Day 27

Assignment 2: Extend the above JUnit tests to use @Before, @After, @BeforeClass, and @AfterClass annotations to manage test setup and teardown.

A)

Sure! Here’s a detailed explanation of the provided code:

MathematicalOperationsTest Class:

This class uses JUnit 5 to test the MathematicalOperations class, which presumably has methods for basic arithmetic operations.

Java code:

```
import org.junit.jupiter.api.*;

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


@TestInstance(TestInstance.Lifecycle.PER_CLASS)

public class MathematicalOperationsTest {

    private MathematicalOperations mathOps;


    @BeforeAll

    public void setUpBeforeClass() {

        System.out.println("Setup before all tests");

        // Code that needs to be executed before all tests

    }


    @AfterAll

    public void tearDownAfterClass() {

        System.out.println("Cleanup after all tests");

        // Code that needs to be executed after all tests

    }
```

```
@BeforeEach
public void setUp() {
    System.out.println("Setup before each test");
    mathOps = new MathematicalOperations();
}

@AfterEach
public void tearDown() {
    System.out.println("Cleanup after each test");
    // Code that needs to be executed after each test
}

@Test
public void testAdd() {
    assertEquals(5, mathOps.add(2, 3));
    assertEquals(-1, mathOps.add(2, -3));
    assertEquals(0, mathOps.add(-2, 2));
}

@Test
public void testSubtract() {
    assertEquals(1, mathOps.subtract(3, 2));
    assertEquals(5, mathOps.subtract(2, -3));
    assertEquals(-4, mathOps.subtract(-2, 2));
}

@Test
public void testMultiply() {
    assertEquals(6, mathOps.multiply(2, 3));
```

```
        assertEquals(-6, mathOps.multiply(2, -3));
        assertEquals(4, mathOps.multiply(-2, -2));
    }

    @Test
    public void testDivide() {
        assertEquals(2, mathOps.divide(6, 3));
        assertEquals(-2, mathOps.divide(6, -3));
        assertEquals(1, mathOps.divide(-3, -3));
    }

    @Test
    public void testDivideByZero() {
        Exception exception = assertThrows(IllegalArgumentException.class, () -> {
            mathOps.divide(1, 0);
        });
        assertEquals("Division by zero is not allowed.", exception.getMessage());
    }
}
```

Explanation:

*- *Test Lifecycle Management*:*

*- *@TestInstance(TestInstance.Lifecycle.PER_CLASS)*: This annotation specifies that a single instance of the test class will be used to run all the test methods. This allows us to use @BeforeAll and @AfterAll instance methods.*

*- *Setup and Teardown Methods*:*

*- *@BeforeAll public void setUpBeforeClass()*: This method runs once before all tests. It's typically used to perform expensive or one-time setup activities. In this example, it prints a setup message.*

*- *@AfterAll public void tearDownAfterClass()*: This method runs once after all tests. It's typically used to clean up resources allocated in @BeforeAll. Here, it prints a cleanup message.*

*- *@BeforeEach public void setUp()*: This method runs before each test. It's used to set up the test environment. In this example, it initializes the mathOps object and prints a setup message.*

*- *@AfterEach public void tearDown()*: This method runs after each test. It's used to clean up the test environment. Here, it prints a cleanup message.*

*- *Test Methods*:*

*- *@Test public void testAdd()*: Tests the add method with different inputs and verifies the results using assertEquals.*

*- *@Test public void testSubtract()*: Tests the subtract method with different inputs and verifies the results using assertEquals.*

*- *@Test public void testMultiply()*: Tests the multiply method with different inputs and verifies the results using assertEquals.*

*- *@Test public void testDivide()*: Tests the divide method with different inputs and verifies the results using assertEquals.*

*- *@Test public void testDivideByZero()*: Tests the divide method to ensure it throws an IllegalArgumentException when attempting to divide by zero. It verifies the exception's message to ensure it matches "Division by zero is not allowed."*

Summary:

This test class sets up and tears down the test environment at appropriate points, ensuring a clean state for each test. It contains methods to verify the functionality of basic arithmetic operations, including edge cases like division by zero, using assertions to compare expected and actual outcomes.