**Summary and Reflections Report**

**Summary**

My testing approach aligned to the software requirements exactly as they were written. All parts of the mobile application were created the way they would be expected to in a real-world environment. The objects were designed with encapsulation to control how data was manipulated according to the required rules such as no null values and maximum lengths. The tests covered all parts of the software while also going above and beyond just asserting that a test was successful by giving each test easy to follow descriptive names and categories which could be viewed in the test results. I created the objects and tested every edge case for error including creating an object that follows all specified rules, creating objects with null values, and creating objects with values that were too long and received the expected results. When testing the each service object I made sure that all methods followed the same rules as when using the constructor to ensure that all edge cases were handled when updating the values of already constructed tasks. In addition, the JUnit tests were effective because they covered each bit of functionality across each class entirely (100% coverage).

I ensured that my code was technically sound by going through every object’s methods and ensuring that I got the expected result for their intended purpose. I also made sure to add checks to the methods and the constructor that would throw an error if the specified rules for each data value were not followed. On top of that I made sure to use proper encapsulation so that every data type could be controlled within the methods when used and did not give public access directly to those data types. If you look at all objects within the software, you will notice that all data types were set to private and then handled using public methods, this gives me more control over how the data is manipulated.

I was able to ensure the code was efficient by not using loops where it was unnecessary. All the service objects (AppointmentService, ContactService, and TaskService) use a map to store the objects rather than a list. This allowed me to find already constructed objects via the id without having to loop through an entire list. This is much more efficient as the Java implementation of the HashMap is O(1) search time compared to the ArrayList search time which is O(n). In other words, although the program is not big enough at this time to make much of a difference in speed, if there were a million objects being created and searched then my code as it is now would be much more efficient than if lists were being used. Searching is the most redone task in the code for these services and by not depending on a list, you are effectively guaranteeing good performance with the current functionality of the product.

**Reflection**

In software development, testing is vital to the success of the project. I have employed different testing techniques for each class required by the milestones. I used both static testing and dynamic testing to ensure that my code would work as expected. The static testing allowed me to make sure that my code would compile and minimize the number of bugs that would be found when the code was ran and the dynamic testing let me find any bugs that may have not been found by my static testing. Unfortunately, I couldn’t do black box testing because I wrote the software and so I had full understanding of the internal design of the code.

The static testing that I performed was I manually reviewed the code and followed it logically to verify that I believed it did what its intended purpose was. I also had my ide checking for errors automatically without needing to run any code. This is the most common form of testing because it is cost effective and intuitive. For all companies and individuals this is an easy and important type of testing for all software projects. IDE’s usually come with some sort of automated analysis for your code and most programmers naturally review their code while writing it, so this is very practical.

For dynamic testing, I wrote unit tests to ensure that my code ran correctly and efficiently. I was unable to do system and integration testing at this time with the application. Unit testing is both practical and critical for companies that need to test their products without completely deploying and configuring the software. I used it to test each individual method of each of my objects as well as their constructors to make sure that everything worked before the product was deployed and potentially caused harm to the company. A company like Facebook would use unit tests to make sure that adding and removing friends is working properly with a test service/database for example before pushing the code to a larger test system or production environment where bad code could cause permanent damage.

My mindset when working on this project was rather standard. I didn’t necessarily have to employ caution when creating software tests because the functionality was simple enough to test without worry of affecting the system negatively. What I mean by that is, if I were to create a system that connected to an external database, then I would have to be more cautious with my tests and how I created the data objects in the test code, so I did not corrupt external data or cause issues with external dev systems. I did have to use caution when creating the Service objects’ methods because they had to follow the same rules as the constructors of the underlying objects that were storing. I limited bias by designing the code in a way that would be in a design spec from a project manager rather than designing it in the quickest way I could personally come up with. The tests also limited my bias on what “good code” was because I had to test every edge case to make sure it met all the requirements. Had I not designed the code the same way I would expect my coworkers to be required to write the code then it could have possibly been a concern. You must be disciplined in your commitment to code quality because if you aren’t then it creates expensive development costs later down the line. If you design your code in a way that is not extensible and in a way that only you can understand it easily then you have created more problems than solutions for your employer. The same goes for writing tests, something may seem trivial and not worth the test but the risk of a small bug being created due to even a typo somewhere could create a domino effect that turns into a big problem on production. This is the reason that I didn’t make my datatypes public and re-added the rules to the service methods because if I allowed direct access to variables or didn’t recheck the input being passed with the service objects then bugs or inconsistencies could be introduced after the objects are created, rendering the rules in the constructor basically useless.