زنگ سی‌شارپ – قسمت سی و نهم

نوشته‌ی مسعود درویشیان

لینک مستقیم این مطلب در وب‌تارگت

در قسمت قبل تا حدودی با ارث‌بری آشنا شدید. در این قسمت به‌ادامه‌ی مبحث ارث‌بری می‌پردازیم.

به نحوی‌دسترسی به‌صورت protected در مثال زیر توجه کنید:

```
using System;
class B
{
    protected int i, j; // private to B, but accessible by D
    public void Set(int a, int b)
    {
        i = a;
        j = b;
    }
    public void Show()
    {
        Console.WriteLine(i + " " + j);
    }
}
class D : B
{
    int k; // private

    // D can access B's i and j
    public void Setk()
    {
        k = i * j;
    }
    public void Showk()
    {
        Console.WriteLine(k);
    }
}
class ProtectedDemo
{
    static void Main()
    {
        D ob = new D();

        ob.Set(2, 3);
        ob.Show();

        ob.Setk();
        ob.Showk();
```

```
    }
}
```

در این مثال، به‌دلیل این‌که کلاس D از B ارث‌بری کرده است و همچنین به‌دلیل این‌که i و j به‌صورت protected تعریف شده‌اند، متد SetK() می‌تواند به آن‌ها دسترسی داشته باشد. توجه داشته باشد اگر i و j به‌صورت private تعریف شوند، کلاس D به آن‌ها دسترسی نخواهد داشت.

این‌که چند مرحله ارث‌بری انجام شود، مهم نیست. مانند public و private، دسترسی protected در این مراحل ارث‌بری، همواره برای یک عضو، protected می‌ماند. از این‌رو، هنگامی‌که یک derived class به‌عنوان یک base class برای یک derived class دیگر مورد استفاده قرار می‌گیرد، تمام اعضای protected در اولین base class که توسط اولین derived class ارث‌بری شده‌اند، در دومین derived class نیز به‌صورت protected ارث‌بری می‌شوند.

با وجود این‌که دسترسی protected بسیار مفید است، در همه‌ی موقعیت‌ها نباید از آن استفاده کرد و فقط باید زمانی از آن بهره برد که می‌خواهیم یک عضو در سلسه‌مراتب ارث‌بری قابل دسترس، و در خارج از این سلسله‌مراتب، غیر قابل دسترسی باشد.

Constructor ها و ارث‌بری

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

به مثال زیر توجه کنید:

```
using System;
```

```
class A
{
    protected int ID { get; set; }
    protected int Number { get; set; }
}
class B : A
{
    protected string Name;

    public B(int id, int number, string name)
    {
        ID = id;
        Number = number;
        Name = name;
    }

    public void Show()
    {
        Console.WriteLine(Name + ": " + ID + " - " + Number);
    }
}
class IndehritDemo
{
    static void Main()
    {
        B ob1 = new B(180, 1, "Filipe");
        ob1.Show();

        B ob2 = new B(190, 2, "Cevat");
        ob2.Show();
    }
}
```

در این‌جا، constructor در کلاس B، متغیرهایی که از کلاس A به ارث برده است را مقداردهی می‌کند.

هنگامی‌که هم در base class و هم در derived class، constructor تعریف شده باشد، روند کار متفاوت‌تر خواهد بود زیرا constructor های base class و derived class هردو بایستی اجرا شوند. در این موارد بایستی از کلمه‌ی کلیدی base، که دو کاربرد دارد، استفاده کنید. کاربرد اول آن برای صدا زدن constructor در base class است. کاربرد دوم آن برای دسترسی به اعضایی از base class است که به دلیل تشابه اسمی در derived class قابل مشاهده نیستند.

فراخوانی constructor های base class

یک derived class می‌تواند constructor ای که در base class اش تعریف شده است را از طریق گسترش دادن فرم constructor در derived class و کلمه‌ی کلیدی base، صدا بزند.

فرم کلی تعریف گسترش یافته‌ی آن به‌شکل زیر است:

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

در این‌جا، arg-list مشخص‌کننده‌ی argument های مورد نیاز constructor در base class است. به نحوه‌ی قرار گرفتن colon نیز توجه داشته باشید. به‌منظور این‌که با کاربرد base آشنا شوید به مثال زیر توجه کنید:

```
using System;
class A
{
    private int i, j;

    public A(int a, int b)
    {
        i = a;
        j = b;
    }

    public void ShowA()
    {
        Console.Write(i + " " + j);
    }
}
class B : A
{
    private int k;

    public B(int a, int b, int c) : base(a, b)
    {
        k = c;
    }

    public void ShowB()
    {
        ShowA();
        Console.WriteLine(" " + k);
    }
}
class InheritanceDemo
{
    static void Main()
    {
        B b = new B(2, 4, 6);

        b.ShowB();
    }
}
```

به نحوه‌ی تعریف constructor در کلاس B توجه کنید:

```
public B(int a, int b, int c) : base(a, b) {
```

در این‌جا، B() با صدا زدن base همراه با پارامترهای a و b موجب می‌شود تا constructor کلاس A اجرا شده و متغیرهای i و j را مقداردهی کند. B دیگر خودش این متغیرها را مقداردهی نکرده و تنها چیزی که (k) مربوط به خودش است را مقداردهی می‌کند. Constructor به هر شکلی در base class تعریف شده باشد، می‌تواند توسط کلمه‌ی کلیدی base صدا زده شود و آن constructor ای اجرا خواهد شد که با argument ها تطابق داشته باشد.

به مثال زیر دقت کنید:

```
using System;
class A
{
    private int i, j;

    public A(int a, int b)
    {
        i = a;
        j = b;
    }

    public A(int a)
    {
        i = j = a;
    }
    public void ShowA()
    {
        Console.Write(i + " " + j);
    }
}
class B : A
{
    private int k;

    public B(int a, int b, int c) : base(a, b) {
        k = c;
    }

    public B(int a) : base(a)
    {
        k = 5;
    }

    public void ShowB()
    {
        ShowA();
```

```
        Console.WriteLine(" " + k);
    }
}
class InheritanceDemo
{
    static void Main()
    {
        B b1 = new B(2, 4, 6);
        b1.ShowB();

        B b2 = new B(1);
        b2.ShowB();
    }
}
```

در این مثال، base class شامل دو constructor بوده که constructor دوم آن شامل یک argument است. بنابراین هنگامی‌که در کلاس B از base استفاده می‌کنید و یک argument به آن می‌دهید، در A آن constructor که یک parameter دارد اجرا خواهد شد.

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

تا این‌جا با اولین کاربرد base آشنا شدید. کاربرد دوم آن برای دسترسی به اعضایی از base class است که به دلیل تشابه اسمی در derived class قابل مشاهده نیستند. derived class می‌تواند عضوی را تعریف کند که مشابه نام یکی از اعضای base class اش باشد. هنگامی‌که چنین اتفاقی افتد، آن عضو base class در derived class دیده نمی‌شود. درحالی‌که این مورد از لحاظ تکنیکی در سی‌شارپ خطا شمرده نمی‌شود، کامپایلر یک پیغام هشدار به شما داده و از این که یکی از اعضا دیده نمی‌شود، شما را باخبر می‌سازد. اگر قصد شما این باشد تا باعث دیده نشدن یکی از اعضای base class شوید، به‌منظور رفع هشدار کامپایلر، در derived class در تعریف آن عضو از کلمه‌ی‌کلیدی new استفاده کنید. دقت داشته باشید که این‌طور استفاده از new با آن حالتی که از آن برای ساختن شیء استفاده می‌کردید، متفاوت است.

به مثال زیر توجه کنید:

```
using System;
class A
{
    public int i = 0;
}
class B : A
{
    new int i; // this i hides the i in A
    public B(int b)
    {
        i = b; // i in B
    }
    public void Show()
    {
        Console.WriteLine("i in derived class: " + i);
    }
}
class NameHiding
{
    static void Main()
    {
        B ob = new B(2);
        ob.Show();
    }
}
```

به نحوه‌ی استفاده‌ی new توجه کنید:

```
new int i; // this i hides the i in A
```

این خط کد به کامپایلر می‌گوید که شما می‌دانید متغیر جدیدی به اسم i ساخته شده است و باعث دیده نشدن i در base class خواهد شد. اگر new را از کد بالا حذف کنید، پیغام هشدار compiler را مشاهده خواهید کرد. از آن‌جا که B متغیر i خودش را تعریف می‌کند و باعث دیده نشدن i در A می‌شود، پس از اجرای متد Show()، مقدار متغیر i که متعلق به B است نمایش داده خواهد شد، نه مقدار i در متغیر A.

استفاده از base برای دسترسی به اعضایی که دیده نمی‌شوند

دومین استفاده از base تا حدودی شبیه به this است با این تفاوت که base همیشه به base class رجوع می‌کند. نحوه‌ی استفاده از base به شکل زیر است:

```
base.member
```

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

به مثال زیر توجه کنید:

```
using System;
class A
{
    public int i = 0;
}
class B : A
{
    new int i; // this i hides the i in A
    public B(int a, int b)
    {
        base.i = a; // i in A
        i = b; // i in B
    }
    public void Show()
    {
        Console.WriteLine("i in base class: " + base.i);
        Console.WriteLine("i in derived class: " + i);
    }
}
class UncoverName
{
    static void Main()
    {
        B ob = new B(1, 2);
        ob.Show();
    }
}
```

خروجی:

```
i in base class: 1
i in derived class: 2
```

همان‌طور که می‌بینید، با وجود این‌که متغیر i در B باعث دیده نشدن متغیر i در A می‌شود، با استفاده از کلمه‌ی کلیدی base توانستیم در کلاس B به آن دسترسی داشته باشیم.

این مورد درباره‌ی متدها نیز صدق می‌کند. برای مثال اگر در کد قبل، هردو کلاس متدی به اسم Show() تعریف کنند با استفاده از base می‌توانید در derived class به متد Show() در base class دسترسی داشته باشید:

```
using System;
class A {
    public int i = 0;
    public void Show()
    {
        Console.WriteLine("i in base class: " + i);
    }
}
class B : A
{
    new int i; // this i hides the i in A

    public B(int a, int b)
    {
        base.i = a; // i in A
        i = b; // i in B
    }

    new public void Show()
    {
        base.Show(); // this calls Show() in A

        // this displays the i in B
        Console.WriteLine("i in derived class: " + i);
    }
}
class UncoverName
{
    static void Main()
    {
        B ob = new B(1, 2);
        ob.Show();
    }
}
```

خروجی:

```
i in base class: 1
i in derived class: 2
```

همان‌طور که می‌بینید، base.Show() باعث شده که متد Show() در base class صدا زده شود. به نحوه‌ی استفاده از new در این مثال، دقت کنید. new در این‌جا باعث می‌شود به کامپایلر بفهمانید که شما می‌دانید متد Show() در کلاس B باعث دیده نشدن Show() در کلاس A می‌شود.
