Data Structures and Algorithms
Lab 5: Exception Handling

I. Objective

After completing this tutorial, you can:

- Understand and implement *exception handling* in Java program.

II. Motivation

There are three types of errors:

- Syntax errors: Occurs when the rule of the language is violated and detected by compiler.
- Run-time errors: Occurs when the computer detects an operation that cannot be carried out (*e.g.*, division by zero; *x/y* is syntactically correct, but if *y* is zero at run-time a run-time error will occur).
- Logic errors: Occurs when a program does not perform the intended task.

Instead of deciding how to deal with an error, Java provides the *exception* mechanism:

- Indicate an error (exception event) has occurred,
- Let the user decide how to handle the problem in a separate section of code specific for that purpose,
- Crash the program if the error is not handled.

III. Exception Indication

1. Use built-in exception class

There are many useful predefined exception classes, for example:

- `ArithmeticException`
- `NullPointerException`
- `IndexOutOfBoundsException`
- `IllegalArgumentException`
- `InputMismatchException`

The following Java programs will illustrate the use of *Exception Indication*.

```
// Program-1
import java.util.Scanner;
import java.util.InputMismatchException;

public class SampleExceptionA {

    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        boolean isError = false;

        do
        {
            System.out.print("Enter an integer number: ");
            try
            {
                int num = sc.nextInt();
                System.out.println("num = " + num);
                isError = false;
            } catch(InputMismatchException e)
            {
                System.err.println("Incorrect input!");
                sc.nextLine();  // skip newline
                isError = true;
            }
        } while(isError);
    }
}
```

```
// Program-2
public class SampleExceptionB {

    public static double factorial(int n) throws IllegalArgumentException
    {
        if(n < 0)
        {
            IllegalArgumentException obj = new IllegalArgumentException(n +
" is invalid.");
            throw obj;
        }
        else
        {
            double output = 1;
            for(int i = 2; i <= n; i++)
            {
                output *= i;
            }
            return output;
        }
    }

    public static void main(String[] args)
    {
        System.out.println("n = 5 --> " + factorial(5));
        System.out.println("n = -1 --> " + factorial(-1));
        System.out.println("n = 6 --> " + factorial(6));
    }
}
```

2. Define new Exception class

New exception classes can be defined by deriving from class *Exception*. Then it can be used in `throw` statements and `catch` blocks.

```
public class MathException extends Exception {

    public MathException()
    {
        super();
    }

    public MathException(String s)
    {
        super(s);
    }
}
```

```
public class SampleExceptionC {

    public static double factorial(int n) throws MathException
    {
        if(n < 0)
        {
            throw new MathException(n + " is invalid.");
        }
        else
        {
            double output = 1;
            for(int i = 2; i <= n; i++)
            {
                output *= i;
            }
            return output;
        }
    }

    public static void main(String[] args) throws MathException
    {
        System.out.println("n = 5 --> " + factorial(5));
        System.out.println("n = -1 --> " + factorial(-1));
        System.out.println("n = 6 --> " + factorial(6));
    }
}
```

IV. Exercises

Review all previous exercises from *Lab-1* to *Lab-4*, and implement the Exception Handling for appropriate exercises.