Day 27:

Assignment 1: Write a set of JUnit tests for a given class with simple mathematical operations (add, subtract, multiply, divide) using the basic @Test annotation.

A)Certainly! Here's a detailed explanation of the provided code:

MathOperations Class

This class contains basic arithmetic operations: addition, subtraction, multiplication, and division.

Java code:

```
public class MathOperations {

    public int add(int a, int b) {
        return a + b;
    }

    public int subtract(int a, int b) {
        return a - b;
    }

    public int multiply(int a, int b) {
        return a * b;
    }

    public double divide(int a, int b) {
        if (b == 0) {
            throw new IllegalArgumentException("Cannot divide by zero");
        }
        return (double) a / b;
    }
}
```

*- *add(int a, int b)*: Takes two integers and returns their sum.*

*- *subtract(int a, int b)*: Takes two integers and returns their difference.*

*- *multiply(int a, int b)*: Takes two integers and returns their product.*

*- *divide(int a, int b)*: Takes two integers and returns their quotient. If the divisor b is zero, it throws an IllegalArgumentException to prevent division by zero.*

MathOperationsTest Class

This class uses JUnit 5 to test the methods in the MathOperations class.

Java code:

```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
public class MathOperationsTest {
    private final MathOperations mathOperations = new MathOperations();
    @Test
    public void testAdd() {
        assertEquals(5, mathOperations.add(2, 3));
        assertEquals(0, mathOperations.add(-2, 2));
        assertEquals(-5, mathOperations.add(-2, -3));
    }
    @Test
    public void testSubtract() {
        assertEquals(1, mathOperations.subtract(3, 2));
        assertEquals(-4, mathOperations.subtract(-2, 2));
        assertEquals(1, mathOperations.subtract(-2, -3));
    }

    @Test
```

```
    public void testMultiply() {
        assertEquals(6, mathOperations.multiply(2, 3));
        assertEquals(-4, mathOperations.multiply(-2, 2));
        assertEquals(6, mathOperations.multiply(-2, -3));
    }
    @Test
    public void testDivide() {
        assertEquals(2.0, mathOperations.divide(6, 3), 0.001);
        assertEquals(-2.0, mathOperations.divide(-6, 3), 0.001);
        assertEquals(2.0, mathOperations.divide(-6, -3), 0.001);

        Exception exception = assertThrows(IllegalArgumentException.class, () -> {
            mathOperations.divide(1, 0);
        });
        assertEquals("Cannot divide by zero", exception.getMessage());
    }
}
```

- *testAdd()*: Verifies the correctness of the add method by checking various cases:
  - Positive integers
  - Negative and positive integer combination
  - Negative integers

- *testSubtract()*: Verifies the correctness of the subtract method by checking various cases:
  - Positive integers
  - Negative and positive integer combination
  - Negative integers

- *testMultiply()*: Verifies the correctness of the multiply method by checking various cases:

- Positive integers

- Negative and positive integer combination

- Negative integers

*- *testDivide()*: Verifies the correctness of the divide method by checking various cases:*

- Positive integers

- Negative and positive integer combination

- Negative integers

- Also tests the scenario where division by zero occurs, expecting an IllegalArgumentException with the message "Cannot divide by zero".

Each test method uses assertions to compare the expected output with the actual result from the MathOperations methods, ensuring the methods behave as expected. The assertThrows method is used to verify that the correct exception is thrown for invalid operations.