CSC 151 Programming Assignment 1

Objectives:

- Comparing Arrays and LinkedLists
- Implementing Bags with LinkedLists(Doubly LinkedLists)
- Initializing collections from ArrayLists and files
- Using Generics
- Using Interfaces

Definition

In this assignment you will implement various operations on doubly linked list bags and use various types to create and call the methods of the linked list bags. You will use your data structures to implement a spell checker. You will be given a list of correct spellings and another list of words that should be checked. You will compare those words and find the correct and incorrect spellings.

The interfaces and the classes in this assignment are partially or completely provided in your textbook. Please complete the partial implementations and submit all files listed in the submission instructions.

Interface BagInterface

- Use the same code provided at the end of the previous assignment.
- It will be in **package assignment** (it is required otherwise **Gradescope** cannot run the tests)

Class DoublyLinkedBag

- Use the partial code provided at the end of the document.
- It will be in **package assignment** (it is required otherwise **Gradescope** cannot run the tests)
- See the details below written as comments and complete the partial code as instructed.
- Test your class in the main method.
- Test your class by JUnit

Class SpellCheckerDoubly

- Use the partial code provided at the end of the document.
- It will be in **package assignment** (it is required otherwise **Gradescope** cannot run the tests)
- See the details below written as comments and complete the partial code as instructed.
- Test your class in the main method.
- Test your class by JUnit

General Submission Instructions

- You may have groups for the assignments **if it is allowed.** If you have a group state it **explicitly in the honor code** by writing all group members' names. Please name the pdf files accordingly. **However, all group members should submit individually.**

- All submissions require a package called **assignment** and your source codes (.java files) will be under it.
- You should submit
  - Your java files to **Gradescope** as required and
  - Your java files and sample outputs for various scenarios as a **single pdf** file named as **YourFullName(s)_AssignmentN.pdf** with the **honor code** at the top as a comment to Nexus under the appropriate link. Replace N in AssignmentN with the appropriate value. (e.g., if I submit Assignment1 I will submit a pdf file whose name is **ZeynepOrhan_Assignment1.pdf.** If I have a group whose members are me and Jane Doe, then the file name will be **JaneDoeZeynepOrhanAssignment1.pdf** and the honor code will be modified). Use the template .docx file attached and modify it accordingly.
- **If you use any resource to get help, please state it in the honor code. Otherwise, it will be an honor code violation!!!**
- **Gradescope provides a feature to check code similarity by comparing all submissions. If the percentage of your code's similarity is above a threshold, then this will be an honor code violation!!!**
- Make sure that you have no compiler errors.
- When you have compiler error free codes and submit appropriately to Gradescope, your code will be tested automatically and graded. You do not need to know the test cases, but you will see the test results. Informative messages will be provided for success and failure cases.
- You can resubmit your files any number of times if you have failing tests until the due date.
- Your final submission or the one you selected will be graded after the due date of the assignment.
- Please start as early as possible so that you may have time to come and ask if you have any problems.
- Read the late submission policy in the syllabus.
- Please be aware that the correctness, obeying the OOP design and submission rules are equally important
- Gradescope testing will generally be 100% and sometimes manual grading will be done. Please be aware that **your grade can be reduced if you do not follow the instructions and apply good programming practices**.
- Use the starter code if provided.
- Do not use any predefined Collection classes of Java unless stated otherwise.

<u>**Assignment specific instructions**</u>

- General submission instructions apply.

- **Group work: ALLOWED(You may have a group of 2-3 students)!!! However, you should submit the same files individually!!!**
- **Due date:** January 30, 2021, until 11:50 PM
- **Nexus:** Submit the code files (**BagInterface.java, DoublyLinkedBag.java, SpellCheckerDoubly.java**) as a single pdf file named as **YourFullName_Assignment4.pdf** with the honor code at the top as a comment under Programming Assignment 1 link.
- **Gradescope:** Submit the java files (**BagInterface.java, DoublyLinkedBag.java, SpellCheckerDoubly.java**) under the Programming Assignment 1
- **Grading:**
  - Gradescope: Yes
  - Manual: Yes
- **Starter code:** Use the code given below

```java
package assignment;
/*
 * Honor code
 */

import java.util.StringJoiner;

/**
 * DoublyLinkedBag class: A class of bags whose entries are stored in a chain of doubly linked nodes.
 * The bag is never full.
 *
 * @author Frank M. Carrano
 * @version 5.0
 */
public class DoublyLinkedBag<T> implements BagInterface<T> {
        private DoublyLinkedNode firstNode; // Reference to first node
        private int numberOfEntries;

        /**
         * Parameterless constructor that initializes this Bag.
         *
         *
         */




         /**
         * add: Adds a new entry to this bag.
         *
         * @param newEntry The object to be added as a new entry
         * @return True if the addition is successful, or false if not.
         */




        /**
         * toArray: Retrieves all entries that are in this bag.
         *
         * @return A newly allocated array of all the entries in this bag.
         */





        /**
         * isEmpty: Sees whether this bag is empty.
         *
         * @return True if this bag is empty, or false if not.
         */




        /**
         * getCurrentSize: Gets the number of entries currently in this bag.
         *
         * @return The integer number of entries currently in this bag.
         */




         /**
         * getFrequencyOf: Counts the number of times a given entry appears in this bag.
         *
         * @param anEntry The entry to be counted.
         * @return The number of times anEntry appears in this bag.
         */




         /**
         * contains: Tests whether this bag contains a given entry.
         *
         * @param anEntry The entry to locate.
         * @return True if the bag contains anEntry, or false otherwise.
         */




        /** clear: Removes all entries from this bag. */



        /**
         * remove: Removes one unspecified entry from this bag, if possible.
         *
         * @return Either the removed entry, if the removal was successful, or null.
         */


        /**
         * remove: Removes one occurrence of a given entry from this bag, if possible.
         *
         * @param anEntry The entry to be removed.
         * @return True if the removal was successful, or false otherwise.
         */





        /**
         * union: Creates a new bag that combines the contents of this bag and anotherBag.
         *
         * @param anotherBag The bag that is to be added.
         * @return A combined bag.
         */



        /**
         * intersection: Creates a new bag that contains those objects that occur in both this bag and
         * anotherBag.
         *
         * @param anotherBag The bag that is to be compared.
         * @return The intersection of the two bags.
         */




        /**
         * difference: Creates a new bag of objects that would be left in this bag after removing
         * those that also occur in anotherBag.
         *
         * @param anotherBag The bag that is to be removed.
         * @return The difference of the two bags.
         */




        /**
         * toString: Convert this bag to a String for displaying.
         * each item will be comma separated and a space after comma enclosed in [ and ]
         * if we have a b c d in the bag a is the most recent one and will be converted as
         * [a, b, c, d]. StringJoiner is a good option to use.
         */




        // private inner class DoublyLinkedNode:
        // A class of nodes for a chain of doubly linked nodes.
        private class DoublyLinkedNode {
                private T data; // Entry in bag
                private DoublyLinkedNode next; // Link to next node
                private DoublyLinkedNode previous; // Link to previous node
```

```
                 // private constructor of class DoublyLinkedNode with a data parameter



                 // private constructor of class DoublyLinkedNode with a data, nextNode and previousNode parameters



                // get and set methods for DoublyLinkedNode class









        } // end DoublyLinkedNode
} // end DoublyLinkedBag
```

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.*;

/**
 * SpellCheckerDoubly class: A spelling checker class.
 *
 * @author Frank M. Carrano
 * @author zorhan modified
 * @version 5.0
 */
public class SpellCheckerDoubly {
        BagInterface<String> correctWords; // Storage for the correct words

         /**
         * Parameterless constructor that initializes this Bag.
         * correctWords should be a new DoublyLinkedBag of String
         *
         */




        /**
         * setDictionary: Initializes the dictionary of correctly spelled words from a file.
         *
         * @param fileName A File name as a String that represents a text file of correctly
         *                 spelled words, one per line.
         * @return True if the dictionary is initialized, if not returns false.
         */




        /**
         * setDictionary: Initializes the dictionary of correctly spelled words from a list.
         *
         * @param correctWordsList An ArrayList list of String that contains correctly spelled words.
         *
         */




        /**
         * setDocument: Initializes the document to be checked from a file.
         *
         * @param fileName A File name as a String that represents a text file of the document
         *             whose words will be checked, one per line.
         * @return DoublyLinkedBag(BagInterface) of document's words.
         */
```

```java
	/**
	 * setDocument: Initialises the document to be checked from a list
	 *
	 * @param wordList An ArrayList of String that contains words to be checked.
	 * @return DoublyLinkedBag(BagInterface) of document's words.
	 */



	/**
	 * checkSpelling: Checks the spelling of a given single word as String.
	 *
	 * @param aWord A string whose spelling is to be checked.
	 * @return True if the given word is spelled correctly, otherwise returns false.
	 */



	/**
	 * checkBagSpelling: Checks the spelling of a given bag of words
	 *
	 *
	 * @param wordBag   A bag of words whose spelling is to be checked.
	 * @param correct   A bag of the words in wordBag whose spellings are correct
	 * @param incorrect A bag of the words in wordBag whose spellings are incorrect
	 */


	/**
	 * checkSpellingFromLists: Checks the spelling of a given bag of words
	 * from the given list of words list and dictionary list.
	 *
	 * @param dictList  A list of dictionary words
	 * @param wordList  A list of words to be checked
	 * @param correct   A bag of the words in wordBag whose spellings are correct
	 * @param incorrect A bag of the words in wordBag whose spellings are incorrect
	 */



	/**
	 * checkSpellingFromFile: Checks the spelling of a given bag of words by
	 * creating the words list and dictionary from the files.
	 *
	 * @param dictFile  A file name of dictionary words
	 * @param wordFile  A file name of words to be checked
	 * @param correct   A bag of the words in wordBag whose spelling are correct
	 * @param incorrect A bag of the words in wordBag whose spelling are incorrect
	 */



	 // Sample main
	public static void main(String[] args) {

		String[] words = { "a", "b", "c", "d", "e" };
		ArrayList<String> wordList = new ArrayList<>();
		Collections.addAll(wordList, words);

		String[] correctWords = { "a", "b", "cc", "d", "e" };
		ArrayList<String> dict = new ArrayList<>();
		Collections.addAll(dict, correctWords);

		SpellCheckerDoubly spD = new SpellCheckerDoubly();
		BagInterface<String> correct = new DoublyLinkedBag<>();
		BagInterface<String> incorrect = new DoublyLinkedBag<>();
		spD.checkSpellingFromLists(dict, wordList, correct, incorrect);

		System.out.println("All words to be checked " + wordList);
		System.out.println("All dictionary words " + dict);
		System.out.println("Correctly spelled" + correct);
		System.out.println("Incorrectly spelled" + incorrect);

		SpellCheckerDoubly spD2 = new SpellCheckerDoubly();
		BagInterface<String> correct2 = new DoublyLinkedBag<>();
		BagInterface<String> incorrect2 = new DoublyLinkedBag<>();
		spD2.checkSpellingFromFile("src\\assignment\\tests\\dict", "src\\assignment\\tests\\words", correct2,
                                  incorrect2);

                System.out.println("Correctly spelled" + correct2);
                System.out.println("Incorrectly spelled" + incorrect2);

        }

}
```

Output

```
All words to be checked [a, b, c, d, e]
All dictionary words [a, b, cc, d, e]
Correctly spelled[a, b, d, e]
Incorrectly spelled[c]
Correctly spelled[b, c, e]
Incorrectly spelled[aa, df, z]
```

Content of the dict file

```
a
b
c
d
e
f
g
```

Content of the words file

```
aa
b
c
df
e
z
```