**Homework 5**

Chapter 8.1: Logic Coverage Criteria

**Name:**

**What to do?**

Consider the following method, and submit your answers for the questions below.

**public static boolean** isSatisfactory(**boolean** good, **boolean** fast, **boolean** cheap ) {  
 **if** ((good && fast) || (good && cheap) || (fast && cheap)) {  
 **return true**;  
 }  
 **return false**;  
}

Using the code above, answer the following questions.,

* 1. Write the predicate (p) to represent the logical expression in the code above. **(5 points)**
  2. List the clauses that go with predicate p. **(5 points)**

a = good b = fast c = cheap

* 1. By using the mathematical approach compute and simplify the conditions under which each clause determines predicate p. Be sure to include details or steps showing how you compute and simplify. **(15 points)**
  2. Write the complete truth table for all clauses. Label your rows starting from 1.

Use the format in the examples we covered in lecture. That is, row 1 should be all clauses true. You should include columns for the conditions under which each clause determines the predicate, and also a column for the value of the predicate itself. **(15 points)**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Row | a | b | c |  |  |  |  |
| 1 | T | T | T | T |  |  |  |
| 2 | T | T |  | T |  |  |  |
| 3 | T |  | T | T |  |  |  |
| 4 | T |  |  |  |  |  |  |
| 5 |  | T | T | T |  |  |  |
| 6 |  | T |  |  |  |  |  |
| 7 |  |  | T |  |  |  |  |
| 8 |  |  |  |  |  |  |  |

* 1. Give a list of pairs of rows from your table that satisfy the following: **(15 points)**
     1. Clause Coverage (CC)

Rows 1,8

* + 1. Predicate Coverage (PC)

Rows 1,8

* + 1. If any, find rows that satisfy Clause Coverage (CC) but do not satisfy Predicate Coverage (PC)

Rows 2,3,5 or Rows 4,6,7

* 1. List all pairs of rows from your table that satisfy Correlated Active Clause Coverage (CACC) with respect to each clause. Implement these tests in JUnit. **(20 points)**
     + For each JUnit test method, clearly identify which test case(s) it is for.
     + Demonstrate success by submitting your JUnit tests (.java file) and a screenshot or output showing the result of execution.

Rows: 2,3

Rows: 6,7

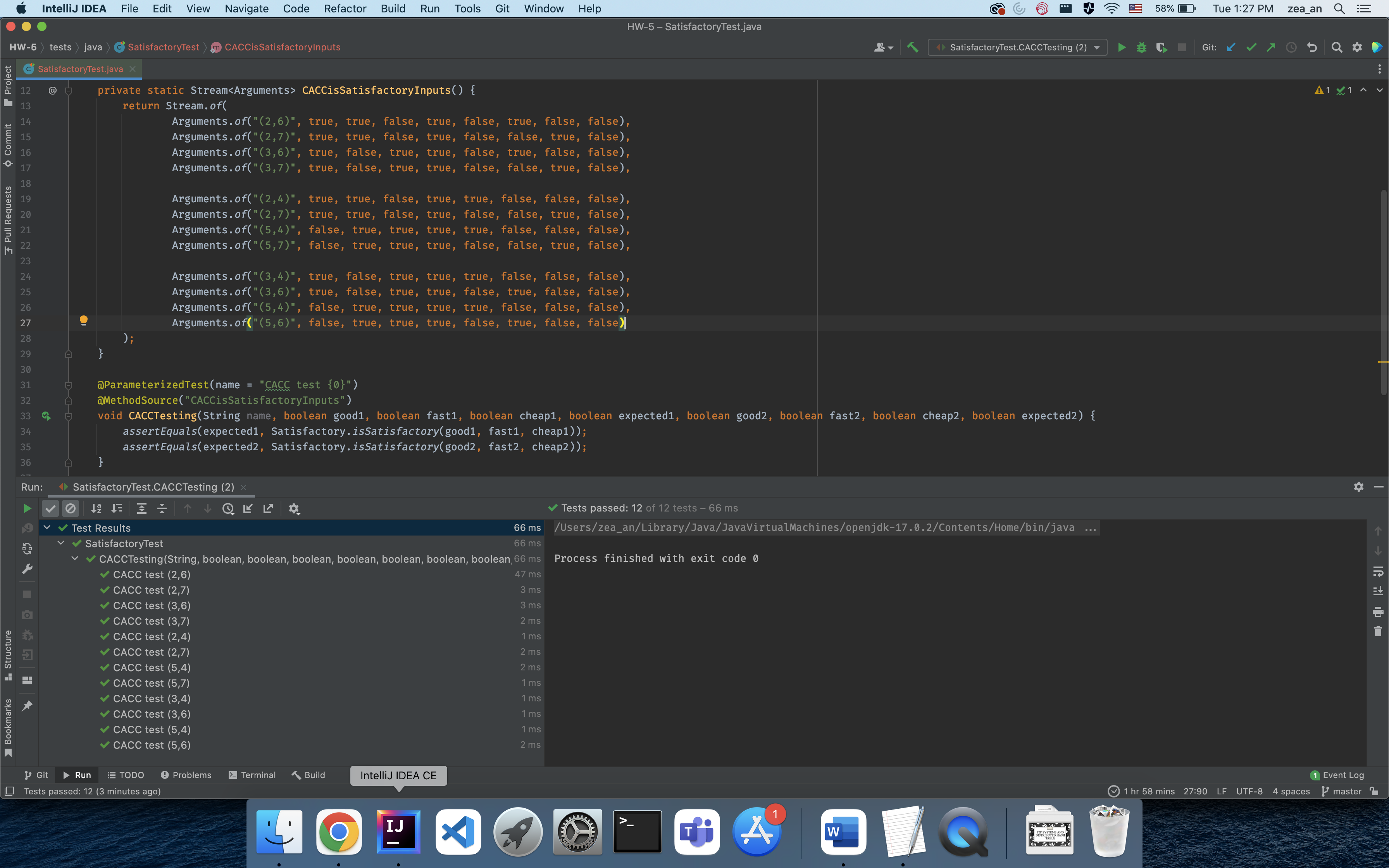
Rows: 2,5

Rows: 4,7

Rows: 3,5

Rows: 4,6

|  |  |  |
| --- | --- | --- |
| Major Clause | Row Pair | Test |
| a | (2,6) | isSatisfactory(true,true, false), isSatisfactory(false, true, false) |
|  | (2,7) | isSatisfactory(true, true, false), isSatisfactory(false, false, true) |
|  | (3,6) | isSatisfactory(true, false, true), isSatisfactory(false, true, false) |
|  | (3,7) | isSatisfactory(true, false, true), isSatisfactory(false, false, true) |
| b | (2,4) | isSatisfactory(true, true, false), isSatisfactory(true, false, false) |
|  | (2,7) | isSatisfactory(true, true, false), isSatisfactory(false, false, true) |
|  | (5,4) | isSatisfactory(false, true, true), isSatisfactory(true, false, false) |
|  | (5,7) | isSatisfactory(false, true, true), isSatisfactory(false, false, true) |
| c | (3,4) | isSatisfactory(true, false, true), isSatisfactory(true, false, false) |
|  | (3,6) | isSatisfactory(true, false, true), isSatisfactory(false, true, false) |
|  | (5,4) | isSatisfactory(false, true, true), isSatisfactory(true, false, false) |
|  | (5,6) | isSatisfactory(false, true, true), isSatisfactory(false, true, false) |



* 1. Suppose we refactor the isSatisfactory() method as shown below. Develop tests that achieves (CACC) for the predicates in the following method. Compare and contrast these tests with the tests you identified in f. (No need to write the tests in JUnit for this question, g) **(15 points)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Row |  |  |  |  |  |
| 1 | T | T | T |  |  |
| 2 | T |  |  |  |  |
| 3 |  | T |  |  |  |
| 4 |  |  |  |  |  |

TR\_A={(1,3)}

assertEquals(true, isSatisfactoryRefactored(true,true,true))

assertEquals(false, isSatisfactoryRefactored(false,true,true))

TR\_B={(1,2)}

assertEquals(true, isSatisfactoryRefactored(true,true,true))

assertEquals(false, isSatisfactoryRefactored(true,false,true))

Interesting that this highlights how the third parameter doesn’t impact this predicate due to P1’s reachability being always.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Row |  |  |  |  |  |
| 1 | T | T | T |  |  |
| 2 | T |  |  |  |  |
| 3 |  | T |  |  |  |
| 4 |  |  |  |  |  |

TR\_A={(1,3)}

assertEquals(true, isSatisfactoryRefactored(true,false,true))

assertEquals(false, isSatisfactoryRefactored(false,false,true))

TR\_C={(1,2)}

assertEquals(true, isSatisfactoryRefactored(true,false,true))

assertEquals(false, isSatisfactoryRefactored(true,false,false))

Second parameter was used to guarantee that second predicate is reached.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Row |  |  |  |  |  |
| 1 | T | T | T |  |  |
| 2 | T |  |  |  |  |
| 3 |  | T |  |  |  |
| 4 |  |  |  |  |  |

TR\_B={(1,3)}

assertEquals(true, isSatisfactoryRefactored(false,true,true))

assertEquals(false, isSatisfactoryRefactored(false,false,true))

TR\_C={(1,2)}

assertEquals(true, isSatisfactoryRefactored(false,true,true))

assertEquals(false, isSatisfactoryRefactored(false,true,false))

First parameter is used to guarantee that third predicate is reached.

I noticed that the tests from f) cover rows 2-7 while the test from g) cover rows 1,3,4,5,6,7. It is like we swapped row 2 for row 1.

Another detail I noticed is that the refactored form led to less ambiguity about which test to choose. For example,

This means I can come up with any combination from these three sets which will satisfy CACC. Compare this with the following.

Although there is less flexibility in the refactored version consideration needs to be taken to make sure a given predicate is reachable. I believe that is the main trade off.

* 1. Write the reachability of the predicates in the code below. Use the format in the example we covered in lecture (Lecture-9). For example, r(p1) = true (always reached) etc. **(10 points)**

**public static boolean** isSatisfactoryRefactored(**boolean** good, **boolean** fast, **boolean** cheap){  
 **if** (good && fast) {  
 **return true**;  
 }  
 **if** (good && cheap) {  
 **return true**;  
 }  
 **if** (fast && cheap) {  
 **return true**;  
 }  
 **return false**;  
}

**What to submit?**

Submit the following file to Blackboard:

* A word document describing your answers to the question above
* The code file requested above