الباب التاسع

الاســـتثناءات

(Exceptions)

محتويات الباب:

- الأخطاء الاستثنائية
- إلقاء الاستثناء (throw)
- إمساك الاستثناء (try - catch)
- معالجة الملفات
- استخدام المقبض النهائى (finally)
- الاستثناءات المبتكرة
- إعادة إلقاء الاستثناء (Rethrowing)

الأخطاء الاستثنائية

هـــناك نوعـــية من الأخطاء تحدث وقت تنفيذ البرنامج وتتسبب فى توقف البرنامج تماماً مع إرسال رسالة فى نافذة خاصة عن نوع الخطأ الـــذى تســـبب فى المشكلة. وتسمى هذه النوعية من الأخطاء بالأخطاء الاستثنائية أو الاستثناءات. وعندما يحدث هذا الخطأ فإننا نصطلح عليه بأن البرنامج قد **ألقـــى استثناءً**. والاستثناء عبارة عن هدف من الفصيلة **System.Exception** ، ويحـــتوى الهـــدف علـــى المعلومات الخاصة بأســـباب حـــدوث هـــذا الاستثناء. والفارق بين الأخطاء (errors) وبين الاســـتثناءات ، أن الخطأ يمكن التنبؤ به وتجنبه ، بينما الاستثناء قد لا يكـــون مـــتوقعاً مـــثل حالة نفاذ الذاكرة أثناء تشغيل البرنامج ، أو عند محاولـــة فـــتح ملف سبق مسحه. فى هذه الحالة عليك بالتعامل مع هذا الاستثناء باستخدام **المقبض** المناسب حتى لا يتوقف البرنامج. والمقبض عـــبارة عـــن بلوك من الكود **يمسك** بالاستثناء ويعالج الموقف ويحاول استمرار البرنامج.

أما إذا لم توفر للاستثناء المقبض المناسب ، فإن المقبض سابق التعريف يتولى الزمام وينهى عمل البرنامج.

وهناك عبارات خاصة للتعامل مع الاستثناءات من خلال برنامج سى# ، نلخصها فيما يلى:

❖ **throw**: إلقاء الاستثناءات

- ❖ **try - catch**: إمساك الاستثناءات
- ❖ **finally** - try: التنظيف بعد إلقاء الاستثناء. والتنظيف يعنى فى أغلـــب الأحوال تنظيف الموارد (مثل إغلاق الملفات المفتوحة) بصرف النظر عن الاستثناء.
- ❖ try - catch - finally: إمساك الاستثناء والتنظيف

إلقاء الاستثناء (throw)

تستخدم العبارة throw لإلقاء هدف استثناء ما بالصورة الآتية:

```
throw [expression];
```

حيث:

- ❖ Expression: هدف الاستثناء ـــ وهو اختيارى.

والاســـتثناءات الـــتى يلقـــيها الـــبرنامج عبارة عن هدف من الفصيلة System.Exception أو أحد الأهداف المشتقة منه التى تتضمن:

InvalidCastException
OverFlowException
ArgumentNullException
ArithmeticException
DivideByZeroException

واستثناءات أخرى كثيرة.

وعـــندما يلقى الاستثناء فإن البرنامج يتعطل لحظيا حيث يجرى البحث عن مقبض لمعالجة هذا الاستثناء. فإن وجد المقبض المناسب فإنه يقوم

بمعالجـــة الموقف بحيث يستمر تنفيذ البرنامج. وقد لاتنجح بيئة التشغيل فى العثور على المقبض وفى هذه الحالة ينتهى التنفيذ عند هذا الحد.

مثال(1-9) الإلقاء

يحتوى البرنامج التالى على العبارة throw التى تلقى استثناء كهدف من الفصيلة ArgumentNullException.

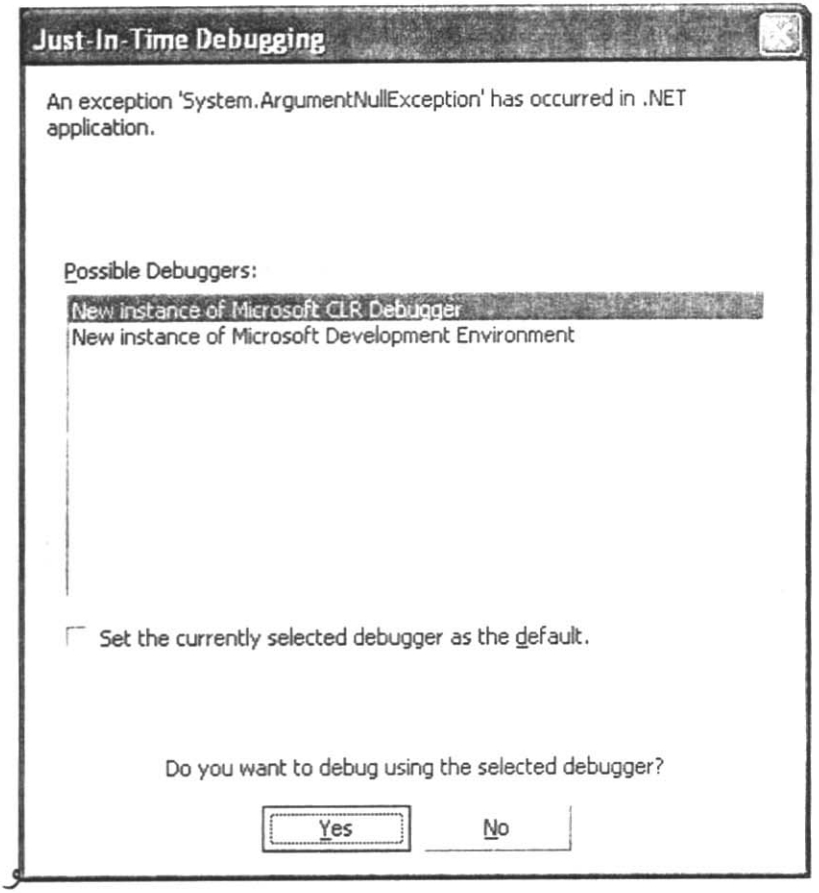
```
// Example 9-1.cs
// throw example

using System;

public class MyClass
{
  s tatic void Main()
  {
    string myString = "Hello.";
    // Print the first statement:
    Console.Write("The string is {0}", myString);

    myString = null;

    if (myString == null)
    {
      throw new ArgumentNullException  (); //  إلقاء الاستثناء
    }

    // The following line will not be executed:
    Console.Write("myString is null.");
  }
}
```

تنفيذ البرنامج:

عند تنفيذ هذا البرنامج يطبع العبارة:

```
The string is Hello.
```

ثم يتوقف قليلاً ويلقى الاستثناء بالصورة الموضحة فى الشكل التالى:



ونرى فى أعلى الشكل اسم الاستثناء الذى ألقاه البرنامج وهو:

System.ArgumentNullException

والنتـــيجة المتوقعة هى أحد اختيارين : إما استخدام أداة بيئة الأستوديو لمعالجة المشكلة (Debugger) أو اختيار أداة أخرى. اختر الزر No لكى تستمر فى تنفيذ برنامجك. وعندئذ ترى رسالة مماثلة للرسالة الآتية:

```
The string is Hello.
Unhandled Exception: System.ArgumentNullException: Value
cannot be null.
   at MyClass.Main()
```

أما العبارة الأخيرة فى البرنامج:

```
Console.Write("myString is null.");
```

فإنها لن تنفذ لأنها جاءت تالية للاستثناء.

ملاحظات على البرنامج السابق:

لـــو أنـــك استبدلت اسم فصيلة الاستثناء ArgumentNullException باسم الفصيلة الأساسية Exception ، أى:

```
throw new Exception();
```

فإنك تحصل على رسالة مختلفة هى:

```
The string is Hello.
Unhandled Exception: System.Exception: Exception of type
System.Exception was thrown.
  at MyClass.Main()
```

أى أن اسم هدف الاستثناء يظهر دائماً فى النتيجة.

وفـــى الفقرات القادمة سوف نتعرف بطريقة إمساك ومعالجة الاستثناء باستخدام مقبض للمعالجة بحيث يستمر تنفيذ البرنامج. أما ما حدث هنا فهو نتيجة استخدام المقبض سابق التعريف فى بيئة التشغيل.

إمساك الاستثناء (try - catch)

تتكون الـــعبارة try-catch من بلوكين: الأول يسمى بلوك المحـــاولة (try). ويوضع الكود الذى نتوقع منه حدوث الخطأ (مثل انتهاء الذاكرة المتاحة أو عدم وجود الملف) بداخل هذا البلوك. وهو يأخذ الصورة:

```
try
{
  // الكود المطلوب تجربته
}
```

أما الجـــزء الثانى من العبارة try-catch فهو بلوك إمـــساك الاستثناء (catch) الـــذى يســـتخدم فى إنشاء مقبض معالجة الاستثناء. وهو يأخذ الصورة:

```
catch [(declaration)]
{
  // كود مقبض المعالجة
}
```

حيث:

❖ declaration: إعلان عن هدف الاستثناء ــ وهو اختيارى.

ولا يجوز استخدام أحد البلوكات بدون الآخر. ولو أنك استخدمت بلوك المحاولة بمفرده فإن المترجم يعترض مرسلاً الرسالة الآتية:

```
error CS1524: Expected catch or finally
```

وقـــد يستخدم البلوك catch بدون بارامترات على الإطلاق وهو يصلح فى هذه الحالة للتعامل مع أى استثناء.

ومن الجائز أن تستخدم عدة بلوكات من بلوكات الإمساك فتأخذ العبارة الكاملة الصورة:

```
try
{
  // الكود المطلوب تجربته
}
catch (declaration-1)
{
  // كود مقبض المعالجة الأول
}
catch (declaration-2)
{
```

```
  كود مقبض المعالجة الثانى //
}
catch (declaration-3)
...
```

وهكذا

وعـــندما يلقى البرنامج استثناءً ما ، تبدأ بيئة التشغيل فى البحث عن المقبض المناسب فى بلوكات الإمساك. فإن تم العثور عليه فإن البرنامج يستمر بعد معالجة المشكلة أو طباعة الرسالة المناسبة التى تشرح الخطأ الذى حدث أثناء تشغيل البرنامج.

مثال (2-9) مقبض المعالجة

فى المثال التالى نستخدم ثلاثة أجزاء تؤدى الوظائف التالية:

- ❖ إلقاء الاستثناء (throw)
- ❖ تجربة الكود باستخدام بلوك المحاولة (try)
- ❖ إمساك الاستثناء باستخدام بلوك الإمساك (catch).

```
// Example 9-2.cs
// Handling an exception

using System;

class MyClass
{
  public void MyMethod(string myString)
  {
    if (myString == null)
     throw(new ArgumentNullException()); //  إلقاء الاستثناء
  }
```

```
  public static void Main()
  {
    MyClass myClass = new MyClass();
    try                                    //        بلوك المحاولة
    {
      string myString = null;
      myClass.MyMethod(myString);
    }

    catch (Exception e)                    //        بلوك الإمساك
    {
      Console.WriteLine("The fo  llowing exception is caught:
\n{0}", e);
    }
    // Continue after handling the exception:  //    استمرار البرنامج
    Console.WriteLine("Now the program continues...");
  }
}
```

تنفيذ البرنامج:

```
The following exception is caught:
System.ArgumentNullException: Value cannot be null.
   at MyClass.Main()
Now the program continues...
```

ملاحظات على البرنامج السابق:

أول مـــا نلاحظـــه علـــى هذا البرنامج أن وجود المقبض يؤدى إلى اســـتمرار البرنامج. وذلك بصرف النظر عن محتويات المقبض. وكما نـــرى فى البرنامج السابق أن المقبض يحتوى على عبارة لطبع بعض المعلومـــات عن الاستثناء ولكن ذلك كان كافياً للتعامل معه ، واستمرار الـــبرنامج إلى الخطوة التالية. ومع ذلك ففى البرامج التطبيقية فإنه من الضـــرورى تصحيح الاستثناء الذى وقع بطريقة ما. فإذا كانت المشكلة مـــثلاً هى نفاذ الذاكرة المخصصة للبرنامج فقد يكون من الأفضل تنبيه

المستخدم وإعطائه الفرصة لإغلاق بعض التطبيقات المفتوحة وإعادة المحاولة.

ترتيب مقابض المعالجة

من المهم أن ترتب المقابض بحسب درجة التوقع ، من الأكثر توقعاً إلى الأقل توقعاً. فلو كان أكثر الاستثناءات توقعاً هو الاستثناء الحسابى ArithmeticException ، فإن هذا المقبض يجب أن يكون فى المقدمة ، أى تالياً لبلوك المحاولة مباشرة. ثم تأتى المقابض الأخرى مرتبة بحسب درجة توقعها ، وتنتهى بالمقبض العام مثل:

```
catch (ArithmeticException e) { ... } // أول مقبض
...
catch (Exception e) { ... }    // آخر مقبض
```

وبالرغم من أن ترتيب المقابض مسألة تقديرية ، ولكنك لو بدأت البلوكات بالمقبض العام فإنك تتلقى رسالة خطأ مثل:

```
error CS0160: A previous catch clause already catches all
exceptions of this or a super type ('System.Exception')
```

مثال (9-3) المقابض الخاصة

فى هذا المثال نجرب العبارة x/y التى تؤدى إلى إلقاء استثناء القسمة على صفر. ونستخدم مقبضين بالبرنامج ، الأول هو مقبض الاستثناء المتوقع (ArithmeticException) والثانى هو المقبض العام (Exception).

```
// Example 9-3.cs
// Ordering exceptions
```

```
using System;

class MyClass
{
  static void Main()
  {
    int x = 0;
    int y = 10;
    try  //     بلوك المحاولة
    {
     int z = y/x;
    }
    // The most expected exception:   مقبض إمساك الاستثناء الأكثر احتمالاً
    catch (ArithmeticExcepti  on e)
    {
      Console.WriteLine("Arithmetic Exception Handler: {0}", e);
    }
    // Catch the general exception:   مقبض إمساك الاستثناءات العام
    catch (Exception e)
    {
      Console.WriteLine("General Exception Handler: {0}", e   );
    }
    // Continue the program:   استمرار البرنامج بعد معالجة الاستثناء
    Console.WriteLine("Program Continues...");
  }
}
```

تنفيذ البرنامج:

```
Arithmetic Exception Handler:
System.DivideByZeroException: Attempted to divide
by zero.
   at MyClass.Main()
Program Continues...
```

ملاحظات على البرنامج السابق:

جـــرب حذف المقبض الخاص بالاستثناء ArithmeticException وشاهد النتيجة عندما يقوم المقبض العام بمعالجة الاستثناء.

تدريب (9-1)

بالرغم من أنه يمكنك التعامل مع "القسمة على صفر" باستخدام الاستثناء ArithmeticException ولكن هناك فصيلة أكثر تخصصاً فى التعامل مع القســـمة علـــى صفر وهى DivideByZeroException. جرب أن تستخدم هذه الفصيلة جنبا إلى جنب مع الفصائل الأخرى فى المثال السابق ، ثم غـــيّر ترتيب المقابض حتى تحصل على الترتيب الذى لا يعترض عليه المترجم.

تتابع الأحداث فى التعامل مع الاستثناءات

هـــناك تتابع معين للأحداث عند إلقاء استثناء ، حيث تبدأ عملية البحث باستعادة سيناريو الدوال الموجودة فى الخزنة. وهى عملية تشبه تشغيل شـــريط الفيديو فى الاتجاه العكسى. فإذا لم يعثر البرنامج على المقبض المناســـب فـــى الأســـلوب الحالى ، فإنه يبدأ فى إعادة شريط الأحداث وينظر بداخل الأساليب الأخرى التى قد تحتوى على المقبض.

مثال (9-4) أحداث معالجة الاستثناء

نقدم فى هذا المثال برنامجين متشابهين إلى حد كبير للمقارنة بينهما. الأول تتسلســـل فـــيه الأحـــداث بصورة عادية حيث يتم إلقاء الاستثناء ومعالجـــته فـــى نفـــس الأســـلوب MyMthod2. أما البرنامج الثانى فإن الاســـتثناء يلقـــى فـــى الأســـلوب MyMthod1 ثـــم يعالج فى الأسلوب

MyMethod2. ولذلـــك ففى البرنامج الثانى تحدث عملية ترجيع الشريط للبحث عن المقابض.

ونـــرى فـــى نتيجة تنفيذ البرامج سجلاً كاملاً لتسلسل الأحداث فى كل برنامج ، حيث نرى متى يبدأ عمل كل أسلوب ومتى ينتهى.

```
// Example 9-4A.cs
// Events sequence

using System;

class MyClass
{
  static void Main()
  {
    Console.WriteLine("Main starts...");
    MyClass mc = new MyClass();
    mc.MyMethod1();
    // Continue the p  rogram:
    Console.WriteLine("Program ends.");
  }
  void MyMethod1()
  {
    Console.WriteLine("Starting Method1.");
    MyMethod2();
    Console.WriteLine("Exiting Method1.");
  }
  void MyMethod2()
  {
    Console.WriteLine("Starting Me  thod2.");
   try
    {
      Console.WriteLine("Starting the try block.");
      throw new Exception();
      Console.WriteLine("Exiting the try block.");
    }
    catch
    {
      Console.WriteLine("Handling the exception.");
```

```
    }
    Console.WriteLine("Exiting Method2.");
  }
}
```

تنفيذ البرنامج:

فـــيما يلـــى نرى نتيجة العبارات التى وضعناها فى مدخل ومخرج كل أسلوب من أساليب البرنامج. وهذا يساعدك على تتبع الأحداث الجارية. وكمـــا نرى فى هذا المثال أن دخول الأسلوب MyMrthod2 أعقبه دخول بلوك المحاولة ومعالجة الاستثناء ، ثم الخروج من الأسلوب.

```
Main starts...
Starting Method1.
Starting Method2.    // الدخول
Starting the try block.
Handling the exception.
Exiting Method2.    // الخروج
Exiting Method1.
Program ends.
```

وهـــذا هو البرنامج الثانى الذى يحدث فيه استعادة الأحداث بالخزنة (Unwinding stack) بهدف البحث عن المقبض.

```
// Example 9-4Bcs
// Ordering exceptions

using System;
class MyClass
{
  static void Main()
  {
  Console.WriteLine("Main starts...");
  MyClass mc = new MyClass();
  mc.My Method1();
  Console.WriteLine("Program ends.");
  }
  void MyMethod1()
```

```
  {
    Console.WriteLine("Starting Method1.");
    try
    {
      Console.WriteLine("Starting the try block.");
      MyMethod2();
      Console.WriteLine("Exiting   the try block.");
    }
    catch
    {
      Console.WriteLine("Handling the exception.");
    }
    Console.WriteLine("Exiting Method1.");
  }
  void MyMethod2()
  {
    Console.WriteLine("Starting Method2.");
    throw new Exception  ();
    Console.WriteLine("Exiting Method2.");
  }
}
```

تنفيذ البرنامج:

نلاحظ في نتيجة التنفيذ هنا أنه بعد دخول بلوك المحاولة بالأسلوب MyMethod1 قد اتجه التنفيذ إلى الأسلوب MyMethod2 للبحث عن مقبض.

```
Main starts...
Starting Method1.
Starting the try block.
Starting Method2.
Handling the exception.
Exiting Method1.
Program ends.
```

تعليق على البرنامجين السابقين:

نلاحـــظ فى كل من البرنامجين 9-4A.cs و 9-4B.cs أننا نرى فى التنفيذ عبارة دخول بلوك المحاولة:

```
Starting the try block.
```

ولكننا لا نرى عبارة الخروج منه. وهذا يعنى أن البرنامج لا يعود إلى النقطة التى حدث عندها إلقاء الاستثناء.

الاستثناءات المتوقعة أثناء معالجة الملفات

هـــناك بـــروتوكولات معيـــنة لفـــتح وإغلاق الملفات باستخدام فصائل دوت.نـــت. فعلى سبيل المثال لكى تفتح ملفاً ما للقراءة بالاسم test.txt فإنك تتخذ الخطوات التالية:

- استخدم حيز الاسم **System.IO**
- أعلن عن متغير ملف باستخدام النمط **StreamReader** كالآتى:

```
StreamReader myFile;
```

- افتح الملف بإحدى الطرق الآتية:

```
myFile = File.OpenText("test.txt");
```

أو

```
myFile = new StreamReader("test.txt");
```

- اقرأ محتويات الملف إما سطراً بسطر بالأسلوب ReadLine() مع تخزين كل سطر فى حرفى:

```
string line = myFile.ReadLine();
```

أو اقرأ الملف دفعة واحدة حتى نهايته بالأسلوب **ReadToEnd()**، ثم خزن محتوياته فى حرفى:

```
string file = myFile.ReadToEnd();
```

❖ اطبع الملف سطراً سطراً أو دفعة واحدة باستخدام العبارة:

```
Console.WriteLine(line);
```

أو

```
Console.WriteLine(file);
```

❖ أغلق الملف باستخدام الأسلوب **Close():**

```
myFile.Close();
```

ملاحظة:

يمكنك فتح الملف للكتابة باستخدام الإعلان التالى:

```
StreamWriter myFile = new StreamWriter("test.txt");
```

كما يمكنك فتح الملف للتذييل (للكتابة فى المؤخرة) بالإعلان:

```
StreamWriter myFile = new StreamWriter("test.txt",true);
```

والاستثناء المتوقع حدوثه هو عدم وجود الملف. وسوف نترك لك برمجة هذا التطبيق كتدريب.

وعند التعامل مع الملفات فإن البرنامج قد يلقى استثناءً فى حالة إذا كان الملف غير موجود على القرص. ولذلك فإنه من المفضل أن تضع عبارات فتح الملف بداخل بلوك محاولة كما فى المثال التالى.

مثال (5-9) استثناء: الملف غير موجود

فى هذا المثال نقرأ ملفاً للنصوص بالاسم "test.txt" بالفهرست الحالى ، ونخزن كل سطر فى الحرفى line ، ثم نطبع الملف سطراً بسطر من خلال حلقة تكرارية. وقد استخدمنا هنا مقبضاً لمعالجة الاستثناء FileNotFoundException على اعتبار أنه الاستثناء الوحيد المتوقع. كما أضفنا الاستثناء العام من باب الاحتياط.

```
// Example 9-5.cs
// Processing files

using System;
using System.IO;

class MyClass
{
  static void Main( )
  {
    int counter = 0;
    string line;
    try
    {
      StreamReader file = File.OpenText("test.txt");
      while((line = file.ReadLine()) != null)
      {
        Console.WriteLine (line);
        counter++;
       }
      file.Close();
    }
    catch (FileNotFoundException)
    {
      Console.WriteLine("The file you are trying to open is not
found.");
    }
    catch
    {
      Console.WriteLine("General catch statement.");
    }
  }
}
```

تنفيذ البرنامج:

بفـــرض أن الملـــف test.txt لا يوجد على القرص فإن البرنامج يطبع الرسالة الآتية:

```
The file you are trying to open is not found.
```

أنشىء الملف test.txt أو انسخ أحد الملفات بأمر مثل:

```
>COPY 9-5.cs test.txt
```

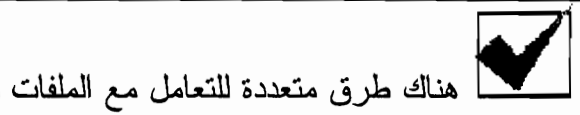
ثم جرب تشغيل البرنامج. وشاهد النتيجة.

ملاحظة:

لاحظ أن المتغيرات المعلن عنها بداخل بلوك المحاولة لا يمكن التوصل إليها من خارج البلوك. ولذلك فإنه من المفضل دائماً أن تعلن عن المتغيرات فى الأسلوب الرئيسى. ويمكنك دائماً شحن المتغيرات بداخل بلوك المحاولة.

تدريب (9-2)

اكتب برنامجاً يؤدى نفس العمليات الواردة فى المثال السابق مستخدما طريقة قراءة الملف دفعة واحدة بالأسلوب ReadToEnd().



هناك طرق متعددة للتعامل مع الملفات قراءة وكتابة ، ولكننا ذكرنا هنا أكثر الطرق شيوعاً. ويمكنك الاطلاع على المزيد عن القنوات StreamReader و StreamWriter على موقع الوب MSDN:

http://msdn.microsoft.com

استخدام المقبض النهائى (finally)

يستخدم بلوك المقبض النهائى (finally) إما مع بلوك المحاولة ، وفى هذه الحالة تسمى العبارة try-finally.

كما يمكن أن يستخدم مع كل من بلوك المحاولة وبلوك الإمساك ، وفى هذه الحالة تسمى العبارة try-catch-finally.

وعند تنفيذ أى برنامج محتوٍ على البلوك النهائى فإن دفة التحكم تنتقل دائماً إلى هذا البلوك بصرف النظر عن وجود استثناء من عدمه. ويستخدم البلوك عادة لتنظيف الموراد أى لإتاحة الموارد التى سبق حجزها.

العبارة try- finally

تأخذ هذه العبارة الصورة الآتية:

```
try
{
  try -block
}
finally
{
  finally -block
}
```

حيث:

- try-block: بلوك المحاولة ـ يحتوى على الكود الطلوب تجربته.
- finally-block: بلوك المقبض النهائى ـ يحتوى على الكود اللازم لتنظيف الموارد وهو ينفذ بصرف النظر عن وقوع الاستثناء من عدمه.

مثال (6-9) تنفيذ عبارات المقبض النهائى

فى هذا المثال نعرض كيف تنفذ العبارات الواردة فى المقبض النهائى. والبرنامج يلقى الاستثناء InvalidCastException نتيجة استخدام الاسقاط

int مع هدف من النمط string (وهو تحويل غير جائز). ويحتوى البلوك finally على بعض العبارات التى تنفذ بالرغم من إلقاء الاستثناء.

```
// Example 9-6.cs
// try-finally example

using System;

public class MyClass
{
  static void Main()
  {
    string myString = "Finally";
    object myObject = myString;

    try
    {
      // The following conversion is inv   alid.
      // It throws an exception.
      int myInt = (int)myObject;
    }

    finally
    {
     // The code in this block is always executed:
     Console.WriteLine("The program continues and ends here:");
  Console.WriteLine("My  String is: {0}", myString);
    }
  // The following code will not be executed:
  Console.WriteLine("Hello again!");  // عبارة لن تنفذ.
  }
}
```

تنفيذ البرنامج:

نلاحظ أن البرنامج سوف يتوقف وتظهر نافذة الاستثناء التقليدية. وبالضغط على الزر No يظهر اسم الاستثناء الذى حدث ثم يستكمل تنفيذ الجزء من البرنامج الموجود بداخل البلوك النهائى.

```
Unhandled Exception: System.InvalidCastException:
Specified cast is not valid.
   at MyClass.Main()
The program continues and ends here: استكمال تنفيذ البلوك النهائى
My String is: Finally
```

ملاحظات على البرنامج السابق:

نلاحظ أن العبارة الأخيرة فى البرنامج ، وهى تقع خارج البلوك النهائى ، لن تنفذ. معنى ذلك أن الاستثناء يمنع وصول دفة التحكم إلى أى جزء فى البرنامج ماعدا البلوك النهائى.

العبارة try-catch- finally

تأخذ هذه العبارة الصورة الآتية:

```
try
{
  try -block
}
catch
{
  catch -block
}
finally
{
  finally -block
}
```

حيث:

- try-block: بلوك المحاولة ــ يحتوى على الكود الطلوب تجربته.
- catch-block: بلوك الإمساك ــ يحتوى على مقبض المعالجة.

❖ finally-block: البلوك النهائى ـ يحتوى على الكود اللازم لتنظيف الموارد وهو ينفذ بصرف النظر عن وقوع الاستثناء من عدمه.

من المؤكد بطبيعة الحال أنه طالما أُلقى استثناء ما فإنه من الأفضل إمساكه ومعالجته بالبلوك catch حتى لا يتوقف البرنامج. لذلك فإن البلوك النهائى يستخدم بهذه الصورة طالما أن إلقاء الاستثناء متوقع.

مثال (9-7) تنفيذ مقبض المعالجة ومقبض البلوك النهائى

فى هذا المثال قد أضفنا إلى المثال السابق مقبض المعالجة وبهذا تكتمل الصورة بإمساك الاستثناء الذى ألقاه بلوك المحاولة ، ويستمر تنفيذ البرنامج بعد تنفيذ البلوك النهائى.

```
// Example 9-7.cs
// try-catch-finally example

using System;

public class MyClass
{
  static void Main()
  {
    string myString = "Try  -Catch-Finally.";
    object myObject = myString;

    try
    {
      // The following conversion is invalid.
      // It throws an exception.
      int myInt = (int)myObject;
    }
    catch(InvalidCastException ex)
    {
```

```
      Console.WriteLi  ne("The exception \"{0}\" was handled.",
ex);
    }

    catch
    {
      Console.WriteLine("Unknown exception handled.");
    }

    finally
    {
     // The code in this block is always executed:
     Console.Write("The program   continues here: ");
     Console.WriteLine("My String is: {0}", myString);
    }
  // The following code will always be executed:
  Console.WriteLine("Goodbye!");
  }
}
```

تنفيذ البرنامج:

```
The exception "System.InvalidCastException: Specified
cast is not valid.
   at MyClass.Main()" was handled.
The program continues here: My String is: Try-Catch-
Finally.
Goodbye! العبارة الأخيرة فى البرنامج
```

ملاحظات على البرنامج السابق:

فـــى هـــذا الـــبرنامج نلاحـــظ أن المقـــبض الأول قد أمسك بالاستثناء InvalidCastException وطبع على الشاشة نتيجة المعالجة. أما المقبض النهائى فقد طبع محتويات الحرفى myString. كما نلاحظ فى هذا المثال أن العبارة الأخيرة قد تم تنفيذها بعد معالجة الاستثناء الذى وقع.

تدريب (3-9)

باعتبار أن الغرض الأساسى من بلوك المقبض النهائى هو تنظيف الموارد ، اكتب برنامجاً لقراءة ملف نصوص مع استخدام بلوك المقبض النهائى فى إغلاق الملف الذى سبق فتحه.

الاستثناءات المبتكرة (User-defined Exceptions)

تستطيع دائما أن تبتكر استثناءً ما بإعلانه كفصيلة تنحدر من الفصيلة ApplicationException. والفائدة التى تعود عليك كمبرمج من استخدام الاستثناءات المبتكرة هى منحك الفرصة لكى تضيف إلى المقبض ما تشاء من النصوص التى تشرح الخطأ الذى حدث بأى درجة من التفصيل.

ومن البديهى أن برنامجك هو الذى يلقى هذا الاستثناء ، أما بيئة التشغيل فلا تعرف شيئاً عن الاستثناء المبتكر.

ويتم وراثة الفصيلة كالمثال الآتى:

```
class MyCustomException: ApplicationException
```

كما يستخدم أسلوب بناء ذى بارامتر حرفى يتم إرساله إلى الفصيلة الموروثة كالآتى:

```
MyCustomException(string message): base(message)
{
}
```

ويحـــتوى البارامـــتر الحـــرفى على الرسالة المراد توصيلها عند إلقاء الاستثناء.

مثال (9-8) استخدام الاستثناء المبتكر

يوضح المثال التالى طريقة إلقاء ومعالجة الاستثناءات المبتكرة.

```
// Example 9-8.cs
// Custom Exceptions

using System;

//    إعلان فصيلة الاستثناء المبتكر
public class MyCustomException: ApplicationException
{
  // The  MyCustomException class constructor:  أسلوب البناء
  public MyCustomException(string message): base(message)
  {
  }
}
class MyClass
{
  static void Main()
  {
    // Create an instance of MyCustomException:    إعلان هدف الاستثناء
    MyCustomExcept ion e =  new MyCustomException("which
includes my custom message!");
    try
    {
      // Throwing the exception:     إلقاء الاستثناء
      throw e;
    }
    // Catching the exception:    إمساك الاستثناء المبتكر ــ مقبض خاص
    catch(MyCustomE xception)
    {
      Console.WriteLine("The exception   \"{0}\" was handled.", e);
    }
    //    المقبض العام
```

```
    catch
    {
      Console.WriteLine("Unknown exception handled.");
    }
    finally
    {
     // The co  de in this block is always executed:
     Console.Write("The program continues here. ");
    }
  }
}
```

تنفيذ البرنامج:

```
The exception "MyCustomException: which includes my
custom message!
   at MyClass.Main()" was handled.
The program continues here.
```

إعادة إلقاء الاستثناء (Rethrowing)

قد يتطلب الأمر فى بعض الأحيان إعادة إلقاء الاستثناء. فقد يتعامل مع الاستثناء مقبض المعالجة بأحد الأساليب ، ثم يعيد إلقاءه إلى أسلوب آخر أو إلى الأسلوب الرئيسى بهدف الإفادة بالمزيد من المعلومات عن الخطأ الذى وقع أو بهدف إجراء المزيد من المعالجة. ويتم إعادة إلقاء الاستثناء الجارى التعامل معه باستخدام العبارة throw بدون بارامترات ، أى:

```
catch
{
  throw;
}
```

كمـــا يجوز أيضاً أن تحتوى العبارة catch على إلقاء نفس الاستثناء مع رسالة تشرح الأمر بوضوح ، مثل:

```
catch (DivideByZeroException e)
{
  throw (new DivideByZeroException("Dividing by zero occurred in
MyMethod()", e));
}
```

مثال (9-9) رفع الاستثناء إلى الأسلوب الرئيسى

فى المثال التالى نرى تسلسل إلقاء الاستثناء كالآتى:

- يقوم الأسلوب MyMethod1() باستدعاء الأسلوب MyMethod2() الذى يُلقى الاستثناء لأول مرة.
- يـــتم إمساك الاستثناء فى الأسلوب MyMethod1() ثم يقوم بإعادة إلقائه بالعبارة throw.
- يتم إمساك الاستثناء بالأسلوب الرئيسى حيث تجرى عليه المزيد من المعالجة.

ونـــرى فى نتيجة تنفيذ البرنامج تسلسل هذه الأحداث بدءاً من أول عملية إمساك.

```
// Example 9-9.cs
// Rethrowing exceptions

using System;

class MyClass
{
  static void Main()
  {
    MyClass mc = new MyClass();
```

```
    try
    {
      mc.MyMethod1();
    }
    //    إمساك الاستثناء بعد إعادة الإلقاء
    catch(E  xception e)
    {
      Console.WriteLine("Caught in Main: {0}", e);
      Console.WriteLine("More cleaning up...");

    }
  }

  public void MyMethod1()
  {
    try
    {
      MyMethod2();
    }
    //    إمساك الاستثناء أول مرة
    catch(Exception)
    {
      Console.WriteLine("Caught in MyMethod1");
      Console.WriteLine("Cleaning up chores...");
      // Rethrow the same exception:    إعادة الإلقاء
      throw;
    }
  }

  public void MyMethod2()
  {
    //    إلقاء الاستثناء لأول مرة
    throw new Exception("thrown by MyMethod2");
  }
}
```

تنفيذ البرنامج:

```
Caught in MyMethod1
Cleaning up chores...
Caught in Main: System.Exception: thrown by MyMethod2
```

```
   at MyClass.MyMethod2()
   at MyClass.MyMethod1()
   at MyClass.Main()
More cleaning up...
```

مثال (9-10) إعادة الإلقاء أثناء المعالجة

فـــى هذا المثال نرى تطبيقاً لإمساك استثناء وإعادة إلقائه بواسطة بلوك المعالجة. ثم يتم إمساك الاستثناء مرة أخرى بداخل الأسلوب الرئيسى.

```
// Example 9-10.cs
// Rethrowing exceptions back to Main

using System;

class MainClass
{
   static void Main()
   {
      MyClass mc = new MyClass();
      try
      {
         mc.MyMethod();
      }
      catch (Exception ex)
      {
         // Catch the rethrown exception:    إمساك الاستثناء بعد إعادة الإلقاء
         Console.WriteLine("Rethrown exception caught in
Main():\n{0}", ex);
      }
   }
}

class MyClass
{
   int x = 0, y = 0;
   public void MyMethod()
   {
      try
      {
```

```
        int z = x / y;        // القسمة على صفر
      }
    catch (DivideByZeroException ex)
    {
      //   إمساك الاستثناء وإعادة إلقائه لرفعه إلى الأسلوب الرئيسى
      // Catch and Rethrow the same exception:
      throw (new DivideByZeroException("Dividing    by zero
occurred in the method MyMethod() \n", ex));
    }
  }
}
```

تنفيذ البرنامج:

لاحـــظ أن تسلسل الأحداث يبدأ من لحظة الإمساك بالاستثناء الذى أعيد إلقاؤه ، أى من النهاية إلى البداية.

```
Rethrown exception caught in Main():
System.DivideByZeroException: Dividing by zero occurred
in the method MyMethod()

 ---> System.DivideByZeroException: Attempted to divide
by zero.
   at MyClass.MyMethod()
   --- End of inner exception stack trace ---
   at MyClass.MyMethod()
   at MainClass.Main()
```