LAB 2 - Sorted List of Words Using Linked Lists

Due Date: L01: Feb.1; L02: Feb.8; L03: Feb.2; L04: Feb. 9; L05: Feb.3; L06: Feb.10; L07: Feb.4; L08: Feb.11; L09: Feb. 5; L10: Feb.12

Assessment: 4% of the total course mark.

Description:

In this assignment you have to implement sorted lists of words using **singly linked lists**. Each list should store the words in alphabetical order. Operations to be performed on the lists of words are mainly insertion of words, deletion of words, and searches. To this end write a `Java` class `WordLinkedList`. Each `WordLinkedList` object represents a collection of distinct words **sorted in alphabetical order** (the order the words appear in the dictionary). Additionally, you have to write a class `Node` to represent nodes of the linked list.

Each list should be implemented using a **singly linked list**, where each node stores a word represented as a `String` object. You are allowed to use a dummy node as the header. The number of nodes in the linked list other than the dummy header (if you use one) has to be equal to the number of words in the word list. Additionally, the list may not contain duplicates. To determine the alphabetical order of two words you may use method `compareTo()` from class `java.lang.String`.

You are allowed to use from `Java API` classes or methods for I/O, for string manipulation (for instance you may use methods `compareTo()` or `charAt()` from class `java.lang.String`), and exceptions, but no other classes or methods. Thus, you are not allowed to use a `Java` API method for sorting arrays and you are not allowed to use the class `ArrayList`. If you intend to use a `Java API` class or method and you are not sure if it is allowed or not, ask the instructor.

Class `WordLinkedList` has to satisfy the specifications described below.

Specifications:

Class `WordLinkedList` has at least the following instance fields:

1) an integer to store the **size of the list**, i.e., the number of words;
2) a reference to the beginning of the linked list (a reference variable of type `Node`).

All instance fields are `private`.

Class `WordLinkedList` contains at least the following constructors:

- `public WordLinkedList()` - constructs an empty `WordLinkedList` ("empty" means with zero words).

- `public WordLinkedList(String[] arrayOfWords)` - constructs a `WordLinkedList` object and stores the words from `arrayOfWords` in the list. Pay attention to the fact that the array passed to the constructor may contain duplicate words and the words might not be sorted. On the other hand, the list **is not allowed to**

contain duplicates and the words have to be sorted in alphabetical order. You may first create an empty list and then insert one word at a time using method `insert()`, to be described shortly. For comparing strings, you may use the method `compareTo()` from class `java.lang.String`. If you decide to use a sorting method, then you need to implement it yourself. You may assume that all strings in `arrayOfWords` consist only of lower case letters, and no other characters.

Class `WordLinkedList` contains at least the following methods:

- `public int getSize()` - returns the size of `this` list (the number of words).

- `public String getWordAt(int i) throws IndexOutOfBoundsException`- returns a new `String` object representing the word at position `i` in the list. Valid positions are between 0 and `n-1`, inclusive, where `n` denotes the size of the list. If the input index does not correspond to a valid position, an exception should be thrown. Since the list is sorted alphabetically, the positions of words have to agree with their alphabetical order.
  Example: For the list {`answer, blue, game, glory, student, tea`}, the method call `getWordAt(2)` returns a `String` object representing the word `game`.

- `public void insert(String newword)` - inserts `newword` in `this WordLinkedList` if `newword` is not in the list. It does nothing if `newword` is already in `this WordLinkedList`. Notice that, after an insertion is performed, the value of the field storing the size of the list has to be updated. You may assume that the string `newword` which is passed to method `insert()` consists only of lower case letters, and no other characters.
  Important: After each insertion the list must remain sorted!
  Example: Consider the list is: {`answer, game, tea`}. Assume now that word `bee` has to be inserted. Then after the insertion the list is {`answer, bee, game, tea`}.

- `public int find(String word)` - returns the position of `word` is in `this WordLinkedList` or -1 if `word` is not in `this WordLinkedList`. **Example:** If `this WordLinkedList` is {`answer, bee, game, tea`}, then after invoking `find("game")` the value 2 is returned, while after invoking `find("task")` the value -1 is returned.

- `public String remove(int i) throws IndexOutOfBoundsException` - removes the word at position `i` from `this WordLinkedList` and returns a new `String` object representing the removed word if `i` is a valid position in the list; Otherwise, an exception is thrown with an appropriate message. Notice that, after a word is removed, the value of the field storing the size of the list has to be updated.
  Important: After each remove operation, the list must remain sorted!
  Example: If `this WordLinkedList` is {`answer, bee, game, tea`}, when invoking `remove(0)` the word `answer` has to be removed from the list. The resulting list is {`bee, game, tea`}.

- `public void mergeTo(WordLinkedList that)` - removes all the words from `that WordLinkedList` and inserts them in `this WordLinkedList`. The words that are already in `this WordLinkedList` are not inserted anymore. This method has to run in $O(n_1 + n_2)$ time in the worst case, where n_1 is the size of `this` list and n_2 is the size of `that` list. The additional memory usage must be $\Theta(1)$ (i.e., only a constant amount of memory may be allocated during the method execution). Only half the mark will be awarded for less efficient implementations.
- `public String toString()` - returns a string representing the list, with the words listed in alphabetical order, each word on a separate line.

Report Description:

Write a report containing a concise description of the algorithms used for methods `insert()`, `mergeTo()` and for the second constructor and an evaluation of the worst-case running time and of the additional amount of memory used in terms of Big-theta (include a justification, not just the final results). Illustrate the algorithm for method `mergeTo()` and for the constructor with sketches of the linked list and of the additional variables involved in the execution at various stages (at least three stages) during the execution. The report may not have more than three pages.

In the report you also **have to specify the following dates**: 1) when you started working on the assignment, 2) when you completed half of it, and 3)when you finished it.

If you get inspiration from other sources you have to acknowledge them in the report. If you discuss the solution with your friends you have to acknowledge this and provide their names. You are not allowed to copy portions of the code or portions of the report from anywhere. You have to write the whole source code and the whole report yourself.

Submission Instructions: Submit the source code of the classes `Node` and `WordLinkedList` in a single text file and the report in a pdf file. Include your student number in the name of the file. For instance, if your student number is 12345 then the files should be named as follows:

- WordLinkedList-Lab2-12345.txt
- Report-Lab2-12345.pdf

Submit the files in the Dropbox on Avenue by 11:59 pm the day of your designated lab session.

To get credit for the assignment you have to demonstrate your code (i.e., class `WordList` and the test class) in front of a TA during your lab session. A 50% penalty will be applied for late demo. A 25% penalty will be applied if the demo is on time, but the electronic submission is late. The report does not have to be presented to the TA during the demo, but it has to be submitted online by the end of the day of your lab session (11:59 pm). A 50% penalty will be applied to the report mark if the report is submitted late.