**Owen Cumley CS320 Project Two Code Reflection**

When it comes to how I conducted unit testing for each of the three features it was generally pretty similar between all of them with only some minor differences. The main tests of each feature like the ID, adding and removing of a feature were very similar and mostly copied code with keyword changes to ensure continuity between each feature. This does not mean that I was careless with my testing implementation though I as made sure that each test was testing the proper scenarios along with their fringe cases. This included things like testing an ID with the absolute minimum and maximum allowed characters for an ID so that the code was formatted properly and would not error if a user tried using a fringe case value. Because of this rigorous testing style, I was able to ensure maximum efficiency of my code by covering as close to 100% of all of the cases as possible, making room for the possibility I did miss some test cases.

My experience of writing JUnit tests was very stressful at first and involved a large portion of time just staring at my screening wondering how and where to start. Thankfully, Mr. Parker made some excellent videos on how to start with the JUnit test assignments and the basics of how to implement a test. Those few videos gave me the confidence and foundation that I needed to start implementing the tests on my own. This over time evolved into me becoming comfortable enough that I was able to implement the date test on lines 62-76 of the Appointment feature with little to no issue. Because of this comfort in testing the correct and incorrect values, I was also able to make the efficiency of my tests somewhat high as well, however there is still room for improvement. The most obvious case would be in my AppointmentService class. On lines 18-29, I ran two separates for loops to search for a duplicate or null value. If I had added both of those tests into a single for loop, I could have increased efficiency as only a single for loop would have needed to have been run.

The testing techniques I used where all purpose specific but also rather simple as well. Taking the testing of valid or invalid ID values for instance. Using the assertEquals test when it came to the minimum or maximum allowed characters for an ID fit for the circumstance as I was testing if the value returned by the getId call was the same as the one entered. If the value got caught with an invalid value, then getId would not return a value and the test would fail. While I decided to use the assertEquals test for my ID values, the assertSame test looks like it would also work, but it actually would not. However, using assertSame would actually fail the test as the object being tested is a difference instance, even contents of the string are the same. So while it may be temping to replace assertEquals with assertSame, caution must be taken to ensure that each assertation is used in its proper context to avoid code that does not fail because of the wrong assertation type. When it comes to practical use cases of the testing methods I mentioned, the ones I used inside of my features will generally be used in similar ways during testing. As for the assertSame test, it will generally be used to test of the object returned is the exact same as the object instance that was passed in rather than a new instance of the same value.

The mindset that I adopted when working on this project was very similar to the mindset that I use when it comes to coding most things, ensure that the code functions and meets the required criteria first and foremost and then work in improving and optimizing it after the fact. While there are obviously some optimizations that I missed like stated before, ensuring that a properly functioning code is the absolute most important part has always driven my coding mindset. At the end of the day, it does not matter if the code you made is the fastest and most optimized if it does not function or meet the required criteria. I was also able to appreciate the interrelationships of each feature as the project used very similar tests for each feature, allowing for allot of crossover in code.

When it comes to bias, I tried to limit this aspect as much as possible by ensuring that my tests were as thorough as I could make them by testing fringe cases instead of just regular ones. If I decided to assume that my code worked flawlessly every time, a simple mistake like a >0 in the ID verification to throw an error for a value that is not between 1 and 10 characters long would have resulted in a failed test as an ID with a string 0 characters long could have been passed though. All of which translates to discipline when it comes to coding. Being able to reflect on mistakes that I made in the past and improve on them in the future shows that I am always willing to learn and grow as a programmer. It also shows that I am not going to try and corner cut or implement parts of code that I am unsure if they will work or not.