نوشته‌ی مسعود درویشیان

لینک مستقیم این مطلب در وب‌تارگت

Overloading Unary Operators

همان‌طور که در قسمت سی و یکم توضیح داده شد، دو حالت از Operator Method وجود دارد: Unary Operators و Binary Operators که با Binary Operators نیز آشنا شدید. Unary Operator ها مانند Binary Operator ها overload می‌شوند و با این تفاوت که در Unary Operators تنها یک operand وجود دارد.

فرم کلی Unary Operators به‌شکل زیر است:

```
// General form for overloading a unary operator
public static ret-type operator op(param-type operand)
{
    // operations
}
```

به‌عنوان مثال متد زیر، unary minus را overload می‌کند:

```
public static TwoD operator -(TwoD ob)
{
    TwoD result = new TwoD();
    result.X = -ob.X;
    result.Y = -ob.Y;
    return result;
}
```

در این‌جا یک شیء جدید return می‌شود که شامل مقدار منفی فیلدهای operand است. دقت کنید که operand تغییر نمی‌کند، برای مثال در عبارت a = -b مقدار منفی b به a اختصاص داده می‌شود در حالی‌که b بدون تغییر می‌ماند.

به نمونه‌ی زیر توجه کنید:

```
using System;
class TwoD
{
    private int X, Y;

    public TwoD()
    {
        X = Y = 1;
    }
    public TwoD(int a, int b)
    {
```

```
        X = a;
        Y = b;
    }

    public static TwoD operator -(TwoD ob)
    {
        TwoD result = new TwoD();
        result.X = -ob.X;
        result.Y = -ob.Y;
        return result;
    }

    public void Show()
    {
        Console.WriteLine("{0}, {1}", X, Y);
    }
}
class UnaryOpertorDemo
{
    static void Main()
    {
        TwoD ob = new TwoD(1, 1);
        TwoD result;
        result = -ob;
        ob.Show();
        result.Show();
    }
}
```

در سی‌شارپ، overload کردن ++ و -- بسیار آسان است. کافی است که مقدار را افزایش یا کاهش دهید و آن را return کنید اما نباید مقدار شیء operand را تغییر دهید. سی‌شارپ به‌طور خودکار حالت‌های postfix و prefix را برای شما در نظر می‌گیرد. برای مثال متد زیر یک operator++() برای کلاس TwoD است:

```
public static TwoD operator ++(TwoD ob)
{
    TwoD result = new TwoD();
    result.X = ob.X + 1;
    result.Y = ob.Y + 1;
    return result;
}
```

به مثال زیر که در آن از ++ و-- و - و + استفاده شده است توجه کنید:

```
using System;
class TwoD
{
    int X, Y;
    public TwoD()
    {
        X = Y = 0;
    }
    public TwoD(int a, int b)
    {
        X = a;
        Y = b;
    }
```

```
    // Overload binary +
    public static TwoD operator +(TwoD ob1, TwoD ob2)
    {
        TwoD result = new TwoD();
        result.X = ob1.X + ob2.X;
        result.Y = ob1.Y + ob2.Y;
        return result;
    }

    // Overload binary -
    public static TwoD operator -(TwoD ob1, TwoD ob2)
    {
        TwoD result = new TwoD();
        result.X = ob1.X - ob2.X;
        result.Y = ob1.Y - ob2.Y;
        return result;
    }

    // Overload unary -
    public static TwoD operator -(TwoD ob)
    {
        TwoD result = new TwoD();
        result.X = -ob.X;
        result.Y = -ob.Y;
        return result;
    }

    // Overload unary ++
    public static TwoD operator ++(TwoD ob)
    {
        TwoD result = new TwoD();
        result.X = ob.X + 1;
        result.Y = ob.Y + 1;
        return result;
    }

    // Overload unary --
    public static TwoD operator --(TwoD ob)
    {
        TwoD result = new TwoD();
        result.X = ob.X - 1;
        result.Y = ob.Y - 1;
        return result;
    }
    public void Show()
    {
        Console.WriteLine(X + ", " + Y);
    }
}
class OpOverloadingDemo
{
    static void Main()
    {
        TwoD a = new TwoD();
        TwoD b = new TwoD(1, 2);
        TwoD c = new TwoD(5, 5);

        Console.Write("Here is a : ");
        a.Show();
        Console.Write("Here is b : ");
```

```
        b.Show();
        Console.Write("Here is c : ");
        c.Show();
        Console.WriteLine();

        a++;
        Console.Write("Here is a after a++ : ");
        a.Show();

        a--;
        Console.Write("Here is a after a-- : ");
        a.Show();

        Console.WriteLine();

        a = c;
        Console.Write("Here is a after a = c : ");
        a.Show();

        a = c + b;
        Console.Write("Here is a after a = c + b : ");
        a.Show();

        a = b - c;
        Console.Write("Here is a after a = b - c : ");
        a.Show();
        Console.WriteLine();

        a = -b;
        Console.Write("Here is a after a = -b : ");
        a.Show();
        Console.Write("Here is b after a = -b : ");
        b.Show();
        Console.WriteLine();

        // reset objects
        a = new TwoD(1, 1);
        b = new TwoD(2, 2);
        c = new TwoD();

        Console.Write("Here is a : ");
        a.Show();
        Console.Write("Here is b : ");
        b.Show();
        Console.Write("Here is c : ");
        c.Show();
        Console.WriteLine();

        c = a++;
        Console.Write("Here is c after c = a++ : ");
        c.Show();
        Console.Write("Here is a after c = a++ : ");
        a.Show();
        Console.WriteLine();

        c = ++a;
        Console.Write("Here is c after c = ++a : ");
        c.Show();
        Console.Write("Here is a after c = ++a : ");
        a.Show();
```

```
        c = --a;
        Console.Write("Here is c after c = --a : ");
        c.Show();
        c = a--;
        Console.Write("Here is c after c = a-- : ");
        c.Show();

    }
}
```

خروجی:

```
Here is b : 1, 2
Here is c : 5, 5

Here is a after a++ : 1, 1
Here is a after a-- : 0, 0

Here is a after a = c : 5, 5
Here is a after a = c + b : 6, 7
Here is a after a = b - c : -4, -3

Here is a after a = -b : -1, -2
Here is b after a = -b : 1, 2

Here is a : 1, 1
Here is b : 2, 2
Here is c : 0, 0

Here is c after c = a++ : 1, 1
Here is a after c = a++ : 2, 2

Here is c after c = ++a : 3, 3
Here is a after c = ++a : 3, 3
Here is c after c = --a : 2, 2
Here is c after c = a-- : 2, 2
```

شما می‌توانید operator methods را هم overload کنید. رایج‌ترین دلیل آن این است که با این کار این امکان را فراهم می‌کنید تا عملیات بین یک class type (مثل کلاسی که خودتان تعریف کردید) و یک built-in type (یک type که در دات نت موجود است، مثل int) انجام شود. به‌عنوان مثال، مجدداً به کلاس TwoD دقت کنید. مشاهده کردید که + را overload کرده‌ایم و این باعث می‌شود تا مختصات یک شیء TwoD با مختصات یک شیء دیگر جمع شود. اما این تنها راه جمع کردن برای شیء TwoD نیست! شما فقط توانستید دو شیء را با هم جمع کنید اما گاهی نیاز دارید یک int را با یک شیء جمع کنید. برای این منظور شما نیاز دارید که + را دوباره overload کنید، ولی به‌صورت زیر:

```
// Overload binary + for TwoD + int.
public static TwoD operator +(TwoD op1, int op2)
{
    TwoD result = new TwoD();
    result.X = op1.X + op2;
    result.Y = op1.Y + op2;
    return result;
}
```

دقت کنید که پارامتر دوم از جنس int است. بنابراین این متد اجازه می‌دهد یک مقدار int به هر یک از فیلدهای TwoD افزوده شود. توجه داشته باشید هنگامی‌که یک binary operator را overload می‌کنید، یکی از operand ها حتماً باید از جنس کلاسی باشد که در آن operand مربوطه overload می‌شود اما بقیه‌ی operand ها می‌توانند از هر نوعی باشند. به همین دلیل است که در متد قبل یکی از parameter ها از جنس TwoD (جنس کلاسی که در آن operand مورد نظر overload شده) و دیگری از یک جنس دل‌خواه مثل int است.

در زیر مشاهده می‌کنید که + دو متد overload شده دارد:

```
using System;
class TwoD
{
    int X, Y;
    public TwoD()
    {
        X = Y = 0;
    }
    public TwoD(int a, int b)
    {
        X = a;
        Y = b;
    }
    // Overload binary + for TwoD + int.
    public static TwoD operator +(TwoD op1, int op2)
    {
        TwoD result = new TwoD();
        result.X = op1.X + op2;
        result.Y = op1.Y + op2;
        return result;
    }
    // Overload binary + for TwoD + TwoD.
    public static TwoD operator +(TwoD op1, TwoD op2)
    {
        TwoD result = new TwoD();
        result.X = op1.X + op2.X;
        result.Y = op1.Y + op2.Y;
        return result;
    }
    public void Show()
    {
        Console.WriteLine(X + ", " + Y);
    }
}
class OpOverloadingDemo
{
    static void Main()
    {
        TwoD ob1 = new TwoD();
        TwoD ob2 = new TwoD(3, 3);
        TwoD result;

        result = ob1 + 2; // TwoD + int
        ob1.Show();
        result.Show();
```

```
        result += ob2; // TwoD + TwoD
        ob2.Show();
        result.Show();
    }
}
```

همان‌طور که می‌بینید، هنگامی‌که + روی دو شیء TwoD اعمال شده، مختصات این دو شیء با هم جمع می‌شود و هنگامی‌که + روی یک شیء TwoD و یک مقدار int اعمال شده ، فیلدهای شیء با مقدار int جمع شده است.

overload کردن + کاربرد مفیدی را به کلاس TwoD می‌افزاید اما هنوز این تمام چیزی نیست که مورد نیاز است و در واقع کار هنوز تمام نشده است. متد operator+(TwoD, int) تنها برای عبارتی از این جمله:

```
ob1 = ob2 + 10;
```

مجاز است و عبارتی مثل:

```
ob1 = 10 + ob2;
```

را نمی‌پذیرد. به این دلیل که argument عدد صحیح، دومین argument است یعنی آن operand که در سمت راست قرار دارد، اما در عبارت قبلی می‌بینید که argument عدد صحیح را در سمت چپ قرار داده‌ایم. برای این‌که هر دو عبارت برای استفاده مجاز باشند، باید یک بار دیگر + را overload کنید. این‌بار باید اولین پارامتر را int و دومین پارامتر را TwoD در نظر بگیرید.

به این ترتیب، یک ورژن از متد operator+() برای TwoD + int و یک ورژن دیگر آن برای int + TwoD است. Overload کردن + (یا هر binary operator دیگری) به این شکل باعث می‌شود تا یک built-in type هم بتواند در سمت چپ و هم در سمت راست قرار بگیرد. در نمونه‌ی زیر می‌بینید که چگونه + همان‌گونه که شرح داده شد، overload شده است:

```
using System;
class TwoD
{
    int X, Y;
    public TwoD()
    {
        X = Y = 0;
    }
    public TwoD(int a, int b)
    {
        X = a;
        Y = b;
    }
    // Overload binary + for TwoD + int.
```

```
    public static TwoD operator +(TwoD op1, int op2)
    {
        TwoD result = new TwoD();
        result.X = op1.X + op2;
        result.Y = op1.Y + op2;
        return result;
    }
    // Overload binary + for int + TwoD.
    public static TwoD operator +(int op1, TwoD op2)
    {
        TwoD result = new TwoD();
        result.X = op1 + op2.X;
        result.Y = op1 + op2.Y;
        return result;
    }
    // Overload binary + for TwoD + TwoD.
    public static TwoD operator +(TwoD op1, TwoD op2)
    {
        TwoD result = new TwoD();
        result.X = op1.X + op2.X;
        result.Y = op1.Y + op2.Y;
        return result;
    }
    public void Show()
    {
        Console.WriteLine(X + ", " + Y);
    }
}
class OpOverloadingDemo
{
    static void Main()
    {
        TwoD a = new TwoD(1, 2);
        TwoD b = new TwoD(10, 10);
        TwoD c = new TwoD();

        Console.Write("Here is a: ");
        a.Show();
        Console.WriteLine();
        Console.Write("Here is b: ");
        b.Show();
        Console.WriteLine();

        c = a + b; // TwoD + TwoD
        Console.Write("Result of a + b: ");
        c.Show();
        Console.WriteLine();

        c = b + 10; // TwoD + int
        Console.Write("Result of b + 10: ");
        c.Show();
        Console.WriteLine();

        c = 15 + b; // int + TwoD
        Console.Write("Result of 15 + b: ");
        c.Show();
    }
}
```

Overload کردن عملگرهای رابطه‌ای

عملگرهای رابطه‌ای (Relational Operators)، مثل == یا < می‌توانند به‌سادگی overload شوند. به‌طور معمول، یک عملگر رابطه‌ای overload شده، مقدار true یا false را return می‌کند، به‌این دلیل که حالت و کاربرد استاندارد عملگرهای رابطه‌ای حفظ شود و بتوان از آن‌ها در عبارت‌های شرطی استفاده کرد. اگر در این موارد به‌جای مقادیر bool چیزی دیگری را return کنید، به شدت کاربرد این operator را محدود کرده‌اید. نکته‌ی مهم دیگر این‌جاست که بایستی relational operators را به‌طور جفتی overload کنید. به‌عنوان مثال اگر < را overload کردید، بایستی > را نیز overload کنید. این مورد برای operator های (=> و =<) و (== و =!) نیز صادق است.

به مثال زیر توجه کنید:

```
using System;
class TwoD
{
    int X, Y;
    public TwoD()
    {
        X = Y = 0;
    }
    public TwoD(int a, int b)
    {
        X = a;
        Y = b;
    }

    public static bool operator <(TwoD op1, TwoD op2)
    {
        return ((op1.X < op2.X) && (op1.X < op2.Y));
    }
    public static bool operator >(TwoD op1, TwoD op2)
    {
        return ((op1.X > op2.X) && (op1.Y > op2.Y));
    }
}
class OpOverloadingDemo
{
    static void Main()
    {
        TwoD ob1 = new TwoD(1, 4);
        TwoD ob2 = new TwoD(2, 3);

        if (ob1 > ob2)
            Console.WriteLine("ob1 is greater than ob2");
        if (ob1 < ob2)
            Console.WriteLine("ob1 is less than ob2");
    }
}
```

در مثال بالا، operator>(TwoD, TwoD) در صورتی true را return می‌کند که هم X و هم Y شیء اول از X و Y شیء دوم بزرگ‌تر باشد. operator<(TwoD, TwoD) در صورتی مقدار true را return می‌کند که هم X و هم Y شیء اول از X و Y شیء دوم کوچک‌تر باشد.

نکته‌ی دیگر این‌که اگر می‌خواهید operator های == و =! را overload کنید، بایستی متدهای Object.Equels() و Object.GetHashCode() را نیز override کنید. در مورد این متدها و تکنیک overriding بعداً صحبت خواهیم کرد.