Philip Trinh

SNHU/ CS-320

07/22/2023

**Module Four Journal**

In my testing approach, I tried my best to align the JUnit tests to ensure that all the software requirements are tested, especially the functionalities specified in the requirements document. For example, in the contact service, the requirements put forth are the requirements are the class must support adding contacts with a unique ID, deleting contacts by contact ID, and updating specific fields (first Name, last Name, Phone, Address) for existing contacts by contact ID. Additionally, the requirements for task service is that it must support adding tasks with a unique ID, deleting tasks by task ID, and updating specific fields (Name, Description) for existing tasks by task ID. These two have very similar requirements so their test cases will be almost the same. First for contact service class, I include test cases to verify parameters of adding a new contact, updating contact details, and deleting a contact by creating a new contact service with default values for each field. Then the service is requested to make some change to the list of contacts and the result is tested using assertEquals to ensure the actual field matches what is expected. The code uses the @Order annotation to control the sequence of test execution based on the automatically assigned contact ID for each contact created. To ensure the correctness of the contact ID, For evidence that the task Id is correct for each test, the records are displayed on the console for manual error check. This is the same for task service but instead of contact name, phone, etc. I check create task ID, ensure the name and description can be added, delete, and update correctly.

A screenshot of a computer

Description automatically generated

Contact service test

A screenshot of a computer

Description automatically generated

Task service test

To evaluate the overall quality of my JUnit tests, I run Junit’s coverage tool to evaluate the percentage and ensure I have perfect coverage in the fields that are required. I also look back at the highlight in Contact Service and Task Service classes to understand why certain lines were not covered by the test. For example, the code breaks out before the contact ID not found line is run because the contact Id I set is in the list so it was created and run correctly; otherwise, the test would have failed at assertion and show on the right display as contact ID not found. Task Service are tested in the same way. By having good coverage, I felt more confident that my tests were effective in identifying potential bugs and ensuring the reliability of the services.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Contact service test w/ all ID found.

Contact service test w/ all ID not found.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Task service test w/ all ID not found.

Task service test w/ all ID found.

In terms of ensuring the technical soundness of my code, I followed best practices and coding standards while writing the test cases. For instance, in the contact service tests, I used proper assertions to check if the actual results matched the expected outcomes, display test name, separately create a new task to update, and comments to explain what my code is expected to do. Here's an example:

A screenshot of a computer

Description automatically generated

In this snapshot, I explained what all the tests will do, then I further explain each step and used assertEquals to verify that the new updated field match expected. Also, I include a display task list which will show the list of contact on the console for manual error checking.

To make my code efficient, I paid attention to unnecessary overhead and optimized where possible. For example, in the task service tests, I made sure to minimize unnecessary database queries by using in-memory databases or mocking the database interactions. This helped speed up the tests and reduced the time required for test execution. Here's a snippet illustrating this:

A computer code with text

Description automatically generated

No task ID 1 found, so nothing is deleted, assertEquals error.

A screenshot of a computer

Description automatically generated

Task ID 6 found, so it no longer in the list.

In this example, I ensured that the task was deleted by verifying that it no longer existed in the service after the deletion.

Overall, I tried my best to align my testing approach with the software requirements, maintain good test coverage, ensure technical soundness, and improve efficiency in my JUnit tests for both the contact service and task service classes.