CSCI 1933 Lab 2

Basic Java Programming

Lab Rules

You may work individually or with *one* partner in lab. We suggest that you have a TA evaluate your progress on each milestone before you move onto the next. The labs are designed to be completable by the end of lab. If you are unable to complete all of the milestones by the end of lab, you have **until the last office hours on Monday** to get those checked off by **any** of the course TAs. We suggest you get your milestones checked off as soon as you complete them since Monday office hours tend to become crowded. If you worked with a partner, both of you must be present when getting checked off to receive credit. You will only receive credit for the milestones you have checked off by a TA. There is nothing to submit to Moodle for this lab.

Attendance

Per the syllabus, students with more than 3 unexcused lab absences over the course of the semester will automatically fail the course. The TAs will take attendance during lab; it is expected that students will be actively working on the lab material. *Your physical presence in lab is not sufficient to be marked as present for the purposes of attendance.*

1 Testing the Complex Number Class

Write tests for the `Complex3` class included with the lab using the (expected value) - (actual value) method of testing. For example if you created a method `int mult(int a, int b)`, to test if it worked you would create a variable `actual` and a variable `expected` and compare them like so:

```
int actual = mult(2, 3);
int expected = 6;
boolean result = expected == actual;
System.out.println("TestMult: " + result);
```

Doing this allows for quicker evaluation of the tests, since you don't need to inspect the results any further than looking for any test that returned `false`.

> **Milestone 1:**
> Show a TA your tests for `Complex3`, then run them. Have a minimum of four different tests.

2 What's my GPA?

Command Line Input

Some lab steps this week will make use of command line inputs. Command line inputs are the arguments listed after the command line to run your Java program. Command line arguments that are interpreted by your Java program as Strings can be supplied as follows:

```
java MyProg argument1 argument2 argument3 ...
```

For example, if you entered the following into your program:

```
java MyProg 1 2 3 a b c
```

The array `args` from your main method would look like `["1", "2", "3", "a", "b", "c"]`. The values can be obtained by referencing `args[0], args[1], args[2]`..., respectively. Since all are Strings, if a scalar integer or double equivalent is needed for one of the arguments, it can be derived by the following:

```
Double.valueOf(args[0]);
```

In order to process all the arguments typed, the length of the array of String structure, `args`, can be found by `args.length`. Access the elements in the `args` array using a loop similar to the following example:

```
// access ith element of args like this: args[i]
for (int i = 0; i < args.length; i++){
    System.out.println(args[i]);
}
```

Calculating GPA

Students and advisors commonly need to find the GPA for specific groups of courses taken. Examples include finding the "technical GPA" or the GPA for prerequisite courses for a specific major.

Write a Java program that takes in letter grades and credit pairs from the command line and computes the GPA.

Use the following chart for grade point values:

Grade	Grade Points
A	4.0
A-	3.667
B+	3.333
B	3.0
B-	2.667
C+	2.333
C	2.0
C-	1.667
D+	1.333
D	1.0
F	0.0

Some examples follow:

```
java GPA a 4 b 4 a 2 b- 4
The GPA is: 3.33

java GPA a 4
The GPA is: 4

java GPA c- 4 b+ 4 a 3
The GPA is: 2.91

java GPA c 4 d 1 a 3 b+ 4 a 4
The GPA is: 3.145

java GPA a- 4 b+ 3 c- 2 d+ 1
The GPA is: 2.933
```

To calculate GPA use the following method:

$$\text{gpa} = \frac{\sum_i (\text{grade}_i \times \text{credit}_i)}{\sum_i \text{credit}_i}$$

For example, if you enter `java GPA a 4 b- 3 c+ 2` your calculation would be:

$$\text{gpa} = \frac{(4.0 \times 4) + (2.667 \times 3) + (2.333 \times 2)}{9} = 3.185$$

Milestone 2:
Run your `GPA` class from the command line using the examples shown above and show us your own tests. When can your solution fail?

3 Write a Rational Number Class

Create your own class called `Rational`. `Rational` should parallel Complex3. It will include a constructor, but no default constructor. It will have **get** and **set** methods for the numerator and denominator attributes. It will also include the following methods:

- `public void addRational(Rational r)`
- `public void subRational(Rational r)`
- `public void mulRational(Rational r)`
- `public void divRational(Rational r)`

that will perform the appropriate rational arithmetic. You do NOT need to reduce the result to lowest terms, however. Write tests for all of the methods in the `Rationals` class.

Reflection: What should the access modifier for your methods be? What shouldn't they be?

Milestone 3:
Show a TA your `Rational` class and run the accompanying tests.