Software Testing Lab Report
JUnit 5 Unit Testing

Mohamed Amine El Kalai

November 26, 2025

1 Introduction

This report describes the practical work on software testing using JUnit 5. The exercises cover basic unit testing and boundary value analysis with temperature regulation systems.

2 Exercise 1: Calculator Testing

2.1 Project Setup

I created a new Maven project in IntelliJ IDEA with the standard directory structure. The project contains separate folders for source code (`src/main/java`) and test code (`src/test/java`).

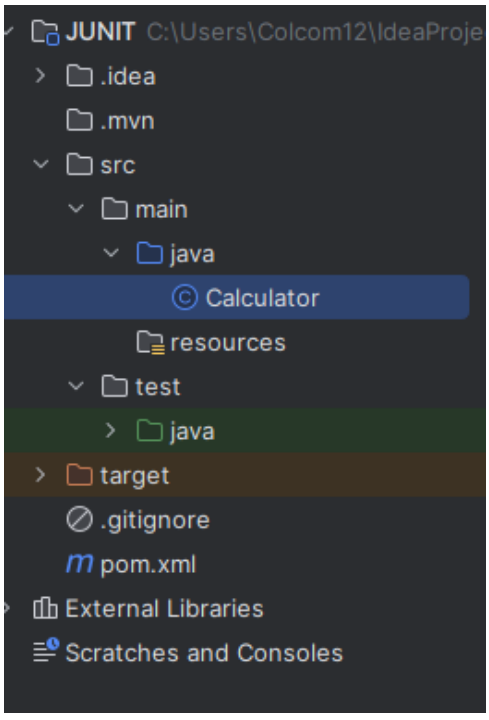

Figure 1: Project structure in IntelliJ IDEA

2.2 Calculator Class Implementation

The Calculator class provides basic arithmetic operations including add, subtract, multiply, and divide methods.

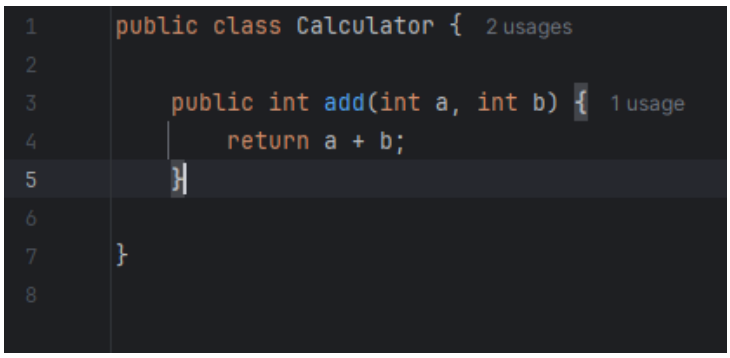

Figure 2: Calculator class with add method

2.3 Test Implementation

I wrote unit tests for all methods using the Arrange-Act-Assert pattern. The test class was generated automatically by IntelliJ IDEA, then I added the test logic.

Figure 3: Test class for the add method

2.4 Test Results

2.4.1 Successful Test Run

When running the test with the correct implementation, all tests passed successfully.

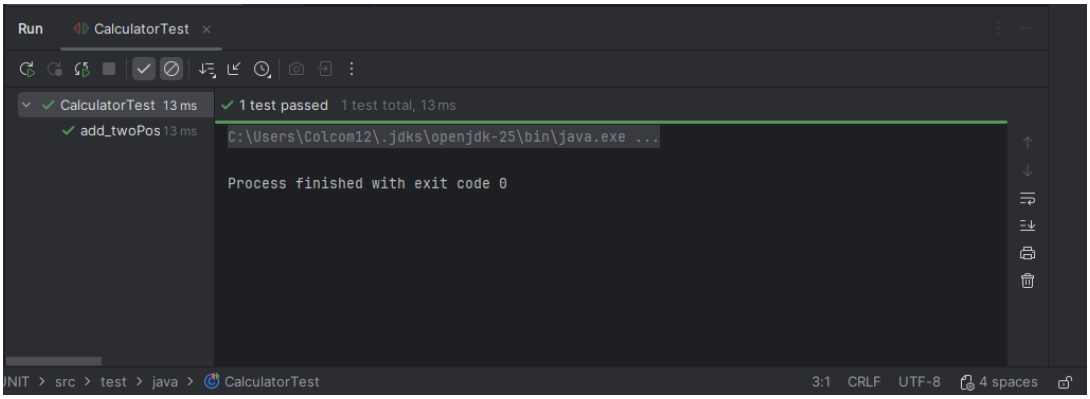

Figure 4: Test passed - all assertions correct

2.4.2 Failed Test (Intentional Bug)

I modified the `add` method to return `a - b` instead of `a + b` to demonstrate how unit tests detect bugs. The test failed as expected.

Figure 5: Test failed - bug detected (expected 5, got -1)

The error message clearly shows that the expected value was 5, but the actual result was -1, proving that the test successfully caught the bug.

2.4.3 All Tests Passing

After implementing all four arithmetic methods and their exception handling, all five tests passed.

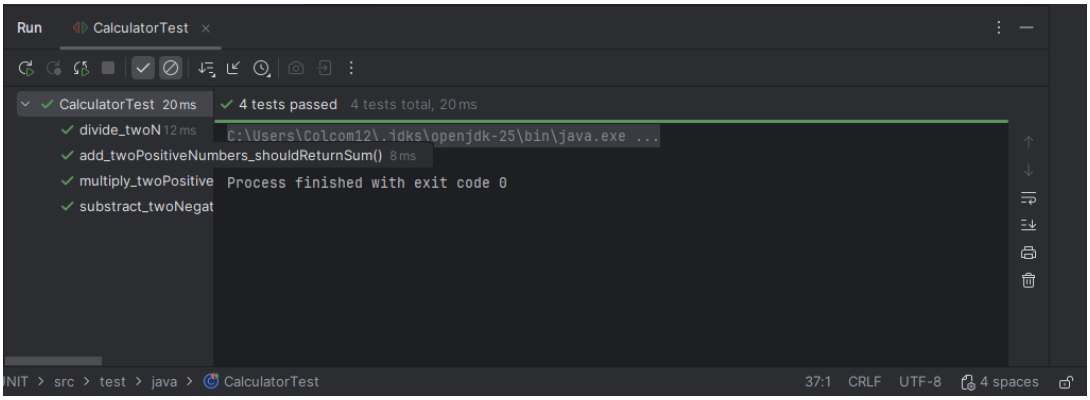

Figure 6: All calculator tests passed (4 tests total shown)

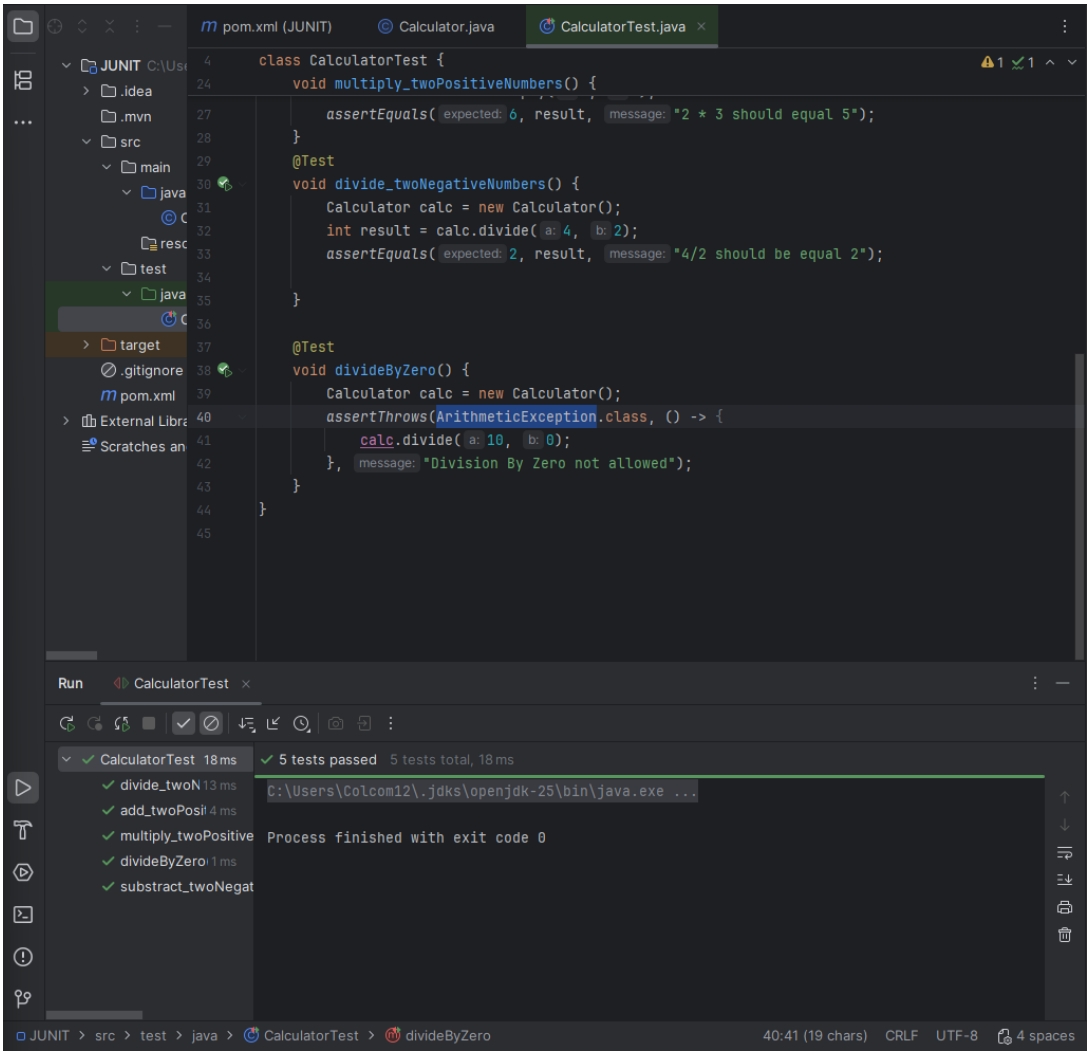

Figure 7: Complete test suite with divide by zero test (5 tests total)

2.5 Key Learnings from Exercise 1

- The Arrange-Act-Assert pattern makes tests clear and organized
- `assertEquals` verifies expected vs actual values

- `assertThrows` checks that exceptions are thrown correctly
- Unit tests can catch bugs immediately when code is modified

3 Exercise 2: Temperature Regulator

3.1 Problem Description

This exercise tests a temperature control system that decides between HEAT, COOL, or STANDBY based on current and target temperatures. The specification defines a tolerance zone of ±0.5°C around the target temperature:

- If current < target - 0.5, then action is HEAT
- If current > target + 0.5, then action is COOL
- Otherwise, action is STANDBY

3.2 Boundary Value Analysis

Boundary Value Analysis (BVA) is a testing technique that focuses on values at the edges of input ranges. I identified the following boundary values for testing:

- Just inside the STANDBY zone: current = target + 0.25 and target - 0.25
- At the exact boundaries: current = target + 0.5 and target - 0.5
- Just outside the STANDBY zone: current = target + 0.75 and target - 0.75

3.3 Test Implementation

I created test cases to check all boundary conditions around the tolerance zone.

```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class TemperatureRegulatorTest {

    @Test
    void currentTempBVA() {
        TemperatureRegulator regulator = new TemperatureRegulator();

        assertEquals(TemperatureRegulator.Action.STANDBY, regulator.compute( current: 0,  target: 0.25),  message: "0.25 + 0.5 > 0 and 0 > 0.25-0.5");
        assertEquals(TemperatureRegulator.Action.STANDBY, regulator.compute( current: 0,  target: 0.5),  message: "0 == 0.5 - 0.5");
        assertEquals(TemperatureRegulator.Action.HEAT, regulator.compute( current: 0,  target: 0.75),  message: "0 < 0.75 - 0.5");
        assertEquals(TemperatureRegulator.Action.STANDBY, regulator.compute( current: 0,  target: -0.5),  message: "0 == 0.5 - 0.5");
        assertEquals(TemperatureRegulator.Action.COOL, regulator.compute( current: 0,  target: -0.75),  message: "0 > -0.75 + 0.5");



    }


}
```

Figure 8: Temperature regulator boundary value test cases

3.4 Test Results

All boundary tests passed successfully, confirming that the implementation correctly handles the tolerance zone according to the specification.

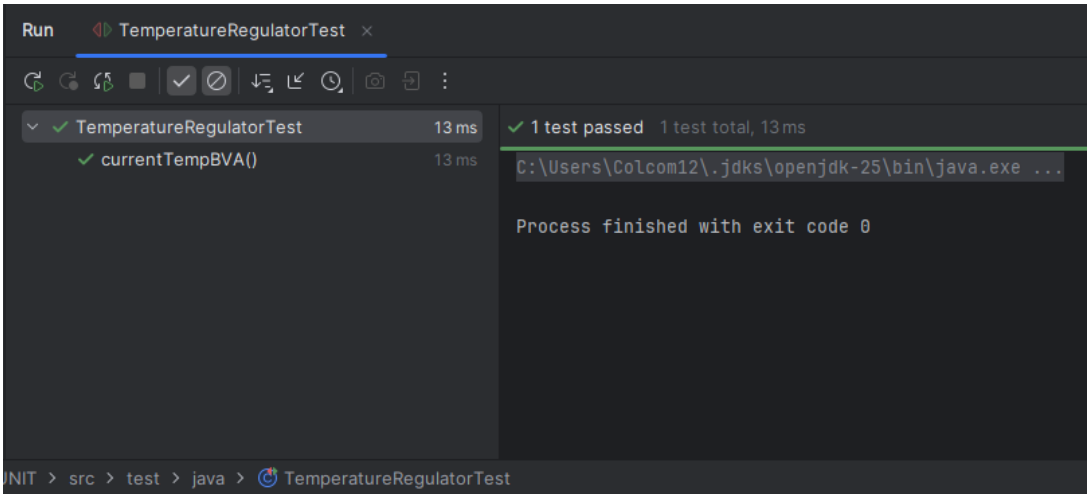

Figure 9: Temperature regulator test passed - all boundary conditions verified

3.5 Key Learnings from Exercise 2

- Boundary Value Analysis helps find edge cases where bugs often occur
- Testing at exact boundaries (± 0.5) and near boundaries (± 0.25, ± 0.75) gives confidence
- The tolerance zone specification was correctly implemented in the code
- Descriptive assertion messages help understand what each test checks

4 Conclusion

This lab helped me understand the basics of unit testing with JUnit 5. I learned how to:

- Set up a Java project with proper test structure in IntelliJ IDEA
- Write and run unit tests using the Arrange-Act-Assert pattern
- Use common assertions like `assertEquals` and `assertThrows`
- Apply Boundary Value Analysis to identify important test cases
- Interpret test results and understand failure messages

The exercises showed that proper testing can catch bugs early and ensure that code works correctly under different conditions, especially at boundary values where errors are most likely to happen.