CSCB07 – Software Design

Lab 4

Objectives

- Getting familiar with object-oriented programming
- Learning how to use Javadoc to generate documentation

Logistics

- This lab will be supervised by your TA during the tutorial session of Week 6 (June 13 - 17, 2022). If you encounter any problem while doing the steps listed in the following sections, ask the TA for help.
- The lab should be done individually.

Instructions

1. Using Eclipse, create a new java project and add to it a package named **lab4**
2. Define class **Lab4Exception** as follows:
    a. It inherits from **Exception**
    b. It has one field of type **String** named **message**
    c. It has a constructor that takes one argument of type **String** and assigns it to **message**
    d. As you will see later on, instances of this class would be thrown at different locations of the code. Do not use try-catch blocks to handle them for now, just declare the exception in the headers of the involved methods.
3. Define an abstract class **SpecialNumber** as follows:
    a. It has an abstract method named **add** that takes one argument of type **SpecialNumber** and returns a **SpecialNumber**. This method is meant to add the calling object with the argument and return the result.
    b. It has an abstract method named **divideByInt** that takes one argument of type **int** and returns a **SpecialNumber.** This method is meant to divide the calling object by the argument and return the result.
    c. It has a concrete method named **computeAverage** that takes one argument of type **List<SpecialNumber>** and returns the average of its elements using **add** and **divideByInt**. If the list is null or empty, the method throws a **Lab4Exception** with the message "List cannot be empty"
4. Define class **RationalNumber** as follows:
    a. It has two fields of type **int** named **numerator** and **denominator**
    b. It has a constructor that takes two arguments of type **int** and initializes **numerator** and **denominator** accordingly. If the argument corresponding to the denominator is zero, the constructor should throw a **Lab4Exception** with the message "Denominator cannot be zero"

c. It inherits from **SpecialNumber**
    i. When implementing **add**, you need to make sure that the argument being added is an instance of **RationalNumber**. Otherwise, a **Lab4Exception** should be thrown with the message "Cannot add an incompatible type"
    ii. When implementing **divideByInt**, you need to make sure that the argument is not zero. Otherwise, a **Lab4Exception** should be thrown with the message "Cannot divide by zero"
d. It implements **Comparable**
e. It overrides **equals** and **hashCode**. Note that two rational numbers could be equal without having the same numerators or denominators (e.g. 1/2 and 2/4)

5. Define class **ComplexNumber** as follows:
    a. It has two fields of type **double** named **real** and **imaginary**
    b. It has a constructor that takes two arguments of type **double** and initializes **real** and **imaginary** accordingly
    c. It inherits from **SpecialNumber**
        i. When implementing **add**, you need to make sure that the argument being added is an instance of **ComplexNumber**. Otherwise, a **Lab4Exception** should be thrown with the message "Cannot add an incompatible type"
        ii. When implementing **divideByInt**, you need to make sure that the argument is not zero. Otherwise, a **Lab4Exception** should be thrown with the message "Cannot divide by zero"
    d. It implements **Comparable**. Complex numbers are to be compared using their magnitudes (i.e. $\sqrt{real^2 + imaginary^2}$)
    e. It overrides **equals** and **hashCode**
6. Test your code using the JUnit tests provided in **Lab4Tests.java**. All tests should pass.
7. Add doc comments to your code to be able to generate HTML documentation later on
    a. Doc comments begin with **/**** and end with ***/**
    b. A doc comment precedes a class, field, or constructor/method declaration. It includes a description followed by block tags (e.g. @param, @return). More information regarding doc comments could be found at the following link: https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html
    c. Add doc comments for class **RationalNumber** and method **computeAverage** in **SpecialNumber**. For the latter, make sure to use @param, @return, and @throws
    d. For example, the comments for method **compareTo** in class ComplexNumber.java could be as follows:

```
/**
* This method compares two ComplexNumber objects
* @param anotherComplexNumber the complex number to be compared
* @return -1 if anotherComplexNumber is less than this ComplexNumber, 0 if they are
*  equal, and 1 otherwise
*/
```

8. Generate documentation for your project in HTML format as follows:
"Project" -> "Generate Javadoc" -> Select "Public" visibility and choose destination -> "Finish"

Submission

Upload the four java files and the Javadoc HTML files as a single archive file to “Lab 4” on Quercus by June 20th at the latest.