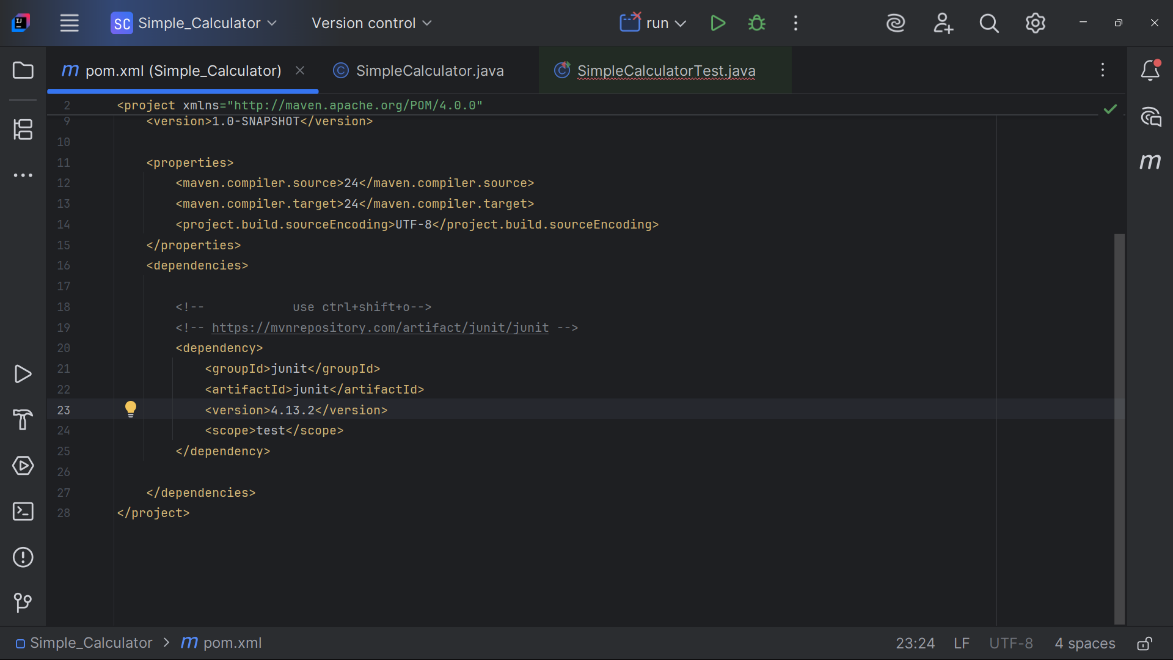
**JUnit\_Basic Testing Exercises**

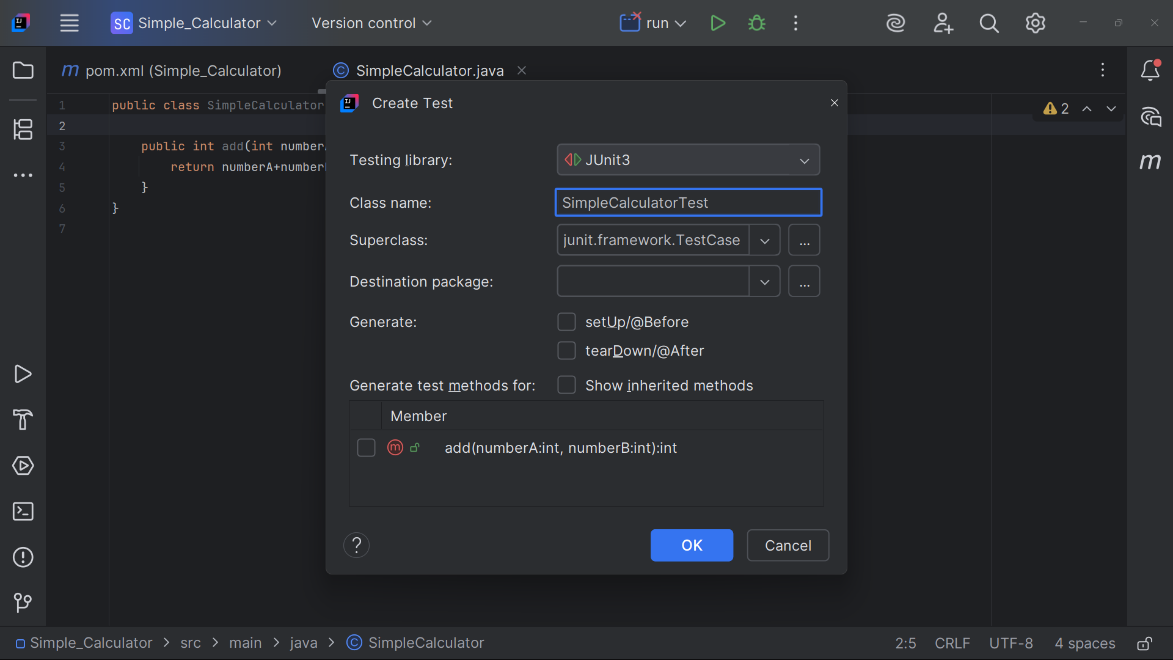
**(All exercise are included)**

**Exercise 1: Setting Up Junit**

--adding dependency in Maven

****

--creating test class

****

**Exercise 3: Assertions in Junit**

**CODE**

// **SimpleCalculator Class**

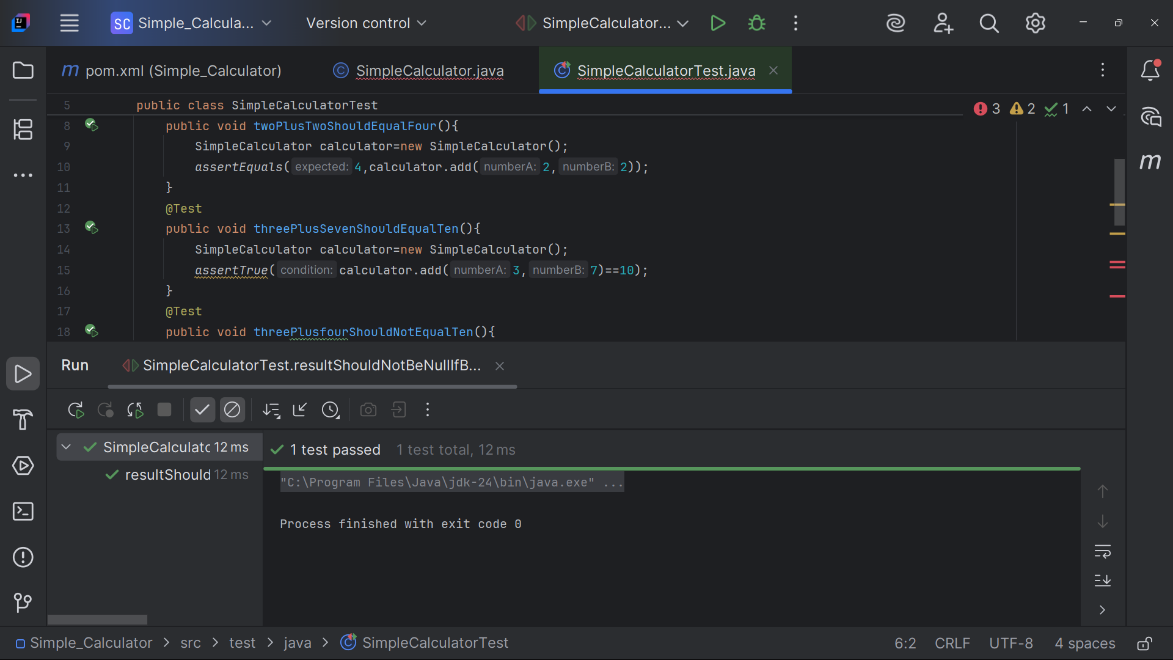
public class SimpleCalculator {  
  
 public int add(int numberA, int numberB){  
 return numberA+numberB;  
 }  
 public Integer safeAdd(Integer a, Integer b) {  
 if (a == null || b == null) {  
 return null;  
 }  
 return Integer.sum(a, b);  
 }  
}

// **SimpleCalculatorTest : Test class**

import static org.junit.Assert.\*;  
import org.junit.Test;  
  
public class SimpleCalculatorTest  
{  
 @Test  
 public void twoPlusTwoShouldEqualFour(){  
 SimpleCalculator calculator=new SimpleCalculator();  
 *assertEquals*(4,calculator.add(2,2));   
 }  
 @Test  
 public void threePlusSevenShouldEqualTen(){  
 SimpleCalculator calculator=new SimpleCalculator();  
 *assertTrue*(calculator.add(3,7)==10);  
 }  
 @Test  
 public void threePlusfourShouldNotEqualTen(){  
 SimpleCalculator calculator=new SimpleCalculator();  
 *assertFalse*(calculator.add(3,4)==10);  
 }  
 @Test  
 public void resultShouldBeNullIfAnyInputIsNull() {  
 SimpleCalculator calculator = new SimpleCalculator();  
 *assertNull*(calculator.safeAdd(null, 5));  
 *assertNull*(calculator.safeAdd(3, null));  
 }

@Test  
 public void resultShouldNotBeNullIfBothInputsAreNotNull() {  
 SimpleCalculator calculator = new SimpleCalculator();  
 *assertNotNull*(calculator.safeAdd(3, 4));  
 }  
}

**OUTPUT**

****

**Exercise 4: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in Junit**

**CODE**

//only changes is made on test class that is **SimpleCalculatorTest**

import static org.junit.Assert.\*;  
import org.junit.Test;  
import org.junit.Before;  
import org.junit.After;  
  
public class SimpleCalculatorTest  
{

private SimpleCalculator calculator;

@Before  
 public void setUp() {  
 calculator = new SimpleCalculator(); // Arrange  
 System.out.println("Setup: Calculator initialized.");  
 }

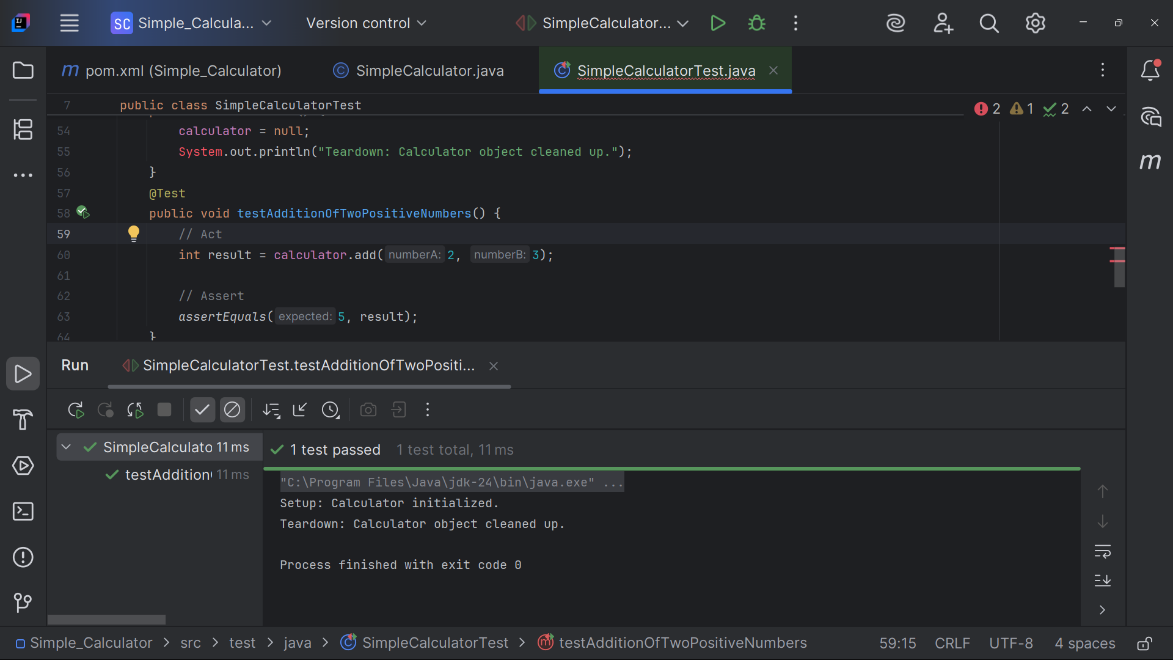
@After  
 public void tearDown() {  
 calculator = null;  
 System.out.println("Teardown: Calculator object cleaned up.");  
 }

@Test  
 public void testAdditionOfTwoPositiveNumbers() {  
   
 int result = calculator.add(2, 3);  
  
   
 assertEquals(5, result);  
 }

@Test  
 public void testAdditionOfPositiveAndNegativeNumber() {  
   
 int result = calculator.add(10, -4);  
  
 // Assert  
 assertEquals(6, result);  
 }

@Test  
 public void testAdditionWithZero() {  
   
 int result = calculator.add(0, 5);  
  
   
 assertEquals(5, result);  
 }  
}

**OUTPUT**

****

**Mockito Exercises**

**Exercise 1: Mocking and Stubbing**

**CODE**

//**external service**

public interface MathService {

int add(int a, int b);

}

//**SimpleCalculator**

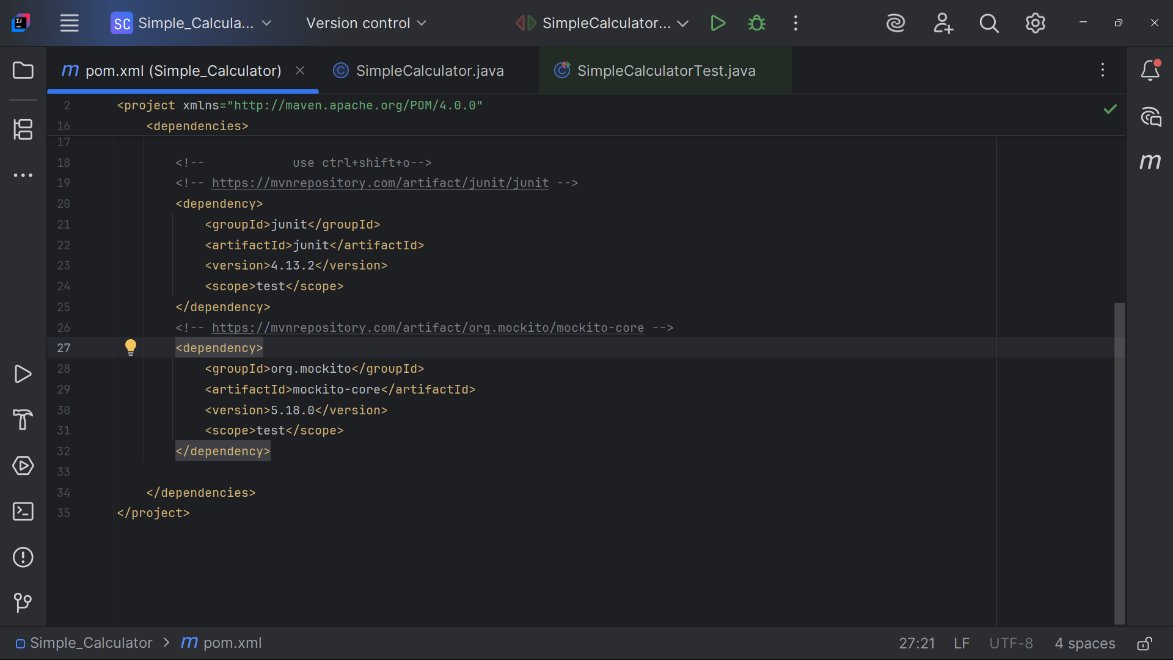
public class SimpleCalculator {  
 private MathService mathService;  
  
 public SimpleCalculator(MathService mathService) {  
 this.mathService = mathService;  
 }  
 public int addNumbers(int numberA, int numberB){  
 return mathService.add(numberA,numberB);  
 }

}

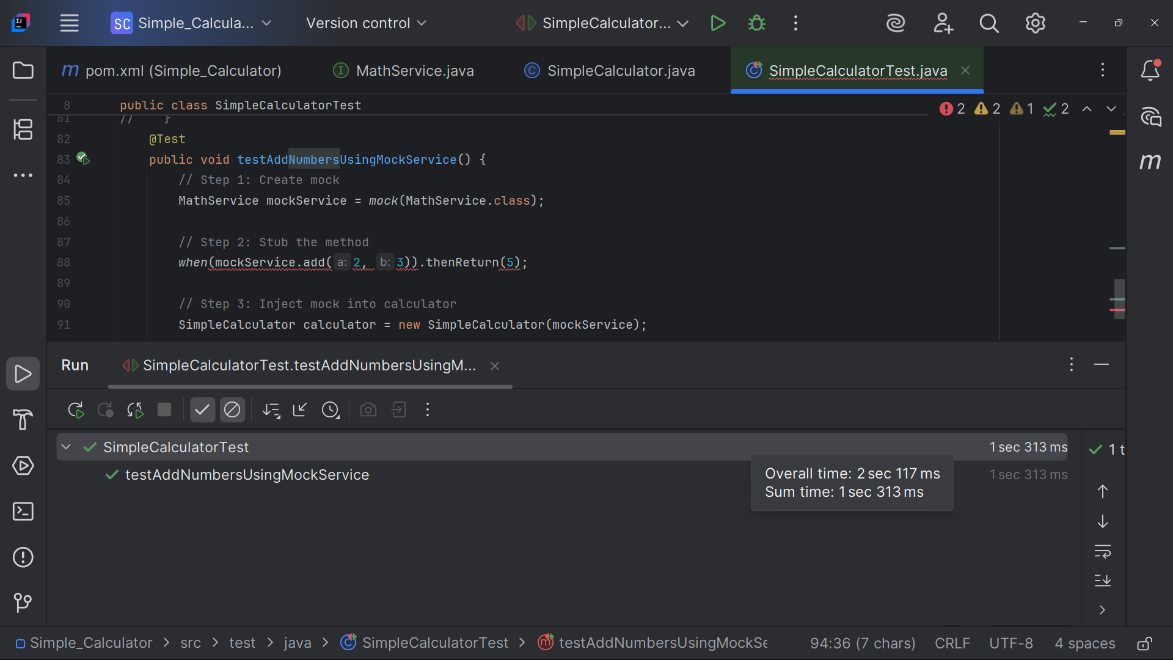
// **SimpleCalculatorTest** test file: where Mockito runs

import static org.junit.Assert.\*;  
import org.junit.Test;  
import static org.mockito.Mockito.\*;  
  
public class SimpleCalculatorTest  
{

@Test  
 public void testAddNumbersUsingMockService() {  
 // Step 1: Create mock  
 MathService mockService = *mock*(MathService.class);  
  
 // Step 2: Stub the method  
 *when*(mockService.add(2, 3)).thenReturn(5);  
  
 // Step 3: Inject mock into calculator  
 SimpleCalculator calculator = new SimpleCalculator(mockService);  
  
 // Step 4: Call and assert  
 int result = calculator.addNumbers(2, 3);  
 *assertEquals*(5, result);}}  
  
**OUTPUT**

****

//running

****

**Exercise 2: Verifying Interactions**

**CODE**

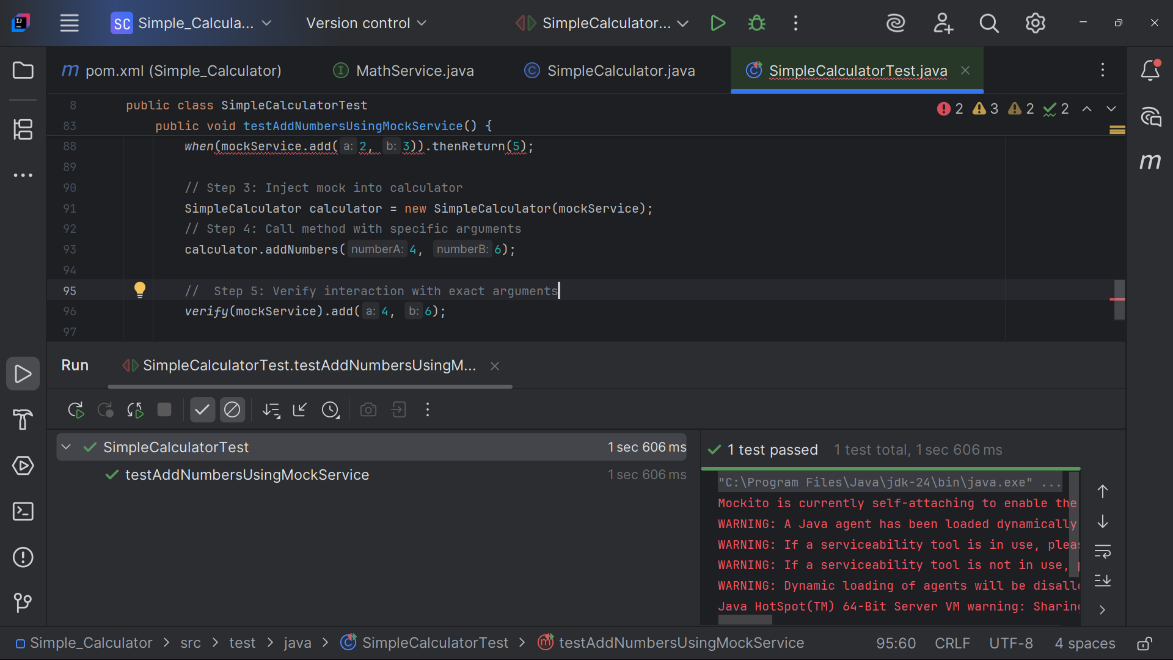
// **SimpleCalculatorTest** test file: where Mockito runs

import static org.junit.Assert.\*;  
import org.junit.Test;  
import static org.mockito.Mockito.\*;  
  
public class SimpleCalculatorTest  
{ @Test  
 public void testAddNumbersUsingMockService() {  
 // Step 1: Create mock  
 MathService mockService = *mock*(MathService.class);  
  
 // Step 2: Stub the method  
 *when*(mockService.add(2, 3)).thenReturn(5);  
  
 // Step 3: Inject mock into calculator  
 SimpleCalculator calculator = new SimpleCalculator(mockService);  
  
 // Step 4: Call method with specific arguments

calculator.addNumbers(4, 6);

// Step 5: Verify interaction with exact arguments  
 *verify*(mockService).add(4, 6);  
 }}

**OUTPUT**



**SL4J Logging Exercises**

**Exercise 1: Logging Error Messages and Warning Levels**

**CODE**

//**external service**

public interface MathService {

int add(int a, int b);

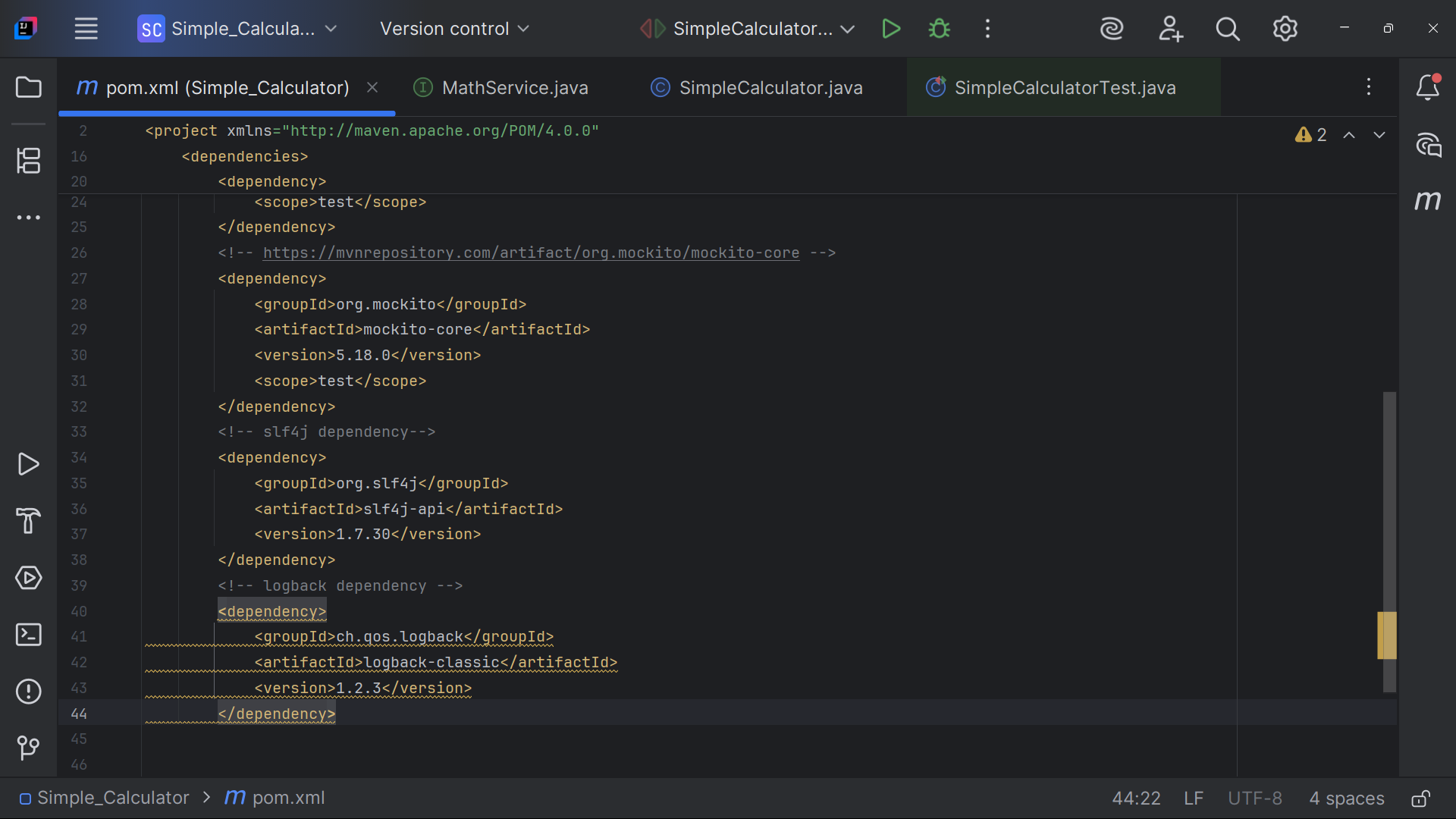
//**SimpleCalculator**

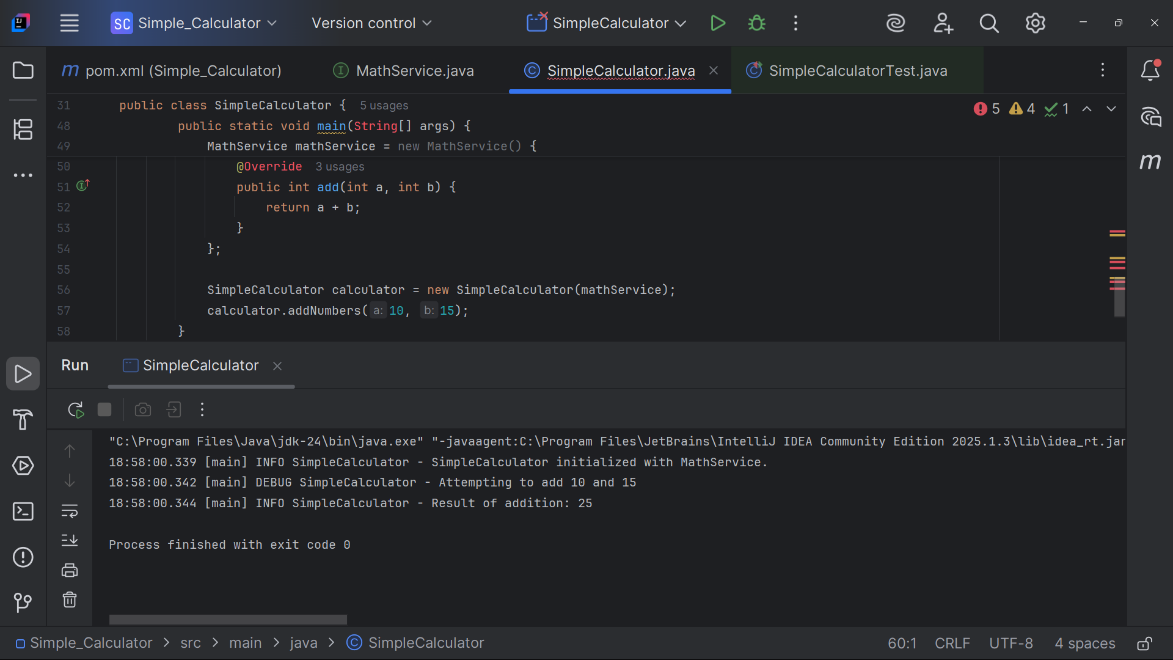
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;

public class SimpleCalculator {  
  
 private static final Logger *logger* = LoggerFactory.getLogger(SimpleCalculator.class);  
 private MathService mathService;  
  
 public SimpleCalculator(MathService mathService) {  
 this.mathService = mathService;  
 *logger*.info("SimpleCalculator initialized with MathService.");  
 }  
  
 public int addNumbers(int a, int b) {  
 *logger*.debug("Attempting to add {} and {}", a, b);  
 int result = mathService.add(a, b);  
 *logger*.info("Result of addition: {}", result);  
 return result;  
 }  
  
 public static void main(String[] args) {  
 MathService mathService = new MathService() {  
 @Override  
 public int add(int a, int b) {  
 return a + b;  
 }  
 };  
  
 SimpleCalculator calculator = new SimpleCalculator(mathService);  
 calculator.addNumbers(10, 15);  
 }  
 }

**OUTPUT**

//putting dependencies



****