

# 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

```
5     class CalculatorTest {
6
7         void Subtraction_Test() {
8             int result = calc.sub( a: 2, b: 3 );
9
10            assertEquals( expected: -1, result, message: "2 - 3 should equal -1");
11        }
12
13        @Test
14        void Multiplication_Test() {
15            Calculator calc = new Calculator();
16            int result = calc.mul( a: 2, b: 3 );
17            assertEquals( expected: 6, result, message: "2 * 3 should equal 6");
18        }
19
20        @Test
21        void Division_Test() {
22            Calculator calc = new Calculator();
23            int result = calc.div( a: 2, b: 3 );
24            assertEquals( expected: 0, result, message: "division by 2 should equal 1");
25        }
26
27    }
28
29    ✅
30
31 }
```

Run CalculatorTest ✖

↻ ↻ ⟳ ⟳ ⌚ ⌚ 📸 📸 :

<span style="color: #007bff;">✓</span> CalculatorTest	54 ms
<span style="color: #007bff;">✓</span> Subtraction_Test()	40 ms
<span style="color: #007bff;">✓</span> add_twoPositiveNumbers	12 ms
<span style="color: #007bff;">✓</span> Multiplication_Test()	1 ms
<span style="color: #007bff;">✓</span> Division_Test()	1ms

✓ 4 tests passed 4 tests total, 54 ms  
"C:\Program Files\Java\jdk-25\bin\java.exe" ...  
Process finished with exit code 0

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

The screenshot shows an IDE interface with a code editor and a run-time console.

**Code Editor Content:**

```

1 import org.junit.jupiter.api.Test ;
2 import static org.junit.jupiter.api.Assertions.*;
3
4
5 class CalculatorTest {
6     @Test
7     void add_twoPositiveNumbers_shouldReturnSum() {
8
9         Calculator calc = new Calculator();
10        int result = calc.add( a: 2, b: 3);
11
12        assertEquals( expected: 5, result, message: "2 + 3 should equal 5");
13    }
14
15    @Test
16    void Subtraction_Test() {
17        Calculator calc = new Calculator();
18        int result = calc.sub( a: 2, b: 3);
19

```

**Run Tab:**

- CalculatorTest
- Run button
- Stop button
- Reset button
- Output tab (highlighted)
- More options

**Output Tab Content:**

```

CalculatorTest 65 ms
  ✅ Subtraction_Test() 37 ms
  ✗ add_twoPositiveNumbers 27 ms
    org.opentest4j.AssertionFailedError: 2 + 3 should equal 5 ==>
      Expected :5
      Actual   :-1
      <Click to see difference>
  ✅ Multiplication_Test() 1ms
  ✅ Division_Test()

```

**Console Output:**

```

C:\Program Files\Java\jdk-25\bin\java.exe" ...

```

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 ( $\pm 0.5$  degrees).

### 3.4 Test Execution

## 4 Conclusion

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