Lab Practice Week 4

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, 2, 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

---

This week you will continue to extend the work you have done in last lab exercise. You will need to demonstrate your understanding in the following concepts. You should read more about these from the lecture notes and textbook to understand the following concepts before doing the exercises. **After you have implemented the requirements of the exercises, use your implementation to explain the following concepts.**

- Object reference
- Information Hiding and Encapsulation
    - Precondition and Postcondition
    - Helper
    - Class interface

1. Add another **public** method called *add* to your *Fraction* class. This method adds another fraction to the 'calling object'. Thus, the method will take a *Fraction* class object as a parameter, add this parameter fraction to the calling object (fraction), and return a Fraction object as a result.

   HINT: We can use cross multiplication to determine the numerator of the resultant Fraction. The denominator of the resultant Fraction is simply the multiplication of the denominators of the two other Fractions.

2. Add a **private** method *simplify()* to your *Fraction* class that converts a fraction to its simplest form. For example, the fraction 20 / 60 should be stored in the class instance variables as 1 / 3 (i.e. numerator = 1, denominator = 3). N.B. you will need a method to determine the Greatest Common Divisor (GCD). Remember, both of these methods (*simplify* and *gcd*) **must be private**. As these methods are private, client programs cannot access them. So, how are they to be used? Given their purpose, it would mean that any *Fraction* class method that modifies the instance variables (e.g.: add, constructor, set) should call the *simplify()* method to reduce the instance variables to their minimum values. Thus, these methods are used only for *housekeeping*; they are **not** to be used by client programs.

   Make sure you understand how the following GCD algorithm work before implementing it.

```
// function gcd; two integer parameters
  int gcd(int n1, int n2)
    // local variable c
    int c
    loop while ( n1 != 0 AND n2 != 0 )
      c = n2
      n2 = n1 % n2 // modulus operator
      n1 = c
    return (n1 + n2)
```

3. Write a client class (*TestFraction2*) that allows the user to add up any number of non-zero fractions. The program should display the running total as an exact fraction (in simplified form as numerator / denominator). The user can finish by entering a fraction that represents a zero fraction.

   Note: If you need any implement any more methods (private or public) in the class to complete this question, you will need to justify and explain it clearly here.

4. Provide an updated UML class diagram to include all added behaviour of your *Fraction* class and also provide the structure diagram for your updated client class (*TestFraction2*).