نوشته‌ی مسعود درویشیان

لینک مستقیم این مطلب در وب‌تارگت

Conversion Operators

در قسمت قبل اندکی با conversion operators آشنا شدید و همین‌طور چگونگی استفاده از implicit conversion را فرا گرفتید. برای تبدیل implicit به‌صورت زیر عمل می‌کردیم:

```
using System;
class TwoD
{
    int X, Y;
    public TwoD()
    {
        X = Y = 0;
    }
    public TwoD(int a, int b)
    {
        X = a;
        Y = b;
    }
    public static implicit operator int(TwoD op)
    {
        return op.X * op.Y;
    }
}
class OpOvDemo
{
    static void Main()
    {
        TwoD ob1 = new TwoD(2, 2);
        int i = ob1;
        Console.WriteLine(i);
    }
}
```

در این حالت، تبدیل به‌طور اتوماتیک انجام می‌شود و مقدار ۴ در i قرار می‌گیرید. اگر conversion را به‌طور explicit تعریف کنید، تبدیل به‌صورت اتوماتیک انجام نمی‌شود و cast مورد نیاز است. در زیر برنامه‌ی بالا را بازنویسی کرده‌ایم اما این‌بار به‌جای implicit از explicit استفاده شده است:

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

```
    }
    public TwoD(int a, int b)
    {
        X = a;
        Y = b;
    }
    public static explicit operator int(TwoD op)
    {
        return op.X * op.Y;
    }
}
class OpOvDemo
{
    static void Main()
    {
        TwoD ob1 = new TwoD(2, 2);
        int i = (int)ob1;
        Console.WriteLine(i);
    }
}
```

همان‌طور که می‌بینید، مقدار شیء ob1 درون i قرار نمی‌گیرد مگر این‌که ابتدا cast انجام شود:

```
int i = (int)ob1;
```

اگر cast را حذف کنید برنامه کامپایل نخواهد شد.

محدودیت‌هایی که در conversion operators وجود دارد:

- Target-type یا source-type در conversion بایستی از جنس همان کلاسی باشد که conversion در آن تعریف شده است. برای مثال نمی‌توانید تبدیل double به int را از نو تعریف کنید.
- نمی‌توانید class type را به نوع داده‌ی object تبدیل کنید.
- نمی‌توانید برای یک source-type و target-type هم تبدیل implicit و هم تبدیل explicit تعریف کنید.
- نمی‌توانید از یک base class به یک derived class تبدیل انجام دهید (با مبحث ارث‌بری بعداً آشنا خواهید شد).
- نمی‌توانید برای یک class-type به/از interface تبدیل انجام دهید (با مبحث interface بعداً آشنا خواهید شد).

علاوه‌بر این قوانین، برای انتخاب بین implicit یا explicit باید دقت کنید. implicit conversion باید زمانی مورد استفاده قرار گیرد که تبدیل کاملاً عاری از خطا باشد. برای کسب اطمینان در این مورد از این دو قانون پیروی کنید: یک، هیچ فقدان اطلاعاتی (مثل کوتاه‌سازی، سرریز، تغییر علامت و...) نباید رخ دهد. دو، تبدیل نباید باعث بروز exception یا خطا در برنامه شود. اگر conversion نتواند این دو قانون را رعایت کند، باید از explicit conversion بهره ببرید.

هیچ نیاز و اجباری نیست که عملکرد اپراتور overload شده با عمکرد اصلی آن operator ارتباط داشته باشد. با این حال، به‌دلیل این‌که ساختار و خوانایی کد حفظ شود، بهتر است اپراتور overload شده بازتابی از رفتار اصلی آن operator باشد. برای مثال، + مربوط به کلاس TwoD از نظر مفهومی، مشابه + در نوع integer است و اینکه این operator رفتاری مشابه / داشته باشد چندان جالب نیست.

توجه کنید که الویت operator ها قابل تغییر نیست و نمی‌توانید این الویت را عوض کنید همچنین تعدادی از operator ها قابل overload شدن نیستند. در جدول زیر operator هایی که قابل overload شدن نیستند مشخص شده‌اند:

&&	()	.	?
??	[]	||	=
=>	->	as	checked
default	is	new	sizeof
typeof	unchecked		

با operator های ناآشنا در جدول بالا، در مقالات آینده آشنا خواهید شد. قابل ذکر است که operator های انتسابی نیز overload نمی‌شوند و همین‌طور operator هایی به‌شکل =+ نیز قابل overload شدن نیستند. البته اگر یک operator را overload کنید که به‌طور کلی حالت ترکیبی مثل =+ را هم دارا باشد، این حالت ترکیبی برای شیء شما نیز به‌صورت اتوماتیک اعمال می‌شود. به‌عنوان مثال اگر + را overload کنید، =+ نیز برای استفاده فعال است:

```
TwoD a = new TwoD(2, 2);
TwoD b = new TwoD(3, 4);
a += b;
```

نکته‌ی دیگر این‌که، اگرچه نمی‌توانید اپراتور [ ] که مربوط به index آرایه است را overload کنید، اما می‌توانید از indexer استفاده کنید که در ادامه به آن می‌پردازیم.

Indexers

همان‌طور که می‌دانید، index گذاری آرایه از طریق اپراتور [ ] انجام می‌شود. تعریف کردن اپراتور [ ] برای کلاس نیز امکان‌پذیر است اما برای این منظور از operator method استفاده نکرده و در عوض از Indexer استفاده می‌کنید. Indexer

اجازه می‌دهد یک شیء مانند یک آرایه index گذاری شود. Indexer ها می‌توانند یک یا بیشتر از یک بعد داشته باشند و ما در این‌جا با Indexer یک بعدی شروع می کنیم.

فرم کلی Indexer یک بعدی به‌شکل زیر است:

```
element-type this[int index] {
    // The get accessor
    get {
        // return the value specified by index
    }

    // The set accessor
    set {
        // set the value specified by index
    }
}
```

در این‌جا، element-type مشخص کننده‌ی نوع عنصر indexer است. از این‌رو، هر عنصری که توسط indexer قابل دسترسی باشد، از نوع element-type است. این نوع با نوع یک آرایه (که برای indexer در نظر می گیرید و اصطلاحاً به آن backing store می گویند) یکسان است. پارامتر index در واقع index عنصری که می‌خواهید به آن دسترسی داشته باشید را مشخص می کند. توجه کنید که نیازی نیست حتماً جنس پارامتر int باشد اما از آن‌جا که indexer ها مشابه با index آرایه مورد استفاده قرار می گیرند، استفاده از int در این مورد رایج است.

درون بدنه‌ی indexer کلمه‌های get و set را مشاهده می کنید که به هر کدام از آن‌ها accessor گفته می‌شود. یک accessor مشابه یک متد است با این تفاوت که return-type و parameter ندارد. هنگامی که از indexer استفاده می کنید این accessor ها به‌طور اتوماتیک فراخوانی می‌شوند و هر دوی accessor ها index را به‌عنوان پارامتر دریافت می کنند. اگر indexer در طرف چپ تساوی قرار گرفته باشد، بنابراین set accessor فراخوانی و یک مقدار به عنصری که توسط index مشخص شده است، اختصاص داده می‌شود. در غیر این‌صورت get accessor فراخوانی شده و عنصر مشخص شده توسط index، return می‌شود. همچنین یک پارامتر به اسم value دارد که شامل مقداری است که به یک index مشخص اختصاص داده می‌شود.

یکی دیگر از مزیت‌های indexer این است که می‌توانید دسترسی به آرایه را دقیقاً تحت کنترل داشته باشید و از دسترسی‌های نامناسب جلوگیری کنید.

به مثال ساده‌ی زیر توجه کنید:

```
using System;
class IndexerDemo
{
    int[] arr; // reference to underlying array (backing store)
    public int Lenght;

    public IndexerDemo(int size)
    {
        arr = new int[size];
        Lenght = size;
    }

    // Indexer
    public int this[int index]
    {
        // get accessor
        get
        {
            return arr[index];
        }

        // set accessor
        set
        {
            arr[index] = value;
        }
    }
}
class idx
{
    static void Main()
    {
        IndexerDemo ob = new IndexerDemo(4);

        ob[0] = 10;
        ob[1] = 20;
        ob[2] = 30;
        ob[3] = 40;

        for (int i = 0; i < ob.Lenght; i++)
        {
            Console.WriteLine(ob[i]);
        }
    }
}
```

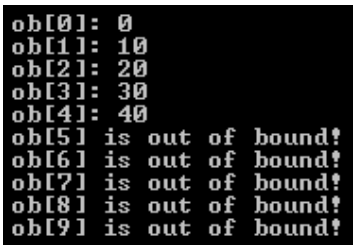
همان‌طور که می‌بینید، یک indexer تعریف کرده‌ایم که با عناصری از نوع int سروکار دارد. Indexer را به‌صورت public تعریف کرده‌ایم تا خارج از کلاس نیز قابل دسترس باشد. در قسمت get accessor مقدار arr[index] را return کرده‌ایم و همین‌طور در قسمت set accessor نیز value به index عنصر مربوطه اختصاص داده می‌شود. Value یک پارامتر بوده و شامل مقداری است که به آرایه اختصاص داده می‌شود. نیازی نیست که یک indexer هم get و set را داشته باشد بلکه می‌توانید یک indexer داشته باشید که تنها get یا set را دارد و read-only یا write-only است. البته در مثال بالا برای سادگی بیشتر، هیچ کنترلی روی مقادیری که قرار است get یا set شوند اعمال نکرده‌ایم.

در مثال زیر روی get و set کنترل بیشتری اعمال کرده‌ایم:

```
using System;
class IndexerDemo
{
    int[] arr;
    public int Length;
    public bool ErrFlag;
    public IndexerDemo(int size)
    {
        arr = new int[size];
        Length = size;
    }

    public int this[int index]
    {
        get
        {
            if (Ok(index))
            {
                ErrFlag = false;
                return arr[index];
            }
            else
            {
                ErrFlag = true;
                return 0;
            }
        }
        set
        {
            if (Ok(index))
            {
                ErrFlag = false;
                arr[index] = value;
            }
            else
                ErrFlag = true;
        }
    }

    private bool Ok(int index)
    {
        if (index >= 0 && index < Length)
            return true;
        return false;
    }
}
class Idx
{
    static void Main()
    {
        IndexerDemo ob = new IndexerDemo(5);

        for (int i = 0; i < 10; i++)
        {
            ob[i] = i * 10;

            if (ob.ErrFlag)
                Console.WriteLine("ob[{0}] is out of bound!", i);
```

```
            else
                Console.WriteLine("ob[{0}]: {1}", i, ob[i]);
        }
    }
}
```

خروجی:



همان‌طور که می‌بینید، پیش از آن‌که get یا set کنیم، ابتدا توسط متد Ok() صحیح بودن index را بررسی کرده‌ایم تا در محدوده‌ی درست بوده و out of bound نباشد. همچنین هنگامی‌که index نامناسبی در حال get یا set شدن است، متغیری به اسم ErrFalg مقداردهی می‌شود که نشان‌دهنده‌ی بروز خطا است. البته برای خطایابی در مقالات آینده با روش مناسب‌تری آشنا خواهید شد اما در حال حاضر همین روش مناسب است.

یک indexer می‌تواند overload شود. در مثال زیر علاوه‌بر indexer های int می‌توانید indexer هایی از نوع double نیز داشته باشید. در این مثال double indexer به نزدیک‌ترین index گرد (round) می‌شود:

```
using System;
class IndexerDemo
{
    int[] arr;
    public int Length;
    public bool ErrFlag;
    public IndexerDemo(int size)
    {
        arr = new int[size];
        Length = size;
    }

    public int this[int index]
    {
        get
        {
            if (Ok(index))
            {
                ErrFlag = false;
                return arr[index];
            }
            else
            {
                ErrFlag = true;
                return 0;
            }
```

```
        }
        set
        {
            if (Ok(index))
            {
                ErrFlag = false;
                arr[index] = value;
            }
            else
                ErrFlag = true;
        }
    }

    public int this[double index]
    {
        get
        {
            int idx = (int)Math.Round(index);

            if (Ok(idx))
            {
                ErrFlag = false;
                return arr[idx];
            }
            else
            {
                ErrFlag = true;
                return 0;
            }
        }
        set
        {
            int idx = (int)Math.Round(index);
            if (Ok(idx))
            {
                ErrFlag = false;
                arr[idx] = value;
            }
            else
            {
                ErrFlag = true;
            }
        }
    }

    private bool Ok(int index)
    {
        if (index >= 0 && index < Length)
            return true;
        return false;
    }
}
class Idx
{
    static void Main()
    {
        IndexerDemo ob = new IndexerDemo(5);

        for (int i = 0; i < 10; i++)
        {
            ob[i] = i * 10;
```

```
            if (ob.ErrFlag)
                Console.WriteLine("ob[{0}] is out of bound!", i);
            else
                Console.WriteLine("ob[{0}]: {1}", i, ob[i]);
        }

        Console.WriteLine();

        ob[1] = 4;
        ob[2] = 8;

        Console.WriteLine("ob[1]: {0}", ob[1]);
        Console.WriteLine("ob[2]: {0}", ob[2]);
        Console.WriteLine("ob[1.3]: {0}", ob[1.3]);
        Console.WriteLine("ob[1.7]: {0}", ob[1.7]);
    }
}
```

خروجی:

```
ob[0]: 0
ob[1]: 10
ob[2]: 20
ob[3]: 30
ob[4]: 40
ob[5] is out of bound!
ob[6] is out of bound!
ob[7] is out of bound!
ob[8] is out of bound!
ob[9] is out of bound!

ob[1]: 4
ob[2]: 8
ob[1.3]: 4
ob[1.7]: 8
```

همان‌طور که در خروجی می‌بینید، index های double توسط متد Math.Round() به نزدیک‌ترین عدد صحیح، گرد شده‌اند. 1.3 به 1 و 1.7 به 2 گرد شده است.

قابل ذکر است که نیازی نیست حتماً یک آرایه برای indexer داشته باشید. مهم این است که به یک کلاس این قابلیت را اضافه کنید تا به‌شکل آرایه نیز بتوان از آن استفاده کرد.

به مثال زیر توجه کنید:

```
using System;
class IndexerDemo
{
    public int this[int index]
    {
        get
        {
            if (index >= 1 && index <= 10)
            {
                return index * 10;
```

```
            }
            else return -1;
        }
    }
}
class Idx
{
    static void Main()
    {
        IndexerDemo ob = new IndexerDemo();

        Console.WriteLine(ob[1]);
        Console.WriteLine(ob[2]);
        Console.WriteLine(ob[3]);
        Console.WriteLine(ob[11]);
        Console.WriteLine(ob[10]);
    }
}
```

همان‌طور که می‌بینید در مثال بالا از آرایه استفاده نکرده و مستقیماً حاصل ضرب index در ۱۰ را return کرده‌ایم و اگر index بین ۱ و ۱۰ نباشد، ۱- بازگشت داده می‌شود. نکته‌ی دیگر این است که تنها از get استفاده کرده‌ایم بدین معنی که این Indexer به‌صورت read-only است و در سمت راست یک تساوی می‌تواند مورد استفاده قرار گیرد، نه در سمت چپ.

یعنی استفاده به‌شکل زیر، نادرست است:

```
ob[2] = 5;
```

اما به‌صورت زیر، کاملاً صحیح است:

```
int i = ob[2];
```

دو محدودیت دیگر برای Indexer ها موجود است. یک، به‌دلیل این‌که indexer ها درواقع storage location (محل ذخیره سازی) تعریف نمی‌کنند و به نوعی متد هستند، استفاده از آن‌ها به‌عنوان پارامتر ref و out غیرمجاز است. دو، indexer نمی‌تواند به‌صورت static تعریف شود.

Indexer های چند بعدی

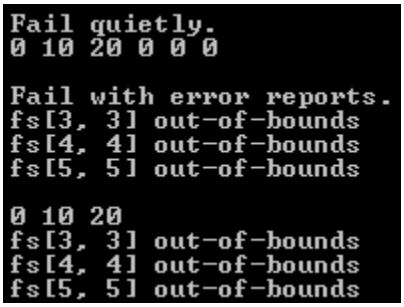
شما می‌توانید برای آرایه‌های چند بعدی نیز، indexer بسازید.

به مثال زیر توجه کنید:

```
// A two-dimensional fail-soft array.
using System;
class FailSoftArray2D
{
    int[,] a; // reference to underlying 2D array
    int rows, cols; // dimensions
    public int Length; // Length is public
    public bool ErrFlag; // indicates outcome of last operation

    // Construct array given its dimensions.
    public FailSoftArray2D(int r, int c)
    {
        rows = r;
        cols = c;
        a = new int[rows, cols];
        Length = rows * cols;
    }

    // This is the indexer for FailSoftArray2D.
    public int this[int index1, int index2]
    {
        // This is the get accessor.
        get
        {
            if (ok(index1, index2))
            {
                ErrFlag = false;
                return a[index1, index2];
            }
            else
            {
                ErrFlag = true;
                return 0;
            }
        }
        // This is the set accessor.
        set
        {
            if (ok(index1, index2))
            {
                a[index1, index2] = value;
                ErrFlag = false;
            }
            else ErrFlag = true;
        }
    }
    // Return true if indexes are within bounds.
    private bool ok(int index1, int index2)
    {
        if (index1 >= 0 & index1 < rows &
        index2 >= 0 & index2 < cols)
            return true;
        return false;
    }
}
// Demonstrate a 2D indexer.
class TwoDIndexerDemo
{
    static void Main()
    {
        FailSoftArray2D fs = new FailSoftArray2D(3, 5);
```

```
        int x;

        // Show quiet failures.
        Console.WriteLine("Fail quietly.");
        for (int i = 0; i < 6; i++)
            fs[i, i] = i * 10;
        for (int i = 0; i < 6; i++)
        {
            x = fs[i, i];
            if (x != -1) Console.Write(x + " ");
        }
        Console.WriteLine();

        // Now, display failures.
        Console.WriteLine("\nFail with error reports.");
        for (int i = 0; i < 6; i++)
        {
            fs[i, i] = i * 10;
            if (fs.ErrFlag)
                Console.WriteLine("fs[" + i + ", " + i + "] out-of-bounds");
        }

        Console.WriteLine();
        for (int i = 0; i < 6; i++)
        {
            x = fs[i, i];
            if (!fs.ErrFlag) Console.Write(x + " ");
            else
                Console.Write("\nfs[" + i + ", " + i + "] out-of-bounds");
        }
        Console.WriteLine();
    }
}
```

خروجی:



مبحث Indexer به پایان رسید، در قسمت بعد با Properties آشنا خواهید شد.