نوشته‌ی مسعود درویشیان

لینک مستقیم این مطلب در وب‌تارگت

Catch کردن exception کلاس‌های مشتق شده

هنگام گرفتن exception type هایی که شامل base و derived class هستند، باید به چیدمان و نحوه‌ی قرار گرفتن دنباله‌ی catch ها دقت کنید زیرا یک catch برای یک base class با تمام کلاس‌های مشتق شده از آن، تطابق دارد. برای مثال، به‌دلیل این‌که کلاس Exception، کلاس والد تمام exception های دیگر است، گرفتن آن موجب گرفتن تمام exception های موجود می‌شود. البته (همان‌طور که قبلاً توضیح داده شد) استفاده از catch بدون مشخص کردن exception type، یک راه دیگر (و خواناتر) برای گرفتن تمامی exception ها است. با این حال، باید دقت کنید که گرفتن derived class exceptions (مخصوصاً) هنگامی‌که exception های خودتان را می‌سازید، از اهمیت بالایی برخوردار است.

اگر می‌خواهید هم exception های base class و هم exception های derived class را بگیرید، باید در دنباله‌ی نوشتن catch ها، نوع derived class را در ابتدا قرار دهید. این کار ضروری است زیرا یک base class catch تمام derived class ها را catch می‌کند. خوشبختانه قانون ذکر شده در سی‌شارپ ضروری است و در صورت عدم رعایت آن با خطای compile-time مواجه می‌شوید.

برنامه‌ی زیر دو exception کلاس با نام‌های ExceptA و ExceptB می‌سازد. ExceptA از کلاس Exception و ExceptB از ExceptA ارث‌بری کرده است. سپس برنامه، exception هر یک از type ها را throw می‌کند. برای این‌که برنامه مختصر باشد، custom exception ها تنها یک constructor را فراهم می‌آورند، که یک رشته را می‌گیرد و خطا را شرح می‌دهد. اما به‌یاد داشته باشید که در برنامه‌های واقعی‌تان باید تمام constructor های کلاس Exception را در custom exception خودتان وارد کنید.

```
using System;

// Create an exception.
class ExceptA : Exception
{
    public ExceptA(string message) : base(message) { }
```

```
    public override string ToString()
    {
        return Message;
    }
}

// Create an exception derived from ExceptA.
class ExceptB : ExceptA
{
    public ExceptB(string message) : base(message) { }
    public override string ToString()
    {
        return Message;
    }
}
class OrderMatters
{
    static void Main()
    {
        for (int x = 0; x < 3; x++)
        {
            try
            {
                if (x == 0) throw new ExceptA("Caught an ExceptA exception");
                else if (x == 1) throw new ExceptB("Caught an ExceptB exception");
                else throw new Exception();
            }
            catch (ExceptB exc)
            {
                Console.WriteLine(exc);
            }
            catch (ExceptA exc)
            {
                Console.WriteLine(exc);
            }
            catch (Exception exc)
            {
                Console.WriteLine(exc);
            }
        }
    }
}

/* Output

Caught an ExceptA exception
Caught an ExceptB exception
System.Exception: Exception of type 'System.Exception' was thrown.
    at OrderMatters.Main()

*/
```

در برنامه‌ی بالا، به نوع و ترتیب قرار گرفتن catch ها دقت کنید. این تنها ترتیبی است که آن‌ها می‌توانند داشته باشند. از آن‌جا که ExcepB از ExceptA مشتق شده است، catch مربوط به ExceptB باید قبل از ExceptA واقع شود. به همین

ترتیب، catch مربوط به کلاس Exception (که base class تمامی exception ها است) باید در آخر قرار گیرد. می‌توانید با جابه‌جا کردن ترتیب catch ها، ببینید که برنامه هنگام اجرا با خطای compile-time مواجه می‌شود.

یکی از مزایای استفاده از catch کردن base class این است که می‌توانید یک دسته‌بندی کلی از exception ها را catch کنید. برای مثال، اگر خطای به‌وجود آمده به هیچ‌کدام یک از catch ها مطابقت نداشت، catch کردن base class موجب می‌شود در نهایت خطا گرفته شود.

استفاده از کلمات کلیدی checked و unchecked

یکی از ویژگی‌های خاص سی‌شارپ مربوط به تولید خطاهای overflow در هنگام محاسبات ریاضی است. همان‌طور که می‌دانید، ممکن است در بعضی از محاسبات ریاضی، نتیجه‌ی تولید شده از حد و اندازه‌ی data type مربوطه بالاتر رود. هنگامی‌که چنین اتفاقی می‌افتد، باعث به‌وجود آمدن overflow (سرریز) می‌شود.

برای نمونه، به قطعه کد زیر توجه کنید:

```
byte a, b, result;
a = 127;
b = 127;

result = (byte)(a * b);
```

در این‌جا، حاصل ضرب a و b از حد مقدار byte فراتر می‌رود. از این‌رو، این حاصل‌ضرب موجب می‌شود تا overflow ایجاد شود.

سی‌شارپ به شما اجازه می‌دهد تا با استفاده از کلمات کلیدی checked و unchecked، هنگامی‌که overflow رخ می‌دهد، یک exception به‌وجود آورید (یا از تولید exception جلوگیری کنید). برای مشخص کردن این‌که یک عبارت برای overflow بررسی شود، از کلمه‌ی کلیدی checked استفاده کنید. برای مشخص کردن این‌که overflow نادیده گرفته شود، از unchecked استفاده کنید. در مورد بالا، نتیجه‌ی حاصل ضرب برای تطابق یافتن با data type مورد نظر، کوتاه می‌شود. کلمه‌ی کلیدی checked به دو صورت مورد استفاده قرار می‌گیرد. در حالت اول فقط یک عبارت مورد بررسی قرار می‌گیرد که به آن operator form می‌گویند. در حالت دوم یک بلوک از کد مورد بررسی قرار می‌گیرد که به آن statement form گفته می‌شود.

```
checked (expr)

checked {
// statements to be checked
}
```

در این‌جا، expr عبارتی است که مورد بررسی قرار می‌گیرد. اگر یک عبارت بررسی شده overflow شود، یک OverFlowException پرتاب خواهد شد. کلمه‌ی کلیدی unchecked نیز به دو صورت نوشته می‌شود. حالت اول operator form است که overflow را برای یک عبارت خاص نادیده گرفته و حالت دوم، overflow را برای یک بلوک کد نادیده می‌گیرد:

```
unchecked (expr)

unchecked {
// statements for which overfl ow is ignored
}
```

در این‌جا، expr عبارتی است که برای overflow بررسی نمی‌شود. در این مواقع، هنگامی‌که overflow رخ می‌دهد، کوتاه‌سازی انجام خواهد شد.

به مثال زیر که در آن checked و unchecked شرح داده شده است، دقت کنید:

```
using System;
class CheckedDemo
{
    static void Main()
    {
        byte a, b;
        byte result;

        a = 127;
        b = 127;

        try
        {
            result = unchecked((byte)(a * b)); // truncation will accur
            Console.WriteLine("Unchecked result: " + result);

            result = checked((byte)(a * b)); // this causes exception
            Console.WriteLine("Checked result: " + result); // won't execute
        }
        catch (OverflowException exc)
        {
            Console.WriteLine(exc);
        }
    }
}

/* Output

Unchecked result: 1
```

```
System.OverflowException: Arithmetic operation resulted in an overflow.
    at CheckedDemo.Main()

*/
```

همان‌طور که مشاهده می‌کنید، در قسمت unchecked کوتاه‌سازی انجام شده اما در قسمت checked یک exception پرتاب شده است. در مثال قبل، استفاده از checked و unchecked برای یک عبارت شرح داده شد. در مثال بعدی استفاده از این دو کلمه‌ی کلیدی را برای بلوکی از کد مشاهده خواهید کرد:

```
using System;
class CheckedBlocks
{
    static void Main()
    {
        byte a, b;
        byte result;

        a = 127;
        b = 127;

        try
        {
            unchecked
            {
                a = 127;
                b = 127;
                result = unchecked((byte)(a * b));
                Console.WriteLine("Unchecked result: " + result);

                a = 125;
                b = 5;
                result = unchecked((byte)(a * b));
                Console.WriteLine("Unchecked result: " + result);
            }
            checked
            {
                a = 2;
                b = 7;
                result = checked((byte)(a * b)); // this is OK
                Console.WriteLine("Checked result: " + result);

                a = 127;
                b = 127;
                result = checked((byte)(a * b)); // this causes exception
                Console.WriteLine("Checked result: " + result); // won't execute
            }
        }
        catch (OverflowException exc)
        {
            Console.WriteLine(exc);
        }
    }
}

/* Output
```

```
Unchecked result: 1
Unchecked result: 113
Checked result: 14
System.OverflowException: Arithmetic operation resulted in an overflow.
    at CheckedBlocks.Main()

 */
```

همان‌طور که می‌بینید، unchecked block موجب شده تا پس از سرریز، کوتاه‌سازی انجام شود اما در checked block بعد اینکه سرریز اتفاق افتاده، یک exception پرتاب شده است.

Delegates، Events و lambda expressions

این بخش را با تعریف اصطلاح delegate شروع می‌کنیم. به زبان ساده، یک delegate برابر است با شیءای که می‌تواند به یک method رجوع کند. بنابراین هنگامی‌که یک delegate می‌سازید، در واقع یک object را به‌وجود می‌آورید که می‌تواند reference به یک متد را در خودش نگاه دارد. از این‌رو، متد می‌تواند از طریق این reference فراخوانی شود.

فرم کلی delegate به‌صورت زیر است:

```
delegate ret-type name(parameter-list);
```

در این‌جا، ret-type نوع بازگشتی متدی است که delegate آن را فراخوانی می‌کند. Name برابر با نام delegate است. پارامترهای مورد نیاز متد که از طریق delegate فراخوانی می‌شوند در قسمت parameter-list قرار می‌گیرد.

به مثال زیر توجه کنید:

```
using System;
using System.IO;

delegate string StrMod(string str);

class DelegateTest
{
    // Replaces spaces with hyphens.
    static string ReplaceSpaces(string s)
    {
        Console.WriteLine("Replacing spaces with hyphens.");
        return s.Replace(' ', '-');
    }
    // Remove spaces.
    static string RemoveSpaces(string s)
    {
        string temp = "";
        int i;
```

```
        Console.WriteLine("Removing spaces.");
        for (i = 0; i < s.Length; i++)
            if (s[i] != ' ') temp += s[i];
        return temp;
    }
    // Reverse a string.
    static string Reverse(string s)
    {
        string temp = "";
        int i, j;
        Console.WriteLine("Reversing string.");
        for (j = 0, i = s.Length - 1; i >= 0; i--, j++)
            temp += s[i];
        return temp;
    }

    public static void Main()
    {
        StrMod strOp = new StrMod(ReplaceSpaces);
        string str;

        // Call methods through the delegate.
        str = strOp("This is a test.");
        Console.WriteLine("Resulting string: " + str);
        Console.WriteLine();

        strOp = new StrMod(RemoveSpaces);
        str = strOp("This is a test.");
        Console.WriteLine("Resulting string: " + str);
        Console.WriteLine();

        strOp = new StrMod(Reverse);
        str = strOp("This is a test.");
        Console.WriteLine("Resulting string: " + str);
    }
}

/* Output

Replacing spaces with hyphens.
Resulting string: This-is-a-test.

Removing spaces.
Resulting string: Thisisatest.

Reversing string.
Resulting string: .tset a si sihT

 */
```

در مثال بالا یک delegate تعریف کردیم که return type آن از نوع string است و در قسمت parameter-list خود یک string می‌گیرد. بنابراین این delegate تنها به متدهایی می‌تواند رجوع کند که همین signature را داشته باشند. همان‌طور که مشاهده می‌کنید تعدادی متد تعریف کرده‌ایم که signature آن‌ها با delegate تعریف شده تطابق دارد. هنگامی‌که یک شیء از delegate می‌سازید، نام متد مربوطه را (تنها نام متد، بدون پارامتر) به delegate می‌دهیم:

```
StrMod strOp = new StrMod(ReplaceSpaces);
```

سپس از طریق delegate متد را فراخوانی می‌کنیم:

```
str = strOp("This is a test.");
```

به این ترتیب متد ReplaceSpaces با پارامتر This is a test فراخوانی می‌شود و سپس رشته‌ی ویرایش شده در str قرار می‌گیرد. در قسمت بعد مشاهده می‌کنید که delegate به متدهای دیگری نیز وصل شده و آن‌ها را فراخوانی کرده است.

به مثال بعدی delegate دقت کنید:

```
using System;

class Program
{
    delegate string UppercaseDelegate(string input);

    static string UppercaseFirst(string input)
    {
        char[] buffer = input.ToCharArray();
        buffer[0] = char.ToUpper(buffer[0]);
        return new string(buffer);
    }

    static string UppercaseLast(string input)
    {
        char[] buffer = input.ToCharArray();
        buffer[buffer.Length - 1] = char.ToUpper(buffer[buffer.Length - 1]);
        return new string(buffer);
    }

    static string UppercaseAll(string input)
    {
        return input.ToUpper();
    }

    static void WriteOutput(string input, UppercaseDelegate del)
    {
        Console.WriteLine("Your string before: {0}", input);
        Console.WriteLine("Your string after: {0}", del(input));
    }

    static void Main()
    {
        // Wrap the methods inside delegate instances and pass to the method.
        WriteOutput("perls", new UppercaseDelegate(UppercaseFirst));
        WriteOutput("perls", new UppercaseDelegate(UppercaseLast));
        WriteOutput("perls", new UppercaseDelegate(UppercaseAll));
    }
}

/* Output

Your string before: perls
Your string after: Perls
```

```
Your string before: perls
Your string after: perlS
Your string before: perls
Your string after: PERLS

*/
```

در این مثال نیز یک delegate تعریف شده است که نوع بازگشتی و پارامتر ورودی آن string است. از طریق متد WriteOutput() می‌توانیم این delegate را به متدهای دل‌خواه وصل کرده و نتیجه را مشاهده کنیم.

استفاده از delegate دو مزیت دارد. Delegates از events پشتیبانی می‌کند (که در قسمت بعد مشاهده خواهید کرد) و دیگر این‌که delegate موجب می‌شود تا برنامه شما بتواند در runtime (زمان اجرا) متدها را اجرا کند بدون اینکه بداند آن متدها در compile time چه چیزی هستند. این قابلیت زمانی مفید واقع می‌شود که (به‌عنوان مثال) در حال ساخت یک framework هستید و از این طریق component ها را به برنامه‌تان plug in می‌کنید.