Corey Gaspar

10/3/2025

**Module 5 Journal**

For the three milestones that I’ve completed and submitted these past few weeks, I have primarily utilized unit testing with JUnit. By using unit testing as my main testing method, I was able to verify that each unit, or individual piece of my program, behaved as expected according to the requirements. For example, for the Module 3 milestone, I created tests for the Contact and ContactService classes. These tests were used to verify that the contact ID, first name, last name, phone number, and address met the required restraints. I also included tests that ensured that invalid IDs and inputs were properly rejected.

The same applies to Module 4, where I tested the Task and TaskService classes. I created JUnit tests to ensure that each task had a unique task ID that was not updatable, null, or longer than 10 characters, a name that was not null or longer than 20 characters, and a description that was not null or longer than 50 characters. My tests also verified that tasks could be added, updated, and deleted properly, while invalid inputs and IDs were rejected.

In Module 5, I used the same approach, where I tested my Appointment and AppointmentService classes. The goal here was to create JUnit tests that verified that each appointment had a unique ID that could not be null, changed, or longer than 10 characters, a date that could not be null or in the past, and a description that could not be longer than 50 characters. To test the date, I created helper methods that took the current date and grabbed the date of the day ahead and the day prior. My tests verified that appointments could be created and managed correctly, while invalid IDs, dates, or descriptions were rejected.

Unit testing allowed me to isolate each class and method, which overall made it much easier for me to test as many edge cases as possible. I was able to verify that the program handled both valid and invalid inputs properly. Whenever there were any changes that I needed to make, I was able to rerun the tests with no errors or failures, which helped me confirm that my program was meeting the requirements. By utilizing unit testing for all three milestones, I was able to catch errors early on and verify that they all met their requirements.

One testing method that I didn’t use throughout my milestones is integration testing, which is a testing method that focuses on verifying that individual pieces, or units, of the program work as expected within the larger system. This method is especially useful within bigger programs that deal with interactions between different services, such as a front-end application communicating with a back-end API. I did not use this method because the milestones focused on the individual pieces, however I plan on utilizing it for the final project.

Another testing method I did not use is acceptance testing, which is a method that checks whether the software meets the requirements of the customer’s or end-user’s requirements. This type of testing is typically used in real-world scenarios and is used to determine if the final product meets the overall requirements of the client and end-user. Since I have only built small pieces of my project, I have yet to really utilize this testing method. I plan on utilizing this during the final project to ensure that the completed program works correctly from start to finish.

Unit testing is practical for the early stages of development, where the goal is to make sure that each individual requirement works as expected before combining it into the full system. It helps developers catch small errors early on into the project, reduces debugging time, and ensures that each piece of code meets its specific requirements. Integration testing is most useful when multiple components or services need to work together, such as when connecting a user interface to a database or external API. It helps identify problems that occur when different components need to work with each other, which is especially important in larger or more complex projects. Acceptance testing, on the other hand, is used near the end of development to verify that the finished product meets customer or user expectations. This testing method ensures that the software performs correctly in real-world situations and is ready for release. Each of these testing methods plays an important role depending on the project’s size, stage, and goals. When used together, they create a more reliable and high-quality software product.