Reflecting on my work with the Contact Service from Module Three and the Task Service from this module, I gained valuable experience in designing and testing code to meet specific requirements. Both services required a deep focus on how functionality was implemented and validated through JUnit tests. My goal was to ensure the programs worked as expected while handling a variety of scenarios.

First, my testing approach was well aligned with the software requirements. For example, in the Contact class, the rules for creating a valid contact ID, first name, last name, phone number, and address were very specific. The ID could not be null, empty, or exceed 10 characters. Similarly, the phone number had to be exactly 10 digits and numeric. In my test case

testConstructorInvalidInputs in ContactTest, I checked for invalid inputs like a null ID or a too-long phone number:

assertThrows(IllegalArgumentException.class, () -> new Contact("12345678901", "John", "Doe", "1234567890", "123 Main Street"));

assertThrows(IllegalArgumentException.class, () -> new Contact("12345", "John", "Doe", "123", "123 Main Street"));

This ensured the constructor behaved as required for all edge cases. Similarly, the Task class validated the task ID, generating one automatically if it was null or blank, which I confirmed with the test testGeneratedTaskIdWhenNull:

Task task = new Task(null, "Test Task", "This is a test description.");

assertNotNull(task.getTaskId());

assertEquals(10, task.getTaskId().length());

These examples illustrate how my tests were aligned with the requirements by validating both valid and invalid inputs.

The quality of my JUnit tests was strong because they thoroughly covered the functionality, ensuring the services behaved correctly in both normal and error conditions. For instance, in ContactServiceTest, I tested adding a contact with a duplicate ID using the test testAddContactDuplicateId:

Contact contact1 = new Contact("12345", "John", "Doe", "1234567890", "123 Main St");

Contact contact2 = new Contact("12345", "Jane", "Smith", "0987654321", "456 Elm St");

service.addContact(contact1);

assertThrows(IllegalArgumentException.class, () -> service.addContact(contact2));

This test ensured that the program correctly handled invalid attempts to add duplicate IDs. In TaskServiceTest, the testUpdateTaskName case ensured tasks could be updated while maintaining data integrity:

service.updateTask("12345", "Updated Name", null);

assertEquals("Updated Name", updatedTask.getName());

assertEquals("Old Description", updatedTask.getDescription());

These examples demonstrate how my tests effectively validated the program’s behavior under various conditions.

To ensure that my code was technically sound, I wrote comprehensive test cases that checked field-level validation and application logic. In the Contact class, the constructor rejected invalid phone numbers using this validation:

if (phone == null || phone.length() != 10 || !phone.matches("\\d+")) {

throw new IllegalArgumentException("Invalid Phone Number!");

}

The corresponding test case in ContactTest, testSettersInvalidInputs, confirmed this behavior:

assertThrows(IllegalArgumentException.class, () -> contact.setPhone("abcdefghij"));

assertThrows(IllegalArgumentException.class, () -> contact.setPhone("123"));

This ensured that the program met the technical requirements for data validation.

Efficiency was also an important focus. The ContactService class used simple and effective logic for CRUD operations. For example, the deleteContact method stopped searching once the contact was found, avoiding unnecessary loops:

if (contacts.get(i).getContactId().equals(contactId)) {

contacts.remove(i);

break; // Exit loop after deleting

}

In the ContactServiceTest case testDeleteContact, I confirmed that deleting a contact reduced the list size and removed the correct item:

service.deleteContact("12345");

assertEquals(0, service.getContacts().size());

These practices ensured that the code operated efficiently while maintaining correctness.

Overall, my experience with the Contact Service and Task Service emphasized the importance of aligning tests with requirements, validating functionality through comprehensive JUnit tests, and ensuring code quality and efficiency. This process prepared me for future projects, such as Project Two, where I will need to analyze and reflect on my work. By focusing on validation, testing, and performance, I ensured the reliability and robustness of my code.