Exercise 7
Application Development Using Exception Handling

Aim:
To develop C# application program to demonstrate exception handling with try, catch, throw and finally statements.

Exception Handling:
Exception handling is the method of catching and recording these errors in code so you can fix them. Usually, errors and exceptions are stored in log files or databases.

In C#, the exception handling method is implemented using the following statements

- try
- catch
- throw
- finally

Example: Handling the DivideByZeroException
Example for the exception-handling mechanism is as follows, the program given below will compile but will show an error during execution. The **division by zero** is a runtime exception, and the program terminates with an error message as given below.

```
using System;
class Program
    {
     static void Main(string[] args)
        {
            int x = 0;
            int div = 0;
            try
            {
               div = 100 / x;
            }
            catch(DivideByZeroException ex)
            {
               Console.WriteLine("Divide By Zero Error Handled!");
            }
            Console.WriteLine(div);
            Console.WriteLine("Other Tasks..");
            Console.WriteLine("Program Ends");
            Console.ReadKey();
        }
 }
```

Here we are using the object of the standard exception class DivideByZeroException to handle the exception caused by division by zero.

Throwing Exceptions in C#
In C#, it is possible to throw an exception programmatically. The 'throw' keyword is used for this purpose. The general form of throwing an exception is as follows.

```
            throw exception_obj;
```

Example: throwing DivideByZeroException

```
using System;
class Program
    {
     static void Main(string[] args)
        {
            try
            {
                throw new DivideByZeroException("Invalid Division");
            }
            catch(DivideByZeroException)
            {
                Console.WriteLine("Exception");
            }
            Console.ReadKey();
        }
 }
```

User-defined Exceptions in C#
In C#, it is possible to create user defined exception classes. But Exception must be the ultimate base class for all exceptions in C#. So the user-defined exception classes must inherit from either the Exception class.

Example: Write a program to get an Age of a person as input and check if it is greater than 18, if not throw an user defined exception called InvalidAgeException.

```
using System;

class InvalidAgeException : Exception
{
    public InvalidAgeException(string msg) : base(msg)
    {
    }
}
class Program
    {

     static string checkAge(int age)
      {
        if (age < 18)
            throw new InvalidAgeException("Error!Age must be greater than 18");
        else
           return "Yes! You are Eligible";
      }

     static void Main(string[] args)
```

```
        {
            Console.Write("Enter your age: ");
            int age = int.Parse(Console.ReadLine());
            try
            {
               Console.WriteLine(checkAge(age));
            }
            catch(InvalidAgeException e)
            {
               Console.WriteLine(e.Message);
            }
            Console.ReadKey();
        }
 }
```

Set A: Complete the followings Questions

1. You will be given two integers x and y as input, you have to compute x/y. If x and y are not 32 bit signed integers or if y is zero, exception will occur and you have to report it. Read sample Input/Output to know what to report in case of exceptions.

Sample Input1:
10
3
Sample Output1:
3

Sample Input2:
10
Hello
Sample Output2:
System.ArgumentException

Sample Input3:
10
0
Sample Output3:
System.DivideByZeroException: / by zero

Sample Input4:
23.323
0
Sample Output 4:
System.ArgumentException

2. You are required to compute the power of a number by implementing a calculator. Create a class MyCalculator which consists of a single method long power(int, int). This method takes two integers, n and p, as parameters and finds n^p. If either n or p is negative, then the method must throw an exception which says "n or p should not be negative". Also, if both n and p are zero, then

the method must throw an exception which says "n or p should not be zero". Complete the function power in class MyCalculator and return the appropriate result after the power operation or an appropriate exception as detailed above.
Sample Input1:
3 5
Sample Output1:
243

Sample Input2:
0 0
Sample Output2:
System.Exception: n and p should not be zero

Sample Input3:
-1 3
Sample Output3:
System.Exception: n or p should not be negative.

Set B: Select the Question based on the formula given below.
Question Number = (Regno%5) + 1

1. Create a class to represent an Employee with empid, name and age as attributes. Create a Main class and create an employee object by getting the empid, name and age as input and check for constraints as given below. Also generate and handle the user defined exceptions if the user inputs are not proper as per the following constraints

➢ If the empid is below 4 digits, throw InvalidEmpidException.
➢ If the name is a number, InvalidNameException must be thrown.
➢ If the age is greater than 50, an InvalidAgeException must be thrown
➢ If no error an object must be created for the entered employee details.

2. Create a class Account with data such as name, balance, atmPin and initialize using parameterized constructor. Include methods such as checkPin() and withdraw(). Create a Main class, then perform checkPin() and withdraw() operations. Also, generate and handle the user defined exceptions for the following situations.

- In checkPin(), check for "atmPin" if not valid throw InvalidATMPinException.
- In withdraw(), throw "NoCashException" if the account is running out of balance.

3. Develop a program to create a user defined exception for the following.

- Create a student class with name, mark1, mark2, and mark3 as data members.
- Save the data using the constructor method.

- When storing name. If the length > 7 throw LengthException and handle the same.
- After saving the marks, calculate the average of marks and do the following
  if avg < 50 throw FailedException
  If avg < 75 && avg > 50 throw NotFirstClassException
  If avg > 75 throw FirstClassException

4. Create a class called Stack to represent a data structure (LIFO) with data such as size, top, elements array. Add constructor and methods such as push() and pop(). In Main class, create a stack object, perform push() and pop() operations. Generate and handle custom exceptions for the following situations.
• While push operation, throw and handle StackFullException if stack is full.
• While pop operation, throw and handle StackEmptyException if stack is empty.

5. Create a class called Queue to represent a data structure (FIFO) with data members such as items array, rear, front, size. Include constructor to store the Queue data and demonstrate the exception handling for the following.

- Do Insert operation, if Queue is full generate an exception “QueueFullException” and handle the same.
- Do Delete operation, if Queue is empty generate an exception “QueueEmptyException” and handle the same.