# Software Testing Lab Report
# Unit Testing

Nour el houda

November 27, 2025

## 1 Exercise 1: Calculator Testing

### 1.1 Implementation

The Calculator class provides basic arithmetic operations:

```java
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }

    public int subtract(int a, int b) {
        return a - b;
    }

    public int multiply(int a, int b) {
        return a * b;
    }

    public int divide(int a, int b) {
        if (b == 0) {
            throw new ArithmeticException("Cannot divide by zero");
        }
        return a / b;
    }
}
```

Listing 1: Calculator.java

### 1.2 Unit Tests

:

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

class CalculatorTest {

```

```java
    @Test
    void add_twoPositiveNumbers_shouldReturnSum() {
        // Arrange
        Calculator calc = new Calculator();
        // Act
        int result = calc.add(2, 3);
        // Assert
        assertEquals(5, result, "2 + 3 should equal 5");
    }

    @Test
    void divide_byZero_shouldThrowArithmeticException() {
        // Arrange
        Calculator calc = new Calculator();

        // Act & Assert
        ArithmeticException exception = assertThrows(
            ArithmeticException.class,
            () -> calc.divide(10, 0),
            "Division by zero should throw ArithmeticException"
        );

        assertEquals("Cannot divide by zero", exception.
   getMessage());
    }
}
```

Listing 2: CalculatorTest.java - Sample Tests

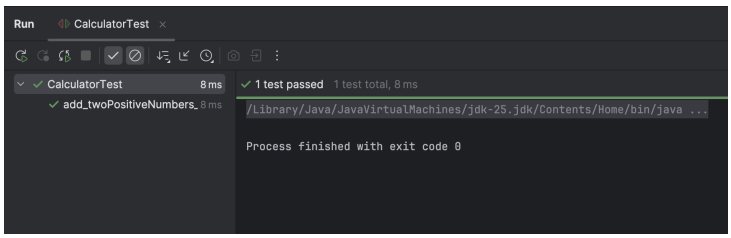


Figure 1: This is my image


Figure 2: This is my image

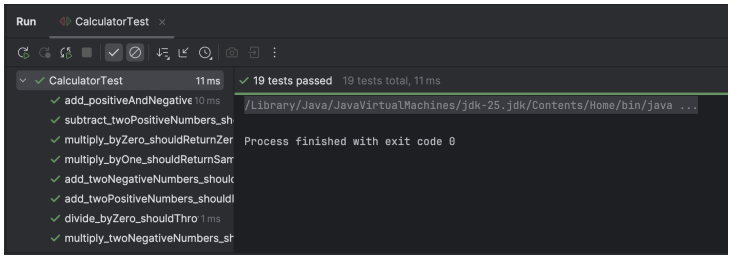


Figure 3: This is my image

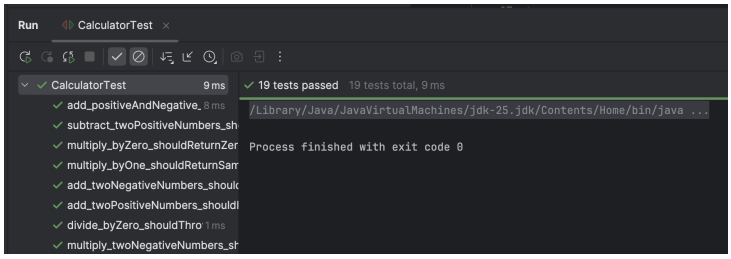


Figure 4: This is my image

# 2 Exercise 2: Temperature Regulator (BVA)

## 2.1 Specification

The Temperature Regulator controls heating/cooling based on tolerance:

- If `current < target - 0.5` → HEAT
- If `current > target + 0.5` → COOL
- Otherwise → STANDBY

## 2.2 Implementation

```java
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;
        }
    }
```

```
}
```

Listing 3: TemperatureRegulator.java

## 2.3 Boundary Value Analysis

For target = 20.0°C, the boundaries are:

- Lower boundary: 19.5°C (target - 0.5)
- Upper boundary: 20.5°C (target + 0.5)

## 2.4 BVA Test Implementation

```
class TemperatureRegulatorTest {

    @Test
    void compute_currentExactlyAtLowerBoundary_shouldReturnSTANDBY() {
        TemperatureRegulator regulator = new TemperatureRegulator();
        double current = 19.5;
        double target = 20.0;

        TemperatureRegulator.Action result = regulator.compute(current, target);

        assertEquals(TemperatureRegulator.Action.STANDBY, result,
            "Current 19.5 is at lower boundary, should STANDBY");
    }

    @Test
    void compute_currentExactlyAtUpperBoundary_shouldReturnSTANDBY() {
        TemperatureRegulator regulator = new TemperatureRegulator();
        double current = 20.5;
        double target = 20.0;

        TemperatureRegulator.Action result = regulator.compute(current, target);

        assertEquals(TemperatureRegulator.Action.STANDBY, result,
            "Current 20.5 is at upper boundary, should STANDBY");
    }
}
```

Listing 4: TemperatureRegulatorTest.java - BVA Tests

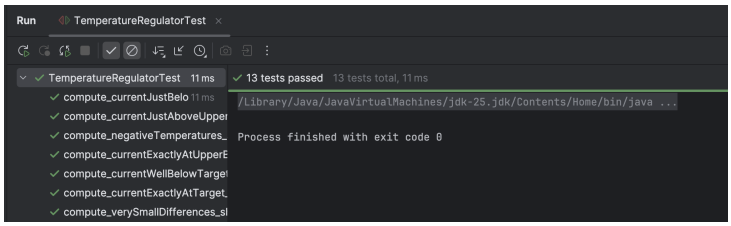


Figure 5: This is my image