ECSE 321 - Intro to Software Engineering
Assignment #3

Camilo Garcia La Rotta

March 11, 2017

1. Clean code and javadoc can be found in attached source folder

2. **NOTE**

   - Even though we handle the case where the input array is null by returning a non positive integer, we consider this to be an invalid operation (null pointer to a variable which was expected to contain a String array). Hence it is our only input considered INVALID and all other partitions are implied to be VALID
   - Please refer to the attached Java code to find the enumerated tests in `testCounter.java`

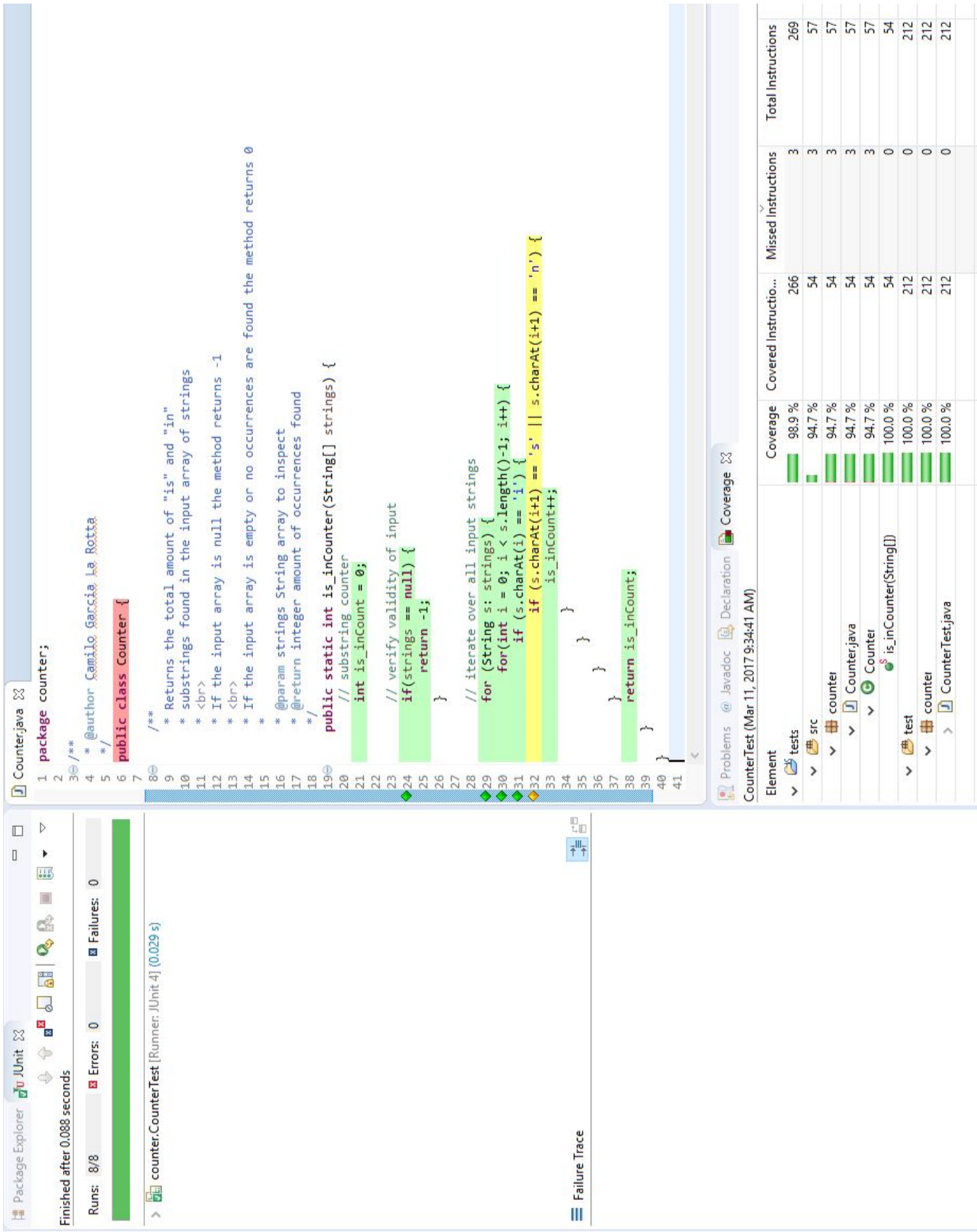
   Splitting our partitions to test into **INPUTS** and **INPUTS** we can derive the following:

   - **INPUTS:**
     (a) The string array can be null (INVALID) or non-null
     (b) The string array empty, with 1 element or more than one element
     (c) The strings inside the array can be have any positive number of characters. Zero, 1 or more than 1 characters
     (d) The strings inside the array can contain no is/in, only is, only in or multiple is/in
     (e) The substrings is/in can be found at the beginning, middle and end of the string
   - **OUTPUTS:**
     (a) -1 if the string is array is null
     (b) 0 if no occurrences of is/in were found or if the array is empty
     (c) a non zero positive integer n equal to the amount of occurrences of is/in

   The code coverage and success of the JUnit test is trivially 100% and 8/8 respectively. More interestingly, the partitions are tested by the following test cases:

   - **INPUTS:**
     (a) partition (a) was tested through test (1)
     (b) partition (b) was tested through tests (2), (3), (4)
     (c) partition (c) was tested through tests (3), (4), (5)
     (d) partition (d) was tested through tests (6), (7), (8)
     (e) partition (e) was tested through tests (6), (7), (8)
   - **OUTPUTS:**
     (a) partition (a) was tested through test (1)
     (b) partition (b) was tested through test (3)
     (c) partition (c) was tested through test (8)

For the class Counter, the visual annotations and source code highlighting can be found in the graph annexed below.



Note that the yellow line marking partial coverage of line 32 represents the fact that 1 out of the 4 branches didn't reach this conditional. This is to be expected, as it is the case when the character at index i is not 'i'.

Also note that the net coverage of the method is_inCounter(String[]) is 100%

3. The transition table for the given state chart is the following:

Transition #	**State**	**Transition**	**New State**
1	initial	create	created
2	created	getShipmentRate	quoteOffered
3	created	cancel	Canceled
4	quoteOffered	getShipmentRate	quoteOffered
5	quoteOffered	confirm	confirmed
6	quoteOffered	cancel	canceled
7	confirmed	getConfirmedRate	confirmed
8	confirmed	denyCancellations	cannotGetCancelled
9	confirmed	cancel	cancelled
10	cannotGetCancelled	getConfirmedRate	cannotGetCancelled

In order to minimize test cases and still achieve 100% state and transition coverage, we will start all test cases from the initial state. To find the total amount of linearly independent sequences we find it's cyclomatic complexity:

$$C(G) = E - V + 2PC(G) = 10 - 6 + 2C(G) = 6$$

Where C is the cyclomatic complexity, E the number of edges, V the number of vertices and P the number of separate components.

The linearly independent set of transitions is the following:

(a) 1-3

(b) 1-2-4

(c) 1-2-6

(d) 1-2-5-7

(e) 1-2-5-9

(f) 1-2-5-8-10

This sequences are tested in the same alphabetic order in the attached test suite. The CancellationTest contains all sequences that lead to the cancelled state. The LoopTest contains all sequences that lead to a steady state (loop to itself)