JAVA PROGRAMMING – EXERCISE 9

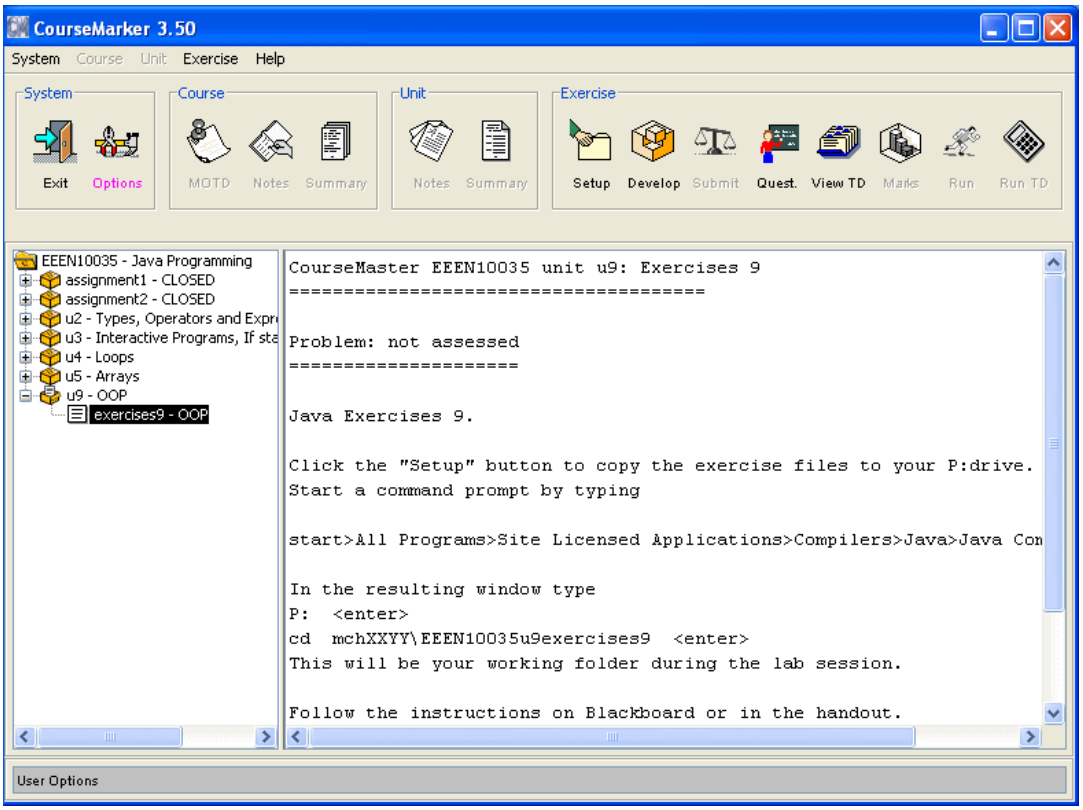
In this lab, we'll be dealing with programs that have multiple classes.

COURSE MARKER

1) Start Course Marker by selecting

Start>All Programs>EPS >EEE >Course Marker 4
After a few moments you will see the login window –
NOTE: User Name is your usual username (mch…), and the password is your student ID number:

2) Once you're in, access the relevant assignment by selecting:
EEEN10035 – Java Programming>u9 – OOP:



3) Click the Setup button on top, and then click OK.
   Clicking the Setup button copies the files needed for this exercise to the folder P:>mchXXYY> EEEN10035u9exercises9 (where mchXXYY is replaced with your username).

   The files copied are:
   Volt.java, TestVolt.java, SafeCheck.java, TestSafeCheck.java, Thanks.java and TestThanks.java .
   These are the java files you'll be working on this time.

4) That's all we need from Course Marker, so now you can close it. Everything we do from here on out will involve *ConTEXT*.

5) Open ConTEXT via,
   Start>All Programs> Start>All Programs> EPS >EEE> ConTEXT

In the resulting window navigate to the folder:

P:>mchXXYY> EEEN10035u9exercises9

6) You should see the following files in the folder: Volt.java, TestVolt.java, SafeCheck.java, TestSafeCheck.java, Thanks.java and TestThanks.java.

<u>Before we move on ahead, a quick review of multiple classes.</u>

MULTIPLE CLASSES

So far, you have written programs with the following structure,

```
public class Anything {

        public static void main(String [ ] args)  {
            ……
            ……

        }

        public static void CalcAvg() {
            ……
            ……
        }

}
```

This is an example of a program containing a **single class** with **multiple methods.**

It is possible to have multiple classes (each having multiple methods) in a single program.
Observe the following program structure,

```
public  class Anything {
        public static void main(String [ ] args)  {
            ……
            ……

        }
}
```

```
public  class Calculator {
        public  void CalcMax ( ) {
            ……
            ……
        }
        public  void CalcMin ( ) {
            ……
            ……
        }
        public  void CalcAvg ( ) {
            ……
            ……
        }
}
```

This is an example of a program containing **multiple classes**. Each class is stored in a separate file and each can have **multiple methods.** The two files are: **Anything.java** and **Calculator.java**.

Note: There is only one main method in the whole program, i.e. only one of the classes can have a main method. Remember that every program starts executing from the main method.

The methods that do all the work in the program are delegated to the Calculator class. This is a common way of designing programs with multiple classes. Notice that *none* of the methods in **Calculator** need to be static.

The class Calculator above contains methods that might be useful in different programs. This means those methods don't have to be re-written over and over, and makes the class re-useable, an advantage often touted for object oriented programs.

Let's look at a working example now:

```
public class Anything  {

    public static void main (String [ ] args)  {
        //declare some data
        int  data [ ] = { 1, 2, 3 };
        Calculator s = new Calculator ( );
        System.out.println (s.getSum ( data ));
        s.getAvg ( data );
    }
}
```

The program always starts executing from the main.

To use the methods belonging to our second class **Calculator**, we need to declare a variable first.
s is the variable name here, and is more formally called an **object reference variable** or just **reference variable**. Think of it as a 'pointer' to an instance of Calculator. The word new creates the instance.

```
public  class Calculator {

        public  int  getSum ( int a [ ] ) {
            int sum = a [0] + a [1] + a [2];
            return sum;
        }
        public  void CalcAvg ( int a [ ] )  {
           int sum = a [0] + a [1] + a [2];
           System.out.println (sum/3);
        }

}
```

When run, the Output of this program will be:
6
2

We use this syntax each time we want to call a method from the class **Calculator**:

<reference variable name>.<method name> ( );

NOTES:

- There are variations possible. For example, it is possible, but unusual (except in graphical programs), to declare multiple the classes in a single file. By keeping them separate, the program is clearer and easier to maintain.
- Additional classes do not contain a main method.

Task 1:
Complete the Thanks.java such that the **printMessage** method prints out "Thank you for using this program!" in a new line. Complete TestThanks.java such that it creates an instance of Thanks using a variable name of your choice, and using the instance variable, call **printMessage**.

Task 2:
Open up SafeCheck.java, and complete it such that the **printSafe** method accepts as a single parameter an array of type double, and prints out "SAFE" if the sum of the values in the array is less than 30 ("UNSAFE" is otherwise). Test your solution by completing and running TestSafeCheck.java.

Task 3:
Complete Volt.java, by completing the method calcAvg, that calculates and returns the average value of a double array that is passed to it as a parameter. Then complete TestVolt.java such that the main prompts the user to enter 5 voltage readings into a double array, then creates an instance of Volt, and passes the array into the method CalcAvg to print out the average.

Task 4:
Create TestVolt2.java, which is based on TestVolt.java, such that the main declares an instance each of Volt, SafeCheck and Thanks. Accept 5 voltage readings into a double array as before. Modify the 'PrintSafe' method in SafeCheck.java to both print out, and return (as a String), "SAFE" or "UNSAFE". If "SAFE" is returned, call the CalcAvg method in Volt.java to print out the average voltage value. Finally call the 'PrintMessage' method in Thanks.java.

Task 5:
Write a new class called Utilities which has one method called PowerCalc ( ).This method should calculate the power, given the values of voltage (assume the current is 5A, and use the equation P = V*I). It should accept an array through parameters and *return* the result as an array. Write a class called TestUtilities to test it.

Task 6:
Create SafeCheck2.java, which is based on SafeCheck.java, such that it passes the voltage readings to the PowerCalc ( ) method of Utilities.java. Print "SAFE" only if *none* of the power values are above 30. Test by running TestSafeCheck.java.

END