

الفصل الخامس عشر معالجة الاستثناءات Exception Handling

أثناء عملك داخل البرامج، تظهر العديد من الأخطاء التي نطلق عليها "استثناءات" Exceptions والتي يجب التنبيه لها ومعالجتها كي تعمل برامجك بشكل صحيح.

بانتهاء هذا الفصل، ستتعرف على:

- أنواع الاستثناءات.
- العمل مع الاستثناءات الغير مُعالجة.
- عبارات معالجة الاستثناءات.
- كيفية معالجة الاستثناءات.
- توفيق سلوك الاستثناءات.
- استخدام نافذة **Call Stack**.
- الاستثناءات المعرفة من قِبل 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 المعروفة داخل 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 يحتوي الجزء التنفيذي بالمنطقة Stack على العديد من عمليات استدعاء الوظائف التي لا يمكنه احتوائها
System.NullReferenceException	حينما لا يحتوي النوع المرجعي المستخدم على أى قيمة
System.ArgumentNullException	حينما يتم تمرير معامل من النوع Null إلى الوظيفة على الرغم من أن هذه الوظيفة لا تقبل هذا النوع من المعاملات
System.ArgumentOutOfRangeException	حينما تكون قيمة أحد المعاملات خارج النطاق المحدد للنوع المستخدم
System.TypeInitializationException	حينما لا تستطيع العبارة Catch المصاحبة النقاط الاستثناء الملقى بواسطة دالة الإنشاء الساكنة الخاصة بالكائن

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

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

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

- الخاصية **Message**: وتقوم بإظهار رسالة على الشاشة في حالة حدوث الاستثناء والتي تحتوي بدورها على سبب حدوث هذا الاستثناء. فمثلاً عند طرح الاستثناء **DivideByZeroException**، تظهر الرسالة التالية:

System.DivideByZeroException: Attempt to divide by zero at classname:methodname

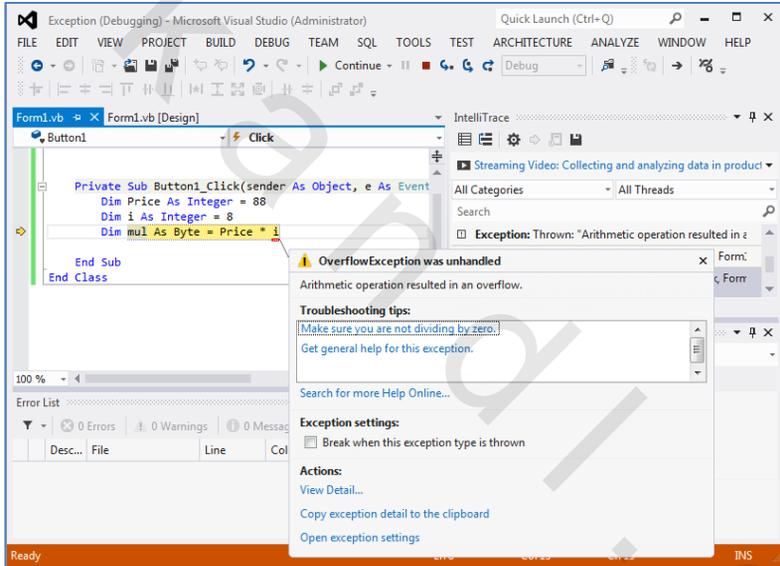
- الخاصية **InnerException**: وتقوم بتوضيح الاستثناء الذي أدى بدوره إلى ظهور الاستثناء الحالي. ففي حالات معينة يتسبب أحد الاستثناءات في ظهور استثناء آخر. أما إذا لم يكن هناك استثناء تسبب في ظهور الاستثناء الحالي، فإن قيمة هذه الخاصية تكون **Null**.

العمل مع الاستثناءات الغير مُعالجة

لتبسيط الشرح الوارد في هذا الفصل، دعنا نرى مثلاً بسيطاً عن المواقف التي يقوم المترجم فيها بإظهار الاستثناء (الخطأ). انظر معي إلى الكود التالي:

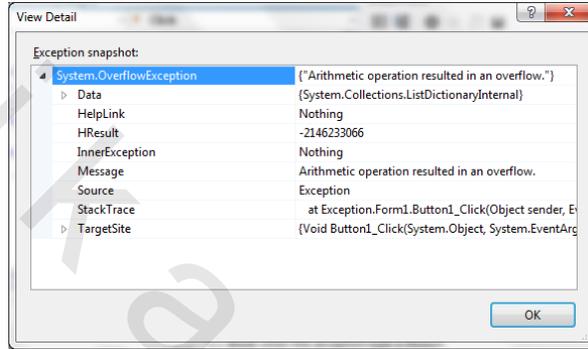
Dim Price As Integer = 88
Dim i As Integer = 8
Dim mul As Byte = Price * i

يتم في 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 "تفح البيانات" **Overflow**، يقوم **Visual Studio 2012** 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 مُعالج
 مربع المجموعة **troubleshooting tips** ويحتوى على عدد من الخيارات التى يحاول كل منها التنبأ بسبب المشكلة وإظهار معلومات عن كيفية التغلب عليها. ففي 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 في طلب المساعدة عن كيفية التعامل مع الاستثناء من خلال بيانات المساعدة الموجودة على الويب.
- الارتباط **View Details** ويستخدم في إظهار مربع حوارى آخر يحتوى على تفاصيل الاستثناء (انظر شكل ١٥-٢).



شكل ١٥-٢ يحتوى هذا المربع على تفاصيل الاستثناء

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

معالجة الاستثناءات

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

العبارة **Throw**

تتكون هذه العبارة من كلمة **Throw** متبوعاً 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 التي 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** و **Finally**، 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** و **Catch**، 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 **Try** والتى تحتوى على سطور الكود التى ربما تقوم بطرح أحد الاستثناءات. وعلى هذا فهناك احتمالان 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 الثاني فهو عبارة **Catch** التي تحتوى على معاميل واحد يتمثل في الاستثناء الذي تقوم العبارة باحتوائه، حيث يجب أن يكون المعامل أحد الأنواع المشتقة من التصنيف الأساسي **System.Exception** وصيغة هذه العبارة كما يلي:

Try

.....

Catch ThisExcpAsException_Type

يتم هنا كتابة الكود اللازم لاحتواء الاستثناءات ذات النوع **'Exception_Type**

End Try

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

من الممكن أن يحتوى كود العبارة **Catch** بداخله على العبارة **Throw** التي تقوم بإعادة طرح الاستثناء، وهذه العبارة تختلف باختلاف عبارة **Catch** التي تعرّف بداخلها. وإذا لم تحتوى عبارة **Throw** على معاميل من الأساس، فهذا يعنى أنها معرفة داخل عبارة **Catch** عامة تستخدم لمعالجة جميع الاستثناءات كما في الكود التالي:

Try

.....

Catch

Throw

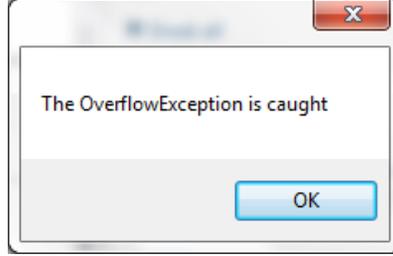
End Try

لتوضيح الرؤية أكثر وللتعرف على كيفية استخدام عبارتي **Try** و **Catch**، دعنا نرى الكود التالي (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 يلي:

- في السطور ٢ و ٣ يتم حث المستخدم على إدخال سعر الكتاب مع استقبال هذه القيمة من لوحة المفاتيح وتحويلها إلى قيمة صحيحة.
 - في 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 وبالتالي تقوم عبارة Try بطرح الاستثناء أو الخطأ OverflowException الذى يتم معالجته فيما بعد من خلال عبارة Catch.
 - تحتوى السطور ٧ و ٨ على كود عبارة Catch الأولى التى تحتوى على معامل من النوع OverflowException أى أن هذه العبارة مخصصة للاستثناءات ذات النوع OverflowException. وفي هذه العبارة يتم طباعة الاستثناء على الشاشة.
 - تحتوى السطور ٩ و ١٠ على كود عبارة Catch الثانية المستخدمة لمعالجة أى نوع من أنواع الاستثناءات حيث يتم أيضاً طباعة الاستثناء على الشاشة.
- في 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 ١٥-٣ يتم تنفيذ كود عبارة Catch المناسبة بمجرد وقوع الاستثناء

العبارة Try-Finally

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

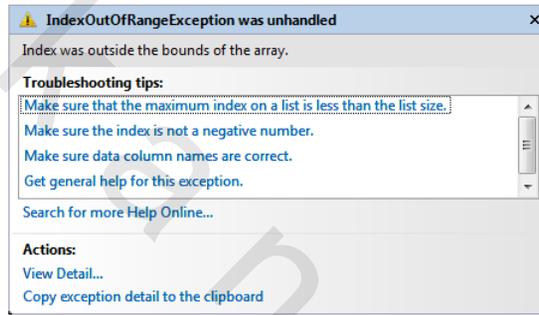
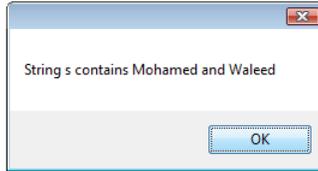
لتوضيح طريقة عمل العبارة **Try-Finally**، انظر معي إلى الكود التالي:

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 يلي:

- في السطر رقم ١ تم تعريف المصفوفة **S** التي تحتوى على عنصرين فقط من النوع **string** وهما **S[0]="Mohamed"** و **S[1]="Waleed"**.
- تحتوى السطور ٢ و ٣ على عبارة **Try** وبها الكود المطلوب اختياره وهو تخصيص قيمة للعنصر الثالث بالمصفوفة. ولأن المصفوفة لا تحتوى أساساً إلا على عنصرين فقط، تقوم العبارة **Try** بطرح الاستثناء **IndexOutOfRangeException** والذي لا يتم معالجته لعدم وجود عبارة **Catch** داخل العبارة **Try**.

- تحتوي السطور من ١١ إلى ١٥ على العبارة **Finally** التي يتم تنفيذها حتى ولو كان هناك استثناء (خطأ) داخل عبارة **Try**. وبالتالي 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 في حالة وجود استثناء داخل عبارة **Try**

العبارة **Try-Catch-Finally**

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

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

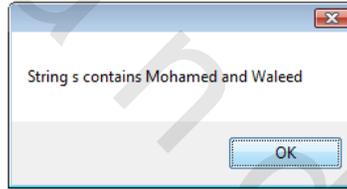
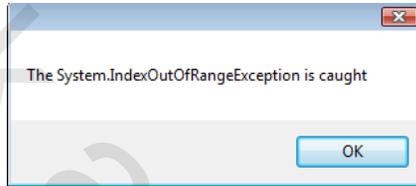


للتعرف عن قرب على طريقة عمل العبارة **Try-Catch-Finally**، قم بتعديل الكود السابق كما يلي:

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 التي لا تحتوي على معاملات وعبارة `Catch` التي تحتوي على المعامل `System.Exception` في 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 التي من الممكن أن ينتج عن تنفيذها خطأ أو أكثر داخل عبارة Try، حيث تقوم هذه العبارة تلقائياً بطرح رسالة الخطأ المناسبة التي 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 والتي يتم استخدامها لمعالجة هذا الخطأ أو الاستثناء.

يطلق على المسار الذي 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 التي 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 ينتهي بث الاستثناء. أما إذا لم يجد النظام عبارة Catch المناسبة، فيتم البحث عن العبارة Finally التي يتم بداخلها معالجة الاستثناء. فإذا قامت عبارة Finally هي الأخرى بإصدار استثناء آخر، يتم قطع معالجة الاستثناء الأول والبدء في 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 التي تليها إلى أن نصل إلى العبارة الخارجية.

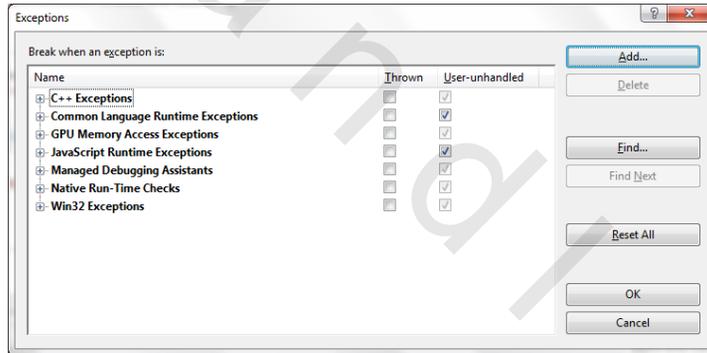


توزيع سلوك الاستثناءات

عند تعيين كود معالجة الاستثناء من خلال عبارات الاستثناء التي 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 في إيقاف تنفيذ البرنامج دون تمكينك من معالجة هذا الخطأ حتى لا يتكرر

مرة ثانية. لذلك، يتيح لك Visual Studio 2012 التحكم في معالجة الاستثناءات وتوفير سلوكها باستخدام المربع الحوارى Exceptions الذى يمكنك من خلاله تحديد ما إذا كان Visual Studio يستمر في 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:

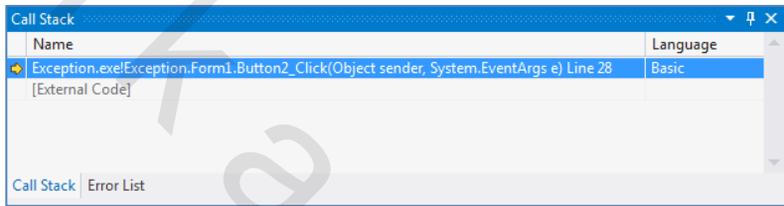
1. افتح قائمة Debug من شريط القوائم ثم اختر Exceptions من القائمة المنسدلة (أو اضغط الاختصار Ctrl+Alt+E من لوحة المفاتيح)، يظهر المربع الحوارى Exceptions (انظر شكل ١٥-٦).



2. شكل ١٥-٦ توفيق سلوك الاستثناءات من خلال المربع الحوارى Exceptions
2. يتم ترتيب الاستثناءات فى صورة هرمية، وبالتالي يمكنك تحديد استثناء معين داخل المجموعة أو مجموعة كاملة من الاستثناءات. فقط قم باختيار الاستثناء المطلوب.
3. 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 التي 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 في طور التوقف، افتح قائمة Debug من شريط القوائم ثم اختر Call Stack و Windows من القوائم المنسدلة (انظر شكل ٧-١٥).



شكل ٧-١٥ 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 وهي تصنيفات الاستثناء التي يقوم المستخدم بتعريفها لضمان إصدار البرنامج رسائل لأخطاء معينة حينما يتم إصدار الاستثناءات أثناء تنفيذ البرنامج.

و حقيقةً 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 ينتهي اسم التصنيف بكلمة "Exception"، كما يجذب أيضاً تجميع كل الاستثناءات المعرفة من قِبَل المستخدم داخل مسمى واحد Namespace لتنظيم الجيد لجميع تصنيفات الاستثناءات سواء المعرفة من قِبَل النظام أو المعرفة من قِبَل المستخدم.

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

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

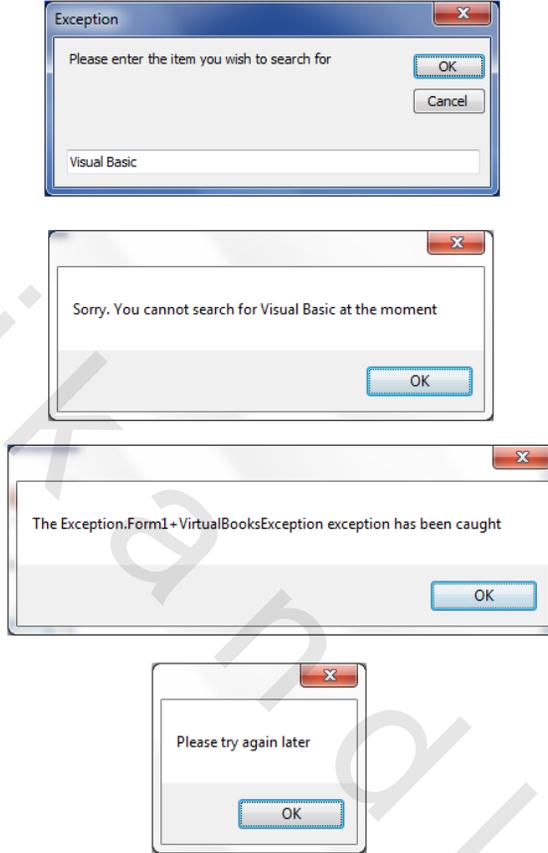
وعن هذا الكود، نوضح ما يلي:

- في 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 معاملاً واحداً من النوع String ويقوم بطباعة رسالة على الشاشة فحوها عدم القدرة على

البحث عن العنصر في الوقت الحالي ثم طرح استثناء من النوع **VirtualBooksException** الذي قمنا بتعريفه في بداية الكود.

- تحتوى السطور من ١١ إلى ١٥ على كود العبارة **try** المستخدمة لاختبار تنفيذ الكود ويتم فيها استقبال عنصر من المستخدم وتمريره إلى الإجراء **Search()** الذي يقوم بدوره بإظهار الرسالة على الشاشة وطرح الاستثناء.
- تحتوى السطور ١٤ و ١٥ على كود العبارة **catch** التي يتم تنفيذها في حالة طرح العبارة **try** للاستثناء **VirtualBooksException** حيث يتم إظهار رسالة بالاستثناء على الشاشة.
- تحتوى السطور ١٦ و ١٧ على كود العبارة **finally** التي تقوم بإظهار رسالة على الشاشة.

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



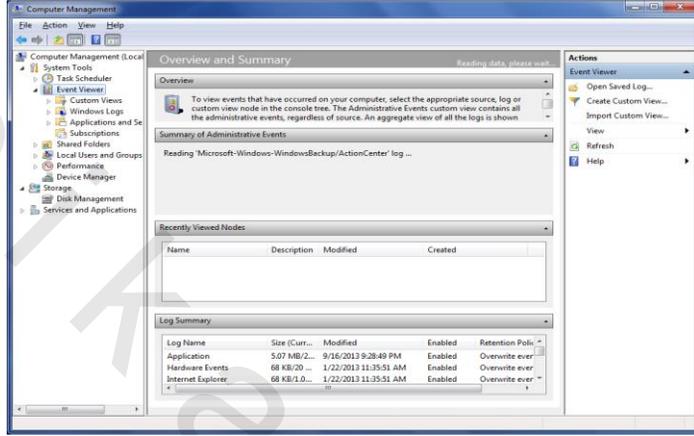
شكل ١٥-٨ استخدام الاستثناءات المعرفة من قِبل المستخدم.

استخدام سجل الأحداث

أثناء العمل داخل بيئة تطوير **Visual Studio 2012**، يمكنك استخدام نافذة **Call Stack** للبحث عن العنصر الذي تسبب في ظهور الاستثناء. أما أثناء التشغيل، فيمكنك استخدام سجل أحداث **Windows** لما يحتويه من ميزات عديدة يمكن إجمال أهمها فيما يلي:

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

- اعتماد مستوى الشبكات ومستخدماً أنظمة التشغيل على برنامج عارض الأحداث المبني داخل Windows (انظر شكل ١٥-٩).



شكل ١٥-٩ التحكم في 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 التفاصيل الفنية في التعامل مع سجل الأحداث، دعنا نقوم بإلقاء نظرة خاطفة على المصطلحات المستخدمة مع سجلات الأحداث، حيث يوجد ثلاثة مصطلحات أساسية يمكن بيانها فيما يلي:
- **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 التي يتم استخدامها لتوضيح نوع الخطأ.
- **سجل الأحداث Event Log** ويحتوى على سجل من الأحداث المرتبطة. فهناك سجل الأحداث **Security** على سبيل المثال والذي يحتوى على الأحداث المرتبطة

بأمان النظام. ويحتوى نظام التشغيل Windows تلقائياً على ثلاثة سجلات هي Application و System و Security إلا أنك تستطيع إضافة المزيد من السجلات بسهولة تامة من خلال عارض الأحداث Event Viewer.

- مصدر الحدث Event Source ويقوم بتعريف الكائن المتسبب في الحدث، والذي قد يكون اسم التطبيق على سبيل المثال. يمكنك التعامل مع سجلات الأحداث داخل Windows من خلال المسمى System.Diagnostics. سنقوم فيما يلي 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 في هذه الحالة هو السجل Application، إلا أنك تستطيع تعيين سجل أحداث مخصص. لربط مصدر الحدث بسجل أحداث مخصص، يمكنك استخدام الوظيفة CreateEventSource ومن ثم يقوم نظام التشغيل بتذكر هذا السجل من هذه اللحظة. ويمكنك بعد ذلك كتابة المعلومات إلى سجل الأحداث من خلال الوظيفة WriteEntry() كما في 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 والثاني 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 الحدث **EventLogEntryType**. وبالتالي 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
AddHandler evtTemp.EntryWritten, AddressOf CheckEvent
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
```

