Homework Turnin

Name:	Akshit K Patel
Email:	akshit@uw.edu
Student ID:	1561387
Section:	DC
Course:	CSE 143 16au
Assignment:	a6
Receipt ID:	14c83c33df5400c2bf1cbc570efd9b86

Turnin Successful!

The following file(s) were received:

Anagrams.java (7185 bytes)

```java
/**
 * @author Akshit Patel
 * @Date 11/10/2016
 * CSE 143D DC
 * TA: Melissa Medsker
 * HW #6 Anagrams
 */
import java.util.*;//Lists, Maps & Stack.

/**
 * This class uses the given list of meaningful words(dictionary) to print at
 * most the given maximum or all of the possible permutations(arrangements) of
 * alphabetical letters in a given word or phrase which can be defined as an
 * Anagram.
 *
 * <p>
 * A possible permutation is an Anagram if and only if it can be found in a
 * given dictionary and all letters have been used from the word/phrase. <br>
 * An example: [<i>cinema</i>] has anagrams [<i>iceman</i>], [<i>min, ace</i>],
 * <i>etc</i>. A word can itself be an anagram like [<i>cinema</i>] has
 * [<i>cinema</i>] as a possible permutation.
 * </p>
 */
public class Anagrams {

    /*
     * Store the LetterInvetory of every word in given dictionary.
     */
    private Map<String, LetterInventory> wordMap;
    /*
     * Reference to the list of words provided as dictionary.
     */
    private List<String> dict;

    /**
     * Constructs a new Anagrams object that uses the given list of strings as
     * its dictionary.
     *
     * @param dictionary list of strings of words which contains no duplicates,
     *        is nonempty collection of nonempty sequences of letters and which
     *        can be used to find anagrams of a given word/phrase.
     */
    public Anagrams(List<String> dictionary) {
        this.dict = dictionary;// reference
        this.wordMap = new HashMap<String, LetterInventory>();
        // store the letterInvetory of every word in dictionary.
        for (String word : this.dict) {
            this.wordMap.put(word, new LetterInventory(word));
        }
    }

    /**
     * Prints at most the given maximum or all of the possible permutations of
     * letters that make up words that all together form an anagram of a given
     * word/phrase with each individual word having meaning i.e. it can be found
     * in the given dictionary. Only alphabetical letters are used to form an
     * anagram ignoring the non-alphabets.
     *
     * <p>
     * If the given maximum is a positive integer like 2 then it will print
     * anagrams of the given word/phrase with two words at max i.e. [inventory]
     * can have anagrams [inventory] or [irony, vent] but not [inn, rev, toy] as
     * it is made up of 3 words. However, if the given maximum is 0 then all
     * possible combinations of words that make up an anagram are printed.
     * </p>
     *
     * <p>
     * The input string can be: "a b c".<br>
     * The output is in the form: [a,* b, c] or [ac, b] where a, b & c are
     * possible words for the anagram inside square brackets, the order of the
     * possible word as anagram is dependent on the dictionary order.
     * </p>
     *
     * @param text String representation of the word/phrase whose anagrams need
     *        to printed.
     * @param max Integer value of the maximum number of words than a possible
     *        anagram should have. 0 for unlimited possibilities or an positive
     *        integer for limited possibilities upto the given maximum.
     * @throws IllegalArgumentException if the given integer maximum limit is a
     *         negative number.
     */
    public void print(String text, int max) {
        if (max < 0) {
            throw new IllegalArgumentException("Max cannot be negative!");
        }
        LetterInventory textInventory = new LetterInventory(text);
        if (max == 0) {
            // max = Integer.MAX_VALUE;
            max = text.length();
        }
        // possible anagram to be stored in stack.
        Stack<String> anagram = new Stack<String>();
        // print the anagrams.
        this.getAnagrams(this.dict, textInventory, max, anagram);
    }

    /**
     * Prints at most the given maximum limit of an anagram of a given
     * word/phrase with each individual word having meaning i.e. it can be found
     * in the given dictionary. Only alphabetical letters are used to form an
     * anagram ignoring the non-alphabets.
     *
     * @param choices List of String of words of the given dictionary, also used
     *        to get the pruned version of larger dictionary made up of relevant
     *        words(possible words that could make an anagram).
     * @param textInventory LetterInvetory of the given text used to make
     *        anagrams and pruning the dictionary to shorter dictionary of
     *        relevant words.
     * @param max integer value of the maximum number of words than a possible
     *        anagram should have. Needs to a number greater than 0.
     * @param anagram Stack of strings that makes up an anagram of the given
     *        word/phrase.
     */
    private void getAnagrams(List<String> choices,
                             LetterInventory textInventory,
                             int max,
                             Stack<String> anagram) {
        // proceed if the inventory contains letters to consider.
        if (textInventory != null) {
            // base case: a possible anagram is found.
            if (textInventory.isEmpty()) {
                System.out.println(anagram);
            } else if (max > 0) {
                // prune the given dictionary to more relevant words.
                List<String> pruneChoices = this.pruneChoices(textInventory,
                                                              choices);
                for (String word : pruneChoices) {
                    anagram.push(word); // choose the word
                    this.getAnagrams(pruneChoices,
                                     textInventory
                                             .subtract(this.wordMap.get(word)),
                                     max - 1,
                                     anagram);
                    anagram.pop();// unchoose the word.
                }
            }
        }
    }

    /**
     * Returns a shorter/pruned version of the given dictionary that has more
     * relevant words(possible words that could make an anagram) that can be
     * considered to avoid unnecessary computation.
     *
     * @param textInventory LetterInvetory of the given text used to prune the
     *        dictionary to shorter dictionary.
     * @param shortDict List of String of words used as dictionary for the
     *        Anagram object.
     * @return List of string with pruned choices of the given dictionary
     *         consisting of only relevant words.
     */
    private List<String> pruneChoices(LetterInventory textInventory,
                                      List<String> shortDict) {
        List<String> choices = new ArrayList<>();
        for (String word : shortDict) {
            // if relevant word found then add it to pruned version of list.
            if (textInventory.subtract(this.wordMap.get(word)) != null) {
                choices.add(word);
            }
        }
        return choices;
    }
}
```