Module 7: Project 2

During the development of my code, I always kept the software requirements in mind to give myself an idea on what I am testing my code on and what restrictions were assigned to each String field. Each ID (whether it was the contact ID, the task ID, or the appointment ID) weren’t able to be updatable since the ID represented the overall contact or task. However, the components of each ID (such as the name, date, phone number, address, and description) were all able to be updated so I created setters for each variable to be called upon each contact, task and appointment. My contact/contact service tests and task/task service tests both aligned with their respective classes by grabbing the non-updatable IDs, updatable names (contact input first and last names respectively, tasks input full name), updatable date, updatable contact phone number and address, and updatable task and appointment description.

My JUnit tests for the contact service were a success because of the two contacts and two tasks inputted. Contact ID “1234567890” was such a success because it fulfilled the requirements of having each the first and last name being fewer than 10 characters, the phone number having exactly 10 characters, and the address being less than 30 characters. Contact ID “2345678901” had the same positive outputs, with different components within the ID.

I made sure to format my code in a way that makes it make sense and keep that same analogy and order of code that makes it make sense. For example, in the testTask function, I started off by declaring a Task variable task to form and test a new Task under ID “4567890123”. A few other tests included testing when either the task ID, name and/or description is null or too long, which go against the software requirements of the program. The tests all passed because each different function included a component respective to that function.

My code overall was efficient because I would keep my code organized and in the right place and would copy/paste any code that repeated within the next few lines, or any that needed to be tweaked a bit. For example, the line of code within the Appointment.java class

if (appId == null || appId.length() > 10) {

throw new IllegalArgumentException("Invalid appointment ID.");

}

was deemed repetitive the following several lines for date and description so instead of typing this line of code over and over again, I simply copy and pasted this line and changed the variable and comment line that was being used in the next few lines.

testing, Performance testing, Unit testing, and User-Acceptance testing. For Milestone One in the contact class, changes within the class included adding, deleting, and updating first/last name, phone number, or address. In the JUnit test classes, I was required to make use of JUnit 5 testing to make sure my code ran properly. I mainly used this method of testing in Milestone Two and Three as well since I found this method extremely effective because of the quickness of test coverage in each test ran. It literally took less than a second for each successful test to run. I found it a little harder at first for the ContactService class to run a successful test but after fixing a part of my code where the setters come from within the public Contact class, I found that the tests were finally successful. It took me up until Milestone Three to figure this out. Afterwards, I changed the lines of the setters within each the Contact, Task, and Appointment public classes and as a whole my tests were running at a successful rate. I also tested for any conditions that happened depending on if the names and/or descriptions were over the character limit or if any of the information was null. Those tests also ran successful as well, for example testing a description that had over 50 characters when the limit is only 50.

Some testing techniques that I did not use for this project mainly tie to the behavioral aspect of code, which include Functional testing and System testing. I found that these type of testing techniques mainly occur when a program has to deal with a lot of projects instead of just the three milestones that we were required to complete. I’m sure that down the path of my future courses that I take and from there on in my career I will be focusing more on functional testing and improving on my non-functional testing experience that I have gained in this project.

As I was completing this project, I made sure to keep the customer requirements up on my other screen to make sure those requirements are met. As I said in our earlier discussion in class, I feel that requirements elicitation is the most important because gathering requirements is crucial for a software development project. I wanted each test to reflect on those requirements so it can show me as a programmer that these tests won’t work if a certain requirement is not met. I have been gaining more knowledge on staying focused and disciplined after each milestone project and learning on different techniques on how to conduct these certain classes each week. I liked utilizing the Date class in Milestone Three because it brought a new twist in my project and offered new ways on how to finalize my project for Project One. For future projects, I plan to look more into requirements even more and take any feedback necessary to improve on my code and avoids errors as much as possible.