

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

```
1 public class Calculator {
2
3     /**
4      * Returns the sum of two integers.
5      */
6     public int add(int a, int b) {
7         return a + b;
8     }
9
10    /**
11     * Returns the difference between two integers.
12     */
13    public int subtract(int a, int b) {
14        return a - b;
15    }
16
17    /**
18     * Returns the product of two integers.
19     */
20    public int multiply(int a, int b) {
21        return a * b;
22    }
23
24    /**
25     * Returns the quotient of two integers.
26     * @throws ArithmeticException if b is zero
27     */
28    public int divide(int a, int b) {
29        if (b == 0) {
30            throw new ArithmeticException("Division by zero is not allowed");
31        }
32    }
33 }
```

```

31     }
32     return a / b;
33 }
34 }

```

Listing 1: Calculator.java - Complete Implementation

2 Step 7 & 8: Complete Test Class

2.1 Test Implementation with All Methods

```

1  import org.junit.jupiter.api.Test;
2  import org.junit.jupiter.api.BeforeEach;
3  import static org.junit.jupiter.api.Assertions.*;
4
5  class CalculatorTest {
6
7      private Calculator calc;
8
9      @BeforeEach
10     void setUp() {
11         calc = new Calculator();
12     }
13
14     // ===== ADD METHOD TESTS =====
15
16     @Test
17     void add_twoPositiveNumbers_shouldReturnSum() {
18         // Arrange
19         // (already done in setUp)
20
21         // Act
22         int result = calc.add(2, 3);
23
24         // Assert
25         assertEquals(5, result, "2+3 should equal 5");
26     }
27
28     @Test
29     void add_twoNegativeNumbers_shouldReturnSum() {
30         // Act
31         int result = calc.add(-2, -3);
32
33         // Assert
34         assertEquals(-5, result, "-2+(-3) should equal -5");
35     }
36
37     @Test
38     void add_positiveAndNegative_shouldReturnSum() {
39         // Act

```

```

40     int result = calc.add(5, -3);
41
42     // Assert
43     assertEquals(2, result, "5+(-3) should equal 2");
44 }
45
46 // ===== SUBTRACT METHOD TESTS =====
47
48 @Test
49 void subtract_twoPositiveNumbers_shouldReturnDifference() {
50     // Act
51     int result = calc.subtract(5, 3);
52
53     // Assert
54     assertEquals(2, result, "5-3 should equal 2");
55 }
56
57 @Test
58 void subtract_resultIsNegative_shouldReturnNegativeDifference
59     () {
60     // Act
61     int result = calc.subtract(3, 5);
62
63     // Assert
64     assertEquals(-2, result, "3-5 should equal -2");
65 }
66
67 @Test
68 void subtract_twoNegativeNumbers_shouldReturnDifference() {
69     // Act
70     int result = calc.subtract(-5, -3);
71
72     // Assert
73     assertEquals(-2, result, "-5-(-3) should equal -2");
74 }
75
76 // ===== MULTIPLY METHOD TESTS =====
77
78 @Test
79 void multiply_twoPositiveNumbers_shouldReturnProduct() {
80     // Act
81     int result = calc.multiply(4, 5);
82
83     // Assert
84     assertEquals(20, result, "4*5 should equal 20");
85 }
86
87 @Test
88 void multiply_byZero_shouldReturnZero() {
89     // Act
90     int result = calc.multiply(5, 0);

```

```

90         // Assert
91         assertEquals(0, result, "5*0 should equal 0");
92     }
93
94
95     @Test
96     void multiply_twoNegativeNumbers_shouldReturnPositive() {
97         // Act
98         int result = calc.multiply(-3, -4);
99
100        // Assert
101        assertEquals(12, result, "-3*-4 should equal 12");
102    }
103
104    @Test
105    void multiply_positiveAndNegative_shouldReturnNegative() {
106        // Act
107        int result = calc.multiply(3, -4);
108
109        // Assert
110        assertEquals(-12, result, "3*-4 should equal -12");
111    }
112
113    // ===== DIVIDE METHOD TESTS =====
114
115    @Test
116    void divide_twoPositiveNumbers_shouldReturnQuotient() {
117        // Act
118        int result = calc.divide(10, 2);
119
120        // Assert
121        assertEquals(5, result, "10/2 should equal 5");
122    }
123
124    @Test
125    void divide_resultWithRemainder_shouldReturnIntegerQuotient()
126    {
127        // Act
128        int result = calc.divide(10, 3);
129
130        // Assert
131        assertEquals(3, result, "10/3 should equal 3 (integer division)");
132    }
133
134    @Test
135    void divide_negativeByPositive_shouldReturnNegativeQuotient()
136    {
137        // Act
138        int result = calc.divide(-10, 2);
139    }

```

```

138         // Assert
139         assertEquals(-5, result, "-10_/_2_should_equal_-5");
140     }
141
142     // ===== EXCEPTION HANDLING TESTS =====
143
144     @Test
145     void divide_byZero_shouldThrowArithmeticException() {
146         // Act & Assert
147         ArithmeticException exception = assertThrows(
148             ArithmeticException.class,
149             () -> calc.divide(10, 0),
150             "Division_by_zero_should_throw_ArithmeticException"
151         );
152
153         // Optionally verify the exception message
154         assertEquals("Division_by_zero_is_not_allowed", exception
155             .getMessage());
156     }
157
158     @Test
159     void divide_zeroByNonZero_shouldReturnZero() {
160         // Act
161         int result = calc.divide(0, 5);
162
163         // Assert
164         assertEquals(0, result, "0_/_5_should_equal_0");
165     }
166 }

```

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