When starting the codes to meet the software requirements I focused first on making sure that the information inputted is correct. I made it so that the codes will ask for the information that is needed to run the JUnit tests such as the name, address, ID, phone number, and description the user inputs. For the most part all three different codes that I created such as the Contact, Task Service, and Appointment shared the same structure for the JUnit tests. The information inputted for each one was different, but they were used to check if the information is correct by comparing it with an example I provided and that they are within the number of characters they can input for the system to check. Some of these examples are as follows: Contact c01 = new Contact("10011001", "Albert", "Alvarez", "1001234501", "1st ST");. This one refers to an example I created for the contact test and this one is what the user inputs: "10011001", "Tommy", "Oliver", "1562876410", "Angel St". By checking that everything is correct the test can update the information of the example I provided to the one the user inputted through this line: assertEquals(true, s.updateContact("10011001", "Tommy", "Oliver", "1562876410", "Angel St"));. Another example based on the Task Service Test code is testing out whether the name is valid. The example is in this line: Task task = new Task("1000110002", "Reading all the time everyday", "Reading manga online");. Based on the information provided the name is not valid because it is not within 20 characters. This test will continue to check that the name is valid based on the number of characters the user inputs until it is correct. The last example that I will show is from the Appointment Service Test class that checks whether the ID inputted is valid based on the number of characters entered. Like checking the name from the Task Service Test this one is based on making sure that the ID is within 10 characters for it to be valid. The example is as follows for an incorrect ID: Appointment appointment = new Appointment("10001100021", "6/6/21", "Adjustments for my account"); addAppointment(appointment); System.out.print("size: " + appointments.size());. With the ID not being correct the appointment cannot be verified and accepted for the user.

Overall, I feel that the JUnit tests that I created are good at checking that the information is inputted correctly. The only issue with my tests is that I did not verify that the date is correct for the Appointment class when inputted by the user. I was unable figure out how to confirm whether the date is in the past. Otherwise, I think I did a good job in creating the JUnit tests for each class to verify that the information is correct. In addition, this was also my first-time writing JUnit tests, and I was unfamiliar of them before taking this course. At first, I assumed that they had to be created as from scratch like a Java class. However, after a couple of weeks I understood that they are already created as an option in the Eclipse software. Once I figured that out, I was able to speed up the process in creating the tests for each assignment.

When working on this project the approach that I followed was like the static testing technique. I focused more on completing each code first to make sure that the information was there needed for the JUnit tests. After everything was completed, I moved on to creating the tests for each class. I started off by creating examples for the tests to be verify that the information that was inputted was correct. After that I moved on to updating the information that was entered by the user for the classes to stay updated with the correct information. The last part I did was add notes to certain lines to make sure I understand what each part is meant for in the JUnit tests. Comparing this to the dynamic testing techniques I did not run the test and focused on making sure everything was listed in the code for testing out inputs. Also, I did not test out if the code had any errors by guessing where the issues will occur later. Instead, I focused on the flow of the program and how it would run based on the for loops and booleans I added. I felt that this process was better because I was unfamiliar with JUnit tests, and I wanted to make sure that each line in the code was correct before executing it.

Working on this project I was very cautious about the JUnit tests I created for each assignment. I understood that I needed to test out how the user inputs the information to make sure it is valid I just did not know how the JUnit tests would be used for this. After the first two assignments I was able to get a better idea on how to use them and started to focus on improving the tests to make sure that there are no errors that gets by from what the user inputs. One of these examples is from the Task Service Test class: Task task = new Task("1000110002", "Reading", "Reading manga online almost every day usually from morning to night"); addTask(task); System.out.print("size: " + tasks.size());. This part of the code was used to check that the description entered was correct to validate that the Task is correct. There were also other examples such as these that are similar in the other JUnit tests that I created for the other assignments to check that the inputs were valid. What I did come to understand was that when going over each class I felt that I was missing some information to make sure the tests executed smoothly. That is why it was important to get feedback from others to make sure that I did not leave anything out and everything was accounted for. Having a second pair of eyes is always important when testing something out to make sure there are not issues later. It is also the reason why there is usually a team setup for testing the software in a company to make sure nothing is missed and ready for deployment. Overall, when working on this project I focused a lot on studying how to create JUnit tests. The first couple of weeks I searched through the web for the definition of JUnit to different methods of creating them. It took a while to get the assignment done each week at first, but repetition was the best way for me to understand how to create the tests and complete the assignments. It always starts with first understanding the assignment then slowly build up to it by learning the basics until you get the hang of it. Over time everyone usually gets the hang of things and can create better codes and tests to make sure they run without any issues.