Assignment 11

Rasika Mohod
rmohod@gmu.edu
G01044774

Introduction to Software Testing (Edition 2): Book by Jeff Offutt and Paul Amman
Exercises Section 8.3 (pages 223), Number 12(b).

```
// Introduction to Software Testing
// Authors: Paul Ammann & Jeff Offutt
// Chapter 8;
// See GoodFastCheapRACC.java, GoodFastCheapMUMCUT.java for JUnit tests
// See also GoodFastCheapRefactored.java

import java.util.*;

// GoodFastCheap: Investigating clause testing with an old engineering joke

public final class GoodFastCheap {

    boolean good  = false;
    boolean fast  = false;
    boolean cheap = false;

    public void makeGood () {
       good = true;
       if (fast && cheap) { cheap = false; }
    }

    public void makeFast () {
       fast = true;
       if (good && cheap) { good = false; }
    }

    public void makeCheap () {
       cheap = true;
       if (fast && good) { fast = false; }
    }

    public void makeBad ()       { good = false; }
    public void makeSlow ()      { fast = false; }
    public void makeExpensive () { cheap = false; }

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

Predicate for original isSatisfactory() method :

```
(good && fast) || (good && cheap) || (fast && cheap)
```

Truth Table:

Row#	good	fast	cheap	P	Pgood	Pfast	Pcheap
1	T	T	T	T			
2	T	T		T	T	T	
3	T		T	T	T		T
4	T					T	T
5		T	T	T		T	T
6		T			T		T
7			T		T	T	
8							

RACC:

good	(2,6), (3,7)
fast	(2,4), (5,7)
cheap	(3,4), (5,6)

Refactored isSatisfactory() method:

```
public boolean isSatisfactory() {
	if (good && fast) return true;
	if (good && cheap) return true;
	if (fast && cheap) return true;

	return false;
}
```

12. b) The RACC tests from the original method do not satisfy RACC on the refactored method. List what is missing, and add the missing tests to the JUnit from the prior exercise.

Predicates for refactored method:

P3: fast && cheap

Possible values for P3 to be reachable:

➔ good && fast and good && cheap cannot be true

➔ Thus, good has to be false

Row#	Good	Fast	Cheap
5	F	T	T
6	F	T	F

7	F	F	T

P2: good && cheap

Possible values for P2 to be reachable:

➔ good && fast cannot be true

➔ Thus, either good or fast has to be false

Row#	Good	Fast	Cheap
3	T	F	T
4	T	F	F
5	F	T	T

P1: good && fast

Possible values for P1 to be reachable:

Row#	Good	Fast	Cheap
2	T	T	F
3	T	F	T
4	T	F	F
5	F	T	T
6	F	T	F

Total tests to be taken: (Row #) 2, 3, 4, 5, 6, 7

Row#	Good	Fast	Cheap
2	T	T	F
3	T	F	T
4	T	F	F
5	F	T	T
6	F	T	F
7	F	F	T

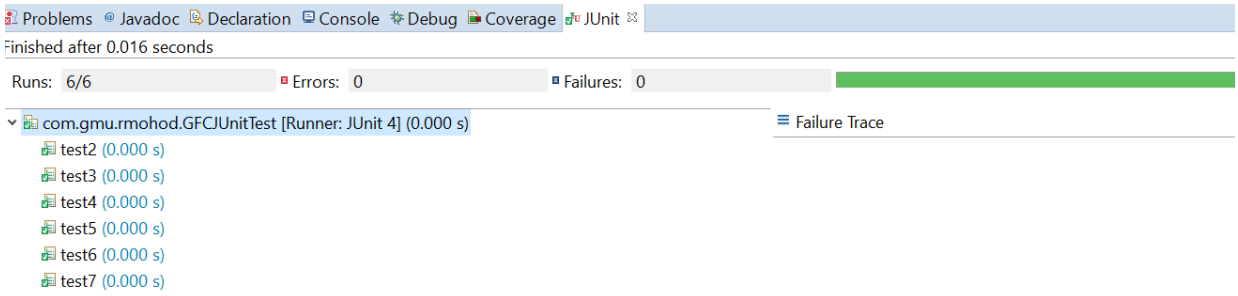
JUnit Tests (GFCJUnitTest.java):

```
package com.gmu.rmohod;

import static org.junit.Assert.*;

import org.junit.Test;
```

```
public class GFCJUnitTest {
	@Test
	public void test2() {
		//2 Good=T Fast=T Cheap=F

		GFCrefactored gfc = new GFCrefactored();
		gfc.makeGood();
		gfc.makeFast();

		assertEquals(true,gfc.isSatisfactory());
	}
	@Test
	public void test3() {
		//3 Good=T Fast=F Cheap=T

		GFCrefactored gfc = new GFCrefactored();
		gfc.makeGood();
		gfc.makeCheap();

		assertEquals(true,gfc.isSatisfactory());
	}
	@Test
	public void test4() {
		//4 Good=T Fast=F Cheap=F

		GFCrefactored gfc = new GFCrefactored();
		gfc.makeGood();

		assertEquals(false,gfc.isSatisfactory());
	}
	@Test
	public void test5() {
		//5 Good=F Fast=T Cheap=T

		GFCrefactored gfc = new GFCrefactored();
		gfc.makeFast();
		gfc.makeCheap();

		assertEquals(true,gfc.isSatisfactory());
	}
	@Test
	public void test6() {
		//6 Good=F Fast=T Cheap=F

		GFCrefactored gfc = new GFCrefactored();
		gfc.makeFast();

		assertEquals(false,gfc.isSatisfactory());
	}
	@Test
	public void test7() {
		//7 Good=F Fast=F Cheap=T

		GFCrefactored gfc = new GFCrefactored();
		gfc.makeCheap();

		assertEquals(false,gfc.isSatisfactory());
```

```
    } }
```

JUnit Test output:



GFCrefactored.java:

```java
package com.gmu.rmohod;

import java.util.*;

// GoodFastCheap: Investigating clause testing with an old engineering joke
public final class GFCrefactored {

    boolean good  = false;
    boolean fast  = false;
    boolean cheap = false;

    public void makeGood () {
       good = true;
       if (fast && cheap) { cheap = false; }
    }

    public void makeFast () {
       fast = true;
       if (good && cheap) { good = false; }
    }

    public void makeCheap () {
       cheap = true;
       if (fast && good) { fast = false; }
    }

    public void makeBad ()       { good = false; }
    public void makeSlow ()      { fast = false; }
    public void makeExpensive () { cheap = false; }

    public boolean isSatisfactory() {   // Refactoring here
       if (good && fast)  return true;
       if (good && cheap) return true;
       if (fast && cheap) return true;

       return false;
    }
}
```