CSC 151 Assignment 4

Objectives:

- Implementing Bags with LinkedLists
- Using Generics
- Using Interfaces

Definition

In this assignment you will implement various operations on linked list bags and use various types to create and call the methods of the linked list bags.

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 LinkedBag

- 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 LinkedBagDriver

- 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.

Assignment specific instructions

- General submission instructions apply.
- **Group work: NOT ALLOWED!!!**
- **Due date:** January 30, 2021 until 11:50 PM

- **Nexus:** Submit the code files (**BagInterface.java, LinkedBag.java, LinkedBagDriver.java**) as a single pdf file named as **YourFullName_Assignment4.pdf** with the honor code at the top as a comment under Assignment 4 link.
- **Gradescope:** Submit the java files (**BagInterface.java, LinkedBag.java, LinkedBagDriver.java**) under the Assignment 4
- **Grading:**
    - Gradescope: Yes
    - Manual: No
- **Starter code:** Use the code given below

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

import java.util.StringJoiner;

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

        /**
         * 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.
         */



        /**
         * 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.
         */
```

```java
        /** clear: Removes all entries from 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.
         */



        /**
         * 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.
         */



        // getReferenceTo: Should be private so not written in Javadoc format
        // parameter is an entry of type T
        // Locates a given entry within this bag.
        // Returns a reference to the node containing the entry, if located,
        // or null otherwise.



        // Node class: Should be private so not written in Javadoc format
        // A class that represents a Node with a data of type generic and
        // a Node type next link
        private class Node {
                // Private instance variables
                // Entry in bag


                // Link to next node


                // Constructors.
                // Constructor with a data of type T


                 // Constructor with a data of type T and a next of type Node



                 // get/set methods









        } // end Node
} // end LinkedBag
```

```java
package assignment;

/**
 * A class of that tests singly linked lists that have String as the data.
 *
 * @author Zeynep Orhan
 */

public class LinkedBagDriver {
	 /**
	 * palindrome: Tests whether this bag contains a given entry.
	 *
	 * @param w a String
	 * @return True if the w is a palindrome, or false otherwise.
	 */




	/**
	 * addAll: adds all the strings in a given String array to this bag.
	 *
	 * @param w a String array
	 * @return a BagInterface<String> which is a LinkedBag by adding all Strings in w
	 */




	/**
	 * allPalindrome: adds all the strings in a given String array to a LinkedBag
	 * and tests if they are all palindrome
	 *
	 * @param w a String array
	 * @return True if all strings are palindromes, or false otherwise.
	 */




	/**
	 * allPalindromes: adds all the strings in a given String array to a LinkedBag
	 * and returns a string of the palindromes
	 *
	 * @param w a String array
	 * @return a string that has all the palindrome strings in the bag separated by space,
	 * most recent one should be at the beginning
	 */



	public static void main(String[] args) {
		// TODO Auto-generated method stub
		BagInterface<String> words = new LinkedBag<>();
		String w[] = { "a", "ab", "aba", "abba", "abcba", "abb" };
		for (String word : w)
			words.add(word);

		System.out.println("Add all words");
		System.out.println(words);

		Object wordArray[] = words.toArray();
		for (Object word : wordArray)
			if (palindrome((String) word))
				System.out.println(word + " is a palindrome");
			else
				System.out.println(word + " is not a palindrome");

		System.out.println("Is the list empty? " + words.isEmpty());
		System.out.println("Number of aa in the list " + words.getFrequencyOf("aa"));
		System.out.println("Number of abb in the list " + words.getFrequencyOf("abb"));
		System.out.println("Add abb " + words.add("abb"));
		System.out.println("Number of abb in the list after adding one more " + words.getFrequencyOf("abb"));
		System.out.println("Number of words in the list " + words.getCurrentSize());
```

```
                words.clear();
                System.out.println("Number of words in the list after clear " + words.getCurrentSize());

                LinkedBag<String> words2 = (LinkedBag<String>) addAll(w);
                System.out.println("Is the list empty? " + words2.isEmpty());
                System.out.println("Number of aa in the list " + words2.getFrequencyOf("aa"));
                System.out.println("Number of abb in the list " + words2.getFrequencyOf("abb"));

                System.out.println("Add abb " + words2.add("abb"));
                System.out.println("Number of abb in the list after adding one more " + words2.getFrequencyOf("abb"));
                System.out.println("Number of words in the list " + words2.getCurrentSize());

                words2.clear();
                System.out.println("Number of words in the list after clear " + words2.getCurrentSize());

                System.out.println("Are all palindromes? " + allPalindrome(w));
                System.out.println("Show all palindromes " + allPalindromes(w));

        }
}
```

Output

```
Add all words
[abb, abcba, abba, aba, ab, a]
abb is not a palindrome
abcba is a palindrome
abba is a palindrome
aba is a palindrome
ab is not a palindrome
a is a palindrome
Is the list empty? false
Number of aa in the list 0
Number of abb in the list 1
Add abb true
Number of abb in the list after adding one more 2
Number of words in the list 7
Number of words in the list after clear 0
Is the list empty? false
Number of aa in the list 0
Number of abb in the list 1
Add abb true
Number of abb in the list after adding one more 2
Number of words in the list 7
Number of words in the list after clear 0
Are all palindromes? false
Show all palindromes abcba abba aba a
```