LaTisha Burns

October 15th, 2025

CS 320

Project Two

**Summary**

**Unit Testing Approach**

For the contact, task, and appointment services, my unit testing approach focused on matching each test directly to the software requirements. I used JUnit to create automated tests that verified all validation rules, data restrictions, and service methods. Each test was built to ensure that the program correctly handled both valid and invalid input.

In the ContactService, one requirement was that the contact ID must not be null or longer than ten characters. I created a test using the assertThrows method to confirm that an exception was raised if an invalid ID was entered. This helped verify that the constructor followed the rules correctly. In the TaskService, I tested that each task ID was unique. I added one valid task, then attempted to add another task with the same ID to confirm that an exception occurred. This confirmed that duplicates were not allowed and that the service managed data properly.

For the AppointmentService, I wrote tests to confirm that appointments could be added, updated, and deleted successfully. I also tested for invalid IDs and null values to make sure errors were correctly handled. By testing both positive and negative cases, I could see how the code behaved in every scenario. This approach aligned fully with the requirements and gave me confidence that the system worked as intended.

**Test Quality Defense**

I can defend the overall quality of my JUnit tests because they covered every rule listed in the project and tested all main paths in the code. My goal was to make sure every function was tested with correct and incorrect input so that both expected results and failure conditions were checked. This provided full coverage across the classes and reduced the chance of hidden bugs.

Each test was automated, which made it easy to run them again whenever I changed the code. This helped confirm that earlier features still worked correctly and allowed me to identify problems early. Running all tests together and reviewing the consistent pass results showed that the test suite was effective and reliable.

**Experience Writing JUnit tests**

While writing JUnit tests, I ensured that my code was technically sound by using clear assertions and exception checks. For example, I used assertThrows(IllegalArgumentException.class, () -> new Contact("12345678901", "John", "Doe", "1234567890", "Address")); to confirm that long IDs were rejected. This demonstrated that my constructors properly enforced requirements.

To keep my tests efficient, I combined related checks when it made sense. For example, in the TaskService, I verified that the name and description of a task could both be updated in the same test method. This allowed me to cover multiple requirements without repeating code. My focus on reducing redundancy made my tests easier to manage while still confirming all behaviors.

Creating JUnit tests also helped me improve my understanding of how different parts of the program worked together. Each test confirmed not just that the code was correct, but that the logic was consistent and the program structure was solid.

**Reflection**

**Testing Techniques**

**Testing techniques I used:**

* Static testing involved reviewing the code and requirements without execution to find early errors. This helped ensure my classes matched the specifications before running the tests.
* Unit testing focused on verifying small, isolated parts of the program such as constructors, setters, and service methods. This allowed me to confirm each individual component worked properly.
* JUnit testing provided a framework for automating these unit tests, which made it faster to check my results and rerun tests after making updates.

**Testing techniques I did not use:**

* Integration testing would have been useful if I needed to test how multiple services interacted, but this project was limited to individual class testing.
* System testing is used when an entire program is ready for real-world testing, such as checking a user interface or database connection, which this project did not include.
* Regression testing retests a program after changes to ensure old functionality still works. Since the milestones were independent, this technique was not needed.

In practical situations, static and unit testing are helpful early in development to catch small issues before they grow. Integration and system testing become important when different parts of a system must work together. Regression testing is especially valuable in long-term projects where code changes frequently, and white-box testing is useful for performance and security validation.

White-box testing examines internal logic and code paths. I focused more on black-box testing, where I tested inputs and outputs without reviewing the internal code structure.

**Mindset**

Throughout this project, I approached testing with caution and attention to detail. I understood that each class and method was connected, so one small error could affect how data was handled elsewhere. For example, if the validation rules in the Contact class failed, incorrect data could carry over to other services. Recognizing these connections helped me appreciate the complexity of the system and remain careful in my testing.

To limit bias while reviewing code, I treated each test as if I were verifying someone else’s work. I intentionally tried to break my own code by using invalid data. This helped me stay objective and spot issues I might have overlooked if I only tested successful cases. I can see how bias might be a concern for developers testing their own code because it is easy to assume the code is correct. Testing from multiple perspectives helps reduce that risk.

Finally, being disciplined about quality is essential for software engineers. Cutting corners can lead to bugs, security risks, and future technical debt. To avoid this, I plan to always include thorough testing and regular reviews in my workflow. For example, in future projects, I will use automated test coverage tools and maintain updated test cases after each code change to ensure long-term reliability and maintainability.