برنامه نویسی شی گرا در سی شارپ (C#) قسمت ۱۰: Exntesion Method ها (نسخه چاپی)

در قسمتی که در مورد کلاس ها و اشیاء صحبت کردیم، گفتیم زمانی که شما کلاسی را تعریف می کنید باید از روی آن کلاس شئ ای بسازید تا به اعضای آن دسترسی داشته باشید. اما حالت هایی وجود دارد که شما می توانید یک کلاس و اعضای آن را به صورتی تعریف کنید که دسترسی به اعضای آن کلاس بدون تعریف شئ از آن امکان پذیر باشد. در طول این دوره آموزشی با یکی از این کلاس ها کار کرده ایم. کلاس Console که شما می توانستید بدون ساختن شئ از روی آن متدهای آن را صدا بزنید. برای مثال متد WriteLine:

```
Console.WriteLine("Welcome to ITPro.ir");
```

دلیل این امر، تعریف کلاس Console و اعضای آن به صورت static است.

کلاس ها و اعضای static

اعضای static، در حقیقت اعضایی هستند که وابسته به شئ نیستند و در بین کل شئ های ساخته شده از یک کلاس مشترک می باشند. افرادی که قبلاً با زبان Visual Basic کار کرده باشند با این اعضاء با نام نام Shared آشنای دارند. در زبان سی شارپ اعضای static با کلمه کلیدی static تعریف می شوند. مثال زیر را در نظر بگیرید:

```
public class Messages
{
    public static void Welcome()
    {
        Console.WriteLine("Welcome to ITPro.ir");
    }
}
```

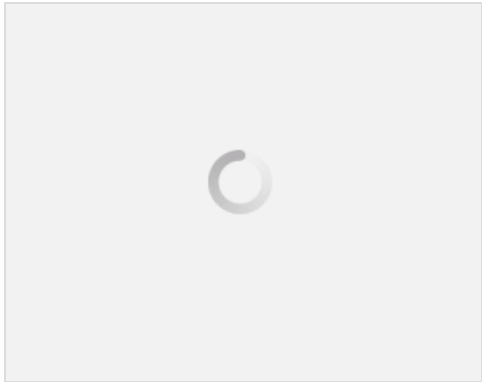
در کلاس بالا، متد Welcome به صورت static تعریف شده است. کافیست برای استفاده از این متد به صورت زیر عمل کنیم:

```
Messages.Welcome();
```

دقت کنید که هیچ شئ ای از کلاس Messages ایجاد نشده است و تنها با نوشتن نام کلاس و تایپ کاراکتر . به اعضای static آن کلاس دسترسی داریم. در صورتی که شئ ای از کلاس Messages بسازیم و لیست اعضای آن را مشاهده کنیم خبری از متد Welcome نخواهد بود.ما می توانیم یک کلاس را به صورت static تعریف کنیم. در صورتی که کلاسی به صورت static تعریف شود، تمامی اعضای آن باید به صورت static تعریف شوند و همچنین دیگر امکان ساخت شئ از روی آن کلاس وجود نخواهد داشت:

```
public static class Messages
{
    public static void Welcome()
    {
        Console.WriteLine("Welcome to ITPro.ir");
    }
}
```

در صورتی که داخل یک کلاس static عضو غیر static تعریف کنیم، با پیغام خطا مواجه خواهیم شد:

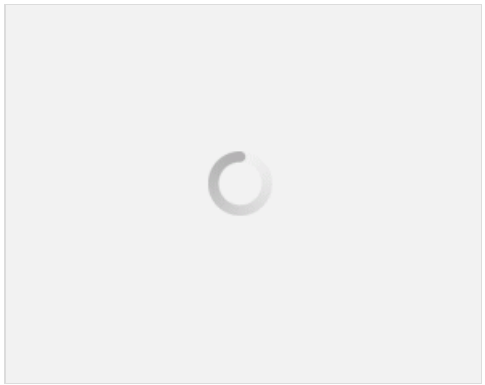


در بخشی که متدها را توضیح می دادیم، هنگام تعریف متدها گفتیم متدهایی که داخل کلاس Program تعریف می شدند را باید به صورت static تعریف کنیم که از داخل متد Main به آنها دسترسی داشته باشیم. دلیل این کار این بود که متد Main خود به صورت static تعریف شده است و شما از داخل یک متد static می توانید تنها متدهای static داخل همان کلاس را صدا بزنید، در غیر اینصورت باید از روی آن کلاس شئ ای بسازید و سپس متد مربوطه را اجرا کنید. به مثال زیر توجه کنید:

```
public class Messages
{
    public static void Welcome()
    {
        Console.WriteLine("Welcome to ITPro.ir");
    }

    public void Goodbye()
    {
            Console.WriteLine("Goodbye. Please comeback soon.");
    }
}
```

کلاس Messages را از حالت static خارج کردیم، پس می توانیم اعضای غیر static داخل آن داشته باشیم. اما فرض کنید می خواهیم از متد Goodbye داخل متد Welcome استفاده کنیم. اگر متد Goodbye را بدون ساختن شئ صدا کنیم پیغام خطا دریافت خواهیم کرد:



برای رفع این مشکل متد Welcome را به صورت زیر باید بنویسیم:

```
public static void Welcome()
{
    Console.WriteLine("Welcome to ITPro.ir");
    var messages = new Messages();
```

```
    messages.Goodbye();
}
```

ابتدا از خود کلاس Messages یک شئ ایجاد کرده و از روی شئ اقدام به فراخوانی متد Goodbye کردیم. به این نکته دقت کنید که امکان استفاده از اعضای static در بخش های غیر static مشکلی ایجاد نمی کند و می توان به صورت مستقیم آن ها را فراخوانی کرد.

```
public class Messages
{
    public static void Welcome()
    {
        Console.WriteLine("Welcome to ITPro.ir");
    }

    public void Goodbye()
    {
        Console.WriteLine("Goodbye. Please comeback soon.");
        Welcome();
    }
}
```

در متد Goodbye ما به صورت مستقیم متد Welcome را صدا زدیم بدون اینکه پیغام خطایی دریافت کنیم. اما اگر برای فراخوانی متد Welcome در مثال بالا، از کلمه کلیدی this استفاده می کردیم، با پیغام خطا مواجه می شدیم. به این خاطر که کلمه کلیدی this همانطور که در قسمت های قبلی گفتیم، به شئ ساخته شده از روی یک کلاس اشاره می کند.در ابتدای همین بخش، گفتیم که اعضای static بین کل اشیاء ساخته شده از یک کلاس مشترک هستند. یعنی به ازای هر شئ، مقدار متفاوتی ندارند، چون وابسته به شئ نیستند. برای مثال، کد زیر یک خاصیت static از نوع int با نام Instances تعریف می کند که تعداد اشیاء ایجاد شده داخل یک کلاس را به ما نمایش می دهد. برای اینکه به ازای ساخته شدن هر شئ مقدار Instances یک واحد اضافه شود، سازنده پیش فرض را برای کلاس به صورت زیر می نویسیم:

```
public class Sample
{
    public static int Instances { get; private set; }

    public Sample()
    {
        Instances++;
    }
}
```

دقت کنید، بخش set خاصیت Instances با سطح دسترسی private تعریف شده است. یعنی ما تنها می توانیم داخل کلاس آن را مقدار دهی کنیم و خارج از کلاس امکان مقدار دهی به آن وجود ندارد. سپس داخل سازنده پیش فرض یک واحد به Instances اضافه می کنیم. حال کد زیر را اجرا می کنیم:

```
Console.WriteLine(Sample.Instances);
```

```
var s1 = new Sample();
Console.WriteLine(Sample.Instances);
var s2 = new Sample();
var s3 = new Sample();
Console.WriteLine(Sample.Instances);
Console.ReadKey();
```

با اجرای قطعه کد بالا در متد Main، به ترتیب عددهای ۰، سپس ۱ و در انتها ۳ که به ترتیب تعداد اشیاء ساخته شده از روی کلاس Sample هستند نمایش داده می شود. به همین دلیل گفته می شود که اعضای static بین کلیه اشیاء ساخته شده در یک کلاس مشترک هستند.

سازنده های static

در قسمت قبل گفتیم که سازنده ها به ما امکان اجرای کد مورد نظر در هنگام ساختن اشیاء را می دهند. اما سازنده ای وجود دارد که کاربردش برای اعضای static است. سازنده های static به صورت زیر تعریف می شوند:

```
static {classname}()
{
}
```

که به جای classname نام کلاسی که سازنده برای آن ایجاد می شود را می نویسیم. مثال:

```
public class StaticConstructor
{
    public static string FirstName { get; set; }
    public static string LastName { get; set; }

    static StaticConstructor()
    {
        FirstName = "None";
        LastName = "None";
    }
}
```

در کد بالا، هنگام اجرای سازنده مقادیر FirstName و LastName به None تغییر می کند. اما سازنده static چه زمانی اجرا می شود؟ اجرای سازنده های static درست زمانی که شما تصمیم دارید به یکی از فیلدهای static یک کلاس دسترسی پیدا کنید، تنها برای یکبار اجرا می شود. تنها برای یکبار، یعنی شما اگر ۱۰ مرتبه فیلدهای static یک کلاس را استفاده کنید، سازنده static تنها برای دسترسی اول اجرا خواهد شد. مثال:

```
Console.WriteLine(StaticConstructor.FirstName);
StaticConstructor.FirstName = "Hossein";
Console.WriteLine(StaticConstructor.FirstName);
```

کد بالا به ترتیب مقادیر None و سپس Hossein را در خروجی چاپ می کند. به نکات زیر هنگام استفاده از سازنده های static توجه کنید:

۱. سازنده های static تنها یکبار در طول اجرای کد و آن هم زمان اولین دسترسی به اعضای static اجرا خواهند شد.

۲. سازنده های static نمی توانند سطح دسترسی غیر از public داشته باشند.
۳. سازنده های static نمی توانند پارامتری را به عنوان ورودی بگیرند.

استفاده از کلاس ها و اعضاء static بایـد با دقت زیاد انجام شود. زیرا این فیلدها خطرات زیادی را برای برنامه ایجاد می کنند، مخصوصاً برنامه های تحت وب که شـما باید مباحث مربوط به همزمانی را در هنگام دسترسـی به اعضای static رعایت کنید. در قسمت همزمانی و آشنایی با برنامه نویسی Aynchronous به صورت کامل با این مبحث آشنا می شوید.

Extension Method ها

بعـد از آشـنایی با کلاس ها و اعضای static به سراغ Exntesion Method هـا می رویم. موقعیتی را در نظر بگیریـد که می خواهیـد به یک کلاس متدی اضافه کنید. اما یا کد کلاس در اختیار شـما نیست و یا نمی خواهید کد اصلی کلاس دستکاری شود. برای اینکار از Extension Method ها اسـتفاده می شود. این قابلیت به ما این اجازه را می دهـد تا به نمونه های یک کلاس رفتاری را اضافه کنیم. این عملیات برای کلاس های تعریف شده توسط خود ما و همچنین نوع های داده اولیه و کلیه کلاس هایی که داخل دات نت تعریف شده اند قابل استفاده می باشد. شیوه کلی تعریف Extension Method ها به صورت زیر است:

۱. ابتدا باید یک کلاس static برای تعریف Extension Method ها تعریف کنیم.

۲. به ازای هر Extension Method، متدی با ساختار زیر داخل کلاس static تعریف شده ایجاد می کنیم:

```
public static {return-type} {name}(this {datatype} {instancename}, {parameters})
{
}
```

۱. return-type نوع داده بازگشتی متد را مشخص می کند.
۲. name نام متدی که قرار است به شئ مورد نظر از یک کلاس اضافه شود. این نام کاملاً دلخواه است.
۳. datatype یا نوع داده ای که تصمیم داریم به شئ های آن یک متد اضافه کنیم.
۴. instancename یا نام متغیری که بواسـطه آن می خواهیم به شـئ ای که متد بر روی آن ایجاد می شود دسترسـی داشـته باشـیم را مشخص می کنیم.این نام کاملاً دلخواه است.
۵. parameters یا لیست پارامترهای ورودی متدی که قصد تعریف آن را داریم مشخص می کند.

برای مثال، فرض کنید میخواهیم به نوع داده int یک متد اضافه کنیم که یک عدد را به عنوان ورودی گرفته و عدد داخل متغیر ایجاد شده را به توان ورودی رسانده و بر می گرداند. برای این کار کافیست ابتدا کلاسی برای Extension Method ها اضافه کنیم:

```
public static class IntExtensions
{


}
```

بهـتر است برای Extension Method هـای هر نوع داده، یک کلاس جـداگانه ایجاد کنیم ماننـد LongExtensions یا CustomExtensions که بخش اول اشاره به کلاسـی می کند که ما تصمیم داریم برای آن Extension Method ایجاد کنیم. در ادامه متدی با نام Pow به نوع داده Int اضافه می کنیم:

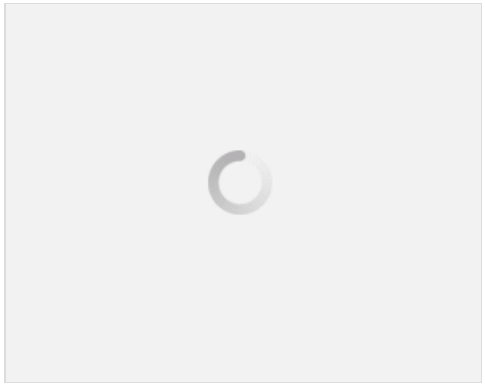
```
public static class IntExtensions
{
    public static int Pow(this int number, int pow)
    {
        int result = 1;
```

```
        for (int counter = 0; counter < pow; counter++)
            result = result*number;
        return result;
    }
}
```

حال برای استفاده از این متد کافیست آن را برای متغیرهایی از نوع int به صورت زیر فراخوانی کنیم:

```
int myNum = 4;
var pow = myNum.Pow(3);
Console.WriteLine(pow);
```

به قسمت تعریف Extension Method بر گردیم، پارامتری با نام number داخل متد Pow تعریف کردیم، در حقیقت این متد به مقدار داخل متغیری اشاره می کند که ما Extension Method را بر روی آن اجرا می کنیم. در کد بالا، پارامتر number به مقدار ۴ که داخل متغیر myNum ریخته شده اشاره می کند.شـما برای هر نوع داده و هر کلاسی می توانید Extension متد تعریف کنید. زمانی که Intellisense باز می شود، متـد های عادی به صورت یک مکعب نمایش داده می شونـد، اما Extension Method هـا به صورت یـک مکعب که یک فلش آبی رنگ به سمت پایین در بالای آن قرار دارد نمایش داده می شوند.



کلاس های partial

کلاس های partial به شـما این امکان را می دهنـد تا یک کلاس را به چند فایل بشـکنید. برای مثال، یک فایل تعریف Property ها و یک فایل تعریف Method ها. در مثال زیر من کلاسـی با نام Customer تعریف کردم که این کلاس به سه بخش تقسـیم شـده است، بخش اول خصوصیات، بخش دوم سازنده ها و بخش سوم متدها:

```
public partial class Customer
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public long Age { get; set; }
}

public partial class Customer
{
    public Customer()
    {
    }
```

```
    public Customer(string firstName, string lastName, long age)
    {
        FirstName = firstName;
        LastName = lastName;
        Age = age;
    }
}

public partial class Customer
{
    public string DisplayFullName()
    {
        return this.FirstName + " " + this.LastName;
    }
}
```

در حقیقت Customer یک کلاس است که به سه قسمت تقسیم شده. شما می توانید برای هر قسمت، یک فایل جداگانه ایجاد کنید. کلاس های partial زمان نوشتن برنامه ها کاربرد زیادی ندارند، به شخصه یاد ندارم داخل پروژه ای از این قابلیت استفاده کرده باشم. بیشترین استفاده ای که از کلاس های partial شده است داخل برنامه هایی از نوع Windows Application که فایل های مربوط به فرم های برنامه به چند بخش شکسته شده اند. در قسمت های بعدی با این نوع از برنامه ها بیشتر آشنا خواهیم شد.بعد از آشنایی با کلاس های static و کلاس های partial و همچنین Extension Method ها، در قسمت بعدی با مبحث Reference Type ها و Value Type ها و همچنین struct ها آشنا خواهیم شد. تا مبحث بعدی شما دوستان عزیز را به خدا می سپارم. **ITPRO باشید**

نویسنده : حسین احمدی

منبع : جزیره برنامه نویسی وب سایت توسینسو

#آموزش_زبان_سی_شارپ #کلاس_های_static_در_سی_شارپ #آموزش_برنامه_نویسی_شئ_گرا
#extension_method_ها_در_سی_شارپ #کلاس_های_partial_در_سی_شارپ #آموزش_برنامه_نویسی

nabiloo.reza@gmail.com

بینهایت سپاسگزارم از آموزش خوبتون.

rostami

عالی بود تشکر

dashaliyekarimi

سلام، یک مطلبی در مورد سازنده های static که علاوه بر دسترسی به مقادیر static، در اولین ساخت شی از آن کلاس نیز اجرا میشود.

https://msdn.microsoft.com*en-us*library/k۹x۶w۰hc.aspx

علی ورزشی

سلام اقای احمدی وقت بخیر

این بخشی که شما اومدید مقادیر رو تو سازنده ها ست کردید رو الان تست گرفتم خطا میده بهم اصلا نمیشناسه اونا رو

اول فیلد ها رو به صورت Private در نظر گرفتم که می خواستم تست کنم بعد دقیقا مثل کد های شما تست گرفتم ولی قبول نمی کنه و خطا داره میده و پیغامی که میده به این صورت هست :

The name 'FirstName' does not exist in the current context

حسین احمدی

سلام، کدی که کلاس رو تعریف کردید اینجا بزارید، احتمالاً نام Property رو اشتباه وارد کردید.

علی ورزشی

```
public partial class Person
  {
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public long Age { get; set; }
  }

  public partial class person
  {
    public person()
    {
    }

    public person(string firstName, string lastName, long age)
    {
      FirstName = firstName;
      LastName = lastName;
      Age = age;
    }
  }


  public partial class person
  {

  }
}
```

علی ورزشی

حسین احمدی

کـدی که شـما نوشتیـد به این خـاطر اجرا نمیشه که یک کلاس داخل کلاس دیگه تعریف کردیـد، یعنی Inner Type استفاده کردید و خصوصیت ها داخل کلاس اصلی و سازنده داخل کلاس داخلی نوشته شده، کد رو به صورت زیر تغییر بدید مشکل حل میشه:

```
public class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public long Age { get; set; }

    public Person()
    {
    }

    public Person(string firstName, string lastName, long age)
    {
        FirstName = firstName;
        LastName = lastName;
        Age = age;
    }
}
```

علی ورزشی

منظورتون رو متوجه نشدم

حسین احمدی

شما داخل کلی که تعریف کردید رو نگاه کنید، دو بار کلاس تعریف شده:

```
public class Person
{
    public class person
    {
    }
}
```

خصوصـیت هـا تو کلاـس اول و سازنـده تو کلاس دوم تعریف شـده بود که امکان دسترسـی به خصوصـیت ها از کلاس داخلی وجود نداشت، به همین خاطر خطا دریافت می کردید.

علی ورزشی

جناب احمدی والا من تو این عکس همچین کدی که شما می فرمایید رو نمی بینم

حسین احمدی

من کد شما رو اشتباه متوجه شده بودم، اینطور اصلاح می کنم:

شما سه کلاس جداگانه تعریف کردید، یکی داخلش خصوصیت ها، و دو کلاس دیگه که Partial هستند، مشکل به این خاطر هست که کلاس اول Person با P بزرگ شروع شده، اما دو کلاس بعدی با p کوچک، به همین خاطر خصوصیت ها از سازنده ها در دو کلاس متفاوت قرار گرفتند و شما نمی تونید به خصوصیت ها دسترسی داشته باشد.

علی ورزشی

خیلی ممنون اصال حواسم به اسم کلاس نبود

مهدی تاران

سلام

خیلی ممنون از مطالبتون . فکر کنم در خطی که درباره سازنده های استاتیک گفتین اشتباه شده :

سازنده های static نمی توانند سطح دسترسی غیر از public داشته باشند.

درستش برگرفته از https://msdn.microsoft.com*en-us*library/k۹x۶w۰hc(v=vs.۸۰).aspx

.A static constructor does not take access modifiers or have parameters

بقیه ویژگی های سازنده های استاتیک که شاید برای کاربران مفید باشه :

A static constructor is called automatically to initialize the class before the first instance is created or any static members are referenced.

.A static constructor cannot be called directly

.The user has no control on when the static constructor is executed in the program

.A typical use of static constructors is when the class is using a log file and the constructor is used to write entries to this file

Static constructors are also useful when creating wrapper classes for unmanaged code, when the constructor can call the LoadLibrary method

مطلب اصلی