فرادرس
فراتر از یک کلاس درس
www.faradars.org

آموزش جامع شیئ گرایی در سی شارپ

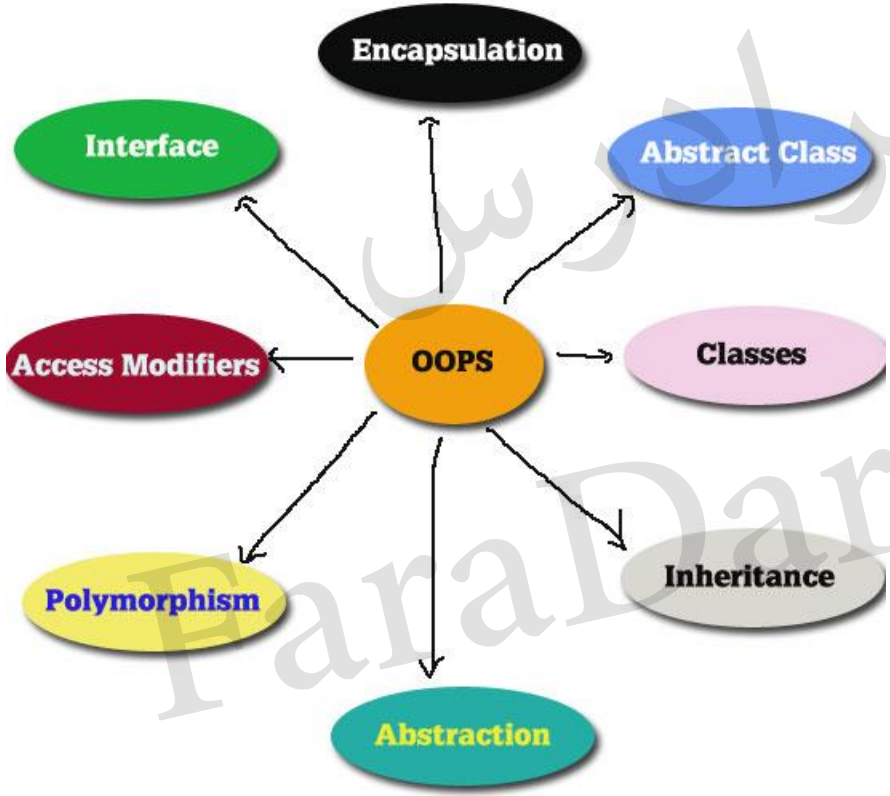
مدرس:

رشید شجاعی

کارشناس ارشد کامپیوتر گرایش نرم افزار

فرادرس
FaraDars.org

OOP: Object Oriented Programming



مفاهیم برنامه نویسی شی گرا:

- کلاس‌ها و اشیاء
- تعیین کننده‌های دسترسی
- کپسوله سازی
- وراثت
- واسط
- چندریختی
- خلاصه سازی و تجرد
- کلاس‌های انتزاعی

فرادرس
FaraDars.org

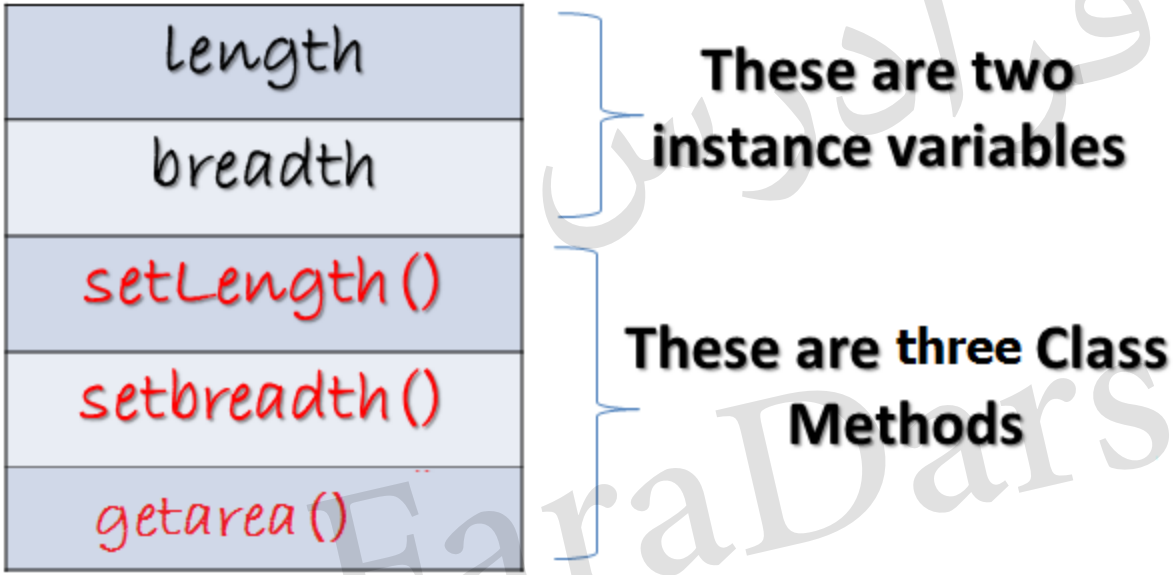
OOAD & UML

برای پروژه‌های بزرگ و پیچیده نمی‌توانید به راحتی پشت کامپیوتر نشسته و شروع به برنامه‌نویسی کنید.
برای دستیابی به بهترین راه حل، بایستی بدقت نیازمندی‌های پروژه را تجزیه و تحلیل کنید(تعیین کنید که سیستم چه کار می‌خواهد انجام دهد) و طرحی را توسعه دهید که آنها را برآورده سازد (سیستم قادر به تصمیم‌گیری صحیح برای انجام وظایف خود باشد). در حالت ایده‌آل قبل از هرگونه کدنویسی، باید به سراغ فرآیند رفته و به دقت طراحی را انجام دهید. اگر این فرآیند مستلزم تحلیل و طراحی سیستم از نقطه نظر شی‌گرا باشد، آن را ***object-orientend analysis and design (OOAD)*** یا ***تحلیل و طراحی شی‌گرا*** می‌گویند. برنامه‌نویسان با تجربه مطلع هستند که تحلیل و طراحی می‌تواند در زمان و هزینه ایجاد برنامه بسیار صرفه‌جویی کند، با اجتناب از اعمال طرح‌های ضعیف که در هر بار برنامه را به شکست کشانده و باعث می‌شوند کار از ابتدا آغاز گردد که همان تحمیل هزینه و زمان است. اگر چه پردازش‌های مختلفی از OOAD وجود دارد، اما یک زبان گرافیکی برای نمایش نتایج هر فرآیند OOAD بیشتر از همه بکار گرفته شده است. این زبان *UML (Unified Modeling Language)* نام دارد. در حال حاضر زبان UML یکی از پرکاربردترین طرح‌های نمایش گرافیکی برای مدل کردن سیستم‌های شی‌گرا است. برای مدل کردن سیستم‌ها از این زبان در سراسر مجموعه آموزشی پیش رو استفاده کرده‌ایم.

فرادرس
FaraDars.Org

مثال: ایجاد یک کلاس برای مستطیل

فرض کنید خواهیم یک کلاس برای اشیایی از نوع مستطیل ایجاد کنیم. برای یک مستطیل می‌توانیم اجزاء داده‌ای همانند طول و عرض در نظر بگیریم. همچنین می‌توانیم متدها یا همان اجزاء تابعی همانند تنظیم کردن عرض و طول و همچنین محاسبه محیط در نظر بگیریم.



فرادرس
FaraDars.org

شکل کلی تعریف کلاس

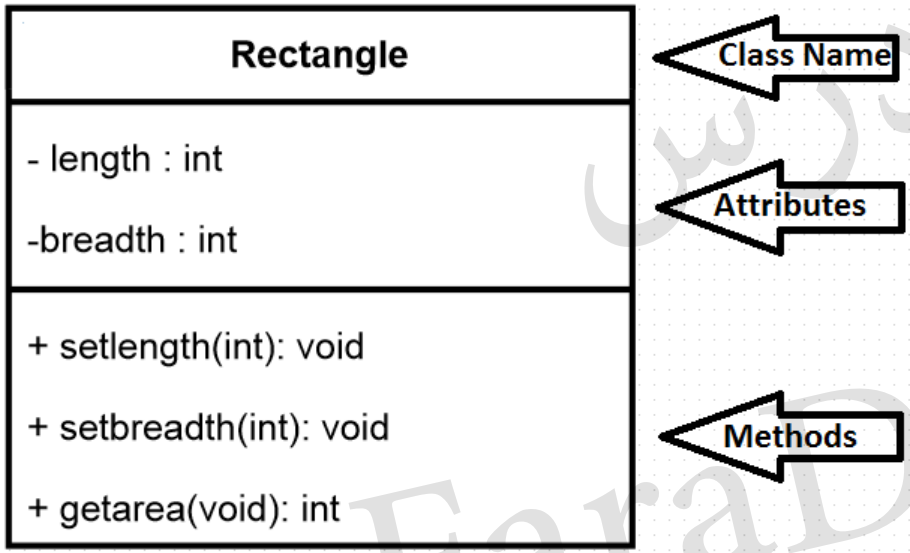
برای ایجاد کلاس در C# از واژه کلیدی class بصورت مقابل استفاده می‌شود:

```
[modifier] class name
{
Classmembers
}
```

- **Modifier**: سطح دستیابی به کلاس را مشخص می‌کند. سطح دستیابی مشخص می‌کند که کلاس تعریف شده چگونه باید دستیابی شود. در اینجا دو سطح دستیابی برای کلاس داریم: public و internal وقتی سطح دستیابی public تعریف شود به معنای این است که این کلاس خارج از فضای نامی که در آن تعریف می‌شود قابل استفاده است. بعبارت دیگر، سطح دستیابی عمومی به معنای عدم وجود محدودیتی در دستیابی است. سطح دستیابی internal مشخص می‌کند که کلاس فقط در همان فضای نامی که تعریف می‌شود قابل استفاده است. Modifier بصورت پیش‌فرض internal تعریف می‌شود.
- **class**: واژه‌ای کلیدی است که به همین صورت برای تعریف کلاس به کار می‌رود.
- **name**: نامی است که توسط برنامه‌نویس برای کلاس انتخاب می‌گردد. نام‌گذاری برای کلاس، از نامگذاری برای شناسه‌ها پیروی می‌کند.
- **classmember**: اعضای کلاس را مشخص می‌کند و شامل اعضاء داده‌ای و اجزاء تابعی کلاس می‌باشند.

فرادرس
FaraDars.org

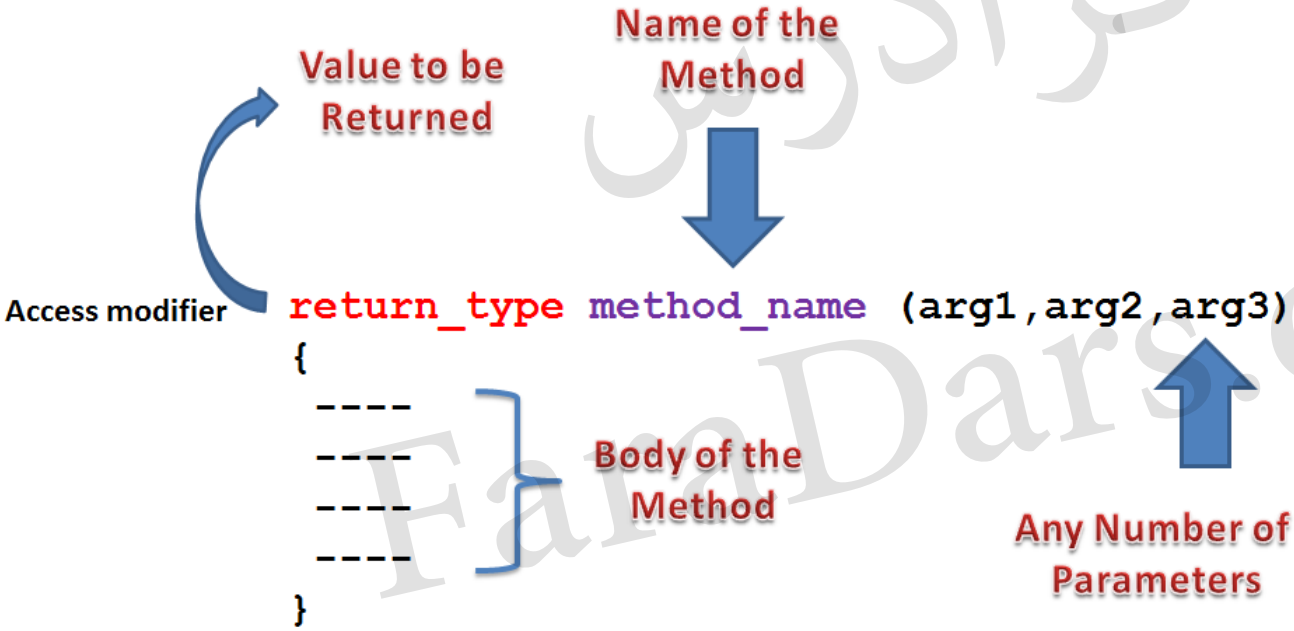
Class Diagram نمودار کلاس برای مستطیل



از مطالب قبل بخاطر دارید که UML یک زبان گرافیکی بکار رفته توسط برنامه‌نویس برای نمایش سیستم‌های شی‌گرا به یک روش استاندارد است. هر کلاس در یک نمودار کلاس بصورت یک مستطیل و با سه قسمت مدل‌سازی می‌شود. بخش فوقانی حاوی نام کلاس است که در وسط قرار گرفته و بصورت توپر نوشته می‌شود. بخش میانی حاوی صفات کلاس است و بخش تحتانی حاوی عملیات کلاس است. دیاگرام مقابل مبادرت به مدل‌سازی داده عضو length و breadth بعنوان اجزاء داده‌ای در بخش میانی کلاس کرده است. UML اعضای داده را در لیستی که در آن نام صفت، یک کولن و نوع صفت قرار گرفته عرضه می‌کند. نوع صفات length و breadth از نوع int است که متناظر با int در C# می‌باشد. اعضاء داده‌ای length و breadth در C# حالت private دارند و از اینرو در دیاگرام کلاس با یک علامت (-) در مقابل نام صفت مشخص شده است. علامت منفی در UML معادل با تصریح‌کننده دسترسی private در C# است. کلاس Rectangle حاوی سه تابع عضو public است، از اینرو در لیست دیاگرام کلاس این سه عملیات در بخش تحتانی یا سوم جای گرفته‌اند. نماد جمع (+) قبل نام هر عملیات نشان می‌دهد که عملیات در C# حالت public دارد. عملیات setlength دارای یک پارامتر از نوع int است. در UML نوع برگشتی از یک عملیات با قرار دادن یک کولن و نوع برگشتی پس از پرانتزهای نام عملیات مشخص می‌شود.

FaraDars.org

شکل کلی ایجاد کردن اجزاء تابعی یا متد های کلاس



فرادرس
FaraDars.org

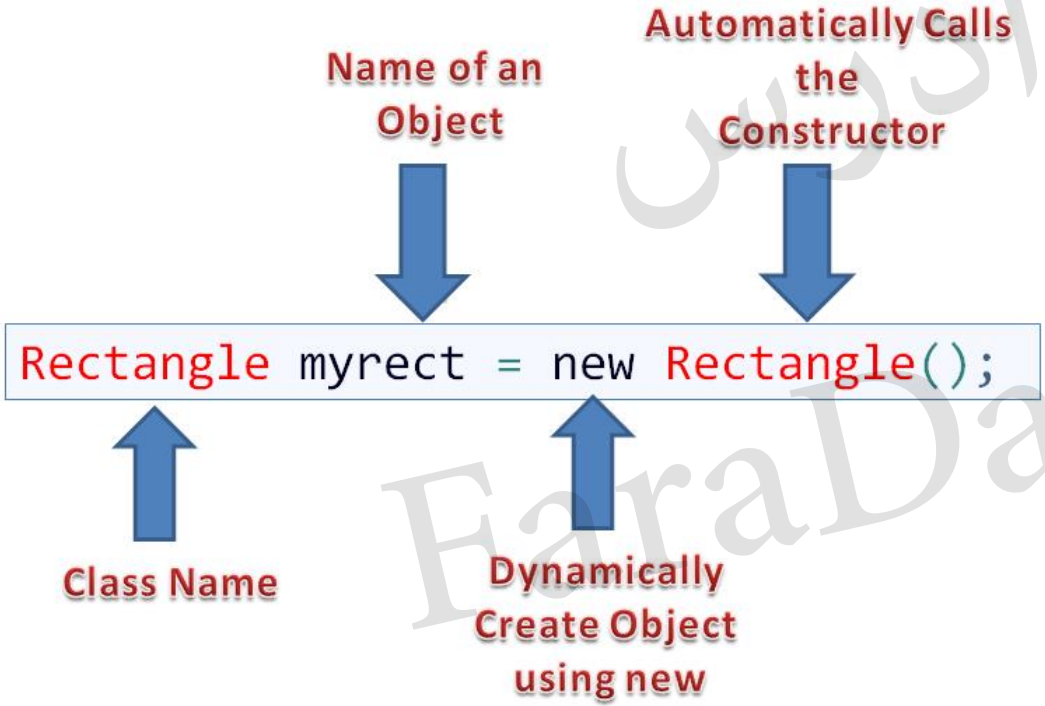
کلاس مستطیل

```
class Rectangle
{
    // two fields
    private int breadth;
    private int length;

    // three methods
    public void setLength(int newValue)
    {
        length = newValue;
    }
    public void setBreadth(int newValue)
    {
        breadth = newValue;
    }
    public int getarea()
    {
        return 2 * (length + breadth);
    }
```

فرادرس
FaraDars.org

شکل کلی ایجاد کردن یک شی از روی کلاس

در شکل مقابل شی بنام myrect از روی کلاس Rectangle ایجاد شده.

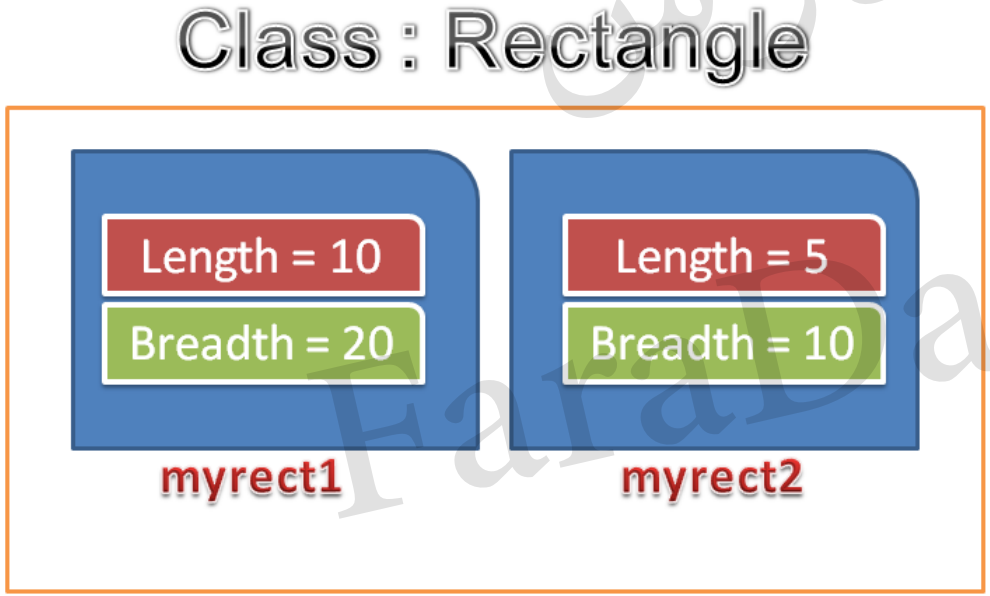


فرادرس
FaraDars.org

تعریف اشیاء

```
Rectangle myrect1 = new Rectangle();
Rectangle myrect2 = new Rectangle();
```

تمام اشیائی که از روی یک کلاس ایجاد می‌شوند بطور اختصاصی حاوی تمام اجزاء کلاس مذکور هستند.



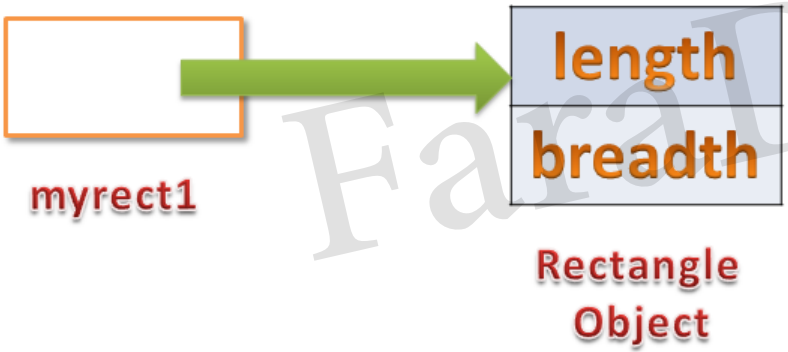
فرادرس
FaraDars.org

ایجاد کردن اشاره گر یا همان مرجع به اشیاء

```
Rectangle myrect1;
```

در صورتیکه از دستورnew در تعریف کردن اشیاء استفاده نکنیم شی ایجاد نمی‌شود، بلکه تنها یک مرجع یا همان اشاره‌گر ایجاد می‌شود. بنابراین مطابق شکل می‌توان تعریف شی را در دو مرحله انجام داد.

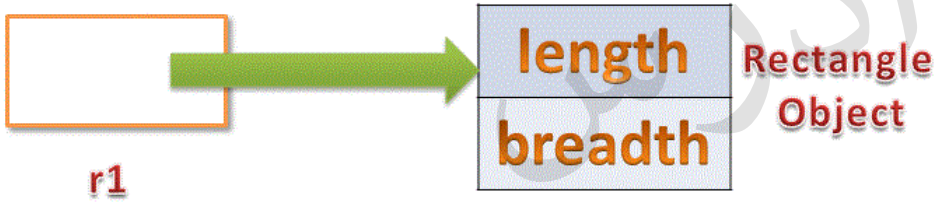
```
myrect1 = new Rectangle();
```



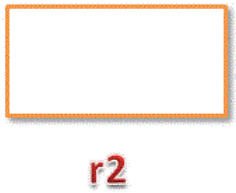
فرادرس
FaraDars.org

نسبت دادن یک مرجع به یک شی موجود

```
Rectangle r1 = new Rectangle();
```




```
Rectangle r2 = r1;
```



در صورتیکه از دستور new در تعریف کردن اشیاء استفاده نکنیم، شی ایجاد نمی‌شود بلکه تنها یک مرجع یا همان اشاره‌گر ایجاد می‌شود که می‌تواند به شئی که قبلا ایجاد شده نسبت داده شود و به عنوان نام دوم شی مذکور در نظر گرفته شود.

فرادرس
FaraDars.org

```
Rectangle R1 = new Rectangle();
```

نحوه فراخوانی اجزاء تابعی کلاس یا همان متدها

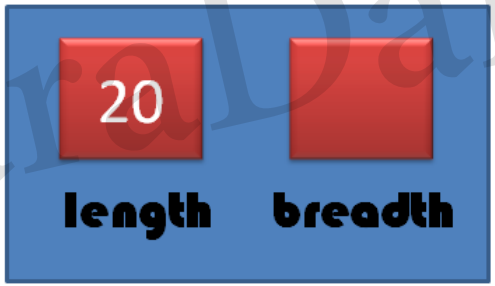
برای فراخوانی متدها ابتدا نام شی، علامت نقطه و سپس نام متد را می‌نویسیم و مقادیر مناسب را به عنوان آرگومان‌های مورد نیاز به متدها ارسال می‌کنیم.

```
R1.setLength(20);
```

Calling Method setLength()
For R1 Object

```
void setLength(int len)
{
length = len;
}
```

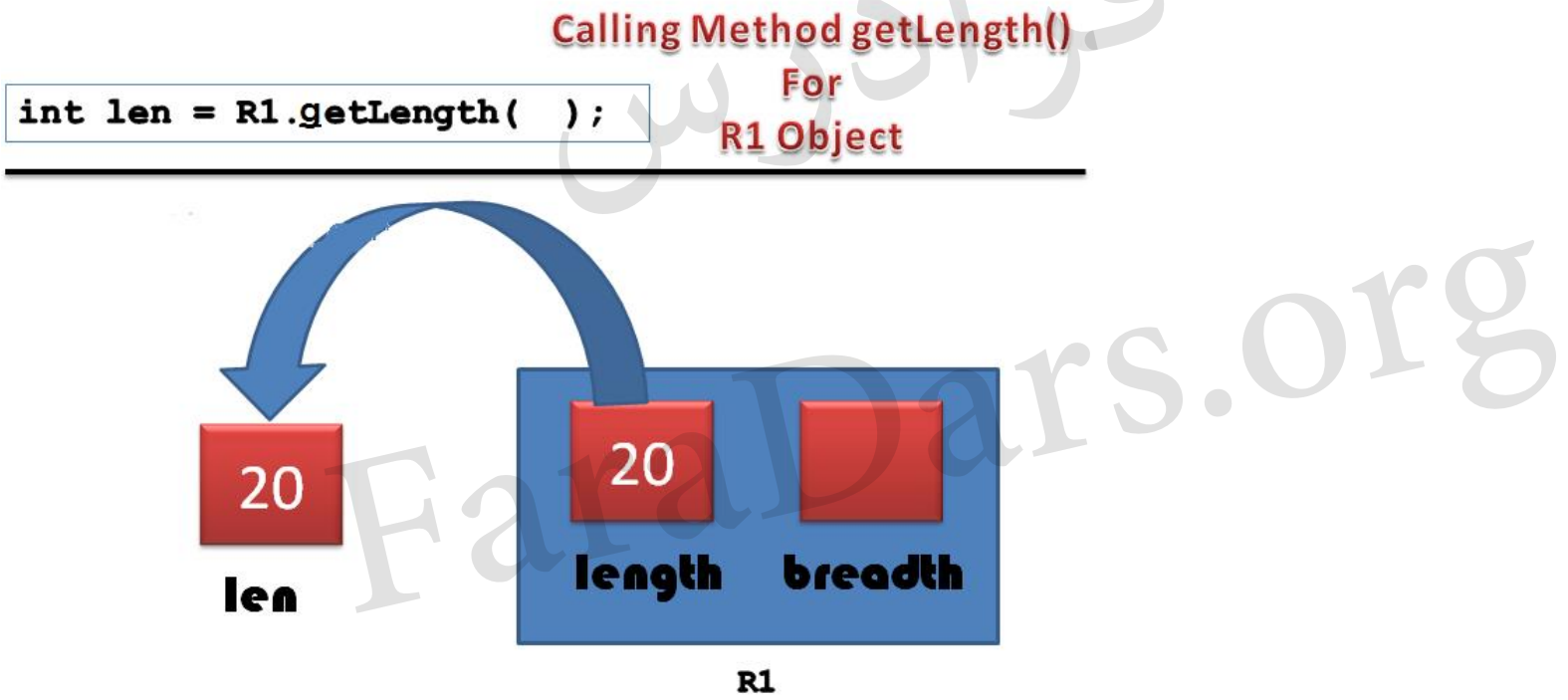

```
length = len;
```



فرادرس
FaraDars.org

فراخوانی متد getlength برای شی R1

```
Rectangle R1 = new Rectangle();
```



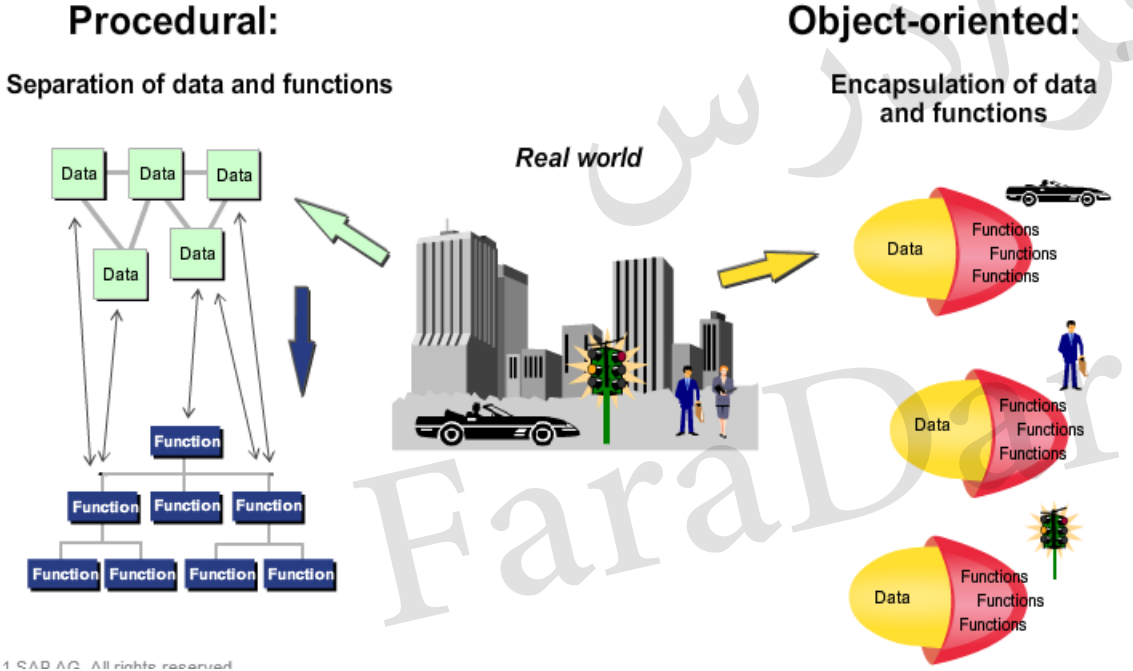
فرادرس
FaraDars.org

Object Oriented Programming برنامه نویسی شی گرا

ما در دنیایی از اشیاء زندگی می‌کنیم. کافیست نگاهی به اطراف خود بیاندازیم. اطراف ما پر است از اتومبیل‌ها، هواپیماها، انسان‌ها، حیوانات، ساختمان‌ها، چراغ‌های ترافیک، بالابرها، و بسیاری از چیزهای دیگر. قبل از اینکه زبان‌های برنامه‌نویسی شی‌گرا ابداع شوند، زبان‌های برنامه‌نویسی (همانند FORTRAN، Pascal، C و Basic) بر روی اعمال یا actions بجای چیزها یا اشیاء تمرکز داشتند. با اینکه برنامه‌نویسان در دنیائی از اشیاء زندگی می‌کردند اما با افعال سرگرم بودند. خود همین تناقض باعث شد تا برنامه‌های نوشته شده از قدرت کافی برخوردار نباشند. هم اکنون که زبان‌های برنامه‌نویسی شی‌گرا همانند C# و Java در دسترس هستند، برنامه‌نویسان به زندگی خود در یک دنیای شی‌گرا ادامه می‌دهند و می‌توانند برنامه‌های خود را با اسلوب شی‌گرا بنویسند. فرآیند برنامه‌نویسی شی‌گرا در مقایسه با برنامه‌نویسی *روالی*(*procedural*) ماهیت بسیار طبیعی‌تری دارد و نتیجه آن هم رضایت بخش‌تر است.

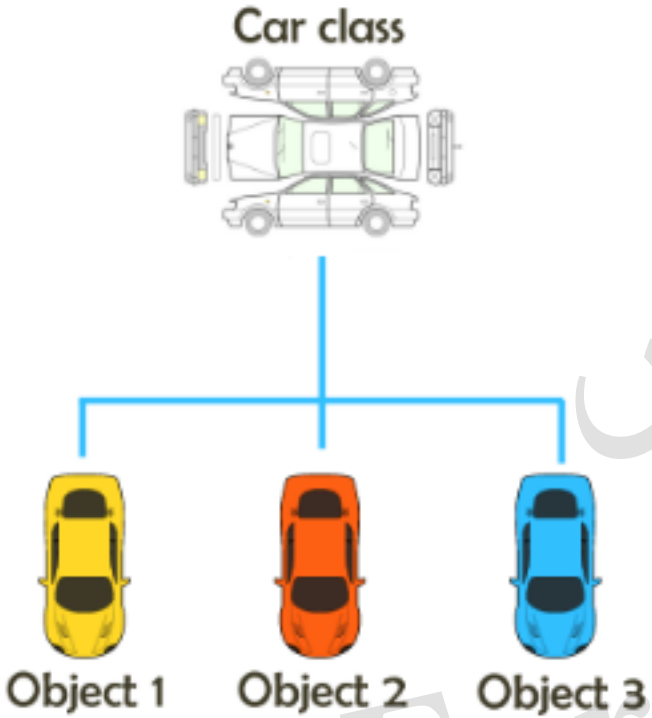
فرادرس
FaraDars.org

برنامه نویسی شی‌گرا در مقایسه با برنامه نویسی روال‌گرا



اشیاء چه هستند و چرا خاص می‌باشند؟ در واقع تکنولوژی شی، یک الگوی بسته(package) است که به ما در ایجاد واحدهای نرم‌افزاری با معنی کمک می‌کند. اشیاء تمرکز زیادی بر نواحی خاص برنامه‌ها دارند. اشیاء می‌توانند از جمله اشیاء تاریخ، زمان، پرداخت، فاکتور، صدا، ویدئو، فایل، رکورد و بسیاری از موارد دیگر باشد. در حقیقت می‌توان هر چیزی را بفرم یک شی عرضه کرد. فرآیند برنامه‌نویسی شی‌گرا در مقایسه با برنامه‌نویسی *روالی* (*procedural*) ماهیت بسیار طبیعی‌تری دارد و نتیجه آن هم رضایت بخش‌تر است.

11 SAP AG. All rights reserved



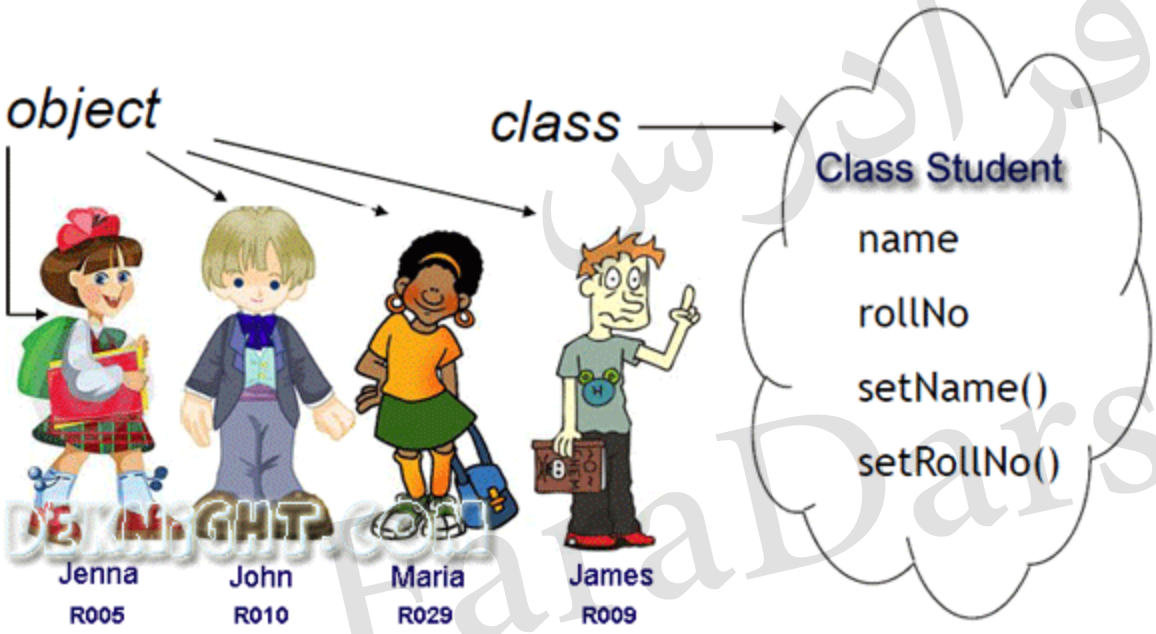
کلاس و شی (Class & object)

تمرکز برنامه‌نویسان oop بر روی ایجاد «*انواع داده‌ای تعریف شده از سوی کاربر*» که *کلاس* نامیده می‌شوند است. قبل از اینکه بتوانید با اتومبیلی رانندگی کنید، باید کسی آن را طراحی و ساخته باشد. معمولاً ساخت اتومبیل، با ترسیم یا نقشه‌کشی مهندسی شروع شود. متأسفانه نمی‌توانید با نقشه‌های ترسیمی یک اتومبیل رانندگی کنید. قبل از اینکه با اتومبیلی رانندگی کنید باید آن اتومبیل از روی نقشه‌های ترسیمی ساخته شود.

کلاس‌ها همانند نقشه‌های ترسیمی خانه‌ها هستند. یک کلاس، نقشه ایجاد یک شی از کلاس است. همانطوری که می‌توانیم خانه‌های متعددی از روی یک نقشه بسازیم، می‌توانیم تعدادی شی از روی یک کلاس نمونه‌سازی کنیم. نمی‌توان در نقشه آشپزخانه مبادرت به آشپزی کرد، آشپزی فقط در آشپزخانه خانه امکان‌پذیر است. اشیاء نمونه‌های واقعی هستند که از روی کلاس‌ها ایجاد می‌شوند.

فرادرس
FaraDars.org

کلاس و شی (Class & object)

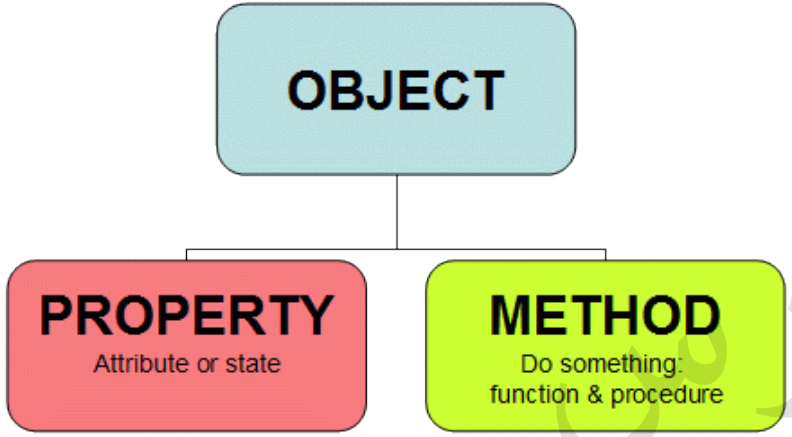


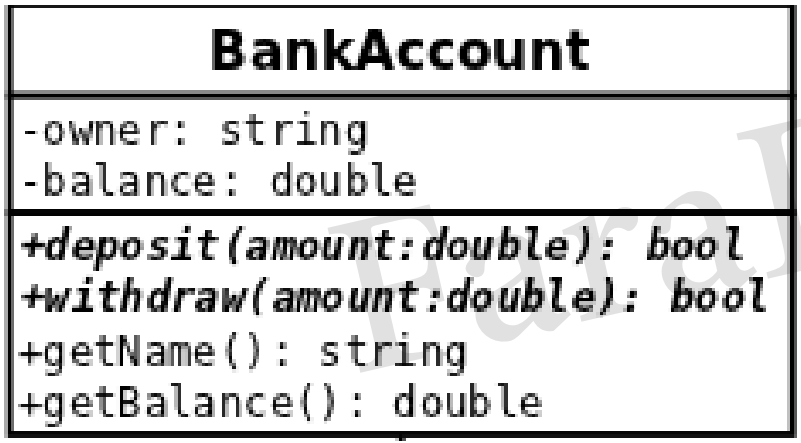
کلاس قالب یک موجودیت را تعریف می‌کند و اشیاء نمونه‌هایی واقعی از آن کلاس می‌باشند. کلاس هیچ حافظه‌ای اشغال نکرده و تنها قالب موجودیت را نمایش می‌دهد اما از روی یک کلاس چندین شی می‌توان تعریف کرد که هر یک برای خود حافظه اشغال کرده‌اند و مستقل از یکدیگرند.

فرادرس
FaraDars.org

کلاس‌ها، اجزاء داده‌ای و اجزاء تابعی

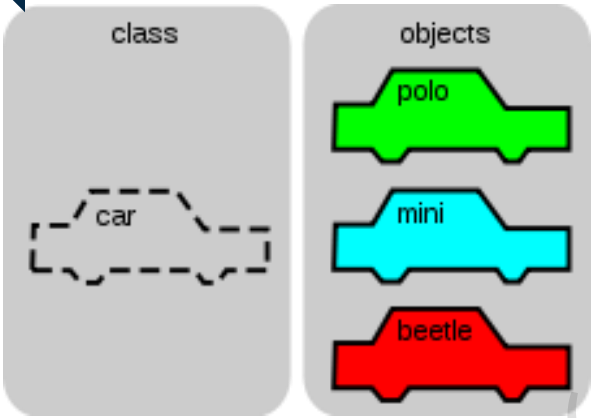
تمرکز برنامه‌نویسان C++ بر روی ایجاد «*انواع داده‌ای تعریف شده از سوی کاربر*» که *کلاس* نامیده می‌شوند است. هر کلاس حاوی اجزاء داده‌ای و هم مجموعه‌ای از متدهاست که بر روی داده‌ها کار می‌کنند و سرویس‌هایی برای *سرویس‌گیرنده‌ها*(کلاس‌ها و توابع دیگری که از کلاس استفاده می‌کنند) تدارک می‌بینند. برای مثال، یک کلاس حساب بانکی می‌تواند شامل یک شماره حساب و یک موجودی باشد که اجزاء داده‌ای کلاس محسوب می‌شوند و به رفتارهای هر موجودیت از یک کلاس، *توابع عضو* گفته می‌شود (معمولاً در سایر زبان‌های برنامه‌نویسی شی‌گرا همانند جاوا به توابع عضو، متد گفته می‌شود). برای مثال، یک کلاس حساب بانکی می‌تواند دارای توابع عضوی برای ایجاد یک پس‌انداز(افزایش‌دهنده موجودی)، برداشت(کاهش‌دهنده موجودی) و نمایش موجودی فعلی باشد. برنامه‌نویس از انواع توکار(و سایر انواع تعریف شده توسط کاربر در ایجاد انواع جدید(کلاس‌ها) استفاده می‌کند.

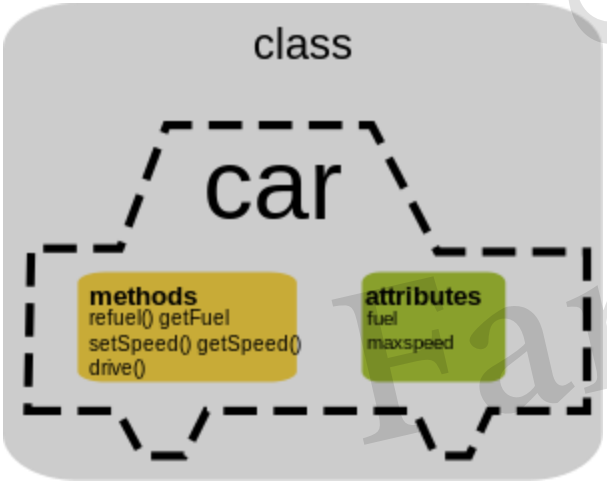




فرادرس
FaraDars.org

کلاس‌ها، اجزاء داده‌ای و اجزاء تابعی





فرض کنید که می‌خواهید با اتومبیلی رانندگی کرده و با فشردن پدال گاز آن را سریعتر به حرکت درآورید. چه اتفاقی قبل از اینکه بتوانید اینکار را انجام دهید، باید رخ دهد؟ بسیار خوب، قبل از اینکه بتوانید با اتومبیلی رانندگی کنید، باید کسی آن را طراحی و ساخته باشد. معمولاً ساخت اتومبیل با ترسیم یا نقشه‌کشی مهندسی شروع شود. همانند طراحی صورت گرفته برای خانه. این ترسیمات شامل طراحی پدال گاز است که راننده با استفاده از آن سبب می‌شود تا اتومبیل سریعتر حرکت کند. تا حدی پدال سبب «پنهان» شدن پیچیدگی مکانیزمی می‌شود که اتومبیل را سریعتر بحرکت درمی‌آورد، همانطوری که پدال ترمز سبب «پنهان» شدن مکانیزمی می‌شود که از سرعت اتومبیل کم می‌کند، فرمان اتومبیل سبب «پنهان» شدن مکانیزمی می‌شود که اتومبیل را هدایت می‌کند و موارد دیگر. با انجام چنین کارهایی، افراد عادی می‌توانند به آسانی اتومبیل را هدایت کرده و براحتی از پدال گاز، ترمز و فرمان، مکانیزم تعویض دنده و سایر «واسطهای» کاربرپسند و ساده استفاده کنند تا پیچیدگی مکانیزم‌های داخلی اتومبیل برای راننده مشخص نباشد. متأسفانه، نمی‌توانید با نقشه‌های ترسیمی یک اتومبیل رانندگی کنید، قبل از اینکه با اتومبیلی رانندگی کنید باید آن اتومبیل از روی نقشه‌های ترسیمی ساخته شود. یک اتومبیل کاملاً ساخته شده دارای پدال گاز واقعی برای به حرکت درآوردن سریع اتومبیل است، اما این هم کافی نیست. اتومبیل بخودی خود شتاب نمی‌گیرد، از اینرو لازم است راننده بر روی پدال گاز فشار آورد تا به اتومبیل دستور حرکت سریع‌تر را صادر کند.

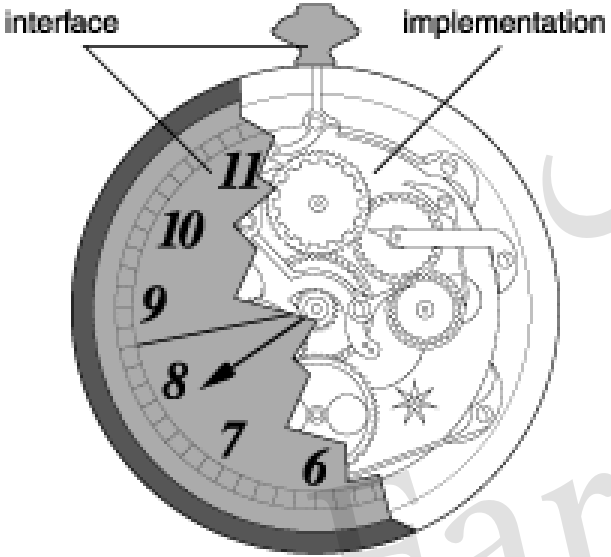
فرادرس
FaraDars.Org

Encapsulation یا بسته بندی

قرار دادن کلیه خصوصیات و متدهای یک کلاس در یک بسته‌بندی مشخص را **Encapsulation** می‌نامند. بسته بندی کردن(package) نرم‌افزار بصورت کلاس‌ها می‌تواند ویژگی استفاده مجدد از نرم‌افزار را عرضه کند.

FaraDars.org

Interface



Interface به عنوان یک قرارداد تعریف می‌شود که همه‌ی گروه‌هایی که آنرا می‌گیرند باید آن را دنبال کنند. Interface قسمت چه باید کرد از یک قرارداد را تعریف می‌کند و گروه‌های مشتق شده قسمت چگونگی انجام این قراردادها را تعریف می‌کنند. Interface تنها شامل اعلام اعضا می‌باشد و اغلب به تأمین یک ساختار استاندارد کمک می‌کنند. تعریف عملکرد اعضاء مسئولیت گروه مشتق شده می‌باشد. Interface نشان می‌دهد که میخواهید یک آبجکت چگونه مورد استفاده قرار گیرد و به این که چگونه پیاده‌سازی می‌شود کاری ندارد. به کمک Interface می‌توان عملکرد وراثت چندگانه را در C# شبیه‌سازی کرد.

فرادرس
FaraDars.org

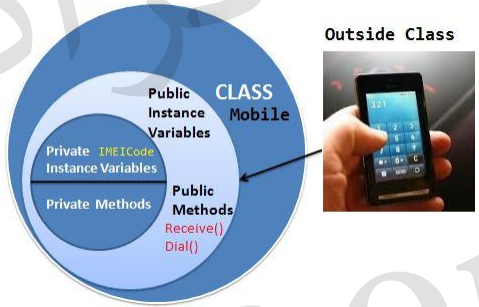
Access Modifiers

Access Modifiers	Applicable to the Application	Applicable to the Current Class	Applicable to the Derived Class
public	✓	✓	✓
private	✗	✓	✗
protected	✗	✓	✓
internal	✗	✓	✓

Accessibility Levels

اجزاء داده‌ای و تابعی درون کلاس می‌توانند یکی از سه سطح دسترسی زیر را داشته باشند:

- Private
- Public
- Protected

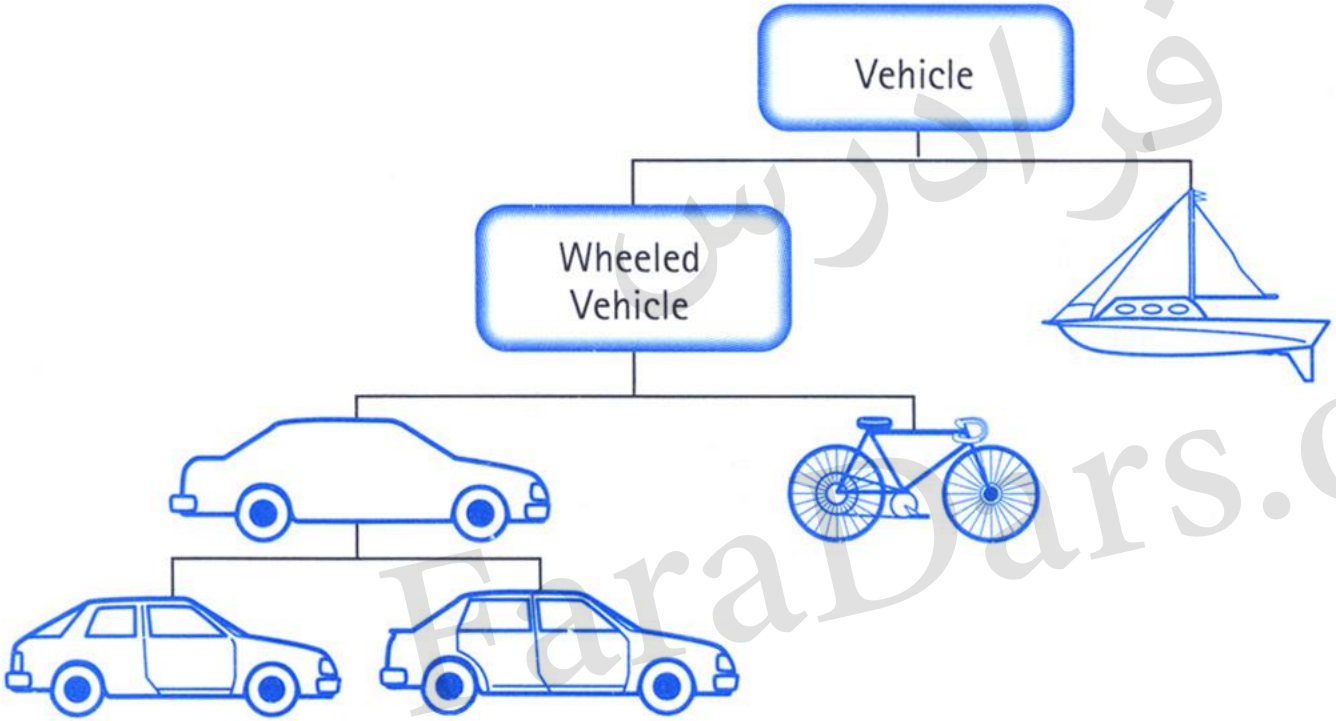


انواع سطوح دسترسی:

1. private: این سطح دسترسی مشخص می‌کند که قطعه کد تعریف شده تنها داخل خود پروژه یا Scope مربوطه قابل دسترس باشند. برای مثال کلاسی که به صورت private تعریف شده باشد، تنها داخل همان پروژه قابل دسترس بوده و از سایر پروژه‌هایی که در solution تعریف شده قابل دسترس نخواهد بود، یا اعضای کلاسی که به صورت private تعریف شده‌اند، تنها در Scope همان کلاس که بین علامت‌های {} می‌باشد قابل دسترس خواهند بود.
2. public: کدهایی که با این سطح دسترسی مشخص شده باشند، در تمامی قسمت‌های پروژه و سایر پروژه‌ها قابل دسترس خواهند بود.
3. internal: سطوح دسترسی internal تنها داخل همان فضای نام قابل دسترس بوده و سایر فضای نام‌ها به آنها دسترسی نخواهند داشت. این سطح دسترسی برای کلاس‌ها کاربرد زیادی دارد.
4. protected: این سطح دسترسی زمانی که از مفهوم inheritance استفاده کنیم کاربرد دارد. در قسمت وراثت این سطح دسترسی را به تفصیل مورد بررسی قرار خواهیم داد.
5. internal protected: همانند قسمت protected این دسترسی نیز در قسمت وراثت توضیح داده خواهد شد که تلفیقی از دسترسی‌های internal و protected می‌باشد.

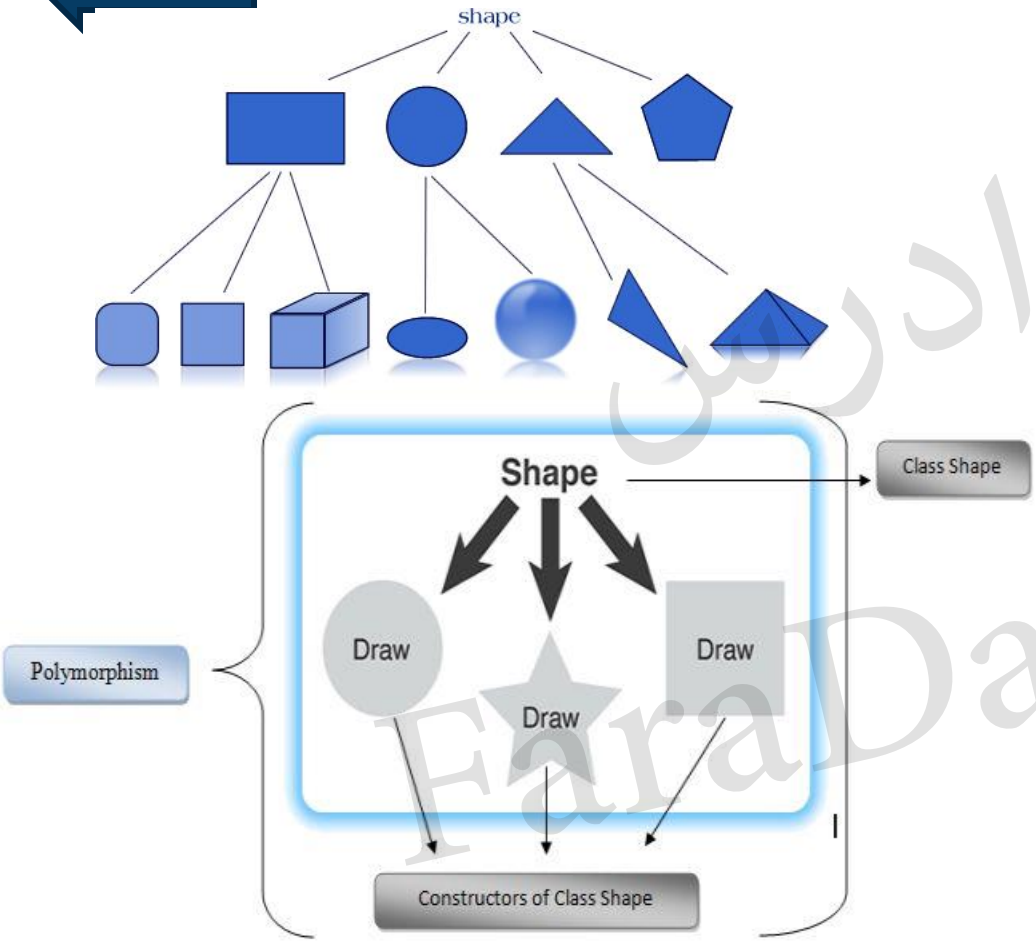
فرادرس
FaraDars.org

وراثت Inheritance



ارث‌بری فرمی از بکارگیری مجدد نرم‌افزار است که در آن کلاس‌های ایجاد شده اجزاء داده‌ای و رفتارهای خود را براساس اطلاعات یک کلاس موجود بدست آورده و در صورت نیاز حاوی قابلیت‌های جدید هستند. بکارگیری مجدد نرم‌افزار سبب کاهش مدت زمان توسعه نرم‌افزار شده و کیفیت آنرا بطور موثری افزایش می‌دهد.

فرادرس
FaraDars.org



چند ریختی Polymorphism

چندریختی امکان می‌دهد تا برنامه‌ها بجای اینکه«*برنامه خاصی*» باشند، حالت یک «*برنامه کلی*» داشته باشند.
در عمل، چند ریختی امکان می‌دهد تا برنامه‌هایی بنویسیم که مبادرت به پردازش اشیاء از کلاس‌هایی کنند که بخشی از همان سلسله مراتب کلاس هستند، همچنانکه همگی آنها اشیائی از سلسله مراتب کلاس مبنا می‌باشند. به کمک چندریختی، می‌توانیم سیستم‌های را طراحی و پیاده‌سازی کنیم که گسترش و بسط‌پذیری آنها آسانتر است. کلاس‌های جدید می‌توانند با کمی تغییر یا اصلاح در بخش‌های عمومی برنامه، به آن افزوده شوند، مادامیکه کلاس‌های جدید بخشی از سلسله مراتب توارثی باشند که برنامه بطور جامع آنرا پردازش می‌کند. تنها بخش‌هایی از برنامه که باید برای تطبیق یافتن با کلاس‌های جدید تغییر داده شوند آنهایی هستند که نیاز دارند تا از وجود کلاس‌های جدید افزوده شده به سلسه مراتب مستقیماً مطلع گردند.

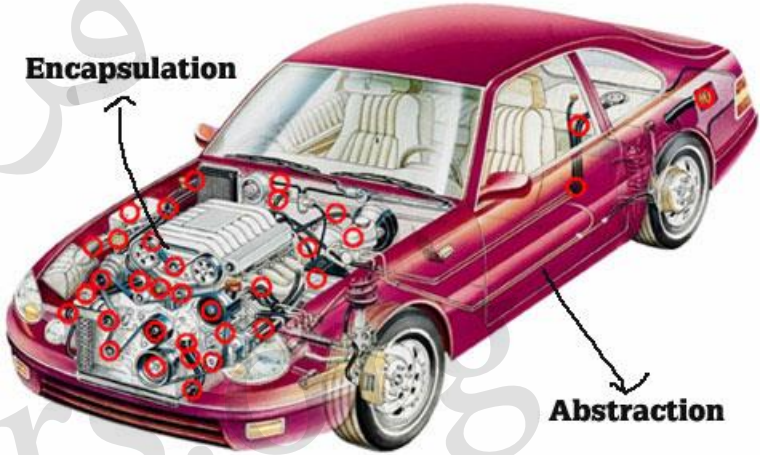
فرادرس
FaraDars.Org

Abstraction

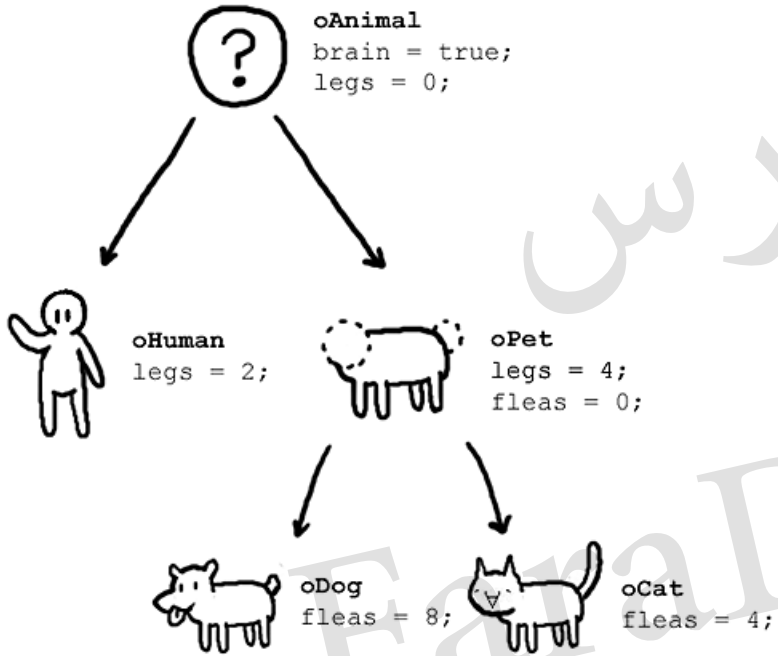
در لغت به معنای تجرد یا انتزاع است و به مفهوم یک برداشت واحد از مجموعه مشاهدات می‌باشد. در واقع انتزاع همان نیرویی است که با دیدن تعداد زیادی درخت مفهوم جنگل را در ذهن انسان تداعی می‌کند یا با دیدن مجموعه‌ای از پیکسل‌های صفحه نمایش کامپیوتر یک تصویر خاص را در ذهن ایجاد می‌کند. در این روش برنامه‌نویس باید سعی کند اشیائی را تعریف کند که برنامه را از تعریف اشیاء بیشتر بی‌نیاز کند. در واقع او همواره باید به دنبال اشیاء مادر بگردد. یعنی بعد از یافتن توابع و داده‌های مورد نیاز برای پیاده‌سازی سیستم مورد نظر باید مجموعه‌ای از توابع و داده‌ها را توسط "یک" کلاس(شئ) پیاده‌سازی کند. Abstraction در اصل به معنی نمایش دادن تنها جزئیات مورد نیاز به سرویس گیرنده است.

FaraDars.org

Abstraction



Abstract Class



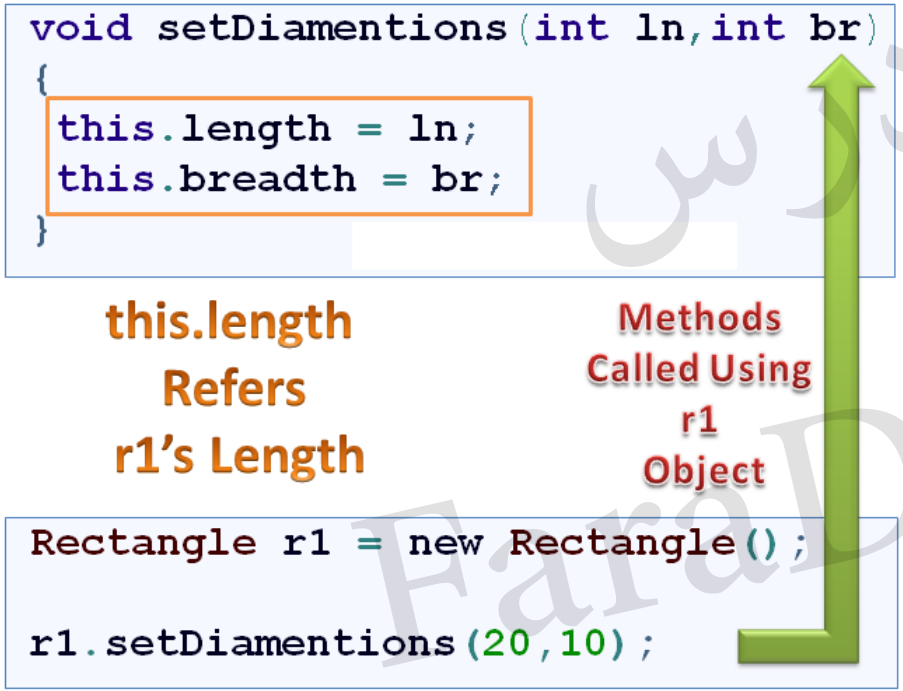
در ارث‌بری کلاس‌های پایه، تنها شامل متدها و خصوصیاتی هستند که در بین کلاس‌هایی که از این کلاس‌ها مشتق می‌شوند مشترک هستند. به عبارت دیگر خود کلاس‌های پایه دارای مفهوم کاملی از یک شئ نمی‌باشند، درنتیجه ایجاد اشیاء جدید از این کلاس‌ها بی‌معنی است. برای اینکه نتوان از کلاس‌های پایه اشیاء نمونه‌ای تعریف کرد. آنها را به صورت abstract تعریف می‌نماییم.

نکته اینکه نمی‌توان اشیاء جدیدی از کلاس‌های abstract ایجاد کرد.

فرادرس
FaraDars.org

مفهوم مرجع this در متدهای کلاس

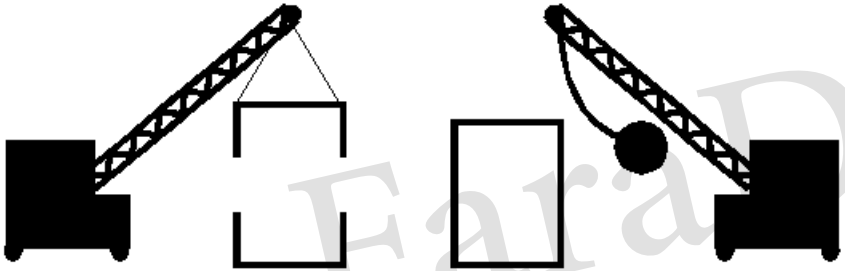
مشاهده کردید که توابع عضو یک شی می‌تواند داده‌های شی را دستکاری کند. چگونه توابع عضو می‌دانند که اعضای داده کدام یک از اشیاء را دستکاری کنند؟ هر شی از طریق یک اشاره‌گر بنام **this** (یک کلمه کلیدی در C#) به آدرس متعلق بخود دسترسی دارد. اشاره‌گر **this** یک شی، بخشی از خود شی نمی‌باشد، اشاره‌گر **this** بصورت یک آرگومان ضمنی به هر تابع عضو غیراستاتیک شی ارسال می‌شود(توسط کامپایلر). اشیاء از اشاره‌گر **this** بصورت ضمنی(که در این بخش آنرا انجام می‌دهیم) یا صریح برای مراجعه اعضای داده و توابع عضو خود استفاده می‌کنند.



فرادرس
FaraDars.org

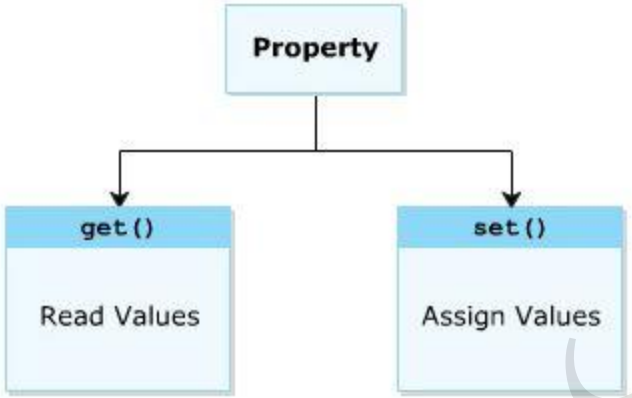
تابع عضو سازنده و تابع عضو مخرب

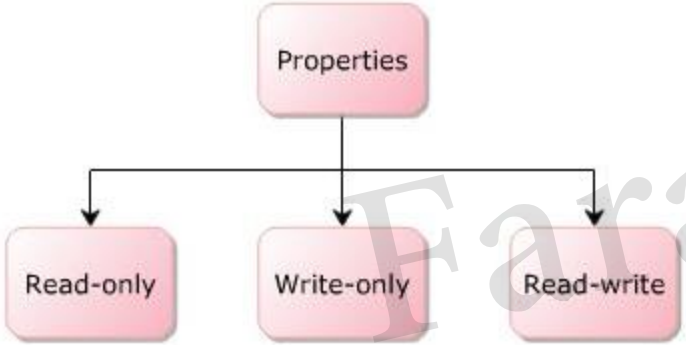
- هر کلاسی که اعلان می‌کنید می‌تواند یک سازنده(*constructor*) داشته باشد که می‌توان با استفاده از آن مبادرت به مقداردهی اولیه یک شی از کلاس به هنگام ایجاد شی کرد. سازنده یک تابع عضو ویژه است که بایستی همنام با نام کلاس تعریف شده باشد، از اینروست که کامپایلر می‌تواند آن را از دیگر توابع عضو کلاس تشخیص دهد. مهمترین تفاوت موجود مابین سازنده‌ها و توابع دیگر در این است که سازنده‌ها نمی‌تواند مقدار برگشت دهند، بنابراین نمی‌توانند نوع برگشتی داشته باشند(حتی **void**). معمولاً سازنده‌ها بصورت **public** اعلان می‌شود. هر زمان که شی از روی یک کلاس ایجاد می‌کنیم اتوماتیک سازنده کلاس اجرا می‌گردد.C# نیازمند فراخوانی یک سازنده برای هر شی است که ایجاد می‌شود، در چنین حالتی مطمئن خواهیم بود که شی قبل از اینکه توسط برنامه بکار گرفته شود بدرستی مقداردهی اولیه شده است. فراخوانی سازنده به هنگام ایجاد شی، بصورت غیرصریح یا ضمنی انجام می‌شود. در هر کلاسی که بصورت صریح سازنده‌ای را مشخص نکرده است، کامپایلر یک *سازنده پیش‌فرض* تدارک می‌بیند این سازنده دارای پارامتر نمی‌باشد.
- *نابودکننده*(مخرب) نوع دیگری از تابع عضو می‌باشد. نام تابع مخرب یک کلاس با کاراکتر مد(~) و نام کلاس مشخص می‌شود. مخرب یک کلاس بصورت ضمنی(غیرصریح) و در زمان از بین رفتن شی فراخوانی می‌شود. مخرب پارامتر دریافت نمی‌کند و مقداری برگشت نمی‌دهد و نمیتوان برای آن سطح دسترسی مشخص کرد. یک کلاس می‌تواند فقط یک مخرب داشته باشد و نمی‌توان آن را سربارگذاری کرد.



FaraDars.Org

خصوصیت یا Property





property (یا خصوصیت) استانداردی در سی شارپ برای دسترسی به اعضای داده‌ای با سطح دسترسی private در داخل یک کلاس می‌باشد. هر property دارای دو بخش است؛ یک بخش جهت مقداردهی بلوک(set) و یک بخش برای دسترسی به مقدار بلوک(get) یک داده private. propertyها باید به صورت public تعریف شوند تا در کلاس‌های دیگر نیز قابل دسترسی باشند. Propertyهایی که بدنه get ندارند Write-Only و آنهایی که بدنه set را ندارند Read-Only می‌گوییم.

فرادرس
FaraDars.org

شکل کلی یک خصوصیت یا Property

ساختار کلی propertyها به صورت مقابل می‌باشد:

```
{access-modifier} {data-type} {property-name}
{
    [access-modifier] get
    {
        // body for get value
    }
    [access-modifier] set
    {
        // body for set value
    }
}
```

- access-modifier: سطح دسترسی به Property را تعیین می‌کند. Property نیز مانند فیلد می‌تواند سطح دسترسی داشته باشد.
- data-type: نوع Property که یکی از Data Typeهای دات نت یا کلاسی است که به صورت دستی نوشته شده باشد.
- property-name: نام Property است و برای نامگذاری آن همیشه از قاعده PascalCase استفاده می‌شود. در حقیقت سطح دسترسی تفاوتی ندارد و همیشه آن را PascalCase تعریف کنید.
- بدنه get: این بدنه، دقیقاً معادل متد Getایست که در قسمت قبلی تعریف کردیم. شما داخل بدنه get می‌توانید هر دستوری بنویسید، در حقیقت این بدنه مانند یک متد عمل کرده و زمانی که شما مقدار Property را می‌خوانید دقت کنید بدنه get حتماً باید مقداری را با دستور return برگرداند. همچنین این بدنه می‌تواند دارای access-modifier باشد، یعنی سطح دسترسی خواندن مقدار را مشخص می‌کند. در صورتی که سطح دسترسی را مشخص نکنید به صورت پیش‌فرض public در نظر گرفته می‌شود.
- بدنه set: این بدنه دقیقاً معادل متد Set در مثال قبلی است. زمانی که شما مقداری را داخل Property ست کنید، بدنه set اجرا می‌شود. داخل بدنه set پارامتر پیش فرضی وجود دارد به نام value که مقدار ست شده برای Property داخل آن قرار گرفته و شما می‌توانید از طریق بدنه set به آن دسترسی داشته باشید. همچنین می‌توان برای بدنه set سطح دسترسی مشخص کرد که اگر سطح دسترسی را مشخص نکنید، به صورت پیش فرض public در نظر گرفته می‌شود.

فرادرس
FaraDars.org

مثالی برای ایجاد Property

فرض کنید بخواهیم یک کلاس برای موجودیت شخص ایجاد کنیم. تصمیم داریم عملیاتی که بوسیله متدهای Getو Set در کلاس Rectangle انجام دادیم را با Propertyها پیاده‌سازی کنیم. کلاس Person را به شکل مقابل ایجاد می‌کنیم. در کد مقابل، دو Property با نام‌های FirstNameو LastName تعریف کردیم که عملیات خواندن و نوشتن از فیلدهای مربوطه را انجام می‌دهند. برای نوشتن Propertyها حتماً نیازی به تعریف Field برای آنها نیست. شما می‌توانید هر کدی را برای بدنه get یا set بنویسید. به عنوان مثال می‌خواهیم برای کلاس Person یک Property تعریف کنیم که نام کامل شخص را برگرداند. نام این خاصیت را FullName می‌گذاریم. نگاهی دوباره به کلاس Person و خاصیت FullName می‌کنیم، اگر دقت کرده باشید این خاصیت تنها بدنه get دارد و بدنه set را برای آن ننوشتیم. دلیل این امر آن است که FullName تنها ترکیبی از firstNameو lastName را برمی‌گرداند و در صورتی که بخواهیم مقداری داخل FullName بریزیم، با پیغام خطا مواجه می‌شویم.

به Propertyهایی که بدنه get ندارند Write-Only، و آنهایی که بدنه set را ندارند Read-Only می‌گوییم. همچنین همانطور که قبلاً هم گفتیم می‌توانیم علاوه بر خود Property برای هر یک از بدنه‌های getو set نیز سطح دسترسی مشخص کنیم.

```
public class Person
{
    private string firstName;
    private string lastName;

    public string FirstName
    {
        get { return firstName; }
        set { firstName = value; }
    }

    public string LastName
    {
        get { return lastName; }
        set { lastName = value; }
    }
}
```

```
public class Person
{
    private string firstName;
    private string lastName;

    public string FirstName
    {
        get { return firstName; }
        set { firstName = value; }
    }

    public string LastName
    {
        get { return lastName; }
        set { lastName = value; }
    }

    public string FullName
    {
        get { return FirstName + " " + LastName; }
    }
}
```

FaraDars.org

Automatic Properties ایجاد

گاهی اوقات Property که تعریف می‌کنیم تنها عملیات خواندن و نوشتن یک فیلد را کنترل می‌کند. برای مثال، کلاس Person را در نظر بگیرید. کد مربوط به خصوصیت بالا را می‌توان به شکل مقابل نیز نوشت. کامپایلر بعد از کامپایل کد بالا، به صورت خودکار یک فیلد برای خاصیت نوشته شده تعریف کرده و بدنه getو set آن را به صورت خودکار می‌نویسد. از مزایای Auto-Propertyها حجم کد کمتر و البته قابلیت کنترل دسترسی به عملیات خواندن و نوشتن Propertyها می‌باشد. مثال بالا را به نحوی تغییر می‌دهیم که خاصیت FirstName تنها داخل کلاس قابل نوشتن باشد. به این نکته توجه داشته باشید، زمانی که از Auto-Propertyها استفاده می‌کنید حتماً باید getو set را بنویسید، در غیر اینصورت پیغام خطا دریافت خواهید کرد. البته این مشکل در نسخه ۶ زبان سی شارپ برطرف شده است.

```
1  public class Person
2  {
3      private string firstName;
4
5      public string FirstName
6      {
7          get { return firstName; }
8          set { firstName = value; }
9      }
10 }
```

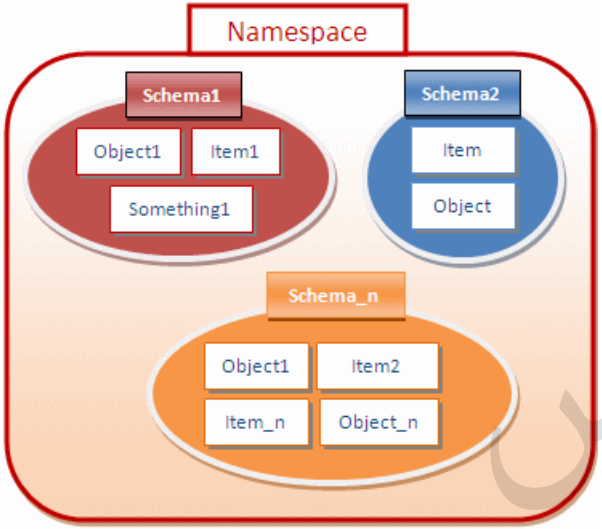
```
1  public class Person
2  {
3      public string FirstName { get; set; }
4  }
```

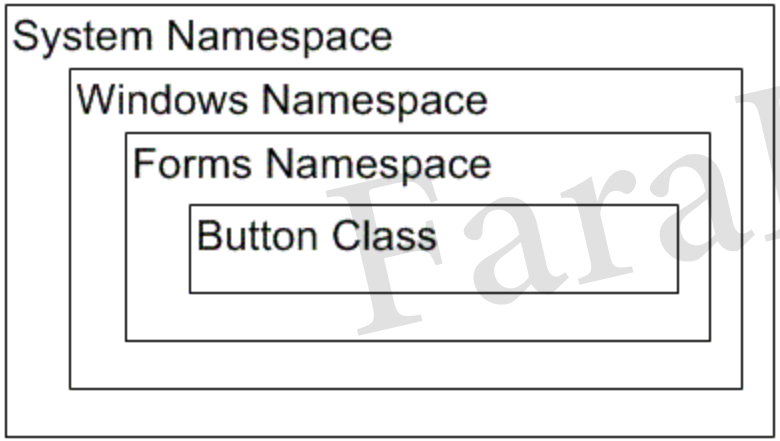
```
1  public class Person
2  {
3      public string FirstName { get; private set; }
4  }
```

FaraDars.org

فضای نام Namespace

فضای نام یک ساختار سازمانی است که کلاس‌ها را دسته‌بندی می‌کند. به عبارت دیگر منطقه‌ای برای تعریف کردن کلاس‌هاست. فضای نام به شما کمک می‌کند که پایه و اساس کدهایتان را پیدا و درک کنید. فضای نام‌ها برای برنامه‌نویسی سی شارپ ضروری نیستند، آنها معمولاً برای بهبود قابلیت فهم کد استفاده می‌شوند. بدون استفاده از فضای نام، تمامی اسامی مورد استفاده در برنامه سعی می‌کنند در یک فضای نام کلی، جایی برای خود در نظر بگیرند که موجب برخورد بین اسامی یکسان می‌شود. اما اگر از فضای نام‌های جداگانه‌ای استفاده شود، هیچ اشکالی در برنامه به وجود نمی‌آید. در واقع فضای نام برای تفکیک مجموعه‌ای از اسامی از مجموعه دیگر به کار می رود.





FaraDars.org
فرادرس

چند نمونه فضای نام موجود در C#

از مهمترین فضای نام‌های موجود در کتابخانه .Net Framework فضای نام System است. این فضای نام دارای فضای نام‌های دیگری در داخل خود می‌باشد. به عنوان مثال Sysytem.IO که حاوی کلاس‌هایی برای کار با ورودی و خروجی و کار با فایل‌ها می‌باشد.

چند نمونه از فضای نام ها:

فضای نام	شرح
System	شامل کلاسهای اساسی و انواع داده می باشد.
System.Data	حاوی کلاسهایی برای دستیابی به داده های بانک اطلاعاتی.
System.Drawing	کلاسهایی برای ترسیم و گرافیک.
System.IO	کلاسهایی برای ورودی و خروجی داده.
System.Windows.Forms	کلاسهایی برای ساخت واسط کاربری.
System.Xml	برای پردازش داده های Xml استفاده میشود.

فرادرس
FaraDars.org

شکل کلی ایجاد فضای نام و استفاده از آن

```
namespace namespace_name
{
// code declarations
}
```

```
using namespace_name;
```

```
namespace namespace_name1
{
// code declarations
namespace namespace_name2
{
// code declarations
}
}
```

```
using namespace_name1.namespace_name2;
```

استفاده از فضای نام و نام کلاس که با نقطه از هم جدا می‌شوند.
به عنوان مثال:

```
System.Convert.ToString();
```

استفاده از دستور Using

```
using System;
Convert.ToString();
```

در این روش ابتدا دستور using را برای استفاده از فضای نام می‌نویسیم، سپس کلاس‌های موجود در آن فضا را بدون ذکر نام و در هر جایی از سند به کار خواهیم گرفت. شما می‌توانید از فضای نام‌ها به صورت تودرتو نیز استفاده کنید.

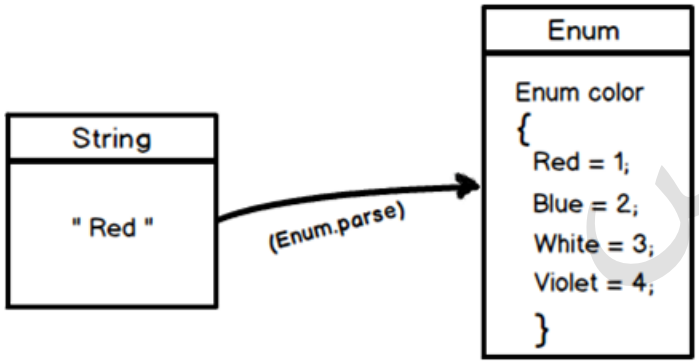
```
System.Windows.Forms.MessageBox.Show("");
```

استفاده از دستور Using

```
using System.Windows.Forms;
MessageBox.Show()
```

فرادرس
FaraDars.org

آشنایی با مفاهیم نوع داده شمارشی یا Enum



نوع داده شمارشی یا Enum جهت تعریف مقادیر ثابت و قابل شمارش در برنامه بسیار کاربرد دارد. مقادیری که در این نوع داده تعریف می‌شوند بطور خودکار از عدد ۰ شماره گذاری می‌شوند و به ترتیب یکی به آن‌ها اضافه می‌شود.

در این حالت متد ()ToString برای نوع داده‌ای Enum عنوان مقادیر ثابت را بر می‌گرداند.

روش عرف برای نمایش مقدار عددی استفاده از تبدیل نوع صریح به int است.

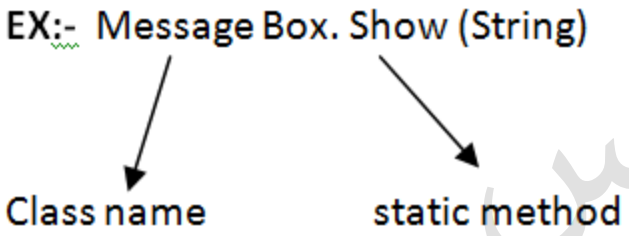
جهت تغییر شماره مقادیر کافیست بصورت زیر عمل کنیم:

```
enum TypesOfColo
{
Black=-5,
Green=20,
White,
Red,
Blue
}
```

```
enum <enum_name>
{
   enumeration list
};
```

فرادرس
FaraDars.org

آشنایی با کلمه کلیدی static



گاهی نیاز دارید که اعضای یک کلاس به هیچ شیئی وابسته نباشند. به‌طور معمول اعضای کلاس از طریق شیئی که از آن کلاس ساخته می‌شود قابل دسترسی هستند اما شما می‌توانید عضوی از کلاس را طوری تعریف کنید که بدون ساخت هیچ شیئی مستقیماً(از طریق نام کلاس و عملگر نقطه) به آن دسترسی داشته باشید. برای ساخت چنین عضوی، قبل از تعریف آن عضو از کلمه‌ی‌کلیدی static استفاده می‌کنید. هنگامی که عضوی از یک کلاس به‌صورت static تعریف می‌شود، آن عضو بدون ساخت هیچ objectای از کلاس قابل دسترسی و در واقع مستقل از اشیاء است و به هیچ شیئی از آن کلاس وصل نمی‌شود. شما می‌توانید هم متدها و هم متغیرها را به‌صورت static تعریف کنید. به عنوان مثال متد show از کلاس MessageBox یک متد استاتیک می‌باشد.

```
Convert.ToDouble()
```

▲ 1 of 18 ▼ double Convert.ToDouble(**bool value**)

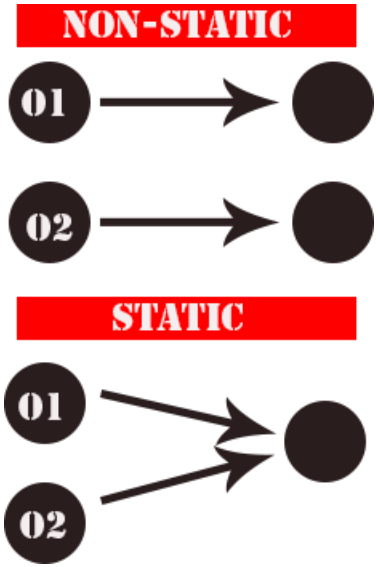
Converts the specified Boolean value to the equivalent double-precision floating-point number.

value: *The Boolean value to convert.*

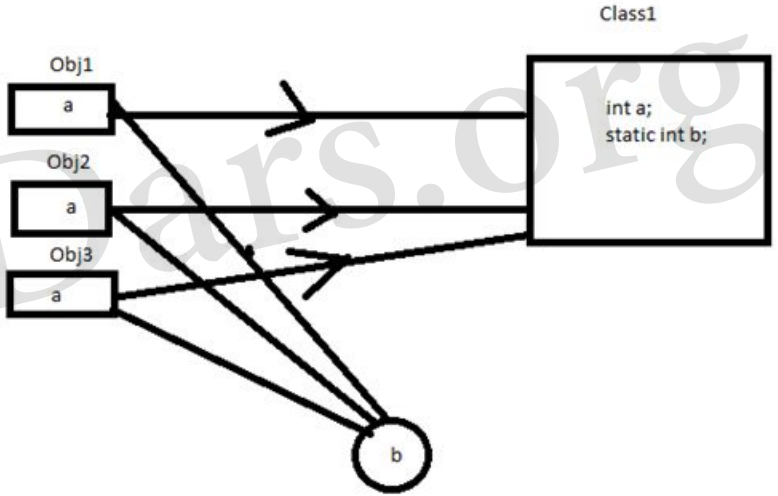
فرادرس
FaraDars.org

اجزاء داده‌ای static

اعضای static در حقیقت اعضایی هستند که وابسته به شئ نیستند و در بین کل اشیاء ساخته شده از یک کلاس مشترک می‌باشند. یعنی به ازای هر شئ مقدار متفاوتی ندارند، چون وابسته به شئ نیستند.




```
01.  Class Class1
02.  {
03.      int a;//instace type
04.      static int b ;// static type
05.  }
```



فرادرس
FaraDars.org

اجزاء تابعی یا متدهای static

متد static تنها می‌تواند به اعضای static دسترسی داشته باشد و نمی‌تواند مستقیماً به اعضای عادی کلاس دسترسی پیدا کند، زیرا اعضای عادی یک کلاس حتماً باید به یک شیء وصل شوند تا مقدارشان در آن شیء ذخیره شود اما اعضای static مستقل از اشیاء هستند و می‌توان مستقیماً به آن‌ها به کمک نام کلاس دسترسی پیدا کرد. برای اینکه بتوانید به اعضای staticای که در کلاس هستند دسترسی داشته باشید، کافی است ابتدا نام کلاس را نوشته و سپس توسط عملگر(.) به آنها دسترسی پیدا کنید.

اگر قصد دارید درون یک متد static به اعضای عادی نیز دسترسی داشته باشید باید از طریق یک شیء این کار را انجام دهید. برای ایجاد یک متد static کافیست کلمه کلیدی static بعد از Access Modifier و قبل از نوع خروجی متد بیاوردید.

```
class Class1
{
public static void method1()
{

}
}
```

```
Class1.method1();
```

فرادرس
FaraDars.org

کلاس static

همچنین می‌توانید یک کلاس را به‌صورت static تعریف کنید. هنگامی‌که یک کلاس را به‌صورت static تعریف می‌کنید:

(۱) دیگر نمی‌توانید از روی این کلاس شیء بسازید.

(۲) همه‌ی اعضای کلاس باید static باشند.

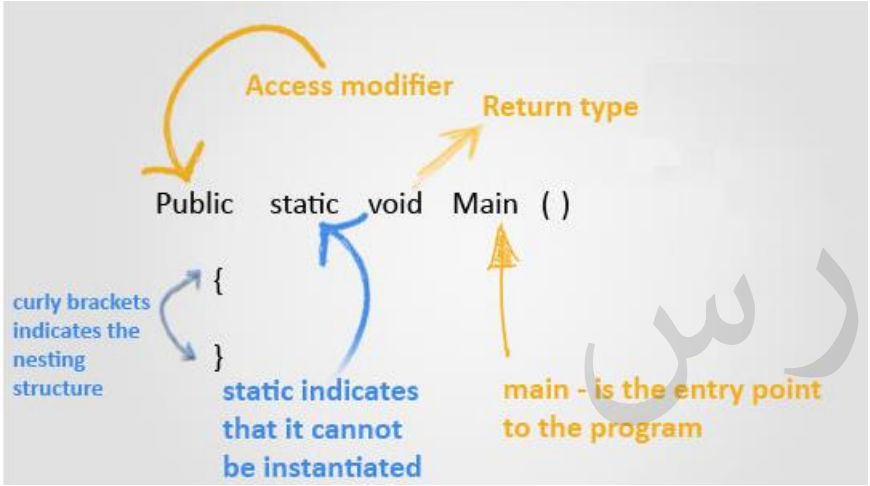
یکی از مزایای استفاده از کلاس‌های استاتیک این است که کامپایلر تضمین می‌کند به هیچ عنوان نمونه‌ای از این کلاس ساخته نشود. کلاس‌های استاتیک sealed هستند درنتیجه نمی‌توان از آنها ارث برد. این کلاس‌ها نمی‌توانند constructor داشته باشند ولی با این حال می‌توان از static constructorها برای مقداردهی به عناصر static کلاس استفاده کرد. توجه کنید که static constructorها پارامتر و modifier ندارند.

```
static class Class1
                    {

                    }

                public static class DistanceConverter
                                                    {
        public static double meterTocmeter(double meter)
                                                    {
                                        return (meter * 100);
                                                    }
    public static double cmeterTometer(double cmeter)
                                                    {
                                        return (cmeter / 100);
                                                    }
public static double cmeterTomilimeter(double cmeter)
                                                    {
                                         return (cmeter * 10);
                                                    }
public static double milimeterTocmeter(double milimeter)
                                                    {
                                        return (milimeter / 10);
                                                    }
                                                }
```

```
                                        double cm;
                                cm=DistanceConverter.meterTocmeter(2.5);
```

FaraDars.org فرادرس

کلمه کلیدی static در متد main



حتماً تا این لحظه‌ی متوجه این کلمه‌ی کلیدی static در متد Main() شده‌اید. از آن‌جا که متد Main() نقطه‌ی شروع برنامه‌تان و یکی از اعضای کلاس است، باید قبل از هرچیز و پیش از ساخت هرگونه شیئی، صدا زده شود. به این دلیل است که متد Main() را به‌صورت static تعریف می‌کنیم تا قبل از اینکه شیئی از کلاس ساخته شود، متد Main() فراخوانی شده تا درون این متد بتوانیم کنترل برنامه را در دست بگیریم.

```
static class Program
{
  static void Main()
  {
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new Form1());
  }
}
```

FaraDars.org

انواع روش ارسال پارامتر به متد

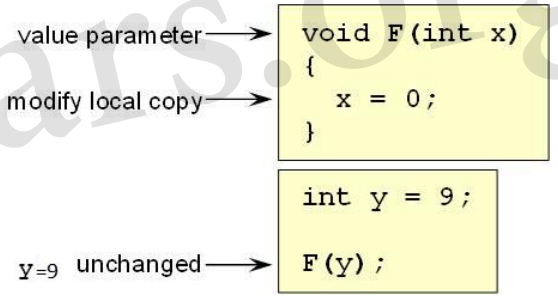
- Three ways to pass parameters

in	Pass by value
in out	Pass by reference
out	Output parameters

۱. روش ارسال با مقدار Pass by value

در این روش در پارامترهای تابع متغیرهای معمولی تعریف می‌کنیم و در زمان فراخوانی مقادیر ثابت یا اسامی متغیرهایی از فراخواننده را به عنوان آرگومان به پارامترهای متد ارسال می‌کنیم. هر تغییری برروی پارامترها در بدنه متد هیچ تأثیری برروی آرگومانهای فراخواننده ندارد و آرگومانها تنها به عنوان مقدار اولیه پارامترها در نظر گرفته می‌شوند. در این شکل ارسال پارامترها تنها به عنوان ورودی در نظر گرفته می شوند. ارسال به روش pass by value به صورت پیش فرض می‌باشد .

```
private void button1_Click(object sender, EventArgs e)
{
    AddUp(15, 5);
}

void AddUp(int firstNumber, int secondNumber)
{


}
```



FaraDars.org
فرادرس

انواع روش ارسال پارامتر به متد

۲. روش ارسال پارامتر با ارجاع Pass by reference

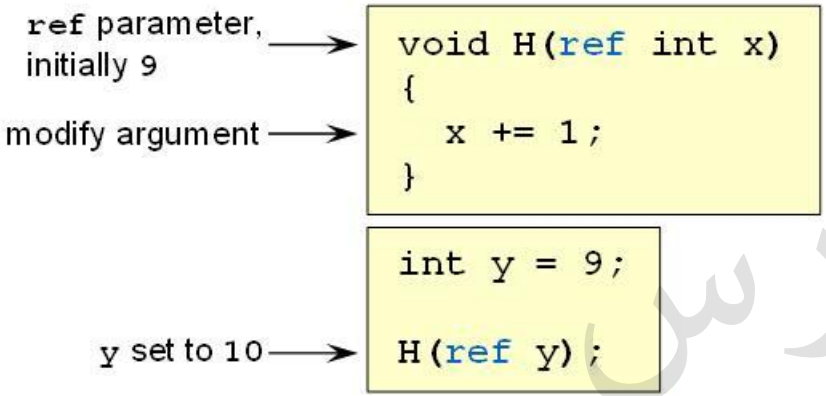
وقتی متغیری را به صورت ref به یک تابع ارسال می‌کنیم، مقدار متغیر ارسال نمی‌شود بلکه آدرس متغیر به بدنه متد فرستاده می‌شود و هر تغییری در متغیر محلی برروی متغیر اصلی نیز اعمال می‌شود. به این نوع پارمترها ارجاعی می‌گویند.

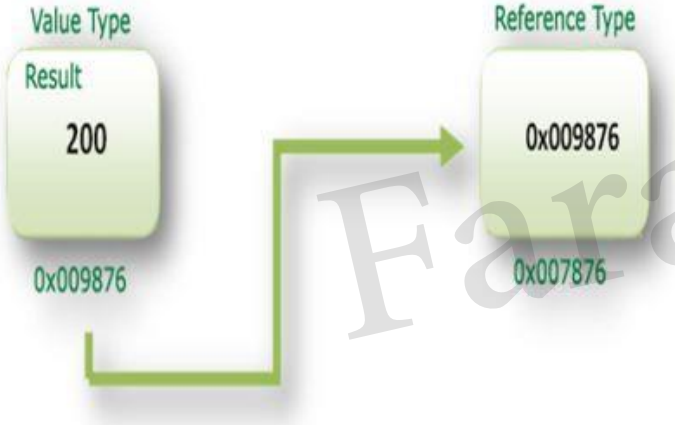
هر تغییری برروی پارامترها در بدنه متد برروی آرگومان‌ها در فراخواننده اعمال می شود.

نوشتن کلمه کلیدی ref هم درپشت آرگومان در زمان ارسال به متد و هم در پشت تعریف پارامتر در متد الزامی می‌باشد.

آرگومانی که به این روش به متد ارسال می‌شود باید مقداردهی اولیه شده باشد.

در این روش پارامترها به عنوان هم ورودی و هم خروجی در نظر گرفته می‌شوند.





FaraDars.org

انواع روش ارسال پارامتر به متد

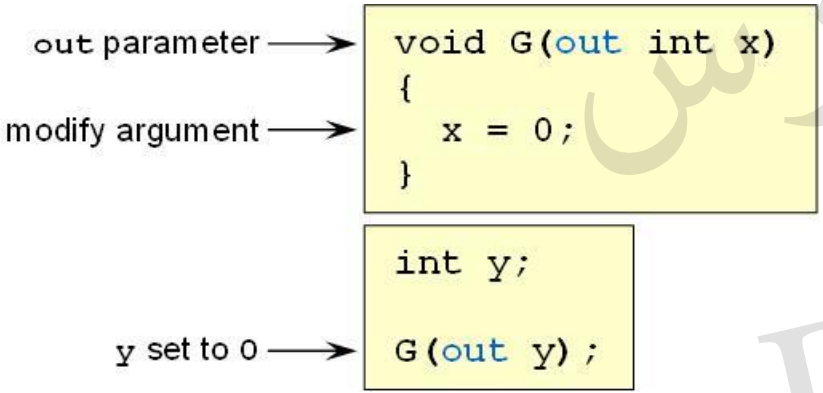
۳. روش ارسال پارامتر به عنوان تنها خروجی با کلمه کلیدی out

این شیوه ارسال هم به صورت ارجاعی می‌باشد. این پارامترها زمانی استفاده می‌شود که قصد ارسال اطلاعاتی به متد را نداشته باشیم، بلکه می‌خواهیم اطلاعات از متد برگردد.

نیازی به مقداردهی قبل از ارسال به متد نیست ولی حتماً باید درون متد (قبل از بازگشت به متدی فراخوانی شده) مقدار بگیرد.

در این روش پارامترها تنها به عنوان خروجی در نظر گرفته می‌شوند.

استفاده از ref و out تنها به فرستادن value typeها محدود نمی‌شود بلکه هنگام فرستادن reference type نیز می‌توانند مورد استفاده قرار گیرد.



FaraDars.org

استفاده از کلمه کلیدی params

گاهی از اوقات نیاز است تا تابعی تعریف کنیم که تعداد آرگومانهای آن متغیر باشند. برای این منظور از کلمه‌ی کلیدی params استفاده می‌شود.

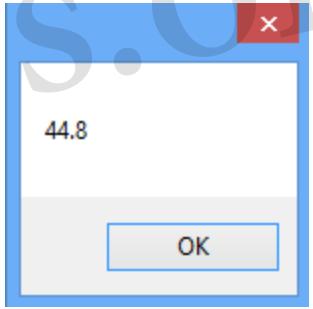
دو نکته در اینجا حائز اهمیت است:

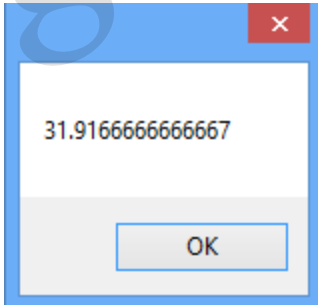
۱- در هر تابعی تنها می‌توان یکبار از params استفاده کرد.

۲- پس از بکار بردن params دیگر نمی‌توان هیچ آرگومانی را تعریف کرد.

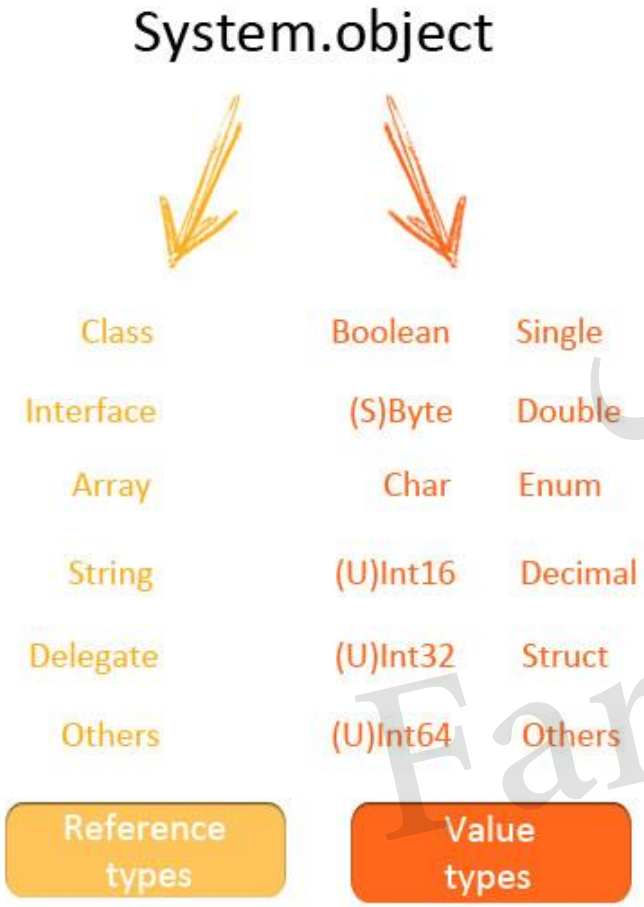
```
class testparams
{
public double Average(params double[] nums)
{
double result = 0;
for (int i = 0; i < nums.Length; i++)
result += nums[i];
return result / nums.Length;
}
}
```

```
private void button16_Click(object sender, EventArgs e)
{
double avg;
testparams c1 = new testparams();
avg=c1.Average(10, 17, 70,80,47);
MessageBox.Show(avg.ToString());
double[] List = { 15, 8, 6, 12, 77.5, 73 };
avg = c1.Average(List);
MessageBox.Show(avg.ToString());
}
```



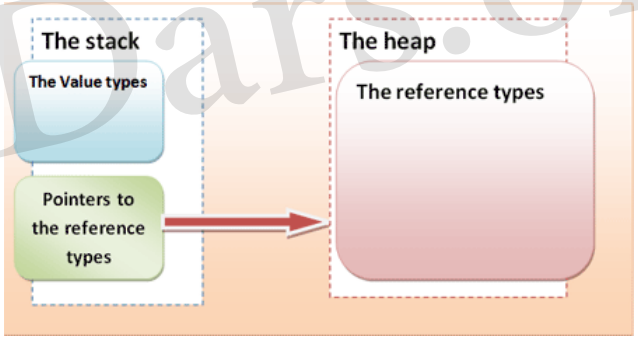


فرادرس
FaraDars.org



تفاوت Value type و Reference type

Value Typeها موقع فرستاده شدن به یک متد، بصورت پیش فرض به روش فراخوانی با مقدار ارسال می‌شوند و مقدار درون خود را درون آرگومان متد کپی می‌کنند و هرگونه تغییر درون آرگومان‌های متد، هیچ گونه تغییری در داده اصلی نخواهد داشت. اما Ref Typeها بصورت پیش فرض بصورت فراخوانی با ارجاع ارسال می‌شوند زیرا با نام خودشان که در واقع ارجاع به آن‌ها می‌باشد فرستاده می‌شوند. در نتیجه هرگونه تغییر درون متد، تأثیر بر روی داده اصلی خواهد گذاشت اگر memory یک برنامه را به دو قسمت تقسیم کنیم: stack و heap. در این صورت همه Value type در stack ذخیره می‌شوند ولی با این تفاوت که در Reference Typeها Instance آن در heap نگهداری می‌شود و در واقع چیزی که در stack خواهد بود چیزی نیست جز آدرس به heap یا همان Reference.



FaraDars.org

کلاس های بخشی یا partial

با استفاده از کلمه کلیدی partial می‌توانید اجزاء یک کلاس را در دو یا بیشتر از دو فایل جداگانه قرار دهید. مشروط بر اینکه فضای نام همه فایل‌ها یکسان باشد.

مزایای استفاده از کلاس‌های partial:

۱- افزایش خوانایی در کلاس‌های بسیار زیادی است.

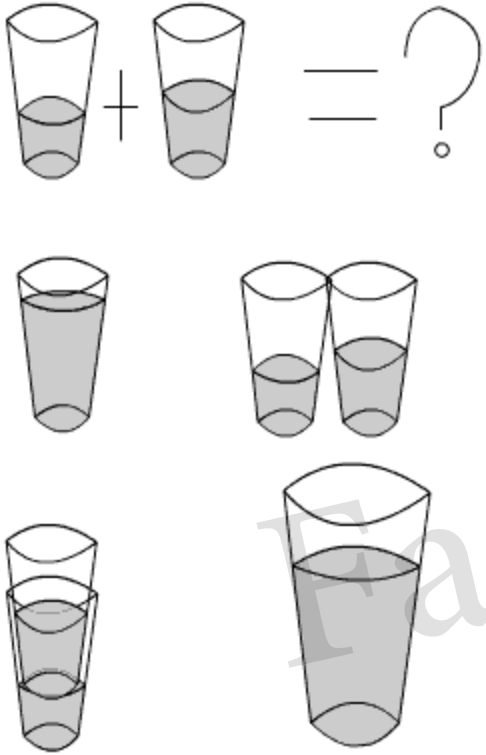
۲- اگر شما برروی کلاس پیچیده‌ای کار می‌کنید براحتی می‌توانید آن را بین چند نفر تقسیم کنید و هر کدام از افراد در فایلی جداگانه کدهای کلاس خود را می‌نویسد.

۳- شما براحتی می‌توانید مکان‌هایی که برخی کدها بصورت اتوماتیک تولید می‌شوند، مثل کد فرم‌های تحت ویندوز (Windows Forms, Web Service)، را از کدهایی که بصورت دستی می‌نویسید تفکیک کنید. در نتیجه اعمال تغییرات بسیار راحت تر می‌شود. بطور مثال وقتی یک فرم ویندوزی ایجاد می‌کنیم قسمت کدی که توسط Designer نوشته می‌شود و مربوط به ظاهر فرم هست در فایلی جداگانه وجود دارد که باعث می‌شود کدی که ما می نویسیم از آن کد جدا باشد و خواناتر باشد.

۴- در تعریف کلاس‌ها می‌توانیم بخش از کلاس‌ها که دارای خصوصیات private است را از بخشی که دارای خصوصیات public است تفکیک کنیم. این کار به خوانایی و روان فهمیدن کلاس کمک فراوانی می‌کند.

فرادرس
FaraDars.Org

بازنویسی عملگرها یا operator overloading



سی‌شارپ به شما اجازه می‌دهد operatorهایی تعریف کنید که مرتبط به کلاس‌هایی است که خودتان می‌سازید. به این پروسه operator overloading گفته می‌شود. با overload کردن یک operator شما کاربرد آن operatorرا به کلاس خودتان اضافه می‌کنید. تأثیری که این operator برروی کلاس شما می‌گذارد کاملاً تحت کنترل خودتان است و ممکن است برای هر کلاس متفاوت باشد. به‌عنوان مثال کلاسی که یک لیست پیوندی تعریف می‌کند، ممکن است از عملگر + برای افزودن یک شیء به انتهای لیست استفاده کند. کلاسی که stack را اجرا می‌کند، ممکن است از عملگر + برای افزودن یک شیء به بالای پشته استفاده کند. کلاسی دیگر ممکن است از عملگر + به‌طور کاملاً متفاوت استفاده کند.

هنگامی‌که یک عملگر overload می‌شود، معنای واقعی خودش را از دست نمی‌دهد. بلکه فقط کاربرد آن به یک کلاس افزوده می‌شود. بنابراین(به‌عنوان مثال) overload کردن عملگر + برای افزودن یک شیء به انتهای لیست پیوندی دلیل نمی‌شود که عملکرد آن operator برای جمع کردن دو عدد صحیح تغییر کند.

مزیت اصلی operator overloading این است که به شما اجازه می‌دهد به‌طور یکپارچه یک کلاس جدید را در محیط برنامه‌نویسی خود ادغام کنید. این ویژگی که به آن type extensibility می‌گویند، یکی از بخش‌های مهم یک زبان برنامه‌نویسی شی‌گرا مثل سی‌شارپ است. هنگامی‌که operatorها برای یک کلاس تعریف می‌شوند، می‌توانید آن operator را برروی اشیای کلاس مربوطه اعمال کنید. این نکته قابل ذکر است که operator overloading یکی از قدرمندترین ویژگی‌های سی‌شارپ است.

فرادرس
FaraDars.org

شکل کلی بازنویسی عملگرها

برای overload کردن یک عملگر، از کلمه‌ی کلیدی operator برای تعریف یک operator method استفاده می‌کنیم که برای یک عمل خاص مربوط به کلاس خودش تعریف می‌شود.

دو حالت از operator method وجود دارد:

1) unary operators عملگرهای تکی و 2) binary operators عملگرهای دوتایی. فرم کلی هرکدام را در زیر می‌بینید. در این‌جا عملگری که آن را overload می‌کنید، مثل + یا / جایگزین op می‌شود. ret-type مشخص کننده‌ی نوع مقداری است که return خواهد شد. اگرچه return type می‌تواند از هر نوعی باشد اما اغلب از نوع همان کلاسی است که operator در آن overload می‌شود. این ارتباط (یکسان بودن return type با جنس کلاس) باعث راحتی استفاده از عمگرهای overload شده می‌شود. برای unary operatorها عملوند در قسمت operand قرار می‌گیرد. برای binary operatorها، عملوندها در قسمت operand1و operand2 قرار خواهد گرفت. توجه داشته باشید که operator methodها باید هم publicو هم static باشند.

در unary operatorها، نوع عملوند(operand) باید با نوع کلاسی که operator در آن تعریف می‌شود یکسان باشد. بنابراین نمی‌توانید operatorهای سی‌شارپ را برای اشیایی که خودتان نساخته‌اید تعریف کنید. برای مثال، نمی‌توانید مجدداً عملگر + را برای intو string تعریف کنید. نکته‌ی دیگر اینکه operator parameters نباید از ref و out استفاده کنند.

```
// General form for overloading a unary operator
public static ret-type operator op(param-type operand)
{
    // operations
}

// General form for overloading a binary operator
public static ret-type operator op(param-type1 operand1, param-type1 operand2)
{
    // operations
}
```

FaraDars.org فرادرس

مثال برای بازنویسی عملگرها

```
class mydate
{
    private int _year;
    private int _month;
    private int _day;
    private static int[] endofmonth = { 0, 31, 31, 31, 31,31,31,30,30,30,30,30,29};
    6 references
    public mydate(int d = 1, int m = 1, int y = 1394)...
    4 references
    public mydate(mydate x)...
    17 references
    public int Year ...
    17 references
    public int Month...
    13 references
    public int Day...
    2 references
    private bool IsEndofMonth()...
    5 references
    private void increment()...
    0 references
    public static mydate operator+(mydate x, int n)...
    0 references
    public static mydate operator +( int n, mydate x)...
    1 reference
    public static mydate operator++(mydate x)...
    1 reference
    public static bool operator >(mydate x, mydate y)...
```

فرادرس
FaraDars.org

بازنویسی عملگر+

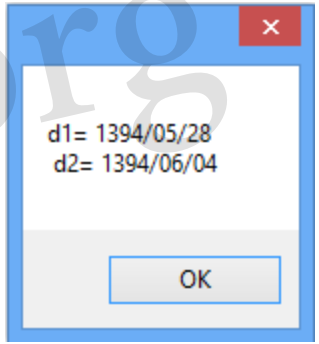
دقت کنید خط سوم برنامه دارای خطا می باشد. این خطا به این دلیل است که عمگر + برای نوع کلاس Mydate تعریف نشده است. می توانیم عملگر + برای کلاس Mydate به شکل زیر بازنویسی کنیم. خطا برطرف می شود و خروجی برنامه را مشاهده می کنیم.

```
private void button2_Click(object sender, EventArgs e)
{
    mydate d1 = new mydate(28, 5, 1394);
    mydate d2 = new mydate();
    d2 = d1 + 7;
    Messa                                              oString());
}
```

(local variable) mydate d1

Error:
Operator '+' cannot be applied to operands of type 'operator_overloading2.mydate' and 'int'

```
public static mydate operator +(mydate x, int n)
                                {
                    mydate t = new mydate(x);
                for (int i = 0; i < n; ++i)
                                    t.increment();
                                    return t;
                                    }
```



فرادرس
FaraDars.org

ادامه بازنویسی عملگر+

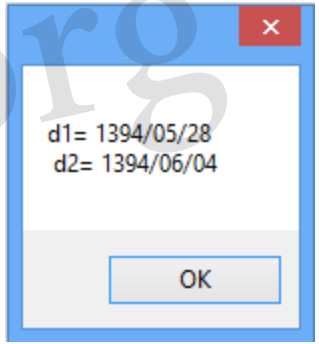
بازنویسی عملگر+ خطای موجود در برنامه قبلی را برطرف کرد اما اگر برنامه نویس از عملگر + به شکل مقابل استفاده نماید مجدد برنامه خطا می دهد برای رفع مشکل عملگر + را مجدد به شکل زیر بازنویسی می کنیم. بنابراینبا ابن روش به این عملگر خاصیت جابجا پذیری داده می شود.

```
private void button3_Click(object sender, EventArgs e)
{
    mydate d1 = new mydate(28, 5, 1394);
    mydate d2 = new mydate();
    d2 = 7 + d1;
    Messa                                          oString());
}
```

struct System.Int32
Represents a 32-bit signed integer.

Error:
Operator '+' cannot be applied to operands of type 'int' and 'operator_overloading2.mydate'

```
public static mydate operator +(int n, mydate x)
                                                {
                                    mydate t = new mydate(x);
                                for (int i = 0; i < n; ++i)
                                                    t.increment();
                                                return t;
                                                }
```



فرادرس
FaraDars.org

بازنویسی عملگرها رابطه ای

عملگرهای رابطه‌ای(Relational Operators) مثل == یا > می‌توانند به‌سادگی overload شوند. به‌طور معمول، یک عملگر رابطه‌ای overload شده مقدار true یا false را return می‌کند، به‌این دلیل که حالت و کاربرد استاندارد عملگرهای رابطه‌ای حفظ شود و بتوان از آن‌ها در عبارت‌های شرطی استفاده کرد. اگر در این موارد به‌جای مقادیر bool چیز دیگری را return کنید، به شدت کاربرد این operator را محدود کرده‌اید. نکته‌ی مهم دیگر این‌جاست که بایستی relational operators را به‌طور جفتی overload کنید. به‌عنوان مثال اگر > را overload کردید، بایستی < را نیز overload کنید. این مورد برای operatorهای (<= >=)و (=! ==)نیز صادق است.

```
public static bool operator >(mydate x, mydate y)
{
if (x.Year > y.Year||(x.Year==y.Year && x.Month>y.Month)||(x.Year==y.Year && x.Month==y.Month && x.Day>y.Day))
return true;
else
return false;
}
```

```
public static bool operator <(mydate x, mydate y)
{
if (x.Year < y.Year||(x.Year==y.Year && x.Month<y.Month)||(x.Year==y.Year && x.Month==y.Month && x.Day<y.Day))
return true;
else
return false;
}
```

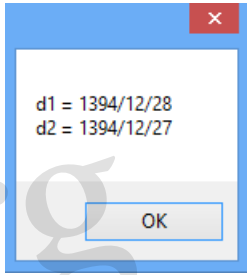
FaraDars.org فرادرس

بازنویسی عملگرهای ++ و --

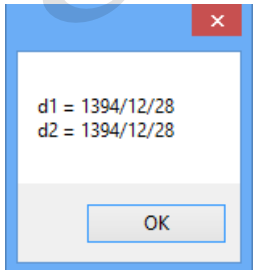
در سی‌شارپ overload کردن ++ و -- بسیار آسان است. کافی است که مقدار را افزایش یا کاهش دهید و آن را return کنید اما نباید مقدار شیء operandرا تغییر دهید. سی‌شارپ به‌طور خودکار حالت‌های postfixو prefix را برای شما در نظر می‌گیرد.

```
public static mydate operator++(mydate x)
                                    {
                        mydate t = new mydate(x);
                                    t.increment();
                                         return t;
                                    }
```

```
                    private void button4_Click(object sender, EventArgs e)
                                                                        {
                                                mydate d1 = new mydate(27, 12, 1394);
                                                        mydate d2 = new mydate();
                                                                        d2 = d1++;
            MessageBox.Show("d1 = " + d1.ToString() + "\nd2 = " + d2.ToString());

                                                                        }
```




```
                    private void button4_Click(object sender, EventArgs e)
                                                                        {
                                                mydate d1 = new mydate(27, 12, 1394);
                                                        mydate d2 = new mydate();
                                                                        d2 = ++d1;
            MessageBox.Show("d1 = " + d1.ToString() + "\nd2 = " + d2.ToString());

                                                                        }
```



فرادرس
FaraDars.org

بازنویسی تبدیل نوع یا conversion

برخی مواقع می‌خواهید از شیء یک کلاس در عبارتی استفاده کنید که شامل data typeهای دیگری نیز است. در بعضی موارد overload کردن یک یا چند operator می‌تواند این کار را برای شما انجام دهد. اما گاهی چیزی که شما نیاز دارید یک تبدیل ساده از نوع کلاس به نوع مورد نظرتان است. برای انجام این دسته از موارد، سی‌شارپ به شما اجازه می‌دهد نوع خاصی از operator method را بسازید که conversion operator نامیده می‌شود. Conversion operator یک شیء از کلاس شما را به نوع دیگری که مد نظرتان است تبدیل می‌کند. دو حالت از conversion operator موجود است: implicitو explicit که فرم کلی آن‌ها به شکل زیر است.

در این‌جا target-type مشخص کننده‌ی نوعی است که قصد دارید source-type را به آن تبدیل کنید و value مقدار کلاس بعد از تبدیل است. Conversion operator اطلاعات را مطابق با target-type باز می‌گرداند(return می‌کند).

اگر conversion operator به‌طور implicit مشخص شود، بنابراین conversion به‌صورت اتوماتیک انجام خواهد شد، مثل حالتی که شیء در یک عبارت همراه با یک data type دیگری از نوع target-type در تعامل است. هنگامی‌که conversion به‌صورت explicit تعریف شده باشد، بنابراین هنگامی که cast مورد نیاز است conversion فراخوانی می‌شود. توجه کنید که نمی‌توانید برای یک source-typeو target-type هم implicitو explicit را تعریف کنید. اگر conversion را به‌طور explicit تعریف کنید، تبدیل به‌صورت اتوماتیک انجام نمی‌شود و cast مورد نیاز است. علاوه‌بر این قوانین، برای انتخاب بین implicit یا explicit باید دقت کنید. implicit conversion باید زمانی مورد استفاده قرار گیرد که تبدیل کاملاً عاری از خطا باشد. برای کسب اطمینان در این مورد از این دو قانون پیروی کنید: اولاً هیچ فقدان اطلاعاتی (مثل کوتاه‌سازی، سرریز، تغییر علامت و...) نباید رخ دهد. ثانیاً تبدیل نباید باعث بروز exception یا خطا در برنامه شود. اگر conversion نتواند این دو قانون را رعایت کند، باید از explicit conversion بهره ببرید.

```
1 public static operator implicit target-type(source-type v) { return value; }
2 public static operator explicit target-type(source-type v) { return value; }
```

FaraDars.org

بازنویسی تبدیل نوع برای مثال Mydate

```
public static explicit operator int(mydate x){
                                        int n=0;
                                for(int i=1;i<=x.Month;++i)
                                        n=n+endofmonth[i];
                                        return n + x.Day;
                                        }

public static implicit operator string(mydate x)
                                        {
                                return x.ToString();
                                        }
```

اگر conversion را به‌طور explicit تعریف کنید، تبدیل به‌صورت اتوماتیک انجام نمی‌شود و cast مورد نیاز است.

- نمی‌توانید برای یک source-type و target-type هم تبدیل implicit و هم تبدیل explicit تعریف کنید.
- نمی‌توانید class type را به نوع داده‌ی object تبدیل کنید.

type یا source-type در conversion بایستی از جنس همان کلاسی باشد که conversion در آن تعریف شده است. برای مثال نمی‌توانید تبدیل double به int را از نو تعریف کنید.

.

```
private void button4_Click(object sender, EventArgs e)
                                        {
                                        int a;
                        mydate d1 = new mydate(22, 7, 1393);
                                        a = (int)d1;
                                MessageBox.Show(a.ToString());
                                        }
```

```
private void button5_Click(object sender, EventArgs e)
                                        {
                        mydate d = new mydate(22, 4, 1392);
                                        string s1 = d;
                                MessageBox.Show(s1);
                                        }
```

فرادرس
FaraDars.org

بازنویسی کردن true و false

کلمات کلیدی trueو falseنیز می‌توانند به‌عنوان unary operators به‌منظور overload کردن مورد استفاده قرار گیرند. نسخه‌ی overload شده‌ی این operatorها با توجه به کلاسی که شما می‌سازید شخصی‌سازی می‌شود. هنگامی‌که trueو false برای یک کلاس overload می‌شوند، می‌توانید از اشیای آن کلاس برای کنترل کردن if، for، whileو do-while و همچنین ? استفاده کنید.
Operatorهای trueو false باید باهم overload شوند و نمی‌توانید فقط یکی از آن‌ها را overload کنید. هر دوی آن‌ها unary operator هستند و فرم کلی آن‌ها به‌صورت زیر است. دقت کنید که هریک مقدار bool را return می‌کند.

```
public static bool operator true(param-type operand)
{
    // return true or false
}
public static bool operator false(param-type operand)
{
    // return true or false
}
```

فرادرس
FaraDars.org

مثال بازنویسی کردن true و false

فرض کنید بخواهیم true و false را به نحوی برای کلاس Mydate بازنویسی کنیم که true بودن یک تاریخ به معنی روز اول سال نو بودن باشد و false به معنی اینست که شی تاریخ، معادل روز اول سال نیست. همانطور که مشاهده می کنید بدون بازنویسی true وfalse برنامه خطا دارد اما در صورتی که درون کلاس Mydate عملگرهای True وfalse را به شکل زیر بازنویسی کنیم خطلا برطرف می گردد.

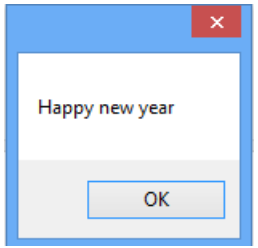
```
private void button5_Click(object sender, EventArgs e)
{
    mydate d1 = new mydate(1, 1, 1395);
    if (d1)

}
```

(local variable) mydate d1

Error:
Cannot implicitly convert type 'operator_overloading2.mydate' to 'bool'

```
public static bool operator true(mydate x)
                                        {
        return x.Day == 1 && x.Month == 1;
                                        }

public static bool operator false(mydate x)
                                        {
      return !(x.Day == 1 && x.Month == 1);
                                        }
```

```
private void button5_Click(object sender, EventArgs e)
                                                {
mydate d1 = new mydate(1, 1, 1395);
                                        if (d1)
MessageBox.Show("Happy new year");
                                                }
```



FaraDars.org
فرادرس

موارد زیر را هنگام سربارگذاری عملگرها به خاطر داشته باشید

Operators	Overloadable
Unary operators +, -, !, ~, ++, —, true, false	Yes
Binary operators +, -, *, /, %, &, |, ^, <<, >>	Yes
Comparison operators ==, !=, <, >, <=, >=	Yes, in pairs (e.g., if < is overloaded, > must be too, and vice versa. Note that if you overload == or !+, you must override the Object.Equals and Object.GetHashCode functions.)
Conditional logical operators &&, ||	No, but they are evaluated using & and |, which can be overloaded
Array-indexing operators []	No, but you can define indexers
Cast operator ()	No, but you can define new conversion operators
Assignment operators +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=	No, but +=, for example, is evaluated using +, which can be overloaded
Miscellaneous operators =, ., ?:, ->, new, is, sizeof, typeof	No

1- تنها عملگرهای ذکر شده در ادامه را مي توان overloadکرد.

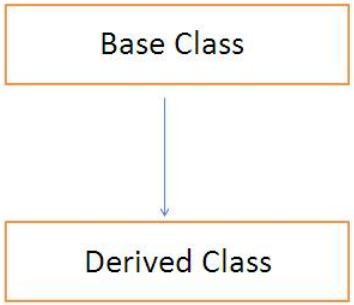
Unary Operators :
+ - ! ~ ++ -- true false

Binary Operators
+ - * / % & | ^ << >> == != > < >= <=

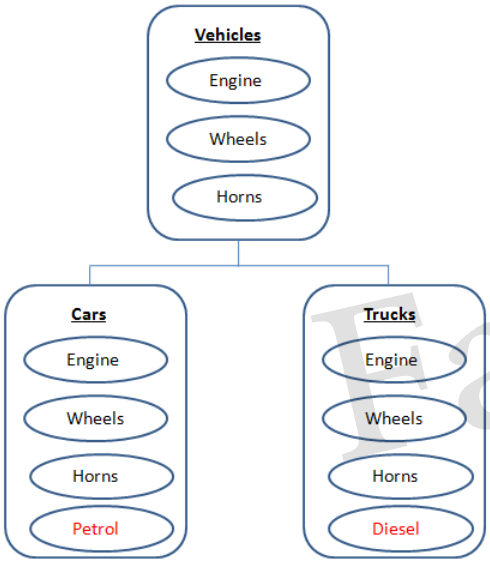
2- خروجي متدهاي بكار گرفته شده در سربارگذاري عملگر ها نمي تواند voidباشد.
3- حداقل يكي از آرگومانهاي بكار گرفته شده در متدي كه براي overloading عملگرها بكار مي رود بايد از نوع كلاس حاوي متد باشد.
4- متدهاي مربوطه بايد به صورت publicو staticتعريف شوند.
5- هنگامي كه اپراتور > را سربارگذاري مي كنيد بايد جفت متناظر آن يعني < را هم سربارگذاري نماييد.
6- هنگاميكه براي مثال + را overloadمي كنيد خودبخود += نيز overloadشده است و نيازي به كدنويسي براي آن نيست.

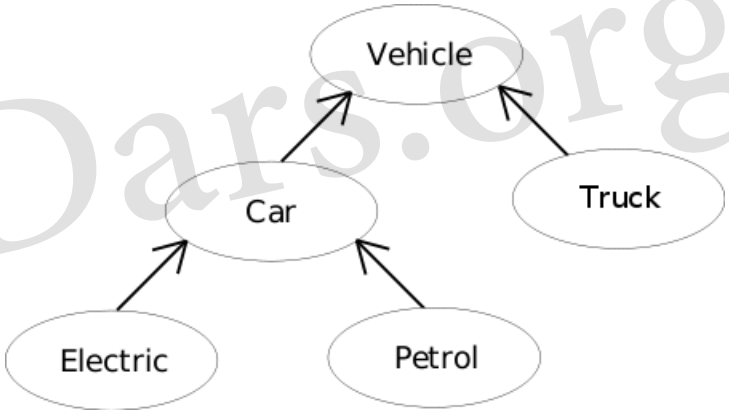
فرادرس FaraDars.org

وراثت، ارث‌بری Inheritance



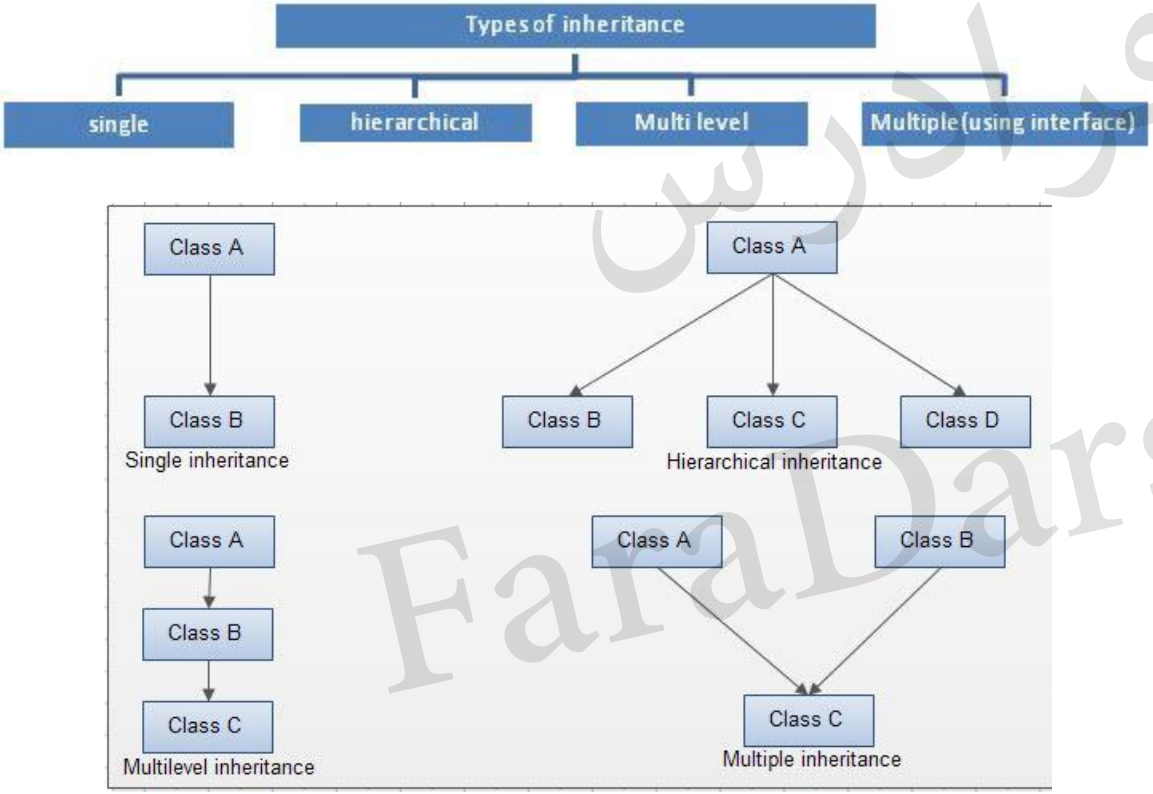
Inheritance یکی از اصول بنیادی برنامه‌نویسی شی‌گرا است که موجب ساخت کلاس‌ها به‌صورت سلسله‌مراتبی می‌شود. همه‌ی زبان‌های برنامه‌نویسی شی‌گرا به دلایل یکسانی از inheritance استفاده می‌کنند. با استفاده از inheritance می‌توانید یک کلاس کلی با یک سری ویژگی تعریف کنید که این ویژگی‌ها می‌توانند در چند بخش مرتبط باهم مشترک باشند. این کلاس کلی می‌تواند توسط کلاس‌های دیگر ارث‌بری شود و مواردی که یکتاست را در اختیار آن‌ها قرار دهد. در زبان سی‌شارپ کلاسی که از آن ارث‌بری می‌شود، base class (کلاس پایه) نام دارد و کلاسی که ارث‌بری را انجام می‌دهد derived class (کلاس مشتق شده) نامیده می‌شود. از این‌رو derived class نسخه‌ی اختصاصی شده‌ی base class است. derived class تمام variableها، methodها، propertyها و indexerهای تعریف شده در base class را به ارث می‌برد و در کنار این‌ها عناصر مخصوص به خود را نیز اضافه می‌کند.





FaraDars.org

Types of Inheritance in C# انواع وراثت

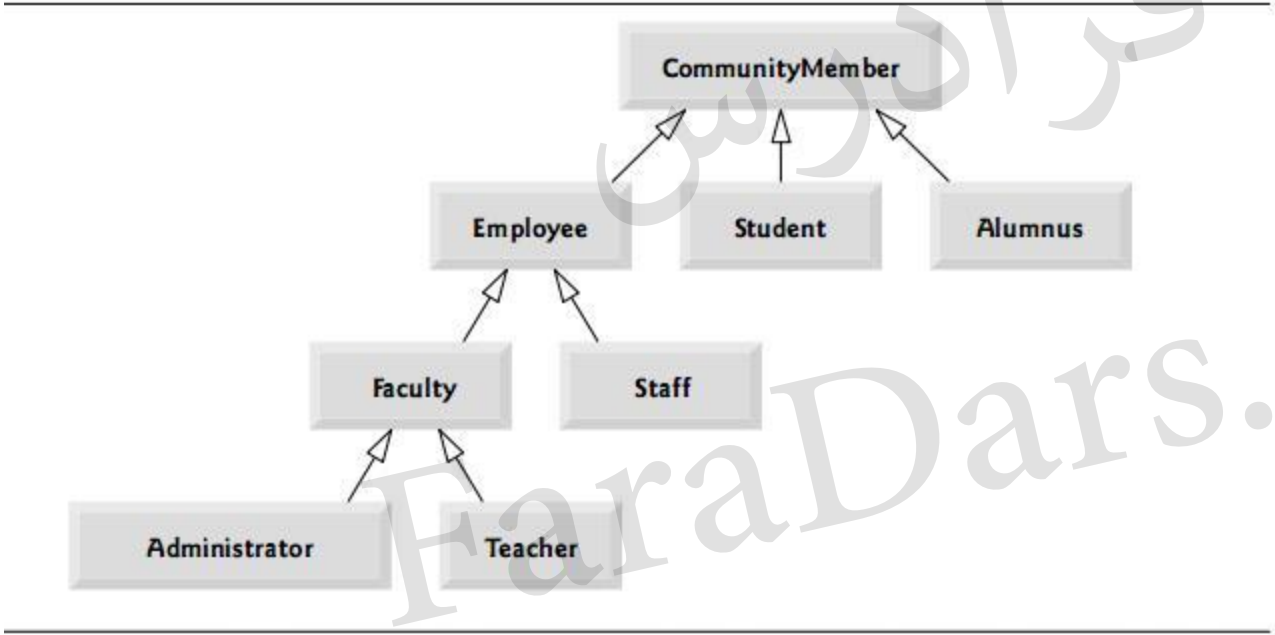


انواع وراثت در سی شارپ:

- وراثت یگانه Single Inheritance
- وراثت چند سطحی Multilevel Inheritance
- وراثت سلسه مراتبی Hierarchical Inheritance
- وراثت چندگانه Multilevel Inheritance (using interface)

فرادرس FaraDars.org

وراثت، ارث‌بری Inheritance



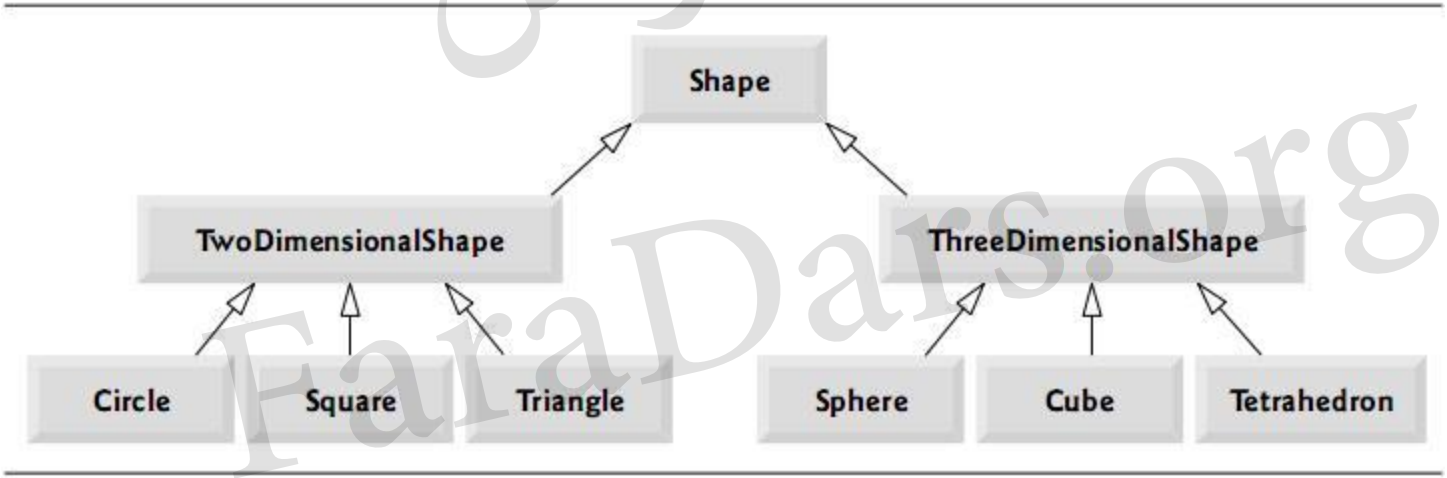
class diagram showing an inheritance hierarchy for university CommunityMembers.

یک جامعه دانشگاهی را با صدها عضوی که دارد در نظر بگیرید. این اعضا متشکل از کارمندان، فارغ‌التحصیلان و دانشجویان هستند. کارمندان می‌توانند اعضای هیئت علمی باشند یا کارمند ساده. اعضای هیئت علمی می‌توانند مدیر یا استاد باشند. در هر فلش این سلسله مراتب، رابطه وجود داشتن برقرار است. برای مثال، اگر فلش‌ها را دنبال کنیم، متوجه می‌شویم که Employee یک CommunityMember است، یا Teacher یک Faculty است. در واقع CommunityMember، کلاس مبنای مستقیم برای Employee، Student و Alumnus است. علاوه بر این، CommunityMember یک کلاس مبنای غیرمستقیم برای تمام دیگر کلاس‌ها در دیاگرام سلسله مراتب است.

فرادرس
FaraDars.org

سلسله مراتب ارث‌بری Shape

این سلسله مراتب با کلاس مبنای Shape آغاز می‌شود. کلاس‌های TwoDimensionalShape(شکل‌های دوبعدی) و ThreeDimensionalShape(شکل‌های سه‌بعدی) از کلاس مبنای Shape مشتق شده‌اند. شکل‌ها یا دوبعدی یا سه‌بعدی هستند. سطح سوم این سلسله مراتب حاوی برخی از انواع مشخص از اشکال دوبعدی و سه‌بعدی است. در این سلسله مراتب کلاس، چندین رابطه is-a وجود دارد. برای نمونه، یک Triangle(مثلث) یک شکل دوبعدی و یک شکل است (shape)، در حالیکه یک Sphere(کره) یک شکل سه‌بعدی و یک شکل است. توجه کنید که این سلسله مراتب می‌توانست حاوی کلاس‌های دیگری همانند مستطیل‌ها، بیضی‌ها و ذوزنقه‌ها باشد که همگی شکل‌های دوبعدی هستند.

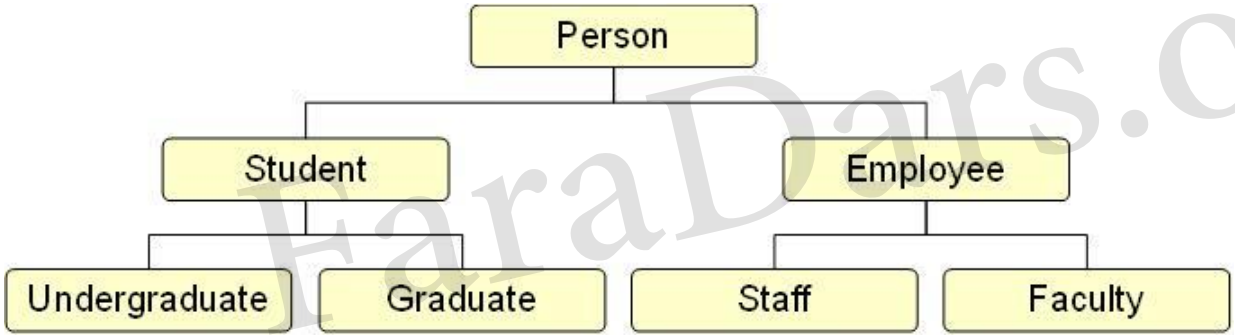


class diagram showing an inheritance hierarchy for Shapes.

FaraDars.org
فرادرس

مثال کاربردی از وراثت

Inheritance یکی از اصول بنیادی برنامه‌نویسی شی‌گرا است که موجب ساخت کلاس‌ها به‌صورت سلسله‌مراتبی می‌شود. همه‌ی زبان‌های برنامه‌نویسی شی‌گرا به دلایل یکسانی از inheritance استفاده می‌کنند. با استفاده از inheritanceمی‌توانید یک کلاس کلی با یک سری ویژگی تعریف کنید که این ویژگی‌ها می‌توانند در چند بخش مرتبط باهم مشترک باشند. این کلاس کلی می‌تواند توسط کلاس‌های دیگر ارث‌بری شود و مواردی که یکتاست را در اختیار آن‌ها قرار دهد. در زبان سی‌شارپ کلاسی که از آن ارث‌بری می‌شود، base class (کلاس پایه) نام دارد و کلاسی که ارث‌بری را انجام می‌دهد derived class (کلاس مشتق شده) نامیده می‌شود. از این‌رو derived class نسخه‌ی اختصاصی شده‌ی base classاست. derived class تمام variableها، methodها، propertyها و indexerهای تعریف شده در base class را به ارث می‌برد و در کنار این‌ها عناصر مخصوص به خود را نیز اضافه می‌کند.



FaraDars.org

ایجاد کلاس ها بدون استفاده از وراثت

در صورتیکه برای موجودیت دانشجو و کارمند کلاس‌های مجزا ایجاد کنیم، بخش زیادی از کد تکراری می‌باشد. این موضوع موجب دوباره‌کاری و همچنین هزینه برشدن نگهداری سیستم می‌شود.

repeated code

```
class Student
{
  string name;
  int    age;
  public void Birthday() { age++; }

  int id;
}
```

```
class Employee
{
  string name;
  int    age;
  public void Birthday() { age++; }

  double salary;
}
```

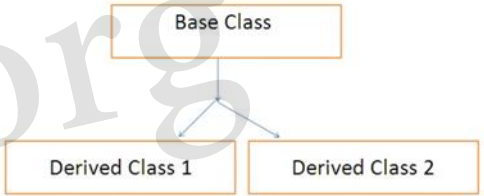
فرادرس
FaraDars.org

وراثت، ارث‌بری Inheritance

همانطور که مشاهده می‌کنید می‌توانیم یک کلاس Person ایجاد کنیم و مشترکات موجودیت‌ها را در آن قرار دهیم و کلاس‌های Student و Employee آن را به ارث ببرند و تنها بخش‌های اختصاصی خود را به آن اضافه کنند. در صورتیکه به کد کلاس‌های مشتق دقت کنید می‌بینید که بعد از نام کلاس با یک : نام کلاس پایه ذکر شده و به این روش ارث بری انجام می‌شود.

base class → common code

```
class Person
{
  string name;
  int    age;

  public void Birthday()
  {
    age++;
  }
}
```




```
class Student : Person
{
  int id;
}
```

← derived classes →

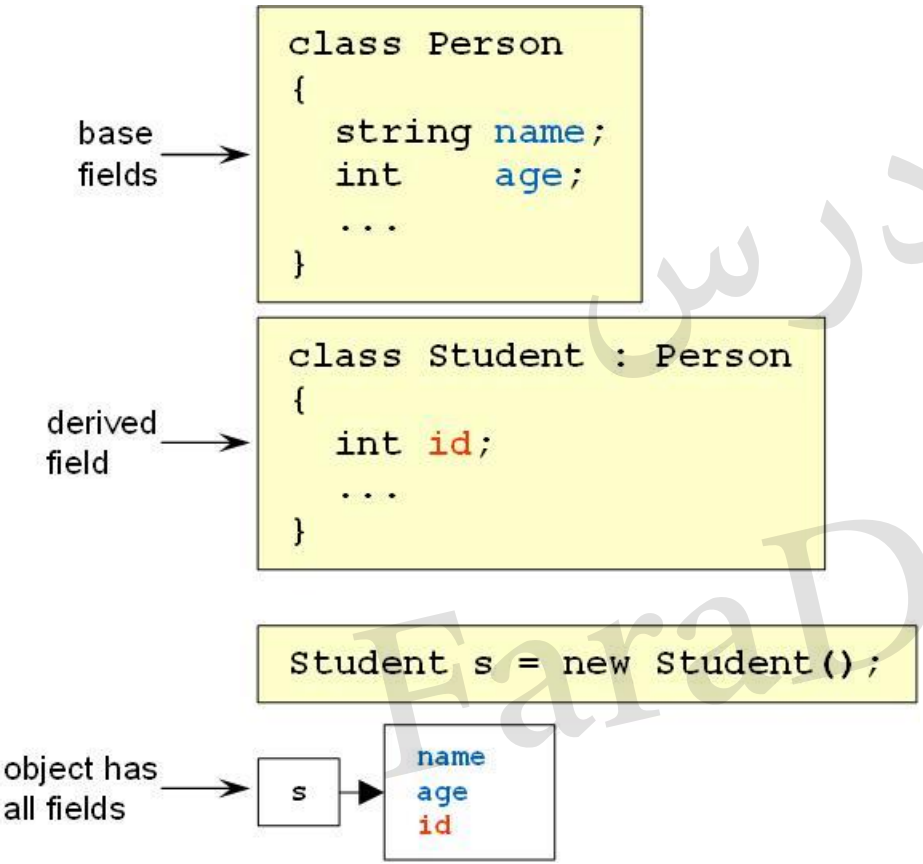
← derived members →

```
class Employee : Person
{
  double salary;
}
```

فرادرس
FaraDars.org

تخصیص حافظه در وراثت

اشیائی از نوع مشتق تمام اجزا کلاس پایه را نیز دارند.



فرادرس
FaraDars.org

Multi level Inheritance وراثت چند سطحی

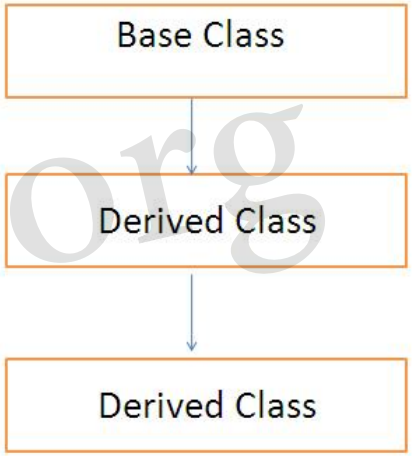
ارث بری می‌تواند چند سطحی باشد. به عنوان مثال کارمند کلاس شخص را به ارث ... کلاس عضو هیئت علمی کلاس کارمند را به ارث می برد.

```
class Person
{
   ...
}
```

```
class Employee : Person
{
   ...
}
```

inherits from all ancestors →

```
class Faculty : Employee
{
   ...
}
```



فرادرس
FaraDars.org

دسترسی به اجزاء اختصاصی کلاس پایه به کمک متد های عمومی کلاس پایه از یک شی کلاس مشتق

از طریق اشیائی از نوع مشتق می‌توانیم متدهای عمومی کلاس پایه را فراخوانی کنیم و متدهای عمومی کلاس پایه نیز می‌توانند به همه اجزای کلاس پایه دسترسی داشته باشند(حتی اجزای اختصاصی) بنابراین در خارج از کلاس مشتق به کمک متدهای عمومی کلاس پایه از یک شی کلاس مشتق می‌توانیم به همه اجزاء کلاس پایه دسترسی داشته باشیم.

base method →

```
class Person
{
  public void Birthday()
  {
    age++;
  }
  ...
}
```

derived object →

call base class method →

```
Student s = new Student();

s.Birthday();

...
```

فرادرس
FaraDars.org

Constructor and inheritance

در سلسله‌مراتب ارث‌بری، هم base classها و هم derived classها می‌توانند constructor خودشان را داشته باشند. در این‌جا این سوال به‌وجود می‌آید که کدام constructor مسئول ساختن شیء derived class است؟ آن‌که در base class است یا آن‌که در derived class قرار دارد؟ یا هردو؟ در واقع constructorای که در base class قرار دارد، بخش base class یک شیء و constructorای که در derived class واقع است، قسمت derived class را می‌سازد. اگر توجه کنید متوجه می‌شوید که این کار منطقی است زیرا base class هیچ دسترسی و اطلاعی از عناصر درون derived class ندارد و از این‌رو constructor آن‌ها باید جداگانه باشد. در مثال‌های قبلی از default constructor که به‌صورت اتوماتیک توسط سی‌شارپ ساخته می‌شوند استفاده شده است اما در عمل بیشتر کلاس‌ها constructor تعریف می‌کنند.

base fields likely require initialization →

```
class Person
{
  string name;
  int    age;
  ...
}
```

base supplies constructor to perform initialization →

```
class Person
{
  string name;
  int    age;

  public Person(string name, int age)
  {
    this.name = name;
    this.age  = age;
  }
  ...
}
```

base fields →

```
class Person
{
  string name;
  int    age;
  ...
}
```

derived field →

```
class Student : Person
{
  int id;
  ...
}
```

construct Student →

```
Student s = new Student("Ann", 23, 12345);
```

call Person constructor →

```
class Person
{
  public Person(string name, int age) { ... }
  ...
}

class Student : Person
{
  int id;

  public Student(string name, int age, int id)
  :base(name, age)
  {
    this.id = id;
  }
}
```

FaraDars.org
فرادرس

فراخوانی constructorهای base class

یک derived class می‌تواند constructorای که در base classاش تعریف شده است را از طریق گسترش دادن فرم constructor در derived class و کلمه‌ی‌کلیدی base، صدا بزند.
فرم کلی تعریف گسترش یافته‌ی آن به‌شکل زیر است:

```
derived-constructor(parameter-list) : base(arg-list)
{
// body of constructor
}
```

در این‌جا، arg-list مشخص‌کننده‌ی argumentهای مورد نیاز constructor در base class است. به نحوه‌ی قرار گرفتن colon نیز توجه داشته باشید.

```
class Person
{
  public Person(string name, int age) { ... }
  ...
}
```

```
class Student : Person
{
  int id;

  public Student(string name, int age, int id)
  :base(name, age)
  {
    this.id = id;
  }
}
```

call Person constructor →

هنگامی‌که یک derived class از کلمه‌ی‌کلیدی base استفاده می‌کند، base مستقیماً به نزدیک‌ترین base class بالای derived class مربوط می‌شود. از این‌رو، هنگامی‌که از سلسله‌مراتب ارث‌بری استفاده می‌کنید، base به نزدیک‌ترین base class در این زنجیره رجوع خواهد کرد. اگر از base استفاده نکنید، constructor پیش‌فرض base class اجرا خواهد شد.

فرادرس
FaraDars.org

No Multi Inheritance

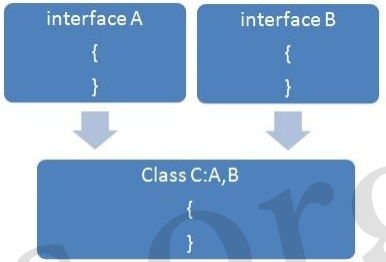
وراثت چندگانه از کلاس‌ها در زبان سی شارپ پشتیبانی نمی‌شود. البته یک کلاس امکان ارث بری از تعداد نامحدود واسط‌ها را دارد. پشتیبانی نکردن از وراثت چندگانه به دلیل اهداف معماری این زبان در CLI و برای جلوگیری از پیچیدگی است. در عوض می‌توان از Interfaceهای مختلف استفاده کرد. یعنی برای یک کلاس که احتمالا مشتق کلاسی دیگر است می‌توان چندین واسط را پیاده سازی(Implement) نمود.

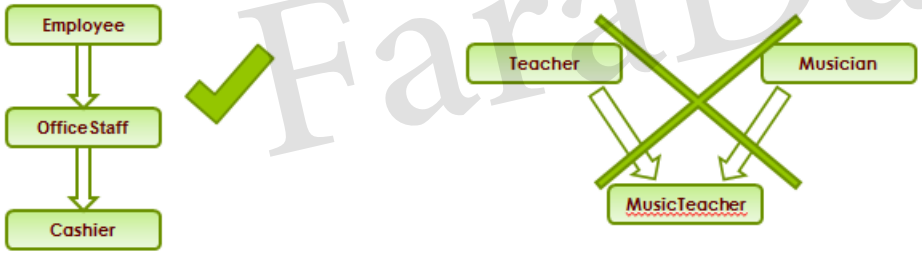
```
class Student : Person
{
   ...
}
```

```
class Employee : Person
{
   ...
}
```

error, only one base allowed →

```
class Graduate : Student, Employee
{
   ...
}
```



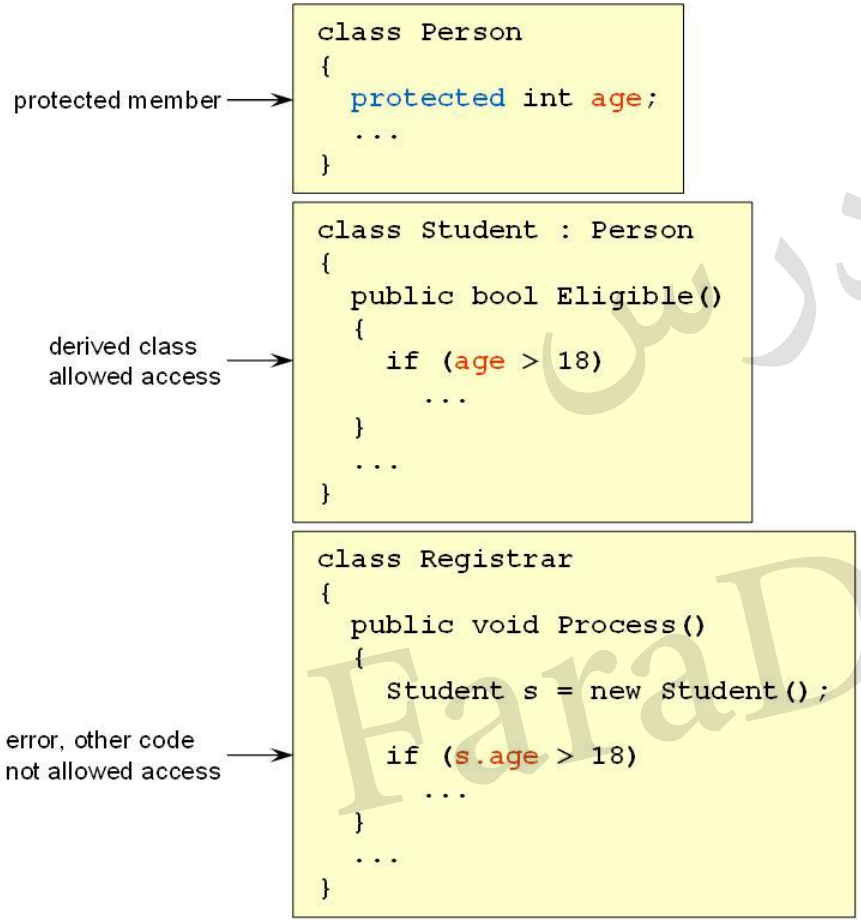



```
1 reference
interface Teacher
{
  //interface body
}
1 reference
interface musician
{
    //interface body
}
0 references
class MusicTeacher :Person ,musician, Teacher
{
   //class body
}
```

FaraDars.org

Protected

دسترسی protected عرضه کننده یک سطح حفاظتی میانی مابین دسترسی‌های public و private است. اعضای یک کلاس مبنای protected می‌توانند فقط در کلاس مبنا یا در هر کلاس مشتق شده از آن کلاس در دسترس قرار گیرند.



فرادرس
FaraDars.org

Hiding inherited member

base class field →
base class method →

```
class Base
{
  public int Field;

  public void Method()
  {
    ...
  }
}
```

field with same name →
method with same signature →

```
class Derived : Base
{
  public int Field;

  public void Method()
  {
    ...
  }
}
```

government assigns id to each person →
set method for person id →

```
class Person
{
  int id;

  public void SetId(int id)
  {
    this.id = id;
  }
  ...
}
```

Student derived from Person →

```
class Student : Person
{
}
```

Student object →
invokes Person class SetId method →

```
Student bob = new Student();

bob.SetId(12345);

...
```

university assigns id to each student →
set for student id →

```
class Student : Person
{
  int id;

  public void SetId(int id)
  {
    this.id = id;
  }
}
```

Student now has two SetId methods →
which one called? →

```
Student bob = new Student();

bob.SetId(12345);

...
```

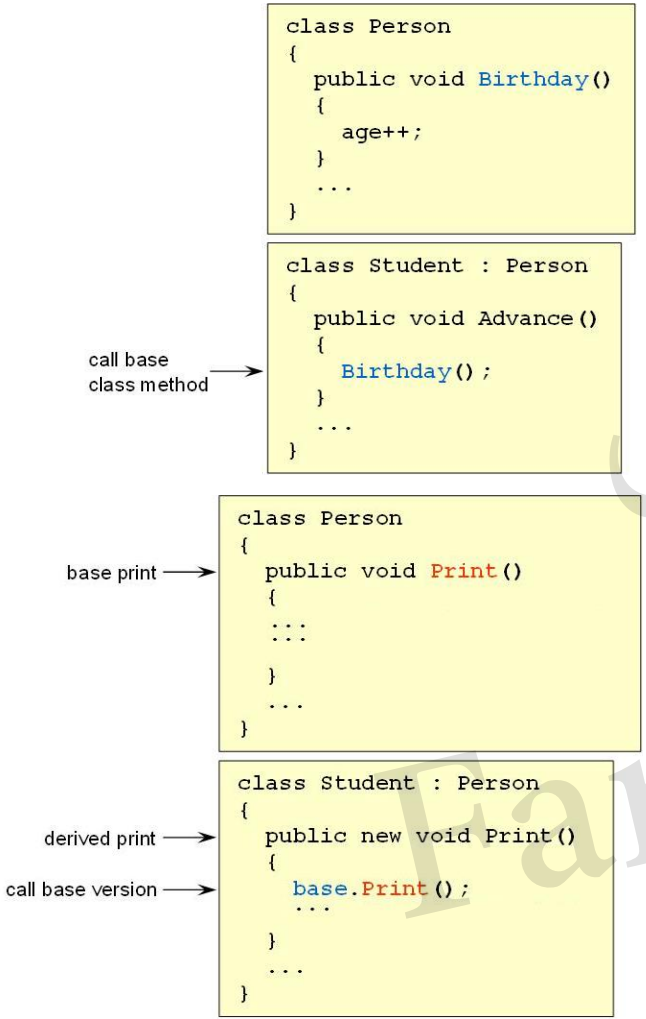
label with new to acknowledge hiding of inherited version →

```
class Student : Person
{
  int id;

  public new void SetId(int id)
  {
    this.id = id;
  }
}
```

derived class می‌تواند عضوی را تعریف کند که مشابه نام یکی از اعضای base classاش باشد. هنگامی‌که چنین اتفاقی افتد، آن عضو base class در derived class دیده نمی‌شود. درحالی‌که این مورد از لحاظ تکنیکی در سی‌شارپ خطا شمرده نمی‌شود، کامپایلر یک پیغام هشدار به شما داده و از این‌که یکی از اعضا دیده نمی‌شود شما را با خبر می‌سازد. اگر قصد شما این باشد تا باعث دیده نشدن یکی از اعضای base class شوید، به‌منظور رفع هشدار کامپایلر در derived class در تعریف آن عضو از کلمه‌ی‌کلیدی new استفاده کنید. دقت داشته باشید که این استفاده از new با حالتی که از آن برای ساختن شیء استفاده می‌کردید متفاوت است.

فرادرس
FaraDars.org

Invoking base class methods



یکی از کاربردهای base برای دسترسی به اعضایی از base class است که به دلیل تشابه اسمی در derived class قابل مشاهده نیستند. derived class می‌تواند عضوی را تعریف کند که مشابه نام یکی از اعضای base classاش باشد. هنگامی‌که چنین اتفاقی افتد، آن عضو base class در derived classدیده نمی‌شود. درحالی‌که این مورد از لحاظ تکنیکی در سی‌شارپ خطا شمرده نمی‌شود، کامپایلر یک پیغام هشدار به شما داده و از این‌که یکی از اعضا دیده نمی‌شود، شما را باخبر می‌سازد. اگر قصد شما این باشد تا باعث دیده نشدن یکی از اعضای base class شوید، به‌منظور رفع هشدار کامپایلر، در derived class در تعریف آن عضو از کلمه‌ی‌کلیدی new استفاده کنید. دقت داشته باشید که این‌طور استفاده از new با آن حالتی که از آن برای ساختن شیء استفاده می‌کردید متفاوت است. استفاده از base تا حدودی شبیه به this است با این تفاوت که base همیشه به base class رجوع می‌کند. نحوه‌ی استفاده از base به شکل زیر است:

base.member

در این‌جا member هم می‌تواند متغیر و هم می‌تواند متد باشد. این نحوه‌ی استفاده از base برای مواقعی است که یک عضو در base class به دلیل تشابه اسمی در derived class دیده نمی‌شود. این مورد درباره‌ی متدها نیز صدق می‌کند.

فرادرس
FaraDars.org

ایجاد کلاس ها بدون استفاده از ارث بری

در این مثال، از یک سلسله مراتب ارث‌بری که حاوی انواع کارمندان در برنامه پرداخت دستمزد یک شرکت است استفاده می‌کنیم تا به توضیح رابطه موجود مابین یک کلاس مبنا و یک کلاس مشتق شده بپردازیم. کارمندان کمیسیون (یا کارمندان حق‌العمل‌کار) که بعنوان اشیاء از کلاس مبنا عرضه خواهند شد، حقوق خود را بصورت درصدی از فروش دریافت می‌کنند، در حالیکه کارمندان کمیسیون مبتنی بر پایه حقوق(که بعنوان اشیائی از کلاس مشتق شده عرضه خواهند شد) یک حقوق پایه به همراه درصدی از فروش را دریافت می‌کنند. بحث خود را که در ارتباط با رابطه موجود مابین این دو نوع کارمند است به دقت و به کمک چند مثال مطرح می‌کنیم. در مثال اول برای هریک از این موجودیت‌ها کلاس‌های مستقل ایجاد می‌کنیم. همانطور که مشاهده می‌کنید بیش از ۹۰ درصد کد نوشته شده تکراری می‌باشد.

```
class CommissionEmployee
{
    private string firstname;
    private string lastname;
    private string socialsecuritynumber;
    private double grosssales;//Monthly
    private  double commissionrate;

    public string Firstname...
    public string Lastname...
    public string SocialSecurityNumber...
    public double GrossSales...
    public double CommissionRate...
    public CommissionEmployee(string fn,string ln,string ssn,double sales,double rate)...
    public double Earnings()
      {
        return CommissionRate * GrossSales;
      }
    public override string ToString()...
}
```

```
class BasePluseCommissionEmployee
{
    private string firstname;
    private string lastname;
    private string socialsecuritynumber;
    private double grosssales;//Monthly
    private double commissionrate;
    private double basesalary;

    public string Firstname...
    public string Lastname...
    public string SocialSecurityNumber...
    public double GrossSales...
    public double CommissionRate...
    public double BaseSalary...
    public BasePluseCommissionEmployee(string fn, string ln, string ssn, double sales, double rate,double salary)...
    public double Earnings()
    {
        return BaseSalary+ CommissionRate * GrossSales;
    }
    public override string ToString()...
}
```

فرادرس
FaraDars.org

استفاده از ویژگی وراثت در ایجاد کلاس‌ها

در این بخش یک نسخه جدید از کلاس BasePlusCommssionEmployeeایجاد می‌کنیم که از کلاس CommissionEmployee مشتق شده است. در این مثال شی BasePlusCommssionEmployee یک CommisionEmployee است چرا که قابلیت‌های کلاس CommissionEmployee را به ارث می‌برد.


```
class CommissiomEmployee
{
    private string firstname;
    private string lastname;
    private string socialsecuritynumber;
    private double grosssales;//Monthly
    private  double commissionrate;
    public string Firstname...
    public string Lastname...
    public string SocialSecurityNumber...
    public double GrossSales...
    public double CommissionRate...
    public CommissiomEmployee(string fn,string ln,string ssn,double sales,double rate)...
    public double Earnings()
      {
        return CommissionRate * GrossSales;
      }
    public override string ToString()...
}
```

```
class BasePluseCommissiomEmployee:CommissiomEmployee
{
    private double basesalary;
    public double BaseSalary...
    public BasePluseCommissiomEmployee(string fn, string ln, string ssn, double sales, double rate,double salary)
        :base( fn,  ln,  ssn,  sales,  rate)...
    public new double Earnings()...
    public override string ToString()...
}
```

فرادرس
FaraDars.org

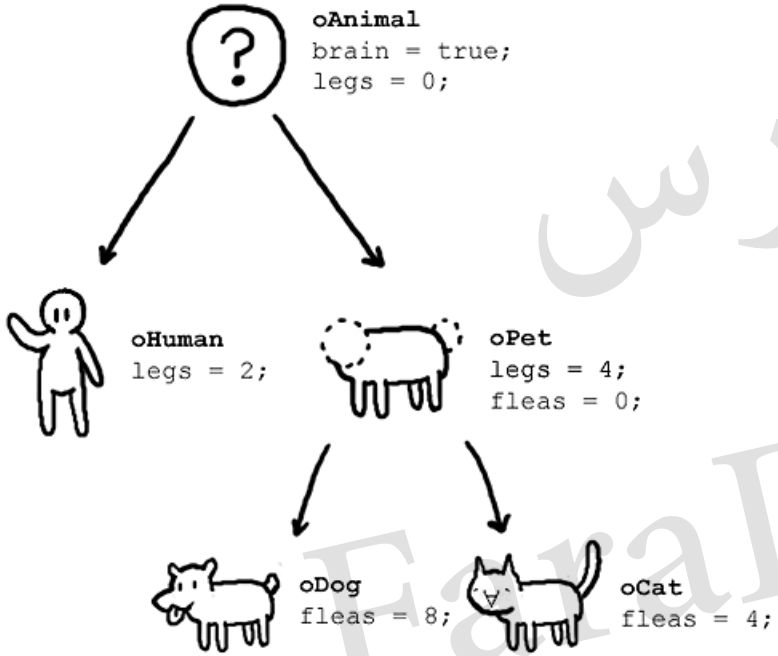
Overriding و متدهای Virtual

```
class ClassA
{
    4 references
    public virtual void SayHello()
    {
        MessageBox.Show("SayHello() in ClassA class.");
    }
}
3 references
class ClassB : ClassA
{
    4 references
    public override void SayHello()
    {
        MessageBox.Show("SayHello() in ClassB class.");
    }
}


class ClassC : ClassB
{

}
```

Virtual method، متدی است که با کلمه‌ی کلیدی virtual و در base class تعریف می‌شود. Virtual method به شکلی است که می‌توانید آن را در derived class مجدداً تعریف کنید. از این‌رو، هر derived class می‌تواند نسخه‌ی اختصاصی خودش را از virtual method داشته باشد. همان‌طور که گفته شد، virtual method در base class با کلمه‌ی کلیدی virtual تعریف می‌شود. هنگامی‌که یک virtual method در derived class مجدداً تعریف می‌شود، باید از override modifier استفاده کنید بنابراین پروسه تعریف مجدد virtual method در derived class را method overriding می‌نامیم. هنگام override کردن یک متد، باید اسم متد، return type و پارامترهای آن را مطابق با virtual method بنویسیم. هنگامی‌که از سلسله مراتب ارث‌بری استفاده می‌کنید، اگر یک derived class، یک virtual method را override نکند، به طرف ابتدای زنجیره‌ی ارث‌بری حرکت کنید، اولین override آن متد که دیده شود اجرا خواهد شد.

فرادرس
FaraDars.org

کلاس انتزاعی Abstract Class



گاهی قصد دارید یک base class بسازید که تنها یک فرم کلی را مشخص می‌کند و آن را با تمام کلاس‌های مشتق شده به اشتراک می‌گذارد و اجازه می‌دهد که خود derived classها بدنه و جزئیات این فرم کلی را تکمیل کنند. به‌عنوان مثال، چنین کلاسی ماهیت یک متد را مشخص می‌کند و derived classها باید این متد را override کنند اما خود base class دیگر نیازی ندارد که برای این متد یک اجرای پیش‌فرض داشته باشد. این حالت ممکن است زمانی رخ دهد که base class نتواند یک اجرای بامعنی برای متد مورد نظر داشته باشد، از این‌رو اجرا را بر عهده‌ی derived classها می‌گذارد. ایجاد اشیاء از این کلاس‌ها بی‌معنی است. برای اینکه نتوان از کلاس‌های پایه اشیاء تعریف کرد آنها را به صورت abstract تعریف می‌نماییم.

فرادرس
FaraDars.Org

abstract Method & abstract Class

یک متد abstract با abstract modifier ساخته می‌شود. abstract method بدنه ندارد و از این‌رو درون base class اجرا نخواهد شد. derived classها حتماً باید این abstract method را override کنند. یک abstract method به‌صورت اتوماتیک virtual نیز هست و در واقع نمی‌توانید از virtual و abstract باهم در یک تعریف استفاده کنید. همان‌طور که می‌بینید در abstract method به بدنه نیاز ندارید. دقت کنید که abstract modifier را نمی‌توانید برای متدهای static استفاده کنید. Properties و indexers نیز می‌توانند abstract باشند.

کلاسی که شامل یک یا بیشتر از یک متد abstract باشد باید به‌صورت abstract تعریف شود. برای تعریف یک کلاس به‌صورت abstract کافی است که قبل از کلمه‌ی کلیدی class از abstract modifier استفاده کنید. از آن‌جا که abstract class نمی‌تواند به‌طور کامل اجرا شود(به‌دلیل وجود متدهای abstract که بدنه ندارند)، به‌همین دلیل نمی‌توانید از abstract class شیء بسازید. هنگامی‌که یک derived class از یک abstract class ارث‌بری می‌کند باید تمام متدهای abstract در base class را override کند در غیر این‌صورت derived class نیز باید به‌صورت abstract تعریف شود.

```
abstract class myclass1
 {
    int id;
     1 reference
     public abstract void func1();
     0 references
     public void func2()
     {
        //......
     }
 }

1 reference
class myclass2 : myclass1
{
    1 reference
    public override void func1()
     {

     }
}
```

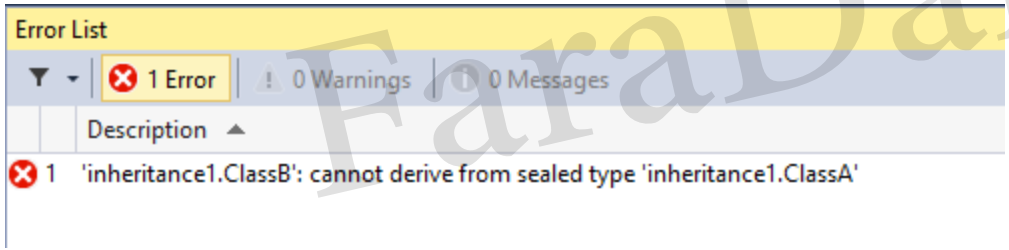
FaraDars.org

استفاده از sealed برای جلوگیری از ارث‌بری

```
sealed  class ClassA
  {

  }

  0 references
  class ClassB : ClassA
  {

  }
```

با این‌که inheritance بسیار مفید و کاربردی است، گاهی نیاز است که از انجام شدن آن پیش‌گیری کنید. این‌که در کجا و در چه شرایطی از انجام inheritance جلوگیری کنید، بستگی به مسأله و منطق خودتان دارد. در سی‌شارپ با استفاده از کلمه‌ی کلیدی sealed به راحتی می‌توانید مانع انجام شدن inheritance شوید.

واژه‌ی sealed به معنای مهر و موم شده است و با استفاده از آن اطمینان می‌یابید که از یک کلاس مهر و موم شده نمی‌توان ارث‌بری کرد. به‌منظور sealed کردن یک کلاس، کافی است که در ابتدای تعریف کلاس از کلمه‌ی کلیدی sealed استفاده کنید. دقت داشته باشید که یک کلاس را نمی‌توان هم به‌صورت sealed و هم به‌صورت abstract تعریف کرد چراکه کلاس abstract به تنهایی کامل نیست و برای این‌که اجرای کاملی داشته باشد وابسته به کلاس‌های مشتق‌شده از خودش است.

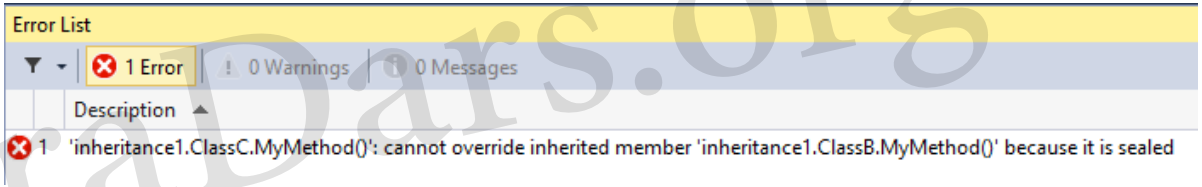


FaraDars.org
فرادرس

استفاده از sealed برای جلوگیری از override متدها

نکته‌ی دیگر این است که sealed می‌تواند در virtual methods نیز برای پیش‌گیری از override شدن مورد استفاده قرار گیرد. در این‌جا، کلاس ClassA یک متد virtual دارد که در کلاس ClassB هم override و هم sealed شده است. از این‌رو کلاس‌هایی که از ClassB ارث‌بری می‌کنند دیگر نمی‌توانند MyMethod() را override کنند زیرا این متد دیگر sealed شده است.

```
class ClassA
{
        1 reference
        public virtual void MyMethod()
        {
        }
}
1 reference
class ClassB : ClassA
{
        1 reference
        sealed public override void MyMethod()
        {
        }
}
0 references
class ClassC : ClassB
{
    // Error! MyMethod() is sealed!
        2 references
    public override void MyMethod()
    {
    }
}
```



FaraDars.org

Indexers و شکل کلی آنها

```
element-type this[int index] {
    // The get accessor
    get {
        // return the value specified by index
    }

    // The set accessor
    set {
        // set the value specified by index
    }
}
```

همان‌طور که می‌دانید، index گذاری آرایه از طریق عملگر [ ] انجام می‌شود. تعریف کردن عملگر [ ] برای کلاس نیز امکان‌پذیر است اما برای این منظور از operator method استفاده نکرده و در عوض از Indexer استفاده می‌کنید. Indexer اجازه می‌دهد یک شیء مانند یک آرایه index گذاری شود. Indexerها می‌توانند یک یا بیشتر از یک بعد داشته باشند. فرم کلی Indexer یک بعدی به‌شکل مقابل است. در این‌جا element-type مشخص کننده‌ی نوع عنصر indexer است. از این‌رو، هر عنصری که توسط indexer قابل دسترسی باشد، از نوع element-type است. این نوع با نوع یک آرایه (که برای indexer در نظر می‌گیرید و اصطلاحاً به آن backing store می‌گویند) یکسان است. پارامتر index در واقع index عنصری که می‌خواهید به آن دسترسی داشته باشید را مشخص می‌کند. توجه کنید که نیازی نیست حتماً جنس پارامتر int باشد اما از آن‌جا که indexerها مشابه با index آرایه مورد استفاده قرار می‌گیرند، استفاده از int در این مورد رایج است.

درون بدنه‌ی indexer کلمه‌های get و set را مشاهده می‌کنید که به هر کدام از آن‌ها accessor گفته می‌شود. یک accessor مشابه یک متد است با این تفاوت که return-type و parameter ندارد. هنگامی‌که از indexer استفاده می‌کنید این accessorها به‌طور اتوماتیک فراخوانی می‌شوند و هر دوی accessorها index را به‌عنوان پارامتر دریافت می‌کنند. اگر indexer در طرف چپ تساوی قرار گرفته باشد، بنابراین set accessor فراخوانی و یک مقدار به عنصری که توسط index مشخص شده است، اختصاص داده می‌شود. در غیر این‌صورت get accessor فراخوانی شده و عنصر مشخص شده توسط index، return می‌شود. Set همچنین یک پارامتر به اسم value دارد که شامل مقداری است که به یک index مشخص اختصاص داده می‌شود. یکی دیگر از مزیت‌های indexer این است که می‌توانید دسترسی به آرایه را دقیقاً تحت کنترل داشته باشید و از دسترسی‌های نامناسب جلوگیری کنید.

فرادرس
FaraDars.org

مثال چندضلعی

کلاس چندضلعی آرایه‌ای از نقاط برای ذخیره‌سازی رئوس دارد. همچنین برای نوشتن و خواندن رئوس نیز از متد استفاده می‌شود. می‌توان به کمک indexer این کلاس را بسیار خواناتر کرد.

data storage →
set →
get →

```
class Polygon
{
  private Point[] vertices;

  public void SetVertex(int i, Point value)
  {
    vertices[i] = value;
  }

  public Point GetVertex(int i)
  {
    return vertices[i];
  }
}
```

set →
get →

```
Polygon triangle = new Polygon(3);

triangle.SetVertex(0, new Point(0,0));

Point p = triangle.GetVertex(0);
```

فرادرس
FaraDars.org

Basic indexer

data storage

```
class Polygon
{
  private Point[] vertices;
  ...
}
```

```
class Polygon
{
  ...
  public Point this [int i] { ... }
}
```

access type this parameter accessors

```
class Polygon
{
  ...
  public Point this [int i]
  {
    set { ... }

    get { ... }
  }
}
```

write

read

```
class Polygon
{
  Point[] vertices;

  public Point this [int i]
  {
    set
    {
      vertices[i] = value;
    }
    ...
  }
  ...
}
```

set accessor

index data

```
class Polygon
{
  Point[] vertices;

  public Point this [int i]
  {
    get
    {
      return vertices[i];
    }
    ...
  }
  ...
}
```

get accessor

return a Point

```
Polygon triangle = new Polygon();

triangle[0] = new Point(0, 0);
triangle[1] = new Point(3, 1);
triangle[2] = new Point(1, 2);

Point apex = triangle[2];
...
```

calls set accessor

calls get accessor

فرادرس
FaraDars.org

Index type

در این‌جا، element-type مشخص کننده‌ی نوع عنصر indexer است. از این‌رو، هر عنصری که توسط indexer قابل دسترسی باشد، از نوع element-type است. این نوع با نوع یک آرایه(که برای indexer در نظر می‌گیرید و اصطلاحاً به آن backing store می‌گویند) یکسان است. پارامتر index در واقع index عنصری که می‌خواهید به آن دسترسی داشته باشید را مشخص می‌کند. توجه کنید که نیازی نیست حتماً جنس پارامتر int باشد اما از آن‌جا که indexerها مشابه با index آرایه مورد استفاده قرار می‌گیرند، استفاده از int در این مورد رایج است.

```
element-type this[int index] {
    // The get accessor
    get {
        // return the value specified by index
    }

    // The set accessor
    set {
        // set the value specified by index
    }
}
```

string parameter →

```
class Department
{
  public Employee this [string name] ...
  ...
}
```

```
Department d = new Department();
...
Employee e = d["Ann"];
```

pass string →

FaraDars.org

Index number

Indexerها می‌توانند یک یا بیشتر از یک بعد داشته باشند.
برای تعریف indexer چند بعدی کافی است تعداد index ها را زیاد کرد.

2 parameters →

```
class Matrix
{
  public int this [int row, int column] ...
  ...
}
```

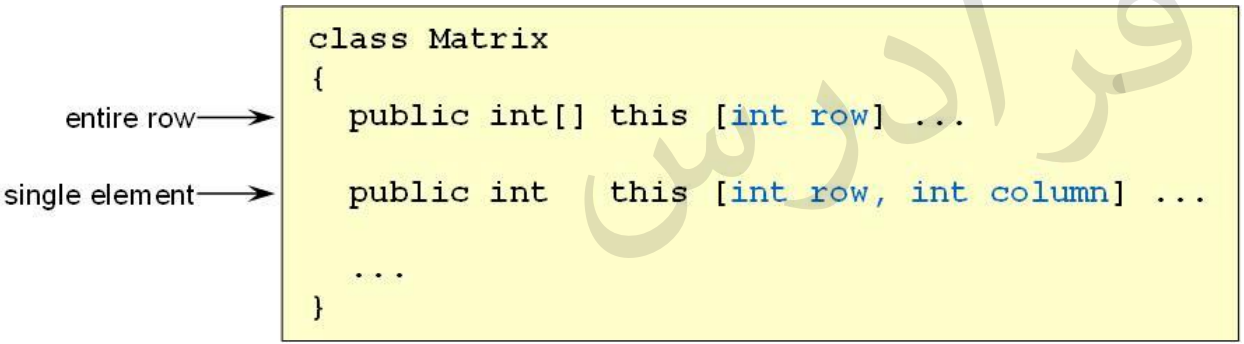
pass 2 indices →

```
Matrix m = new Matrix();
...
int v = m[1,2];
```

فرادرس
FaraDars.org

Indexer overloading

می‌توان برای یک کلاس چندین indexer تعریف کرد.



فرادرس
FaraDars.org

مثال چندضلعی

در این مثال از متدهای setvertex و getvertex برای خواندن و نوشتن در آرایه رئوس استفاده شده که استفاده از کلاس را سخت کرده و متحدالشکل بودن با آرایه را ندارد و این عدم یکسانی موجب عدم خوانایی و عدم راحتی کار با اشیائی از نوع Polygon1 می‌گردد.

```
class Polygon1
{
Point[] vertices;
public Polygon1(int n)
{
vertices= new Point[n];
}
public void setvertex(int i, Point value)
{
vertices[i] = value;
}
public Point getvertex(int i){
return vertices[i];
}
}
```

```
private void button1_Click(object sender, EventArgs e)
{
Polygon1 triangle = new Polygon1(3);
triangle.setvertex(0, new Point(10, 10));
triangle.setvertex(1, new Point(40, 10));
triangle.setvertex(2, new Point(40, 70));
}
```

فرادرس
FaraDars.org

مثال چندضلعی با indexer

در این مثال بجای متدهای setvertexو getvertex برای خواندن و نوشتن در آرایه رئوس از indexer استفاده شده که استفاده از کلاس را با آرایه‌ها متحدالشکل کرده و این یکسانی موجب خوانایی و راحتی کار با اشیائی از نوع Polygon2 می‌گردد.

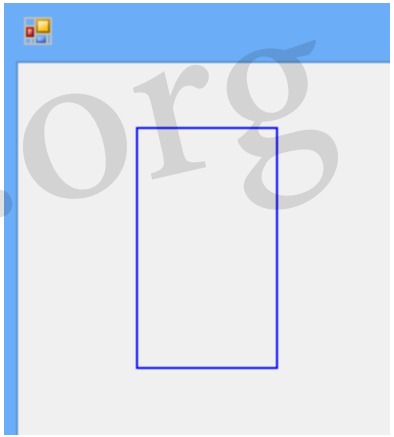
```
class Polygon2
{
    Point[] vertices;
    public Polygon2(int n)
    {
        vertices= new Point[n];
    }
    public Point this[int i]{
        set
        {
            vertices[i] = value;
        }
        get
        {
            return vertices[i];
        }
    }
}
```

```
private void button2_Click(object sender, EventArgs e)
{
    Polygon2 triangle = new Polygon2(3);
    triangle[0] = new Point(10, 10);
    triangle[1] = new Point(50, 50);
    triangle[2] = new Point(10, 70);
}
```

فرادرس
FaraDars.org

مثال چندضلعی با indexer

در این مثال برای کلاس چند ضلعی یک متد بنام draw ایجاد کردیم که با ترسیم خط بین رئوس چندضلعی را رسم می‌کند. به سهولت و خوانایی استفاده از کلاس Polygon2 در کد روبرو دقت کنید تا مزیت استفاده از indexer را بهتر درک کنید. کلاس Point یک کلاس موجود در فضای نام System.Drawing می‌باشد.

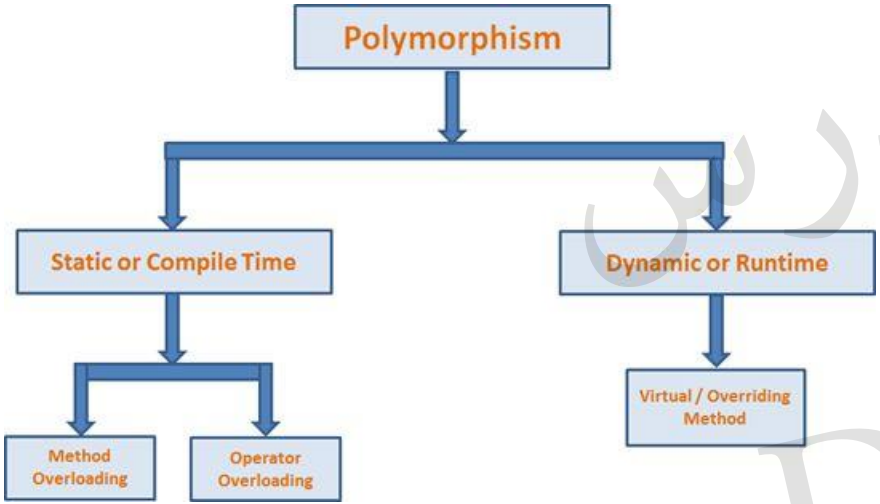


```
class Polygon2
{
    Point[] vertices;
    public Polygon2(int n)
    {
        vertices= new Point[n];
    }
    public Point this[int i]{
        set{ vertices[i] = value;  }
        get{ return vertices[i];   }
    }
    public void draw(Graphics x,Color c){
        int i;
        for ( i = 0; i < vertices.Length - 1;++i )
            x.DrawLine(new Pen(c),vertices[i],vertices[i+1]);
        x.DrawLine(new Pen(c), vertices[i], vertices[0]);
    }
}
```

```
private void button3_Click(object sender, EventArgs e)
{
    Polygon2 rec1 = new Polygon2(4);
    rec1[0] = new Point(10, 10);
    rec1[1] = new Point(80, 10);
    rec1[2] = new Point(80, 130);
    rec1[3] = new Point(10, 130);
    rec1.draw(pictureBox1.CreateGraphics(), Color.Blue);
}
```

FaraDars.org

چندریختی Polymorphism



چندریختی امکان می‌دهد تا برنامه‌ها بجای اینکه «برنامه خاصی» باشند، حالت یک «برنامه کلی» داشته باشند.
به کمک چندریختی، می‌توانیم سیستم‌هایی را طراحی و پیاده‌سازی کنیم که گسترش و بسط‌پذیری آنها آسانتر است. کلاس‌های جدید می‌توانند با کمی تغییر یا اصلاح در بخش‌های عمومی برنامه به آن افزوده شوند، مادامیکه کلاس‌های جدید بخشی از سلسله مراتب توارثی باشند که برنامه بطور جامع آنرا پردازش می‌کند. تنها بخش‌های از برنامه که باید برای تطبیق یافتن با کلاس‌های جدید تغییر داده شوند آنهایی هستند که نیاز دارند تا از وجود کلاس‌های جدید افزوده شده به سلسله مراتب مستقیماً مطلع گردند.

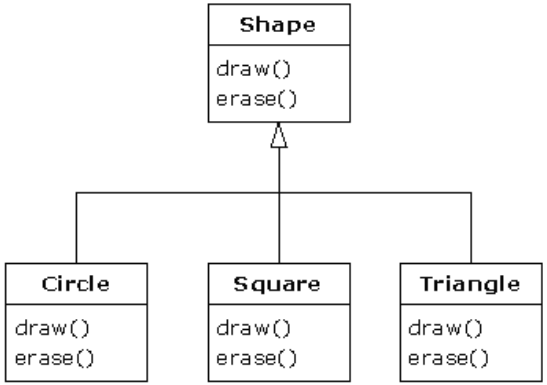
فرادرس
FaraDars.org

مثال چند ریختی

فرض کنید می‌خواهیم برنامه‌ای بنویسیم که صدای چند نوع حیوان را شبیه‌سازی کند. کلاس‌های cat(گربه)، Dog(سگ) و Dock(اردک) نشان‌دهنده سه نوع حیوان تحت بررسی هستند. تصور کنید که هر یک از این کلاس‌ها از کلاس مبنای Animal ارث‌بری دارند، که حاوی یک تابع speak() بوده که صدای مربوط به حیوان را نشان می‌دهد. هر کلاس مشتق شده متد speak را پیاده‌سازی می‌کند. برنامه مبادرت به نگهداری یک بردار(vector) از اشاره‌گرها به اشیائی از انواع کلاس‌های مشتق شده‌ی Animal می‌کند. برای شبیه‌سازی حرکت حیوانات، برنامه به هر شی در هر ثانیه یک پیغام بنام speak() ارسال می‌کند. با این وجود، هر نوع خاص از حیوان به این پیغام به روش خود پاسخ می‌دهد، برنامه بطور جامع یک پیغام را به هر شی ارسال می‌کند، اما هر شی از صدای خود مطلع است و بر مبنای آن حرف می‌زند. بر پایه اینکه هر شی از نحوه «انجام فعل صحیح» مطلع است، واکنش به فراخوانی متد یکسان، مفهوم کلیدی چندریختی یاpolymorphism است. پیغام یکسان در این مورد speak() که به انواع اشیاء ارسال می‌شود نتایج مختلفی بدنبال دارد و از اینرو نشاندهنده مفهوم چندریختی است. به کمک چندریختی، می‌توانیم سیستم‌هایی را طراحی و پیاده‌سازی کنیم که گسترش و بسط‌پذیری آنها آسانتر است. کلاس‌های جدید می‌توانند با کمی تغییر یا اصلاح در بخش‌های عمومی برنامه به آن افزوده شوند، مادامیکه کلاس‌های جدید بخشی از سلسله مراتب توارثی باشند که برنامه بطور جامع آنرا پردازش می‌کند. تنها بخش‌هایی از برنامه که باید برای تطبیق یافتن با کلاس‌های جدید تغییر داده شوند آنهایی هستند که نیاز دارند تا از وجود کلاس‌های جدید افزوده شده به سلسله مراتب مستقیماً مطلع گردند. برای مثال، اگر کلاس Bird(پرنده) که از کلاس Animal ارث‌بری دارد را ایجاد کنیم(که می‌تواند متد speak را بازنویسی کند)، فقط نیاز است تا کلاس Bird و آن بخشی که یک نمونه از شی Bird را شبیه‌سازی می‌کند را بنویسیم.

فرادرس
Polymorphism
FaraDars.org

مثال چند ریختی



فرض کنید که مجموعه‌ای از کلاس‌های شکل همانند Circle(دایره)، Triangle(مثلث) و Square(مربع) داریم که همگی از کلاس مبنا Shape مشتق شده‌اند. هر کدامیک از کلاس‌ها می‌توانند از قابلیت ترسیم خود از طریق یک تابع عضو بنام draw برخوردار باشد. اگرچه هر کلاسی دارای تابع draw متعلق بخود است، لیکن عملکرد این تابع برای هر شکل با دیگری کاملاً متفاوت خواهد بود. در برنامه‌ای که مجموعه‌ای از اشکال را ترسیم می‌کند، قابلیت تلقی کردن تمام اشکال بصورت اشیائی از کلاس مبنای Shape سودمند خواهد بود. سپس برای ترسیم هر شکل می‌توانیم به آسانی از یک اشاره‌گر Shape کلاس مبنا برای فراخوانی تابع draw استفاده کرده و به برنامه اجازه دهیم تا بصورت دینامیک(یعنی در زمان اجرا) تعیین کند که کدام تابع draw کلاس مشتق شده برحسب نوع شی که اشاره‌گر Shape به آن اشاره می‌کند، بکار گرفته شود.

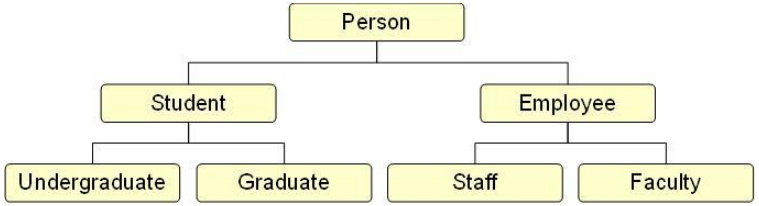
برای داشتن چنین رفتاری، ابتدا تابع draw را در کلاس مبنا بعنوان یک تابع virtual اعلان کرده و تابع draw در هر کلاس مشتق شده را برای ترسیم شکل مقتضی override می‌کنیم.

اگر برنامه‌ای مبادرت به فراخوانی یک تابع virtual از طریق اشاره‌گر یک کلاس مبنا به یک شی از کلاس مشتق شده کند، برنامه بصورت دینامیک(یعنی در زمان اجرا) تابع صحیح draw را براساس نوع شی و نه نوع اشاره‌گر انتخاب خواهد کرد. انتخاب تابع مقتضی برای فراخوانی در زمان اجرا(بجای زمان کامپایل) بعنوان مقیدسازی دینامیک(dynamic binding) شناخته می‌شود.

زمانیکه یک تابع virtual توسط مرجعی به یک شی خاص و به کمک نام و عملگر انتخاب عضو نقطه فراخوانی می‌شود، احضار تابع در زمان کامپایل مقرر می‌شود(که به این حالت مقیدسازی استاتیک گفته می‌شود) و تابع virtual که فراخوانی شده یک تابع تعریف شده برای کلاسی از شی مشخص است، که این رفتار نشان‌دهنده چندریختی استاتیک است.

FaraDars.Org

Type Compatibility



هر شی از کلاس مشتق یک شی از نوع کلاس پایه نیز می‌باشد و برای این نوع تبدیل نیاز به casting نمی باشد و بصورت ضمنی انجام می‌شود.

Person supports the Birthday operation →

```
class Person
{
   string name;
   int    age;

   public void Birthday()
   {
     age++;
   }
}
```

derive Student from Person →

add method →

```
class Student : Person
{
  int id;

  public void SetId(int id)
  {
    this.id = id;
  }
}
```

Person reference to Student object →

```
Person p = new Student();
```

Person reference to any derived object →

```
Person a = new Person();
Person b = new Student();
Person c = new Graduate();
Person d = new Undergraduate();
Person e = new Employee();
Person f = new Faculty();
Person g = new Staff();
```

Person reference to Student object →

Person method ok →

error, no access to Student method →

```
Person p = new Student();

p.Birthday();

p.SetId(12345);
...
```

فرادرس
FaraDars.org

Type Compatibility مزایای

علی‌رقم محدودیت‌های سازگاری اشیائی از نوع مشتق با مراجعی از نوع کلاس پایه، همانگونه که مشاهده می‌کنید تمام اشیائی که از کلاس‌های مشتق شده ایجاد می‌شوند را می‌توان به متد CheckAge ارسال کرد. بنابراین یک متد کلی ایجاد شده است.

method works for all people →

```
public bool CheckAge(Person p)
{
  return p.age >= 21;
}
```

```
Student       s = new Student();
Undergraduate u = new Undergraduate();

if (CheckAge(s))
  ...

if (CheckAge(u))
  ...
```

pass Student →

pass Undergraduate →

فرادرس
FaraDars.org

Method binding

نسبت دادن متدها به اشیاء به دو روش Static و Dynamic انجام می‌شود. روش اول در زمان کامپایل انجام می‌شود و روش دوم در زمان اجرا.

all employees support the `Promote` operation →

```
class Employee : Person
{
  public void Promote() { ... }
  ...
}
```

`Faculty` version →

```
class Faculty : Employee
{
  public void Promote() { ... }
  ...
}
```

`Staff` version →

```
class Staff : Employee
{
  public void Promote() { ... }
  ...
}
```

nothing to do for generic employee →

```
class Employee : Person
{
  public void Promote()
  {
  }
  ...
}
```

`Faculty` implementation →

```
class Faculty : Employee
{
  public void Promote()
  {
    salary *= 1.2;

    level++;
    if (level > 4)
      tenure = true;
  }
  ...
}
```

`Staff` implementation →

```
class Staff : Employee
{
  public void Promote()
  {
    salary *= 1.1;
  }
  ...
}
```

which version? →

```
void Evaluate(Employee e)
{
  e.Promote();
}
```

pass `Faculty` →
pass `Staff` →

```
Faculty f = new Faculty();
Staff   s = new Staff  ();

Evaluate(f);
Evaluate(s);
```

فرادرس
FaraDars.org

Dynamic Polymorphism

بر پایه اینکه هر شی از نحوه «انجام فعل صحیح» مطلع است، واکنش به فراخوانی متد یکسان، مفهوم کلیدی چندریختی یا polymorphism است. پیغام یکسان در این مورد Promote() که به انواع اشیاء ارسال می‌شود، نتایج مختلفی بدنبال دارد و از اینرو نشاندهنده مفهوم چندریختی است. به کمک چندریختی می‌توانیم سیستم‌هایی طراحی و پیاده‌سازی کنیم که گسترش و بسط‌پذیری آنها آسانتر است.

virtual on base version

```
class Employee : Person
{
  public virtual void Promote() {}
}
```

override on derived versions

```
class Faculty : Employee
{
  public override void Promote() { ... }
}
```

```
class Staff : Employee
{
  public override void Promote() { ... }
}
```

binding based on object type

```
void Evaluate(Employee e)
{
  e.Promote();
}
```

Faculty object

Staff object

```
Faculty f = new Faculty();
Staff   s = new Staff  ();

Evaluate(f);
Evaluate(s);
```

new type

```
class Administrator : Employee
{
  public override void Promote() { ... }
}
```

generic code unchanged

```
void Evaluate(Employee e)
{
  e.Promote();
}
```

Administrator object

```
Administrator a = new Administrator();

Evaluate(a);
```

FaraDars.org

Downcasting

نسبت دادن یک Reference از جنس Faculty به شیء e از نوع Employee یعنی اینکه "دید خودمون" رو از سطح بالا (Employee) به سطح پائین تر (Faculty) تغییر بدم. اینکار به صورت پیش فرض(implicit-غیر صریح) امکان‌پذیر نیست و در صورتیکه شما بخواهید این کار را انجام دهید حتماً باید صراحتاً(explicit) مسئولیت این کار را با castingبه عهده بگیرید.

Faculty object, Employee reference →
can only access Employee members →
error, no access to Faculty members →

```
Employee e = new Faculty();

e.Promote();

e.level++;
```

error, no implicit conversion →

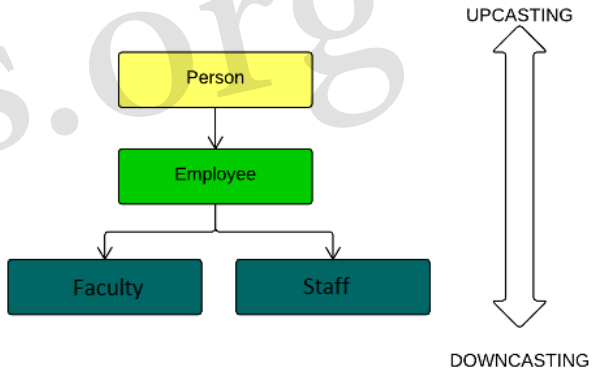
```
void Evaluate(Employee e)
{
  Faculty f = e;

  e.level++;
  ...
}
```

cast syntax →
access Faculty members →

```
void Evaluate(Employee e)
{
  Faculty f = (Faculty)e;

  f.level++;
}
```

operator as →
test for success →
access Faculty members →

```
void Evaluate(Employee e)
{
  Faculty f = e as Faculty;

  if (f != null)
  {
    f.level++;
  }
}
```



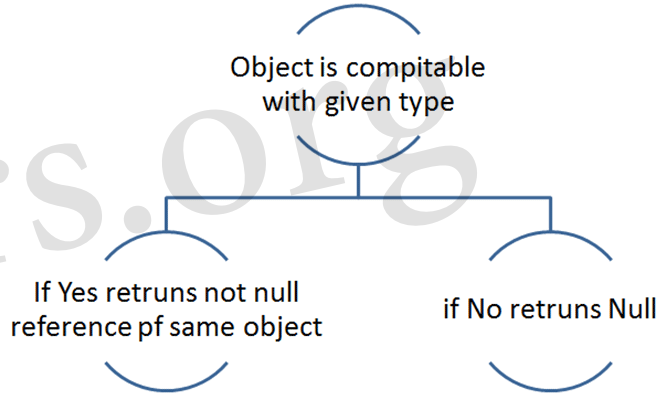
فرادرس
FaraDars.org

Type testing

به کمک عملگر is می‌توان تشخیص داد که آیا یک مرجع از نوع یک کلاس است یا خیر. به عنوان مثال آیا مرجع e که از نوع کلاس پایه employee می‌باشد به شئی از نوع کلاس Faculty اشاره می کند؟

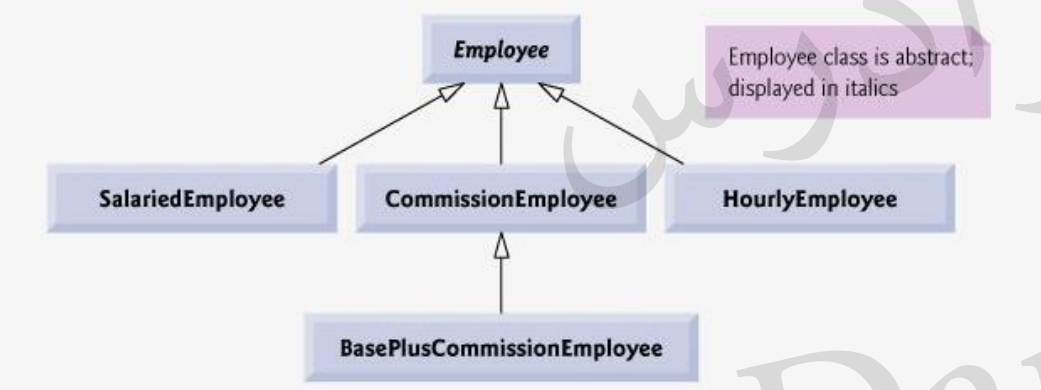
test type of object →

```
void Evaluate(Employee e)
{
  if (e is Faculty)
    ...
}
```



فرادرس
FaraDars.org

مثال کاربردی: سیستم پرداخت حقوق با استفاده از چند ریختی



شرکتی به کارمندان خود بطور هفتگی حقوق پرداخت می‌کند. کارمندان به چهار دسته تقسیم شده‌اند: کارمندانی که یک حقوق ثابت صرف نظر از ساعات کاری در هفته دریافت می‌کنند، کارمندانی که براساس ساعت کاری و اضافه کاری در طول هفته مازاد بر ۴۰ ساعت حقوق دریافت می‌کنند، کارمندانی که براساس فروش حقوق دریافت می‌کنند و کارمندانی که علاوه بر حقوق ثابت درصدی از فروش نیز کمیسیون به آنها تعلق می‌گیرد. شرکت تصمیم دارد که تا پرداخت حقوق‌های جاری به کارمندانی که حقوق پایه همراه با کمیسیون از فروش دریافت می‌کنند، ۱۰ درصد به میزان فروش آنها پاداش اضافه نماید. شرکت مایل به پیاده‌سازی یک برنامه‌ای است تا محاسبات پرداخت حقوق را به روش چند ریختی انجام دهد.

هر کارمندی صرفنظر از روش محاسبه حقوق وی، دارای نام، نام خانوادگی و شماره تأمین اجتماعی بوده و از اینرو اعضای داده خصوصی عبارتند از: firstName، lastNameو socialSecurityNumber که در کلاس مبنای انتزاعی Employee ظاهر می‌شوند.

FaraDars.org

مثال کاربردی: سیستم پرداخت حقوق با استفاده از چند ریختی

	Earnings	ToString
Employee	abstract	*firstName lastName* social security number: *SSN*
Salaried-Employee	weeklySalary	salaried employee: *firstName lastName* social security number: *SSN* weekly salary: *weeklysalary*
Hourly-Employee	*If hours <= 40* wage * hours *If hours > 40* 40 * wage + (hours - 40) * wage * 1.5	hourly employee: *firstName lastName* social security number: *SSN* hourly wage: *wage* hours worked: *hours*
Commission-Employee	commissionRate * grossSales	commission employee: *firstName lastName* social security number: *SSN* gross sales: *grossSales* commission rate: *commissionRate*
BasePlus-Commission-Employee	(commissionRate * grossSales) + baseSalary	base salaried commission employee: *firstName lastName* social security number: *SSN* gross sales: *grossSales* commission rate: *commissionRate* base salary: *baseSalary*

دیاگرام شکل مقابل نمایشی از پنج کلاس موجود در سلسله مراتب است که در سمت چپ آن و توابع earningsو ToStringدر سرستون‌ها قرار گرفته‌اند. برای هر کلاس، دیاگرام نتیجه دلخواه هر تابع را نشان می‌دهد. دقت کنید که کلاس Employee با "abstract" برای تابع earnings همراه شده و نشان می‌دهد که این تابع یک تابع virtual محض است. هر کلاس مشتق شده تابع مناسب خود را برای انجام مقاصد مقتضی انتخاب می‌کند(یعنی تابع earnings را override می‌کند).

FaraDars.org

ایجاد کلاس مبنای انتزاعی Employee

متد Earning از کلاس Employee از نوع abstract می‌باشد زیرا در این کلاس برای پیاده‌سازی آن مفاهیم کافی موجود نیست. این متد در کلاس‌هایی که از کلاس Employee مشتق می‌شوند پیاده‌سازی می‌گردد. به دلیل اینکه کلاس Employee حاوی یک متد انتزاعی می‌باشد، باید به شکل یک کلاس انتزاعی تعریف گردد.

```
public abstract class Employee
{
    2 references
    public string FirstName...
    2 references
    public string LastName ...
    2 references
    public string SocialSecurityNumber ...
    1 reference
    public Employee(string first, string last, string ssn)...
    4 references
    public override string ToString()
    {
        string s = "\nName : " + FirstName + " " + LastName + "\nSSN : " + SocialSecurityNumber.ToArray();
        return s;
    }
    1 reference
    public abstract decimal Earnings(); // no implementation here
}
```

FaraDars.org فرادرس

ایجاد کلاس مشتق شده غیرانتزاعی SalariedEmployee

```
public class SalariedEmployee : Employee
{
    private decimal weeklySalary;
    1 reference
    public SalariedEmployee(string first, string last, string ssn,decimal salary): base(first, last, ssn)...
    3 references
    public decimal WeeklySalary...
    1 reference
    public override decimal Earnings()
    {
        return WeeklySalary;
    }
    4 references
    public override string ToString()
    {
        string s = "salaried employee" + base.ToString() + "\nWeekly salary:" + WeeklySalary.ToString();
        return s;
    }
}
```

فرادرس
FaraDars.org

ایجاد کلاس مشتق شده غیرانتزاعی HourlyEmployee

```
public class HourlyEmployee : Employee
{
   private decimal wage;
   private decimal hours; // hours worked for the week
   0 references
   public HourlyEmployee( string first, string last, string ssn,decimal hourlyWage, decimal hoursWorked )
      : base( first, last, ssn )...
   5 references
   public decimal Wage...
   5 references
   public decimal Hours...
   2 references
   public override decimal Earnings()
   {
      if ( Hours <= 40 )
         return Wage * Hours;
      else
         return ( 40 * Wage ) + ( ( Hours - 40 ) * Wage * 1.5M );
   }
   8 references
   public override string ToString()
   {  string s = "hourly employee" + base.ToString();
       s += "\nhourly wage: " + Wage.ToString()+"\nhours worked: " + Hours.ToString();
       return s;
   }
}
```

فرادرس
FaraDars.org

ایجاد کلاس مشتق شده غیرانتزاعی CommissionEmployee

```
public class CommissionEmployee : Employee
{
    private decimal grossSales;
    private decimal commissionRate;
    0 references
    public CommissionEmployee(string first, string last, string ssn,decimal sales, decimal rate)
        : base(first, last, ssn)...
    3 references
    public decimal GrossSales...
    3 references
    public decimal CommissionRate...
    3 references
    public override decimal Earnings()
    {
        return CommissionRate * GrossSales;
    }
    12 references
    public override string ToString()
    {
        string s= "commission employee"+ base.ToString()+"\ngross sales: "+ GrossSales.ToString();
        s+= "\ncommission rate: "+ CommissionRate.ToString();
        return s;
    }
}
```

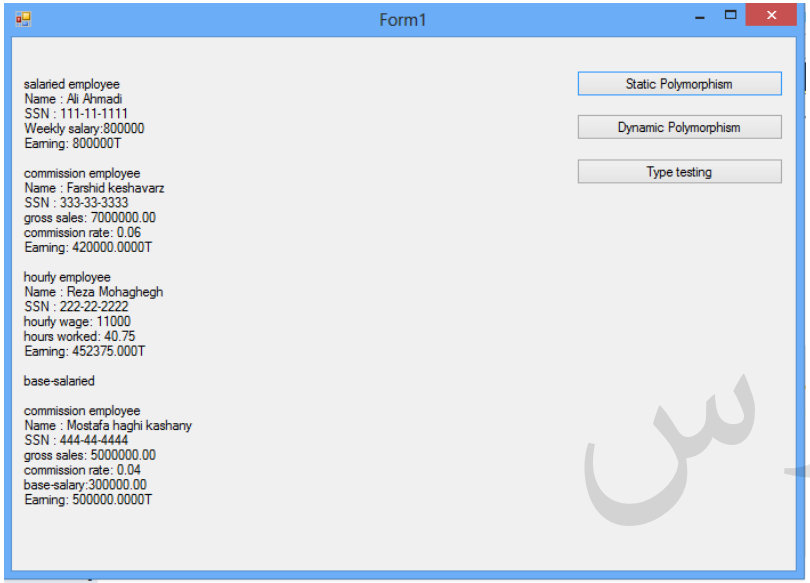
فرادرس
FaraDars.org

کلاس مشتق شده غیرانتزاعی BasePlusCommissionEmploye

```
public class BasePlusCommissionEmployee : CommissionEmployee
{
    private decimal baseSalary;
    0 references
    public BasePlusCommissionEmployee(string first, string last,string ssn, decimal sales, decimal rate,decimal salary)
        : base(first, last, ssn, sales, rate)...
    3 references
    public decimal BaseSalary...
    5 references
    public override decimal Earnings()
    {
        return BaseSalary + base.Earnings();
    }
    15 references
    public override string ToString()
    {
        string s;
        s="base-salaried "+ base.ToString()+"\nbase-salary:" + BaseSalary.ToString();
        return s;
    }
}
```

فرادرس
FaraDars.org

Static Polymorphism تست

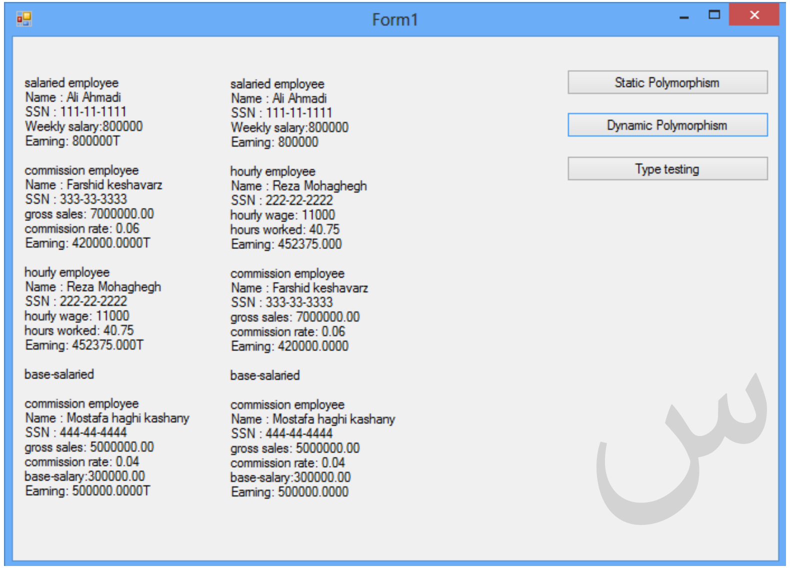


Static polymorphism زمانی اتفاق می‌افتد که در زمان Compile یک متد از طریق نام شی فراخوانی می‌گردد.

فرادرس
FaraDars.Org

```
private void button1_Click(object sender, EventArgs e)
{
    SalariedEmployee salariedEmployee =new SalariedEmployee("Ali", "Ahmadi", "111-11-1111", 800000);
    HourlyEmployee hourlyEmployee =new HourlyEmployee("Reza","Mohaghegh","222-22-2222", 11000, 40.75m);
    CommissionEmployee commissionEmployee =new CommissionEmployee("Farshid", "keshavarz","333-33-3333", 7000000.00M, .06M);
    BasePlusCommissionEmployee basePlusCommissionEmployee =new BasePlusCommissionEmployee("Mostafa", "haghi kashany","444-44-4444", 5000000.00M, .04M, 300000.00M);
    label1.Text = salariedEmployee.ToString() +"\nEarning: "+ salariedEmployee.Earnings()+"T";
    label1.Text += commissionEmployee.ToString() + "\nEarning: " + commissionEmployee.Earnings() + "T";
    label1.Text += hourlyEmployee.ToString() + "\nEarning: " + hourlyEmployee.Earnings() + "T";
    label1.Text += basePlusCommissionEmployee.ToString() + "\nEarning: " + basePlusCommissionEmployee.Earnings() + "T";
}
```

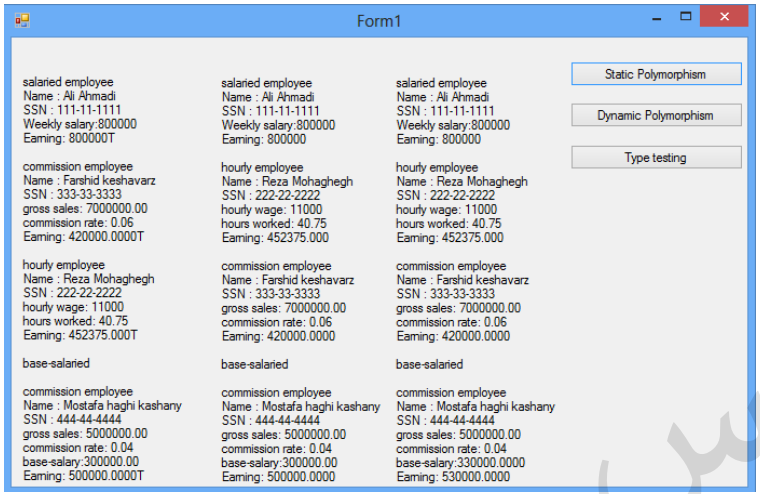
Dynamic Polymorphism تست



Dynamic polymorphism زمانی اتفاق می‌افتد که در زمان اجرا یک متد صحیح و مربوط به کلاس مشتق از طریق مرجع شی پایه فراخوانی می‌گردد. بر پایه اینکه هر شی از نحوه «انجام فعل صحیح» مطلع است، واکنش به فراخوانی متد یکسان از طریق مرجع شی پایه، مفهوم کلیدی چندریختی پویا یا polymorphism است. پیغام یکسان در این مورد Earning() که از طریق مراجع کلاس پایه به انواع اشیاء ارسال می‌شود، نتایج مختلفی بدنبال دارد و از اینرو نشان‌دهنده مفهوم چندریختی است.

```
private void button2_Click(object sender, EventArgs e) {
SalariedEmployee salariedEmployee = new SalariedEmployee("Ali", "Ahmadi", "111-11-1111", 800000);
HourlyEmployee hourlyEmployee = new HourlyEmployee("Reza", "Mohaghegh", "222-22-2222", 11000, 40.75m);
CommissionEmployee commissionEmployee = new CommissionEmployee("Farshid", "keshavarz", "333-33-3333", 7000000.00M, .06M);
BasePlusCommissionEmployee basePlusCommissionEmployee = new BasePlusCommissionEmployee("Mostafa", "haghi kashany", "444-44-4444", 5000000.00M, .04M, 300000.00M);
Employee[] employees = new Employee[4];
employees[0] = salariedEmployee;
employees[1] = hourlyEmployee;
employees[2] = commissionEmployee;
employees[3] = basePlusCommissionEmployee;
foreach ( Employee currentEmployee in employees )
label2.Text+=currentEmployee.ToString()+"\nEarning: "+currentEmployee.Earnings().ToString();}
```

فرادرس
FaraDars.org



Type Testing

زمانی که مرجعی از نوع کلاس پایه داریم، می‌توان کنترل کرد که آیا به شئی از نوع یک کلاس مشتق خاص اشاره می‌کند یا خیر. برای این منظور از عملگر is به شکل زیر استفاده می‌کنیم:

```
if(currentEmployee is BasePlusCommissionEmployee)
```

```
private void button2_Click(object sender, EventArgs e) {
SalariedEmployee salariedEmployee = new SalariedEmployee("Ali", "Ahmadi", "111-11-1111", 800000);
HourlyEmployee hourlyEmployee = new HourlyEmployee("Reza", "Mohaghegh", "222-22-2222", 11000, 40.75m);
CommissionEmployee commissionEmployee = new CommissionEmployee("Farshid", "keshavarz", "333-33-3333", 7000000.00M, .06M);
BasePlusCommissionEmployee basePlusCommissionEmployee = new BasePlusCommissionEmployee("Mostafa", "haghi kashany", "444-44-
4444", 5000000.00M, .04M, 300000.00M);
Employee[] employees = new Employee[4];
employees[0] = salariedEmployee;
employees[1] = hourlyEmployee;
employees[2] = commissionEmployee;
employees[3] = basePlusCommissionEmployee;
foreach (Employee currentEmployee in employees)
{
if (currentEmployee is BasePlusCommissionEmployee) {
BasePlusCommissionEmployee employee = (BasePlusCommissionEmployee)currentEmployee;
employee.BaseSalary *= 1.10M; }
label3.Text += currentEmployee.ToString() + "\nEarning: " + currentEmployee.Earnings().ToString();} }
```

فرادرس
FaraDars.org

واسط ها Interfaces

بعضی مواقع در برنامه‌نویسی شی‌گرا تعریف اینکه یک کلاس چه کاری را باید انجام دهد می‌تواند مفید باشد، اما اینکه این‌کار را به چه روشی انجام می‌دهد مهم نیست. شما پیش از این با چنین نمونه‌ای که abstract method نام داشت آشنا شدید. یک abstract method متدی را با یک return type و یک نام، تعریف می‌کند اما چیزی را اجرا نمی‌کند بلکه derived class باید abstract methodهایی که در base class تعریف شده‌اند را پیاده‌سازی کند. از این‌رو، abstract method مشخص کننده‌ی interface یک متد است و نه قسمت اجرایی. اگرچه abstract classes و abstract methods مفید و کاربردی هستند، اما می‌توان این مفهوم را به شکل کامل‌تری نیز بیان کرد. در سی‌شارپ شما می‌توانید interface یک کلاس را به‌طور کامل از بخش اجرایی آن جدا کنید، که این کار توسط کلمه‌ی کلیدی interface انجام می‌شود. Interface از نظر syntax مشابه با abstract class است. در interface نیز متدها بدنه ندارند و این بدین معنی است که در interface متدها اجرا نمی‌شوند. Interface مشخص می‌کند که چه کاری باید انجام شود اما به چگونگی انجام شدن آن اهمیت نمی‌دهد و شما هرطور که مایل هستید متد مورد نظر را اجرا می‌کنید.

FaraDars.Org

بین Interface و Abstract Class کدام‌یک را انتخاب کنیم؟

یکی از قسمت‌های مهم برنامه‌نویسی سی‌شارپ دانستن این موضوع است که هنگامی‌که قصد دارید قابلیت‌های یک کلاس را شرح دهید، چه زمانی از interface و چه زمانی از abstract class باید استفاده کنید درحالی‌که قسمت اجرایی ندارید. قانون کلی بدین صورت است که هرگاه بخواهید مفهوم کلی را شرح دهید و فقط به انجام شدن کارها تاکید داشته باشید و در واقع چگونگی انجام شدن آن برای شما اهمیت نداشته باشد، باید از interface استفاده کنید. اگر نیاز دارید که بعضی از جزئیات اجرا شدن را از قبل وارد کنید، آن‌گاه باید abstract class را مورد استفاده قرار دهید.

oops interface vs abstract class

Interface	Abstract class
Interface support multiple inheritance	Abstract class does not support multiple inheritance
Interface does'n Contains Data Member	Abstract class contains Data Member
Interface does'n contains Cunstructors	Abstract class contains Cunstructors
An interface Contains only incomplete member (signature of member)	An abstract class Contains both incomplete (abstract) and complete member
An interface cannot have access modifiers by default everything is assumed as public	An abstract class can contain access modifiers for the subs, functions, properties
Member of interface can not be Static	Only Complete Member of abstract class can be Static

فرادرس
FaraDars.org

قالب کلی تعریف واسط interface

interface definition →

```
interface IMyInterface
{
   ...
}
```

```
interface IMyInterface
{
  void Process(int arg1, double arg2);

  float this [int index] { get; set; }

  string Name { get; set; }
}
```

method →

indexer →

property →

یک واسط توسط کلمه‌ی کلیدی interface تعریف می‌شود. در مقابل فرم ساده شده‌ی یک interface را می‌بینید. اسم interface توسط name مشخص می‌شود. متدها نیز توسط return type، نام و پارامترها (signature) تعریف می‌شوند. این متدها در واقع abstract method هستند. همان‌طور که پیش‌تر ذکر شد، در interface، متدها بدنه‌ی اجرایی ندارند و از این‌رو کلاسی که interface دارد باید تمام متدهای تعریف شده در interface را پیاده‌سازی کند. در یک interface، متدها implicitly public هستند. یعنی به‌صورت پیش‌فرض و خودکار public هستند و شما اجازه‌ی تغییر این حالت را ندارید.

همانند متدها، properties نیز در interface بدون بدنه تعریف می‌شوند. اگر تنها از get یا set استفاده کنید، property شما read-only یا write-only خواهد بود. اگرچه تعریف property در interface مشابه با تعریف auto-implemented property در کلاس است، اما این دو یکی نیستند. این روش تعریف property در interface باعث نمی‌شود که auto-implement باشد بلکه این تنها مشخص کننده‌ی نام و نوع property است. همچنین اجازه تغییر access modifier را در قسمت get و set ندارید.

یک interface می‌تواند indexer را نیز در خود داشته باشد. همانند قبل اگر تنها از get یا set استفاده کنید، indexer شما read-only یا write-only خواهد بود. همچنین مجاز به استفاده از access modifier در accessorهای indexer تعریف شده در interface نیستید.

فرادرس
FaraDars.org

Interface implementation

هنگامی‌که یک interface تعریف می‌شود، هر تعداد کلاس که شما مد نظر دارید می‌توانند این interface را پیاده‌سازی کنند. همچنین یک class می‌تواند به تعداد دلخواه interface پیاده‌سازی کند. برای اجرای یک interface، کلاس باید بدنه‌ی متدهای تعریف شده در interface را فراهم آورد. هر کلاس، آن‌طور که بخواهد برای اجرای این متدها(بدنه‌هایی که در کلاس خودش برای متدهای interface آماده کرده است) اقدام می‌کند. بنابراین دو کلاس می‌توانند یک interface را به روش‌های مختلفی اجرا کنند اما هر دو کلاس شامل تمام متدهایی که در interface مشخص شده است می‌باشند.

interface

```
interface IFighter
{
  void Punch(int side);
  void Kick (int side);
  void Block();
}
```

class implements interface

implement methods

```
class Soldier : IFighter
{
  public void Punch(int side)
  {
    Move(arms[side], Forward);
  }

  public void Kick (int side)
  {
    Move(legs[side], Forward);
  }

  public void Block()
  {
    Move(arms[Left ], Up);
    Move(arms[Right], Up);
  }
  ...
}
```

```
interface IFighter
{
  void Punch(int side);
  void Kick (int side);
  void Block();
}
```

```
interface IWrestler
{
  void Takedown(int legs);
  void Escape  ();
}
```

IFighter methods

IWrestler methods

```
class Soldier : IFighter, IWrestler
{
  public void Punch(int side) { ... }
  public void Kick (int side) { ... }
  public void Block() { ... }

  public void Takedown(int legs) { ... }
  public void Escape() { ... }

  ...
}
```

base class

interface

interface

```
class Soldier : Person, IFighter, IWrestler
{
  ...
}
```

FaraDars.org

استفاده از reference variableهای interface

شما در سی‌شارپ می‌توانید یک reference variable از interface تعریف کنید. به عبارت دیگر، در سی‌شارپ می‌توانید interface reference variable بسازید. این‌چنین متغیری می‌تواند به هر شیئی که interfaceاش را اجرا می‌کند رجوع کند. هنگامی‌که متد یک شیء را از طریق interface reference صدا می‌زنید، آن نسخه از متد که شیء مربوط به آن interface را پیاده‌سازی کرده است اجرا می‌شود. این پروسه شبیه به استفاده از base class reference برای دسترسی به شیء derived class است (که در قسمت‌های قبلی با آن آشنا شدید).

interface reference →

```
IFighter f;
```

interface reference refers to object of implementing class →

```
IFighter f = new Soldier();
```

can only call `IFighter` methods when using `IFighter` reference →

```
IFighter f = new Soldier();

f.Punch(Left);
f.Kick(Right);
f.Block();
...
```

فرادرس
FaraDars.org

Generic code

Generic code به برنامه‌ای گفته می‌شود که بطور کلی نوشته شده و برای موارد زیادی قابل استفاده می‌باشد. به عنوان مثال متد Warmup به عنوان ورودی مرجعی از نوع IFighter می‌گیرد، بنابراین اشیایی از تمام کلاس‌هایی که این interface را پیاده‌سازی می‌کنند را می‌توان به این متد ارسال کرد بنابراین این متد یک کد کلی می‌باشد.

generic code, works for all `IFighter` types

```
void WarmUp(IFighter f)
{
  f.Punch(Left );
  f.Punch(Right);
  f.Kick (Left );
  f.Kick (Right);
}
```

```
Soldier s = new Soldier();

WarmUp(s);
```

ok since `Soldier` implements `IFighter`

```
class Monkey : IFighter
{
  ...
}
```

implement `IFighter`

```
class Robot : IFighter
{
  ...
}
```

```
Monkey m = new Monkey();
Robot  r = new Robot ();

WarmUp(m);
WarmUp(r);
```

ok, both implement `IFighter`

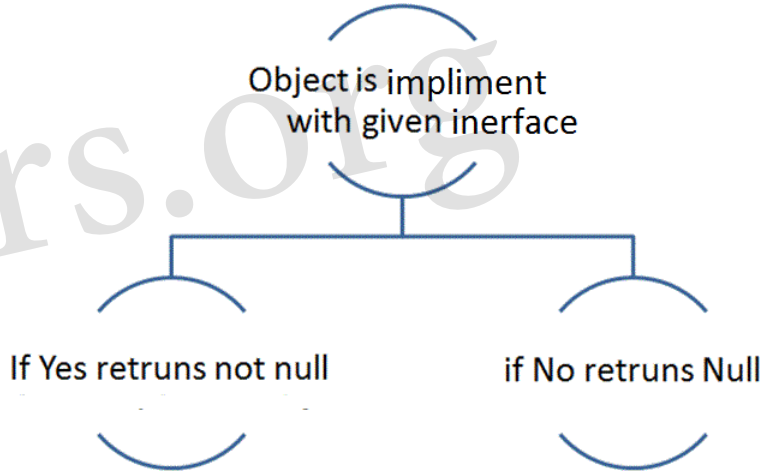
FaraDars.org

Type testing

به کمک عملگر is می‌توان تشخیص داد که آیا یک مرجع یک interface را پیاده‌سازی کرده یا خیر. به عنوان مثال آیا مرجع s که از نوع کلاس Soldier می باشد intefcace بنام Ifighter را پیاده‌سازی کرده است یا خیر.

test if `Soldier` implements `IFighter`

```
void March(Soldier s)
{
   if (s is IFighter)
      ...
}
```



فرادرس
FaraDars.org

Interface inheritance ارث بری واسطها

یک interface می‌تواند از یک interface دیگر ارث‌بری کند. برای انجام این امر، از syntax مشابه ارث‌بری در کلاس‌ها استفاده می‌شود. هنگامی‌که که یک کلاس قصد اجرای interfaceای را دارد که آن interface از interface دیگری ارث بری کرده است، کلاس باید تمام اعضای تعریف شده در زنجیره‌ی ارث‌بری را پیاده‌سازی کند. هنگامی که یک interface از interface دیگری ارث‌بری می‌کند این امکان وجود دارد که در derived interface یک عضو تعریف شود و این عضو با یکی از اعضای base interface هم‌نام باشد. در این مواقع عضو موجود در base interface دیگر دیده نمی‌شود و شما یک پیغام هشدار را خواهید دید. برای رفع پیغام هشدار می‌توانید قبل از تعریف آن عضو در derived interface، از کلمه‌ی کلیدی new استفاده کنید.

base →
```
interface IFighter
{
  void Punch(int side);
  void Kick (int side);
  void Block();
}
```

derived →
```
interface IStreetFighter : IFighter
{
  void Bite();
}
```

```
interface IFighter
{
  void Punch(int side);
  void Kick (int side);
  void Block();
}
```

```
interface IWrestler
{
  void Takedown(int legs);
  void Escape  ();
}
```

multiple inheritance allowed →
```
interface IStreetFighter : IFighter, IWrestler
{
  void Bite();
}
```

from IFighter →
from IWrestler →
from IStreetFighter →
```
class Soldier : IStreetFighter
{
  public void Punch(int side) { ... }
  public void Kick (int side) { ... }
  public void Block()         { ... }

  public void Takedown(int legs) { ... }
  public void Escape()           { ... }

  public void Bite() { ... }

  ...
}
```

FaraDars.org

ابهام(Ambiguity)

هنگامی‌که یکی از اعضای interface را پیاده‌سازی می‌کنید، می‌توانید نام آن عضو را به همراه نام interfaceاش بنویسید انجام این کار باعث ساختن explicit interface member می‌شود. ساختن explicit implementation یا به‌طور خلاصه implementation از interface method می‌تواند دو دلیل داشته باشد: **۱.** هنگامی‌که یک interface method را از طریق explicit implementation می‌سازید، متد ساخته شده از طریق اشیای کلاس قابل دسترسی نخواهد بود بلکه از طریق interface reference به آن دسترسی خواهید داشت. از این‌رو، explicit implementation روش دیگری برای پیاده‌سازی interface method است اما این متد، دیگر یک عضو public از کلاس‌تان نیست. **۲.** برای یک کلاس امکان‌پذیر است که دو interface را پیاده‌سازی(implement) کند و این امکان وجود دارد که هردوی آن‌ها متدهایی با یک نام و یک signature داشته باشند در این موارد استفاده از explicit implementation باعث رفع ابهام می‌شود چراکه شما قبل از نام متد، نام interface آن را نیز مشخص می‌کنید.

```
interface IFighter
{
  void Block();
  ...
}
```

different interfaces

same signature

```
interface IWrestler
{
  void Block();
  ...
}
```

interfaces contain method with same signature

class codes one method only

```
class Soldier : IFighter, IWrestler
{
  public void Block()
  {
    ...
  }
  ...
}
```

interfaces contain 2 methods with same signature

IFighter version

IWrestler version

```
class Soldier : IFighter, IWrestler
{
  void IFighter.Block()   { ... }

  void IWrestler.Block() { ... }

  ...
}
```

Soldier reference

error, ambiguous to call Block since two versions exist

```
Soldier s = new Soldier();

s.Block();

...
```

IFighter reference

calls IFighter.Block

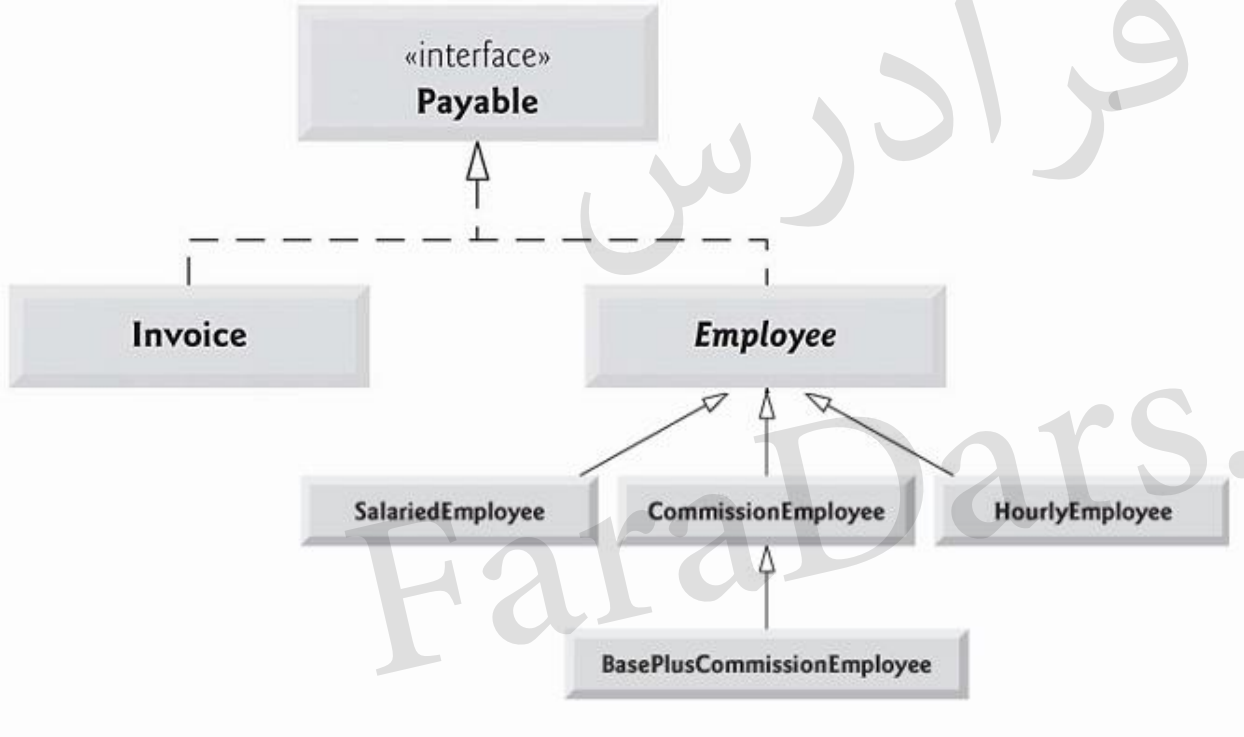
IWrestler reference

calls IWrestler.Block

```
Soldier s = new Soldier();

IFighter f = s;
f.Block();

IWrestler w = s;
w.Block();

...
```

فرادرس
FaraDars.org

مثال کاربردی: سیستم پرداخت یکسان



همانگونه که در فصل چندریختی مشاهده کردید، سیستم پرداخت حقوق و دستمزد شرکت بر پایه چندریختی طراحی گردید. فرض کنید شرکت مایل به پیاده‌سازی برنامه‌ای است تا تمام محاسبات پرداخت‌ها از قبیل پرداخت حقوق و یا پرداخت مبالغ فاکتورها و سایر پرداخت‌ها را به روش چند ریختی و یکسان در سیستم انجام دهد. با اینکه پرداخت‌ها معانی متفاوتی دارند، لیکن ابتدا یک interface با یک متد جهت پرداخت تعریف می‌کنیم و تمام کلاس‌های مرتبط با هر نوع پرداخت، interface مذکور را پیاده‌سازی می‌کنند.

فرادرس
FaraDars.org

مثال کاربردی: سیستم پرداخت

ابتدا یک interface بنام Ipayable ایجاد می‌کنیم. این interface قالب یک متد بنام GetPaymentAmount را تعریف می‌کند که ورودی ندارد و مبلغ پرداخت را برمی‌گرداند. کلاس Employee واسط IPayable را پیاده‌سازی می‌کند بنابراین باید متد مذکور را پیاده‌سازی نماید اما چون برای پیاده‌سازی این متد در کلاس Employee اطلاعات کافی موجود نبوده است بنابراین به شکل abstract تعریف شده و تمام کلاس‌هایی که Employee را به ارث می‌برند همانند کلاس SalariedEmployee مؤظف به پیاده‌سازی این متد هستند.

```
public interface IPayable
{
    4 references
    decimal GetPaymentAmount();
}
```

```
public abstract class Employee : IPayable
{
    2 references
    public string FirstName { get; private set; }
    2 references
    public string LastName { get; private set; }
    2 references
    public string SocialSecurityNumber { get; private set; }
    1 reference
    public Employee( string first, string last, string ssn )...
    3 references
    public override string ToString()...
    4 references
    public abstract decimal GetPaymentAmount();
}
```

```
public class SalariedEmployee : Employee
{
    private decimal weeklySalary;
    2 references
    public SalariedEmployee(string first, string last, string ssn,decimal salary):base( first, last, ssn )...
    3 references
    public decimal WeeklySalary...
    4 references
    public override decimal GetPaymentAmount()
    {
        return WeeklySalary;
    }
    3 references
    public override string ToString()...
}
```

FaraDars.org

مثال کاربردی: سیستم پرداخت یکسان

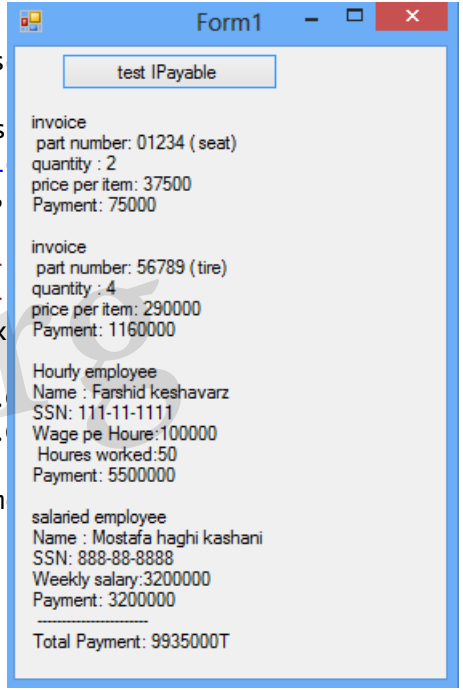
```
public class Invoice : IPayable
{
    private int quantity;
    private decimal pricePerItem;
    2 references
    public string PartNumber { get; set; }
    2 references
    public string PartDescription { get; set; }
    2 references
    public Invoice( string part, string description, int count,decimal price )...
    3 references
    public int Quantity...
    3 references
    public decimal PricePerItem...
    3 references
    public override string ToString()...
    4 references
    public decimal GetPaymentAmount()...
}
```

همانگونه که می‌بینید کلاس Invoice یا همان فاکتور نیز واسط Ipayable را پیاده‌سازی کرده است و بنابراین دارای متد GetPaymentAmount می‌باشد و باید آنرا مطابق کاربرد خود پیاده‌سازی نماید. تمام کلاس‌هایی که Interface مذکور را پیاده‌سازی می‌کنند دارای متدی بنام GetPaymentAmount هستند که مبلغ پرداخت را تعیین می‌کند، بنابراین برنامه‌های یکسان برای تمام موجودیت‌ها ایجاد می‌گردد.

فرادرس
FaraDars.org

مثال کاربردی: سیستم پرداخت یکسان-نمونه خروجی

```
private void button1_Click(object s

IPayable[] payableObjects
doubl
payableObjects[0] = new Invoice("01234",
payableObjects[1] = new Invoice("56789",
payableObjects[2] = new HourlyEmployee("Farshid", "keshavarz", "111-11-
payableObjects[3] = new SalariedEmployee("Mostafa", "haghi kashani", "888-
foreach (IPayable x

label1.Text += x.ToString() + "\nPayment: " + x.
TotalPayment += x.

label1.Text += "\n ----------------------\nTotal Payment: " + TotalPaym
```



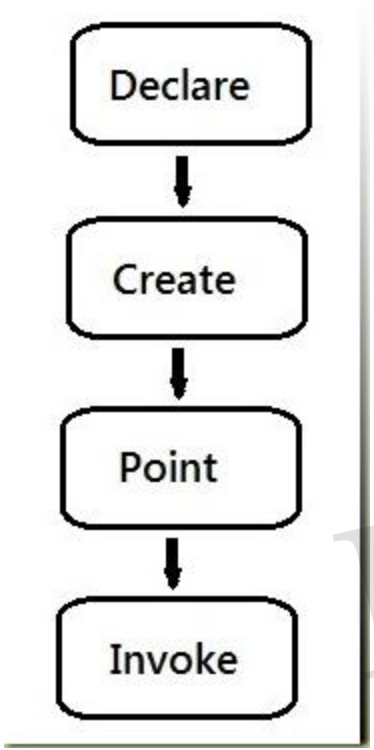
فرادرس
FaraDars.org

Delegates

به زبان ساده یک delegate برابر است با شیئی که می‌تواند به یک method رجوع کند. بنابراین هنگامی‌که یک delegate می‌سازید، در واقع یک object را به‌وجود می‌آورید که می‌تواند reference به یک متد را در خودش نگاه دارد. از این‌رو، متد می‌تواند از طریق این reference فراخوانی شود. Delegates می‌تواند شی باشد که شامل لیستی از متدهای یکسان(هم امضا و دارای مقادیر بازگشتی یکسان). delegate موجب می‌شود تا برنامه شما بتواند در runtime(زمان اجرا) متدها را اجرا کند بدون اینکه بداند آن متدها در compile time چه چیزی هستند. لازم به ذکر است که به لیست Reference Methods مربوط به یک delegate اصطلاحاً Invocation List گفته می‌شود.

فرادرس
FaraDars.org

Step By Step Creating and Using the Delegate



Step 1 Declare a Delegate

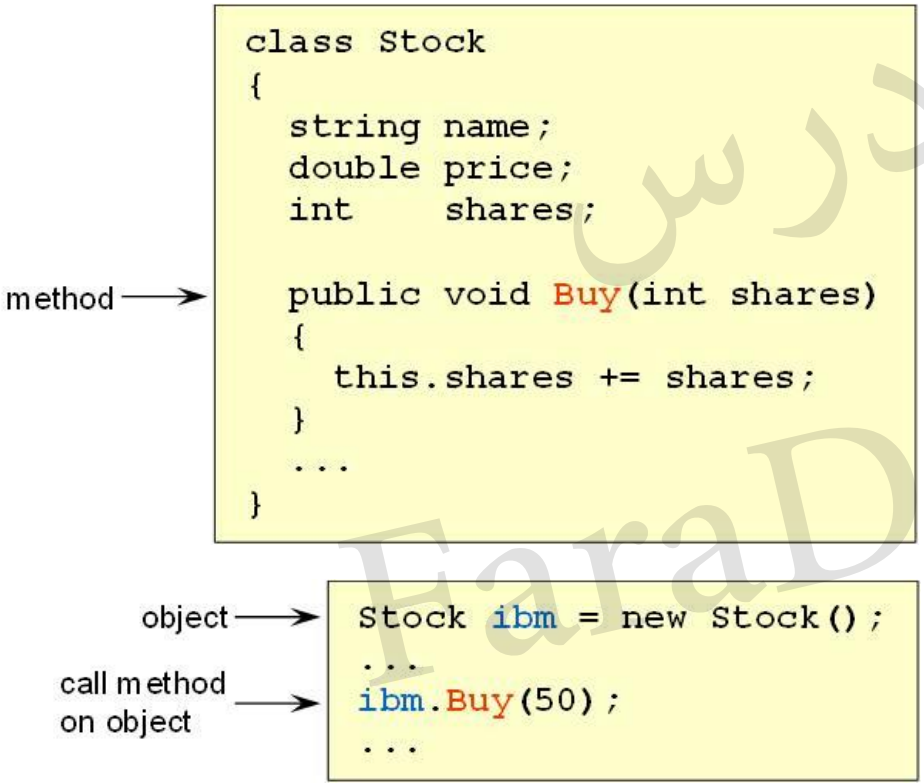
Step 2 Create a Delegate reference

Step 3 Point the reference pointer to methods

Step 4 Invoke the methods through delegate

فرادرس
FaraDars.org

Background - object-oriented method call



روش فراخوانی متدهای اشیاء به شکل مقابل می‌باشد.

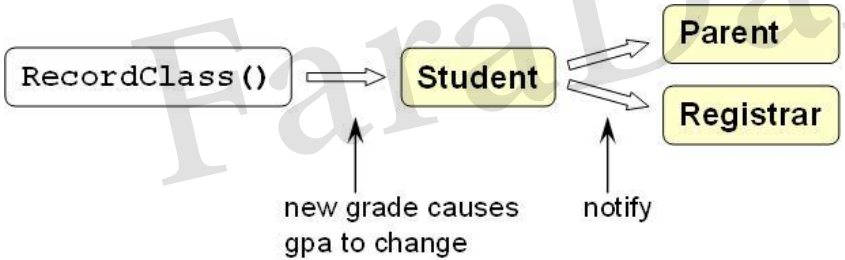
فرادرس
FaraDars.org

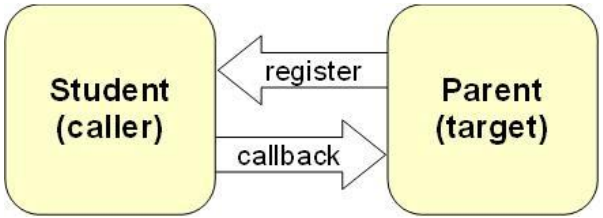
آموزش جامع شیئ گرایی
در سی شارپ
faradars.org/fvcs9404

Notification

store state →
change state →

```
class Student
{
  string name;
  double gpa;
  int    units;

  public void RecordClass(int grade)
  {
    gpa = (gpa * units + grade) / (units + 1);

    units++;
  }
  ...
}
```





فرادرس
FaraDars.org

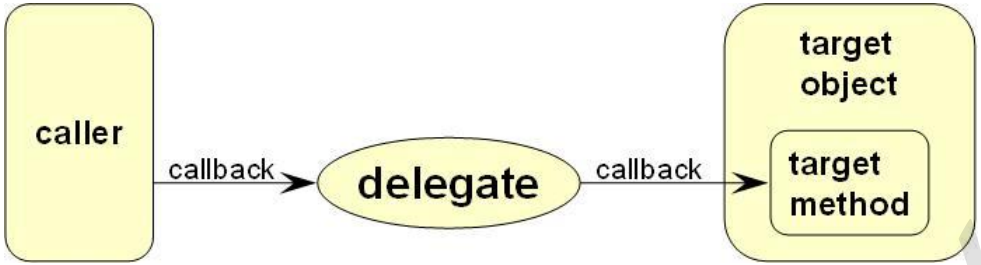
Delegate definition

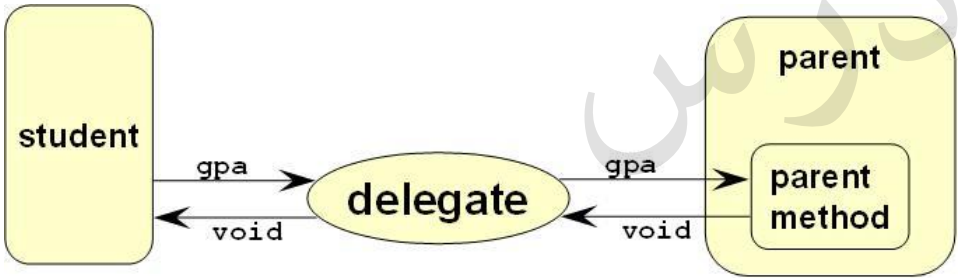
فرم کلی delegate به صورت زیر است:

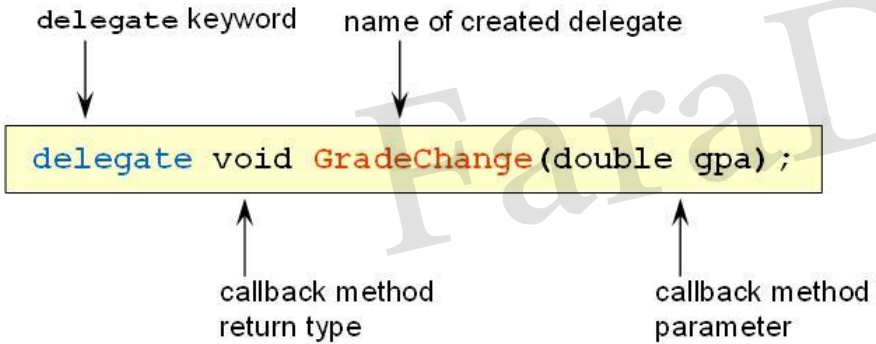
```
delegate ret-type name(parameter-list);
```

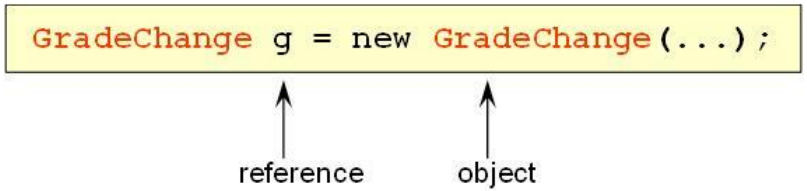
در این‌جا ret-type نوع بازگشتی متدی است که delegate آن را فراخوانی می‌کند. Name برابر با نام delegate است. پارامترهای مورد نیاز متد که از طریق delegate فراخوانی می‌شوند در قسمت parameter-list قرار می‌گیرد.

هنگامی‌که یک شیء از delegate می‌سازید، نام متد مربوطه را (تنها نام متد، بدون پارامتر) به delegate می‌دهیم:



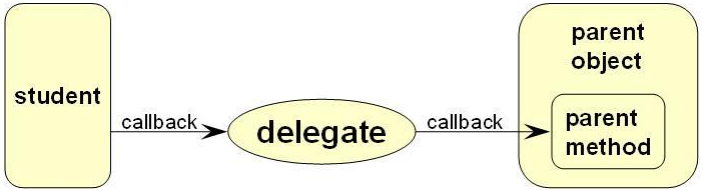






فرادرس
FaraDars.org

Delegate use



همانطور که در کد مقابل می‌بینید، امضای متد Report با تعریف GradeChange مطابقت دارد، پس این متد می‌تواند به Invocation List آبجکت Targetsما اضافه شود. هنگامی‌که یک شیء از delegate می‌سازید، نام متد مربوطه را(تنها نام متد، بدون پارامتر) به delegate می‌دهیم.

delegate defines required signature

```
delegate void GradeChange(double gpa);
```

method signature matches delegate

```
class Parent
{
  public void Report(double gpa)
  {
    ...
  }
}
```

delegate reference

```
class Student
{
  public GradeChange Targets;
  ...
}
```

caller

target

delegate

register

```
void Run()
{
  Student ann = new Student("Ann");

  Parent  mom = new Parent();

  GradeChange g = new GradeChange(mom.Report);

  ann.Targets = g;
  ...
}
```

target object

target method

reference to delegate

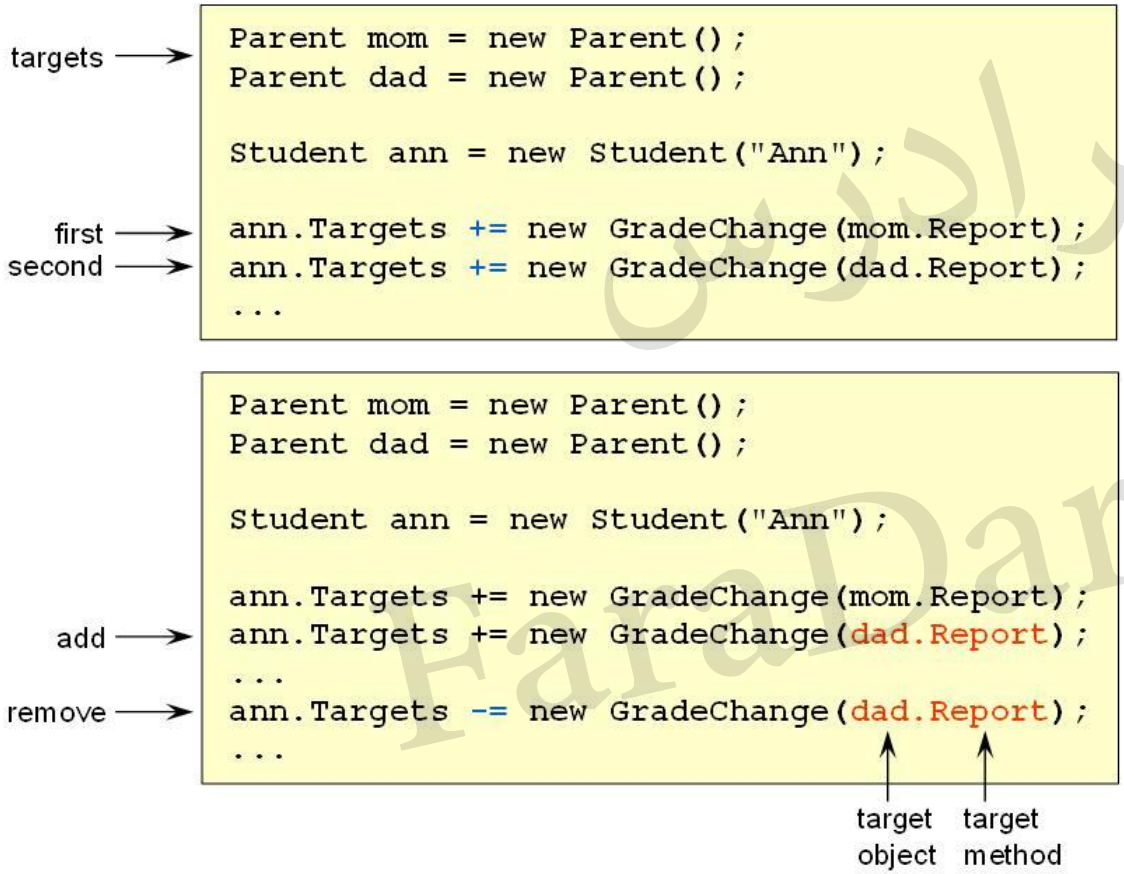
invoke callback through delegate

```
class Student
{
  public GradeChange Targets;

  public void RecordClass(int grade)
  {
    // update gpa
    ...
    Targets(gpa);
  }
}
```

callback takes double argument

فرادرس
FaraDars.Org

Multiple targets



توجه داشته باشید که هیچ لزومی ندارد شما توابع اشیاء یک کلاس خاص را در داخل یک شیء Delegate ثبت نمایید. شما می‌توانید توابع مربوط به اشیائی که از کلاس‌های مختلفی ایجاد شده‌اند را نیز تنها به شرط اینکه پارامترهای ورودی و خروجی آنها یکسان باشند در داخل شیء Delegate تبت نمایید.

دقت کنید که برای ثبت اولین تابع درداخل یک شیء Delegate، می‌توان از عملگر = (مساوی) استفاده کنید. در صورتیکه بخواهید توابع دیگری را نیز در داخل همان شیء Delegate ثبت نمایید، باید از عملگر += (بعلاوه مساوی) برای این منظور استفاده نمایید. لازم به ذکر است که اگر در این حالت سهواً به جای استفاده از عملگر += از عملگر = استفاده کنید، تمامی توابع از قبل ثبت شده در داخل شیء Delegate به طور خودکار Unregistered می‌گردند!

در صورتیکه تابعی از یک شی را در داخل یک شیء Delegate ثبت کرده‌اید و بخواهید که شیء مذکور را از بین ببرید، ابتدا باید تابع مربوطه را با استفاده از عملگر -= (منها مساوی) Unregistered نموده و سپس نسبت به از بین بردن آن شیء اقدام نمایید:

فرادرس
FaraDars.org

مثال

```
public delegate void GradeChange(double avg);
```

خط اول تعریف Delegate می‌باشد.

کلاس Student همان caller یا Publisher می‌باشد.

```
3 references
public class student
{
    string fullname;
    double gpa;
    int units;
    0 references
    public string FullName...
    1 reference
    public student(string name,double avg=0,int u = 0)...
    1 reference
    public void RecordClass(double grade)
    {
        gpa = (gpa * units + grade) / (units + 1);
        units++;
        Targets(gpa);
    }
    public GradeChange Targets;
}
```

فرادرس
FaraDars.org

مثال

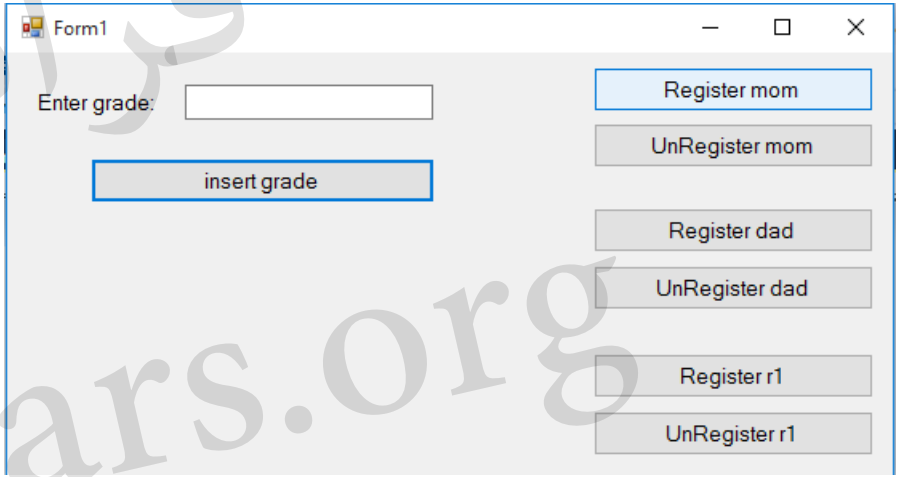
کلاس Parent و کلاس Registarar همان Targetsها یا همان Subscripbers هستند.

```
class parent
{
    string fname;
    2 references
    public parent(string n)...
    4 references
    public void reported( double avg)
    {
        string s;
        s = "I m " + fname + "\nOne of parrents" + "\nMy chid gpa has reported to me and that is:"+avg.ToString();
        System.Windows.Forms.MessageBox.Show(s);
    }
}
//--------------------
2 references
class registrar
{
    2 references
    public void log( double avg)
    {
        string s = "I'm the registrar \nI've loged your gpa\nYour gpa is: " + avg.ToString();
        System.Windows.Forms.MessageBox.Show(s);
    }
}
```

فرادرس
FaraDars.org

مثال

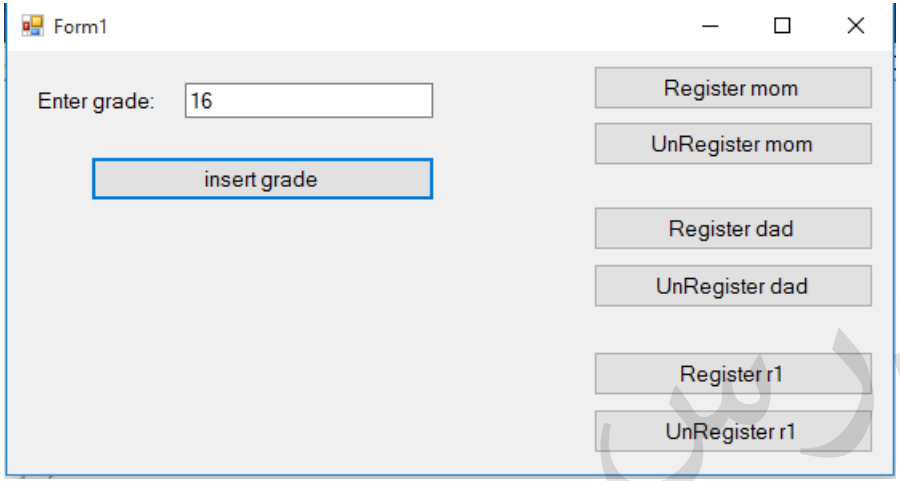
```
public partial class Form1 : Form
{
    student s = new student("ali", 0, 0);
    parent dad = new parent("reza");
    parent mom = new parent("maryam");
    registrar r1 = new registrar();
    1 reference
    public Form1()...
    1 reference
    private void BtnRegMom_Click(object sender, EventArgs e)
    {
        s.Targets += new GradeChange(mom.reported);
    }
    1 reference
    private void BtnRegDad_Click(object sender, EventArgs e)
    {
        s.Targets += new GradeChange(dad.reported);
    }
    1 reference
    private void BtnUnRegdad_Click(object sender, EventArgs e)
    {
        s.Targets -= new GradeChange(dad.reported);
    }
    1 reference
    private void BtnRegR1_Click(object sender, EventArgs e)
    {
        s.Targets += new GradeChange(r1.log);
    }
     1 reference
     private void BtnUnRegR1_Click(object sender, EventArgs e)
    {
        s.Targets -= new GradeChange(r1.log);
    }
}
```



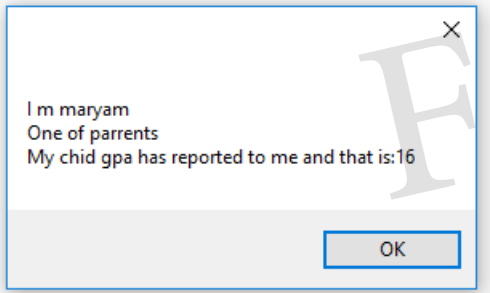
فرادرس
FaraDars.org

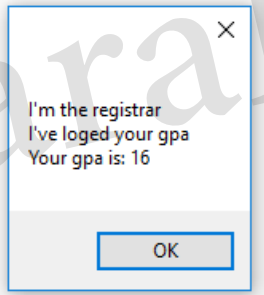
مثال

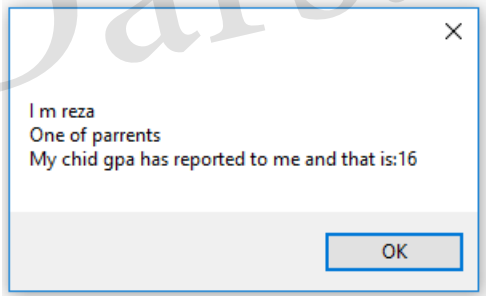
حال اگر آبجکت delegate مورد نظر را Invoke کنیم، تمام متدهایی که در Invocation List آن موجود هستند اجرا می‌شوند، نحوه‌ی Invoke کردن یک آبجکت delegate مانند فراخوانی یک متد و ارسال پارامتر(در صورت نیاز) به آن می‌باشد. درون متد RecordClass از کلاس Student آبجکت delegate مورد نظر را Invoke می‌کنیم.




```
private void BtnInsertGrade_Click(object sender, EventArgs e)
{
    s.RecordClass(Convert.ToDouble(textBox1.Text));
}
```







فرادرس
FaraDars.org

Null reference

reference field defaults to null

test before call

```
class Student
{
  public GradeChange Targets;

  public void RecordClass(int grade)
  {
    // update gpa
    ...
    if (Targets != null)
      Targets(gpa);
  }
```

فرادرس
FaraDars.org

مثال از Null reference

```
public class student
{
    string fullname;
    double gpa;
    int units;
    0 references
    public string FullName...
    1 reference
    public student(string name,double
    1 reference
    public void RecordClass(double gra
    {
        gpa = (gpa * units + grade) /
        units++;
        Targets(gpa);
    }
    public GradeChange Targets;
}
```

NullReferenceException was unhandled

An unhandled exception of type 'System.NullReferenceException' occurred in EventsANDDelegate.exe

Additional information: Object reference not set to an instance of an object.

Troubleshooting tips:

- Check to determine if the object is null before calling the method.
- Use the "new" keyword to create an object instance.
- Get general help for this exception.

Search for more Help Online...

Exception settings:

Break when this exception type is thrown

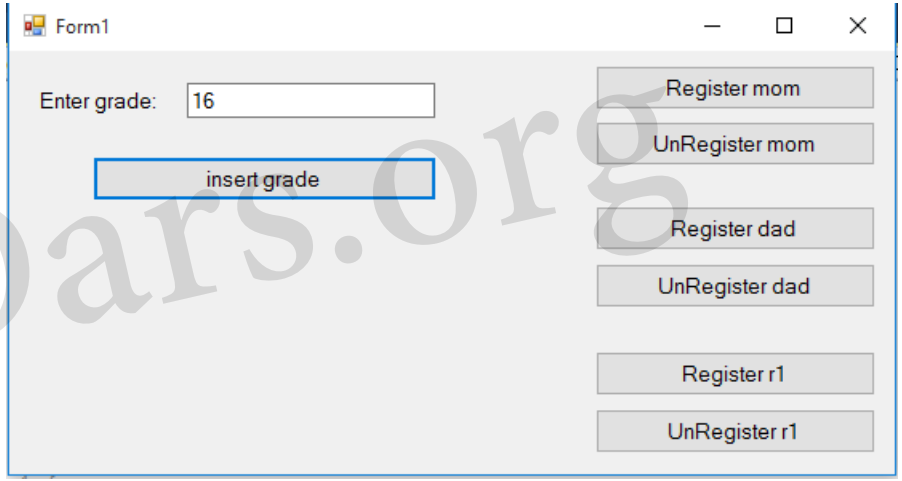
Actions:

View Detail...

Copy exception detail to the clipboard

Open exception settings

```
public class student
{
    string fullname;
    double gpa;
    int units;
    0 references
    public string FullName...
    1 reference
    public student(string name,double avg=0,int u = 0)...
    1 reference
    public void RecordClass(double grade)
    {
        gpa = (gpa * units + grade) / (units + 1);
        units++;
        if(Targets!=null)
        Targets(gpa);
    }
    public GradeChange Targets;
}
```



فرادرس
FaraDars.org

Static methods

دقت کنید که شما قادر به ثبت توابع استاتیک(Static) کلاس‌ها نیز در داخل اشیاء Delegate خواهید بود و همانطور که عنوان کردیم تنها شرط آن این است که پارامترهای ورودی و خروجی این توابع استاتیک نیز با ساختار کلاس Delegate مطابقت داشته باشند. برای دسترسی به توابع static از نام کلاس استفاده می‌کنیم

static method →

```
class Registrar
{
  public static void Log(double gpa)
  {
    ...
  }
}
```

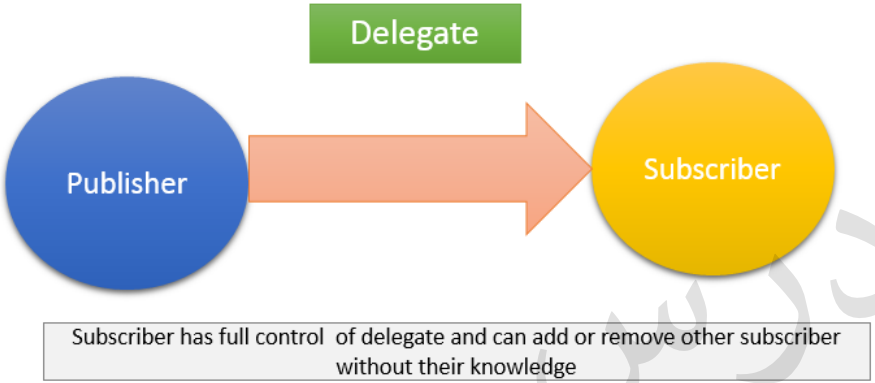
register →

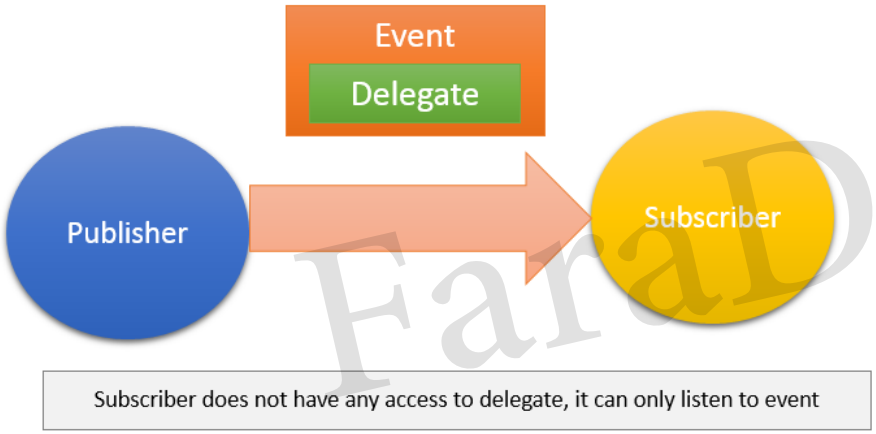
```
void Run()
{
  Student ann = new Student("Ann");

  ann.Targets = new GradeChange(Registrar.Log);
  ...
}
```

فرادرس
FaraDars.org

Events

یک object می‌تواند برای یک event تعدادی event handler را register کند و هنگامی‌که یک event اتفاق می‌افتد، تمامی handlerهای register شده فراخوانی می‌شوند. Event handlerها باید مطابق با delegate باشند.

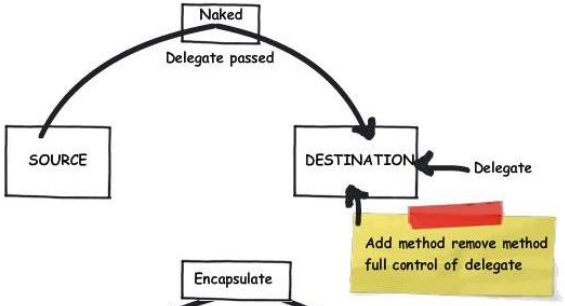


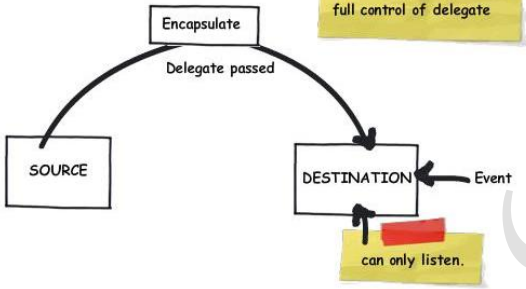


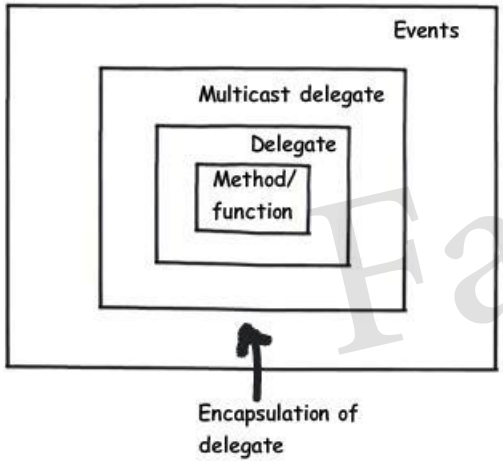
فرادرس
FaraDars.org

Events

یک object می‌تواند برای یک event تعدادی event handler را register کند و هنگامی‌که یک event اتفاق می‌افتد(Event Raise)، تمامی handlerهای register شده فراخوانی می‌شوند. Event handlerها باید مطابق با delegate باشند.







فرادرس
FaraDars.org

Events

یک object می‌تواند برای یک event تعدادی event handler را register کند و هنگامی‌که یک event اتفاق می‌افتد، تمامی handlerهای register شده فراخوانی می‌شوند. Event handlerها باید مطابق با delegate باشند.

public delegate →

```
class Student
{
  public GradeChange Targets;
  ...
}
```

```
Parent  mom = new Parent();
Parent  dad = new Parent();
Student ann = new Student("Ann");
...
ann.Targets = new GradeChange(mom.Report);
ann.Targets = new GradeChange(dad.Report);
...
ann.Targets(4.0);
...
```

overwrite mom handler →

invoke →

event →

```
class Student
{
  public event GradeChange Targets;
  ...
}
```

```
Parent  mom = new Parent();
Student ann = new Student("Ann");

ann.Targets += new GradeChange(mom.Report);
ann.Targets -= new GradeChange(mom.Report);

ann.Targets  = new GradeChange(mom.Report);
ann.Targets(4.0);
...
```

ok to use += →

ok to use -= →

error to use = →

error to invoke →

فرادرس
FaraDars.org

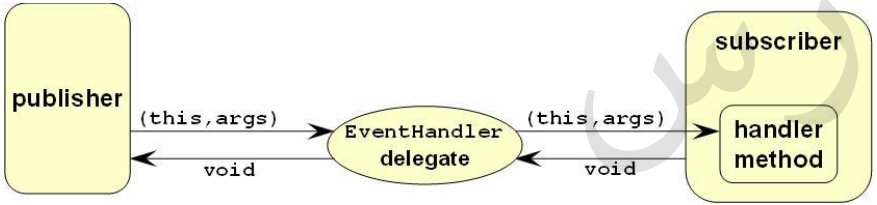
Applications

```
delegate void EventHandler(object sender, EventArgs args);
```

nothing returned to event publisher

object that published event

event details



Click event defined in base Control class

```
public class Control ...
{
   public event EventHandler Click;
   ...
}
```

inherits Click event

```
public class Button : Control
{
...
}
```

GUI contains several buttons

callback method for ok button

register with Click event of ok button

```
public class MyForm : Form
{
   private Button ok;
   private Button cancel;
   private Button help;

   void callback(object sender, EventArgs args)
   {
      ...
   }

   public MyForm()
   {
      ...
      ok.Click += new EventHandler(this.callback);
      ...
   }
}
```

فرادرس
FaraDars.org

Applications

```
delegate void MouseEventHandler(object sender, MouseEventArgs e);
```

specific delegate for mouse events

mouse event details

info specific to mouse event

```
class MouseEventArgs : EventArgs
{
  public MouseButtons Button { get; }
  public int          Clicks { get; }
  public int          X      { get; }
  public int          Y      { get; }
  ...
}
```

mouse event

```
public class Control : Component ...
{
  public event MouseEventHandler MouseDown;
  ...
}
```

control offering mouse events

callback method

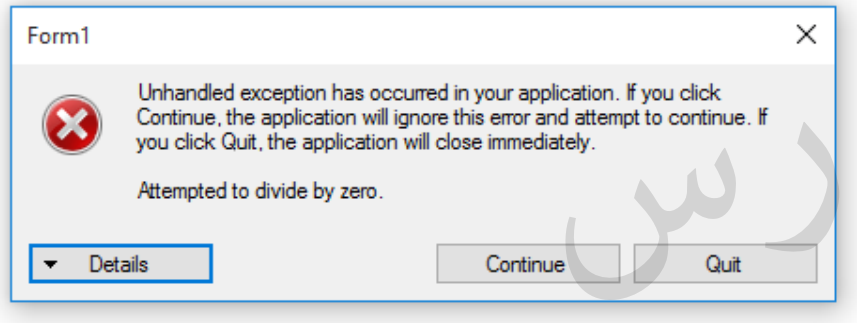
event details

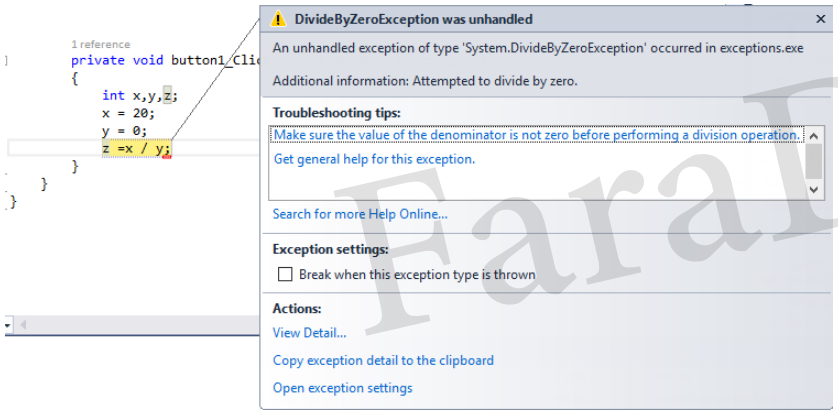
register

```
public class MyForm : Form
{
  private Panel drawingSurface;

  void mouseCallback(object sender, MouseEventArgs args)
  {
    ...
    int xCoord = args.X;
    int yCoord = args.Y;
    ...
  }

  public MyForm()
  {
    ...
    drawingSurface.MouseDown += new MouseEventHandler(this.mouseCallback);
    ...
  }
}
```

فرادرس
FaraDars.org

Exceptions

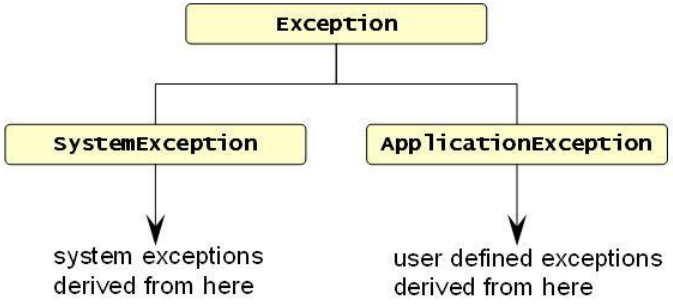


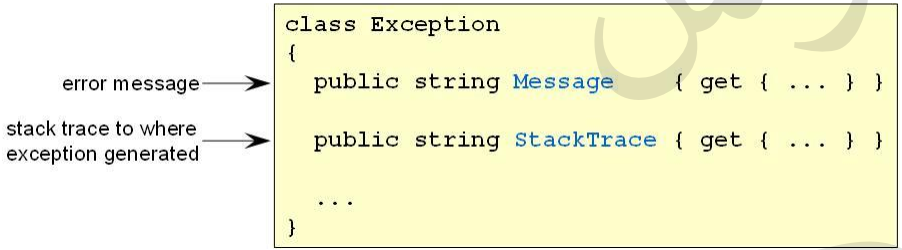


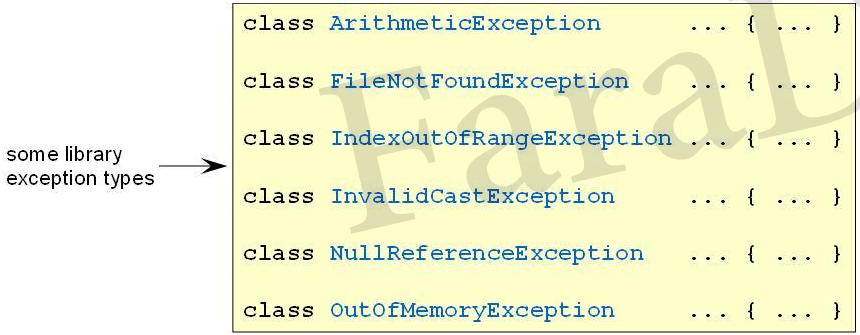
یک exception خطایی است که در runtime(زمان اجرا) اتفاق می‌افتد. با استفاده از زیرسیستم exception-handling در سی‌شارپ، شما می‌توانید از یک روش کنترل شده و سازمان‌یافته، خطاهای runtime را handle کنید. یکی از مزیت‌های اصلی exception handling این است که به‌طور خودکار خطاگیری را انجام می‌دهد و این درصورتی است که پیش از به‌وجود آمدن این ویژگی در برنامه‌نویسی، باید خودتان خطاگیری را انجام می‌دادید که هم خسته‌کننده و هم مستعد خطا بود. Exception handling یک بلاک کد(که exception handler نامیده می‌شود) تعریف می‌کند که هنگام بروز خطا به‌صورت خودکار اجرا می‌شود. بنابراین دیگر نیازی نیست که به‌صورت دستی موفق بودن یا عدم موفق بودن هر قسمت از برنامه را بررسی کنید. اگر یک خطا در runtime به‌وجود آید توسط exception handler بررسی خواهد شد. سی‌شارپ exceptionهای استاندارد را برای خطاهای رایج در یک برنامه(مانند خطاهای divide-by-zero و index-out-of-range) تعریف می‌کند که این موضوع یکی دیگر از دلایل اهمیت exception handling این است.

فرادرس
FaraDars.org

Exception hierarchy



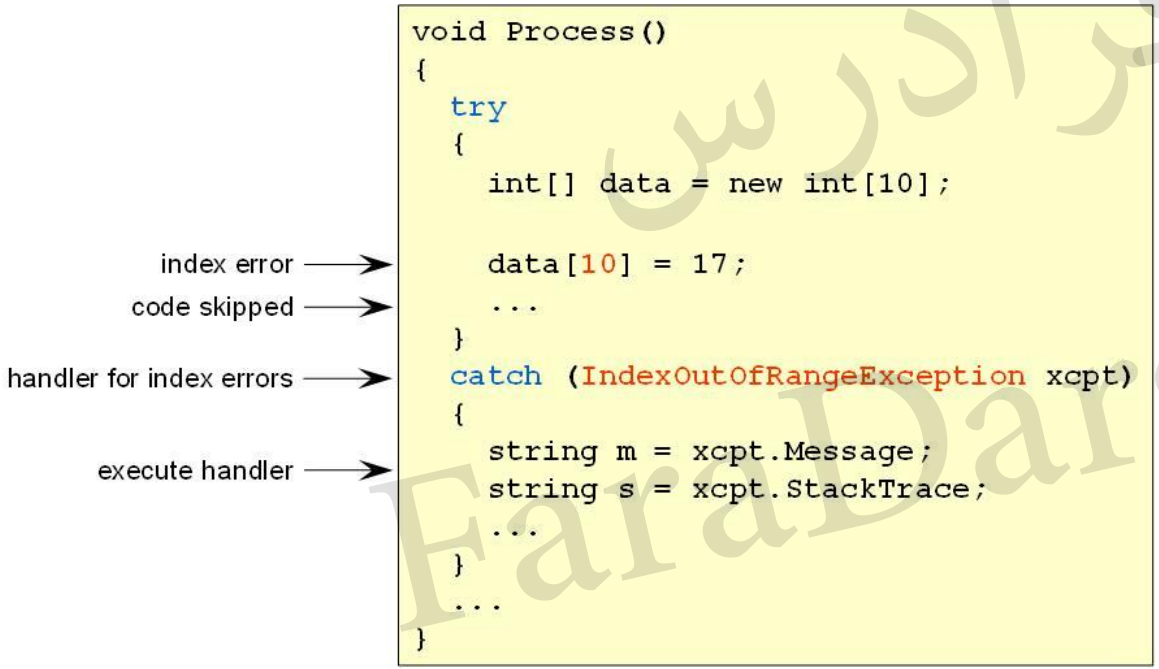




در سی‌شارپ exceptionها توسط کلاس‌ها ارائه می‌شوند. همه‌ی کلاس‌های exception(مثل کلاس‌های استاندارد دات‌نت برای خطاگیری) باید از کلاس Exception مشتق شوند که خودش بخشی از System namespace است. بنابراین همه‌ی exceptionها زیرکلاس Exception هستند. یکی از زیرکلاس‌های مهم Exception، کلاس SystemException است که مشخص‌کننده‌ی base class برای exceptionهای از پیش تعریف شده در System namespace است. کلاس SystemException چیزی را به کلاس Exception نمی‌افزاید بلکه فقط در صدر زنجیره‌ی exceptionهای استاندارد دات‌نت فریم‌ورک قرار می‌گیرد. دات‌نت فریم‌ورک exceptionهای توکار(built-in) بسیار زیادی را تعریف می‌کند که از SystemException ارث‌بری می‌کنند. برای مثال، هنگامی‌که خطای تقسیم بر صفر رخ می‌دهد، یک exception از نوع DivideByZeroExcepion به‌وجود می‌آید. به‌زودی متوجه خواهید شد که چگونه کلاس‌های exception خودتان را با ارث‌بری از کلاس Excepion بنویسید. تعدادی property دارد که سه عدد از مهم‌ترین آن‌ها Message، StackTrace وTargetSite هستند. این propertyها هرسه read-only هستند. Message شامل یک رشته است که ماهیت خطا را شرح می‌دهد. StackTrace شامل یک رشته است که این رشته شامل فراخوانی‌هایی است که منجر به خطا شده‌اند. TargetSite شامل یک شیء بوده که مشخص‌کننده متد تولیدکننده‌ی exception است. Exception Class همچنین شامل چندین متد است.

FaraDars.org

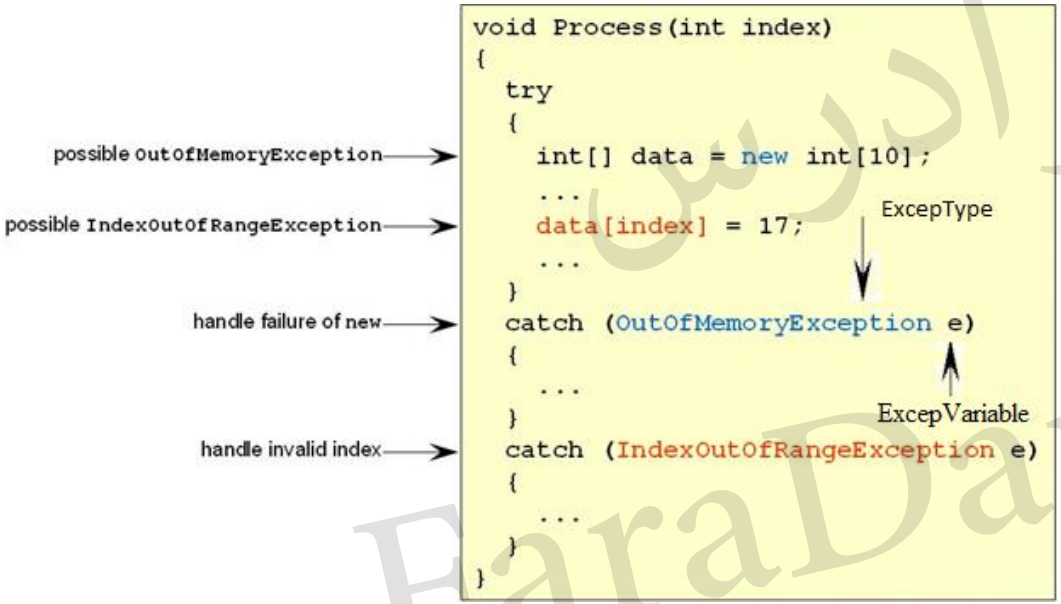
Handling an exception



در سی‌شارپ توسط چهار کلمه‌کلیدی try، catch، throwو finally مدیریت می‌شود. این‌ها یک زیرسیستم مرتبط را به‌وجود می‌آورند که استفاده از هرکدام، اشاره به استفاده از دیگری دارد. در طول بررسی مبحث exception handling هر کدام از کلمات کلیدی با جزییات توضیح داده خواهند شد اما نگاهی مختصر به وظایف هرکدام می‌تواند در این‌جا مفید واقع شود. آن قسمت از کدهای برنامه که قصد دارید خطاهای(exceptions) آن را بررسی کنید، درون try block قرار می‌گیرند. اگر یک exception درون try block رخ دهد، این exception(به اصطلاح) پرتاب(throw) می‌شود. کد شما می‌تواند این exception را در قسمت catch block دریافت و به روشی منطقی آن را handle کند. Exceptionهای استاندارد سیستم، خودشان به‌صورت خودکار throw می‌شوند اما برای throwکردن یک exception به‌صورت دستی باید از کلمه‌کلیدی throw استفاده کنید. هر کدی که در نهایت تحت هر شرایطی باید اجرا شود در قسمت finally block قرار می‌گیرد.

فرادرس
FaraDars.org

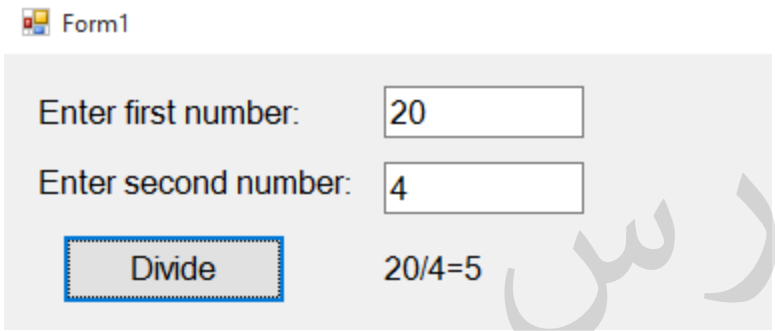
Multiple catch



در این‌جا، ExcepType نوع exceptionای می‌باشد که رخ داده است. هنگامی‌که یک exception پرتاب می‌شود، توسط جزء catch مرتبط با خودش گرفته شده و سپس exception در آن قسمت با یک روش منطقی handle می‌شود. همان‌طور که فرم کلی try/catch در شکل مقابل نشان می‌دهد، بیشتر از یک جزء catch می‌تواند به try وابسته باشد. در واقع نوع exception مشخص می‌کند که کدام catch باید اجرا شود. از این‌رو، هنگامی‌که یک exception با یک catch مطابقت داشت، فقط همان catch اجرا می‌شود و بقیه‌ی catchها نادیده گرفته می‌شوند. هنگامی‌که یک exception گرفته می‌شود، متغیر e مقدار آن را دریافت می‌کند. در واقع مشخص کردن e (exception variable) اختیاری است اگر exception handler نیازی به دسترسی به exception object نداشته باشد(که اغلب به همین صورت است)، نیازی به مشخص کردن e نیست و مشخص کردن نوع exception به تنهایی کفایت می‌کند به همین دلیل اکثر مثال‌هایی که می‌بینید فاقد e هستند. نکته‌ی مهم این است که اگر هیچ exceptionای پرتاب نشود، try block به‌صورت معمول اجرا خواهد شد و همه‌ی catchهای وابسته به آن نادیده گرفته می‌شوند و اجرا از آخرین catch به بعد ادامه می‌یابد. بنابراین تنها زمانی یک catch اجرا می‌شود که یک exception پرتاب شده باشد. استفاده از catch بدون مشخص کردن exception type یک روش برای گرفتن تمامی exceptionها است.

فرادرس
FaraDars.org

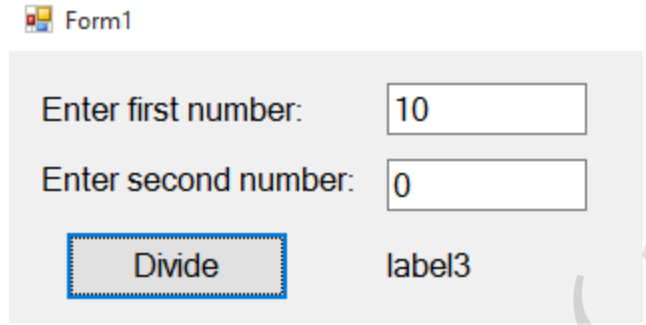
مثال مدیریت استثناء



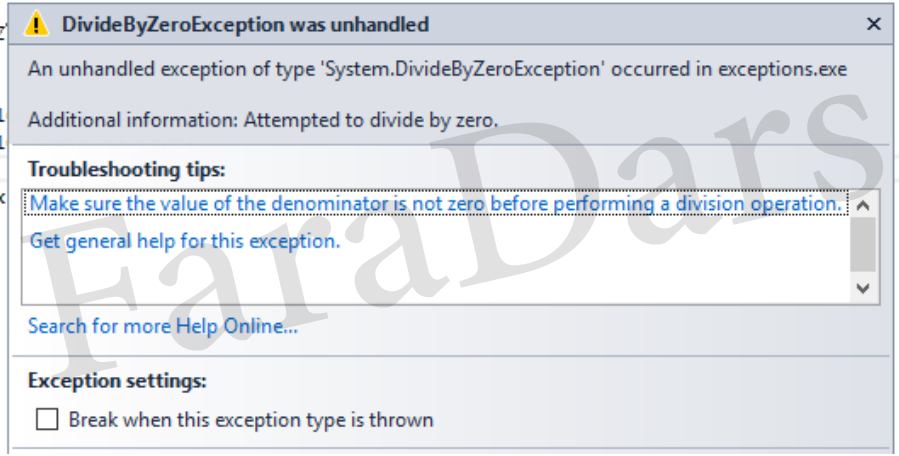

```
private void button1_Click(object sender, EventArgs e)
{
    int x,y,z;
    x = Convert.ToInt16(textBox1.Text);
    y = Convert.ToInt16(textBox2.Text);
    z = x / y;
    lblresult.Text = x.ToString() + "/" + y.ToString() + "=" + z.ToString();
}
```

فرادرس
FaraDars.org

مثال مدیریت استثناء



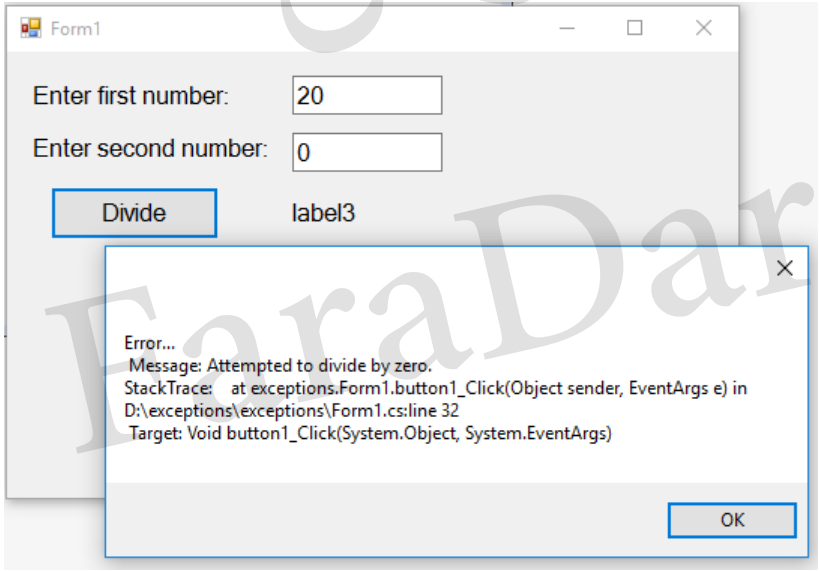

```
1 reference
private void button1_C
{
    int x,y,z;
    x = Convert.ToInt1
    y = Convert.ToInt1
    z = x / y;
    lblresult.Text = x
}
```



فرادرس
FaraDars.org

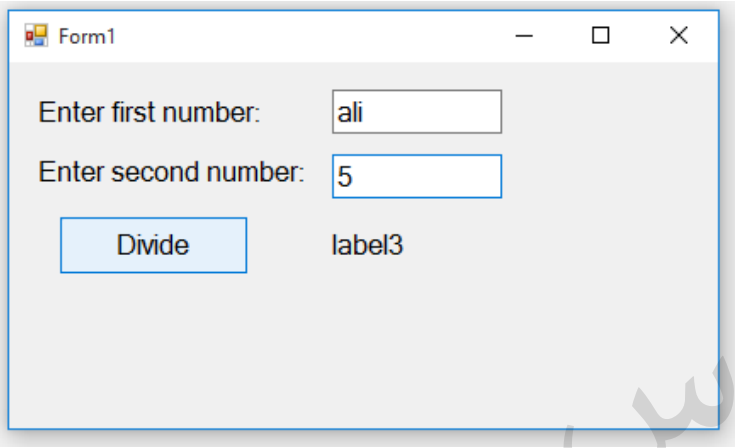
مثال مدیریت استثناء

```
private void button1_Click(object sender, EventArgs e)
{
    int x,y,z=0;
    x = Convert.ToInt16(textBox1.Text);
    y = Convert.ToInt16(textBox2.Text);
    try
    {
        z = x / y;
    }
    catch (DivideByZeroException err)
    {
        MessageBox.Show("Error..." + "\n Message: " + err.Message + "\nStackTrace: " + err.StackTrace+"\n Target: "+err.TargetSite.ToString());
    }
  lblresult.Text = x.ToString() + "/" + y.ToString() + "=" + z.ToString();
}
```



فرادرس
FaraDars.org

مثال مدیریت استثناء




```
1 reference
private void button1_Click(object sender, Even
{
    int x,y,z=0;
    x = Convert.ToInt16(textBox1.Text);
    y = Convert.ToInt16(textBox2.Text);
    try
    {
        z = x / y;
    }
    catch (DivideByZeroException err)
    {
        MessageBox.Show("Error..." + "\n Messa
    }
  lblresult.Text = x.ToString() + "/" + y.ToSt
}
```

FormatException was unhandled

An unhandled exception of type 'System.FormatException' occurred in mscorlib.dll

Additional information: Input string was not in a correct format.

Troubleshooting tips:

When converting a string to DateTime, parse the string to take the date before putting each variable into the DateTime object.

Make sure your method arguments are in the right format.

Get general help for this exception.

Search for more Help Online...

Exception settings:

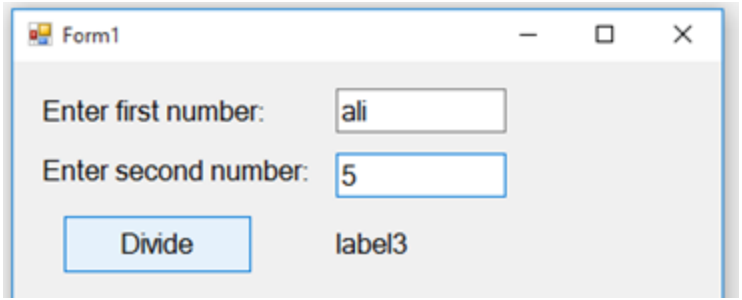
Break when this exception type is thrown

Actions:

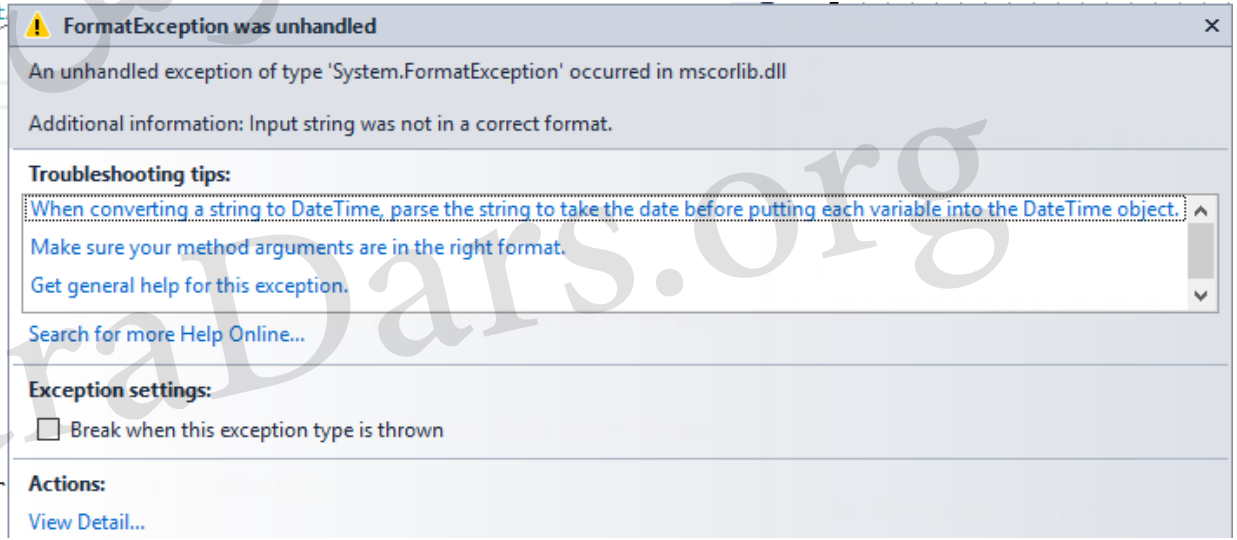
View Detail...

فرادرس
FaraDars.org

مثال مدیریت استثناء



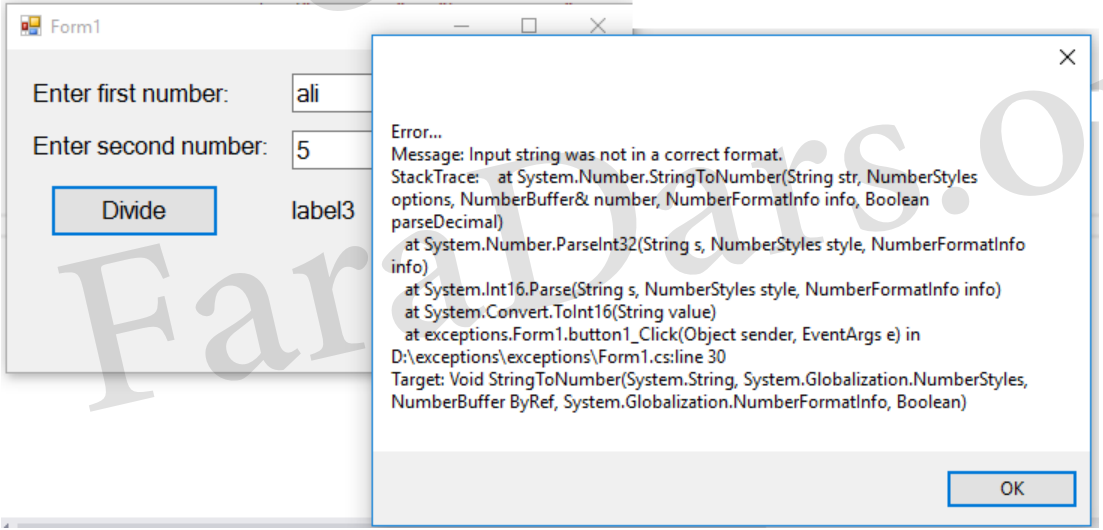

```
private void button1_Click(object sender, Event
{
    int x=0,y=0,z=0;
    x = Convert.ToInt16(textBox1.Text);
    y = Convert.ToInt16(textBox2.Text);
    try
    {
      z = x / y;
    }
    catch (DivideByZeroException err)
    {
      MessageBox.Show("Error..." + "\nMessage:
    }
    catch(FormatException err)
    {
      MessageBox.Show("Error..." + "\nMessage:
    }
  lblresult.Text = x.ToString() + "/" + y.ToStr
}
```



فرادرس
FaraDars.org

مثال مدیریت استثناء

```
private void button1_Click(object sender, EventArgs e)
{
    int x=0,y=0,z=0;
    try
    {
        x = Convert.ToInt16(textBox1.Text);
        y = Convert.ToInt16(textBox2.Text);
        z = x / y;
    }
    catch (DivideByZeroException err)
    {
      MessageBox.Show("Error..." + "\nMessage: " + err.Message + "\nStackTrace: "+ err.StackTrace+"\nTarget: "+err.TargetSite.ToString());
    }
    catch(FormatException err)
    {
      MessageBox.Show("Error..." + "\nMessage: " + err.Message + "\nStackTrace: "+ err.StackTrace+"\nTarget: "+err.TargetSite.ToString());
    }
  lblresult.Text = x.ToString() + "/" + y.ToString() + "=" + z.ToString();
}
```



فرادرس
FaraDars.org

Generating an exception

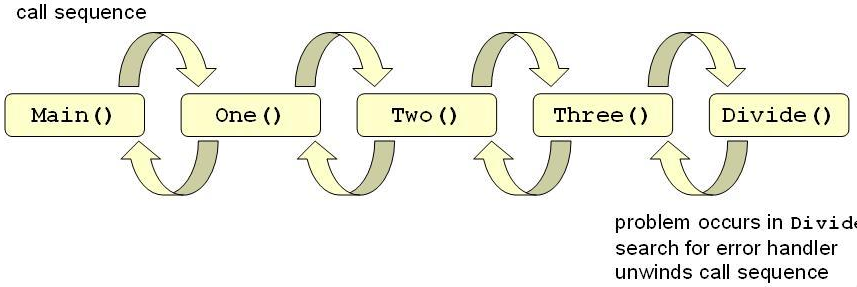
در مثال‌های قبل exceptionهایی که به‌طور خودکار توسط runtime system تولید شده بودند، گرفته می‌شدند. اما شما می‌توانید به‌صورت دستی یک exception را با استفاده از کلمه‌کلیدی throw پرتاب کنید. در این‌جا، مقدار جلوی throw باید یک شیء از یک کلاس با نوع exception باشد که از Exception ارث‌بری کرده است. همان‌طور که می‌بینید، DivideByZeroException در قسمت throw با استفاده از new ساخته شده است. به یاد داشته باشید که throw یک شیء را پرتاب می‌کند. بنابراین شما باید یک شیء برای آن بسازید تا آن را پرتاب کند. این بدان معناست که نمی‌توانید یک type را پرتاب کنید. در این مورد، برای ساخت شیء DivideByZeroException از default constructor استفاده شده است اما constructorهای دیگر نیز برای exceptionها موجود هستند. در اکثر موارد exceptionهایی که پرتاب می‌کنید اشیای exception classهایی هستند که خودتان ساخته‌اید. در ادامه‌ی این مبحث متوجه خواهید شد که چگونه exception classهای خودتان را بسازید.

throw →

```
int Divide(int numerator, int denominator)
{
  if (denominator == 0)
    throw new DivideByZeroException("Error: divide by zero");

  return numerator / denominator;
}
```

FaraDars.org

Locating a handler



به صورت پیش فرض مدیریت یک استثناء درون بلاکی‌ست که اتفاق افتاده است. در صورتیکه درون بلاک جاری مدیریت خطای مناسب اتفاق نیفتاده باشد سیستم برعکس مراحل فراخوانی بلاک‌ها به دنبال بلاک مدیریت کننده مناسب می‌گردد تا به بدنه اصلی برنامه یا همان main برسد. بیشتر برنامه‌نویسان از try block خارجی برای handle کردن خطاهایی که سخت قابل اصلاح کردن هستند استفاده می‌کنند و از try block داخلی برای درست کردن خطاهایی که راحت‌تر اصلاح می‌شوند، بهره می‌برند. شما همچنین می‌توانید از try block خارجی به‌عنوان catch all handler برای handle کردن errorهایی که در try block داخلی handle نشده‌اند، استفاده کنید.

no handler available

```
int Divide(int numerator, int denominator)
{
  if (denominator == 0)
    throw new DivideByZeroException("Error: divide by zero");

  return numerator / denominator;
}
```

matching handler →
will be executed →

```
void One()
{
  try
  {
    Two();
  }
  catch (DivideByZeroException e)
  {
    ...
  }
}
```

type of handler does not match →

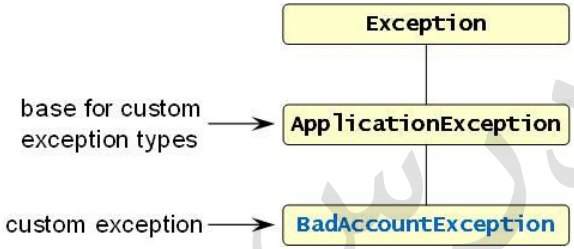
```
void Two()
{
  try
  {
    Three();
  }
  catch (IOException e)
  {
    ...
  }
}
```

no handler available

```
void Three()
{
  int q;

  q = Divide(3, 0);

  ...
}
```

FaraDars.org

Custom exception




```
class BadAccountException : ApplicationException
{
  public int id;

  public BadAccountException(string msg, int id)
    :base(msg)
  {
    this.id = id;
  }
}
```

custom data

base constructor

اگرچه exceptionهای built-in سی‌شارپ اکثر خطاهای رایج را handle می‌کنند، اما مکانیزم exception-handling در سی‌شارپ به این خطاها محدود نیست. در واقع، بخشی از قدرت رویکرد سی‌شارپ به exceptionها، توانایی handle کردن exception typeهایی است که خودتان می‌سازید. شما می‌توانید exceptionهای خودتان را برای handle کردن خطاهای به‌وجود آمده بسازید (custom exception). ساختن یک exception بسیار ساده است، تنها کافی است یک کلاس تعریف کنید و آن کلاس از Exception ارث‌بری کرده باشد. کلاس مشتق شده‌ی شما در واقع نیازی برای اجرای هیچ چیزی ندارد. هنگامی‌که کلاس‌های شما از Exception ارث‌بری می‌کنند، موجب می‌شود تا بتوانید از آن‌ها به‌عنوان exception استفاده کنید. exceptionهایی که شما می‌سازید شامل propertyو methodهایی است که ApplicationException برای آن فراهم می‌سازد. همچنین می‌توانید تعدادی از اعضای آن را در exception classای که خود ساخته‌اید، override کنید.

فرادرس
FaraDars.Org

مثال از Custom exception

مجدداً به کلاس چند ضلعی در مبحث indexer فکر کنید. در صورتیکه از یک اندیس خارج از محدود استفاده کنیم یک خطای زمان اجرا رخ می‌دهد و برنامه خانمه پیدا می‌کند می‌توان با تعریف یک استثناء و مدیریت آن در برنامه از خانمه یافتن برنامه جلوگیری کرد.

```
private void button3_Click(object sender, EventArgs e)
{
    Polygon2 rec1 = new Polygon2(4);
    rec1[0] = new Point(10, 10);
    rec1[1] = new Point(80, 10);
    rec1[2] = new Point(80, 130);
    rec1[5] = new Point(10, 130);
    rec1.draw(pictureBox1.CreateGraphics(), Color.Blue);
}
```

```
class Polygon2
{
    Point[] vertices;
     2 references
     public Polygon2(int n)
    {
       vertices= new Point[n];
    }
     7 references
    public Point this[int i]{
        set
        {
            vertices[i] = value;
        }
        get
        {
            return vertices[i];
        }
 }
```

IndexOutOfRangeException was unhandled

An unhandled exception of type 'System.IndexOutOfRangeException' occurred in indexer.exe

Additional information: Index was outside the bounds of the array.

Troubleshooting tips:

Make sure data column names are correct.

Make sure the index is not a negative number.

Make sure that the maximum index on a list is less than the list size.

Get general help for this exception.

Search for more Help Online...

Exception settings:

Break when this exception type is thrown

Actions:

View Detail...

فرادرس
FaraDars.org

مثال از Custom exception

می‌توانیم برای برنامه خود یک استثناء تعریف کنیم تنها کافیست کلاس Exception و یا کلاس ApplicationException به ارث ببرد. می‌توانیم برای کلاس استثناء ایجاد شده اجزا داده‌ای و تابعی جدید تیز در نظر بگیریم و همچنین اجرا تابعی و خصوصیت‌های کلاس پایه را override کنیم.

```
class MyException:ApplicationException
{
    private int vertexcount;
    1 reference
    public  MyException(string msg,int count) : base(msg)
    {
        vertexcount = count;
    }
    2 references
    public override string Message
    {
        get
        {
            return base.Message+"\n باید یک عدد از ۰ تا عدد "+vertexcount.ToString()+"باشد";
        }
    }
}
```

فرادرس
FaraDars.org

مثال از Custom exception

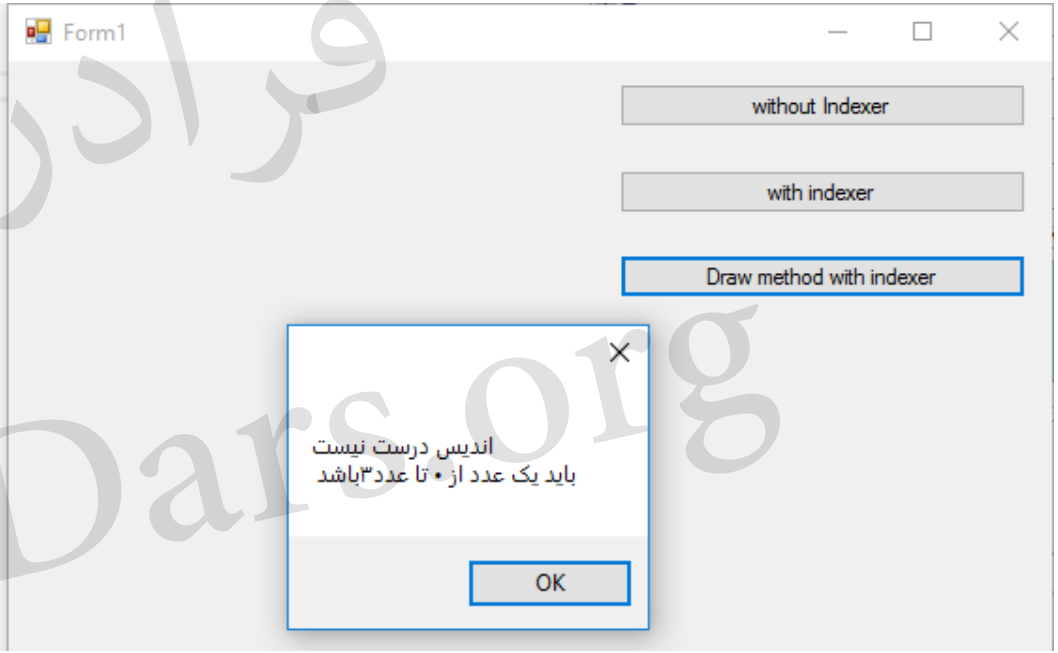
در صورتیکه در indexer اندیس ورودی در محدوده مجاز نباشد یک استثناء از نوع MyException ایجاد کرده و به بیرون پرتاب می‌کند.
بلاک مدیریت کننده این خطا در هر جایی می‌تواند ایجاد گردد.

```
class Polygon2
{
    Point[] vertices;
     2 references
     public Polygon2(int n)...
    7 references
    public Point this[int i]
    {
        set
        {
            if (i >= vertices.Length || i < 0)
                throw new MyException("اندیس درست نیست", vertices.Length - 1);
            vertices[i] = value;
        }
        get...
    }
    2 references
    public void draw(Graphics x,Color c)...
}
```

فرادرس
FaraDars.org

مثال از Custom exception

```
private void button3_Click(object sender, EventArgs e)
{
    try {
        Polygon2 rec1 = new Polygon2(4);
        rec1[0] = new Point(10, 10);
        rec1[1] = new Point(80, 10);
        rec1[2] = new Point(80, 130);
        rec1[5] = new Point(10, 130);
        rec1.draw(pictureBox1.CreateGraphics(), Color.Blue);
    }
    catch(MyException e1)
    {
        MessageBox.Show(e1.Message);
    }
}
```



فرادرس
FaraDars.org

Catching base class

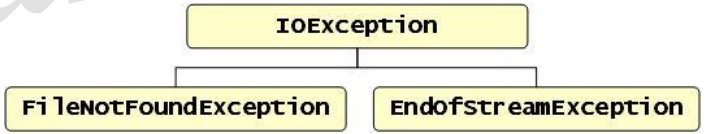
catch IOException
and all derived classes

```
void Process()
{
  try
  {
    ...
  }
  catch (IOException e)
  {
    ...
  }
}
```

catch any
exception

```
void Process()
{
  try
  {
    ...
  }
  catch (Exception e)
  {
    ...
  }
  ...
}
```

هنگام گرفتن exception typeهایی که شامل baseو derived class هستند، باید به چیدمان و نحوه‌ی قرار گرفتن دنباله‌ی catchها دقت کنید زیرا یک catch برای یک base class با تمام کلاس‌های مشتق شده از آن، تطابق دارد. برای مثال، به‌دلیل این‌که کلاس Exception، کلاس والد تمام exceptionهای دیگر است، گرفتن آن موجب گرفتن تمام exceptionهای موجود می‌شود. البته(همان‌طور که قبلاً توضیح داده شد) استفاده از catch بدون مشخص کردن exception type، یک راه دیگر(و خواناتر) برای گرفتن تمامی exceptionها است. با این حال، باید دقت کنید که گرفتن derived class exceptions(مخصوصاً) هنگامی‌که exceptionهای خودتان را می‌سازید، از اهمیت بالایی برخوردار است. یکی از مزایای استفاده از catch کردن base class این است که می‌توانید یک دسته‌بندی کلی از exceptionها را catch کنید. برای مثال، اگر خطای به‌وجود آمده به هیچ‌کدام یک از catchها مطابقت نداشت، catch کردن base class موجب می‌شود در نهایت خطا گرفته شود.



FaraDars.org

مثال ترتیب قرارگرفتن catch ها

اگر می‌خواهید هم exceptionهای base class و هم exceptionهای derived class را بگیرید، باید در دنباله‌ی نوشتن catchها، نوع derived class را در ابتدا قرار دهید. این کار ضروری است زیرا یک base class catch تمام derived classها را catch می‌کند. خوشبختانه قانون ذکر شده در سی‌شارپ ضروری است و در صورت عدم رعایت آن با خطای compile-time مواجه می‌شوید. برنامه‌ی زیر دو exception کلاس با نام‌های ExceptAو ExceptB می‌سازد. ExceptA از کلاس Exception و ExceptB از ExceptA ارث‌بری کرده است. سپس برنامه exception هر یک از typeها را throw می‌کند. برای این‌که برنامه مختصر باشد، custom exceptionها تنها یک constructor را فراهم می‌آورند، که یک رشته را می‌گیرد و خطا را شرح می‌دهد.

```
class ExceptA : Exception
{
    2 references
    public ExceptA(string message) : base(message) { }

}
3 references
class ExceptB : ExceptA
{
    1 reference
    public ExceptB(string message) : base(message) { }

}
```

فرادرس
FaraDars.org

مثال ترتیب قرارگرفتن catch ها

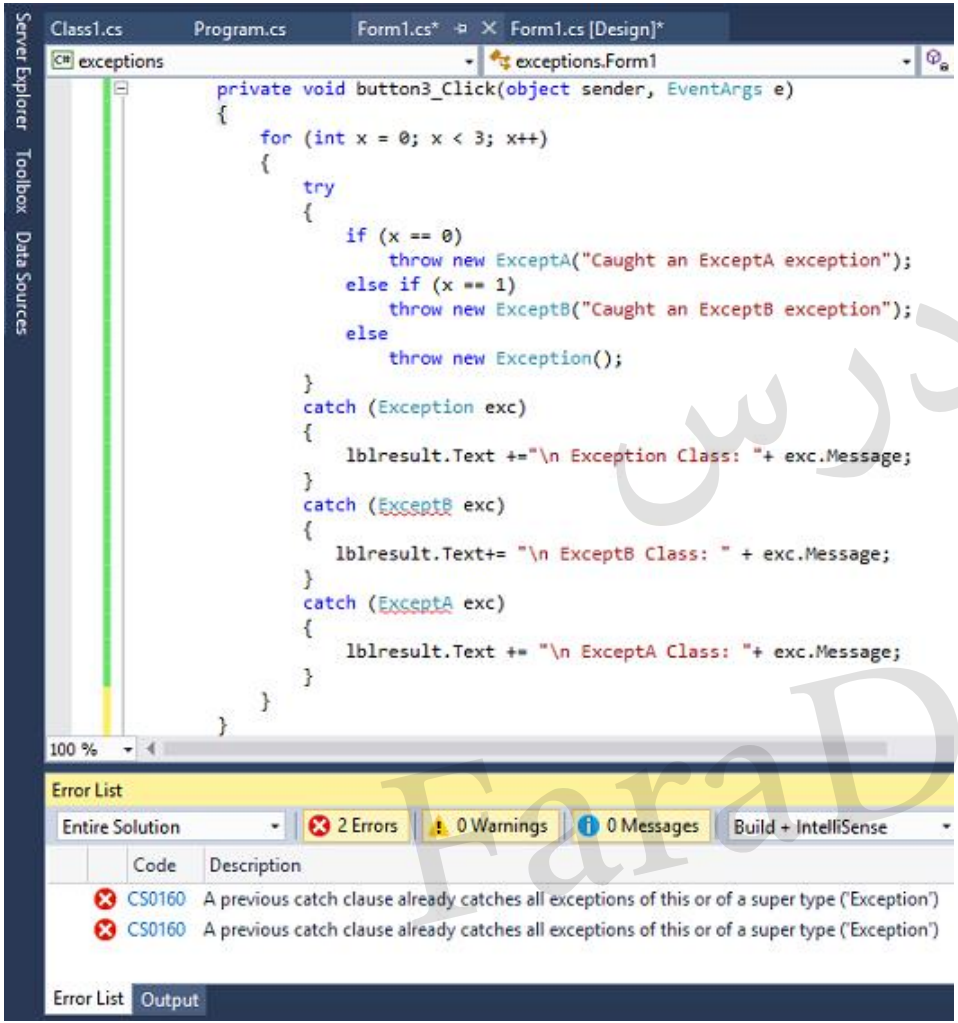
```
private void button3_Click(object sender, EventArgs e)
{
    for (int x = 0; x < 3; x++)
    {
        try
        {
            if (x == 0)
                throw new ExceptA("Caught an ExceptA exception");
            else if (x == 1)
                throw new ExceptB("Caught an ExceptB exception");
            else
                throw new Exception();
        }
        catch (ExceptB exc)
        {
           lblresult.Text+= "\n ExceptB Class: " + exc.Message;
        }
        catch (ExceptA exc)
        {
            lblresult.Text += "\n ExceptA Class: "+ exc.Message;
        }
        catch (Exception exc)
        {
            lblresult.Text +="\n Exception Class: "+ exc.Message;
        }
    }

}
```

در برنامه‌ی مقابل به نوع و ترتیب قرار گرفتن catchها دقت کنید. این تنها ترتیبی است که آن‌ها می‌توانند داشته باشند. از آن‌جا که ExceptB از ExceptA مشتق شده است، catch مربوط به ExceptB باید قبل از ExceptA واقع شود. به همین ترتیب، catch مربوط به کلاس Exception(که base class تمامی exceptionها است) باید در آخر قرار گیرد. می‌توانید با جابه‌جا کردن ترتیب catchها، ببینید که برنامه هنگام اجرا با خطای compile-time مواجه می‌شود.

یکی از مزایای استفاده از catch کردن base class این است که می‌توانید یک دسته‌بندی کلی از exceptionها را catch کنید. برای مثال اگر خطای به‌وجود آمده به هیچ‌کدام یک از catchها مطابقت نداشت، catch کردن base class موجب می‌شود در نهایت خطا گرفته شود.

فرادرس
FaraDars.org

عدم ترتیب صحیح catch ها



اگر می‌خواهید هم exceptionهای base class و هم exceptionهای derived class را بگیرید، باید در دنباله‌ی نوشتن catchها، نوع derived class را در ابتدا قرار دهید. این کار ضروری است زیرا یک base class catch تمام derived classها را catch می‌کند. خوشبختانه قانون ذکر شده در سی‌شارپ ضروری است و در صورت عدم رعایت آن با خطای compile-time مواجه می‌شوید.

در مثال قبلی با جابه‌جا کردن ترتیب catchها می‌ببینید که برنامه هنگام اجرا با خطای compile-time مواجه می‌شود.

فرادرس
FaraDars.org

Finally

```
void Process()
{
  FileStream data;

  data = new FileStream("File.dat", FileMode.Create);

  data.WriteByte(0x10);

  ...

  data.Close();
}
```

may throw exception

skipped when exception thrown

```
void Process()
{
  FileStream data = null;

  try
  {
    data = new FileStream("File.dat", FileMode.Create);
    ...
  }
  finally
  {
    data.Close();
  }
}
```

always executed

گاهی‌اوقات شما می‌خواهید یک بلوک از کد حتماً پس از try/catch اجرا شود. برای مثال ممکن است یک exception باعث شود تا ادامه‌ی اجرای یک متد پایان یابد اما آن متد یک فایل یا یک network connection را باز کرده است که حتماً باید در نهایت بسته شود. این چنین شرایطی در برنامه‌نویسی زیاد هستند و سی‌شارپ راه حل ساده و مناسبی برای آن ارائه داده که این راه حل استفاده از finally block است. Finally block باید در انتهای دنباله‌ی catchها قرار بگیرد. Finally block تحت هر شرایطی اجرا می‌شود. این بدان معناست که مهم نیست try block با موفقیت اجرا شود یا خیر، در نهایت finally block اجرا خواهد شد.

فرادرس
FaraDars.org

این اسلاید ها بر مبنای نکات مطرح شده در فرادرس
«آموزش جامع شی گرایی در سی شارپ»
تهیه شده است.

برای کسب اطلاعات بیشتر در مورد این آموزش به لینک زیر مراجعه نمایید.

faradars.org/fvcs9404

فرادرس
FaraDars.org