Software Testing Practical Session Report

Younes Menfalouti

November 26, 2025

1 Introduction

This report documents the practical session on software testing, focusing on unit testing with JUnit in IntelliJ IDEA. The session involved implementing a Calculator class, a TemperatureRegulator class, and writing comprehensive unit tests using the Arrange-Act-Assert pattern.

2 Exercise 1: Calculator

2.1 Implementation

The Calculator class provides basic arithmetic operations.

```
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }
    public int sub(int a, int b) {
        return a - b;
    }
    public int mul(int a, int b) {
        return a * b;
    }
    public int div(int a, int b) {
        return a / b;
    }
}
```

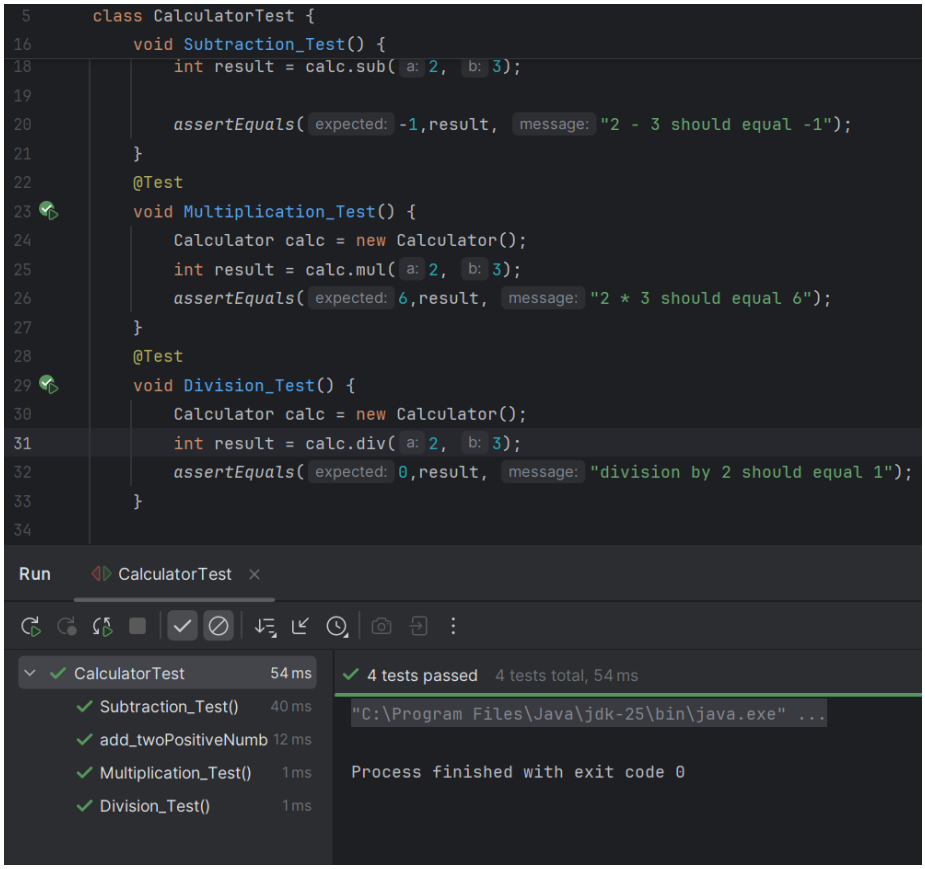
2.2 Unit Tests

The CalculatorTest class contains unit tests for the Calculator.

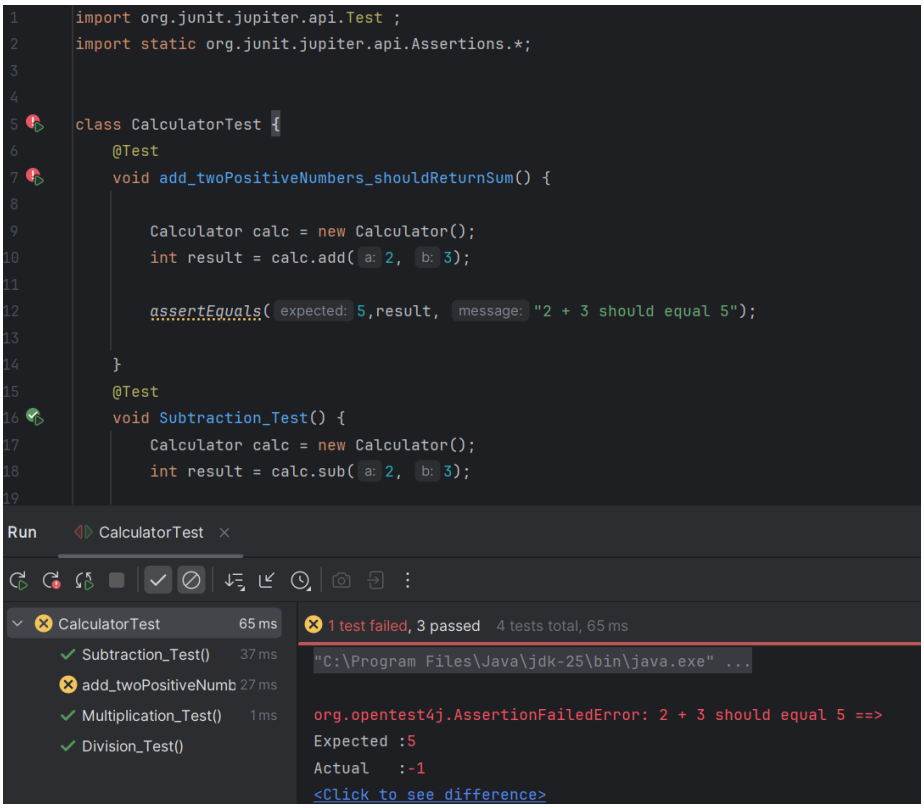
```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
```

```
class CalculatorTest {
    @Test
    void add_twoPositiveNumbers_shouldReturnSum() {
        Calculator calc = new Calculator();
        int result = calc.add(2, 3);
        assertEquals(5, result, "2 + 3 should equal 5");
    }
    @Test
    void Subtraction_Test() {
        Calculator calc = new Calculator();
        int result = calc.sub(2, 3);
        assertEquals(-1, result, "2 - 3 should equal -1");
    }
    @Test
    void Multiplication_Test() {
        Calculator calc = new Calculator();
        int result = calc.mul(2, 3);
        assertEquals(6, result, "2 * 3 should equal 6");
    }
    @Test
    void Division_Test() {
        Calculator calc = new Calculator();
        int result = calc.div(2, 3);
        assertEquals(0, result, "division by 2 should equal 1");
    }
}
```

2.3 Test Execution



Here, the add method was temporarily modified to return `a - b` instead of `a + b`.



Running the add test again results in failure, as the expected sum (5) does not match the incorrect subtraction result (-1).

3 Exercise 2: TemperatureRegulator

3.1 Implementation

The TemperatureRegulator class computes the action based on current and target temperatures.

```
public class TemperatureRegulator {

    public enum Action { HEAT, COOL, STANDBY }

    public Action compute(double current, double target) {
        final double TOL = 0.5;

        double diff = current - target;

        if (diff < -TOL) {
```

```
            return Action.HEAT;
        } else if (diff > TOL) {
            return Action.COOL;
        } else {
            return Action.STANDBY;
        }
    }
}
```

3.2 Unit Tests

The TemperatureRegulatorTest class contains unit tests for the TemperatureRegulator.

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

class TemperatureRegulatorTest {

    @Test
    void compute_currentMuchLowerThanTarget_shouldReturnHeat() {
        TemperatureRegulator regulator = new TemperatureRegulator();
        double current = 20.0;
        double target = 22.0;
        TemperatureRegulator.Action result = regulator.compute(current, target);
        assertEquals(TemperatureRegulator.Action.HEAT, result);
    }

    @Test
    void compute_currentMuchHigherThanTarget_shouldReturnCool() {
        TemperatureRegulator regulator = new TemperatureRegulator();
        double current = 22.0;
        double target = 20.0;
        TemperatureRegulator.Action result = regulator.compute(current, target);
        assertEquals(TemperatureRegulator.Action.COOL, result);
    }

    @Test
    void compute_currentWithinTolerance_shouldReturnStandby() {
        TemperatureRegulator regulator = new TemperatureRegulator();
        double current = 21.0;
        double target = 21.0;
        TemperatureRegulator.Action result = regulator.compute(current, target);
        assertEquals(TemperatureRegulator.Action.STANDBY, result);
    }
```

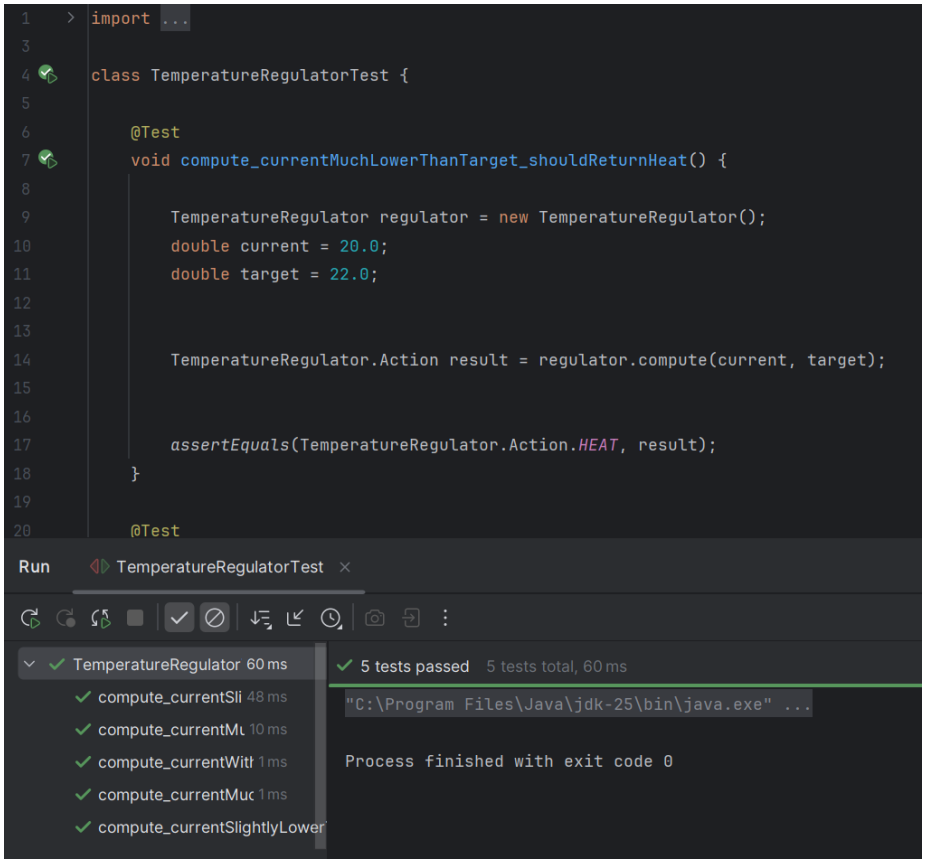
```
    @Test
    void compute_currentSlightlyHigherThanTarget_shouldReturnStandby() {
        TemperatureRegulator regulator = new TemperatureRegulator();
        double current = 20.7;
        double target = 21.0;
        TemperatureRegulator.Action result = regulator.compute(current, target);
        assertEquals(TemperatureRegulator.Action.STANDBY, result);
    }

    @Test
    void compute_currentSlightlyHigherThanTarget_shouldReturnStandby()
        TemperatureRegulator regulator = new TemperatureRegulator();
        double current = 20.7;
        double target = 21.0;
        TemperatureRegulator.Action result = regulator.compute(current, target);
        assertEquals(TemperatureRegulator.Action.STANDBY, result);
    }
}
```

3.3 Analysis

The test results indicate that the TemperatureRegulator class behaves according to the specification. The compute method correctly determines the action based on the temperature difference and tolerance: - Returns HEAT when current temperature is more than 0.5 degrees below the target. - Returns COOL when current temperature is more than 0.5 degrees above the target. - Returns STANDBY when the difference is within the tolerance (± 0.5 degrees).

3.4 Test Execution



4 Conclusion

The session demonstrated the basics of unit testing in Java using JUnit.