Lab 02

Introducing Java Applications I

Objectives: ***At the end of this lab session the student should be able to:***

- Use the Eclipse IDE to develop and run Java applications.
- Use Java input/output statements.
- Use control statements.
- Debug Java programs.

Part 1: Problem Solving Exercises (*Instructor-led*) (Time: 60 minutes)

Creating and running a Java application:

A Java source file is a text file that contains one or more class definitions. The Java compiler expects these files to be stored with the `.java` filename extension. When Java source code is compiled, each individual class is put into its own output file named after the class with a `.class` extension.

Exercise 1: Experiment with a Java Program

- Create a Java project called `project21` and under the `src` folder of `project21` create a class called `Hello.java` in which you type the following class:

```
public class Hello {
     public static void main (String args[]) {
         System.out.print("Hello, welcome to CMPS252");
     }
}
```

- Run the class as a "Java Application". What was the output? _____
- Perform the following experiments and record your findings:
  - Remove the first **public** keyword. Does the class compile? _____ Does it run? ____ Restore the public keyword.
  - Remove the second **public** keyword. Does the class compile? ___ Does it run? ___ Restore the public keyword.
  - Remove the **static** keyword. Does the class compile? ____ Does it run? _____ Restore the static keyword.
  - Remove the **void** keyword. Does the class compile? _____ Does it run? _____ Restore the void keyword.
  - Replace the **void** keyword with **int** and add a return 0; statement just before the end of the main method. Does the class compile? _____ Does it run? _____ Restore the void keyword.
  - Rewrite the method name as **Main** instead of **main**. Does the class compile? _____ Does it run? _____ Restore the main method name.

- Change the type of the **args[]** array from String to **int**. Does the class compile? ____ Does it run? _____ Restore **args** data type.
- Change the argument name from **args** to **myArgs**. Does the class compile? _____ Does it run? _____ Restore **args** name.
- Change the argument from **args[]** to **[]args**. Does the class compile? _____ Does it run? _____

- What is the conclusion from the above experiments? _____
  _____
  _____
- Change the program so that it outputs your name repeated 6 times, printed in one line, separated with the tab character.
  (Hint: use a ***while*** loop, following the same syntax that you learnt in course CMPS152).
- Repeat the above, but using a ***do-while*** loop and let the 6 names be printed in one line, separated with the tab character.

Exercise 2: Selection

2.1 (*Do not type, just study and then answer)* What is output by the following program? Assume the user enters 65 for one execution of the program and 59 for a second execution.

```
import java.util.Scanner;
public class Compare1 {
     public static void main( String args[] ) {
          int mark;
          Scanner cin = new Scanner(System.in);
          System.out.println("Enter mark (an integer):");
          mark = cin.nextInt();
          if   ( (mark/20)>=3 )
               System.out.println( "Pass" );
          else
               System.out.println(  "Fail" );
          cin.close();
     }
}
```

Your Answer: _____
_____

2.2 Switch Statement

- Unlike **if** statement (single selection) and **if-else** (double selection), **switch** is a multiple selection statement which allows a variable to be tested for equality against a list of values.
- Each value is called a case, and the variable being switched on is checked for each case.
- A switch works with the byte, short, char, and int primitive data types and with Strings (New as of Java7).

Perform the following:

1. Make a copy of the Compare1 class and name it Compare2.
2. Replace the if-else statement with a switch that displays the following messages based on the values of the switch selector:

Value(s)	**Message**
4,5	Excellent
3	Very Good
2	Good
All other values	Fail

Exercise 3: Introducing printf() and String.format()

- The following application outputs temperature in Fahrenheit and its equivalent in Centigrade.

```
public class TemperatureConverter {
    public static void main(String args[]) {
        System.out.println("Fahrenheit"+"\t"+"Centigrade");
        for (int fahr=0;fahr<=100; fahr+=10){
              double centigrade = 5.0*(fahr-32)/9.0;
              System.out.println(fahr+"\t"+centigrade);
        }
    }
}
```

- Perform the following:
  - Create a new Java project.
  - Create a new class called "`TemperatureConverter`" and copy to it the above code and then run it. The output should be similar to the following (partly shown here):

```
Fahrenheit      Centigrade
0       -17.77777777777778
10      -12.222222222222221
20      -6.666666666666667
:
:
90      32.22222222222222
100     37.77777777777778
```

- Answer the following questions:
  - What is the purpose of the "`\t`" which is used in the above code? _____ _____
  - Is it possible to use print or println() to force the output of *Centigrade* values to be shown to two decimal places? _____
  - The word 'Fahrenheit' has 10 letters: is it possible to output the values of *Fahrenheit* to be right-justified in a field width of 10 characters, similar to the following?_____

```
Fahrenheit
         0
        10
       100
```

Using printf()

The Java printf() method provides us with the ability to do the above, and much more. Unlike print() and println(), the printf() method takes a formatting string that has "place holders" used to control the output format for our variables/expressions which must be passed as parameters to printf() (that is to say, they should be separated with a comma). For example:

%d Is used to format numbers of type int, long or byte. If you want to specify a field width of say, 10 characters, then use **%10d** in the format-string. The numbers will be output right-justified within this field.

%f Is used to format numbers of type float or double. If you want to specify a field width of say, 15 characters, and decimal places of 3, then use **%15.3f** in the format-string. The numbers will be output right-justified within this field.

%s Is used for String formatting.

[Please refer to your textbook for further details, when needed]

- Now, modify the TemperatureConverter class so that its output should be similar to the following:

```
Fahrenheit  Centigrade
         0      -17.78
        10      -12.22
        20       -6.67
         :
         :
        90       32.22
       100       37.78
```

Using String.format()

The String.format() method allows you to format the output and store it in a string variable. This variable can then be output using the usual print() method. This method has the same signature as the printf() method and works in a similar fashion, except that the output is stored in a String variable rather than printed.
The advantage of this alternative becomes important when some GUI messages need to be formatted before placing them on the GUI component, as will be seen later. You can also store the formatted output in a String variable and then use the print() method to display it on the Console in a single operation, as in the following:

- Now, make a copy of your TemperatureConverter class (Simply click on the file name and then type **CTRL-C** followed by **CTRL-V**. Then rename the file to TemperatureConverter2.
- Add a String variable inside the *for-loop* just after the line that defines and calculates the centigrade variable.:
  `String formattedLine = String.format(`*insert the same args of the printf() method*`);`
- Replace `printf(…)` with `print(formattedLine)`, then save and run. You should get the same output as before.

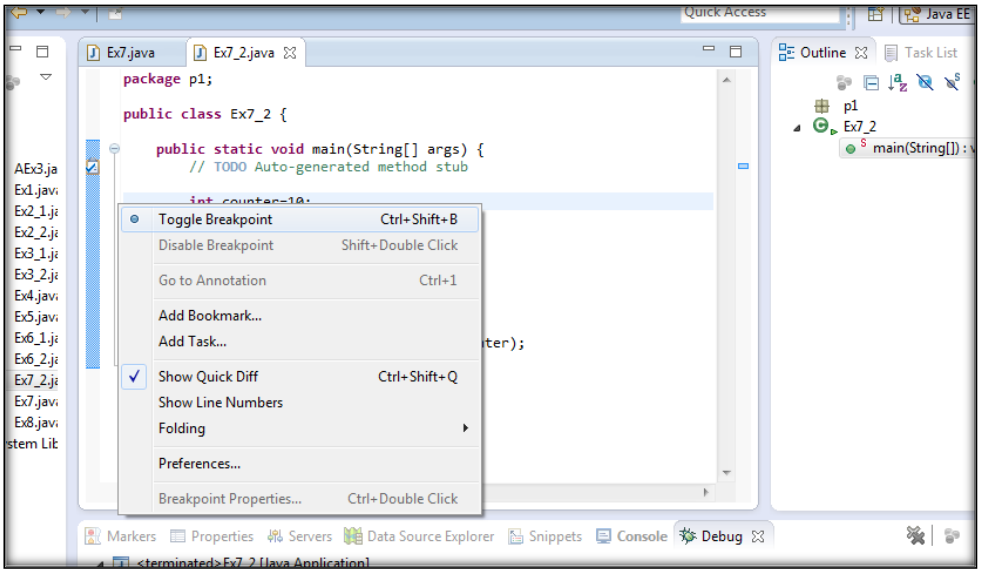
Exercise 4: Debugging

- Debugging allows you to run a program interactively while watching the source code and the variables during the execution.
- By breakpoints in the source code you specify where the execution of the program should stop.
- Once the program is stopped you can investigate variables and change their content.
- Eclipse has a special Debug perspective, which gives you a preconfigured set of views. In this perspective, you control the execution process of your program and can investigate the state of the variables.

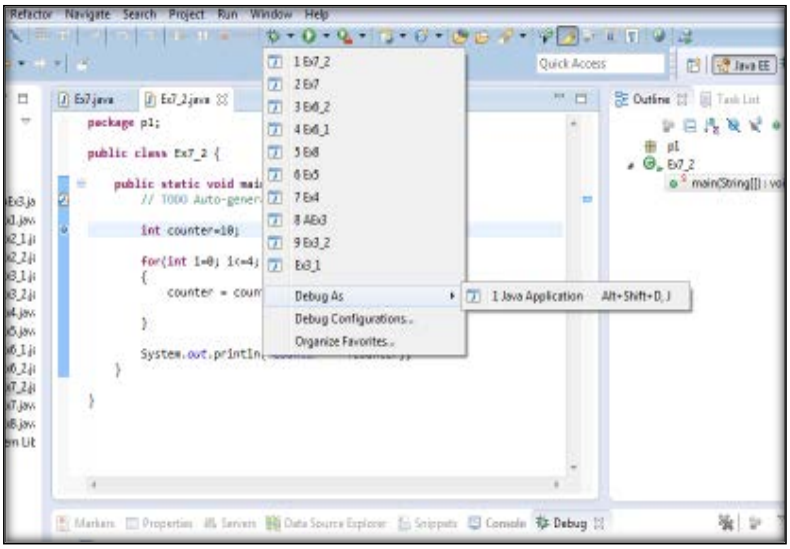
- Let's debug the following program and monitor how the for loop is executed.

```
public class Experiment1 {
		public static void main (String args[] )
		{
			int counter=10;
			for (int i=0; i<=4; i++)
			{
				counter = counter + i;
			}
			System.out.println("Counter = "+ counter);
		}
}
```

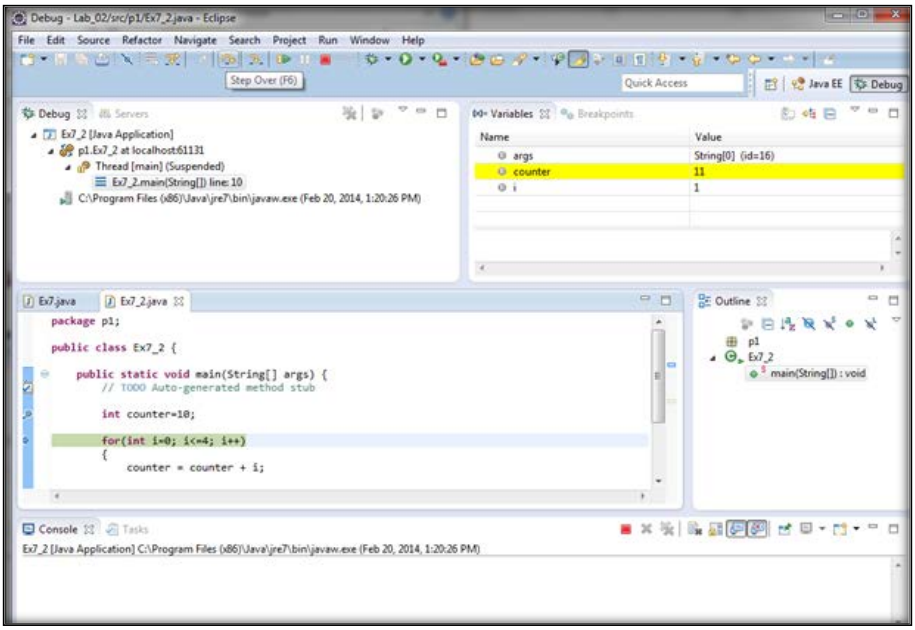
- To start debugging, add a breakpoint on the variable which you want to start from by right clicking on the left bar and selecting Toggle Breakpoint.



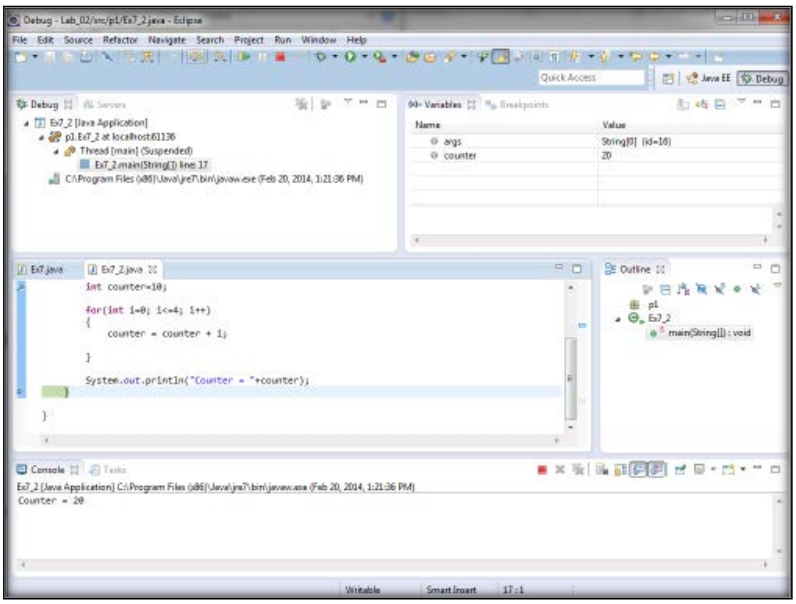
- Start debugging by clicking on Debug as shown in the figure.



- As you Step Over you will see how the program is executed and the changes occurring on the variables.



- The last statement is printed when it is reached at the end of the program.



Part 2: Practice Exercises (Time: 60 minutes)

Exercise 5: Switch Statement

- Write an application that prints the month of the year depending on the value of an integer variable month.

Sample Output:

```
Enter Month: 9
September
```

- Modify the previous application to such that it reads the name of the month and prints its number. *[Hint: use a String variable]*

Sample Output:

```
Enter Month: May
5
```

Exercise 6: Number Calculations

Write a Java application that performs the following:

- Read the marks of a student in three subjects: math, computer and music (assume all are integer numbers).
- Calculate the total mark and the average mark.
- Determine the highest and the lowest of the three marks.
- Display the total mark, the average mark, the smallest mark and the highest mark as shown in the sample output below.

[Hint: To read integers, use the `nextInt()` method of the **cin** object as in Exercise 2]

Sample Output:

```
Enter Math's mark: 90
Enter Computer's mark: 95
Enter Music's mark: 85

Highest mark is 95
Lowest mark is 85
Total mark is 270
Average mark is 90
```

Exercise 7:

Write an application that outputs Distance values measured in inches and its equivalent in Centimeters for all Distance values between 0 and 50 inches in intervals of 2 inches. The equation to convert inches to centimeters is:

*centimeters = 2.54*inches*

(*format your inches values as integers and the centimeters values as double values displaying 2 decimal places after the decimal point.*)

Additional Practice Problems

1. Write an application to read the wattage (or watts) of a standard light bulb and assign to a variable called lumens the expected brightness of that bulb. Use this table:

Watts	Brightness (lumens)
15	125
25	215
40	500
60	880
75	1000
100	1675

Assign the value `-1` if the value of *watts* is not in the table. *[Hint: use the **switch** statement]*

2. Write Java application to calculate the following equation:

   `Y=1/2+3/4+5/6+...+n/(n+1)` `n` is an odd number entered by user.

   - Extend your application to solve the following equation

     `Y=1/2!+3/4!+5/6!+...+n/(n+1)!`