Sergio Irianda

CS-320

Project 2

1. Summary

For each of the three main features in the Appointment class uniqueID, date, and description a unit testing was conducted using JUnit. My approach ensured that both valid and invalid inputs were tested to confirm my validation methods. ID validation included scenarios where the ID was null, too long, or valid. Date validation tests checked if the date was null, in the past, or valid. The description validation tests ensured that descriptions were not too long and null.

The unit tests aligned with the software requirements by enforcing constraints outlined in the class. For example:

The test testBadDate() ensures past dates are rejected:

A black background with blue text

AI-generated content may be incorrect.

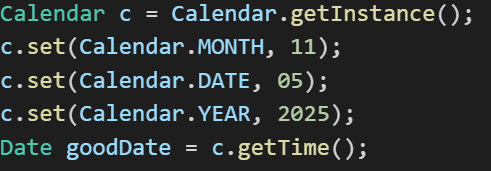
The effectiveness of the JUnit tests was measured using test coverage analysis. The tests covered all major input conditions, ensuring all cases were handled correctly. Achieving a high test coverage percentage confirmed that all branches of the code logic were exercised. Writing the JUnit tests involved structure validation of input handling, error throwing, and edge cases. The process reinforced the importance of validating logic and provided insights into programming practices.

The use of assertions in JUnit ensured correctness ensured technical soundness, such as:



This validates that exceptions were triggered correctly under invalid conditions.

The tests minimized redundant logic by leveraging reusable date objects and ensuring efficiency:



By reusing a properly formatted date, unnecessary object creation was avoided.

1. Reflection

Testing Techniques:

The project focused on validating input and output without considering internal implementation details. I verified edge cases, such as input lengths (ID max length, description max length) and date constraints. The testing that was not used was White-Box Testing. This method examines internal structures but was not used as JUnit tests mainly evaluated expected outputs. Regression Testing is another valuable method. However, since this was an isolated project, tests were not rerun against modified versions of the software.

Mindset:

As a tester, caution is important to avoid overlooking edge cases. The Understanding between input validation methods ensured comprehensive testing. For example, acknowledging that null descriptions and empty descriptions were distinct cases led to separate test cases (testNullDesc() vs. testEmptyDesc()).

Bias was minimized by systematically considering invalid inputs that one might assume would never occur. For instance, assuming that a date will always be provided could lead to missing null handling. Testing with:

A black background with blue text

AI-generated content may be incorrect.

This ensures that assumptions are challenged.

Maintaining quality in software testing is important to prevent hidden bugs, extra work later, and problems with maintenance. To do this, I made sure to test different situations, including edge cases, to catch potential issues. I could have also had my peers review my code to get a fresh perspective and find mistakes I might have missed. Plus, I kept learning about the best ways to test software to improve my approach. By staying disciplined and following these steps, future software projects can be more reliable and easier to maintain.