2021

CSCI5308 Assignment 2

PROCEDURAL REFACTORING, SOLID REFACTORING AND EXCEPTIONS

DHRUMIL AMISH SHAH (B00857606)

CSCI 5308 Assignment 2

Task 1 – A list of procedural refactorings done in the code

1. Improved the formatting of classes **IndexableList** in file **IndexableList.java** and **OrderedIndexableList** in file **OrderedIndexableList.java** according to the formatting guidelines specified in the **JavaStyleGuide.pdf** file. Below is a list of steps performed:
    a. Removed the extra space between the import statement and the class declaration in **OrderedIndexableList** class.
    b. Indented the **OrderedIndexableList** code to follow two spaces indentation.
    c. Removed the space between the class comment and class declaration in class **IndexableList**.
    d. Removed extra spaces in class **OrderedIndexableList**.
    e. Single line inserted(wherever not present) in both classes **IndexableList** and **OrderedIndexableList** after:
        i. Class declaration(i.e., single verticle white-space after class declaration)
        ii. Member declaration(i.e., single verticle white-space after class member)
        iii. Constructor declaration(i.e., single verticle white-space after constructor)
        iv. Method declaration(i.e., single verticle white-space after each method)
    - Guideline followed – **Formatting** guideline in JavaStyleGuide.pdf

2. Refactored the variable name "x" to "leftNumElements" and "y" to "rightNumElements" in class **IndexableList** according to the naming guidelines specified in the **JavaStyleGuide.pdf** file. Class member variable names must be meaningful. Variable names "x" and "y" conveyed no meaning. Hence they were changed to "leftNumElements"(from x) and "rightNumElements"(from y).
    - Rule violated – One-character names should not be used except as indexes in loops.
    - Fixed – By changing the variable names to more meaningful names.

    The variables leftNumElements, rightNumElements, leftArray, and rightArray in class **IndexableList** are marked protected. Further, the class **OrderedIndexableList** extends **IndexableList** and has the same variables with private access scope. These variables in class **OrderedIndexableList** are of no use since the class **OrderedIndexableList** already extends the **IndexableList** and hence can use the protected variables of the class **IndexableList**. Thus, private variables leftNumElements, rightNumElements, leftArray, and rightArray (all four variables) are removed from class **OrderedIndexableList**.
    - Guideline followed – **Naming** guideline in JavaStyleGuide.pdf

3. Added the remaining comments to improve the readability of the code **IndexableList** and **OrderedIndexableList** according to the guidelines specified in the **JavaStyleGuide.pdf** file.
    - Guideline followed – **Commenting** guideline in JavaStyleGuide.pdf

4. Variable names in methods(Local variable names) were changed according to the naming guidelines specified in the **JavaStyleGuide.pdf**. Below is a list of changes done:

File – **OrderedIndexableList.java**

a. Method – add(E e)
   Changed variable name "i" to "insertPosition".

File – **IndexableList.java**

b. Method – remove(Object o)
   Changed variable name "i" to "removePosition".
c. Method - set(int index, E element)
   Changed variable name "e" to "prevElement".
d. Method – toScanner()
   Changed variable name "s" to "elementsString".

- Guideline followed – **Naming** guideline in JavaStyleGuide.pdf

Task 2 – A list of SOLID principles refactoring done in the code

1. **A brief description of each issue**
   - Class OrderedIndexableList was tightly coupled and, due to that, it violated the Interface Segregation Principle(ISP). Further, its constructor accepted the ArrayList implementation as an argument instead of the general List interface. Since ArrayList was the only accepted argument, other list implementations like LinkedList and Vector cannot be passed even they all implement the List interface. Thus, it violates the Dependency Inversion Principle(DIP).

   Where the issue is
   - In file OrderedIndexableList.java (Class OrderedIndexableList)

   What SOLID principle is violated
   - Interface Segregation Principle(ISP) and Dependency Inversion Principle(DIP)

   How to fix the issue
   - Create a new interface OrderedList (File OrderedList.java). This interface provides a contract to implement a functionality to add elements in a logical order based on the Comparable provided. Class OrderedIndexableList implements the interface OrderedList and overrides the method to provide custom logic to add elements in order and thus resolves the ISP.

   - Instead of passing ArrayList implementation in OrderedIndexableList, pass the List implementation. Doing so allows the passing of different list implementations at run-time. Thus, DIP is resolved.

   Apply the fix
   - File OrderedList.java and OrderedIndexableList.java

   Test your fix
   - File Tests.java contains method testSOLIDPrinciple1Refactoring() to test this issue.

2. **A brief description of each issue**
   - Similar to issue 1, class IndexableList was tightly coupled and, due to that, it violated the Interface Segregation Principle(ISP). Further, its constructor accepts the ArrayList implementation as an argument instead of the general List interface. Since ArrayList was the only accepted argument, other list implementations like LinkedList and Vector cannot be passed even they all implement the List interface. Thus, it violates the Dependency Inversion Principle(DIP). Also, its method subList(int fromIndex, int toIndex) returned ArrayList implementation, which was again specific. Therefore, it also violates the DIP.

   Where the issue is
   - In file IndexableList.java (Class IndexableList)

   What SOLID principle is violated
   - Interface Segregation Principle(ISP) and Dependency Inversion Principle(DIP)

   How to fix the issue
   - Create a new interface, IndexedList (File IndexedList.java). This interface provides a contract to implement a functionality to add, delete, update elements. Class IndexableList implements the interface OrderedList and overrides all the methods to provide custom logic to manage elements and resolves the ISP.

   - Instead of passing ArrayList implementation in IndexableList, pass the List implementation. Doing so allows the passing of different list implementations at run-time. Similarly, instead of returning ArrayList, return List to resolve the DIP.

   Apply the fix
   - File IndexedList.java and IndexableList.java

   Test your fix
   - File Tests.java contains method testSOLIDPrinciple2Refactoring() to test this issue.

3. **A brief description of each issue**
   - The toScanner() method in the IndexableList class is responsible for returning the elements of the list in a String representation wrapped by the Scanner class. This method has no relation with the class IndexableList. The class IndexableList is only responsible for managing the list and performing operations (i.e., insert, delete, set, etc…) on the list.

   Where the issue is
   - With toScanner() in class IndexableList.

   What SOLID principle is violated
   - Having method toScanner() in class IndexableList violates the Single Responsibility Principle(SIP).

How to fix the issue

- Create a new class IndexedListPrinter and move the method toScanner() to this class from IndexableList class. Create a constructor in class IndexedListPrinter and pass the IndexableList object in the constructor and use the dot(.) operator to call methods of the class IndexableList.

- Remove the Scanner class import from the IndexableList class and add it to IndexableListPrinter.

Apply the fix

- File IndexedListPrinter.java contains the fix.

Test your fix

- File Tests.java contains method testSOLIDPrinciple3Refactoring () to test this issue.

Task 3 – Methods requiring exceptions

Table 1: Exceptions in class OrderedIndexableList

Methods where exception should be used	What exception should be used
public boolean add(E e)	NullPointerException

Table 2: Exceptions in class IndexableList

Methods where exception should be used	What exception should be used
public E get(int index)	ArrayIndexOutOfBoundsException
E set(int index, E element)	ArrayIndexOutOfBoundsException

Test exceptions

- File Tests.java contains method testExceptions() to test this issue.