

### استفاده از sealed برای جلوگیری از ارث‌بری

با این که inheritance بسیار مفید و کاربردی است، گاهی نیاز است که از انجام شدن آن پیش‌گیری کنید. این که در کجا و در چه شرایطی از انجام inheritance جلوگیری کنید، بستگی به مساله و منطق خودتان دارد. در سی‌شارپ با استفاده از کلمه‌ی کلیدی sealed به راحتی می‌توانید مانع انجام شدن inheritance شوید.

واژه‌ی sealed به معنای **مهر و موم شده** است و با استفاده از آن اطمینان می‌یابید که از یک کلاس مهر و موم شده نمی‌توان ارث‌بری کرد. به‌منظور sealed کردن یک کلاس، کافی است که در ابتدای تعریف کلاس از کلمه‌ی کلیدی sealed استفاده کنید. دقت داشته باشید که یک کلاس را نمی‌توان هم به‌صورت sealed و هم به‌صورت abstract تعریف کرد چرا که کلاس abstract به تنهایی کامل نیست و برای این که اجرای کاملی داشته باشد وابسته به کلاس‌های مشتق‌شده از خودش است.

به مثال زیر دقت کنید:

```
using System;
sealed class A {
    // ...
}
class B : A { // ERROR! Can't derive from class A
    // ...
}
class SealedDemo
{
    static void Main()
    {
        B ob = new B();
    }
}
```

در مثال بالا، کلاس A به‌صورت sealed تعریف شده است بنابراین هیچ کلاسی نمی‌تواند از این کلاس ارث‌بری کند.

اگر قصد compile کردن برنامه‌ی بالا داشته باشید با این خطا مواجه می‌شوید:

'B': cannot derive from sealed type 'A'

به این معنا که B نمی‌تواند از کلاس sealed شده‌ی A ارث‌بری کند.

نکته‌ی دیگر این است که sealed می‌تواند در virtual methods نیز برای پیش‌گیری از override شدن مورد استفاده قرار گیرد.

به نمونه‌ی زیر توجه کنید:

```
class B
{
    public virtual void MyMethod() { /* ... */ }
}
class D : B
{
    sealed public override void MyMethod() { /* ... */ }
}
class X : D
{
    // Error! MyMethod() is sealed!
    public override void MyMethod() { /* ... */ }
}
```

در این‌جا، کلاس B یک متد virtual دارد که در کلاس D هم override و هم sealed شده است. از این‌رو کلاس‌هایی که از D ارث‌بری می‌کنند دیگر نمی‌توانند MyMethod() را override کنند زیرا این متد دیگر sealed شده است.

## The object Class

سی‌شارپ یک کلاس خاص به اسم object دارد که تمام type های دیگر است. بدین معنا که تمام type ها (هم reference types و هم value types) از object ارث‌بری کرده‌اند. بنابراین یک reference variable از نوع object می‌تواند به هر نوعی رجوع کند. در قسمت بیست و هفتم زنگ سی‌شارپ با boxing و unboxing آشنا شدید. هنگامی که یک متغیر value type به نوع object تبدیل می‌شود، boxing و هنگامی که یک نوع object به value type تبدیل می‌شود unboxing اتفاق می‌افتد.

نمونه‌ی زیر نشان می‌دهد که چگونه یک متغیر از نوع object می‌تواند هر گونه data type را بپذیرد:

```
using System;
```

```

class ObjectTest
{
    public int i = 10;
}

class MainClass2
{
    static void Main()
    {
        object a;
        a = 1; // an example of boxing
        Console.WriteLine(a);
        Console.WriteLine(a.GetType());
        Console.WriteLine(a.ToString());

        a = new ObjectTest();
        ObjectTest classRef;
        classRef = (ObjectTest)a;
        Console.WriteLine(classRef.i);
    }
}
/* Output
1
System.Int32
1
10
*/

```

از لحاظ فنی اسم `object` در سی‌شارپ، اسم دیگر `System.Object` بوده که بخشی از `.Net Framework class library` است. کلاس `object` تعدادی متد دارد که در جدول زیر می‌بینید:

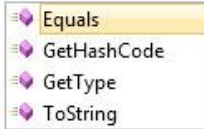
Name	Description
<code>Equals(Object)</code>	Determines whether the specified object is equal to the current object.
<code>Equals(Object, Object)</code>	Determines whether the specified object instances are considered equal.
<code>Finalize</code>	Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection.
<code>GetHashCode</code>	Serves as the default hash function.
<code>GetType</code>	Gets the <code>Type</code> of the current instance.
<code>MemberwiseClone</code>	Creates a shallow copy of the current <code>Object</code> .
<code>ReferenceEquals</code>	Determines whether the specified <code>Object</code> instances are the same instance.
<code>ToString</code>	Returns a string that represents the current object.

هنگامی که یک کلاس می‌سازید تعدادی از این متدها را مستقیماً در اختیار دارید:

```

1 using System;
2 class A
3 {
4 }
5 class MainClass
6 {
7     static void Main()
8     {
9         A a = new A();
10        a.
11    }
12 }

```



در ادامه مثال‌هایی از ارث‌بری، برای درک بهتر این موضوع می‌بینید.

به مثال زیر توجه کنید:

```

using System;
class Automobile
{
    public string Color { get; set; }
    public int Weight { get; set; }
    public int TopSpeed { get; set; }

    public virtual void Accelerate()
    {
        Console.WriteLine("Automobile is accelerating");
    }

    public void Brake()
    {
        Console.WriteLine("Automobile is braking");
    }
}

class Car : Automobile
{
    public int TrunkSize { get; set; }
}

class Truck : Automobile
{
    // ...
    public override void Accelerate()
    {
        Console.WriteLine("Truck is accelerating");
    }
}

class Van : Automobile
{
    // ...
}

```

```

class MainClass
{
    static void Main()
    {
        // ...
    }
}

```

در مثال بالا، کلاس Automobile شامل تعدادی method و property بوده که یکی از این متدها virtual است. کلاس‌های Car و Van و Truck از کلاس Automobile ارث‌بری می‌کنند زیرا همه‌گی به‌نحوی اتومبیل هستند و یک‌سری ویژگی مشترک دارند. هر یک از این کلاس‌ها می‌تواند علاوه بر مواردی که به ارث برده است، موارد مورد نیاز خود را نیز اضافه کند. به‌عنوان مثال، همان‌طور که می‌بینید کلاس Car یک property اضافه‌تر دارد و کلاس Truck متد Accelerate را override کرده است تا این متد را متفاوت اجرا کند.

به مثال زیر توجه کنید:

```

using System;
class Vehicle
{
    protected int someInt;
    public int SomeInt
    {
        get { return someInt; }
        set { someInt = value; }
    }

    public void PrintSomeInt()
    {
        Console.WriteLine(this.someInt);
    }

    public void Transport()
    {
        Console.WriteLine("Vehicle is transporting");
    }

    public virtual void Stop()
    {
        Console.WriteLine("Vehicle is stopping");
    }
}

class Car : Vehicle
{
    public Car()
    {
        this.someInt = 6;
    }
}

```

```

}

class Plane : Vehicle
{
    public void IncreaseAlt()
    {
        Console.WriteLine("Plane is increasing altitude");
    }

    public override void Stop()
    {
        Console.WriteLine("Land");
        base.Stop();
    }
}

class Ship : Vehicle
{
    // ...
}

class MainClass
{
    static void Main()
    {
        Vehicle a = new Vehicle();
        a.Stop();

        Plane b = new Plane();
        b.Stop();

        //...
    }
}

```

در این مثال، کلاس Vehicle شامل method و متغیر و property است. متغیر someInt به صورت protected تعریف شده است، بدین معنا که این متغیر فقط در زنجیره‌ی ارث‌بری قابل مشاهده بوده و خارج از این زنجیره، private است. متد Stop() به صورت virtual تعریف و در کلاس Plane نیز override شده و همان‌طور که می‌بینید متد Stop() در کلاس Vehicle نیز توسط کلمه‌ی کلیدی base، فراخوانی شده است.

به مثال زیر توجه کنید:

```

using System;
class Shape
{
    public int X { get; set; }
    public int Y { get; set; }

    public Shape(int x, int y)
    {
        this.X = x;
    }
}

```

```

        this.Y = y;
    }
}
class Circle : Shape
{
    public int Radius { get; set; }

    public Circle(int x, int y, int radius)
        : base(x, y)
    {
        this.Radius = radius;
    }
}

class MainClass
{
    static void Main()
    {
        Circle a = new Circle(5, 6, 100);
        Shape b = new Shape(5, 10);
    }
}

```

در این مثال، constructor کلاس Shape به دو پارامتر x و y نیاز دارد. کلاس Circle که از Shape ارث‌بری کرده است، باید در constructor خودش این مقادیر را به base class بدهد. همان‌طور که می‌بینید، توسط کلمه‌ی کلیدی base این مقادیر به base class داده شده است.

به مثال زیر توجه کنید:

```

using System;
class Shape
{
    public int X { get; set; }
    public int Y { get; set; }
    public int Z { get; set; }

    public Shape(int x, int y, int z)
    {
        this.X = x;
        this.Y = y;
        this.Z = z;
    }

    public Shape(int def)
    {
        this.X = def;
        this.Y = def;
        this.Z = def;
    }
}

class Oval : Shape

```

```

{
    public int BigRadius { get; set; }
    public int SmallRadius { get; set; }

    public Oval(int big, int small, int def)
        : base(def)
    {
        this.BigRadius = big;
        this.SmallRadius = small;
    }

    public Oval(int r, int def)
        : base(def)
    {
        this.BigRadius = r;
        this.SmallRadius = r;
    }

    public Oval(int r, int x, int y, int z)
        : base(x, y, z)
    {
        this.BigRadius = r;
        this.SmallRadius = r;
    }
}

class Circle : Oval
{
    public string Color { get; set; }
    public Circle(string color, int r, int def)
        : base(r, def)
    {
        this.Color = color;
    }

    public Circle(int r, int def)
        : base(r, def)
    {
        this.Color = "red";
    }

    public Circle()
        : base(1, 0, 0, 0)
    {
        this.Color = "red";
    }
}

class MainClass
{
    static void Main()
    {
        Shape a = new Shape(5);
        Shape b = new Shape(6, 8, 2);

        Oval c = new Oval(6, 5);
        Oval d = new Oval(6, 10);

        Circle e = new Circle("blue", 5, 9);
    }
}

```



```

    Circle f = new Circle(5, 7);

    Console.WriteLine(f.SmallRadius);
    Console.WriteLine(e.Color);
}
}

```

در برنامه‌ی بالا به زنجیره‌ی ارث‌بری و constructor ها دقت کنید. در این مثال کلاس Circle از Oval و Oval از Shape ارث‌بری کرده است. هنگامی که شما یک شیء از Circle می‌سازید و دومین constructor آن را صدا می‌زنید:

```

public Circle(int r, int def)
    : base(r, def)
{
    this.Color = "red";
}

```

این constructor باعث فراخوانی constructor ای در کلاس Oval می‌شود که شامل دو پارامتر است:

```

public Oval(int r, int def)
    : base(def)
{
    this.BigRadius = r;
    this.SmallRadius = r;
}

```

و این constructor نیز باعث فراخوانی constructor ای در کلاس Shape می‌شود که شامل یک پارامتر است:

```

public Shape(int def)
{
    this.X = def;
    this.Y = def;
    this.Z = def;
}

```

به این ترتیب است که مقادیر مربوطه مقداردهی می‌شوند.

به مثال زیر توجه کنید:

```

using System;
class Shape
{
    private int x;
    public int X
    {
        get { return x; }
        set { x = value; }
    }

    private int y;
    public int Y

```

```

{
    get { return y; }
    set { y = value; }
}

private int width;
public int Width
{
    get { return width; }
    set { width = value; }
}

private int height;
public int Height
{
    get { return height; }
    set { height = value; }
}

private string fillColor;
public string FillColor
{
    get { return fillColor; }
    set { fillColor = value; }
}

private string strokeColor;
public string StrokeColor
{
    get { return strokeColor; }
    set { strokeColor = value; }
}

private int strokeWidth;
public int StrokeWidth
{
    get { return strokeWidth; }
    set { strokeWidth = value; }
}

public Shape(int x, int y, int width, int height)
{
    this.x = x;
    this.y = y;
    this.width = width;
    this.height = height;

    this.fillColor = "white";
    this.strokeColor = "black";
    this.strokeWidth = 1;
}

public Shape(int x, int y, int width, int height,
            string fillColor, string strokeColor, int strokeWidth)
{
    this.x = x;
    this.y = y;
    this.width = width;

```

```

        this.height = height;

        this.fillColor = fillColor;
        this.strokeColor = strokeColor;
        this.strokeWidth = strokeWidth;
    }

    public virtual double Area
    {
        get
        {
            return this.width * this.height;
        }
    }

    public virtual double Perimeter()
    {
        return 2 * (width + height);
    }
}
class Circle : Shape
{
    public Circle(int x, int y, int diameter)
        : base(x, y, diameter, diameter)
    {
    }

    public Circle()
        : base(0, 0, 100, 100)
    {
    }

    public double Radius
    {
        get
        {
            return this.Width / 2.0;
        }
    }

    public override double Area
    {
        get
        {
            return Math.PI * this.Radius * this.Radius;
        }
    }

    public override double Perimeter()
    {
        return 2 * Math.PI * this.Radius;
    }
}
class Square : Shape
{
    public Square(int dim)
        : base(0, 0, dim, dim)
    {
    }
}

```

```

    }

    public Square()
        : base(0, 0, 100, 100)
    {
    }
}
class Rectangle : Shape
{
    public Rectangle(int x, int y, int width, int height)
        : base(x, y, width, height)
    {
    }

    public Rectangle()
        : base(0, 0, 100, 100)
    {
    }
}
class Oval : Shape
{
    public Oval(int x, int y, int width, int height)
        : base(x, y, width, height)
    {
    }

    public Oval()
        : base(0, 0, 100, 100)
    {
    }
}
class Line : Shape
{
    public Line(int x, int y, int width, int height)
        : base(x, y, width, height)
    {
    }

    public Line()
        : base(0, 0, 100, 100)
    {
    }
}

class MainClass
{
    static void Main()
    {
        Rectangle r = new Rectangle();
        Console.WriteLine(r.Perimeter());

        Square s = new Square(10);
        Console.WriteLine(s.Area);

        Circle c = new Circle();
        Console.WriteLine(c.Area);

        Shape a = new Shape(5, 2, 3, 4, "Yellow", "", 9);
    }
}

```

```
Console.WriteLine(a.FillColor);
Console.WriteLine(a.Perimeter());
Console.WriteLine(a.Area);

Line b = new Line();
Console.WriteLine(b.Width);
}
}
/*
Output:
400
100
7853.98163397778
Yellow
14
12
100
*/
```

به نحوه‌ی ارث‌بری، override شدن و استفاده از base در constructor ها در برنامه‌ی بالا دقت کنید.

در این قسمت مبحث ارث‌بری به پایان رسید. در قسمت آینده با Interface آشنا خواهید شد.

---

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.