Assignment 1

Handout: **Thursday, 3 October 2024**
Due: **14:00:00, Thursday, 17 October 2024**

Goals:

- To practice the use of Java data types and control structures.
- To understand the importance of information hiding.
- To design and implement Java classes.
- To get used to the IntelliJ Idea IDE.

NOTES

1. JDK 21[1] and IntelliJ IDEA Community Edition Version 2024.2[2] will be used in grading your assignments. Make sure you use the same versions of tools for your development.

2. For each class "XYZ" in the directory "src", you may right-click on it in the editor and select "Go to\Test" to view the sample tests we prepared for the class, and you can also right-click on the test class or the "test" folder and run the test(s). If any sample test fails, your implementation is buggy. Note, however, that since the sample tests only check a small number of input/output pairs, passing all those tests does NOT mean your implementation is correct. We use another (more comprehensive) set of JUnit tests to grade your code.

3. You may define additional methods when you see fit. However, you MUST NOT change the names of existing Java files or the signatures/return types of the existing methods in those files since the tests refer to the classes and methods based on their current declaration, and those changes will cause the tests to fail. Failed tests due to changes to the file names and/or method signatures/return types will be treated the same as failed tests due to incorrect implementations.

4. Your code may invoke methods defined in the String class, but no methods from the other library classes should be invoked. Refer to the Java documentation[3] for available String methods.

5. Full mark: 50

6. Please refer to LEC00 Course-info.pdf for the late submission policy.

[1] https://www.oracle.com/java/technologies/javase/jdk21-archive-downloads.html

[2] https://www.jetbrains.com/idea/download/other.html

[3] https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/String.html

1. Floating Point Vs. Integer Numbers (15 points)

Suppose we have a new data type called `TinyFloat`, which is like `float` but uses **only** eight bits. From left to right, the meaning of the bits is as follows:

- 1 bit for the sign: 0 for positive and 1 for negative.
- 4 bits in two's complement for the exponent.
- 3 bits for the mantissa.

Consider, for example, a `TinyFloat` object with the bit sequence `00100110`. The value is positive (`0`), the exponent is 4_{10} (= 0100_2), and the significand is 1.75_{10} (=1.110_2). Therefore, the whole value is $1.75 \times 2^4 = 28$.

What to do:

[Task 1] a. Complete the method binary2Integer to convert the string value to integer value for the exponent
b. Complete the method binary2Decimal to convert the string value to float value for the mantissa.
c. Complete the method TinyFloat.fromString based on the two methods above. The method takes a String of eight characters, each being '1' or '0', and returns the value of the corresponding TinyFloat object (as a float value);

[Task 2] Complete method TinyFloat.numberOfIntegers;
The method returns the number of all TinyFloat objects that are integers. When the method is called in TinyFloat.main, the result is printed as below. The last line XXXX shows the number of the integers. Do not change the code in TinyFloat.main.

00000000 == 1

00001000 == 2

00001100 == 3

…

XXXX

Note:

- The absolute value of a TinyFloat object is always calculated using formula 1.mantissa x $2^{exponent}$.
- Method getValidBitSequences returns all the 256 (=2^8) tinyFloats in String.

2. Rational Numbers (12 points)

> In mathematics, a rational number is any number that can be expressed as the quotient or fraction p/q of two integers, a numerator p and a non-zero denominator q. Since q may be equal to 1, every integer is a rational number.
>
> -- Wikipedia

Write a Java class for rational numbers. The class should have

1. two fields of type int, one for the numerator and the other for the denominator.
2. a constructor with two parameters, the numerator and denominator, by calling the setter methods.
3. four methods called add, subtract, multiply, and divide, respectively; Each method takes another rational number as the parameter, does the calculation using this and the parameter rational number, and returns the rational result. It is not necessary to simplify the results for these four methods.
4. a simplify method that simplifies this rational number. Each simplified rational number should satisfy the following three requirements:
    a. Common factors between the numerator and the denominator should be cancelled out; For example, 12/30 should become 2/5 after simplification.
    b. The denominator should always be positive, while the numerator could be positive, negative, or zero;

c. The denominator should always be 1 when the rational number is an integer.

- a toString method which returns the string representation of this in the form numerator/denominator. For example, given a simplified rational number has a denominator 1 and a numerator 3, the string format of the rational number should be 3/1.

Notes:

- You may assume the following when completing the class: 1) The constructor will never be used to instantiate a rational number with a denominator equal to 0; 2) The parameters of add, subtract, multiply, and divide will never be null; 3) The parameter of divide will never be equal to 0.

What to do: In Rational.java,

[Task 3] add the missing fields to the class Rational.

[Task 4] complete the constructor and the methods add, subtract, multiply, divide, simplify, and toString.

3. Complex Numbers (13 points)

> A complex number is a number that can be expressed in the form a + bi, where a and b are real numbers, and i is the imaginary unit, which satisfies the equation $i^2 = -1$. In this expression, a is the real part, and b is the imaginary part of the complex number.
>
> -- Wikipedia

Write a Java class for complex numbers with both the real and the imaginary parts of type Rational. The class should have

1. two fields of type Rational, one for the real part and the other for the imaginary part.
2. a constructor with two parameters, one for the real part and the other for the imaginary part. The set methods are used to set these two parameters.
3. four methods called add, subtract, multiply, and divide, respectively; Each method takes another complex number as the parameter, does the calculation using this and the parameter, and returns the result complex.
4. a simplify method that simplifies the real and imaginary parts of this.
5. a toString method which returns the string representation of this in the form (real, imaginary).

Notes:

- You may assume the following when completing the class: 1) The parameters of the constructor are never null; 2) The parameters of add, subtract, multiply, and divide will never be null; 3) The parameter of divide will never be equal to 0+0i or 0-0i.

What to do: In Complex.java

[Task 5] Add the missing fields to the class Complex.

[Task 6] Complete the constructor as well as the methods add, subtract, multiply, divide, and toString.

4. Balanced Brackets (10 points)

A non-empty `String` containing **only six characters**, namely '(', ')', '{', '}', '[', and ']', is called *balanced* if the brackets can be paired into "()"s, "[]"s, and/or "{}"s without changing their positions. For example, "()", "(){}", "({})", and "(()[{}])" are balanced, but "((", "({)}", and "(){" are not.

What to do: In `BalancedBrackets.java`,

[Task 7] complete method `isBalanced` in class `BalancedBrackets` so that the method returns `true` if and only if the argument `String` i) is non-empty, ii) contains only the six characters, and iii) is balanced.

What to hand in:

The whole **Assignment1** folder (including the completed methods) compressed into a ZIP file.