Grand Strand Systems:

Contact Service Summary

Chris Tapia

December 12, 2021

Overall, my unit testing approach was solid as I utilized the Assertions class to run all of my tests. I also based my tests to align with the business requirements. By doing so, I ensured that the tests checked for error through input length if it was too long, if an input was null, adding contacts and the method for deleting those through assertAll() and its lambda expression. It was very helpful to view the coverage percentage of the tests that I ran. This showed the overall quality of the tests by ensuring that I tested each applicable method. This is the definition of unit testing, testing every single unit/method. I also made sure that these tests would be efficient by testing for every input parameter and ensuring it matches through assertEquals(). I also achieved this checking to see if the input validation for the object generation was invalid by using an assertThrows(IllegalArgumentException.class) and throwing in values that would certainly throw an exception when validated.

On the topic of implementing software testing techniques while maintaining efficiency, I implemented HashMaps to make the system efficient which resulted in the reduction of code complexity and validating what the system takes in. I also implemented secure coding techniques such as input validation for the objects that get created and setting up errors to be thrown. This was vital for when I executed unit tests to see if I can recreate those errors. When testing, I found that using testing like the Assertion methods, made sure that I could test for each requirement. I used techniques that primarily placed its attention on the class of creating objects using assertEquals() to confirm those objects were being made accurately. I also made sure to make the class variables private to instead use getter and setter methods to manipulate those variables which led to having another layer of security.

Throughout my progression of each service type, I separated the tests from the main classes so that they would be in their own suite of tests. For the AppointmentService class, one of the requirements were to validate the input of a date field to ensure it wasn’t a date in the past. To achieve this form of input validation, I used the same design as the others for validating input and then I utilized the Date class to compare dates and the SimpleDateFormat to convert the string input into a Date object that can be compared to.

Throughout the development process, I employed caution the most when it came to testing. I wanted to be sure that if the test results came back positive with a green check, that it wasn’t a false positive. Therefore, for testing, I added in my system print statements so that I can view the methods were doing what I wanted them to do via the console. By focusing heavily on both good testing and printing out the correct data to the console, I inadvertently avoided some bias. In a way, it was me not trusting the test results were truly positive until I double checked it through print statements. This helped a lot because in the beginning I was receiving positive test results but when I went to check it through actual print statements, the methods weren’t doing what I needed them to do.

It is vital, as a software engineer, not to cut corners and remain disciplined regardless of how tenured I may become. The quality of design I produce is a direct reflection on myself. I want to create code that’s clean, efficient, and secure but also code that is reusable and easy to maintain. By doing so, I’m creating, not only a brand for myself, but I’m setting a standard/example for other software engineers in the years to come.