Lab Practice Week 3

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

What to submit: your answers to exercises 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

---

Before starting the exercises, make sure you have read the relevant material for this and previous topics/labs.

In the following exercises, you will need to demonstrate how you can do the following and describe how you do them using the internal documentation.

Note: The example code given in the textbook and lecture slides may not have all the following, therefore you cannot just copy the style here, as they are there to teach some basics only.

- **Modularisation code**: not putting everything in one method; separate the computing and error checking from the main method.
- **Code reuse**: methods that can be reused; any code that cannot be reuse (for example interacting with users asking for input etc) should be in the client class.
- **Information hiding**: public instance variables vs private variables.
- **Any other software design principles used**: You should describe and explain any other software design principles you have used

1. Write a class called *Fraction*; this class will be designed to handle fractions. The *Fraction* class has only **two** instance variables:

    - An integer numerator
    - An integer denominator

The two instance variables can be publicly accessible (at this stage). The Fraction class should never store a value of 0 for the denominator.

Supply an output method for the Fraction class; this method will be used by the client to print out a fraction (as two separate integers) to the screen.

An output fraction would be in the form:

numerator / denominator

An example: the fraction for a half would be input as 1 for the numerator and 2 for the denominator. The output should be printed out as 1 / 2.

Once this is done, write a client class program called *TestFraction*; this class is to be used to test your *Fraction* class.

Note that the client program should use the *Fraction* class methods or instance variables (via the dot notation) to get the inputs and display the fractions. The client class should use a loop for getting fractions from the user and displaying them to the screen. For this exercise, stop when a negative value entered by the user for the numerator.

Write the pseudocode first before coding the class and the client program in Java.

2. This exercise requires you to ensure that the input values representing fractions are stored with denominators that are positive integers. You **cannot** require the user to only enter a positive denominator value; the user should not be inconvenienced by such a restriction. For example, whilst values of 1 / -2 are acceptable inputs for a fraction, the output representation should be -1 / 2. Your solution should check the denominator input; if it is negative, swap the sign of **both** numerator and denominator instance variables. Test this functionality with your *TestFraction* class.

3. Now change your *Fraction* class so that its instance variables (numerator and denominator) are hidden (i.e. private). Check that your client class (after this change) will now **not** compile. Why does it now not compile?

   Provide the changes to the *Fraction* class and the *TestFraction* class such that the program can work with the private instance variables by implementing appropriate **accessor** and **mutator** methods.

4. Provide the UML class diagram for your Fraction class after completing question 3 (refer lecture notes topic 2), and also provide the structure chart for the client class after completing question 3 (you should have learned how to do a structure chart in ICT159).