**WEEK – 2**

**PL/SQL programming - PLSQL\_Exercises**

**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**

**Create Tables**

**--**Customers

CREATE TABLE customers (

customer\_id NUMBER PRIMARY KEY,

name VARCHAR2(50),

age NUMBER,

balance NUMBER,

isvip VARCHAR2(5)

);

--Loans

CREATE TABLE loans (

loan\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

interest\_rate NUMBER(5,2),

due\_date DATE,

CONSTRAINT fk\_customer

FOREIGN KEY (customer\_id)

REFERENCES customers(customer\_id)

);

**Insert Sample Data**

INSERT INTO customers VALUES (1, 'Ravi', 65, 12000, 'FALSE');

INSERT INTO customers VALUES (2, 'Anita', 45, 8000, 'FALSE');

INSERT INTO customers VALUES (3, 'Kumar', 70, 15000, 'FALSE');

INSERT INTO loans VALUES (101, 1, 9.5, SYSDATE + 20);

INSERT INTO loans VALUES (102, 2, 10.0, SYSDATE + 10);

INSERT INTO loans VALUES (103, 3, 11.0, SYSDATE + 25);

**Scenario 1 - Apply 1% Discount to Interest Rate for Customers Aged Above 60**

**Scenario 2 - Set IsVIP Flag to True for Customers with Balance > 10000**

**Scenario 3 - Print Loan Reminders Due Within Next 30 Days**

SET SERVEROUTPUT ON;

BEGIN

-- ========== DISCOUNT ==========

DBMS\_OUTPUT.PUT\_LINE('DISCOUNT:');

FOR customer\_rec IN (

SELECT customer\_id FROM customers WHERE age > 60

) LOOP

FOR loan\_rec IN (

SELECT loan\_id, interest\_rate FROM loans

WHERE customer\_id = customer\_rec.customer\_id

) LOOP

UPDATE loans

SET interest\_rate = interest\_rate - 1

WHERE loan\_id = loan\_rec.loan\_id;

DBMS\_OUTPUT.PUT\_LINE(' Applied 1% discount to loan ID: ' || loan\_rec.loan\_id ||

' for customer ID: ' || customer\_rec.customer\_id);

END LOOP;

END LOOP;

-- ========== VIP ==========

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || 'VIP:');

FOR customer\_rec IN (

SELECT customer\_id, balance FROM customers

WHERE balance > 10000

) LOOP

UPDATE customers

SET isvip = 'TRUE'

WHERE customer\_id = customer\_rec.customer\_id;

DBMS\_OUTPUT.PUT\_LINE(' Promoted customer ID ' || customer\_rec.customer\_id || ' to VIP.');

END LOOP;

-- ========== REMINDER ==========

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || 'REMINDER:');

FOR loan\_rec IN (

SELECT l.loan\_id, l.due\_date, c.name, c.customer\_id

FROM loans l

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

WHERE l.due\_date BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE(' Loan ID ' || loan\_rec.loan\_id ||

' for customer "' || loan\_rec.name ||

'" (ID: ' || loan\_rec.customer\_id || ') is due on ' ||

TO\_CHAR(loan\_rec.due\_date, 'DD-MON-YYYY'));

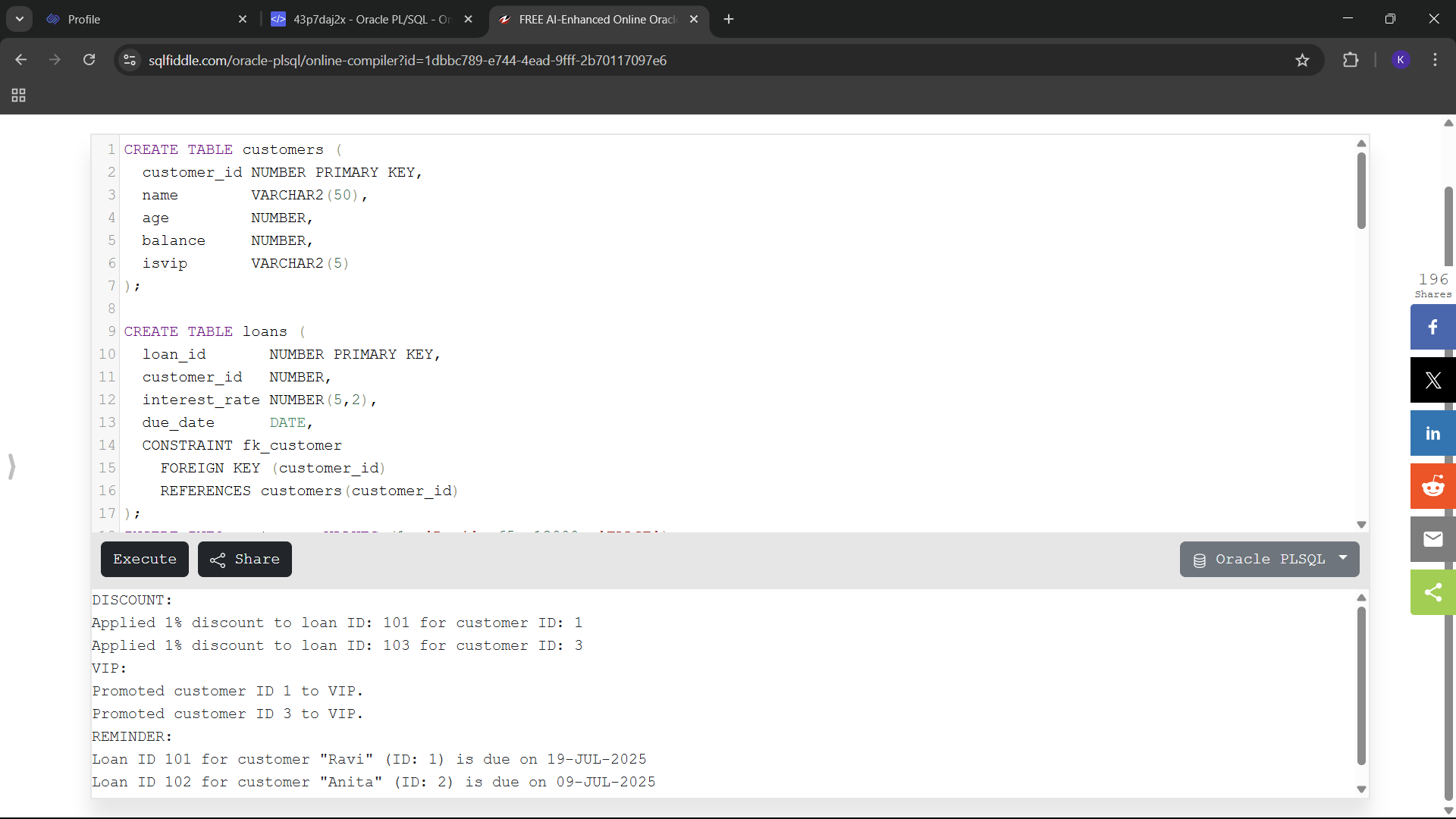
END LOOP;

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**

**Create tables**

-- Savings accounts

CREATE TABLE savings\_accounts (

account\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

balance NUMBER

);

-- Employees

CREATE TABLE employees (

emp\_id NUMBER PRIMARY KEY,

name VARCHAR2(50),

salary NUMBER,

dept\_id NUMBER

);

-- Generic accounts for transfers

CREATE TABLE bank\_accounts (

account\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

balance NUMBER

);

**Insert Sample Data**

-- Insert data for savings\_accounts

INSERT INTO savings\_accounts VALUES (101, 1, 10000);

INSERT INTO savings\_accounts VALUES (102, 2, 15000);

INSERT INTO savings\_accounts VALUES (103, 3, 20000);

-- Insert data for employees

INSERT INTO employees VALUES (201, 'Alice', 50000, 10);

INSERT INTO employees VALUES (202, 'Bob', 55000, 10);

INSERT INTO employees VALUES (203, 'Charlie', 60000, 20);

-- Insert data for bank\_accounts

INSERT INTO bank\_accounts VALUES (301, 1, 8000);

INSERT INTO bank\_accounts VALUES (302, 2, 5000);

INSERT INTO bank\_accounts VALUES (303, 3, 7000);

-- Turn on output

SET SERVEROUTPUT ON;

--------------------------------------------------------------------------------

-- 1. Stored Procedure: ProcessMonthlyInterest

--------------------------------------------------------------------------------

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

DBMS\_OUTPUT.PUT\_LINE('PROCESSING INTEREST:');

FOR acc IN (SELECT account\_id, balance FROM savings\_accounts) LOOP

UPDATE savings\_accounts

SET balance = balance + (acc.balance \* 0.01)

WHERE account\_id = acc.account\_id;

DBMS\_OUTPUT.PUT\_LINE(' Added 1% interest to account ID: ' || acc.account\_id);

END LOOP;

COMMIT;

END;

/

--------------------------------------------------------------------------------

-- 2. Stored Procedure: UpdateEmployeeBonus

--------------------------------------------------------------------------------

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_dept\_id IN NUMBER,

p\_bonus\_percent IN NUMBER

) IS

BEGIN

DBMS\_OUTPUT.PUT\_LINE('UPDATING BONUSES FOR DEPT ' || p\_dept\_id || ':');

FOR emp IN (

SELECT emp\_id, salary FROM employees WHERE dept\_id = p\_dept\_id

) LOOP

UPDATE employees

SET salary = salary + (emp.salary \* p\_bonus\_percent / 100)

WHERE emp\_id = emp.emp\_id;

DBMS\_OUTPUT.PUT\_LINE(' Bonus applied to employee ID: ' || emp.emp\_id);

END LOOP;

COMMIT;

END;

/

--------------------------------------------------------------------------------

-- 3. Stored Procedure: TransferFunds

--------------------------------------------------------------------------------

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) IS

v\_balance NUMBER;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('PROCESSING TRANSFER:');

SELECT balance INTO v\_balance

FROM bank\_accounts

WHERE account\_id = p\_from\_account

FOR UPDATE;

IF v\_balance < p\_amount THEN

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

END IF;

UPDATE bank\_accounts

SET balance = balance - p\_amount

WHERE account\_id = p\_from\_account;

UPDATE bank\_accounts

SET balance = balance + p\_amount

WHERE account\_id = p\_to\_account;

DBMS\_OUTPUT.PUT\_LINE(' Transferred ' || p\_amount || ' from account ' || p\_from\_account ||

' to account ' || p\_to\_account);

COMMIT;

END;

/

--------------------------------------------------------------------------------

-- Execute all 3 procedures

--------------------------------------------------------------------------------

BEGIN

DBMS\_OUTPUT.PUT\_LINE('============================');

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

DBMS\_OUTPUT.PUT\_LINE('============================');

ProcessMonthlyInterest;

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '============================');

DBMS\_OUTPUT.PUT\_LINE('RUNNING: UpdateEmployeeBonus (10%)');

DBMS\_OUTPUT.PUT\_LINE('============================');

UpdateEmployeeBonus(10, 10);

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '============================');

DBMS\_OUTPUT.PUT\_LINE('RUNNING: TransferFunds (2000 from 301 to 302)');

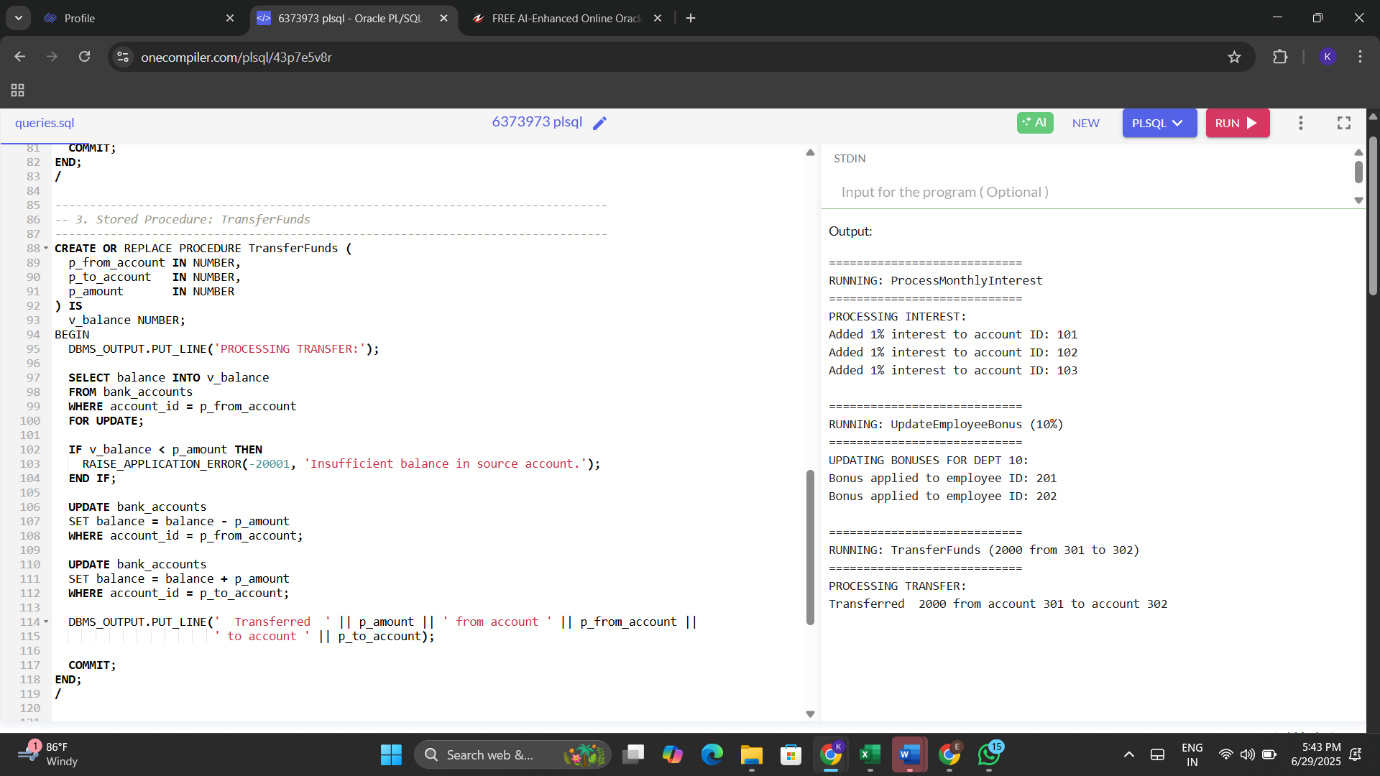
DBMS\_OUTPUT.PUT\_LINE('============================');

TransferFunds(301, 302, 2000);

END;

/

**OUTPUT**



**TDD USING JUNIT5 AND MOCKITO**

**JUnit\_Basic Testing Exercises**

**EXERCISE 1:** **Setting Up Junit**

**SCENARIO**

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

**CODE**

**Main.java**

package com.example;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

PrimeUtils utils = new PrimeUtils();

Scanner scanner = new Scanner(System.in);

System.out.print("Enter a number to check if it is prime: ");

int number = scanner.nextInt();

boolean result = utils.isPrime(number);

if (result) {System.out.println(number + " is a prime number."); }

else {System.out.println(number + " is NOT a prime number."); }

}}

**PrimeUtils.java**

package com.example;

public class PrimeUtils {

public boolean isPrime(int n) {

if (n < 2) return false;

if (n == 2) return true;

if (n % 2 == 0) return false;

for (int i = 3; i \* i <= n; i += 2) {

if (n % i == 0) return false;

}

return true;

}

}

**PrimeUtilsTest.java**

package com.example;

import org.junit.Test;

import com.example.PrimeUtils;

import static org.junit.Assert.\*;

public class PrimeUtilsTest {

private final PrimeUtils utils = new PrimeUtils();

@Test

public void testSmallPrimes() {

assertTrue(utils.isPrime(2));

assertTrue(utils.isPrime(3));

assertTrue(utils.isPrime(5));

assertTrue(utils.isPrime(7));

}

@Test

public void testSmallComposites() {

assertFalse(utils.isPrime(0));

assertFalse(utils.isPrime(1));

assertFalse(utils.isPrime(4));

assertFalse(utils.isPrime(6));

assertFalse(utils.isPrime(9));

}

@Test

public void testLargeNumbers() {

assertTrue(utils.isPrime(97));

assertFalse(utils.isPrime(100));

}

@Test

public void testNegativeNumbers() {

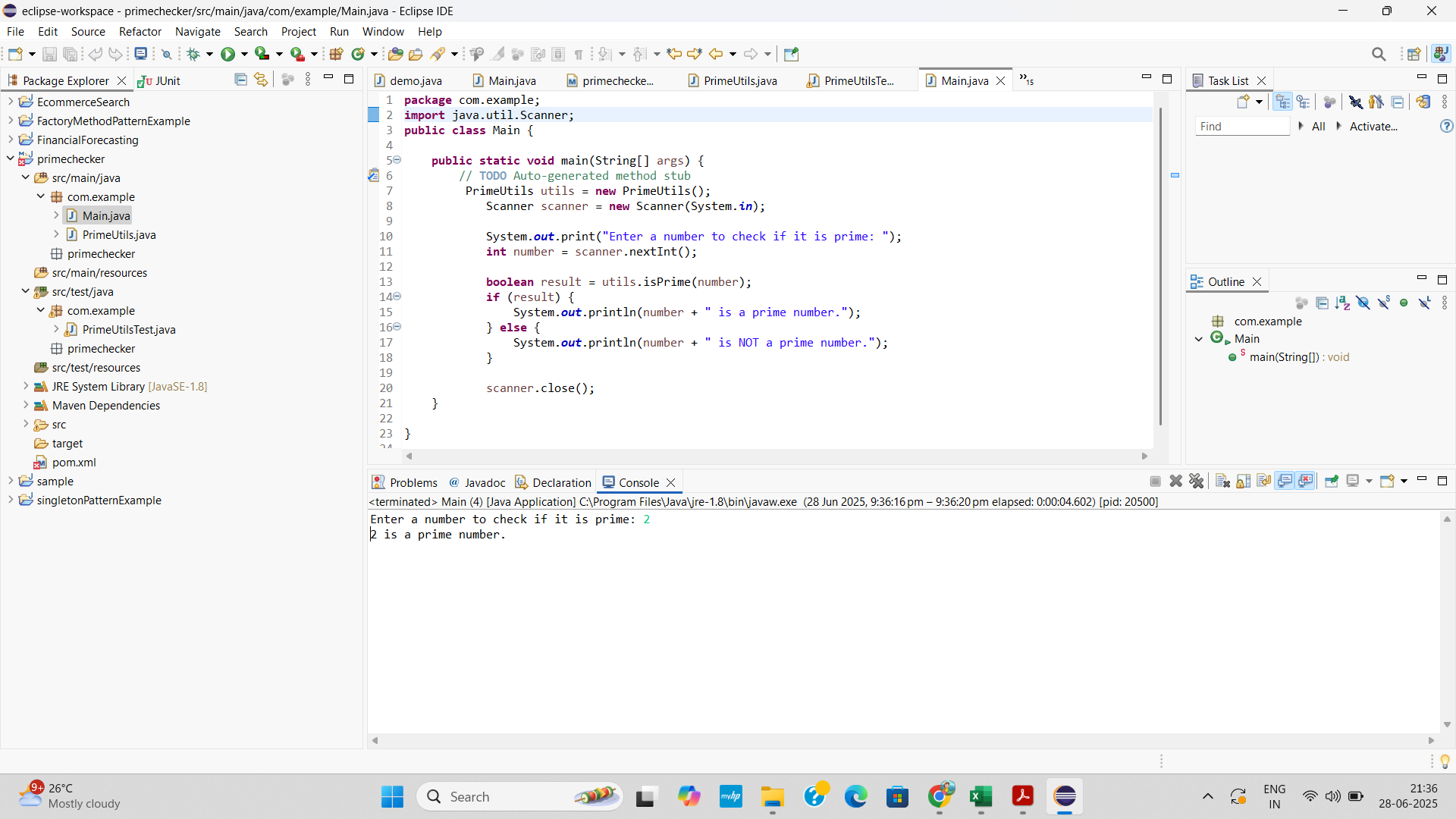
assertFalse(utils.isPrime(-11));

assertFalse(utils.isPrime(-2));

}

}

**OUTPUT**

****

**EXERCISE 3: ASSERTIONS IN JUNIT**

**SCENARIO**

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

**CODE**

**AssertionsTest.java**

package com.example;

import static org.junit.Assert.\*;

import org.junit.Test;

public class AssertionsTest {

public void testAssertions() {

@Test

assertEquals(5, 2 + 3);

assertTrue(5 > 3);

assertFalse(5 < 3);

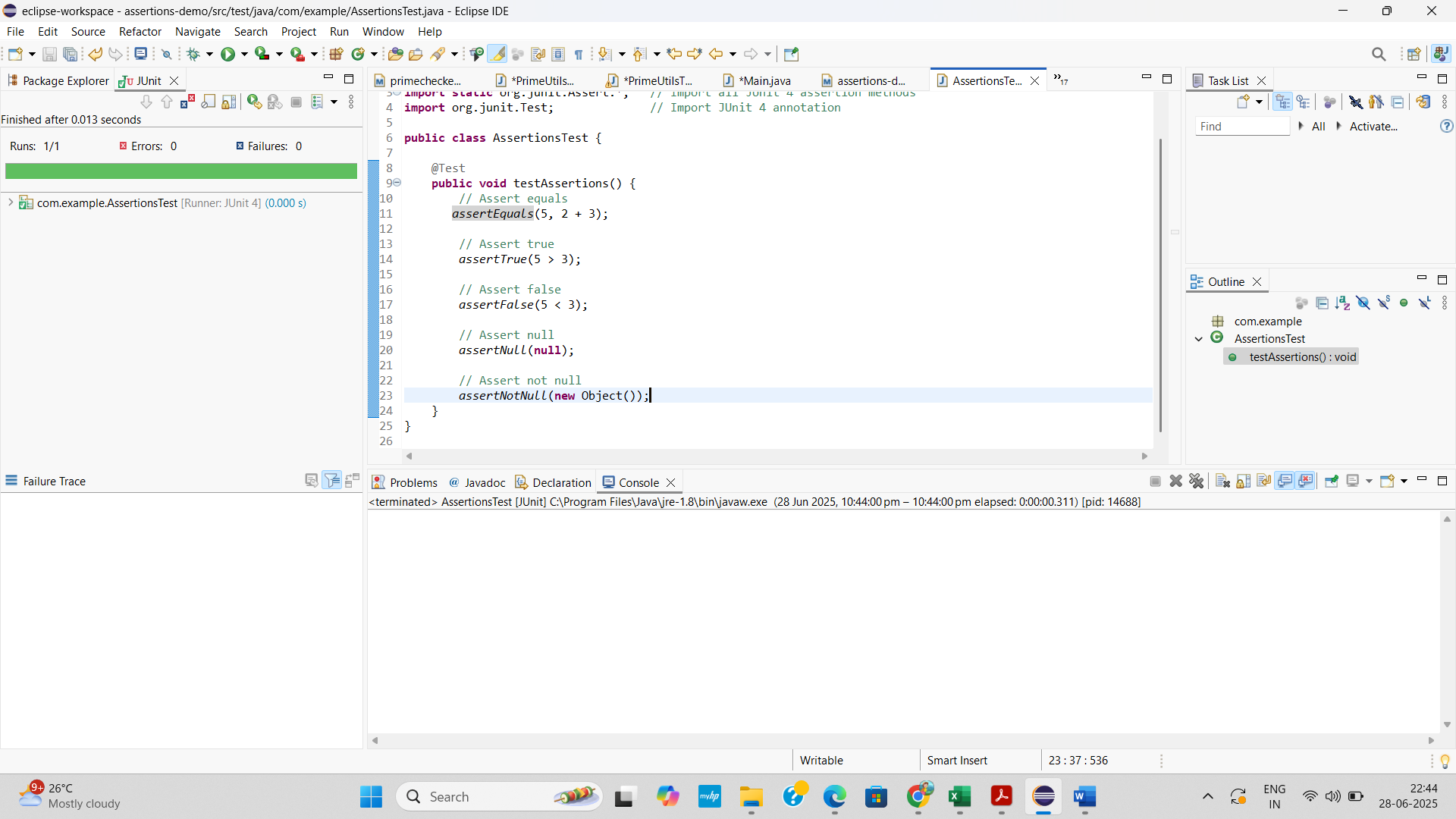
assertNull(null);

assertNotNull(new Object());

}

}

**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.

**CODE**

**OrderProcessor.java**

package com.example;

public class OrderProcessor {

private boolean orderPlaced;

public boolean placeOrder() {

orderPlaced = true;

return orderPlaced;

}

public boolean cancelOrder() {

if (orderPlaced) {

orderPlaced = false;

return true;

}

else {

return false;

}

}

public boolean isOrderPlaced() {

return orderPlaced;

}

}

**OrderProcessorTest.java**

package com.example;

import static org.junit.Assert.\*;

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

public class OrderProcessorTest {

private OrderProcessor orderProcessor;

@Before

public void setUp() {

orderProcessor = new OrderProcessor();

System.out.println("Setup: New OrderProcessor created.");

}

@After

public void tearDown() {

orderProcessor = null;

System.out.println("Teardown: OrderProcessor destroyed.");

}

@Test

public void testPlaceOrder() {

boolean result = orderProcessor.placeOrder();

assertTrue(result);

assertTrue(orderProcessor.isOrderPlaced());

}

@Test

public void testCancelOrder() {

orderProcessor.placeOrder();

boolean result = orderProcessor.cancelOrder();

assertTrue(result);

assertFalse(orderProcessor.isOrderPlaced());

}

@Test

public void testCancelWithoutPlacing() {

boolean result = orderProcessor.cancelOrder();

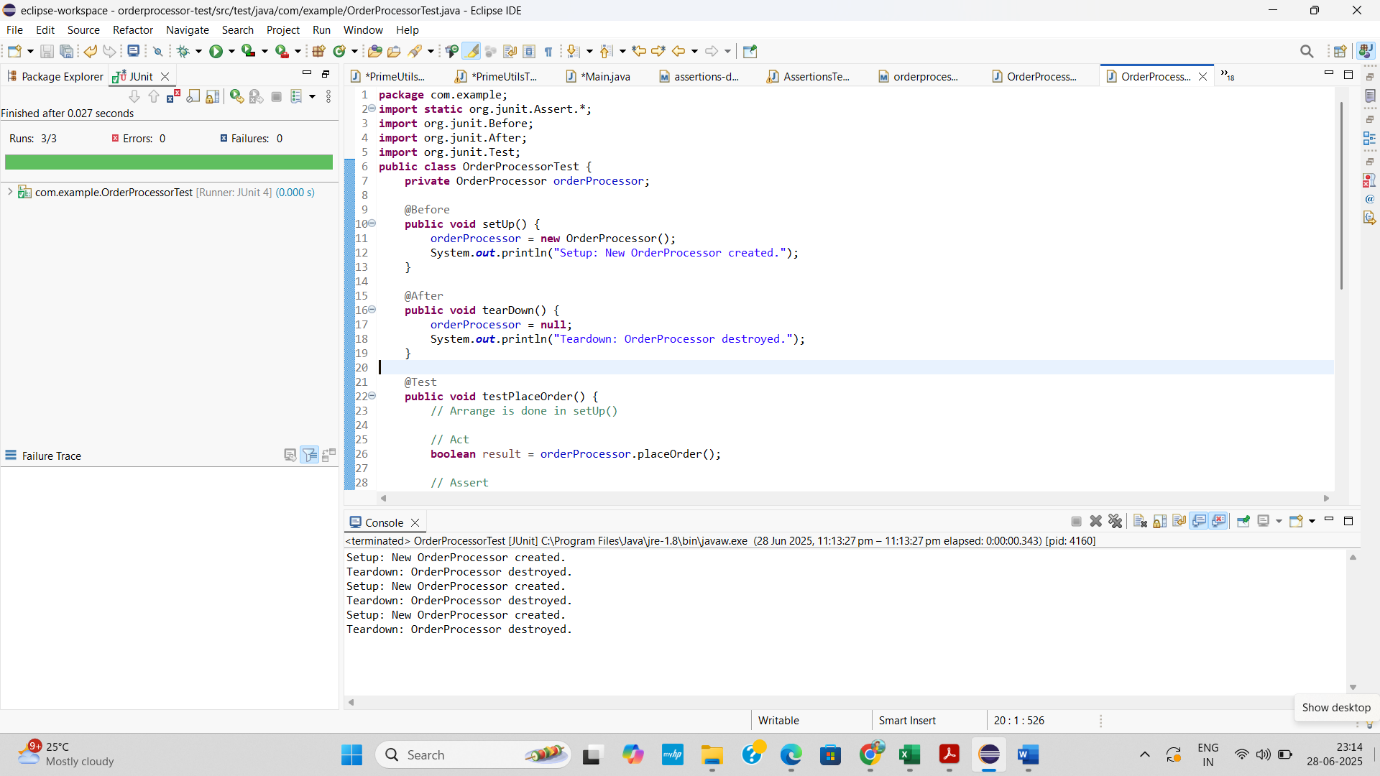
assertFalse(result);

assertFalse(orderProcessor.isOrderPlaced());

}

}

**OUTPUT**

****

**TDD USING JUNIT5 AND MOCKITO - MOCKITO 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.

**CODE**

**ExternalApi.java**

package com.example;

public interface ExternalApi {

String getData();

}

**MyService.java**

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java**

package com.example;

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() {

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

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

MyService service = new MyService(mockApi);

String result = service.fetchData();

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

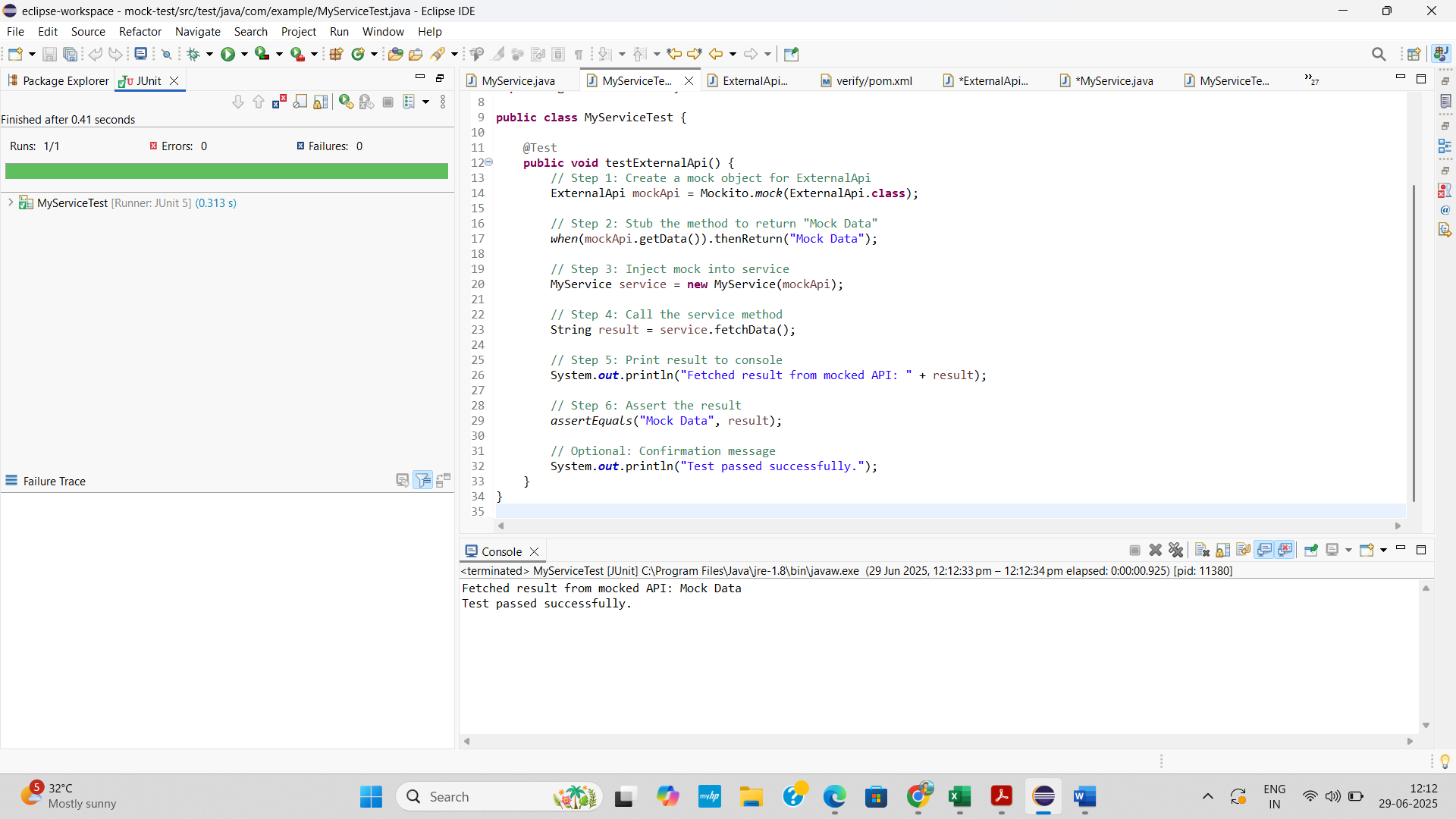
assertEquals("Mock Data", result);

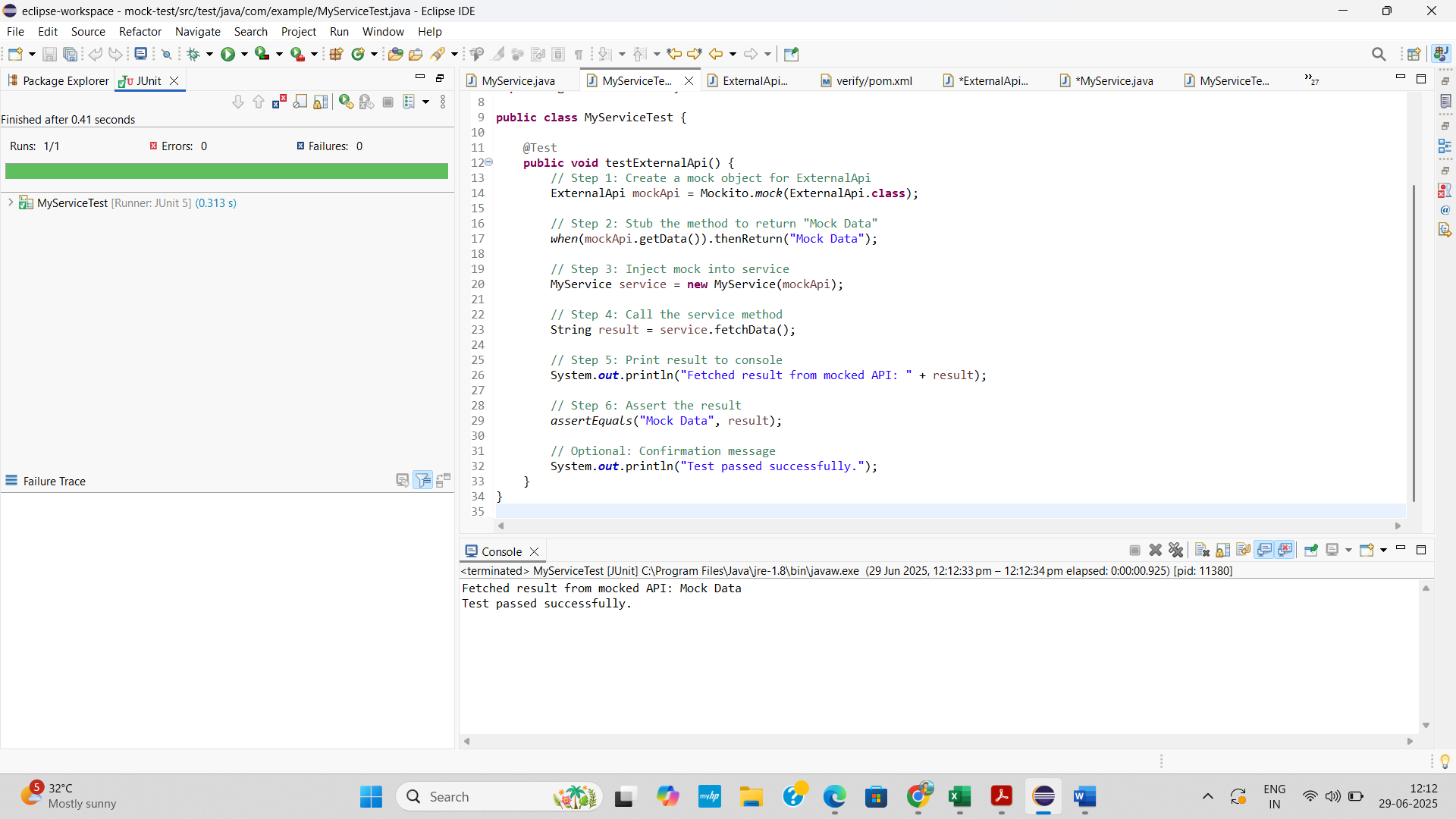
System.out.println("Test passed successfully.");

}

}

**OUTPUT**





**EXERCISE 2: VERIFYING INTERACTIONS**

**SCENARIO**

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

**CODE**

**ExternalApi.java**

package com.example;

public interface ExternalApi {

String getData();

}

**MyService.java**

package com.example;

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}}

**MyServiceTest.java**

package com.example;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

ExternalApi mockApi = mock(ExternalApi.class);

when(mockApi.getData()).thenReturn("Mocked Response");

MyService service = new MyService(mockApi);

String result = service.fetchData();

System.out.println("Fetched result: " + result);

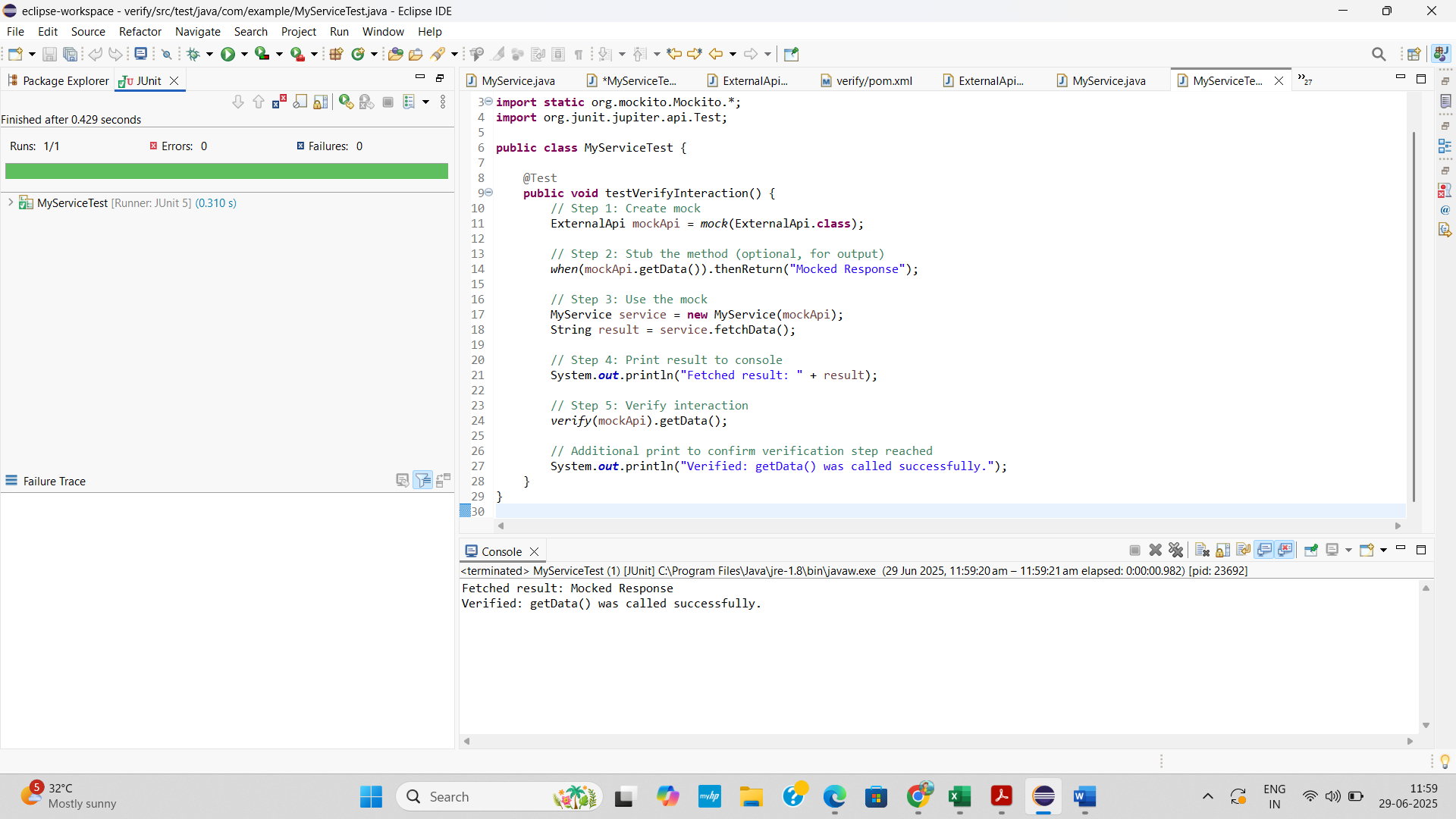
verify(mockApi).getData();

System.out.println("Verified: getData() was called successfully.");

}

}

**OUTPUT**

****

**SLF4J LOGGING FRAMEWORK- SL4J LOGGING EXERCISES**

**EXERCISE 1: LOGGING ERROR MESSAGES AND WARNING LEVELS**

**TASK**

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

.

**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) {

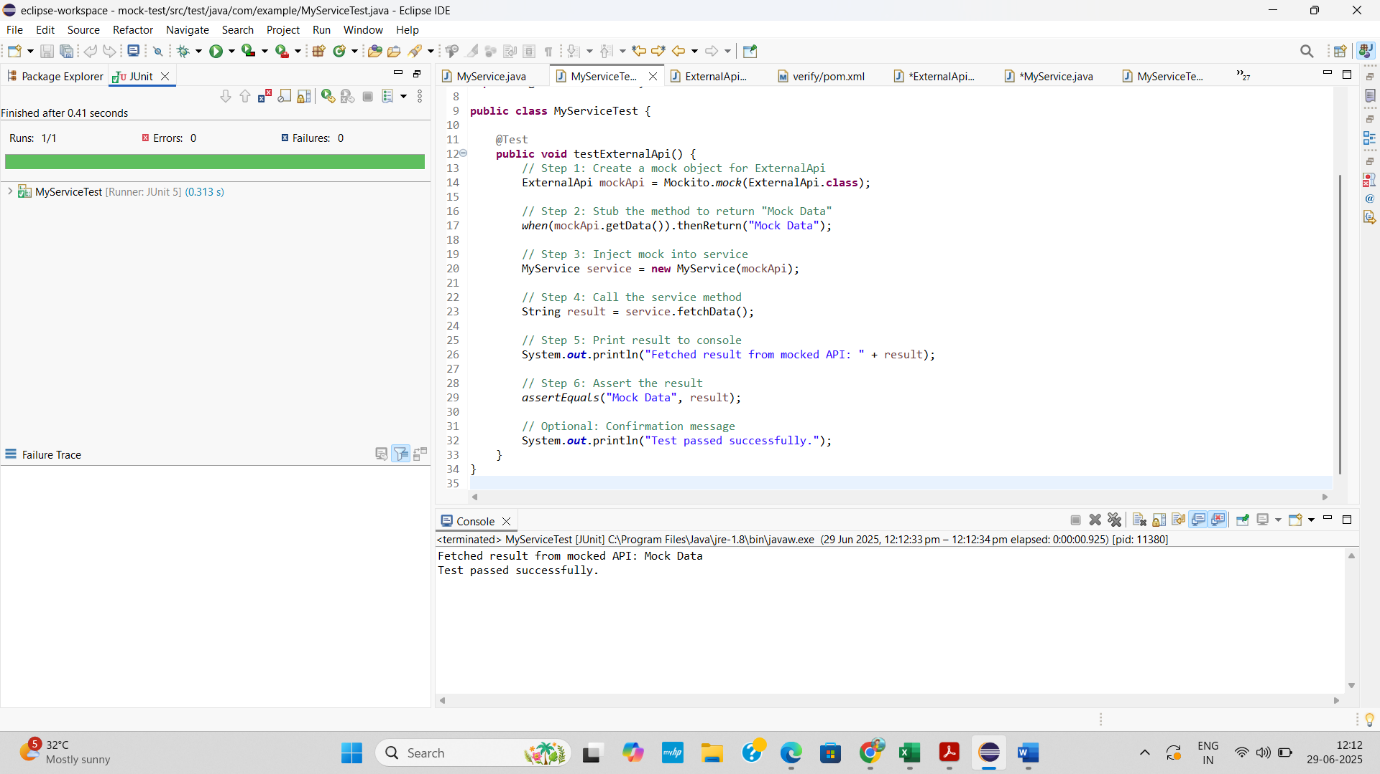
*logger*.error("This is an error");

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

}

}

**OUTPUT**

****