CSC220 Lab06
Anagrams

The goal of this week's assignment is:

1. Practice sorting
2. Continue learning the significance of special cases!
3. Learning how to write test to check your implementation!

Things you must do:

1. There are many details in this assignment. Make sure you read the whole thing carefully before writing any code and closely follow this instruction.
2. You must complete your assignment **individually**.
3. Always remember Java is case sensitive.
4. Your file names, class names, and package name must match exactly as they are specified here.

Things you must not do:

1. You must not change the file names, class names, package names.
2. You must not change the signature of any of these methods (name, parameters, …).

Part 0

- Create a new Java project and call it **Lab06**.
- Create a package inside your project and call it **lab06**.
- Download the Lab06-Assignment06 ZIP folder from Blackboard. Copy and paste the following files into your new lab06 package:
  - **AnagramUtil.java**

Part 1 – Problem Description

For this lab (and assignment), you are asked to construct a program that has the capability to determine if two words are anagrams and to find the largest group of anagrams in a list of words. Two words are anagrams if they contain the same letters with the same frequency. For example, **alert** and **later** are anagrams.

Now that we know how to implement an insertion sort, we are eager to use sorting to solve these two problems this week:

- To check if two words are anagrams, simply sort the characters in each word. If the sorted versions are the same, the words are anagrams of each other.
- Words with the same letters with the same frequency, but different cases are still anagrams. E.g., **Begin** and **being** are anagrams. Hint: Java's built-in String method `toLowerCase` will come in handy when sorting and determining if words are anagrams, but you must still output them in their original case.
- There are many websites devoted to listing anagrams, such as this one (http://www.english-for-students.com/Complete-List-of-Anagrams.html) if you are unclear about the concept yet.
- To find the largest group of anagrams in a list of words, sort the list with

a Comparator that compares the sorted character representations of the words. After the sort, any group of words that are anagrams of each other will be adjacent in the list.

- We will concern ourselves only with word anagrams and not anagrams that are phrases, which may contain whitespace and punctuation.
- You may assume that the word list given as input to both getLargestAnagramGroup methods **does not contain duplicates**.
- **Please follow this algorithm.** It is not the most optimal solution, as we will later discuss. But to get full credit for this Lab, you must follow this algorithm.

Part 2 - Requirements

We start by implementing the easier methods first. Please implement them in the order given and follow the notes closely, your job will be much easier! As always, you are more than welcome to create extra helper methods you need.

- `public static String sort(String inputString)`
    - o This method returns the sorted version of the input string. The sorting must be accomplished using an insertion sort
    - o You are not allowed to use Arrays.sort, you need to implement the insertion sort yourself.
    - o We do not care about lower/upper case. Hint: Java's built-in String method toLowerCase will come in handy when sorting
    - o return null if the input string is null or empty.

- `public static boolean areAnagrams(String string1, String string2)`
    - o This method returns true if the two input strings are anagrams of each other, otherwise returns false.

    - To check if two words are anagrams, simply sort the characters in each word. If the sorted versions are the same, the words are anagrams of each other.

- public static void insertionSort(String[] inputList)
    - This method sorts the input array using an insertion sort and using a **Comparator** object that compares the sorted character
      representations of the words (or strings) in the inputlist.
    - **NOTE:** You are required to use Comparator (nothing else).
    - First write a class that implements Comparator interface. Let's call it "OrderStrings". This class needs to implement a compare method that accepts two strings and compares them.
    - For instance: assume we pass the following array of strings to this function: {"joy", "ski", "fed", cat"}. Here is how the output should look like {"cat", "fed", "ski", "joy"}. Make sure you understand this example before start coding this function.

Part 3 - Testing

THIS PART IS NOT OPTIONAL

Unlike previous labs/assignments you are not given any tests as a starting point. You must create your own tests to examine all and every method you implemented in the previous part. Make sure you check the functionality of your methods very carefully. Here is a test scenario:

1) Test sort: Create a string that includes your name (first name or last name), then pass that string to your sort method. If you look at the signature of this method, it returns a String. Print the string and make sure it has been reordered properly (i.e., the characters are shuffled in the alphabetical order). If not, debug your sort method.

2) What special case you should have taken care of?
3) Test areAnagrams : Do not start this test before your sort method is correct simply because this method is using sort, so if your sort method is not working properly yet, there is no way this method would work! Now that your sort method is working you can test your areAnagrams method. How can you do that? think a second before you to go the next line
    a. Create a second string that includes the previous string in backward order (for instance). Now pass both of them to your areAnagrams. You should get true, why?
    b. Now create two strings, one including your firstname and one including your lastname. Pass both of them to your areAnagrams method. Obviously, you should get false unless your first and last names are exactly the same! Otherwise, go back and debug your code.

4) Test insertionSort: The first thing you need to do for testing this method is to make sure your compare method is working properly. How will you do that?
    a. Your previous test (and previous assignment comes handy here).
        i. Define two strings (you can put anything you want in them, your first name and last name for instance).
        ii. Instantiate from your OrderStrings class (see insertionSort method description in the previous part if you don’t know what I am referring to in here and read the instruction carefully!)
        iii. Use the compare method from OrderStrings class to compare your two strings and see if it returns the correct values for the comparison (i.e., alphabetical order comparison). If you are not sure what compare method should return, you need to go back and learn it from lecture notes!
    b. Now that you made sure your compare method is working properly, create an array that includes a set of strings and test the example provided in the method description. If you got that working great, continue reading, otherwise go back and debug your code.

i. add a couple of names to it (your friends, family, etc).

ii. Call your insertionSort and pass the array of strings you just created to it. What should be the output? Write those names on a piece of paper and sort the characters of each word first. Now sort the words alphabetically.

iii. what special cases you need to worry about?

   1. One is what would happen if you have an empty array.
   2. What else?

Here is also a list of scenarios you need to consider:

- sorting
  1) check empty or null string
  2) check string is sorted correctly

- areAnagrams
  1) empty string vs another string
  2) exactly same string
  3) shuffled strings
  4) two different strings

- insertionSort
  1) an empty list
  2) a list with one element
  3) a list with two elements
  4) a sorted list of strings
  5) a random list of strings

Make sure to upload your code to Box when you have completed the lab.
Don’t forget: lab is due tomorrow (Thursday) night @ 11:59pm!