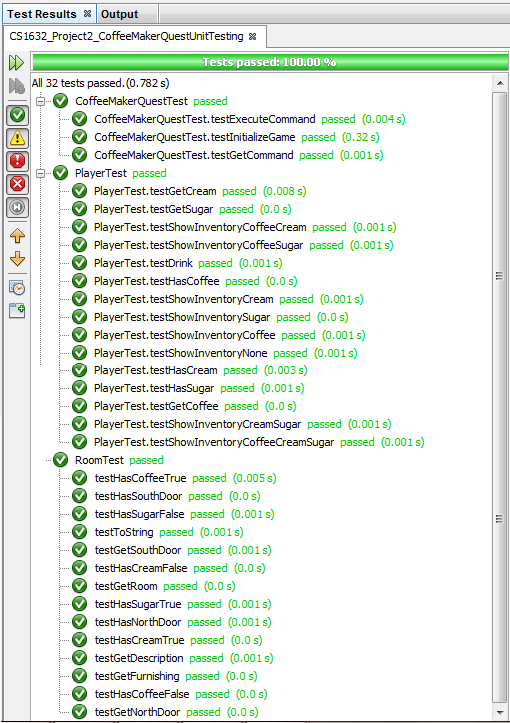
|  |
| --- |
| Coffee Maker Quest |
| CS 1632 — DELIVERABLE 2: Unit Testing and Code Coverage |
| <https://github.com/MeSz/CS1632_Project2_CoffeeMakerQuestUnitTesting> |

|  |
| --- |
| Joe Meszar (jwm54@pitt.edu)  David Sweeney (dms163@pitt.edu)  2/16/2016 |

# SUMMARY

This deliverable was much harder than we had predicted. The process of getting JUnit and Mockito working was a hassle because of class paths being rather uncooperative. Also, my partner and I were utilizing different IDE’s which created problems of its own. The approach we took to the deliverable was to follow the guidelines from class and set each test up so that it failed first, then to write the test so it would pass. The most challenging aspect of this deliverable was the Mock doubles and method stubbing. We created the program with little interdependence so finding places to create mocks was challenging. Furthermore, we can both say that this project seemed to be difficult to test because of its simplicity. Now this seems oxymoronic; but when the program is mostly linear, it does not leave much room for error and in turn does not leave much room for testing fringe cases. The program either works as expected or there is a very noticeable bug in the code (no hidden or missing operations statements, no “off-by-one” errors to look for, no real edge or corner cases, no boundary values).

To boot, we had many discussions about “What is Mockito? Where do we use it? When do we implement it?”. These talks seemed to take just as long as actually implementing the mocks in our code. Overcoming these new terms (and coding strategies) was something I do not think either one of us gave cadence to before we started this project. There is a newfound respect for creating tests within our code, not for the tediousness of it but for the security and faith of the product it creates.

EXECUTED UNIT TESTS 

CODE COVERAGE

