Project 2

This project was a very interesting and enjoyable experience overall. I utilized many different aspects of logic and learning to ensure that the finished product I presented was a good representation of a program that was well tested and accurate to its requirements listed. The program contains six classes each with their own Junit test. The priority of the project was to create separate parts of a program that ultimately all worked together. With what I programmed, I tried to keep the individual parts similar enough that they would all look relatively similar and function relatively similar for the sake of ease of editability later down the road, while ensuring that the programs were distinctly different in their application. I followed that thought with my Junit tests as well throughout the programs classes, and additionally sought to test all parameters possible where I could. With each of the Junit tests, I tested for the possible values in each fields range. For example, with the appointment class, in terms of the description variable, the parameters were that the description had to be at least one character, could not be more than fifty characters, and could not be null in nature. For my testing, I ensured that I tested in each range, the one to fifty range, the fifty plus range and null, to ensure that the program functioned and responded in the manner that it was intended too.

Given the small size of the potential amount of characters for each of the parameters, I could have gone through and tested each individual character count, and tested away, however, the resulting output would have no doubt been lengthy. By identifying key points in each of the fields, I was able to efficiently and maximally ensure that coverage was correct and accurate. Additionally, based on my one-hundred percent coverage rate, I was able to deduce that my program ran and functioned in the manner that it was supposed to, with the inputs that were given. This experience overall was very enjoyable, as it gave me the means to work through coding with security in mind, which in theory should be evident in my work. In order to ensure that my code was technically sound, I threw exceptions with many of the scripted inputs that would have caused a response in the terminal to be output. I also used best practices when writing the “bones” of the program by inputting in line descriptions to label and detail what I had done in each section and what was still needed to be done. In terms of efficiency, as I stated before, I ensured that my code was written in a similar and minimalistic manner, while ensuring maximal coverage. For example, with the AppointmentServiceTest class, I started out by ensuring each time that my assertions were true and that the code functioned properly with correct input. The only redundancy I have here is that I tested with three different sets of values, with the aims of showing that the program functioned correctly given separate but correct input. I followed up this practice by effectively inputting incorrect details and eliciting a response from the system. This was done three times but using one main focus group in each example, with the other two being correct, so it was easily identifiable. All in all, I believe that the program that I wrote was not only efficient in its testing but effectively covers all the parameters with a one hundred percent coverage rate.

As I reflect on this program and the techniques I used, most of them centered on establishing exactly what the program was and was not capable of. There are plenty of other tests at a programmers disposal, such as compatibility testing, GUI testing, security testing and functional testing, but in this case bounds testing was the main bread and butter of this program. Security testing is ordinarily incredibly important when it comes to testing the validity of a program in the real world, but as this program did not store any sensitive data and all data at that was stored system arrays rather than off system servers, there was no necessity for security testing. In terms of the bounds testing that I did conduct, it is exactly as I stated earlier where I tested for the programs implicit values in the statements written, and then tested for the values not explicitly written in the bounds to ensure the program ran efficiently and effectively.

My mindset going into the project was to write a program that was effective at what was required of it, and efficient in code so that if at any point it needed to be edited further down the line, the area of edit was easy to pinpoint and change throughout. This project was done thinking forward instead of at the time of its inception. By this I mean that at all times of the projects construction, I was writing with the purpose of ensuring the code was of easy to understand language and well following logic. At all points I was cautious of writing something into the program that made it overly complex for the simplicity of the program. The code was very dependent on the writing between the subject class and the subject-service class, where the subject class respectively established the bones for the subject-service class to function and utilize different functions. Writing something that would have decontextualized that interdependent relationship would have made the program more complex than it needed to be. As I was the person testing my own code, I tried to reduce bias at all points of this by not growing overly attached to any particular bit of the program. This way, if I needed to make changes or edits, and if testing did not work the way it needed too, I could easily change and update the code regardless of where it fell and what it was, as I am proud of it all, and not just any singular part. Finally, it is important to highlight the value of staying disciplined in terms of the dedication to quality code rather than fast code. Not that these two things are mutually separate all the time, but quality code is not necessarily always found on a deadline, as shown by many data breaches over the past few years. The dedication to quality is important, and especially so while testing the code as this is one of the last few layers before a program enters a different medium. For instance, if a Microsoft developer were to create a patch for office 365 in response to a data breach, which would better protect against attacks, the customer would often feel better about that patch knowing that it has been rigorously tested through many different testing apparatuses before it goes live than to hear the tester only tested for the code they changed. It is important to not cut corners here and make sure that the rigor of the testing is not scoped narrowly for speeds sake, as it is common to have unintended side effects from changes made within a preestablished code. This often comes as a result of other time related issues such as technical debt, where writing the code takes longer than you may have allotted. Personally, the way I avoid this, is by being overly meticulous in planning as well as utilizing code that I have written to do similar tasks. That being said, the code is of course modified to meet the needs of what I am doing, but overall saves me much time given that I know it works and has been tested. The only thing for me to do from there is to test that it works and doesn’t alter what is preexisting within the program.