Exception Handling

Exceptions in any language are classified into **Compile Time Exceptions, Runtime Exceptions** and **Logical Exceptions**.

Compile Time Exceptions

Compile time exceptions are the exceptions that are found during compilation. In general these exceptions are syntactical errors in the program. It is not possible to execute the program without rectifying the compile time exceptions. Hence exception handling is not related to compile time exceptions.

Runtime Exceptions

Runtime exceptions are the exceptions that occur at runtime i.e. while executing the program. When a runtime exception occurs then the program will be terminated abnormally without giving a proper message to the user regarding the exception. Exception handling is used to handle runtime exceptions and give proper message to the user regarding exception and avoid abnormal termination of the program. Exception handling in C# was based on the keywords **try , catch, finally** and **throw**.

Ramesh Bollepalli

Try block is used to write the statements that may cause an exception at runtime. When an exception was raised in the try block, then control will be taken to **catch** block where we have to write the exception handling code. Once control reaches the **catch** block it is not possible to take it back to **try**. The complete code in the **try** block will be executed only when there is no exception raised and the code in **catch** block is executed only when an exception was raised. But you may have the code to be executed both when an exception was raised and no exception was raised. **Finally** block is used to write the code that needs to be executed both when an exception was raised and no exception was raised. **Throw** is used to raise user defined exception.

```
Try
{

}
Catch
{

}
Finally
{

}
```

Example : The following example accepts two integers and performs division. If the value given for denominator is zero then an exception will be raised and the program handles this exception with exception handling.

```
namespace ExceptionHandling
{
    class Program
    {
        static void Main(string[] args)
        {
            int A, B, R;
            Console.WriteLine("Enter Two Integers");
            A = int.Parse(Console.ReadLine());
            B = int.Parse(Console.ReadLine());
            try
            {
                R = A / B;
                Console.WriteLine("Ratio Of {0} And {1} Is {2}", A, B, R);
            }
            catch (DivideByZeroException Ex)
            {
                Console.WriteLine("Division With Zero Not Possible");
            }
        }
    }
}
```

Handling Multiple Exceptions

Within a method there is possibility for more than one exception. In this case no need to write separate **try…catch** for each exception and related to a single **try** block you can create any number of **catch** blocks, one for each type of exception.

Example : The following example handles two exceptions dividebyzero exception and overflow exception that will be raised when the value given for variable is out of range for its data type.

```
namespace ExceptionHandling
{
    class MultipleExceptions
    {
        static void Main()
        {
            int A, B, R;
            Console.WriteLine("Enter Two Integers");
            try
            {
                A = int.Parse(Console.ReadLine());
                B = int.Parse(Console.ReadLine());
                R = A / B;
                Console.WriteLine("Ratio Of {0} And {1} Is {2}", A, B, R);
            }
```

Ramesh Bollepalli

```
            catch (DivideByZeroException Ex)
             {
                 Console.WriteLine("Division With Zero Not Possible");
             }
             catch (OverflowException Ex)
             {
             Console.WriteLine("Value Must Be Within The Range Of Integer");
             }
             Console.Read();
         }
     }
}
```

Handling Any Type of Exception

A programmer will write the exception handling code for all the exceptions that are up to his expectation. But there is possibility for an exception that was not up to the expectation of the programmer. Hence exception handling will be complete only when there is a catch block that can catch any type of exception. For this you have to create the catch block by specifying the exception type as **Exception**. Every exception in .net is by default inherited from **Exception** class and hence it can catch any type of exception. This type of **catch** block must be the last **catch** in a series of **catch** blocks.

Example : The following example handles two exceptions dividebyzero exception and overflow exception and can also handle any other type of exception.

```
namespace ExceptionHandling
{
    class AnyException
    {
        static void Main()
        {
            int A, B, R;
            Console.WriteLine("Enter Two Integers");
            try
            {
                A = int.Parse(Console.ReadLine());
                B = int.Parse(Console.ReadLine());
                R = A / B;
                Console.WriteLine("Ratio Of {0} And {1} Is {2}", A, B, R);
            }
            catch (DivideByZeroException Ex)
            {
                Console.WriteLine("Division With Zero Not Possible");
            }
            catch (OverflowException Ex)
            {
            Console.WriteLine("Value Must Be Within The Range Of Integer");
            }
            catch (Exception Ex)
            {
                Console.WriteLine(Ex.Message);
            }
```

Ramesh Bollepalli

```
            Console.Read();
        }
    }
}
```

User Defined Exceptions

There may be a situation where system will not raise an exception for your requirement and you want to raise the exception manual. In this case you have to create a class for your exception and it must be inherited from **Exception** class. To raise the exception, use **throw** keyword.

Example : The following example creates a user defined exception that will be raised when user enters a negative value for either A or B.

Ramesh Bollepalli

```
namespace ExceptionHandling
{
    class MyException : Exception
    {
        public string MyMessage;
        public MyException(string Mes)
        {
            MyMessage = Mes;
        }
    }
    class UserExceptions
    {
        static void Main()
        {
            int A, B, R;
            Console.WriteLine("Enter Two Integers");
            try
            {
                A = int.Parse(Console.ReadLine());
                if(A<0)
                    throw new MyException("-ve Values Are Not Allowed");
                B = int.Parse(Console.ReadLine());
                if(B<0)
                    throw new MyException("-ve Values Are Not Allowed");
                R = A / B;
                Console.WriteLine("Ratio Of {0} And {1} Is {2}", A, B, R);
            }
            catch (DivideByZeroException Ex)
            {
                Console.WriteLine("Division With Zero Not Possible");
            }
            catch (OverflowException Ex)
            {
            Console.WriteLine("Value Must Be Within The Range Of Integer");
            }
            catch(MyException Ex)
            {
                Console.WriteLine(Ex.MyMessage);
            }
            catch (Exception Ex)
```

```
            {
                Console.WriteLine(Ex.Message);
            }
            Console.Read();
        }
    }
}
```

Logical Exceptions

Logical exception is an exception that will not be found during compilation or at runtime and will cause the program output to be wrong is called as logical exception. To find and rectify the logical exceptions tracing and debugging is used. Executing the program step by step and finding logical errors is called as tracing and debugging. To execute the program step by step shortcut is **F11**.

Break Point

When you don't want to execute entire program step by step and you want to execute only a particular method step by step then **break points** are used. When a **break point** is placed in the program then during execution when control reaches the **break point** then program will enter in to debug mode. Shortcut to place or remove a **break point** is **F9**.

Debug Tools

For debugging a .net application, debugging tools are provided and are as follows.

Watch Window : This can be used to add a variable to it and observe how the value of that variable changes while executing the program step by step. VS.net provides four watch windows and shortcuts to open watch windows are

CTRL + ALT + W, 1

CTRL + ALT + W, 2

CTRL + ALT + W, 3

CTRL + ALT + W, 4

For adding a variable to the watch window, select the variable, right click on it and then choose **add watch**. Otherwise select the variable and drag and drop it in watch window. You can also modify the value of a variable in watch window. For this right click on the variable in watch window and choose **edit value**.

Ramesh Bollepalli

Locals Window : This is used to observe how the values of local variables of current method are changed as the program is executed step by step. There is no need to add variables to locals window and all local variables of the current method are automatically added to the locals window. Shortcut to open locals window is **CTRL + ALT + V, L**. Same as in case of watch window in locals window also you can edit the value of a variable.

Immediate Window : This is used to print the values of variable or expressions and even to change the values of variables. "**?**" is used to print the value of a variable or expression within the immediate window. To change the value of a variable, write the assignment statement. Shortcut to open immediate window is **CTRL + ALT + I**.

Ramesh Bollepalli