**Summary and reflections report**

**Contact Service**: My approach to unit testing the contact service was to ensure all of CRUD which is create, read, update, and delete. Each method I implemented tested for expected outputs and even potential edge cases. For example, when a new contact was created it required tests for valid and invalid inputs, and when updating contacts, it included checks for non-existent IDs.

**Task Service**: For task service I focused on validating task creations, updates and, and deletion functionalities or you can say CUD. I created tests to verify correct handling of task priorities, deadlines, and completion statuses. Using boundary conditions, such as tasks with the earliest and latest possible dates, I also tested.

**Appointment Service**: My appointment service unit tests concentrated on correct scheduling, rescheduling, and cancelling operations. I included tests that verified appointment conflicts, handling time zones, and managing overlapping appointments. I also implemented validation checks for appointment durations and allowed times.

**Alignment to Software Requirements**

Contact – Requirement to prevent duplicate contacts were tested by trying to add contacts with exact same details. A computer code with text

Description automatically generated with medium confidence

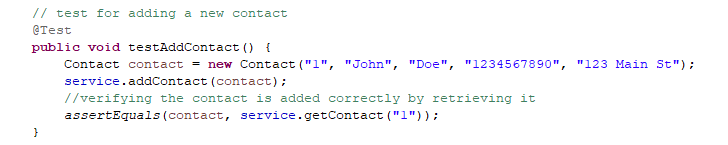
Task - Requirement to ensure tasks with unique IDs are added correctly and duplicates not being allowed: A screenshot of a computer program

Description automatically generated

Appointment – The requirement to cover invalid and valid appointmentId, appointmentDate and description I handle well for instance: A screenshot of a computer

Description automatically generated

**Overall Quality and Experience**

My overall quality of the Junit tests was ensured through testing of edge cases and high code coverage. This reduces the likelihood of undetected bugs. While writing these JUnit tests I tried to cover all functional aspects of the services while maintaining readability and efficiency. For example, I made sure through assertions that I verified the correctness of each test. Like so:

I also strived to avoid unnecessary operations by just writing focused and concise tests. I feel like this is a good example of what I strived for in all my code: A computer code with text

Description automatically generated

**Reflection:**

I applied Unit testing and Exception testing in this project. Some characteristics I applied from unit testing include ensuring each part of code functioned as expected (like methods and classes). Obviously using the framework JUnit to write and run these tests. Testing individual units or even components of code in isolation. For characteristics of exception testing, I tested how the system handled unexpected inputs, ensured it provided error messages, and this often involved testing invalid inputs and testing boundary conditions.

**Unit testing practical uses**:

* Essential for validating the behavior of functions or methods
* Facilitates the refactoring of functionality by ensuring it is preserved
* Useful for catching bugs early in the dev process

**Exception testing practical uses**:

* Important for validating input validation logic
* Ensures stability of the system under unexpected conditions.

**Mindset**

I strived to and I think confidently adopted a cautious mindset. Trying to actively recognize the complexity and relationship in the code. For instance, when implementing the task and task service classes I used specific checks in the constructors and setter methods to enforce these constraints. To limit my bias I tried to focus on writing tests that challenged the code in different scenarios, covering major areas. While developing and self-testing I can get major confirmation bias, where I write tests that confirm unexpected behavior rather than challenge it. Keeping quality at an all time high throughout the whole project was key to me as well as it helps me avoid technical mistakes or even go into technical ‘debt’ later down the line. A fresh set of eyes on my work every day is important, as well as going through regular refactoring helps my code become more readable, scalable, and maintainable. I feel as though these practices will help me in the long run.

References:

*Java unit testing with junit 5: Best Practices & Techniques explained*. BairesDev. (2023, September 8). https://www.bairesdev.com/blog/java-unit-testing/

Bansal, W. by: A. (2024, May 11). *Best practices for unit testing in Java*. Baeldung. https://www.baeldung.com/java-unit-testing-best-practices

Stefan Bechtold, S. B. (n.d.). JUnit 5 user guide. https://junit.org/junit5/docs/current/user-guide/index.html

*A case study*. Refactoring Unit Tests. (n.d.). https://www.kdgregory.com/index.php?page=junit.refactoring