نوشته‌ی مسعود درویشیان

لینک مستقیم این مطلب در وب‌تارگت

گرفتن تمام exception ها

بعضی وقت‌ها، ممکن است بخواهید تمام exception ها را بدون در نظر گرفتن نوع آن‌ها، بگیرید. برای انجام این‌کار، یک مدل catch مشخص می‌کنید که exception type و exception variable ندارد.

فرم کلی آن به‌شکل زیر است:

```
catch {
    // handle exceptions
}
```

خط کد بالا باعث به‌وجود آمدن یک catch all exception handler می‌شود و تضمین می‌کند که تمامی exception های به‌وجود آمده گرفته شوند.

به مثال زیر دقت کنید:

```
// Use the "catch all" catch.
using System;
class ExcDemo5
{
    static void Main()
    {
        // Here, numer is longer than denom.
        int[] numer = { 4, 8, 16, 32, 64, 128, 256, 512 };
        int[] denom = { 2, 0, 4, 4, 0, 8 };

        for (int i = 0; i < numer.Length; i++)
        {
            try
            {
                Console.WriteLine(numer[i] + " / " +
                denom[i] + " is " +
                numer[i] / denom[i]);
            }
            catch
            {
                // A "catch-all" catch.
                Console.WriteLine("Some exception occurred.");
            }
        }
    }
}
```

```
/* Output

4 / 2 is 2
Some exception occurred.
16 / 4 is 4
32 / 4 is 8
Some exception occurred.
128 / 8 is 16
Some exception occurred.
Some exception occurred.

*/
```

نکته‌ی مهم دیگر این است که catch all exception handler باید آخرین catch در لیست catch ها باشد.

دقت داشته باشید که نباید در همه‌ی موارد از catch all handler استفاده کنید و به‌طور معمول بهتر است که هر نوع exception را جداگانه handle کنید. همچنین handle کردن تمام exception ها درون یک handler مشکل است. بنابراین از catch all handler در شرایط خاصی استفاده می‌شود و نباید همیشه exception ها را به‌طور کلی با استفاده از آن handle کرد.

Try block های تو در تو

یک try block می‌تواند درون یک try block دیگر قرار گیرد. Exception به‌وجود آمده در try block داخلی که توسط catch مرتبط با همان try گرفته نشده باشد، می‌تواند توسط catch مربوط به try block خارجی گرفته شود.

در مثال زیر، IndexOutOfRangeException توسط try block داخلی گرفته نشده و try block خارجی آن را می‌گیرد:

```
// Use a nested try block.
using System;
class NestTrys
{
    static void Main()
    {
        // Here, numer is longer than denom.
        int[] numer = { 4, 8, 16, 32, 64, 128, 256, 512 };
        int[] denom = { 2, 0, 4, 4, 0, 8 };

        try
        {
            // outer try
            for (int i = 0; i < numer.Length; i++)
            {
                try
                {
```

```
                    // nested try
                    Console.WriteLine(numer[i] + " / " +
                        denom[i] + " is " +
                        numer[i] / denom[i]);
                }
                catch (DivideByZeroException)
                {
                    Console.WriteLine("Can't divide by Zero!");
                }
            }
        }
        catch (IndexOutOfRangeException)
        {
            Console.WriteLine("No matching element found.");
            Console.WriteLine("Fatal error -- program terminated.");
        }
    }
}

/* Output

4 / 2 is 2
Can't divide by Zero!
16 / 4 is 4
32 / 4 is 8
Can't divide by Zero!
128 / 8 is 16
No matching element found.
Fatal error -- program terminated.

*/
```

در این مثال، exception ای که می‌تواند توسط try block داخلی handle شود (در این‌جا divide-by-zero)، به برنامه اجازه می‌دهد تا ادامه یابد. اما exception به‌وجود آمده به‌دلیل گذشتن از حد آرایه، توسط try block خارجی گرفته شده و موجب به پایان رسیدن برنامه می‌شود.

Try block های تودرتو به error های مختلف اجازه می‌دهد تا از روش‌های متفاوتی handle شوند. بعضی از error ها قابل اصلاح کردن نیستند و بعضی دیگر خطاهای کوچکی هستند که می‌توانند بلافاصله درست شوند. بیشتر برنامه‌نویسان از try block خارجی برای handle کردن خطاهایی که سخت قابل اصلاح کردن هستند استفاده می‌کنند و از try block داخلی برای درست کردن خطاهایی که راحت‌تر اصلاح می‌شوند، بهره می‌برند. شما همچنین می‌توانید از try block خارجی به‌عنوان catch all handler برای handle کردن error هایی که در try block داخلی handle نشده‌اند، استفاده کنید.

پرتاب کردن یک Exception

در مثال‌های قبل، exception هایی که به‌طور خودکار توسط runtime system تولید شده بودند، گرفته می‌شدند. اما شما می‌توانید به‌صورت دستی یک exception را با استفاده از کلمه‌کلیدی throw پرتاب کنید.

فرم کلی آن به شکل زیر است:

```
throw exceptOb;
```

در این‌جا، exceptOb باید یک شیء از کلاس یک exception باشد که از Exception ارث‌بری کرده است.

به مثال زیر دقت کنید:

```
using System;
class ThrowDemo
{
    static void Main()
    {
        try
        {
            Console.WriteLine("Before throw.");
            throw new DivideByZeroException();
        }
        catch (DivideByZeroException)
        {
            Console.WriteLine("Exception caught.");
        }
        Console.WriteLine("After try/catch statement.");
    }
}

/* Output

Before throw.
Exception caught.
After try/catch statement.

*/
```

همان‌طور که می‌بینید، DivideByZeroException در قسمت throw با استفاده از new ساخته شده است. به یاد داشته باشید که throw یک شیء را پرتاب می‌کند. بنابراین شما باید یک شیء برای آن بسازید تا آن را پرتاب کند. این بدان معناست که نمی‌توانید یک type را پرتاب کنید. در این مورد، برای ساخت شیء DivideByZeroException از default constructor استفاده شده است اما constructor های دیگر نیز برای exception ها موجود هستند. در اکثر موارد، exception هایی که پرتاب می‌کنید اشیای exception class هایی هستند که خودتان ساخته‌اید. در ادامه‌ی این مبحث متوجه خواهید شد که چگونه exception class های خودتان را بسازید.

پرتاب مجدد یک exception

یک exception گرفته شده توسط یک catch می‌تواند مجدداً پرتاب شود و این exception سپس می‌تواند توسط یک outer catch گرفته شود. یکی از مهم‌ترین دلایل پرتاب مجدد یک exception این است handler های بیشتری می‌توانند به exception دسترسی داشته باشند. به‌عنوان مثال، ممکن است exception handler اول، یک جنبه از exception و exception handler دوم، جنبه‌ی دیگری از exception به‌وجود آمده را handle کند. برای پرتاب مجدد یک exception کافی است فقط به تنهایی از کلمه‌ی throw استفاده کنید. به‌شکل زیر:

```
throw;
```

یه‌یاد داشته باشید هنگامی‌که یک exception را مجدداً پرتاب می‌کنید، این exception باید توسط یک outer block گرفته شود.

برنامه‌ی زیر پرتاب مجدد یک exception را نشان می‌دهد:

```
using System;
class Program
{
    static void Main()
    {
        // Comment out the first 1-2 method invocations.
        try
        {
            A();
            B();
            C(null);
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
    }

    static void A()
    {
        // Rethrow syntax.
        try
        {
            int value = 1 / int.Parse("0");
        }
        catch
        {
            throw;
        }
    }

    static void B()
    {
```

```
        // Filtering exception types.
        try
        {
            int value = 1 / int.Parse("0");
        }
        catch (DivideByZeroException ex)
        {
            throw ex;
        }
    }

    static void C(string value)
    {
        // Generate new exception.
        if (value == null)
        {
            throw new ArgumentNullException("value");
        }
    }
}

/* Output

Attempted to divide by zero.

//// if you comment out the first and second method invocations:
Value cannot be null.
Parameter name: value

*/
```

اگر برنامه‌ی بالا را اجرا کنید پیغام Attempted to divide by zero را مشاهده خواهید کرد. در این هنگام، تنها متد A() فراخوانی شده است و متدهای B() و C() فراخوانی نمی‌شوند. هنگامی که در ابتدا متد A() اجرا می‌شود، exception به‌وجود آمده (که ابتدا درون متد A() یک‌بار throw و یک‌بار catch شده است) دوباره throw شده و به متد Main() فرستاده می‌شود. اکنون درون متد Main() پس از فراخوانی A() یک exception به‌وجود آمده است که باید handle شود. بنابراین بلافاصله کنترل برنامه به catch درون Main() داده شده و پیغام خطای به‌وجود آمده نمایش داده می‌شود. از این‌رو متدهای B() و C() دیگر اجرا نمی‌شوند. اگر متدهای A() و B() را comment کنید خروجی متفاوتی می‌بینید. در مورد متد C() دیگر پرتاب مجدد exception نداریم زیرا exception به‌وجود آمده برای اولین‌بار throw شده و درون متد اصلی برنامه handle می‌شود.

به مثال دیگری در این مورد توجه کنید:

```
// Rethrow an exception.
using System;
class Rethrow
{
    public static void GenException()
    {
```

```
        // Here, numer is longer than denom.
        int[] numer = { 4, 8, 16, 32, 64, 128, 256, 512 };
        int[] denom = { 2, 0, 4, 4, 0, 8 };

        for (int i = 0; i < numer.Length; i++)
        {
            try
            {
                Console.WriteLine(numer[i] + " / " +
                denom[i] + " is " +
                numer[i] / denom[i]);
            }
            catch (DivideByZeroException)
            {
                Console.WriteLine("Can't divide by Zero!");
            }
            catch (IndexOutOfRangeException)
            {
                Console.WriteLine("No matching element found.");
                throw; // rethrow the exception
            }
        }
    }
}
class RethrowDemo
{
    static void Main()
    {
        try
        {
            Rethrow.GenException();
        }
        catch (IndexOutOfRangeException)
        {
            // recatch exception
            Console.WriteLine("Fatal error -- " + "program terminated.");
        }
    }
}

/* Output

4 / 2 is 2
Can't divide by Zero!
16 / 4 is 4
32 / 4 is 8
Can't divide by Zero!
128 / 8 is 16
No matching element found.
Fatal error -- program terminated.

*/
```

در این برنامه، DivideByZeroException در متد GenException() به‌وجود آمده و در همان‌جا نیز handle شده است. اما handle کردن IndexOutOfRangeException به متد Main() واگذار شده است.

استفاده از finally

گاهی‌اوقات شما می‌خواهید یک بلوک از کد حتماً پس از try/catch اجرا شود. برای مثال، ممکن است یک exception باعث شود تا ادامه‌ی اجرای یک متد پایان یابد اما آن متد یک فایل یا یک network connection را باز کرده است که حتماً باید در نهایت بسته شود. این چنین شرایطی در برنامه‌نویسی زیاد هستند و سی‌شارپ راه حل ساده و مناسبی برای آن ارائه داده که این راه حل، استفاده از finally block است. Finally block باید در انتهای دنباله‌ی catch ها قرار بگیرد. فرم کلی try/catch که شامل finally است، به‌صورت زیر می‌باشد:

```
try {
    // block of code to monitor for errors
}
catch (ExcepType1 exOb) {
    // handler for ExcepType1
}
catch (ExcepType2 exOb) {
    // handler for ExcepType2
}...
finally {
    // fi nally code
}
```

Finally block تحت هر شرایطی اجرا می‌شود. این بدان معناست که مهم نیست try block با موفقیت اجرا شود یا خیر، در نهایت finally block اجرا خواهد شد.

به مثال زیر توجه کنید:

```
// Use finally.
using System;
class UseFinally
{
    public static void GenException(int what)
    {
        int t;
        int[] nums = new int[2];
        Console.WriteLine("Receiving " + what);
        try
        {
            switch (what)
            {
                case 0:
                    t = 10 / what; // generate div-by-zero error
                    break;
                case 1:
                    nums[4] = 4; // generate array index error
                    break;
                case 2:
                    return; // return from try block
            }
```

```
        }
        catch (DivideByZeroException)
        {
            Console.WriteLine("Can't divide by Zero!");
            return; // return from catch
        }
        catch (IndexOutOfRangeException)
        {
            Console.WriteLine("No matching element found.");
        }
        finally
        {
            Console.WriteLine("Leaving try.");
        }
    }
}
class FinallyDemo
{
    static void Main()
    {
        for (int i = 0; i < 3; i++)
        {
            UseFinally.GenException(i);
            Console.WriteLine();
        }
    }
}

/* Output

Receiving 0
Can't divide by Zero!
Leaving try.

Receiving 1
No matching element found.
Leaving try.

Receiving 2
Leaving try.

*/
```

همان‌طور که خروجی نشان می‌دهد، مهم نیست برنامه به چه طریقی از try block خارج می‌شود، finally block همیشه اجرا خواهد شد. از لحاظ تکنیکی، هنگامی‌که یک finally block دقیقاْ بعد از یک try block قرار گیرد دیگر catch block نمی‌تواند بعد از آن‌ها بیاید و finally block بعد از try block اجرا خواهد شد اما هیچ exception ای handle نشده است.