Exception Handling

The Exception Hierarchy

Exception is represented by classes. All the exceptions are subclasses in the built-in exception class named `Exception`, wherein it is a part of namespace `System`.

Two (2) Types of Exceptions

- **ApplicationException** – These exceptions are user program-generated.
- **SystemException** – These exceptions are generated by Common Language Runtime (CLR).

Here are the few exceptions which are commonly used:

- **`System.Exception`** – This is at the top of the standards' exceptions hierarchy. The runtime system in C# generates all the exceptions.
- **`System.ArithmeticException`** – Errors in arithmetic or conversion operation will be thrown in this exception.
- **`System.OverflowException`** – When an overflow occurs in a `checked` operation, it will be thrown in `OverflowException`.
- **`System.ArgumentException`** – Any invalid argument in a method will be thrown in this exception.
- **`System.ArgumentNullException`** – If there is an unacceptable argument passed to a method, it will be thrown in `ArgumentNullException`.
- **`System.IndexOutOfRangeException`** – Throw in this exception when attempting to index an array through an index that is either less than zero or greater than the maximum length of index.
- **`System.OutOfMemoryException`** – If the available memory becomes too low to accommodate a memory allocation request, it will be thrown in `OutOfMemoryException`.
- **`System.StackOverflowException`** – The exception `StackOverflowException` is called when the execution stack is exhausted by having too many pending method calls.
- **`System.FormatException`** – This exception checks the format of the string or argument if it is invalid.

The *try-catch* Block

Handling exceptions in C# uses four (4) keywords:

- **`try`** – This keyword is used to check for the occurrence of any exceptions enclosed to it.
- **`catch`** – This keyword catches the exception that is thrown on the occurrence of exception in a `try` block.
- **`throw`** – It is used to throw an exception manually.
- **`finally`** – This keyword executes a given statement even if the exception is thrown or not thrown. This block cannot transfer control by using `break`, `continue`, `return`, or `goto`.

```
try{
   // code to be check for exceptions
}catch(Type_of_Exception var_name){
   // exception are handled here.
}catch(Type_of_Exception var_name){
   // exception are handled here
}finally{
   // clean up any codes that are allocated in try block
}
```

Throwing an Exception

To catch exceptions, use the `try-catch` block to check the errors in the code and generate in the runtime system automatically. Manually throwing using the **`throw`** keyword can also be used. The syntax is

```
throw new exception_Object;
```

The **`exception_Object`** is an instance of a class derived from the `Exception` class. The **`new`** operator is used to create a new object.

```
private int num1, num2;
private void btnCompute_Click(object sender, EventArgs e){
   num1 = Convert.ToInt32(txtNum1.Text);
   num2 = Convert.ToInt32(txtNum2.Text);
   if(num1 == 0 || num2 == 0)
      throw new DivideByZeroException("Invalid Input");
   else
      MessageBox.Show("Total: " + GetQuotient(num1, num2));
}
public int GetQuotient(int x, int y){
   return x/y;
}
```

Listing 1. Using throw without the try-catch block

Listing 1 shows how **DivideByZeroException** is thrown manually. The exception was caught in an **if** condition that validates the input using the **new** operator. If the condition is equal to 0, the thrown exception will be called. The **throw** keyword can be used outside of the **try-catch** block, but without the **try-catch** block, there will be an interruption in the process. This will show an error message to the console and the application will be closed. If throwing an exception, a **catch** block is needed.

```
private void btnCompute_Click(object sender, EventArgs e)
{
   try{
      num1 = Convert.ToInt32(txtNum1.Text);
      num2 = Convert.ToInt32(txtNum2.Text);
      if(num1 == 0 || num2 ==0)
         throw new DivideByZeroException();
      else
         MessageBox.Show("Total: " + GetQuotient(num1, num2));
   }
   catch(DivideByZeroException dze)
   {
      Console.WriteLine("Divide By Zero Message: " + dze.Message);
   }
   catch(FormatException fe)
   {
      Console.WriteLine("Format Exception Message: " + fe.Message);
   }
}
public int GetQuotient(int x, int y){
   return x/y;
}
```

Listing 2. Using throw with try-catch block

Listing 2 contains two (2) **catch** blocks to check the format and check if the entered number is zero. In contrast, *Listing 1* only shows an exception that will be thrown and will not catch when the exception occurs, while *Listing 2* demonstrates how a **catch** block catches a manually thrown exception. The **DividebyZeroException** is thrown by using **throw** keyword and caught in a **catch** block and display the message.

Creating Own Exception

Creating one's own exception is possible in .NET Framework because it provides a facility to create custom exceptions. In a customized exception, it requires or inherits those exceptions in **System.Exception** class or one of its standard derived classes. The simplest form for a custom exception class is

```
public class CustomizeException: Exception{
   //code here
}
```

Once the custom exception class is created and is derived from the Exception class, add a constructor using the following format:

```
public class CustomizeException: Exception{
    public CustomException(string str): base(str){
    }
}
```

After that, the custom exception acts like other standard exceptions. One can pass a string that describes the cause of error.

Listing 3 demonstrates the creation of the custom exception class `InvalidUserInputException` that throws manually and catches the exception using the `catch` block.

```
///InvalidUserInputException Class
public class InvalidUserInputException : Exception{
    public InvalidUserInputException(string age): base(age){
    }
}

///CustomExcep Class with Form
public partial class CustomExcep : Form
{
    public CustomExcep()
    {
        InitializeComponent();
    }
    private void btnCheck_Click(object sender, EventArgs e)
    {
        try{
            checkAge(Int32.Parse(txtStudAge.Text));
        }
        catch(InvalidUserInputException ex)
        {
            Console.WriteLine("Message: " + ex.Message);
        }
    }
    public void checkAge(int age){
        if(age == 0 || age < 18)
        {
            throw new InvalidUserInputException("Not in legal age!");
        }
        else
        {
            MessageBox.Show("Legal Age!");
        }
     }
}
```

Listing 3. Custom Exception

REFERENCES:

Deitel, P. & Deitel, H. (2015). *Visual C# 2012 how to program* (5th ed.). USA: Pearson Education, Inc.

Gaddis, T. (2016). *Starting out with visual C#* (4th ed.). USA: Pearson Education, Inc.

Harwani, B. (2015). *Learning object-oriented programming in C# 5.0.* USA: Cengage Learning PTR.

Miles, R. (2016). *Begin to code with C#.* Redmond Washington: Microsoft Press.

Doyle, B. (2015). *C# programming: from problem analysis to program design* (5th ed.). Boston, MA: Cengage Learning.