Brandon Mullins

CS-320

7-2 Project Two Submission

Southern New Hampshire University

For each of the three milestones – Contact, Task, and Appointment I took a direct and detailed approach to unit testing. I stuck closely to the requirements that were given and tested both valid and invalid input values. My main focus was making sure that each class met all constraints, such as length limits, null checks, and immutability were required. My testing approach used white-box testing, meaning I knew the internal logic and wrote tests specifically targeting those conditions. For example, in the ContactTest, I tested all five requirements: ID not being null or over 10 characters, first and last names being under 10 characters, phone number being exactly 10 digits, and address being no more than 30 characters. I had one test to confirm a valid object could be created, and others that checked exceptions were thrown away when requirements weren’t met. That same method was applied to TaskTest and AppointmentTest.

I know my tests were effective because the code coverage results backed it up. My service classes all reached 100% coverage, and my model classes were in the 75-90% range after making improvements. For example, my original AppointmentTest didn’t check if the date was actually stored properly or test an ID longer than 10 characters. After getting that feedback, I updated the tests to include assertions that directly check those values using assertEquals.

One thing I focused on was keeping my code technically sound. In every model class, I used setter methods inside constuctors instead of repeating validation logic. That way, if the logic needs to be updated later, it only needs to be done once. In the appointment class I used this structure:

> this.id =id;

setAppointmentDate(appointmentDate);

setDescription(description);

This helps keep my code clean and avoids repeating validation conditions. Also, I used defensive copying when returning Date objects in the getter method of avoiding external modifications of internal state. I also worked on keeping the test code efficient. I reused valid inputs by making constraints for valid IDS, names, phone numbers, and dates. This made the tests earlier to read and update. For example, in AppointmentTest, I created methods like getFutureDate() and getPastDate() using Calendar to make the tests dynamic instead of hardcoding specific dates.

The main testing techniques I used were unit testing. Each test targeted one method or one scenario. I also followed the AAA pattern in every test and gave them descriptive names like testAdd\_withDuplicateContactID\_throwsIllegalArgumentException() to clearly state what was being tested. Another technique I used, even if it was indirectly, was functional testing. I didn’t test the UI or full application behavior, but I did make sure the services did exactly what the requirements expected, like not allowing updates to IDs or rejecting invalid data.

I had to shift my mindset for this project. I wasn’t just writing code; I was reviewing it from a tester’s point of view. I tried to be cautious and did not assume that my code was going to work perfectly. I made sure to test failure conditions and I also validated positive cases to confirm that expected behavior happened.