**http://curriculum.kcdistancelearning.com/courses/PROG1s-HS-A08/s/images/icon_homework_sm.gif Assignment: User-Defined Exceptions**

**10 points**

**Now it's time to apply the information you have learned in this unit so far. Using the information and examples from the previous screens and the Web, research and find at least 5 examples of user-defined exceptions. Describe each exception in detail and how it might be used.  
You will be graded according to the Writing Rubric.  
Remember to submit your assignment.**

To handle unusual events, programmers often create their own user-defined exceptions. These exceptions are similar to pre-defined exceptions, which are already defined classes that inherit features from the System.Exceptions class. Similar to the pre-defined exceptions, the user-defined exceptions follow an organization pattern to monitor, catch, and express that an error has occurred. The Try block monitors an exception. Code is pasted here in the try block so the program checks whether there is an error or not in the code. If there is an error, the Throw statement throws the error to the Catch block, which catches the error and delivers a message that it occurred. Finally, the Finally block says that the error-finding process is finished. This block is also used to clean up unwanted objects. For example, if the try block establishes a connection to a database with a connection object, the finally block must cut this connection after the information is gathered.

One example of a user defined exception is below. This code from tutorialspoint.com is made to throw an exception when the temperature object is zero. The debugging tool goes into the Sub Main() and once it reaches the temp.showTemp() code, it goes into the Class Temperature and into this method to execute the If-Then statement. If the temperature is zero, an exception gets thrown, but if not, the temperature is printed. If temperature is zero, the debugging tool goes into the Class TempIsZeroException and uses the message variable there to locate the “Zero Temperature Found” message from the Throw statement. After it reaches this message, the debugging tool goes back into the Sub Main() to Catch the TempIsZeroException and print the Method message that carries the original message from the Throw statement using the e variable, that is an instance of the TempIsZeroException Class. Although there is no Finally block, the message is printed successfully to the screen.

Module exceptionProg

Public Class TempIsZeroException : Inherits ApplicationException

Public Sub New(ByVal message As String)

MyBase.New(message)

End Sub

End Class

Public Class Temperature

Dim temperature As Integer = 0

Sub showTemp()

If (temperature = 0) Then

Throw (New TempIsZeroException("Zero Temperature found"))

Else

Console.WriteLine("Temperature: {0}", temperature)

End If

End Sub

End Class

Sub Main()

Dim temp As Temperature = New Temperature()

Try

temp.showTemp()

Catch e As TempIsZeroException

Console.WriteLine("TempIsZeroException: {0}", e.Message)

End Try

Console.ReadKey()

End Sub

End Module

Another example’s goal is to print that there are not enough funds if the user enters a negative bank account balance. The debugging tool enters the Sub Main() and gets user input in a variable x, and then goes into the Try block to access the withdraw1 function, at the same time passing the inputted variable, x. In the Class Bank, containing the Sub withdraw1, if the inputted x value is negative, then the Insufficientfund() user-defined exception is thrown. In the Insufficientfund class, the MyBase keyword is used to represent errors that occur when the program is executed. With this keyword, an error message is passed, which the debugging tool then can print after entering the Catch block.

Module Module1

'User define Exception class.

Public Class Insuficentfund : Inherits Exception

Sub New()

MyBase.New("error occurred due to insuficent funds")

End Sub

Sub New(ByVal msg As String)

MyBase.New(msg)

End Sub

Sub New(ByVal msg As String, ByVal inner As Exception)

MyBase.New(msg, inner)

End Sub

End Class

Public Class Bank

Public Sub withdraw1(ByVal amount As Integer)

If (amount < 0) Then

' Use the user define Exception Class

Throw New Insuficentfund()

Else

Console.WriteLine("Welcome to Bank Account")

Console.WriteLine("Your bank Account Balance is = " & amount)

End If

End Sub

End Class

Sub Main()

Dim obj As New Bank

Console.WriteLine("if you enter =Ve value exception is Occurr")

Dim x As Integer

x = Int32.Parse(Console.ReadLine())

Try

obj.withdraw1(x)

'independent catch block for the user define Exception

Catch ex As Insuficentfund

Console.WriteLine(ex.Message)

Console.WriteLine("\*\*\*Exception in Bank Account Blance\*\*\*")

End Try

Console.ReadKey()

End Sub

End Module

The third example shows the basic steps for how a user-defined exception is thrown. The user has defined an ApplicationException somewhere else that is now being thrown by the Throw statement. The error is caught by the Catch block in the variable e, and the message in the Finally block is then executed.

Module exceptionProg

Sub Main()

Try

Throw New ApplicationException("A custom exception is being thrown here...")

Catch e As Exception

Console.WriteLine(e.Message)

Finally

Console.WriteLine("Now inside the Finally Block")

End Try

Console.ReadKey()

End Sub

End Module

Another example’s goal is to throw an exception when a string cannot be successfully parsed into an integer. In the Try block the code being monitored for an exception is the code that tries to parse a string to an integer. However, since the string is “x” not a number declared as a string, a FormatException occurs and is caught by the Catch block in the variable f. In this block, the error message is delivered. The Finally block implies the process is finished.

Module Module1

Sub Main()

' Declare the string value

Dim strValue As String = "x"

Try

'Try to parse string to get integer

Dim intValue As Integer = Integer.Parse(strValue)

System.Console.WriteLine("Parsing...")

'Throw New Exception("The string value could not be parsed")

Catch f As FormatException

System.Console.WriteLine("Format Exception! The string value could not be parsed:" & strValue)

Finally

' Say the process is finished

System.Console.WriteLine("Press ENTER to continue...")

End Try

' Pause output

Console.ReadKey()

End Sub

End Module

The last example’s goal is to throw an exception when a key is not found in a hashtable. After the elements are added to the hashtable, the debugging tool goes into the Try block to access the showKey() method from the Class Key. If hashtable contain key5, then the program will print the key is present. But since there are only four elements paired with four keys, the NotKeyException is thrown. The Throw statement contains a message that is passed via the message variable in the Class NotKeyException. Then, the error is caught in the Catch block as a NotKeyException. Following that, the Finally block prints the program is over.

Module Module1

Public Class NotKeyException : Inherits Exception

Public Sub New(ByVal message As String)

MyBase.New(message)

End Sub

End Class

Public Class Key

Sub showKey(ByVal o As Hashtable)

If o.ContainsKey("key5") Then

Console.WriteLine("The key is in the hashtable.")

Else

Throw (New NotKeyException("EXCEPTION! WARNING! The key was not found"))

End If

End Sub

End Class

Sub Main()

Dim o As New Hashtable()

Dim key As Key = New Key()

o.Add("key1", "cat")

o.Add("key2", "dog")

o.Add("key3", "fish")

o.Add("key4", "hamster")

Try

key.showKey(o)

Catch k As NotKeyException

System.Console.WriteLine(k.Message)

Finally

System.Console.WriteLine()

System.Console.WriteLine("Program is over")

End Try

Console.ReadKey()

End Sub

End Module

In conclusion, these user-defined examples show how errors can be prepared for in advance with the Try, Throw, Catch, and Finally blocks that monitor, throw, catch, and declare errors, respectively. Errors can occur in different situations, such as when a variable is not allowed to be a certain value, parsing from a String to an Integer is not successful, or when an invalid key is called for in a hashtable.

Sources:   
<http://www.dotnetperls.com/integer-parse-vbnet>  
<http://www.tutorialspoint.com/vb.net/vb.net_exception_handling.htm>  
<http://www.dotnetheaven.com/article/user-define-exception-in-vb.net>