Chapter 31: Error Handling

Section 31.1: Try...Catch...Finally Statement

Structure:

```
Try
    'Your program will try to run the code in this block.
    'If any exceptions are thrown, the code in the Catch Block will be executed,
    'without executing the lines after the one which caused the exception.
Catch ex As System.IO.IOException
    'If an exception occurs when processing the Try block, each Catch statement
    'is examined in textual order to determine which handles the exception.
    'For example, this Catch block handles an IOException.
Catch ex As Exception
    'This catch block handles all Exception types.
    'Details of the exception, in this case, are in the "ex" variable.
    'You can show the error in a MessageBox with the below line.
    MessageBox.Show(ex.Message)
Finally
    'A finally block is always executed, regardless of if an Exception occurred.
End Try
```

Example Code:

```
Try
    Dim obj = Nothing
    Dim prop = obj.Name 'This line will throw a NullReferenceException

    Console.WriteLine("Test.") ' This line will NOT be executed
Catch ex As System.IO.IOException
    ' Code that reacts to IOException.
Catch ex As NullReferenceException
    ' Code that reacts to a NullReferenceException
    Console.WriteLine("NullReferenceException: " & ex.Message)
    Console.WriteLine("Stack Trace: " & ex.StackTrace)
Catch ex As Exception
    ' Code that reacts to any other exception.
Finally
    ' This will always be run, regardless of if an exception is thrown.
    Console.WriteLine("Completed")
End Try
```

Section 31.2: Creating custom exception and throwing

You can create a custom exception and throw them during the execution of your function. As a general practice you should only throw an exception when your function could not achieve its defined functionality.

```
Private Function OpenDatabase(Byval Server as String, Byval User as String, Byval Pwd as String)
    if Server.trim="" then
        Throw new Exception("Server Name cannot be blank")
    elseif User.trim ="" then
        Throw new Exception("User name cannot be blank")
    elseif Pwd.trim="" then
        Throw new Exception("Password cannot be blank")
    endif

    'Here add codes for connecting to the server
```

```
End function
```

Section 31.3: Try Catch in Database Operation

You can use Try..Catch to rollback database operation by placing the rollback statement at the Catch Segment.

```
    Try
        'Do the database operation...
        xCmd.CommandText = "INSERT into ...."
        xCmd.ExecuteNonQuery()

        objTrans.Commit()
        conn.Close()
    Catch ex As Exception
        'Rollback action when something goes off
        objTrans.Rollback()
        conn.Close()
    End Try
```

Section 31.4: The Un-catchable Exception

Although `Catch ex As Exception` claims that it can handle all exceptions - there are one exception (no pun intended).

```
Imports System
Static Sub StackOverflow() ' Again no pun intended
    StackOverflow()
End Sub
Static Sub Main()
    Try
        StackOverflow()
    Catch ex As Exception
        Console.WriteLine("Exception caught!")
    Finally
        Console.WriteLine("Finally block")
    End Try
End Sub
```

Oops... There is an un-caught `System.StackOverflowException` while the console didn't even print out anything! According to MSDN,

> Starting with the .NET Framework 2.0, you can’t catch a StackOverflowException object with a try/catch block, and the corresponding process is terminated by default. Consequently, you should write your code to detect and prevent a stack overflow.

So, `System.StackOverflowException` is un-catchable. Beware of that!

Section 31.5: Critical Exceptions

Generally most of the exceptions are not that critical, but there are some really serious exceptions that you might not be capable to handle, such as the famous `System.StackOverflowException`. However, there are others that might get hidden by `Catch ex As Exception`, such as `System.OutOfMemoryException`, `System.BadImageFormatException` and `System.InvalidProgramException`. It is a good programming practice to leave these out if you cannot correctly handle them. To filter out these exceptions, we need a helper method:

```vb
Public Shared Function IsCritical(ex As Exception) As Boolean
    Return TypeOf ex Is OutOfMemoryException OrElse
           TypeOf ex Is AppDomainUnloadedException OrElse
           TypeOf ex Is AccessViolationException OrElse
           TypeOf ex Is BadImageFormatException OrElse
           TypeOf ex Is CannotUnloadAppDomainException OrElse
           TypeOf ex Is ExecutionEngineException OrElse ' Obsolete one, but better to include
           TypeOf ex Is InvalidProgramException OrElse
           TypeOf ex Is System.Threading.ThreadAbortException
End Function
```

Usage:

```vb
Try
    SomeMethod()
Catch ex As Exception When Not IsCritical(ex)
    Console.WriteLine("Exception caught: " & ex.Message)
End Try
```