را انجام می‌دهیم.
۱. اشاره‌گر ماوس را روی خصیصه مورد نظر قرار داده، کلیک راست نموده، گزینه Refactor و سپس گزینه Encapsulate Field را انتخاب می‌نماییم.



۲. پس از کلیک گزینه Encapsulate Field پنجره زیر ظاهر می‌شود.



۳. دکمه OK را کلیک می‌نماییم.

نکته: در صورتی که یک خصیصه به صورت readonly تعریف شده‌باشد، پس از انجام مراحل فوق فقط متد get برای خواندن آن ایجاد خواهد شد. به عبارت دیگر امکان اختصاص مقدار به متغیرهای readonly خارج از کلاس وجود نخواهد داشت.

نکته: در صورت لزوم می‌توانید یکی از متدهای accessor را به صورت private، protected یا internal تعریف کرده و به این ترتیب سطح دسترسی آن کنترل کنیم. برای این منظور به صورت زیر عمل می‌کنیم.

```
private int x;
public int X {
    get {
        return x;
    }
    private set {
        x = value;
    }
}
```

انواع ارسال پارامتر به متد

۱. پارامتر مقدار<sup>۱</sup>: تغییراتی که متد روی پارامتر ارسالی اعمال می‌کند تاثیری روی مقدار متغیر اصلی نخواهد داشت. به عبارت دیگر پارامتر ارسالی کپی متغیر اصلی می‌باشد و تغییرات آن در سطح متد می‌باشد. برای درک بهتر مطلب به مثال زیر توجه نمایید.

```
static void ShowDouble(int val) {
    val *= 2;
    Console.WriteLine("val doubled = {0}", val);
}
```

در صورتی که قطعه کدی به صورت زیر در بدنه متد main() وجود داشته باشد.

```
int myNumber = 5;
Console.WriteLine("myNumber = {0}", myNumber);
ShowDouble(myNumber);
Console.WriteLine("myNumber = {0}", myNumber);
```

خروجی زیر نمایش داده خواهد شد.

```
myNumber = 5
val doubled = 10
myNumber = 5
```

همان‌طور که ملاحظه می‌کنید مقدار متغیر قبل و بعد از فراخوانی متد یکسان است.

۲. پارامتر ارجاع<sup>۲</sup>: تغییراتی که متد روی پارامتر ارسالی اعمال می‌کند، روی مقدار متغیر اصلی اعمال خواهد شد. به عبارت دیگر پارامتر ارسالی اشاره‌گری به متغیر اصلی می‌باشد. برای ارسال پارامتر به روش فراخوانی با ارجاع به صورت زیر عمل می‌کنیم. در انواع داده‌های پیچیده به خاطر وجود مقدار داده‌های بزرگی که هنگام انتقال با مقدار باید کپی‌برداری شود، انتقال با ارجاع موثرتر است.

```
object1.method1(parameter1, ref parameter2,...);
```

همان‌طور که ملاحظه می‌کنید، در این عبارت از کلمه کلیدی ref به منظور ارسال پارامتر با استفاده از روش فراخوانی با ارجاع استفاده شده است. در این مثال parameter1 به روش فراخوانی با مقدار و parameter2 به روش فراخوانی با ارجاع ارسال شده است. برای درک بهتر مطلب به مثال زیر توجه نمایید.

```
static void ShowDouble(ref int val) {
    val *= 2;
    Console.WriteLine("val doubled = {0}", val);
}
```

در صورتی که قطعه کدی به صورت زیر در بدنه متد main() وجود داشته باشد.

```
int myNumber = 5;
Console.WriteLine("myNumber = {0}", myNumber);
ShowDouble(ref myNumber);
Console.WriteLine("myNumber = {0}", myNumber);
```

خروجی زیر نمایش داده خواهد شد.

```
myNumber = 5
val doubled = 10
myNumber = 10
```

همان‌طور که ملاحظه می‌کنید مقدار متغیر قبل و بعد از فراخوانی متد یکسان نمی‌باشد.

۳. پارامتر خروجی<sup>۳</sup>: یک متد در حالت معمول فقط یک مقدار بازگشتی دارد. در صورتی که لازم باشد یک متد چند مقدار بازگشتی داشته باشد، باید از پارامترهای خروجی به عنوان پارامترهای متد استفاده نماییم. لازم به ذکر است که اگر پارامتر ارسالی به متد از نوع پارامتر خروجی می‌باشد، نیازی به مقداردهی آن قبل از ارسال وجود نداشته و پارامتر مذکور مقدار بازگشتی متد را دریافت خواهد کرد. به منظور ارسال پارامتر خروجی به یک متد به صورت زیر عمل می‌کنیم.

```
object1.method1(out parameter1);
```

1- Value Parameter  
2- Reference Parameter  
3- Output Parameter

```
object1.method1(params data type[] parameter1, ref parameter2);
```

برای درک بهتر مطلب به مثال زیر توجه نمایید.

```
static int MaxValue(int[] intArray, out int maxIndex)
{
    int maxVal = intArray[0];
    maxIndex = 0;
    for (int i = 1; i < intArray.Length; i++)
    {
        if (intArray[i] > maxVal)
        {
            maxVal = intArray[i];
            maxIndex = i;
        }
    }
    return maxVal;
}
```

در صورتی که قطعه کدی به صورت زیر در بدنه متد main() وجود داشته باشد.

```
int[] myArray = { 1, 8, 3, 6, 2, 5, 9, 3, 0, 2 };
int maxIndex;
Console.WriteLine("The maximum value in myArray is {0}",
    MaxValue(myArray, out maxIndex));
Console.WriteLine("The first occurrence of this value is at element
    {0}", maxIndex + 1);
```

خروجی زیر نمایش داده خواهد شد.

```
The maximum value in myArray is 9
The first occurrence of this value is at element 7
```

همان‌طور که ملاحظه می‌کنید مقدار متغیر قبل و بعد از فراخوانی متد یکسان است

### ایجاد متدهای همنام

در برنامه‌نویسی ساخت‌یافته در صورتی که دو متد همنام وجود داشته باشد، برنامه کامپایل نشده و پیغام Duplicated Name مشاهده می‌شود. در برنامه‌نویسی شیء‌گرا امکان ایجاد متدهای همنام<sup>۱</sup> در یک کلاس وجود دارد. تنها کافی است تعداد آرگومان‌ها یا نوع آن‌ها متفاوت باشد. برای درک بهتر مطلب به مثال زیر توجه نمایید.

```
public int Add(int x, int y) {
    int z = x + y;
    return z;
}
public string Add(string string1, string string2) {
    string string3 = string1 + string2;
    return string3;
}
```

### توصیف‌کننده‌های متد

۱. static: برای فراخوانی این گونه متدها به جای نام شیء از نام کلاس استفاده می‌شود.
۲. public: امکان فراخوانی این گونه متدها در کل برنامه‌ی کاربردی وجود دارد.
۳. private: این گونه متدها فقط داخل کلاسی که تعریف شده‌اند، قابل استفاده هستند.
۴. protected: این گونه متدها فقط داخل کلاسی که تعریف شده‌اند و زیرکلاس‌های آن قابل استفاده هستند.
۵. internal: فقط داخل Assembly که تعریف شده‌اند، قابل استفاده هستند.
۶. extern: این نوع متدها در زبان C# هیچ محدودیتی نداشته و حتی داخل یک زبان دیگر هم قابل استفاده هستند.



۷. **abstract**: در صورتی که بخواهیم متدی را در یک کلاس معرفی کنیم و پیاده‌سازی آن را در کلاس‌های فرزند آن انجام دهیم، باید از کلمه کلیدی **abstract** استفاده کنیم. متدهای **abstract** فقط داخل کلاس‌های **abstract** قابل تعریف هستند.

۸. **virtual**: در صورتی که بخواهیم متد یک کلاس را در کلاس فرزندش **override** نماییم باید آن را به صورت **virtual** تعریف کنیم.

۹. **override**: برای **override** کردن یک متد **virtual** یا **abstract** باید قبل از کلمه کلیدی **override** استفاده کنیم.

```
abstract class ShapesClass {
    abstract public int Area();
}
```

```
class Square : ShapesClass {
    int x, y;
    public override int Area() {
        return x * y;
    }
}
```

۱۰. **new**: به منظور **override** کردن یک متد از کلمه کلیدی **new** نیز می‌توانیم استفاده نماییم.

```
public class BaseC {
    public int x;
    public void Invoke() { }
}
public class DerivedC : BaseC {
    new public void Invoke() { }
}
```

### متد سازنده

متد سازنده<sup>۱</sup> هنگام مقداردهی اولیه‌ی کلاس فراخوانی می‌شود. به عبارت دیگر هنگامی که از شیئی از یک کلاس ساخته می‌شود، متد سازنده فراخوانی می‌شود. متد سازنده باید هم نام کلاس بوده و نباید دارای مقدار بازگشتی باشد. نکته: در صورتی که کلاس دارای سازنده باشد، سازنده پیش‌فرض توسط کامپایلر ایجاد نخواهد شد.

نکته: تعریف سازنده برای کلاس اجباری نیست. اگر سازنده‌ای برای کلاس تعریف نکنید کامپایلر یک سازنده‌ی پیش‌فرض می‌سازد. این سازنده بسیار ابتدایی بوده و فقط فیلدهای عضو را مقداردهی اولیه می‌کند.

```
<modifier> <constructor name>(param1, param2, ...) {
    ...
}
```

مثال:

```
public class Employee {
    string Name;
    public Employee() {
        Name = "John";
    }
    public Employee(string EmployeeName) {
        Name = EmployeeName;
    }
}
```

با توجه به تعریف متدهای سازنده کلاس **Employee** برای تعریف شیئی از کلاس **Employee** به یکی از روش‌های زیر عمل می‌کنیم.

1. `Employee employee1 = new Employee ("Smith");`
2. `Employee employee1 = new Employee ();`

**نکته:** متد سازنده پیش‌فرض نباید دارای پارامتر باشد.

**نکته:** برای فراخوانی متد سازنده کلاس والد به صورت زیر عمل می‌کنیم.

```
class Employee {
    public Employee() {
        ...
    }
}
class Salary : Employee {
    public Salary() : base() {
        ...
    }
}
```

عبارت `base()` در قطعه کد فوق بیانگر متد سازنده کلاس والد می‌باشد.

**نکته:** امکان Overload کردن متد سازنده نیز وجود دارد برای این منظور به صورت زیر عمل می‌کنیم.

```
public class Time {
    private int Year;
    private int Month;
    private int Date;
    private int Hour;
    private int Minute;
    private int Second;
    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 Year, int Month, int Date, int Hour, int Minute, int Second){
        this.Year = Year;
        this.Month = Month;
        this.Date = Date;
        this.Hour = Hour;
        this.Minute = Minute;
        this.Second = Second;
    }
}
```

به منظور ایجاد شیء از کلاس فوق به یکی از روش‌های زیر عمل می‌کنیم

1. `System.DateTime currentTime = System.DateTime.Now;`  
`Time t = new Time(currentTime);`
2. `Time t2 = new Time(2000, 11, 18, 11, 03, 30);`

**نکته:** امکان تعریف متد سازنده به صورت `static` وجود دارد. سازنده‌های ایستا فقط یک بار اجرا می‌شوند. وقتی

کلاس دارای فیلدهای `static` باشد، از متد سازنده‌ی ایستا برای مقداردهی اولیه فیلدهای ایستا استفاده می‌شود.

### مخرب‌ها

مخرب‌ها<sup>۱</sup> متدهایی هستند که هنگام تخریب اشیاء توسط Garbage Collector فراخوانی می‌شوند. برای تعریف متد مخرب کلاسی به نام `MyClass` به صورت زیر عمل می‌کنیم.

```
~MyClass() { // Implementation. }
```

کامپایلر C# قطعه کد فوق را به صورت زیر ترجمه می‌کند.

```
protected override void Finalize()
```

1- Destructor

```
{
    try{// Implementation.}
    finally{ base.Finalize( );}
}
```

### توصیف‌کننده‌های کلاس

۱. `public`: برای تعریف کلاس‌هایی که توسط تمام برنامه‌ها قابل استفاده باشد، کاربرد دارد.
۲. `internal`: برای تعریف کلاس‌هایی که فقط داخل `Assembly` که تعریف شده‌است، قابل استفاده باشد، کاربرد دارد.
۳. `static`: اگر کلاسی فقط دارای خصوصیات و توابع ایستا باشد، خود کلاس هم می‌تواند ایستا باشد. یک کلاس ایستا از لحاظ عملکرد مانند زمانی است که کلاسی را با یک سازنده ایستای خصوصی تعریف کنید. هرگز نمی‌توان یک نمونه از کلاس ایجاد کرد.
۴. `abstract`: برای تعریف کلاس‌هایی که امکان ایجاد شیء از آنها وجود ندارد ولی امکان ارث‌بری از آنها وجود دارد.
۵. `sealed`: برای تعریف کلاس‌هایی که امکان ایجاد شیء از آنها وجود دارد ولی امکان ارث‌بری از آنها وجود ندارد.

### ساختار

برای تعریف ساختار به صورت زیر عمل می‌کنیم.

```
struct Dimensions {
    public double Length;
    public double Width;
    Dimensions(double length, double width){
        Length = length;
        Width = width;
    }
    public double Diagonal{
        get{
            return Math.sqrt(Length*Length*Width*Width);
        }
    }
}
```

نکته: ساختارها از امکان وراثت پشتیبانی نمی‌کنند.

نکته: ساختارها به صورت پیش فرض از کلاس `ValueType` ارث می‌برند.

نکته: تعریف متغیر از ساختار و استفاده از آن به یکی از صورت‌های زیر است.

1. `Dimensions point = new Dimensions();`  
`point.Length = 3;`  
`point.Width = 6;`
2. `Dimensions point;`  
`point.Length = 3;`  
`point.Width = 6;`

نکته: امکان تعریف سازنده‌ی بدون پارامتر برای ساختارها وجود ندارد.

### کلاس‌های جزئی

با استفاده از کلمه‌ی کلیدی `partial` می‌توان یک کلاس، ساختار یا واسط را در چند فایل قرار داد. در شرایطی که چند برنامه‌نویس روی یک کلاس کار می‌کنند، این قابلیت مفید می‌باشد. دو فایل زیر را در نظر بگیرید.

```
partial class TheBigClass : TheBigBaseClass, IBigClass{
    public void MethodOne(){

    }
}
```

```
partial class TheBigClass : IOtherBigClass{
    public void MethodTwo(){

    }
}
```

پس کامپایل یکی شده و به صورت زیر در خواهد آمد.

```
class TheBigClass : TheBigBaseClass, IBigClass, IOtherBigClass {
    public void MethodOne(){

    }
    public void MethodTwo(){

    }
}
```

**نکته:** در صورتی که کلاس دارای والد نباشد، به صورت پیش فرض از کلاس Object ارث می‌برد.

## وراثت

برای افزایش قابلیت استفاده<sup>۱</sup> در زبان‌های شیء‌گرا مفهومی به نام وراثت<sup>۲</sup> در نظر گرفته شده است. مفهوم وراثت به این معنی است که یک کلاس خصایص و متدهای کلاس دیگر را به ارث می‌برد. برای این منظور در زبان برنامه‌نویسی C# به صورت زیر عمل می‌کنیم.

```
class Sub-Class : Super-Class
```

کلاسی که ارث می‌برد را Sub Class، child یا کلاس فرزند و کلاسی که از آن ارث برده می‌شود، را parent، Super Class یا کلاس والد می‌نامیم.

مثال:

```
class Employee
{
    public void EmployeeName() {
        ...
    }
}
class Salary : Employee {
    public void CalculateSalary() {
        ...
    }
}
class Bonus {
    static void Main() {
        Salary salary1 = new Salary();
        salary1.EmployeeName();
        salary1.CalculateSalary();
    }
}
```

همان‌طور که در مثال فوق مشاهده می‌کنید شیء salary1 نمونه‌ای از کلاس Salary می‌باشد و با توجه به اینکه کلاس Salary فرزند کلاس Employee می‌باشد، امکان فراخوانی متد EmployeeName() در آن وجود دارد.

1- Reusability  
2- Inheritance

**نکته:** ساختارها همواره از System.ValueType مشتق می‌شوند.

**نکته:** ساختار توانایی ارث‌بری از ساختارهای دیگر را ندارد.

### عملگر is

با توجه به اینکه تبدیل صریح اشیاء به یکدیگر ممکن است باعث بروز خطای زمان اجرا شود و مدیریت اینگونه خطاها با استفاده از Exception زمان‌بر می‌باشد، به منظور جلوگیری از بروز خطا از عملگرهای is و as استفاده می‌کنیم. در صورتی که از عملگر is استفاده کنید، نتیجه یک متغیر Boolean خواهد بود که نشان‌دهنده‌ی امکان تبدیل است. برای درک بهتر مطلب به مثال زیر توجه نمایید.

```
using System;
public class TestCating {
    static void Main(){
        String derivedObj = "Dummy";
        Object baseObj1 = new Object();
        Object baseObj2 = derivedObj;
        Console.WriteLine("baseObj2 {0} String", baseObj2 is String ? "is" : "is not");
        Console.WriteLine("baseObj1 {0} String", baseObj1 is String ? "is" : "is not");
        Console.WriteLine("derivedObj {0} String", derivedObj is String ? "is" : "is not");
    }
}
```

### عملگر as

عملگر as مشابه عملگر is می‌باشد با این تفاوت که مقدار بازگشتی آن شیء نتیجه عمل تبدیل است و در صورتی که عمل تبدیل به درستی انجام نشود، مقدار بازگشتی null خواهد بود. برای درک بهتر مطلب به مثال زیر توجه نمایید.

```
using System;
public class BaseType {}

public class DerivedType : BaseType {}

public class TestCasting{
    static void Main() {
        DerivedType derivedObj = new DerivedType();
        BaseType baseObj1 = new BaseType();
        BaseType baseObj2 = derivedObj;
        DerivedType derivedObj2 = baseObj2 as DerivedType;
        if( derivedObj2 != null ) {
            Console.WriteLine( "Conversion Succeeded" );
        } else {
            Console.WriteLine( "Conversion Failed" );
        }
        derivedObj2 = baseObj1 as DerivedType;
        if( derivedObj2 != null ) {
            Console.WriteLine( "Conversion Succeeded" );
        } else {
            Console.WriteLine( "Conversion Failed" );
        }
        BaseType baseObj3 = derivedObj as BaseType;
        if( baseObj3 != null ) {
            Console.WriteLine( "Conversion Succeeded" );
        } else {
            Console.WriteLine( "Conversion Failed" );
        }
    }
}
```

```

    }
}

```

خروجی مثال فوق به صورت زیر است.

```

Conversion Succeeded
Conversion Failed
Conversion Succeeded

```

## پنهان‌سازی توابع<sup>۱</sup>

در صورتی که در سلسله مراتب اشیاء دو کلاس که رابطه پدر و فرزندی با یکدیگر دارند، دو متد همانام وجود داشته‌باشد، متد کلاس فرزند متد کلاس والد را Override می‌کند. برای درک بهتر مطلب به مثال زیر توجه کنید.

مثال ۱:

۱. کلاس والد

```

public class parent {
    public void print() {
        Console.WriteLine("Base Class");
    }
    public void print2() {
        Console.WriteLine("Not Overload");
    }
}

```

۲. کلاس فرزند

```

public class child :parent {
    public void print() {
        Console.WriteLine("Child Class");
    }
}

```

در این مثال اگر شیئی از کلاس child ساخته شود و متد ( ) print فراخوانی شود عبارت Child Class در خط فرمان مشاهده خواهد شد. در حالی که متد ( ) print2 فراخوانی شود، عبارت Not Overloaded مشاهده خواهد شد. همانطور که ملاحظه می‌کنید در حالت اول متد کلاس والد Override شده و اجرا نمی‌شود.

نکته: در صورتی که می‌خواهید متد کلاس والد را از طریق متد کلاس فرزند فراخوانی کنید از کلمه کلیدی base به صورت زیر استفاده کنید.

```

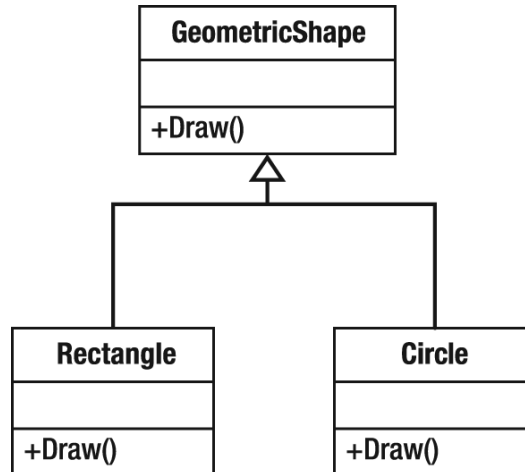
public class parent {
    public void print() {
        Console.WriteLine("Base Class");
    }
}

public class child :parent {
    public void print() {
        Console.WriteLine("Child Class");
        Baser.print();
    }
}

```

## چندریختی

Poly به معنای بسیار و morph به معنای فرم است. به عبارت دیگر چندریختی<sup>۱</sup> به معنای امکان استفاده از فرم‌های متفاوت بدون درگیر شدن در جزئیات آن‌ها می‌باشد. به عنوان نمونه وقتی شرکت مخابرات سیگنال زنگ را به خط تلفن ارسال می‌کند، اطلاعاتی از نوع تلفن ندارد. به عبارت دیگر شرکت مخابرات سیگنال را برای نوع داده‌ای پایه یا همان تلفن می‌فرستند. برای درک بهتر مطلب به مثال زیر توجه نمایید.



همان‌طور که ملاحظه می‌کنید کلاس‌های Rectangle و Circle از کلاس GeometricShape ارث می‌برند و کلاس GeometricShape دارای متدی به نام Draw() است که توسط کلاس‌های فرزند آن (Rectangle و Circle) Override شده‌است.

چندریختی بیانگر شرایطی مانند مثال فوق می‌باشد که یک متغیر می‌تواند رفتار متفاوتی از خود بروز دهد. برای درک بهتر مطلب به قطعه کد زیر توجه نمایید.

```

public class GeometricShape {
    public virtual void Draw(){
        // Do some default drawing stuff.
    }
}
public class Rectangle : GeometricShape {
    public override void Draw() {
        // Draw a rectangle
    }
}
public class Circle : GeometricShape {
    public override void Draw() {
        // Draw a circle
    }
}
public class Polymorphism {
    private static void DrawShape( GeometricShape shape ) {
        shape.Draw();
    }
    static void Main() {
        Circle circle = new Circle();
        Rectangle rectangle = new Rectangle();
        DrawShape(rectangle);
        DrawShape( circle );
    }
}
  
```

همانطور که ملاحظه می‌کنید دو شیئی از کلاس‌های Circle و Rectangle تعریف شده‌است و پارامتر متد DrawShape از نوع کلاس GeometricShape می‌باشد. بنابراین هنگامی که متد DrawShape فراخوانی می‌گردد، آرگومان ارسالی به GeometricShape تبدیل می‌شود. هنگامی که متد draw() در بدنه متد DrawShape فراخوانی می‌شود با توجه به آرگومان ارسالی متد مربوط در یکی از کلاس‌های Circle یا Rectangle فراخوانی می‌گردد.

### واسط

واسط<sup>۱</sup> می‌تواند شامل تعریف متدها، خصوصیات، شاخص‌ها و رخدادها باشد. برای تعریف یک واسط به صورت زیر عمل می‌کنیم. لازم به ذکر است که نام واسط‌ها در .Net Framework با حرف I آغاز می‌شود. پیشنهاد می‌شود که شما هم از همین استاندارد استفاده کنید.

```
interface <interface name> {
    ...
}
```

واسط نمی‌تواند شامل متد سازنده و عملگرهای سفارشی باشد. در ضمن تمام متدهای واسط‌ها public بوده و نمی‌توانند virtual یا static باشند.

هنگام تعریف واسط نیازی به ذکر توصیف‌کننده وجود ندارد، زیرا تمام واسط‌ها به صورت پیش‌فرض public هستند. لازم به ذکر است که یک کلاس می‌تواند چندین واسط را پیاده‌سازی نماید. برای درک بهتر مطلب به مثال زیر توجه نمایید.

```
interface IEmployee {
    ...
}

interface ISalary {
    ...
}

class Employee: IEmployee, ISalary {
    ...
}
```

هنگامی که یک کلاس واسطی را پیاده‌سازی می‌کند، باید تمام متدهای آن را پیاده‌سازی کند. بنابراین کلاس Employee باید تمام متدهای واسط‌های Employee و Salary را پیاده‌سازی کند. لازم به ذکر است که قاعده وراثت در واسط‌ها نیز برقرار است. با این تفاوت امکان وراثت چندگانه<sup>۲</sup> در مورد واسط‌ها وجود دارد.

### آرایه

یک آرایه تعدادی متغیر با یک نوع داده‌ای را دربر می‌گیرد. هر کدام از این متغیرها دارای یک اندیس می‌باشند. از اندیس عناصر آرایه برای دستیابی به داده‌های آن‌ها استفاده می‌شود. توجه داشته‌باشید که آرایه در زبان C# با آرایه‌های زبان ++C تفاوت دارند. در واقع هر آرایه در زبان C# یک شیء از نوع کلاس Array بوده و دارای متدها و خصایص خاص خود می‌باشد.

### انواع آرایه

۱. آرایه یک بعدی: عناصر این نوع آرایه فقط یک اندیس دارند.

```
<Data Type>[ ] <Variable> = new <Data Type>[Digit];
```

1- Interface  
2- Multi Inheritance



مثال:

```
int [] Integer = new int [20];
```

مثال:

```
int [] Integer;
Integer = new int [20];
```

۲. آرایه چندبعدی: عناصر این نوع آرایه بیش از یک اندیس دارند. آرایه‌های چندبعدی به صورت ممکن تعریف شوند.

ا. آرایه مستطیلی<sup>۱</sup>: تعداد عناصر این نوع آرایه در سطرهاى مختلف برابر است.

مثال:

```
int [,] Integer = { { 2,3 }, { 3,4 }, { 4,5 } };
int [, ,] Integer = { { 1,2,3 }, { 2,3,4 }, { 3,4,5 } };
```

ب. آرایه ناهموار<sup>۲</sup>: تعداد عناصر این نوع آرایه در سطرهاى مختلف متفاوت است.

مثال:

```
int [][] Integer = new int [2][];
Integer[0] = new int [2];
Integer[1] = new int [5];
```

نکته: برای مقداردهی اولیه به عناصر آرایه می‌توانید از حلقه‌ها استفاده کنید.

```
int [,] Integer = new int [5,10];
for (int x = 0; x < 5; x++) {
    for (int y = 0; y < 10; y++)
        Integer [x,y] = x*y;
}
```

### خصایص و متدهای کلاس Array

۱. **Length**: طول آرایه یا تعداد عناصر موجود در آن. اگر آرایه چند بعدی باشد، تعداد عناصر تمام رتبه‌ها را بر می‌گرداند.

```
int i = Integer.Length;
```

برای تعیین طول آرایه‌های چند بعدی به صورت زیر عمل می‌کنیم.

```
int i = Integer.GetLength(1);
```

۲. **LongLength**: خاصیت **LongLength** طول آرایه را در قالب یک مقدار **long** باز می‌گرداند. اگر تعداد عناصری بیش‌تر از یک مقدار عددی ۳۲ بیتی باشد، برای دریافت تعداد عناصر آن به این خصوصیت نیاز خواهیم داشت.

۳. **Rank**: با خصوصیت رتبه، می‌توانید تعداد ابعاد آرایه را به دست آورید.

۴. **CreateInstance()**: کلاس **Array** انتزاعی<sup>۳</sup> است، بنابراین نمی‌توان آرایه را با استفاده از یک سازنده ایجاد کرد. برای ایجاد آرایه باید از متد ایستای **CreateInstance()** استفاده کرد.

```
Array intArray = Array.CreateInstance(typeof(int), 5)
for(int i=0; i<5; i++){
    intArray.SetValue(33, i);
}
for(int i=0; i<5; i++){
    Console.WriteLine(intArray.GetValue(i));
}
```

1- Rectangular  
2- Orthogonal  
3- Abstract

۵. متد Clone() هنگامی که یک متغیر آرایه را به متغیر آرایه دیگر نسبت می‌دهید هر دو متغیر به یک نقطه از حافظه اشاره می‌کنند، بنابراین برای ایجاد کپی از آرایه به صورت زیر عمل می‌کنیم.

```
int[] intArray2 = intArray.Clone();
```

۶. متد Sort(): کلاس Array از روش مرتب‌سازی حبابی برای مرتب‌سازی عناصر آرایه به وسیله متد Sort() استفاده می‌کند.

```
Array.Sort(intArray);
```

مثال:

```
int[] marks = { 70, 62, 53, 44, 75, 68 };
int I = marks.Length;
Array.sort(marks);
for(int x = 0; x < I; x++){
    Console.WriteLine(x);
}
```

تمرین: سایر متدها و خصایص کلاس Array را با توجه به مستندات MSDN شرح دهید.

### کلمه‌ی کلیدی params

در صورتی که قبل از نام و نوع داده‌ای پارامتر یک متد از کلمه کلیدی params استفاده کنید، امکان ارسال تعداد دلخواهی از متغیرها با نوع داده‌ای متناسب با پارامتر تعریف شده وجود خواهد داشت. در واقع مقادیر ارسالی به عنوان یک آرایه در نظر گرفته می‌شوند. برای درک بهتر مطلب به مثال زیر توجه نمایید.

```
public class Tester {
    static void Main() {
        Tester t = new Tester();
        t.DisplayVals(5, 6, 7, 8);
        int[] explicitArray = new int[5] { 1, 2, 3, 4, 5 };
        t.DisplayVals(explicitArray);
    }
    public void DisplayVals(params int[] intVals) {
        foreach (int i in intVals) {
            Console.WriteLine("DisplayVals {0}", i);
        }
    }
}
```

همان‌طور که ملاحظه می‌کنید، تفاوتی بین ارسال آرایه و ارسال تعدادی متغیر با نوع داده‌ای int وجود ندارد.

نکته: همان‌طور که ملاحظه کردید جهت دستیابی به عناصر یک آرایه، نام آرایه را به همراه اندیس عنصر موردنظر به صورت زیر درج می‌نماییم.

```
ArrayName[index]
```

### کلاس ArrayList

از این کلاس به منظور ایجاد آرایه با طول متغیر و قابل افزایش استفاده می‌شود. کلاس ArrayList در فضای نامی System.Collection قرار دارد. کلاس ArrayList واسطه IList را پیاده‌سازی می‌نماید. هنگامی که شیئی از این کلاس ایجاد می‌کنید، فضای حافظه به آن اختصاص داده می‌شود. امکان تعیین فضای حافظه ی اولیه هنگام تعریف شیء ArrayList وجود دارد. امکان افزودن عناصر جدید به اشیاء ArrayList وجود دارد. در صورتی که حجم حافظه اختصاص داده شده به ArrayList پر شود، فضای حافظه جدید به آن اختصاص داده می‌شود. برای درک بهتر مطلب به مثال زیر توجه نمایید.

```
using System;
using System.Collections;
public class ArrayList1 {
    public static void Main() {
```

```

        ArrayList list1 = new ArrayList();
        list1.Add("This");
        list1.Add(" is");
        list1.Add(" a");
        list1.Add("sample ");
        list1.Add("ArrayList.");
    }
}

```

در این مثال ابتدا شیئی از کلاس ArrayList ایجاد شده و سپس با استفاده از متد Add() پنج عنصر به آن اضافه شده‌است.

### نوع داده شمارشی

نوع داده‌ی شمارشی<sup>۱</sup> مقادیر را به صورت مجموعه‌ی کاربرپسند<sup>۲</sup> ذخیره می‌کند. این مجموعه را لیست شمارنده<sup>۳</sup> می‌نامیم. نوع داده پیش‌فرض انواع داده‌ای شمارشی integer می‌باشد. برای استفاده از نوع داده شمارشی ابتدا باید یک شمارنده تعریف نماییم. برای تعریف شمارنده به صورت زیر عمل می‌کنیم.

```

<modifier> enum <enumeration name> {
    ...
};

```

توصیف‌کننده‌های مجاز برای نوع داده شمارشی عبارتند از: new, public, protected, private, internal. برای درک بهتر نوع داده شمارشی به مثال زیر توجه نمایید.

```

public enum months {
    January, February, March, April, May, June, July,
    August, September, October, November, December
}

```

مثال فوق لیست شمارشی از ماه‌های سال به وجود می‌آورد، تنها اشکال قطعه کد فوق این است که نوع داده شمارشی به صورت پیش‌فرض از صفر شروع می‌شود، در حالی که شماره ماه‌های سال از یک شروع می‌شود. برای رفع این مشکل می‌توان شماره اولین عنصر نوع داده شمارشی را به صورت زیر تعیین نمود.

```

public enum months {
    January = 1, February, March, April, May, June, July,
    August, September, October, November, December
}

```

مثال:

```

namespace ConsoleApplication1 {
    enum orientation : byte {
        north = 1,
        south = 2,
        east = 3,
        west = 4
    }
    struct route {
        public orientation direction;
        public double distance;
    }
    class Program {
        static void Main(string[] args) {
            route myRoute;
            int myDirection = -1;
            double myDistance;
            Console.WriteLine("1) North\n2) South\n3) East\n4) West");
        }
    }
}

```

1- Enumeration  
2- User Friendly  
3- Enumerator List

```

do
{
    Console.WriteLine("Select a direction:");
    myDirection = Convert.ToInt32(Console.ReadLine());
}
while ((myDirection < 1) || (myDirection > 4));
Console.WriteLine("Input a distance:");
myDistance = Convert.ToDouble(Console.ReadLine());
myRoute.direction = (orientation)myDirection;
myRoute.distance = myDistance;
Console.WriteLine("myRoute specifies a direction of {0} and a " +
    "distance of {1}", myRoute.direction, myRoute.distance);
Console.ReadKey();
}
}
}

```

## رشته‌ها

برای استفاده از رشته‌ها در زبان برنامه‌نویسی C# از کلاس String یا StringBuilder استفاده می‌شود.

## کلاس String

```
String str = "Hello";
```

توابع کلاس String به شرح زیر می‌باشد.

۱. Compare(): جهت مقایسه دو رشته استفاده می‌شود.
۲. Concat(): چند رشته را ترکیب کرده و آن‌ها را در یک رشته قرار می‌دهد.
۳. CopyTo(): تعداد کاراکترهای مشخصی را از یک اندیس مشخص، در یک رشته‌ی جدید کپی می‌کند.
۴. Format(): جهت تعیین فرمت رشته استفاده می‌شود.
۵. IndexOf(): اولین موقعیت یک کاراکتر یا زیر رشته را باز می‌گرداند. برای دستیابی به آخرین موقعیت می‌توانید از متد LastIndexOf() استفاده نمایید.
۶. IndexOfAny(): در صورتی که بخواهید در بین مجموعه‌ای از کاراکترها، اولین موقعیت یک کاراکتر را تعیین نمایید، از این متد استفاده می‌نمایید. برای دستیابی به آخرین موقعیت می‌توانید از متد LastIndexOfAny() استفاده نمایید.
۷. Insert(): یک رشته را در محل مشخصی از رشته‌ی دیگر درج می‌کند.
۸. Join(): با ترکیب آرایه‌ای از رشته‌ها یک رشته جدید ایجاد می‌کند.
۹. PadLeft(): یک کاراکتر تکرار مشخص را به سمت چپ رشته اضافه می‌کند.
۱۰. PadRight(): یک کاراکتر تکرار مشخص را به سمت راست رشته اضافه می‌کند.
۱۱. Replace(): برای جایگزین کردن یک رشته با رشته‌ی دیگر استفاده می‌شود.
۱۲. Split(): جهت تقسیم یک رشته به چند زیر رشته استفاده می‌شود.
۱۳. SubString(): زیررشته‌ای از رشته‌ی اصلی را از اندیس مشخص شده و به تعداد کاراکتر موردنظر جدا می‌کند.
۱۴. ToLower(): جهت تبدیل کاراکترهای رشته به حروف کوچک استفاده می‌شود.
۱۵. ToUpper(): جهت تبدیل کاراکترهای رشته به حروف بزرگ استفاده می‌شود.
۱۶. Trim(): فضاهای خالی ابتدا و انتهای رشته را حذف می‌کند.

**نکته:** برای اضافه کردن یک رشته به انتهای رشته‌ی دیگر به صورت زیر عمل می‌کنیم.

```

string string1 = "John";
string string2 = "Floyd";

```

```
string string3 = string1 + string2;
```

**تمرین:** تمام متدهای کلاس String را با ذکر مثال تشریح نمایید. (با توجه به MSDN)

**نکته:** نقطه ضعف کلاس String این است که هرگاه از متدهای آن یا عملگر + استفاده می‌کنیم، رشته‌ی جدیدی ایجاد می‌شود و اشاره‌گر رشته به محل جدید اشاره می‌کند. در این حالت رشته قبلی جزء متغیرهای بدون ارجاع خواهد بود و توسط زباله‌روب جمع‌آوری خواهد شد. برای رفع این مشکل از StringBuilder استفاده می‌کنیم.

### کلاس StringBuilder

```
StringBuilder sb = new StringBuilder("Hello");
```

علاوه‌براین می‌توان یک شیء از نوع StringBuilder با ظرفیت مشخص و بدون محتوا ایجاد نمود.

```
StringBuilder sb = new StringBuilder(20);
```

توابع کلاس StringBuilder به شرح زیر می‌باشد.

1. Append(): یک رشته به رشته جاری اضافه می‌کند.
2. AppendFormat(): رشته‌ای را که نقش تصریح‌کننده‌ی فرمت را دارد، اضافه می‌کند.
3. Insert(): یک زیررشته را به رشته‌ی جاری اضافه می‌کند.
4. Remove(): یک زیررشته را از رشته‌ی جاری حذف می‌کند.
5. Replace(): یک زیررشته از رشته‌ی جاری را با زیررشته‌ی دیگر جایگزین می‌کند.

**تمرین:** کلاس StringBuilder، خصایص و متدهای آن را با ذکر مثال تشریح نمایید. (با توجه به MSDN)

**تمرین:** برنامه‌ای بنویسید که رشته‌ای را از ورودی خوانده و تشخیص دهد که رشته‌ی ورودی یک رشته Palindrome می‌باشد یا خیر؟

**تمرین:** برنامه‌ای بنویسید که یک عدد چهار رقمی را از ورودی خوانده، آن را با استفاده از روش زیر رمزنگاری کرده و سپس چاپ نماید.

روش رمزنگاری: ابتدا هر کاراکتر را با ۷ جمع کند و سپس باقیمانده آن را بر ۱۰ حساب کند. سپس جای رقم اول را با رقم سوم و رقم دوم را با رقم چهارم عوض کند.

**تمرین:** برنامه‌ای بنویسید که عددی را از ورودی خوانده و فاکتوریل آن را محاسبه و چاپ نماید.

**تمرین:** برنامه‌ای بنویسید که عددی را از ورودی خوانده و مقدار  $e^x$  را چاپ نماید

$$e^x = 1 + x/1! + x^2/2! + x^3/3! + \dots$$

**تمرین:** برنامه‌ای بنویسید که ۲۰ عدد را از ورودی خوانده و هر کدام را که تکراری نباشد به آرایه اضافه نماید.

**تمرین:** برنامه‌ای بنویسید که رشته‌ای را از ورودی پذیرفته و کلمات رشته را داخل "" قرار دهد.

**تمرین:** برنامه‌ای بنویسید که آرایه‌ای را از ورودی خوانده، Binary Search و Quick Sort را روی آن انجام داده و نتیجه را چاپ نماید.

### پیاده‌سازی عملگرهای سفارشی

عبارت فوق معادل Operator Overloading است و فقط به منظور درک بهتر مطلب به این صورت ترجمه شده‌است. برای ایجاد یک عملگر از کلمه کلیدی Operator به صورت زیر استفاده می‌کنیم.

```
public static Matrix operator +(Matrix lhs, Matrix rhs){
    ...
}
```

همان‌طور که ملاحظه می‌کنید برای ایجاد یک عملگر سفارشی پس از کلمه کلیدی operator نام عملگر ذکر می‌گردد. توجه داشته باشید که وجود کلمه کلیدی static الزامی می‌باشد.

برای استفاده از عملگر ایجاد شده به صورت زیر عمل می‌کنیم.

```
Matrix theSum = first + second;
```

توجه داشته باشید که عبارت فوق توسط کامپایلر C# به قطعه کد زیر تبدیل می‌شود.

`Matrix theSum = Matrix.operator +(first, second)`

**تمرین:** برنامه‌ای بنویسید که توانایی جمع و ضرب ماتریس‌ها را داشته‌باشد. برای این منظور از عملگرهای سفارشی استفاده کنید.

### عامل

عامل<sup>۱</sup> نوع خاصی از کلاس می‌باشد که فقط شامل آدرس متدها می‌باشد. عامل‌ها اشیایی هستند که امکان ارسال متدها به عنوان پارامتر به متدهای دیگر را فراهم می‌آورند. به عبارت دیگر یک عامل امکان ذخیره ارجاع<sup>۲</sup> به یک متد را فراهم می‌آورد. برای درک بهتر مطلب به مثال زیر توجه نمایید.

```
class Program {
    delegate double ProcessDelegate(double param1, double param2);
    static double Multiply(double param1, double param2) {
        return param1 * param2;
    }
    static double Divide(double param1, double param2) {
        return param1 / param2;
    }
    static void Main(string[] args) {
        ProcessDelegate process;
        Console.WriteLine("Enter 2 numbers separated with a comma:");
        string input = Console.ReadLine();
        int commaPos = input.IndexOf(',');
        double param1 = Convert.ToDouble(input.Substring(0, commaPos));
        double param2 = Convert.ToDouble(input.Substring(commaPos+1, input.Length-commaPos-1));
        Console.WriteLine("Enter M to multiply or D to divide:");
        input = Console.ReadLine();
        if (input == "M") process = new ProcessDelegate(Multiply);
        else process = new ProcessDelegate(Divide);
        Console.WriteLine("Result: {0}", process(param1, param2));
        Console.ReadKey();
    }
}
```

همان طور که در مثال فوق ملاحظه می‌نمایید، در ابتدا یک عامل به صورت زیر تعریف شده‌است.

```
delegate double ProcessDelegate(double param1, double param2);
```

عامل فوق دارای دو پارامتر و نوع داده‌ی بازگشتی `double` می‌باشد.

متد `Main()` با تعریف متغیری به نام `process` از نوع `ProcessDelegate` آغاز می‌شود. در ادامه با توجه به ورودی کاربر مقداردهی متغیر `process` به صورت زیر انجام می‌شود.

```
if (input == "M")
    process = new ProcessDelegate(Multiply);
else
    process = new ProcessDelegate(Divide);
```

همان طور که ملاحظه می‌نمایید با توجه به ورودی کاربر ارجاع به یکی از متدهای `Multiply` یا `Divide` انجام می‌شود. سپس فراخوانی عامل صرف نظر از متدی که به آن ارجاع می‌کند، انجام می‌شود.

```
Console.WriteLine("Result: {0}", process(param1, param2));
```

همان طور که ملاحظه می‌کنید فراخوانی عامل تفاوتی با فراخوانی متد ندارد. تنها تفاوتی که بین متد و عامل وجود دارد، این است که امکان ارسال عامل به عنوان پارامتر به متد وجود دارد. برای این منظور به صورت زیر عمل می‌کنیم.

```
static void ExecuteFunction(ProcessDelegate process) {
    process(2.2, 3.3);
}
```

1- Delegate

2- Reference

## توابع بی‌نام

یک تابع بی‌نام قطعه‌کدی است که به عنوان پارامتر عامل استفاده می‌شود. برای درک بهتر مطلب به مثال زیر توجه کنید.

```
class Program {
    delegate String DelegateTest(String val);
    static void Main() {
        string mid = ", middle part, ";
        DelegateTest test = DelegateTest(String param){
            param += mid;
            param += " and this was added to the string.";
            return param;
        };
        Console.WriteLine(test("Start of string"));
        Console.ReadKey();
    }
}
```

## عبارات Lambda

در C# 3.0 یک ساختار دستوری جدید به نام عبارات Lambda برای تعریف توابع بی‌نام ایجاد شده است. عبارات Lambda را می‌توان به همراه عامل‌ها استفاده کرد. برای درک بهتر مطلب مثال قبل را به صورت زیر تغییر می‌دهیم.

```
class Program {
    delegate String DelegateTest(String val);
    static void Main() {
        string mid = ", middle part, ";
        DelegateTest test = param=>
        {
            param += mid;
            param += " and this was added to the string.";
            return param;
        };
        Console.WriteLine(test("Start of string"));
        Console.ReadKey();
    }
}
```

سمت چپ عبارت Lambda (سمت چپ =>) بیانگر پارامترهای تابع بی‌نام است و سمت راست آن پیاده‌سازی تابع را در بر می‌گیرد.

**نکته:** رویدادها مبتنی بر عامل‌ها هستند. به عبارت دیگر هر رویداد یک عامل است.

## Indexer

ممکن است مواردی پیش بیاید که لازم باشد به عناصر یک کلاس به صورت آرایه دسترسی داشته باشید. برای این منظور در زبان C# از Indexerها استفاده می‌نماییم. برای اینکه از Indexer در یک کلاس استفاده کنیم، باید ابتدا آنرا تعریف نماییم. نحوه تعریف Indexer به صورت زیر است.

`<modifier> <type> this [parameter-list]`  
عبارت `<type>` بیانگر نوع داده بازگشتی Indexer است. عبارت `this` نام Indexerها نام صریح ندارند، و `parameter-list` لیست پارامترهای قابل قبول برای عناصر کلاس می‌باشد. یک Indexer مانند یک Property می‌باشد، بنابراین امکان تعریف متدهای `get` و `set` برای آنها وجود دارد. با این تفاوت که در مورد Indexerها به جای یک خصیصه خاص از خود شیء استفاده می‌کنیم.  
برای درک بهتر مطلب به مثال زیر توجه نمایید.

```

public class SpellingList {
    protected string[] words = new string[size];
    static public int size = 10;
    public SpellingList() {
        for (int x = 0; x < size; x++)
            words[x] = String.Format("Word{0}", x);
    }
    public string this[int index] {
        get {
            string tmp;
            if( index >= 0 && index <= size-1 )
                tmp = words[index];
            else
                tmp = " ";
            return ( tmp );
        }
        set {
            if( index >= 0 && index <= size-1 )
                words[index] = value;
        }
    }
}

public class Indexer {
    public static void Main() {
        SpellingList myList = new SpellingList();
        myList[3] = "=====";
        myList[4] = "Brad";
        myList[5] = "was";
        myList[6] = "Here!";
        myList[7] = "=====";
        for ( int x = 0; x < SpellingList.size; x++ )
            Console.WriteLine(myList[x]);
    }
}

```

همان‌طور که در این مثال مشاهده می‌کنید، در صورتی که از Indexer استفاده نمایید، به جای عبارت `myList.words[i]` می‌توانید از `myList[i]` استفاده نمایید.

**نکته:** برای تبدیل انواع داده‌ای پایه به کلاس به صورت زیر عمل می‌کنیم.

```

int x = 100;
object y = (object) x;
sbyte z = (sbyte) y;

```

### Generic

یک ویژگی جدید CLR 2.0 معرفی Generic‌ها است. در CLR 1.0 برای تعریف کلاس‌های انعطاف‌پذیر و غیر مشخص در زمان کامپایل از کلاس Object استفاده می‌شد. اشکال این روش عدم وجود امنیت و نیاز به انجام عملیات تبدیل نوع است.

با وجود Generic‌ها دیگر نیازی به استفاده از کلاس Object نیست. کلاس‌های Generic امکان استفاده از انواع Generic را که در صورت نیاز با انواع داده‌ای مورد نظر جایگزین می‌شوند را فراهم می‌آورد. لازم به ذکر است که استفاده از Generic‌ها کارایی را افزایش می‌دهد، زیرا نیازی به انجام عملیات تبدیل نوع ندارد. به مثال زیر توجه کنید.

```

ArrayList list = new ArrayList();
list.Add(44);
int i1 = (int)list[0];

```



```
foreach(int i2 in list){
    Console.WriteLine(i2);
}
```

در این مثال هنگام فراخوانی متد Add و هنگام خواندن عناصر آرایه تبدیل نوع صورت می‌گیرد و انجام این عمل زمان‌بر است. برای رفع این مشکل می‌توان قطعه کد فوق را به صورت زیر تغییر داد.

```
List<int> list = new List<int>( );
list.Add(44);
int i1 = (int)list[0];
foreach(int i2 in list){
    Console.WriteLine(i2);
}
```

مزیت دیگر استفاده از Generic ها امنیت نوع است. در صورتی که از قطعه کد اول استفاده کنید، هر نوع شیئی را می‌توانید به آرایه اضافه کنید، در حالی که در آرایه قطعه کد دوم فقط نوع داده‌ای int را می‌پذیرد و برای انواع داده‌ای دیگر خطا می‌دهد.

تعریف کلاس‌های Generic و خصوصیات آن‌ها، توابع Generic، واسط‌های Generic، عامل‌های Generic به دانشجویان واگذار می‌شود.

### Thread

Thread کوچکترین واحد قابل اجرای یک برنامه‌ی کاربردی است. به عبارت دیگر یک Thread کوچکترین واحدی از برنامه است که زمان CPU به آن اختصاص می‌یابد. تصمیم‌گیری در مورد روند اجرای برنامه به عهده‌ی Thread است. بنابراین Thread ها ابزار مناسبی برای اجرای برنامه‌های کاربردی پیچیده و همچنین اجرای همزمان چندین برنامه هستند. علاوه بر این امکان اجرای همزمان چندین Thread در یک برنامه‌ی کاربردی وجود دارد. این گونه برنامه‌ها را برنامه‌های Multi Thread می‌نامند. هنگامی که کامپایلر C# یک برنامه‌ی کاربردی Multi Thread را اجرا می‌کند، چندین Thread به طور همزمان اجرا می‌شوند. اجرای همزمان چندین Thread موجب کاهش زمان اجرای برنامه‌ی کاربردی می‌شود.

در یک برنامه‌ی کاربردی Multi Thread امکان اجرای چند فعالیت به طور همزمان وجود دارد. برای مثال فرض کنید دستور چاپ ۱۰۰ صفحه را به چاپگر ارسال کرده‌اید. با توجه به اینکه عمل چاپ زمان‌گیر است در صورتی که برنامه‌ی کاربردی Multi Thread باشد، امکان ادامه عملیات همزمان با انجام عملیات چاپ وجود خواهد داشت. در این حالت دو Thread وجود خواهد داشت یک Thread برای انجام عمل چاپ و Thread دیگر برای انجام هر فعالیت دیگری که مدنظر کاربر باشد.

تمام برنامه‌های کاربردی با Thread ها سروکار دارند، ولی موقعیت‌هایی وجود دارد که استفاده از Thread ها بسیار مفید است. برخی از موقعیت‌ها را در قالب چند مثال بررسی می‌کنیم.

۱. همان‌طور که قبلاً هم ذکر کردیم برای انجام عملیات زمان‌بر می‌توانید از Thread ها استفاده کنید. در این

حالت می‌توانید دو Thread داشته باشید. Thread اول را *Worker Thread* و Thread دوم را *User Thread* می‌نامیم. Worker Thread عملیات زمان‌بر را انجام می‌دهد و User Thread عملیات مورد نظر کاربر را.

۲. انتقال اطلاعات در شبکه. برای مثال فرض کنید نیاز دارید حجم زیادی از اطلاعات را از یک شعبه اداره به شعبه دیگر انتقال دهید. در این حالت یک Thread در شعبه مبداء ایجاد می‌کنید که به سرویس‌دهنده شعبه مقصد وصل شود.

۳. برای اجرای برنامه‌های کاربردی که چندین عملیات متفاوت را در یک زمان انجام می‌دهند. برای مثال هنگامی که کاربر داده‌ای را در بانک اطلاعاتی وارد می‌کند، داده‌های بانک اطلاعاتی مادر هم به‌هنگام شوند. در این حالت می‌توانید دو Thread داشته باشید. Thread اول را *Worker Thread* و Thread دوم را *User Thread*

می‌نامیم. Worker Thread عملیات به‌هنگام‌سازی بانک اطلاعاتی مادر را انجام می‌دهد و User Thread عملیات ثبت داده‌های ورودی کاربر در بانک اطلاعاتی.

**نکته:** به منظور افزایش کارایی برنامه‌ی کاربردی تعداد Thread های برنامه کاربردی را تا حد ممکن کم کنید.

### ایجاد Thread

به‌منظور ایجاد Thread باید یک شیء از کلاس Thread که در فضای نامی System.Threading قرار دارد، ایجاد کنید. سپس باید delegate به نام ThreadStart() را فراخوانی کنید. ThreadStart() در فضای نامی System.Threading قرار دارد. برای درک بهتر مطلب به مثال زیر توجه کنید.

```
using System;
using System.Threading;
class Class1 {
    public void Method1() {
        Console.WriteLine("Method1 is the starting point of execution of the
                           thread");
    }
    public static void Main() {
        Class1 newClass = new Class1();
        Thread Thread1 = new Thread(new ThreadStart(newClass.Method1));
    }
}
```

نام شیء Thread در این مثال Thread1 است. همان‌طور که ملاحظه می‌کنید پارامتر ThreadStart() نام متدی است که Thread را آغاز می‌کند. پس از ایجاد شیء Thread نوبت فراخوانی متد Start() است. به این منظور از عبارت Thread1.Start() استفاده می‌شود.

**نکته:** به این خاطر که امکان استفاده از Thread در تمام بخش‌های برنامه کاربردی وجود دارد. توصیه می‌شود که به Thread ی که ایجاد می‌کنید، یک نام اختصاص بدهید. اختصاص نام به Thread موجب می‌شود که امکان استفاده مجدد از Thread برای برنامه‌نویسان دیگر ایجاد شود.

قطعه کد زیر Thread جهت انجام عملیات به‌هنگام‌سازی بانک اطلاعاتی ایجاد می‌کند. نام Thread ایجادشده Update Records Thread می‌باشد.

```
Thread Thread1 = new Thread(new ThreadStart(newClass.Method1));
Thread1.Name = "Update Records Thread";
```

علاوه بر خصیصه‌ی name که برای تعیین نام Thread استفاده می‌شود، خصایصی در کلاس Thread وجود دارد که امکان تعیین وضعیت Thread در حال اجرا را فراهم می‌آورد.

۱. خصیصه IsAlive : از این خصیصه جهت تعیین در حال اجرا بودن Thread استفاده می‌شود. مقدار بازگشتی این متد برای Thread های در حال اجرا برابر true و برای Thread هایی که در حال اجرا نیستند، برابر false خواهد بود.

۲. خصیصه ThreadSafe : این خصیصه وضعیت اجرای Thread را مشخص می‌کند. به عبارت دیگر مقدار بازگشتی این متد بیانگر این است که آیا اجرای Thread شروع شده‌است یا نه؟

### خاتمه Thread

گاهی اوقات لازم است، یک Thread در حال اجرا را خاتمه دهید. به عنوان نمونه مثال قبلی که از یک Thread جهت چاپ ۱۰۰ صفحه استفاده می‌شود را در نظر بگیرید. فرض کنید یک مورد چاپ ضروری پیش آمده است. در این صورت ابتدا باید Thread در حال اجرا را خاتمه دهید. به این منظور می‌توانید از متد Abort() کلاس Thread استفاده کنید.

```
Thread1.Abort();
```

## توالی Thread ها

این امکان در زبان C# وجود دارد که کامپایلر قبل از شروع اجرای یک Thread منتظر خاتمه اجرای Thread دیگر بماند. برای این منظور از متد Join() به صورت زیر استفاده می‌شود.

```
Thread1.Join();
```

فراخوانی متد Join() موجب می‌شود که Thread ی که توسط آن فراخوانی متد Join() شیء Thread1 انجام شده‌است، در انتظار خاتمه اجرای Thread1 بماند. در صورتی که زمان خاتمه Thread1 را نمی‌دانید می‌توانید زمان خاتمه اجرای Thread1 را به عنوان پارامتر متد Join() در نظر بگیرید.

## متوقف ساختن Thread

هنگامی که متد Abort() یک Thread فراخوانی می‌شود، دیگر امکان اجرای مجدد آن وجود ندارد، در حالی که هرگاه متد Suspend() یک Thread فراخوانی می‌شود، امکان اجرای مجدد آن وجود خواهد داشت. متد Suspend() به صورت زیر است.

```
Thread1.Suspend();
```

فراخوانی متد Suspend() شیء Thread را از بین نمی‌برد، فقط اجرای آن را متوقف می‌کند، بنابر این هرگاه که لازم باشد امکان فعال کردن Thread وجود خواهد داشت. به این منظور از متد Resume() به صورت زیر استفاده می‌کنیم.

```
Thread1.Resume();
```

## متوقف ساختن Thread برای یک بازه زمانی خاص

برای متوقف ساختن اجرای یک متد برای یک بازه زمانی مشخص می‌توانید از متد Sleep() استفاده نمایید. متد Sleep() دارای پارامتری از نوع Time می‌باشد و اجرای Thread را به اندازه زمان تعیین شده توسط پارامتر دریافتی متوقف می‌کند.

```
Thread.Sleep(2000);
```

همان‌طور که ملاحظه می‌کنید، برای فراخوانی متد Sleep() از نام کلاس استفاده شده‌است نه نام شیء. برای درک بهتر نحوه استفاده از کلاس Thread و نحوه استفاده از آن به مثال زیر توجه نمایید.

```
using System;
using System.Threading;
class Class1 {
    public void Method1() {
        Console.WriteLine("Method1 is the starting point of execution of the thread");
    }
    public static void Main() {
        Class1 newClass = new Class1();
        Thread Thread1 = new Thread(new ThreadStart(newClass.Method1));
        Thread1.Name = "Sample Thread";
        Thread1.Start();
        Console.WriteLine("The execution of Sample Thread has started.");
        Thread1.Abort();
    }
}
```

همان‌طور که در مثال فوق مشاهده می‌کنید، با استفاده از ThreadStart ارجاعی به متد Class1.Method1() ایجاد شده‌است.

هنگامی که متدهای Thread را فراخوانی می‌کنید، وضعیت (State) آن تغییر می‌کند. برای مثال هنگامی که متد Start() یک Thread فراخوانی می‌شود، وضعیت آن برابر running خواهد شد.

### وضعیت‌های Thread

| نام متد   | وضعیت Thread                                                                  |
|-----------|-------------------------------------------------------------------------------|
| Start()   | هرگاه متد Start() فراخوانی شود، وضعیت Thread برابر Running می‌شود.            |
| Sleep()   | هرگاه متد Sleep() فراخوانی شود، وضعیت Thread برابر Stopped می‌شود.            |
| Suspend() | هرگاه متد Suspend() فراخوانی شود، وضعیت Thread برابر SuspendRequested می‌شود. |
| Resume()  | هرگاه متد Resume() فراخوانی شود، وضعیت Thread برابر Running می‌شود.           |
| Abort()   | هرگاه متد Abort() فراخوانی شود، وضعیت Thread برابر AbortRequested می‌شود.     |

### اولویت Thread

اولویت Threadها ترتیب اجرای آنها را تعیین می‌کند. برای مثال در صورتی که بخواهیم دو Thread را روی یک دستگاه اجرا نماییم. Threadی که اولویت بالاتری دارد زودتر اجرا می‌شود. اولویت یک Thread یکی از مقادیر زیر می‌باشد.

۱. Highest: در صورتی که اولویت یک Thread برابر Highest باشد، قبل از بقیه Threadها اجرا می‌شود. در صورتی که کامپایلر C# با یک Thread که اولویت آن برابر Highest می‌باشد مواجه شود، تمام Threadهای دیگر را متوقف کرده و Thread مذکور را اجرا می‌کند.

۲. AboveNormal: در صورتی که اولویت یک Thread برابر AboveNormal باشد، قبل از بقیه Threadها به جز Threadهایی که اولویت آنها Highest می‌باشد، اجرا می‌گردد.

۳. Normal: در صورتی که اولویت یک Thread برابر Normal باشد، در لیست Threadهای در حال انتظار قرار گرفته و بخشی از زمان CPU به آن اختصاص داده می‌شود.

۴. Lowest و BelowNormal: Threadهایی که اولویت آنها برابر Lowest یا BelowNormal باشد، تنها در صورتی اجرا می‌شوند که هیچ Thread دیگری با اولویت بالاتر برای اجرا وجود نداشته باشد. برای تعیین اولویت یک Thread به صورت زیر عمل می‌کنیم.

```
Thread Thread1 = new Thread(new ThreadStart(newClass.Method1));
Thread1.Priority = ThreadPriority.Highest;
```

### همزمان‌سازی

همزمان‌سازی به معنای اطمینان از دسترسی تنها یک Thread به متغیر در یک لحظه می‌باشد. برای اینکه به اهمیت همزمان‌سازی پی ببرید به مثال زیر توجه نمایید.

فرض کنید دو Thread به نامهای Thread1 و Thread2 با اولویت یکسان وجود دارد و به صورت همزمان روی یک سیستم اجرا می‌شود. در صورتی که Thread1 در بازه زمانی خودش بخواهد مقداری به متغیری به نام variable1 با سطح دسترسی public نسبت دهد و Thread2 در بازه زمانی خودش بخواهد مقدار متغیر variable1 را بخواند و مقداری متغیر در بازه زمانی Thread1 خاتمه نیافته باشد، باعث بروز خطا در روند اجرایی برنامه خواهد شد. برای این منظور در زبان برنامه‌نویسی C# از کلمه کلیدی lock به صورت زیر استفاده می‌شود.

```
lock (variable1) {
    ...
}
```

### اعتبارسنجی و مدیریت خطا

اعتبارسنجی و مدیریت خطا جزء عملیات پشت‌زمینه تمام برنامه‌های کاربردی می‌باشد. برنامه‌ی کاربردی باید در برابر تمام خطاهای زمان اجرا مقاوم بوده و ایستادگی نماید. علاوه‌براین هنگام بروز خطا باید پیغام مناسبی به کاربر نمایش داده شود.

خطای زمان اجرا ممکن است به دلیل بروز شرایط نامناسب به‌وقوع بپیوندد. برای مثال فرض کنید برنامه‌ی کاربردی شما به صورتی طراحی شده‌است که یک فایل را به صورت فقط نوشتنی باز نماید و برای شرایطی که عملیات باز کردن فایل با موفقیت انجام نشود، پیش‌بینی صورت گرفته باشد. در این حالت اگر برنامه‌ی کاربردی دیگری فایل را باز کرده باشد و برنامه کاربردی شما اجرا شود، با خطای زمان اجرا مواجه خواهید شد. برای رفع این گونه مشکلات وجود مکانیزم مدیریت خطا در برنامه کاربردی ضروری است.

### اعتبارسنجی

به خاطر داشته‌باشید که همیشه قبل از ثبت داده‌های ورودی فرم در بانک اطلاعاتی لازم است عملیات اعتبارسنجی آنها را انجام دهید. فواید عملیات اعتبارسنجی به شرح زیر می‌باشد.

- کاهش زمان پاسخ: زمان پاسخ برنامه‌ی کاربردی کاهش می‌یابد، زیرا داده‌های نادرست برای بانک اطلاعاتی فرستاده نشده و باعث بروز خطا نخواهد شد.
- افزایش دقت اطلاعات: اطلاعات نادرست در بانک اطلاعاتی ثبت نمی‌شوند.
- افزایش کارایی بانک اطلاعاتی: با توجه به کاهش تعداد تراکنش‌ها کارایی بانک افزایش می‌یابد.

### تعیین مکانیزم اعتبارسنجی

راه کارهای متفاوتی برای انجام عملیات اعتبارسنجی فرم‌ها وجود دارد. برخی از این روش‌ها عبارتند از :

- انتخاب کنترل مناسب جهت دریافت ورودی کاربر
- اعتبارسنجی فرم، منظور کنترل تهی بودن فیلدهای اجباری، کنترل فرمت فیلدها و طول داده‌ای آن‌هاست.

### انتخاب کنترل مناسب

اغلب مواقع انتخاب کنترل مناسب به منظور دریافت ورودی کاربر از بروز خطای احتمالی پیشگیری می‌کند. برای مثال به جای استفاده از TextBox برای دریافت تاریخ می‌توانید از DateTimePicker استفاده کنید.

### اعتبارسنجی فرم

امکان تعیین عملیات اعتبارسنجی کنترل‌های فرم در رخداد کلیک یکی از دکمه‌های فرم وجود دارد. اعتبارسنجی کنترل‌های فرم به صورت همزمان شما را از نوشتن کد برای رخدادهای تک‌تک کنترل‌های فرم بی‌نیاز می‌کند. فرض کنید یک فرم دارای دکمه‌ای به نام Update و دو TextBox باشد، قطعه کد زیر را در رخداد Click دکمه‌ی Update جهت کنترل طول داده و فرمت آن درج نمایید.

```
if (TextBox1.Text.Length < 6) {
    MessageBox.Show("Please specify a value to TextBox2");
    TextBox2.Focus();
    return ;
}
if (TextBox2.Text.Length < 1 || Convert.ToInt32(TextBox2.Text) < 1) {
    MessageBox.Show("Please specify a valid value to TextBox2");
    TextBox2.Focus();
    return ;
}
```

### استفاده از ErrorProvider

به جای نمایش messageBox به ازای ورود هر داده‌ی ناقص یا نادرست توسط کاربر می‌توانید از ErrorProvider استفاده نمایید. برای افزودن ErrorProvider به فرم به صورت زیر عمل می‌کنیم.

۱. فرم را در حالت Design View باز کنید.
۲. کنترل ErrorProvider را در پنجره Toolbox انتخاب کرده و آن را به فرم اضافه کنید.
۳. مقدار خصیصه ی Name کنترل ErrorProvider را تعیین کنید.

پس از انجام مراحل فوق باید متن پیغام خطای کنترل‌های فرم را تعیین نمایید. به عنوان مثال فرض کنید فرمی دارای یک دکمه به نام Save باشد. قطعه کد مربوط به دکمه Save این فرم به صورت زیر است.

```
bool flag;
flag = true;
if (textBox1.Text=="") {
    errorMessage.SetError(textBox1,"Please specify a valid car number.");
    flag = false;
}
else
    errorMessage.SetError(textBox1,"");
if (textBox2.Text=="") {
    errorMessage.SetError(textBox2,"Please specify a valid name.");
    flag = false;
}
else
    errorMessage.SetError(textBox2,"");
if (textBox3.Text=="") {
    errorMessage.SetError(textBox3,"Please specify a valid address.");
    flag = false;
}
else
    errorMessage.SetError(textBox3,"");
if (textBox4.Text=="") {
    errorMessage.SetError(textBox4,"Please specify a valid make.");
    flag = false;
}
else
    errorMessage.SetError(textBox4,"");
if (flag==false)
    return;
```

همان‌طور که ملاحظه می‌کنید در این مثال از یک متغیر به نام flag جهت تعیین تهي بودن فیلدها استفاده شده‌است. هرگاه مقدار یکی از فیلدها تهي باشد، مقدار متغیر flag برابر true شده و پیغام خطا برای کنترل errorMessage تعیین می‌شود و هرگاه فیلد مقداردهی شود، مقدار متغیر flag برابر false شده و پیغام خطا پاک می‌شود.

### مدیریت استثناء<sup>۱</sup>

C# مانند اغلب زبان‌های برنامه‌نویسی شیء‌گرای دیگر از Exceptionها جهت مدیریت شرایط غیرعادی استفاده می‌کند. یک Exception شیء است که اطلاعات مربوط به شرایط غیرعادی را در خود نگه می‌دارد.

درک تفاوت bug، error و exception بسیار مهم است. هر آنچه مربوط به اشتباه برنامه‌نویس هنگام نوشتن کد می‌باشد، bug نام دارد. اینگونه خطاها باید در اسرع وقت و حداقل امکان قبل از فروش نرم افزار رفع گردند. با وجود اینکه یک bug ممکن است باعث بروز Exception شود ولی برای مدیریت مشکل به وجود آمده از Exceptionها استفاده نکرده و مشکل را بر طرف می‌کنیم.

1- Exception

دسته دوم خطاها یا همان error مربوط به کاربر برنامه‌ی کاربردی است. برای مثال در صورتی که کاربر مقدار یک فیلد اجباری را درج نکند و دکمه‌ی ثبت را کلیک کند، Exception به وقوع خواهد پیوست. برای رفع اینگونه خطاها نیز از Exception استفاده نکرده و آنها را با استفاده از عملیات اعتبارسنجی مدیریت می‌کنیم.

دسته سوم خطاهای هستند که برنامه‌نویس کنترلی روی آنها ندارد. به عنوان مثال می‌توان خطای تقسیم بر صفر، out of memory یا اجرای دستور بازکردن فایلی که وجود ندارد، را نام برد.

هنگامی که یک Exception به وقوع می‌پیوندد اجرای برنامه متوقف می‌شود و در صورتی که قطعه کد لازم برای مدیریت وجود نداشته باشد در همان وضعیت باقی خواهد ماند.

### عبارات try, catch و finally

یکی از روش‌های مدیریت Exception استفاده از عبارات try, catch و finally می‌باشد. در این روش قطعه کدی را باعث بروز Exception می‌شود، را درون بلوک try قرار می‌دهیم. به عنوان مثال هرگاه قطعه کدی جهت درج اطلاعات در بانک اطلاعاتی یا تبدیل فرمت داده می‌نویسید، احتمال بروز Exception وجود دارد. بنابراین لازم است قطعه کد مذکور را در بلوک try قرار دهید.

هرگاه عبارات درون بلوک try با Exception مواجه شوند، عبارت درون بلوک catch مدیریت Exception را انجام خواهد داد. مدیریت Exception توسط catch منوط به استفاده از کلاس exception مناسب می‌باشد. برای مثال اگر به جای استفاده از یک متغیر با نوع داده‌ای int از متغیری با نوع داده‌ای String استفاده کنید، Exception به وقوع پیوسته از نوعFormatException می‌باشد. بنابراین اگر catch از کلاس FormatException جهت مدیریت Exception استفاده کرده باشد، عبارت داخل بلوک catch اجرا خواهد شد.

عبارات درون بلوک finally در هر صورت اجرا می‌شوند. به عبارت دیگر در صورتی که بخواهید قسمتی از برنامه در صورت وقوع و عدم وقوع Exception اجرا شود آن را داخل بلوک finally قرار می‌دهیم. برای مثال در صورتی که برنامه‌ی کاربردی از منابع خارجی نظیر فایل‌ها استفاده می‌کند، قطعه کد مربوط به بستن فایل را درون بلوک finally قرار می‌دهیم تا در هر دو صورت اجرا شود.

ذکر چند نکته در مورد Exception‌ها لازم به نظر می‌رسد.

۱. امکان تعریف چند بلوک catch برای یک try وجود دارد.

۲. هر try فقط دارای یک finally است.

۳. در صورت وجود حداقل یک catch ذکر finally الزامی نمی‌باشد و در غیر این صورت وجود آن الزامی است.

لازم به ذکر است که در صورتی که نیازی به ایجاد روش‌های متفاوت برای مدیریت Exception‌های متفاوت ندارید، می‌توانید از کلاس Exception در بلوک catch استفاده کنید.

ساختار try ... catch به صورت زیر است.

```
try {
    //The statements that might generate an error
    Statement(s);
}
catch (filter) {
    //The statements written here are executed when the statements listed in
    //the Try block fail and the filter specified is true.
    Statement(s);
}
catch (filter) {
    //The statements written here are always execute
    Statement(s);
}
```

قطعه کد مدیریت رخداد کلیک دکمه Update پس از افزودن قطعه کد مدیریت Exception به صورت زیر در خواهد آمد.

```
private void btnUpdate_Click(object sender, System.EventArgs e) {
    if (txtCarNo.Text.Length < 6) {
        MessageBox.Show("Please specify a valid car Number");
        txtCarNo.Focus();
        return;
    }
    try {
        if (Convert.ToInt32(txtWorkerId.Text) < 1) {
            MessageBox.Show("Please specify a valid worker ID");
            txtWorkerId.Focus();
            return;
        }
        if (Convert.ToDateTime(dateTimePicker1.Value) > DateTime.Today) {
            MessageBox.Show("Please specify a valid date");
            dateTimePicker1.Focus();
            return;
        }
    }
    catch (Exception exception) {
        MessageBox.Show(exception.Message);
    }
}
```

در متد فوق از عبارات تبدیل مقادیر فیلدهای txtWorkerId و dateTimePicker1 از فرمت String به فرمت‌های int و DateTime را داخل بلوک try قرار دادیم. بنابراین در صورتی که عمل تبدیل نوع با مشکل مواجه شود، قطعه کد بلوک catch اجرا شده و دلیل بروز Exception به کاربر نمایش داده می‌شود.

**تمرین:** کاربرد و نحوه استفاده از کلاسهای Debug و Trace را به صورت عملی تشریح نمایید.

**تمرین:** برنامه ای بنویسید که دو عدد را از ورودی خوانده و خارج قسمت تقسیم عدد اول بر دوم را محاسبه کند. برنامه باید به صورتی باشد که اگر عدد دوم برابر صفر باشد، خللی در روند اجرایی برنامه به وجود نیاید.

**تمرین:** اعتبار سنجی و Exception Handling فرمهای فوق را با توجه به اجباری/اختیاری بودن فیلدها و نوع داده‌ای آنها تکمیل کنید.

## بانک اطلاعاتی

قدم اول در جهت توسعه پروژه طراحی یک بانک اطلاعاتی مناسب می‌باشد. بانک اطلاعاتی محلی جهت ذخیره و بازیابی اطلاعات می‌باشد. جهت ذخیره و بازیابی اطلاعات در بانک اطلاعاتی از زبان SQL استفاده می‌شود.

## عبارات SQL

SQL زبان استاندارد جهت برقراری ارتباط با بانک اطلاعاتی می‌باشد. زبان SQL جهت واکشی و به‌هنگام‌سازی داده‌های موجود در بانک اطلاعاتی کاربرد دارد. بانک اطلاعاتی از جداول تشکیل می‌شود. جداول ساختارهایی هستند که اطلاعات را در قالب سطر و ستون نگهداری می‌کنند.

| City     | Sales | Date        |
|----------|-------|-------------|
| New York | 23600 | Jul-14-2002 |
| Atlanta  | 16400 | Jul-12-2002 |
| Seattle  | 17300 | Jul-11-2002 |
| Chicago  | 19700 | Jul-14-2002 |



|               |       |             |
|---------------|-------|-------------|
| San Francisco | 24200 | Jul-14-2002 |
|---------------|-------|-------------|

این جدول دارای سه ستون City، Sales و Date می‌باشد. سطرهای این جدول شامل پنج رکورد می‌باشد که حاوی اطلاعات مربوط به میزان فروش ۵ شهر در یک تاریخ خاص می‌باشد.

### عبارت Select

عبارت Select جهت واکنشی داده‌های موجود در بانک اطلاعاتی مورد استفاده قرار می‌گیرد. ساده‌ترین حالت عبارت Select به صورت زیر است.

Select \* from Sales

عبارت فوق تمام رکوردهای موجود در جدول Sales را باز می‌گرداند. ساختار کلی عبارت Select به صورت زیر است.

Select [select-list] from [table name]

علاوه بر این امکان تعیین شرط برای محدود کردن رکوردهایی که توسط عبارت Select بازگردانده می‌شود، وجود دارد. برای مثال فرض کنید که فقط رکوردهایی که فیلد تاریخ آنها Jul-14-2002 می‌باشد، مدنظر است. در این صورت با عبارت Select را به صورت زیر بنویسیم.

Select city, sales from Sales where date = 'jul-14-2002'

در این حالت ساختار عبارت Select به صورت زیر خواهد بود.

Select [select-list] from [table name] where [search condition]

عملگرهایی که در قسمت where قابل استفاده هستند به شرح زیر می‌باشد.

| عملگر   | توصیف                                          |
|---------|------------------------------------------------|
| =       | تساوی                                          |
| <       | کوچکتر از                                      |
| >       | بزرگتر از                                      |
| <=      | کوچکتر مساوی                                   |
| >=      | بزرگتر مساوی                                   |
| <>      | نا مساوی                                       |
| Between | داده‌هایی که داخل مجموعه‌ی تعیین شده می‌گنجند. |
| Like    | داده‌هایی که با الگوی تعیین شده مطابقت دارند.  |

برای درک بهتر مطلب مثال‌های زیر را در نظر بگیرید.

| City    | DOB         | EmailAddress          | SurName | FirstName |
|---------|-------------|-----------------------|---------|-----------|
| Atlanta | Jan-04-1971 | slewis@aol.com        | Lewis   | Sandra    |
| Chicago | Oct-27-1979 | ethorn@yahoo.com      | Thorn   | Elaine    |
| Atlanta | Aug-25-1976 | gthomas@freemail.com  | Thomas  | George    |
| Memphis | Mar-18-1978 | swatson@fastmail.com  | Watson  | Simon     |
| Atlanta | Jun-12-1981 | lgates@mymail.com     | Gates   | Larry     |
| Memphis | Feb-02-1972 | mbrown@aol.com        | Brown   | Michael   |
| Chicago | Oct-04-1982 | sjudd@zipmail.com     | Judd    | Sarah     |
| Detroit | Apr-24-1977 | jjohnson@slowmail.com | Johnson | Joshua    |
| Chicago | Dec-07-1975 | dallison@aol.com      | Allison | Daniel    |
| Detroit | Mar-13-1979 | nharvey@buzz.com      | Harvey  | Nicholas  |
| Memphis | Sep-12-1973 | lhansen@hotmail.com   | Hansen  | Laura     |

**مثال ۱:** لیست دانشجویانی را که در شهر Chicago زندگی می‌کنند، مدنظر است. به این منظور از عبارت Select به صورت زیر استفاده می‌کنیم.

Select \* from Students where City = 'Chicago'

نتیجه‌ی اجرای عبارت Select فوق به صورت زیر خواهد بود.

| City    | DOB         | EmailAddress     | SurName | FirstName |
|---------|-------------|------------------|---------|-----------|
| Chicago | Oct-27-1979 | ethorn@yahoo.com | Thorn   | Elaine    |

|        |         |                   |             |         |
|--------|---------|-------------------|-------------|---------|
| Sarah  | Judd    | sjudd@zipmail.com | Oct-04-1982 | Chicago |
| Daniel | Allison | dallison@aol.com  | Dec-07-1975 | Chicago |

**مثال ۲:** لیست دانشجویانی را که نام خانوادگی آنها با حرف n خاتمه می‌یابد، مدنظر است. به این منظور از عبارت Select به صورت زیر استفاده می‌کنیم.

Select \* from Students where SurName like '%n'

نتیجه‌ی اجرای عبارت Select فوق به صورت زیر خواهد بود.

| FirstName | SurName | EmailAddress          | DOB         | City    |
|-----------|---------|-----------------------|-------------|---------|
| Elaine    | Thorn   | ethorn@yahoo.com      | Oct-27-1979 | Chicago |
| Simon     | Watson  | swatson@fastmail.com  | Mar-18-1978 | Memphis |
| Michael   | Brown   | mbrown@aol.com        | Feb-02-1972 | Memphis |
| Joshua    | Johnson | jjohnson@slowmail.com | Apr-24-1977 | Detroit |
| Daniel    | Allison | dallison@aol.com      | Dec-07-1975 | Chicago |
| Laura     | Hansen  | lhansen@hotmail.com   | Sep-12-1973 | Memphis |

**مثال ۳:** لیست دانشجویانی را که نام آنها از نظر ترتیب حروف الفبا بین Jashua و Michael می‌باشد، مدنظر است. به این منظور از عبارت Select به صورت زیر استفاده می‌کنیم.

Select \* from Students where FirstName Between 'Jashua' and 'Michael'

نتیجه‌ی اجرای عبارت Select فوق به صورت زیر خواهد بود.

| FirstName | SurName | EmailAddress          | DOB         | City    |
|-----------|---------|-----------------------|-------------|---------|
| Joshua    | Johnson | jjohnson@slowmail.com | Apr-24-1977 | Detroit |
| Larry     | Gates   | lgates@mymail.com     | Jun-12-1981 | Atlanta |
| Laura     | Hansen  | lhansen@hotmail.com   | Sep-12-1973 | Memphis |
| Michael   | Brown   | mbrown@aol.com        | Feb-02-1972 | Memphis |

**نکته:** امکان تعیین ستون‌های مورد نظر در عبارت Select وجود دارد.

**مثال ۴:** عبارت Select ی بنویسید که فقط ستون‌های نام، نام خانوادگی و شهر را بازگرداند.

Select FirstName, SurName, City from Students

نتیجه‌ی اجرای عبارت Select فوق به صورت زیر خواهد بود.

| FirstName | SurName | City    |
|-----------|---------|---------|
| Sandra    | Lewis   | Atlanta |
| Elaine    | Thorn   | Chicago |
| George    | Thomas  | Atlanta |
| Simon     | Watson  | Memphis |
| Larry     | Gates   | Atlanta |
| Michael   | Brown   | Memphis |
| Sarah     | Judd    | Chicago |
| Joshua    | Johnson | Detroit |
| Daniel    | Allison | Chicago |
| Nicholas  | Harvey  | Detroit |
| Laura     | Hansen  | Memphis |

## عبارت Insert

عبارت Insert جهت درج یک سطر (رکورد) جدید در جدول کاربرد دارد. برای مثال جهت درج یک رکورد جدید در جدول Students از عبارت Insert به صورت زیر استفاده می‌کنیم.

Insert into Students values ('Sarah', 'Lee', 'slee@yahoo.com', 'Mar-22-1977', 'Detroit')

لازم به ذکر است که امکان درج داده در ستون‌های مورد نظر نیز وجود دارد. به این منظور به صورت زیر عمل می‌کنیم.

Insert into students (FirstName, SurName) values ('Jessica', 'Parker')

## عبارت Update

عبارت Update جهت تغییر داده‌های یک جدول استفاده می‌شود. برای مثال فرض کنید Laura Hasen نام خانوادگی‌اش را تغییر داده و نام خانوادگی Brown را انتخاب کرده‌است. برای تغییر نام خانوادگی Laura از عبارت زیر استفاده می‌کنیم.

Update Students set SurName = 'Brown' where FirstName = 'Laura' and SurName = 'Hansen'

### عبارت Delete

عبارت Delete جهت یک تعداد یا تمام رکوردهای یک جدول استفاده می‌شود. برای مثال جهت حذف اطلاعات دانش‌آموزانی که در شهر Detroit زندگی می‌کنند، از عبارت Delete به صورت زیر استفاده می‌شود.

Delete from Students where City = 'Detroit'

### تعامل با بانک اطلاعاتی از طریق ADO.NET

اغلب برنامه‌های کاربردی نیازمند برقراری ارتباط با بانک اطلاعاتی می‌باشند. برای این منظور در Visual Studio.Net می‌توانید از ADO.NET استفاده کنید. امکان استفاده از مدل دسترسی داده ADO.NET جهت برقراری ارتباط با بانک‌های اطلاعاتی نظیر SQL Server یا Oracle وجود دارد.

ADO.NET اساساً جهت دسترسی به داده‌های موجود در بانک اطلاعاتی در برنامه‌های کاربردی توزیع شده<sup>1</sup> نظیر برنامه‌های کاربردی وب طراحی شده‌است. علاوه بر دسترسی به داده‌های موجود در بانک اطلاعاتی توسط ADO.NET امکان درج اطلاعات، حذف و به هنگام سازی داده‌های موجود در بانک نیز با استفاده از ADO.NET وجود دارد.

جهت برقراری ارتباط با بانک اطلاعاتی ابتدا باید شیئی از کلاس Connection ایجاد کرده و خصایص آن را مقداردهی کنیم. برای این منظور از قطعه کد زیر استفاده می‌کنیم.

```
SqlConnection connection = new SqlConnection();
connection.ConnectionString = "";
connection.Open();
```

لازم به ذکر است که مقدار خصیصه connectionString بستگی به بانک اطلاعاتی موردنظر دارد. پس از ایجاد Connection نوبت ایجاد شیء Command می‌باشد. از این شیء جهت برقراری اجرای دستورات SQL استفاده می‌شود. برای ایجاد شیء Command یکی از قطعه کدهای زیر را استفاده می‌کنیم.

1. `SqlCommand command = new SqlCommand();`  
`command.Connection = connection;`
2. `SqlCommand command = connection.CreateCommand();`

پس ایجاد شیء command نوبت اجرا دستورات SQL فرا می‌رسد. برای این منظور ابتدا باید خصیصه commandText را به صورت زیر مقداردهی نماییم.

```
command.CommandText = "select * from table";
```

لازم به ذکر است که هر چهار دستور select، insert، update و delete را به خصیصه commandText نسبت داده و در صورتی که دستور موردنظر select باشد از متد ExecuteReader() به صورت زیر استفاده می‌کنیم.

```
SqlDataReader reader = command.ExecuteReader();
```

روش دیگر استفاده از قطعه کد زیر است. توجه داشته باشید که این روش نسبت به روش اول دارای مزایایی می‌باشد. بررسی این مزایا به عهده‌ی دانشجویان می‌باشد.

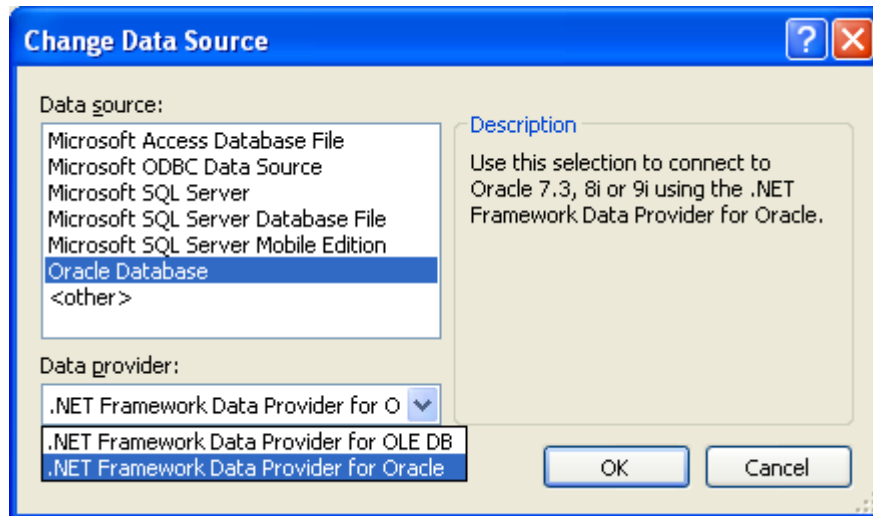
```
SqlDataAdapter adapter = new SqlDataAdapter(command);
DataSet dataSet = new DataSet();
adapter.Fill(dataSet);
```

در صورتی که دستور مورد نظر select نباشد از قطعه کد زیر استفاده می‌کنیم.

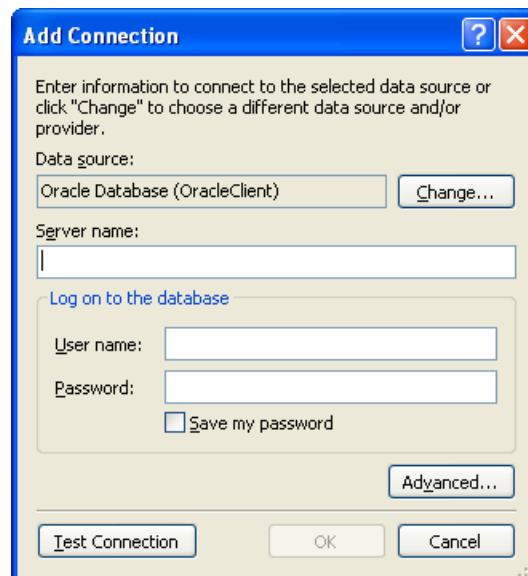
```
command.ExecuteNonQuery();
```

روش فوق بدون استفاده از Wizard می‌باشد. جهت برقراری ارتباط با بانک اطلاعاتی امکان استفاده از wizard نیز وجود دارد. مراحل برقراری ارتباط به شرح زیر است.

۱. گزینه Connect to Database را از منوی Tools انتخاب نمایید تا پنجره زیر نمایش داده شود.



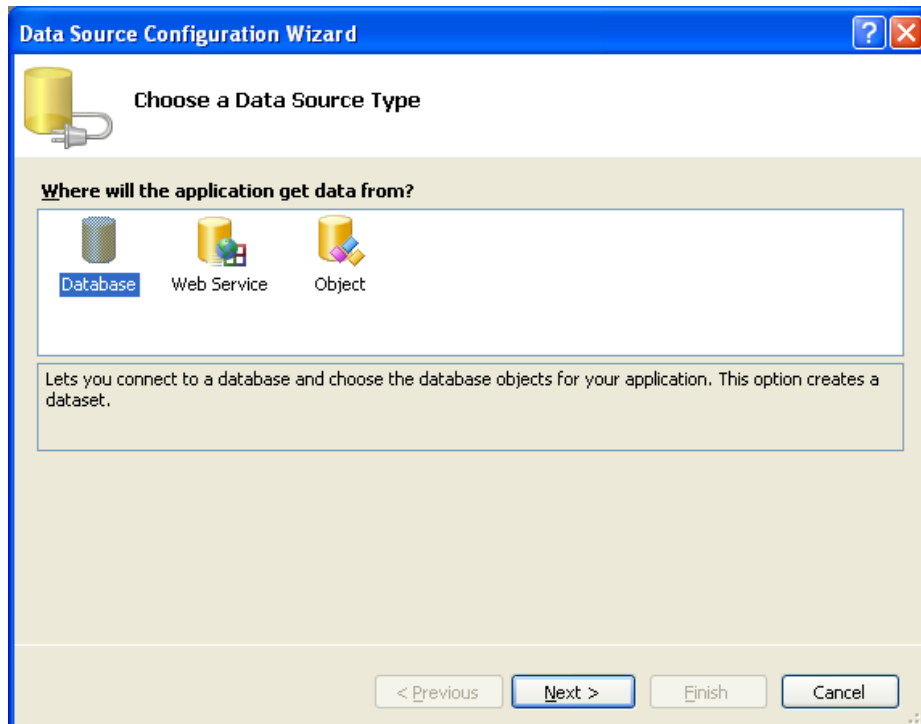
۲. در پنجره فوق نام برنامه‌ی کاربردی سرویس‌دهنده‌ی بانک اطلاعاتی را تعیین کرده و دکمه OK را کلیک کنید تا پنجره زیر نمایش داده شود.



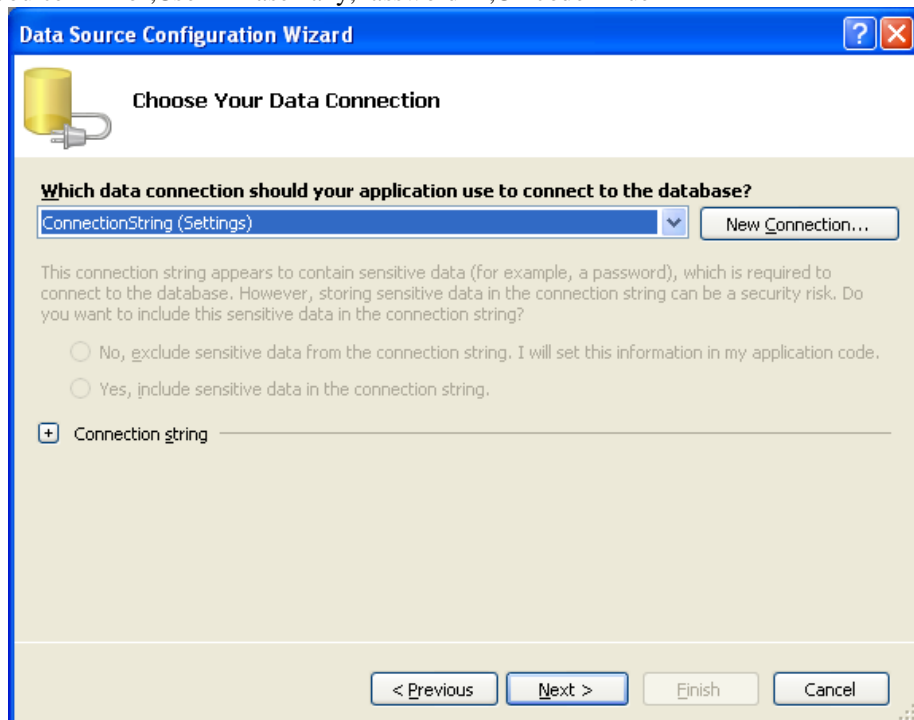
مقدار فیلد Server name نامی است که برای TNS Listener بانک اطلاعاتی Oracle تعیین کرده‌اید. پس از درج مقادیر لازم دکمه Test Connection را کلیک کنید تا از صحت برقراری ارتباط با بانک اطلاعاتی اطمینان حاصل کنید. سپس دکمه OK را کلیک کنید تا این پنجره بسته شود.

لازم به ذکر است در صورتی که تنظیمات این مرحله با توجه به بانک اطلاعاتی مورد استفاده متفاوت خواهد بود.

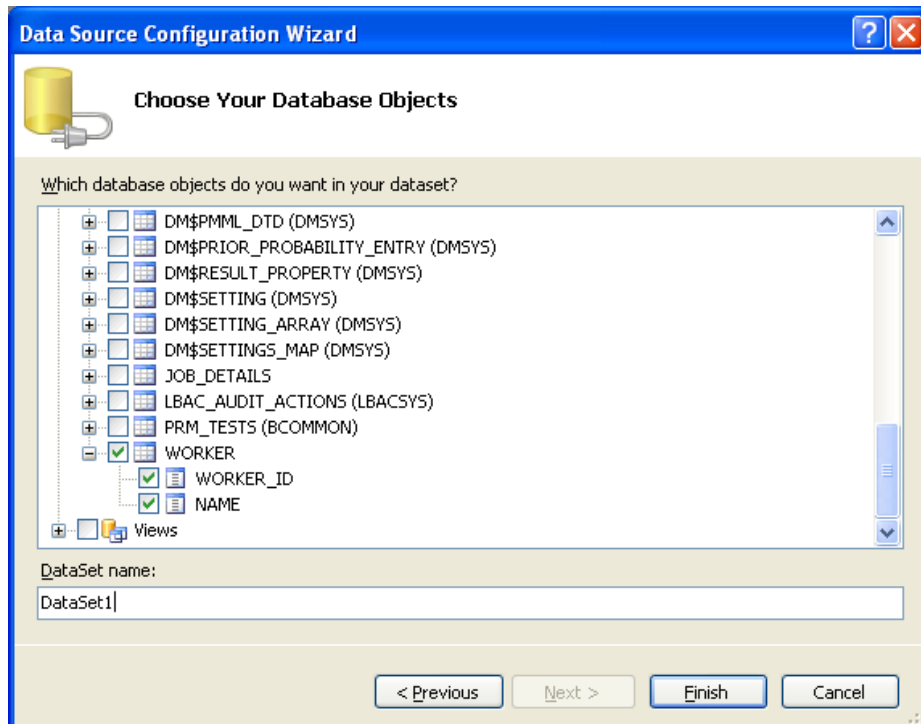
۳. گزینه Add New Data Source را از منوی Data انتخاب کنید تا پنجره زیر نمایش داده شود. گزینه Database را انتخاب کرده و دکمه Next را کلیک کنید.



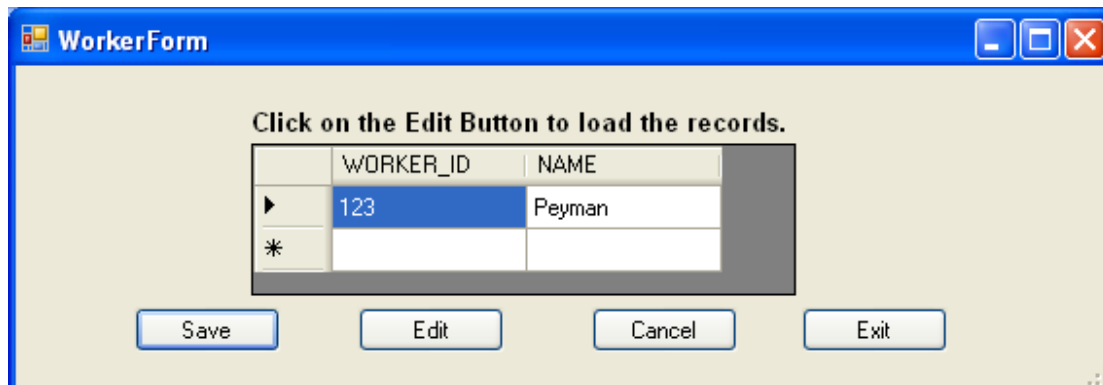
۴. با توجه به اینکه در مراحل قبلی ارتباط با بانک اطلاعاتی برقرار شده‌است، نیازی به ایجاد Connection وجود ندارد. در صورتی که نماد + کنار عبارت Connection String را کلیک نمایید، عبارت زیر مشاهده خواهد شد.  
Data Source=mirror;User ID=asemany;Password=1;Unicode=True



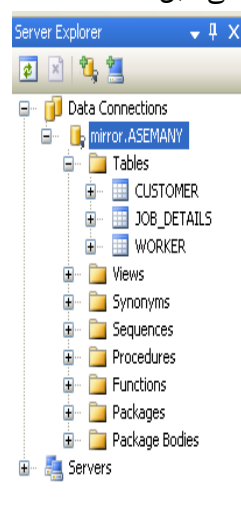
۵. در این مرحله امکان انتخاب جدول یا View مورد نظر وجود دارد. همان‌طور که ملاحظه می‌کنید جدول Worker انتخاب شده‌است. دکمه Finish را کلیک نمایید تا عملیات خاتمه یابد.



۶. با توجه به فیلدهای انتخاب شده از جدول DataGrid View به صورت زیر ایجاد خواهد شد.



۷. از منوی View گزینه Server Explorer را انتخاب نمایید تا پنجره‌ی زیر نمایش داده شود. همان‌طور که ملاحظه می‌کنید لیست تمام اشیاء بانک اطلاعاتی قابل مشاهده است.



ایجاد عملیات دکمه Show

در این بخش متد رخداد کلیک دکمه Edit را به صورتی ایجاد می‌کنیم که رکوردهای جدول را خوانده و آن‌ها را با استفاده از DataGridView نمایش دهد.

```
private void btnEdit_Click(object sender, EventArgs e) {
    personDataSet.Clear();
    personTableAdapter.Fill(personDataSet.PERSON);
}
```

متد Clear() کلاس DataSet داده‌های موجود در شیء personDataSet را حذف می‌کند و سپس شیء personDataSet با استفاده از متد Fill() مجدداً مقداردهی می‌شود. هنگامی که کاربر دکمه Edit را کلیک می‌کند، رکوردهای جدول Person درون DataGridView بارگذاری می‌شود.

### ایجاد عملیات دکمه Update

هنگامی که داده‌های جدید را در DataGridView درج می‌کنید، رکوردهای جدید فقط در DataSet ذخیره می‌شود. برای اعمال تغییرات در بانک اطلاعاتی باید متد رخداد کلیک دکمه Update را به صورت زیر تغییر دهید.

```
private void btnSave_Click(object sender, EventArgs e) {
    personTableAdapter.Update(personDataSet.PERSON);
    MessageBox.Show("The Person table is updated.");
}
```

همان‌طور که ملاحظه می‌کنید به منظور به‌هنگام‌سازی اطلاعات بانک اطلاعاتی از متد Update استفاده شده‌است. این متد رکوردهایی را که به DataGridView اضافه شده‌اند، را در جدول person درج می‌کند.

### ایجاد عملیات دکمه Cancel

این متد داده‌های DataSet را مجدداً بارگذاری می‌کند.

```
private void btnCancel_Click(object sender, EventArgs e) {
    workerDataSet.Clear();
    WORKERTableAdapter.Fill(workerDataSet.WORKER);
}
```

### ایجاد عملیات دکمه Exit

هرگاه دیگر نیازی به فرم WorkerForm نداشته باشید، لازم است فرم مربوطه را ببندید تا به فرم اصلی برنامه دسترسی پیدا کنید.

```
private void btnExit_Click(object sender, System.EventArgs e) {
    this.Close();
}
```