Name: ______________________________ Andrew Id: ____________________

15-121 Fall 2023 Assessment 5

Up to 50 minutes. No calculators, no notes, no books, no computers. Show your work!

1. **Anagram Map**

In this problem you will create a data structure to store anagrams of words. It is very similar to what you built in HW4 in this course, except that you will build it with a `HashMap` instead of a binary search tree.

While solving this problem, you may *not* add any new instance variables to `AnagramMap`

```
public class AnagramMap {
    private HashMap<String, ArrayList<String>> wordMap;

    public AnagramMap() {
        wordMap = new HashMap<String, ArrayList<String>>();
    }

    public static String sortWord(String word) {
        char[] sortedArray = word.toCharArray();
        Arrays.sort(sortedArray);
        return new String(sortedArray);
    }

    /**
     * Adds a word to the AnagramMap. Anagrams are stored in ArrayLists in the map, with the key to each
     * list being a sorted version of the word.
     *
     * Only unique words should be added, so no word should be allowed to appear twice in the Anagram Map.
     * If an attempt is made to add a duplicate, raise an IllegalArgumentException.
     *
     * @param word The word to add to the AnagramMap. You can assume it will consist of only lowercase letters.
     */
    public void addWord(String word) {
        // You will write this code
    }

    /**
     * Return the total number of words that have been added to the AnagramMap. This is not the same thing as
     * the number of keys in the map.
     *
     * Because you are not allowed to add instance variables to this class, this method should go through the
     * entire map and sum the number of words in the ArrayLists.
     *
     * @return The number of words in the AnagramMap.
     */
    public int numWords() {
        // You will write this code
    }

    /**
     * Searches the map given a word and returns a list of all the words that are anagrams of it. If there
     * are no anagrams of the word in the map, then return an empty list.
     *
     * @param word A word to search for anagrams of. You can assume it will consist of only lowercase letters.
     * @return An ArrayList containing all the words in the tree that are anagrams of word.
     */
    public ArrayList<String> findMatches(String word) {
        // You will write this code
    }

    public static void main(String[] args) {
        AnagramMap anagrams = new AnagramMap();
        String[] words = {"rats", "sham", "snuggle", "arts", "mash", "mash"};
        for(String w: words) {
            anagrams.addWord(w);
        }
        System.out.println(anagrams.numWords()); // Prints 5
        System.out.println(anagrams.findMatches("tars")); // Prints [rats, arts]
        System.out.println(anagrams.findMatches("hasm")); // Prints [sham, mash]
        System.out.println(anagrams.findMatches("bob")); // Prints []
    }
}
```

(a) (7 points) Write the method `addWord`.

```
/**
 * Adds a word to the AnagramMap. Anagrams are stored in ArrayLists in the map,
 * with the key to each list being a sorted version of the word.
 *
 * Only unique words should be added, so no word should be allowed to appear
 * twice in the Anagram Map. If an attempt is made to add a duplicate, raise
 * an IllegalArgumentException.
 *
 * @param word The word to add to the AnagramMap. You can assume it will consist
 *             of only lowercase letters.
 */
public void addWord(String word) {
```

(b) (4 points) Write the method `numWords`.

```java
/**
 * Return the total number of words that have been added to the AnagramMap. This
 * is not the same thing as the number of keys in the map.
 *
 * Because you are not allowed to add instance variables to this class, this
 * method should go through the entire map and sum the number of words in the
 * ArrayLists.
 *
 * @return The number of words in the AnagramMap.
 */
public int numWords() {
```

(c) (4 points) Write the method `findMatches`.

```java
/**
 * Searches the map given a word and returns a list of all the words that are
 * anagrams of it. If there are no anagrams of the word in the map, then return
 * an empty list.
 *
 * @param word A word to search for anagrams of. You can assume it will consist
 *             of only lowercase letters.
 * @return An ArrayList containing all the words in the tree that are anagrams
 *         of word.
 */
public ArrayList<String> findMatches(String word) {
```

2. **Free Response**

   (a) (20 points) A `Song` class.

   In this problem you are going to write a `Song` class. Here are some important requirements for your class:

   - A `Song` has a title, artist, and runtime (in seconds). When a new `Song` is created, these items are passed to the constructor. (You must provide the constructor.)
   - The only way for code outside of `Song` to read and modify the attributes of a `Song` must be using appropriately named getters and setters. (You must provide the getters and setters.)
   - A title must not be `null`. If an attempt is made to set the title to something invalid, an `IllegalArgumentException` must be thrown.
   - An artist must not be `null`. If an attempt is made to set the artist to something invalid, an `IllegalArgumentException` must be thrown.
   - The artist may not be “Taylor Swift”. If an attempt is made to set the artist to “Taylor Swift” then an `IllegalArgumentException` must be thrown.
   - The runtime must be a positive integer. If an attempt is made to set the runtime to something invalid, an `IllegalArgumentException` must be thrown.
   - When printing a `Song`, its type and the value of its instance variables should appear. (Such as when `System.out.println(someSongObjectHere)` is used.)
   - A `Song` needs to be able to be properly used in `HashSet`s and `HashMap`s.
   - An array of `Song`s must be sortable using `Arrays.sort(someArrayOfMovies)`. The natural order is based on the artist name, with ties being broken by runtime (longest to shorted).

   Write the `Song` class to the above specifications. Pay careful attention to which classes you may need to extend or implement as well as which methods you need to write or override.

   Restriction: You may not use the the method `Objects.hash`.

Extra space for Question 2(a).

Extra space for Question 2(a).

(b) (5 points) Without modifying the `Song` class, write some code that would allow you sort an array of `Song`s by runtime (shortest to longest). You do not need to write a complete example with a main function and a testcase, instead just focus on what is needed in order to use the `Arrays.sort` method to sort an array of `Song`s named `mySongArray`.