Homework Turnin

Name:	Xuqing Wu
Account:	xw88 (xw88@uw.edu)
Student ID:	1933202
Section:	AD
Course:	CSE 143 20wi
Assignment:	a6
Receipt ID:	53465a61ab7de37591244b8966ab37db

Turnin script completed with output:

Turnin Successful!

The following file(s) were received:

AnagramSolver.java (3261 bytes, sha256: bf80cc283567fb76f1311b33f2f7d8af)

```
1.  // Xuqing Wu
2.  // 2/27/2020
3.  // CSE143
4.  // TA: Eric Fan
5.  // Assignment #6
6.  //
7.  // Class AnagramSolver allows clients use a given dictionary to
8.  // find all combinations of certain number of words that have
9.  // the exactly same letters as the given phrase and print them in
10. // same order as the dictionary
11.
12. import java.util.*;
13.
14. public class AnagramSolver {
15.    private Map<String, LetterInventory> map;    //map to store all
16.       // strings from the given list and their corresponding letterinventory
17.    private List<String> replicate; //list contains all words from dictionary
18.
19.    // post: It is the constructor that uses the given list as its dictionary
20.    //       and put all word from list and their letterinventory in to the map.
21.    //       The list is unchanged. The dictionary is a nonempty collection
22.    //       of nonempty sequences of letters and contains no duplicates
23.    public AnagramSolver(List<String> list) {
24.       replicate = list;
25.       map = new HashMap<>();
26.       for(int i = 0; i < list.size(); i++) {
27.          String word = list.get(i);
28.          LetterInventory single = new LetterInventory(word);
29.          map.put(word, single);
30.       }
31.    }
32.
33.    // pre: max is bigger than or equal 0
34.    //      (throw an IllegalArgumentException if not)
35.    // post: Find all combinations of words that have the
36.    //       same letters as the given string and print them out.
37.    //       All combinations from the dictionary that are anagrams of
38.    //       the string include at most the passed integer "max" words
39.    //       (unlimited number of words if max is 0). First filter out
40.    //       words that are included in the string.
41.    public void print(String s, int max) {
```

```
42.         if(max < 0) {
43.            throw new IllegalArgumentException
44.               ("Max words to include is smaller than 0!");
45.         }
46.         LetterInventory target = new LetterInventory(s);
47.         LetterInventory current = new LetterInventory("");
48.         List<String> list = new LinkedList<>();
49.         Stack<String> stack = new Stack<>();
50.         for(String single: replicate) {
51.            LetterInventory letters = map.get(single);
52.            if(target.subtract(letters) != null) {
53.               list.add(single);
54.            }
55.         }
56.         print(target, current, max, list, stack);
57.      }
58.
59.      // post: the helper method of the public print method. Use
60.      //       recursive backtracking to find combinations of words
61.      //       that have the same letters as the given string. Use satck to
62.      //       store combinations, use list to get all related words, use
63.      //       Letterinventorys to keep track of what letters are left to fill.
64.      private void print(LetterInventory target, LetterInventory current,
65.         int max, List<String> list, Stack<String> stack) {
66.         if(target.isEmpty()) {
67.            System.out.println(stack);
68.         }
69.         if(max == 0 || max != stack.size()) {
70.            for(int i = 0; i < list.size(); i++) {
71.               String s = list.get(i);
72.               current = new LetterInventory(s);
73.               if(target.subtract(current) != null) {
74.                  stack.push(s);
75.                  print(target.subtract(current), current, max, list, stack);
76.                  stack.pop();
77.               }
78.            }
79.         }
80.      }
81.   }
```