Jerry Barboza

CS 320

Module 7

Project Two

Summary

For the ContactTest.java, the first test I created was to test the success when creating a new contact. The success of this contact had to follow the requirements and code from Contact.java. For a success, the ID, name, last name had to be at most 10 characters each, the address string at most 30 characters and finally phone number must be exactly 10 digits. If any of this fail, it will cause a failure in the code therefore I also did tests for when these ends failing.

```
// fails to Contact ID since there are more than 10 characters
@Test
void testCreateContactContactIdFails() {
      Assertions.assertThrows(IllegalArgumentException.class, () -> {
          new Contact("12345678901", "Harry", "Potter", "123 Main Street", "7141234567");
        });
}
// fails the Constact First
@Test
void testCreateContactFirstNameFails() {
      Assertions.assertThrows(IllegalArgumentException.class, () -> {
          new Contact("1234567", "Harryyyyyyyyyyyy", "Potter", "123 Main Street", "7141234567");
        });
}
```

Then for the ContactService.java file, I created some tests that would add the contacts success and show when they would fail. For example, when all the inputs where correctly inputted, it would add the contacts as a success. Some ways it would fail is when there were duplicates contacts with similar exact same contact ID. Then I added a test to see the success when deleting a contact and when there was a failure deleting it the invalidContactId would equal to True. Similarly, I did the same thing to tests for TaskTest.java and TaskService.java. The only difference was that the task class requirements now had a description with a string of no longer than 50 characters, a name string of no more than 20 characters and a task ID with no longer than 10 characters.

For the Appointment tests, I programmed tests that helped me see if the application service is adding appointments with unique appointment ID's. Also, some of the tests was to see of the appointment service is also deleting appointments per appointment ID. On the file AppointmentTest.java, the first test I made was to test the success when creating an appointment. Then next test I made was to see appointment date fails. The failure would be if current date was not used. Then I created a test to see description fails by having over 50 characters since the max characters it can have been 50 therefore over 50 will give us a fail. Then I created a test to see the success when updating an appointment. Then another fail test when updating a date from the past. Then on the Java file: AppointmentServiceTest.java, I made a test to see the success of adding multiple appointments so in that test I created appointment1 and appointment2 with different ID and different descriptions as shown below.

```
@Test
void testAddMultipleAppointmentSuccess() {
    Appointment appointment1 = new Appointment("123458", currentDate, "Hello my name is Ron");
    Appointment appointment2 = new Appointment("123459", currentDate, "Hello my name is Harry");

    assertTrue(appointmentService.addAppointment(appointment1));
    assertTrue(appointmentService.addAppointment(appointment2));

}
```

After this I made a similar test however instead of success it would give us a fail when appointmnet1 information was the same as appointmnet2.

The ways I made sure my code was technically sound and efficient was to follow the requirements carefully and understand how JUnit tests are doing when testing my code. The first solution given to us by the instructor helped me really understand my mistakes for the first assignment since it was my first time writing a JUnit tests and once I had a better understanding of it, I was able to approach the other features with a better understanding on what tests to use.

Reflection: Testing Techniques

Some of the software testing techniques that I employed to this project was having both success and failure tests since this helps us have a better understanding of how well our code is to prevent any problems later when deploying the software. Other techniques that can be used for more complex applications that I did not used for this project is performance testing. Performance testing measures the performance of latency, cpu, memory and more to make sure our services scale properly as the load increases.

Reflection: Mindset

Caution

The mindset I adopted to this project is to be caution for errors, successes and fails of the tests. The mindset I had was to make sure all the test files have enough success and fails tests without being biased. I followed similar patterns to all three features while following the requirements.

Bias

It is very important to limit bias in our code for our results are as clean as possible. Bias can also cause us to not explore further causing us to completely miss the problem. One way we can also minimize bias is to test for fail and success tests since only testing positive tests can cause us to miss problems that a negative test would of show us. In my code I tested both success and failures tests to minimize bias and get better clean results.

Disciplined

Our most recent discussion we talked about how bugs in the code and lack of tests ended up costing millions of dollars to come organizations and other ones even causing harm and death to people. It is very important to be disciplined in our commitment to quality as software engineer professionals and we should NOT cut corners when it comes to writing or testing code since this

can ended up becoming a big problem in the future if bugs and other errors are not detected early. One example is the Hitomi Satellite: Japan's US $268 million blowout. In this incident, the satellite was destroyed in space because of a glitched that scientist didn't detect early causing the satellite to spin to its' destruction. If the scientists have spent more research and testing it before launching it to space, it would have been avoided. Therefore, it is very important to always send more time testing the code to avoid technical debt.

Citations:

https://medium.com/@coderacademy/when-coding-goes-wrong-e46d84c6565f

https://smartbear.com/blog/bias-and-the-human-side-of-software-testing/