



البرمجة الإجرائية Procedural Programming

IPG202

الفصل الخامس معالجة الاستثناءات Exception Handling

الكلمات المفتاحية

الاستثناء، try، catch، throw، finally.

ملخص الفصل

يركز هذا الفصل على مفهوم الاستثناءات، حيث يعرف بمصطلح الاستثناء، ثم يقدم عرضاً لأشكال الاستثناءات المحتملة في البرنامج المكتوب بلغة C# حيث يعرض أنواع الاستثناءات مسبقة التعريف في اللغة وكيفية التقاطها ومعالجتها، ومن ثم يعرفنا على كيفية إنشاء وإلقاء استثناءات خاصة بنا، ويختتم بتقديم بعض الأمثلة التي تساعد على اكتساب مهارات معالجة الاستثناءات.

أهداف الفصل

بنهاية هذا الفصل سيكون الطالب قادراً على:

- تعريف الاستثناء.
- توضيح أسلوب لغة C# في معالجة الاستثناءات.
- استخدام البنية try,catch,finally لمعالجة الاستثناءات.
- إلقاء الاستثناءات الخاصة بالمستخدم ومعالجتها.

محتويات الفصل

1. مقدمة
2. ماهو الاستثناء.
3. النقاط الاستثناءات ومعالجتها.
4. كتل catch غير الصالحة.
5. تداخل الكتل try-catch.
6. الكلمة المفتاحية throw.
7. أصناف الاستثناءات مسبقة التعريف.
8. تمارين وأنشطة.

1- مقدمة.

يمكن تصنيف الأخطاء التي يمكن أن تقع في البرنامج المكتوب بلغة C# إلى ثلاثة أصناف:

الصنف الأول - أخطاء الصيغة syntax error: وهي أخطاء تتجم عن مخالفة قواعد اللغة وأشكال أوامرها وصيغها، وهذه الأخطاء تكتشف دائماً من قبل المترجم compiler ولا ينتقل إلى مرحلة التنفيذ ما لم تتم معالجة هذه الأخطاء وتصحيحها.

الصنف الثاني - الأخطاء المنطقية logic errors: وهي أخطاء تتجم عن عدم التحليل السليم للمسألة المطروحة وبناء الخوارزمية الملائمة، مما يؤدي إلى عمل البرنامج ولكنه يعطي في كثير من الأحيان نتائج غير دقيقة أو يسلك سلوكاً غير مرغوب.

الصنف الثالث - أخطاء وقت التنفيذ runtime errors: وهي أخطاء تحصل في وقت تنفيذ البرنامج وعند بعض الحالات غير المتوقعة، مما يؤدي إلى إحباط البرنامج وإيقاف تنفيذه.

واجهنا خلال جميع مراحل دراستنا العديد من الأخطاء من الصنفين الأول والثاني وتعلمنا كيف نقوم بالتعامل معها وتصحيحها، ونحاول في هذا الفصل الإضاءة على الصنف الأخير ونتعلم كيفية معالجته.

2- ماهو الاستثناء

الاستثناء exception هو عبارة عن خطأ من أخطاء وقت التنفيذ يتم إطلاقه بنتيجة حصول مشكلة غير متوقعة مما يؤدي إلى إيقاف البرنامج.

لتوضيح هذا المفهوم، لنقم بتجربة تنفيذ المقطع التالي:

```
static void Main()
{
    // get numerator
    Console.Write( "Please enter an integer numerator: " );
    int numerator = Convert.ToInt32( Console.ReadLine() );
    // get denominator
    Console.Write( "Please enter an integer denominator: " );
    int denominator = Convert.ToInt32( Console.ReadLine() );
    // divide the two integers, then display the result
    int result = numerator / denominator;
    Console.WriteLine( "\nResult: {0:D} / {1:D} = {2:D}",
        numerator, denominator, result );
} // end Main
```

يمكن أن تكون إحدى حالات التنفيذ من الشكل التالي:

```
C:\WINDOWS\system32\cmd.exe
Please enter an integer numerator: 100
Please enter an integer denominator: 7

Result: 100 / 7 = 14
Press any key to continue . . .
```

لقد عمل البرنامج بشكل سليم وقام بحساب ناتج القسمة، ومن الممكن أن يستمر بالعمل بشكل سليم لفترة طويلة ولكن لا يخلو الأمر من إمكانية حصول حالات غير متوقعة، كما في الحالتين التاليتين:

الحالة الأولى: في حال محاولة إدخال قيمة الصفر للمحول denominator

```
C:\WINDOWS\system32\cmd.exe
Please enter an integer numerator: 100
Please enter an integer denominator: 0

Unhandled Exception: System.DivideByZeroException: Attempted to divide by zero.
   at FirstProgram.Program.Main() in c:\IPG101\FirstProgram\FirstProgram\Program.cs:line 20
Press any key to continue . . .
```

نلاحظ أن البرنامج قد توقف عن التنفيذ بشكل غير متوقع مشيراً إلى حصول استثناء غير متوقع يدعى `DividByZeroException` وذلك بسبب محاولة إجراء عملية تقسيم على الصفر.

الحالة الثانية: في حال محاولة إدخال قيمة غير صحيحة لأحد المتحولين nominator أو denominator

```
C:\WINDOWS\system32\cmd.exe
Please enter an integer numerator: 100
Please enter an integer denominator: hello

Unhandled Exception: System.FormatException: Input string was not in a correct format.
   at System.Number.StringToNumber(String str, NumberStyles options, NumberBuffer& number, NumberFormatInfo info, Boolean parseDecimal)
   at System.Number.ParseInt32(String s, NumberStyles style, NumberFormatInfo info)
   at System.Convert.ToInt32(String value)
   at FirstProgram.Program.Main() in c:\IPG101\FirstProgram\FirstProgram\Program.cs:line 18
Press any key to continue . . .
```

بالمثل، نلاحظ أن البرنامج قد توقف عن التنفيذ بشكل غير متوقع مشيراً إلى حصول استثناء غير متوقع يدعى `FormatException` وذلك بسبب محاولة إدخال قيمة نصية لمتحول رقمي.

في كلا الحالتين يعتبر ظهور مثل هذه الحالات أمراً غير مرغوب وقد يكون ذا كلفة باهظة أثناء تشغيل البرامج، وبالتالي لا بد من إجراء مايلزم لمعالجة مثل هذه الحالات.

3- التقاط الاستثناءات ومعالجتها

يجب أن يتم معالجة الاستثناءات لمنع احتمال توقف البرنامج بشكل غير متوقع، وتسجيل الخطأ الحاصل ومتابعة تنفيذ باقي المهام.

تقدم لغة C# حلاً لالتقاط الاستثناءات ومعالجتها يتمثل بالبنية try catch finally والتي تملك الصيغة العامة التالية:

```
try
{
    // put the code here that may raise exceptions
}
catch
{
    // handle exception here
}
finally
{
    // final cleanup code
}
```

ننوه هنا إلى الأمور الأساسية التالية:

الكلمة try: يجب وضع أي مقطع برمجي يشتبه باحتمال أن يؤدي إلى ظهور استثناءات داخل كتلة try {}. أثناء التنفيذ ، في حالة حدوث استثناء ، ينتقل تدفق عنصر التحكم إلى أول كتلة catch مطابقة.

الكلمة catch: وهي عبارة عن كتلة معالج استثناء حيث يمكنك تنفيذ بعض الإجراءات مثل تسجيل استثناء وتدقيقه. تأخذ الكلمة catch بارامتراً من نوع استثناء يمكنك من خلالها الحصول على تفاصيل استثناء ما.

الكلمة finally: سيتم دائماً تنفيذ الكلمة النهائية finally سواء تم إطلاق استثناء أم لا. عادة ، يجب استخدام الكلمة finally لتحرير الموارد ، على سبيل المثال ، لإغلاق أي دفق أو ملفات تم فتحها في الكلمة try.

ملاحظة: الكلمة finally هي كتلة اختيارية ويمكن حذفها ويجب أن تأتي بعد الكتل try و catch.

على سبيل المثال، لمعالجة الاستثناءات المحتملة في المثال السابق (مثال القسم) يمكن أن نعدل المقطع بحيث يصبح على النحو التالي:

```

static void Main()
{
    try
    {
        // get numerator
        Console.Write( "Please enter an integer numerator: " );
        int numerator = Convert.ToInt32( Console.ReadLine() );
        // get denominator
        Console.Write( "Please enter an integer denominator: " );
        int denominator = Convert.ToInt32( Console.ReadLine() );
        // divide the two integers, then display the result
        int result = numerator / denominator;
        Console.WriteLine( "\nResult: {0:D} / {1:D} = {2:D}",
            numerator, denominator, result );
    }
    catch
    {
        Console.WriteLine( "Error occurred " );
    }
    finally
    {
        Console.WriteLine( "Re-try with a different numbers. " );
    }
} // end Main

```

الآن لنعد محاولة تنفيذ البرنامج بنفس الحالات الثلاث السابقة:

C:\WINDOWS\system32\cmd.exe

```

Please enter an integer numerator: 100
Please enter an integer denominator: 7

Result: 100 / 7 = 14
Re-try with a different number.
Press any key to continue . . .

```

C:\WINDOWS\system32\cmd.exe

```

Please enter an integer numerator: 100
Please enter an integer denominator: 0
Error occurred.
Re-try with a different number.
Press any key to continue . . .

```

```
C:\WINDOWS\system32\cmd.exe
Please enter an integer numerator: 100
Please enter an integer denominator: hello
Error occurred.
Re-try with a different number.
Press any key to continue . . .
```

نلاحظ مايلي:

- تم تنفيذ الكتلة finally في جميع الحالات.
- قام البرنامج بالانتقال إلى الكتلة catch وتنفيذ محتواها في كل مرة وقع فيها استثناء.

في الحالات السابقة التي وقع فيها الاستثناء لم نستطع أن نميز ما هو الاستثناء الحاصل واكتفينا بمعرفة أن خطأ ما قد حصل. قياسياً، يجب أن تتضمن الكتلة catch بارامتراً من الصنف المسبق التعريف exception class (أو من صنف معرف من قبل المستخدم) للحصول على تفاصيل الخطأ.

فيما يلي صيغة معدلة من المثال السابق تتضمن استخدام بارامتر من النمط Exception يقوم بالتقاط جميع أنواع الاستثناءات.

```
static void Main()
{
    try
    {
        // get numerator
        Console.Write( "Please enter an integer numerator: ");
        int numerator = Convert.ToInt32( Console.ReadLine() );
        // get denominator
        Console.Write( "Please enter an integer denominator: ");
        int denominator = Convert.ToInt32( Console.ReadLine() );
        // divide the two integers, then display the result
        int result = numerator / denominator;
        Console.WriteLine( "\nResult: {0:D} / {1:D} = {2:D}",
            numerator, denominator, result );
    }
    catch (Exception ex)
    {
        Console.WriteLine( "Error occurred " + ex.Message);
    }
    finally
    {
        Console.WriteLine( "Re-try with a different numbers. ");
    }
} // end Main
```

بإعادة التجربة مع حالات الاستثناء السابقة نلاحظ أن الخرج قد اختلف وظهر لنا في كل مرة توصيف للاستثناء الذي تم التقاطه:

```
C:\WINDOWS\system32\cmd.exe
Please enter an integer numerator: 100
Please enter an integer denominator: 0
Error occurred.Attempted to divide by zero.
Re-try with a different number.
Press any key to continue . . .
```

```
C:\WINDOWS\system32\cmd.exe
Please enter an integer numerator: 100
Please enter an integer denominator: hello
Error occurred.Input string was not in a correct format.
Re-try with a different number.
Press any key to continue . . .
```

حيث استخدمنا هنا الصفة Message للصنف Exception لعرض توصيف الاستثناء.

ملاحظة: يملك الصنف Exception العديد من الصفات الأخرى التي توصفه ولكننا لن نهتم بها هنا لأنها خارج نطاق دراستنا.

بالنظر إلى المثال الأخير، نلاحظ أن المعالجة نفسها ستحصل لجميع الاستثناءات التي يتم التقاطها من قبل البرنامج، ماذا لو كنا نرغبنا بكتابة معالجة مختلفة لكل نوع من أنواع الاستثناءات، في هذه الحالة يمكن كتابة أكثر من كتلة catch وكل كتلة تستخدم بارامتراً من نوع مخصص من الاستثناءات.

فيما يلي الشكل المعدل من هذا البرنامج الذي يتضمن هذه المعالجة:

```
static void Main()
{
    try
    {
        // get numerator
        Console.WriteLine( "Please enter an integer numerator: " );
        int numerator = Convert.ToInt32( Console.ReadLine() );
        // get denominator
        Console.WriteLine( "Please enter an integer denominator: " );
        int denominator = Convert.ToInt32( Console.ReadLine() );
        // divide the two integers, then display the result
        int result = numerator / denominator;
        Console.WriteLine( "\nResult: {0:D} / {1:D} = {2:D}",
            numerator, denominator, result );
    }
    catch (FormatException formatException)
    {
        Console.WriteLine("\n" + formatException.Message);
        Console.WriteLine("You must enter two integers. Please try again.\n");
    }
    catch (DivideByZeroException divideByZeroException)
    {
        Console.WriteLine("\n" + divideByZeroException.Message);
        Console.WriteLine("Zero is an invalid denominator. Please try again.\n");
    }
    finally
    {
        Console.WriteLine( "Re-try with a different numbers. " );
    }
} // end Main
```

بتكرار نفس الاختبارات السابقة، نحصل على مايلي:

```
C:\WINDOWS\system32\cmd.exe
Please enter an integer numerator: 100
Please enter an integer denominator: 0

Attempted to divide by zero.
Zero is an invalid denominator. Please try again.

Re-try with a different number.
Press any key to continue . . .
```

```
C:\WINDOWS\system32\cmd.exe
Please enter an integer numerator: 100
Please enter an integer denominator: hello

Input string was not in a correct format.
You must enter two integers. Please try again.

Re-try with a different number.
Press any key to continue . . .
```

4- كتل catch غير الصالحة

عند إلقاء استثناء ضمن المقطع try فإن المترجم يقوم بحاولة التقاطه من قبل كتل catch المعرفة بالتتالي، وبالتالي فإن أي تكرار لالتقاط ومعالجة أي استثناء لأكثر من مرة سيعني أن كتلة catch المكررة لا معنى لها ولن تنفذ مطلقاً وبالتالي يعطي المترجم رسالة خطأ في هذه الحالة.

```
static void Main()
{
    try
    {
        //code that may raise an exception
    }
    catch //cannot have both catch and catch(Exception ex)
    {
        Console.WriteLine("Exception occurred");
    }
    catch(Exception ex) //cannot have both catch and catch(Exception ex)
    {
        Console.WriteLine("Exception occurred");
    }
} // end Main
```

سيعطي المقطع السابق رسالة الخطأ التالية:

- ❌ 1 Catch clauses cannot follow the general catch clause of a try statement
- ❌ 2 A previous catch clause already catches all exceptions of this or of a super type ('object')

وتعني:

- 1- الاستثناءان الملتقطان في مقطعي catch هما نفسيهما.
- 2- الكتلة catch بدون بارامتر من نوع Exception أو ببارامتر من نوع Exception عام يجب أن تكون آخر كتلة لأنها تعني معالجة كل الاستثناءات المحتملة... وبالتالي يجب أن توضع كآخر كتلة بحيث تعالج كل ما لم يتم التقاطه من قبل الكتل السابقة.

يوضح المثال التالي الملاحظة الأخيرة:

```
static void Main()
{
    try
    {
        //code that may raise an exception
    }

    catch(Exception ex) // catch(Exception ex) must be the last one
    {
        Console.WriteLine("Exception occurred");
    }
    catch (FormatException formatException)
    {
        Console.WriteLine(formatException.Message);
    }
    catch (DivideByZeroException divideByZeroException)
    {
        Console.WriteLine(divideByZeroException.Message);
    }
} // end Main
```

5- تداخل الكتل try-catch

تتيح لغة C# كتابة كتل try-catch متداخلة، عند استخدام الكتل المتداخلة سيتم التقاط الاستثناء من قبل أول كتلة catch ملائمة تلي الكتلة try التي حصل فيها الاستثناء.

مثال:

```
static void Main(string[] args)
{
    int divider = 0;
    try
    {
        try
        {
            int result = 100/divider;
        }
        catch
        {
            Console.WriteLine("Inner catch");
        }
    }
    catch
    {
        Console.WriteLine("Outer catch");
    }
}
```

بتنفيذ هذا المقطع سيعطي على خرجه العبارة Inner catch.

فيما يلي الشكل المعدل من هذا البرنامج الذي يتضمن هذه المعالجة:

```
static void Main()
{
    int numerator=1, denominator=1;
    try
    {
        try
        {
            // get numerator
            Console.Write("Please enter an integer numerator: ");
            numerator = Convert.ToInt32(Console.ReadLine());
        }
        catch (FormatException formatException)
        {
            Console.WriteLine("\n" + formatException.Message);
        }
    }
}
```

```

        Console.WriteLine("numerator must be integer. Please try again.\n");
    }
    try
    {
        // get denominator
        Console.Write("Please enter an integer denominator: ");
        denominator = Convert.ToInt32(Console.ReadLine());
    }
    catch (FormatException formatException)
    {
        Console.WriteLine("\n" + formatException.Message);
        Console.WriteLine("denominator must be integer. Please try again.\n");
    }
    // divide the two integers, then display the result
    int result = numerator / denominator;
    Console.WriteLine( "\nResult: {0:D} / {1:D} = {2:D}",
        numerator, denominator, result );
}
catch (DivideByZeroException divideByZeroException)
{
    Console.WriteLine("\n" + divideByZeroException.Message);
    Console.WriteLine("Zero is an invalid denominator. Please try again.\n");
}
finally
{
    Console.WriteLine( "Re-try with a different numbers. " );
}
} // end Main

```

فيما يلي عينة عن التنفيذ:

```

C:\WINDOWS\system32\cmd.exe
Please enter an integer numerator: hello
Input string was not in a correct format.
numerator must be integer. Please try again.
Please enter an integer denominator: 0
Attempted to divide by zero.
Zero is an invalid denominator. Please try again.
Re-try with a different numbers.
Press any key to continue . . .

```

6- الكلمة المفتاحية throw

رأينا سابقاً كيف يتم معالجة الاستثناءات التي يتم إلقاؤها تلقائياً من قبل البرنامج، إلا أن لغة C# تتيح لك تعريف استثناءاتك وقبورك الخاصة وإلقاء استثناءات في حال مخالفتها.

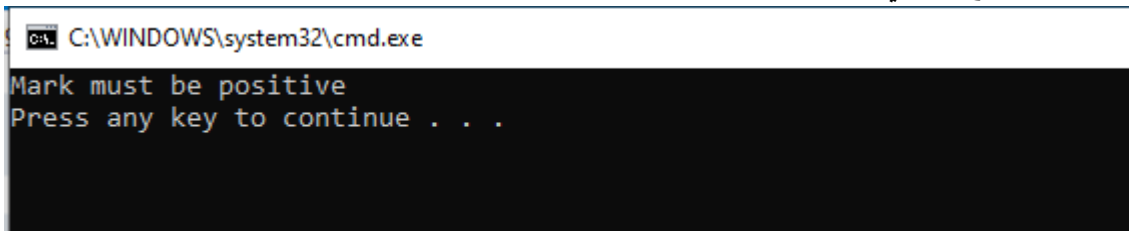
يمكن للاستثناءات أن يتم إلقاؤها بشكل يدوي باستخدام الكلمة المفتاحية throw وبالتالي يمكن إلقاء أي استثناء من الاستثناءات المستقة من الصنف Exception يدوياً باستخدام هذه الكلمة المفتاحية.

مثال:

```
static void Main(string[] args)
{
    int mark = -10;
    try
    {
        PrintMark(mark);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}

public static void PrintMark(int m)
{
    if (m < 0)
        throw new Exception("Mark must be positive");
    Console.WriteLine(m);
}
```

بتنفيذ هذا المقطع سيعطي:



```
C:\WINDOWS\system32\cmd.exe
Mark must be positive
Press any key to continue . . .
```

7- أصناف الاستثناءات مسبقة التعريف

فيما يلي قائمة بأهم أنواع الاستثناءات مسبقة التعريف في لغة C#:

Exception Class	Description
ArgumentException	Raised when a non-null argument that is passed to a method is invalid.
ArgumentNullException	Raised when null argument is passed to a method.
ArgumentOutOfRangeException	Raised when the value of an argument is outside the range of valid values.
DivideByZeroException	Raised when an integer value is divide by zero.
FileNotFoundException	Raised when a physical file does not exist at the specified location.
FormatException	Raised when a value is not in an appropriate format to be converted from a string by a conversion method such as Parse.
IndexOutOfRangeException	Raised when an array index is outside the lower or upper bounds of an array or collection.
InvalidOperationException	Raised when a method call is invalid in an object's current state.
KeyNotFoundException	Raised when the specified key for accessing a member in a collection is not exists.
NotSupportedException	Raised when a method or operation is not supported.
NullReferenceException	Raised when program access members of null object.
OverflowException	Raised when an arithmetic, casting, or conversion operation results in an overflow.
OutOfMemoryException	Raised when a program does not get enough memory to execute the code.
StackOverflowException	Raised when a stack in memory overflows.
TimeoutException	The time interval allotted to an operation has expired.

8- تمارين وأنشطة

التمرين الأول:

قم بكتابة برنامج حل معادلة من الدرجة الثانية:

$$A x^2 + B x + C = 0$$

يقوم البرنامج بسؤال المستخدم لإدخال القيم A, B, C .

بعدها يُظهر البرنامج جذور المعادلة إن وجدت أو عبارة "لا جذور حقيقية".

يجب أن يُظهر البرنامج رسائل مناسبة في حال إدخال المستخدم لقيم غير مقبولة (نصوص مثلاً عوضاً عن الأعداد).

في حال وقوع أي مشكلة في الإدخال يعاود البرنامج سؤال المستخدم عن القيم لإدخالها من جديد.

التمرين الثاني:

قم بكتابة برنامج لطباعة قواسم عدد صحيح موجب مع مراعاة احتمال حصول استثناء إدخال قيمة غير صالحة

(تصية، عشرية ... إلخ).

التمرين الثالث:

قم بكتابة برنامج لحساب معدل علامات طالب في عشر مواد بحيث لا يقبل القيم السالبة للعلامات.