Summary and Reflections Report

**Summary:**

Graphical user interface, text, application, Word

Description automatically generated During my time creating the code for the three main classes, Contact, Task, and Appointment, I implemented JUnit tests for the main classes as well as the service classes. This was done to ensure that proper values passed when creating the various objects and to ensure that when I passed an invalid value, then an error was thrown. The Junit tests that I created helped to ensure that all software requirements were addressed and that the code functioned as it was supposed to keep everything aligned with the requirements. For example, when I created the Contact, Task, and Appointment classes, each of the objects was supposed to have a unique ID that was passed during their creation. One of the requirements for all three classes was that this unique ID must be less than or equal to 10 characters long and it must not be null. To ensure that this was the case, in the constructor, I added code to say that if the ID was greater than 10 characters long or it was null, then an exception would be thrown with a message “Invalid ID.”

Then, I created Junit tests to show that when I pass an ID value that is not null and is 10 characters long, the value could then be asserted to exist in the object as passed. I also created code to show that if I passed a null ID or an 11-character ID, an exception could be asserted to be thrown. You can see that the tests created assert that the values pass when proper, and exceptions are given when the values are not.

Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated with low confidenceGraphical user interface, text, application

Description automatically generatedFor each method in each class I created, I made sure to create a JUnit test. The JUnit tests for each method tests all values and operations worked as they are supposed to. I did this one by one to ensure full coverage of code was tested. I then ran the tests to ensure that everything passed, and since I created individual tests for each method, I was able to verify all requirements were met. This showed that the quality of my tests was good enough and that my tests were effective. It was also each to show that my tests were technically sound. This is due to the fact that the JUnit tests show pass and fail results, as well as errors. Since there were no errors when I made tests, I was certain that the tests themselves were technically sound. The pass and fall results also pointed to the fact that the tests themselves were made correctly.

Graphical user interface, text, application, email

Description automatically generated Each of the tests was made with a unique name that described what it was testing. This was done to ensure that everything in the code was being tested. The tests were each very minimal in the code required for each of them, so each aspect being tested was done one line per test.

**Reflection:**

There are a couple main testing techniques employed in my code. The first technique is my own ability to see errors in the code as I created it. This is done by eye as well as by the coding interface, Eclipse. If there was an error within my code that would cause a major issue, it was pointed out by Eclipse. I was also diligent in my work. What I mean by this is that I meticulously went through the requirements one by one to ensure that each of them was met at every step of the development of my code. We will call this first technique Human Testing. Aside from Human Testing, I also employed JUnit tests. This form of testing is well described in the above section of this document.

There are other types of testing that were not done in my development. This is due to the fact that these techniques were not needed Integration testing is useful because it helps to ensure that the connections between the various components work correctly. They must take the correct inputs, and they must produce the correct outputs consistently. System testing is important because we need to be sure that when the entire system is put together, with all components assembled, that each method is accessible and works as it should. There are various techniques that can be used in software testing. Some of them involve running the code and trying various inputs, like boundary value testing. That is when you try inputting values at the boundaries of the acceptable values for different variables.

Graphical user interface, text, application, email

Description automatically generatedDuring my development of this project, I adopted the mindset of both a developer who wants to create correct code that meets requirements, as well as a tester who wants to make sure the developer didn’t mess up. I was careful throughout my work to make sure that every requirement was met on both sides of the work I did. This was because I wanted to be cautious of mistakes. I created multiple classes, each with a variety of variables and requirements for each of those variables. On top of that each class had its own methods that needed testing. There were a lot of moving parts in the project, so I needed to be diligent. This is why each test was giving a specific name to remind me what I was testing.

I understand that there is the potential for me to be biased in my review of the code. This is why we employ the various software testing techniques. This is also why I was diligent in my work. It is important to be disciplined in software testing because there are many things that need to be covered and the goal is for the code to be secure, efficient, and correct. Without discipline, we might make errors which will lead to bad code. That can cause issues once software is released. This is why I will practice diligence in my work as a software engineer.

Citations:

No outside sources other than information in my own head were used.