زنگ سی‌شارپ – قسمت سی و هشتم

نوشته‌ی مسعود درویشیان

لینک مستقیم این مطلب در وب‌تارگت

Inheritance (وراثت، ارث‌بری)

فهمیدن و درک کردن کلاس‌ها به شما کمک می‌کند تا بتوانید اشیاء را دسته‌بندی و سازماندهی کنید. با دانستن inheritance می‌توانید این دسته‌بندی و سازماندهی را دقیق‌تر انجام دهید. برای مثال، اگر تا به‌حال در مورد Braford چیزی نشنیده باشید، تشکیل تصویر آن در مغزتان کار دشوار و سختی خواهد بود. وقتی می‌دانید که Braford یک حیوان است، تصویر آن می‌تواند راحت‌تر از قبل در ذهن شما مجسم شود. بعد از آن‌که دانستید این حیوان از نوع پستاندار است، تصویر آن در ذهن شما بیشتر شکل می‌گیرد و پس از آن‌که دانستید این حیوان یک گاو است، می‌توانید تقریباً به‌طور کامل آن را در ذهنتان مجسم کنید. وقتی‌که پی بردید Braford یک گاو است، ویژگی‌ها و مشخصاتی که در همه‌ی گاوها وجود دارند در ذهن شما مجسم می‌شوند. برای شناسایی یک Braford، تنها کافی است که جزئیات کوچکی مثل رنگ، نشانه‌ها و... را بدانید اما Braford ویژگی و مشخصات کلی خود را به‌ترتیب از حیوانات، پستانداران و گاوها به ارث برده است.

هنگامی‌که واژه‌ی ارث‌بری به‌کار می‌رود، ناخودآگاه ذهن‌تان به سمت ارث‌بری ژنتیکی هدایت می‌شود. با استفاده از علم بیولوژی می‌دانید که نوع خون و رنگ چشم شما حاصل ژن‌هایی است که از والدین‌تان به ارث برده‌اید. می‌توان گفت که بسیاری از ویژگی‌ها و رفتارهای شما نیز ارثی هستند. برای مثال، ممکن است لبخندهای شما مانند لبخند مادربزرگتان باشد. همچنین ممکن است دقیقاً همان‌طور که پدرتان راه می‌رود شما نیز، به شکلی مشابه، راه بروید. شما علاوه‌بر این‌که رفتارها و ویژگی‌های خاصی را از والدین خود به ارث می‌برید، ویژگی‌ها و رفتارهای خاص خود را نیز دارید.

Inheritance یکی از اصول بنیادی برنامه‌نویسی شی‌گرا است که موجب ساخت کلاس‌ها به‌صورت سلسله‌مراتبی می‌شود. همه‌ی زبان‌های برنامه‌نویسی شی‌گرا به دلایل یکسانی از inheritance استفاده می‌کنند. با استفاده از inheritance می‌توانید یک کلاس کلی با یک سری ویژگی تعریف کنید که این ویژگی‌ها می‌توانند در تعدادی بخش مرتبط باهم، مشترک باشند. این کلاس کلی، می‌تواند توسط کلاس‌های دیگر، ارث‌بری شود و مواردی که یکتاست را در اختیار آن‌ها قرار دهد. در زبان سی‌شارپ، کلاسی که از آن ارث‌بری می‌شود، base class (کلاس پایه) نام دارد و کلاسی که

ارث‌بری را انجام می‌دهد derived class (کلاس مشتق شده) نامیده می‌شود. از این‌رو، derived class نسخه‌ی اختصاصی شده‌ی base class است. derived class تمام variable ها، method ها، property ها و indexer های تعریف شده در base class را به ارث می‌برد و در کنار این‌ها عناصر مخصوص به خود را نیز اضافه می‌کند.

برای آشنا شدن با روند کار inheritance به کلاس زیر توجه کنید:

```
class Animal
{
    public bool Mammal;
    public string Color;
    public int Weight;
    public string Gender;

    public void Greet()
    {
        Console.WriteLine("The Animal says hello!");
    }
    public void Talk()
    {
        Console.WriteLine("The Animal Talk!");
    }
    public void Eat()
    {
        Console.WriteLine("The Animal Eat!");
    }
}
```

در کلاس Animal تعدادی متد و متغیر می‌بینید که می‌تواند در همه‌ی حیوانات مشترک باشد. این کلاس را به‌عنوان base class در نظر می‌گیریم. در مرحله‌ی بعد کلاس Dog عناصر این base class را به ارث می‌برد. دقت کنید که کلاس Dog را چگونه تعریف می‌کنیم:

```
class Dog : Animal
{
    public string DogType { get; set; }
    public string Name { get; set; }

    public void Bark()
    {
        Console.WriteLine(Name + " is barking!");
    }
}
```

کلاس Dog نوع خاصی از کلاس Animal را می‌سازد. کلاس Dog شامل همه‌ی موارد تعریف شده در کلاس Animal است و علاوه‌بر آن‌ها، دو فیلد و یک متد اضافه‌تر نیز دارد.

به نحوه‌ی استفاده از این کلاس‌ها توجه کنید:

```
using System;
class Animal
{
    public bool Mammal { get; set; }
    public string Color { get; set; }
    public int Weight { get; set; }
    public string Gender { get; set; }
    public int Age { get; set; }

    public void Greet()
    {
        Console.WriteLine("The Animal says hello!");
    }
    public void Talk()
    {
        Console.WriteLine("The Animal Talk!");
    }
    public void Eat()
    {
        Console.WriteLine("The Animal Eat!");
    }
}
class Dog : Animal
{
    public string Breed { get; set; }
    public string Name { get; set; }

    public void Bark()
    {
        Console.WriteLine(Name + " is barking!");
    }
}
class InheritanceDemo
{
    static void Main()
    {
        Dog dog = new Dog();

        dog.Breed = "Brittany";
        dog.Name = "Ace";
        dog.Gender = "Male";
        dog.Mammal = true;
        dog.Weight = 10;
        dog.Color = "White";
        dog.Age = 1;

        dog.Talk();
        dog.Eat();
        dog.Greet();
        dog.Bark();
    }
}
```

همان‌طور که می‌بینید، توسط شیء dog میتوانیم به تمام فیلدها و متدهای کلاس Animal دسترسی داشته باشیم. زیرا کلاس Dog از کلاس Animal مشتق شده است.

به syntax ارث‌بری توجه کنید:

```
class Dog : Animal {
```

هنگامی که یک کلاس از کلاس دیگری ارث‌بری می‌کند، نام base class بعد از نام derived class قرار می‌گیرد که این دو توسط colon از هم جدا شده‌اند.

اگرچه کلاس Animal برای کلاس Dog یک base class است اما این کلاس یک کلاس کاملاً مستقل است. base class بودن برای یک derived class بدین معنا نیست که base class نمی‌تواند به‌صورت مجزا استفاده شود. برای مثال، استفاده از کلاس Animal به شکل زیر هیچ مشکلی ندارد:

```
Animal animal = new Animal();
animal.Mammal = true;
animal.Talk();
```

همچنین یک base class هیچ اطلاعاتی در مورد derived class ها ندارد.

به مثال بعدی توجه کنید:

```
using System;
class TwoDShape
{
    public double Width;
    public double Height;
    public void ShowDim()
    {
        Console.WriteLine("Width and height are " + Width + " and " + Height);
    }
}
class Triangle : TwoDShape
{
    public string Style;
    public double Area()
    {
        return Width * Height / 2;
    }
    public void ShowStyle()
    {
        Console.WriteLine("Triangle is " + Style);
    }
}
class Shapes
{
    static void Main()
    {
        Triangle t1 = new Triangle();
        Triangle t2 = new Triangle();

        t1.Width = 4.0;
        t1.Height = 4.0;
        t1.Style = "isosceles";

        t2.Width = 8.0;
```

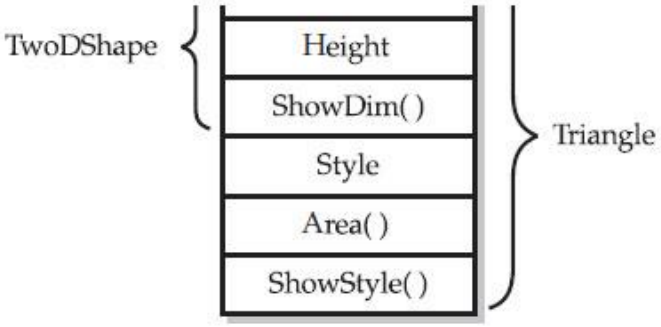
```
        t2.Height = 12.0;
        t2.Style = "right";

        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());
    }
}
```

در این مثال، کلاس TwoDShape شامل فیلدها و متدهای یک شکل دو بعدی است. کلاس Triangle از این base class ارث‌بری کرده و فیلد و متدهای مربوط به خود را به این کلاس افزوده است:



شما تنها می‌توانید یک base class برای هر derived class مشخص کنید. در سی‌شارپ نمی‌توانید چندین base class برای یک derived class داشته باشید زیرا سی‌شارپ از multiple inheritance پشتیبانی نمی‌کند اما می‌توانید multiple inheritance را شبیه‌سازی کنید. شما می‌توانید سلسله‌مراتب ارث‌بری را به‌وجود آورید که در آن یک derived class برای کلاس دیگر، base class باشد. توجه کنید که هیچ کلاسی نمی‌تواند (چه به‌طور مستقیم، چه به‌طور غیر مستقیم) base class خودش باشد.

یکی دیگر از مزیت‌های inheritance این است که می‌توانید از یک base class به تعداد دل‌خواه derived class داشته باشد. هر derived class یک ورژن اختصاصی از base class است. برای مثال، کلاس زیر یک derived class دیگر از کلاس Animal است:

```
class Cat : Animal
{
    public string Breed { get; set; }
    public void Run()
```

```
    {
        Console.WriteLine("The cat is running!");
    }
}
```

این کلاس نیز مشخصات خود را اضافه کرده است.

اعضای یک کلاس ممکن است بنا به دلایلی، private باشند. هنگامی که از یک کلاس ارث‌بری می‌کنید، قادر به دیدن اعضای private آن کلاس نخواهید بود. از این‌رو، اگرچه در inheritance می‌توانید به اعضای base class دسترسی داشته باشید، اما به اعضای private آن دسترسی نخواهید داشت. شاید در ابتدا فکر کنید این یک محدودیت جدی است که derived class به اعضای private یک base class دسترسی ندارد اما سی‌شارپ راه‌حل‌هایی برای این مسئله در نظر گرفته است. یکی از این راه‌حل‌ها، اعضای protected هستند که در ادامه به شرح آن‌ها خواهیم پرداخت. یک راه‌حل دیگر استفاده از property های public برای دسترسی و کنترل اطلاعات private است.

به مثال زیر توجه کنید:

```
using System;
class Human
{
    private int pri_age;
    private int pri_weight;

    public int Age
    {
        get
        {
            return pri_age;
        }
        set
        {
            pri_age = value >= 0 ? value : -value;
        }
    }

    public int Weight
    {
        get
        {
            return pri_weight;
        }
        set
        {
            pri_weight = value >= 0 ? value : -value;
        }
    }
}
class Male : Human
{
    public string Name { get; private set; }
    public int ID { get; private set; }
```

```
    public Male(string name, int age, int weight, int id)
    {
        Name = name;
        Age = age;
        Weight = weight;
        ID = id;
    }
}
class InheritDemo
{
    static void Main()
    {
        Male m1 = new Male("John", 26, 70, 180081);

        Console.WriteLine("Name: " + m1.Name);
        Console.WriteLine("Age: " + m1.Age);
        Console.WriteLine("Weight: " + m1.Weight);
        Console.WriteLine("ID: " + m1.ID);

        m1.Age = -30;
        Console.WriteLine(m1.Age); // prints 30

        m1.Weight = -65;
        Console.WriteLine(m1.Weight); // prints 65

        // m1.Name = "Jeremy"; // wrong! Name property is read-only
        // ID = 15567; // wrong! ID property is read-only
    }
}
```

همان‌طور که می‌بینید، توسط property دسترسی به اعضای private امکان پذیر شده و این دسترسی کنترل شده است. به‌یاد داشته باشید که اعضای private یک کلاس، پیوسته مختص آن کلاس است و هیچ کدی خارج از آن کلاس نمی‌تواند به آن دسترسی داشته باشد، حتی derived classes.

دسترسی به‌صورت Protected (محفوظ)

همان‌طور که گفته شد، اعضای private یک base class برای derived class قابل دسترسی نیست. بنابراین به‌نظر می‌رسد به‌منظور این‌که یکی از اعضای base class در derived class قابل دسترسی باشد، باید آن را به‌صورت public تعریف کنیم و همان‌طور که می‌دانید، اعضای public برای دیگر کدها نیز قابل دسترسی هستند که شاید در این مورد مطلوب نباشد. در این شرایط سی‌شارپ به شما اجازه می‌دهد تا عضوی را به‌صورت protected تعریف کنید. یک عضو protected برای سلسله‌مراتب derived class ها قابل رویت و دسترسی است (public) اما خارج از این سلسله‌مراتب قابل دسترسی نیست (private) .

هنگامی که عضوی از کلاس به‌صورت protected تعریف می‌شود، آن عضو در واقع با یک استثنا، private است و این استثنا زمانی است که از آن عضو protected، ارث‌بری می‌شود. در این مورد، یک عضو protected در base class تبدیل به یک عضو protected در یک derived class می‌شود و این روند همین‌طور ادامه خواهد یافت. بنابراین اعضای protected فقط در سلسله‌مراتب ارث‌بری قابل دسترسی بوده و خارج از این سلسله‌مراتب، private هستند. از این‌رو، با استفاده از protected می‌توانید اعضایی تعریف کنید که private هستند اما می‌توان از آن‌ها در ارث‌بری استفاده کرد.

به مثال زیر توجه کنید:

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

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

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

    public void SetK()
    {
        // B can access A's i and j
        k = i * j;
    }

    public void ShowK()
    {
        Console.WriteLine(k);
    }
}
class ProtectedDemo
{
    static void Main()
    {
        B ob = new B();

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

        ob.SetK();
        ob.ShowK();
    }
}
```