CSCI 5308 Assignment 2

Due date: 11:59pm, Friday, Jul 16, 2021, submitted via Git

Objectives

- Practice procedural refactoring.
- Practice implementing good/quality code.
- Practice SOLID principles.

Preparation:

Clone the code on the repository https://git.cs.dal.ca/courses/2021-summer/csci-5308/assignment2/ronnie

Problem Statement

You have three main tasks in this assignment:

- Refactor the code using what you learnt from lectures 6 and 7.
- Increase code quality using the Java guidelines provided.
- Refactor the code by applying SOLID principles.
- Use exceptions wherever is necessary.

Task

1. [**30 marks**] Identify three (3) procedural refactoring opportunities that can be done in the code using the guidelines. Provide:
   a) A brief description of each issue that require refactoring
   b) Fix the issue.
   c) Test your fix (when applied).

2. [**60 marks**] Identify three (3) refactoring that can be done in the code by applying SOLID Principles. Give:
   a) a brief description of each issue.
   b) where the issue is.
   c) what SOLID principle is violated.
   d) how to fix the issue.
   e) Apply your fix.
   f) Test your fix.

3. **[10 marks]** Identify three (3) methods requiring exceptions. Give:
   a) method where exception should be used.
   b) what the exception should be used.
   c) Add the exception to the code.

4. Push back your work to **YOUR** remote repository, not to the one that you cloned. Everyone has their own repository to deliver this assignment, e.g., https://git.cs.dal.ca/courses/2021-summer/csci-5308/assignment2/yourscid.

5. Remember to check that all your files have been submitted using the web interface to git.

Submission

1. Edit this document with your written answers
2. All the code (fixed) should be committed and pushed back to your remote Git repository. **Important! Use individual commits to each answer.**

Answers

Add your written answers here.

Dhrumil Rakesh Shah
B00870600

Answers:

The very first step I performed was setup the project JDK which was not defined in IntelliJ. So, I set the project JDK to JAVA Version 15.

Task 1

Formatting:

1. Inserted a blank line after class declaration.
2. Removed extra spaces around the variables and operators.
3. Added spaces around the variables and the operators.
4. Moved the return statement to the next line.
5. Removed any extra blank lines.
6. Added blank lines after each variable (field) declaration.

Fix Test:

1. No errors faced after fixing the above issues.

Refactoring the naming:

1. Changed the variable names from generic to more specific and readable names.
    a. Like changed x → numElementLeftArray
    b. Like changed y → numElementRightArray
2. Changed the name of the arrays.
    a. Like changed E[] → Element[]
    b. Like changed E → Element
3. Changed the name of the Element object to keep it uniform throughout the application
    a. Like changed Element e → Element element

4. Changed the name of the Object variables to keep it uniform throughout the application
    a. Like changed Object o → Object object
5. Changed the variable name for better readability.
    a. Like changed i → index
6. Changed the local variable name as it was similar to that of the parameter passed
    a. Like changed Element e → Element elementNumber
7. Changed the String variable name
    a. Like changed String s → String string
8. Changed the local variable name
    a. Like changed int sz → int sizeOfLeftAndRightArray

Fix Test:

1. No errors faced after fixing the above issues.

Refactoring:

1. Added a missing Object[] initialization for leftArray in indexableList.java file.
2. Removing magic constants inside the add method.
    a. Using a constant to store the value of 1000.
    b. Using a constant to store the value of -3.
3. Data level refactoring done in the add() and remove() method in IndexableList.java file.
    a. A local variable int size is declared storing the size returned by the size() method as below.
    b. int halfSize = ((size() + 1) / 2);
4. Also made a local variable named size that stores the value returned by the size() method.
    a. int size = size();

Fix Test:

1. No errors faced after fixing the above issues.

Commenting the code:

1. Main file comments for classes and interfaces.
2. Individual methods comments.
3. Inline comments for the variables declared.
4. Constructor comments.
5. Test file comments added.

Task 2

Applying SOLID Principles

1. **Issue I:**
    a. Both the classes IndexableList and OrderedIndexableList violates the **Open/Closed principle** of the SOLID principles.
        i. Both the classes use implementation of ArrayList instead of List.
        ii. So, if I want to pass any other type of List to the class, then I won't be able to do it without modifying the code.
2. **Fix:**

a. To fix the Open/Closed principle, I replaced the ArrayList with List, leaving room for using other lists if required.

3. **Test:**
    a. Replacing the ArrayList with List did not throw any error, except under the subList() method a new ArrayList is instantiated as a List is abstract so it cannot be instantiated.

1. ***Issue II:*** ***Removing this change as it creates duplicate code***
    a. *Both the classes IndexableList and OrderedIndexableList violates the* ***Single Responsibility principle*** *of the SOLID principles.*
    b. *Both classes have two parameterized constructors*
        i. *One implementing List*
        ii. *Another implementing an Array*

1. **Issue III:**
    a. The extendLeftArray() and extendRightArray() methods in IndexableList class violate the Dependency Inversion principle of the SOLID principles.
    b. They should not be accessible outside the respective class, as the size of the array should only be extended when the array is full from within the class itself.
2. **Fix:**
    a. The fix for the Dependency Inversion principle violation is by making the extendLeftArray() and extendRightArray() methods private so they won't be accessible outside their respective class.
3. **Test:**
    a. After fixing the Dependency Inversion principle violation, no errors were faced in the code.

1. **Issue IV:**
    a. All the classes violate the Interface Segregation principle of the SOLID principles.
    b. There are no interfaces whatsoever that are being implemented by the classes.
    c. No entry point and segregation for the methods being implemented.
2. **Fix:**
    a. The fix for the Interface Segregation principle is by making separate interfaces for the Indexable class and OrderedIndexable class.
3. **Test:**
    a. After fixing the Interface Segregation principle violation, no errors were faced in the code.

4. **Issue V:**
    a. The class IndexableList has a method toScanner() which is not related to the functionalities of the class.
    b. So, here the Single Responsibility principle of the SOLID principles is violated.
5. **Fix:**
    a. The fix for the Single Responsibility principle being violated is by creating a separate class containing the toScanner() method which will be used by the class.

b. A sperate Interface is also made for the ScannerClass, which also fixes the Interface Segregation principle.

6. **Test:**
    a. After fixing the Single Responsibility principle violation, no errors were faced in the code.

Task 3

Exceptions:

1. The add() method in IndexableList class can throw an ArrayIndexOutOfBoundsException and a NullPointerException.
    a. A try-catch block is used to handle the ArrayIndexOutOfBoundsException and a NullPointerException caused by the add() method.

2. The extendLeftArray() method in IndexableList class can throw an ArrayIndexOutOfBoundsException and a NullPointerException..
    a. A try-catch block is used to handle the ArrayIndexOutOfBoundsException and a NullPointerException caused by the extendLeftArray () method.

3. The extendRightArray () method in IndexableList class can throw an ArrayIndexOutOfBoundsException and a NullPointerException.
    a. A try-catch block is used to handle the ArrayIndexOutOfBoundsException caused by the extendRightArray () method.

4. The contains() method in IndexableList class can throw a NullPointerException.
    a. A try-catch block is used to handle the NullPointerException caused by the contains() method.

5. The get() method in IndexableList class can throw an ArrayIndexOutOfBoundsException and a NullPointerException.
    a. A try-catch block is used to handle the ArrayIndexOutOfBoundsException and a NullPointerException caused by the get() method.

6. The indexOf() method in IndexableList class can throw an ArrayIndexOutOfBoundsException and a NullPointerException.
    a. A try-catch block is used to handle the ArrayIndexOutOfBoundsException and a NullPointerException caused by the indexOf () method.

7. The remove() method in IndexableList class can throw an IndexOutOfBoundsException.
    a. A try-catch block is used to handle the IndexOutOfBoundsException caused by the remove() method.

8. The set() method in IndexableList class can throw an ArrayIndexOutOfBoundsException and an NullPointerException.

    a. A try-catch block is used to handle the ArrayIndexOutOfBoundsException and an NullPointerException caused by the set() method.

9. The subList() method in IndexableList class can throw an IndexOutOfBoundsException.
    a. A try-catch block is used to handle the IndexOutOfBoundsException caused by the subList() method.

10. The addElement() method in OrderedIndexableList class can throw an ArrayIndexOutOfBoundsException and a NullPointerException.
    a. A try-catch block is used to handle the ArrayIndexOutOfBoundsException and a NullPointerException caused by the addElement() method.