VB.NET - EXCEPTION HANDLING

An exception is a problem that arises during the execution of a program. An exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

Exceptions provide a way to transfer control from one part of a program to another. VB.Net exception handling is built upon four keywords: **Try**, **Catch**, **Finally** and **Throw**.

- **Try**: A Try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more Catch blocks.
- **Catch**: A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The Catch keyword indicates the catching of an exception.
- **Finally**: The Finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must be closed whether an exception is raised or not.
- **Throw**: A program throws an exception when a problem shows up. This is done using a Throw keyword.

Syntax

Assuming a block will raise an exception, a method catches an exception using a combination of the Try and Catch keywords. A Try/Catch block is placed around the code that might generate an exception. Code within a Try/Catch block is referred to as protected code, and the syntax for using Try/Catch looks like the following:

```
Try
    [ tryStatements ]
    [ Exit Try ]
[ Catch [ exception [ As type ] ] [ When expression ]
    [ catchStatements ]
    [ Exit Try ] ]
[ Catch ... ]
[ Finally
    [ finallyStatements ] ]
End Try
```

You can list down multiple catch statements to catch different type of exceptions in case your try block raises more than one exception in different situations.

Exception Classes in .Net Framework

In the .Net Framework, exceptions are represented by classes. The exception classes in .Net Framework are mainly directly or indirectly derived from the **System.Exception** class. Some of the exception classes derived from the System.Exception class are the **System.ApplicationException** and **System.SystemException** classes.

The **System.ApplicationException** class supports exceptions generated by application programs. So the exceptions defined by the programmers should derive from this class.

The **System.SystemException** class is the base class for all predefined system exception.

The following table provides some of the predefined exception classes derived from the Sytem.SystemException class:

Exception Class	Description
System.IO.IOException	Handles I/O errors.

System.IndexOutOfRangeException	Handles errors generated when a method refers to an array index out of range.
System.ArrayTypeMismatchException	Handles errors generated when type is mismatched with the array type.
System.NullReferenceException	Handles errors generated from deferencing a null object.
System.DivideByZeroException	Handles errors generated from dividing a dividend with zero.
System.InvalidCastException	Handles errors generated during typecasting.
System.OutOfMemoryException	Handles errors generated from insufficient free memory.
System.StackOverflowException	Handles errors generated from stack overflow.

Handling Exceptions

VB.Net provides a structured solution to the exception handling problems in the form of try and catch blocks. Using these blocks the core program statements are separated from the error-handling statements.

These error handling blocks are implemented using the **Try**, **Catch** and **Finally** keywords. Following is an example of throwing an exception when dividing by zero condition occurs:

```
Module exceptionProg
   Sub division(ByVal num1 As Integer, ByVal num2 As Integer)
      Dim result As Integer
      Try
          result = num1 \ num2
      Catch e As DivideByZeroException
          Console.WriteLine("Exception caught: {0}", e)
      Finally
          Console.WriteLine("Result: {0}", result)
      End Try
   End Sub
   Sub Main()
      division(25, 0)
      Console.ReadKey()
  End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
Exception caught: System.DivideByZeroException: Attempted to divide by zero.
at ...
Result: 0
```

Creating User-Defined Exceptions

You can also define your own exception. User-defined exception classes are derived from the **ApplicationException** class. The following example demonstrates this:

```
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
```

When the above code is compiled and executed, it produces the following result:

```
TempIsZeroException: Zero Temperature found
```

Throwing Objects

You can throw an object if it is either directly or indirectly derived from the System.Exception class.

You can use a throw statement in the catch block to throw the present object as:

```
Throw [ expression ]
```

The following program demonstrates this:

```
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
```

When the above code is compiled and executed, it produces the following result:

```
A custom exception is being thrown here...
Now inside the Finally Block
```