**SUMMARY:**

**Contact Service:**  
For this project, I focused on testing the main features of the Contact Service, adding, updating, and deleting a contact. I used JUnit to make sure each feature worked the way it was supposed to. In the testAddContact() method, for example, I created a contact with the ID "1234567890" and name "Juan Matos", added it to the service, and checked if all the contact details. First name, last name, phone number, and address were stored correctly. I used things like assertEquals() and assertNotNull() to do that. I made sure the tests followed the requirements, like keeping contact IDs unique, phone numbers to exactly 10 digits, and names under 10 characters. I also tested for things that could go wrong like trying to use a name that’s too long or a phone number with letters,just to make sure the code would throw the right errors. My tests worked well, with ContactServiceTest getting 100% code coverage and ContactTest reaching 74%. I made sure to test both normal cases and edge cases, like in testSetFirstName\_invalid(), where I checked if the code would throw an error when a name was too long. Overall, the contact service worked as expected, and my tests helped make sure it was reliable.

**Task Service:**  
In this part, I tested task creation, deletion, and updates using JUnit 5. I split the tests into two classes: TaskTest and TaskServiceTest, to stay organized. I made sure each test matched what the project needed.For example, task IDs had to be 10 characters or fewer, names under 20 characters, and descriptions under 50 characters. I also made sure that each task ID had to be unique. I tested for errors, like adding a task with a duplicate ID or giving it an invalid name or description. My coverage was solid.91.8% for TaskTest and 89.2% for TaskServiceTest. I used assertEquals and assertThrows to check that things worked or failed as expected. I also used helper methods like assertThrowsForInvalidTask() to avoid repeating the same code in different tests. When checking valid data, I used assertAll to check multiple things in one go. These tests helped make sure that the task system worked properly and handled bad input the right way.

**Appointment:**  
I used JUnit 5 to test the main parts of the appointment system, like creating, deleting, and checking input. I had two test classes: AppointmentTest and AppointmentServiceTest. I followed the project rules,like appointment IDs had to be 10 characters or less, dates couldn’t be in the past, and descriptions had to be under 50 characters. For example, in testPastDateThrowsException, I checked that the code wouldn’t let me set a date that already passed. My tests reached 95.9% coverage for AppointmentTest and 86.6% for AppointmentServiceTest. I also tested for weird or bad input, like null values or duplicate IDs, using assertThrows. I even made helper methods to make the tests cleaner and easier to read. Thanks to these tests, I could confidently say the appointment system was solid and handled both good and bad data correctly.

**REFLECTION:**

**Contact Service:**  
In this part of the project, I focused on unit testing using JUnit to make sure the contact features worked properly. I tested all the important methods like adding a contact, updating names or phone numbers, and deleting contacts. I tested edge cases too, like names right at the character limit or phone numbers that were too short to make sure the program caught those issues. I also tried different kinds of inputs to see how the system would react. I didn’t use any system or integration testing because this project was small and didn’t really need it. But I still thought about how different parts of the code worked together. I didn’t just assume my code would work—I made sure to test every possible situation I could think of. In the end, ContactServiceTest had 100% test coverage and ContactTest had 74%, which showed me that good testing really helps keep your code strong. In the future, I plan to keep using testing to make sure I catch bugs early and keep my code clean.

**Task Service:**  
For this section, I stuck with unit testing to check all the task features. I didn’t do any system or integration testing since the project was simple and didn’t need that. Unit testing worked well to make sure each function, like adding, deleting, and updating tasks did what it was supposed to. I also tested invalid inputs and edge cases, like tasks with too-long names or missing IDs. I tried to think like a user who might mess up, just to see if the program could handle it. I didn’t assume anything worked unless I tested it first. This helped me avoid mistakes and build a stronger code. In the future, I want to keep up with good testing habits, maybe even use test-driven development, and aim for high coverage from the start. It’s a good way to save time and avoid big bugs later.

**Appointment:**  
For the appointment features, I focused on unit testing too. I tested creating and deleting appointments, checking for invalid data, and making sure all the rules were followed. Since the code wasn’t super complex and didn’t rely on other systems, unit testing was enough. I made sure to test things like making appointments in the past, using too-long IDs, or entering null values. For example, I wrote a test that made sure an error would pop up if I tried to use a past date. I didn’t just assume the code would behave. I wrote failing tests on purpose to catch anything I missed. This helped me write stronger, more reliable code. Moving forward, I’ll keep writing solid tests, clean up my validation code where I can, and regularly check my work to keep everything running smoothly.

References:

GeeksforGeeks. (2025, March 26). *Unit Testing – Software Testing*. Retrieved from <https://www.geeksforgeeks.org/unit-testing-software-testing/>

Programmingempire. (2023, September 17). *JUnit and Code Coverage*. Retrieved from <https://www.programmingempire.com/junit-and-code-coverage/>

GeeksforGeeks. (2024, January 16). *JUnit 5 – JaCoCo Code Coverage*. Retrieved from <https://www.geeksforgeeks.org/junit-5-jacoco-code-coverage/>