Lab11: Exception Handling and Threads

In this lab, the following topics will be covered:

1. **Exceptions**
2. **Types of Exceptions**
3. **Threads**
4. **Using threads**
5. **Timer application.**

1. Exceptions

An ***execption*** is a run-time event that indicates that something has gone wrong. The java language uses *exceptions* to provide error handling capabilities for its programs.

Keywords:

try, *catch*, *throws* and *finally*

There are many different Java *Exception* errors. Some examples below;

- *IOException* - I/O exception error
- *RunTimeException* - Run-time error
- *ArithmeticException* - Divide-by-zero error exeception
- *ArrayIndexOutOfBoundsExecption* - Array index out of bounds error
- *FileNotFound* - Attempt to get file that does not exist
- *EndOfFileException* - End of file marker error

```
//  General Syntax

In Java, exception handling is managed by try blocks.
try {
    //try block
}
catch (ExceptionType1 param1) {
    //exception-handling block
}
catch (ExceptionType2 param2) {
    //exception-handling block
}
...
catch (ExceptionType3 param3) {
    //exception-handling block
}
finally {
    //finally block
    //This block is executed in all circumstances!
}
```

Example of program without exception handling

```
/*
 * Program without Exception Handling
*/

import java.util.Scanner;

public class WithoutException
{
    public static void main(String[] args)
    {
        Scanner input = new Scanner(System.in);
        int donutCount, teaCount;
        double donutsPerGlass;

        System.out.println("Enter number of donuts:");
        donutCount = input.nextInt( );

        System.out.println("Enter number of cup of tea:");
        teaCount = input.nextInt( );

        if (teaCount < 1)
        {
            System.out.println("No Tea!");
            System.out.println("Go buy some tea.");
        }
        else
        {
            donutsPerGlass = donutCount/(double)teaCount;
```

```
            System.out.println(donutCount + " donuts.");
            System.out.println(teaCount + " cup  of tea.");
            System.out.println("You have " + donutsPerGlass + " donuts
for each cup of tea.");
        }
        System.out.println("End of program.");
    }
}
```

Compile and run the program according to the following

```
Enter number of donuts:
2
Enter number of cup of tea:
0
No Tea!
Go buy some tea.
End of program.
```

Example of program with exception handling

```
// An Example of Exception Handling

import java.util.Scanner;
public class ExceptionDemo {

    public static void main(String[] args){
        Scanner input = new Scanner(System.in);
         int donutCount, teaCount;
        double donutsPerGlass;

        try  //try block
        {
            System.out.println("Enter number of donuts:");
            donutCount = input.nextInt( );

            System.out.println("Enter number of cup of tea:");
            teaCount = input.nextInt( );

            if (teaCount < 1)
              //when the throw statement is executed, the block is
stopped
              throw new Exception("Exception: No Tea!");

            donutsPerGlass = donutCount/(double)teaCount;
            System.out.println(donutCount + " donuts.");
            System.out.println(teaCount + " cup of tea.");
            System.out.println("You have " + donutsPerGlass + " donuts
for each cup of tea.");
        }
```

```
        catch(Exception e)  // catch block
        {
            System.out.println(e.getMessage( ));
            System.out.println("Go buy some tea.");
        }

        System.out.println("End of program.");
    }
}
```

Compile and run the program according to the following

```
First run:
Enter number of donuts:
3
Enter number of cup of tea:
2
3 donuts.
2 cup of tea.
You have 1.5 donuts for each cup of tea.
End of program.

Second run:
Enter number of donuts:
2
Enter number of cup of tea:
0
Exception: No Tea!
Go buy some tea.
End of program.
```

In Java, methods that can fail are declared with a *throws* clause. The *throws IOException* clause means that this method can be invoked only within try clause.

```
/*
 * Divide by zero error Exception Handling
*/

//Example of Exception Handling  program
import java.util.Scanner;
import java.io.*;
public class TestExceptions
{
  static int getInt() throws IOException
  {
     Scanner input =  new Scanner(System.in);
```

```
    System.out.print("Enter an integer: ");
    int s = input.nextLine();
    return (s);
  }
  public static void main(String[] args)
  { int n1=0, n2=1, n3=0;
    try {
      n1 = getInt();
      n2 = getInt();
      n3 = n1/n2;
    }
    catch (Exception e)
    { System.out.println("[" + e + "]"); }
    System.out.println(n1 + "/" + n2 + " = " + n3);
  }
}
```

Compile and run the program according to the following

```
Sample output (Everything works fine here);
Enter an integer: 22
Enter an integer: 7
22/7 = 3
Sample output (error! - cannot divide by zero);
Enter an integer: 22
Enter an integer: 0
[java.lang.ArithmeticException: / by zero]
22/0 = 0
Sample output (Incorrect input data type);
Enter an integer: 22
Enter an integer: w
[java.lang.ArithmeticException: w]
22/1 = 0
```

Summary:

1. An Exception is an object of a clsss that is descendant of the class Exception
2. There are predefined exception classes. You may also define your own exception classes
3. Certain Java statements themselves might throw an exception
4. An exception is thrown in a try block
5. An exception is caught in a catch block
6. Every exception has a getMessage() method
7. Do not overuse exception

2. Threads

A thread can be thought as an execution path within a process. A process (i.e. a Java application) will at least have one running thread, the thread main. Up to this point, we have been creating single threaded java applications. However, almost all computer application are multi-threaded.

When creating more than one thread, these threads are run simultaneously, the CPU will switch from one thread to another very quickly that programmer would have the feeling that they run in parallel.

Creating your first thread: Counter

We will create a class Counter with one constructor that has the time to count down in seconds. To make the class a thread, it must extend the java.lang.Thread class.

```
// An Example of Thread
public class Counter extends Thread
{
	int time;
	public Counter(int seconds){
		time = seconds;
	}
	public void run(){
		while(time > 0){

	System.out.println(Thread.currentThread().getName()+"  "+time);
			try {
				Thread.currentThread().sleep(1000);
				time--;
			}
			catch(InterruptedException ex){
				System.out.println("error :"+ex.getMessage());
			}
		}
	}
}
```

And to test this class with three different threads

```
/*
 * An Example of Test class
*/

public class MainThread
{
      public static void main(String[]arfs)
      {
            Counter c1 = new Counter(10);
            Counter c2 = new Counter(4);
            Counter c3 = new Counter(2);
            c1.start();
            c2.start();
            c3.start();
      }
}
```

Note:

The start() method for a thread class will invoke its run method. The "run" method for a thread class is the place from which thread execution start.

Another way of creating a thread is by implementing the interface Runnable.

The counter example looks like

```
public class Counter implements Runnable
{
…
}
```

Accordingly the test class will be modified to the following:

Thread obj = new Thread (runnableObject);

We note that Counter class is a class that implements runnable.

```
public class MainThread {
      public static void main(String[]arfs) {
            Thread c1 = new Thread(new Counter(10));
            Thread c2 = new Thread(new Counter(4));
            Thread c3 = new Thread(new Counter(2));
            c1.start();
            c2.start();
            c3.start();
      }
}
```

Using Threads: Timer Application

Examples of threads are those applications that need a timer; think of an application of a window that will close in 5 seconds!

```
/*
 * An Example of Test class
*/

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class TimedBomb extends JFrame implements Runnable
{
      int time = 5;
      JLabel lbl = new JLabel("Will Self-distruct In :"+time);
      public TimedBomb()
      {
            super("Self Destruction");
            setSize(300,100);
            add(lbl);
            setVisible(true);
            Thread t = new Thread(this);
            //this refer to this class, as it implements runnable
            t.start();
      }
      public void run()
      {
            while(time > 0)
            {
                  try
                  {
                        Thread.sleep(1000);
                        time--;
                        lbl.setText("Will Self-distruct In :"+time);
                  }
                  catch(InterruptedException e)
                  {
                  }
            }
                  JOptionPane.showMessageDialog(this,"Window is
closing.. good bye!");
                  System.exit(0);
      }
      public static void main(String[]args)
      {
            new TimedBomb();
      }
}
```

Project

- Your final demonstration of mini group project according to the following criteria.
  - **Presentation (20 Marks)**
    - Introduction
    - Problem statement
    - UML Diagram (Class diagram + relationships)
    - OOP Concepts (Encapsulation, Associatin, Aggregation, Inheritance, Polymorphism, Abstraction, Interface)
    - Database Design (Main Tables for data storage)
    - Screen Shots
  - **Demonstration (20 Marks)**
    - Graphical User Interface (GUI)
    - Input /output with validation and verification
    - Database connectivity
    - Exception Handling
  - **Final Report (10 Marks)**
    - Contains above all with test data.
    - Hard Copy + Code (zip/rar Source Code and mail to **tariq.bimcs@gmail.com**)