**Summary and Reflections Report**

Created by Sarah Warden

August 17, 2025

My approach to the unit testing for this project was to utilize the JUnit testing method. With this method, I can write a test for each software requirement and method in each class of the project. This way, I know whether each individual part of the code works and if the entire class is covered with all the written tests with the run as configuration application. For example, we needed to Task class to have a task ID that could not be null or more than 10 characters. I wrote a test into both the class and a separate test class to check whether these requirements were met. If they were, then the ID would be created, but if they were not, then an exception argument would be thrown with an output of my choosing. I have supplied the example code for this below:

// check ID no more than 10 characters & not null

        if (taskID == null || taskID.length() >= 11) {

            throw new IllegalArgumentException("Invalid ID.");

        } // end if

This way of testing the requirements was done for each system necessity that was given to us when creating the code. When running the unit tests to make sure that the code is working correctly, we have the option to run as a simple JUnit test to check for failures or errors, or to run with a configuration percentage. Doing the testing this way lets us check what percentage of each method in each class is checked, whether it passes, and if the entire class is checked for functional requirements. By utilizing the configuration percentage, I am confident that my testing is complete and effective.

In order to check the technical soundness of my code, I utilized the assertions package in Java. The assertions package allows me to check the attributes for each class and ensure that they are actually being set with what we input. I have selected the setContactAddress code as an example:

void testValidSetContactAddress() {

        String testID = "5";

        String testFirstName = "Sarah";

        String testLastName = "Warden";

        String testNumber = "1234567890";

        String testAddress = "123 Unicorn Lane, Candy Land";

        Contact testContact = new Contact(testID, testFirstName, testLastName, testNumber, testAddress);

        testContact.setContactAddress("Testing");

        assertTrue(testContact.getContactAddress().equals("Testing"));

    } // end valid address test

As you can see, the contact is created with a set text for each attribute. Once the contact is added to the contact list, I updated the contact address to a new entry, then checked to make sure that the new entry was what would be returned when using the getter method to check the entry. I combined some test cases to ensure efficiency within my code. For example, when testing for null attributes, I simply passed an entry with all attributes as null, rather than writing a new test for each attribute as null, with the others following protocol. This made a potential of five tests combined into one singular test.

For this project, I utilized the JUnit testing technique. As mentioned previously, this method allows me to write a specific test for each method of a class, ensuring that all the code works properly and that all software requirements are met. For example, in the AppointmentService class, the project required an appointment to be added to the calendar, and also deleted. I first wrote a test to check that the appointment list started as empty, then added the class and checked that the size of the list was now one. After removing the appointment, I finished the test for this section by checking that the size of the list was once again zero.

There are a variety of other testing methods that we could have used and all have their benefits and drawbacks. The testNG method is designed to function best with the most modern software development and is used to test code in an end-to-end fashion. This method is great for code that has many dependencies, such as regression testing. This is helpful for code that will continuously be updated and worked on, since we all know code is never fully “finished” as improvements can always be made.

I would image that there would be a good amount of bias in testing your own code. Similar to when you proofread your own paper. Since you know what you wanted to say, sometimes your brain will read what you intended and not what you physically wrote out. I am not sure this would translate exactly into code since the code may not function if not written properly; however, it would be difficult to decide if there was a more efficient way to write the program. That is, assuming I approached the project the way I thought was best, proofing would most likely not change my opinion, unless new information had been gathered that greatly changed the scope or requirements of the project. Being disciplined is very important when writing a program. For example, in this project, I did my best to write each class in the same format and order, so that they would be easiest to follow if a new developer or tester needed to go in and edit or proofread them. Part of this discipline, in my opinion, is making sure there are accurate and consistent contextual notes throughout. The code we wrote was not outrageously complicated, but imagine if the code you are working with had multiple nested sections. It could become difficult to keep track of which brackets and parentheses close which and where your code is running into errors. This is why I make sure to comment what each closing brace, bracket, etc., is closing, ensuring a little confusion as possible.