Lab3

Camden Fergen

Part 1:

- Q1
  - Junit creates a new instance of ProfileTest
  - It then calls the before method to initialize each of the variables
  - It then calls the test function
  - Repeat for every test function
- Q2

```
st
lic void matchAnswersFalseWhenMustMatchCriteriaNotMet() {
 profile.add(new Answer(question, Bool.FALSE));
 criteria.add(
         new Criterion(new Answer(question, Bool.TRUE), Weight.MustMatch));
 boolean matches = profile.matches(criteria);
 assertFalse(matches);

 *
st
lic void matchAnswersTrueForAnyDontCareCriteria() {
 profile.add(new Answer(question, Bool.FALSE));
 criteria.add(
         new Criterion(new Answer(question, Bool.TRUE), Weight.DontCare));
 boolean matches = profile.matches(criteria);
 assertTrue(matches);
```

Part 2:

- All code can be found commented in the files

```java
import org.junit.Test;
import java.util.List;
import static org.junit.Assert.*;
import lab3.letterCombos;

new *
public class letterComboTest {
    new *
    @Test
    public void testGenerateCombinations() {
        List<String> combinations = letterCombos.generateCombinations();

        assertEquals( message: "The number of combinations should be  676",  expected: 676, combinations.size());
        assertTrue( message: "The combinations should include 'aa'", combinations.contains("aa"));
        assertTrue( message: "The combinations should include 'zz'", combinations.contains("zz"));
    }
}
```

- Showing the tests for the 2 letter combinations

```java
/* 2 Letter combinations
public static List<String> generateCombinations() {
    String alphabet = "abcdefghijklmnopqrstuvwxyz";
    List<String> combinations = new ArrayList<>();
    for (int i =  0; i < alphabet.length(); i++) {
        for (int j =  0; j < alphabet.length(); j++) {
            combinations.add(alphabet.charAt(i) + "" + alphabet.charAt(j));
        }
    }
    return combinations;
}
 */
```

```
    }
     */

    new *
    @Test
    public void testGenerateCombinations() {
        List<String> combinations = letterCombos.generateCombinations();

        // Adjusted size to 17,576 for 3-letter combinations
        assertEquals( message: "The number of combinations should be 17,576",  expected: 17576, combinations.size()

        // Example checks for specific combinations
        assertTrue( message: "The combinations should include 'aaa'", combinations.contains("aaa"));
        assertTrue( message: "The combinations should include 'zzz'", combinations.contains("zzz"));
    }
}
```

- Showing tests for 3 letter combinations

```
/* 3 Letter combinations
public static List<String> generateCombinations() {
    String alphabet = "abcdefghijklmnopqrstuvwxyz";
    List<String> combinations = new ArrayList<>();
    for (int i = 0; i < alphabet.length(); i++) {
        for (int j = 0; j < alphabet.length(); j++) {
            for (int k = 0; k < alphabet.length(); k++) {
                combinations.add(alphabet.charAt(i) + "" + alphabet.charAt(j) + "" + alphabet.charAt(k));
            }
        }
    }
    return combinations;
}
```

- P2Q3
  - The limitations of the algorithm are defined by the computer you are running it on. Since its recursive it can take an in number under the 32bit integer limit, and try to compute it. Since we now have a recursive algorithm there is no limitation, but when we had the code to generate the two and 3 combinations the limit was 2 and 3 respectively.
  - For my computer my limit was length 6, as any greater than that it would crash

```java
    /* 1 Letter combinations tests */
    new *
    @Test
    public void testGenerateCombinations1() {
        List<String> combinations = letterCombos.generateCombinations( length: 1);

        assertEquals( message: "The number of combinations should be  26",  expected: 26, combinations.size());
        assertTrue( message: "The combinations should include 'a'", combinations.contains("a"));
        assertTrue( message: "The combinations should include 'z'", combinations.contains("z"));
    }

    /* 2 Letter combinations tests */
    new *
    @Test
    public void testGenerateCombinations2() {
        List<String> combinations = letterCombos.generateCombinations( length: 2);

        assertEquals( message: "The number of combinations should be  676",  expected: 676, combinations.size());
        assertTrue( message: "The combinations should include 'aa'", combinations.contains("aa"));
        assertTrue( message: "The combinations should include 'zz'", combinations.contains("zz"));
    }

    /* 3 Letter combinations tests */
    new *
    @Test
    public void testGenerateCombinations3() {
        List<String> combinations = letterCombos.generateCombinations( length: 3);

        // Adjusted size to 17,576 for 3-letter combinations
        assertEquals( message: "The number of combinations should be 17,576",  expected: 17576, combinations.size());

        // Example checks for specific combinations
        assertTrue( message: "The combinations should include 'aaa'", combinations.contains("aaa"));
        assertTrue( message: "The combinations should include 'zzz'", combinations.contains("zzz"));
    }
```

- Tests for general algorithm

```java
/* General letter combinations */
4 usages  new *
public static List<String> generateCombinations(int length) {
    String alphabet = "abcdefghijklmnopqrstuvwxyz";
    List<String> combinations = new ArrayList<>();
    if (length == 0) {
        return combinations;
    }
    generateCombinationsRecursive( current: "", alphabet, length, combinations);
    return combinations;
}

2 usages  new *
private static void generateCombinationsRecursive(String current, String alphabet, int length, List<String> combinations) {
    if (length == 0) {
        combinations.add(current);
        return;
    }

    for (int i = 0; i < alphabet.length(); i++) {
        generateCombinationsRecursive( current: current + alphabet.charAt(i), alphabet,  length: length - 1, combinations);
    }
}
}
```