|  |
| --- |
|  |
| CS 1632 – DELIVERABLE 2 |
| Unit Testing and Code Coverage |

|  |
| --- |
|  |

Github URL: https://github.com/swanc12/deliv2.git

Colin Swan – CRS110@pitt.edu

February 16th, 2016

# Difficulties

The largest source of difficulty with this assignment was due to lack of experience writing quality testable code. Uncertainty as to the best practice for writing certain segments of the code was frequently a block to progress. As I started writing the program I attempted to follow TDD practices, which lead me to question certain practices I was used to following. I had to consider what best practice would be for certain situations in which TDD seemed to contradict what I had learned to be “best practice” in java in the past.

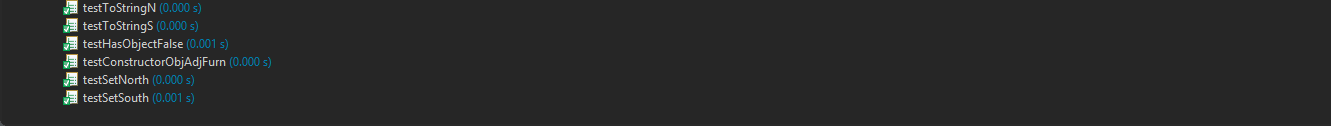
For example, while creating a basic class file for a Room class, I was unsure whether to instantiate class variables as private or public. On the one hand as I was learning to program in Java I was told to always keep them private. But for TDD this would prevent me from decoupling setter and getter methods from each other; if I can’t directly access variables for the class I’m testing then I would need to use getter methods to test the setter methods, which would then have me relying on code that potentially hadn’t been fully implemented yet.

I was able to get around this difficulty by finding a middle ground between private and public. By setting the class variables as protected, I was able to have the test suite for each class extend the class it was testing so that I would be able to access the class variables I needed. This seemed reasonable to me, and I thought I recalled seeing similar practices in the past, so I decided to roll with it. In this way I can keep the variables private from the view of other classes, but I am still able to access the variables in my tests.

Another large source of difficulty came when I began dealing with inputs. I was unsure how to mock inputs, as the Scanner class can’t be mocked. I figured I could mock System.in or manipulate the InputStream going into the Scanner in some other way. However, in the end I realized it might be best to just leave the Scanner in the main method for the program. This helped with testing as I no longer had to mock InputStreams or anything, as I was simply passing in Strings and Chars to all my methods. I felt this also would be a reasonable practice because it makes the program a bit more general and flexible, as none of the methods explicitly require user input, instead using only Strings, chars, etc.

One final source of difficulty came when deciding how to handle returns for methods that mainly were meant to provide output to the user. In particular, the parseInput method seems quite large, and is almost entirely print statements. I considered making this method instead return a String, so that prints would be confined to the main method. However, the main method required a Boolean statement to be returned so that it could determine whether to continue the main loop for the program or not. Because of this I chose to leave the print statements in the method, and used a Junit setUp in a test suite separate from the regular suite that already existed for the CoffeeQuest class. Doing this prevents needless overhead over methods that don’t need System.out to be modified for testing purposes.

# Unit Tests



# Coverage

