LAB 1 - Sorted List of Words Using Arrays

Due Date: L01: Jan.18; L02: Jan.25; L03: Jan.19; L04: Jan.26, L05: Jan. 20; L06: Jan. 27; L07: Jan.21; L08: Jan.28; L09: Jan.22; L10: Jan. 29

Assessment: 4% of the total course mark.

Description:

In this assignment you have to implement sorted lists of words, using **arrays**. Each list must 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 you have to write a `Java` class `WordList`. Each `WordList` object represents a collection of distinct words **sorted in alphabetical order** (the order the words appear in the dictionary). Each word has to be represented using a `String` object and each list has to be implemented using an **array** of references to `String` objects. Notice that the size of the array may be larger than the number of words in the list, thus some positions in the array may be unoccupied (to have room for future insertions). Unoccupied positions must store `NULL` references. A hard requirement is that all **occupied positions in the array to appear before any unoccupied position**. 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`.

Notice that the implementation of the list of words must use **dynamic array expansion**, which means the following. When the array is full (i.e., all positions are occupied) and a new insertion is required, then a larger array has to be allocated, all the data copied from the old into the new array and the new word inserted in the new array. Another important requirement is that the size of the larger array to be a constant times the size of the old array. For instance, the new array may be twice as large as the old array. This requirement is in order to ensure that insertions are efficient.

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.

Your code has to be written in a good programming style, including **instructive comments** and **well-formatted, well-indented** code. Use self-explanatory names for variables as much as possible.

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

Specifications:

Class `WordList` has only the following instance fields:

1) an integer to store the **size of the list**, i.e., the number of words;
2) an array of references to `String` objects to store the words.

3) an integer to store the **capacity of the list**, i.e., the size of the allocated array of `Strings`.

All instance fields are `private`.

Class `WordList` contains at least the following constructors:

- `public WordList(int capacity)` - constructs an empty `WordList` ("empty" means with zero words) of the specified capacity. You may assume that the parameter `capacity` passed to the constructor is always positive, thus no error checking is required.

- `public WordList(String[] arrayOfWords)` - constructs a `WordList` object and stores the words from `arrayOfWords` in the list. The capacity of this object should be twice the size of the input array `arrayOfWords`. 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 `WordList` contains at least the following methods:

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

- `public int getCapacity()` - returns the capacity of `this` list (the size of the array).

- `public String getWordAt(int i) throws ArrayIndexOutOfBoundsException`- 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 WordList` if `newword` is not in the list. It does nothing if `newword` is already in `this WordList`. 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.

If an insertion is required, but the array is already full, then a larger array has to be allocated, all the data copied from the old into the new array and the new word inserted in the new array. **The size of the larger array has to be a constant times the size of the old array. For instance, the new array may be twice as large as the old array.** If the new array has only one (or two, or three, etc.) more elements than the old array, then the insertions are too inefficient (because of the need of copying the words from one array to the other too often)!!
Important: After each insertion the list must remain sorted!
Example: Assume that before the insertion the list is: {`answer, game, tea`}, the name of the array to store the list is `arr` and its size is 6. Then `arr[0]` stores `answer`, `arr[1]` contains `game` and `arr[2]` contains `tea`. Assume now that word `bee` has to be inserted. After the insertion the list is {`answer, bee, game, tea`} and the words are stored in the array as follows: `answer` in `arr[0]`, `bee` in `arr[1]`, `game` in `arr[2]` and `tea` in `arr[3]`.

- `public int find(String word)` - returns the position of `word` is in `this WordList` or -1 if `word` is not in `this WordList`. **This method must use the binary search algorithm. Example:** If `this WordList` 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 void remove(String word)` - removes `word` from `this WordList`; it does nothing if `word` is not in `this WordList`. 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 WordList` is {`answer, bee, game, tea`}, after removing the word `answer` the list is {`bee, game, tea`} and the words are stored in the array as follows: `bee` in `arr[0]`, `game` in `arr[1]` and `tea` in `arr[2]`.

- `public WordList sublist(char init, char fin)` - returns a new `WordList` object which contains exactly the words from `this` list whose first letter is in the range `init` $\cdots$ `fin`, inclusive. The capacity of the new list should be twice its size. You may assume that the parameters `init` and `fin` are lower case characters, thus no error checking is required. Method `charAt()` from class `java.lng.String` may be used to determine the first character of a string. The `Java` objects stored in the new list must be different from the `Java` objects stored in `this` list, even if they represent the same words.
Example: If `this` list is {`answer, blue, game, glory, student, tea`},
then the list created by invoking `sublist('c','e')` is empty (has zero words), while the list created by invoking `sublist('g','m')` is {`game, glory`}.

- `public int countInRange( char init, char fin )` - returns the number of words of `this` list whose first letter is in the range `init` $\cdots$ `fin`, inclusive. You may assume that the parameters `init` and `fin` are lower case characters, thus no error checking is required. This method **must perform** $O(\log n)$ word comparisons in

the worst-case, where n denotes the size of the list. (Hint: use the idea of binary search to find the first element of the sublist and then use it again to find the last element of the sublist). **No mark** will be awarded if the efficiency requirement is not satisfied.

- `public String toString()` - returns a string representing the list, with 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()`, `remove()` and `countInRange()`. For each of the methods `insert()` and `remove()`, consider a list with at least five words and draw a sketch of the data structure (i.e., of the array and of the other fields) at the beginning, in the middle and at the end of method execution. Indicate the values of the local variables involved in the execution as well. 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.

Test Class:

You are also required to write a test class. Pay attention to test that all specifications are met (for all the methods) and to test special cases arising in your code. For instance, when testing method `getWordAt(i)` test separately the cases when (1) $0 \leq$ `i`$\leq$`n-1`; (2) `i` < 0; (3) `i` $>$ `n-1`. When testing method `insert()` test separately the cases when the new word is inserted (1) inside the list; (2) at the beginning; (3) at the end; (4) in an empty list; (5) no insertion occurs because the word is already in the list, and so on.

Include comments or output messages in your test class to indicate what cases or what specifications you test.

Submission Instructions: Submit the source code of the class `WordList` in a text file, the source code for the test class in another 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 have to be be named as follows:

- WordList-Lab1-12345.txt
- TestWordList-Lab1-12345.txt
- Report-Lab1-12345.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.