# Software Testing Exercise 1 Solution

## Exercise Overview

This exercise involves creating a `Calculator` class and writing comprehensive unit tests using JUnit 5, including exception handling for edge cases.

## 1 Step 7: Extended Calculator Class

### 1.1 Implementation

```java
public class Calculator {

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

    /**
     * Returns the difference between two integers.
     */
    public int subtract(int a, int b) {
        return a - b;
    }

    /**
     * Returns the product of two integers.
     */
    public int multiply(int a, int b) {
        return a * b;
    }

    /**
     * Returns the quotient of two integers.
     * @throws ArithmeticException if b is zero
     */
    public int divide(int a, int b) {
        if (b == 0) {
            throw new ArithmeticException("Division by zero is not allowed");
        }
        return a / b;
    }
}
```

Listing 1: Calculator.java - Complete Implementation

## 2 Step 7 & 8: Complete Test Class

### 2.1 Test Implementation with All Methods

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

class CalculatorTest {

    private Calculator calc;

    @BeforeEach
    void setUp() {
        calc = new Calculator();
    }

    // ========== ADD METHOD TESTS ==========

    @Test
    void add_twoPositiveNumbers_shouldReturnSum() {
        // Arrange
        // (already done in setUp)

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

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

    @Test
    void add_twoNegativeNumbers_shouldReturnSum() {
        // Act
        int result = calc.add(-2, -3);

        // Assert
        assertEquals(-5, result, "-2␣+␣(-3)␣should␣equal␣-5");
    }

    @Test
    void add_positiveAndNegative_shouldReturnSum() {
        // Act
        int result = calc.add(5, -3);

        // Assert
        assertEquals(2, result, "5␣+␣(-3)␣should␣equal␣2");
    }

    // ========== SUBTRACT METHOD TESTS ==========

    @Test
    void subtract_twoPositiveNumbers_shouldReturnDifference() {
        // Act
        int result = calc.subtract(5, 3);

        // Assert
        assertEquals(2, result, "5␣-␣3␣should␣equal␣2");
    }

    @Test
    void subtract_resultIsNegative_shouldReturnNegativeDifference
       () {
        // Act
        int result = calc.subtract(3, 5);

        // Assert
        assertEquals(-2, result, "3␣-␣5␣should␣equal␣-2");
    }

    @Test
    void subtract_twoNegativeNumbers_shouldReturnDifference() {
        // Act
        int result = calc.subtract(-5, -3);

        // Assert
        assertEquals(-2, result, "-5␣-␣(-3)␣should␣equal␣-2");
    }

    // ========== MULTIPLY METHOD TESTS ==========

    @Test
    void multiply_twoPositiveNumbers_shouldReturnProduct() {
        // Act
        int result = calc.multiply(4, 5);

        // Assert
        assertEquals(20, result, "4␣*␣5␣should␣equal␣20");
    }

    @Test
    void multiply_byZero_shouldReturnZero() {
        // Act
        int result = calc.multiply(5, 0);
```

```java

        // Assert
        assertEquals(0, result, "5␣*␣0␣should␣equal␣0");
    }

    @Test
    void multiply_twoNegativeNumbers_shouldReturnPositive() {
        // Act
        int result = calc.multiply(-3, -4);

        // Assert
        assertEquals(12, result, "-3␣*␣-4␣should␣equal␣12");
    }

    @Test
    void multiply_positiveAndNegative_shouldReturnNegative() {
        // Act
        int result = calc.multiply(3, -4);

        // Assert
        assertEquals(-12, result, "3␣*␣-4␣should␣equal␣-12");
    }

    // ========== DIVIDE METHOD TESTS ==========

    @Test
    void divide_twoPositiveNumbers_shouldReturnQuotient() {
        // Act
        int result = calc.divide(10, 2);

        // Assert
        assertEquals(5, result, "10␣/␣2␣should␣equal␣5");
    }

    @Test
    void divide_resultWithRemainder_shouldReturnIntegerQuotient()
        {
        // Act
        int result = calc.divide(10, 3);

        // Assert
        assertEquals(3, result, "10␣/␣3␣should␣equal␣3␣(integer␣
           division)");
    }

    @Test
    void divide_negativeByPositive_shouldReturnNegativeQuotient()
        {
        // Act
        int result = calc.divide(-10, 2);

        // Assert
        assertEquals(-5, result, "-10 / 2 should equal -5");
    }

    // ========== EXCEPTION HANDLING TESTS ==========

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

        // Optionally verify the exception message
        assertEquals("Division by zero is not allowed", exception
           .getMessage());
    }

    @Test
    void divide_zeroByNonZero_shouldReturnZero() {
        // Act
        int result = calc.divide(0, 5);

        // Assert
        assertEquals(0, result, "0 / 5 should equal 0");
    }
}
```

Listing 2: CalculatorTest.java - Complete Test Suite

## 3 Test Execution Results

### 3.1 Expected Test Outcomes

When running the complete test suite, all 16 tests should pass:

- **Add tests (3):** Verify addition with positive, negative, and mixed numbers
- **Subtract tests (3):** Verify subtraction in various scenarios
- **Multiply tests (4):** Verify multiplication including edge cases with zero
- **Divide tests (4):** Verify division including integer division behavior
- **Exception tests (2):** Verify proper exception handling for division by zero

### 3.2 Key Testing Concepts Demonstrated

1. **Arrange-Act-Assert Pattern:** Each test follows the AAA structure for clarity

2. **@BeforeEach:** The `setUp()` method initializes a fresh Calculator instance before each test
3. **Multiple Assertions:** Using `assertEquals` for value verification
4. **Exception Testing:** Using `assertThrows` to verify exception behavior
5. **Edge Cases:** Testing with zero, negative numbers, and boundary conditions
6. **Descriptive Names:** Test method names clearly describe what is being tested

## 4 Running the Tests

### 4.1 In IntelliJ IDEA

1. Right-click on `CalculatorTest.java`
2. Select **Run 'CalculatorTest'**
3. View results in the Run tool window
4. Green checkmarks = all tests passed

### 4.2 Expected Output

```
Test Results:
  CalculatorTest
     add_twoPositiveNumbers_shouldReturnSum
     add_twoNegativeNumbers_shouldReturnSum
     add_positiveAndNegative_shouldReturnSum
     subtract_twoPositiveNumbers_shouldReturnDifference
     subtract_resultIsNegative_shouldReturnNegativeDifference
     subtract_twoNegativeNumbers_shouldReturnDifference
     multiply_twoPositiveNumbers_shouldReturnProduct
     multiply_byZero_shouldReturnZero
     multiply_twoNegativeNumbers_shouldReturnPositive
     multiply_positiveAndNegative_shouldReturnNegative
     divide_twoPositiveNumbers_shouldReturnQuotient
     divide_resultWithRemainder_shouldReturnIntegerQuotient
     divide_negativeByPositive_shouldReturnNegativeQuotient
     divide_byZero_shouldThrowArithmeticException
     divide_zeroByNonZero_shouldReturnZero

Total: 15 tests passed
```