7-2 Project Two

# Justin Brown

# CS-320

**Describe your unit testing approach for each of the three features.**

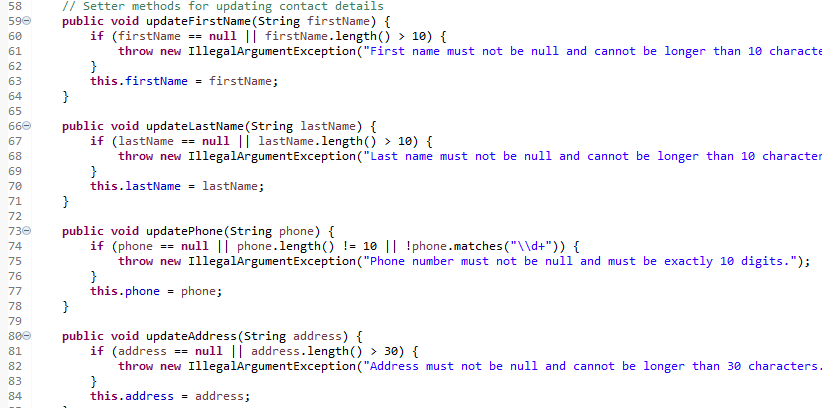
**Contact Service:** Validation tests ensured that contact operations met constraints specified in the Contact class, such as name formats and phone number validations. The ContactService tests covered scenarios for adding, deleting, and updating contacts following the listed requirements.

**Task Service:** Testing for tasks operations ensured guidelines set in the Task class were met, including validations for task name lengths and descriptions.

**Appointment Service:** Tests confirmed that appointments were added with valid IDs, descriptions and past dates are not available for booking.

**To what extent was your approach aligned to the software requirements? Support your claims with specific evidence.**

The approach for all services were thorough to make sure all requirements were met for each one Specifically, in task creation, each requirement was checked before a task could be created, such as task ID length not being greater than 10 characters while also not being empty, name length not being greater than 20 while also not being empty as well as description lengths not being greater than 50 characters.

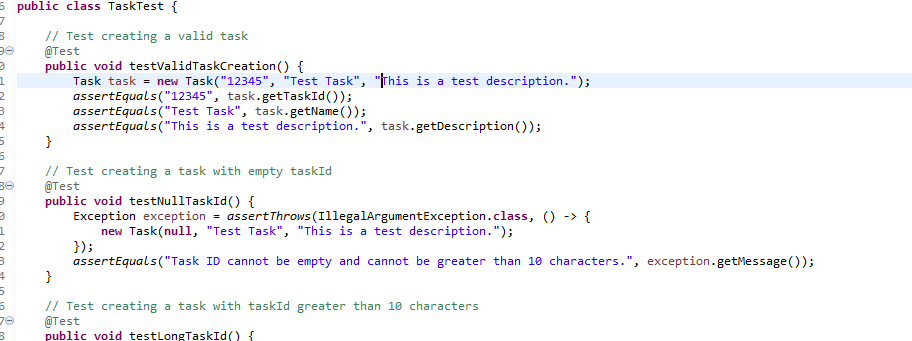


**Appointment Service:**

**Defend the overall quality of your JUnit tests. In other words, how do you know your JUnit tests were effective based on the coverage percentage?**

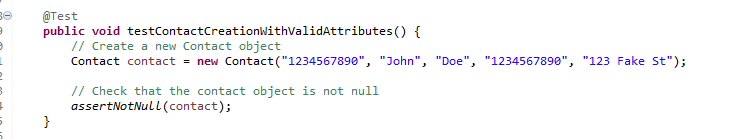
The quality of the JUnit tests was high, as evidenced by a comprehensive coverage percentage. I scored around 89% coverage percentage. While not perfect, the tests were effective in catching errors and edge cases, as each test focused on specific inputs and outputs, ensuring robust validation.

**How did you ensure that your code was technically sound? Cite specific lines of code from your tests to illustrate.**

While optimizations can certainly be made, I feel that the following test showcases my thought process in making sure that the tests were thorough: Tests were ran for each required argument to create a task, making sure that all arguments are not only filled, but also that they meet the requirements provided. (Not going over character limits, unique IDs, etc.)

**How did you ensure that your code was efficient? Cite specific lines of code from your tests to illustrate.**

Looking at contacttest.java, I want to highlight the method of contact creation with valid attributes. The test focuses on immediately checking that the contact object is not empty and meets all the requirements outlined by the client, without using much space when run. This test simply checked whether or not all requirements were fulfilled then acted accordingly and proceeded with the other tests.



#### **Reflection**

**What were the software testing techniques that you employed in this project? Describe their characteristics using specific details.**

**Unit Testing:** Unit testing focuses on testing individual components or units of code. Each test isolates a single method or attribute to ensure correct behavior. The purpose was to test specific required functionalities.

**Boundary Testing:** Tests the limits of input ranges to ensure the system handles edge cases correctly. Used to test empty strings, invalid formats, and out-of-range values for attributes like phone numbers.

**What are the other software testing techniques that you did not use for this project? Describe their characteristics using specific details.**

**Integration Testing**: Tests interactions between different modules to ensure they work together correctly. Useful for verifying combined functionality but not used here as the focus was on individual units.

**System Testing** Involves end-to-end testing of the entire application to ensure it meets requirements. This was not applied in this project due to the focus on unit testing.

**For each of the techniques you discussed, explain the practical uses and implications for different software development projects and situations.**

**Unit Testing:** Mostly used in early-stage development to catch issues at the component level. This ensures individual functions work properly, reducing the likelihood of bugs.

**Integration Testing:** This form of testing verifies proper interactions between modules in larger projects. This is useful for catching integration issues early and allowing for adjustment if needed.

**System Testing:** System testing validates the entire system before deployment. This makes sure that all parts of the system work as intended and meet the requirements of the client.

**Assess the mindset that you adopted working on this project. In acting as a software tester, to what extent did you employ caution? Why was it important to appreciate the complexity and interrelationships of the code you were testing? Provide specific examples to illustrate your claims.**

As a software tester, exercising caution is important as it ensures that software behaves as expected in as many scenarios as possible. This cautious approach helps to identify potential issues early in the development cycle, preventing bugs from reaching production and accounting for potential oversights. Appreciating the complexity and interrelationships of the code being tested is also just as important. Understanding these aspects allows for comprehensive test cases that cover a wide range of scenarios and helps identify areas of the code that are prone to issues due to their complexity or their interactions with other parts of the system.

While not applicable to this specific assignment, an example I can think of is programming within an atm. Imagine if a user were to walk away from an atm without backing out of a transaction. An idle timer may be used to protect the customer’s bank account from the next person who may use it. It’s important to account for as many use cases as possible while also keeping the requirements of the client in mind.

**Assess the ways you tried to limit bias in your review of the code. On the software developer side, can you imagine that bias would be a concern if you were responsible for testing your own code? Provide specific examples to illustrate your claims.**

When reviewing my code, the first thing I question I always asked myself was “does this code fulfill all the requirements of the client?” While my final product worked as intended and did fulfill these requirements, I made sure to look back and check for any potential optimizations and improvements I could have made. While it is nice to feel accomplished, as someone who is not very familiar with Java I struggled a lot, and as such there are undoubtedly errors and inefficient ways I wrote my code. I made improving my number one priority, so I made sure honest review and critique was always the goal. If I were in a professional setting and testing my own code, it might be easier to overlook small mistakes that most people may not notice or care enough to correct me on. But it would prevent me from growing and learning from my mistakes, so it is incredibly important to remove bias when reviewing one’s own code.

**Finally, evaluate the importance of being disciplined in your commitment to quality as a software engineering professional. Why is it important not to cut corners when it comes to writing or testing code? How do you plan to avoid technical debt as a practitioner in the field? Provide specific examples to illustrate your claims.**

While cutting corners may seem tempting, especially for more tedious tasks that may seem inconsequential, over time these shortcuts will build up and lead to shoddy, poorly put together software. Even improper testing habits may result in disastrous consequences, missing features, unintended exploits, etc. An immediate example that comes to mind is in all of the service classes I wrote, they all work with character limits while also expecting some form of input. What if the user enters a bunch of blank spaces and an object gets created when it shouldn’t? That would be an issue that would have to be fixed later down the line when it didn’t have to be in the first place. It’s also important to consider scalability as well. If software isn’t built with scalability kept in mind, then the potential for backtracking and wasted time/resources can be quite large. I made sure to validate all inputs and handle exceptions appropriately, thereby preventing the introduction of technical debt in this project. As for the future, I will make sure to not cut corners, even if what I am tasked with is tedious/inconsequential.