Assignment 1 Report

Amirhossein Sohrabbeig - STD: 1744420

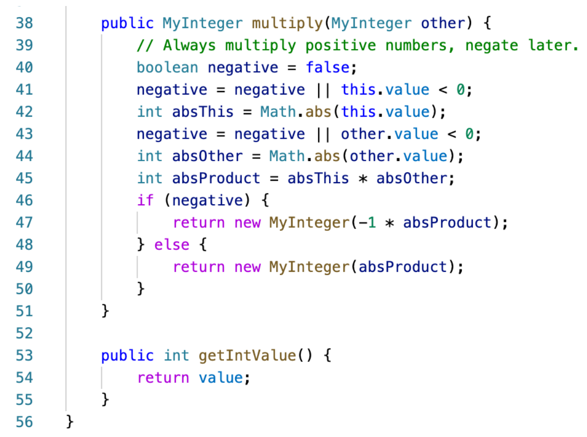
2022-02-09

Java libraries of this project are too large for uploading my files to the eclass, so to make the overall size of the project small, I excluded the libraries before uploading.

4.2 Discover a bug

I did all the steps and generated the tests. All corresponding files are available under the `4.2` sub-directory.

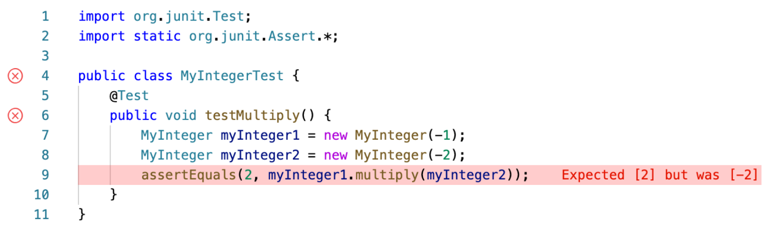
(This is not part of the assignment!) I have to mention that there exists another bug in the code which I found and Randoop was not able to find that. In the following image, you can see the implementation of multiply method.



```java
    public MyInteger multiply(MyInteger other) {
        // Always multiply positive numbers, negate later.
        boolean negative = false;
        negative = negative || this.value < 0;
        int absThis = Math.abs(this.value);
        negative = negative || other.value < 0;
        int absOther = Math.abs(other.value);
        int absProduct = absThis * absOther;
        if (negative) {
            return new MyInteger(-1 * absProduct);
        } else {
            return new MyInteger(absProduct);
        }
    }

    public int getIntValue() {
        return value;
    }
}
```

Figure 1: implementation of multiply function

I wrote a junit test for that. That test was not passed and it showed a bug in the code:



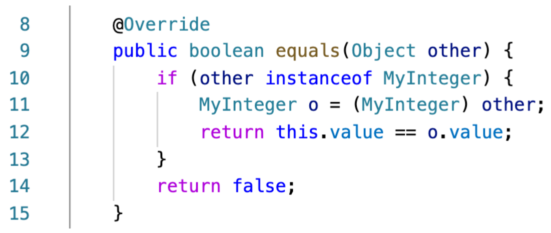
```java
import org.junit.Test;
import static org.junit.Assert.*;

public class MyIntegerTest {
    @Test
    public void testMultiply() {
        MyInteger myInteger1 = new MyInteger(-1);
        MyInteger myInteger2 = new MyInteger(-2);
        assertEquals(2, myInteger1.multiply(myInteger2));    Expected [2] but was [-2]
    }
}
```

Figure 2: Junit test for multiply function

The problem is that the negative variable stores the result of `OR` operation on `this.value` and `other.value`, which should be `XOR`.

4.3 Fix the bug

To fix the bug, first of all, I had to cast the received object to `MyInteger` type. I did that in line 11 of the code by creating a new instance of `MyInteger` class named `o`. Then I compared the `value` of our instance and that of the newly created one. If they were equal, `True` is returned.



```java
 8   @Override
 9   public boolean equals(Object other) {
10       if (other instanceof MyInteger) {
11           MyInteger o = (MyInteger) other;
12           return this.value == o.value;
13       }
14       return false;
15   }
```

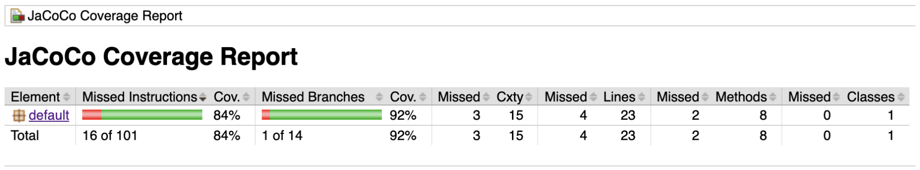
Figure 3: Implementation of equal function

Then I compiled `MyInteger.java` file and run Randoop Test again. Since I implemented the equals function correctly, Error-reveailing tests were all gone. All corresponding files are available under the `4.3` sub-directory.

4.4 Test coverage

I created test-classes directory, compiled the generated test files and put all output classes file to test-classes folder. Next, I generated the coverage analysis.

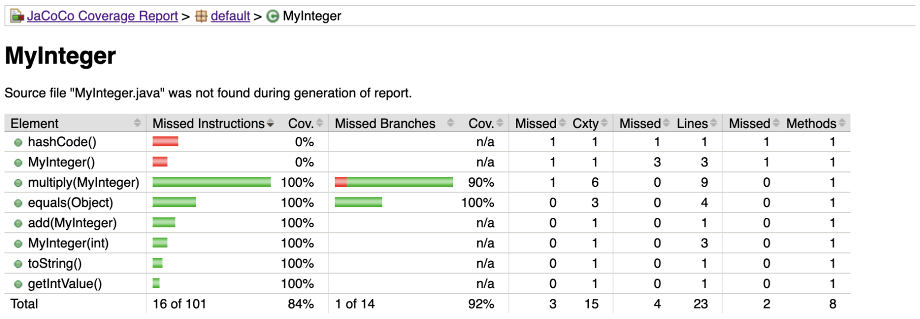
Based on report provided in Figure 2(a), 85 of 101 instructions (84%) and 13 of 14 branches (92%) were covered during runnig these tests.
All related files are available under the `4.4` sub-directory.


JaCoCo Coverage Report

JaCoCo Coverage Report

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
default		84%		92%	3	15	4	23	2	8	0	1
Total	16 of 101	84%	1 of 14	92%	3	15	4	23	2	8	0	1

(a) JaCoCo Coverage Report


JaCoCo Coverage Report > default > MyInteger

MyInteger

Source file "MyInteger.java" was not found during generation of report.

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
hashCode()		0%		n/a	1	1	1	1	1	1
MyInteger()		0%		n/a	1	1	3	3	1	1
multiply(MyInteger)		100%		90%	1	6	0	9	0	1
equals(Object)		100%		100%	0	3	0	4	0	1
add(MyInteger)		100%		n/a	0	1	0	1	0	1
MyInteger(int)		100%		n/a	0	1	0	3	0	1
toString()		100%		n/a	0	1	0	1	0	1
getIntValue()		100%		n/a	0	1	0	1	0	1
Total	16 of 101	84%	1 of 14	92%	3	15	4	23	2	8

(b) JaCoCo Coverage Report > default > MyInteger

Figure 4: Test coverage

5.1 Generate Joda-Time Datetime tests

I generated tests for Joda-time Datetime classes using Randoop and the following command:

```
java -classpath ../lib/joda/randoop-all-4.2.7.jar:  \
../lib/joda/joda-time-2.10.13.jar randoop.main.Main gentests \
--testclass=org.joda.time.DateTime --time-limit=100
```

It generated 4 Error-revealing tests and 1272 Regression tests. As the result, these files were created:

- ErrorTest.java
- ErrorTest0.java
- RegressionTest.java
- RegressionTest0.java
- RegressionTest1.java
- RegressionTest2.java

Then I compiled the generated test using the following commands:

```
export JUNITPATH=../lib/joda/junit-4.13.2.jar:../lib/joda/hamcrest-core-1.3.jar:\
../lib/joda/joda-time-2.10.13.jar
javac -classpath .:$JUNITPATH ErrorTest*.java RegressionTest*.java
-d test-classes
```

and ran them:

`java -classpath .:$JUNITPATH:test-classes org.junit.runner.JUnitCore ErrorTest` ⇒ Resulted 4 failures of 4 tests.

`java -classpath .:$JUNITPATH:test-classes org.junit.runner.JUnitCore RegressionTest` ⇒ All regression tests where successfully passed.

All related files are available under the `5.1` sub-directory.

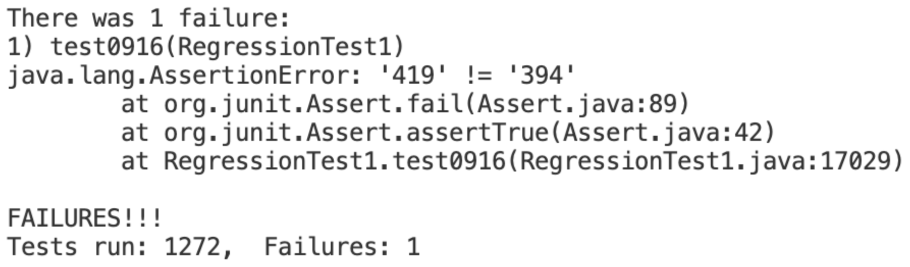
5.2 Flaky tests

Four Error-revealing tests were generated. Based on the definition of flaky tests, which says: ”flaky test is a test that both passes and fails periodically without any code changes,” none of the Error-revealing tests are flaky since I ran them several times and every time all of them failed. But, according to Randoop Manual page, a Regression test can also be flaky if the result of testing changes without changing the code. I ran Regression tests several times and I found that **test0557** sometimes passes and sometimes fails. So, I can conclude that this test is flaky.
In the following lines I will explain most important causes of flakiness and the solution to solve them:

- **Inconsistent assertion timing**: when your application state is not consistent between test runs you will notice that expect/assert statements fail randomly. The fix for this is to construct tests so that you wait for the application to be in a consistent state before asserting. I am not talking about “wait” statements either. You should have predicates in place to poll the application state until it reaches a known good state where you can assert.

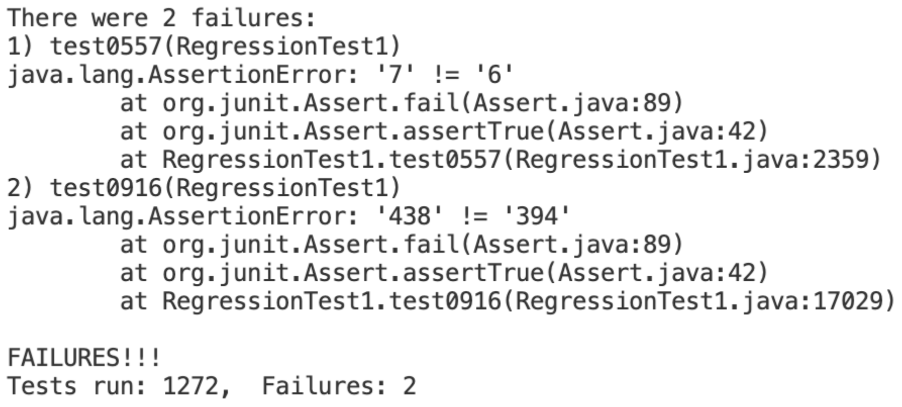
- **Reliance on test order**: global state is the main culprit that causes tests to be reliant on other tests. If you see that you cannot run a test in isolation and it only passes when the whole suite is run then you have this problem. The solution is to entirely reset the state between each test run and reduce the need for global state.

- **End to end tests**: end to end tests are flaky by nature. Write fewer of them. Instead of having 500 end to end tests for your organization, have 5.[1]



```
There was 1 failure:
1) test0916(RegressionTest1)
java.lang.AssertionError: '419' != '394'
        at org.junit.Assert.fail(Assert.java:89)
        at org.junit.Assert.assertTrue(Assert.java:42)
        at RegressionTest1.test0916(RegressionTest1.java:17029)

FAILURES!!!
Tests run: 1272,  Failures: 1
```

(a) test0557 passed



```
There were 2 failures:
1) test0557(RegressionTest1)
java.lang.AssertionError: '7' != '6'
        at org.junit.Assert.fail(Assert.java:89)
        at org.junit.Assert.assertTrue(Assert.java:42)
        at RegressionTest1.test0557(RegressionTest1.java:2359)
2) test0916(RegressionTest1)
java.lang.AssertionError: '438' != '394'
        at org.junit.Assert.fail(Assert.java:89)
        at org.junit.Assert.assertTrue(Assert.java:42)
        at RegressionTest1.test0916(RegressionTest1.java:17029)

FAILURES!!!
Tests run: 1272,  Failures: 2
```

(b) test0557 failed

Figure 5: flaky tests

References

[1] https://engineering.atspotify.com/2019/11/18/test-flakiness-methods-for-identifying-and-dealing-with-flaky-tests/