

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 در صورتی 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، هنگامی که خطای تقسیم بر صفر رخ می‌دهد، یک

exception از نوع **DivideByZeroException** به وجود می آید. به زودی متوجه خواهید شد که چگونه کلاس های **exception** خودتان را با ارث بری از کلاس **Exception** بنویسید.

اصول Exception Handling

Exception handling در سی شارپ توسط چهار کلمه کلیدی **try**، **catch**، **throw** و **finally** مدیریت می شود. این ها یک زیرسیستم مرتبط را به وجود می آورند که استفاده از هر کدام، اشاره به استفاده از دیگری دارد. در طول بررسی مبحث **exception handling** هر کدام از کلمات کلیدی با جزئیات توضیح داده خواهند شد اما نگاهی مختصر به وظایف هر کدام می تواند در این جا مفید واقع شود.

آن قسمت از خط کدهای برنامه که قصد دارید خطاهای (**exceptions**) آن را بررسی کنید، درون **try block** قرار می گیرند. اگر یک **exception** درون **try block** رخ دهد، این **exception** (به اصطلاح) پرتاب (**throw**) می شود. کد شما می تواند این **exception** را در قسمت **block catch** دریافت و به روشی منطقی آن را **handle** کند. **Exception** های استاندارد سیستم، خودشان به صورت خودکار **throw** می شوند اما برای **throw** کردن یک **exception** به صورت دستی باید از کلمه کلیدی **throw** استفاده کنید. هر کدی که در نهایت تحت هر شرایطی باید اجرا شود در قسمت **finally block** قرار می گیرد.

استفاده از try و catch

در قلب **exception handling** کلمات کلیدی **try** و **catch** قرار دارند. این کلمات کلیدی با هم کار می کنند و شما نمی توانید یک **catch** بدون **try** داشته باشید.

فرم کلی بلوک **try/catch exception-handling** به شکل زیر است:

```
1 try {
2     // block of code to monitor for errors
3 }
4 catch (Exception exOb) {
5     // handler for Exception
```

```

6    }
7    catch (ExceptionType2 exOb) {
8        // handler for ExceptionType2
9    }
10   .
11   .
12   .

```

در این جا، **ExceptionType** نوع **exception** ای می باشد که رخ داده است. هنگامی که یک **exception** پرتاب می شود، توسط جزء **catch** مرتبط با خودش گرفته شده و سپس **exception** در آن قسمت با یک روش منطقی **handle** می شود. همان طور که فرم کلی **try/catch** در بالا نشان می دهد، بیشتر از یک جزء **catch** می تواند به **try** وابسته باشد. در واقع نوع **exception** مشخص می کند که کدام **catch** باید اجرا شود. از این رو، هنگامی که یک **exception** با یک **catch** مطابقت داشت، فقط همان **catch** اجرا می شود و بقیه ی **catch** ها نادیده گرفته می شوند. هنگامی که یک **exception** گرفته می شود، متغیر **exOb** مقدار آن را دریافت می کند.

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 می کند. به همین دلیل، اکثر مثال هایی که در این جا می بینید فاقد **exOb** هستند.

نکته ی مهم این است که اگر هیچ **exception** ای پرتاب نشود، **try block** به صورت معمول اجرا خواهد شد و همه ی **catch** های وابسته به آن نادیده گرفته می شوند و اجرا از آخرین **catch** به بعد ادامه می یابد. بنابراین تنها زمانی یک **catch** اجرا می شود که یک **exception** پرتاب شده باشد.

مثال زیر نشان می دهد که چگونه از **try/catch** استفاده کنیم. همان طور که می دانید، اگر قصد دسترسی به خارج از حد یک آرایه را داشته باشید با پیغام خطا روبه رو می شوید. هنگامی که این خطا رخ می دهد، به طور خودکار یک

IndexOutOfRangeException (که یک **exception** استاندارد تعریف شده در دات نت فریم ورک است)،

پرتاب می شود.

به مثال زیر توجه کنید:

```
1  using System;
2  class MainClass
3  {
4      static void Main()
5      {
6          int[] nums = new int[4];
7
8          try
9          {
10             Console.WriteLine("Before exception is generated.");
11
12             // generate an index out-of-bounds exception
13             for (int i = 0; i < 10; i++)
14             {
15                 nums[i] = i;
16                 Console.WriteLine("nums[{0}]: {1}", i, nums[i]);
17             }
18
19             Console.WriteLine("this won't be displayed.");
20         }
21         catch (IndexOutOfRangeException)
22         {
23             // catch the exception
24             Console.WriteLine("Index out-of-bounds!");
25         }
26         Console.WriteLine("After catch block.");
```

```

27     }
28 }
29
30 /* Output
31
32 Before exception is generated.
33 nums[0]: 0
34 nums[1]: 1
35 nums[2]: 2
36 nums[3]: 3
37 Index out-of-bounds!
38 After catch block.
39
40 */

```

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. هنگامی که 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 این که اجرای **catch block** به پایان رسید، اجرای برنامه از خط کدهای بعد از **catch block** ادامه خواهد یافت بنابراین این وظیفه‌ی **exception handler** شماسست 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 این جا 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 نوشتن **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:

```
1 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 این که 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 به شرطی که متد، خودش **exception** را نگیرد.

به مثال زیر دقت کنید:

```
1 using System;
2 class ExcTest
3 {
4     public static void GenException ()
5     {
6         int[] nums = new int[4];
        Console.WriteLine("Before exception is generated.");
```

```
7
8      // Generate an index out-of-bounds exception.
9      for (inti = 0; i < 10; i++)
10     {
11         nums[i] = i;
12         Console.WriteLine("nums[{0}]: {1}", i, nums[i]);
13     }
14     Console.WriteLine("this won't be displayed.");
15 }
16 }
17 class MainClass
18 {
19     static void Main()
20     {
21         try
22         {
23             ExcTest.GenException();
24         }
25         catch (IndexOutOfRangeException)
26         {
27             Console.WriteLine("OOPS! Index out-of-bounds");
28         }
29         Console.WriteLine("After catch block.");
30     }
31 }
32
33 /* Output
34
35 Before exception is generated.
36 nums[0]: 0
37 nums[1]: 1
```

```

38  nums[2] : 2
39  nums[3] : 3
40  OOPS! Index out-of-bounds!
41  After catch block.
42
43  */
44

```

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. هنگامی که 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:

```

1  using System;
2  class NotHandled
3  {
4      static void Main()
5      {
6          int[] nums = new int[4]; Console.WriteLine("Before exception is
7  generated.");
8          // Generate an index out-of-bounds exception.
9          for (int i = 0; i < 10; i++)
10         {

```



```

11         nums[i] = i;
12         Console.WriteLine("nums[{0}]: {1}", i, nums[i]);
13     }
14 }
15 }
16
17 /* Output
18
19 Before exception is generated.
20 nums[0]: 0
21 nums[1]: 1
22 nums[2]: 2
23 nums[3]: 3
24
25 Unhandled Exception: System.IndexOutOfRangeException:
26     Index was outside the bounds of the array.
27     at NotHandled.Main()
28
29 */

```

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 این چنین پیغام خطایی در هنگام **debug** کردن می‌تواند مفید باشد اما این که 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. هنگامی که که **exception** رخ می‌دهد، این **exception** گرفته نشده و

برنامه به شکل غیرعادی پایان می‌یابد.

```
1  // This won't work!
2  using System;
3  class ExcTypeMismatch
4  {
5      static void Main()
6      {
7          int[] nums = new int[4];
8          try
9          {
10             Console.WriteLine("Before exception is generated.");
11
12             // Generate an index out-of-bounds exception.
13             for (int i = 0; i < 10; i++)
14             {
15                 nums[i] = i;
16                 Console.WriteLine("nums[{0}]: {1}", i, nums[i]);
17             }
18             Console.WriteLine("this won't be displayed");
19         }
20
21         /* Can't catch an array boundary error with a
22         DivideByZeroException. */
23         catch (DivideByZeroException)
24         {
25             // Catch the exception.
26             Console.WriteLine("Index out-of-bounds!");
27         }
28
```

```

29         Console.WriteLine("After catch block.");
30     }
31 }

```

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

Exception ها به شما اجازه می‌دهند تا خطاها را به شکلی 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:

```

1  using System;
2  class ExcDemo
3  {
4      static void Main()
5      {
6          int[] numer = { 4, 8, 16, 32, 64, 128 };
7          int[] denom = { 2, 0, 4, 4, 0, 8 };
8
9          for (int i = 0; i < numer.Length; i++)
10         {
11             try
12             {
13                 Console.WriteLine(numer[i] + " / " +
14                 denom[i] + " is " +
15                 numer[i] / denom[i]);

```

```

16         }
17         catch (DivideByZeroException)
18         {
19             // Catch the exception.
20             Console.WriteLine("Can't divide by Zero!");
21         }
22     }
23 }
24 }
25
26 /* Output
27
28 ۴ / ۲ is 2
29 Can't divide by Zero!
30 ۱۶ / ۴ is 4
31
32 ۳۲ / ۴ is 8
33 Can't divide by Zero!
34 ۱۲۸ / ۸ is 16
35
36 */

```

از این رو، 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:

```
1  using System;
2  class ExcDemo
3  {
4      static void Main()
5      {
6          // Here, number is longer than denom.
7          int[] numer = { 4, 8, 16, 32, 64, 128, 256, 512 };
8          int[] denom = { 2, 0, 4, 4, 0, 8 };
9
10         for (int i = 0; i < numer.Length; i++)
11         {
12             try
13             {
14                 Console.WriteLine(numer[i] + " / " +
15                 denom[i] + " is " +
16                 numer[i] / denom[i]);
17             }
18             catch (DivideByZeroException)
19             {
20                 Console.WriteLine("Can't divide by Zero!");
21             }
22             catch (IndexOutOfRangeException)
23             {
24                 Console.WriteLine("No matching element found.");
25             }
26         }
27     }
28 }
29
30 /* Output
31
```

```
32  ۴ / ۲ is 2
33  Can't divide by Zero!
34  ۱۶ / ۴ is 4
35
36  ۳۲ / ۴ is 8
37  Can't divide by Zero!
38  ۱۲۸ / ۸ is 16
39  No matching element found.
40  No matching element found.
41

*/
```

همان طور که خروجی نشان می دهد، هر **catch** فقط به **exception** های مرتبط به خودش پاسخ می دهد.

در کل، لیست **catch** ها از بالا به پایین بررسی می شود تا مشخص شود کدام یک با **exception** مطابقت دارد. اولین

catch که با **exception** مطابقت داشته باشد اجرا شده و بقیه ی **catch** ها نادیده گرفته می شوند.