Let's extend the `Calculator` class by adding more methods. I'll include additional common methods like subtraction, multiplication, division, and a method for calculating the factorial of a number. Afterward, I'll also update the `CalculatorTest` class with tests for each of these new methods.

Updated `Calculator` Class with More Methods

```
package com.example;

public class Calculator {

    // Method to add two numbers
    public int add(int a, int b) {
        return a + b;
    }

    // Method to subtract two numbers
    public int subtract(int a, int b) {
        return a - b;
    }

    // Method to multiply two numbers
    public int multiply(int a, int b) {
        return a * b;
    }

    // Method to divide two numbers
    public double divide(int a, int b) {
        if (b == 0) {
            throw new ArithmeticException("Cannot divide by zero");
        }
        return (double) a / b;
    }

    // Method to calculate factorial of a number
    public int factorial(int n) {
        if (n < 0) {
            throw new IllegalArgumentException("Factorial is not defined
for negative numbers");
        }
```

```
        int result = 1;
        for (int i = 1; i <= n; i++) {
            result *= i;
        }
        return result;
    }
}
```

Explanation of Each Method Added

1. **`subtract(int a, int b)`**:
    - Returns the difference of `a - b`.
2. **`multiply(int a, int b)`**:
    - Returns the product of `a * b`.
3. **`divide(int a, int b)`**:
    - Returns the result of dividing `a` by `b`. It checks if `b` is 0 to prevent division by zero, and throws an `ArithmeticException` if `b` is 0.
4. **`factorial(int n)`**:
    - Returns the factorial of `n` (n!). Throws an `IllegalArgumentException` if `n` is negative, as factorials are only defined for non-negative integers.
    - Uses a `for` loop to calculate the factorial.

Updated `CalculatorTest` Class with Tests for New Methods

We will now add corresponding test cases for the new methods we added to the `Calculator` class.

```
package com.example;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

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

public class CalculatorTest {
```

```
    private Calculator calculator;

    // Initialize a fresh Calculator instance before each test
    @BeforeEach
    public void setup() {
        calculator = new Calculator();
    }

    // Test for addition
    @Test
    public void testAddition() {
        int result = calculator.add(2, 3);
        assertEquals(5, result, "The addition result should be 5");
    }

    // Test for subtraction
    @Test
    public void testSubtraction() {
        int result = calculator.subtract(5, 3);
        assertEquals(2, result, "The subtraction result should be 2");
    }

    // Test for multiplication
    @Test
    public void testMultiplication() {
        int result = calculator.multiply(4, 3);
        assertEquals(12, result, "The multiplication result should be 12");
    }

    // Test for division
    @Test
    public void testDivision() {
        double result = calculator.divide(6, 2);
        assertEquals(3.0, result, 0.001, "The division result should be
3.0");

        // Test division by zero
        assertThrows(ArithmeticException.class, () -> calculator.divide(1,
0), "Division by zero should throw an ArithmeticException");
    }

    // Test for factorial
```

```
    @Test
    public void testFactorial() {
        int result = calculator.factorial(5);
        assertEquals(120, result, "The factorial of 5 should be 120");

        // Test factorial of 0
        result = calculator.factorial(0);
        assertEquals(1, result, "The factorial of 0 should be 1");

        // Test factorial of negative number (should throw exception)
        assertThrows(IllegalArgumentException.class, () ->
calculator.factorial(-1), "Factorial of negative numbers should throw an
IllegalArgumentException");
    }
}
```

Explanation of the Test Methods

1. **`testAddition()`**:
   - Tests the addition method, verifying that adding 2 and 3 returns 5.
2. **`testSubtraction()`**:
   - Tests the subtraction method, verifying that subtracting 3 from 5 returns 2.
3. **`testMultiplication()`**:
   - Tests the multiplication method, verifying that multiplying 4 and 3 returns 12.
4. **`testDivision()`**:
   - Tests the division method, verifying that dividing 6 by 2 returns 3.0.
   - Also tests division by zero to ensure it throws an `ArithmeticException`.
5. **`testFactorial()`**:
   - Tests the factorial method, verifying that the factorial of 5 is 120.
   - Also verifies that the factorial of 0 is 1.
   - Ensures that passing a negative number to the factorial method throws an `IllegalArgumentException`.
