Lab Practice Week 5

You need to show working versions of your answers to all questions to your tutor. Your tutor will expect to see them by your next session.

What to submit: your answers to exercises 1, 3, and 4.

For external students, you will need make arrangement with your tutor for submissions.

You should submit your Lab Practice work on a weekly basis to your tutor. Your tutor will expect to see them by your next session. Do not fall behind or leave it to end of the term as the workload will be very heavy towards the end of the term. Zero mark will be given for the week for late submission without acceptable reason.

Note: even though you only need to submit those exercises mentioned above, you should attempt all exercises in each lab practice to help broaden your understanding; this may also help with your assignment work.

Do all the programs in IDE.

NOTE: Include internal documentation in your code, if you are not sure, read chapter 2.4 in your textbook and talk to your lecturer

---

Exercise 1 below provides an opportunity to do more work with arrays in Java. Review the appropriate lecture material and textbook readings. The exercise also requires you to reinforce your skills in modular design and programming.

Exercises 2 and 3 will give you practice with class design and then testing your user-defined class. They also provide the opportunity to work with arrays that are used to store class objects. Think carefully about this concept. If we can encapsulate different data types (that represent an entity) in a class, we can then store multiple such class objects into a bulk storage structure (such as an array). This is not dissimilar to storing structs in an array, as you did in ICT159 second assignment.

Before starting the exercises, make sure you have read the relevant material.

1. Write a program to store an array of integer random numbers. Your program should find out from the user how many numbers to store. It should then generate and store that many random integers (the random numbers must be between 1 and 999 inclusive). The program should then determine the smallest number, the largest number, and the average of all the numbers stored in the array. Finally, it should print out all the numbers on the screen, five numbers to a line with spaces in between. Once the contents of the array have been printed to screen, display the smallest number, largest number, and average determined previously. You should ensure that your program design in modular. **<u>You should explain clearly whether you can use static variables and methods here, and why.</u>**

The Random class of Java library (java.util.Random) implements a random number generator. To generate random numbers, you construct an object of q the class Random, and then use the method nextInt(n) which returns a number between 0 (inclusive) and n (exclusive). Eg:

```
import java.util.Random;
.....
Random generator = new Random();
.....
//generate a random number between 0 and 99 (inclusive)
int nextRand = generator.nextInt(100);
```

Alternatively use Math.random(), which returns a random value
in the range 0.0 to 1.0 (then adjust to the correct range)

2. Create a UML diagram to help design the class described in exercise 3 below. Complete this exercise **before** you attempt to code the solution. Think about what instance variables will be required to describe a Baby class object; should they be private or public? Determine what class methods are required; should they be private or public?

3. Write Java code for a Baby class. A Baby has a name of type String and an age of type integer.

   Supply two constructors: one will be the default constructor, which just sets default values for the name and age; the second constructor will take two parameters, a string to set the name and an integer to set the age. Also, supply methods for setting the name, setting the age, getting the name and getting the age.

   The class should **not** contain I/O methods; use set method or constructor to set the value passed in as a parameter. The instance variables for output can be obtained using the get methods. The *set* method for the name instance variable should ensure that the value passed is not empty or contain whitespaces (otherwise set a default value). The *set* method for the age instance variable should validate the value passed to be between 1 and 4 inclusive (otherwise set a default value).

   Give Java code for an *equals* method for the Baby class. Babies count as being the same (i.e. equal) if their names and their ages are exactly identical (names should not be case sensitive). The method will take a Baby type parameter and use the calling object (thus comparing these two objects via name and age); it should return Boolean - true or false as appropriate. Remember, if comparing Strings, you must use String comparison methods.

4. Test your Baby class by writing a client program which uses an array to store information about 4 babies. That is, each of the four elements of the array must store a Baby object.

    If you have an array for baby names and another array for baby ages, then you have missed the point of the exercise and therefore not met the requirement of this exercise.

    A Baby class object stores the required information about a Baby. So each Baby object will have its own relevant information, and thus each object must be stored in one element of the array.

    The client program should:

    a. Enter details for each baby (name and age) and thus populate the Baby array
    b. Output the details of each baby from the array (name and age)
    c. Calculate and display the average age of all babies in the array
    d. Determine whether any two babies in the array are the same

    As the required information for these tasks is stored in the Baby array, you will need to use a loop to access each array element (and use the dot notation to access the appropriate set and get methods to assign/retrieve the information).

    For part d above, a nested loop is required.