الاستثناءات
Exceptions

الكلمات المفتاحية

الاستثناء، معالجة الاستثناء، الصفّ Exception، الاستثناءات مسبقة التعريف.

ملخص الفصل

نوضِّح في هذا الفصل مفهوم الاستثناءات وكيفية التقاطها ومعالجتها باستخدام try-catch-finally، ونشرح الصفّ Exception وكيفية استخدامه ووراثته لإنشاء استثناءات خاصّة بالمستخدم، ونسلّط الضوء على أشهر الاستثناءات مسبقة التعريف.

الأهداف التعليمية

يتعرّف الطالب في هذا الفصل على:

- الاستثناءات وأسباب حدوثها
- كيفية معالجة الاستثناءات
- الصفّ Exception
- كيفية تعريف استثناءات خاصّة من قِبَل المبرمج
- الاستثناءات مسبقة التعريف

مقدمة

أثناء القيام بتنفيذ برنامج ما، قد تحدث حالات غير متوقّعة ومختلفة عن السيناريو الأساسي المصمَّم من أجله البرنامج، تُسمّى هذه الحالات بالاستثناءات Exceptions. وعند حدوث إحدى هذه الحالات، يتوقّف تنفيذ البرنامج وتظهر رسالة خطأ. وتوجد عدّة أسباب لحدوث الاستثناءات، كارتكاب الأخطاء من قِبَل المبرمجين أو فشل الاتصال بقاعدة معطيات أو انقطاع في الاتصال مع أحد المخدِّمات أو عدم توافر أحد الملفّات المطلوب القراءة منها، ويجب على المبرمجين توقُّع السيناريوهات التي تؤدّي إلى حدوث الاستثناءات ووضع الحلول لمعالجتها.

تسمح العديد من لغات البرمجة بمعالجة الاستثناءات من خلال القيام بردّ الفعل المناسب وتحرير الموارد المحجوزة عند الضرورة. وفي لغة #C، الاستثناءات هي صفوف يتمّ استخدامها من أجل توصيف الأخطاء.

1. معالجة الاستثناءات

تتمّ معالجة الاستثناءات باستخدام الكلمات المفتاحية try وcatch وfinal:

- تُستخدَم try قبل كتلة التعليمات التي نتوقّع حدوث استثناء فيها.
- تُستخدَم catch لمعالجة الاستثناءات الحاصلة ضمن كتلة try. يمكن استخدام أكثر من كتلة catch كلّ منها مخصّص لمعالجة استثناء واحد.
- كتلة finally اختيارية، تحتوي على التعليمات الواجب تنفيذها سواء حدث استثناء أم لم يحدث، وعادة ما تتضمّن تعليمات تقوم بتحرير الموارد التي يتمّ حجزها في كتلة try.
- عند حصول استثناء، يتمّ الانتقال إلى أوّل معالج استثناء catch ضمن المكدّس Stack، وفي حال عدم وجوده يتوقّف البرنامج وتظهر رسالة خطأ من قِبَل نظام التشغيل. تُسمّى عملية حدوث استثناء بقذف الاستثناء Exception Throwing، ولمعالجة الاستثناء لا بُدّ من التقاطه أوّلاً.

```
try  {
            // Block of statements
     }
catch(Exception ThrownException1) {
           // Catch Block 1
    }
catch(Exception ThrownException2) {
            // Catch Block 2
    }
finally {
      // place statements here
   }
```

مثال:

لنأخذ عملية قراءة محتوى ملفّ نصّي وكتابته على الشاشة. تتطلّب هذه العملية إنشاء غرض من الصفّ FileStream للوصول إلى الملفّ، ويتمّ ذلك باستخدام بانٍ يأخذ ثلاثة وسائط دخل: المسار الكامل للملفّ، ونمط التعامل مع الملفّ، ونمط الوصول إليه. ومن الأخطاء الممكن حدوثها في هذه الحالة: عدم وجود الملفّ في المجلد المحدّد بالمسار المُعطى كوسيط دخل أو وجود خطأ في اسم الملفّ أو وجود خطأ في كتابة المسار المؤدّي إلى الملفّ. وفي حال حدوث أحد هذه الأخطاء لن يتمّ إنشاء الغرض، وهذا بدوره سيؤدّي إلى عدم إنشاء غرض من الصفّ StreamReader، وبالتالي لن تتمّ عملية القراءة، ولا حاجة لإغلاق الغرض القارئ ولا الغرض الموصل للملفّ لأنّ هذين الغرضين لم يتمّ إنشاؤهما أصلاً. ولن يتمّ اكتشاف هذه الأخطاء من قبل المترجم لأنّها ليست من النمط Syntax Error.

ونوضّح من خلال الرمّاز الآتي كيفية معالجة مثل هذه الأخطاء الاستثنائية، حيث قمنا بوضع جميع التعليمات المتعلّقة بالسيناريو السابق ضمن كتلة try، ثمّ تبعناها بتعليمة catch التي تعمل على التقاط استثناء عند حدوثه وكتابة الرسالة الموضّحة له على الشاشة.

```
using System;
using System.IO;
namespace TextFiles
{
    class FileReading
    {
        static void readObject(string path)
        {
            FileStream fs;
            StreamReader fr;
            try
            {
                //create file stream object
                fs = new FileStream(path, FileMode.Open, FileAccess.Read);
                //create reader objec
                fr = new StreamReader(fs);
                string content;
                while (!fr.EndOfStream)
                {
                    content = fr.ReadLine();
                    Console.WriteLine(content);
                }
                fr.Close();
                fs.Close();
            }
            catch (Exception e) { Console.WriteLine(e.Message); }
        }
        static void Main()
        {
            string filePath = @"C:\Users\User\Desktop\student.txt";
            readObject(filePath);
            Console.ReadKey();
        }//end Main
    }// end class FileReading
```

```
}// end namespace TextFile
/* The student.txt content is:
   C# Programming
   C Programming
   C++ Programming
*/
```

ولقد تقصّدنا حذف الملفّ من المسار المشار إليه في المتغيّر filePath، ثمّ قمنا بالتنفيذ فحصلنا على الخرج الآتي:

```
Could not find file "C:\Users\User\Desktop\student.txt"
```

وتدلّ رسالة الخطأ التي ظهرت في الخرج على أنّه لم يتمّ العثور على الملفّ في المسار المطلوب. وفي هذه الحالة، كلّ ما على المستخدم النهائي أن يقوم به هو التحقق من وجود الملفّ واسمه وصحة كتابة المسار المؤدّي إليه، ثمّ محاولة التنفيذ مرّة أخرى.

بعد التحقق من كلّ هذه الأمور وإعادة التنفيذ، تمّت طباعة محتوى الملفّ على الشاشة كما هو متوقّع، وحصلنا على الخرج الآتي:

```
C# Programming
C Programming
C++ Programming
```

2. عناصر الصفّ Exception

يحتوي الصفّ Exception المنتمي إلى فضاء الأسماء System على مجموعة من الخصائص المتاحة للاستخدام من قِبَل تعليمة catch من أجل وصف الاستثناء الذي يتمّ التقاطه، ويوضّح الجدول التالي بعض هذه الخصائص:

الخاصّية	توصيف
HelpLink	تشير إلى عنوان لملفّ يضمّ معلومات أكثر عن الاستثناء
Message	تعيد رسالة توضّح سبب حدوث الاستثناء
Source	تشير إلى اسم التطبيق أو الغرض الذي تسبّب بالاستثناء
StackTrace	إعطاء سلسلة محرفية توضّح مسار استدعاءات الطرائق التي أدّت إلى الاستثناء انطلاقاً من الطريقة Main
TargetSite	اسم الطريقة التي قذفت الاستثناء

مثال:

نوضّح في المثال التالي كيفية استخدام خصائص الصف Exception، حيث نقوم في الرمّاز الآتي بتحويل قسري بين الغرض MyObject من الصفّ MainExceptionsMembers الذي يتمّ تعريفه والغرض Formattable من الواجهة IFormattable، وهذا التحويل غير ممكن.

```
using System;
class MainExceptionsMembers
{
    public static void Main()
    {
        try
        {
            MainExceptionsMembers MyObject = new MainExceptionsMembers();
            IFormattable Formattable;// IFormattable  is a predefined
                                     // interface that prescribes an
extended ToString method
            Formattable = (IFormattable)MyObject;
            // wait for user to acknowledge the results
            Console.WriteLine("Hit Enter to terminate...");
            Console.ReadKey();
        }// end try
        catch (InvalidCastException e)
```

```
        {
            Console.WriteLine("\nThe reason for the current exception :
{0}",
                        e.Message);
            Console.WriteLine("\nThe name of the application or the object
that causes the exception : {0}", e.Source);
            Console.WriteLine("\nThe name of the method that threw the
exception : {0}", e.TargetSite);
            Console.WriteLine("\nThe stack of method calls that were
underway when the exception was thrown: {0}", e.StackTrace);
            // wait for user to acknowledge the results
            Console.ReadKey();
        }// end catch
    }// end Main
}// end class MainExceptionsMembers
```

وبعد التنفيذ، نحصل على الخرج الآتي:

```
The reason for the current exception : Unable to cast object of type
'MainExceptionsMembers' to type 'System.IFormattable'.
The name of the application or the object that causes the exception:
ExceptionsMembers
The link to the help file that provides more information about the
exception:    https://docs    Microsoft.com/en-us/dotnet/api/system.
Exception.helplink?view=netcore-3.1
The name of the method that threw the exception : Void Main()
The stack of method calls that were underway when the exception was
thrown:      at  MainExceptionsMembers.Main()  in  D:\VS\  Exceptions\
ExceptionsMembers \ ExceptionsMembers.cs: line 8
```

ويتضمّن الخرج ما يأتي:

- سبب حدوث الاستثناء المتمثّل بعدم المقدرة على إجراء التحويل بين النمط MainExceptionsMembers والنمط System.IFormattable.
- اسم التطبيق الذي أصدر الاستثناء ExceptionMembers (يمثّل في هذا المثال اسم المشروع الذي ضمّ الرمّاز المصدري).
- عنوان لصفحة ويب تتضمّن معلوماتٍ عن الاستثناء.
- اسم الطريقة التي قذفت الاستثناء (الطريقة Main).
- مسار الطريقة التي أدّت إلى الاستثناء ورقم السطر.

3. صفوف الاستثناءات المعرَّفة من قِبَل المستخدم

يمكن للمستخدم (المبرمج) تعريف استثناءات خاصّة عن طريق وراثة الصفّ ApplicationException، والموروث بدوره من الصفّ Exception. وبالتالي، يمكن تسمية الاستثناء بأي اسم يختاره المبرمج ويضمنه الرسالة التي يرغب بها، وعادة ما يتمّ اختيار اسم يدلّ على سبب حدوث الاستثناء.

ففي الرمّاز الآتي الحاوي للصفّ Point الممثّل لنقطة ثنائية الأبعاد لها الإحداثيتين xCoordinate وyCoordinate، نقوم بتعريف الاستثناء CoordinateOutOfRangeException الذي يتمّ قذفه عندما تكون قيمة xCoordinate خارج المجال [0-700] أو عندما تكون قيمة yCoordinate خارج المجال [0-500].

مثال:

```
using System;
namespace Exceptions
{
    class CoordinateOutOfRangeException : ApplicationException
    {
        public CoordinateOutOfRangeException() :
            base("The supplied coordinate is out of range.")
        { }
    }// end class CoordinateOutOfRangeException
    class Point
    {
        //  0 =< X =< 700 ,  0 =< Y =< 500
        private int xCoordinate;
        private int yCoordinate;
        public int X
        {
            get { return xCoordinate; }
            set
            {
                if ((value >= 0) && (value <= 700))
                    xCoordinate = value;
                else
                    throw new CoordinateOutOfRangeException();
            }// end set
        }// end X
```

```
        public int Y
        {
            get { return yCoordinate; }
            set
            {
                if ((value >= 0) && (value <= 500))
                    yCoordinate = value;
                else
                    throw new CoordinateOutOfRangeException();
            } // end set
        } // end Y
        public static void Main()
        {
            Point MyPoint = new Point();
            try
            {
                MyPoint.X = 2100; MyPoint.Y = 200;
                Console.WriteLine("({0}, {1})", MyPoint.X, MyPoint.Y);
            }// end try
            catch (CoordinateOutOfRangeException CaughtException)
            {
                Console.WriteLine(CaughtException.Message);

            }// end catch

            finally
            {
                Console.WriteLine("End from finally");
                Console.ReadKey();
            }// end finally
        }// end Main
    }// end class Point
}//end namespace Exceptions
```

وعند محاولة تنفيذ الرمّاز من أجل القيمتين (2100) لـ xCoordinate و(200) لـ yCoordinate على الترتيب، نحصل على الخرج الآتي:

```
The supplied coordinate is out of range.
End from finally
```

نلاحظ ظهور الرسالة التي تمّ تضمينها في الاستثناء الذي قمنا بتعريفه، والتي تدلّ على أنّ إحدى القيمتين لا تنتمي إلى المجال الموافق المطلوب، ونلاحظ أيضاً أنّه قد تمّ تنفيذ كتلة تعليمات finally والتي تضمّ طباعة رسالة نصّية فقط.

نعيد التنفيذ من دون تغيير قيمة yCoordinate، ومع تبديل قيمة xCoordinate إلى القيمة (100)، فنحصل على الخرج:

```
(100, 200)
End from finally
```

ويتضّح أنّ كتلة finally قد تمّ تنفيذها سواء أحدث الاستثناء أم لم يحدث.

4. صفوف الاستثناءات مسبقة التعريف

جميع الاستثناءات مشتقّة من الصفّ System.Exception، ويوجد عدد كبير من الاستثناءات معرّفة مسبقاً ضمن إطار عمل .NET، ونوضّح بعضاً منها في الجدول الآتي:

الاستثناء	الخطأ المسبّب
System.OutOfMemoryException	محاولة إنشاء غرض مع عدم توفّر الحجم اللازم له في الذاكرة
System.StackOverflowException	امتلاء المكدّس بسبب وجود سلسلة من استدعاءات الطرائق التي لم تنجز عملها، وعادة ما تحصل عند تنفيذ طريقة عودية عدد كبير من المرّات
System.NullReferenceException	أحد الأغراض أو المراجع لديه القيمة null
System.TypeInitializationException	عملية تهيئة فاشلة
System.InvalidCastExpression	محاولة فاشلة لتغيير نمط واجهة أو أي نمط مشتقّ أثناء التنفيذ
System.ArrayTypeMismatchException	محاولة تخزين عنصر في مصفوفة لا يتوافق نمط معطياته مع نمط معطيات عناصر المصفوفة
System.ArithmeticException	الصفّ الأساس للاستثناءات التي تحدث أثناء العمليات الحسابية مثل: System.DivideByZeroException و System.OverflowException
System.ArgumentOutOfRangeException	قيمة أحد معاملات دخل الطريقة لا تنتمي إلى المجال المعرَّفة من أجله الطريقة
System.IndexOutOfRangeException	محاولة تخزين عنصر ضمن مصفوفة باستخدام دليل لا تنتمي قيمته إلى المجال المطلوب
System.DivideByZeroException	محاولة القسمة على صفر
System.OverflowException	القيمة الناتجة من تنفيذ عملية حسابية تكون أكبر من القيمة العظمى الممكن تخزينها في نمط بيانات هذه القيمة، مثلاً إذا كانت القيمة الواجب إسنادها لمتحوّل من النمط int هي 7845100000 سنحصل على استثناء من هذا النمط لأنّ أكبر قيمة يمكن تخزينها في متحوّل من النمط int هي: 2147483647

أمثلة

مثال1: محاولة القسمة على صفر

في الرمّاز الآتي، نوضّح كيفية معالجة من النمط DivideByZeroException:

```
using System;
class ExceptionTest
{
    static double SafeDivision(double x, double y)
    {
        if (y == 0)
            throw new DivideByZeroException();
        return x / y;
    }// end SafeDivision
    static void Main()
    {
        // Change the values
        // to see exception handling behavior.
        double a = 98, b = 0;
        double result = 0;

        try
        {
            result = SafeDivision(a, b);
            Console.WriteLine("{0} divided by {1} = {2}", a, b, result);
        }// end try

        catch (DivideByZeroException e)
        {
            Console.WriteLine("{0}", e.Message);
        }//end catch
        Console.ReadKey();
    } // end Main
      }// end ExceptionTest
```

تقوم الطريقة SafeDivision باستقبال عددين من النمط double، وتعيد ناتج قسمة العدد الأوّل على العدد الثاني بشرط ألّا تكون قيمة العدد الثاني مساوية لـ (0). وفي حال عدم تحقّق الشرط، يتمّ قذف استثناء من النمط DivideByZeroException وهو صفّ مسبق التعريف في منصّة .NET..

وفي الطريقة Main، تمّ تعريف المتحوّلين a وb وإسناد القيمة (98) لـ a والقيمة (0) لـ b، وتمّ تعريف المتحوّل result الذي سيضمّ ناتج قسمة a على b. ثمّ تمّ استدعاء الطريقة SafeDivision ضمن جسم التعليمة try لأنّه من الممكن حدوث استثناء (قسمة على صفر)، وتمّ استدعاء أمر طباعة لإظهار رسالة توضّح ناتج عملية القسمة. وفي حال حدوث الاستثناء المتوقّع، يتمّ التقاطه ومعالجته ضمن جسم التعليمة catch التي تأخذ كوسيط دخل الغرض e المنتسخ من الصفّ DivideByZeroException، وتقوم بإظهار رسالة تشرح فيها الخطأ الذي أدّى إلى حصول الاستثناء. وبعد تنفيذ البرنامج وحدوث الاستثناء، ظهرت الرسالة الآتية:

```
Attempted to divide by zero.
```

وهذا يدلّ على أنّه تمّ التقاط الاستثناء ومعالجته ضمن جسم التعليمة catch.

مثال 2:

دليل المصفوفة خارج المجال المطلوب

في الرمّاز الآتي، تضم المصفوفة IntegerArray خمسة عناصر، ولذلك، يجب أن تكون قيم دليلها ضمن المجال [0-4]، وعند محاولة إسناد قيمة في للمصفوفة للعنصر الموافق للدليل (10)، سيتمّ قذف استثناء من نمط IndexOutOfRangeException، والتقاطه من قبل التعليمة catch التي ستطبع رسالة توضّح سبب حدوث الاستثناء.

```
using System;
class ExceptionTest
{
    public static void Main()
    {
        try
        {
            int[] IntegerArray = new int[5];
            IntegerArray[10] = 123;
            // wait for user to acknowledge the results
            Console.WriteLine("Hit Enter to terminate...");
            Console.ReadKey();
        }
        catch (IndexOutOfRangeException)
```

```
        {
            Console.WriteLine("An invalid element index access was
attempted.");
            // wait for user to acknowledge the results
            Console.ReadKey();
        }
        Console.ReadKey();
    } // end Main
}// end ExceptionTest
```

وبعد التنفيذ، نحصل على الخرج الآتي:

```
An invalid element index access was attempted.
```

نلاحظ أنّنا لم نقم بإنشاء غرض من الصفّ IndexOutOfRangeException لأنّه لا حاجة للاستعانة بأي من أعضائه، وقمنا بطباعة رسالة خاصّة تشرح الاستثناء.

مثال3: احتمال حدوث أكثر من استثناء

في بعض الحالات، قد تتضمّن تعليمات كتلة try أكثر من تعليمة قابلة للتسبّب بحدوث استثناء. ولمعالجة هذه الحالة، نكتب لكلّ استثناء تعليمة catch خاصّة به لمعالجته. فعند حدوث أحد الاستثناءات المتوقّعة ستلتقطه تعليمة catch الخاصّة به وتعالجه ولا تتمّ متابعة باقي التعليمات الموجودة ضمن كتلة try. ففي الرمّاز الآتي، تتضمّن كتلة تعليمة try تعليمتين قابلتين لقذف استثناء:

```
using System;
namespace Exceptions
{
    class ExcepTester
    {
        public static void Main()
        {
            int[] Array;
            try
            {
                Array = new int[500000000000];
                Array = new int[5];
                Array[10] = 123;
            }
```

```
            catch (OverflowException e)
            {
                Console.WriteLine("Overflow: " + e.Message + e.StackTrace);
            }
            catch (IndexOutOfRangeException e)
            {
                Console.WriteLine("IndexOutOfRangeException: " + e.Message);
            }
            finally
            {
                Console.WriteLine("End Finally");
                Console.ReadKey();
            }
        } // end Main
    }// end ExcepTester
}//end namespace Exceptions
```

وبعد تنفيذ الرمّاز، نحصل على الخرج الآتي الذي يشير إلى استثناء حصل في السطر السابع، وهو من النمط Overflow، ولم يحدث استثناء في السطر التاسع على الرّغم من استخدام دليل للمصفوفة من خارج المجال المطلوب.

```
Overflow: Arithmetic operation resulted in an overflow.    at
Exceptions.ExcepTester.Main()                              in
D:\VS\Exceptions\Test\Program.cs:line7
End Finally
```

خاتمة

لقد تمّ في هذا الفصل التعرّف على الصفّ Exception وأعضائه وكيفية استخدامه والأنماط المشتقّة منه. وفي الفصل القادم، سيتمّ التعرّف على صفوف أخرى تسهل عمل المبرمجين. ونظراً لأهمية بنى المعطيات عموماً والمصفوفات بشكل خاصّ، سيتمّ تخصيص الفصل القادم للتعرّف على بنى معطيات كثيرة الاستخدام من قِبَل المبرمجين.

الأنشطة المرافقة

1. قم بإنشاء تطبيق برمجي بلغة C# لمؤسّسة يوجد فيها موظّفون، ويتمّ فيها توصيف كلّ موظّف من خلال رقمه الذاتي واسمه وكنيته ودخله الشهري ورقم هاتفه. وينقسم الموظّفون وفقاً لطريقة حساب مداخيلهم الشهرية إلى ثلاثة أقسام:
   - موظّف يتقاضى راتباً شهرياً
   - موظّف يتقاضى أجوراً ساعية مقدراها 5$ لكلّ ساعة عمل
   - موظّف يتقاضى راتباً شهرياً وأجوراً ساعية عن كلّ ساعة إضافية يعملها خلال الشهر

2. يتمّ تخزين الموظّفين ضمن مصفوفة، ويجب أن يقوم التطبيق بالمهام الآتية:
   - إضافة شخص إلى المصفوفة
   - طباعة معلومات موظّف بعد تمرير رقمه الذاتي
   - طباعة معلومات كافّة الموظّفين الذين يزيد دخلهم الشهري عن قيمة معينة
   - طباعة عدد الموظّفين
   - طباعة المبلغ الممثّل لمجموع رواتب الموظّفين

   ثمّ قم بإنشاء الصفّ `Tester` الحاوي على الطريقة `Main` من أجل اختبار التطبيق

3. يجب معالجة جميع القيم المدخلة من قِبَل المستخدم وغير المقبولة: قيم غير رقمية بدلاً من أرقام، قيم سالبة عوضاً عن أعداد موجبة، أرقام بدلاً من سلاسل نصّية. ويجب التحقّق من أنّ لقيمة رقم الهاتف الصيغة `###-###-####` حيث يعبّر # عن رقم.

المراجع

1. https://docs.microsoft.com/en-us/dotnet/csharp/