**WEEK 2**

PL-SQL

**Exercise 1: Control Structures**

**Scenario 1:** The bank wants to apply a discount to loan interest rates for customers above 60 years old.

**Question:** Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

**Scenario 2:** A customer can be promoted to VIP status based on their balance.

**Question:** Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.

**Scenario 3:** The bank wants to send reminders to customers whose loans are due within the next 30 days.

**Question:** Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

CODE:

SET SERVEROUTPUT ON;

CREATE TABLE Customers (

CustomerID        NUMBER PRIMARY KEY,

  Age               NUMBER,

  LoanInterestRate  NUMBER,

  Balance           NUMBER,

  IsVIP             VARCHAR2(5)

);

CREATE TABLE Loans (

  LoanID      NUMBER PRIMARY KEY,

  CustomerID  NUMBER REFERENCES Customers(CustomerID),

  DueDate     DATE

);

INSERT INTO Customers VALUES (1, 65, 10.5, 15000, 'FALSE');

INSERT INTO Customers VALUES (2, 45, 12.0, 9000, 'FALSE');

INSERT INTO Customers VALUES (3, 70, 11.0, 11000, 'FALSE');

INSERT INTO Loans VALUES (101, 1, SYSDATE + 10);

INSERT INTO Loans VALUES (102, 2, SYSDATE + 35);

INSERT INTO Loans VALUES (103, 3, SYSDATE + 5);

COMMIT;

BEGIN

  DBMS\_OUTPUT.PUT\_LINE('--- Scenario 1: Apply 1% discount for customers above 60 ---');

  FOR cust\_rec IN (

    SELECT CustomerID, LoanInterestRate

    FROM Customers

    WHERE Age > 60

  ) LOOP

    UPDATE Customers

    SET LoanInterestRate = LoanInterestRate - 1

    WHERE CustomerID = cust\_rec.CustomerID;

    DBMS\_OUTPUT.PUT\_LINE('Customer ID ' || cust\_rec.CustomerID ||

                         ' interest rate reduced to ' || (cust\_rec.LoanInterestRate - 1));

  END LOOP;

  DBMS\_OUTPUT.PUT\_LINE('--- Scenario 2: Promote customers to VIP if Balance > 10000 ---');

  FOR vip\_rec IN (

    SELECT CustomerID, Balance

    FROM Customers

    WHERE Balance > 10000

  ) LOOP

    UPDATE Customers

    SET IsVIP = 'TRUE'

    WHERE CustomerID = vip\_rec.CustomerID;

    DBMS\_OUTPUT.PUT\_LINE('Customer ID ' || vip\_rec.CustomerID ||

                         ' promoted to VIP (Balance: ' || vip\_rec.Balance || ')');

  END LOOP;

  DBMS\_OUTPUT.PUT\_LINE('--- Scenario 3: Loan due reminders (next 30 days) ---');

  FOR loan\_rec IN (

    SELECT LoanID, CustomerID, DueDate

    FROM Loans

    WHERE DueDate BETWEEN SYSDATE AND SYSDATE + 30

  ) LOOP

    DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || loan\_rec.LoanID ||

                         ' for Customer ID ' || loan\_rec.CustomerID ||

                         ' is due on ' || TO\_CHAR(loan\_rec.DueDate, 'DD-MON-YYYY'));

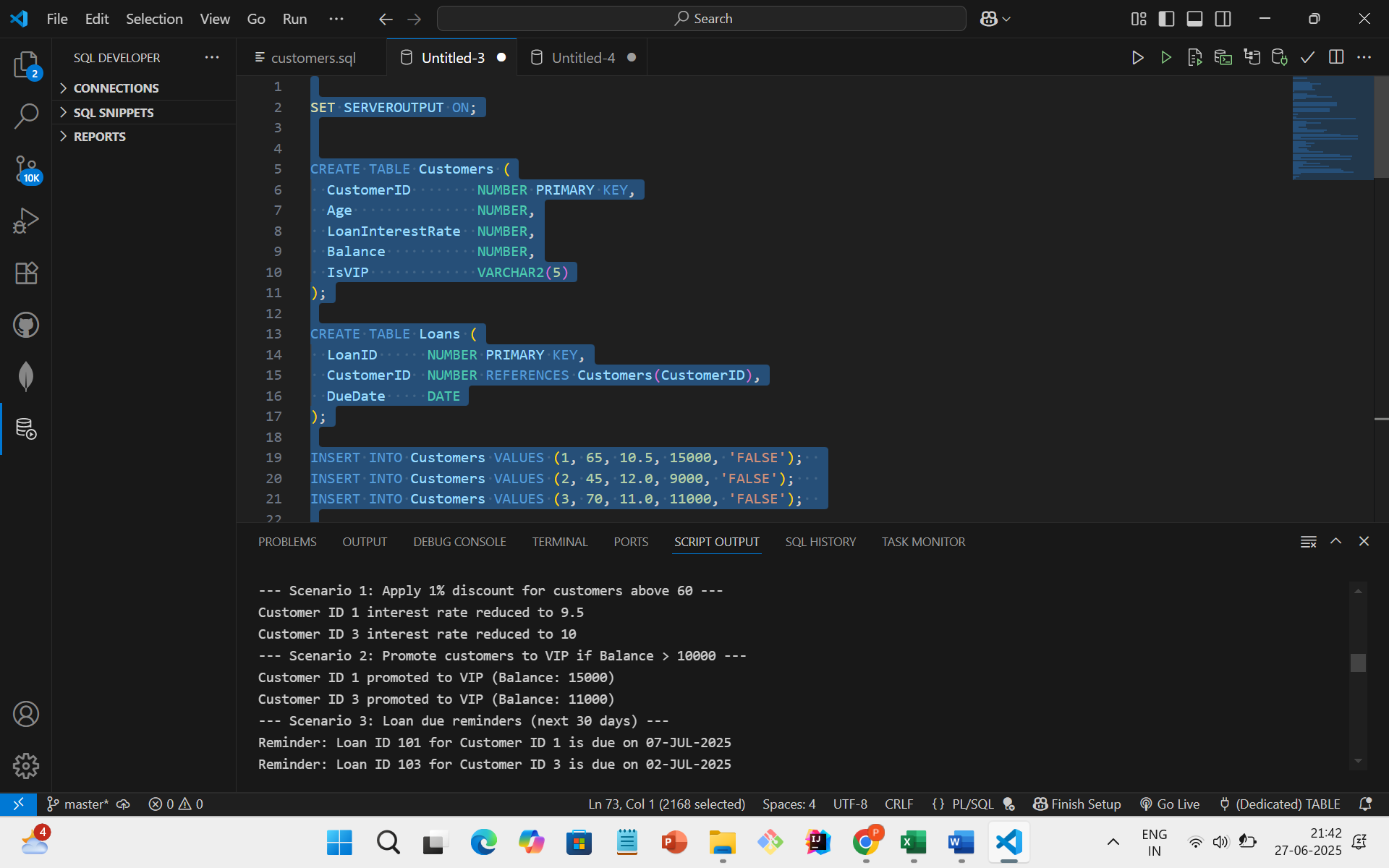
  END LOOP;

  COMMIT;

END;

/

OUTPUT:



**Exercise 3: Stored Procedures**

**Scenario 1:** The bank needs to process monthly interest for all savings accounts.

**Question:** Write a stored procedure **ProcessMonthlyInterest** that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

**Scenario 2:** The bank wants to implement a bonus scheme for employees based on their performance.

**Question:** Write a stored procedure **UpdateEmployeeBonus** that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

**Scenario 3:** Customers should be able to transfer funds between their accounts.

**Question:** Write a stored procedure **TransferFunds** that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer

CODE:

SET SERVEROUTPUT ON;

CREATE TABLE SavingsAccounts (

  AccountID    NUMBER PRIMARY KEY,

  CustomerID   NUMBER,

  Balance      NUMBER

);

CREATE TABLE Employees (

  EmployeeID   NUMBER PRIMARY KEY,

  DeptID       NUMBER,

  Salary       NUMBER

);

CREATE TABLE Accounts (

  AccountID    NUMBER PRIMARY KEY,

  CustomerID   NUMBER,

  Balance      NUMBER

);

INSERT INTO SavingsAccounts VALUES (101, 1, 10000);

INSERT INTO SavingsAccounts VALUES (102, 2, 5000);

INSERT INTO Employees VALUES (1, 10, 30000);

INSERT INTO Employees VALUES (2, 10, 35000);

INSERT INTO Employees VALUES (3, 20, 40000);

INSERT INTO Accounts VALUES (201, 1, 15000);

INSERT INTO Accounts VALUES (202, 2, 5000);

COMMIT;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

  FOR acc\_rec IN (

    SELECT AccountID, Balance

    FROM SavingsAccounts

  ) LOOP

    UPDATE SavingsAccounts

    SET Balance = Balance + (Balance \* 0.01)

    WHERE AccountID = acc\_rec.AccountID;

    DBMS\_OUTPUT.PUT\_LINE('Savings Account ID ' || acc\_rec.AccountID ||

                         ': Interest applied. New Balance = ' || TO\_CHAR(acc\_rec.Balance \* 1.01));

  END LOOP;

  COMMIT;

END;

/

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

  p\_DepartmentID IN NUMBER,

  p\_BonusPercent IN NUMBER

) IS

BEGIN

  FOR emp\_rec IN (

    SELECT EmployeeID, Salary

    FROM Employees

    WHERE DeptID = p\_DepartmentID

  ) LOOP

    UPDATE Employees

    SET Salary = Salary + (Salary \* (p\_BonusPercent / 100))

    WHERE EmployeeID = emp\_rec.EmployeeID;

    DBMS\_OUTPUT.PUT\_LINE('Employee ID ' || emp\_rec.EmployeeID ||

                         ': Salary updated to ' || TO\_CHAR(emp\_rec.Salary \* (1 + p\_BonusPercent / 100)));

  END LOOP;

  COMMIT;

END;

/

CREATE OR REPLACE PROCEDURE TransferFunds (

  p\_FromAccountID IN NUMBER,

  p\_ToAccountID   IN NUMBER,

  p\_Amount        IN NUMBER

) IS

  v\_FromBalance NUMBER;

BEGIN

  SELECT Balance INTO v\_FromBalance

  FROM Accounts

  WHERE AccountID = p\_FromAccountID

  FOR UPDATE;

  IF v\_FromBalance < p\_Amount THEN

    RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds in source account.');

  END IF;

  UPDATE Accounts

  SET Balance = Balance - p\_Amount

  WHERE AccountID = p\_FromAccountID;

  UPDATE Accounts

  SET Balance = Balance + p\_Amount

  WHERE AccountID = p\_ToAccountID;

  DBMS\_OUTPUT.PUT\_LINE('Transferred ' || p\_Amount ||

                       ' from Account ID ' || p\_FromAccountID ||

                       ' to Account ID ' || p\_ToAccountID);

  COMMIT;

END;

/

BEGIN

  DBMS\_OUTPUT.PUT\_LINE('--- Running: ProcessMonthlyInterest ---');

  ProcessMonthlyInterest;

  DBMS\_OUTPUT.PUT\_LINE('--- Running: UpdateEmployeeBonus (Dept 10, 10%) ---');

  UpdateEmployeeBonus(10, 10);

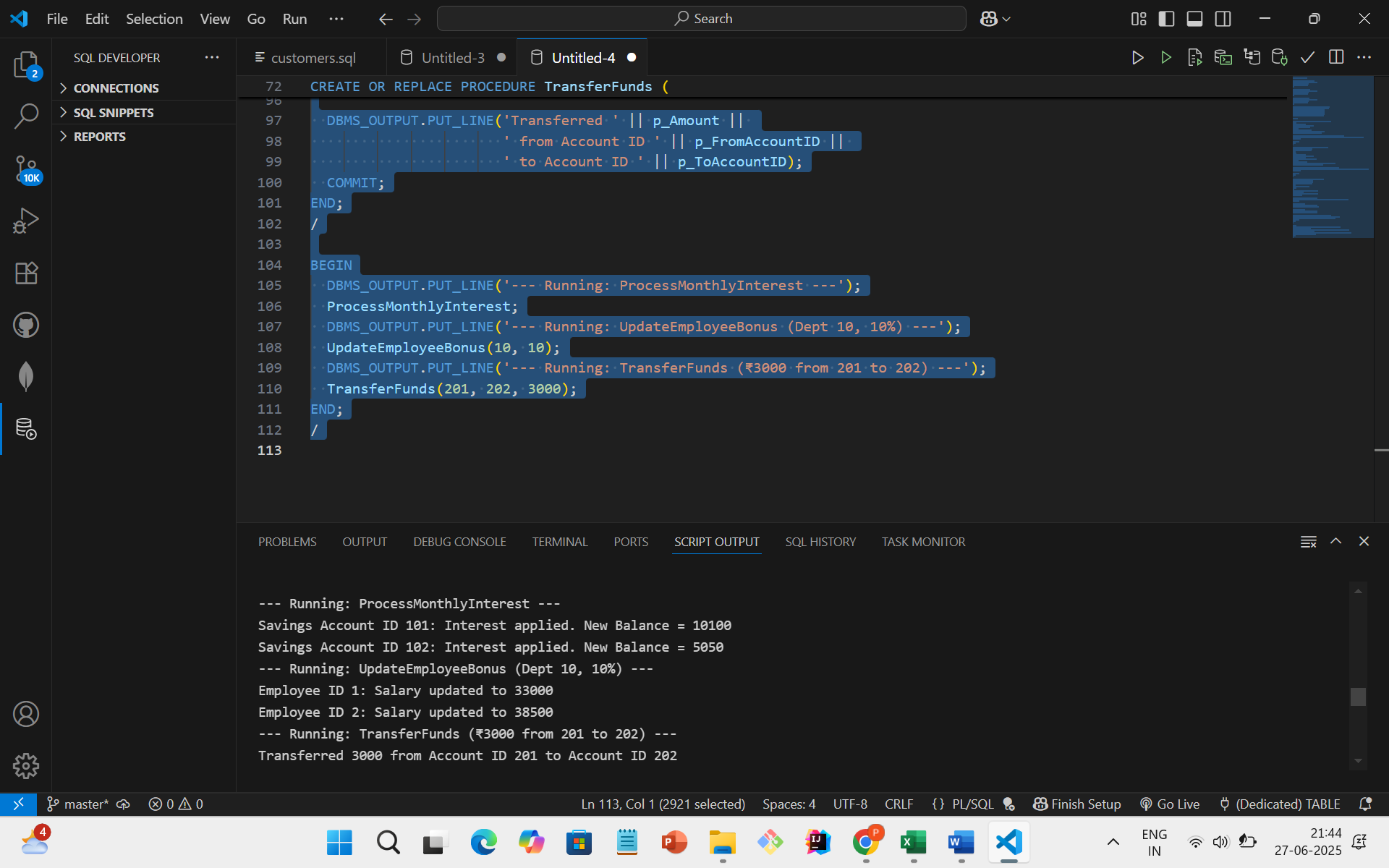
  DBMS\_OUTPUT.PUT\_LINE('--- Running: TransferFunds (₹3000 from 201 to 202) ---');

  TransferFunds(201, 202, 3000);

END;

/

OUTPUT:



**JUnit Testing Exercises**

**Exercise 1: Setting Up JUnit**

Scenario:

You need to set up JUnit in your Java project to start writing unit tests.

Steps:

1. Create a new Java project in your IDE (e.g., IntelliJ IDEA, Eclipse).
2. Add JUnit dependency to your project. If you are using Maven, add the following to your pom.xml:

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

CODE:

Calculator.java

**package** com.example;

**public** **class** Calculator {

// Addition

**public** **int** add(**int** a, **int** b) {

**return** a + b;

}

// Subtraction

**public** **int** subtract(**int** a, **int** b) {

**return** a - b;

}

// Multiplication

**public** **int** multiply(**int** a, **int** b) {

**return** a \* b;

}

// Division (integer division)

**public** **int** divide(**int** a, **int** b) {

**if** (b == 0) {

**throw** **new** ArithmeticException("Cannot divide by zero");

}

**return** a / b;

}

}

CalculatorTest.java:

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

Calculator calc = new Calculator();

@Test

public void testAdd() {

int result = calc.add(2, 3);

System.out.println("Addition Result: " + result);

assertEquals(5, result);

}

@Test

public void testSubtract() {

int result = calc.subtract(5, 4);

System.out.println("Subtraction Result: " + result);

assertEquals(1, result);

}

@Test

public void testMultiply() {

int result = calc.multiply(3, 4);

System.out.println("Multiplication Result: " + result);

assertEquals(12, result);

}

@Test

public void testDivide() {

int result = calc.divide(10, 5);

System.out.println("Division Result: " + result);

assertEquals(2, result);

}

@Test(expected = ArithmeticException.class)

public void testDivideByZero() {

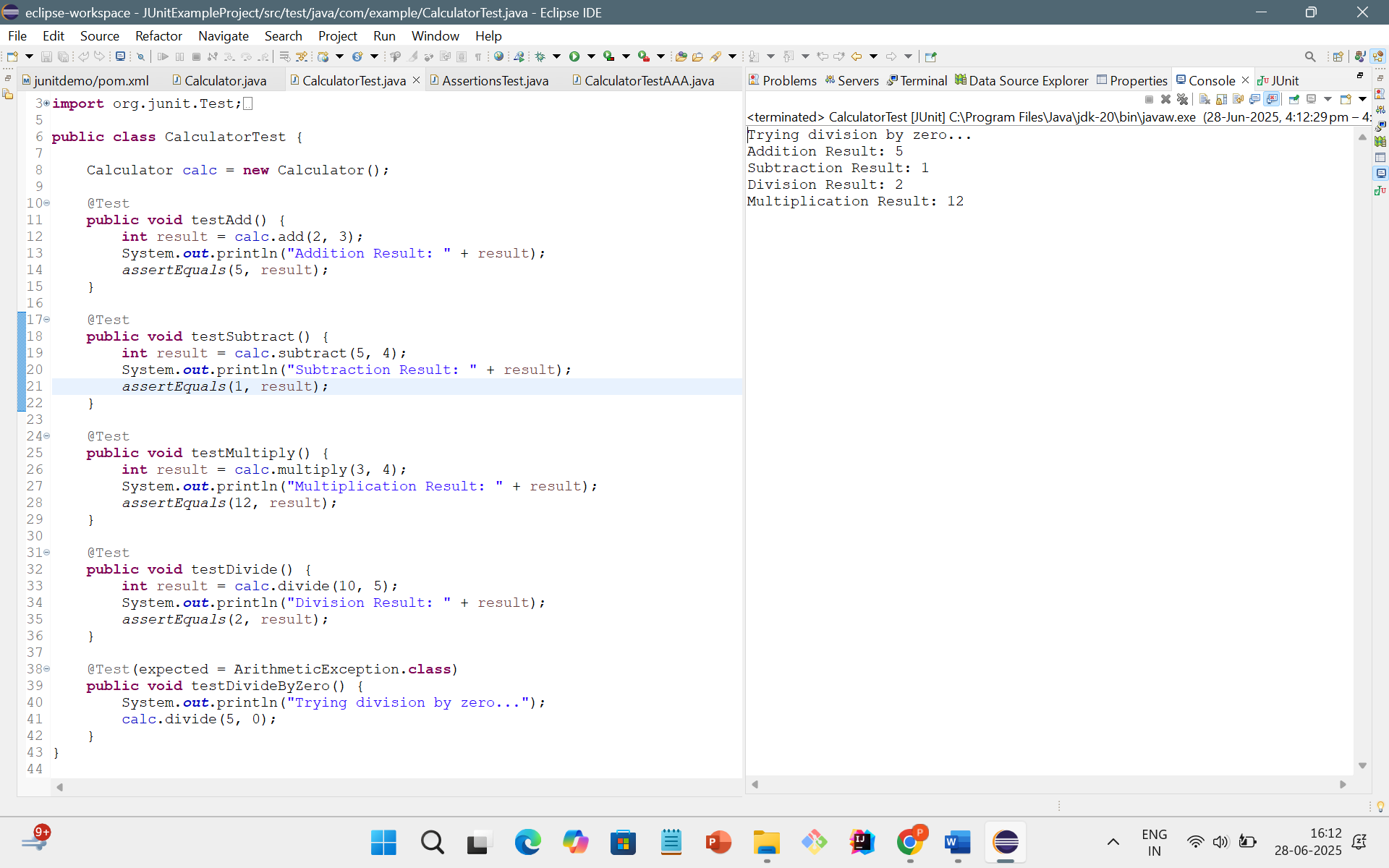
System.out.println("Trying division by zero...");

calc.divide(5, 0);

}

}

OUTPUT:



**Exercise 3: Assertions in JUnit**

Scenario:

You need to use different assertions in JUnit to validate your test results.

Steps:

1. Write tests using various JUnit assertions.

Solution Code:

public class AssertionsTest {

@Test public void testAssertions() {

// Assert equals assertEquals(5, 2 + 3);

// Assert true assertTrue(5 > 3);

// Assert false assertFalse(5 < 3);

// Assert null assertNull(null);

// Assert not null

assertNotNull(new Object());

} }

CODE:

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

@Test

public void testAssertions() {

// Assert equals

int sum = 2 + 3;

assertEquals(5, sum);

System.out.println("assertEquals passed: 2 + 3 = " + sum);

boolean conditionTrue = 5 > 3;

assertTrue(conditionTrue);

System.out.println("assertTrue passed: 5 > 3 = " + conditionTrue);

boolean conditionFalse = 5 < 3;

assertFalse(conditionFalse);

System.out.println("assertFalse passed: 5 < 3 = " + conditionFalse);

Object obj1 = null;

assertNull(obj1);

System.out.println("assertNull passed: obj1 = " + obj1);

// Assert not null

Object obj2 = new Object();

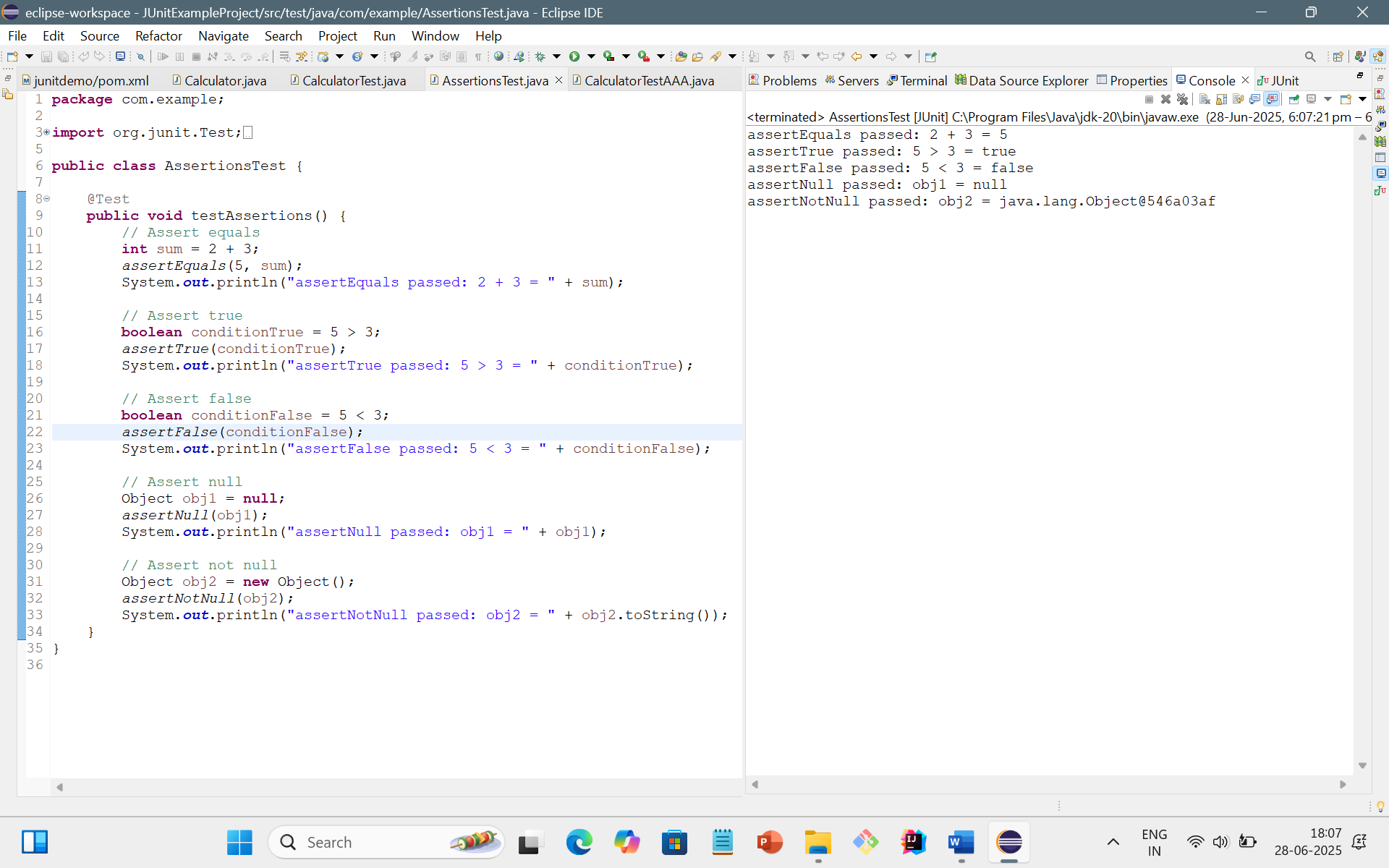
assertNotNull(obj2);

System.out.println("assertNotNull passed: obj2 = " + obj2.toString());

}

}

OUTPUT:



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

**Teardown Methods in JUnit**

Scenario:

You need to organize your tests using the Arrange-Act-Assert (AAA) pattern and use setup and teardown methods.

Steps:

1. Write tests using the AAA pattern.
2. Use @Before and @After annotations for setup and teardown methods.

CODE:

package com.example;

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*

public class CalculatorTestAAA {

private Calculator calculator;

@Before

public void setUp() {

calculator = new Calculator();

System.out.println("setUp: Calculator instance created");

}

@After

public void tearDown() {

System.out.println("tearDown: Test completed\n");

}

@Test

public void testAddition() {

int a = 10;

int b = 5;

int result = calculator.add(a, b);

assertEquals(15, result);

System.out.println("testAddition: " + a + " + " + b + " = " + result);

}

@Test

public void testSubtraction() {

int a = 8;

int b = 3;

int result = calculator.subtract(a, b);

assertEquals(5, result);

System.out.println("testSubtraction: " + a + " - " + b + " = " + result);

}

@Test(expected = ArithmeticException.class)

public void testDivideByZero() {

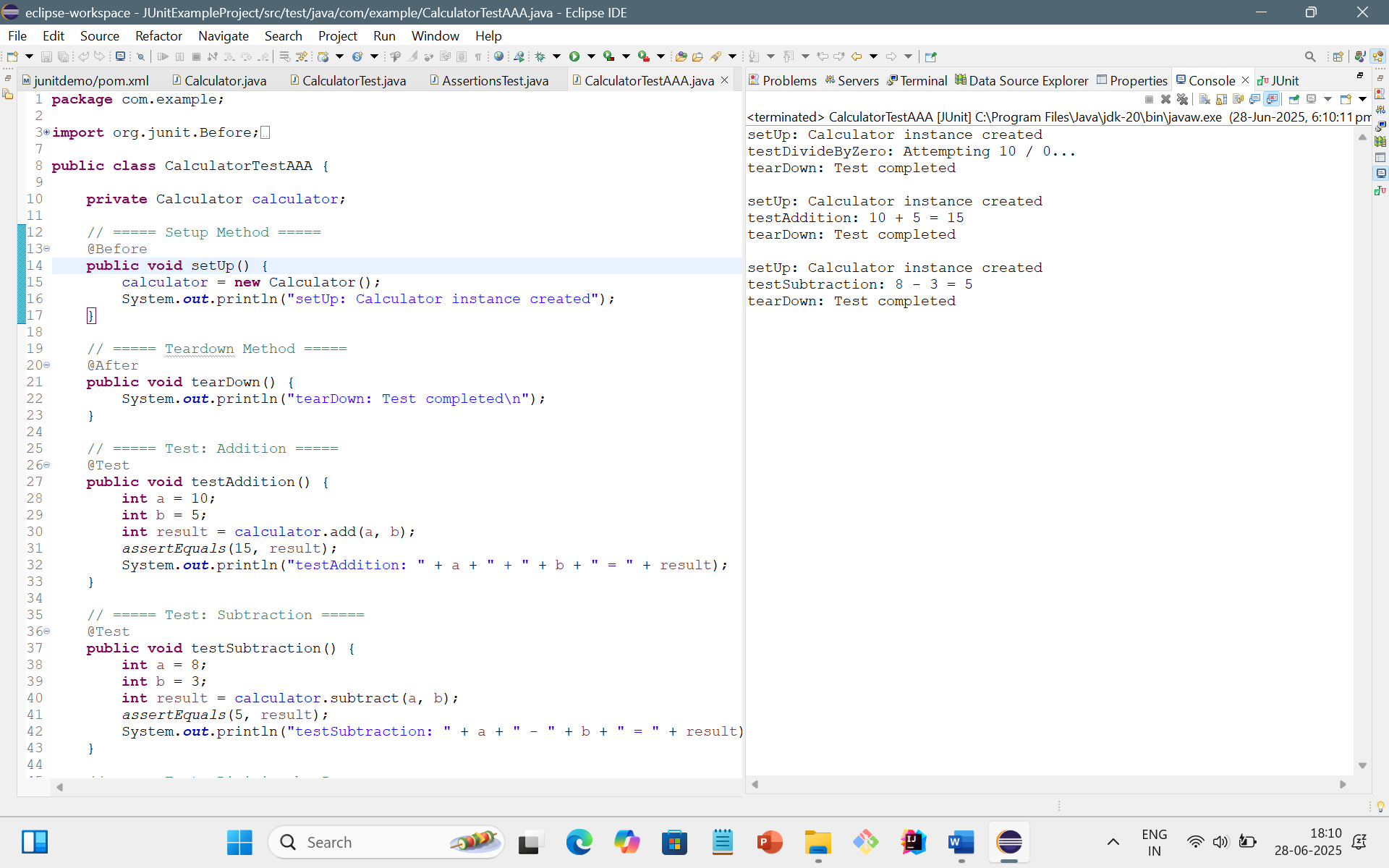
System.out.println("testDivideByZero: Attempting 10 / 0...");

calculator.divide(10, 0);

}

}

OUTPUT:



**Mockito Hands-On Exercises**

**Exercise 1: Mocking and Stubbing**

Scenario:

You need to test a service that depends on an external API. Use Mockito to mock the external API and stub its methods.

Steps:

1. Create a mock object for the external API.
2. Stub the methods to return predefined values.
3. Write a test case that uses the mock object.

Solution Code:

import static org.mockito.Mockito.\*; import org.junit.jupiter.api.Test; import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testExternalApi() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class); when(mockApi.getData()).thenReturn("Mock Data");

MyService service = new MyService(mockApi); String result = service.fetchData(); assertEquals("Mock Data", result);

}

}

CODE:

ExternalApi.java:

**public** **interface** ExternalApi {

String getUserName(**int** userId);

**int** getUserAge(**int** userId);

String getData();

}

MyService.java:

**public** **class** MyService {

**private** ExternalApi api;

**public** MyService(ExternalApi api) {

**this**.api = api;

}

**public** String fetchData() {

**return** api.getData();

}

**public** String getUserSummary(**int** userId) {

String name = api.getUserName(userId);

**int** age = api.getUserAge(userId);

**return** "User: " + name + ", Age: " + age;

}

}

MyServiceTest.java:

**import** **static** org.mockito.Mockito.\*;

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

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

**public** **class** MyServiceTest {

@Test

**public** **void** testGetUserSummary() {

// Create mock for ExternalApi

ExternalApi mockApi = *mock*(ExternalApi.**class**); *when*(mockApi.getData()).thenReturn("Mock Data");

*when*(mockApi.getUserName(101)).thenReturn("Alice");

*when*(mockApi.getUserAge(101)).thenReturn(25);

MyService service = **new** MyService(mockApi);

String summary = service.getUserSummary(101);

String result = service.fetchData();

System.***out***.println("Fetched from mocked API: " + result);

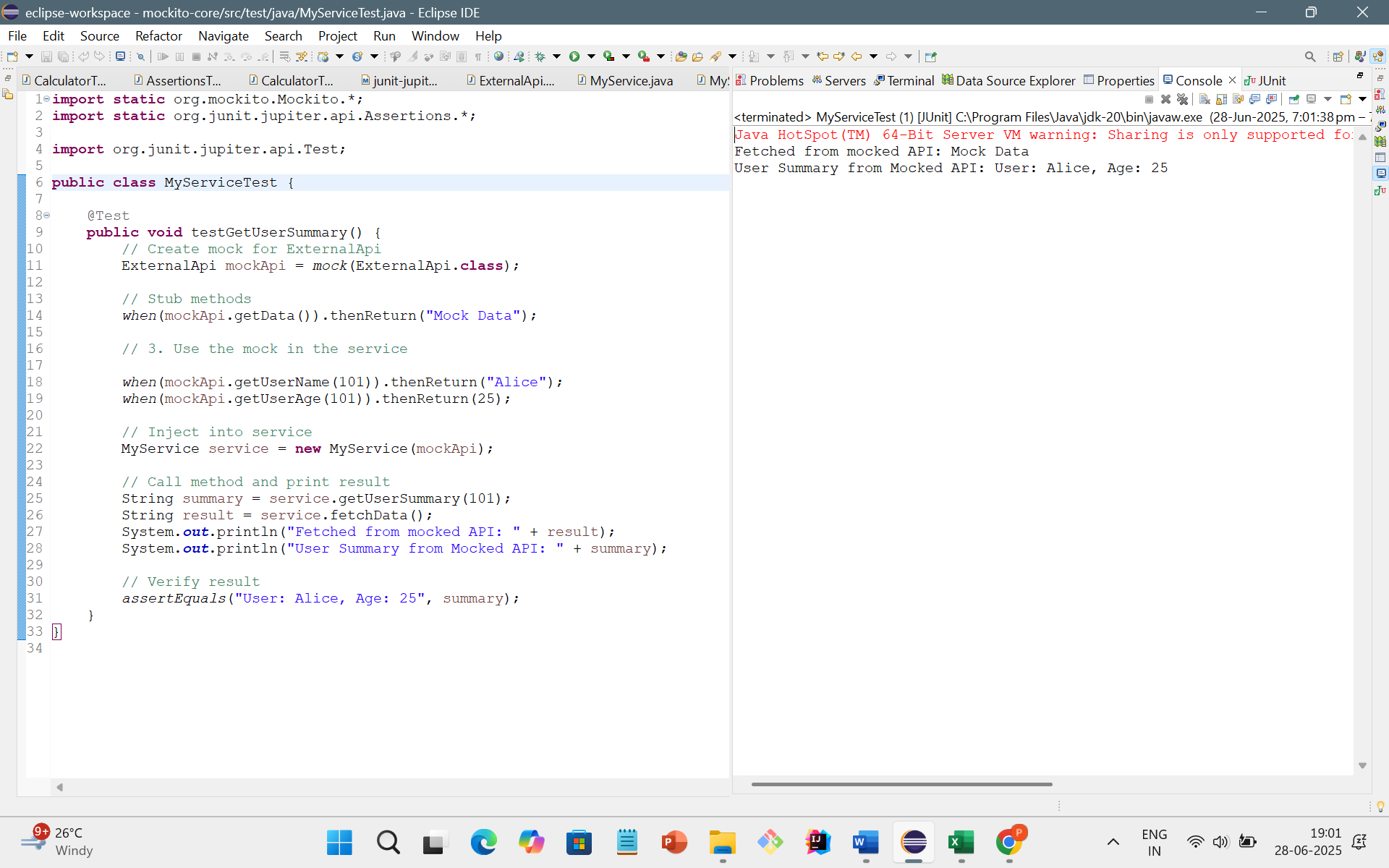
System.***out***.println("User Summary from Mocked API: " + summary);

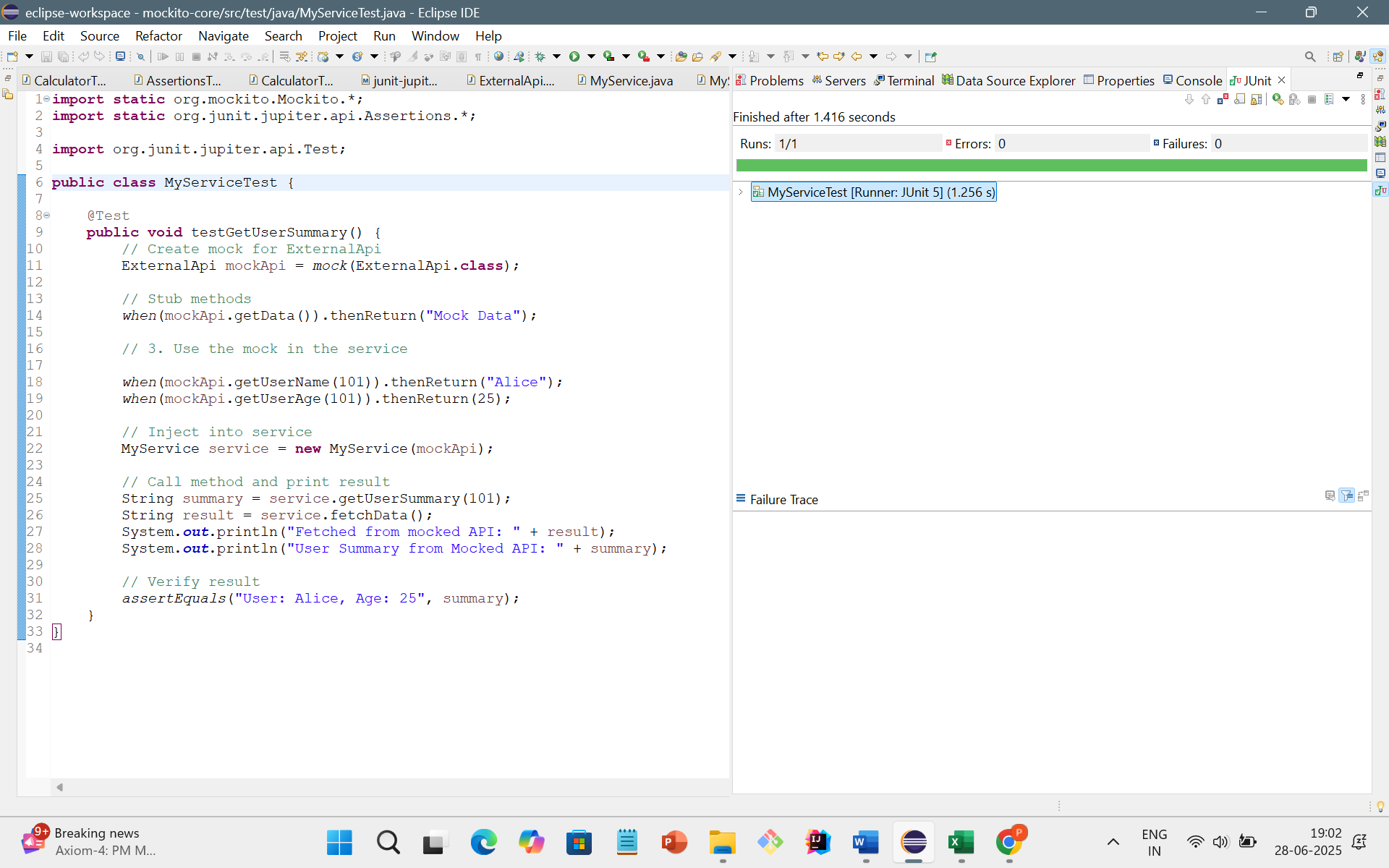
*assertEquals*("User: Alice, Age: 25", summary);

}

}

OUTPUT:





**Exercise 2: Verifying Interactions**

Scenario:

You need to ensure that a method is called with specific arguments.

Steps:

1. Create a mock object.
2. Call the method with specific arguments.
3. Verify the interaction.

Solution Code:

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test; import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class); MyService service = new MyService(mockApi); service.fetchData(); verify(mockApi).getData();

}

}

CODE:

ExternalApi.java:

**public** **interface** ExternalApi {

**void** getData(); // No return type for this example

}

MyService.java:

**public** **class** MyService {

**private** ExternalApi api;

**public** MyService(ExternalApi api) {

**this**.api = api;

}

**public** **void** fetchData() {

System.***out***.println("Inside MyService.fetchData()");

api.getData();

}

}

MyServiceTest.java:

**import** **static** org.mockito.Mockito.\*;

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

**public** **class** MyServiceTest {

@Test

**public** **void** testVerifyInteraction() {

ExternalApi mockApi = *mock*(ExternalApi.**class**);

MyService service = **new** MyService(mockApi);

service.fetchData();

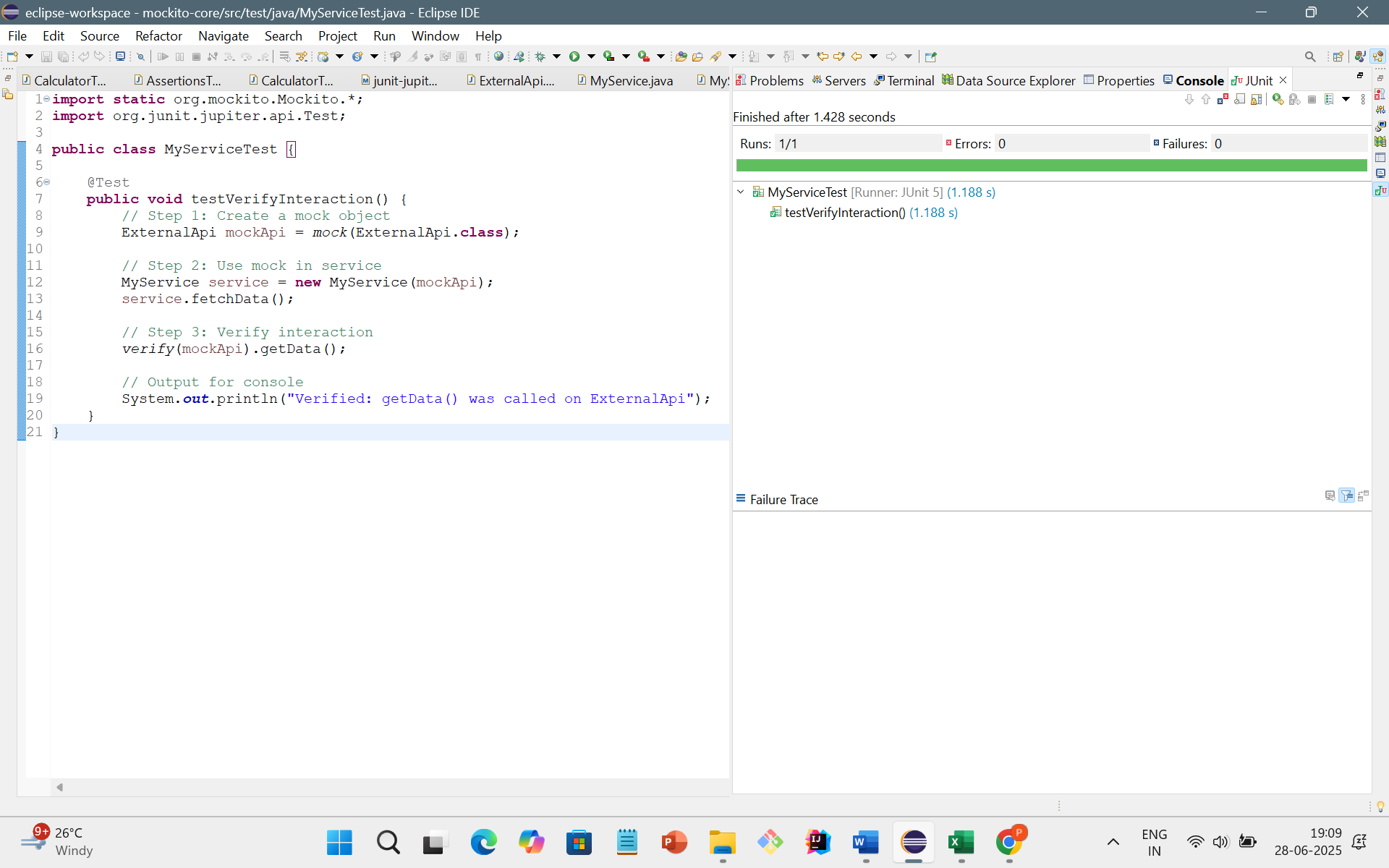
*verify*(mockApi).getData();

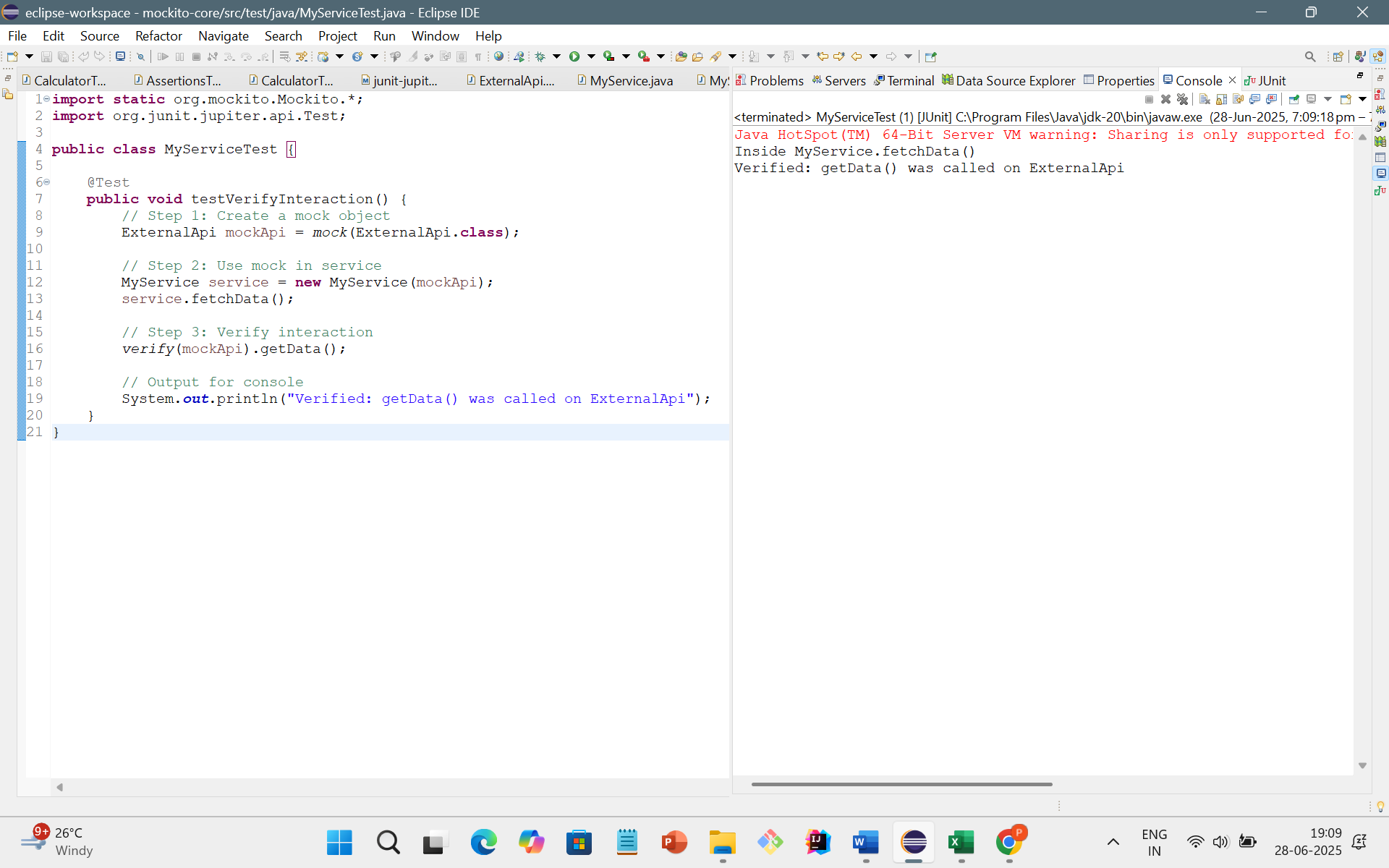
System.***out***.println("Verified: getData() was called on ExternalApi");

}

}

OUTPUT:





**Logging using SLF4J**

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

Task: Write a Java application that demonstrates logging error messages and warning levels using SLF4J.

Step-by-Step Solution:

1. Add SLF4J and Logback dependencies to your `pom.xml` file:

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.30</version>

</dependency>

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.2.3</version> </dependency>

1. Create a Java class that uses SLF4J for logging:

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

public class LoggingExample { private static final Logger logger = LoggerFactory.getLogger(LoggingExample.class);

public static void main(String[] args) { logger.error("This is an error message"); logger.warn("This is a warning message");

} }

CODE:

LoggingExample.java:

**package** com.example;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**public** **class** LoggingExample {

**private** **static** **final** Logger ***logger*** = LoggerFactory.*getLogger*(LoggingExample.**class**);

**public** **static** **void** main(String[] args) {

***logger***.error("This is an error message");

***logger***.warn("This is a warning message");

}

}

OUTPUT:

