

# TP08

Salma Bellaou

November 2025

## 1 exercise 01

```
public class Calculator { no usages

    /**
     * Returns the sum of two integers.
     */
    public int add(int a, int b) {
        return a + b;
    }
}
```

Figure 1: Calculator class

```

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

public class CalculatorTest {

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

        // Act
        int result = calc.add(a: 2, b: 3);

        // Assert
        assertEquals( expected: 5, result, message: "2 + 3 should equal 5");
    }
}

```

Figure 2: test class

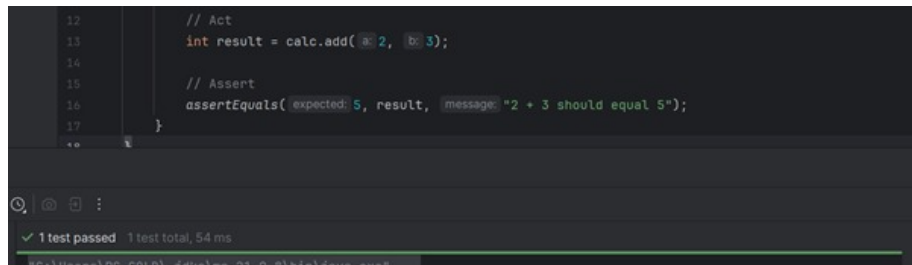


Figure 3: Test passed

```

public class Calculator { 2 usages

    /**
     * Returns the sum of two integers.
     */
    public int add(int a, int b) {
        return a - b;
    }
}

```

Figure 4: replacing the addition with the subtraction

```
11
12     // Act
13     int result = calc.add( a: 2, b: 3);
14
15     // Assert
16     assertEquals( expected: 5, result, message: "2 + 3 should equal 5");
17 }
18
19
```

1 test failed 1 test total, 87 ms

"C:\Users\BPC\AppData\Local\Temp\21\_0\_0\bin\java.exe"

Figure 5: Test failed

```
public class Calculator { 2 usages

    /**
     * Returns the sum of two integers.
     */
    public int add(int a, int b) {
        return a + b;
    }

    public int subtract(int a,int b){ no usages
        return a - b;
    }

    public int multiply(int a, int b){ no usages
        return a * b;
    }

    public float divide(int a, int b){ no usages
        return a / b;
    }
}
```

Figure 6: adding new methods to Calculator

```
void subtractTwoPositiveNumbersShouldReturnSubtraction() { no usages
    // Arrange
    Calculator calc = new Calculator();

    // Act
    int result = calc.subtract( a: 2, b: 3);

    // Assert
    assertEquals( expected: -1, result, message: "2 - 3 should equal -1");
}
```

Figure 7: adding subtract tester

```

}

void multiplyTwoNumbersShouldReturnmultiplication() { no usages
    // Arrange
    Calculator calc = new Calculator();

    // Act
    int result = calc.multiply( a: 2, b: 3);

    // Assert
    assertEquals( expected: 6, result, message: "2 * 3 should equal 6");
}

```

Figure 8: adding multiply tester

```

void divideTwoNumbersShouldReturndivision() { no usages
    // Arrange
    Calculator calc = new Calculator();

    // Act
    float result = calc.divide( a: 3, b: 2);

    // Assert
    assertEquals( expected: 1.5, result, message: "2 / 3 should equal 1.5");
}
}

```

Figure 9: Adding divide tester

```

42 // Act
43 float result = calc.divide( a: 3, b: 2);
44
45 // Assert
46 assertEquals( expected: 1.5, result, message: "2 / 3 should equal 1.5");
47 }
48 }

```

8 ms ✓ 1 test passed 1 test total, 58 ms

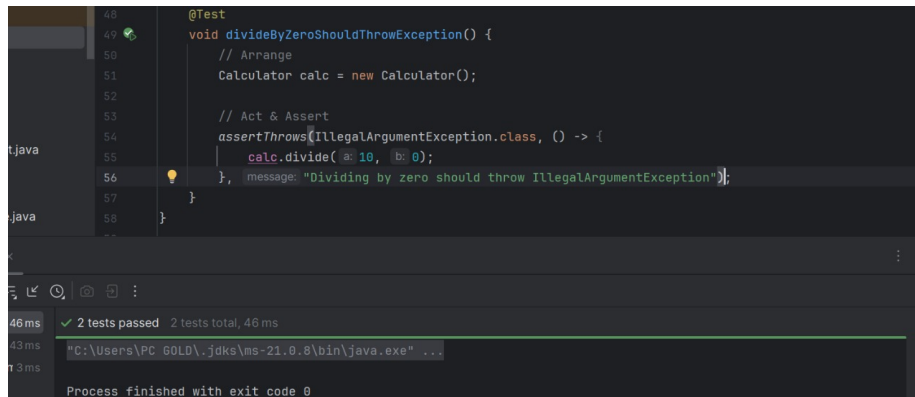
Figure 10: Test passed

```

public float divide(int a, int b){ 2 usages
    if (b == 0) {
        throw new IllegalArgumentException("Cannot divide by zero");
    }
    return a / b;
}

```

Figure 11: Editing the divide method



```
48      @Test
49      void divideByZeroShouldThrowException() {
50          // Arrange
51          Calculator calc = new Calculator();
52
53          // Act & Assert
54          assertThrows(IllegalArgumentException.class, () -> {
55              calc.divide(10, 0);
56          }, message: "Dividing by zero should throw IllegalArgumentException");
57      }
58  }
```

46 ms ✓ 2 tests passed 2 tests total, 46 ms  
"C:\Users\PC 60LD\jdk\ms-21.0.8\bin\java.exe" ...  
Process finished with exit code 0

Figure 12: Editing CalculatorTest and we find that the test has passed

## 2 Exercise 02

### Task 1: Boundary Value Analysis Test Cases

The regulator has a tolerance zone of  $\pm 0.5^{\circ}\text{C}$  around the target temperature. The following test cases are derived using Boundary Value Analysis (BVA):

1. **Test 1:** Current temperature = target -  $0.51^{\circ}\text{C}$  → Expected action: HEAT
2. **Test 2:** Current temperature = target -  $0.50^{\circ}\text{C}$  → Expected action: STANDBY
3. **Test 3:** Current temperature = target -  $0.49^{\circ}\text{C}$  → Expected action: STANDBY
4. **Test 4:** Current temperature = target → Expected action: STANDBY
5. **Test 5:** Current temperature = target +  $0.49^{\circ}\text{C}$  → Expected action: STANDBY
6. **Test 6:** Current temperature = target +  $0.50^{\circ}\text{C}$  → Expected action: STANDBY
7. **Test 7:** Current temperature = target +  $0.51^{\circ}\text{C}$  → Expected action: COOL