CS145 – PROGRAMMING ASSIGNMENT #5

ANAGRAMS

OVERVIEW

This program focuses on programming topics including Array Sorting, Object creation, interfaces, string manipulation, and general programming topics. You will also be introduced to the idea of multiple front ends.

INSTRUCTIONS

Your deliverable will be to turn in two files. One file named `Word.java` and the other file will be called `AnagramManager.java` the course web site.

You will need support files `AnagramMain.java`, `AnagramMainGUI.java`, `dicitonary.txt`, `dictionary2.txt`, from the course web site; place them in the same folder as your project.

ANAGRAMS

An anagram is a word or phrase made by rearranging the letters of another word or phrase. For example, the words "midterm" and "trimmed" are anagrams. The same is true of "eat", "tea", "ate" and "eta".

The canonical form of any word is a list of the letters of that word in alphabetical order starting with the letter "a". For example the canonical form of the word "*live*" is "*eilv*" while the canonical form the word "*veil*" is also "*eilv*".

PROGRAM DESCRIPTION

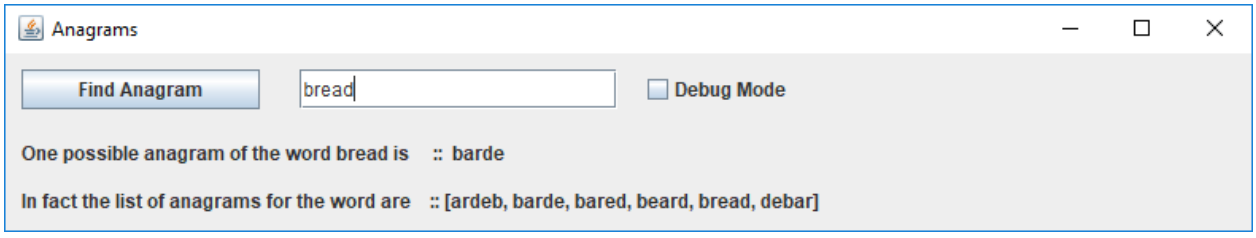
In this assignment, you will construct both an object that contains both a word and its canonical form, and a manager class to keep track of an array of these words.

However, the files that you write should work with both of the front-end methods. With no changes the files that you turn in should work with both `AnagramMain.java` and `AnagramMainGUI.java`.

CONSOLE - SAMPLE EXECUTION

```
Welcome to the CS145 Anagram Practice

Press enter to start, or any other input for debug mode

Please type in a word to anagram or QUIT to quit :
evil
One possible anagram of your word evil is live
In fact the list of anagrams for your word are :
  [evil, live, veil, vile]

Please type in a word to anagram or QUIT to quit :
llllll
Your word was not found in the list
In fact the list of anagrams for your word are : []

Please type in a word to anagram or QUIT to quit :
bread
One possible anagram of your word bread is ardeb
In fact the list of anagrams for your word are :
   [ardeb, barde, bared, beard, bread, debar]

Please type in a word to anagram or QUIT to quit :

What do you want to generate (Enter to quit)?
```

GUI - SAMPLE EXECUTION



YOUR INSTRUCTIONS

You will be given two client programs called `AnagramMain.java` and `AnagramMainGUI.java` that do the file processing and user interaction. You are to write a class called `AnagramManager` that manipulates the list of words and handles all the sorting and the searching. You will also write a class called Word that will implement the methods for dealing with an individual word. You may/probably want to write the Word class first.

NECESSARY METHODS – WORD CLASS

The Word class will be a class that handles a single word, and its canonical form. It should be made as an interface to Comparable for use. The object should be relatively small, only needing two private fields to store both the original word and the canonically form of that word.

PUBLIC WORD(STRING X)

In the `Word(String x)`constructor you should store both the word given and the canonical form of the word. You may also want to *adjust* the case of the word for consistency.

PUBLIC GETWORD()

In the `getWord()` should return the original unmodified word.

PUBLIC GETFORM()

In the `getForm()` should return the canonical form of the word.

PUBLIC TOSTRING()

The `toString()` method should return a string that shows both the original word and its canonical form in the format "`[word=dorw]`".

PUBLIC COMPARETO(WORD X)

The `compareTo` method should implement the comparable interface for `Word` objects based upon the canonical form of the two words involved.

This means that “eat” and “tea” are considered equal to each other. While “dad” is smaller than “bob” *(because of their associated forms).*

NECESSARY METHODS – ANAGRAM MANAGER

The `AnagramManager` class will be your primary class to manage your list of words and to find the necessary anagrams. It will use the Word class described later to assist.

Internally you must use an array to keep track of the elements. For this assignment, you may not use a List or any other ADT. Part of this assignment is learning to use the techniques on an array. Using a List or ADT will result in the loss of points. *I am aware that the List would be more effective and easier, but restrictions breed creativity.*

PUBLIC ANAGRAMMANAGER(LIST<STRING> LIST)

In the public `AnagramManager(List<String> list)` constructor you should initialize a new manager for an ARRAY of word type objects provided for you in the list. It should take each word from the provided list and convert it into a `Word` type object (*described later*) and then added to an array of words for storages. Do not modify the list passed.

You should throw an `IllegalArgumentException` if the list is null, or empty (size 0).

Your class should keep an internal list of `Word`s as an array. Part of this assignment is the understanding of how to work with an array, so while a List might be a better option, to receive full credit you should use an array of `Word`s.

PUBLIC VOID SORTBYWORD()

In `sortByWord()` you should sort your internal array of words by the actual word value.

PUBLIC VOID SORTBYFORM()

In the `sortByForm()` method, you should sort your internal array of words by the canonical form of the word as described above.

PUBLIC STRING GETANAGRAM(STRING X)

In the `getAnagram(String x)` method you should accept in a word parameter and return a random word from the array that has the same canonical form as the word you entered. If no word is found that is canonically the same then return an empty string.

For example if you do `getAnagram("garage")` then the program should return “garage” as that is the only word that is canonically the same. However if you run `getAnagram("live")` then you may get back any of [evil, live, veil, vile] as they are all canonically the same.

PUBLIC SET<STRING> GETANAGRAMS(STRING X)

In the `getAnagrams(String x)` method you should accept in a word parameter and return a set of all the words that are canonically the same. So `getAnagrams("llllll")` would return an empty set, `getAnagrams("garage")` would return a set of size one, and `getAnagrams("live")` would return a set of size four.

PUBLIC STRING TOSTRING()

The `toString()` should return a string that prints off the first five Words of the current array and then print off the last five Words of the current array, depending on how it is sorted. For example if you do a `toString()` using the smaller dictionary, your output should look like.

```
[acme=acem][ate=aet][book=bkoo][came=acem][cook=ckoo][...][tea=aet][teaks=aekst][tee=eet][terse=eerst][trees=eerst]
```

Do not attempt to print off the entire list if you are using the primary dictionary, as it will be extremely large and make look like your program crashed.

DEVELOPMENTS STRATEGY AND HINTS

None of the methods of this assignment are particularly hard, but you will want to work in pieces to get them all working.

For this program you must store the contents in an array.

Other than the Array requirement, you are allowed to use whatever constructs you want from the Java class libraries.

You have two sorts to do this problem, but only one compareTo method. Think about how to best utilize the method you do have access to. In order to get full credit for this assignment, you want to use the most efficient ways you know of to process the data.

STYLE GUIDELINES AND GRADING:

Part of your grade will come from appropriately utilizing object methods.

In many ways, the fact that you have two different interfaces should not matter at all. You are programming to a set of instructions, so once you get it working, they should work the same for both. However, you need to make sure to test both versions.

Your class may have other methods besides those specified, but any other methods you add should be private.

One of your sorting methods should be really short, and one will require more work.

Comment your code descriptively in your own words at the top of your class, each method, and on complex sections of your code. Comments should explain each method's behavior, parameters, return, pre/post-conditions, and exceptions.

The correct answer when sorting by **word** is:

```
[aa=aa][aah=aah][aahed=aadeh][aahing=aaghin][aahs=aahs][...][zymotic=cimotyz][zymurgies=egimrsuyz][zymurgy=gmruyyz][zyzzyva=avyyzzz][zyzzyvas=asvyyzzz]
```

The correct answer when sorting by **form** is:

```
[aa=aa][abracadabra=aaaaabbcdrr][baccalaureates=aaaabcceelrstu][baccalaureate=aaaabcceelrtu][bacchanalia=aaaabcchiln][...][tut=ttu][tutu=ttuu][ut=tu][tux=tux][xu=ux]
```