CS-320 Project Two - Summary and Reflections Report

Owen McCostis

For each of the three features (Contact, Task, and Appointment), my unit testing approach was strictly aligned with the specific software requirements outlined by the customer. I used them to guide the development of my test cases, which helped me focus on testing the key functionality and behaviors of my code. An example of this was in ContactTest.java, where I confirmed that an exception is thrown for a contact ID that is longer than 10 characters:

```
assertThrows(IllegalArgumentException.class, () ->

new Contact("12345678901", "John", "Doe", "1234567890", "123 Main St"));
```

This test was directly based on the software requirements, which enabled me to verify compliance with them. I tested both positive and negative test cases whenever input was received and made sure to include edge cases to confirm that the customer-defined constraints were properly implemented.

Overall, I would argue that the quality of my JUnit tests was high. I achieved strong coverage across all classes by designing tests that covered each behavior. I also tested data validation and exception handling. Additionally, I made sure the code was efficient by breaking down the functions and testing each one independently. An example of this was in TaskServiceTest.java, where I isolated and tested the updateName function to ensure it worked exactly as expected:

```
service.updateName("001", "Updated Task");

assertEquals("Updated Task", service.getTask("001").getName());
```

My experience writing the JUnit tests was very structured. I followed a pattern of validating inputs, testing each function individually, and including a negative test. This approach helped ensure that the code was both technically sound and efficient. For technical soundness, I wrote tests to check if the validation rules were correctly enforced. For efficiency, I reused test objects whenever possible and used `@BeforeEach` to avoid redundant object creation.

I employed unit testing, boundary testing, negative testing, and positive testing for this project. I isolated and tested individual components of my code with unit tests, I tested input constraints with boundary testing, I ensured invalid data is rejected with negative testing, and I confirmed that valid data behaves as expected with positive testing. I did not use integration testing, where interactions between modules are tested since they were each tested independently. I also did not test user interface/experience or system performance. Unit testing assists with regularly testing the components of a system throughout development, boundary testing is vital for ensuring compliance with constraint requirements, and negative testing identifies improper handling of invalid user inputs.

Throughout the project, I adopted the mindset of a software tester, approaching each test with skepticism and caution. My top priority was validating the requirements provided by the customer. Each field was treated as though it could break the entire system and both expected and unexpected user behavior was considered. In AppointmentTest.java, I made sure to not only check that the appointment date was valid, but also that it stayed valid after being returned from a getter.

To limit bias, I tried my best to never assume anything. Even when it seemed as though something was working properly, I tested it from the opposite angle to catch any mistakes I might have missed. It’s easy to assume your code is perfect when it seems to function and skip over negative tests, so I prioritized failure testing when creating my JUnit tests.

Being disciplined in your commitment to quality is central to being a software engineer. Lazy testing practices will likely lead to more problems down the road, especially if you are handing your code off to someone else. To avoid technical debt, I wrote complete tests, handled edge cases, and ignored the temptation to cut corners. An example of this is my inclusion of defensive copies in Appointment.java, which prevent bugs caused by accidental changes to a shared Date object.

This project gave me hands-on experience with the full development and testing process. By keeping both efforts closely aligned with the customer’s requirements, I increased the likelihood that the services I designed will meet their expectations.