زنگ سی‌شارپ – قسمت چهل و ششم

نوشته‌ی مسعود درویشیان

لینک مستقیم این مطلب در وب‌تارگت

در قسمت‌های قبل با یکی از مهم‌ترین بخش‌های سی‌شارپ، interface، آشنا شدید. در این قسمت به مبحث بسیار مهم exception handling که یک سیستم خطاگیری بسیار قدرتمند در سی‌شارپ است، می‌پردازیم.

Exception Handling

یک exception خطایی است که در runtime (زمان اجرا) اتفاق می‌افتد. با استفاده از زیرسیستم exception-handling در سی‌شارپ، شما می‌توانید از یک روش کنترل شده و سازمان‌یافته، خطاهای runtime را handle کنید. یکی از مزیت‌های اصلی exception handling این است که به‌طور خودکار خطاگیری را انجام می‌دهد و این درصورتی است که پیش از به‌وجود آمدن این ویژگی در برنامه‌نویسی، باید خودتان خطاگیری را انجام می‌دادید که هم خسته‌کننده و هم مستعد خطا بود. Exception handling یک بلاک کد (که exception handler نامیده می‌شود) تعریف می‌کند که هنگام بروز خطا به‌صورت خودکار اجرا می‌شود. بنابراین دیگر نیازی نیست که به‌صورت دستی موفق بودن یا عدم موفق بودن هر قسمت از برنامه را بررسی کنید. اگر یک خطا در runtime به‌وجود آید توسط exception handler بررسی خواهد شد. سی‌شارپ exception های استاندارد را برای خطاهای رایج در یک برنامه (مانند خطاهای divide-by-zero و index-out-of-range) تعریف می‌کند که این موضوع یکی دیگر از دلایل اهمیت exception handling این است.

کلاس System.Exception

در سی‌شارپ، exception ها توسط کلاس‌ها ارائه می‌شوند. همه‌ی کلاس‌های exception (مثل کلاس‌های استاندارد دات‌نت برای خطاگیری) باید از کلاس Exception مشتق شوند که خودش بخشی از System namespace است. بنابراین همه‌ی exception ها زیرکلاس Exception هستند.

یکی از زیرکلاس‌های مهم Exception، کلاس SystemException است که مشخص‌کننده‌ی base class برای exception های از پیش تعریف شده در System namespace است. کلاس SystemException چیزی را به کلاس Exception نمی‌افزاید بلکه فقط در صدر زنجیره‌ی exception های استاندارد دات‌نت فریم‌ورک قرار می‌گیرد.

دات‌نت فریم‌ورک exception های توکار (built-in) بسیار زیادی را تعریف می‌کند که از SystemException ارث‌بری می‌کنند. برای مثال، هنگامی‌که خطای تقسیم بر صفر رخ می‌دهد، یک exception از نوع DivideByZeroExcepion به‌وجود می‌آید. به‌زودی متوجه خواهید شد که چگونه کلاس‌های exception خودتان را با ارث‌بری از کلاس Excepion بنویسید.

اصول Exception Handling

Exception handling در سی‌شارپ توسط چهار کلمه‌کلیدی try، catch، throw و finally مدیریت می‌شود. این‌ها یک زیرسیستم مرتبط را به‌وجود می‌آورند که استفاده از هرکدام، اشاره به استفاده از دیگری دارد. در طول بررسی مبحث exception handling هر کدام از کلمات کلیدی با جزییات توضیح داده خواهند شد اما نگاهی مختصر به وظایف هرکدام می‌تواند در این‌جا مفید واقع شود.

آن قسمت از خط کدهای برنامه که قصد دارید خطاهای (exceptions) آن را بررسی کنید، درون try block قرار می‌گیرند. اگر یک exception درون try block رخ دهد، این exception (به اصطلاح) پرتاب (throw) می‌شود. کد شما می‌تواند این exception را در قسمت catch block دریافت و به روشی منطقی آن را handle کند. Exception های استاندارد سیستم، خودشان به‌صورت خودکار throw می‌شوند اما برای throw کردن یک exception به‌صورت دستی باید از کلمه‌کلیدی throw استفاده کنید. هر کدی که در نهایت تحت هر شرایطی باید اجرا شود در قسمت finally block قرار میگیرد.

استفاده از try و catch

در قلب exception handling کلمات کلیدی try و catch قرار دارند. این کلمات کلیدی با هم کار می‌کنند و شما نمی‌توانید یک catch بدون try داشته باشید.

فرم کلی بلوک try/catch exception-handling به‌شکل زیر است:

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

در این‌جا، ExcepType نوع exception ای می‌باشد که رخ داده است. هنگامی‌که یک exception پرتاب می‌شود، توسط جزء catch مرتبط با خودش گرفته شده و سپس exception در آن قسمت با یک روش منطقی handle می‌شود. همان‌طور که فرم کلی try/catch در بالا نشان می‌دهد، بیشتر از یک جزء catch می‌تواند به try وابسته باشد. در واقع نوع exception مشخص می‌کند که کدام catch باید اجرا شود. از این‌رو، هنگامی‌که یک exception با یک catch مطابقت داشت، فقط همان catch اجرا می‌شود و بقیه‌ی catch ها نادیده گرفته می‌شوند. هنگامی‌که یک exception گرفته می‌شود، متغیر exOb، مقدار آن را دریافت می‌کند.

در واقع، مشخص کردن exOb (exception variable) اختیاری است. اگر exception handler نیازی به دسترسی به exception object نداشته باشد (که اغلب به همین صورت است)، نیازی به مشخص کردن exOb نیست و مشخص کردن نوع exception به تنهایی کفایت می‌کند. به همین دلیل، اکثر مثال‌هایی که در این‌جا می‌بینید فاقد exOb هستند.

نکته‌ی مهم این است که اگر هیچ exception ای پرتاب نشود، try block به‌صورت معمول اجرا خواهد شد و همه‌ی catch های وابسته به آن نادیده گرفته می‌شوند و اجرا از آخرین catch به بعد ادامه می‌یابد. بنابراین تنها زمانی یک catch اجرا می‌شود که یک exception پرتاب شده باشد.

مثال زیر نشان می‌دهد که چگونه از try/catch استفاده کنیم. همان‌طور که می‌دانید، اگر قصد دسترسی به خارج از حد یک آرایه را داشته باشید با پیغام خطا روبه‌رو می‌شوید. هنگامی‌که این خطا رخ می‌دهد، به‌طور خودکار یک IndexOutOfRangeExeption (که یک exception استانداردِ تعریف شده در دات‌نت فریم‌ورک است)، پرتاب می‌شود.

به مثال زیر توجه کنید:

```
using System;
class MainClass
{
    static void Main()
    {
        int[] nums = new int[4];

        try
        {
            Console.WriteLine("Before exception is generated.");

            // generate an index out-of-bounds exception
            for (int i = 0; i < 10; i++)
            {
                nums[i] = i;
                Console.WriteLine("nums[{0}]: {1}", i, nums[i]);
            }

            Console.WriteLine("this won't be displayed.");
        }
        catch (IndexOutOfRangeException)
        {
            // catch the exception
            Console.WriteLine("Index out-of-bounds!");
        }
        Console.WriteLine("After catch block.");
    }
}

/* Output

Before exception is generated.
nums[0]: 0
nums[1]: 1
nums[2]: 2
nums[3]: 3
Index out-of-bounds!
After catch block.

*/
```

دقت کنید که nums آرایه‌ای از جنس int با ۴ عنصر است. اما حلقه‌ی for سعی دارد تا از index صفر تا ۹ را مقداردهی کند که این کار باعث می‌شود تا IndexOutOfRangeException رخ دهد.

برنامه‌ی بالا چند نکته‌ی کلیدی را در مورد exception handling نشان می‌دهد. کدهایی که قصد مانیتور کردن آن‌ها را دارید در یک try block قرار می‌گیرند. هنگامی‌که یک exception رخ می‌دهد، این exception به بیرون از try block پرتاب شده و توسط catch گرفته می‌شود. در این لحظه، کنترل برنامه به catch block داده شده و try block به پایان می‌رسد. این بدان معناست که catch block فراخوانی نمی‌شود بلکه ادامه‌ی اجرای برنامه به catch block داده می‌شود. به همین دلیل است که عبارت this won't be displayed بعد از حلقه‌ی for هیچ‌گاه نمایش داده نخواهد شد. بعد از این‌که اجرای catch block به پایان رسید، اجرای برنامه از خط کدهای بعد از catch block ادامه خواهد یافت بنابراین این وظیفه‌ی exception handler شماست که مشکل رخ داده را برطرف کند تا برنامه با موفقیت ادامه یابد.

دقت کنید که در قسمت catch از exception variable استفاده نکرده‌ایم. در عوض، تنها مشخص کردن نوع exception (که در این‌جا نوع آن، IndexOutOfRangeException است) کفایت می‌کند. همان‌طور که پیش‌تر ذکر شد، exception variable تنها زمانی مورد نیاز است که بخواهید به exception object دسترسی داشته باشید. exception handler در بعضی موارد می‌تواند از مقدار exception object برای بدست آوردن اطلاعات بیشتر در مورد error، استفاده کند اما در بیشتر موارد تنها کافی است که بدانید یک exception رخ داده است. بنابراین ننوشتن exception variable در exception handler غیرمعمول نیست و کاملاً صحیح است.

همان‌طور که توضیح داده شد، اگر try block هیچ‌گونه exception ای را throw نکند، هیچ‌کدام از catch block ها اجرا نشده و برنامه از ادامه‌ی آخرین catch اجرا می‌شود. برای اثبات این موضوع، در برنامه قبل، حلقه‌ی for را به‌صورت زیر بنویسید:

```
for(int i=0; i < nums.Length; i++) {
```

اکنون دیگر حلقه از حد آرایه‌ی nums فراتر نمی‌رود و عملاً هیچ‌گونه exception ای throw نشده و catch block اجرا نخواهد شد.

همه‌ی کدهای قابل اجرا در try block برای این‌که مشخص شود exception وجود دارد یا خیر، بررسی می‌شوند. این exception ممکن است بعد از فراخوانی یک متد که در try block وجود دارد، رخ دهد. یک exception پرتاب شده توسط یک method که از درون یک try block فراخوانی شده است، می‌تواند توسط همان try block گرفته شود. البته به‌شرطی که متد، خودش exception را نگیرد.

به مثال زیر دقت کنید:

```
using System;
class ExcTest
{
    public static void GenException()
    {
        int[] nums = new int[4];
        Console.WriteLine("Befor exception is generated.");

        // Generate an index out-of-bounds exception.
        for (int i = 0; i < 10; i++)
        {
            nums[i] = i;
            Console.WriteLine("nums[{0}]: {1}", i, nums[i]);
        }
        Console.WriteLine("this won't be displayed.");
    }
}
class MainClass
{
    static void Main()
    {
        try
        {
            ExcTest.GenException();
        }
        catch (IndexOutOfRangeException)
        {
            Console.WriteLine("OOPS! Index out-of-bounds");
        }
        Console.WriteLine("After catch block.");
    }
}

/* Output

Before exception is generated.
nums[0]: 0
nums[1]: 1
nums[2]: 2
nums[3]: 3
OOPS! Index out-of-bounds!
After catch block.

*/
```

همان‌طور که بیان شد، به دلیل این‌که GenException() از درون یک try block فراخوانی شده است، exception ای که درون متد به‌وجود آمده (و catch نشده است)، درون متد Main() گرفته (catch) می‌شود. دقت کنید که اگر این exception درون متد GenException() گرفته شده بود، دیگر به متد Main() فرستاده نمی‌شد.

گرفتن exception های استاندارد، همان‌طور که در مثال‌های قبل دیدید، یک مزیت مهم دارد و آن هم این است که برنامه‌ی شما دیگر به‌صورت غیرعادی پایان نمی‌یابد. هنگامی‌که یک exception پرتاب می‌شود، باید توسط قسمتی از

کد گرفته شود. به‌طور کلی، اگر برنامه شما یک exception را نگیرد، این exception توسط runtime system گرفته خواهد شد. مشکل این‌جاست که runtime system خطا را گزارش می‌دهد و برنامه را می‌بندد.

به مثال زیر که در آن از exception handling استفاده نشده است دقت کنید:

```
using System;
class NotHandled
{
    static void Main()
    {
        int[] nums = new int[4]; Console.WriteLine("Before exception is generated.");
        // Generate an index out-of-bounds exception.
        for (int i = 0; i < 10; i++)
        {
            nums[i] = i;
            Console.WriteLine("nums[{0}]: {1}", i, nums[i]);
        }
    }
}

/* Output

Before exception is generated.
nums[0]: 0
nums[1]: 1
nums[2]: 2
nums[3]: 3

Unhandled Exception: System.IndexOutOfRangeException:
        Index was outside the bounds of the array.
    at NotHandled.Main()

*/
```

اگر برنامه‌ی بالا اجرا کنید می‌بینید که بعد از رخ دادن خطا، برنامه از ادامه‌ی اجرا باز می‌ایستد و پیغام خطایی را به شما نشان می‌دهد و حاکی از آن است که برنامه‌ی شما یک unhandled Exception دارد. اگرچه این‌چنین پیغام خطایی در هنگام debug کردن می‌تواند مفید باشد اما این‌که بقیه نیز بتوانند این پیغام را مشاهده کنند، نمی‌تواند جالب باشد. پس بهتر است که برنامه‌ی شما، خودش exception هایش را handle کند.

همان‌طور که پیش‌تر ذکر شد، نوع exception پرتاب شده باید با نوع exception مشخص شده در catch تطابق داشته باشد. برای مثال در برنامه‌ی زیر یک exception از نوع IndexOutOfRangeException پرتاب می‌شود اما در قسمت catch، نوع exception مشخص شده، DivideByZeroException (یکی دیگر از exception های built-in) است. هنگامی‌که که exception رخ می‌دهد، این exception گرفته نشده و برنامه به‌شکل غیرعادی پایان می‌یابد.

```
// This won't work!
```

```
using System;
class ExcTypeMismatch
{
    static void Main()
    {
        int[] nums = new int[4];
        try
        {
            Console.WriteLine("Before exception is generated.");

            // Generate an index out-of-bounds exception.
            for (int i = 0; i < 10; i++)
            {
                nums[i] = i;
                Console.WriteLine("nums[{0}]: {1}", i, nums[i]);
            }
            Console.WriteLine("this won't be displayed");
        }

        /* Can't catch an array boundary error with a
        DivideByZeroException. */
        catch (DivideByZeroException)
        {
            // Catch the exception.
            Console.WriteLine("Index out-of-bounds!");
        }

        Console.WriteLine("After catch block.");
    }
}
```

اگر برنامه را اجرا کنید متوجه می‌شوید که مشخص کردن exception نامربوط باعث می‌شود catch block اجرا نشده و برنامه به‌صورت غیرعادی متوقف شود.

Exception ها به شما اجازه می‌دهند تا خطاها را به‌شکلی مطلوب handle کنید. یکی از مزیت‌های کلیدی exception handling این است که می‌تواند به هر خطا پاسخ دهد و به اجرای ادامه‌ی برنامه بپردازد. برای نمونه، مثال زیر عناصر یک آرایه را بر عناصر یک آرایه‌ی دیگر تقسیم می‌کند و اگر خطای تقسیم بر صفر رخ دهد، یک DivideByZeroException تولید می‌شود. در برنامه، این exception تنها با گزارش یک پیغام handle می‌شود و اجرای برنامه ادامه می‌یابد.

به مثال زیر دقت کنید:

```
using System;
class ExcDemo
{
    static void Main()
    {
        int[] numer = { 4, 8, 16, 32, 64, 128 };
        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)
            {
                // Catch the exception.
                Console.WriteLine("Can't divide by Zero!");
            }
        }
    }
}

/* 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

*/
```

از این‌رو، هنگامی که قصد دارید یک عدد را بر صفر تقسیم کنید، اجرای برنامه‌ی شما به یک خطای runtime به‌طوری ناگهانی پایان نمی‌یابد بلکه به‌روشی تمیز و مرتب یک پیغام را نمایش می‌دهید که این کار غیر ممکن است و به اجرای ادامه‌ی برنامه می‌پردازید.

شما همچنین می‌توانید بیشتر از یک catch وابسته به یک try داشته باشید. اما هر catch باید نوع متفاوتی از exception را بگیرد.

به مثال زیر دقت کنید:

```
using System;
class ExcDemo
{
    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 (DivideByZeroException)
            {
                Console.WriteLine("Can't divide by Zero!");
            }
            catch (IndexOutOfRangeException)
            {
                Console.WriteLine("No matching element found.");
            }
        }
    }
}

/* 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.
No matching element found.

*/
```

همان‌طور که خروجی نشان می‌دهد، هر catch فقط به exception های مرتبط به خودش پاسخ می‌دهد.

در کل، لیست catch ها از بالا به پایین بررسی می‌شود تا مشخص شود کدام‌یک با exception مطابقت دارد. اولین catch که با exception مطابقت داشته باشد اجرا شده و بقیه‌ی catch ها نادیده گرفته می‌شوند.