آموزش برنامه نویسی شی گرا در سی شارپ (C#) قسمت ۶ : وراثت (نسخه PDF)

یکی از مباحث بسـیار مهم در برنامه نویسی شئ گرا، مبحث وراثت یا Inheritance است. در قسمت مقدمه برنامه نویسی شئ گرا، در مورد این مبحث به صورت مختصر صحبت کردیم. در ادامه تصمیم داریم که مبحث را بیشتر مورد بررسی قرار دهیم و با نحوه کاربرد آن در زبان سی شارپ بیشتر آشنا شویم. مبحث وراثت از این نظر مهم است که به شما کمک می کند از کدهای نوشته شده مجدداً استفاده کنید و در نتیجه حجم کده نوشته شده در برنامه شما به صورت محسوسی کاهش پیدا کند.

تعریف وراثت یا Inheritance و پیاده سازی آن در زبان C#

همانطور که در مقدمه مبحث برنامه نویسـی شـئ گرا خدمت دوسـتان توضـیح دادم، وراثت به معنی به ارث بردن یکسری خصوصیات و رفتار بوسـیله فرزند از والد است. در برنامه نویسـی شئ گرا، زمانی که صحبت از وراثت می کنیم، در حقیقت می خواهیم برای یک کلاسی، یک کلاس والد مشـخص کنیم. وراثت در برنامه نویسـی شئ گرا کاربردهای بسیاری دارد، به صورتی که اصلی ترین و بنیادی ترین قابلیت در برنامه نویسی شئ گرا نام برده می شود. قبل از شروع به نکته زیر توجه کنید:

زمانی که کلاس A به عنوان والد کلاس B معرفی می شود، یعنی کلاس B فرزند کلاس A می باشد، می گوییم کلاس B از کلاس A مشتق شده است. در طول این دوره از واژه مشـتق شده به تکرار اسـتفاده خواهیم کرد.در ابتدا با شـیوه کلی اسـتفاده از وراثت در کلاس ها آشنا می شویم. فرض کنید کلاسی داریم با نام A:

```
public class A
{


}
```

حال تصـمیم داریم کلاسی تعریف کنیم با نام B که از کلاس A مشـتق شده است، یعنی تمامی خصوصیات و رفتارهای کلاس A را به ارث می برد. برای اینکار کلاس B را به صورت زیر تعریف می کنیم:

```
public class B : A
{
}
```

بوسـیله دستور بالا، کلاس A به عنوان کلاس والـد کلاس B در نظر گرفته خواهـد شـد. گفتیم یکی از مزایـای اسـتفاده از وراثت در برنامه نویسـی شـئ گرا، استفاده مجدد از کدهایی است که در کلاس والد تعریف شده است. در مثال بالا، کد کلاس A خصوصیتی با نام Item۱ و Item۲ تعریف می کنیم:

```
public class A
{
    public string Item1 { get; set; }
    public string Item2 { get; set; }
}
```

به دلیل اینکه کلاس B از کلاس A مشتق شده است، می توانیم از خصوصیت های Item۱ و Item۲ برای کلاس B استفاده کنیم:

```
B obj = new B();
```

```
obj.Item1 = "Hossein Ahmadi";
obj.Item2 = "ITPro.ir";
```

حال، کلاس سومی تعریف می کنیم با نام C. این کلاس نیز از کلاس A مشتق می شود:

```
public class C : A
{
}
```

زمانی که شئ ای از کلاس C بسازیم، میبینیم که خصوصیات Item۱ و Item۲ برای این شئ کلاس C نیز وجود دارند، در حقیقت ما این خصوصیات را تنها یکبار در کلاس A تعریف کردیم و با قابلیت وراثت از این کدها برای کلاس های B و C مجدداً استفاده کردیم. زمانی که کلاسی یک کلاس والد مشتق می شود، علاوه بر اینکه دارای خصوصیات و رفتارهای کلاس های والد می باشد، می توان برای کلاس فرزند خصوصیات و رفتارهای جدید تعریف کرد. کلاس C را که در بالا تعریف کردیم به صورت زیر تغییر می دهیم:

```
public class C : A
{
    public string Item3 { get; set; }
}
```

حال زمانی که ما شئ ای از روی کلاس C بسازیم علاوه بر خصوصیت های Item۱ و Item۲ که در کلاس A تعریف شده اند، به خصوصیت دیگری نیز نام Item۳ که در کلاس C تعریف شده دسترسی خواهیم داشت:

```
var instanceOfC = new C();
instanceOfC.Item1 = "Hossein Ahmadi";
instanceOfC.Item2 = "ITPro.ir";
instanceOfC.Item3 = "C# Course";
```

وراثت در زبان سی شارپ، به صورت درختی می باشد، یعنی زمانی که کلاس C از کلاس A مشتق شد می توان کلاس D را نوشت که از کلاس C مشتق شده است:

```
public class D : C
{
    public string Item4 { get; set; }
}
```

در مثال بالا، کلاس D علاوه بر خصوصیات کلاس A و کلاس C خصوصیات مربوط به خودش را نیز شامل می شود. در حقیقت زنجیره وراثت را در این مثال مشاهده می کنید. در مثال های بالا، کلاس ها تنها شامل Property بودند، زمانی که شما برای کلاسی یک رفتار تعریف می کنید، کلاس های فرزند آن رفتار را نیز به ارث می برند:

```
public class A
{
    public string Item1 { get; set; }
    public string Item2 { get; set; }
```

```
    public void PrintItem1()
    {
        Console.WriteLine(Item1);
    }
}
```

حال شئ ای از کلاس D می سازیم و رفتار PrintItem۱ را صدا می زنیم:

```
var obj = new D();
d.PrintItem1();
```

در قسمت های قبلی دیدیم که کلاس D از کلاس C مشتق شده است و خود کلاس C از کلاس A. پس کلیه خصوصیات و رفتارهای کلاس A برای سطوح پایین تر وراثت قابل دسترس هستند.

کلمه کلیدی base

در قسمت های قبلی، در مورد کلمه کلیدی this توضیح دادیم و گفتیم که این کلمه کلیدی به شئ ای اشاره می کند که از روی کلاس ساخته شده. کلمه کلیدی دیگری وجود دارد با نام base که اشاره به کلاس والد دارد. برای مثال، کلاس B را به صوورت زیر تغییر می دهیم:

```
public class B : A
{
    public void PrintParentItems()
    {
        Console.WriteLine(base.Item1 + " " + base.Item2);
    }
}
```

در مثال بالا، کلمه کلیدی base به کلیه اعضای والد اشاره می کند. زمانی که شـما از کلمه کلیدی base داخل کلاس استفاده می کنید، تنها اعضای کلاس والد به شـما نمایش داده شده و اعضای کلاس فرزند به شـما نمایش داده نمی شوند. در قسـمت های بعدی با کاربردهای دیگر کلمه base آشنا می شویم.

تبدیل کلاس های مشتق شده به کلاس والد

زمانی که شما از روی یک کلاس، شئ ای می سازید باید نوع آن کلاس را مشخص کنید یا از کلمه کلیدی var استفاده کنید:

```
B obj = new B();
```

زمانی که کلاسـی از یک شئ مشتق شده باشد، می توان هنگام تعریف شئ از روی آن کلاس، نوع متغیر را به جای خود کلاس، کلاس وارد قرار داد. برای مثال، در مثال زیر ما یک شئ از روی کلاس C می سازیم:

```
A obj = new C();
```

دقت کنید که نوع متغیر obj را از نوع A در نظر گرفتیم، اما شئ ای از نوع C داخل آن ریختیم. دلیل این امر آن است که کلاس C از کلاس A مشتق شده و به نوعی قابل تبدیل به کلاس A می باشد. اما به این نکته توجه داشته باشید، زمانی که نوع متغیر را از نوع کلاس والد در نظر می گیریم، هنگام اسـتفاده از شـئ ساخته شده، تنها خصوصیات و رفتارهایی هایی قابل استفاده هستند که در کلاس والد تعریف شده

اند. به عنوان مثال، کد زیر صـحیح نمی باشد، به این خاطر که Item۳ داخل کلاس C تعریف شده و ما تنها به Item۱ و Item۲ که داخل A تعریف شده اند دسترسی داریم.

```
A instanceOfC = new C();
instanceOfC.Item3 = "C# Course";
```

یکی از مهمترین کاربردهای استفاده از نوع داده کلاس والد، مبحث Polymorphism می باشد که در بخش های بعدی با این مفهوم بیشتر آشنا خواهیم شد.

کلاس Object

در کتابخانه دات نت، کلاسی وجود دارد به نام Object یا شئ. در دات نت، کلیه نوع های داده و کلاس ها، چه آنهایی که به صورت دستی می نویسیم و چه آنهایی که در کتابخانه دات نت وجود دارند، از کلاس object مشتق شده اند، به جز کلاس هایی که برای آنها کلاس والد را مشخص کرده ایم. حتی کلاس هایی که برای آنها کلاس والد مشخص شده، باز هم شاخه اصلی زنجیره وراثت به کلاس object ختم می شود. پس به این صورت می گوییم که کلاس object، کلاس پایه ای برای کلیه کلاس های دات نت می باشد. یکسـری رفتارها برای کلاس Object تعریف شده اند که در تمامی کلاس ها در دسترس هستند، زیرا کلیه کلاس ها از کلاس object مشتق شده اند. این رفتارها به شرح زیر می باشند:

۱. Equals: این رفتار بررسی می کند که دو شئ با یکدیگر برابر هستند یا خیر.
۲. GetHashCode: این رفتار عددی را برمی گرداند که شناسه شئ ایجاد شده می باشد.
۳. GetType: نوع یا Type شئ را بر میگرداند. این متد را در بخش Reflection بیشتر بررسی خواهیم کرد.
۴. ToString: زمانی که این رفتار را برای یک شـئ صـدا می زنیـد، رشـته ای مرتبط با آن شـئ را بر میگرداند که به صورت پیش فرض Type Name یا نام نوع آن کلاس را بر میگرداند. در بخش Polymorphism با این متد بیشتر آشنا می شوید.

گفتیم زمان تعریف کردن یک شئ، نوع داده والد را به جای خود کلاس برای متغیر در نظر گرفت:

```
object number = 12;
object name = "Hossein Ahmadi";
object instance = new A();
```

همینطور که مشاهـده می کنیـد فرقی نمی کنـد مقدار متغیر چه باشد، هر مقداری را می توان در متغیر از نوع object ذخیره کرد. در این قسـمت سـعی کردیم مقـدماتی از مبحث وراثت را با هم مرور کنیم. در بخش بعدی بحث وراثت را با بررسـی مفهوم Polymorphism ادامه خواهیم داد.

نویسنده : حسین احمدی

منبع : جزیره برنامه نویسی وب سایت توسینسو

مطلب اصلی