



## زنگ سی‌شارپ – قسمت چهارم و یکم

نوشته‌ی مسعود درویشیان  

[لینک مستقیم این مطلب در وب‌تارگت](#)

در قسمت قبل اندکی با virtual method آشنا شدید. همان‌طور که ذکر شد، پروسه‌ی تعریف مجدد virtual method در derived class را method overriding می‌نامند.

همان‌طور که گفته شد، virtual method در base class با کلمه‌ی کلیدی virtual تعریف می‌شود. هنگامی که یک virtual method در derived class مجدداً تعریف می‌شود، باید از override modifier استفاده کنید و هنگام override کردن یک متد، باید اسم متد، return type و پارامترهای آن را مطابق با virtual method بنویسید.

به مثال زیر توجه کنید:

```
using System;
class Human
{
    public virtual void SayHello(string name)
    {
        Console.WriteLine("SayHello in base class");
    }
}
class Man : Human
{
    public override void SayHello(string name)
    {
        Console.WriteLine("Hello " + name);
    }
}
class OverrideDemo
{
    static void Main()
    {
        Man ob = new Man();
        ob.SayHello("Stefan");
    }
}
```

متد SayHello() در کلاس Human به‌صورت virtual تعریف شده است و یک پارامتر دارد. در کلاس Man که از Human ارث‌بری کرده، متد مربوطه override شده است. همان‌طور که می‌بینید این متد در کلاس Man از override

modifier استفاده کرده است. دقت کنید که override کردن یک متد ضروری نیست و در صورتی که متدی را override نکنید، آن نسخه از متد که در base class وجود دارد اجرا خواهد شد.

به مثال زیر توجه کنید:

```
using System;
class A
{
    public virtual void SayHello()
    {
        Console.WriteLine("SayHello in base class");
    }
}
class B : A
{
    public override void SayHello()
    {
        Console.WriteLine("SayHello in B");
    }
}
class C : A
{
    // this class doesn't override SayHello()
}
class OverrideDemo
{
    static void Main()
    {
        A a = new A();
        B b = new B();
        C c = new C();

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

خروجی:

```
SayHello in base class
SayHello in B
SayHello in base class
Press any key to continue . . .
```

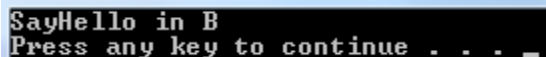
در این جا، کلاس C متد SayHello() را override نمی کند بنابراین زمانی که متد SayHello() از طریق شیء C فراخوانی می شود، متد SayHello() در کلاس A اجرا خواهد شد.

هنگامی که از سلسله مراتب ارث‌بری استفاده می‌کنید، اگر یک derived class، یک virtual method را override نکند، به طرف ابتدای زنجیره‌ی ارث‌بری حرکت کنید، اولین override آن متد که دیده شود اجرا خواهد شد.

به مثال زیر توجه کنید:

```
using System;
class A
{
    public virtual void SayHello()
    {
        Console.WriteLine("SayHello in base class");
    }
}
class B : A
{
    public override void SayHello()
    {
        Console.WriteLine("SayHello in B");
    }
}
class C : B
{
    // this class doesn't override SayHello()
}
class D : C
{
    // this class doesn't override SayHello()
}
class OverrideDemo
{
    static void Main()
    {
        D d = new D();
        d.SayHello();
    }
}
```

خروجی:



```
SayHello in B
Press any key to continue . . . _
```

همان‌طور که در مثال بالا می‌بینید، کلاس D از C و کلاس C از B و کلاس B از A ارث‌بری کرده است. کلاس D و C متد SayHello() را override نکرده‌اند اما کلاس B این متد را override کرده است. بنابراین هنگامی که از طریق شیء کلاس D این متد را صدا می‌زنید، در زنجیره‌ی ارث‌بری اولین کلاسی که متد SayHello() را فراخوانی کرده است کلاس B است. بنابراین همان‌طور که در خروجی می‌بینید، نسخه‌ی override شده‌ی این متد، موجود در کلاس B، اجرا خواهد شد.

شد. قابل ذکر است که properties و indexers نیز می‌توانند با استفاده از virtual و override به همین شکل مورد استفاده قرار گیرند.

## علت استفاده از متدهای override شده چیست؟

متدهای override شده به سی‌شارپ اجازه می‌دهند تا از ویژگی runtime polymorphism بهره ببرد. Polymorphism توانایی ساخت متدهایی است که با توجه به موقعیت، می‌توانند اجرای متفاوتی داشته باشند. برای مثال شما می‌تواند هم به ماشین و هم به سگ غذا بدهید اما خوب می‌دانید که معنای غذا دادن به این دو کاملاً متفاوت است. Polymorphism به این دلیل برای برنامه‌نویسی شی‌گرا اهمیت دارد که به یک کلاس کلی، اجازه می‌دهد متدهایی داشته باشد که در همه‌ی کلاس‌های مشتق شده از آن کلاس، مشترک هستند. این درحالی است که به derived class ها این اجازه را می‌دهد تا هرطور که می‌خواهند آن متدها را اجرا کنند و در صورت نیاز، نحوه‌ی اجرای آن متدها را تغییر دهند. متدهای override شده، روش دیگری برای اجرای این جنبه از polymorphism که می‌گوید "one interface, multiple methods" هستند.

به مثال زیر توجه کنید:

```
// Use virtual methods and polymorphism.
using System;
class TwoDShape
{
    double pri_width;
    double pri_height;

    // A default constructor.
    public TwoDShape()
    {
        Width = Height = 0.0;
        name = "null";
    }

    // Parameterized constructor.
    public TwoDShape(double w, double h, string n)
    {
        Width = w;
        Height = h;
        name = n;
    }
}
```

```

// Construct object with equal width and height.
public TwoDShape(double x, string n)
{
    Width = Height = x;
    name = n;
}

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

// Properties for Width and 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 string name { get; set; }

public void ShowDim()
{
    Console.WriteLine("Width and height are " +
        Width + " and " + Height);
}
public virtual double Area()
{
    Console.WriteLine("Area() must be overridden");
    return 0.0;
}
}

// A derived class of TwoDShape for triangles.
class Triangle : TwoDShape
{
    string Style;

    // A default constructor.
    public Triangle()
    {
        Style = "null";
    }

    // Constructor for Triangle.
    public Triangle(string s, double w, double h) :
        base(w, h, "triangle")
    {
        Style = s;
    }
}

```

```

// Construct an isosceles triangle.
public Triangle(double x)
    : base(x, "triangle")
{
    Style = "isosceles";
}

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

// Override Area() for Triangle.
public override double Area()
{
    return Width * Height / 2;
}

// Display a triangle's style.
public void ShowStyle()
{
    Console.WriteLine("Triangle is " + Style);
}
}

// A derived class of TwoDShape for rectangles.
class Rectangle : TwoDShape
{
    // Constructor for Rectangle.
    public Rectangle(double w, double h) :
        base(w, h, "rectangle") { }

    // Construct a square.
    public Rectangle(double x) :
        base(x, "rectangle") { }

    // Construct a copy of a Rectangle object.
    public Rectangle(Rectangle ob) : base(ob) { }

    // Return true if the rectangle is square.
    public bool IsSquare()
    {
        if (Width == Height) return true;
        return false;
    }

    // Override Area() for Rectangle.
    public override double Area()
    {
        return Width * Height;
    }
}

class DynShapes
{
    static void Main()

```

```

{
    TwoDShape[] shapes = new TwoDShape[5];

    shapes[0] = new Triangle("right", 8.0, 12.0);
    shapes[1] = new Rectangle(10);
    shapes[2] = new Rectangle(10, 4);
    shapes[3] = new Triangle(7.0);
    shapes[4] = new TwoDShape(10, 20, "generic");

    for (int i = 0; i < shapes.Length; i++)
    {
        Console.WriteLine("object is " + shapes[i].name);
        Console.WriteLine("Area is " + shapes[i].Area());

        Console.WriteLine();
    }
}

```

خروجی:

```

object is triangle
Area is 48

object is rectangle
Area is 100

object is rectangle
Area is 40

object is triangle
Area is 24.5

object is generic
Area() must be overridden
Area is 0

```

در برنامه‌ی بالا، ابتدا `Area()` به صورت `virtual` در کلاس `TwoDShape` تعریف شده و سپس توسط کلاس‌های `Triangle` و `Rectangle` نیز `override` شده است. در `TwoDShape` می‌بینید که `Area()` فقط به صورت `virtual` تعریف شده است و تنها کاری که انجام می‌دهد این است که اطلاع می‌دهد این متد باید `override` شود. هر `override` از متد `Area()` باید بستگی به شکل شیء‌ای داشته باشد که `derived class` نشان دهنده‌ی آن است. به عنوان مثال اگر شکل مورد نظر مستطیل است نحوه‌ی محاسبه‌ی مساحت آن متناسب با مستطیل خواهد بود و اگر شکل مورد نظر مثلث باشد، نحوه‌ی محاسبه‌ی مساحت آن نیز متناسب با مثلث است. نکته‌ی مهم دیگر برنامه‌ی بالا درون متد `Main()` است. همان‌طور که می‌بینید `shapes` آرایه‌ای از اشیای `TwoDShape` است اما عناصری که در این آرایه قرار دادیم `reference` های `Triangle` و `Rectangle` هستند. همان‌طور که قبلاً ذکر شد، این مورد به این دلیل صحیح است که `TwoDShape` `base class`

reference می‌تواند به derived class object رجوع کند. این برنامه سپس توسط یک حلقه، اطلاعات عناصر موجود در آرایه را نمایش می‌دهد.

## استفاده از کلاس‌های Abstract

گاهی قصد دارید یک base class بسازید که تنها یک فرم کلی را مشخص می‌کند و آن را با تمام کلاس‌های مشتق شده، به اشتراک می‌گذارد و اجازه می‌دهد که خود derived class ها بدنه و جزئیات این فرم کلی را تکمیل کنند. به‌عنوان مثال، این چنین کلاسی ماهیت یک متد را مشخص می‌کند و derived class ها باید این متد را override کنند اما خود base class دیگر نیازی ندارد که برای این متد یک اجرای پیش‌فرض داشته باشد. این حالت ممکن است زمانی رخ دهد که base class نتواند یک اجرای بامعنی برای متد مورد نظر داشته باشد، از این‌رو اجرا را بر عهده‌ی derived class ها می‌گذارد. مانند مثال قبل که متد Area() در کلاس TwoDShape، هیچ‌گونه محاسباتی را انجام نمی‌داد. در چنین مواقعی، می‌توانید مانند مثال قبل به‌سادگی یک پیغام هشدار درون متد قرار دهید اما این روش چندان مناسب نیست و ممکن است در شرایط خاصی مثل debug کردن، مناسب باشد. گاهی ممکن است متدهایی در base class داشته باشید که derived class ها حتماً باید آن‌ها را اجرا کنند، در چنین شرایطی باید از abstract method استفاده کنید.

یک متد abstract با abstract modifier ساخته می‌شود. abstract method بدنه ندارد و از این‌رو درون base class اجرا نخواهد شد. derived class ها حتماً باید این abstract method را override کنند. یک abstract method به‌صورت اتوماتیک virtual نیز است و در واقع نمی‌توانید از virtual و abstract باهم در یک تعریف استفاده کنید.

فرم کلی abstract method به‌شکل زیر است:

```
abstract type name(parameter-list);
```

همان‌طور که می‌بینید، در abstract method به بدنه نیاز ندارید. دقت کنید که abstract modifier را نمی‌توانید برای متدهای static استفاده کنید. properties و indexers نیز می‌توانند abstract باشند.



کلاسی که شامل یک یا بیشتر از یک متد abstract باشد باید به صورت abstract تعریف شود. برای تعریف یک کلاس به صورت abstract کافی است که قبل از کلمه‌ی کلیدی class از abstract modifier استفاده کنید. از آنجا که abstract class نمی‌تواند به طور کامل اجرا شود (به دلیل وجود متدهای abstract که بدنه ندارند)، به همین دلیل نمی‌توانید از abstract class شیء بسازید.

هنگامی که یک derived class از یک abstract class ارث‌بری می‌کند باید تمام متدهای abstract در base class را override کند در غیر این صورت derived class نیز باید به صورت abstract تعریف شود.

به مثال زیر توجه کنید:

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

    // Parameterized constructor.
    public TwoDShape(double w, double h, string n)
    {
        Width = w;
        Height = h;
        name = n;
    }

    // Properties for Width and 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 string name { get; set; }

    // Now, Area() is abstract.
    public abstract double Area();
}

// A derived class of TwoDShape for triangles.
class Triangle : TwoDShape
{
    string Style;

    // Constructor for Triangle.
    public Triangle(string s, double w, double h)
```

```

        : base(w, h, "triangle")
    {
        Style = s;
    }

    // Override Area() for Triangle.
    public override double Area()
    {
        return Width * Height / 2;
    }
}

// A derived class of TwoDShape for rectangles.
class Rectangle : TwoDShape
{
    // Constructor for Rectangle.
    public Rectangle(double w, double h) :
        base(w, h, "rectangle") { }

    // Override Area() for Rectangle.
    public override double Area()
    {
        return Width * Height;
    }
}

class AbsShape
{
    static void Main()
    {
        Triangle triangle = new Triangle("right", 8.0, 12.0);
        Rectangle rectangle = new Rectangle(10, 4);

        Console.WriteLine("Triangle | Area: " + triangle.Area());
        Console.WriteLine("Rectangle | Area: " + rectangle.Area());
    }
}

```

همان‌طور که برنامه نشان می‌دهد، همه‌ی derived class ها بایستی Area() را override کنند (یا اینکه خودشان باید abstract باشند). نکته‌ی دیگر این است که یک abstract class می‌تواند متدهایی داشته باشد که abstract نیستند و derived class ها می‌توانند در صورت نیاز آن‌ها را override کنند درحالی که هیچ اجباری در کار نیست.

---

کلیه حقوق مادی و معنوی برای وبسایت [وب‌تارگت](#) محفوظ است.

استفاده از این مطلب در سایر وبسایت‌ها و نشریات چاپی تنها با ذکر و درج لینک منبع مجاز است.