Introduction

Definition: "Exception is a runtime error which arises because of abnormal conditions in a code sequence". In C# Exception is a class in the system namespace. An object of an exception is that describe the exceptional conditions occur in a code. That means, we are catching an exception, creating an object of it, and then throwing it. C# supports exceptions in a very much the same way as Java and C++.

Before going into detail, I must say the usefulness of knowing and performing exception handling:

- They cannot be ignored, as if calling code does not handle the error, it causes program termination.
- They do not need to be handled at the point where error took place. This makes them very suitable for library or system code, which can signal an error and leave us to handle it
- They can be used when passing back a return value cannot be used.

Try... Catch

Exceptions are handled by using try...catch statements. Code which may give rise to exceptions is enclosed in a try block, which is followed by one or more catch blocks. Well if we don't write like such we get errors like as follows:

```csharp
class A {
    static void Main() {
        try {
        } catch { }
    }
}
TEMP.cs(3,5): error CS1003: Syntax error, 'try' expected

class A {
    static void Main() {
        finally {
        }
    }
}
TEMP.cs(3,5): error CS1003: Syntax error, 'try' expected

class A {
    static void Main() {
        try {
        }
    }
}
TEMP.cs(6,3): error CS1524: Expected catch or finally
```

The try block contains the code segment expected to raise an exception. This block is executed until an exception is thrown. The catch block contains the exception handler. This block catches the exception and executes the code written in the block. If we do not know what kind of exception is going to be thrown, we can simply omit the type of exception. We can collect it in an Exception object as shown in the following program:

```csharp
int a, b = 0;
Console.WriteLine("My program starts");
try {
    // Code that may throw an exception
} catch (Exception exception) {
    // Handle the exception
    Console.WriteLine(exception.Message);
}
```
The output of the program is:

My program starts
System.DivideByZeroException: Attempted to divide by zero. at ConsoleApplication4.Class1.Main(String[] args) in d:\dont delete\c#(c sharp)\swapna\programs\consoleapplication4\consoleapplication4\class1.cs:line 51
Remaining program

The exception 'Divide by zero' was caught, but the execution of the program did not stop.

**Exception Types**

There are a number of exception classes provided by C#, all of which inherit from the System.Exception class. Following are some common exception classes.

<table>
<thead>
<tr>
<th>Exception Class</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>SystemException</td>
<td>A failed run-time check; used as a base class for other.</td>
</tr>
<tr>
<td>AccessException</td>
<td>Failure to access a type member, such as a method or field.</td>
</tr>
<tr>
<td>ArgumentException</td>
<td>An argument to a method was invalid.</td>
</tr>
<tr>
<td>ArgumentNullException</td>
<td>A null argument was passed to a method that doesn’t accept it.</td>
</tr>
<tr>
<td>ArgumentOutOfRangeException</td>
<td>Argument value is out of range.</td>
</tr>
<tr>
<td>ArithmeticException</td>
<td>Arithmetic overflow - or underflow has occurred.</td>
</tr>
<tr>
<td>ArrayTypeMismatchException</td>
<td>Attempt to store the wrong type of object in an array.</td>
</tr>
<tr>
<td>BadImageFormatException</td>
<td>Image is in the wrong format.</td>
</tr>
<tr>
<td>CoreException</td>
<td>Base class for exceptions thrown by the runtime.</td>
</tr>
<tr>
<td>DivideByZeroException</td>
<td>An attempt was made to divide by zero.</td>
</tr>
<tr>
<td>FormatException</td>
<td>The format of an argument is wrong.</td>
</tr>
<tr>
<td>IndexOutOfRangeException</td>
<td>An array index is out of bounds.</td>
</tr>
<tr>
<td>InvalidCastExpression</td>
<td>An attempt was made to cast to an invalid class.</td>
</tr>
<tr>
<td>InvalidOperationException</td>
<td>A method was called at an invalid time.</td>
</tr>
<tr>
<td>MissingMemberException</td>
<td>An invalid version of a DLL was accessed.</td>
</tr>
<tr>
<td>NotFiniteNumberException</td>
<td>A number is not valid.</td>
</tr>
<tr>
<td>NotSupportedException</td>
<td>Indicates that a method is not implemented by a class.</td>
</tr>
<tr>
<td>NullReferenceException</td>
<td>Attempt to use an unassigned reference.</td>
</tr>
<tr>
<td>OutOfMemoryException</td>
<td>Not enough memory to continue execution.</td>
</tr>
<tr>
<td>StackOverflowException</td>
<td>A stack has overflowed.</td>
</tr>
</tbody>
</table>

The finally block is used to do all the cleanup code. It does not support the error message, but all the code contained in the finally block is executed after the exception is raised. We can use this block along with try-catch and only with catch too.

The finally block is executed even if the error is raised. Control is always passed to the finally block regardless of how the try blocks exits.

This is shown in the following example:

```csharp
int a, b = 0;
Console.WriteLine("My program starts");
try
{
    a = 10 / b;
}
catch (InvalidOperationException e)
{
    Console.WriteLine(e);
}
Console.WriteLine("Remaining program");
```
My program starts
System.DivideByZeroException: Attempted to divide by zero.
at ConsoleApplication4.Class1.Main(String[] args) in d:\programs\consoleapplication4\class1.cs:line 51
finally
Remaining program

But then what's the difference? We could have written

Console.WriteLine("finally");

after the catch block, and not write the finally block at all. Writing finally did not make much of a difference. Anyway the code written after catch gets executed. The answer to this is not clear in this program. It will be clear when we see the try-finally and the throw statement.

**Throwing an error**

Consider the following code snippet

```csharp
int a, b = 0;
Console.WriteLine("My program starts");
try
{
a = 10 / b;
}
finally
{
    Console.WriteLine("finally");
}
Console.WriteLine("Remaining program");
```

Here the output is

My program starts
Exception occurred: System.DivideByZeroException: Attempted to divide by zero.
at ConsoleApplication4.Class1.Main(String[] args) in d:\programs\consoleapplication4\class1.cs:line 51
finally

Note that "Remaining program" is not printed out. Only "finally" is printed which is written in the finally block.

The throw statement throws an exception. A throw statement with an expression throws the exception produced by evaluating the expression. A throw statement with no expression is used in the catch block. It re-throws the exception that is currently being handled by the catch block.

Consider the following program:

```csharp
int a, b = 0;
Console.WriteLine("My program starts");
try
{
a = 10 / b;
}
catch (Exception e)
My program starts

Exception occurred: System.DivideByZeroException: Attempted to divide by zero.

This shows that the exception is re-thrown. Whatever is written in finally is executed and the program terminates. Note again that "Remaining program" is not printed.

I am 22 male. BS(Computer Science), MCSE, Brainbench certified. Member of .NET Open source, Mono Project. E-author on various websites, including www.dotnetextreme.com, www.csharphelp.com and so on. Currently working as a asst. network engineer, and as a freelance programmer, programming in Microsoft.NET platform. Love to use linux box with my NT-Machine.

Related Content

- Exceptions and Performance in .NET