East West University
Department of Computer Science and Engineering
A/2, Jahurul Islam Avenue, Jahurul Islam City, Aftabnagar, Dhaka

Lab Manual: 10

Lab Topic: Exception Handling

Course Code: CSE110 (Object Oriented Programming)

Course Instructor: Mahamudul Hasan, Senior Lecturer, CSE

Lab Objective

1. **Learn a** mechanism to handle Exception in Java program

Lab Activities:

A. Built-in Exceptions Handle

```
class ArithmeticException_Demo
{
    public static void main(String args[])
    {
        try {
            int a = 30, b = 0;
            int c = a/b;
            System.out.println ("Result = " + c);
        }
        catch(ArithmeticException e) {
            System.out.println (e);
            System.out.println ("Can't divide a number by 0");
        }
    }
}

Class Testthrows1 {
  Void m () throws IOException
  {
    throw new IOException("device error");//checked exception
  }
}
```

Lab problem 1:

- **Write a program that creates a *Calculator* class.** The class contains two private variables of double type. Design a constructor that accepts two values as parameter and set those values. Also create a no-argument constructor and accessor, mutator.
- Design four methods named *Add (), Subtract (), multiply (), Division ( )* for performing addition, subtraction, multiplication and division of two numbers.
- For addition and subtraction, the sum of two numbers should be positive. If the sum is a negative number then throw an exception in respective methods. So design an exception handler (*ArithmeticException*) in *add ()* and *subtract ()* methods respectively to check whether the sum of the both number is negative or not.
- For division and multiplication two numbers should not be zero. If zero is entered for any number then throw an exception in respective methods. So design an exception handler (*ArithmeticException*) in *multiply ()* and *division ()* methods respectively to check whether any number is zero or not.
- Write a main class and declare four objects of *Calculator* class. Perform addition (obj1), subtraction (obj2), multiply (obj3) and division (obj4) operations for these objects. If any non numeric values are provided as input; then you should throw an exception (*NumberFormatException*) and display a message that informs the user of the wrong input before exiting. Your program should take input infinitely until the non-numeric inputs

are provided.

B. User Defined Exceptions

Lab problem 2:

- **Create an exception class named *FruitException*** that extend a base class named *Exception*
- Design a constructor in your class that accepts a string value set it to the super class constructor to display the exception message.
- Create a main class named *Fruit*. Write a method inside the class called *fruitPrice(double price, double weight)* that accepts price and weight per kg of the fruit. Inside the method, if the price per kg is less than 50 then throw an exception with string "Available Fruit!" and if the price per kg is greater than 500 then throw an exception with string "Rare Fruit!" otherwise print the total price of the fruit by multiplying the price and weight.
- Inside the main method declare single object of the product class and call the *fruitPrice()* method to display the fruit information.

C. Java Multi-Catch block

- At a time only one exception occurs and at a time only one catch block is executed.
- All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception.

```
public class MultipleCatchBlock1 {

    public static void main(String[] args) {

        try{
            int a[]=new int[5];
            a[5]=30/0;
           }
        catch(ArithmeticException e)
              {
               System.out.println("Arithmetic Exception occurs");
              }
        catch(ArrayIndexOutOfBoundsException e)
              {
               System.out.println("ArrayIndexOutOfBounds Exception
occurs");
              }
        catch(Exception e)
              {
               System.out.println("Parent Exception occurs");
              }
      System.out.println("rest of the code");
   }
}
```

- Try for the following code:

```
1. try{
    int a[]=new int[5];
    System.out.println(a[10]);
   }
2.
   try{
      int a[]=new int[5];
      a[5]=30/0;
      System.out.println(a[10]);        }

3.
 try{
    String s=null;
```

```
                        System.out.println(s.length());
                    }



                    4.
                     class MultipleCatchBlock5{
                        public static void main(String args[]){
                         try{
                             int a[]=new int[5];
                             a[5]=30/0;
                         }
  catch(Exception e){System.out.println("common task completed");}
  catch(ArithmeticException e){System.out.println("task1 is completed");}
  catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}

  System.out.println("rest of the code...");
}  }
```

- Grouping exception in catch block

```
catch (NoSuchPaddingException | NoSuchAlgorithmException
            | InvalidKeyException | BadPaddingException
            | IllegalBlockSizeException | IOException ex) {
      System.err.println(ex);
}
```

Lab problem 3:
Write a sample code to implement multi catch (minimum 5 catch block) and a finally block.

D. Java Nested try block:

Check the following code.

```
class Excep6{
    public static void main(String args[]){
           try{
                try{
                   System.out.println("going to divide");
                   int b =39/0;
                }
                catch(ArithmeticException e)
                 {System.out.println(e);}

                try{
                   int a[]=new int[5];
                   a[5]=4;
                }
                catch(ArrayIndexOutOfBoundsException e)
                 {System.out.println(e);}
                System.out.println("other statement);
            }
            catch(Exception e)
               {System.out.println("handeled");}
              System.out.println("normal flow..");
        }

    }
```

Lab Problem 4: Modify the previous code with at least 5 nested try catch block.

E. Problem

Lab Problem 5:

You are given a piece of Java code (You can safely assume any code that follows our criteria described below). You have to complete the code by writing down the handlers for exceptions

thrown by the code. The exceptions the code may throw along with the handler message are listed below:

Division by zero: Print "Invalid division".
String parsed to a numeric variable : Print "Format mismatch".
Accessing an invalid index in string : Print "Index is invalid".
Accessing an invalid index in array : Print "Array index is invalid".

MyException : This is a user defined Exception which you need to create. It takes a parameter param. When an exception of this class is encountered, the handler should print "MyException[param]", here param is the parameter passed to the exception class.

Exceptions other than mentioned above : Any other exception except the above ones fall in this category. Print "Exception encountered".

Finally, **after the exception is handled, print "Exception Handling Completed"**.

Example: For an exception of MyException class if the parameter value is 5, the message will look like MyException[5].

Input Format:
The code handles all the input itself.

Output Format:
If any exception is encountered in the code, print the respective handler code. The last line of output should be "Exception Handling Completed".