For Project One, I developed unit tests for three primary features: the contact service, task service, and appointment service. My approach centered around validating each feature's core functionalities, as well as their adherence to the specified requirements. For example, in the contact service, I ensured that the contact ID was unique and immutable, first and last names were capped at 10 characters, and phone numbers contained exactly 10 digits. In the task service, I focused on validating that tasks had unique IDs, descriptions were limited to 50 characters, and that task statuses were updated correctly. Finally, for the appointment service, I verified that appointments had valid dates, descriptions were correctly constrained to 50 characters, and that no appointments were set in the past. While constructing testing for each of these milestones, I focused on making tests involved and attached to one aspect of the main classes. By doing this I should have had a large coverage percentage, but as a result of compiler issues a lot of the tests that I produced had issues themselves. The quality of my JUnit tests is reinforced by the methodical approach I took to achieve high coverage and address core functionalities. An example would be in my TaskService class; tests were carefully constructed to validate constraints on task IDs, task names, and descriptions, ensuring they adhered to specified character limits and unique identifiers. Writing JUnit tests has been a struggle. I believe that I have gotten a lot out of trying to write testing code that would run my applications to ensure their quality; Although I clearly need to reevaluate a lot of the work that I have produced during this course. I think one of the biggest benefits that I have received throughout this class is being able to think about what kind of testing each functionality of the classes would need. I thought that I was able to meet the needs of each class in regard to testing but, because of challenges such as compiler errors I missed a lot of mistakes in the tests themselves.

The JUnit tests I wrote were designed to ensure the technical soundness of the AppointmentService class by validating key constraints that were crucial for the application's functionality. Here are some examples found in **AppointmentServiceTest.java**:

The test on line 49 (testAddAppointment()) ensures that adding an appointment with a unique ID is successful, while also checking that adding an appointment with a duplicate ID will throw an IllegalArgumentException. This prevents any potential data integrity issues in the appointment management process.

On line 70 (testUpdateAppointmentDate()), the code ensures that trying to set an appointment in the past raises an IllegalArgumentException. This test is critical for maintaining the integrity of the appointment data, ensuring that users cannot mistakenly schedule events in the past.

I made sure that the code in my tests were efficient by writing JUnit tests that not only validate functionality but also minimize redundancy and unnecessary computations. Specific strategies include targeting key operations, avoiding repetitive tasks, and ensuring that tests focus on critical paths here are some examples.

On line 35 of **AppointmentServiceTest.java**, in the setUp() method, I initialize the AppointmentService object once before running any tests. This avoids repeatedly creating a new instance for each test case, reducing overhead and improving test execution speed.

In line 49, the testAddAppointment() method combines the validation of successful appointment addition and duplicate ID detection in a single test. By handling both scenarios together, the test efficiently validates multiple conditions without redundant setup or teardown, ensuring the code remains streamlined.

In this project, I utilized several software testing techniques, each selected to align with the project's goals of ensuring code reliability, functionality, and correctness. Some of the techinques I used were:

Junit testing: Involves testing smaller units of code checking for specific functionalities and outcomes.

Exception testing: Ensures that the code responds predictably when given both desired conditions and undesired conditions.

Some techniques that I did not use in this project are:

System testing: Involves testing all components as a consolidated unit program which come together to create a system. I did not use this technique because the focus of this course was to break down the code into manageable pieces and focus on testing those individual pieces.

Integration testing: Similar in concept to system testing the focus of this technique is to see how modular code works with each other which I did not use as it was not the focus of this course.

Unit testing holds value as you can disseminate larger amounts of work to multiple team members so that the product gets completed faster. This helps mitigate stress and burnout amongst the team, but if system testing or integration testing are not being used, then when all members present their individual chunks, there could be issues. Exceptions testing should be used regardless of the amount of work as being able to make a flexible predictable piece of code is the goal.

Realizing that the focus of these assignments was to visualize and attack potential flaws in the code through testing cases, I took a mindset of caution and tried to keep my perspective wide. I attempted to look at each requirement through a big picture perspective then when implementing cases for the testing portions I would focus my scope onto core functionalities which I used to implement my testing cases.

To minimize my personal bias and thoughts of things working as they were supposed to, I would develop the code for the core classes first then I would separate myself and reapproach the code to try and find potential flaws and inconsistencies in the testing. By doing this I was allowed an unfamiliar perspective to my own code which I utilized. Bias is almost certainly an issue when it comes to testing your own code as you expect it to work as you intended it to when you wrote it. This can lead to assumed outcomes and frustration when trying to find the shortcomings. It can also lead you to being blindsided towards outlandish issues. You can create your testing and responses to cater to one kind of issue and completely neglect a different category which can cause issues.

Discipline in quality commitment is extremely important as it can and will cause more issues in the future if not adhered to. When testing standards are met with quick fixes and disregard then bigger issues emerge that are not so easily responded to. Starting with discipline at the forefront of the testing approach will mitigate the amount of time and issues there are to respond to and is something that I plan on taking with me beyond this class. I ensured that there was comprehensive unit testing for each class to the best of my ability and integrated solutions to each fault that was brought to my attention. I plan on conducting code reviews and automated testing in the future to minimize and detect issues early in the development process so that I can confront and remove them,