TP 8

Zineb El Ansari

November 2025

1 Exercice 1

```java
public class Calculator {  2 usages
    /**
     * Returns the sum of two integers.
     */
    public int add(int a, int b) {
        return a + b;
    }
}
```

Figure 1: Create the Calculator class

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


class CalculatorTest {

    @Test
    void add_twoPositiveNumbers_shouldReturnSum() {
        // 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: Create the Test Class

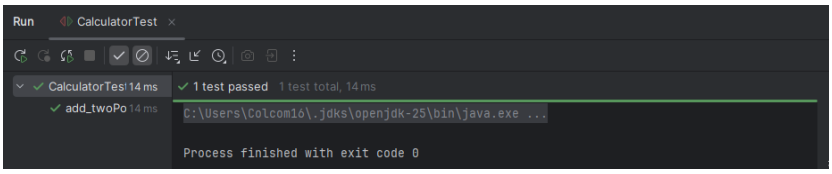


Figure 3: Test Output

```
public class Calculator {  2 usages
    /**
     * Returns the sum of two integers.
     */
    public int add(int a, int b) {
        return a - b;
    }
}
```

Figure 4: Temporarily modification of the implementation of the add method

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


class CalculatorTest {

    @Test
    void add_twoPositiveNumbers_shouldReturnSum() {
        // 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 5: Test Class of the temporarily modification of the implementation of the add method

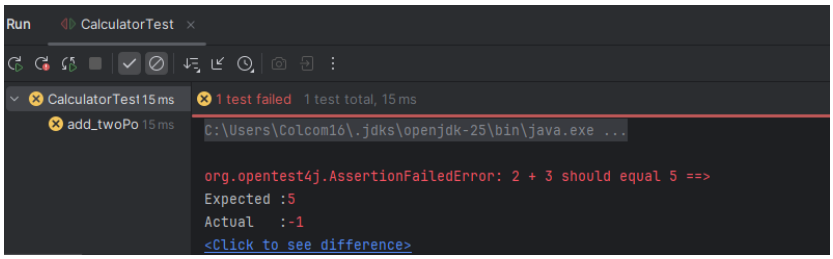
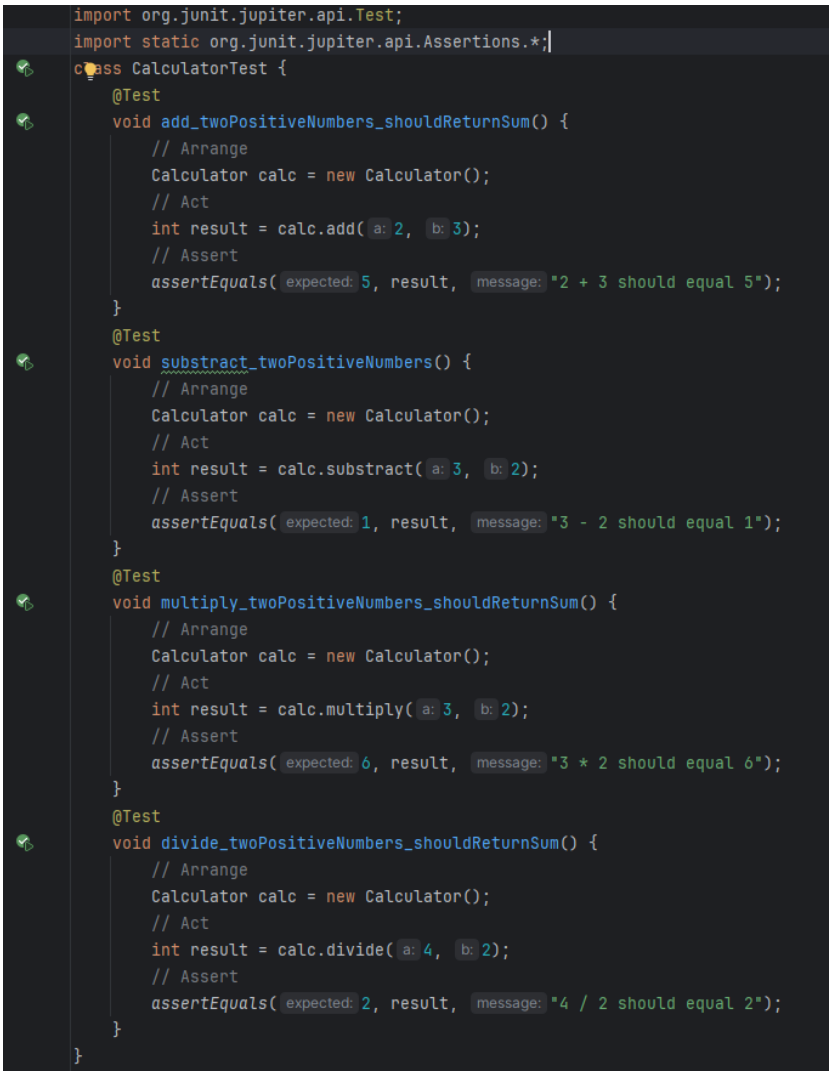


Figure 6: The test Output

```java
public class Calculator {  2 usages
    /**
     * Returns the sum of two integers.
     */
    public int add(int a, int b) {
        return a + b;
    }
    public int substract(int a, int b) {  no usages
        return a - b;
    }
    public int multiply(int a, int b) {  no usages
        return a * b;
    }
    public int divide(int a, int b) {  no usages
        return a / b;
    }
}
```

Figure 7: Calculator class with different methods



```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
class CalculatorTest {
    @Test
    void add_twoPositiveNumbers_shouldReturnSum() {
        // 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");
    }
    @Test
    void substract_twoPositiveNumbers() {
        // Arrange
        Calculator calc = new Calculator();
        // Act
        int result = calc.substract( a: 3,  b: 2);
        // Assert
        assertEquals( expected: 1, result,  message: "3 - 2 should equal 1");
    }
    @Test
    void multiply_twoPositiveNumbers_shouldReturnSum() {
        // Arrange
        Calculator calc = new Calculator();
        // Act
        int result = calc.multiply( a: 3,  b: 2);
        // Assert
        assertEquals( expected: 6, result,  message: "3 * 2 should equal 6");
    }
    @Test
    void divide_twoPositiveNumbers_shouldReturnSum() {
        // Arrange
        Calculator calc = new Calculator();
        // Act
        int result = calc.divide( a: 4,  b: 2);
        // Assert
        assertEquals( expected: 2, result,  message: "4 / 2 should equal 2");
    }
}
```

Figure 8: Test class with different methods

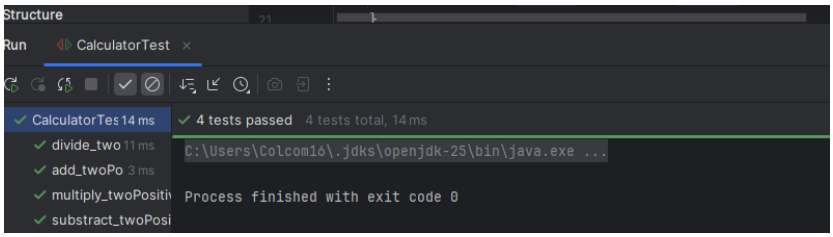


Figure 9: The test Output

```java
public class Calculator {  8 usages
    /**
     * Returns the sum of two integers.
     */
    public int add(int a, int b) {
        return a + b;
    }
    public int substract(int a, int b) {  1 usage
        return a - b;
    }
    public int multiply(int a, int b) {  1 usage
        return a * b;
    }
    public int divide(int a, int b) {  1 usage
        if (b == 0) {
            throw new ArithmeticException("Cannot divide by zero");
        }
        return a / b;
    }
}
```

Figure 10: Calculator Class with Exception Handling

```
    }
    @Test
    public void testDivideByZeroThrowsException() {
        Calculator calc = new Calculator();
        assertThrows(ArithmeticException.class, () -> {
            calc.divide( a: 10,  b: 0);
        });
    }
}
```

Figure 11: Test Class of Exception Handling

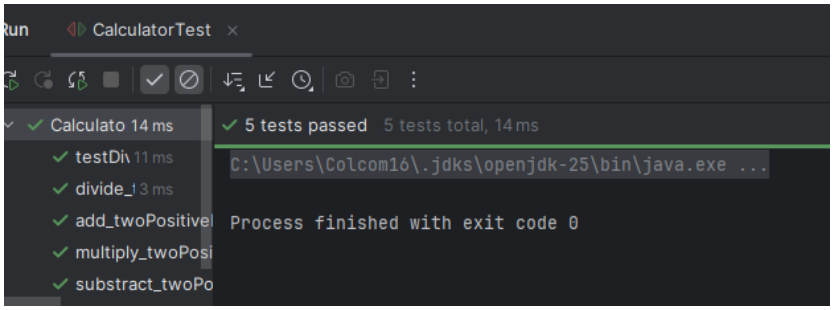


Figure 12: The Test Output

2 Exercice 2

```
public class TemperatureRegulator {  no usages

    public enum Action { HEAT, COOL, STANDBY }  4 usages
    public Action compute(double current, double target) {  no usages
        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;
        }
    }
}
```

Figure 13: Create Class Temperature Regulator

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

class TemperatureRegulatorTest {

    private final TemperatureRegulator regulator = new TemperatureRegulator();  9 usages

    @Test
    void testHeatBelowLowerBoundary() {
        assertEquals(TemperatureRegulator.Action.HEAT,
                regulator.compute( current: 19.4,  target: 20.0));
    }

    @Test
    void testStandbyAtLowerBoundary() {
        assertEquals(TemperatureRegulator.Action.STANDBY,
                regulator.compute( current: 19.5,  target: 20.0));
    }

    @Test
    void testStandbyAboveLowerBoundary() {
        assertEquals(TemperatureRegulator.Action.STANDBY,
                regulator.compute( current: 19.6,  target: 20.0));
    }

    @Test
    void testStandbyBelowUpperBoundary() {
        assertEquals(TemperatureRegulator.Action.STANDBY,
                regulator.compute( current: 20.4,  target: 20.0));
    }

    @Test
    void testStandbyAtUpperBoundary() {
        assertEquals(TemperatureRegulator.Action.STANDBY,
                regulator.compute( current: 20.5,  target: 20.0));
    }
```

Figure 14: Create The Test Class

```java
    @Test
    void testCoolAboveUpperBoundary() {
        assertEquals(TemperatureRegulator.Action.COOL,
                regulator.compute( current: 20.6,  target: 20.0));
    }

    @Test
    void testStandbyExactMatch() {
        assertEquals(TemperatureRegulator.Action.STANDBY,
                regulator.compute( current: 20.0,  target: 20.0));
    }

    @Test
    void testHeatFarBelow() {
        assertEquals(TemperatureRegulator.Action.HEAT,
                regulator.compute( current: 18.0,  target: 20.0));
    }

    @Test
    void testCoolFarAbove() {
        assertEquals(TemperatureRegulator.Action.COOL,
                regulator.compute( current: 23.0,  target: 20.0));
    }

}
```

Figure 15: The rest of the Test Class

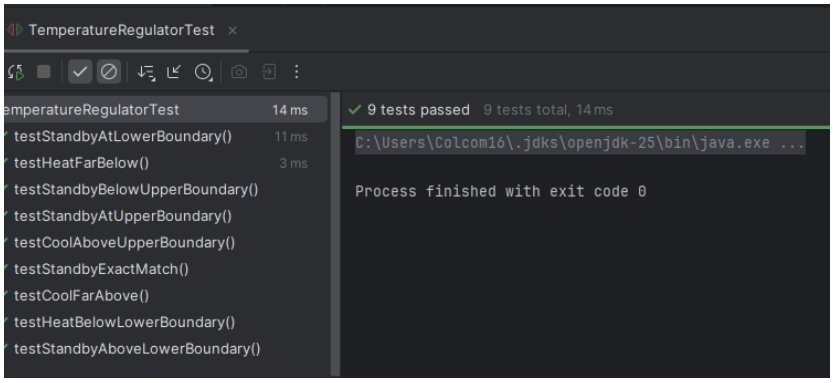


Figure 16: The Test Output

3 Exercice 3

```java
public class PolynomeSecondDegre {  no usages
    private double a;  5 usages
    private double b;  5 usages
    private double c;  2 usages

    public PolynomeSecondDegre(double a, double b, double c) {  no usages
        this.a = a;
        this.b = b;
        this.c = c;
    }

    public double[] roots() {  no usages
        double dis = Math.pow(b, 2) - 4 * a * c;
        if (dis > 0) {
            double root1 = (-b - Math.sqrt(dis)) / (2 * a);
            double root2 = (-b + Math.sqrt(dis)) / (2 * a);
            return new double[]{root1, root2};
        } else if (dis == 0) {
            double root = -b / (2 * a);
            return new double[]{root};
        } else {
            throw new ArithmeticException("There are no real roots");
        }
    }
}
```

Figure 17: Create Class Polynome Second Degre