Dominique Flipping

This report provides a summary and reflection of the testing approaches, strategies, and mindset adopted while completing Project One for CS 320. In this project, I developed and tested three core features of a mobile application: the Contact Service, Task Service, and Appointment Service. Unit tests were written in JUnit to ensure that each feature met the requirements and functioned correctly. The following sections summarize my approach, defend the effectiveness of my tests, and reflect on the techniques and mindset that I applied throughout the process.

# Summary

## Unit Testing Approach

For each of the three features, I developed targeted unit tests that aligned closely with the software requirements. For the Contact Service, tests ensured that all contacts could be added, updated, and deleted correctly while maintaining the unique ID constraint. For the Task Service, the focus was on verifying that tasks had valid IDs, names, and descriptions and that updates respected the length restrictions. For the Appointment Service, I tested that appointments included a valid ID, date, and description, and that invalid inputs were handled gracefully. By writing test cases for both valid and invalid scenarios, I ensured coverage of normal operation and edge cases.

## Alignment to Requirements

My testing approach was directly aligned with the requirements outlined in the specifications. For example, the requirement that a contact ID must not exceed ten characters was tested with both valid and invalid inputs. Similarly, the requirement that task names must be under 20 characters was enforced in the unit tests. These tests demonstrated that the code adhered to the functional constraints defined in the project guidelines.

## Effectiveness of Tests

The effectiveness of the JUnit tests was demonstrated through their high coverage percentage. Coverage tools showed that most branches and methods were exercised by the test cases, including handling of exceptions and invalid input. This high level of coverage provided confidence that the application behaved as expected under a wide range of conditions.

## Experience Writing JUnit Tests

Writing the JUnit tests was an iterative process that helped me strengthen both my technical and critical thinking skills. To ensure the code was technically sound, I wrote assertions that verified each outcome precisely. For example, in the TaskServiceTest, I included assertions that verified task names could not exceed the character limit. I also included tests for null inputs to confirm that the application handled them appropriately. To ensure efficiency, I avoided redundant tests and grouped related assertions within the same test method, reducing duplication while maintaining clarity.

# Reflection

## Testing Techniques Employed

The main testing techniques I employed in this project included unit testing and boundary value testing. Unit testing allowed me to focus on individual components in isolation, such as validating the behavior of the Contact or Task Service. Boundary value testing was applied to enforce constraints, such as the maximum length of IDs, names, and descriptions. These techniques ensured that the application was both robust and aligned with requirements.

## Other Techniques Not Used

Other software testing techniques that I did not employ included integration testing, system testing, and regression testing. Integration testing would involve validating how different modules interact with each other, which was beyond the scope of this project. System testing would test the full application in an environment similar to production. Regression testing would ensure that new changes did not break existing functionality. Although these techniques were not applied here, they are essential in larger projects where multiple components must work together reliably.

## Uses and Implications of Techniques

Unit and boundary value testing are most practical during the early stages of development to catch small errors before they escalate. Integration and system testing are more appropriate for later stages of development to validate end-to-end behavior. Regression testing becomes essential when projects grow and changes are frequent. By understanding which technique to use in different situations, developers can build more reliable and maintainable systems.

## Mindset: Caution

While working on this project, I adopted a mindset of caution by carefully validating each requirement against the code. For example, I considered the potential consequences of invalid input for contact IDs and task descriptions, writing tests to verify that the system handled these gracefully. It was important to appreciate the complexity of interrelated code since an error in one service could potentially affect others.

## Mindset: Limiting Bias

To limit bias, I approached the code as though I had not written it myself. I asked, 'If I were a different developer, what assumptions might I challenge?' This helped me write stronger tests that checked for unexpected inputs rather than only confirming what I expected to work. For instance, I tested null values, overly long strings, and invalid dates to ensure that my assumptions did not lead to overlooked bugs. As a developer, bias could be a concern if I only tested for cases I knew were correct, so I actively worked to test for incorrect or edge cases as well.

## Mindset: Discipline and Quality

Discipline was essential in maintaining a commitment to quality. Cutting corners by omitting edge cases or skipping validations could have introduced technical debt that would create more work later. For example, ignoring validation on task descriptions might have led to runtime errors in production. By thoroughly testing now, I avoided introducing issues that would require costly fixes later. As a practitioner, I plan to avoid technical debt by consistently applying testing best practices, maintaining clear documentation, and ensuring that code reviews include test coverage verification.

# Conclusion

This project reinforced the importance of aligning testing strategies with requirements and remaining disciplined in applying testing best practices. By combining unit and boundary value testing, writing efficient and sound test code, and adopting a careful and unbiased mindset, I was able to deliver a robust application with strong coverage. The lessons from this project will guide me in future software engineering work, ensuring that I continue to balance technical soundness with efficiency and quality.