زنگ سی‌شارپ – قسمت چهل‌ام

نوشته‌ی مسعود درویشیان

لینک مستقیم این مطلب در وب‌تارگت

آیا سی‌شارپ یک زبان **strongly typed** است یا **weakly typed**؟!

احتمالاً این اصطلاحات را بین برنامه‌نویسان زیاد شنیده‌اید اما در واقع این اصطلاحات بی‌معنی هستند و بهتر است که از گفتن آن اجتناب ورزید. ویکی‌پدیا معانی متفاوتی را برای strongly typed لیست کرده که تعدادی از آن‌ها یکدیگر را نقض می‌کنند. هر زمان که دو نفر در مورد strongly typed و weakly typed صحبت می‌کنند احتمالاً معانی متفاوتی از این اصطلاحات در ذهن‌شان نقش بسته است بنابراین هردو برداشت متفاوتی از این اصطلاحات دارند و از طرفی گمان می‌کنند که هردو منظورشان یکسان است. در واقع یک زبان نسبت به یک زبان دیگر، در صورتی‌که در type system (type system مجموعه‌ای از قوانین در زبان برنامه‌نویسی است که در صورت عدم پیروی از آن‌ها با خطا مواجه خواهید شد) محدودیت بیشتری را اعمال کند، بیشتر strongly typed است و از طرف دیگر، آن زبان که محدودیت کمتری در type system در مقایسه با یک زبان دیگر دارد، نسبت به آن زبان، بیشتر weakly typed است.

بنابراین گفتن این‌که آیا سی‌شارپ (یا هر زبان دیگر) strongly typed یا weakly typed است، صحیح نیست زیرا این موضوع بستگی به زبانی که مقایسه را با آن انجام می‌دهید و همچنین بستگی به دید کسی که در این مورد صحبت می‌کند، دارد و عملاً بهتر است به‌جای استفاده از این اصطلاحات از محدودیت‌ها و ویژگی‌های type system یک زبان صحبت کرد.

ادامه‌ی مبحث ارث‌بری

تا این‌جا ما تنها از یک base class و یک derived class استفاده می‌کردیم اما شما می‌توانید در سلسله‌مراتب ارث‌بری هر تعداد که می‌خواهید base class و derived class داشته باشید. همان‌طور که قبلاً ذکر شد، یک derived class می‌تواند برای یک کلاس دیگر به‌عنوان base class در نظر گرفته شود.

برای مثال اگر سه کلاس A، B و C داشته باشید و C از B ارث‌بری کند و B هم A آن‌گاه کلاس C به همه‌ی عناصر A و B دسترسی دارد.

به مثال زیر توجه کنید:

```
using System;
class A
{
    protected int Width;
    protected int Height;

    public A(int w, int h)
    {
        Width = w;
        Height = h;
    }
}
class B : A
{
    protected string Style;

    public B(string s, int w, int h)
        : base(w, h)
    {
        Style = s;
    }
}
class C : B
{
    protected string Color;

    public C(string c, string s, int w, int h)
        : base(s, w, h)
    {
        Color = c;
    }

    public void Show()
    {
        Console.WriteLine("{0}, {1}: Width: {2} - Height: {3}", Style, Color, Width, Height);
    }
}
class MyShape
{
    static void Main()
    {
        C ob = new C("Red", "Rectangular", 500, 200);
        ob.Show();
    }
}
```

همان‌طور که می‌بینید، متد Show() در کلاس C به همه‌ی عناصر کلاس A و B دسترسی دارد و دلیل آن این است که C از B و B از A ارث‌بری کرده است. با این‌که C مستقیماً از A ارث‌بری نکرده اما به‌دلیل وجود این زنجیره‌ی ارث‌بری، کلاس C که در آخر این زنجیره قرار دارد همه‌ی عناصر زنجیره را به ارث می‌برد.

چه زمانی constructor ها فراخوانی می‌شوند؟

شاید در مبحث ارث‌بری و سلسله‌مراتب ارث‌بری، این سوال برای‌تان به‌وجود آمده باشد، هنگامی‌که یک شیء از derived class ساخته می‌شود کدام constructor زودتر اجرا خواهد شد؟ درواقع در سلسله‌مراتب ارث‌بری، constructor ها به ترتیب از ابتدای زنجیره‌ی ارث‌بری اجرا خواهند شد.

به مثال زیر توجه کنید:

```
using System;
class A
{
    public A()
    {
        Console.WriteLine("Constructing A.");
    }
}
class B : A
{
    public B()
    {
        Console.WriteLine("Constructing B.");
    }
}
class C : B
{
    public C()
    {
        Console.WriteLine("Constructing C.");
    }
}
class InheritanceDemo
{
    static void Main()
    {
        C ob = new C();
    }
}
```

خروجی:

```
Constructing A.
Constructing B.
Constructing C.
```

همان‌طور که در خروجی می‌بینید، constructor ها از ابتدای زنجیره تا انتها، به ترتیب، اجرا شده‌اند. دلیل آن این است که base class هیچ اطلاعی از derived class ندارد و هرگونه مقداردهی و اجرای مورد نیاز در base class جدا از derived class است و از طرفی به‌دلیل این‌که derived class عناصر base class را به ارث می‌برد، نیاز است که base class مقداردهی‌ها و اجراهای‌اش را انجام داده باشد. از این‌رو base class بایستی زودتر اجرا شود.

در سی‌شارپ، reference variable یک کلاس در حالت عادی نمی‌تواند به اشیای کلاس‌های دیگر رجوع کند. برای مثال به برنامه‌ی زیر که دو کلاس در آن تعریف شده است توجه کنید:

```
using System;
class A
{
    int x;
    public A(int x)
    {
        this.x = x;
    }
}
class B
{
    int x;
    public B(int x)
    {
        this.x = x;
    }
}
class IncompatibleRef
{
    static void Main()
    {
        A a = new A(10);
        A a2;
        B b = new B(5);

        a2 = a; // OK, both of same type

        // a2 = b; // Error, not of same type
    }
}
```

اگرچه کلاس A و B محتوای یکسانی دارند، اما در سی‌شارپ این اجازه را ندارید که یک reference از نوع B را به متغیری از نوع A اختصاص دهید زیرا این دو، type متفاوتی هستند (type system زبان سی‌شارپ این اجازه را به شما نمی‌دهد). بنابراین اگر در برنامه‌ی بالا، خط کد زیر را از حالت comment خارج کنید با خطای compile-time مواجه خواهید شد:

```
// a2 = b; // Error, not of same type
```

در حالت کلی، reference variable یک شیء تنها می‌تواند به اشیایی از جنس خودش رجوع کند اما سی‌شارپ در این مورد یک استثنا دارد و آن این است که reference variable یک base class می‌تواند به تمام اشیایی که از base class مشتق شده‌اند رجوع کند. این استثنا به این دلیل صحیح است که یک نمونه از derived class، نمونه‌ای از base class را در خود دارد و از این‌رو base class می‌تواند به آن رجوع کند.

به مثال زیر توجه کنید:

```
using System;
class A
{
    public int a;
    public A(int a)
    {
        this.a = a;
    }
}
class B : A
{
    public int b;
    public B(int a, int b)
        : base(a)
    {
        this.b = b;
    }
}
class BaseRef
{
    static void Main()
    {
        A aOb = new A(5);
        B bOb = new B(3, 6);

        A aOb2;

        aOb2 = aOb; // OK, both of same type
        Console.WriteLine("aOb2.a: " + aOb2.a);

        aOb2 = bOb; // OK because B is derived from A
```

```
        Console.WriteLine("aOb2.a: " + aOb2.a);

        // A references know only about A members
        aOb2.a = 10; // Ok
        // aOb2.b = 20 // Error! A doesn't have a b member
    }
}
```

در این برنامه، B از A ارث‌بری کرده است بنابراین خط کد زیر:

```
aOb2 = bOb; // OK because B is derived from A
```

صحیح است زیرا یک base class reference (که در این‌جا aOb2 است) می‌تواند به یک derived class object رجوع کند. هنگامی‌که به base class reference، یک refenence از شیء derived class اختصاص می‌دهید، در نهایت شما فقط به آن بخش که در base class مشخص شده است دسترسی دارید. به همین دلیل است که aOb2 با اینکه به شیء B رجوع می‌کند، نمی‌تواند به b دسترسی داشته باشد. این امر منطقی است زیرا base class هیچ اطلاعی ندارد که derived class چه عناصر دیگری را افزوده است. به همین دلیل نیز، آخرین خط برنامه comment شده است.

یکی از موقعیت‌هایی که derived class reference به base class variable اختصاص می‌یابد زمانی است که constructor ها در سلسله‌مراتب ارث‌بری صدا زده می‌شوند. همان‌طور که می‌دانید یک کلاس می‌تواند constructor ای داشته باشد که یک شیء از جنس کلاس خودش را به‌عنوان پارامتر بگیرد. کلاس‌هایی که از این کلاس ارث‌بری کرده باشند می‌توانند از ویژگی ذکر شده در مثال قبل استفاده کنند.

به مثال زیر دقت کنید:

```
using System;
class TwoDShape
{
    double pri_width;
    double pri_height;

    public TwoDShape()
    {
        Width = Height = 0.0;
    }

    public TwoDShape(double w, double h)
    {
        Width = w;
        Height = h;
    }
```

```
    public TwoDShape(double x)
    {
        Width = Height = x;
    }

    // Construct a copy of a TwoDShape object.
    public TwoDShape(TwoDShape ob)
    {
        Width = ob.Width;
        Height = ob.Height;
    }

    public double Width
    {
        get { return pri_width; }
        set { pri_width = value < 0 ? -value : value; }
    }
    public double Height
    {
        get { return pri_height; }
        set { pri_height = value < 0 ? -value : value; }
    }
    public void ShowDim()
    {
        Console.WriteLine("Width and height are " +
        Width + " and " + Height);
    }
}
// A derived class of TwoDShape for triangles.
class Triangle : TwoDShape
{
    string Style;

    public Triangle()
    {
        Style = "null";
    }

    public Triangle(string s, double w, double h)
        : base(w, h)
    {
        Style = s;
    }

    public Triangle(double x)
        : base(x)
    {
        Style = "isosceles";
    }

    // Construct a copy of a Triangle object.
    public Triangle(Triangle ob)
        : base(ob)
    {
        Style = ob.Style;
    }

    public double Area()
```

```
    {
        return Width * Height / 2;
    }

    public void ShowStyle()
    {
        Console.WriteLine("Triangle is " + Style);
    }
}
class Shapes7
{
    static void Main()
    {
        Triangle t1 = new Triangle("right", 8.0, 12.0);

        // Make a copy of t1.
        Triangle t2 = new Triangle(t1);

        Console.WriteLine("Info for t1: ");
        t1.ShowStyle();
        t1.ShowDim();
        Console.WriteLine("Area is " + t1.Area());

        Console.WriteLine();

        Console.WriteLine("Info for t2: ");
        t2.ShowStyle();
        t2.ShowDim();
        Console.WriteLine("Area is " + t2.Area());
    }
}
```

خروجی:

```
Info for t1:
Triangle is right
Width and height are 8 and 12
Area is 48

Info for t2:
Triangle is right
Width and height are 8 and 12
Area is 48
```

در این برنامه، t2 از t1 ساخته شده است از این‌رو با آن یکسان است.

به constructor کلاس Triangle دقت کنید:

```
public Triangle(Triangle ob) : base(ob) {
    Style = ob.Style;
}
```

این constructor یک شیء از نوع Triangle دریافت می‌کند و از طریق base، این شیء را به constructor کلاس TwoDShape می‌فرستد:

```
public TwoDShape(TwoDShape ob) {
    Width = ob.Width;
    Height = ob.Height;
}
```

نکته این‌جا است که TwoDShape انتظار دارد یک شیء از جنس TwoDShape دریافت کند اما ()Triangle یک شیء از نوع Triangle به آن داده است. همان‌طور که گفته شد، علت این‌که این کد کار می‌کند این است که base class reference می‌تواند به یک شیء derived class رجوع کند. بنابراین هیچ مشکلی ندارد که به ()TwoDShape یک reference بفرستید که این reference به شیءای رجوع می‌کند که از کلاس TwoDShape ارثبری کرده است. ()TwoDShape تنها آن بخشی از شیء derived class را می‌سازد که عناصر کلاس TwoDshape هستند و بقیه‌ی قسمت‌های شیء derived class به آن مربوط نیست.

Overriding و متدهای Virtual

Virtual method، متدی است که با کلمه‌ی‌کلیدی virtual در base class تعریف می‌شود. Virtual method به‌شکلی است که می‌توانید آن را در derived class مجدداً تعریف کنید. از این‌رو، هر derived class می‌تواند نسخه‌ی اختصاصی خودش را از virtual method داشته باشد. همان‌طور که گفته شد، virtual method در base class با کلمه‌ی‌کلیدی virtual تعریف می‌شود. هنگامی‌که یک virtual method در derived class مجدداً تعریف می‌شود، باید از override modifier استفاده کنید بنابراین پروسه تعریف مجدد virtual method در derived class را method overriding می‌نامیم. هنگام override کردن یک متد، باید اسم متد، return type و پارامترهای آن را مطابق با virtual method بنویسیم.

به مثال زیر توجه کنید:

```
using System;
class A
{
```

```
    public virtual void Hello()
    {
        Console.WriteLine("Hello() in A class.");
    }
}
class B : A
{
    public override void Hello()
    {
        Console.WriteLine("Hello() in B class.");
    }
}
class C : A
{
    public override void Hello()
    {
        Console.WriteLine("Hello() in C class.");
    }
}
class MyClass
{
    static void Main()
    {
        A a = new A();
        B b = new B();
        C c = new C();

        a.Hello();
        b.Hello();
        c.Hello();
    }
}
```

خروجی:

```
Hello() in A class.
Hello() in B class.
Hello() in C class.
```

ادامه‌ی بحث virtual method و overriding را در قسمت بعد دنبال کنید.