الفصل الخامس عشر
معالجة الاستثناءات
Exception Handling

أثناء عملك داخل البرامج، تظهر العديد من الأخطاء التى نطلق عليها"استثناءات" Exceptionsوالتى يجب التنبه لها ومعالجتها كى تعمل برامجك بشكل صحيح.

بانتهاء هذا الفصل، ستتعرف على:

- أنواع الاستثناءات.
- العمل مع الاستثناءات الغير مُعالَجة.
- عبارات معالجة الاستثناءات.
- كيفية معالجة الاستثناءات.
- توفيق سلوك الاستثناءات.
- استخدام نافذة **Call Stack**.
- الاستثناءات المعرفة من قِبَل المستخدم.
- استخدام سجل الأحداث.

يعتبر ظهور الأخطاء داخل البرامج من الأشياء المعتادة لدى المبرمجين، فلا تتوقع أبداً أن تقوم بتطوير تطبيق متكامل دون وجود أى خطأ من الأخطاء. فتطوير البرنامج يتم على عدة مراحل حتى نحصل على الشكل النهائى الخالى من الأخطاء.

وعامةً هناك نوعان أساسيان من الأخطاء، الأول عبارة عن الأخطاء التى تحدث وقت الترجمة وتسمى **Compile-time Errors** وتنتج غالباً عن عدم كتابة الكود بشكل صحيح ولا تضر البرنامج بشئ لأن البرنامج لن يتم تنفيذه من الأساس إلا بعد إصلاح هذه الأخطاء. أما النوع الثانى فهو الأخطاء التى تحدث وقت التنفيذ وتسمى **Runtime Errors** وهذا النوع لا يمكن التنبؤ به بدرجة عالية من الدقة، لذا يطلق عليه استثناءات **Exceptions**، حيث تتطلب البرمجة الصحيحة احتواء وتوقع جميع الاستثناءات التى قد تحدث أثناء التشغيل ومعالجتها.

وعامةً يطلق على عملية إصدار الاستثناءات أثناء التشغيل "طرح الاستثناءات" **Exception Throwing** وتتم هذه العملية فى حالتين أساسيتين هما:

- عند وجود عبارة **Throw** التى تقوم بطرح الاستثناء؛أى إصدار الخطأ،ثم توجيه البرنامج إلى جزء من الكود يقوم باحتواء هذا الاستثناء ومعالجته.
- عندما يخرج برنامج **Visual Basic 2012** على غير المتوقع، وحينئذٍ يتم إصدار أو طرح الاستثناء.

سنقوم فى هذا الفصل بالتعرض للأخطاء الشهيرة التى تحدث أثناء التشغيل والطرق المختلفة لمعالجتها.

أنواع الاستثناءات

تدعم لغة **Visual Basic 2012**العديد من الاستثناءات القياسية المعرَّفةداخل تصنيفات الاستثناءات. يوضح جدول ١٥-١ التالى تصنيفات الاستثناءات المعتادة وسبب حدوثها.

جدول ١٥-١ تصنيفات الاستثناءات المعتادة وسبب حدوثها

اسم الاستثناء	سبب حدوثه
System.OutOfMemoryException	حينما لا يتمكن المعامل new من حجز أماكن بالذاكرة
System.StackOverflowException	حينما يحتوى الجزء التنفيذى بالمنطقة Stack على العديد من عمليات استدعاء الوظائف التى لا يمكنه احتوائها
System.NullReferenceException	حينما لا يحتوى النوع المرجعى المستخدم على أى قيمة
System.ArgumentNullException	حينما يتم تمرير معامل من النوع Null إلى الوظيفة على الرغم من أن هذه الوظيفة لا تتقبل هذا النوع من المعاملات
System.ArgumentOutOfRangeException	حينما تكون قيمة أحد المعاملات خارج النطاق المحدد للنوع المستخدم
System.TypeInitializationException	حينما لا تستطيع العبارة Catch المصاحبة التقاط الاستثناء الملقى بواسطة دالة الإنشاء الساكنة الخاصة بالكائن

اسم الاستثناء	سبب حدوثه
System.ArrayTypeMismatchException	عند محاولة تخزين نوع خاطئ لأحد كائنات المصفوفة
System.IndexOutOfRangeException	عند محاولة استخدام فهرس أقل من الصفر أو أكبر من المدى المتاح للمصفوفة
System.InvalidCastException	عند فشل التحويل من النوع الأساسى إلى النوع المشتق أثناء التشغيل
System.MulticastNotSupportedException	عند محاولة جمع وسيطتين لا تدعمان نوع البيانات Object
System.ArithmeticException	هذا هو التصنيف الأساسى لجميع الاستثناءات التى تحدث أثناء إجراء العمليات الحسابية
System.DivideByZeroException	عند قسمة عدد صحيح على الرقم 0
System.OverflowException	عند حدوث فيضان Overflow فى أى عملية حسابية
System.NotFiniteNumberException	عند استخدام رقم غير صحيح
System.NotSupportedException	عند محاولة الوصول إلى وظيفة لم يتم تمثيلها داخل التصنيف الحالى

اسم الاستثناء	سبب حدوثه
System.MissingMemberException	حينما يكون إصدار DLL المستخدم غير صحيح
System.AccessViolationException	عند عدم إمكانية الوصول إلى نوع عنصر معين داخل التصنيف
System.FormatException	حينما يكون تنسيق أحد المعاملات غير صحيح
System.BadImageFormatException	حينما يكون تنسيق الصورة المستخدمة غير صحيح

يعتبر التصنيف System.Exception التصنيف الأساسى لجميع تصنيفات الاستثناءات الأخرى، لذا يتم تطبيق الخاصيتين التاليتين على جميع التصنيفات المشتقة من هذا التصنيف:

- الخاصية Message: وتقوم بإظهار رسالة على الشاشة فى حالة حدوث الاستثناء والتى تحتوى بدورها على سبب حدوث هذا الاستثناء. فمثلاً عند طرح الاستثناء DivideByZeroException، تظهر الرسالة التالية:

```
System.DevideByZeroException: Attempt to divide by zero
at classname:methodname
```

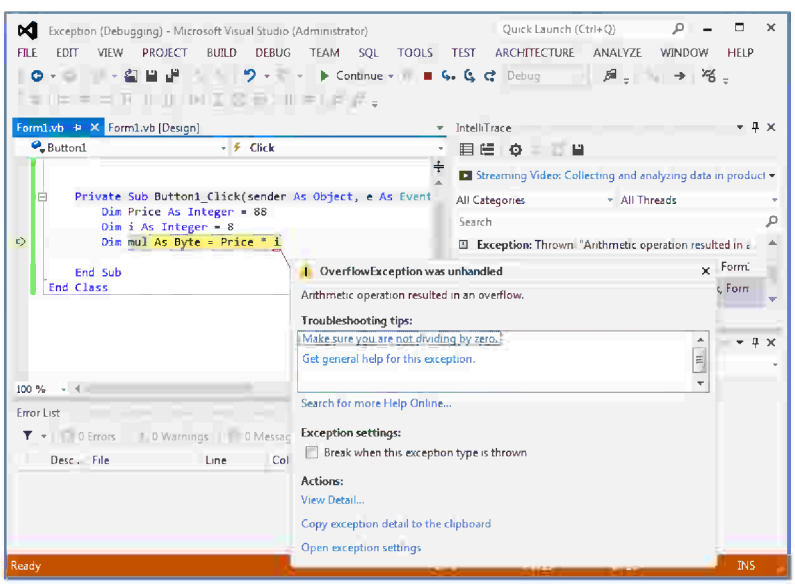
- الخاصية InnerException: وتقوم بتوضيح الاستثناء الذى أدى بدوره إلى ظهور الاستثناء الحالى. ففى حالات معينة يتسبب أحد الاستثناءات فى ظهور استثناء آخر. أما إذا لم يكن هناك استثناء تسبب فى ظهور الاستثناء الحالى، فإن قيمة هذه الخاصية تكون Null.

العمل مع الاستثناءات الغير مُعالَجة

لتبسيط الشرح الوارد فى هذا الفصل، دعنا نرى مثالاً بسيطاً عن المواقف التى يقوم المترجم فيها بإظهار الاستثناء (الخطأ). انظر معى إلى الكود التالى:

```
Dim Price As Integer = 88
Dim i As Integer = 8
Dim mul As Byte = Price * i
```

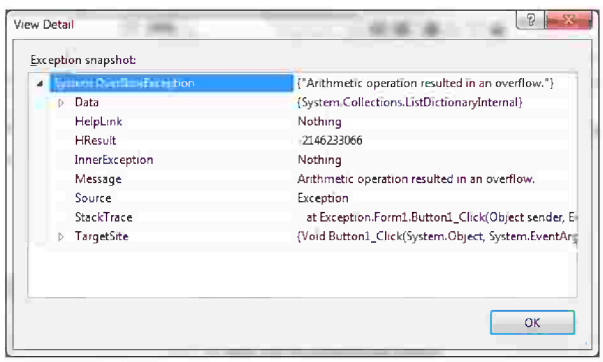
يتم فى السطر الثالث من هذا الكود إيجاد حاصل ضرب قيمتين من النوع **Integer** وإرجاع القيمة الناتجة إلى متغير من النوع **Byte**. ولأن نوع البيانات **Byte** لا يقدر على احتواء القيمة الناتجة وهو ما يسمى "طفح البيانات" **Overflow**، يقوم**Visual Studio2012** بإظهار مربع حوارى يحتوى على نوع الاستثناء وشرح مختصر له مع إعطاء بعض الخيارات من خلال مجموعة من الارتباطات التى نوضحها فيما يلى (انظر شكل ١٥-١):



شكل ١٥-١ يظهر هذا المربع عند حدوث استثناء غير مُعالَج

- مربع المجموعة **troubleshooting tips** ويحتوى على عدد من الخيارات التى يحاول كل منها التنبأ بسبب المشكلة وإظهار معلومات عن كيفية التغلب عليها. ففى الاستثناء الذى بين أيدينا يوجد خياران، الأول **Make sure you are not dividing by zero** والذى بنقره يحثك **Visual Basic** على التأكد من عدم القسمة على الرقم 0، والثانى **Get general help for this exception** والذى بنقره يتم عرض شرح للاستثناء وأسباب ظهوره وكيفية معالجته.

- الارتباط **Search for more Help Online** ويستخدم فى طلب المساعدة عن كيفية التعامل مع الاستثناء من خلال بيانات المساعدة الموجودة على الويب.
- الارتباط **View Details** ويستخدم فى إظهار مربع حوارى آخر يحتوى على تفصيلات الاستثناء (انظر شكل ١٥-٢).



شكل ١٥-٢يحتوى هذا المربع على تفاصيل الاستثناء

- الارتباط **Copy exception detail to the clipboard** ويستخدم فى نسخ تفاصيل الاستثناء إلى الحافظة حتى تتمكن من لصقها فى أى مكان آخر باستخدام الأمر **Paste**.

عبارات معالجة الاستثناءات

يتم معالجة الاستثناءات داخل لغة **Visual Basic 2012** من خلال مجموعة من العبارات. وفيما يلى نتعرض بالشرح لكل عبارة من هذه العبارات.

العبارة Throw

تتكون هذه العبارة من كلمة **Throw** متبوعةً بالتعبير الذى من أجله تم طرح الاستثناء، حيث يتم طرح الاستثناء بمجرد إيجاد قيمة هذا التعبير، كما يجب أن يكون هذا الاستثناء أحد حالات التصنيف **System.Exception** أو تصنيفاته المشتقة. يوضح الكود التالى العبارة **Throw** التى تقوم بطرح الاستثناء **ArrayTypeMismatchException**:

```
Throw (New ArrayTypeMismatchException())
```

من الممكن ألا تحتوى عبارة Throw على التعبير المصاحب وذلك عند استخدام العبارة داخل كتلة كود تسمى Catch كما سنرى بعد قليل.

العبارة Try

تقوم العبارة Try بتحديد سطور الكود التى ربما تقوم بطرح الاستثناء، حيث تتكون هذه العبارة من كلمة Try متبوعةً بمحتوى العبارة الذى يحيط بدوره بجزء الكود الذى ربما يقوم بطرح أحد الاستثناءات. كما تقوم العبارة أيضاً بتوجيه البرنامج إلى الكود الذى يتم تنفيذه فى حالة حدوث الاستثناء. يمكنك أيضاً داخل Visual Basic استخدام العديد من عبارات Try المتداخلة. ودائماً يتم استخدام عبارة Try مع العبارات Catch وFinally، لذا فهى دائماً تكون فى إحدى الصور الثلاثة الآتية:

- العبارة Try-Catch
- العبارة Try-Finally
- العبارة Try-Catch-Finally

العبارة Try-Catch

تعتبر العبارة Try-Catch مزيج من عبارتى Try وCatch، حيث تتكون من العبارة Try متبوعةً بحالة أو أكثر من العبارة Catch التى تحتوى بدورها على الكود اللازم لمعالجة الاستثناءات المختلفة التى تم طرحها بواسطة العبارة Tryوالتى تحتوى على سطور الكود التى ربما تقوم بطرح أحد الاستثناءات. وعلى هذا فهناك احتمالين لنتيجة عبارة Try، الاحتمال الأول هو تنفيذ الكود بنجاح دون حدوث أى مشاكل، أما الاحتمال الثانى فهو طرح أحد الاستثناءات وفى هذه الحالة يتم توجيه البرنامج مباشرةً إلى كود العبارة Catch التى تتولى معالجة الاستثناء الذى تم طرحه.

يوجد نوعان من عبارات Catch، فهناك عبارة Catch العامة التى لا تحتوى على معاملات والتى تستخدم لاحتواء أى نوع من الاستثناءات وصيغتها العامة كما يلى:

```
Try
.....
```

```
Catch
        يتم هنا كتابة الكود اللازم لاحتواء الاستثناءات'
End Try
```

أما النوع الثانى فهو عبارة **Catch** التى تحتوى على معامل واحد يتمثل فى الاستثناء الذى تقوم العبارة باحتوائه، حيث يجب أن يكون المعامل أحد الأنواع المشتقة من التصنيف الأساسى **System.Exception** وصيغة هذه العبارة كما يلى:

```
Try
.....
Catch ThisExcpAsException_Type
'Exception_Typeيتم هنا كتابة الكود اللازم لاحتواء الاستثناءات ذات النوع
End Try
```

إذا احتوت عبارة **Catch** على المعامل **System.Exception** فهذا يعنى أنها قادرة على احتواء جميع الاستثناءات التى تشتق أساساً؛ كما نعلم، من هذا التصنيف. وحينما تُتبع عبارة **Try** بأكثر من حالة للعبارة **Catch**، فإن ترتيب هذه العبارات أمرٌ فى غاية الأهمية، حيث يقوم البرنامج بتنفيذ كل عبارة من هذه العبارات تبعاً لترتيبها بالكود، لذا يجب تعريف العبارات التى تقوم بمعالجة استثناءات معينة قبل تعريف العبارة العامة التى لا تحتوى على معاملات والتى تستخدم لمعالجة جميع الاستثناءات بأنواعها المختلفة.

من الممكن أن يحتوى كود العبارة **Catch** بداخله على العبارة **Throw** التى تقوم بإعادة طرح الاستثناء، وهذه العبارة تختلف باختلاف عبارة **Catch** التى تعرَّف بداخلها. وإذا لم تحتوى عبارة **Throw** على معامل من الأساس، فهذا يعنى أنها معرفة داخل عبارة **Catch** عامة تستخدم لمعالجة جميع الاستثناءات كما فى الكود التالى:

```
Try
.....

Catch
        Throw
End Try
```

لتوضيح الرؤية أكثر وللتعرف على كيفية استخدام عبارتى **Try** و**Catch**، دعنا نرى الكود التالى (يوجد هذا الكود على القرص المدمج المرفق داخل مشروع باسم **Exception**):

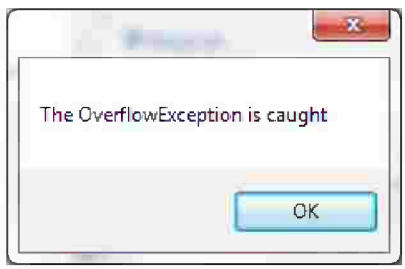
```
1.  Dim Price As String
2.  Price = InputBox("Please enter the price in dollars")
3.  Dim iprice As Integer = Int32.Parse(Price)
4.  Dim i As Integer = 0
5.  Try
6.  Dim div As Integer = iprice / i
7.  Catch div As OverflowException
8.  MessageBox.Show("The OverflowException is caught")

9.  Catch ex As System.Exception
10. MessageBox.Show("The " + ex.GetType.ToString +
"Exception is caught, please try again")
11. End Try
```

وعن هذا الكود، نوضح ما يلى:

- فى السطور ٢ و٣ يتم حث المستخدم على إدخال سعر الكتاب مع استقبال هذه القيمة من لوحة المفاتيح وتحويلها إلى قيمة صحيحة.
- فى السطور من ٥ إلى ١١ يتم وضع الكود المراد اختباره داخل عبارة **Try** حيث يتم اختبار تنفيذ قسمة عدد صحيح على عدد صحيح آخر يحتوى على القيمة **0** وبالتالى تقوم عبارة **Try** بطرح الاستثناء أو الخطأ **OverflowException** الذى يتم معالجته فيما بعد من خلال عبارة **Catch**.
- تحتوى السطور ٧ و٨ على كود عبارة **Catch** الأولى التى تحتوى على معامل من النوع **OverflowException**أى أن هذه العبارة مخصصة للاستثناءات ذات النوع **OverflowException**. وفى هذه العبارة يتم طباعة الاستثناء على الشاشة.
- تحتوى السطور ٩و١٠ على كود عبارة **Catch** الثانية المستخدمة لمعالجة أى نوع من أنواع الاستثناءات حيث يتم أيضاً طباعة الاستثناء على الشاشة.

فى الكود السابق بمجرد اكتشاف الاستثناء داخل عبارة **Try**، يتم تنفيذ الكود الموجود داخل عبارة **Catch** الأولى واستمرار التطبيق (انظر شكل ١٥-٣). قم بتنفيذ التطبيق ثم أدخل القيمة صفر للحصول على رسالة الاستثناء.



شكل ١٥-٣يتم تنفيذ كود عبارة Catch المناسبة بمجرد وقوع الاستثناء

العبارة *Try-Finally*

تستخدم العبارة **Finally** لإجراء عملية تنظيف عامة لجميع الموارد المستخدمة داخل عبارة **Try** التى تسبقها. وفى حالة وجود العبارة **Finally** يتم تنفيذها بغض النظر عن الطريقة التى تم بها مغادرة عبارة **Try**، كما يتم تنفيذها قبل الاستثناءات التى تم طرحها بواسطة عبارة **Try**.

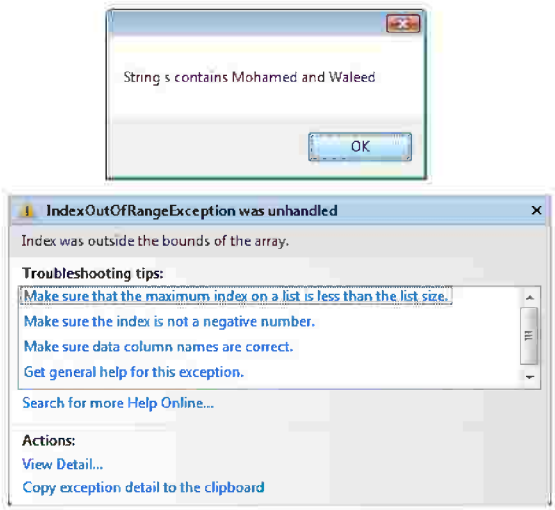
لتوضيح طريقة عمل العبارة **Try-Finally**، انظر معى إلى الكود التالى:

```
1.  Dim s() As String = {"Mohamed", "Waleed"}
2.  Try
3.  s(2) = "Elsayed"
4.  Finally
5.  MessageBox.Show("String s contains Mohamed and
                                        Waleed")
6.  End Try
```

وعن هذا الكود، نوضح ما يلى:

- فى السطر رقم ١ تم تعريف المصفوفة S التى تحتوى على عنصرين فقط من النوع stringوهما "S[0]="Mohamed و"S[1]="Waleed.
- تحتوى السطور ٢و٣ على عبارة Try وبها الكود المطلوب اختباره وهو تخصيص قيمة للعنصر الثالث بالمصفوفة. ولأن المصفوفة لا تحتوى أساساً إلا على عنصرين فقط، تقوم العبارة Try بطرح الاستثناء IndexOutOfRangeException والذى لا يتم معالجته لعدم وجود عبارة Catch داخل العبارة Try.

- تحتوى السطور من ١١ إلى ١٥ على العبارة **Finally** التى يتم تنفيذها حتى ولو كان هناك استثناء (خطأ) داخل عبارة **Try**. وبالتالى فعند تنفيذ هذا الكود، تظهر الرسالة الموجودة بعبارة **Finally** ثم يظهر الاستثناء (انظر شكل ١٥-٤).



شكل ١٥-٤ يتم تنفيذ كود عبارة **Finally** حتى فى حالة وجود استثناء داخل عبارة **Try**

العبارة Try-Catch-Finally

لعلك الآن تتوقع الصيغة العامة لهذه العبارة، فهى حقيقةً بمثابة الصيغة الكاملة لعبارة **Try**، حيث يتم وضع الكود الذى ربما يقوم بطرح استثناءات داخل العبارة **Try**. فإذا ظهرت أى استثناءات، يتم تنفيذ الكود الموجود بالعبارة **Catch** وأخيراً تقوم العبارة **Finally** بتحرير الموارد المستخدمة من قِبَل عبارتى **Try** و**Catch** بغض النظر عن وجود استثناءات أم لا.

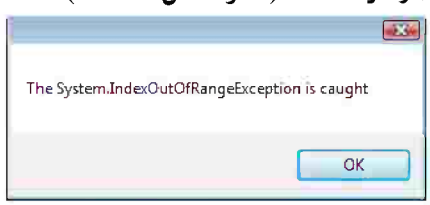
لا يمكنك استخدام عبارة **return** داخل كود العبارة **Finally** لأنها لا تقوم بإرجاع أى بيانات.

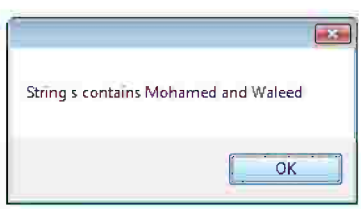
للتعرف عن قرب على طريقة عمل العبارة **Try-Catch-Finally**، قم بتعديل الكود السابق كما يلى:

```
1.  Dim s() As String = {"Mohamed", "Waleed"}
2.  Try
3.  s(2) = "Elsayed"
```

```
4.    Catch exp As Seystem.Exception
MessageBox.Show("The " + exp.GetType.ToString + "
is caught ")
5.    Finally
6.    MessageBox.Show("String s contains Mohamed and
                                        Waleed")
7.    End Try
```

وعند تنفيذ هذا الكود، يتم معالجة الاستثناء عن طريق الرسالة الموجودة بالعبارة **Catch** ثم تظهر الرسالة الموجودة بعبارة **Finally** (انظر شكل ١٥-٥).





شكل ١٥-٥ استخدام العبارة **Try-Catch-Finally**.

لا يمكنك استخدام عبارة **Try** مع عبارة **Catch** العامة التى لا تحتوى على معاملات وعبارة **Catch** التى تحتوى على المعامل **System.Exception** فى نفس الوقت.

لا يمكنك استخدام العبارات **break** و**continue** و**return** و**goto** لنقل التحكم خارج العبارة **Finally** حيث يقوم المترجم بالاعتراض وإظهار رسالة الخطأ المناسبة.

كيفية معالجة الاستثناءات

لمعالجة الاستثناءات كما رأينا، يتم وضع سطور الكود التى من الممكن أن ينتج عن تنفيذها خطأ أو أكثر داخل عبارة **Try**، حيث تقوم هذه العبارة تلقائياً بطرح رسالة الخطأ المناسبة التى نطلق عليها "استثناء" **Exception**. يمكنك أيضاً إصدار الاستثناء صراحةً من خلال العبارة **Throw**. وحينما يحدث خطأ ما أو استثناء ما، يقوم النظام بالبحث عن أقرب عبارة **Catch**والتى يتم استخدامها لمعالجة هذا الخطأ أو الاستثناء.

يطلق على المسار الذى يأخذه النظام بداية من لحظة اكتشاف الخطأ أو الاستثناء وحتى يتم الوصول إلى معالج الاستثناء المناسب أو العبارة **Catch** "بث الاستثناء" **Exception Propagation**.

إذا صاحب عبارة **Try** التى قامت بإصدار الاستثناء أكثر من عبارة **Catch**، يتم تنفيذ هذه العبارات طبقاً لترتيبها داخل الكود، حيث يتم مقارنة الاستثناء بمعامل عبارة **Catch**، فإذا حدث توافق يتم تنفيذ كود العبارة وإلا يتم الذهاب إلى العبارة التالية إلى أن نصل إلى عبارة **Catch** المناسبة وحينئذٍ ينتهى بث الاستثناء. أما إذا لم يجد النظام عبارة **Catch** المناسبة، فيتم البحث عن العبارة **Finally** التى يتم بداخلها معالجة الاستثناء. فإذا قامت عبارة **Finally** هى الأخرى بإصدار استثناء آخر، يتم قطع معالجة الاستثناء الأول والبدء فى معالجة الاستثناء الأخير، وإلا يتم معالجة الاستثناء الأول إلى نهايته. وعلى الجانب الآخر، إذا لم يجد النظام لا عبارة **Catch** ولا عبارة **Finally**، يتم قطع البرنامج مباشرةً.

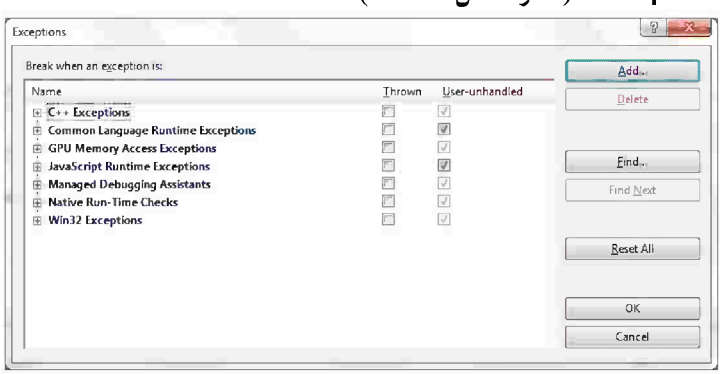
إذا وجدت مجموعة من العبارات **Try** المتداخلة، تبدأ عملية المعالجة على العبارة الداخلية أولاً ثم العبارة التى تليها إلى أن نصل إلى العبارة الخارجية.

توفيق سلوك الاستثناءات

عند تعيين كود معالجة الاستثناء من خلال عبارات الاستثناء التى ذكرناها منذ قليل وتقوم بتنفيذ التطبيق ويظهر الاستثناء، يتم توجيه التطبيق مباشرةً إلى معالج الاستثناء المناسب (راجع شكل ١٥-١). ولكن ربما لا تحبذ كمبرمج أن يتم دائماً تنفيذ معالج الاستثناء تلقائياً لأن ذلك يتسبب فى إيقاف تنفيذ البرنامج دون تمكينك من معالجة هذا الخطأ حتى لا يتكرر

مرة ثانية. لذلك، يتيح لك **Visual Studio2012** التحكم فى معالجة الاستثناءات وتوفيق سلوكها باستخدام المربع الحوارى **Exceptions** الذى يمكنك من خلاله تحديد ما إذا كان **Visual Studio** يستمر فى معالجة الاستثناء بمجرد ظهوره أم يتوقف تبعاً لنوع هذا الاستثناء. والوضع الافتراضى هو الاستمرار فى حالة وجود كود المعالجة المناسب أو التوقف فى حالة عدم وجود هذا الكود. وعلى ذلك يمكنك إخبار **Visual Studio** بالتوقف عند استثناءات معينة حتى فى حالة وجود كود المعالجة المناسب. لتوضيح ذلك، تابع معنا الخطوات الآتية:

١. افتح قائمة **Debug** من شريط القوائم ثم اختر **Exceptions** من القائمة المنسدلة (أو اضغط الاختصار **Ctrl+Alt+E** من لوحة المفاتيح)، يظهر المربع الحوارى **Exceptions** (انظر شكل ١٥-٦).



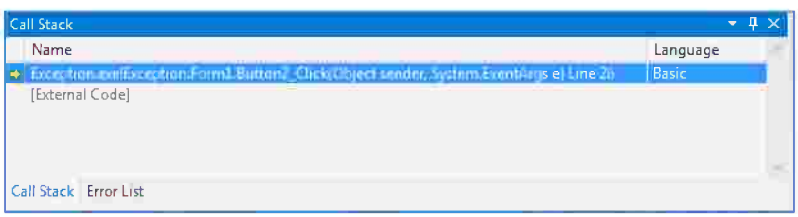
شكل ١٥-٦ توفيق سلوك الاستثناءات من خلال المربع الحوارى **Exceptions**

٢. يتم ترتيب الاستثناءات فى صورة هرمية، وبالتالى يمكنك تحديد استثناء معين داخل المجموعة أو مجموعة كاملة من الاستثناءات. فقط قم باختيار الاستثناء المطلوب.

٣. إذا أردت توقف البرنامج مؤقتاً وإظهار المربع الحوارى الخاص بالاستثناء بمجرد طرح الاستثناء، نشط مربع الاختيار المقابل للاستثناء بالعمود **Thrown**. وإذا أردت التوقف مؤقتاً حتى فى حالة وجود كود معالجة للاستثناء، نشط مربع الاختيار المقابل للاستثناء بالعمود **User-unhandled**.

استخدام نافذة Call Stack

تعتبر نافذة **Call Stack** من الأدوات الهامة التى يمكنك استخدامها عند معالجة الاستثناء أثناء طور التوقف المؤقت **Break Mode**، حيث تحتوى هذه النافذة دائماً على قائمة بعمليات استدعاء الإجراءات المختلفة، وتظهر قوة هذه النافذة أساساً عند عمليات الاستدعاء المتداخلة. لإظهار هذه النافذة فى طور التوقف، افتح قائمة **Debug** من شريط القوائم ثم اختر **Windows** و**Call Stack** من القوائم المنسدلة (انظر شكل ١٥-٧).



شكل ١٥-٧ نافذة **Call Stack**

الاستثناءات المعرفة من قِبَل المستخدم

تعرضنا حتى الآن إلى تصنيفات الاستثناءات المعرفة من قِبَل النظام، إلا أن لغة **Visual Basic** تدعم أيضاً إنشاء واستخدام التصنيفات المعرفة من قِبَل المستخدم وهى تصنيفات الاستثناء التى يقوم المستخدم بتعريفها لضمان إصدار البرنامج رسائل لأخطاء معينة حينما يتم إصدار الاستثناءات أثناء تنفيذ البرنامج.

وحقيقةً فإن هناك بعض القيود على تصنيفات الاستثناء المعرفة من قِبَل المستخدم، حيث يجب أن تقوم هذه التصنيفات بالإعلان عن معظم دوال الإنشاء المعتادة داخل التصنيفات الأساسية، كما يجب أن ينتهى اسم التصنيف بكلمة "**Exception**"، كما يجذ أيضاً تجميع كل الاستثناءات المعرفة من قِبَل المستخدم داخل مسمى واحد **Namespace** للتنظيم الجيد لجميع تصنيفات الاستثناءات سواءً المعرفة من قِبَل النظام أو المعرفة من قِبَل المستخدم.

للتعرف على طريقة تعريف واستخدام تصنيفات الاستثناءات المعرفة من قِبَل المستخدم، دعنا نرى الكود التالى:

```
1.  Imports System
2.  Public Class VirtualBooksException
3.  Inherits System.Exception
4.  ' يمكنك هنا تعريف الكائنات المختلفة
5.  End Class

6.  Public Sub Search(s As String)
7.  MessageBox.Show("Sorry. You cannot search for " + s
                                        + " at the moment")
8.  Throw (New VirtualBooksException())
9.  End Sub

10. Private Sub Button4_Click(sender As Object, e As
    EventArgs) Handles Button1.Click

11. Try
12. Dim str As String = InputBox("Please enter the item
                                    you wish to search for")
13. Search(str)
14. Catch vbe As VirtualBooksException
15. MessageBox.Show("The " + vbe.GetType.ToString + "
                                    exception has been caught")
16. Finally

17. MessageBox.Show("Please try again later")
18. End Try

19. End Sub
```

وعن هذا الكود، نوضح ما يلى:

- فى السطر رقم ٢ تم تعريف تصنيف استثناء باسم **VirtualBooksException** مشتق من تصنيف الاستثناءات العام **System.Exception**.
- تحتوى السطور من ٦ إلى ٩ على كود الإجراء**()Search** الذى يحتوى على معامل واحد من النوع **String** ويقوم بطباعة رسالة على الشاشة فحواها عدم القدرة على

البحث عن العنصر فى الوقت الحالى ثم طرح استثناء من النوع VirtualBooksException الذى قمنا بتعريفه فى بداية الكود.

- تحتوى السطور من ١١ إلى ١٥على كود العبارة try المستخدمة لاختبار تنفيذ الكود ويتم فيها استقبال عنصر من المستخدم وتمريره إلى الإجراء()Search الذىيقوم بدوره بإظهار الرسالة على الشاشة وطرح الاستثناء.
- تحتوى السطور ١٤و١٥ على كود العبارة catch التى يتم تنفيذها فى حالة طرح العبارة try للاستثناء VirtualBooksException حيث يتم بإظهار رسالة بالاستثناء على الشاشة.
- تحتوى السطور ١٦و١٧ على كود العبارة finally التى تقوم بإظهار رسالة على الشاشة.

إذا قمت بترجمة الكود السابق وتنفيذه، تحصل على مجموعة من الرسائل المشابهة لتلك الموجودة بشكل ١٥-٨ التالى.

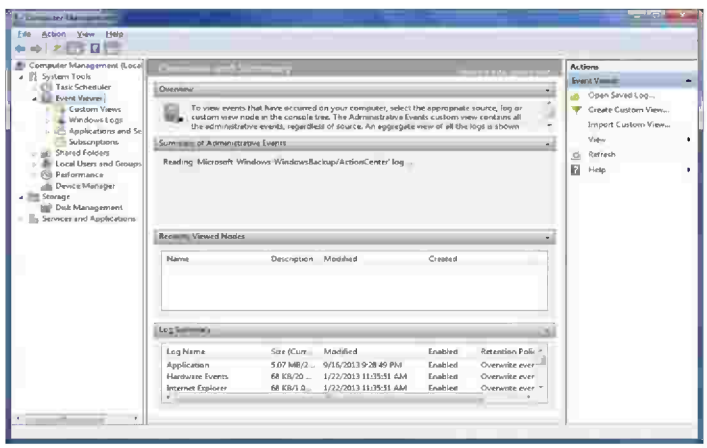
شكل ١٥-٨ استخدام الاستثناءات المعرفة من قِبَل المستخدم.

استخدام سجل الأحداث

أثناء العمل داخل بيئة تطوير Visual Studio2012، يمكنك استخدام نافذة Call Stack للبحث عن العنصر الذى تسبب فى ظهور الاستثناء. أما أثناء التشغيل، فيمكنك استخدام سجل أحداث Windows لما يحتويه من ميزات عديدة يمكن إجمال أهمها فيما يلى:

- يعتبر سجل الأحداث منطقة مركزية يمكنك من خلالها التعرف على الأحداث التى تتم على الخادم بالكامل.

- اعتياد مسئولى الشبكات ومستخدمى أنظمة التشغيل على برنامج عارض الأحداث المبنى داخل **Windows** (انظر شكل ١٥-٩).



شكل ١٥-٩ التحكم فى الأحداث المختلفة من خلال عارض الأحداث

- يمكنك من خلال عارض الأحداث اختبار أحد سجلات الأحداث عن بعد.
- يمكنك استخدام كود **Visual Basic** للوصول إلى أحد سجلات الأحداث والتعديل فيها.

وقبل أن نتناول التفصيلات الفنية فى التعامل مع سجل الأحداث، دعنا نقوم بإلقاء نظرة خاطفة على المصطلحات المستخدمة مع سجلات الأحداث، حيث يوجد ثلاثة مصطلحات أساسية يمكن بيانها فيما يلى:

- *عنصر الحدث* ***Event Entry*** وهو عبارة عن حدث على الحاسب تم إضافته إلى أحد السجلات. وليس بالضرورة أن تكون هذه الأحداث عبارة عن أخطاء. فحينما تقوم بتشغيل الحاسب على سبيل المثال، يتم إدخال العديد من الأحداث للإعلام فقط. يحتوى عارض الأحداث على مجموعة من الرموز التى يتم استخدامها لتوضيح نوع الخطأ.
- *سجل الأحداث* ***Event Log*** ويحتوى على سجل من الأحداث المرتبطة. فهناك سجل الأحداث **Security** على سبيل المثال والذى يحتوى على الأحداث المرتبطة

بأمان النظام. ويحتوى نظام التشغيل Windows تلقائياً على ثلاثة سجلات هى Application وSystem وSecurity إلا أنك تستطيع إضافة المزيد من السجلات بسهولة تامة من خلال عارض الأحداث Event Viewer.

- *مصدر الحدث* ***Event Source*** ويقوم بتعريف الكائن المتسبب فى الحدث، والذى قد يكون اسم التطبيق على سبيل المثال.

يمكنك التعامل مع سجلات الأحداث داخل Windows من خلال المسمى System.Diagnostics. سنقوم فيما يلى بالتعرف على كيفية الكتابة لسجل الأحداث أو القراءة منه.

كتابة الأخطاء إلى سجل الأحداث

لكتابة الأخطاء إلى أحد سجلات الأحداث، يجب أن تقوم بتعيين اسم مصدر الحدث، وهو عبارة عن قيمة حرفية تكون غالباً اسم تطبيقك. ومع كل مصدر من مصادر الأحداث، يقوم Windows بالاحتفاظ بسجل الأحداث المصاحب، وسجل الأحداث الافتراضى فى هذه الحالة هو السجل Application، إلا أنك تستطيع تعيين سجل أحداث مخصص. لربط مصدر الحدث بسجل أحداث مخصص، يمكنك استخدام الوظيفة CreateEventSource ومن ثمَّ يقوم نظام التشغيل بتذكر هذا السجل من هذه اللحظة. ويمكنك بعد ذلك كتابة المعلومات إلى سجل الأحداث من خلال الوظيفة WriteEntery() كما فى الكود التالى:

```
Public Sub LogException(FunctionName As String,
NewException As Exception)

Dim evtLog As New System.Diagnostics.EventLog()
If Not evtLog.SourceExists(FunctionName) Then
evtLog.CreateEventSource(FunctionName, "MyLog")
End If
evtLog.WriteEntry(FunctionName, NewException.ToString,
                                  EventLogEntryType.Error)
evtLog.Close()
```

```
End Sub
```

يحتوى هذا الكود على الإجراء LogException() الذى يقوم باستقبال معاملين، أحدهما اسم الدالة والثانى نوع الاستثناء، ثم يقوم بكتابة الاستثناء إلى سجل أحداث مخصص باسم MyLog من خلال الوظيفة WriteEntry() التى تحتوى على ثلاثة معاملات، اسم الدالة والاستثناء ونوع الحدثEventLogEntryType. وبالتالى يمكنك إضافة هذا الإجراء إلى تطبيقك وكتابة الأخطاء إلى سجل الأحداث بمجرد حدوثها من خلال سطر واحد من الكود يتم فيها استدعاء الإجراء مع تمرير المعاملات المناسبة كما فى الكود التالى:

```
Try
        Call ProcessInvoice(sngAmount,arDetails)
        Catch ex As Exception
        Dim NewEx As New Exception("Error in
        ProcessInvoice",ex)
        LogException("InvoiceApp",MyEx)
        Throw MyEx
End Try
```

قراءة سجل الأحداث

بالإضافة إلى إمكانية الكتابة إلى سجل الأحداث، حيث يتم تخزين الأحداث دائماً داخل المجموعة Entries المصاحبة للكائن EventLog. يمكنك أيضاً القراءة من سجل الأحداث باستخدام كود Visual Basic 2012.ففى الكود التالى على سبيل المثال يتم قراءة محتويات السجل MyLog، فإذا احتوى على عنصر جديد به كلمة Problem، يتم إظهار رسالة بذلك للمستخدم:

```
Private Sub btnStart_Click(sender As Object, e As EventArgs)
Handles btnStart.Click
    Dim evtTemp As System.Diagnostics.EventLog
evtTemp = New System.Diagnostics.EventLog("MyLog")
evtTemp.EnableRaisingEvents = True
AddHandlerevtTemp.EntryWritten, AddressOfCheckEvent
End Sub

Private Sub CheckEvent(sender As Object, args As
System.Diagnostics.EntryWrittenEventArgs)
    If InStr(args.Entry.Message.ToUpper, "PROBLEM") <> 0
```

```
        Then
MessageBox.Show("we have a problem!")
    End If
End Sub
```