Anagrams.java

```
package com.example;


import java.util.Arrays;
import java.util.HashMap;

/**
 * An anagram is a word or phrase formed by rearranging the letters of a different word or phrase,
 * typically using all the original letters exactly once.[1]
 * For example, the word anagram itself can be rearranged into nag a ram,
 * also the word binary into brainy and the word adobe into abode.
 * Reference from https://en.wikipedia.org/wiki/Anagram
 */
public class Anagrams {

    // 4 approaches are provided for anagram checking. approach 2 and approach 3 are similar but
    // differ in running time.
    public static void main(String[] args) {
        String first = "deal";
        String second = "lead";
        // All the below methods takes input but doesn't return any output to the main method.
        Anagrams nm = new Anagrams();
        System.out.println(nm.approach2(first, second)); /* To activate methods for different approaches*/
        System.out.println(nm.approach1(first, second)); /* To activate methods for different approaches*/
        System.out.println(nm.approach3(first, second)); /* To activate methods for different approaches*/
        System.out.println(nm.approach4(first, second)); /* To activate methods for different approaches*/
        /**
         * OUTPUT :
         * first string ="deal" second string ="lead"
         * Output: Anagram
         * Input and output is constant for all four approaches
         * 1st approach Time Complexity : O(n logn)
         * Auxiliary Space Complexity : O(1)
         * 2nd approach Time Complexity : O(n)
         * Auxiliary Space Complexity : O(1)
         * 3rd approach Time Complexity : O(n)
         * Auxiliary Space Complexity : O(1)
         * 4th approach Time Complexity : O(n)
         * Auxiliary Space Complexity : O(n)
         * 5th approach Time Complexity: O(n)
         * Auxiliary Space Complexity: O(1)
         */
    }

    boolean approach1(String s, String t) {
        if (s.length() != t.length()) {
            return false;
        } else {
            char[] c = s.toCharArray();
            char[] d = t.toCharArray();
            Arrays.sort(c);
            Arrays.sort(d); /* In this approach the strings are stored in the character arrays and
                               both the arrays are sorted. After that both the arrays are compared
                               for checking anangram */

            return Arrays.equals(c, d);
        }
    }

    boolean approach2(String a, String b) {
        if (a.length() != b.length()) {
            return false;
        } else {
            int[] m = new int[26];
            int[] n = new int[26];
            for (char c : a.toCharArray()) {
                m[c - 'a']++;
            }
            // In this approach the frequency of both the strings are stored and after that the
            // frequencies are iterated from 0 to 26(from 'a' to 'z' ). If the frequencies match
            // then anagram message is displayed in the form of boolean format Running time and
            // space complexity of this algo is less as compared to others
            for (char c : b.toCharArray()) {
                n[c - 'a']++;
            }
2             for (int i = 0; i < 26; i++) {
1                 if (m[i] != n[i]) {
1                     return false;
                }
            }
1             return true;
        }
    }

    boolean approach3(String s, String t) {
1         if (s.length() != t.length()) {
1             return false;
        }
        // this is similar to approach number 2 but here the string is not converted to character
        // array
        else {
            int[] a = new int[26];
            int[] b = new int[26];
            int k = s.length();
3             for (int i = 0; i < k; i++) {
2                 a[s.charAt(i) - 'a']++;
2                 b[t.charAt(i) - 'a']++;
            }
2             for (int i = 0; i < 26; i++) {
2                 if (a[i] != b[i]) return false;
            }
1             return true;
        }
    }

    boolean approach4(String s, String t) {
1         if (s.length() != t.length()) {
1             return false;
        }
        // This approach is done using hashmap where frequencies are stored and checked iteratively
        // and if all the frequencies of first string match with the second string then anagram
        // message is displayed in boolean format
        else {
            HashMap<Character, Integer> nm = new HashMap<>();
            HashMap<Character, Integer> kk = new HashMap<>();
            for (char c : s.toCharArray()) {
1                 nm.put(c, nm.getOrDefault(c, 0) + 1);
            }
            for (char c : t.toCharArray()) {
1                 kk.put(c, kk.getOrDefault(c, 0) + 1);
            }
            // It checks for equal frequencies by comparing key-value pairs of two hashmaps
2             return nm.equals(kk);
        }
    }

    boolean approach5(String s, String t) {
1         if (s.length() != t.length()) {
1             return false;
        }
        // Approach is different from above 4 aproaches.
        // Here we initialize an array of size 26 where each element corresponds to the frequency of
        // a character.
        int[] freq = new int[26];
        // iterate through both strings, incrementing the frequency of each character in the first
        // string and decrementing the frequency of each character in the second string.
3         for (int i = 0; i < s.length(); i++) {
1             int pos1 = s.charAt(i) - 'a';
1             int pos2 = s.charAt(i) - 'a';
1             freq[pos1]++;
1             freq[pos2]--;
        }
        // iterate through the frequency array and check if all the elements are zero, if so return
        // true else false
2         for (int i = 0; i < 26; i++) {
1             if (freq[i] != 0) {
1                 return false;
            }
        }
1         return true;
    }
}
```

Mutations

```
23   1. removed call to java/io/PrintStream::println → NO_COVERAGE
24   1. removed call to java/io/PrintStream::println → NO_COVERAGE
25   1. removed call to java/io/PrintStream::println → NO_COVERAGE
26   1. removed call to java/io/PrintStream::println → NO_COVERAGE
46   1. negated conditional → KILLED
47   1. replaced boolean return with true for com/example/Anagrams::approach1 → NO_COVERAGE
51   1. removed call to java/util/Arrays::sort → KILLED
52   1. removed call to java/util/Arrays::sort → KILLED
56   1. replaced boolean return with false for com/example/Anagrams::approach1 → KILLED
     2. replaced boolean return with true for com/example/Anagrams::approach1 → SURVIVED
61   1. negated conditional → KILLED
62   1. replaced boolean return with true for com/example/Anagrams::approach2 → NO_COVERAGE
67   1. Replaced integer subtraction with addition → KILLED
     2. Replaced integer addition with subtraction → KILLED
74   1. Replaced integer subtraction with addition → KILLED
     2. Replaced integer addition with subtraction → KILLED
76   1. changed conditional boundary → KILLED
     2. negated conditional → SURVIVED
77   1. negated conditional → KILLED
78   1. replaced boolean return with true for com/example/Anagrams::approach2 → NO_COVERAGE
81   1. replaced boolean return with false for com/example/Anagrams::approach2 → KILLED
86   1. negated conditional → KILLED
87   1. replaced boolean return with true for com/example/Anagrams::approach3 → NO_COVERAGE
95   1. changed conditional boundary → KILLED
     2. Changed increment from 1 to -1 → KILLED
     3. negated conditional → SURVIVED
96   1. Replaced integer subtraction with addition → KILLED
     2. Replaced integer addition with subtraction → KILLED
97   1. Replaced integer subtraction with addition → KILLED
     2. Replaced integer addition with subtraction → KILLED
99   1. changed conditional boundary → KILLED
     2. negated conditional → SURVIVED
100  1. replaced boolean return with true for com/example/Anagrams::approach3 → NO_COVERAGE
     2. negated conditional → KILLED
102  1. replaced boolean return with false for com/example/Anagrams::approach3 → KILLED
107  1. negated conditional → KILLED
108  1. replaced boolean return with true for com/example/Anagrams::approach4 → NO_COVERAGE
117  1. Replaced integer addition with subtraction → KILLED
120  1. Replaced integer addition with subtraction → KILLED
123  1. replaced boolean return with false for com/example/Anagrams::approach4 → KILLED
     2. replaced boolean return with true for com/example/Anagrams::approach4 → SURVIVED
128  1. negated conditional → KILLED
129  1. replaced boolean return with true for com/example/Anagrams::approach5 → NO_COVERAGE
137  1. changed conditional boundary → KILLED
     2. Changed increment from 1 to -1 → KILLED
     3. negated conditional → SURVIVED
138  1. Replaced integer subtraction with addition → KILLED
139  1. Replaced integer subtraction with addition → KILLED
140  1. Replaced integer addition with subtraction → KILLED
141  1. Replaced integer subtraction with addition → KILLED
145  1. changed conditional boundary → KILLED
     2. negated conditional → SURVIVED
146  1. negated conditional → KILLED
147  1. replaced boolean return with true for com/example/Anagrams::approach5 → NO_COVERAGE
150  1. replaced boolean return with false for com/example/Anagrams::approach5 → KILLED
```

Active mutators

- BOOLEAN_FALSE_RETURN
- BOOLEAN_TRUE_RETURN
- CONDITIONALS_BOUNDARY_MUTATOR
- EMPTY_RETURN_VALUES
- INCREMENTS_MUTATOR
- INVERT_NEGS_MUTATOR
- MATH_MUTATOR
- NEGATE_CONDITIONALS_MUTATOR
- NULL_RETURN_VALUES
- PRIMITIVE_RETURN_VALS_MUTATOR
- VOID_METHOD_CALL_MUTATOR

Tests examined

- com.example.AnagramsTest.isAlphabetical(com.example.AnagramsTest) (1 ms)

Report generated by PIT 1.5.0