آشنائی با زبان #C

www.IRVirus.Com

مایکروسافت در مصاف با جاوا، بدنبال ارائه یك زبان کامل بود که سایه جاوا را در میادین برنامه نویسی کم رنگ تر نماید. شاید بهمین دلیل باشد که #C را ایجاد کرد. 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 گستردگی و مقبولیتی به مراتب بیشتر از جاوا را نزد پیاده کنندگان نرم افزار پیدا کند.

با توجه به نقش محوری این زبان، از آن بعنوان مادر زبانهای برنامه نویسی در دات 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 که در ادامه به برخی از آنها اشاره می گردد.

مطرح شدن بعنوان یك استاندارد صنعتی

انجمن تولیدکنندگان کامپیوتر اروپا (ECMA) زبان #C را در سوم اکتبر سال 2001 بعنوان یك استاندارد پذیرفته (ECMA-334) و بدنبال آن تلاش های وسیعی برای کسب گواهی ISO نیز انجام شده است. زبان فوق در ابتدا توسط شرکت مایکروسافت و بعنوان بخشی از دات نت پیاده سازی و بلافاصله پس از آن توسط شرکت های اینتل، هیولیت پاکارد و مایکروسافت مشترکا، جهت استاندارسازی پیشنهاد گردید.

زبان #C بگونه ای طراحی شده است که نه تنها وابستگی به یك Platform خاص را ندارد، بلکه 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 که توسط برخی از شرکت های جانبی ارائه شده است، کدهای #C را به بایت کدهای جاوا کمپایل می کنند. یکی از چنین کامپایلرهائی را می توان در سایت Halcyonsoft.com مشاهده نمود. بنابراین کدهای #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) بخوبی مطابقت می نمایند. CLI قلب و روح دات نت و CLR(Common Language Runtime) است. اولین نسخه از کامپایلر زبان #C که از CLI استفاده می کند، NET Framwork. مایکروسافت است.

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

#C چیست ؟

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

طراحان زبان #C، با درك اهمیت موضوع فوق، این ویژگی را کماکان 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 از Keyword های مربوطه اعلان نمایند(فراخوانی یك توانائی و استفاده از آن).

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

کد مدیریت یافته

با استفاده از نسخه پیاده سازی شده #C توسط مایکروسافت، می توان همواره کد مدیریت یافته ای را تولید کرد. یك برنامه #C پس از کامپایل، بصورت برنامه ای در خواهد آمد که شامل دستورالعمل های تلفیق شده (CIL (Common Intermediate Language است (درست بر خلاف دستورالعمل های مختص یك ماشین خاص). CIL (گاها 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 CLI نظیر CLR مایکروسافت خواهد بود. این برنامه ها بدین دلیل کد مدیریت یافته، نامیده می شوند که CLR مسئولیت تبدیل این دستورالعمل ها به کدهای قابل اجرا برروی ماشین و ارائه اغلب سرویس های اساسی برای کدینگ نظیر : Garbage Collection، مدیریت Heap و عمر مفید یك Object و یا Type Verification را فراهم می کند.

روش یادگیری #C

یادگیری این زبان برای افرادیکه دارای سابقه آشنائی با یکی از زبانهای برنامه نویسی ++C، C و یا جاوا باشند کار مشکلی نخواهد بود، حتی افرادیکه دارای آشنائی اولیه با جاوااسکریپت و یا دیگر زبانهای برنامه نویسی نظیر ویژوال بیسك 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 بیسك 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 یك 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 بیشماری 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 یك شئ.

خلاصه

بدون شك فراگیری و تسلط بر زبان #C بمنزله کسب یك 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 یك 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

قسمت اول :

طي سلسله مقالاتي مي خواهيم با C# بيشتر آشنا شويم. فرض اين مقالات بر اين است كه آشنايي مختصري با زبانهاي برنامه نويسي داريد ، هر چند كار ما تقريبا از صفر شروع مي شود و هدف آن سادگي هر چه بيشتر است.

C# از دو زبان C++ و Java متولد شده است! حاوي بسياري از جنبه هاي C++ مي باشد اما ويژگي هاي شيء گرايي خودش را از جاوا به ارث برده است.

C# اگرچه از C++ گرفته شده است اما يك زبان "خالص" شيء گرا (Object oriented) مي باشد. هر دو زبان ياد شده جزو زبانهاي هيبريد محسوب مي شوند اما طراحان C# اين مورد را به اندازه ي C++ مهم تلقي نكرده اند. يك زبان هيبريد اجازه ي برنامه نويسي با شيوه هاي مختلف را ميسر مي كند. دليل اينكه C++ هيبريد است ، اين است كه قرار بوده تا با زبان C سازگار باشد و همين امر سبب گرديده تا بعضي از جنبه هاي C++ بسيار پيچيده شوند.

زبان سي شارپ فرض اش بر اين است كه شما مي خواهيد تنها برنامه نويسي شيء گرا انجام دهيد و همانند C++ مخلوطي از برنامه نويسي رويه ايي (Procedural) و شيء گرا را نمي خواهيد به پايان برسانيد. بنابراين بايد طرز فكر خودتان را با دنياي شيء گرايي تطبيق دهيد. در ادامه خواهيد ديد كه در سي شارپ هر چيزي شيء است حتي يك برنامه ي سي شارپ.

برنامه ي اول :

Visual studio.net را اجرا كنيد و سپس در صفحه ي ظاهر شده New Project را برگزينيد. حالا از گزينه ي Visual C# projects قسمت Console applications را انتخاب نماييد. نامي دلخواه همانند ex01 را وارد نموده و سپس Ok نماييد. كد زير به صورت خودكار براي شما توليد 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 سري 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 (شبيه 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 با C# 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 شبيه به عبارت زير مي باشد:

```
using System;

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

دكمه ي F5 را فشار دهيد تا برنامه اجرا شود.

تعريف متغيرها در سي شارپ:

سي شارپ عناصري را كه بكار مي گيرد همانند اعداد و كاراكترها ، به صورت نوع ها (Types) طبقه بندي مي كند. اين انواع شامل موارد زير مي شوند :

نوع هاي پايه ايي از پيش تعريف شده مانند اعداد و غيره.
نوع هاي تعريف شده توسط كاربر 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;
```

كه در اينجا متغير a بعنوان يك متغير حاوي اعداد صحيح تعريف شده است.
نكته ي مهمي كه در اينجا حائز اهميت است ، مقدار دهي اوليه ي متغيرها مي باشد. در غير اينصورت كامپايلر سي شارپ برنامه را بايك خطا متوقف مي كند. دليل اين امر هم اين است كه از استفاده از متغيرهاي بدون مقدار در طول برنامه جلوگيري شود تا ميزان خطاهاي در حين اجرا كاهش يابد.

نوع هاي داده اي پايه ي زير در در سي شارپ به صورت پيش فرض مهيا هستند:

object : نوعي است نامحدود كه مي تواند تمام انواع ديگر را نيز شامل شود. مثال :

```
object = null;
```

string : رشته ؛ در اينجا يك رشته توالي كاراكترهاي يونيكد مي باشد. مثال :

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

sbyte : نوع داده ايي صحيح 8 بيتي علامت دار.
byte : نوع داده ايي صحيح 8 بيتي بدون علامت. مثال :

```
sbyte val = 12;
```

short : نوع داده ايي صحيح 16 بيتي علامت دار.
ushort : نوع داده ايي صحيح 16 بيتي بدون علامت. مثال :

```
short val = 12;
```

int : نوع داده ايي صحيح 32 بيتي علامت دار.
unit : نوع داده ايي صحيح 32 بيتي بدون علامت. مثال :

```
int val = 12;
```

long : نوع داده ايي صحيح 64 بيتي علامت دار.
ulong : نوع داده ايي صحيح 64 بيتي بدون علامت. مثال :

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

كلا در اينجا u به معناي unsigned است.

float : نوع اعشاري با single precision .
double : نوع اعشاري با double precision . مثال :

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

bool : نوع داده ايي Boolean كه مي تواند true و يا false باشد. مثال :

```
Bool val = true;
```

char : كاراكتر، دراينجا char يك كاراكتر يونيكد است.

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

به نحوه ي تعريف كاراكتر ها و همچنين رشته ها در سي شارپ دقت كنيد.

decimal : نوع داده ايي دسيمال با 28 رقم معني دار.

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

يك نكته :

- 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 گويند.
- 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 ايي 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!

}
}
}
```

نكاتي در مورد كد فوق:

- بك اسلش ان ، در زبانهاي مشتق شده از سي به معناي new line مي باشد.
- در كد فوق نحوه ي تعريف چند متغير در يك خط و حالتهاي مقدار دهي مختلف را ملاحظه مي كنيد.
- از متد ReadLine براي نگه داشتن خروجي و مشاهده ي آن در اينجا استفاده كرديم.
- عادت كنيد به صورت دندانه دار كد بنويسيد. اينكار خوانايي كد را صد برابر مي كند. در اينجا كدهاي داخل متد main ، كاملا چند دندانه از آكولادهاي باز و بسته كردن آن جلو تر هستند.
- در كد بالا در متد WriteLine اعداد و رشته ها با هم جمع شده اند! اين مورد بدليل وجود overload هاي زياد اين تابع و ... ميسر گشته است. اصلا به آن دل نبنديد! چون در آينده كامپايلر سي شارپ اگر چنين اعمالي را در جاهاي ديگري مرتكب شويد به شدت با شما برخورد خواهد كرد!! براي جمع كردن اعداد با رشته ها حتما بايد عدد به رشته تبديل گردد و بعد ... . در اين مورد در مقالات بعدي بحث خواهد گرديد.

قسمت دوم :

مقدمه :
در اين قسمت مي خواهيم با يك سري از اصول اوليه ي شيء گرايي در سي شارپ كمي آشنا شويم. لازم به ذكر است ، بسياري از مواردي كه در اين قسمت مطرح مي شوند فقط براي آشنايي شما است و در آينده بيشتر بحث و مرور خواهند شد.

آشنايي با فضاهاي نام (NameSpaces) :
فضاهاي نام روشي براي مديريت كد نويسي هستند. براي مثال آنها ايجاد شده اند تا تداخلي بين نام هاي توابع در برنامه شما رخ ندهد. اين مساله در پروژه هاي بزرگ خود را نشان مي دهد و ممكن است دو آيتم در يك پروژه نام هاي يكساني را پيدا كنند. بدين وسيله اين شانس تصادم و تداخل كاهش پيدا مي كند. براي ايجاد يك فضاي نام به صورت زير عمل مي شود:

```
namespace anyName
{
........

Class anyClassName
{
..........
}

.......

}
```

يكي از فضاهاي نام پايه اي در دات نت فريم ورك ، فضاي نام System مي باشد. براي استفاده از آن مي توان از كد زير كمك گرفت :

```
using System;
```

تمام فضاهاي نام به صورت پيش فرض public مي باشند و در خارج از كد شما قابل دسترسي هستند. روش استفاده از آنها به صورت زير است:

```
ProjectName.NameSpace.ClassName.MemberName
```

نكته : اگر دقت كرده باشيد هنگامي كه كرسر ماوس را روي هر آيتمي در منوي autocomplete نگه مي داريد و يا آنرا انتخاب مي كنيد يك راهنماي كوچك نمايش داده مي شود كه در حقيقت كامنت مربوط به آن تابع مي باشد. روش نوشتن چنين كامنت حرفه اي كه در منوهاي ويژوال استوديو ظاهر شود به صورت زير است كه بهتر است (!) قبل از هر تابع يا خاصيت يا كلاس و .... نوشته شود

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

كلاس ها :
چون سي شارپ تمام سر و كارش با كلاس ها است بنابراين بايد در مورد نحوه ي تعريف و استفاده از آنها تسلط كافي داشته باشيم.
يك پروژه ي جديد console در VS.NET باز كنيد و نام آنرا در ابتدا ex03 وارد نماييد.
بعد از باز شدن پروژه ، از منوي Project گزينه ي Add class را انتخاب كنيد تا كلاسي جديد به نام clsDate.cs را اضافه نماييم. ساختار فايل ايجاد شده توسط VS.NET به صورت زير است :

```
using System;

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

```
}
}
```

تابع يا متد clsDate كه در اينجا به صورت پيش فرض ايجاد شده است اصطلاحا سازنده (constructor) نام دارد. اين تابع هر بار كه يك شيء جديد از كلاس مي سازيم به صورت خودكار اجرا مي شود.

از اين كلاس مي خواهيم براي نمايش تاريخ/ ساعت و غيره استفاده كنيم.

براي مثال مي خواهيم تاريخ جاري سيستم را به صورت يك خاصيت از اين كلاس دريافت كنيم. براي اين منظور كد زير را به برنامه اضافه مي نماييم:

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

توضيح كد فوق :
خاصيتي را كه مي خواهيم از برنامه دريافت كنيم با كلمه ي كليدي get معرفي مي نماييم. هر چيزي كه اين قسمت برگرداند خروجي currentSystemDate خواهد بود. اين دستور زبان كه در بالا معرفي شد استاندارد است و در همه جا به يك صورت تعريف و بكار برده مي شود. پس شكل آنرا به خاطر بسپاريد.
از كلمه ي كليدي return براي برگرداندن يك خروجي از خاصيت و يا تابع استفاده مي شود.

براي استفاده از اين خاصيت جديد ، در فايل Class1.cs كه متد main برنامه ي ما در آنجا قرار دارد به صورت زير عمل مي كنيم :

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

توضيح كد فوق :
براي استفاده از يك كلاس بايد يك متغير از آن را تعريف كنيم. در هر زباني يك سري نوع هاي استاندارد مانند int و string و غيره وجود دارند. كلاس هم در حقيقت يك نوع داده ي بسيار بسيار قدرتمند به شمار مي آيد. براي تعريف يك متغير از نوع جديد روش كار مانند سابق است. براي مثال زماني كه يك متغير عدد صحيح را تعريف مي كنيد به صورت زير عمل مي شود :

```
int i=0;
```

راي تعريف يك متغير از نوع داده اي كه خودمان تعريف كرده ايم نيز بايد به همين صورت عمل شود.

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

از كلمه ي كليدي new اينجا به صورت استاندارد براي مقدار دهي اوليه به اين متغير جديد استفاده مي نماييم.

سپس به روش دستيابي به اين خاصيتي كه به كلاس اضافه كرده ايم مي رسيم.

```
m_var.currentSystemDate
```

كلا چه يك خاصيت و يا يك متد را به كلاس اضافه نماييم براي دستيابي به آن از عملگر نقطه پس از ذكر نام متغير تعريف شده از نوع كلاس خود ، استفاده مي نماييم. براي استفاده از خاصيت ها نيازي به آوردن () بعد از ذكر نام خاصيت نمي باشد.

عموما از خاصيت ها براي برگرداندن و يا تنظيم يك مقدار ساده استفاده مي شود و در آنها عمليات پيچيده اي مد نظر نمي باشد.

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

شما به ويژگي هاي يك شيء با استفاده از خواص آن مي توانيد دسترسي پيدا كنيد. يك property عضوي است كه امكان دسترسي به ويژگي شيء يا كلاس را فراهم مي كند. براي مثال طول يك رشته (string) ، سايز يك فونت ، عنوان يك فرم و نام يك مصرف كننده ، خاصيت هستند .
بسياري از اشياء ذاتي دات نت فريم ورك ، خواص مفيد زيادي را به همراه دارند. براي مثال شيء DateTime را در نظر بگيريد. با استفاده از خاصيت Today آن مي توان تاريخ جاري سيستم را بدست آورد. براي استفاده از يك خاصيت لازم است تا كلاس تعريف كننده شيء در برنامه مهيا باشد. منظور همان استفاده از فضاي نام مربوطه مي باشد. پس از وارد كردن فضاي نام كلاس مورد نظر مي توانيد از شيء و خواص آن استفاده كنيد. دو راه وجود دارد يا به صورت كامل تمام موارد بايد ذكر شوند مانند System.DateTime.Now; و يا با وارد كردن فضاي نام System كوتاه سازي صورت مي گيرد.

براي استفاده از هر متد و يا شيء ايي در سي شارپ بايد اين شيء قابل دسترسي باشد. براي مثال شيء Console كه از آن براي چاپ كردن خروجي بر روي صفحه ي نمايش استفاده مي كنيم در فضاي نام System واقع شده است. يا بايد در ابتداي برنامه ذكر كرد using System ; و سپس خيلي راحت از اين شيء استفاده كرد و يا مي توان اينكار را انجام نداد و نوشت : System.Console و الي آخر. با ذكر فضاي نام در ابتدا با استفاده از using مي توان خلاصه نويسي كرد.

نتيجه ي نهايي مثال اين فصل :

محتويات فايل 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!
}
}
}
```

محتويات فايل clsDate.cs كه به برنامه اضافه كرديم:

```
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() ;
}
}


}
}
```

قسمت سوم :

ساختارهاي تصميم گيري :

در بسياري از موارد هنگام برنامه نويسي لازم است تا از عبارات شرطي استفاده كنيم. براي انجام اينكار دو روش عمده وجود دارد. استفاده از if و يا switch . از if بيشتر براي مقايسه هايي تكي و كوچك استفاده مي شود و حاصل مقايسه ي آن يا true است و يا

false . از عبارت switch هنگامي استفاده مي شود كه مقايسه هاي متعددي بايد در مورد يك مقدار صورت گيرد.
هر دو عبارت if و switch توسط عبارتهايي Boolean كنترل مي شوند (true و يا false) . در هنگام استفاده از if اگر عبارت Boolean حاصل اش true باشد اولين قسمت شرط اجرا مي شود و سپس برنامه از انتهاي if ادامه پيدا مي كند. اگر حاصل عبارت Boolean مساوي false باشد كنترل برنامه به قسمت else منتقل مي شود.

مثال :

يك پروژه ي جديد console باز كنيد و نام آنرا ex04 بگذاريد. سپس كد زير را در آن وارد و جرا كنيد :

```
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!");

}
}
}
```

نكاتي در مورد كد فوق :

1- سي شارپ به كوچكي و بزرگي حروف حساس است . براي مثال cUserInput با cUserinput فرق مي كند.
2- حتما بايد بعد از if پرانتزها ذكر گردد.
3- حتما بايد داخل if يك عبارت Boolean ذكر شود مانند 5<if(x) .
4- در سي شارپ مقايسه ي تساوي دو عبارت با == و انتساب با = انجام مي شود. (موارد 1 و 4 مواردي هستند كه اغلب تازه كاران با آن مشكل دارند!) براي مثال if(i==3) صحيح است اما if(i=3) در سي شارپ معنايي ندارد.
5- اگر بعد از if يك خط كد قرار گيرد نيازي به آوردن آكولاد ها نيست. هنگامي نياز به آكولادها مي باشد كه بيش از يك خط بايد بعد از if قرار گيرد.
6- در سي شارپ همانند اسلاف خودش براي تبديل نوع هاي داده ايي مي توان به صورت زير نيز عمل كرد : (char) Console.Read() ; يعني دريافتي Read به char تبديل مي شود .در اين مورد باز هم صحبت خواهد شد.
7- همانطور كه ذكر شد در سي شارپ همه چيز شيء است حتي نوع هاي پايه ايي مانند char . با استفاده از متد IsDigit آن مي توان چك كرد كه آيا ورودي آن عدد است يا خير؟ (در مورد متدها صحبت خواهد شد)

استفاده از switch :

بهتر است اين مورد را با يك مثال دنبال كنيم.
پروژه ي سي شارپ جديدي به نام ex05 در حالت console در VS.NET باز كنيد. دراينجا مي خواهيم يك كلاس جديد تعريف كرده و توسط خاصيتي كه در آن ايجاد مي كنيم متوجه شويم روز جاري مطابق سيستم چه روزي است .
يك كلاس جديد از منوي پروژه ،با استفاده از گزينه ي class Add به برنامه اضافه كنيد و نام آنرا در ابتدا clsDate بگذاريد.

```
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 = "چهار شنبه";
break;

case "Thursday":
res = "پنج شنبه";
break;

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

return res ;
}

}


}
}
```

هنگام ذخيره كردن اين كد ويژوال استوديو به شما اخطار مي دهد كه كد داراي حروف يونيكد است. از منوي فايل گزينه ي advanced save options را انتخاب كنيد. در اينجا مي توان نوع ذخيره سازي را يونيكد انتخاب كرد.

براي استفاده از كلاس فوق مانند مطالبي كه در قسمت قبل گفته شد عمل مي كنيم :

```
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 يونيكد را پشتيباني نمي كند ولي اصل برنامه براي ما مهم است و در آينده بيشتر از آن استفاده خواهيم كرد.

همانطور كه ملاحظه كرديد اگر از switch استفاده نمي شد بايد از 7 عدد if استفاده مي گرديد كه اصلا ظاهر حرفه اي و شكيلي نداشت!
با استفاده از عبارت زير كار مقايسه شروع مي شود. روز سيستم در يافت شده و وارد بدنه ي switch مي گردد. سپس توسط case ها چك مي شود تا تساوي آن با عبارت بعد از case به اثبات برسد.

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

اگر هر كدام از عبارات بعد از case صحيح بودند كار پس از آن كه در اينجا انتساب است انجام شده و سپس توسط break كنترل برنامه از switch خارج مي شود و ادامه ي كار دنبال مي گردد.
اگر هيچكدام از case ها صحيح نبودند مي توان از گزينه ي default هم در صورت نياز استفاده كرد. اين حالت در يك چنين مواقعي اجرا مي گردد

قسمت چهارم :
آرايه ها در سي شارپ :

هنگامي آرايه ها ايجاد مي شوند كه بخواهيم با مجموعه اي از اطلاعات همجنس كار كنيم. براي نمونه از يك آرايه براي ذخيره تعدادي كاراكتر مي خواهيم استفاده نماييم. آرايه ها هم يك نوع متغير هستند پس بايد تعريف و مقدار دهي اوليه شوند ، نوع و تعداد اعضاي آنها نيز بايد معين گردد.
فرض كنيد 10 داده ي هم جنس داريم (براي مثال رشته (string)) و مي خواهيم آنها را ذخيره كنيم. يا مي توان 10 متغير مختلف را تعريف كرد و سپس تك تك آنها را مقدار دهي نمود و يا يك آرايه تعريف نمود و سپس در خانه هاي مختلف آن اين ده عضو را چيد. اين مطلب زماني حائز اهميت مي شود كه داده هاي همجنس و به نوعي مرتبط ما تعداد زيادي داشته باشند.

براي تعريف آرايه چندين راه مختلف وجود دارد :
براي تعريف آرايه ابتدا نوع آنرا مشخص مي كنيد سپس [] را بايد جلوي تعريف نوع بگذاريد اين دستور زبان است و چون چرا نداردا! در زبان سي كمي متفاوت بود. اين كروشه ها بعد از نام متغير مي آمدند. و سپس در اينجا نام يك متغير را كه بعدا به آن ارجا مي دهيم خواهيد گذاشت. براي مثال

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

حد پايين آرايه صفر بوده براي مثال اگر آرايه chrData[] ده عضو داشته باشد، اولين عضو آن chrData[0] و آخرين عضو آن chrData[9] است.
مطلب ديگري كه در مورد آرايه ها خيلي مهم است اندازه ي آن است. يعني يك آرايه حاوي چند خانه ي خالي است كه ما اجازه داريم آنرا پر كنيم. 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- تعريف آرايه اي از رشته ها و مقدار دهي اوليه آن.

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

2- تعريف و مقدار دهي اوليه

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

كه آرايه اي از نوع رشته اي به طول 2 عضو با مقدار دهي اوليه ايجاد شده است. در اين حالت نيازي به تعيين طول آن نمي باشد.

3- روشي ديگر براي مقدار دهي اوليه

```
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 سمي 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 دهي 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

قسمت پنجم :
حلقه ها در سي شارپ :

مقدمه :

اگر نياز باشد تا قطعه اي از كد بيش از يكبار اجرا شود نياز به استفاده از حلقه ها مي باشد. براي مثال فرض كنيد آرايه اي به طول 1000 تعريف كرده ايد. اكنون مي خواهيد آنرا با هزار عدد متوالي پركنيد. بديهي است كه روش زير كارآمد نيست! :

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

نوشتن اين خطوط متوالي احتمالا با كپي و پيست و اصلاح آن حداقل نيم ساعت طول مي كشد! بنابراين نياز به وسيله اي حس مي شود كه بتوان بوسيله ي آن امثال اينگونه كارها را انجام داد.

تعريف حلقه ها و استفاده از آنها :

براي تعريف حلقه ها ابزارهاي متعددي مانند for , do , while, foreach وجود دارند. استفاده و انتخاب آنها بستگي به سليقه ي شما و منطق برنامه دارد. در هر حال يك مساله بديهي است كه همواره بيش از يك راه حل براي يك مساله وجود خواهد داشت.

استفاده از حلقه ي for :

عموما كدنويسي را با كد نويسي مي توان آموخت! بنابراين در مورد انواع حلقه ها مثالهايي ارائه خواهد گرديد.

يك برنامه ي سي شارپ جديد console را در VS.NET باز كنيد و نام آنرا درابتدا ex07 انتخاب نماييد. سپس كد زير را درون آن بنويسيد :

```
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();

}
}
}
```

توضيحاتي در مورد كد فوق :
1- براي تعريف حلقه ي for همانطور كه مي بينيد بايد تعداد بار اجراي حلقه (اينجا از 0 تا 999 است) و همچنين نحوه ي رسيدن از 0 به 1000 را مشخص كرد (در اينجا i++ است يعني هر بار يك واحد به شمارشگر حلقه اضافه مي شود.)
2- در زبان سي i++ يعني i=i+1 و i-- يعني i=i-1 و كلا i-=n يعني i=i-n و به همين ترتيب. براي مثال i*=n يعني i=i*n و i+=n يعني i=i+n و ...
3- اگر پس از حلقه ي for يك خط كد داشته باشيم نيازي به آكولاد نيست (مانند قسمت اول كد). ولي اگر تعداد خطوط مربوط به بدنه ي for زياد بود بايد حتما از آكولاد استفاده شود (مانند قسمت دوم كد). (اين قاعده اي كلي است در زبانهاي مشتق شده از زبان سي در مورد هر چيزي!)
4- فرض كنيد در قسمت اول كد بالا بجاي 1000 مي نوشتيد 1001 . سريعا با يك خطاي زمان اجرا مواجه مي شديد. زيرا مي خواستيد به عضوي از آرايه دسترسي پيدا كنيد كه تعريف نشده است. راه مدرن چك كردن اين مسائل استفاده از خاصيت Length آرايه است كه در قسمت دوم كد در عمل مشاهده مي نماييد. هميشه از اين روش استفاده كنيد.
5- حلقه ي اول يعني اينكه كار پر كردن آرايه intData را از صفر تا 999 يكي يكي (i++) انجام بده.

استفاده از حلقه ي while :

يك برنامه ي سي شارپ جديد console را در VS.NET باز كنيد و نام آنرا درابتدا ex08 انتخاب نماييد. سپس كد زير را درون آن بنويسيد :

```
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();
}
}
}
```

توضيحاتي در مورد كد فوق :
1- حلقه ي while در بالا كار انجام حلقه را تا هنگامي انجام مي دهد كه شرط ذكر شده در ابتداي آن صادق و برقرار باشد. يعني در حلقه ي فوق تا وقتي 6>n است اين حلقه ادامه خواهد يافت.
2- حلقه ي while صفر يا بيشتر بار ممكن است اجرا شود.
3- در كد فوق از {0} استفاده گرديده است. متد WriteLine به شما اين اجازه را مي دهد كه n تا آرگومان براي آن تعريف كنيد و مقادير هر كدام را كه خواستيد در كد نمايش دهيد از {x} استفاده كنيد. 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 كنيد و نام آنرا درابتدا ex09 انتخاب نماييد. سپس كد 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 كد فوق :
1- اين حلقه به حلقه ي do...while معروف است و هر دو جزء آن بايد ذكر گردد.

2- اين حلقه تا زماني كه شرط ذكر شده در قمست while صحيح است ادامه مي يابد.
3- اين حلقه در ابتداي كار بدون توجه به قسمت while حداقل يكبار 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 كنيد و نام آنرا درابتدا ex10 انتخاب نماييد. سپس كد 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();

}
}
}
```

توضيحاتي در مورد كد فوق :
1- از foreach براي حركت در بين اعضاي يك آرايه (مانند مثال بالا) و يا مجموعه ايي از اشياء استفاده مي شود (روشي شكيل ، مدرن و مطمئن! و تقريبا به ارث رسيده از ويژوال بيسيك!!).
2- در زبانهاي مشتق شده از C ، عملگر % ، باقيمانده را محاسبه مي كند.
3- در كد فوق با استفاده از حلقه ي foreach تك تك اعضاي آرايه در مورد زوج و يا فرد بودند مورد بررسي قرار گرفته اند و تعداد اعضاي زوج و فرد در آخر نمايش داده مي شود

قسمت ششم :

دو مورد تكميلي در مورد حلقه ها در سي شارپ :

1- هر جايي خواستيد به هر دليلي حلقه را پايان دهيد مي توانيد از دستور break; استفاده كنيد. در اين حالت به صورت آني حلقه خاتمه يافته و كدهاي ادامه ي برنامه پس از حلقه اجرا مي شوند.
2- نحوه ي استفاده از دستور continue : فرض كنيد حلقه ي شما در راند 15 خودش است! حالا در اين راند شما مي خواهيد يك سري از دستورات درون حلقه اجرا نشوند و حلقه به راند بعدي منتقل شده و كارش را ادامه دهد. اينجا است كه از دستور continue استفاده مي شود. بهتر است به يك مثال ساده در اين زمينه توجه كنيم.

مثال : يك برنامه ي سي شارپ جديد console را در VS.NET باز كنيد و نام آنرا درابتدا ex11 انتخاب نماييد. سپس كد زير را درون آن بنويسيد :

```
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 تكميلي مربوط به رد و بدل كردن مقادير به/از كلاس ها :

در قسمت بعدي مي خواهيم خاصيتي را تعريف كنيم كه يك مقدار را از كاربر مي گيرد و در برنامه مي توان توسط قسمت هاي ديگر از آن استفاده كرد.

ابتدا يك متغير عمومي بايد در سطح كلاس تعريف كرد تا مقدار دريافت شده توسط set را در خود نگاه داري كند (در مورد scope متغيرها (متغيرهاي عمومي و محلي و امثال اينها) در هنگام معرفي توابع بيشتر بحث خواهد شد) . سپس از طريق كلمه ي كليدي value مقدار دريافت شده به متغير انتساب مي يابد و چون در سطح كلاس عمومي است در تمام كلاس قابل درسترسي است.

مثال : يك برنامه ي سي شارپ جديد console را در VS.NET باز كنيد و نام آنرا درابتدا ex12 انتخاب نماييد. سپس از منوي پروژه يك كلاس جديد به آن اضافه نماييد (به نام clsDate) و كد 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 كنيم:

```
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();
}
}
}
```

توضيحاتي در مورد كد فوق:

1- نحوه ي تعريف متغير از يك كلاس جزو اساسي ترين قسمت هاي كار با يك كلاس محسوب مي شود كه در قسمت هاي پيشين نيز معرفي گرديد.
2- هنگامي كه از if استفاده مي كنيم لزومي ندارد حتما بنويسيم m_var.IsLeapYear==true . همين كه اين خاصيت ذكر مي شود در وهله ي اول true بودن آن چك خواهد شد.
3- نحوه ي مقدار دهي به يك خاصيت را هم در كد فوق ملاحظه مي نماييد. در هنگام استفاده از خاصيت ها نيازي به آوردن پرانتزها () در مقابل نام آنها وجود ندارد.
4- براي مرور ، نحوه ي معرفي خاصيت ها با get نيز بيان گرديد. با استفاده از set و get مي توان به كلاس ها ، مقادير متغيرها را پاس كرد و يا مقداري را دريافت نمود

قسمت هفتم :

تعريف متدها در سي شارپ

در اين قسمت به يكي از مهمترين مباحث برنامه نويسي سي شارپ مي رسيم.
متدها در سي شارپ و يا همان توابع در زبان C ، اعضاي يك شيء يا كلاس هستند و مجموعه اي از يك سري از كارها را انجام مي دهند. فرض كنيد در برنامه ي شما ، قسمتي بايد يك عمليات رياضي خاص را انجام دهد و اين قسمت از كد كه شامل چندين خط نيز مي گردد بايد بارها و بارها در برنامه صدا زده شود. براي نظم بخشيدن به برنامه ، آنها را مي توان به صورت توابع بسته بندي كرد و بجاي نوشتن چندين خط تكراري، فقط نام اين بسته (تابع) و پارامترهاي آن را فراخواني نمود.

در سي شارپ يك تابع به صورت زير تعريف مي شود :

(نوع و اسامي پارامترها) نام تابع نوع خروجي تابع سطح دسترسي به تابع
{
بدنه ي تابع
}

براي تعريف يك متد يا تابع ابتدا سطح دسترسي به آن مانند public و private سپس نوع خروجي تابع مانند void (هيچي) ذكر مي

گردد كه داخل اين پرانتزها مي توان ورودي هاي تابع يا بقولي آرگومان هاي ورودي را معرفي كرد. سپس تابع بايد با { شروع و با يك } خاتمه يابد.

براي مثال :

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

هر تابعي مي تواند صفر تا تعداد بيشماري آرگومان ورودي و صفر تا تعداد بيشماري خروجي داشته باشد. بوسيله يك تابع مي توان پيچيدگي كار را مخفي كرد و صرفا با صدا زدن نام آن ، يك سري از عمليات را انجام داد. گاهي از اوقات لازم مي شود دو يا چند تابع با يك نام داشته باشيم بطوريكه پارامترهاي ورودي يا مقادير خروجي و يا نوع آرگومان هاي ورودي آنها با هم متفاوت باشد به اين كار overloading مي گويند.
بسياري از كلاس هاي دات نت فريم ورك متدها و يا توابع مفيد حاضر و آماده اي را دارند. براي مثال كلاس DateTime ، متدي به نام ToLongDatastring دارد كه تاريخ را به صورت يك رشته طولاني بر مي گرداند.

توابع void :

توابعي كه با نوع void معرفي مي شوند هيچ خروجي ندارند و در زبان ويژوال بيسيك به آنها sub و در دلفي به آنها procedure مي گويند.

بازگرداندن يك مقدار از يك تابع :

پس از اينكه عمليات يك مجموعه از كدها درون تابع به پايان رسيد با استفاده از كلمه ي return مي توان خروجي تابع را معرفي كرد. لازم به ذكر است ، هرجايي اين كلمه ي return ذكر شود كار تابع خاتمه مي يابد.

بهتر است موارد فوق را با چند مثال مرور كنيم :

مثال : يك برنامه ي سي شارپ جديد console را در VS.NET باز كنيد و نام آنرا درابتدا 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;
}
}
}
```

توضيحاتي در مورد كد فوق :

1- از شيء Math در سي شارپ مي توان براي انجام يك سري عمليات رياضي ابتدايي 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 استاتيك را فراخواني كرد. فعلا اين نكته را بخاطر را داشته باشيد تا در مقالات بعدي بيشتر راجع به آن صحبت شود.
3- بد نيست تابع تعريف شده را كمي 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 :

يكي از زيبايي هاي برنامه نويسي شيء گرا نظم و ترتيب و بسته بندي كارها مي باشد كه اصطلاحا در اينجا به آن encapsulation مي گويند. يعني ما يك سري از توابع و خواص را درون كپسولي به نام كلاس قرار مي دهيم تا به سادگي بارها و بارها از آن استفاده نماييم.
براي اينكار به سادگي يك توابع را به صورت معمول درون كلاس تعريف مي نماييم و سپس همانند خواص كه در مورد آنها صحبت شد ، از توابع مي توان استفاده كرد با اين تفاوت كه هنگام كار با توابع حتي اگر آنها هيچ آرگومان و يا پارامتر ورودي هم نداشته باشند ذكر پرانتزها الزامي است.

مثالي ديگر در اين زمينه :

مثال : يك برنامه ي سي شارپ جديد console را در VS.NET باز كنيد و نام آنرا درابتدا ex14 انتخاب نماييد. سپس از منوي پروژه يك كلاس 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 كرد :

```
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 كد فوق :
1- تابع intCalc ما دو عدد صحيح مثبت را مي گيرد و سپس جذر كوچكترين دو عدد ورودي را محاسبه مي كند.
2- براي تبديل نوع هاي عددي مختلف به هم مي توان از شيء Convert استفاده كرد.
3- بدون استفاده از شيء Convert يكبار برنامه را اجرا كنيد و دليل خطاي بوجود آمده را بيان نماييد

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 بگوييم ، 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 درابتدا 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 تغييري 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 :

اين كلمه ي كليدي نيز دقيقا همانند out عمل مي كند و نحوه ي تعريف و استفاده از آن نيز مشابه است با اين تفاوت كه آرگوماني كه به اين نوع توابع فرستاده مي شود بايد مقدار دهي اوليه شده باشد.

مثال : يك برنامه ي سي شارپ جديد console را در VS.NET باز كنيد و نام آنرا درابتدا ex16 انتخاب نماييد. سپس كد زير را درون آن بنويسيد :

```
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();
}
}
}
```

توضيحاتي در مورد كد فوق :

1- همانطور كه ملاحظه مي كنيد در اينجا هنگام استفاده از تابع FillArray بايد آرگوماني را كه مي خواهيم به آن پاس كنيم مقدار دهي اوليه كنيم.

2- پس مي توان نتيجه گرفت آرگومانهايي كه با out تعريف مي شوند به صورت خالص خروجي هستند و نيازي به مقدار دهي اوليه هنگام استفاده از آنها وجود ندارد. از ref هنگامي استفاده مي كنيم كه بخواهيم روي متغير موجود و مقدار دهي شده ي خارج از تابع ، درون تابع عملياتي صورت گيرد و سپس همان متغير دستكاري شده ، عودت داده شود.

تعريف تابعي با تعداد آرگومانهاي نامعلوم :

گاهي از اوقات نياز است تا تابعي تعريف كنيم كه تعداد آرگومانهاي آن متغير باشند .براي اين منظور از كلمه ي كليدي params استفاده مي شود.

دونكته در اينجا حائز اهميت است:

1- در هر تابعي تنها مي توان يكبار از params استفاده كرد.

2- پس از بكار بردن params ديگر نمي توان هيچ آرگوماني را تعريف كرد.

يكي از مثالهايي كه در اين زمينه مي توان ارائه داد استفاده از آرايه ها به عنوان آرگومان ورودي است. در اين حالت يا مي توان يك آرايه را به صورت كامل به تابع معرفي كرد و يا تنها نام آنرا به تابع پاس كرد. مثال زير را ملاحظه كنيد :

مثال : يك برنامه ي سي شارپ جديد console را در VS.NET باز كنيد و نام آنرا درابتدا ex17 انتخاب نماييد. سپس كد زير را درون آن بنويسيد :

```
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();

}
}
}
```

توضيحاتي در مورد كد فوق :

1- در تابع main به دو صورت از تابع UseParams ما استفاده شده است. يا اينكه خيلي ساده هر تعداد آرگومان را مي توان به تابع فرستاد و يا اينكه در ادامه آرايه ايي رسما تعريف و سپس به تابع فرستاده شود.
2- نحوه ي تعريف و استفاده از آرايه ها به صورت آرگومان ورودي را نيز مي توان در مثال فوق آموخت

قسمت نهم :
مبحث overloading :

گاهي از اوقات لازم است تا نگارش هاي مختلفي از يك تابع داشته باشيم. براي مثال تعريف سه تابع با يك نام اما با آرگومانهاي مختلف. به اين نوع توابع و يا متدها اصطلاحا Overloaded Methods مي گويند . (فكر كنم آنرا سربارگذاري توابع ترجمه كرده اند!)
براي مثال :

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

مثال : يك برنامه ي سي شارپ جديد console را در VS.NET باز كنيد و نام آنرا درابتدا ex18 انتخاب نماييد. سپس كد زير را درون آن بنويسيد :

```
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 );
}


}
}
```

توضيحاتي در مورد كد فوق :

1- نحوه ي تعريف دو تابع با يك نام را ملاحظه مي نماييد. اينكار در زبان سي ممنوع است!
2- كامپايلر به صورت هوشمند بر اساس نوع و تعداد آرگومانهاي ورودي ، ورژن مناسب را انتخاب و اجرا مي كند.

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

مباحث تكميلي 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 تشكليل شده از رديف ها و ستون ها در نظر گرفت و الي آخر...
سي شارپ دو نوع آرايه ي چند بعدي را پشتيباني مي كند : rectangular and jagged
در يك آرايه ي rectangular هر رديف ، طولش با رديف بعدي يكي است. آرايه ي jagged در حقيقت آرايه ايي از آرايه ها است ، بنابراين هر كدام 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 كنيد و نام آنرا درابتدا ex19 انتخاب نماييد. سپس كد 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
{
rectangularArray[i,j] = i+j;
}
}
// report the contents of the array
for (int i = 0;i < rows;i++)
{
for (int j = 0;j
{
Console.WriteLine("rectangularArray[{0},{1}] = {2}",
i,j,rectangularArray[i,j]);
}
}

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

توضيحاتي در مورد كد فوق :
1- نحوه ي تعريف ، مقدار دهي اوليه و استفاده از آرايه هاي دو بعدي را در مثال فوق ملاحظه مي نماييد.
2- در يك آرايه ي دوبعدي محل قرار گيري رديف ها و ستون ها براي مثال به صورت زير است :

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

استفاده از آرايه هاي چند بعدي :

مثال : يك برنامه ي سي شارپ جديد console را در VS.NET باز كنيد و نام آنرا درابتدا ex20 انتخاب نماييد. سپس كد زير را درون آن بنويسيد :

```
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]);
}
}
}
}
}
```

توضيحاتي در مورد كد فوق :
1- در حقيقت مثال فوق تعريف آرايه ايي از آرايه ها بود.
2- چون مقدار دهي اوليه به صورت واضحي انجام شده نيازي به ذكر ابعاد آرايه به صورت صحيح وجود نداشت

قسمت دهم :

Jagged arrays آرايه اي از آرايه ها است و همانطور كه ذكر شد لزومي ندارد كه هر رديف آن با رديف بعدي هم طول باشد . هنگام تعريف اين نوع آرايه شما تعداد رديف ها را مشخص مي نماييد. هر رديف يك آرايه را نگهداري مي كند. در اينجا هر آرايه بايد تعريف شود. روش تعريف Jagged array به صورت زير است

```
type [] []...
```

در اينجا تعداد براكت ها بيانگر ابعاد آرايه مي باشد. براي مثال آرايه ي زير دو بعدي است :

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

و براي مثال براي دسترسي به پنجمين عنصر آرايه ي سوم به صورت زير عمل مي شود :

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

مثال : يك برنامه ي سي شارپ جديد console را در VS.NET باز كنيد و نام آنرا درابتدا ex21 انتخاب نماييد. سپس كد زير را درون آن بنويسيد :

```
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();

}
}
}
```

توضيحاتي در مورد كد فوق :

هنگام كار با آرايه هاي rectangular براي درسترسي به اعضا به صورت زير عمل مي شد :

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

اما در اينجا بدين صورت است :

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

استفاده از System.Array :

دات نت فريم ورك كلاسي را معرفي كرده است به نام Array. توسط اين كلاس كار با آرايه ها و اعمال روي آنها براي مثال سورت كردن و غيره به شدت ساده مي شود .

مثال : يك برنامه ي سي شارپ جديد console را در VS.NET باز كنيد و نام آنرا درابتدا ex22 انتخاب نماييد. سپس كد زير را درون آن بنويسيد :

```
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() ;

}
}
}
```

توضيحاتي در مورد كد فوق :
از دو متد Sort و Reverse در اينجا براي سورت كردن و نمايش آرايه به ترتيب معكوس (از انتها به ابتدا) استفاده گرديده است.

تعريف آرايه هاي ديناميك در سي شارپ :

يكي از مشكلاتي كه با آرايه هاي معمول وجود دارد اين است كه قبل از هر كاري بايد طول آنها را مشخص كرد. گاهي از اوقات ما دقيقا نمي دانيم برنامه چه تعداد عضو را دريافت مي كند تا آرايه اي از پيش تعريف شده با همان تعداد عضو ايجاد كنيم. براي حل اين مشكل از كلاس ArrayList تعريف شده در دات نت فريم ورك مي توان استفاده كرد.

هنگام استفاده از ArrayList نيازي به دانستن تعداد اعضايي كه بايد اضافه شوند نمي باشد و با استفاده از متد Add آن به سادگي مي توان اعضاء را به آن اضافه نمود . تعدادي از خواص و متدهاي اين كلاس به صورت زير هستند :

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

مثال : يك برنامه ي سي شارپ جديد console را در VS.NET باز كنيد و نام آنرا درابتدا 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.Write("{0} ", empArray[i].ToString( ));
}
Console.WriteLine("\n");
Console.WriteLine("empArray.Capacity: {0}",
empArray.Capacity);

Console.ReadLine();

}
}
}
```

توضيحاتي در مورد كد فوق :

1- با كلمه ي كليدي override در قسمت هاي بعدي آشنا خواهيم شد.
2- براي استفاده از ArrayList لازم بود تا فضاي نامي را كه اين كلاس در آن تعريف شده است ، به برنامه اضافه كرد.
3- در مثال فوق نحوه ي تعريف دو كلاس را در يك فضاي نام مشاهده مي نماييد.
4- نحوه ي تعريف و مقدار دهي ArrayList و همچنين استفاده از خواص آن در مثال فوق بررسي شده است

قسمت يازدهم :

از اين قسمت به بعد مي خواهيم نگاهي دقيق تر به بحث شيء گرايي در سي شارپ بياندازيم؛ همانند فضاهاي نام ، كلاس ها ، ارث بري ، پلي مرفيسم و غيره.

در قسمت هاي قبل آشنايي مختصري با فضاهاي نام پيدا كرديم. در ادامه جزئيات بيشتري را در مورد آن بررسي خواهيم كرد.
فضاهاي نام (namespaces) براي اداره كردن و نظم بخشيدن به كدها ارائه شده اند. همچنين از امكان تشابه اسمي در بين قسمت هاي مختلف برنامه نيز جلوگيري مي كنند. استفاده از آنها عادت پسنديده اي است هنگاميكه قصد داريم از كد نوشته شده بارها و بارها استفاده كنيم.

مثال : يك برنامه ي سي شارپ جديد console را در VS.NET باز كنيد و نام آنرا درابتدا ex24 انتخاب نماييد. سپس كد زير را درون آن بنويسيد :

```
// 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.");
}
}
}

}
```

توضيحاتي در مورد كد فوق :
يكي از روش هاي مناسب براي معرفي فضاهاي نام ، ارائه ي آنها به صورت سلسله مراتبي مي باشد. قسمت هاي عمومي تر در بالا و قسمت هاي اختصاصي تر در فضاهاي نام داخلي تر قرار داده مي شوند. اين روش به معرفي فضاهاي نام تو در تو منتهي مي شود (nested namespaces) ، همانند مثال بالا.

كد فوق را به صورت زير با استفاده از عملگر دات (.) مي توان خلاصه نويسي كرد و نتيجه با قبل تفاوتي 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 درابتدا 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.");
}
}
}
```

توضيحاتي در مورد كد فوق :
در كد فوق نحوه ي استفاده از اعضاي تعريف شده در فضاهاي نام را مي توان مشاهده كرد. نحوه ي استفاده از آنها همانطور كه در قسمت هاي قبل نيز گفته شد به صورت زير است :

```
ProjectName.NameSpace.ClassName.MemberName
```

براي مثال در فضاي نام tutorial كلاس myExample1 قرار دارد و داخل آن متد myPrint1 تعريف شده است. پس نحوه ي دسترسي به متد آن به صورت زير است :

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

كلاس هاي myExample1 و myExample2 هر دو به يك فضاي نام (ex25.tutorial) تعلق دارند ، هر چند جدا از هم نوشته شده اند. حتي آنها را با حفظ سلسله مراتب خودشان مي توان در فايلهاي جداگانه اي نيز نوشت.

استفاده از using :

مثال : يك برنامه ي سي شارپ جديد console را در VS.NET باز كنيد و نام آنرا درابتدا ex26 انتخاب نماييد. سپس كد زير را درون آن بنويسيد :

```
// 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.");
}
}
}
```

توضيحاتي در مورد كد فوق :

همانند مثال بالا ، براي خلاصه نويسي مي توان از كلمه ي using به همراه نام namespace مورد نظر استفاده كرد. براي مثال اگر متد WriteLine را بخواهيم كامل بنويسيم به صورت زير است :

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

اما با قيد كردن و الحاق كردن فضاي نام آن ، ديگر نيازي به ذكر System در ابتداي آن نيست.

نكته :

باز هم مي توان خلاصه نويسي بيشتري را ارائه داد

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

در اين صورت تنها كافي است متد كلاس تعريف شده در آنرا به صورت زير فراخواني كنيم :

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

قسمت دوازدهم :

كلاس ها در سي شارپ :

تا بحال در حد كاربرد ، با كلاس ها آشنا شده ايم . اما در اين قسمت مي خواهيم نگاهي دقيق تر به كلاس ها بيندازيم.

هر كدي در سي شارپ قسمتي از يك كلاس مي باشد و تركيب تمام خواص و متدهاي موجود در يك كلاس يك نوع داده ي جديد تعريف شده از طرف ما را پديد مي آورد. هر متغيري كه از كلاس ساخته شود ، شيء ناميده مي شود و يك كپي منحصر به فرد است. براي مثال برنامه ي زير را درنظر 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 اينجا كلاس Data تعريف شده است و داراي يك عضو به نام x مي باشد. به اين نوع داده در كلاس فيلد گفته مي شود و هنگاميكه به صورت public معرفي مي شود يعني خارج از كلاس نيز قابل دسترسي است. در كد بالا دو متغير از كلاس تعريف و مقدار دهي اوليه شده اند. خروجي برنامه به صورت زير است :

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

دليل اين خروجي آن است كه هر instance (نمونه) از كلاس منحصر بفرد است و در اينجا نمي توان انتظار داشت كه هر دو خروجي يكي شوند.

براي مقدار دهي اوليه متغيرهايي كه به صورت فيلد تعريف مي شوند ، بهتر است مقدار دهي آنها را در سازنده ي كلاس (constructor) انجام دهيم.

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

همانطور كه پيشتر نيز ذكر شد ، متدي كه هم نام كلاس است ، سازنده نام مي گيرد. يك كلاس مي تواند بيش از يك سازنده داشته باشد. براي مثال :

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

از آنجائيكه كه سازنده ي بدون پارامتر ذكر شده در كد فوق private تعريف شده است بنابراين خارج از كلاس ديگر قابل دسترسي نمي باشد . بنابراين كدي خارج از كلاس ، تنها مي تواند از دو سازنده ي ديگر استفاده كند. براي مثال تعريف دو متغير جديد از اين كلاس به صورت زير مي باشد :

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

سي شارپ به شما اجازه مي دهد تا سازنده ها را در يك كلاس توسط كلمه ي كليدي this نيز فراخواني كنيد يعني بجاي ذكر نام متد سازنده از كلمه ي this استفاده شود (در خود كلاس) .

اگر مي خواهيد متغيري را بين نمونه (instance) هاي مختلف يك كلاس به اشتراك بگذاريد كلمه ي كليدي static وارد صحنه مي شود. به مثال زير توجه كنيد :

```
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);
}
}
```

بايد خاطر نشان كرد كه متغيرهاي استاتيك توسط نمونه هاي كلاس قابل دستيابي نيستند و فقط درون كلاس به شكل زير مي توان از آنها استفاده كرد :

.

در مثال فوق دو نمونه از كلاس Counted تعريف شده است. با هر بار فراخواني كلاس ، خودبخود سازنده اجرا شده و يك عدد به اين شمارشگر استاتيك اضافه مي شود. همانطور كه ذكر شد، براي اينكه بتوان به اين متغير استاتيك در خارج از كد دسترسي پيدا كرد يك متد غير استاتيك تعريف شده است.

در مثال فوق تابع GetInstanceCount تنها يك عدد را بر مي گرداند. در برنامه نويسي شيء گرا مرسوم است كه در اين حالت به جاي توابع از خواص استفاده شود كه به اندازه ي كافي در مورد آنها در قسمت هاي قبل توضيح داده شد. در اين صورت تعريف فوق به صورت زير در مي آيد :

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

و در اين صورت قسمت بعدي كد به صورت زير اصلاح مي شود (فراخواني خواص ، بدون ذكر پرانتزها بعد از نام آنها صورت مي گيرد):

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

اگر يك خاصيت هم خواندني و هم نوشتني باشد به صورت زير تعريف مي شود :

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

فيلدهاي پابليك را مي توان خواند و يا تغيير داد. اگر لازم باشد تا كاربر نتواند آنها را تغيير دهد مي توان از كلمه ي كليدي readonly قبل از تعريف آنها استفاده كرد. مثال :

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

قسمت سيزدهم :

با استفاده از ايندكسرها مي توان با يك كلاس همانند آرايه ها رفتار كرد. به مثال زير توجه كنيد :

```
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 ي تعريف و استفاده از ايندكسرها را مي توان مشاهده كرد. كلاس IntIndexer حاوي آرايه اي به نام myData مي باشد. بدليل private بودن آن در خارج از كلاس قابل دسترسي نيست. اين آرايه در سازنده ي كلاس (متد IntIndexer) با كلمه ي empty مقدار دهي اوليه شده است.
عضو بعدي كلاس Indexer مي باشد و با كلمه ي كليدي this و براكتها مشخص شده ست (this[int pos]). همانطور كه ملاحظه مي فرماييد نحوه ي تعريف ايندكسرها شبيه به تعريف خواص مي باشد.

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

خروجي مثال فوق به صورت زير 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 صحيح روشي است متداول براي دسترسي به اعضاي آرايه ها در بسياري از زبانها اما ايندكسرها در سي شارپ فراتر از اين مي رود. ايندكسرها را مي توان با پارامترهاي متعددي تعريف كرد و هر پارامتر با نوعي مختلف (دقيقا همانند پارامترهاي ورودي متدها). البته محدوديتي كه اينجا وجود دارد در مورد نوع پارامتر ها است كه تنها مي تواند integers, enums, and strings باشد . بعلاوه قابليت Overloading ايندكسرها نيز وجود دارد. به همين جهت به آنها آرايه هاي هوشمند هم گفته مي شود (smart arrays). 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 اولين ايندكسر با يك پارامتر از نوع اعداد صحيح تعريف شده است و در ايندكسر دوم از نوع رشته.
خروجي برنامه ي فوق به صورت زير 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
```

نكته :
1- امضاي (ليست پارامترهاي) ايندكسر ها در يك كلاس بايد منحصر بفرد باشد .
2- تعريف يك ايندكسر به صورت استاتيك مجاز نيست.

در صورت نياز به ايندكسرهايي با پارمترهاي ورودي متعدد مي توان به صورت زير عمل كرد :

```
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;
}
}
}
```

قسمت چهاردهم :

ارث بري (Inheritance) :

ارث بري يكي از مفاهيم اوليه ي برنامه نويسي شيء گرا مي باشد. با استفاده از آن استفاده مجدد از كد موجود به نحوي مؤثر ميسر مي گردد و صرفه جويي قابل توجهي را در زمان برنامه نويسي پديد مي آورد. به كد زير دقت كنيد :

```
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.
```

كد فوق از دو كلاس استفاده مي كند. كلاس بالايي ParentClass و كلاس اصلي ChildClass مي باشد. كاري كه انجام شده است استفاده از كدهاي كلاس والد ParentClass در كلاس بچه (!) ChildClass مي باشد. براي اينكه ParentClass را بعنوان كلاس پايه براي ChildClass معرفي كنيم به صورت زير عمل شد :

```
public class ChildClass : ParentClass
```

كلاس پايه با استفاده از معرفي كولون ":" ، پس از كلاس مشتق شده تعريف مي شود. در سي شارپ تنها ارث بري يگانه پشتيباني مي شود. بنابراين تنها يك كلاس پايه را براي ارث بري مي توان تعريف كرد.

ChildClass دقيقا توانايي هاي ParentClass را دارا است. بنابراين مي توان گفت ChildClass همان ParentClass است. براي مثال در كد فوق ChildClass داراي متد print نمي باشد اما آنرا از كلاس ParentClass به ارث برده است و در متد Main برنامه از آن استفاده گرديده است.

هنگام ساختن يك شيء از كلاس مشتق شده (derived) ، ابتدا يك نمونه از كلاس والد خود بخود ساخته مي شود. اين مورد در خروجي كد فوق هنگامي كه متدهاي سازنده ها روي صفحه چاپ شده اند قابل مشاهده است.

تبادل اطلاعات بين كلاس والد و كلاس فرزند :

به مثال زير دقت كنيد :

```
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.
```

كلاس فرزند با كلاس والد در هنگام instantiation مي تواند تبادل اطلاعات كند. همانطور كه در مثال فوق بارز است با استفاده از كلمه ي كليدي base ، كلاس فرزند تابع سازنده ي كلاس والد را فراخواني كرده است. اولين خط خروجي بيانگر اين موضوع است.

گاهي از اوقات ما مي خواهيم تابعي را كه در كلاس والد تعريف شده است را در كلاس فرزند با تعريف ديگري و مخصوص به خودمان ارائه دهيم. در اينصورت تابع تعريف شده در كلاس فرزند ، تابع هم نام والد را مخفي خواهد كرد و ديگر آن تابع والد فراخواني نخواهد گرديد. در اين حالت تنها يك راه براي دسترسي به تابع اصلي والد وجود دارد و آن استفاده از base. مي باشد كه در كد فوق پياده سازي شده است.
با استفاده از base. مي توان به تمام اعضاي public و يا protected كلاس والد از درون كلاس فرزند دسترسي داشت.
راه ديگري كه براي اين منظور وجود دارد در آخرين خط كد فوق در متد Main پياده سازي شده است :

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

براي تبديل نوع هاي مختلف در سي شارپ مي توان از پرانتز و سپس ذكر نوع اصلي استفاده كرد به اين عمل casting و يا boxing هم مي گويند. در كد فوق درحقيقت child به نوعي از parent تبديل شده است. بنابراين مانند اين است كه يك نمونه از كلاس والد متد print همان كلاس را فراخواني مي كند.

قسمت پانزدهم :

پلي مرفيسم (Polymorphism)

يكي ديگر از مفاهيم اوليه ي شيء گرايي پلي مرفيسم (چند ريختي) مي باشد. پلي مرفيسم به معناي توانايي استفاده كردن از فرم هاي مختلف يك نوع است بدون توجه به جزئيات آن .
براي مثال هنگاميكه سيگنال تلفني شما فرستاده مي شود ، از نوع تلفني كه در انتهاي خط موجود است خبري ندارد. تلفن انتهاي خط ، مي خواهد يكي از تلفن هاي عهد عتيق باشد و يا تلفني با آخرين امكانات روز .
شركت مخابرات (!) تنها از نوع پايه اي به نام phone خبر دارد و فرض مي كند كه هر instance از اين نوع مي داند كه چگونه صداي زنگ تلفن شما را به صدا در آورد. بنابراين شركت مخابرات از تلفن شما به صورت پلي مرف استفاده مي كند.
در عمل پلي مرفيسم هنگامي مفيد خواهد بود كه بخواهيم گروهي از اشياء را به يك آرايه نسبت دهيم و سپس متدهاي هر يك را فراخواني كنيم. الزاما اين اشياء از يك نوع نخواهند بود.

نحوه ي ايجاد متدهاي پلي مرفيك :

براي ايجاد متدي كه نياز است تا پلي مرفيسم را پشتيباني نمايد ، تنها كافي است آنرا از نوع virtual در كلاس پايه تعريف كنيم. مثال :
فرض كنيد تابع DrawWindow در كلاس Window تعريف شده است. براي ايجاد قابليت پلي مرفيسم در آن به صورت زير عمل مي شود :

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

در اين حالت هر كلاسي كه از Window مشتق شود ، مجاز است نگارش خاص خودش را از DrawWindow ارائه كند. در اين صورت در كلاسي كه از كلاس پايه ي ما ارث مي برد ، تنها كافي است كه كلمه ي كليدي override را قبل از نام تابع مذكور ذكر نماييم.

يك مثال كامل :

```
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;
}
}
```

كلاس DrawingObject ، كلاسي پايه براي تمام كد ما كه از آن به ارث مي برد ، مي باشد. متد Draw در آن با كلمه ي كليدي virtual معرفي شده است. يعني تمام كلاس هاي فرزند اين كلاس والد مي توانند اين متد را override كنند (تحريف كردن و يا تحت الشعاع

قرار دادن هم ترجمه شده است!).
در ادامه سه كلاس تعريف شده اند كه تمامي آنها از كلاس مبنا ارث مي برند و تابع Draw را تحريف كرده اند (!). با استفاده از كلمه ي كليدي override مي توان تابع مجازي كلاس مبنا را با تعريفي جديد در زمان اجراي برنامه ارائه داد. تحريف شدن تنها زماني رخ مي دهد كه كلاس ، توسط ريفرنس كلاس مبنا مورد ارجاع واقع شده باشد.
و در متد Main برنامه از اين كلاس ها در عمل استفاده گرديده است. در متد Main ، آرايه اي از نوع DrawingObject تعريف و مقدار دهي اوليه شده است تا بتواند 4 شيء از نوع اين كلاس را در خودش ذخيره كند.
بدليل رابطه ي ارث بري موجود مي توان آرايه ي dObj را با نوع هايي از كلاس هاي Line، Circle و Square مقدار دهي كرد (همانند كدهاي بعدي متد Main) . اگر ارث بري در اينجا وجود نمي داشت مي بايست به ازاي هر كلاس يك آرايه تعريف مي شد.
سپس از حلقه ي زيباي foreach براي حركت در بين اعضاي اين آرايه استفاده گرديده است. در اينجا هر شيء متد خاص خودش را در مورد Draw فراخواني مي كند و نتيجه را روي صفحه نمايش خواهد داد.
خروجي نهايي به صورت زير خواهد بود :

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

قسمت شانزدهم :

كلاس ها ي abstract
كلاس ها را همچنين مي توان به صورت abstract تعريف كرد. از اين نوع كلاس ها نمي توان instance ايي را ايجاد نمود. در اين كلاس هاي پايه ، صرفا تعريف متدها و خواص هايي عنوان گرديده و در آينده در كلاس 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 كه شايد پيش بيايد اين است كه اگر interface ها صرفا تعريف توابع و خواص را مي توانند در خود جاي دهند پس چه دليلي براي بكار بردن آنها و طولاني كردن كار كد نويسي وجود دارد؟
كاربردهاي زيادي را مي توان براي اينترفيس ها برشمرد. اينترفيس يك رفتار را تعريف مي كند. فرض كنيد در حال توسعه ي برنامه ايي هستيد كه بر روي دو كامپيوتر مختلف بايد با هم در ارتباط مستقيم بوده و برهم كنش داشته باشند و هر برنامه از ماژولي به نام communication object CCommObj استفاده مي نمايد. يكي از متدهاي اين شيء ، SendData() مي باشد كه رشته اي را دريافت كرده و به برنامه ي ديگر مي فرستد. اين فراخواني از نوع asynchronous است زيرا ما نمي خواهيم اگر خطايي در شبكه رخ داد، برنامه براي هميشه منتظر باقي بماند. اما چگونه برنامه ي A كه تابع ذكر شده را فراخواني كرده است مي تواند تشخيص دهد كه پيغام به مقصد رسيده است يا خير و يا آيا خطايي در شبكه مانع رسيدن پيغام گشته است يا خير؟ جواب بدين صورت است كه CCommObj هنگام دريافت پيغام ، رخدادي را سبب خواهد شد و اگر خطايي رخ داده باشد خير. در اين حالت نياز به يك ماژول logging نيز احساس مي گردد تا خطاهاي رخ داده را ثبت نمايد. يك روش انجام آن اين است كه CCommObj پياده سازي اين امكان را نيز بعهده گرفته و اگر فردا نيز خواستيم ماژول ديگري را به برنامه اضافه كنيم هر روز بايد CCommObj را تغيير دهيم. تمام اين كارها را به سادگي مي توان در يك اينترفيس مدل كرد. روش آن نيز در ادامه بيان مي گردد:
در ابتدا يك اينترفيس ايجاد مي كنيم تا ليست تمام امكانات ممكن را "منتشر" كند:

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

شي ء CCommObj ما از اين توابع كه بعدا توسعه داده خواهند شد براي با خبر سازي كلاينت ها استفاده مي نمايد. تمام متدها در يك اينترفيس ذاتا پابليك هستند بنابراين نيازي به ذكر صريح اين مطلب نمي باشد و اگر اينكار را انجام دهيد كامپايلر خطاي زير را گوشزد خواهد كرد :

The modifier 'public' is not valid for this item

در ادامه كلاينت CClientApp_A را پياده سازي خواهيم كرد :

```
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 كد فوق كلاس CClientApp_A از ICommObjEvents ارث برده و تمام متدهاي اين اينترفيس را پياده سازي نموده است. هنگامي كه CCommObj تابع OnDataSent را فراخواني مي كند اين كلاينت پيغام را دريافت خواهد كرد. لازم به ذكر است كه كلاس كلاينت ما چون از يك اينترفيس ارث بري مي نمايد پس بايد تمام توابع و خواص كلاس پايه را پياده سازي كند در غير اينصورت هر چند برنامه كامپايل خواهد شد اما هنگامي كه شيء CCommObj هر كدام از توابع اين كلاس را فراخواني كد ، خطاي زمان اجرا رخ خواهد داد. متد Init كلاس فوق آرگوماني را از نوع CCommObj دريافت نموده و در يك متغير private آنرا ذخيره مي نمايد. همچنين در اين متد ، متد Advise از كلاس CCommObj نيز فراخواني 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 ();
}
}
```

قسمت هفدهم :

در كلاس CCommObj كه با آن آشنا شديم ، آرايه اي Private از نوع ICommObjEvents به نام m_arSinkColl وجود دارد. اين آرايه تمام اينترفيس هاي sink شده را ذخيره مي كند. واژه ي sink در اينجا به كلاسي گفته مي شود كه دريافت كننده ي رخدادها است. متد Advise تنها sink وارده به آنرا در يك آرايه ذخيره مي كند و سپس انديس آرايه را كه در اينجا cookie ناميده شده است بر مي گرداند. اين كوكي توسط كلاينتي كه ديگر نمي خواهد از آن آيتم هيچونه رخدادي را دريافت كند به سرور فرستاده مي شود و سپس سرور اين آيتم را از ليست خودش حذف خواهد كرد.

نحوه ي فراخواني متد advise توسط كلاينت نيز جالب است.

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

در اينجا تنها يك this بعنوان آرگومان به متد advice فرستاده شده است در حاليكه انتظار مي رفت آرگوماني از نوع ICommObjEvents به تابع فرستاده شود. دليل صحت اين عمل بدين صورت است كه كلاس ClientApp_A از اينترفيس ICommObjEvents ارث برده است و آنرا پياده سازي نموده است.
در ادامه ليست كامل برنامه ي نوشته شده را در حالت Console ملاحظه مي فرماييد.

```
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 كلاينت تعريف كرده ايم و يك نمونه از CCommObj را. دو كلاينت instance هاي CCommObj را بعنوان آرگومان دريافت كرده اند. در هنگام فراخواني init توسط هر كلاينت متد advise فراخواني مي گردد. در خاتمه Read مربوط به كلاينت 1 فراخواني شده است كه سبب مي شود تا رخداد OnDataSend شيء CCommObj اجرا شود و به تمام كلاينت ها فرستاده شود.

هدف از اين 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 كه يك شيء واقعا يك اينترفيس را پياده سازي كرده است؟
دو روش براي فهميدن اين موضوع وجود دارد:
- استفاده از كلمه ي كليدي is
- استفاده از كلمه ي كليدي as

اولين مثال زير از كلمه ي كليدي is استفاده مي كند :

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

كلمه ي كليدي is مقدار true را بر مي گرداند اگر اپراتور سمت چپ ، اينترفيس سمت راست را پياده سازي كرده 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 شيء theClient5 به ICommObjEvents مي باشد. چون CClientApp_C اينترفيس را پياده سازي نمي كند حاصل خط اول نال خواهد بود.

به صورت خلاصه :
يك اينترفيس قراردادي است كه به كلاينت گارانتي مي دهد يك كلاس خاص چگونه رفتار خواهد كرد. هنگاميكه كلاسي يك اينترفيس را پياده سازي مي كند به تمام كلاينت ها مي گويد كه : من تمام موارد ذكر شده در اينترفيس را ارائه و پياده سازي خواهم كرد. نمونه ي عملي استفاده از اينترفيس ها بحث dot net remoting است.

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 شبيه به جاوا و سي پلاس پلاس استفاده مي نمايد.

دلايلي كه بايد در برنامه exception handling حتما صورت گيرد به شرح زير است:
- قابل صرفنظر كردن نيستند و اگر كدي اين موضوع را در نظر نگيرد 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 مديريت 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 مديريت كرد. و اگر از اين قطعات خطايابي استفاده نشود برنامه به صورتهاي زير 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 مي كند. سپس وارد بلوك و يا قطعه ي try مي گردد. اگر هيچ خطايي هنگام اجراي دستورات داخل آن رخ ندهد ، برنامه به خط آخر جهش خواهد كرد و كاري به قطعات catch ندارد.
اما در اينجا در اولين try عددي بر صفر تقسيم شده است بنابراين كنترل برنامه به بلوك catch منتقل مي شود و صرفا نوع خطاي رخ داده شده نوشته و نمايش داده مي شود. سپس برنامه به كار عادي خودش ادامه مي دهد.

تعدادي از كلاس هاي exception در سي شارپ كه از كلاس System.Exception ارث برده اند به شرح زير 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.
• InvalidCastExpression - 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 كد فوق صرفا عمومي ترين نوع از اين كلاس ها كه شامل تمامي اين موارد مي شود مورد استفاده قرار گرفت يعني :

```
catch ( Exception e )
```

اگر نيازي به خطايابي دقيقتر باشد مي توان از كلاس هاي فوق براي اهداف مورد نظر استفاده نمود.

مثالي ديگر: (در اين مثال خطايابي دقيق تر با استفاده از كلاس هاي فوق و همچنين مفهوم finally نيز 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 :

اين عبارت سبب ايجاد يك خطا در برنامه مي شود.

مثال :

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

در اين حالت قسمت فاينالي اجرا شده و برنامه بلافاصله خاتمه پيدا مي كند .

قسمت نوزدهم :

سربارگذاري عملگر ها (Operator OverLoading)

به تعريف مجدد راه و روش اجراي عملگر ها توسط ما ، سربارگذاري عملگرها گفته مي شود. فرض كنيد مي خواهيد عدد 2 را به يك مقدار datetime اضافه كنيد. خطاي زير حاصل خواهد شد:

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

جالب بود اگر مي توانستيم عدد 2 را به datetime اضافه كنيم و نتيجه ي آن تعداد روزهاي مشخص بعلاوه ي دو مي بود. اينگونه توانايي ها را مي توان بوسيله ي operator overloading ايجاد كرد.

تنها عملگر هاي زير را مي توان overload كرد:

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

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

نحوه ي انجام اينكار نيز در حالت كلي به صورت زير است:

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

به مثال زير توجه كنيد:

```
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. يكبار توسط آن مي توان يك عدد صحيح را به يك تاريخ اضافه كرد و بار ديگر يك يك تاريخ را مي توان به عدد صحيح افزود.

موارد زير را هنگام سربارگذاري عملگرها به خاطر داشته باشيد:

1- تنها اپراتورهاي ذكر شده را مي توان overload كرد. اپراتورهايي مانند new,typeof, sizeof و غيره را نمي توان سربارگذاري نمود.
2- خروجي متدهاي بكار گرفته شده در سربارگذاري عملگر ها نمي تواند void باشد.
3- حداقل يكي از آرگومانهاي بكار گرفته شده در متدي كه براي overloading عملگرها بكار مي رود بايد از نوع كلاس حاوي متد باشد.
4- متدهاي مربوطه بايد به صورت public و static تعريف شوند.
5- هنگامي كه اپراتور < را سربارگذاري مي كنيد بايد جفت متناظر آن يعني > را هم سربارگذاري نماييد.
6- هنگاميكه براي مثال + را overload مي كنيد خودبخود =+ نيز overload شده است و نيازي به كدنويسي براي آن نيست.

يكي از موارد جالب بكار گيري سربارگذاري عملگرها در برنامه نويسي سه بعدي و ساختن كلاسي براي انجام عمليات ماتريسي و برداري مي باشد

قسمت بيستم :

Delegates در سي شارپ روشي مطمئن و typesafe را براي بكار گيري مفهوم pointer function ارائه مي دهند. يكي از ابتدايي ترين استفاده هاي function pointers پياده سازي callback مي باشد. اما در ابتدا لازم است تا با اصول اوليه ي كاري آن آشنا شويم.

مثال يك :
يك delegate چگونه تعريف و استفاده مي شود؟
Delegate يك شيء است كه بيانگر يك تابع مي باشد بنابراين مي تواند بعنوان آرگومان ورودي يك تابع ديگر و يا عضوي از يك كلاس بكار رود.
در زبان " "function-pointer، Func1() اشاره گري به Func2() را بعنوان پارامتر دريافت كرده و نهايتا آنرا فراخواني مي كند.
در زبان " "delegate، Func1() يك شيء delegate از Func2() را دريافت كرده و سپس آنرا فراخواني مي كند.
در مثال زير از دو تابع براي شرح اين مطلب سود جسته شده است:
Func1() از delegate استفاده مي كند.
Func2() يك delegate است.

(شماره گذاري خطوط ، در كد زير ، صرفا براي راحت تر شدن توضيحات در مورد آنها است و لزومي به تايپ آنها در برنامه ي اصلي نيست.)

```
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 شيء delegate را براي Func2 تعريف مي كند.

LINE 04-06
تابعي را تعريف كرده است كه آرگومان ورودي آن از نوع Delg است.

LINE 07-09
تابعي را تعريف مي كند كه بايد به صورت delegate به تابع ديگر فرستاده شود.

LINE 10-14
تابع Main اجراي برنامه را با ايجاد يك شيء delegate براي Func2 آغاز كرده و سپس تابع Func1 را فراخواني مي كند.

مثال 2:
چگونه مي توان از delegates در كارهاي عملي استفاده كرد؟

طرح يك مساله:
شخصي تقاضاي ثبت نام در يك مؤسسه ي آموزشي و همچنين تقاضاي كاريابي در يك شركت را داده است. هر كدام از اين نهادها روشي خاص خود را براي ارزيابي شخص دارند.

راه حل (با روشي شيء گرا):
شخص مشخصاتي همچون سن / جنس / ميزان تحصيلات قبلي / تجربيات كاري و مدارك مرتبط دارد.
مؤسسه ي آموزشي تعدادي از اين مشخصات را براي ارزيابي شخص استفاده مي كند و اين امر در مورد شركت ياد شده نيز صادق است.
شيء شركت و شيء آموزشگاه هر كدام توابع ارزيابي خاص خودشان را پياده سازي مي كنند.
شخص ، اينترفيسي واحدي را در اختيار شركت / آموزشگاه براي ارزيابي خود قرار مي دهد.

پياده سازي (با استفاده از سي شارپ):
ما delegateايي را تعريف مي كنيم كه بيانگر اينترفيسي است كه به شركت و آموزشگاه اجازه ي چك كردن شخص را مي دهد.
سه كلاس school و company و person را تعريف مي نماييم.
كلاس 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 نياز را تعريف مي كند.

LINE 04-23
كلاس person را تعريف مي كند. اين كلاس تابعي پابليك را ارائه مي دهد كه آرگومان ورودي آن از نوع GetChecker مي باشد.

LINE 24-30
كلاس school را تعريف مي كند و سپس تابعي را كه delegate است ارائه مي دهد.

LINE 31-37
كلاس company را تعريف مي كند و سپس تابعي را كه delegate است ارائه مي دهد.

LINE 38-36
كلاس test را پياده سازي مي نمايد. سپس يك شيء شخص ساخته مي شود. در ادامه new GetChecker(School.SchoolCheck) و new GetChecker(Company.CompanyCheck) شيء ايي را ايجاد مي كند از نوع delegate مورد نياز و آنرا به تابع CheckMe مي فرستد. خروجي نتيجه ي ارزيابي اين شخص مي باشد.

اگر چك كردن اشخاص بيشتري نياز باشد به اين صورت عمل مي 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 بين دات نت فريم ورك و سي شارپ چه نقشي دارد؟

طرح يك مساله:
نمايش دادن ميزان پيشرفت خواندن يك فايل هنگامي كه حجم فايل بسيار زياد است.

راه حل (با استفاده از سي شارپ):
در مثال زير از كلاس FileReader براي خواندن يك فايل حجيم استفاده شده است. هنگاميكه برنامه مشغول خواندن فايل است ' Still reading.. را نمايش مي دهد و در پايان 'reading Finished..'. را عرضه مي كند.
براي اينكار از فضاي نام System.IO استفاده شده است. اين فضاي نام حاوي delegate ايي مهيا شده براي ما مي باشد. بدين ترتيب مي توانيم به دات نت فريم ورك بگوييم كه ما تابعي را تعريف كرده ايم كه او مي تواند آنرا فراخواني كند.
سؤال: چه نيازي وجود دارد تا دات نت فريم ورك تابع ما را فراخواني و اجرا كند؟ با استفاده از تابع ما كه دات نت فريم آنرا صدا خواهد زد در طول خواندن فايل به ما گفته مي شود كه بله! من هنوز مشغول خواندن هستم! به اين عمليات Callback نيز گفته مي شود. به اينكار پردازش asynchronous نيز مي گويند!

```
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,
0,
arrByte.Length,
callbackOnFinish,
null);
14 Console.WriteLine("Still reading..");
15 }
16 else Console.WriteLine("Finished reading..");
17 }

18 public void readFile(){
19 sInput = File.OpenRead(@"C:\big.dat");
20 sInput.BeginRead(arrByte,
```

```
0,
arrByte.Length,
callbackOnFinish,
null);
21 for(long i=0;i<=1000000000;i++){
// just to introduce some delay
22 }
23 }

24 public static void Main(){
25 FileReader asynctest=new FileReader();
26 asynctest.readFile();
27 }
28 }
```

02 LINE
فضاي نام System.IO را به برنامه ملحق مي كند. اين فضاي نام به صورت خودكار حاوي تعريف delegate زير مي باشد:

```
public delegate void AsyncCallback (IAsyncResult ar);
```

LINE 03-10
تعريف كلاس

LINE 06
شيء delegate را تعريف مي كند.

LINE 07-10
سازنده ي كلاس را پياده سازي مي كنند. در اينجا ما تصميم گرفته ايم كه بافري حاوي 256 بايت را در هر لحظه بخوانيم.

LINE 09
شيء delegate نمونه سازي شده است.

LINE 18-23
readFile را پياده سازي مي كند.

LINE 12-16
نحوه ي استفاده از شيء IAsyncResult را بيان مي كند.

LINE 12
sInput.EndRead(result) تعداد بايتهاي خوانده شده را بر مي گرداند. اين خواندن تاجايي كه تعداد بايتهاي خوانده شده صفر است ادامه پيدا مي كند و در اينجا 'Finished reading..' اعلام مي گردد.