University of Pittsburgh

Coffee Maker Quest

CS 1699 – Deliverable2: Unit Testing and Code Coverage

John Abraham & Joel Roggeman
Bill Laboon
2/17/2015

TABLE OF CONTENTS:

1. SUMMARY AND TESTING CONCERNS

For this assignment our group wrote unit tests for the Coffee Maker Quest game using the JUnit framework, the Mockito framework for test doubles and stubbing of methods, and the Eclipse tool EclEmma to generate a code coverage report.

One of the issues that we ran into while testing the code was determining what to test. While we could use the project requirements as a guideline to determine if something made sense to test, in general, trying to decide what methods should be tested and what shouldn't was a bit hazy at times (balancing too little testing with going down the rabbit hole takes practice). For example, in Room.java, both getAdjective() and getNoun() were private methods, so we couldn't reasonably run any tests on them without reflection, and even then, a reasonable and accurate test for randomness (the main thing we may want to test) would require repeated calls, statistical tests, etc. Additionally, generateDescription() was private, and even if we used reflection, it would've required either stubbed other methods or hoping that the two methods it called were correct.

Overall, there were a number of void methods that had a bunch of side effects (getCoffee() and the like) that weren't really testable without redirecting System.out or using reflection to check private variables, and there were a number of private methods that made testing the public ones a bit harder because of their dependencies on these private methods.

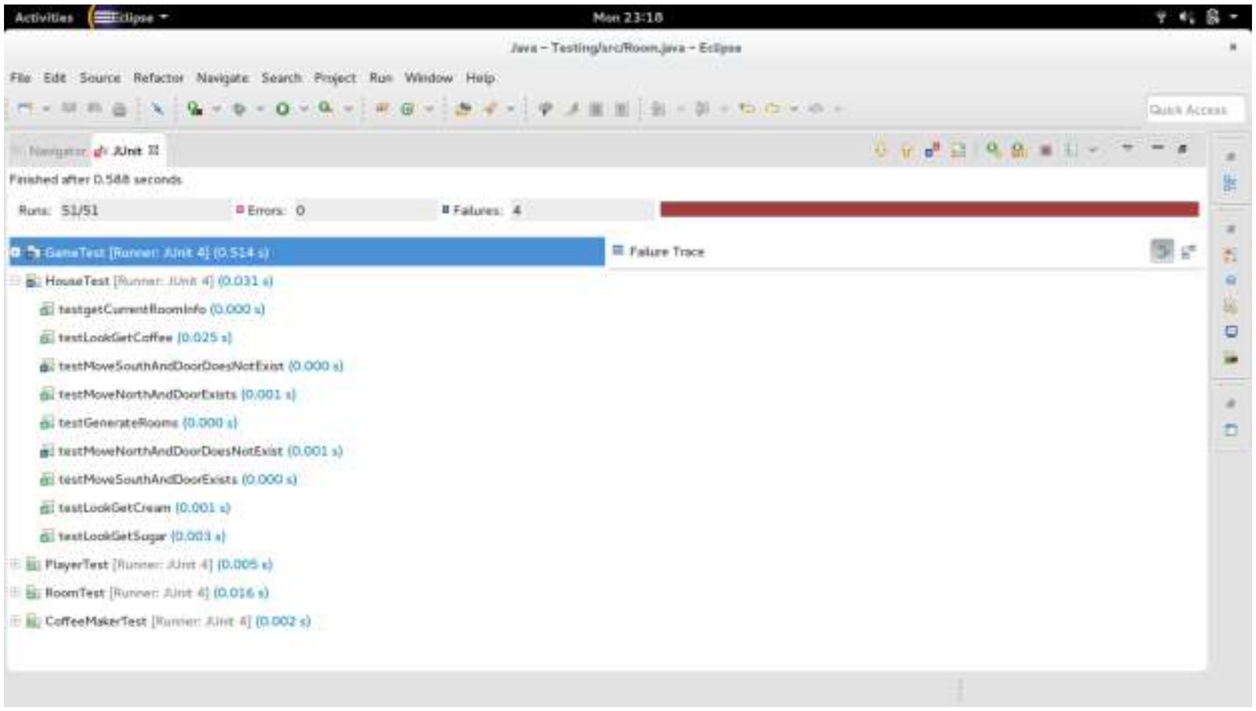
Four of the tests that we wrote failed. These tests were written based on the requirements of Coffee Maker Quest. These failures weren't unexpected because this is not a bug free program as we saw in the first deliverable when we did manual testing on the same program. The tests that failed were: *testMoveNorthAndDoorDoesNotExist*, *testMoveSouthAndDoorDoesNotExist*, *testDoSomethingMoveNorthLowerCase*, and *testDoSomethingHelpCommandUpperCase*. These are the same defects that were found under manual testing. No additional bugs were found. testMoveNorthAndDoorDoesNotExist tests if a player can move North when he/she is in a room in the house (i.e, the North most room in the house) and there is no indication that there's a door going North, and *testMoveSouthAndDoorDoestNotExist* was tested when the player is in a room in the house (i.e. the first room of the house when the program first starts) and there does not exist a door going South. To test those two methods the private variable _currentRoom in the Room class was accessed using reflection and compared to the expected behavior based on Requirement #4 (FUN-MOVE). When the player takes the North exist and there isn't a North exit _currentRoom should remain the same but as observed it changes to 0 and takes the player back to the beginning. Same for the moveSouth method. When the player is in the first room in the house and he/she goes South _currentRoom should stay at 0 but as observed is changes to -1. *testDoSomethingMoveNorthLowerCase* is another test that failed. This test uses a mocked House object to verify that a player moves North when when the lower case character "n" is entered . Requirement #3 (FUN-INPUT-CAPS) requires that inputs be case insensitive and character "n" doesn't let the player move North so this was a defect. Finally *testDoSomethingHelpCommandUpperCase* tests if the input command "H" for help is supported because Requirement #9 (FUN-HELP) requires that there be a help command. House and Player objects were mocked and the return value of the doSomething method was checked and compared to the number -1 because if a command was processed successfully -1 should be returned and 1 is return if the command wasn't processed successfully. The doSomething method returns 1, so the test failed.

Overall, going forward, we expect that we wouldn't have too many problems if we made a few alterations to the code base. First of all, we need to fix the bugs present in the codebase. Second, it might be nice to refactor some parts of the code a bit to reduce dependencies and make them more testable. Finally, if we follow TDD, or at the very least, ensure that our methods are relatively modular and not heavily coupled or dependent on other classes/methods, we shouldn't have a problem writing a fairly robust (and testable) game.

2. UNIT TEST CODE

Our repository can be found at https://github.com/abrahamjj/CS1699

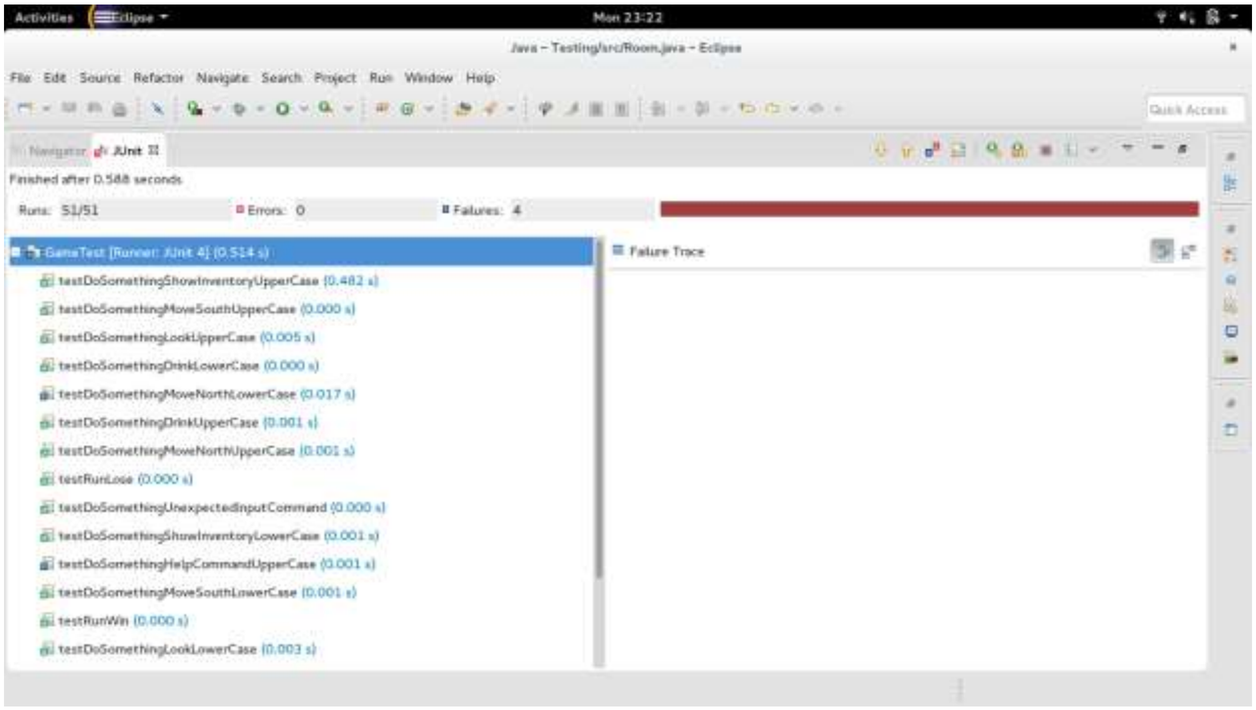
3. SCREENSHOT OF EXECUTED UNIT TESTS



```
Java – Testing/src/Room.java – Eclipse

Finished after 0.588 seconds
Runs: 51/51    Errors: 0    Failures: 4

GameTest [Runner: JUnit 4] (0.514 s)
HouseTest [Runner: JUnit 4] (0.031 s)
    testgetCurrentRoomInfo (0.000 s)
    testLookGetCoffee (0.025 s)
    testMoveSouthAndDoorDoesNotExist (0.000 s)
    testMoveNorthAndDoorExists (0.001 s)
    testGenerateRooms (0.000 s)
    testMoveNorthAndDoorDoesNotExist (0.001 s)
    testMoveSouthAndDoorExists (0.000 s)
    testLookGetCream (0.001 s)
    testLookGetSugar (0.003 s)
PlayerTest [Runner: JUnit 4] (0.005 s)
RoomTest [Runner: JUnit 4] (0.016 s)
CoffeeMakerTest [Runner: JUnit 4] (0.002 s)
```



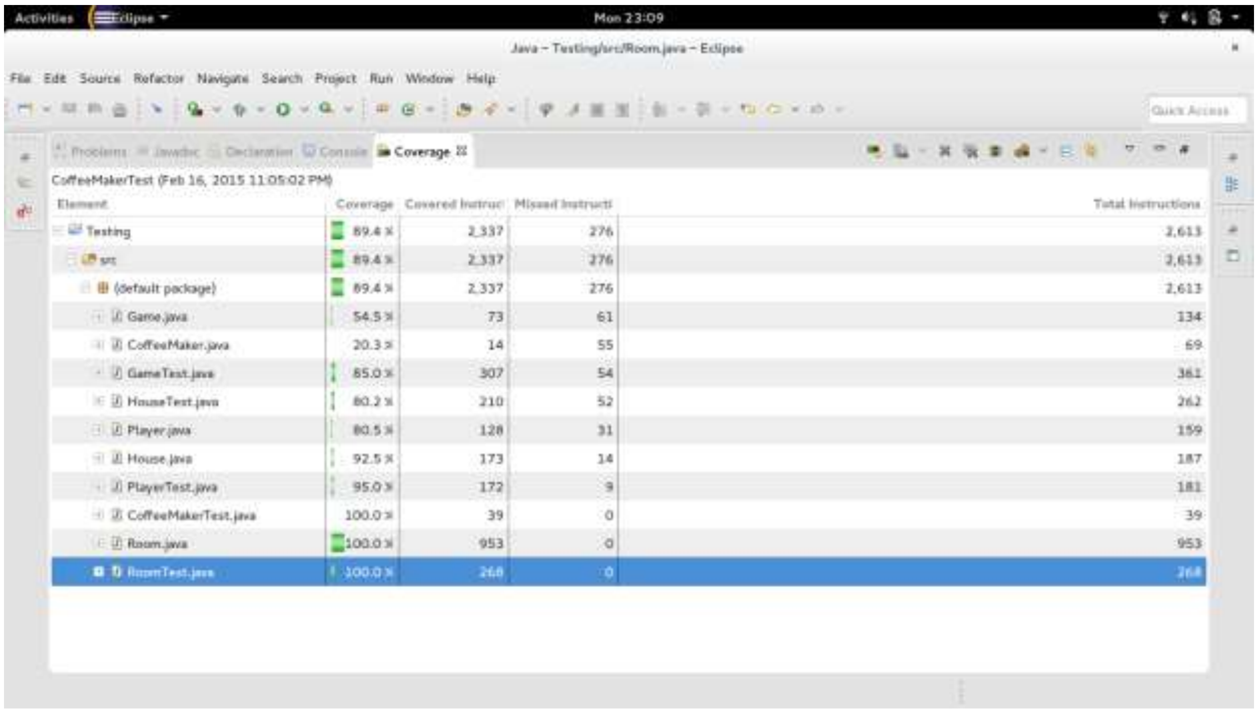
```
Java – Testing/src/Room.java – Eclipse

Finished after 0.588 seconds
Runs: 51/51    Errors: 0    Failures: 4

GameTest [Runner: JUnit 4] (0.514 s)
    testDoSomethingShowInventoryUpperCase (0.482 s)
    testDoSomethingMoveSouthUpperCase (0.000 s)
    testDoSomethingLookUpperCase (0.005 s)
    testDoSomethingDrinkLowerCase (0.000 s)
    testDoSomethingMoveNorthLowerCase (0.017 s)
    testDoSomethingDrinkUpperCase (0.001 s)
    testDoSomethingMoveNorthUpperCase (0.001 s)
    testRunLose (0.000 s)
    testDoSomethingUnexpectedInputCommand (0.000 s)
    testDoSomethingShowInventoryLowerCase (0.001 s)
    testDoSomethingHelpCommandUpperCase (0.001 s)
    testDoSomethingMoveSouthLowerCase (0.001 s)
    testRunWin (0.000 s)
    testDoSomethingLookLowerCase (0.003 s)
```

Rather than including multiple screenshots, we show the two that reveal which specific tests fail.

4. UNIT TEST COVERAGE REPORT


Activities
Eclipse
Mon 23:09
Java – Testing/src/Room.java – Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Problems Javadoc Declaration Console Coverage

CoffeeMakerTest (Feb 16, 2015 11:05:02 PM)

Element	Coverage	Covered Instruc	Missed Instructi	Total Instructions
Testing	89.4 %	2,337	276	2,613
src	89.4 %	2,337	276	2,613
(default package)	89.4 %	2,337	276	2,613
Game.java	54.5 %	73	61	134
CoffeeMaker.java	20.3 %	14	55	69
GameTest.java	85.0 %	307	54	361
HouseTest.java	80.2 %	210	52	262
Player.java	80.5 %	128	31	159
House.java	92.5 %	173	14	187
PlayerTest.java	95.0 %	172	9	181
CoffeeMakerTest.java	100.0 %	39	0	39
Room.java	100.0 %	953	0	953
RoomTest.java	100.0 %	268	0	268

Most of the code is covered fairly thoroughly, except for CoffeeMaker.java. The majority of this class is its main method, which we didn't test.