زنگ سی‌شارپ – قسمت چهل و پنجم

نوشته‌ی مسعود درویشیان

لینک مستقیم این مطلب در وب‌تارگت

بین Interface و Abstract Class کدام‌یک را انتخاب کنیم؟

یکی از قسمت‌های مهم برنامه‌نویسی سی‌شارپ دانستن این موضوع است، هنگامی‌که قصد دارید قابلیت‌های یک کلاس را شرح دهید، چه زمانی از interface و چه زمانی از abstract class باید استفاده کنید درحالی‌که قسمت اجرایی ندارید. قانون کلی بدین صورت است که هرگاه بخواهید مفهوم کلی را شرح دهید و فقط به انجام شدن کارها تاکید داشته باشید و در واقع چگونه‌گی انجام شدن آن برای شما اهمیت نداشته باشد، باید از interface استفاده کنید. اگر نیاز دارید که بعضی از جزئیات اجرا شدن را از قبل وارد کنید، آن‌گاه باید abstract class را مورد استفاده قرار دهید.

Structures

همان‌طور که می‌دانید، کلاس‌ها reference type هستند. این بدان معنا است که اشیای کلاس از طریق یک reference قابل دسترسی هستند. از این‌رو reference type ها با value type ها که مستقیماً قابل دسترسی‌اند، متفاوت هستند. اما دسترسی مستقیم به یک شیء (به شکلی مشابه با value type ها) نیز گاهی می‌تواند سودمند باشد. یکی از دلایل این‌کار، افزایش بهره‌وری است. درسترسی به اشیاء از طریق reference باعث به‌وجود آمدن overhead (سربار) می‌شود و این overhead ها فضا اشغال می‌کنند. برای هر شیء کوچک، این فضاها می‌تواند قابل توجه باشد. برای رفع این نگرانی، سی‌شارپ structure را ارائه داده است. یک structure مشابه با class است با این تفاوت که structure، value type اما کلاس reference type است.

Structure ها توسط کلمه‌کلیدی struct تعریف می‌شوند و syntax آن‌ها مشابه class است. فرم کلی struct به شکل زیر است:

```
struct name : interfaces
{
    // member declarations
}
```

در این‌جا، اسم این structure توسط name مشخص شده است. structure ها نمی‌توانند از structure ها یا از class های دیگر ارث‌بری کنند و همچنین نمی‌توان از آن‌ها برای structure ها و یا class های دیگر به عنوان base استفاده کرد. این بدین معنی است که inheritance در structure ها کاملاً بی استفاده است (اما به‌صورت پیش‌فرض structure از System.ValueType ارث‌بری می‌کند که System.ValueType خودش از object ارث‌بری می‌کند). با این حال یک structure می‌تواند یک یا چند interface را اجرا کند که این interface ها بعد از نام structure مشخص می‌شوند و لیست آن‌ها توسط کاما از هم جدا می‌شود. همچون کلاس، structure می‌تواند شامل اعضایی چون method، field، indexer، property، operator method و event باشد. structure ها همچنین می‌توانند constructor داشته باشند اما destructor ندارند. نکته‌ی دیگر این است که نمی‌توانید در structure ها default constructor (constructor بدون پارامتر) تعریف کنید زیرا default constructor به‌صورت پیش‌فرض برای همه‌ی structure ها تعریف شده است و این default constructor قابل تغییر نیست. Default constructor فیلدهای structure را به مقادیر پیش‌فرض‌شان مقداردهی می‌کند. از آن‌جایی که structure از inheritance پشتیبانی نمی‌کند، منطقی است که مجاز به استفاده از اعضای structure به‌عنوان abstract، virtual یا protected نخواهید بود.

یک شیء structure می‌تواند مشابه با کلاس با استفاده از new ساخته شود اما استفاده از new ضروری نیست. هنگامی‌که از new استفاده شود، constructor مشخص شده نیز فراخوانی خواهد شد. هنگامی‌که از new استفاده نشود، شیء ساخته می‌شود اما مقداردهی نشده است بنابراین نیاز است تا مقادیر آن را مقداردهی کنید.

به مثال زیر دقت کنید:

```
using System;
struct Gamepad
{
    public string name;
    public string color;

    public Gamepad(string name, string color)
    {
        this.name = name;
        this.color = color;
    }

    public void Show()
    {
        Console.WriteLine("Name : " + name);
        Console.WriteLine("Color: " + color);
    }
}
class MainClass
```

```
{
    static void Main()
    {
        Gamepad gamepad1 = new Gamepad("Xbox One Wireless Controller", "Black"); // explicit
constructor

        Gamepad gamepad2 = new Gamepad(); // default constructor
        Gamepad gamepad3; // no constructor

        gamepad1.Show();

        Console.WriteLine();
        if (gamepad2.name == null)
            Console.WriteLine("gamepad2.name is null!");

        // Now, give gamepad2 some info
        gamepad2.name = "PS4 Wireless Controller";
        gamepad2.color = "Black";

        Console.WriteLine();
        Console.WriteLine("gamepad2 now contains: ");
        gamepad2.Show();

        Console.WriteLine();

        // Console.WriteLine(gamepad3.name); // must be initialize first

        gamepad3.name = "Steam Controller";
        gamepad3.color = "Gray";
        gamepad3.Show();
    }
}

/* Output

Name : Xbox One Wireless Controller
Color: Black

gamepad2.name is null!

gamepad2 now contains:
PS4 Wireless Controller
Black

Name : Steam Controller
Color: Gray

 */
```

همان‌طور که برنامه‌ی بالا نشان می‌دهد، structure هم می‌تواند از طریق new و فراخوانی constructor ساخته شود و هم می‌تواند به‌سادگی فقط تعریف شود. اگر از new استفاده شود یا از default constructor و یا از constructor تعریف شده توسط برنامه‌نویس استفاده خواهد شد. اما اگر مانند gamepad3 از new استفاده نشود، پیش از استفاده از شیء بایستی حتماً آن را مقداردهی کنید.

هنگامی که یک structure را به یک structure دیگر اختصاص می‌دهید، یک کپی از شیء ساخته می‌شود. تفاوت مهم struct و class در همین نکته است. همان‌طور که قبلاً توضیح داده شد، هنگامی که یک class reference را به یک class reference دیگر اختصاص می‌دهید، reference سمت چپ تساوی به همان شیءای رجوع می‌کند که reference سمت راست تساوی به آن رجوع می‌کند. این بدان معنی است که اگر مقادیر شیء را تغییر دهید، هر دوی class reference ها تغییرات را می‌بینند. اما هنگامی که یک struct را به یک struct دیگر اختصاص می‌دهید، یک کپی مستقل از شیء ایجاد می‌کنید و تغییر یکی از اشیاء ربطی به دیگری ندارد.

به مثال زیر توجه کنید:

```
// Copy a struct.
using System;
// Define a structure.
struct MyStruct
{
    public int x;
}
// Demonstrate structure assignment.
class StructAssignment
{
    static void Main()
    {
        MyStruct a;
        MyStruct b;

        a.x = 10;
        b.x = 20;

        Console.WriteLine("a.x {0}, b.x {1}", a.x, b.x);

        a = b;
        b.x = 30;

        Console.WriteLine("a.x {0}, b.x {1}", a.x, b.x);
    }
}

/* Output

a.x 10, b.x 20
a.x 20, b.x 30

*/
```

همان‌طور که خروجی نشان می‌دهد، بعد از اختصاص

```
a = b;
```

structure variable های a و b همچنان جدا و مجزا هستند. این بدان معناست که a به b رجوع نمی‌کند و ارتباطی بین آن‌ها نیست. فقط و فقط یک کپی مجزا از شیء b به a اختصاص داده شده است.

اگر a و b هردو class reference بودند، تغییراتی متفاوت با مثال بالا رخ می‌داد. به مثال زیر که class version مثال بالا است دقت کنید:

```
// Use a class.
using System;
// Now a class.
class MyClass
{
    public int x;
}
// Now show a class object assignment.
class ClassAssignment
{
    static void Main()
    {
        MyClass a = new MyClass();
        MyClass b = new MyClass();

        a.x = 10;
        b.x = 20;

        Console.WriteLine("a.x {0}, b.x {1}", a.x, b.x);

        a = b;
        b.x = 30;

        Console.WriteLine("a.x {0}, b.x {1}", a.x, b.x);
    }
}

/* Output

a.x 10, b.x 20
a.x 30, b.x 30

*/
```

همان‌طور که می‌بینید، a و b بعد از اختصاص‌دهی هردو به یک شیء رجوع می‌کنند.

چرا باید از structure استفاده کنیم؟

ممکن است این سوال در ذهن‌تان به‌وجود آمده باشد که چرا در سی‌شارپ struct وجود دارد درحالی‌که کلاس ورژن کامل‌تری است. پاسخ این است که struct بهره‌وری و سرعت اجرای بیشتری دارد. دلیل آن این است که structure به‌صورت value type است و دیگر reference ای وجود ندارد بنابراین دسترسی به آن مستقیم است. از این‌رو در بعضی

موارد از memory کمتری نیز استفاده می‌شود. درکل، هرگاه نیاز دارید که گروهی از اطلاعات مرتبط را کنار هم قرار دهید و در عین حال نیازی به inheritance و دسترسی به شیء از طریق reference ندارید، آن‌گاه struct می‌تواند انتخاب موثرتری باشد.

Enumerations

یک enumeration مجموعه‌ای از integer های نام‌گذاری شده است. نوع enumeration توسط کلمه‌کلیدی enum تعریف می‌شود. فرم کلی یک enumeration به شکل زیر است:

```
enum name { enumeration list };
```

در این‌جا، نام این enumeration توسط name مشخص شده است و enumeration list لیستی از شناسه‌ها است که توسط کاما از هم جدا شده‌اند.

در مثال زیر یک enumeration به اسم Apple وجود دارد که انواع مختلفی از Apple را لیست کرده است:

```
enum Apple
{
    Jonathan, GoldenDel, RedDel, Winesap,
    Cortland, McIntosh
};
```

نکته‌ی کلیدی در مورد enumeration ها این است که هر symbol در آن، جایگزین یک integer value است. دقت کنید که برای convert کردن بین این دو باید از explicit cast استفاده کنید. از آن‌جا که enumerations مقادیر integer را ارائه می‌دهند، می‌توانید از enumeration برای کنترل کردن (مثلاً) switch و حلقه‌ی for استفاده کنید.

شماره‌ی symbol های enumeration از صفر شروع می‌شود بنابراین در نمونه‌ی بالا، Jonathan مقدار صفر و GoldenDel مقدار یک دارد.

اعضای یک enumeration از طریق dot operator قابل دسترسی هستند. برای مثال:

```
Console.WriteLine(Apple.RedDel + " has the value " + (int)Apple.RedDel);
```

در خروجی، RedDel has the value 2 را نمایش می‌دهد.

همان‌طور که خروجی نشان می‌دهد، هنگامی‌که یک enumerated value نمایش داده می‌شود، از نام آن استفاده شده است. برای دیدن مقدار integer آن، باید از cast استفاده کرد.

به مثال زیر دقت کنید:

```
// Demonstrate an enumeration.
using System;
class EnumDemo
{
    enum Apple
    {
        Jonathan, GoldenDel, RedDel, Winesap,
        Cortland, McIntosh
    };
    static void Main()
    {
        string[] color = {
            "Red",
            "Yellow",
            "Red",
            "Red",
            "Red",
            "Reddish Green"
                          };
        Apple i; // declare an enum variable

        // Use i to cycle through the enum.
        for (i = Apple.Jonathan; i <= Apple.McIntosh; i++)
            Console.WriteLine(i + " has value of " + (int)i);

        Console.WriteLine();

        // Use an enumeration to index an array.
        for (i = Apple.Jonathan; i <= Apple.McIntosh; i++)
            Console.WriteLine("Color of " + i + " is " +
            color[(int)i]);
    }
}

/* Output

Jonathan has value of 0
GoldenDel has value of 1
RedDel has value of 2
Winesap has value of 3
Cortland has value of 4
McIntosh has value of 5

Color of Jonathan is Red
Color of GoldenDel is Yellow
Color of RedDel is Red
Color of Winesap is Red
Color of Cortland is Red
Color of McIntosh is Reddish Green

 */
```

دقت کنید که چگونه حلقه‌ی for با متغیر نوع Apple کنترل می‌شود. همان‌طور که گفته شد، هیچ‌گونه implicit conversion بین نوع integer و enumeration type وجود ندارد و در صورت نیاز حتماً باید از explicit cast استفاده کنید.

نکته‌ی دیگر این است که همه‌ی enumeration ها به‌طور پیش‌فرض از System.Enum ارث‌بری می‌کنند که System.Enum از System.ValueType و خود System.ValueType از object ارث‌بری می‌کند.

مقداردهی یک Enumeration

شما می‌توانید مقدار یک یا چند symbol را با استفاده از علامت تساوی و مقدار مورد نظرتان، مقداردهی کنید. بقیه‌ی symbol هایی که بعد از مقداردهی شما واقع هستند، به ترتیب مقدارشان یکی یکی بیشتر از مقداری که مشخص کرده‌اید می‌شود.

به نمونه‌ی زیر دقت کنید:

```
enum Apple
{
    Jonathan, GoldenDel, RedDel = 10, Winesap,
    Cortland, McIntosh
};
```

اکنون مقدار symbol ها به شکل زیر است:

```
/* Values of these symbols

Jonathan    =   0

GoldenDel   =   1

RedDel      =   10

Winesap     =   11

Cortland    =   12

McIntosh    =   13

*/
```

به‌صورت پیش‌فرض، enumeration ها بر اساس نوع int هستند اما شما می‌توانید یک enumeration با هر نوع عددی دیگری بسازید. برای این منظور باید به‌شکل زیر عمل کرد:

```
enum Apple : byte
{
    Jonathan, GoldenDel, RedDel, Winesap,
    Cortland, McIntosh
};
```

همان‌طور که می‌بینید، بعد از نام enumeration با استفاده از colon و مشخص کردن نوع byte، این کار را انجام داده‌ایم.

استفاده از Enumeration

ممکن است در نگاه اول با خود فکر کنید که enumeration بخشی از سی‌شارپ است که خیلی اهمیت ندارد اما در واقع این‌طور نیست و enumeration بسیار مفید و کاربردی است. برای مثال تصور کنید برنامه‌ای نوشتید که یک Conveyor را در یک کارخانه کنترل می‌کند و در برنامه‌ی شما متدی به نام ()Conveyor وجود دارد که فرمان‌های start، stop، forward و reverse را به‌عنوان پارامتر دریافت می‌کند. اکنون به‌جای فرستادن مقادیر عددی (مثلاً ۱ برای start و ۲ برای stop و...) به این متد می‌توانیم از enumeration استفاده کنیم که به این مقادیر عددی یک کلمه اختصاص می‌دهد.

به مثال زیر دقت کنید:

```
// Simulate a conveyor belt.
using System;
class ConveyorControl
{
    // Enumerate the conveyor commands.
    public enum Action { Start, Stop, Forward, Reverse };

    public void Conveyor(Action com)
    {
        switch (com)
        {
            case Action.Start:
                Console.WriteLine("Starting conveyor.");
                break;
            case Action.Stop:
                Console.WriteLine("Stopping conveyor.");
                break;
            case Action.Forward:
                Console.WriteLine("Moving forward.");
                break;
            case Action.Reverse:
                Console.WriteLine("Moving backward.");
                break;
```

```
        }
    }
}
class ConveyorDemo
{
    static void Main()
    {
        ConveyorControl c = new ConveyorControl();

        c.Conveyor(ConveyorControl.Action.Start);
        c.Conveyor(ConveyorControl.Action.Forward);
        c.Conveyor(ConveyorControl.Action.Reverse);
        c.Conveyor(ConveyorControl.Action.Stop);
    }
}

/* Output

Starting conveyor.
Moving forward.
Moving backward.
Stopping conveyor.

*/
```

به دلیل این‌که متد Conveyor() یک argument از نوع Action می‌گیرد، فقط مقادیر مشخص شده در Action باید به آن فرستاده شوند. به‌عنوان مثال نمی‌توانید مقدار ۲۲ را به آن بدهید. به نحوه‌ی استفاده از enumeration برای کنترل دستور switch نیز دقت کنید.