

مقدمه

با عرض سلام خدمت مشتاقان برنامه نویسی در مقدمه این کتاب باید بگویم که این کتاب توسط بنده سید مهدی همیلی Homeily@Gmail.com جمع اوری شده است تمام مصالب این کتاب از منابع متفاوتی و به صورت پراکنده بوده که توسط بنده جمع اوری شده است که تمام منابع ان در زیر ذکر شده است .

1 [Http://www.iranasp.com](http://www.iranasp.com)

2 [Http://www.irandevlopers.com](http://www.irandevlopers.com)

سید مهدی همیلی

Homeily@Gmail.com

آشنائی با زبان C#

مایکروسافت در مصاف با جاوا، بدنیاال ارائه يك زبان كامل بود كه سایه جاوا را در میادین برنامه نویسی كم رنگ تر نماید. شاید بهمین دلیل باشد كه #C را ایجاد كرد. شباهت های بین دو زبان بسیار چشمگیر است. مایکروسافت در رابطه با میزان استفاده و گسترش زبان فوق بسیار خوشبین بوده و امیدوار است بسرعت زبان فوق گسترده و مقبولیتی به مراتب بیشتر از جاوا را نزد پیاده کنندگان نرم افزار پیدا كند.

با توجه به نقش محوری این زبان، از آن بعنوان مادر زبانهای برنامه نویسی در ذات نت نام برده می شود. مورد فوق به تنهائی، می تواند دلیل قانع کننده ای برای یادگیری این زبان باشد، ولی دلایل متعدد دیگری نیز وجود دارد كه در ادامه به برخی از آنها اشاره می گردد.

مطرح شدن بعنوان يك استاندارد صنعتی

انجمن تولیدکنندگان کامپیوتر اروپا (ECMA) زبان #C را در سوم اکتبر سال 2001 بعنوان يك استاندارد پذیرفته (ECMA-334) و بدنیاال آن تلاش های وسیعی برای كسب گواهی ISO نیز انجام شده است. زبان فوق در ابتدا توسط شركت مایکروسافت و بعنوان بخشی از ذات نت پیاده سازی و بلافاصله پس از آن توسط شركت های اینتل، هیولیت پاكارد و مایکروسافت مشتركا، جهت استانداردسازی پیشنهاد گردید.

زبان #C بگونه ای طراحی شده است كه نه تنها وابستگی به يك Platform خاص را ندارد، بلكه در اغلب موارد وابستگی Runtime نیز ندارد. کامپایلر #C می تواند بر روی هر نوع معماری سخت افزاری طراحی و اجرا گردد. در برخی از نسخه های اولیه کامپایلر زبان فوق كه توسط برخی از شركت های جانبی ارائه شده است، كدهای # را به بایت كدهای جاوا كمپایل می كند. یکی از چنین كامپایلرهای را می توان در سایت Halcyonsoft.com مشاهده نمود. بنابراین كدهای #C براحتی قابلیت حمل بر روی محیط های متفاوت را دارا خواهند بود.

مشخصات تعریف شده زبان #C با سایر استانداردهای تعریف شده ECMA نظیر CLI (Common Language Infrastructure) (ECMA-335) (Infrastructure) بخوبی مطابقت می نمایند CLI قلب و روح ذات نت و (Common Language Runtime) CLR است. اولین نسخه از كامپایلر زبان #C كه از استفاده می كند، NET Framework. مایکروسافت است.

با توجه به موارد گفته شده، مشخص می گردد كه این زبان بسرعت بسمت استاندارد شدن حرکت و با تائید استانداردهای مربوطه از طرف انجمن های معتبر بین المللی و حمایت فراگیر شركت های معتبر کامپیوتری در دنیا مسیر خود را بسمت جهانی شدن بخوبی طی می نماید.

C چیست ؟

طراحان زبان #C با تاكید و الگوبرداری مناسب از مزایای زبانهای نظیر C++، C و جاوا و نادیده گرفتن برخی از امکانات تامل برانگیز و كم استفاده شده در هر يك از زبانهای فوق، يك زبان برنامه نویسی مدرن شی گراء را طراحی كرده اند. در مواردی، برخی از ویژگی های استفاده نشده و درست درك نشده در هر يك از زبانهای گفته شده، حذف و یا با اعمال كنترل های لازم بر روی آنها، زمینه ایجاد يك زبان آسان و ایمن برای اغلب پیاده کنندگان نرم افزار بوجود آمده است. مثلا C++ می توانند مستقیما با استفاده از اشاره گرهای عملیات دلخواه خود را در حافظه انجام دهند. وجود توانائی فوق برای نوشتن برنامه های کامپیوتری با كارائی بالا ضرورت اساسی دارد. اما در صورتیکه عملیاتی اینچنین بدرستی كنترل و هدایت نگردند، خود می توانند باعث بروز مسائل Bug (بیشماري گردد).

طراحان زبان #C، با درك اهمیت موضوع فوق، این ویژگی را كمكان در آن گنجانده ولی بمنظور ممانعت از استفاده نادرست و ایجاد اطمینان های لازم مسئله حفاظت نیز مورد توجه قرار گرفته است. جهت استفاده از ویژگی فوق، برنامه نویسان می بایست با صراحت و به روشنی خواسته خود را از طریق استفاده Keyword های مربوطه اعلان نمایند (فراخوانی يك توانائی و استفاده از آن).

C بعنوان يك زبان شی گراء عالی است. این زبان First-Class را برای مفهوم (Data Member Property) (بهمراه سایر خصائص عمومی برنامه نویسی شی گراء حمایت می كند. در C++ و جاوا يك مند get/set اغلب برای دستیابی به ویژگی های هر Property استفاده می گردد. همچنان تعریف Property را به متدهای get/set ترجمه كرده تا بدین طریق بتواند دارای حداكثر ارتباط متقابل با سایر زبانهای برنامه نویسی باشد. بصورت فطری Events, Declared Value, Reference Type, Overloading Operator را نیز حمایت می كند.

كد مدیریت یافته

با استفاده از نسخه پیاده سازی شده #C توسط مایکروسافت، می توان همواره كد مدیریت یافته ای را تولید كرد. يك برنامه #C پس از كامپایل، بصورت برنامه ای در خواهد آمد كه شامل دستورالعمل های تلفیق شده (Common Intermediate Language) است (درست بر خلاف دستورالعمل های مختص يك ماشین خاص (CIL) (گاهها با نام

MSIL (Microsoft Intermediate Language) با به اختصار IL نیز نامیده می شود (، در مفهوم مشابه بایت کدهای جاوا بوده و شامل مجموعه ای از دستورالعمل های سطح پایین قابل فهم توسط تکنولوژی مبتنی بر CLR نظیر CLR مایکروسافت خواهد بود. این برنامه ها بدین دلیل کد مدیریت یافته، نامیده می شوند CLR مسئولیت تبدیل این دستورالعمل ها به کدهای قابل اجرا بر روی ماشین و ارائه اغلب سرویس های اساسی برای کدینگ نظیر Garbage Collection، مدیریت Heap و عمر مفید يك Object و یا Type Verification را فراهم می کند.

روش یادگیری #C

یادگیری این زبان برای افرادی که دارای سابقه آشنائی با یکی از زبانهای برنامه نویسی +C، C و یا جاوا باشند کار مشکلی نخواهد بود، حتی افرادی که دارای آشنائی اولیه با جاوا اسکریپت و یا دیگر زبانهای برنامه نویسی نظیر ویژوال بیسک می باشند، امکان پذیر و راحت خواهد بود. برخی از برنامه نویسان حرفه ای بر این باور هستند که #C نسبت به VB.NET با اقبال بیشتر و سریعتری مواجه خواهد شد، چراکه #C نسبت به ویژوال بیسک خلاصه تر است. حتی برنامه های بزرگ و پیچیده ای که توسط #C نوشته می گردند خواناتر، کوتاه و زیبا خواهند بود. برخی از ویژگی های ارائه شده در #C نظیر Unsigned Integer، Operator OverLoading، و امنیت بیشتر Type ها، در VB.NET وجود نداشته و این امر می تواند دلیلی بر فراگیرتر شدن #C نسبت به VB.NET نزد برنامه نویسان با تجربه باشد.

برای یادگیری هر يك از زبانهای حمایت شده در دات نت، می بایست از BCL (Basic Class Library) مربوط به NET Framework شروع کرد. #C خود صرفا دارای ۷۷ کلمه کلیدی یا Keyword بوده که برای اکثر برنامه نویسان غریب نخواهند بود. در مقابل BCL، دارای ۴۵۰۰ کلاس و تعداد بیشماري متد و Property است که برنامه نویسان #C، می توانند از آنها برای انجام عملیات دلخواه خود استفاده نمایند. شاید یکی از مسائل قابل توجه جهت یادگیری این زبان برای برخی از برنامه نویسان حرفه ای عدم وجود برخی از ویژگی ها و امکاناتی باشد که در گذشته و از طریق سایر زبانهای استفاده شده، بخدمت گرفته می شدند. مثلا عدم وجود امکاناتی جهت توارث چندگانه (MI) سلسله مراتبی يك شیء.

خلاصه

بدون شك فراگیری و تسلط بر زبان #C بمنزله کسب يك پتانسیل با ارزش بوده که ثمرات آن برای برنامه نویسان در حال و آینده ای نه چندان دور بیشتر هویدا خواهد شد. استاندارد بودن و وجود کتابخانه ای مملو از کلاس این اطمینان را بوجود خواهد آورد که با فراگیری زبان فوق و کسب، مهارت های لازم، به يك توانائی فرا محیطی جدید دست پیدا خواهیم کرد که امکان استفاده از آن بر روی محیط های متفاوت وجود خواهد داشت. ویژگی ها و قابلیت های بیشمار این زبان از جمله دلایل قانع کننده دیگری است که فراگیری آن را توجیه پذیر و منطقی می کند.

آموزش #C قسمت اول

طی سلسله مقالاتی می خواهیم با #C بیشتر آشنا شویم. فرض این مقالات بر این است که آشنایی مختصری با زبانهای برنامه نویسی دارید ، هر چند کار ما تقریباً از صفر شروع می شود و هدف آن سادگی هر چه بیشتر است .

#C از دو زبان ++C و Java متولد شده است ! حاوی بسیاری از جنبه های ++C می باشد اما ویژگی های شیء گرای خود را از جاوا به ارث برده است .

#C اگرچه از ++C گرفته شده است اما یک زبان "خالص" شیء گرا (Object oriented) می باشد. هر دو زبان یاد شده جزو زبانهای هیبرید محسوب می شوند اما طراحان #C این مورد را به اندازه ی ++C مهم تلقی نکرده اند. یک زبان هیبرید اجازه ی برنامه نویسی با شیوه های مختلف را میسر می کند. دلیل اینکه ++C هیبرید است ، این است که قرار بوده تا با زبان C سازگار باشد و همین امر سبب گردیده تا بعضی از جنبه های ++C بسیار پیچیده شوند .

زبان سی شارپ فرض اش بر این است که شما می خواهید تنها برنامه نویسی شیء گرا انجام دهید و همانند ++C مخلوطی از برنامه نویسی رویه ای (Procedural) و شیء گرا را نمی خواهید به پایان برسانید. بنابراین باید طرز فکر خودتان را با دنیای شیء گرای تطبیق دهید. در ادامه خواهیم دید که در سی شارپ هر چیزی شیء است حتی یک برنامه ی سی شارپ .

برنامه ی اول :

Visual studio.net را اجرا کنید و سپس در صفحه ی ظاهر شده New Project را برگزینید. حالا از گزینه ی Visual C# projects قسمت Console applications را انتخاب نمایید. نامی دلخواه همانند ex01 را وارد نموده و سپس OK نمایید. کد زیر به صورت خودکار برای شما تولید خواهد شد :

```
using System;

namespace ex01
{
    ///
    /// Summary description for Class1.
    ///
    class Class1
    {
        ///
        /// The main entry point for the application.
        ///
        [STAThread]
        static void Main(string[] args)
        {
            //
            // TODO: Add code to start application here
            //
        }
    }
}
```

اگر يك سرې از مفاهيم آنرا متوجه نمي شويد اصلا مهم نيست! در مقالات آتي تمام اين موارد مفصل توضيح داده خواهند شد .
متد استاندارد Main در اينجا قسمتي است كه عمليات اصلي برنامه در حالت Console (شبیه به برنامه هاي تحت داس اما 32 بيتي) در آن انجام مي شود .
بدون متد Main برنامه هاي سي شارپ قادر به اجرا نخواهند بود . نوع آن در اينجا void تعريف شده است يعني اين متد خروجي ندارد . حتي اگر برنامه هاي استاندارد ويندوز را هم بخواهيد Cl # بنويسيد بازهم متد Main حضور خواهد داشت ، هر چند به صورت خودكار ويژوال استوديو آنرا توليد مي كند .
طريقه ي نوشتن توضيحات (Comments) در سي شارپ همانند C ++ مي باشد يعني :

```
/* any comments */
```

ويا

```
// any comments
```

و تنها برنامه نويس براي نوشتن توضيحاتي در مورد كدهاي خود از آنها استفاده مي كند و در خروجي برنامه ظاهر نمي شوند .

فعلا براي پايان قسمت اول از شيء Console و متد WriteLine آن براي نمايش يك جمله ي ساده استفاده مي كنيم . راجع به متدها ، متغيرها و غيره در آينده بيشتر صحبت مي كنيم .
در آخر برنامه ي ما چيزي شبیه به عبارت زیر مي باشد :

```
using System;  
  
namespace ex01  
{  
class Class1  
{  
[STAThread]  
static void Main(string[] args)  
{  
Console.WriteLine("Hello C#!");  
}  
}  
}
```

دکمه ي F5 را فشار دهيد تا برنامه اجرا شود .

تعريف متغيرها در سي شارپ :

سي شارپ عناصر ي را كه بكار مي گيرد همانند اعداد و کاراکترها ، به صورت نوع ها (Type) طبقه بندي مي كند . اين انواع شامل موارد زیر مي شوند :
نوع هاي پايه ايي از پيش تعريف شده مانند اعداد و غيره .
نوع هاي تعريف شده توسط کاربر كه شامل STRUCT ها و ENUM ها مي شوند .

نحوه ي تعريف متغيرها از نوع هاي پايه ايي از پيش تعريف شده :
همانطور كه مي دانيد از متغيرها براي نگهداري اطلاعات استفاده مي شود . در سي شارپ ابتدا نوع متغير و سپس نام متغير و در آخر يك سمي كولون بكار برده مي شود . براي مثال :

```
int a;
```

که در اینجا متغیر a بعنوان يك متغیر حاوي اعداد صحیح تعریف شده است .
نکته ي مهمي که در اینجا حائز اهمیت است ، مقدار دهی اولیه ي متغیرها مي باشد .در غیر اینصورت
کامپایلر سي شارپ برنامه را بایک خطا متوقف مي کند .دلیل این امر هم این است که از استفاده از
متغیرهای بدون مقدار در طول برنامه جلوگیری شود تا میزان خطاهای در حین اجرا کاهش یابد .

نوع های داده ای پایه ي زیر در در سي شارپ به صورت پیش فرض مهیا هستند :

object : نوعی است نامحدود که مي تواند تمام انواع دیگر را نیز شامل شود .مثال :

```
object = null;
```

string : رشته ؛ در اینجا يك رشته توالی کاراکترهای یونیکد مي باشد .مثال :

```
string s= "hello";
```

sbyte : نوع داده ای صحیح 8 بیتی علامت دار .

byte : نوع داده ای صحیح 8 بیتی بدون علامت .مثال :

```
sbyte val = 12;
```

short : نوع داده ای صحیح 16 بیتی علامت دار .

ushort : نوع داده ای صحیح 16 بیتی بدون علامت .مثال :

```
short val = 12;
```

int : نوع داده ای صحیح 32 بیتی علامت دار .

uint : نوع داده ای صحیح 32 بیتی بدون علامت .مثال :

```
int val = 12;
```

long : نوع داده ای صحیح 64 بیتی علامت دار .

ulong : نوع داده ای صحیح 64 بیتی بدون علامت .مثال :

```
Long val1 = 12; long val2 = 34L;
```

کلا در اینجا u به معنای unsigned است .

float : نوع اعشاری با single precision

double : نوع اعشاری با double precision . مثال :

```
float val = 1.23f;
```

bool : نوع داده ای Boolean که مي تواند true و یا false باشد .مثال :

```
Bool val = true;
```

char : کاراکتر، در اینجا char يك کاراکتر یونیکد است .

```
char val = 'h';
```

به نحوه ي تعریف کاراکتر ها و همچنین رشته ها در سي شارپ دقت کنید .

decimal : نوع داده ای دسیمال با 28 رقم معنی دار .

```
decimal val = 1.23M;
```

يك نکته :

-بهتر است هنگام تعريف يك متغير ، نامي با مسما براي آن انتخاب شود تا در هنگام كار خواندن كد ساده تر گردد . همچنين رسم شده است كه نوع متغير را به صورت خلاصه به نام متغير اضافه مي كنند . براي مثال بجاي FirstName بهتر است بنويسيم strFirstName . به اين نوع نگارش Hungarian notation مي گویند .
-تمام نوع هاي پيش فرض تعريف شده در سي شارپ شيء هستند . در آينده بيشتر در اين مورد صحبت خواهيم كرد .

مثال اين قسمت :

يك برنامه ي console جديد در را VS.NET باز كنيد . نام آنرا در ابتدا ex02 انتخاب نماييد . در اينجا مي خواهيم دو متغير رشته ابي و صحيح را تعريف و سپس در خروجي نمايش دهيم .

كد نهايي به صورت زير مي باشد :

```
using System;

namespace ex02
{
    ///
    /// Summary description for Class1.
    ///
    class Class1
    {
        ///
        /// The main entry point for the application.
        ///
        [STAThread]
        static void Main(string[] args)
        {
            int intVar1 = 0;

            int intVar2;
            intVar2=1;

            int intV3=15 , intV4 = 12;

            string strText1 = "abcd";

            Console.WriteLine(
                "The value for variables are : \n intVar1="+intVar1 +
                "\n intVar2="+ intVar2 +
                "\n intV3=" + intV3 +
                "\n intV4=" + intV4 +
                "\n strText1=" + strText1);

            Console.WriteLine("\n\n Press any key to terminate");
            Console.ReadLine(); // pause screen!

        }
    }
}
```

```
}  
}
```

نکاتی در مورد کد فوق :

- يك اسلس ان ، در زبانهاي مشتق شده از سي به معنای new line مي باشد .
- در کد فوق نحوه ي تعريف چند متغير در يك خط و حالتهاي مقدار دهی مختلف را ملاحظه مي کنید .
- از متد ReadLine براي نگه داشتن خروجي و مشاهده ي آن در اینجا استفاده کردیم .
- عادت کنید به صورت دندانہ دار کد بنویسید .اینکار خوانایی کد را صد برابر می کند .در اینجا کدهای داخل متد main ، کاملاً چند دندانہ از آکولادهای باز و بسته کردن آن جلو تر هستند .
- در کد بالا در متد WriteLine اعداد و رشته ها با هم جمع شده اند !این مورد بدلیل وجود overload های زیاد این تابع و ...میسر گشته است .اصلاً به آن دل نبندید !چون در آینده کامپایلر سی شارپ اگر چنین اعمالی را در جاهای دیگری مرتکب شوید به شدت با شما برخورد خواهد کرد !!برای جمع کردن اعداد با رشته ها حتما باید عدد به رشته تبدیل گردد و بعددر این مورد در مقالات بعدی بحث خواهد گردید .

آموزش C# قسمت دوم

در این قسمت می خواهیم با یک سری از اصول اولیه ی شیء گرایی در سی شارپ کمی آشنا شویم . لازم به ذکر است ، بسیاری از مواردی که در این قسمت مطرح می شوند فقط برای آشنایی شما است و در آینده بیشتر بحث و مرور خواهند شد .

آشنایی با فضاهاي نام (NameSpaces)

فضاهای نام روشی برای مدیریت کد نویسی هستند . برای مثال آنها ایجاد شده اند تا تداخلی بین نام های توابع در برنامه شما رخ ندهد . این مساله در پروژه های بزرگ خود را نشان می دهد و ممکن است دو آیتم در یک پروژه نام های یکسانی را پیدا کنند . بدین وسیله این شانس تصادم و تداخل کاهش پیدا می کند . برای ایجاد یک فضای نام به صورت زیر عمل می شود :

```
namespace anyName
{
.....

Class anyClassName
{
.....
}

.....

}
```

یکی از فضاهای نام پایه ای در دات نت فریم ورک ، فضای نام System می باشد . برای استفاده از آن می توان از کد زیر کمک گرفت :

```
using System;
```

تمام فضاهای نام به صورت پیش فرض public می باشند و در خارج از کد شما قابل دسترسی هستند . روش استفاده از آنها به صورت زیر است :

```
ProjectName.Namespace.ClassName.MemberName
```

نکته :

اگر دقت کرده باشید هنگامی که کرسر ماوس را روی هر آیتمی در منوی autocomplete نگه می دارید و یا آنرا انتخاب می کنید یک راهنمای کوچک نمایش داده می شود که در حقیقت کامنت مربوط به آن تابع می باشد . روش نوشتن چنین کامنت حرفه ای که در منوهای ویژوال استودیو ظاهر شود به صورت زیر است که بهتر است!) (قبل از هر تابع یا خاصیت یا کلاس و نوشته شود

```
///
///
///
///
```

کلاس ها :

چون سي شارپ تمام سر و کارش با کلاس ها است بنابراین باید در مورد نحوه ي تعريف و استفاده از آنها تسلط کافي داشته باشیم .

يك پروژه ي جديد console در VS.NET باز کنید و نام آنرا در ابتدا ex03 وارد نمایید .
بعد از باز شدن پروژه ، از منوي Project گزینه ي Add class را انتخاب کنید تا کلاسي جديد به نام clsDate.cs را اضافه نمایيم .ساختار فایل ایجاد شده توسط VS.NET به صورت زیر است :

```
using System;

namespace ex03
{
    ///
    /// Summary description for clsDate.
    ///
    public class clsDate
    {
    public clsDate()
    {
    //
    // TODO: Add constructor logic here (chashm!)
    //
    }
    }
}
```

تابع يا متد clsDate که در اینجا به صورت پیش فرض ایجاد شده است اصطلاحاً سازنده (constructor) نام دارد .این تابع هر بار که يك شيء جديد از کلاس مي سازيم به صورت خودکار اجرا مي شود .

از این کلاس مي خواهيم براي نمایش تاريخ /ساعت و غيره استفاده کنیم .

براي مثال مي خواهيم تاريخ جاري سيستم را به صورت يك خاصیت از این کلاس دریافت کنیم .براي این منظور کد زیر را به برنامه اضافه مي نمایيم :

```
public string currentSystemDate
{
get
{
return System.DateTime.Today.ToString() ;
}
}
```

توضیح کد فوق :

خاصیتي را که مي خواهيم از برنامه دریافت کنیم با کلمه ي کلیدی get معرفي مي نمایيم .هر چیزی که این قسمت برگرداند خروجي currentSystemDate خواهد بود .این دستور زبان که در بالا معرفي شد استاندارد است و در همه جا به يك صورت تعريف و بکار برده مي شود .پس شکل آنرا به خاطر بسپاريد .
از کلمه ي کلیدی return براي برگرداندن يك خروجي از خاصیت و يا تابع استفاده مي شود .

براي استفاده از این خاصیت جديد ، در فایل Class1.cs که متد main برنامه ي ما در آنجا قرار دارد به صورت زیر عمل مي کنیم :

```
clsDate m_var = new clsDate(); // initialize variable
Console.WriteLine ( m_var.currentSystemDate );
Console.ReadLine();//pause!
```

توضیح کد فوق :

برای استفاده از یک کلاس باید یک متغیر از آن را تعریف کنیم. در هر زبانی یک سری نوع های استاندارد مانند int و string و غیره وجود دارند. کلاس هم در حقیقت یک نوع داده ی بسیار بسیار قدرتمند به شمار می آید. برای تعریف یک متغیر از نوع جدید روش کار مانند سابق است. برای مثال زمانی که یک متغیر عدد صحیح را تعریف می کنید به صورت زیر عمل می شود :

```
int i=0;
```

رای تعریف یک متغیر از نوع داده ای که خودمان تعریف کرده ایم نیز باید به همین صورت عمل شود .

```
clsDate m_var = new clsDate();
```

از کلمه ی کلیدی new اینجا به صورت استاندارد برای مقدار دهی اولیه به این متغیر جدید استفاده می نمایم .

سپس به روش دستیابی به این خاصیتی که به کلاس اضافه کرده ایم می رسم .

```
m_var.currentSystemDate
```

کلا چه یک خاصیت و یا یک متد را به کلاس اضافه نمایم برای دستیابی به آن از عملگر نقطه پس از ذکر نام متغیر تعریف شده از نوع کلاس خود ، استفاده می نمایم. برای استفاده از خاصیت ها نیازی به آوردن (بعد از ذکر نام خاصیت نمی باشد .

عموما از خاصیت ها برای برگرداندن و یا تنظیم یک مقدار ساده استفاده می شود و در آنها عملیات پیچیده ای مد نظر نمی باشد .

توضیحی در مورد (System.DateTime.Today.ToString());
استفاده از خواص :

شما به ویژگی های یک شیء با استفاده از خواص آن می توانید دسترسی پیدا کنید. یک property عضوی است که امکان دسترسی به ویژگی شیء یا کلاس را فراهم می کند. برای مثال طول یک رشته string) ، سایز یک فونت ، عنوان یک فرم و نام یک مصرف کننده ، خاصیت هستند . بسیاری از اشیاء ذاتی ذات نت فریم ورك ، خواص مفید زیادی را به همراه دارند. برای مثال شیء DateTime را در نظر بگیرید. با استفاده از خاصیت Today آن می توان تاریخ جاری سیستم را بدست آورد . برای استفاده از یک خاصیت لازم است تا کلاس تعریف کننده شیء در برنامه مهیا باشد. منظور همان استفاده از فضای نام مربوطه می باشد. پس از وارد کردن فضای نام کلاس مورد نظر می توانید از شیء و خواص آن استفاده کنید. دو راه وجود دارد یا به صورت کامل تمام موارد باید ذکر شوند مانند System.DateTime.Now ; و یا با وارد کردن فضای نام System کوتاه سازی صورت می گیرد . برای استفاده از هر متد و یا شیء ایی در سی شارپ باید این شیء قابل دسترسی باشد. برای مثال شیء Console که از آن برای چاپ کردن خروجی بر روی صفحه ی نمایش استفاده می کنیم در فضای نام System واقع شده است. یا باید در ابتدای برنامه ذکر کرد using System ; و سپس خیلی راحت از این شیء استفاده کرد و یا می توان اینکار را انجام نداد و نوشت System.Console و الی آخر. با ذکر فضای نام در ابتدا با استفاده از using می توان خلاصه نویسی کرد .

نتیجه ی نهایی مثال این فصل :

محتویات فایل Class1.cs

```

using System;

namespace ex03
{
    ///
    /// Summary description for Class1.
    ///
    class Class1
    {
        ///
        /// The main entry point for the application.
        ///
        [STAThread]
        static void Main(string[] args)
        {
            clsDate m_var = new clsDate(); // initialize variable
            Console.WriteLine ( m_var.currentSystemDate );

            Console.ReadLine();//pause!
        }
    }
}

```

محتویات فایل clsDate.cs که به برنامه اضافه کردیم :

```

using System;

namespace ex03
{
    ///
    /// Summary description for clsDate.
    ///
    public class clsDate
    {
        public clsDate()
        {
            //
            // TODO: Add constructor logic here
            //
        }

        public string currentSystemDate
        {
            get
            {
                return System.DateTime.Today.ToString() ;
            }
        }

    }
}

```

آموزش C# قسمت سوم

ساختارهای تصمیم گیری :

در بسیاری از موارد هنگام برنامه نویسی لازم است تا از عبارات شرطی استفاده کنیم. برای انجام اینکار دو روش عمده وجود دارد. استفاده از if و یا switch. از if بیشتر برای مقایسه هایی تکي و کوچک استفاده مي شود و حاصل مقایسه ي آن يا true است و يا false. از عبارت switch هنگامی استفاده مي شود که مقایسه های متعددی باید در مورد يك مقدار صورت گیرد. هر دو عبارت if و switch توسط عبارتهایی Boolean کنترل مي شوند (true و يا false). در هنگام استفاده از if اگر عبارت Boolean حاصل اش true باشد اولین قسمت شرط اجرا مي شود و سپس برنامه از انتهای if ادامه پیدا مي کند. اگر حاصل عبارت Boolean مساوي false باشد کنترل برنامه به قسمت else منتقل مي شود.

مثال :

يك پروژه ي جدید console باز کنید و نام آنرا ex04 بگذارید. سپس کد زیر را در آن وارد و اجرا کنید :

```
using System;

namespace ex04
{
class Class1
{
[STAThread]
static void Main(string[] args)
{
Console.WriteLine("Enter 1 character to be evaluated");

char cUserInput = (char) Console.Read();

if ( char.IsDigit( cUserInput ) )
Console.WriteLine("The char is a number!");
else
Console.WriteLine("The char is not a number!");

}
}
}
```

نکاتی در مورد کد فوق :

- 1- سي شارپ به کوچکی و بزرگی حروف حساس است. برای مثال cUserInput با cUserinput فرق مي کند.
- 2- حتما باید بعد از if پرانتزها ذکر گردد.
- 3- حتما باید داخل if يك عبارت Boolean ذکر شود مانند if(x).
4- در سي شارپ مقایسه ي تساوي دو عبارت با == و انتساب با = انجام مي شود. (موارد 1 و 4 مواردی هستند که اغلب تازه کاران با آن مشکل دارند!) برای مثال if(i==3) (صحیح است اما if(i=3) در سي شارپ معنایی ندارد.
- 5- اگر بعد از if يك خط کد قرار گیرد نیازی به آوردن آکولاد ها نیست. هنگامی نیاز به آکولادها مي باشد که بیش از يك خط باید بعد از if قرار گیرد.
- 6- در سي شارپ همانند اسلاف خودش برای تبدیل نوع های داده ای می توان به صورت زیر نیز عمل کرد : Console.Read (char)) ; یعنی دریافتی Read به char تبدیل مي شود. در این مورد باز هم صحبت

خواهد شد .
7- همانطور که ذکر شد در سی شارپ همه چیز شیء است حتی نوع های پایه ایی مانند char . با استفاده از متد IsDigit آن می توان چک کرد که آیا ورودی آن عدد است یا خیر؟ (در مورد متدها صحبت خواهد شد)

استفاده از switch

بهتر است این مورد را با یک مثال دنبال کنیم .
پروژه ی سی شارپ جدیدی به نام ex05 در حالت console در VS.NET باز کنید . در اینجا می خواهیم یک کلاس جدید تعریف کرده و توسط خاصیتی که در آن ایجاد می کنیم متوجه شویم روز جاری مطابق سیستم چه روزی است .
یک کلاس جدید از منوی پروژه ، با استفاده از گزینه ی class Add به برنامه اضافه کنید و نام آنرا در ابتدا clsDate بگذارید .

```
using System;

namespace ex05
{
    ///
    /// Summary description for clsDate.
    ///
    public class clsDate
    {
        public clsDate()
        {
            //
            // TODO: Add constructor logic here
            //
        }

        public string systemDayOfWeek
        {
            get
            {

                string res="";
                switch( System.DateTime.Now.DayOfWeek.ToString())
                {
                    case "Saturday" :
                        res = "شنبه";
                        break;

                    case "Sunday" :
                        res = "یک شنبه";
                        break;

                    case "Monday":
                        res = "دوشنبه";
                        break;

                    case "Tuesday":
                        res = "سه شنبه";
                        break;
                }
            }
        }
    }
}
```

```

case "Wednesday":
res = "چهارشنبه";
break;

case "Thursday":
res = "پنجشنبه";
break;

case "Friday":
res = "جمعه" ;
break;
}

return res ;
}

}

}
}
}
}

```

هنگام ذخیره کردن این کد ویژوال استودیو به شما اخطار می دهد که کد دارای حروف یونیکد است . از منوی فایل گزینه ی advanced save options را انتخاب کنید . در اینجا می توان نوع ذخیره سازی را یونیکد انتخاب کرد .

برای استفاده از کلاس فوق مانند مطالبی که در قسمت قبل گفته شد عمل می کنیم :

```

using System;

namespace ex05
{
class Class1
{
[STAThread]
static void Main(string[] args)
{
clsDate m_var = new clsDate();
Console.WriteLine( m_var.systemDayOfWeek );
Console.ReadLine();
}
}
}
}

```

هر چند حالت console یونیکد را پشتیبانی نمی کند ولی اصل برنامه برای ما مهم است و در آینده بیشتر از آن استفاده خواهیم کرد .

همانطور که ملاحظه کردید اگر از switch استفاده نمی شد باید از 7 عدد if استفاده می کردید که اصلاً ظاهر حرفه ای و شکلی نداشت !
با استفاده از عبارت زیر کار مقایسه شروع می شود . روز سیستم در یافت شده و وارد بدنه ی switch می گردد . سپس توسط case ها چک می شود تا تساوی آن با عبارت بعد از case به اثبات برسد .

switch(System.DateTime.Now.DayOfWeek.ToString())

اگر هر کدام از عبارات بعد از case صحیح بودند کار پس از آن که در اینجا انتساب است انجام شده و سپس توسط break کنترل برنامه از switch خارج می شود و ادامه ی کار دنبال می گردد .
اگر هیچکدام از case ها صحیح نبودند می توان از گزینه ی default هم در صورت نیاز استفاده کرد . این حالت در يك چنین موافعی اجرا می گردد.

آموزش C# قسمت چهارم

آرایه ها در سی شارپ :

هنگامی آرایه ها ایجاد می شوند که بخواهیم با مجموعه ای از اطلاعات همجنس کار کنیم. برای نمونه از یک آرایه برای ذخیره تعدادی کاراکتر می خواهیم استفاده نماییم. آرایه ها هم یک نوع متغیر هستند پس باید تعریف و مقدار دهی اولیه شوند ، نوع و تعداد اعضای آنها نیز باید معین گردد . فرض کنید 10 داده ی هم جنس داریم (برای مثال رشته string) (و می خواهیم آنها را ذخیره کنیم. یا می توان 10 متغیر مختلف را تعریف کرد و سپس تک تک آنها را مقدار دهی نمود و یا یک آرایه تعریف نمود و سپس در خانه های مختلف آن این ده عضو را چید. این مطلب زمانی حائز اهمیت می شود که داده های همجنس و به نوعی مرتبط ما تعداد زیادی داشته باشند .

برای تعریف آرایه چندین راه مختلف وجود دارد :
برای تعریف آرایه ابتدا نوع آنرا مشخص می کنید سپس [] را باید جلوی تعریف نوع بگذارید این دستور زبان است و چون چرا ندارد! در زبان سی کمی متفاوت بود. این گروه ها بعد از نام متغیر می آمدند. و سپس در اینجا نام یک متغیر را که بعدا به آن ارجا می دهیم خواهید گذاشت. برای مثال

```
int[] table; // not int table[];
```

حد پایین آرایه صفر بوده برای مثال اگر آرایه chrData[] ده عضو داشته باشد، اولین عضو آن chrData[0] و آخرین عضو آن chrData[9] است .
مطلب دیگری که در مورد آرایه ها خیلی مهم است اندازه ی آن است. یعنی یک آرایه حاوی چند خانه ی خالی است که ما اجازه داریم آنرا پر کنیم. مثال :

```
int[] numbers; // declare numbers as an int array of any size  
numbers = new int[10]; // numbers is a 10-element array  
numbers = new int[20]; // now it's a 20-element array
```

1- تعریف آرایه ای از رشته ها و مقدار دهی اولیه آن .

```
String[] strData = new string[2];
```

2- تعریف و مقدار دهی اولیه

```
string [] strData = { "1234","abcd" };
```

که آرایه ای از نوع رشته ای به طول 2 عضو با مقدار دهی اولیه ایجاد شده است. در این حالت نیازی به تعیین طول آن نمی باشد .

3- روشی دیگر برای مقدار دهی اولیه

```
strData[0] = "1234";  
strData[1] = "abcd";
```

مثال : يك پروژه ي جديد Console سي شارپ را باز كنيد و نام آنرا در ابتدا ex06 بگذاريد . در اين مثال مي خواهيم نحوه ي كار با آرايه ها را مرور كنيم :

```
using System;

namespace ex06
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            string[] sGoalList = new string[3];
            string sReplyStatement = "You have choosen Goal ";

            // Store goals in the array
            sGoalList[0] = "Hike the Appalachian Trail";
            sGoalList[1] = "Run the marathon";
            sGoalList[2] = "Give $1 million to worthwhile causes";

            // Store response to goals in the array
            //(declaring and initializing on same line)
            string[] sGoalResponse = {
                "If you are staring from GA, you should get "
                + "started in early spring, so you will "+
                "not get caught in snow.",
                "Make sure that you have a good pair of shoes.",
                "Start saving as soon as possible."};

            // Give the user a list of goals to choose from
            Console.WriteLine("GOAL LIST");

            for(int i = 0; i < sGoalList.Length; i++)
            {
                Console.WriteLine("Goal " + i +
                    " - " + sGoalList[i]);
            }

            // Request the user to choose a goal.
            Console.WriteLine(""); // Write an empty line for space
            Console.Write("Please choose the number of the "
                + "goal that you want to achieve [0,1,2]: ");

            Console.ReadLine();

        }
    }
}
```

نكاتي در مورد كد فوق :

- 1- نحوه ي استفاده از عملگر + را براي اتصال رشته هاي بلند در كد فوق مي توان ديد .
- 2- در سي شارپ پايان خط سمعي كولون مي باشد . بنابراين نگراني در مورد چند خطي شدن يك دستور وجود ندارد .
- 3- هنگامي كه آرايه اي را با مقادير درون آكولادها ، مقدار دهني اوليه مي كنيد لزومي ندارد طول آن آرايه را مشخص كنيد ؛ مانند آرايه sGoalResponse در بالا . در غير اينصورت حتما بايد طول يك آرايه را كه معرف تعداد خانه هاي خالي آن است ، معرفي كنيد مانند آرايه sGoalList
- 4- فعلا حلقه ي for را در اين مثال بخاطر داشته باشيد تا در مقاله ي بعدي راجع به آن صحبت كنيم

آموزش C# قسمت پنجم

حلقه ها در سي شارپ :

مقدمه :

اگر نیاز باشد تا قطعه ای از کد بیش از یکبار اجرا شود نیاز به استفاده از حلقه ها می باشد. برای مثال فرض کنید آرایه ای به طول 1000 تعریف کرده اید. اکنون می خواهید آنرا با هزار عدد متوالی پر کنید. بدیهی است که روش زیر کارآمد نیست !

```
int[] intData = new int[1000];
intData[0]=0;
.
.
.
intData[999]=1000;
```

نوشتن این خطوط متوالی احتمالا با کپی و پیست و اصلاح آن حداقل نیم ساعت طول می کشد! بنابراین نیاز به وسیله ای حس می شود که بتوان بوسیله ی آن امثال اینگونه کارها را انجام داد .

تعریف حلقه ها و استفاده از آنها :

برای تعریف حلقه ها ابزارهای متعددی مانند `for`, `while`, `do` و `foreach` وجود دارند. استفاده و انتخاب آنها بستگی به سلیقه ی شما و منطق برنامه دارد. در هر حال یک مساله بدیهی است که همواره بیش از یک راه حل برای یک مساله وجود خواهد داشت .

استفاده از حلقه ی `for`

عموما کدنویسی را با کد نویسی می توان آموخت! بنابراین در مورد انواع حلقه ها مثالهایی ارائه خواهد گردید .

یک برنامه ی سی شارپ جدید `console` را در `VS.NET` باز کنید و نام آنرا در ابتدا `ex07` انتخاب نمایید. سپس کد زیر را درون آن بنویسید :

```
using System;

namespace ex07
{
class Class1
{
[STAThread]
static void Main(string[] args)
{
int[] intData = new int[1000];

for (int i=0 ; i<1000 ; i++ )
intData[i]=i;

for(int i=0 ; i< intData.Length ; i++)
{
int j = intData[i];
```

```

Console.WriteLine("intData[" + i + "]=" + j);
}

Console.ReadLine();

}
}
}

```

توضیحاتی در مورد کد فوق :

- 1- برای تعریف حلقه ی for همانطور که می بینید باید تعداد بار اجرای حلقه (اینجا از 0 تا 999 است) و همچنین نحوه ی رسیدن از 0 به 1000 را مشخص کرد (در اینجا ++ است یعنی هر بار یک واحد به شمارشگر حلقه اضافه می شود.)
- 2- در زبان سی ++ یعنی $i=i+1$ و -- یعنی $i=i-1$ و کلا $i=n$ یعنی $i=i-n$ و به همین ترتیب . برای مثال $i*=n$ یعنی $i=i*n$ و $i+=n$ یعنی $i=i+n$ و ...
- 3- اگر پس از حلقه ی for یک خط کد داشته باشیم نیازی به آکولاد نیست (مانند قسمت اول کد) . ولی اگر تعداد خطوط مربوط به بدنه ی for زیاد بود باید حتما از آکولاد استفاده شود (مانند قسمت دوم کد) . این قاعده ای کلی است در زبانهای مشتق شده از زبان سی در مورد هر چیزی!
- 4- فرض کنید در قسمت اول کد بالا بجای 1000 می نوشتید 1001 . سرریجا با یک خطای زمان اجرا مواجه می شدید . زیرا می خواستید به عضوی از آرایه دسترسی پیدا کنید که تعریف نشده است . راه مدرن چک کردن این مسائل استفاده از خاصیت Length آرایه است که در قسمت دوم کد در عمل مشاهده می نمایید . همیشه از این روش استفاده کنید .
- 5- حلقه ی اول یعنی اینکه کار پر کردن آرایه intData را از صفر تا 999 یکی یکی ++i) انجام بده .

استفاده از حلقه ی while

یک برنامه ی سی شارپ جدید console را در VS.NET باز کنید و نام آنرا در ابتدا ex08 انتخاب نمایید . سپس کد زیر را درون آن بنویسید :

```

using System;

namespace ex08
{
class Class1
{
[STAThread]
static void Main(string[] args)
{
int n = 1;

while (n < 6)
{
Console.WriteLine("Current value of n is {0}", n);
n++;
}

Console.ReadLine();
}
}
}

```

توضیحاتی در مورد کد فوق :

- 1- حلقه ی while در بالا کار انجام حلقه را تا هنگامی انجام می دهد که شرط ذکر شده در ابتدای آن صادق و برقرار باشد . یعنی در حلقه ی فوق تا وقتی 6 است این حلقه ادامه خواهد یافت .

2- حلقه ي while صفر يا بيشتر بار ممکن است اجرا شود .
3- در کد فوق از {0} استفاده گردیده است .متد WriteLine به شما اين اجازه را مي دهد که n تا آرگومان براي آن تعريف کنيد و مقادير هر کدام را که خواستيد در کد نمايش دهيد از \x استفاده کنيد . در اين مورد مقدار آرگومان x ام نمايش داده مي شود .

استفاده از حلقه ي do

يك برنامه ي سي شارپ جديد console را در VS.NET باز کنيد و نام آنرا در ابتدا ex09 انتخاب نماييد . سپس کد زير را درون آن بنويسيد :

```
using System;

namespace ex09
{

class Class1
{
[STAThread]
static void Main(string[] args)
{
int x;
int y = 0;

do
{
x = y++;
Console.WriteLine(x);

}while(y < 5);

Console.ReadLine();

}
}
}
```

توضيحاتي در مورد کد فوق :

- 1- اين حلقه به حلقه ي do...while معروف است و هر دو جزء آن بايد ذکر گردد .
- 2- اين حلقه تا زماني که شرط ذکر شده در قسمت while صحيح است ادامه مي يابد .
- 3- اين حلقه در ابتداي کار بدون توجه به قسمت while حداقل يکبار اجرا مي شود. (مثال زير را اجرا نماييد)

```
int n = 10;
do
{
Console.WriteLine("Current value of n is {0}", n);
n++;
} while (n < 6);
```

استفاده از حلقه ي foreach

يك برنامه ي سي شارپ جديد console را در VS.NET باز کنيد و نام آنرا در ابتدا ex10 انتخاب نماييد . سپس کد زير را درون آن بنويسيد :

```

using System;

namespace ex10
{
class Class1
{
[STAThread]
static void Main(string[] args)
{
int odd = 0, even = 0;
int[] arr = new int [] {0,1,2,5,7,8,11};

foreach (int i in arr)
{
if (i%2 == 0)
even++;
else
odd++;
}

Console.WriteLine(
"Found {0} Odd Numbers, and {1} Even Numbers.",
odd, even) ;

Console.ReadLine();

}
}
}

```

توضیحاتی در مورد کد فوق :

- 1- از foreach برای حرکت در بین اعضای یک آرایه (مانند مثال بالا) و یا مجموعه ایی از اشیاء استفاده می شود (روشهای تشکیل ، مدرن و مطمئن! و تقریباً به ارث رسیده از ویژوال بیسیک!!).
- 2- در زبانهای مشتق شده از C ، عملگر %، باقیمانده را محاسبه می کند .
- 3- در کد فوق با استفاده از حلقه ی foreach تک تک اعضای آرایه در مورد زوج و یا فرد بودند مورد بررسی قرار گرفته اند و تعداد اعضای زوج و فرد در آخر نمایش داده می شود

آموزش C# قسمت ششم

دو مورد تکمیلی در مورد حلقه ها در سی شارپ :

- 1- هر جایی خواستید به هر دلیلی حلقه را پایان دهید می توانید از دستور break استفاده کنید. در این حالت به صورت آنی حلقه خاتمه یافته و کدهای ادامه ی برنامه پس از حلقه اجرا می شوند .
- 2- نحوه ی استفاده از دستور continue : فرض کنید حلقه ی شما در راند 15 خودش است !حالا در این راند شما می خواهید یک سری از دستورات درون حلقه اجرا نشوند و حلقه به راند بعدی منتقل شده و کارش را ادامه دهد. اینجا است که از دستور continue استفاده می شود. بهتر است به یک مثال ساده در این زمینه توجه کنیم .

مثال : یک برنامه ی سی شارپ جدید console را در VS.NET باز کنید و نام آنرا در ابتدا ex11 انتخاب نمایید . سپس کد زیر را درون آن بنویسید :

```
using System;

namespace ex11
{
class Class1
{
[STAThread]
static void Main(string[] args)
{

Console.WriteLine(
"for (int i = 1; i <= 100; i++) -> break at i==5" );
for (int i = 1; i <= 100; i++)
{
if (i == 5)
break;
Console.WriteLine(i);
}
Console.ReadLine();

Console.WriteLine(
"for (int i = 1; i <= 10; i++) -> continue if i<9" );
for (int i = 1; i <= 10; i++)
{
if (i < 9)
continue;
Console.WriteLine(i);
}
Console.ReadLine();

}
}
}
```


موارد تکميلي مربوط به رد و بدل کردن مقادير به/از کلاس ها :

در قسمت بعدي مي خواهيم خاصيتي را تعريف کنيم که يك مقدار را از کاربر مي گيرد و در برنامه مي توان توسط قسمت هاي ديگر از آن استفاده کرد .

ابتدا يك متغير عمومي بايد در سطح کلاس تعريف کرد تا مقدار دريافت شده توسط set را در خود نگاه داري کند (در مورد scope متغيرها) متغيرهاي عمومي و محلي و امثال اينها) در هنگام معرفي توابع بيشتري بحث خواهد شد) . سپس از طريق کلمه ي کلیدی value مقدار دريافت شده به متغير انتساب مي يابد و چون در سطح کلاس عمومي است در تمام کلاس قابل دسترسي است .

مثال : يك برنامه ي سي شارپ جديد console را در VS.NET باز کنيد و نام آنرا در ابتدا ex12 انتخاب نماييد . سپس از منوي پروژه يك کلاس جديد به آن اضافه نماييد (به نام clsDate) و کد زير را درون آن بنويسيد :

```
using System;

namespace ex12
{
public class clsDate
{
private int Year;

public clsDate()
{
}

public int setYear
{
set
{
Year = value;
}
}

public bool IsLeapYear
{
get
{
return System.DateTime.IsLeapYear(Year);
}
}

}
}
```

براي استفاده از آن در متد main برنامه به صورت زير عمل مي کنيم :

```
using System;

namespace ex12
{
class Class1
{
```

```

[STAThread]
static void Main(string[] args)
{
    clsDate m_var = new clsDate();

    m_var.setYear = 1990;

    if (m_var.IsLeapYear)
        Console.WriteLine("1990 is a leap year.");
    else
        Console.WriteLine("1990 is not a leap year.");

    Console.ReadLine();
}
}
}

```

توضیحاتی در مورد کد فوق :

- 1- نحوه ی تعریف متغیر از یک کلاس جزو اساسی ترین قسمت های کار با یک کلاس محسوب می شود که در قسمت های پیشین نیز معرفی گردید .
- 2- هنگامی که از if استفاده می کنیم لزومی ندارد حتما بنویسیم `m_var.IsLeapYear==true` . همین که این خاصیت ذکر می شود در وهله ی اول `true` بودن آن چک خواهد شد .
- 3- نحوه ی مقدار دهی به یک خاصیت را هم در کد فوق ملاحظه می نمایید . در هنگام استفاده از خاصیت ها نیازی به آوردن پرانتزها (در مقابل نام آنها وجود ندارد) .
- 4- برای مرور ، نحوه ی معرفی خاصیت ها با `get` نیز بیان گردید . با استفاده از `set` و `get` می توان به کلاس ها ، مقادیر متغیرها را پاس کرد و یا مقداری را دریافت نمود.

آموزش C# قسمت هفتم

تعریف متدها در سی شارپ

در این قسمت به یکی از مهمترین مباحث برنامه نویسی سی شارپ می‌رسیم .
متدها در سی شارپ و یا همان توابع در زبان C ، اعضای یک شیء یا کلاس هستند و مجموعه ای از یک سری از کارها را انجام می‌دهند . فرض کنید در برنامه ی شما ، قسمتی باید یک عملیات ریاضی خاص را انجام دهد و این قسمت از کد که شامل چندین خط نیز می‌گردد باید بارها و بارها در برنامه صدا زده شود . برای نظم بخشیدن به برنامه ، آنها را می‌توان به صورت توابع بسته بندی کرد و بجای نوشتن چندین خط تکراری، فقط نام این بسته (تابع) و پارامترهای آن را فراخوانی نمود .

در سی شارپ یک تابع به صورت زیر تعریف می‌شود :

```
(نوع و اسامی پارامترها) نام تابع نوع خروجی تابع سطح دسترسی به تابع  
}  
بدنه ی تابع  
{
```

برای تعریف یک متد یا تابع ابتدا سطح دسترسی به آن مانند public و private سپس نوع خروجی تابع مانند void (هیچی) ذکر می‌گردد که داخل این پرانتزها می‌توان ورودی‌های تابع یا بقولی آرگومان‌های ورودی را معرفی کرد . سپس تابع باید با { شروع و با یک } خاتمه یابد .

برای مثال :

```
public int myFunc( int x )  
{  
.....  
}
```

هر تابعی می‌تواند صفر تا تعداد بیشماري آرگومان ورودی و صفر تا تعداد بیشماري خروجی داشته باشد . بوسیله یک تابع می‌توان پیچیدگی کار را مخفی کرد و صرفاً با صدا زدن نام آن ، یک سری از عملیات را انجام داد . گاهی از اوقات لازم می‌شود دو یا چند تابع با یک نام داشته باشیم بطوریکه پارامترهای ورودی یا مقادیر خروجی و یا نوع آرگومان‌های ورودی آنها با هم متفاوت باشد به این کار overloading می‌گویند . بسیاری از کلاس‌های دات نت فریم‌ورک متدها و یا توابع مفید حاضر و آماده ای را دارند . برای مثال کلاس DateTime ، متدی به نام ToString دارد که تاریخ را به صورت یک رشته طولانی بر می‌گرداند .

توابع void

توابعی که با نوع void معرفی می‌شوند هیچ خروجی ندارند و در زبان ویژوال بیسیک به آنها subها و در دلفی به آنها procedure می‌گویند .

بازگرداندن یک مقدار از یک تابع :

پس از اینکه عملیات یک مجموعه از کدها درون تابع به پایان رسید با استفاده از کلمه ی return می‌توان خروجی تابع را معرفی کرد . لازم به ذکر است ، هرجایی این کلمه ی return ذکر شود کار تابع خاتمه می‌یابد .

بهتر است موارد فوق را با چند مثال مرور کنیم :

مثال : يك برنامه ي سي شارپ جديد console را در VS.NET باز كنيد و نام آنرا در ابتدا ex13 انتخاب نماييد . در اينجا مي خواهيم تابعي را تعريف كنيم كه سه برابر جذر يك عدد را بر مي گرداند .

```
using System;

namespace ex13
{
class Class1
{
[STAThread]
static void Main(string[] args)
{
Console.WriteLine( int3SQL(3) );
Console.ReadLine();
}

public static double int3SQL( double intInput )
{
double i=0;
i = Math.Sqrt( intInput );
return i;
}
}
}
```

توضیحاتي در مورد كد فوق :

- 1- از شيء Math در سي شارپ مي توان براي انجام يك سري عمليات رياضي ابتدائي استفاده كرد . در اينجا از متد جذر گرفتن آن استفاده شده است .
- 2- در تعريف تابع خودمان از كلمه ي كليدي static استفاده شده است . درون تابع Main نمي توان توابع غير استاتيک را فراخواني كرد . فعلا اين نکته را بخاطر را داشته باشيد تا در مقالات بعدي بيشتر راجع به آن صحبت شود .
- 3- بد نيست تابع تعريف شده را كمی بيشتر آناليز كنيم :

```
public static double int3SQL( double intInput )
{
double i=0;
i = Math.Sqrt( intInput );
return i;
}
```

ابتدا سطح دسترسي به تابع ذكر شده است . پابليك ، يعني اين تابع خارج از كلاس يك برنامه نيز قابل دسترسي است . سپس از كلمه ي static استفاده گرديده كه توضيح مختصري را در مورد آن ملاحظه كرديد . در ادامه نوع خروجي تابع كه در اينجا double مي باشد معرفي گرديده است . دقت كنيد كه حتما بايد نوع تعريف شده با مقداري كه يك تابع بر مي گرداند يكسان باشد و گرنه با يك خطا برنامه متوقف مي شود . سپس نام تابع تعريف شده است . داخل پرانتز ها نوع و نام آرگوماني ارائه شده است كه در بدنه ي تابع استفاده مي گردد . اگر به تعداد بيشتري پارامتر و يا آرگومان نياز بود مي توان آنها را با ، از هم جدا كرد . پس از اينكه عمليات تابع خاتمه مي يابد با استفاده از return اين خروجي را معرفي مي نماييم . براي استفاده از اين تابع به سادگي نام تابع و سپس پرانتزها به همراه يك عدد دلخواه را مي نويسم كه آنرا در متد Main برنامه مي توان مشاهده كرد .

تعريف توابع در كلاس هاي ديگر برنامه و نحوه ي استفاده از آنها :

يکي از زيبايي هاي برنامه نويسي شيء گرا نظم و ترتيب و بسته بندي کارها مي باشد که اصطلاحا در اينجا به آن encapsulation مي گويند . يعني ما يك سري از توابع و خواص را درون کپسولي به نام کلاس قرار مي دهيم تا به سادگي بارها و بارها از آن استفاده نماييم .
براي اينکار به سادگي يك توابع را به صورت معمول درون کلاس تعريف مي نماييم و سپس همانند خواص که در مورد آنها صحبت شد ، از توابع مي توان استفاده کرد با اين تفاوت که هنگام کار با توابع حتي اگر آنها هيچ آرگومان و يا پارامتر ورودي هم نداشته باشند ذکر پرانتزها الزامي است .

مثالي ديگر در اين زمينه :

مثال : يك برنامه ي سي شارپ جديد console را در VS.NET باز کنيد و نام آنرا در ابتدا ex14 انتخاب نماييد . سپس از منوي پروژه يك کلاس جديد را به برنامه اضافه نماييد (نام آنرا clsTools بگذاريد) .

```
using System;

namespace ex14
{
public class clsTools
{
public clsTools()
{
}

public uint intCalc ( uint a , uint b )
{
uint c = Math.Min (a,b);
double x = Math.Sqrt(c) ;
uint w = Convert.ToUInt32 ( x);
return w;
}
}
}
```

سپس در متد Main برنامه مي توان به صورت زير از آن استفاده کرد :

```
using System;

namespace ex14
{
class Class1
{
[STAThread]
static void Main(string[] args)
{
clsTools m_var = new clsTools();
Console.WriteLine( m_var.intCalc(4,9));
Console.ReadLine();
}
}
}
```

توضيحاتي در مورد کد فوق :

1-تابع intCalc ما دو عدد صحيح مثبت را مي گيرد و سپس جذر کوچکترين دو عدد ورودي را محاسبه مي

- کند .
- 2- برای تبدیل نوع های عددی مختلف به هم می توان از شیء Convert استفاده کرد .
- 3- بدون استفاده از شیء Convert یکبار برنامه را اجرا کنید و دلیل خطای بوجود آمده را بیان نمایید

آموزش C# قسمت هشتم

چگونه از يك تابع بيش از يك خروجي دريافت كنيم .

ظاهرا به نظر مي رسد كه توابع فقط مي توانند يك return داشته باشند و بلافاصله پس از فراخواني return كار تابع پايان يافته است . در سي شارپ دو كلمه ي كليدي به نام هاي ref و out اضافه شده اند كه اين امر را ساده تر مي كنند .

استفاده از كلمه ي كليدي out

از out در تعريف تابع قبل از معرفي نوع آرگومان ورودي استفاده مي كنيم . در اين حالت بجاي اينكه به اين آرگومان ، آرگومان ورودي بگويم ، مي توان آنرا آرگومان خروجي ناميد . تا يك مثال را در اين زمينه با هم مرور نكنيم اين مورد مفهوم نخواهد بود :

مثال : يك برنامه ي سي شارپ جديد console را در VS.NET باز كنيد و نام آنرا در ابتدا ex15 انتخاب نماييد . سپس كد زير را درون آن بنويسيد :

```
using System;

namespace ex15
{
class Class1
{

public static int TestOut(out char i)
{
i = 'b';
return -1;
}

[STAThread]
static void Main(string[] args)
{
char i; // variable need not be initialized
Console.WriteLine(TestOut(out i));
Console.WriteLine(i);
Console.ReadLine();

}
}
}
```

توضيحاتي در مورد كد فوق :

- 1- در تابع TestOut آرگومان i از با كلمه ي كليدي out مشخص شده است . يعني اينكه درون تابع هر گونه تعييري روي i انجام شود ، خارج از تابع قابل دسترسي است .
- 2- توابعي كه داراي آرگومانهايي تعريف شده با كلمه ي كليدي out هستند نيز مي توانند از return هم استفاده كنند . همانند مثال فوق .

استفاده از كلمه ي كليدي ref

این کلمه ی کلیدی نیز دقیقا همانند out عمل می کند و نحوه ی تعریف و استفاده از آن نیز مشابه است با این تفاوت که آرگومانی که به این نوع توابع فرستاده می شود باید مقدار دهی اولیه شده باشد .

مثال : یک برنامه ی سی شارپ جدید console را در VS.NET باز کنید و نام آنرا در ابتدا ex16 انتخاب نمایید . سپس کد زیر را درون آن بنویسید :

```
using System;

namespace ex16
{
class Class1
{
public static void FillArray(ref int[] arr)
{
// Create the array on demand:
if (arr == null)
arr = new int[10];
// Otherwise fill the array:
arr[0] = 123;
arr[4] = 1024;
}

[STAThread]
static void Main(string[] args)
{
// Initialize the array:
int[] myArray = {1,2,3,4,5};

// Pass the array using ref:
FillArray(ref myArray);

// Display the updated array:
Console.WriteLine("Array elements are:");
for (int i = 0; i < myArray.Length; i++)
Console.WriteLine(myArray[i]);

Console.ReadLine();
}
}
}
```

توضیحاتی در مورد کد فوق :

1 - همانطور که ملاحظه می کنید در اینجا هنگام استفاده از تابع FillArray باید آرگومانی را که می خواهیم به آن پاس کنیم مقدار دهی اولیه کنیم .
2 - پس می توان نتیجه گرفت آرگومانهایی که با out تعریف می شوند به صورت خالص خروجی هستند و نیازی به مقدار دهی اولیه هنگام استفاده از آنها وجود ندارد . از ref هنگامی استفاده می کنیم که بخواهیم روی متغیر موجود و مقدار دهی شده ی خارج از تابع ، درون تابع عملیاتی صورت گیرد و سپس همان متغیر دستکاری شده ، عودت داده شود .

تعریف تابعی با تعداد آرگومانهای نامعلوم :

گاهی از اوقات نیاز است تا تابعی تعریف کنیم که تعداد آرگومانهای آن متغیر باشند . برای این منظور از کلمه ی کلیدی params استفاده می شود .

دونکته در اینجا حائز اهمیت است :

- 1- در هر تابعی تنها می توان یکبار از params استفاده کرد .
- 2- پس از یکبار بردن params دیگر نمی توان هیچ آرگومانی را تعریف کرد .

یکی از مثالهایی که در این زمینه می توان ارائه داد استفاده از آرایه ها به عنوان آرگومان ورودی است . در این حالت یا می توان یک آرایه را به صورت کامل به تابع معرفی کرد و یا تنها نام آنرا به تابع پاس کرد .مثال زیر را ملاحظه کنید :

مثال : یک برنامه ی سی شارپ جدید console را در VS.NET باز کنید و نام آنرا در ابتدا ex17 انتخاب نمایید . سپس کد زیر را درون آن بنویسید :

```
using System;

namespace ex17
{
class Class1
{
public static void UseParams(params int[] list)
{
for ( int i = 0 ; i < list.Length ; i++ )
Console.WriteLine(list[i]);
Console.WriteLine();
}

[STAThread]
static void Main(string[] args)
{
UseParams(1, 2, 3);

int[] myarray = new int[3] {10,11,12};
UseParams(myarray);

Console.ReadLine();

}
}
}
```

توضیحاتی در مورد کد فوق :

- 1- در تابع main به دو صورت از تابع UseParams ما استفاده شده است . یا اینکه خیلی ساده هر تعداد آرگومان را می توان به تابع فرستاد و یا اینکه در ادامه آرایه ای رسماً تعریف و سپس به تابع فرستاده شود .
- 2- نحوه ی تعریف و استفاده از آرایه ها به صورت آرگومان ورودی را نیز می توان در مثال فوق آموخت

آموزش C# قسمت نهم

مبحث overloading

گاهی از اوقات لازم است تا نگارش های مختلفی از یک تابع داشته باشیم. برای مثال تعریف سه تابع با یک نام اما با آرگومانهای مختلف. به این نوع توابع و یا متدها اصطلاحاً Overloaded Methods می گویند. (فکر کنم آنرا سربارگذاری توابع ترجمه کرده اند!) برای مثال :

```
void myMethod(int p1);  
void myMethod(int p1, int p2);  
void myMethod(int p1, string s1);
```

مثال : یک برنامه ی سی شارپ جدید console را در VS.NET باز کنید و نام آنرا در ابتدا ex18 انتخاب نمایید . سپس کد زیر را درون آن بنویسید :

```
using System;  
  
namespace ex18  
{  
class Class1  
{  
[STAThread]  
static void Main(string[] args)  
{  
writeIT();  
  
writeIT(12);  
  
Console.ReadLine();  
}  
  
public static void writeIT()  
{  
Console.WriteLine(" writeIT() Ver." );  
}  
  
public static void writeIT(int intI)  
{  
Console.WriteLine(" writeIT(intI) Ver. = " + intI );  
}  
  
}  
}
```

توضیحاتی در مورد کد فوق :

- 1- نحوه ی تعریف دو تابع با یک نام را ملاحظه می نمایید. اینکار در زبان سی ممنوع است !
- 2- کامپایلر به صورت هوشمند بر اساس نوع و تعداد آرگومانهای ورودی ، ورژن مناسب را انتخاب و اجرا می کند .

نمونه ي ضعيفي از اين بحث در وي بي 6 به صورت تعريف توابعي با پارامترهاي Optional وجود داشت .

مباحث تکميلي آرايه ها (آرايه هاي چند بعدي) :

آرايه هاي معمولي (يك بعدي) را مي توان يك رديف با تعدادي خانه خالي آماده ي پر شدن در نظر گرفت . آرايه ي دوبعدي را مي توان مانند يك جدول تشکليل شده از رديف ها و ستون ها در نظر گرفت و الي آخر ... سي شارپ دو نوع آرايه ي چند بعدي را پشتيباني مي کند rectangular and jagged در يك آرايه ي rectangular هر رديف ، طولش با رديف بعدي يکي است . آرايه ي jagged در حقيقت آرايه اي از آرايه ها است ، بنابراین هر کدام از آنها مي تواند طول مختلفي داشته باشد .

تعريف يك آرايه ي دوبعدي به صورت زير است :

type [,] array-name

مثال : يك برنامه ي سي شارپ جديد console را در VS.NET باز کنيد و نام آنرا در ابتدا ex19 انتخاب نماييد . سپس کد زير را درون آن بنويسيد :

```
using System;

namespace ex19
{
class Class1
{
[STAThread]
static void Main(string[] args)
{
const int rows = 4;
const int columns = 3;
// declare a 4x3 integer array
int[,] rectangularArray = new int[rows, columns];
// populate the array
for (int i = 0; i < rows; i++)
{
for (int j = 0; j < columns; j++)
{
rectangularArray[i,j] = i+j;
}
}
// report the contents of the array
for (int i = 0; i < rows; i++)
{
for (int j = 0; j < columns; j++)
{
Console.WriteLine("rectangularArray[{0},{1}] = {2}",
i,j,rectangularArray[i,j]);
}
}

Console.ReadLine();
}
}
}
```

توضیحاتی در مورد کد فوق :

- 1- نحوه ی تعریف ، مقدار دهی اولیه و استفاده از آرایه های دو بعدی را در مثال فوق ملاحظه می نمایید .
- 2- در یک آرایه ی دوبعدی محل قرار گیری ردیف ها و ستون ها برای مثال به صورت زیر است :

```
new int[rows, columns]-
```

استفاده از آرایه های چند بعدی :

- مثال : یک برنامه ی سی شارپ جدید console را در VS.NET باز کنید و نام آنرا در ابتدا ex20 انتخاب نمایید . سپس کد زیر را درون آن بنویسید :

```
using System;

namespace ex20
{
class Class1
{
[STAThread]
static void Main(string[] args)
{
const int rows = 4;
const int columns = 3;
// imply a 4x3 array
int[,] rectangularArray =
{
{0,1,2},
{3,4,5},
{6,7,8},
{9,10,11}
};
for (int i = 0;i < rows;i++)
{
for (int j = 0;j
{
Console.WriteLine("rectangularArray[{0},{1}] = {2}",
i,j,rectangularArray[i,j]);
}
}
}
}
}
}
```

توضیحاتی در مورد کد فوق :

- 1- در حقیقت مثال فوق تعریف آرایه ای از آرایه ها بود .
- 2- چون مقدار دهی اولیه به صورت واضحی انجام شده نیازی به ذکر ابعاد آرایه به صورت صحیح وجود نداشت

آموزش C# قسمت دهم

Jagged arrays

Jagged arrays آرایه ای از آرایه ها است و همانطور که ذکر شد لزومی ندارد که هر ردیف آن با ردیف بعدی هم طول باشد. هنگام تعریف این نوع آرایه شما تعداد ردیف ها را مشخص می نمایید. هر ردیف یک آرایه را نگهداری می کند. در اینجا هر آرایه باید تعریف شود. روش تعریف array Jagged به صورت زیر است
type [] []...

در اینجا تعداد براکت ها بیانگر ابعاد آرایه می باشد. برای مثال آرایه ی زیر دو بعدی است :

```
int [] [] myJaggedArray;
```

و برای مثال برای دسترسی به پنجمین عنصر آرایه ی سوم به صورت زیر عمل می شود :

```
myJaggedArray[2][4]
```

مثال : یک برنامه ی سی شارپ جدید console را در VS.NET باز کنید و نام آنرا در ابتدا ex21 انتخاب نمایید . سپس کد زیر را درون آن بنویسید :

```
using System;

namespace ex21
{
class Class1
{
[STAThread]
static void Main(string[] args)
{

const int rows = 4;
// declare the jagged array as 4 rows high
int[][] jaggedArray = new int[rows][];
// the first row has 5 elements
jaggedArray[0] = new int[5];
// a row with 2 elements
jaggedArray[1] = new int[2];
// a row with 3 elements
jaggedArray[2] = new int[3];
// the last row has 5 elements
jaggedArray[3] = new int[5];
// Fill some (but not all) elements of the rows
jaggedArray[0][3] = 15;
jaggedArray[1][1] = 12;
jaggedArray[2][1] = 9;
jaggedArray[2][2] = 99;
jaggedArray[3][0] = 10;
jaggedArray[3][1] = 11;
jaggedArray[3][2] = 12;
jaggedArray[3][3] = 13;
```

```

jaggedArray[3][4] = 14;
for (int i = 0; i < 5; i++)
{
    Console.WriteLine("jaggedArray[0][{0}] = {1}",
        i, jaggedArray[0][i]);
}
for (int i = 0; i < 2; i++)
{
    Console.WriteLine("jaggedArray[1][{0}] = {1}",
        i, jaggedArray[1][i]);
}
for (int i = 0; i < 3; i++)
{
    Console.WriteLine("jaggedArray[2][{0}] = {1}",
        i, jaggedArray[2][i]);
}
for (int i = 0; i < 5; i++)
{
    Console.WriteLine("jaggedArray[3][{0}] = {1}",
        i, jaggedArray[3][i]);
}

Console.ReadLine();

}
}
}

```

توضیحاتی در مورد کد فوق :

هنگام کار با آرایه های rectangular برای دسترسی به اعضا به صورت زیر عمل می شود :

```
rectangularArrayrectangularArray[i,j]
```

اما در اینجا بدین صورت است :

```
jaggedArray[3][i]
```

استفاده از System.Array

دات نت فریم ورک کلاسی را معرفی کرده است به نام Array. توسط این کلاس کار با آرایه ها و اعمال روی آنها برای مثال سورت کردن و غیره به شدت ساده می شود .

مثال : یک برنامه ی سی شارپ جدید console را در VS.NET باز کنید و نام آنرا در ابتدا ex22 انتخاب نمایید . سپس کد زیر را درون آن بنویسید :

```

using System;

namespace ex22
{
    class Class1
    {

```

```

public static void PrintMyArray(object[] theArray)
{
    foreach (object obj in theArray)
    {
        Console.WriteLine("Value: {0}", obj);
    }
    Console.WriteLine("\n");
}

[STAThread]
static void Main(string[] args)
{
    String[] myArray = {
        "Who", "is", "John", "Galt"
    };
    PrintMyArray(myArray);
    Array.Reverse(myArray);
    PrintMyArray(myArray);
    String[] myOtherArray = {
        "We", "Hold", "These", "Truths",
        "To", "Be", "Self", "Evident" };

    PrintMyArray(myOtherArray);
    Array.Sort(myOtherArray);
    PrintMyArray(myOtherArray);

    Console.ReadLine() ;

}
}
}

```

توضیحاتی در مورد کد فوق :

از دو متد Sort و Reverse در اینجا برای سورت کردن و نمایش آرایه به ترتیب معکوس (از انتها به ابتدا) استفاده گردیده است .

تعریف آرایه های دینامیک در سی شارپ :

یکی از مشکلاتی که با آرایه های معمول وجود دارد این است که قبل از هر کاری باید طول آنها را مشخص کرد .گاهی از اوقات ما دقیقاً نمی دانیم برنامه چه تعداد عضو را دریافت می کند تا آرایه ای از پیش تعریف شده با همان تعداد عضو ایجاد کنیم .برای حل این مشکل از کلاس ArrayList تعریف شده در دات نت فریم ورک می توان استفاده کرد .

هنگام استفاده از ArrayList نیازی به دانستن تعداد اعضایی که باید اضافه شوند نمی باشد و با استفاده از متد Add آن به سادگی می توان اعضاء را به آن اضافه نمود .تعدادی از خواص و متدهای این کلاس به صورت زیر هستند :

FixedSize , ReadOnly , Repeat , Synchronized , Capacity,Count , IsFixedSize Adapter
 Clear , Clone , IsReadOnly , IsSynchronized , Item , SyncRoot , Add , AddRange , BinarySearch
 InsertRange , LastIndexOf , Contains , CopyTo , GetEnumerator , GetRange , IndexOf , Insert
 SetRange , Sort , ToArray , TrimToSize Remove , RemoveAt , RemoveRange , Reverse

مثال : يك برنامه ي سي شارپ جديد console را در VS.NET باز كنيد و نام آنرا در ابتدا ex23 انتخاب نماييد .
سپس كد زير را درون آن بنويسيد :

```
using System;
using System.Collections;

namespace ex23
{
// a simple class to store in the array
public class Employee
{
public Employee(int empID)
{
this.empID = empID;
}
public override string ToString( )
{
return empID.ToString( );
}
public int EmpID
{
get
{
return empID;
}
set
{
empID = value;
}
}
private int empID;
}

class Class1
{
[STAThread]
static void Main(string[] args)
{

ArrayList empArray = new ArrayList( );
ArrayList intArray = new ArrayList( );
// populate the array
for (int i = 0;i<5;i++)
{
empArray.Add(new Employee(i+100));
intArray.Add(i*5);
}
// print all the contents
for (int i = 0;i
{
Console.Write("{0} ", intArray[i].ToString( ));
}
Console.WriteLine("\n");
```



```
// print all the contents of the button array
for (int i = 0; i
{
Console.WriteLine("{0} ", empArray[i].ToString( ));
}
Console.WriteLine("\n");
Console.WriteLine("empArray.Capacity: {0}",
empArray.Capacity);

Console.ReadLine();

}
}
}
```

توضیحاتی در مورد کد فوق :

- 1- با کلمه ی کلیدی override در قسمت های بعدی آشنا خواهیم شد .
- 2- برای استفاده از ArrayList لازم بود تا فضای نامی را که این کلاس در آن تعریف شده است ، به برنامه اضافه کرد .
- 3- در مثال فوق نحوه ی تعریف دو کلاس را در يك فضای نام مشاهده می نمایيد .
- 4- نحوه ی تعریف و مقدار دهی ArrayList و همچنین استفاده از خواص آن در مثال فوق بررسی شده است

آموزش C# قسمت یازدهم

از این قسمت به بعد می خواهیم نگاهی دقیق تر به بحث شیء گرایی در سی شارپ بیاندازیم؛ همانند فضاهای نام ، کلاس ها ، ارث بری ، پلی مرفیسم و غیره .

در قسمت های قبل آشنایی مختصری با فضاهای نام پیدا کردیم . در ادامه جزئیات بیشتری را در مورد آن بررسی خواهیم کرد .

فضاهای نام (namespaces) برای اداره کردن و نظم بخشیدن به کدها ارائه شده اند . همچنین از امکان تشابه اسمی در بین قسمت های مختلف برنامه نیز جلوگیری می کنند . استفاده از آنها عادت پسندیده ای است هنگامیکه قصد داریم از کد نوشته شده بارها و بارها استفاده کنیم .

مثال : یک برنامه ی سی شارپ جدید console را در VS.NET باز کنید و نام آنرا در ابتدا ex24 انتخاب نمایید . سپس کد زیر را درون آن بنویسید :

```
// Namespace Declaration
using System;

namespace ex24
{
    namespace tutorial
    {
        // Program start class
        class NamespaceCSS
        {
            // Main begins program execution.
            public static void Main()
            {
                // Write to console
                Console.WriteLine("This is the new Namespace.");
            }
        }
    }
}
```

توضیحاتی در مورد کد فوق :
یکی از روش های مناسب برای معرفی فضاهای نام ، ارائه ی آنها به صورت سلسله مراتبی می باشد .
قسمت های عمومی تر در بالا و قسمت های اختصاصی تر در فضاهای نام داخلی تر قرار داده می شوند .
این روش به معرفی فضاهای نام تو در تو منتهی می شود (nested namespaces) ، همانند مثال بالا .

کد فوق را به صورت زیر با استفاده از عملگر دات (.) می توان خلاصه نویسی کرد و نتیجه با قبل تفاوتی ندارد :

```
// Namespace Declaration
using System;

namespace ex24.tutorial
{
    // Program start class
```

```

class NamespaceCSS
{
// Main begins program execution.
public static void Main()
{
// Write to console
Console.WriteLine("This is the new Namespace.");
}
}
}

```

طريقه ي فراخواني اعضاي فضاهاي نام :

مثال : يك برنامه ي سي شارپ جديد console را در VS.NET باز كنيد و نام آنرا در ابتدا ex25 انتخاب نماييد . سپس كد زير را درون آن بنويسيد :

```

// Namespace Declaration
using System;

namespace ex25
{
// nested namespace
namespace tutorial
{
class myExample1
{
public static void myPrint1()
{
Console.WriteLine("calling another namespace member1.");
}
}
}

// Program start class
class NamespaceCalling
{
// Main begins program execution.
public static void Main()
{
// Write to console
tutorial.myExample1.myPrint1();
tutorial.myExample2.myPrint2();
}
}

// same namespace as nested namespace above
namespace ex25.tutorial
{
class myExample2
{
public static void myPrint2()
{

```

```

Console.WriteLine("calling another namespace member2.");
}
}
}

```

توضیحاتی در مورد کد فوق :
در کد فوق نحوه ی استفاده از اعضای تعریف شده در فضاها ی نام را می توان مشاهده کرد . نحوه ی استفاده از آنها همانطور که در قسمت های قبل نیز گفته شد به صورت زیر است :

ProjectName.Namespace.ClassName.MemberName

برای مثال در فضا ی نام tutorial کلاس myExample1 قرار دارد و داخل آن متد myPrint1 تعریف شده است . پس نحوه ی دسترسی به متد آن به صورت زیر است :

```
tutorial.myExample1.myPrint1();
```

کلاس های myExample1 و myExample2 هر دو به یک فضا ی نام (ex25.tutorial) تعلق دارند ، هر چند جدا از هم نوشته شده اند . حتی آنها را با حفظ سلسله مراتب خودشان می توان در فایل های جداگانه ای نیز نوشت .

استفاده از using

مثال : یک برنامه ی سی شارپ جدید console را در VS.NET باز کنید و نام آنرا در ابتدا ex26 انتخاب نمایید . سپس کد زیر را درون آن بنویسید :

```

// Namespace Declaration
using System;
using ex26.tutorial;

// Program start class
class UsingDirective
{
// Main begins program execution.
public static void Main()
{
// Call namespace member
myExample.myPrint();
}
}

// C# Namespace
namespace ex26.tutorial
{
class myExample
{
public static void myPrint()
{
Console.WriteLine("Example of using a using directive.");
}
}
}
}

```

توضیحاتی در مورد کد فوق :

همانند مثال بالا ، برای خلاصه نویسی می توان از کلمه `using` به همراه نام `namespace` مورد نظر استفاده کرد . برای مثال اگر متد `WriteLine` را بخواهیم کامل بنویسیم به صورت زیر است :

```
System.Console.WriteLine(...);
```

اما با قید کردن و الحاق کردن فضای نام آن ، دیگر نیازی به ذکر `System` در ابتدای آن نیست .

نکته :

باز هم می توان خلاصه نویسی بیشتری را ارائه داد

```
using csTut = ex26.tutorial.myExample; // alias
```

در این صورت تنها کافی است متد کلاس تعریف شده در آنرا به صورت زیر فراخوانی کنیم :

```
csTut.myPrint();
```

آموزش C# قسمت دوازدهم

کلاس ها در سي شارپ :

تا بحال در حد کاربرد ، با کلاس ها آشنا شده ایم . اما در این قسمت مي خواهيم نگاهی دقیق تر به کلاس ها بیاندازيم .

هر کدي در سي شارپ قسمتي از يك کلاس مي باشد و ترکیب تمام خواص و متدهاي موجود در يك کلاس يك نوع داده ي جديد تعريف شده از طرف ما را پديد مي آورد . هر متغيري که از کلاس ساخته شود ، شيء نامیده مي شود و يك کپي منحصر به فرد است . براي مثال برنامه ي زير را در نظر بگيريد :

```
using System;

class Data
{
public int x;
}
class App
{
public static void Main()
{
Data d1 = new Data();
d1.x = 1;
Data d2 = new Data();
d2.x = 2;
Console.WriteLine("d1.x = {0}", d1.x);
Console.WriteLine("d2.x = {0}", d2.x);
}
}
```

در اینجا کلاس Data تعريف شده است و داراي يك عضو به نام x مي باشد . به این نوع داده در کلاس فيلد گفته مي شود و هنگامیکه به صورت public معرفي مي شود يعني خارج از کلاس نیز قابل دسترسي است . در کد بالا دو متغير از کلاس تعريف و مقدار دهی اولیه شده اند . خروجي برنامه به صورت زیر است :

```
d1.x = 1
d2.x = 2
```

دلیل این خروجي آن است که هر instance (نمونه) از کلاس منحصر بفرد است و در اینجا نمي توان انتظار داشت که هر دو خروجي یکی شوند .

براي مقدار دهی اولیه متغيرهايي که به صورت فيلد تعريف مي شوند ، بهتر است مقدار دهی آنها را در سازنده ي کلاس (constructor) انجام دهيم .

```
class Data
{
public int x;
public Data(){x = 99;}
}
```

همانطور که پیشتر نیز ذکر شد ، متدی که هم نام کلاس است ، سازنده نام می گیرد . یک کلاس می تواند بیش از یک سازنده داشته باشد . برای مثال :

```
class Data
{
public int x;
private Data(){ }
public Data(int y){x = y;}
public Data(int y, int z){x = y + z;}
}
```

از آنجائیکه که سازنده ی بدون پارامتر ذکر شده در کد فوق private تعریف شده است بنابراین خارج از کلاس دیگر قابل دسترسی نمی باشد . بنابراین کدی خارج از کلاس ، تنها می تواند از دو سازنده ی دیگر استفاده کند . برای مثال تعریف دو متغیر جدید از این کلاس به صورت زیر می باشد :

```
Data d1 = new Data(44);
Data d2 = new Data(22, 33);
```

سی شارپ به شما اجازه می دهد تا سازنده ها را در یک کلاس توسط کلمه ی کلیدی this نیز فراخوانی کنید یعنی بجای ذکر نام متد سازنده از کلمه ی this استفاده شود (در خود کلاس) .

اگر می خواهید متغیری را بین نمونه (instance) های مختلف یک کلاس به اشتراک بگذارید کلمه ی کلیدی static وارد صحنه می شود . به مثال زیر توجه کنید :

```
using System;
```

```
class Counted
{
public static int count = 0;
public Counted()
{
count++;
}
public int GetInstanceCount()
{
return count;
}
}
class App
{
public static void Main()
{
Counted d1 = new Counted();
Console.WriteLine("current total {0}", d1.GetInstanceCount());
Counted d2 = new Counted();
Console.WriteLine("current total {0}", d2.GetInstanceCount());
Console.WriteLine("total {0}", Counted.count);
}
}
```

باید خاطر نشان کرد که متغیرهای استاتیک توسط نمونه های کلاس قابل دستیابی نیستند و فقط درون کلاس به شکل زیر می توان از آنها استفاده کرد :

در مثال فوق دو نمونه از کلاس Counted تعریف شده است. با هر بار فراخوانی کلاس ، خودیخود سازنده اجرا شده و یک عدد به این شمارشگر استاتیک اضافه می شود. همانطور که ذکر شد، برای اینکه بتوان به این متغیر استاتیک در خارج از کد دسترسی پیدا کرد یک متد غیر استاتیک تعریف شده است .

در مثال فوق تابع GetInstanceCount تنها یک عدد را بر می گرداند. در برنامه نویسی شیء گرا مرسوم است که در این حالت به جای توابع از خواص استفاده شود که به اندازه ی کافی در مورد آنها در قسمت های قبل توضیح داده شد. در این صورت تعریف فوق به صورت زیر در می آید :

```
class Counted
{
public static int x = 0;
public Counted()
{
x++;
}
public int InstanceCount // property
{
get{return x;}
}
}
```

و در این صورت قسمت بعدی کد به صورت زیر اصلاح می شود (فراخوانی خواص ، بدون ذکر پرانتزها بعد از نام آنها صورت می گیرد) :

```
Counted d1 = new Counted();
Console.WriteLine("current total {0}", d1.InstanceCount);
Counted d2 = new Counted();
Console.WriteLine("current total {0}", d2.InstanceCount);
```

اگر یک خاصیت هم خواندنی و هم نوشتنی باشد به صورت زیر تعریف می شود :

```
private string name;
public string Name
{
get{return name;}
set{name = value;}
}
```

فیلدهای پابلیک را می توان خواند و یا تغییر داد. اگر لازم باشد تا کاربر نتواند آنها را تغییر دهد می توان از کلمه ی کلیدی readonly قبل از تعریف آنها استفاده کرد. مثال :

```
class Data
{
public readonly int x = 42;
}
```

بحث کلاس ها ادامه دارد...

آموزش C# قسمت سیزدهم

با استفاده از ایندکسرها می توان با يك كلاس همانند آرایه ها رفتار کرد. به مثال زیر توجه کنید :

```
using System;

///
/// A simple indexer example.
///
class IntIndexer
{
    private string[] myData;

    public IntIndexer(int size)
    {
        myData = new string[size];

        for (int i=0; i < size; i++)
        {
            myData[i] = "empty";
        }
    }

    public string this[int pos]
    {
        get
        {
            return myData[pos];
        }
        set
        {
            myData[pos] = value;
        }
    }

    static void Main(string[] args)
    {
        int size = 10;

        IntIndexer myInd = new IntIndexer(size);

        myInd[9] = "Some Value";
        myInd[3] = "Another Value";
        myInd[5] = "Any Value";

        Console.WriteLine("\nIndexer Output\n");

        for (int i=0; i < size; i++)
```

```

{
Console.WriteLine("myInd[{0}]: {1}", i, myInd[i]);
}
}
}
}

```

در مثال فوق نحوه ی تعریف و استفاده از ایندکسرها را می توان مشاهده کرد. کلاس `IntIndexer` حاوی آرایه ای به نام `myData` می باشد. بدلیل `private` بودن آن در خارج از کلاس قابل دسترسی نیست. این آرایه در سازنده ی کلاس (`IntIndexer`) با کلمه ی `empty` مقدار دهی اولیه شده است. عضو بعدی کلاس `Indexer` می باشد و با کلمه ی کلیدی `this` و براکتها مشخص شده ست (`this[int pos]`). همانطور که ملاحظه می فرمایید نحوه ی تعریف ایندکسرها شبیه به تعریف خواص می باشد.

```

this [argument list]
{
get
{
// Get codes goes here
}
set
{
// Set codes goes here
}
}
}

```

خروجی مثال فوق به صورت زیر است :

```

myInd[0]: empty
myInd[1]: empty
myInd[2]: empty
myInd[3]: Another Value
myInd[4]: empty
myInd[5]: Any Value
myInd[6]: empty
myInd[7]: empty
myInd[8]: empty
myInd[9]: Some Value

```

استفاده از اعداد صحیح روشی است متداول برای دسترسی به اعضای آرایه ها در بسیاری از زبانها اما ایندکسرها در سی شارپ فراتر از این می رود. ایندکسرها را می توان با پارامترهای متعددی تعریف کرد و هر پارامتر با نوعی مختلف (دقیقا همانند پارامترهای ورودی متدها). البته محدودیتی که اینجا وجود دارد در مورد نوع پارامترها است که تنها می تواند `integers, enums, and strings` باشد. بعلاوه قابلیت `Overloading` ایندکسرها نیز وجود دارد. به همین جهت به آنها آرایه های هوشمند هم گفته می شود (`arrays smart`). مثال :

```

using System;

///
/// Implements overloaded indexers.
///
class OvrIndexer
{
private string[] myData;
private int arrSize;

```

```

public OvrIndexer(int size)
{
    arrSize = size;
    myData = new string[size];

    for (int i=0; i < size; i++)
    {
        myData[i] = "empty";
    }
}

public string this[int pos]
{
    get
    {
        return myData[pos];
    }
    set
    {
        myData[pos] = value;
    }
}

public string this[string data]
{
    get
    {
        int count = 0;

        for (int i=0; i < arrSize; i++)
        {
            if (myData[i] == data)
            {
                count++;
            }
        }
        return count.ToString();
    }
    set
    {
        for (int i=0; i < arrSize; i++)
        {
            if (myData[i] == data)
            {
                myData[i] = value;
            }
        }
    }
}

static void Main(string[] args)
{

```

```

int size = 10;
OvrIndexer myInd = new OvrIndexer(size);

myInd[9] = "Some Value";
myInd[3] = "Another Value";
myInd[5] = "Any Value";

myInd["empty"] = "no value";

Console.WriteLine("\nIndexer Output\n");

for (int i=0; i < size; i++)
{
Console.WriteLine("myInd[{0}]: {1}", i, myInd[i]);
}

Console.WriteLine("\nNumber of \"no value\" entries: {0}", myInd["no value"]);
}
}

```

در مثال فوق اولین ایندکسر با یک پارامتر از نوع اعداد صحیح تعریف شده است و در ایندکسر دوم از نوع رشته .
خروجی برنامه ی فوق به صورت زیر است :

```

myInd[0]: no value
myInd[1]: no value
myInd[2]: no value
myInd[3]: Another Value
myInd[4]: no value
myInd[5]: Any Value
myInd[6]: no value
myInd[7]: no value
myInd[8]: no value
myInd[9]: Some Value

```

Number of "no value" entries: 7

نکته :

- 1- امضای (لیست پارامترهای) ایندکسر ها در یک کلاس باید منحصر بفرد باشد .
- 2- تعریف یک ایندکسر به صورت استاتیک مجاز نیست .

در صورت نیاز به ایندکسرهایی با پارمترهای ورودی متعدد می توان به صورت زیر عمل کرد :

```

public object this[int param1, ..., int paramN]
{
get
{
// process and return some class data
}
set
{
// process and assign some class data
}
}

```

```
}  
}
```

يك مثال ديگر :

```
using System;  
  
class IndexExample  
{  
    string Message;  
  
    public static void Main()  
    {  
        IndexExample obj=new IndexExample("Welcome");  
  
        /* This will access the String variable Message  
        using array like notation  
        */  
        for(int i=0;i < obj.Length;i++)  
        {  
            Console.WriteLine(obj[i]);  
        }  
        obj[obj.Length-1]="e to C#";  
  
        Console.WriteLine(obj.Message);  
  
    }  
  
    public IndexExample(string s)  
    {  
        Message=s;  
    }  
  
    public string this[int i]  
    {  
        get  
        {  
            if(i >= 0 && i < Message.Length)  
            {  
                return Message.Substring(i,1);  
            }  
            else  
            {  
                return "";  
            }  
        }  
        set  
        {  
            if(i >= 0 && i < Message.Length)  
            {  
                Message=Message.Substring(0,i) + value + Message.Substring(i+1);  
            }  
        }  
    }  
}
```

```
}
```

```
public int Length
```

```
{
```

```
get
```

```
{
```

```
if(Message!=null)
```

```
{
```

```
return Message.Length;
```

```
}
```

```
else
```

```
return 0;
```

```
}
```

```
}
```

```
}
```

آموزش C# قسمت چهاردهم

ارث بری (Inheritance)

ارث بری یکی از مفاهیم اولیه ی برنامه نویسی شیء گرا می باشد. با استفاده از آن استفاده مجدد از کد موجود به نحوی مؤثر میسر می گردد و صرفه جویی قابل توجهی را در زمان برنامه نویسی پدید می آورد. به کد زیر دقت کنید :

```
using System;

public class ParentClass
{
public ParentClass()
{
Console.WriteLine("Parent Constructor.");
}
public void print()
{
Console.WriteLine("I'm a Parent Class.");
}
}

public class ChildClass : ParentClass
{
public ChildClass()
{
Console.WriteLine("Child Constructor.");
}
public static void Main()
{
ChildClass child = new ChildClass();
child.print();
}
}
```

Output:
Parent Constructor.
Child Constructor.
I'm a Parent Class.

کد فوق از دو کلاس استفاده می کند. کلاس بالایی ParentClass و کلاس اصلی ChildClass می باشد. کاری که انجام شده است استفاده از کدهای کلاس والد ParentClass در کلاس بچه (ChildClass) می باشد. برای اینکه ParentClass را بعنوان کلاس پایه برای ChildClass معرفی کنیم به صورت زیر عمل شد :

```
public class ChildClass : ParentClass
```

کلاس پایه با استفاده از معرفی کولون ":" ، پس از کلاس مشتق شده تعریف می شود. در سی شارپ تنها ارث بری یگانه پشتیبانی می شود. بنابراین تنها یک کلاس پایه را برای ارث بری می توان تعریف کرد .

ChildClass دقیقاً توانایی های ParentClass را دارا است. بنابراین می توان گفت ChildClass همان ParentClass است. برای مثال در کد فوق ChildClass دارای متد print نمی باشد اما آنرا از کلاس ParentClass به ارث برده است و در متد Main برنامه از آن استفاده گردیده است .

هنگام ساختن یک شیء از کلاس مشتق شده (derived) ، ابتدا یک نمونه از کلاس والد خود بخود ساخته می شود. این مورد در خروجی کد فوق هنگامی که متدهای سازنده ها روی صفحه چاپ شده اند قابل مشاهده است .

تبادل اطلاعات بین کلاس والد و کلاس فرزند :

به مثال زیر دقت کنید :

```
using System;

public class Parent
{
    string parentString;

    public Parent()
    {
        Console.WriteLine("Parent Constructor.");
    }

    public Parent(string myString)
    {
        parentString = myString;
        Console.WriteLine(parentString);
    }

    public void print()
    {
        Console.WriteLine("I'm a Parent Class.");
    }
}

public class Child : Parent
{
    public Child() : base("From Derived")
    {
        Console.WriteLine("Child Constructor.");
    }

    public void print()
    {
        base.print();
        Console.WriteLine("I'm a Child Class.");
    }

    public static void Main()
    {
```



```
Child child = new Child();
child.print();
((Parent)child).print();
}
}
```

Output:

```
From Derived
Child Constructor.
I'm a Parent Class.
I'm a Child Class.
I'm a Parent Class.
```

کلاس فرزند با کلاس والد در هنگام instantiation می تواند تبادل اطلاعات کند. همانطور که در مثال فوق بارز است با استفاده از کلمه ی کلیدی base ، کلاس فرزند تابع سازنده ی کلاس والد را فراخوانی کرده است. اولین خط خروجی بیانگر این موضوع است .

گاهی از اوقات ما می خواهیم تابعی را که در کلاس والد تعریف شده است را در کلاس فرزند با تعریف دیگری و مخصوص به خودمان ارائه دهیم. در اینصورت تابع تعریف شده در کلاس فرزند ، تابع هم نام والد را مخفی خواهد کرد و دیگر آن تابع والد فراخوانی نخواهد گردید. در این حالت تنها یک راه برای دسترسی به تابع اصلی والد وجود دارد و آن استفاده از base. می باشد که در کد فوق پیاده سازی شده است . با استفاده از base. می توان به تمام اعضای public و یا protected کلاس والد از درون کلاس فرزند دسترسی داشت .
راه دیگری که برای این منظور وجود دارد در آخرین خط کد فوق در متد Main پیاده سازی شده است :

```
((Parent)child).print();
```

برای تبدیل نوع های مختلف در سی شارپ می توان از پرانتز و سپس ذکر نوع اصلی استفاده کرد به این عمل casting و یا boxing هم می گویند. در کد فوق در حقیقت child به نوعی از parent تبدیل شده است . بنابراین مانند این است که یک نمونه از کلاس والد متد print همان کلاس را فراخوانی می کند.

آموزش #C قسمت پانزدهم

پلی مورفیسم (Polymorphism)

یکی دیگر از مفاهیم اولیه ی شیء گرایی پلی مورفیسم (چند ریختی) می باشد . پلی مورفیسم به معنای توانایی استفاده کردن از فرم های مختلف یک نوع است بدون توجه به جزئیات آن .
برای مثال هنگامیکه سیگنال تلفنی شما فرستاده می شود ، از نوع تلفنی که در انتهای خط موجود است خبری ندارد . تلفن انتهای خط ، می خواهد یکی از تلفن های عهد عتیق باشد و یا تلفنی با آخرین امکانات روز .

شرکت مخابرات)!(تنها از نوع پایه ای به نام phone خبر دارد و فرض می کند که هر instance از این نوع می داند که چگونه صدای زنگ تلفن شما را به صدا در آورد . بنابراین شرکت مخابرات از تلفن شما به صورت پلی مرف استفاده می کند .

در عمل پلی مورفیسم هنگامی مفید خواهد بود که بخواهیم گروهی از اشیاء را به یک آرایه نسبت دهیم و سپس متدهای هر یک را فراخوانی کنیم . الزاما این اشیاء از یک نوع نخواهند بود .

نحوه ی ایجاد متدهای پلی مرفیک :

برای ایجاد متدی که نیاز است تا پلی مورفیسم را پشتیبانی نماید ، تنها کافی است آنرا از نوع virtual در

کلاس پایه تعریف کنیم . مثال :

فرض کنید تابع DrawWindow در کلاس Window تعریف شده است . برای ایجاد قابلیت پلی مورفیسم در آن

به صورت زیر عمل می شود :

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

در این حالت هر کلاسی که از Window مشتق شود ، مجاز است نگارش خاص خودش را از DrawWindow ارائه کند . در این صورت در کلاسی که از کلاس پایه ی ما ارث می برد ، تنها کافی است که کلمه ی کلیدی override را قبل از نام تابع مذکور ذکر نماییم .

یک مثال کامل :

```
using System;
```

```
public class DrawingObject
{
public virtual void Draw()
{
Console.WriteLine("I'm just a generic drawing object.");
}
}
public class Line : DrawingObject
{
public override void Draw()
{
Console.WriteLine("I'm a Line.");
}
}
```

```

public class Circle : DrawingObject
{
public override void Draw()
{
Console.WriteLine("I'm a Circle.");
}
}

public class Square : DrawingObject
{
public override void Draw()
{
Console.WriteLine("I'm a Square.");
}
}

public class DrawDemo
{
public static int Main(string[] args)
{
DrawingObject[] dObj = new DrawingObject[4];

dObj[0] = new Line();
dObj[1] = new Circle();
dObj[2] = new Square();
dObj[3] = new DrawingObject();

foreach (DrawingObject drawObj in dObj)
{
drawObj.Draw();
}

return 0;
}
}

```

کلاس DrawingObject ، کلاسی پایه برای تمام کد ما که از آن به ارث می برد ، می باشد .متد Draw در آن با کلمه ی کلیدی virtual معرفی شده است .یعنی تمام کلاس های فرزند این کلاس والد می توانند این متد را override کنند (تحریف کردن و یا تحت الشعاع قرار دادن هم ترجمه شده است!) . در ادامه سه کلاس تعریف شده اند که تمامی آنها از کلاس مبنا ارث می برند و تابع Draw را تحریف کرده اند (.) . با استفاده از کلمه ی کلیدی override می توان تابع مجازی کلاس مبنا را با تعریفی جدید در زمان اجرای برنامه ارائه داد . تحریف شدن تنها زمانی رخ می دهد که کلاس ، توسط ریفرنس کلاس مبنا مورد ارجاع واقع شده باشد .

و در متد Main برنامه از این کلاس ها در عمل استفاده گردیده است .در متد Main ، آرایه ای از نوع DrawingObject تعریف و مقدار دهی اولیه شده است تا بتواند 4 شیء از نوع این کلاس را در خودش ذخیره کند .

بدلیل رابطه ی ارث بری موجود می توان آرایه ی dObj را با نوع هایی از کلاس های Circle ، Square و Line مقدار دهی کرد (همانند کدهای بعدی متد Main) . اگر ارث بری در اینجا وجود نمی داشت می بایست به ازای هر کلاس یك آرایه تعریف می شد . سپس از حلقه ی زیبای foreach برای حرکت در بین اعضای این آرایه استفاده گردیده است .در اینجا هر شیء متد خاص خودش را در مورد Draw فراخوانی می کند و نتیجه را روی صفحه نمایش خواهد داد . خروجی نهایی به صورت زیر خواهد بود :

Output:
I'm a Line.
I'm a Circle.
I'm a Square.
I'm just a generic drawing object.

آموزش #C قسمت شانزدهم

کلاس ها ي abstract
کلاس ها را همچنین مي توان به صورت abstract تعريف کرد. از اين نوع کلاس ها نمي توان instance ايي را ايجاد نمود. در اين کلاس هاي پایه ، صرفا تعريف متدها و خواص هايي عنوان گرديده و در آینده در کلاس هاي فرزند توسعه داده خواهند شد. براي مثال :

```
public abstract class Named
{
public abstract String Name {get; set;} // property
public abstract void PrintName(); // method
}
public class B : Named
{
private String name = "empty";
public override String Name
{
get{return name;}
set{name=value;}
}
public override void PrintName()
{
Console.WriteLine("Name is {0}", name);
}
}
```

والي که شايد پيش ببايد اين است که اگر interface ها صرفا تعريف توابع و خواص را مي توانند در خود جاي دهند پس چه دليلي براي بکار بردن آنها و طولاني کردن کار کد نويسي وجود دارد؟ کاربردهاي زيادي را مي توان براي اينترفيس ها برشمرد. اينترفيس يك رفتار را تعريف مي کند. فرض کنيد در حال توسعه ي برنامه اي هستيد که بر روي دو کامپيوتر مختلف بايد با هم در ارتباط مستقيم بوده و برهم کنش داشته باشند و هر برنامه از ماژولي به نام CCommObjz communication object استفاده مي نمايد. يکي از متدهاي اين شيء ، SendData) مي باشد که رشته اي را دريافت کرده و به برنامه ي ديگر مي فرستد. اين فراخواني از نوع asynchronous است زيرا ما نمي خواهيم اگر خطايي در شبکه رخ داد، برنامه براي هميشه منتظر باقي بماند. اما چگونه برنامه ي A که تابع ذکر شده را فراخواني کرده است مي تواند تشخيص دهد که پيغام به مقصد رسیده است يا خير و يا آيا خطايي در شبکه مانع رسيدن پيغام گشته است يا خير؟ جواب بدین صورت است که CCommObjz هنگام دريافت پيغام ، رخدادی را سبب خواهد شد و اگر خطايي رخ داده باشد خير. در اين حالت نياز به يك ماژول logging نیز احساس مي گردد تا خطاهای رخ داده را ثبت نمايد. يك روش انجام آن اين است که CCommObjz پياده سازي اين امکان را نیز بعهده گرفته و اگر فردا نیز خواستيم ماژول ديگري را به برنامه اضافه کنيم هر روز بايد CCommObjz را تغيير دهيم. تمام اين کارها را به سادگي مي توان در يك اينترفيس مدل کرد. روش آن نیز در ادامه بيان مي گردد :

در ابتدا يك اينترفيس ايجاد مي کنيم تا ليست تمام امکانات ممکن را "منتشر" کند :

```
interface ICommObjEvents
{
```

```

void OnDataSent();
void OnError();
}

```

شيء CCommObj ما از این توابع که بعداً توسعه داده خواهند شد برای باخبر سازی کلاینت ها استفاده می نماید. تمام متدها در یک اینترفیس ذاتاً پابلیک هستند بنابراین نیازی به ذکر صریح این مطلب نمی باشد و اگر اینکار را انجام دهید کامپایلر خطای زیر را گوشزد خواهد کرد :

The modifier 'public' is not valid for this item

در ادامه کلاینت CClientApp_A را پیاده سازی خواهیم کرد :

```

class CClientApp_A:ICommObjEvents
{
public void OnDataSent()
{
Console.WriteLine("OnDataSent");
}
public void OnError()
{
Console.WriteLine("OnError");
}
private CCommObj m_Server;
public void Init(CCommObj theSource)
{
m_Server = theSource;
theSource.Advise (this);
string strAdd = ("N450:1");
m_Server.read (strAdd,10);
}
}

```

در کد فوق کلاس CClientApp_A از ICommObjEvents ارث برده و تمام متدهای این اینترفیس را پیاده سازی نموده است. هنگامی که CCommObj تابع OnDataSent را فراخوانی می کند این کلاینت پیغام را دریافت خواهد کرد. لازم به ذکر است که کلاس کلاینت ما چون از یک اینترفیس ارث بری می نماید پس باید تمام توابع و خواص کلاس پایه را پیاده سازی کند در غیر اینصورت هر چند برنامه کامپایل خواهد شد اما هنگامی که شیء CCommObj هر کدام از توابع این کلاس را فراخوانی کند ، خطای زمان اجرا رخ خواهد داد .

متد Init کلاس فوق آرگومانی را از نوع CCommObj دریافت نموده و در یک متغیر private آنرا ذخیره می نماید. همچنین در این متد ، متد Advise از کلاس CCommObj نیز فراخوانی گشته است .

```

public class CCommObj
{
private int m_nIndex;
public ICommObjEvents [] m_arSinkColl;
public CCommObj()
{
m_arSinkColl = new ICommObjEvents[10];
m_nIndex = 0;
}
public int Advise(ICommObjEvents theSink)
{
m_arSinkColl[m_nIndex] = theSink;
}
}

```

```
int lCookie = m_nIndex;
m_nIndex++;
return lCookie
}
public void SendData(string strData)
{
foreach ( ICommObjEvents theSink in m_arSinkColl)
if(theSink != null )
theSink.OnDataSent ();
}
}
```

آموزش C# قسمت هفدهم

در کلاس CCommObj که با آن آشنا شدیم ، آرایه ای Private از نوع ICommObjEvents به نام m_arSinkColl وجود دارد . این آرایه تمام اینترفیس های sink شده را ذخیره می کند . واژه ی sink در اینجا به کلاسی گفته می شود که دریافت کننده ی رخدادها است .متد Advise تنها sink وارده به آنرا در یک آرایه ذخیره می کند و سپس اندیس آرایه را که در اینجا cookie نامیده شده است بر می گرداند . این کوکی توسط کلاینتی که دیگر نمی خواهد از آن آیتم هیچونه رخدادی را دریافت کند به سرور فرستاده می شود و سپس سرور این آیتم را از لیست خودش حذف خواهد کرد .

نحوه ی فراخوانی متد advise توسط کلاینت نیز جالب است .

```
public void Init(CCommObj theSource)
{
    m_Server = theSource;
    theSource.Advise (this);
    string strAdd = ("Hello");
    m_Server.read (strAdd,10);
}
```

در اینجا تنها یک this بعنوان آرگومان به متد advice فرستاده شده است در حالیکه انتظار می رفت آرگومانی از نوع ICommObjEvents به تابع فرستاده شود . دلیل صحت این عمل بدین صورت است که کلاس ClientApp_A از اینترفیس ICommObjEvents ارث برده است و آنرا پیاده سازی نموده است . در ادامه لیست کامل برنامه ی نوشته شده را در حالت Console ملاحظه می فرمایید .

```
namespace CSharpCenter
{
    using System;

    public interface ICommObjEvents
    {
        void OnDataSent();
        void OnError();
    }
}
```

```

public class CCommObj
{
private int m_nIndex;
public ICommObjEvents [] m_arSinkColl;
public CCommObj()
{
m_arSinkColl = new ICommObjEvents[10];
m_nIndex = 0;
}

public void Advise(ICommObjEvents theSink)
{

m_arSinkColl[m_nIndex] = theSink;
m_nIndex++;
}
public void SendData(string strData)
{
foreach ( ICommObjEvents theSink in m_arSinkColl)
{
if(theSink != null )
{
theSink.OnDataSent ();
}
}
}
}
class CClientApp_A:ICommObjEvents
{
public void OnDataSent()
{
Console.WriteLine("OnDataSent Client App A");
}
public void OnError()
{
Console.WriteLine("OnError");
}
public void Read()
{
string strAdd = ("Hello");
m_Server.SendData (strAdd);

}
private CCommObj m_Server;
public void Init(CCommObj theSource)
{

```



```

m_Server = theSource;
theSource.Advise (this);
}
}
class CClientApp_B:ICommObjEvents
{
public void OnDataSent()
{
Console.WriteLine("OnDataSent Client App B");
}
public void OnError()
{
Console.WriteLine("OnError");
}
private CCommObj m_Server;
public void Init(CCommObj theSource)
{
m_Server = theSource;
theSource.Advise (this);
}
}
class ConsoleApp
{
public static void Main()
{
CClientApp_A theClient = new CClientApp_A ();
CClientApp_B theClient2 = new CClientApp_B ();
CCommObj theComm = new CCommObj ();
theClient.Init (theComm);
theClient2.Init (theComm);
theClient.Read();

}
}
}

```

در متد Main برنامه ي فوق ، ما دو کلاينت تعريف کرده ایم و يك نمونه از CCommObj را دو کلاينت instance هاي CCommObj را بعنوان آرگومان دریافت کرده اند . در هنگام فراخواني init توسط هر کلاينت متد فراخواني مي گردد . در خاتمه Read مربوط به کلاينت 1 فراخواني شده است که سبب مي شود تا رخداد OnDataSend شيء CCommObj اجرا شود و به تمام کلاينت ها فرستاده شود .

هدف از این مثال ارائه ي بعضي از جنبه هاي اينترفيس ها و نحوه ي استفاده از آنها بود . دو مطلب ديگر در مورد اينترفيس ها باقي مانده اند تا به پايان بحث مربوط به آنها برسيم :

چگونه مي توان متوجه شد که يك شيء واقعا يك اينترفيس را پياده سازي کرده است؟

دو روش برای فهمیدن این موضوع وجود دارد :
-استفاده از کلمه ی کلیدی is
-استفاده از کلمه ی کلیدی as

اولین مثال زیر از کلمه ی کلیدی is استفاده می کند :

```
CClientApp_C theClient3 = new CClientApp_C ();  
if(theClient3 is ICommObjEvents)  
Console.WriteLine ("theClient3 implements ICommObjEvents");  
else  
Console.WriteLine ("theClient3 doesnot implement ICommObjEvents");
```

کلمه ی کلیدی is مقدار true را بر می گرداند اگر اپراتور سمت چپ ، اینترفیس سمت راست را پیاده سازی کرده باشد .

```
ICommObjEvents theClient5 = theClient3 as ICommObjEvents;  
if(theClient5 != null )  
Console.WriteLine ("Yes theClient implements interface");  
  
else  
Console.WriteLine ("NO,Yes theClient doesn't implements interface");
```

در مثال فوق اپراتور as در حال casting شیء theClient5 به ICommObjEvents می باشد . چون CClientApp_C اینترفیس را پیاده سازی نمی کند حاصل خط اول نال خواهد بود .

به صورت خلاصه :

یک اینترفیس قراردادی است که به کلاینت گارانتی می دهد یک کلاس خاص چگونه رفتار خواهد کرد . هنگامیکه کلاسی یک اینترفیس را پیاده سازی می کند به تمام کلاینت ها می گوید که :من تمام موارد ذکر شده در اینترفیس را ارائه و پیاده سازی خواهم کرد . نمونه ی عملی استفاده از اینترفیس ها بحث dot net remoting است.

آموزش C# قسمت هجدهم

مقابله با خطاها در سي شارپ (Exception Handling in C#)

EXCEPTION يك خطاي زمان اجر است كه بدليل شرايطي غيرنرمال در برنامه ايجاد مي شود. در سي شارپ exeption كلاسي است در فضاي نام سيستم. شيء ايي از نوع exception بيانگر شرايطي است كه سبب رخ دادن خطا در كد شده است. سي شارپ از exception ها به صورتي بسيار شبیه به جاوا و سي پلاس پلاس استفاده مي نمايد .

دلایلي كه بايد در برنامه exception handling حتما صورت گيرد به شرح زیر است :
-قابل صرفنظر كردن نيستند و اگر كدي اين موضوع را در نظر نگیرد با يك خطاي زمان اجرا خاتمه پيدا خواهد كرد .

-سبب مشخص شدن خطا در يك نقطه از برنامه شده و ما را به اصلاح آن سوق مي دهد .

بوسيله ي عبارات try...catch مي توان مدیریت خطاها را انجام داد. كدي كه احتمال دارد خطايي در آن رخ دهد درون try قرار گرفته و سپس بوسيله ي يك يا چند قطعه ي catch مي توان آنرا مدیریت كرد. و اگر از اين قطعات خطايابي استفاده نشود برنامه به صورتهای زیر متوقف خواهد شد :

```
class A {static void Main() {catch {}}}  
TEMP.cs(3,5): error CS1003: Syntax error, 'try' expected
```

```
class A {static void Main() {finally {}}}  
TEMP.cs(3,5): error CS1003: Syntax error, 'try' expected
```

```
class A {static void Main() {try {}}}  
TEMP.cs(6,3): error CS1524: Expected catch or finally
```

بهتر است يك مثال ساده را در اين زمينه مرور كنيم :

```
int a, b = 0 ;  
Console.WriteLine( "My program starts " ) ;  
try  
{  
a = 10 / b;  
}
```

```

catch ( Exception e )
{
Console.WriteLine ( e ) ;
}
Console.WriteLine ( "Remaining program" ) ;

```

The output of the program is:
My program starts
System.DivideByZeroException: Attempted to divide by zero.
at ConsoleApplication4.Class1.Main(String[] args) in
d:\dont delete\consoleapplication4\class1.cs:line 51
Remaining program

برنامه شروع به اجرا می کند ، سپس وارد بلوک و یا قطعه ی try می گردد . اگر هیچ خطایی هنگام اجرای دستورات داخل آن رخ ندهد ، برنامه به خط آخر جهش خواهد کرد و کاری به قطعات catch ندارد . اما در اینجا در اولین try عددی بر صفر تقسیم شده است بنابراین کنترل برنامه به بلوک catch منتقل می شود و صرفاً نوع خطای رخ داده شده نوشته و نمایش داده می شود . سپس برنامه به کار عادی خودش ادامه می دهد .

تعدادی از کلاس های exception در سی شارپ که از کلاس System.Exception ارث برده اند به شرح زیر هستند :

- Exception Class - - Cause
- SystemException - A failed run-time check;used as a base class for other.
- AccessException - Failure to access a type member, such as a method or field.
- ArgumentException - An argument to a method was invalid.
- ArgumentNullException - A null argument was passed to a method that doesn't accept it.
- ArgumentOutOfRangeException - Argument value is out of range.
- ArithmeticException - Arithmetic over - or underflow has occurred.
- ArrayTypeMismatchException - Attempt to store the wrong type of object in an array.
- BadImageFormatException - Image is in the wrong format.
- CoreException - Base class for exceptions thrown by the runtime.
- DivideByZeroException - An attempt was made to divide by zero.
- FormatException - The format of an argument is wrong.
- IndexOutOfRangeException - An array index is out of bounds.
- InvalidCastException - An attempt was made to cast to an invalid class.
- InvalidOperationException - A method was called at an invalid time.
- MissingMemberException - An invalid version of a DLL was accessed.
- NotFiniteNumberException - A number is not valid.
- NotSupportedException - Indicates sthat a method is not implemented by a class.
- NullReferenceException - Attempt to use an unassigned reference.
- OutOfMemoryException - Not enough memory to continue execution.
- StackOverflowException - A stack has overflown.

در کد فوق صرفاً عمومی ترین نوع از این کلاس ها که شامل تمامی این موارد می شود مورد استفاده قرار گرفت یعنی :

```

catch ( Exception e )

```

اگر نیازی به خطایابی دقیقتر باشد می توان از کلاس های فوق برای اهداف مورد نظر استفاده نمود .

مثالی دیگر: (در این مثال خطایابی دقیق تر با استفاده از کلاس های فوق و همچنین مفهوم finally نیز گنجانده شده است)

```

int a, b = 0 ;
Console.WriteLine( "My program starts" ) ;
try
{
a = 10 / b;
}
catch ( InvalidOperationException e )
{
Console.WriteLine ( e ) ;
}
catch ( DivideByZeroException e)
{
Console.WriteLine ( e ) ;
}
finally
{
Console.WriteLine ( "finally" ) ;
}
Console.WriteLine ( "Remaining program" ) ;

```

The output here is:
My program starts
System.DivideByZeroException: Attempted to divide by zero.
at ConsoleApplication4.Class1.Main(String[] args) in
d:\dont delete\consoleapplication4\class1.cs:line 51
finally
Remaining program

قسمت موجود در قطعه ي فاينالي همواره صرفنظر از قسمت هاي ديگر اجرا مي شود .

به مثال زير دقت كنيد :

```

int a, b = 0 ;
Console.WriteLine( "My program starts" )
try
{
a = 10 / b;
}
finally
{
Console.WriteLine ( "finally" ) ;
}
Console.WriteLine ( "Remaining program" ) ;

```

Here the output is
My program starts
Exception occurred: System.DivideByZeroException:
Attempted to divide by zero.at ConsoleApplication4.Class1.
Main(String[] args) in d:\dont delete\consoleapplication4
\class1.cs:line 51
finally

قسمت چاپ Remaining program اجرا نشده است .

عبارت throw

این عبارت سبب ایجاد یک خطا در برنامه می شود .

مثال :

```
int a, b = 0 ;
Console.WriteLine( "My program starts" ) ;
try
{
a = 10 / b;
}
catch ( Exception e)
{
throw
}
finally
{
Console.WriteLine ( "finally" ) ;
}
```

در این حالت قسمت فاینالی اجرا شده و برنامه بلافاصله خاتمه پیدا می کند

آموزش C# قسمت نوزدهم

سربارگذاری عملگرها (Operator OverLoading)

به تعریف مجدد راه و روش اجرای عملگرها توسط ما ، سربارگذاری عملگرها گفته می شود. فرض کنید می خواهید عدد 2 را به یک مقدار datetime اضافه کنید. خطای زیر حاصل خواهد شد :

CS0019: Operator '+' cannot be applied to operands of type 'System.DateTime' and 'int'

جالب بود اگر می توانستیم عدد 2 را به datetime اضافه کنیم و نتیجه ی آن تعداد روزهای مشخص بعلاوه ی دو می بود. اینگونه توانایی ها را می توان بوسیله ی operator overloading ایجاد کرد .

تنها عملگرهای زیر را می توان overload کرد :

Unary Operators

+ - ! ~ ++ -- true false

Binary Operators

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

نحوه ی انجام اینکار نیز در حالت کلی به صورت زیر است :

```
return_datatype operator operator_to_be_overloaded (agruments)
{
}
```

به مثال زیر توجه کنید :

```

using System;
class MyDate
{
public DateTime tempDate;
public MyDate(int year,int month,int day)
{
tempDate=new DateTime(year,month,day);
}
public static DateTime operator + (MyDate D,int noOfDays)
{
return D.tempDate.AddDays(noOfDays);
}
public static DateTime operator + (int noOfDays,MyDate D)
{
return D.tempDate.AddDays(noOfDays);
}
}

class Test
{
static void Main()
{
MyDate MD=new MyDate(2001,7,16);
Console.WriteLine(MD + 10 );
}
}

```

output:
2001-07-26

در مثال فوق عملگر + دوبار overload شده است. یکبار توسط آن می توان یک عدد صحیح را به یک تاریخ اضافه کرد و بار دیگر یک یک تاریخ را می توان به عدد صحیح افزود .

موارد زیر را هنگام سربرارگذاری عملگرها به خاطر داشته باشید :

- 1- تنها اپراتورهای ذکر شده را می توان overload کرد. اپراتورهایی مانند new, sizeof, typeid و غیره را نمی توان سربرارگذاری نمود .
- 2- خروجی متدهای بکار گرفته شده در سربرارگذاری عملگرها نمی تواند void باشد .
- 3- حداقل یکی از آرگومانهای بکار گرفته شده در متدی که برای overloading عملگرها بکار می رود باید از نوع کلاس حاوی متد باشد .
- 4- متدهای مربوطه باید به صورت public و static تعریف شوند .
- 5- هنگامی که اپراتور < را سربرارگذاری می کنید باید جهت متناظر آن یعنی > را هم سربرارگذاری نمایید .
- 6- هنگامیکه برای مثال + را overload می کنید خودبخود += نیز overload شده است و نیازی به کدنویسی برای آن نیست .

یکی از موارد جالب بکار گیری سربرارگذاری عملگرها در برنامه نویسی سه بعدی و ساختن کلاسی برای انجام عملیات ماتریسی و برداری می باشد

آموزش C# قسمت بیستم (آخر)

Delegates

Delegates در سی شارپ روشی مطمئن و typesafe را برای بکارگیری مفهوم function pointer ارائه می دهند. یکی از ابتدایی ترین استفاده های function pointers پیاده سازی callback می باشد. اما در ابتدا لازم است تا با اصول اولیه ی کاری آن آشنا شویم .

مثال یک :

یک delegate چگونه تعریف و استفاده می شود؟
Delegate یک شیء است که بیانگر یک تابع می باشد بنابراین می تواند بعنوان آرگومان ورودی یک تابع دیگر و یا عضوی از یک کلاس بکار رود .
در زبان "function-pointer" ، Func1) اشاره گری به Func2) را بعنوان پارامتر دریافت کرده و نهایتاً آنرا فراخوانی می کند .
در زبان "delegate" ، Func1) یک شیء delegate از Func2) را دریافت کرده و سپس آنرا فراخوانی می کند .
در مثال زیر از دو تابع برای شرح این مطلب سود جستته شده است :
Func1) از delegate استفاده می کند .
Func2) یک delegate است .

(شماره گذاری خطوط ، در کد زیر ، صرفاً برای راحت تر شدن توضیحات در مورد آنها است و لزومی به تایپ آنها در برنامه ی اصلی نیست.)

```
01 using System;  
02 delegate void Delg(string sTry);  
03 public class Example1{  
  
// function which uses the delegate object  
04 private static void Func1(Delg d){
```

```

05 d("Passed from Func1");
06 }

// function which is passed as an object
07 private static void Func2(string sToPrint){
08 Console.WriteLine("{0}",sToPrint);
09 }

// Main execution starts here
10 public static void Main(){
11 Delg d = new Delg(Func2);
12 Func1(d);
13 }
14 }

```

LINE 02

يك شئيء delegate را براي Func2 تعريف مي كند .

LINE 04-06

تابعي را تعريف کرده است که آرگومان ورودی آن از نوع Delg است .

LINE 07-09

تابعي را تعريف مي كند که باید به صورت delegate به تابع ديگر فرستاده شود .

LINE 10-14

تابع Main اجراي برنامه را با ايجاد يك شئيء delegate براي Func2 آغاز کرده و سپس تابع Func1 را فراخواني مي كند .

مثال 2

چگونه مي توان از delegates در کارهاي عملي استفاده کرد؟

طرح يك مساله :

شخصي تقاضاي ثبت نام در يك مؤسسه ي آموزشي و همچنين تقاضاي كاريابي در يك شركت را داده است . هر کدام از اين نهادها روشي خاص خود را براي ارزيايي شخص دارند .

راه حل (با روشي شئيء گرا) :

شخص مشخصاتي همچون سن /جنس /ميزان تحصيلات قبلي /تجربيات كاري و مدارك مرتبط دارد . مؤسسه ي آموزشي تعدادي از اين مشخصات را براي ارزيايي شخص استفاده مي كند و اين امر در مورد شركت ياد شده نيز صادق است .

شئيء شركت و شئيء آموزشگاه هر کدام توابع ارزيايي خاص خودشان را پياده سازي مي كنند . شخص ، اينترفيسي واحدی را در اختيار شركت /آموزشگاه براي ارزيايي خود قرار مي دهد .

پياده سازي (با استفاده از سي شارپ) :

ما delegateاي را تعريف مي كنيم که بيانگر اينترفيسي است که به شركت و آموزشگاه اجازه ي چك کردن شخص را مي دهد .

سه کلاس school و company و person را تعريف مي نماييم . کلاس test را براي آزمودن اين موارد ايجاد مي كنيم .

```
01 using System;
```

```
02 using System.Collections;
```

```
03 public delegate bool GetChecker(Person p);
```

```

// Person has his information with him as he
// applies for School and Company
04 public class Person
05 {
06 public string Name;
07 public int Age;
08 public bool Graduate;
09 public int YearsOfExp;
10 public bool Certified;

11 public Person(string name,
int age,
bool graduate,
int yearsOfExp,
bool certified)
12 {
13 Name=name;
14 Age=age;
15 Graduate=graduate;
16 YearsOfExp=yearsOfExp;
17 Certified=certified;
18 }
19 public bool CheckMe(GetChecker checker)
20 {
21 return(checker(this));
22 }
23 }

// A school, the person applied for higher studies
24 public class School
25 {
26 public static bool SchoolCheck(Person p)
27 {
28 return (p.Age>10 && p.Graduate);
29 }
30 }

// A Company, the person wants to work for
31 public class Company
32 {
33 public static bool CompanyCheck(Person p)
34 {
35 return (p.YearsOfExp>5 && p.Certified);
36 }
37 }

// A Test class, displays delegation in action
38 public class Test
39 {
40 public static void Main()
41 {
42 Person p1 = new Person("Jack",20,true,6,false);
43 Console.WriteLine("{0} School Check : {1}",

```

```

p1.Name,
p1.CheckMe(new GetChecker(School.SchoolCheck));
44 Console.WriteLine("{0} Company Check : {1}",
p1.Name,
p1.CheckMe(new GetChecker(Company.CompanyCheck)));
45 }
46 }

```

LINE 03

Delegate مورد نیاز را تعریف می کند .

LINE 04-23

کلاس person را تعریف می کند . این کلاس تابعی پابلیک را ارائه می دهد که آرگومان ورودی آن از نوع GetChecker می باشد .

LINE 24-30

کلاس school را تعریف می کند و سپس تابعی را که delegate است ارائه می دهد .

LINE 31-37

کلاس company را تعریف می کند و سپس تابعی را که delegate است ارائه می دهد .

LINE 38-36

کلاس test را پیاده سازی می نماید . سپس یک شیء شخص ساخته می شود . در ادامه new GetChecker(School.SchoolCheck) و GetChecker(Company.CompanyCheck) (شیء ایی را ایجاد می کند از نوع delegate مورد نیاز و آنرا به تابع CheckMe می فرستد . خروجی نتیجه ی ارزیابی این شخص می باشد .

اگر چک کردن اشخاص بیشتری نیاز باشد به این صورت عمل می شود :

```

Person p1 = new Person("Jack",20,true,6,false);
Person p2 = new Person("Daniel",25,true,10,true);
GetChecker checker1= new GetChecker(School.SchoolCheck);
GetChecker checker2= new GetChecker(School.CompanyCheck);

Console.WriteLine("{0} School Check : {1}",
p1.Name,p1.CheckMe(checker1));
Console.WriteLine("{0} Company Check : {1}",
p1.Name,p1.CheckMe(checker2));
Console.WriteLine("{0} School Check : {1}",
p2.Name,p2.CheckMe(checker1));
Console.WriteLine("{0} Company Check : {1}",
p2.Name,p2.CheckMe(checker2));

```

مثال 3

Delegates در تعامل بین دات نت فریم ورک و سی شارپ چه نقشی دارد؟

طرح یک مساله :

نمایش دادن میزان پیشرفت خواندن یک فایل هنگامی که حجم فایل بسیار زیاد است .

راه حل) با استفاده از سی شارپ (:

در مثال زیر از کلاس FileReader برای خواندن یک فایل حجیم استفاده شده است . هنگامیکه برنامه مشغول خواندن فایل است 'Still reading' . را نمایش می دهد و در پایان '..reading Finished' . را عرضه می کند .

برای اینکار از فضای نام System.IO استفاده شده است. این فضای نام حاوی delegate ای می باشد که برای ما می باشد. بدین ترتیب می توانیم به دات نت فریم ورک بگوییم که ما تابعی را تعریف کرده ایم که او می تواند آنرا فراخوانی کند. سؤال: چه نیازی وجود دارد تا دات نت فریم ورک تابع ما را فراخوانی و اجرا کند؟ با استفاده از تابع ما که دات نت فریم آنرا صدا خواهد زد در طول خواندن فایل به ما گفته می شود که بله! من هنوز مشغول خواندن هستم! به این عملیات Callback نیز گفته می شود. به اینکار پردازش asynchronous نیز می گویند!

```
01 using System;
02 using System.IO;

03 public class FileReader{
04 private Stream sInput;
05 private byte[] arrByte;
06 private AsyncCallback callbackOnFinish;

07 public FileReader(){
08 arrByte=new byte[256];
09 callbackOnFinish = new AsyncCallback(this.readFinished);
10 }

11 public void readFinished(IAsyncResult result){

12 if(sInput.EndRead(result)>0){
13 sInput.BeginRead(arrByte,
14 0,
15 arrByte.Length,
16 callbackOnFinish,
17 null);
18 Console.WriteLine("Still reading..");
19 }
20 else Console.WriteLine("Finished reading..");
21 }

22 public void readFile(){
23 sInput = File.OpenRead(@"C:\big.dat");
24 sInput.BeginRead(arrByte,
25 0,
26 arrByte.Length,
27 callbackOnFinish,
28 null);
29 for(long i=0;i<=1000000000;i++){
30 // just to introduce some delay
31 }
32 }

33 public static void Main(){
34 FileReader asyncTest=new FileReader();
35 asyncTest.readFile();
36 }
37 }
```

02 LINE

فضای نام System.IO را به برنامه ملحق می کند. این فضای نام به صورت خودکار حاوی تعریف delegate

زیر می باشد :

```
IAsyncResult ar public delegate void AsyncCallback
```

LINE 03-10

تعریف کلاس

LINE 06

شیء delegate را تعریف می کند .

LINE 07-10

سازنده ی کلاس را پیاده سازی می کنند. در اینجا ما تصمیم گرفته ایم که بافری حاوی 256 بایت را در هر لحظه بخوانیم .

LINE 09

شیء delegate نمونه سازی شده است .

LINE 18-23

readFile را پیاده سازی می کند .

LINE 12-16

نحوه ی استفاده از شیء IAsyncResult را بیان می کند .

LINE 12

sInput.EndRead(result) تعداد بایتهای خوانده شده را بر می گرداند. این خواندن تاجایی که تعداد بایتهای خوانده شده صفر است ادامه پیدا می کند و در اینجا Finished reading .. اعلام می گردد.

منابع :

1 [Http://www.iranasp.com](http://www.iranasp.com)

2 [Http://www.irandevolvers.com](http://www.irandevolvers.com)

جمع اوری مطالب

سید مهدی همیلی Homeily@Gmail.Com

و با سپاس از

سید محمود همیلی Mahmood.Homeily@Gmail.Com