CSCB07 – Software Design

Lab 3

Objectives

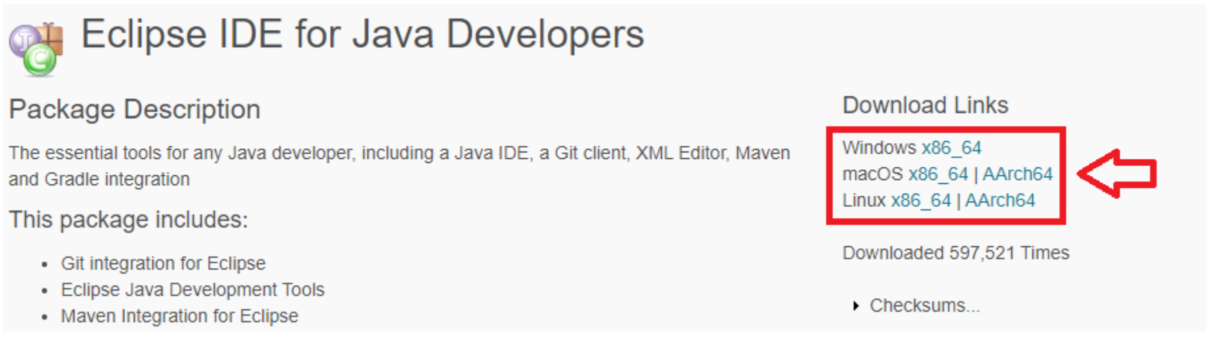
- Applying object-oriented programming principles
- Getting familiar with unit testing
- Learning how to use Javadoc to generate documentation

Logistics

- This lab is worth 2% and it will be supervised by your TA during the tutorial sessions of Weeks 5 and 7.
- If you encounter any problem while doing the steps listed in the following sections, ask the TA for help.
- Attendance will be taken during the tutorial. If you are unable to attend any of the two sessions in Weeks 5 & 7, please send your TA an email explaining why and make sure to submit the deliverables by the due date. Failing to do so might result in a 10% penalty.
- The lab should be done individually.
- The due date is Oct 22, 2023.

Installing Eclipse

1. Download Eclipse IDE for Java Developers using the following link. The package is already equipped with a JRE.
   https://www.eclipse.org/downloads/packages/release/2023-03/r/eclipse-ide-java-developers



Eclipse IDE for Java Developers

Package Description

The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Maven and Gradle integration

This package includes:

- Git integration for Eclipse
- Eclipse Java Development Tools
- Maven Integration for Eclipse

Download Links

Windows x86_64
macOS x86_64 | AArch64
Linux x86_64 | AArch64

Downloaded 597,521 Times

Checksums...

2. Unzip the folder and double click "eclipse.exe"
3. Select a directory as workspace and click "Launch"
4. To create a new Java project: "File" -> "New" -> "Java Project" -> Provide a project name, un-check "Create module-info.java file", and click "Finish"

N.B. For Mac users, the process is similar except that the package is downloaded as a dmg file.

Instructions

1. Using Eclipse, create a new java project and add to it a package named **lab3**
2. Define class **Lab3Exception** 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 **Lab3Exception** 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 **Lab3Exception** 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 **Lab3Exception** 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 **Lab3Exception** 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 **Lab3Exception** 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 **Lab3Exception** 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 **Lab3Tests.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 (.rar or .zip) to “Lab 3” on Quercus.