Unit 1.5
Language Preliminaries & .Net Basic (8-Hrs)

Introduction to Exception Handling

- An exception is an event, which occurs during the execution of a program that interrupts the normal flow of the program's instruction.
- Exception handling is the process of responding to exceptions when a computer program runs.
- An exception occurs when an unexpected event happens that requires special processing.
- When an Exception occurs the normal flow of the program is disrupted and the program terminates abnormally, which is not recommended, therefore, these exceptions **are to be handled using exception handling mechanism**.
- The main purpose of exception handling is to provide a means to detect and report an error to the user.
- An exception can occur for many different reasons. Following are some scenarios where an exception occurs.
  i. A user has entered an invalid data.
  ii. A file that needs to be opened cannot be found.
  iii. A network connection has been lost in the middle of communications
  And many more….

Standard Exception class in C#

- Some of the exception class derived from class **System.Exception class** for handling exception are given below
  1. **DivideByZeroException** i.e. Thrown when division by zero operation is made.
  2. **FormatException** i.e. Thrown when the format of the argument is invalid.
  3. **NullReferenceException i.e.** Thrown when a we attempt to dereference a null object reference.
  4. **StackOverflowException i.e.** Thrown when execution stacks overflow for example: to many nested call.
  5. **NotImplementedException i.e.** Thrown when the requested method or operation is not implemented
  6. **IndexOutOfRangeException i**.e. thrown when an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

Exception handling mechanism in c#

- The steps in exception handling mechanism in c# are
  1. Find the problem. (Hit the exception)
  2. Inform that an error has occurred. (Throw the exception)
  3. Receive the error information (Catch the exception)
  4. Take corrective actions. (Handel the exception)
- Exceptions in c# are handled by using the keywords **try, catch, throw and finally**.
- Try block consists of code that needs to be monitored for exceptions. Try block throw error if occur.
- We can catch that exception using catch keyword and handle it.
- **Throw keyword is used to manually** throw the exception however system generated exception are **automatically thrown by the CLR itself.**
- Finally block consist of code that must be executed without concerning which either try or catch block was executed.
- The general form of an exception handling block is given below

```
try
{
        //section of code that needs to be monitored
}
catch(ExceptionType1 obj1)
{
        //Exception handler for ExceptionType1
}
catch(ExceptionType2 obj2)
{
        //Exception handler for ExceptionType2
}
finally
{
        //block of code to be executed whether or not an exception is thrown.
}
```

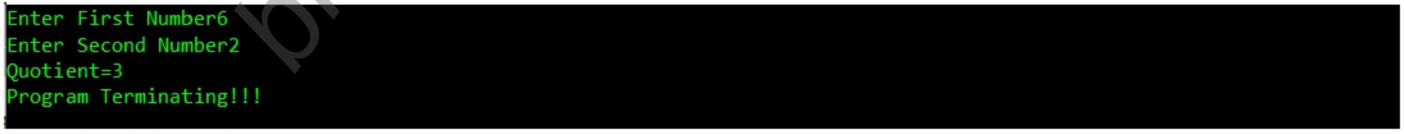
- **The above form is also called as exception handling with multiple catch statement or multilevel exception handling**.

Example

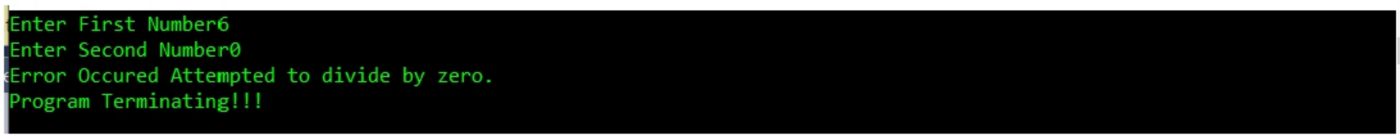
1. **WAP to demonstrate handling DivideByZeroException.**

```
using System;
namespace ConsoleApp
{
    class Program
    {
        public static void Main(String[] args)
        {
            try
            {
                Console.Write("Enter First Number");
                int fn = Convert.ToInt32(Console.ReadLine());
                Console.Write("Enter Second Number");
                int sn = Convert.ToInt32(Console.ReadLine());
                int res = fn / sn;
                Console.WriteLine("Quotient=" + res);
            }
            catch (DivideByZeroException e)
            {
                Console.Write("Error Occured " + e.Message);
            }
            finally
            {
                Console.WriteLine("Program Terminating!!!");
            }
            Console.ReadKey();
        }
    }
}
```

Output:



```
Enter First Number6
Enter Second Number2
Quotient=3
Program Terminating!!!
```



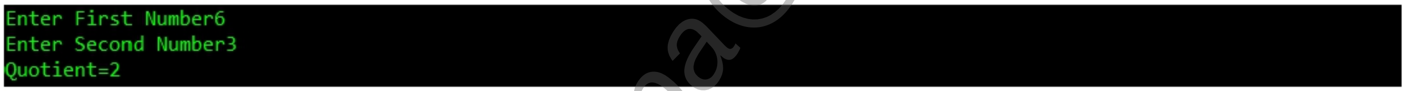
```
Enter First Number6
Enter Second Number0
Error Occured Attempted to divide by zero.
Program Terminating!!!
```

2. **WAP to demonstrate multilevel exception handling.**

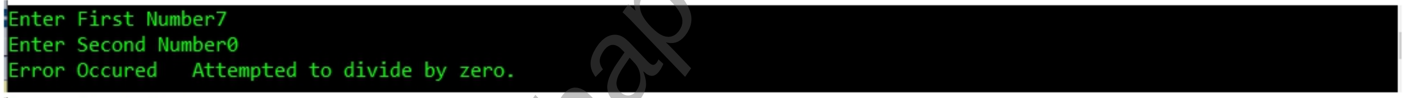
```
using System;
```

```
namespace ConsoleApp
{
    class Program
    {
        public static void Main(String[] args)
        {

            try
            {
                Console.Write("Enter First Number");
                int fn = Convert.ToInt32(Console.ReadLine());
                Console.Write("Enter Second Number");
                int sn = Convert.ToInt32(Console.ReadLine());
                int res = fn / sn;
                Console.WriteLine("Quotient=" + res);
            }
            catch(DivideByZeroException e)
            {
                Console.Write("Error Occured\t" + e.Message);
            }
            catch(FormatException e)
            {
                Console.Write("Error Occured\t" + e.Message);
            }
            Console.ReadKey();
        }
    }
}
```

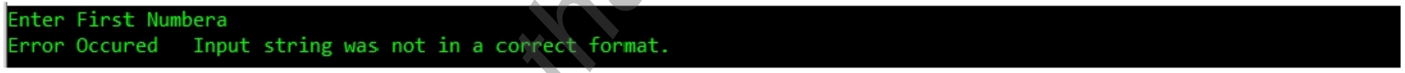
Output:



```
Enter First Number6
Enter Second Number3
Quotient=2
```



```
Enter First Number7
Enter Second Number0
Error Occured   Attempted to divide by zero.
```



```
Enter First Numbera
Error Occured   Input string was not in a correct format.
```

3. **WAP to demonstrate how a single exception handler can handle all the exception.**
Note: Since all the exception are derived from **System.Exception Class, so we use it to handle all the exception.**

```
using System;
namespace ConsoleApp
{
    class Program
    {
        public static void Main(String[] args)
        {
            try
            {
                Console.Write("Enter First Number");
                int fn = Convert.ToInt32(Console.ReadLine());
                Console.Write("Enter Second Number");
                int sn = Convert.ToInt32(Console.ReadLine());
                int res = fn / sn;
                Console.WriteLine("Quotient=" + res);
            }
            catch(Exception e)
            {
                Console.Write("Error Occured\t" + e.Message);
            }
            Console.ReadKey();
        }
```

```
            }
        }
```

The throw Statement

- We can throw an exception, that you just caught, by using the **throw** keyword.
- Basically the common language runtime throws the exceptions and exception handler catch them but if we want to throw the exceptions explicitly/manualy then we have to use throw statement.
- Any type of exceptions which is derived from Exception class can be raised using the throw keyword.
- The general form of throw is

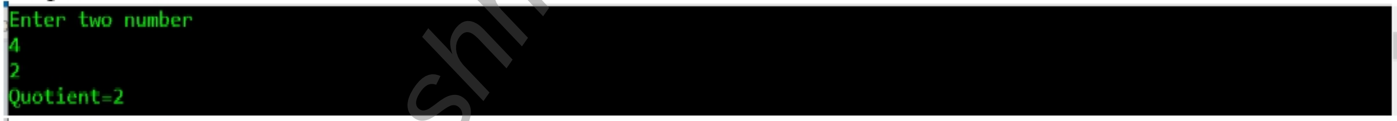
  throw ExceptionInstance

  The ExceptionInstance must be an object sub type Excepition class.

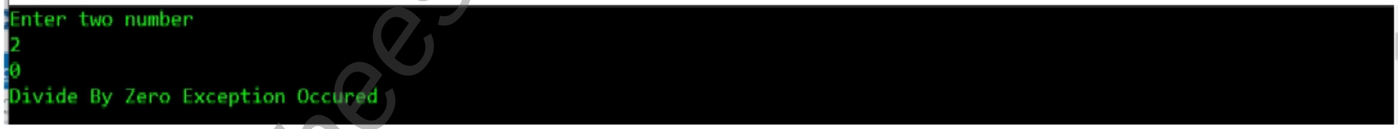
Example:

```
using System;
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                Console.WriteLine("Enter two number");
                int a = Convert.ToInt16(Console.ReadLine());
                int b = Convert.ToInt16(Console.ReadLine());
                if (b == 0)
                {
                    throw new ArithmeticException("Divide By Zero Exception Occured");
                }
                int c = a / b;
                Console.WriteLine("Quotient=" + c);
            }
            catch (ArithmeticException e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadKey();
        }
    }
}
```

Output:



```
Enter two number
4
2
Quotient=2
```



```
Enter two number
2
0
Divide By Zero Exception Occured
```

s

Throwing your own exceptions / creating your own exception subclass

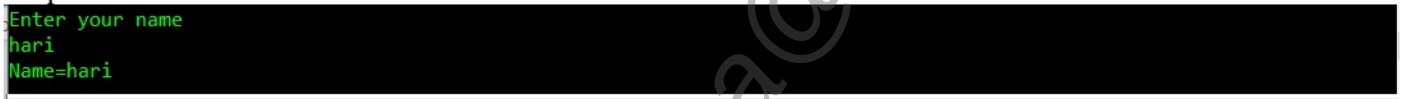
- We can create our own exception class by defining it as a subclass of Exception class.
- Then we can use throw keyword to throw our own exception also.

Example:

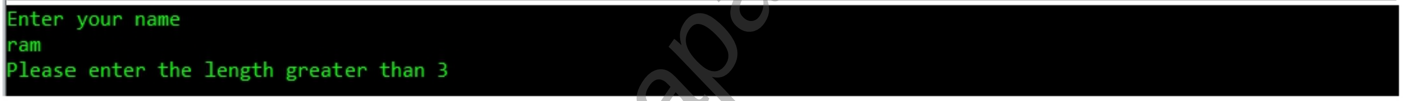
C# program to demonstrate custom exception class

```
using System;
using System.IO;
namespace ConsoleApplication12
{

    class Program
    {
        static void Main(string[] args)
        {


            try
            {
                Console.WriteLine("Enter your name");
                string str = Console.ReadLine();
                if (str.Length <= 3)
                {
                    throw new InvalidLengthException("Please enter the length greater than 3");
                }
                Console.WriteLine("Name=" + str);
            }
            catch (InvalidLengthException e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadKey();
        }
    }
    class InvalidLengthException : Exception
    {
        public InvalidLengthException(String msg):base(msg)
        {
        }
    }
}
```

Output:



```
Enter your name
hari
Name=hari
```



```
Enter your name
ram
Please enter the length greater than 3
```

File Input/Output

- A file is a collection of data stored in a disk with a specific name and a directory path. When a file is opened for reading or writing, it becomes a stream.
- The file becomes stream when we open the file for writing and reading.
- A stream is a sequence of bytes which is used for communication.
- Two stream can be formed from file one is input stream which is used to read the file and another is output stream is used to write in the file.
- In C#, **System.IO** namespace contains classes which handle input and output streams and provide information about file and directory structure.
- Some of the class defined inside namespace System.IO includes
  1. File
  2. FileInfo
  3. Directory
  4. DirectoryInfo
  5. StreamWriter
  6. StreamReader

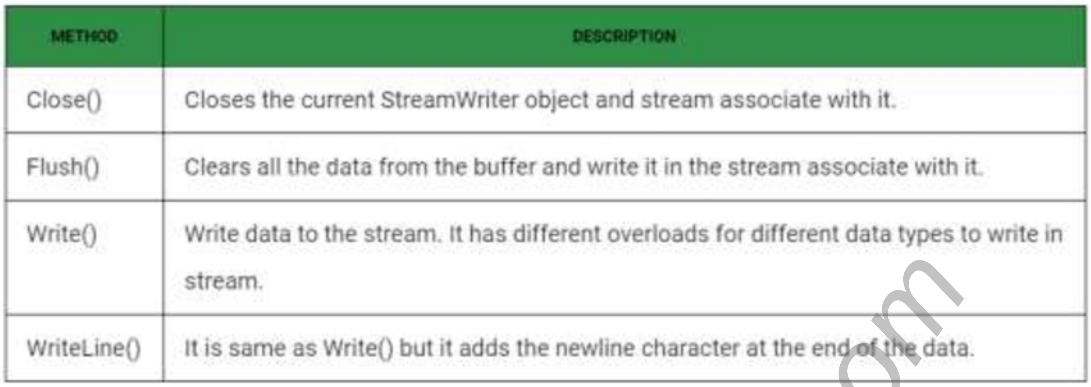
StreamWriter Class

- The StreamWriter class implements TextWriter for writing character to stream in a particular format.
- It is used to write characters to a stream.
- Constructors

1. Streamwriter(String path)
2. Streamwriter(String path, bool append)

Where, path is the disk location to save the file and append is Boolean value which if made true will create the outstream in append mode otherwise create mode.

- Some commonly used methods include



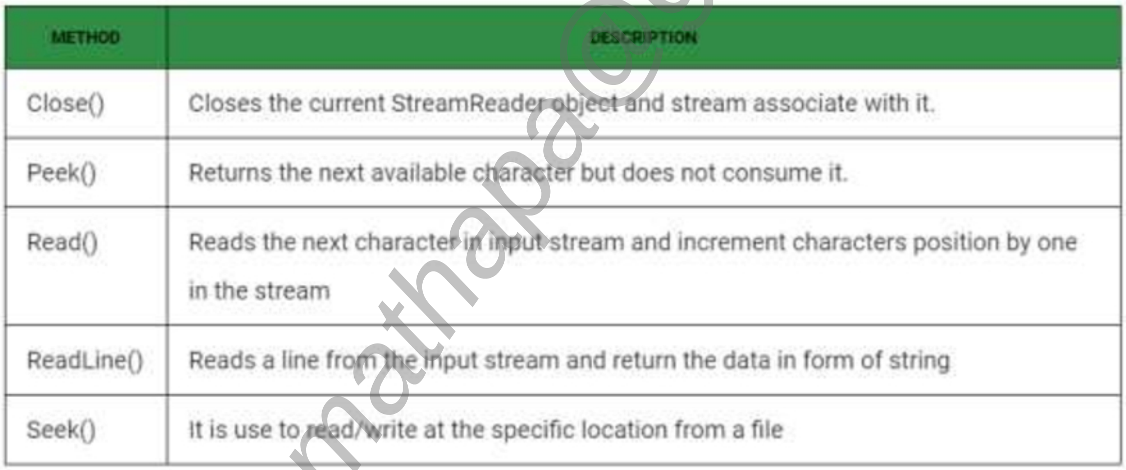
METHOD	DESCRIPTION
Close()	Closes the current StreamWriter object and stream associate with it.
Flush()	Clears all the data from the buffer and write it in the stream associate with it.
Write()	Write data to the stream. It has different overloads for different data types to write in stream.
WriteLine()	It is same as Write() but it adds the newline character at the end of the data.

StreamReader Class

- The StreamReader class implements TextReader for reading character from the stream in a particular format.
- It is used to read characters from a byte Stream
- Constructor

StreamReader(String path)
Where, path is the location to file to be opened for read operation.

- Some commonly used method includes



METHOD	DESCRIPTION
Close()	Closes the current StreamReader object and stream associate with it.
Peek()	Returns the next available character but does not consume it.
Read()	Reads the next character in input stream and increment characters position by one in the stream
ReadLine()	Reads a line from the input stream and return the data in form of string
Seek()	It is use to read/write at the specific location from a file

WAP to write “Hellow World” in file named abc.txt.

```
using System.IO;
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            StreamWriter sw = new StreamWriter("abc.txt");
            sw.Write("Hellow World");
            sw.Flush();
            sw.Close();
        }
    }
}
```

WAP to read the entire content of file named abc.txt

```
using System;
using System.IO;
```

```
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            StreamReader sr = new StreamReader("abc.txt");
            String line;
            while ( (line= sr.ReadLine()) != null)
            {
                Console.WriteLine(line);
            }
            sr.Close();
            Console.ReadKey();
        }
    }
}
```

WAP to count the number of vowel character in filed named abc.txt.

```
using System;
using System.IO;
namespace ConsoleApplication

    class Program
    {
        static void Main(string[] args)
        {
            int count = 0;
            StreamReader sr = new StreamReader("abc.txt");
            int ch;
            while ( (ch= sr.Read()) != -1)
            {
                char c = (char)ch;
                c = Char.ToLower(c);
                if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u')
                {
                    count++;
                }
            }
            Console.WriteLine("Total number of vowel characters: " + count);
            sr.Close();
            Console.ReadKey();
        }
    }
}


Or


using System;
using System.IO;
namespace ConsoleApp
{
    class Program
    {
        static void Main(string[] args)
        {
            int count = 0;
            StreamReader sr = new StreamReader("abc.txt");
            char c;

            while (sr.Peek() != -1)  //Testing EOF
            {
                c =(char)  sr.Read();
                c = Char.ToLower(c);
                if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u')
```

```
                    {
                        count++;
                    }
                }
                Console.WriteLine("Total number of vowel characters: " + count);
                sr.Close();
                Console.ReadKey();
            }

        }
    }
```

A file named abc.txt consist name and roll number of students. WAP to append new student record in the file.

```
    using System;
    using System.IO;
    namespace ConsoleApplication
    {
        class Program
        {
            static void Main(string[] args)
            {
                StreamWriter sw = new StreamWriter("abc.txt", true);
                Console.WriteLine("Enter name and roll of a student");
                string name = Console.ReadLine();
                string roll = Console.ReadLine();
                string rec = name + "\t" + roll;
                sw.WriteLine(rec);
                sw.Flush();
                sw.Close();
            }
        }
    }
```

A file named abc.txt consist name and roll number of students. WAP to append 10 new student record in the file.

```
    using System;
    using System.IO;
    namespace ConsoleApplication12
    {
        class Program
        {
            static void Main(string[] args)
            {
                StreamWriter sw = new StreamWriter("abc.txt", true);
                for (int i = 0; i < 2; i++)
                {
                    Console.WriteLine("Enter name and roll of a student");
                    string name = Console.ReadLine();
                    string roll = Console.ReadLine();
                    string rec = name + "\t" + roll;
                    sw.WriteLine(rec);
                }
                sw.Flush();
                sw.Close();
            }
        }
    }
```

WAP to copy the content of file named abc.txt to def.txt.

```
using System;
using System.IO;
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            StreamReader sr = new StreamReader("abc.txt");
            StreamWriter sw = new StreamWriter("def.txt");
            int ch;
            while ((ch = sr.Read()) != -1)
            {
                sw.Write((char)ch);
            }
            sr.Close();
            sw.Flush();
            sw.Close();
        }
    }
}
```

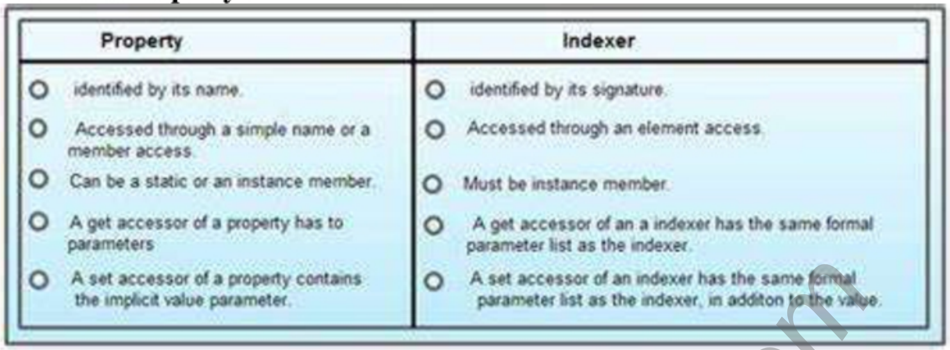
WAP to delete all the vowel letters from file named abc.txt.

```
using System;
using System.IO;
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            StreamReader sr = new StreamReader("abc.txt");
            StreamWriter sw = new StreamWriter("tmp.txt");

            int ch;
            while ((ch = sr.Read()) != -1)
            {
                char c = (char)ch;
                c = Char.ToLower(c);
                if (c != 'a' && c != 'e' && c != 'i' && c != 'o' && c != 'u')
                {
                    sw.Write((char)ch);
                }
            }
            sr.Close();
            sw.Flush();
            sw.Close();
            File.Delete("abc.txt");//abc.txt file will be deleted
            File.Move("tmp.txt", "abc.txt");//tmp.txt will be renamed/moved as abc.txt
        }
    }
}
```

Assignments:

- **Namespace concept with programming example and its importance**
- **WAP to check if the input number is Armstrong number or not.**
- **WAP to input a sentence and display it after removing all the vowels character.**
- **For loop vs foreach loop in c#**
- **Create a class called Quadratic having the fields a, b, c and three function members called SetData(double,double,double) to assign value to these fields from user and DisplyRoots() to display the real roots of quadratic equation. Write a main program to test your class.**

- **Create a class Rectangle with two data members (length and breadth), a function called ReadData() to take detail of sides and a function called DisplayArea() to display area of rectangle. In main create two objects of a class Rectangle and for each object, call both of the function.**
- **Differentiate between Property and Indexers.**



Property	Indexer
○ identified by its name.	○ identified by its signature.
○ Accessed through a simple name or a member access.	○ Accessed through an element access.
○ Can be a static or an instance member.	○ Must be instance member.
○ A get accessor of a property has to parameters	○ A get accessor of an a indexer has the same formal parameter list as the indexer.
○ A set accessor of a property contains the implicit value parameter.	○ A set accessor of an indexer has the same formal parameter list as the indexer, in additon to the value.

- **A company needs to keep record of its following Employees:**

i) Manager ii) Supervisor

The record requires name and salary of both employees. In addition, it also requires section_name (i.e. name of section, example Accounts, Marketing, etc.) for the Manager and group_id (Group identification number, e.g. 205, 112, etc.) for the Supervisor. Design classes for the above requirement. Each of the classes should have a function called set() to assign data to the fields and a function called get() to return the value of the fields. Write a main program to test your classes. What form of inheritance will the classes hold in this case?

<End of Chapter-1>