Members: Gregory Hartranft (gdh12@pitt.edu)

CS 1632 - DELIVERABLE 2: Unit Testing and Code Coverage

Github: https://github.com/Sindalf/CoffeeMaker

The main problem that I had was creating testable code. I had a rather difficult time designing the code itself. I know what I would do, and it would be done quickly and it would work but I don't think it would be very testable. Although I personally don't think it would need to be just due to the simplicity of the program and there not being much logic involved. So designing it become the biggest hurdle of them all.

From the testing standpoint everything seems to be in working order when I finally figured out how to use my IDE properly for testing. Some of the tests were designed before then thus is why @before and @after are not used for System.out tests but I did get a decent work around. Looks ugly but it works. System.in tests I couldn’t figure out a way to test them. Mockito said you couldn’t mock scanner and InputStream with System.setIn() wasn't working. Not sure what to do about this but because of this some of my code in CoffeeMaker is untested. However, I am happy to say that the rest of the code has a near 100% code coverage.

As such CoffeeMaker.testGameLoop() fails due to CoffeeMaker.getInput() trying to execute. All other tests pass.

If I did this again in the future I would probably think less about testable code and think more about code. While that seems counter intuitive I believe a good proper code is testable code. Code with well-defined methods that do exactly what it says they do with proper return values. Using this method, we can easily extend our code or modify aspects of it without interfering with the rest of the code base. If done properly this should lead to good testable code that makes sense to the programmer. In making of this testable code I found myself a little overwhelmed by what I wrote. It seemed like there was too much code for the given job with organization choices made just to be more OOP related. While I can definitely see this being more useful in situations with a larger code base and a program that will grow for our specific situation it's hard to look past its size and coding depth. That's why if you ever do this project again in the future I would suggest something with a bit more meat to the project. For example, the Room object was only used by House object. The room object could have been represented in simpler ways. However, what if there were two Houses or something else that also needed that Room object that was independent of House? In this case this is where the testing aspects would be really useful because from at least my perspective it puts weight on the object itself to be coded in a certain way and to be testable for various types of data and instructions acting upon it all at once. At least that is what I would look at.

Anyway I think what I have learned with this project is rather important I actually had no idea for a while that I could test something intendent of my own program which is something I am definitely going to look into doing more in the future.



