**WEEK 2**

**PL/SQL programming & JUnit, Mockito and SL4J**

**Name: Tarun Tadepalli**

**Superset ID:- 4991850**

**1: Control Structures**

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

* 1. 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.

* 1. 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.

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

**Bank Database:**

Create database bank;

Use bank;

**Create tables:**

CREATE TABLE customers (

id INT PRIMARY KEY,

name VARCHAR(100),

age INT,

balance DECIMAL(10, 2),

isvip BOOLEAN DEFAULT FALSE

);

CREATE TABLE loans (

id INT PRIMARY KEY,

customer\_id INT,

interest\_rate DECIMAL(5, 2),

due\_date DATE,

FOREIGN KEY (customer\_id) REFERENCES customers(id)

);

**Insert Values:**  
INSERT INTO customers VALUES

(1, 'Ravi', 65, 12000.00, FALSE),

(2, 'Anita', 45, 8000.00, FALSE),

(3, 'Sunil', 70, 15000.00, FALSE),

(4, 'Geeta', 30, 9500.00, FALSE);

INSERT INTO loans VALUES

(101, 1, 10.5, CURDATE() + INTERVAL 10 DAY),

(102, 2, 11.0, CURDATE() + INTERVAL 40 DAY),

(103, 3, 9.5, CURDATE() + INTERVAL 5 DAY),

(104, 4, 10.0, CURDATE() + INTERVAL 20 DAY);

**Create Procedures:**

apply\_interest\_discount:

Apply 1% Interest Discount for Age > 60:

CREATE PROCEDURE apply\_interest\_discount()

BEGIN

UPDATE loans

SET interest\_rate = interest\_rate - 1

WHERE customer\_id IN (

SELECT id FROM customers WHERE age > 60

);

END;

update\_vip\_status:

Set isvip = TRUE for balance > 10,000:

CREATE PROCEDURE update\_vip\_status()

BEGIN

UPDATE customers

SET isvip = TRUE

WHERE balance > 10000;

END;

show\_due\_loan\_reminders:

Print Loan Reminders (within 30 days)

CREATE PROCEDURE show\_due\_loan\_reminders()

BEGIN

SELECT

c.name AS customer\_name,

l.id AS loan\_id,

l.due\_date,

CONCAT('Reminder: Loan #', l.id, ' for ', c.name, ' is due on ', DATE\_FORMAT(l.due\_date, '%d-%b-%Y')) AS message

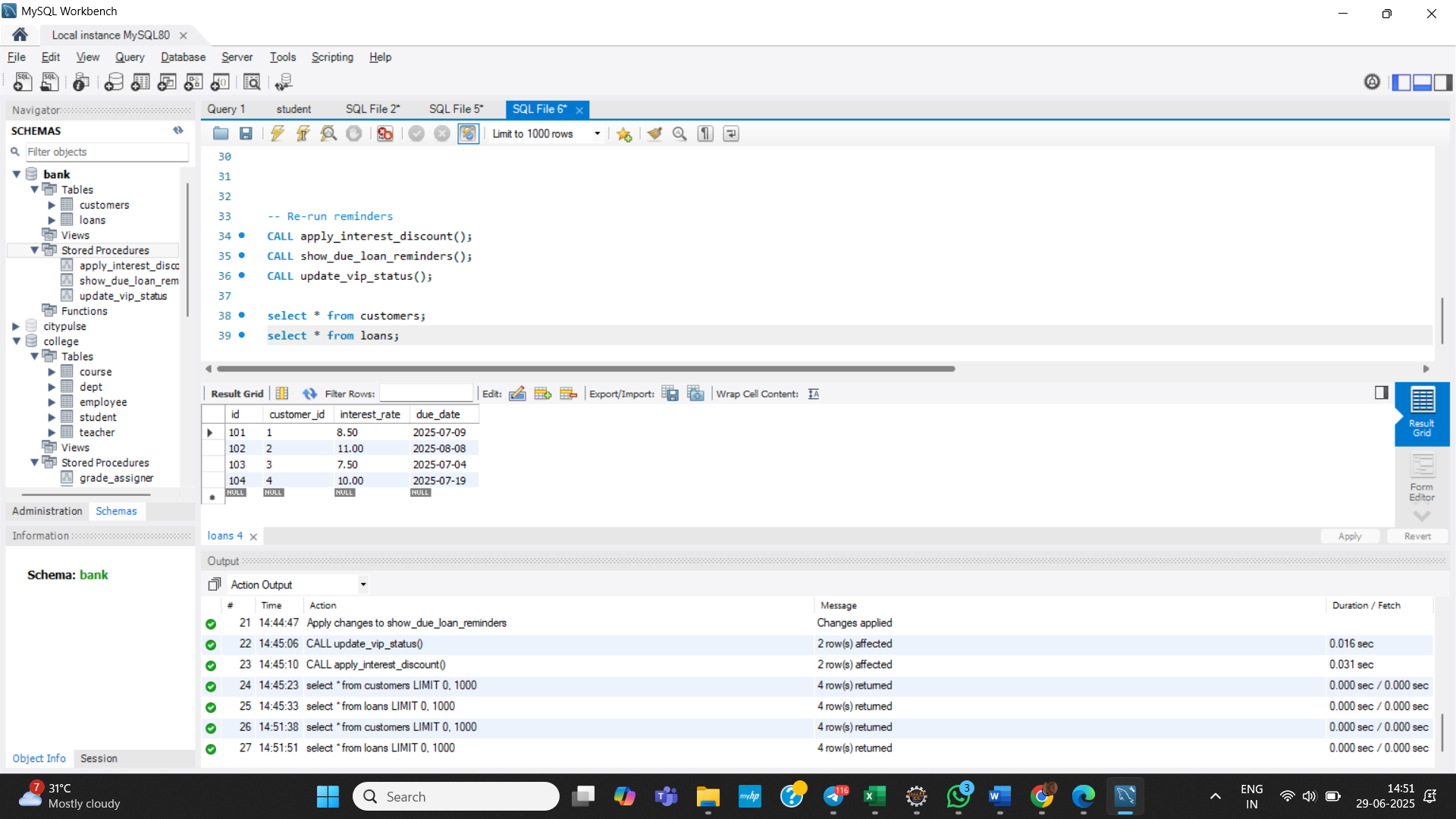
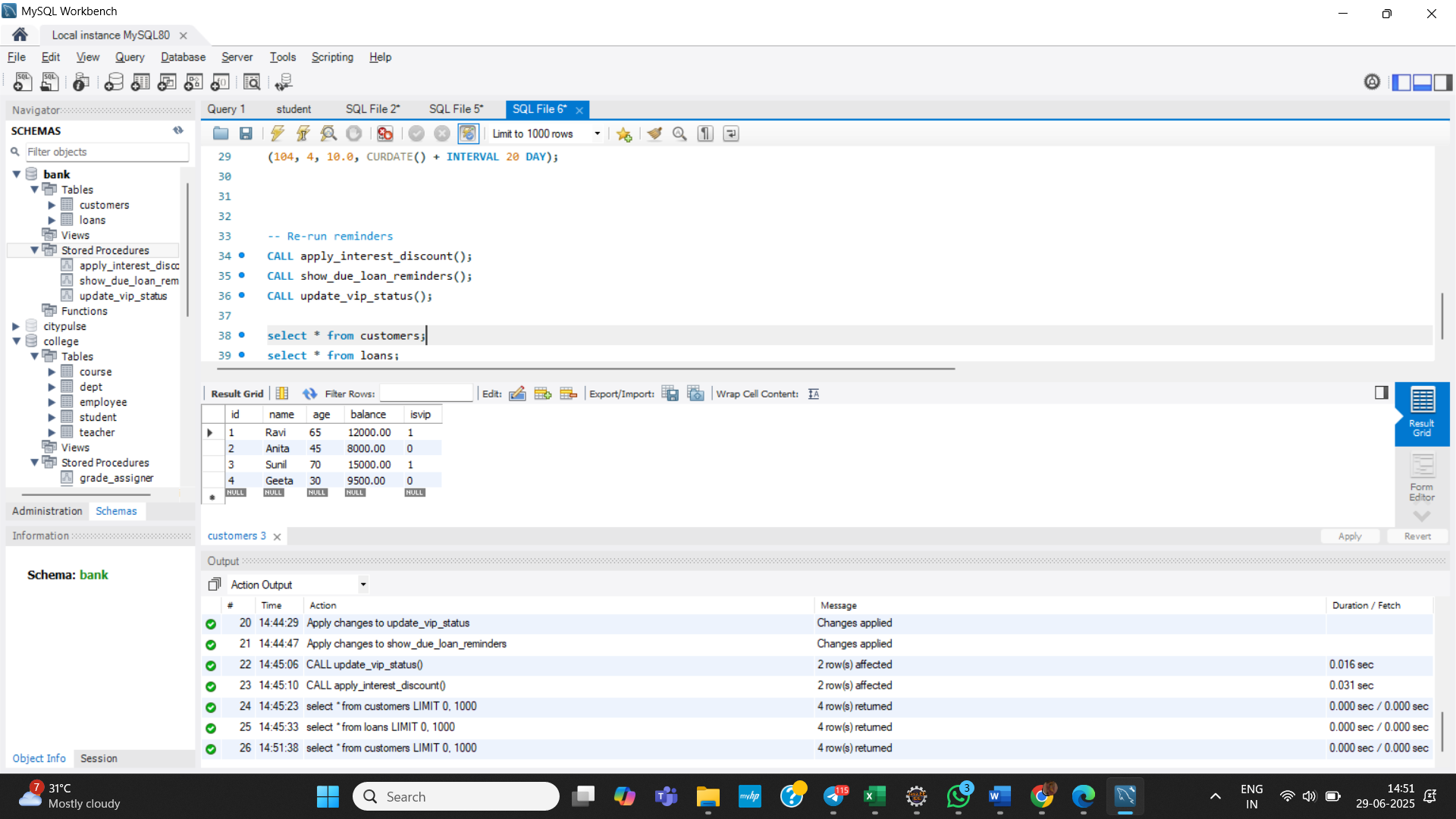
FROM loans l

JOIN customers c ON l.customer\_id = c.id

WHERE l.due\_date BETWEEN CURDATE() AND CURDATE() + INTERVAL 30 DAY;

END;

**Output:**



**2. Stored Procedures**

**Monthly Interest for Savings Accounts:**

**Create schema:**

CREATE TABLE savings\_accounts (

account\_id INT PRIMARY KEY,

customer\_id INT,

balance DECIMAL(10,2)

);

INSERT INTO savings\_accounts VALUES

(1, 101, 10000.00),

(2, 102, 5000.00),

(3, 103, 20000.00);

**ProcessMonthlyInterest**

CREATE PROCEDURE ProcessMonthlyInterest ()

BEGIN

UPDATE savings\_accounts

SET balance = balance + (balance \* 0.01);

END;

**Bonus for Employees Based on Department:**

**Create Schema:**

CREATE TABLE employees (

emp\_id INT PRIMARY KEY,

name VARCHAR(100),

department VARCHAR(50),

salary DECIMAL(10,2)

);

INSERT INTO employees VALUES

(1, 'Ravi', 'IT', 50000.00),

(2, 'Anita', 'HR', 45000.00),

(3, 'Sunil', 'IT', 55000.00),

(4, 'Geeta', 'Sales', 40000.00);

**UpdateEmployeeBonus:**

CREATE PROCEDURE UpdateEmployeeBonus(IN dept\_name VARCHAR(50), IN bonus\_percent DECIMAL(5,2))

BEGIN

UPDATE employees

SET salary = salary + (salary \* bonus\_percent / 100)

WHERE department = dept\_name;

END;

**Transfer Funds Between Accounts:**

**Create Schema:**

CREATE TABLE accounts (

acc\_id INT PRIMARY KEY,

customer\_id INT,

balance DECIMAL(10,2)

);

INSERT INTO accounts VALUES

(1, 201, 15000.00),

(2, 201, 3000.00),

(3, 202, 8000.00);

**TransferFunds:**

CREATE PROCEDURE TransferFunds(

IN source\_acc INT,

IN target\_acc INT,

IN transfer\_amt DECIMAL(10,2)

)

BEGIN

DECLARE current\_balance DECIMAL(10,2);

-- Check balance

SELECT balance INTO current\_balance

FROM accounts

WHERE acc\_id = source\_acc;

IF current\_balance >= transfer\_amt THEN

UPDATE accounts

SET balance = balance - transfer\_amt

WHERE acc\_id = source\_acc;

UPDATE accounts

SET balance = balance + transfer\_amt

WHERE acc\_id = target\_acc;

ELSE

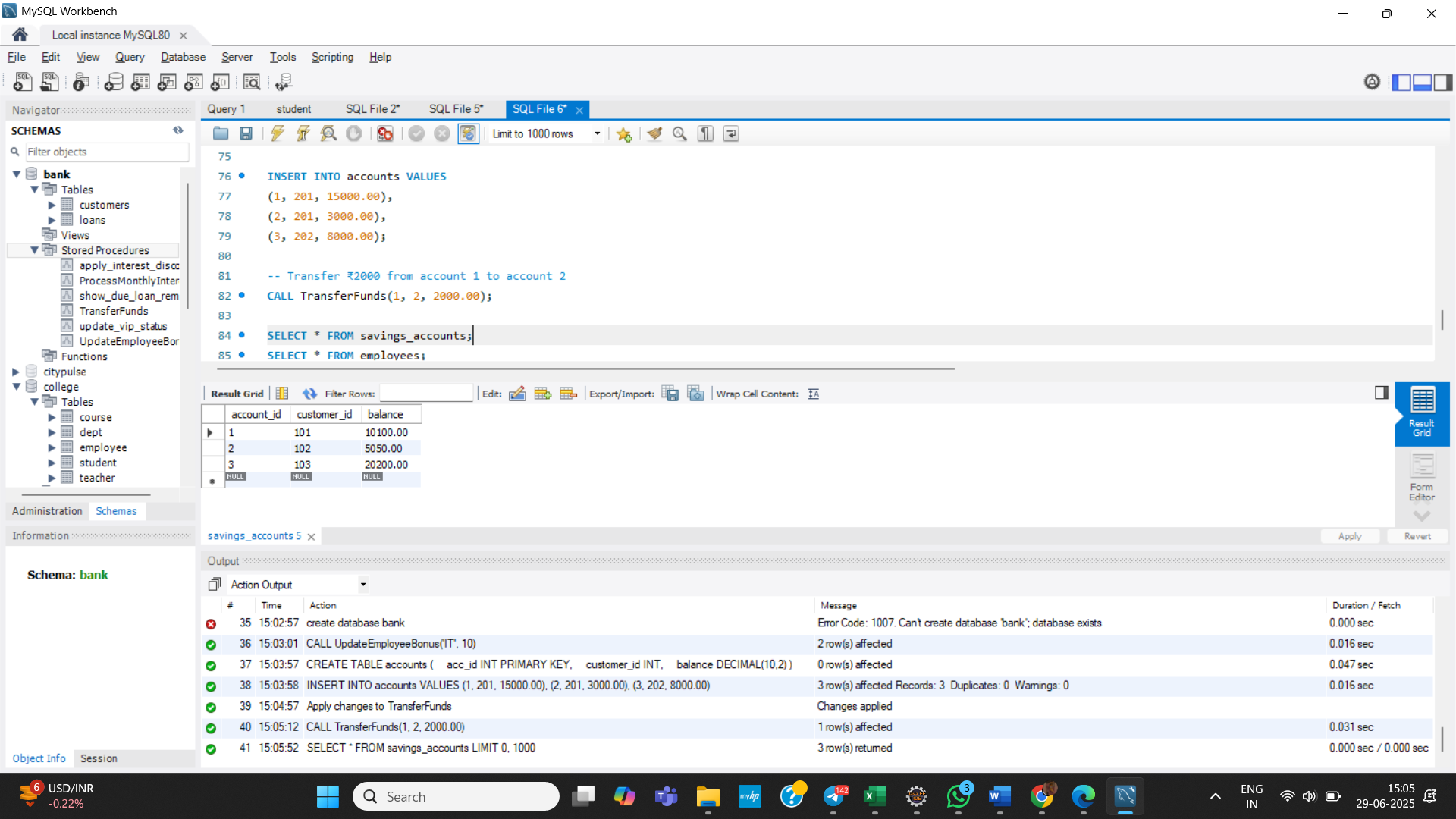
SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Insufficient balance in source account.';

END IF;

END;

**Output:**

 A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.

**JUnit, Mockito and SL4J**

**3. Setting Up Junit**

**Created a new Java project in your IDE – Week2**

**Added JUnit dependency to project**

pom.xml:

<dependency>

<groupld>junit</groupld>

<artifactld>junit</artifactld>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

**Created a class Calculator.java:**

**package** com.cognizant.Week2;

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

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

**if** (b == 0) **throw** **new** IllegalArgumentException("Cannot divide by zero");

**return** a / b;

}

}

**Created CalculatorTest.java to Test :**

**package** com.cognizant.Week2;

**import** **static** org.junit.Assert.*assertEquals*;

**import** **static** org.junit.Assert.*fail*;

**import** org.junit.Test;

**public** **class** CalculatorTest {

@Test

**public** **void** test() {

*fail*("Not yet implemented");

}

Calculator calc = **new** Calculator();

@Test

**public** **void** testAdd() {

*assertEquals*(5, calc.add(2, 3));

}

@Test

**public** **void** testSubtract() {

*assertEquals*(1, calc.subtract(4, 3));

}

@Test

**public** **void** testMultiply() {

*assertEquals*(12, calc.multiply(3, 4));

}

@Test

**public** **void** testDivide() {

*assertEquals*(5, calc.divide(10, 2));

}

@Test(expected = IllegalArgumentException.**class**)

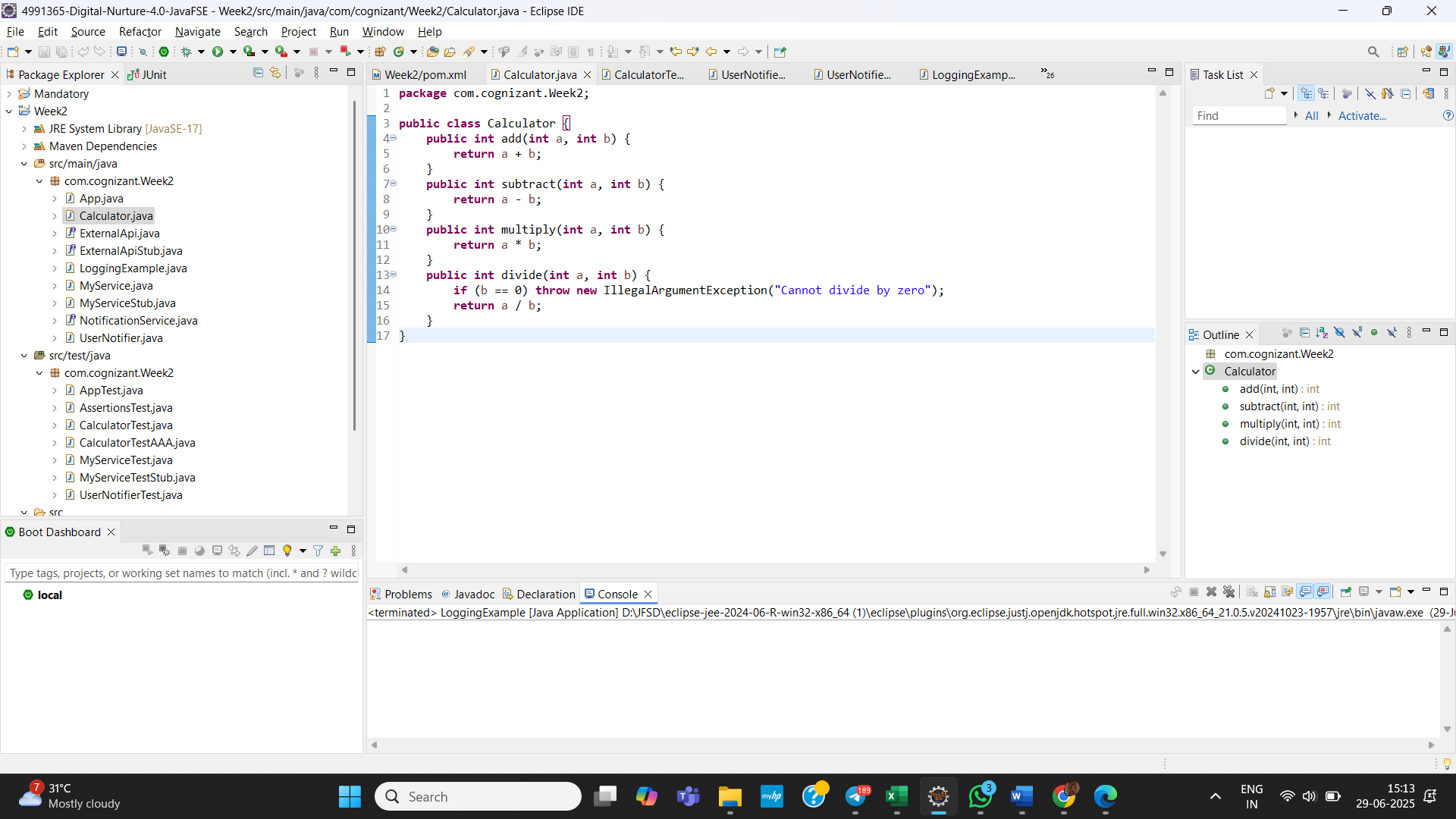
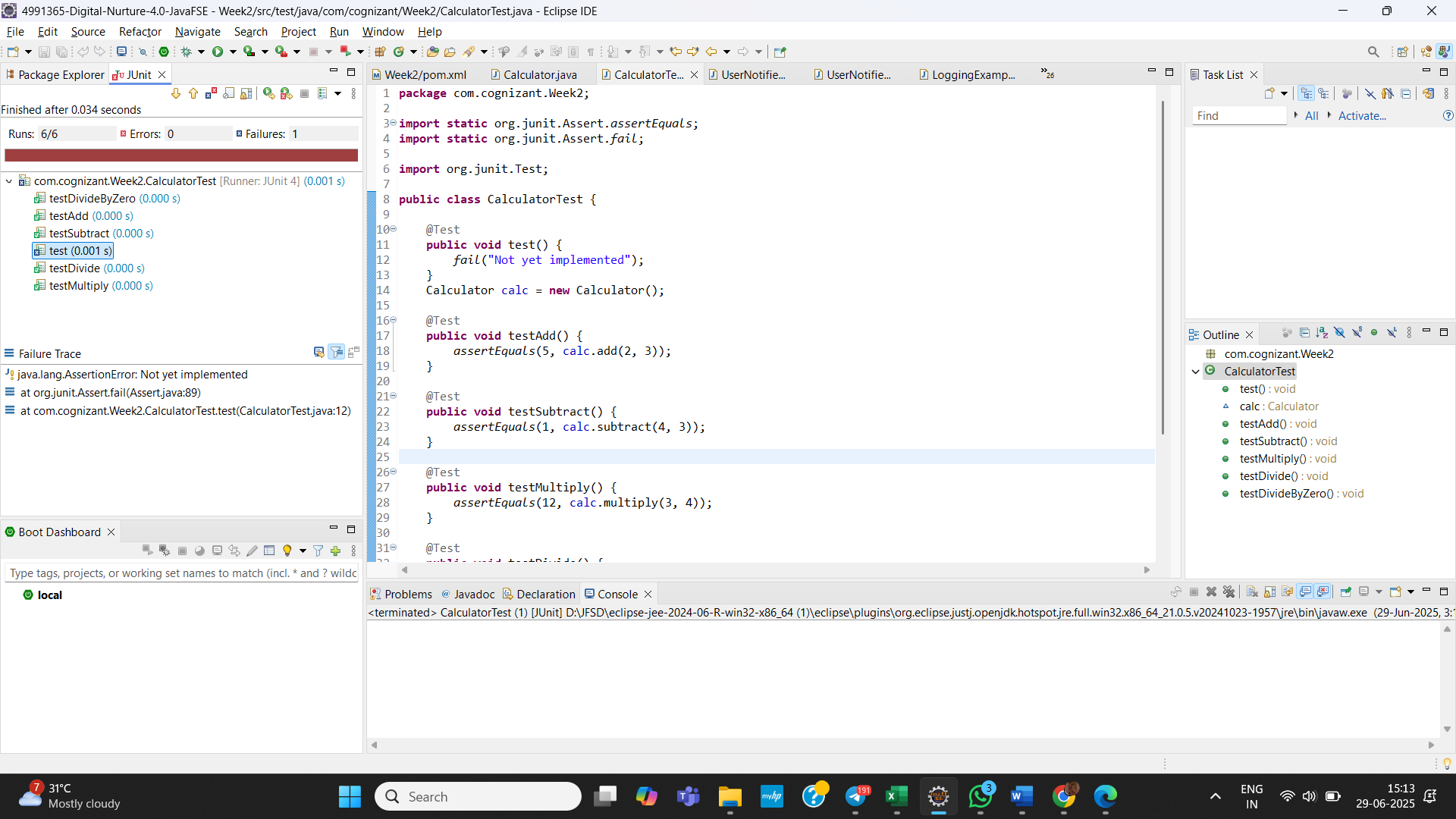
**public** **void** testDivideByZero() {

calc.divide(10, 0);

}

}

**Output:**

**4. Assertions in Junit:-**

**Created a test file named with AssertionsTest.java.**

**package** com.cognizant.Week2;

**import** org.junit.Test;

**import** **static** org.junit.Assert.\*;

**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());

}

}

**Output:**

A screenshot of a computer

AI-generated content may be incorrect.

**5. Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in Junit**

**Create a test file named with CalculatorTestAAA.java.**

**package** com.cognizant.Week2;

**import** **static** org.junit.Assert.\*;

**import** org.junit.After;

**import** org.junit.Before;

**import** org.junit.Test;

**public** **class** CalculatorTestAAA {

**private** Calculator calculator;

// Setup method (runs before every test)

@Before

**public** **void** setUp() {

calculator = **new** Calculator();

System.***out***.println("Setting up Calculator instance...");

}

// Teardown method (runs after every test)

@After

**public** **void** tearDown() {

calculator = **null**;

System.***out***.println("Cleaning up Calculator instance...");

}

@Test

**public** **void** testAdd() {

// Arrange

**int** a = 2, b = 3;

// Act

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

// Assert

*assertEquals*(5, result);

}

@Test

**public** **void** testSubtract() {

**int** result = calculator.subtract(10, 4);

*assertEquals*(6, result);

}

@Test

**public** **void** testMultiply() {

**int** result = calculator.multiply(4, 3);

*assertEquals*(12, result);

}

@Test

**public** **void** testDivide() {

**int** result = calculator.divide(20, 4);

*assertEquals*(5, result);

}

@Test(expected = IllegalArgumentException.**class**)

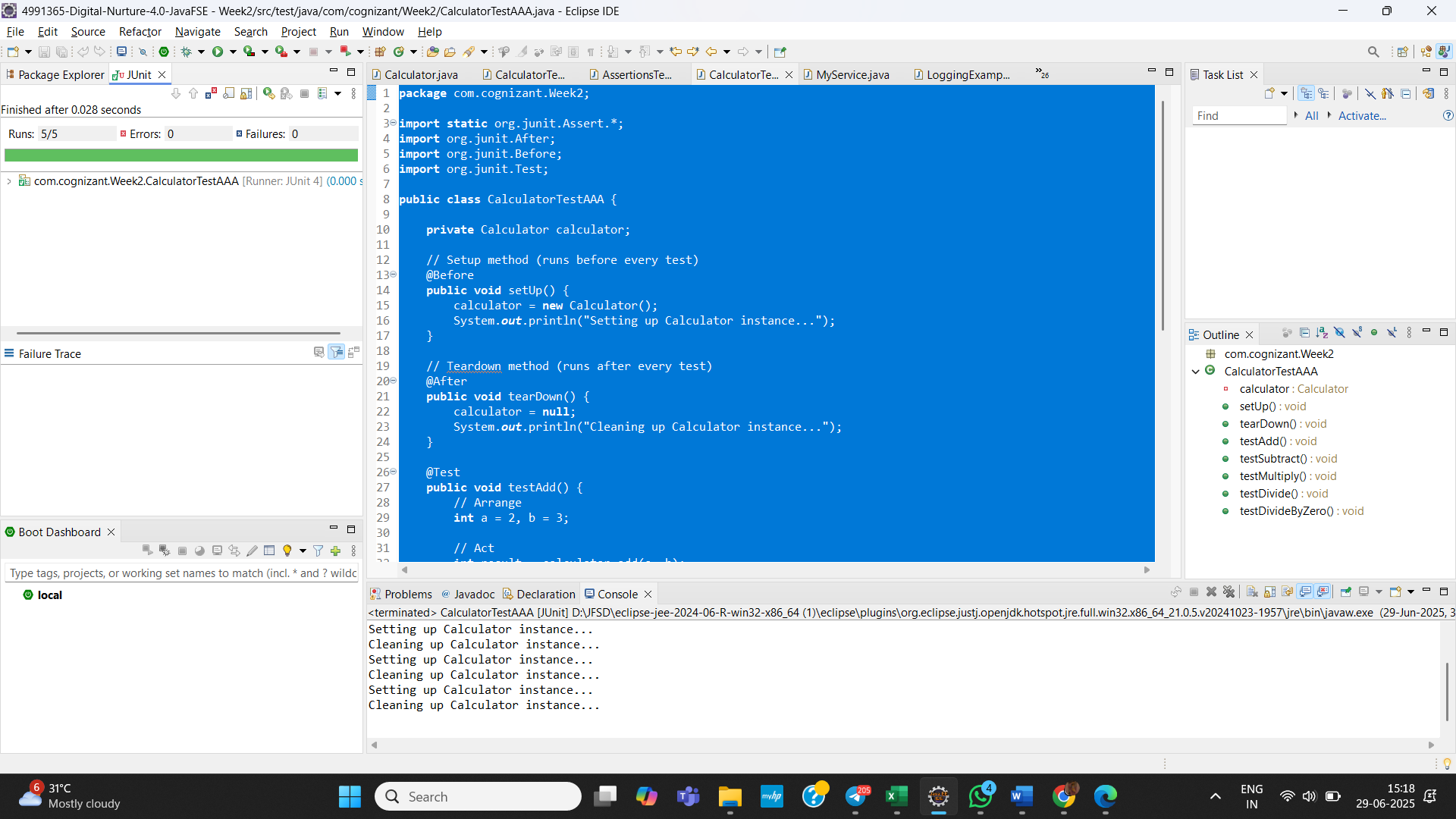
**public** **void** testDivideByZero() {

calculator.divide(10, 0);

}

}

Output:



**TDD using JUnit5 and Mockito**

**6. Mocking and Stubbing**

**Add Dependencies:**

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.11.0</version> <!-- or latest -->

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version> <!-- or latest -->

<scope>test</scope>

</dependency>

**Created the externalAPI interface:**

**package** com.cognizant.Week2;

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

String getData();

}

**Created the MyService Class:**

**package** com.cognizant.Week2;

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

**private** ExternalApi externalApi;

**public** MyService(ExternalApi externalApi) {

**this**.externalApi = externalApi;

}

**public** String fetchData() {

**return** externalApi.getData();

}

}

**Created the myServiceTest Class:**

**package** com.cognizant.Week2;

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

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

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

**import** org.mockito.Mockito;

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

@Test

**public** **void** testExternalApi() {

// Step 1: Create a mock of ExternalApi

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

// Step 2: Stub the getData() method

*when*(mockApi.getData()).thenReturn("Mock Data");

// Step 3: Inject the mock into MyService

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

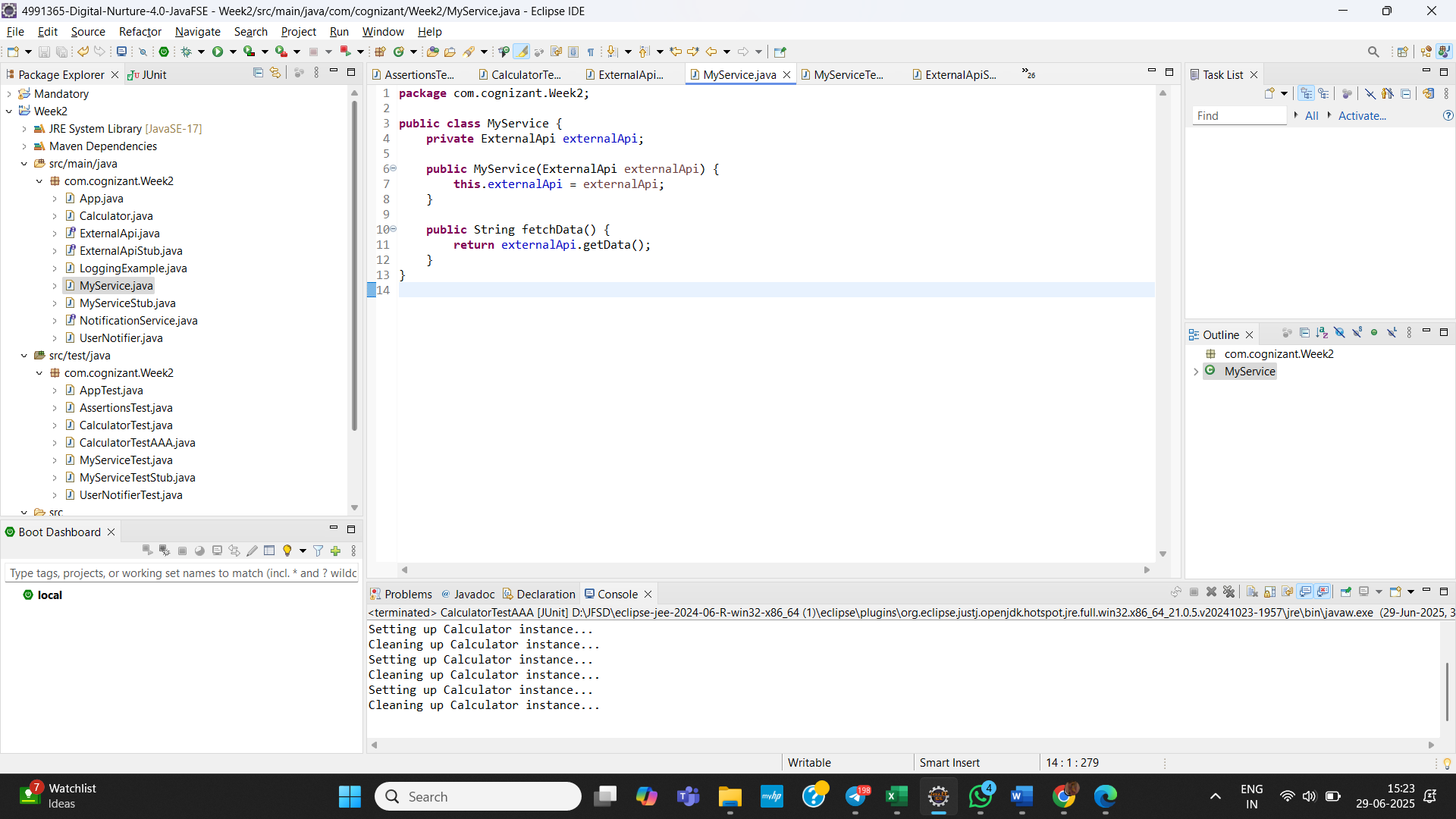
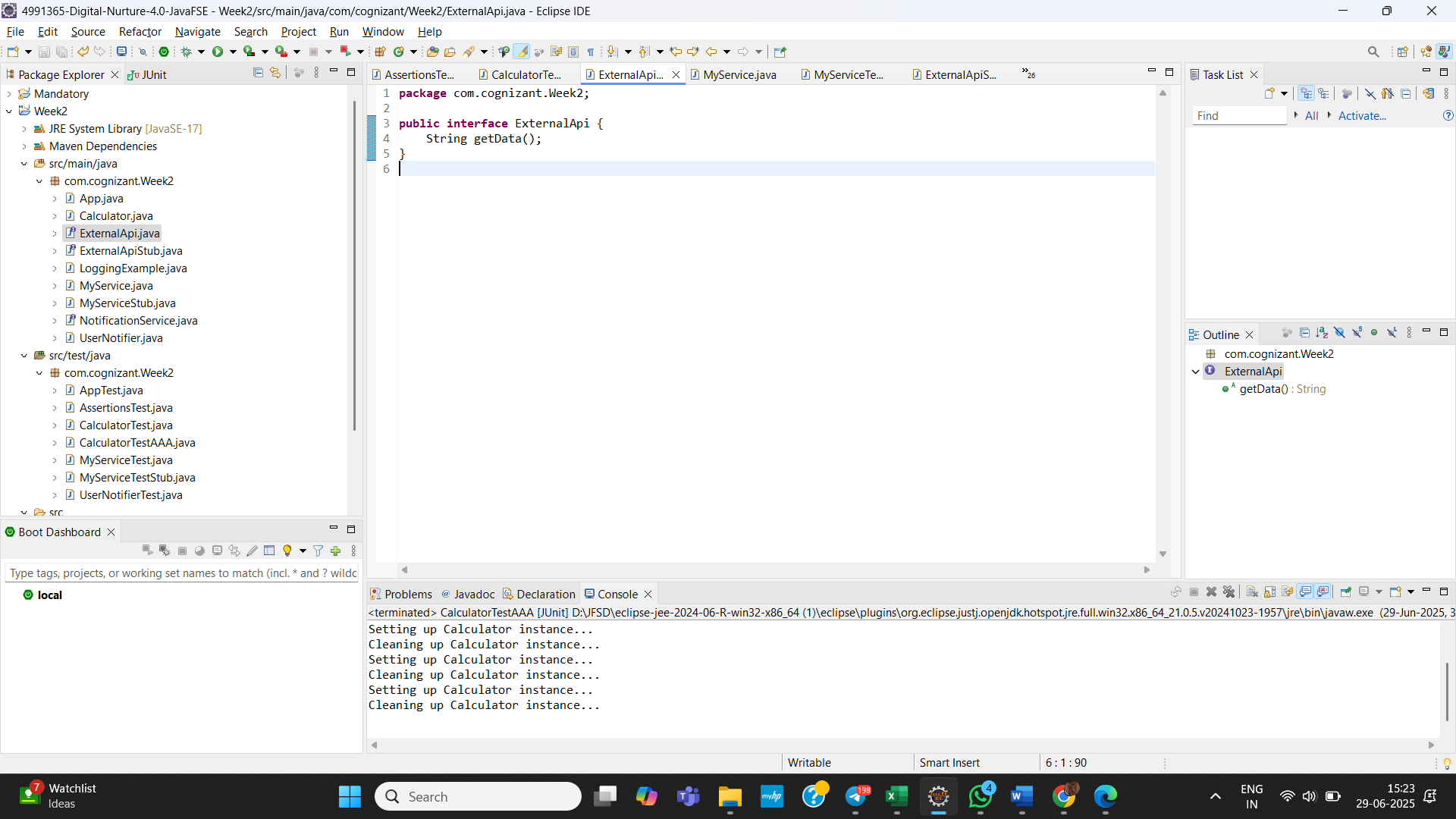
// Step 4: Call the method and assert the result

String result = service.fetchData();

*assertEquals*("Mock Data", result);

}

}

**Output:** A screenshot of a computer

AI-generated content may be incorrect.

**7. Mocking and Stubbing with Multiple Returns:**

**Created the externalAPIStub interface:**

**package** com.cognizant.Week2;

**public** **interface** ExternalApiStub {

String getData();

}

**Created the MyServiceStub Class:**

**package** com.cognizant.Week2;

**public** **class** MyServiceStub {

**private** ExternalApiStub externalApiStub;

**public** MyServiceStub(ExternalApiStub externalApiStub) {

**this**.externalApiStub = externalApiStub;

}

**public** String fetchFirstCall() {

**return** externalApiStub.getData();

}

**public** String fetchSecondCall() {

**return** externalApiStub.getData();

}

}

**Created the MyServiceStubTest Class:**

**package** com.cognizant.Week2;

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

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

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

**public** **class** MyServiceTestStub {

@Test

**public** **void** testExternalApiMultipleReturns() {

// Step 1: Create mock object

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

// Step 2: Stub to return different values for consecutive calls

*when*(mockApi.getData())

.thenReturn("First Response")

.thenReturn("Second Response");

// Step 3: Inject into service

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

String first = service.fetchFirstCall();

String second = service.fetchSecondCall();

System.***out***.println("First: " + first); // <- Print here

System.***out***.println("Second: " + second);

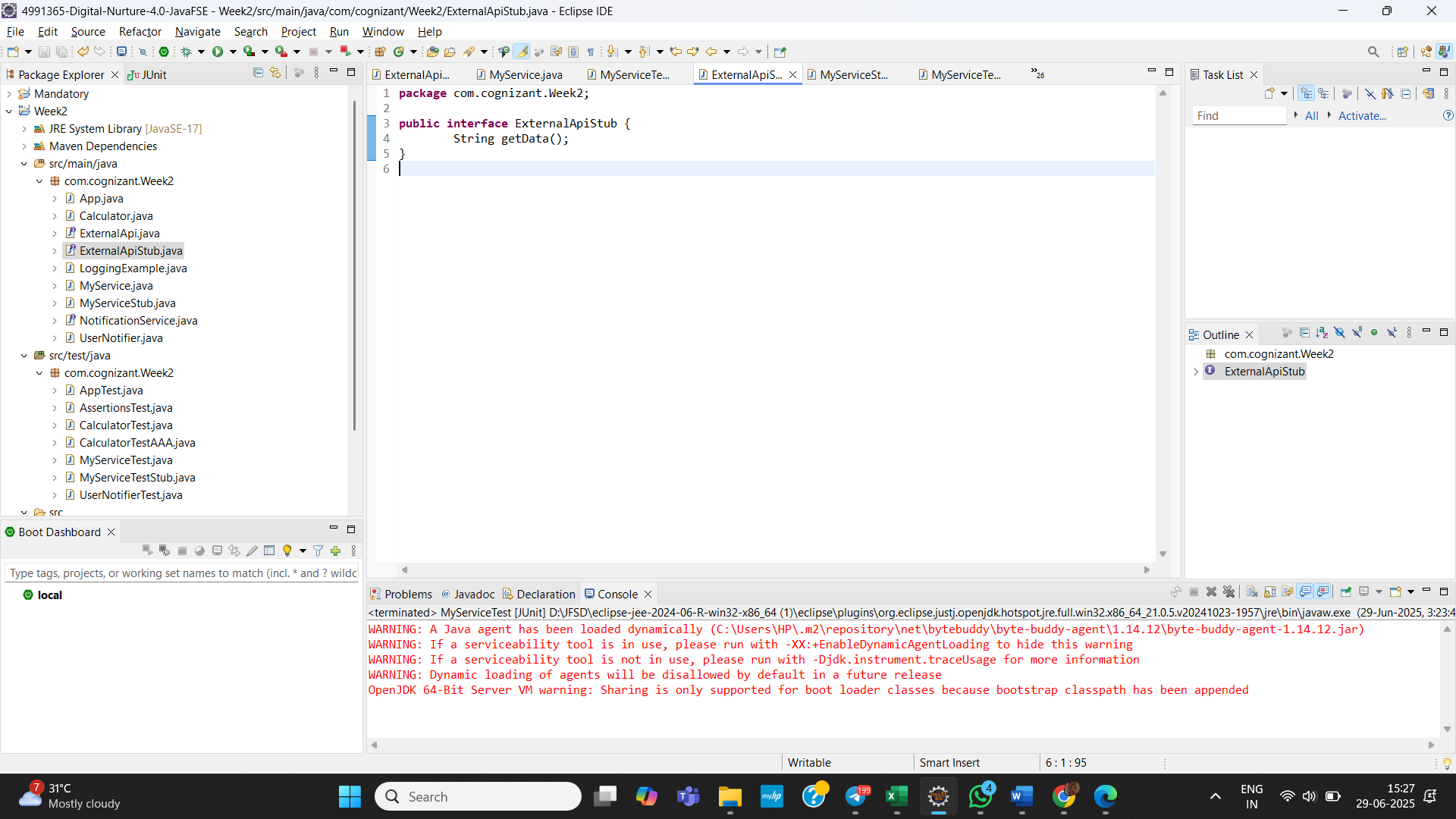
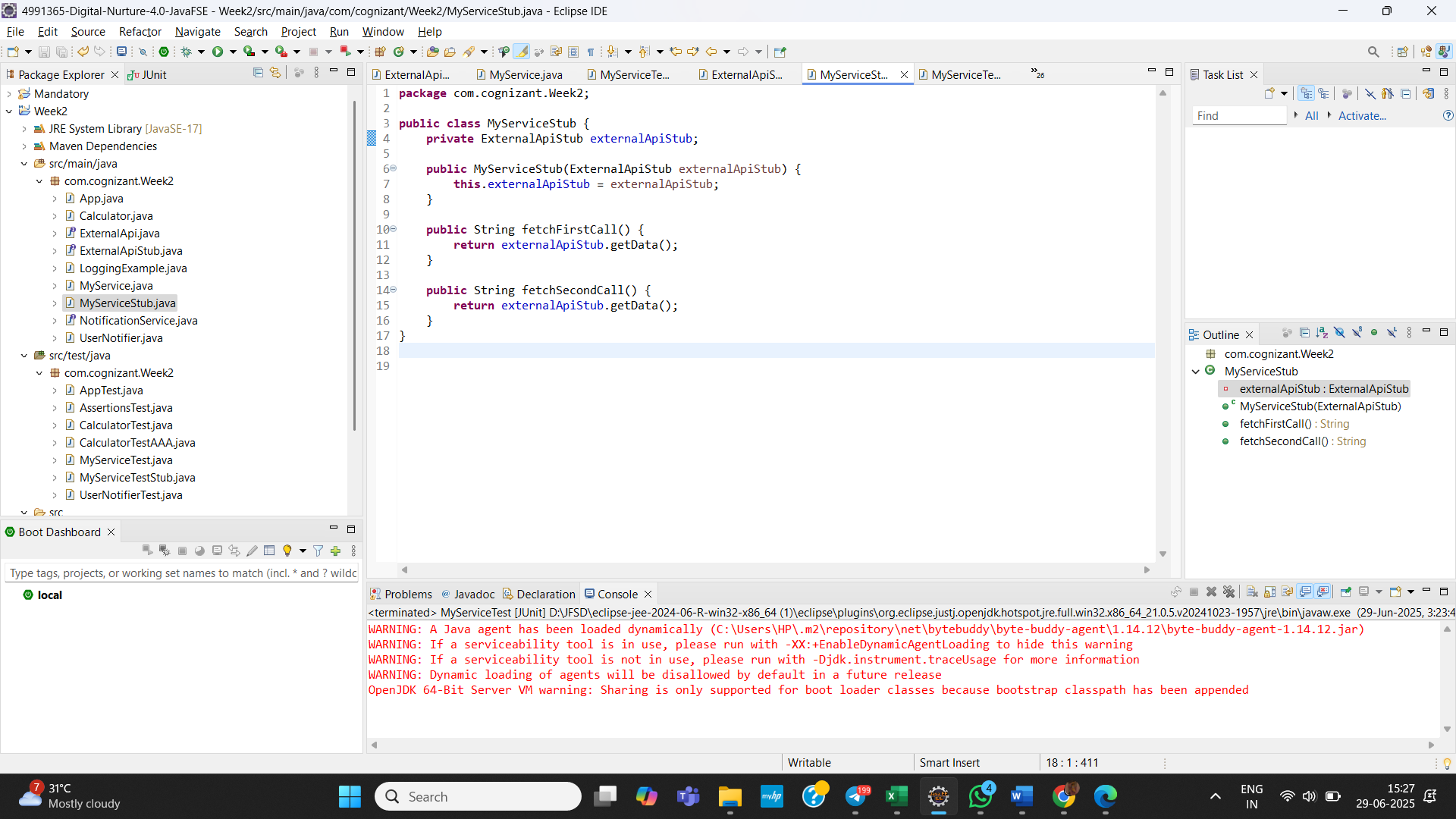
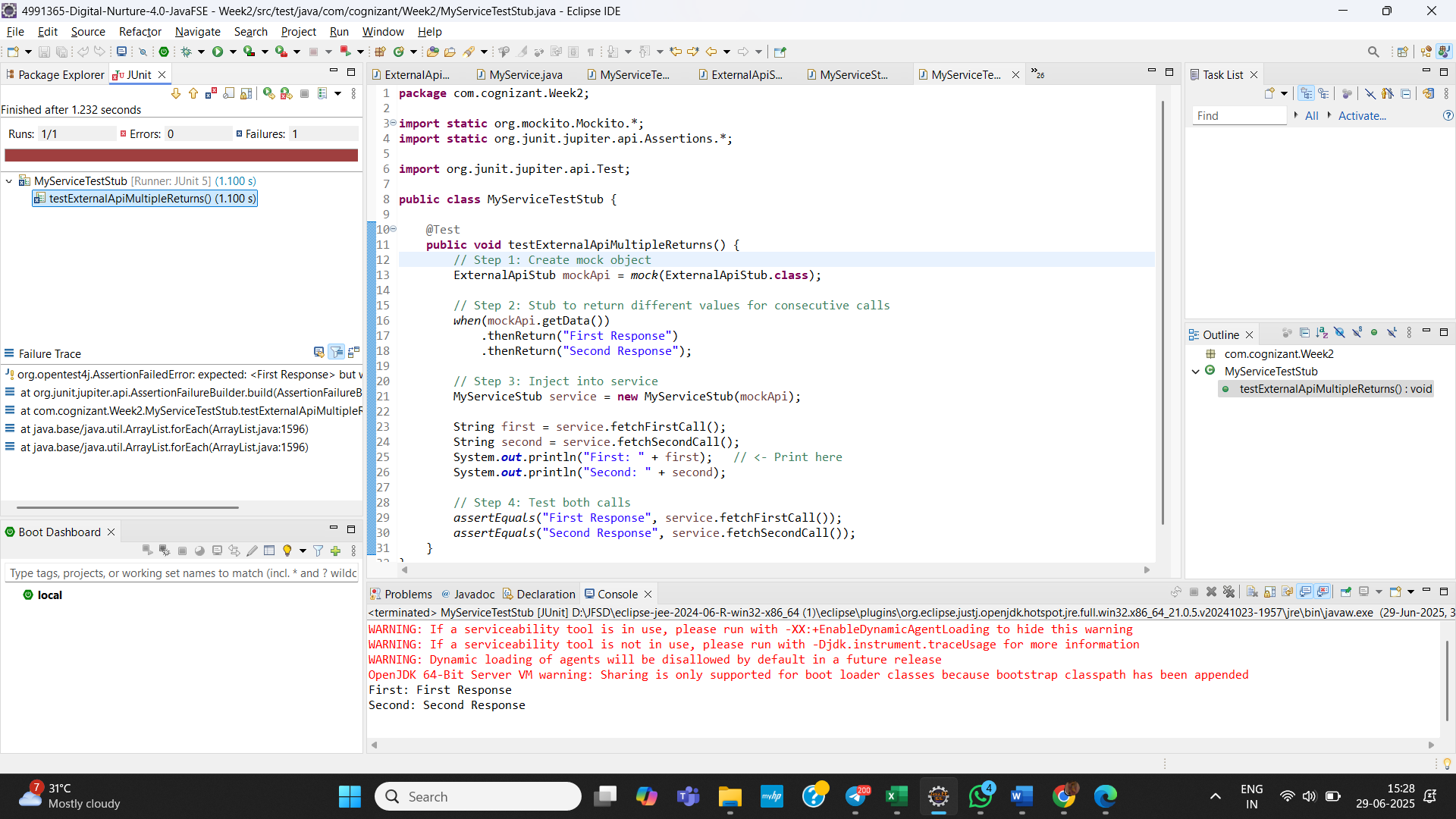
// Step 4: Test both calls

*assertEquals*("First Response", service.fetchFirstCall());

*assertEquals*("Second Response", service.fetchSecondCall());

}

}

**Output:**   

**8. Verifying Interactions:**

**NotificationService:**

**package** com.cognizant.Week2;

**public** **interface** NotificationService {

**void** sendEmail();

**void** sendSMS();

**void** sendPush();

}

**UserNotifier:**

**package** com.cognizant.Week2;

**public** **class** UserNotifier {

**private** NotificationService service;

**public** UserNotifier(NotificationService service) {

**this**.service = service;

}

**public** **void** notifyUser() {

service.sendEmail();

service.sendSMS();

service.sendPush();

}

}

**UserNotifierTest:**

**package** com.cognizant.Week2;

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

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

**import** org.mockito.InOrder;

**public** **class** UserNotifierTest {

@Test

**public** **void** testNotificationOrder() {

// Step 1: Create a mock

NotificationService mockService = *mock*(NotificationService.**class**);

// Step 2: Call methods in a specific order

UserNotifier notifier = **new** UserNotifier(mockService);

notifier.notifyUser();

// Step 3: Verify interaction order

InOrder inOrder = *inOrder*(mockService);

inOrder.verify(mockService).sendEmail();

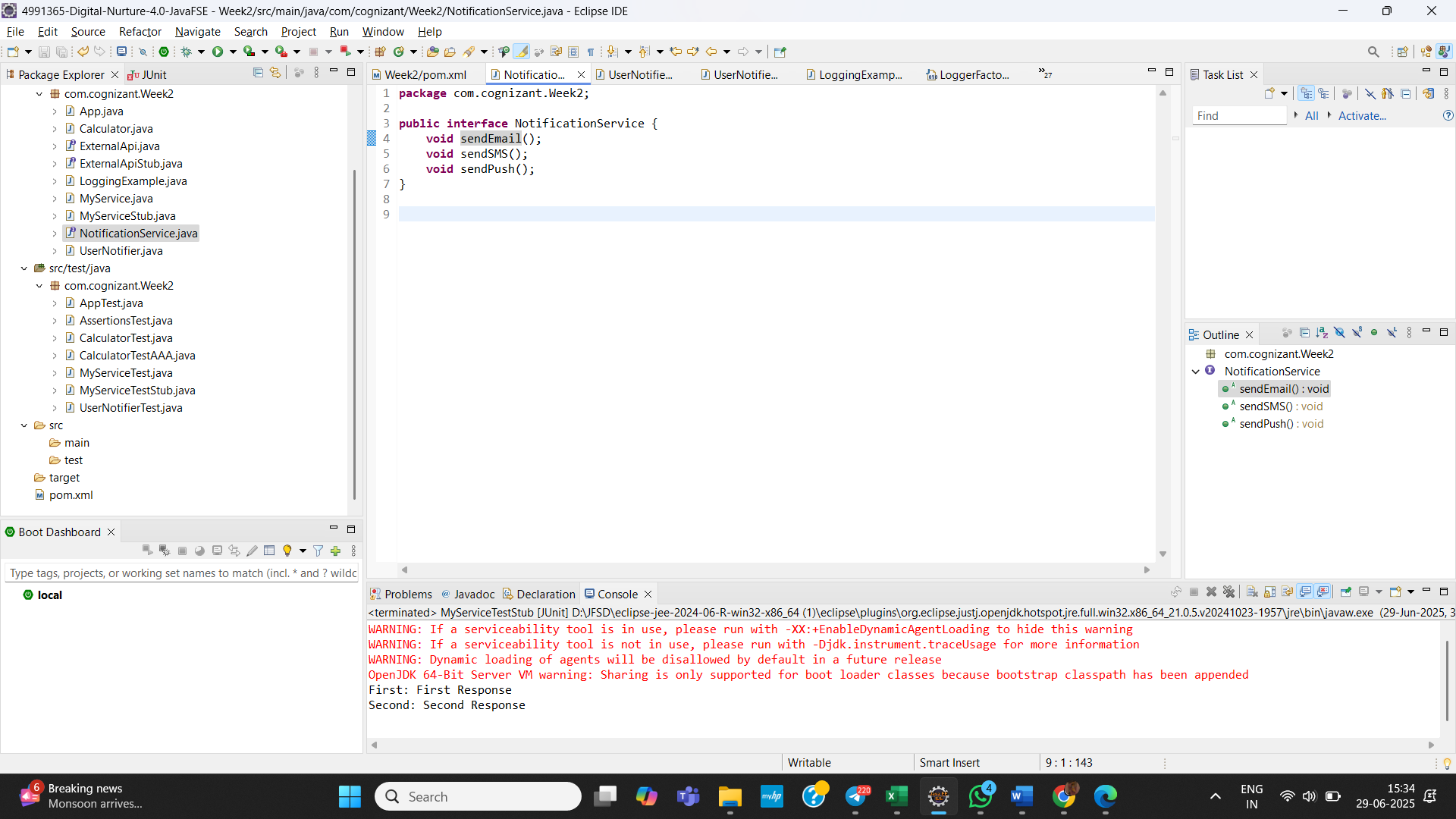
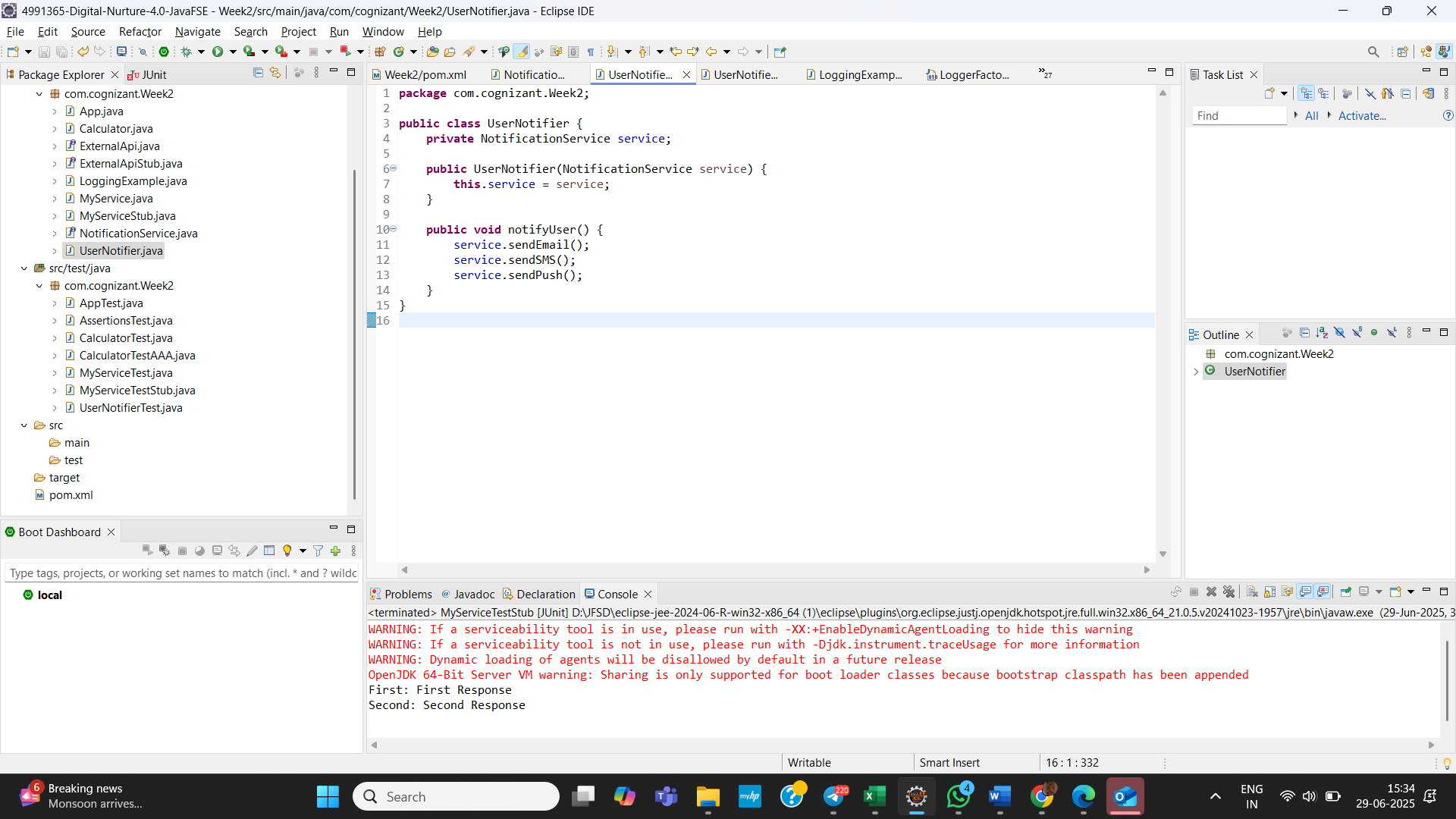
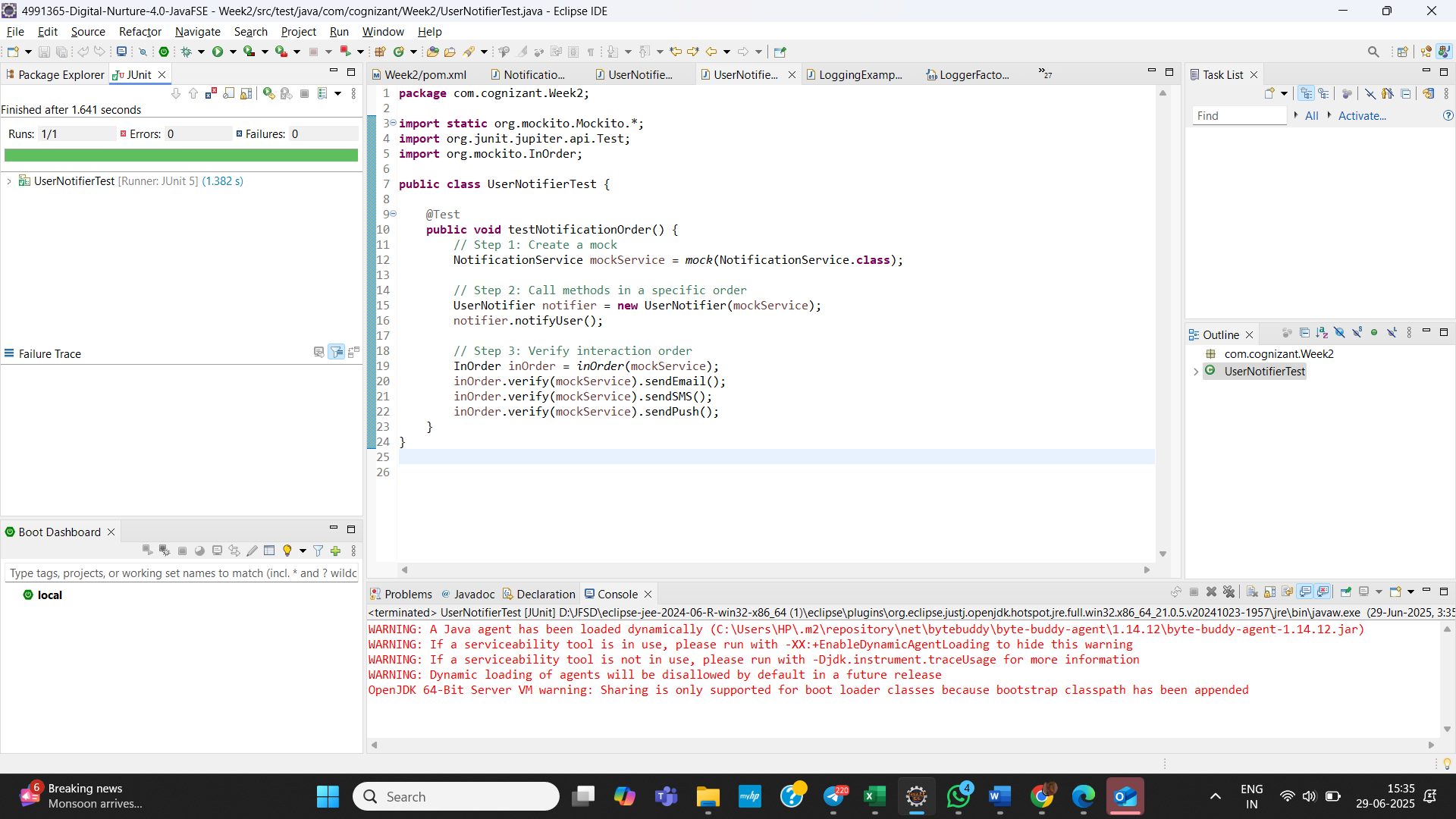
inOrder.verify(mockService).sendSMS();

inOrder.verify(mockService).sendPush();

}

}

**Output:**

**SLF4J logging framework**

**9. Logging Error Messages and Warning Levels:**

**Added Dependencies:**

<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>

**Create LoggingExample.java:**

**package** com.cognizant.Week2;

**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:**

