**CS 1632 – DELIVERABLE 2:**

*Unit Testing and Code Coverage*

Brandon S. Hang

I faced a number of challenges while completing this project, the greatest of which was producing testable code. I normally make liberal use of private modifiers as a way of data hiding and limiting unwanted (or unforeseen) changes to data. However, these methods would then be rather difficult to test as private methods are not testable by JUnit conventions. Thus, I had to figure out ways to change my code in order for it to be testable while limiting the excessive use of public modifiers. I also had to move many of my methods from the main Java class file in order for them to be testable as well. In the end, I feel as if my code is much more testable now at the expense of being more complex with a greater number of “setter”, “getter”, and public methods than my program originally had.

Another challenge I faced was dealing with “impure” methods. By nature, Coffee Maker Quest is a very impure program as many methods simply print text to the console window. I tried to combat this by having many of those methods return a string (or in some cases, an integer). This string, labeled *lastCommand* in CoffeeGame.java, was strictly used for debugging purposes and allowed me to properly test the gameplay aspect of the program using a litany of test cases.

Other challenges I encountered included simply learning how to use both JUnit and Mockito and also ensuring my test cases covered a significant amount of my code. The former simply took time while the latter required reshuffling of code away from the main Java class file. As it stands now, my main Java class file has very little code in it, aside from the main method.

**My Question:** Is there a magic ratio between the tradeoff of using more public methods for testability over private methods for security?