**WEEK -2**

**PL/SQL programming**

**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;

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE loans CASCADE CONSTRAINTS';

EXCEPTION WHEN OTHERS THEN

IF SQLCODE != -942 THEN RAISE; END IF;

END;

/

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE customers CASCADE CONSTRAINTS';

EXCEPTION WHEN OTHERS THEN

IF SQLCODE != -942 THEN RAISE; END IF;

END;

/

CREATE TABLE customers (

cust\_id NUMBER PRIMARY KEY,

age NUMBER(3),

balance NUMBER(12,2),

vip\_flag CHAR(1) DEFAULT 'N'

);

CREATE TABLE loans (

loan\_id NUMBER PRIMARY KEY,

cust\_id NUMBER NOT NULL

CONSTRAINT loans\_cust\_fk REFERENCES customers(cust\_id),

int\_rate NUMBER(5,2),

due\_on DATE

);

INSERT INTO customers VALUES (1, 65, 12000, DEFAULT);

INSERT INTO customers VALUES (2, 45, 8000, DEFAULT);

INSERT INTO customers VALUES (3, 70, 15000, DEFAULT);

INSERT INTO loans VALUES (101, 1, 10, TO\_DATE('04-JUL-2025','DD-MON-YYYY'));

INSERT INTO loans VALUES (102, 2, 9, TO\_DATE('01-SEP-2025','DD-MON-YYYY'));

INSERT INTO loans VALUES (103, 3, 8, TO\_DATE('29-JUN-2025','DD-MON-YYYY'));

COMMIT;

DECLARE

v\_today DATE := TRUNC(SYSDATE);

BEGIN

DBMS\_OUTPUT.PUT\_LINE(' Scenario 1 – 1 % discount for seniors');

FOR s1 IN (

SELECT l.loan\_id, l.int\_rate, c.cust\_id

FROM loans l

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

WHERE c.age > 60

) LOOP

UPDATE loans

SET int\_rate = ROUND(int\_rate \* 0.99, 2)

WHERE loan\_id = s1.loan\_id;

DBMS\_OUTPUT.PUT\_LINE(

' Loan '||s1.loan\_id||' → new rate '

||TO\_CHAR(ROUND(s1.int\_rate\*0.99,2),'FM99990.99')||'%');

END LOOP;

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

DBMS\_OUTPUT.PUT\_LINE(' Scenario 2 – VIP customers');

FOR s2 IN (

SELECT cust\_id, balance

FROM customers

WHERE balance > 10000

) LOOP

UPDATE customers

SET vip\_flag = 'Y'

WHERE cust\_id = s2.cust\_id;

DBMS\_OUTPUT.PUT\_LINE(

' Customer '||s2.cust\_id||' → VIP (balance $'

||TO\_CHAR(s2.balance,'FM999G999D00')||')');

END LOOP;

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

DBMS\_OUTPUT.PUT\_LINE(' Scenario 3 – Loans due in next 30 days');

FOR s3 IN (

SELECT loan\_id, cust\_id, due\_on

FROM loans

WHERE due\_on BETWEEN v\_today AND v\_today + 30

ORDER BY due\_on

) LOOP

DBMS\_OUTPUT.PUT\_LINE(

' Loan '||s3.loan\_id

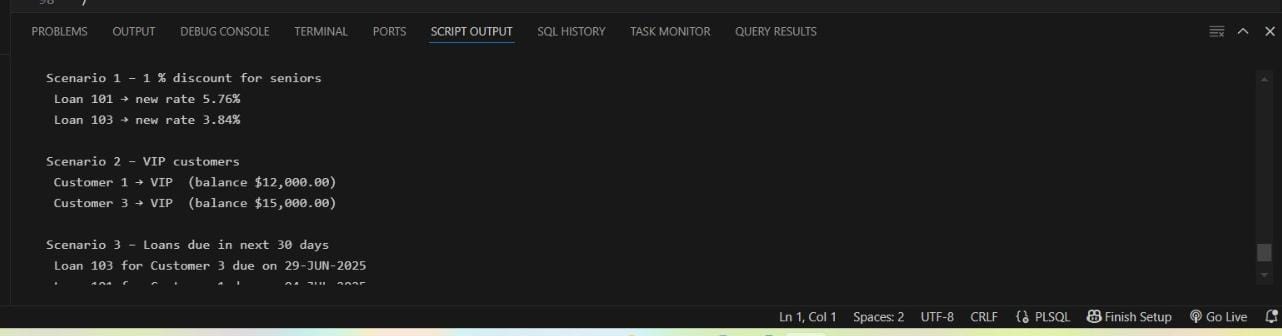
||' for Customer '||s3.cust\_id

||' due on '||TO\_CHAR(s3.due\_on,'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

DECLARE

  PROCEDURE zap(p VARCHAR2) IS

  BEGIN EXECUTE IMMEDIATE 'DROP PROCEDURE '||p;

  EXCEPTION WHEN OTHERS THEN IF SQLCODE != -4043 THEN NULL; END IF; END;

BEGIN

  zap('PROCESSMONTHLYINTEREST');

  zap('UPDATEEMPLOYEEBONUS');

  zap('TRANSFERFUNDS');

END;

/

BEGIN EXECUTE IMMEDIATE 'DROP TABLE accounts   CASCADE CONSTRAINTS';

EXCEPTION WHEN OTHERS THEN IF SQLCODE != -942 THEN NULL; END IF; END;

/

BEGIN EXECUTE IMMEDIATE 'DROP TABLE employees  CASCADE CONSTRAINTS';

EXCEPTION WHEN OTHERS THEN IF SQLCODE != -942 THEN NULL; END IF; END;

/

BEGIN EXECUTE IMMEDIATE 'DROP TABLE departments CASCADE CONSTRAINTS';

EXCEPTION WHEN OTHERS THEN IF SQLCODE != -942 THEN NULL; END IF; END;

/

CREATE TABLE accounts (

  account\_id   NUMBER PRIMARY KEY,

  balance      NUMBER(14,2),

  account\_type VARCHAR2(10)

);

CREATE TABLE departments (

  dept\_id   NUMBER PRIMARY KEY,

  dept\_name VARCHAR2(30)

);

CREATE TABLE employees (

  emp\_id   NUMBER PRIMARY KEY,

  emp\_name VARCHAR2(30),

  dept\_id  NUMBER REFERENCES departments(dept\_id),

  salary   NUMBER(14,2)

);

INSERT INTO accounts VALUES (101, 1200, 'SAVINGS');

INSERT INTO accounts VALUES (102, 3000, 'CURRENT');

INSERT INTO accounts VALUES (103, 5000, 'SAVINGS');

INSERT INTO departments VALUES (10,'IT');

INSERT INTO departments VALUES (20,'Finance');

INSERT INTO employees VALUES (1001,'Asha',   10,60000);

INSERT INTO employees VALUES (1002,'Bhavesh',10,55000);

INSERT INTO employees VALUES (1003,'Chetan', 20,52000);

COMMIT;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

  FOR r IN (SELECT account\_id, balance

            FROM   accounts

            WHERE  account\_type = 'SAVINGS') LOOP

    UPDATE accounts

    SET    balance = ROUND(balance\*1.01,2)

    WHERE  account\_id = r.account\_id;

    DBMS\_OUTPUT.PUT\_LINE(

      '  Account '||r.account\_id||

      ' new balance ₹'||

      TO\_CHAR(ROUND(r.balance\*1.01,2),'FM999G999D00'));

  END LOOP;

END;

/

SHOW ERRORS

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

  p\_dept\_id    IN NUMBER,

  p\_bonus\_pct  IN NUMBER) IS

  v\_new NUMBER;

  v\_rows PLS\_INTEGER := 0;

BEGIN

  FOR r IN (SELECT emp\_id, salary FROM employees

            WHERE dept\_id = p\_dept\_id) LOOP

    v\_new := ROUND(r.salary\*(1+p\_bonus\_pct/100),2);

    UPDATE employees SET salary = v\_new WHERE emp\_id = r.emp\_id;

    DBMS\_OUTPUT.PUT\_LINE(

      ' Emp '||r.emp\_id||' new salary ₹'||

      TO\_CHAR(v\_new,'FM999G999D00'));

    v\_rows := v\_rows+1;

  END LOOP;

  IF v\_rows = 0 THEN

    DBMS\_OUTPUT.PUT\_LINE(

      '  No employees in department '||p\_dept\_id);

  END IF;

END;

/

SHOW ERRORS

CREATE OR REPLACE PROCEDURE TransferFunds(

  p\_src\_id IN NUMBER,

  p\_dst\_id IN NUMBER,

  p\_amt    IN NUMBER) IS

  v\_src\_bal NUMBER;

  v\_dummy   NUMBER;

BEGIN

  BEGIN

    SELECT balance INTO v\_src\_bal

    FROM   accounts

    WHERE  account\_id = p\_src\_id

    FOR UPDATE;

  EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

      DBMS\_OUTPUT.PUT\_LINE('  Source account '||p\_src\_id||' not found.'); RETURN;

  END;

  BEGIN

    SELECT 1 INTO v\_dummy

    FROM   accounts

    WHERE  account\_id = p\_dst\_id

    FOR UPDATE;

  EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

      DBMS\_OUTPUT.PUT\_LINE('  Destination account '||p\_dst\_id||' not found.'); RETURN;

  END;

  IF v\_src\_bal < p\_amt THEN

    DBMS\_OUTPUT.PUT\_LINE('  Insufficient funds in account '||p\_src\_id); RETURN;

  END IF;

  UPDATE accounts SET balance = balance - p\_amt WHERE account\_id = p\_src\_id;

  UPDATE accounts SET balance = balance + p\_amt WHERE account\_id = p\_dst\_id;

  DBMS\_OUTPUT.PUT\_LINE('  ₹'||p\_amt||

                       ' transferred from '||p\_src\_id||' → '||p\_dst\_id);

END;

/

SHOW ERRORS

BEGIN

  DBMS\_OUTPUT.PUT\_LINE('! Scenario 1 : Monthly Interest ');

  ProcessMonthlyInterest;

  DBMS\_OUTPUT.PUT\_LINE('');

  DBMS\_OUTPUT.PUT\_LINE('! Scenario 2 : 10 % Bonus to dept 10 ');

  UpdateEmployeeBonus(10,10);

  DBMS\_OUTPUT.PUT\_LINE('');

  DBMS\_OUTPUT.PUT\_LINE('! Scenario 3 : Transfer ₹2 000 (103 → 102) ');

  TransferFunds(103,102,2000);

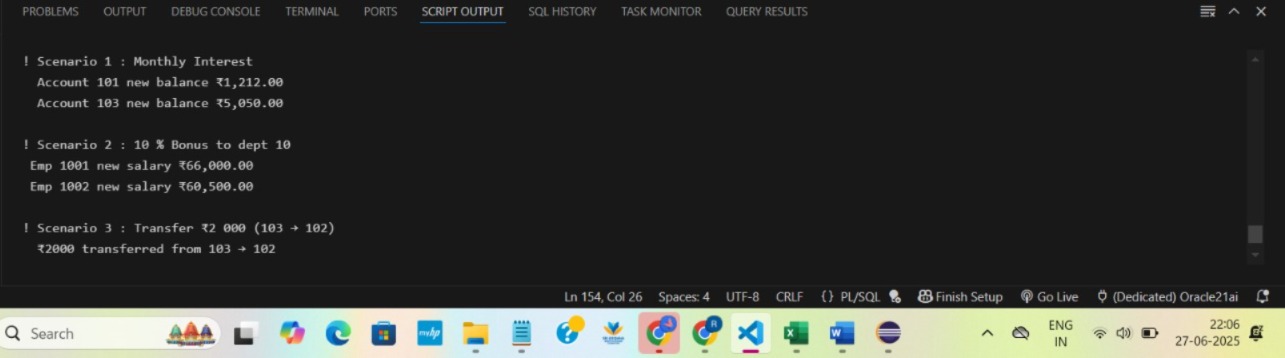
  DBMS\_OUTPUT.PUT\_LINE('');

  COMMIT;

END;

/

**OUTPUT:**

****

**TDD using JUnit5 and Mockito**

**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>

3. Create a new test class in your project.

**CODE:**

**Calculator.java:**

**package** com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

**CalculatorTest.java:**

package com.example;

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

import org.junit.Test;

public class CalculatorTest {

@Test

public void testAdd() {

Calculator calc = new Calculator();

int result = calc.add(5, 3);

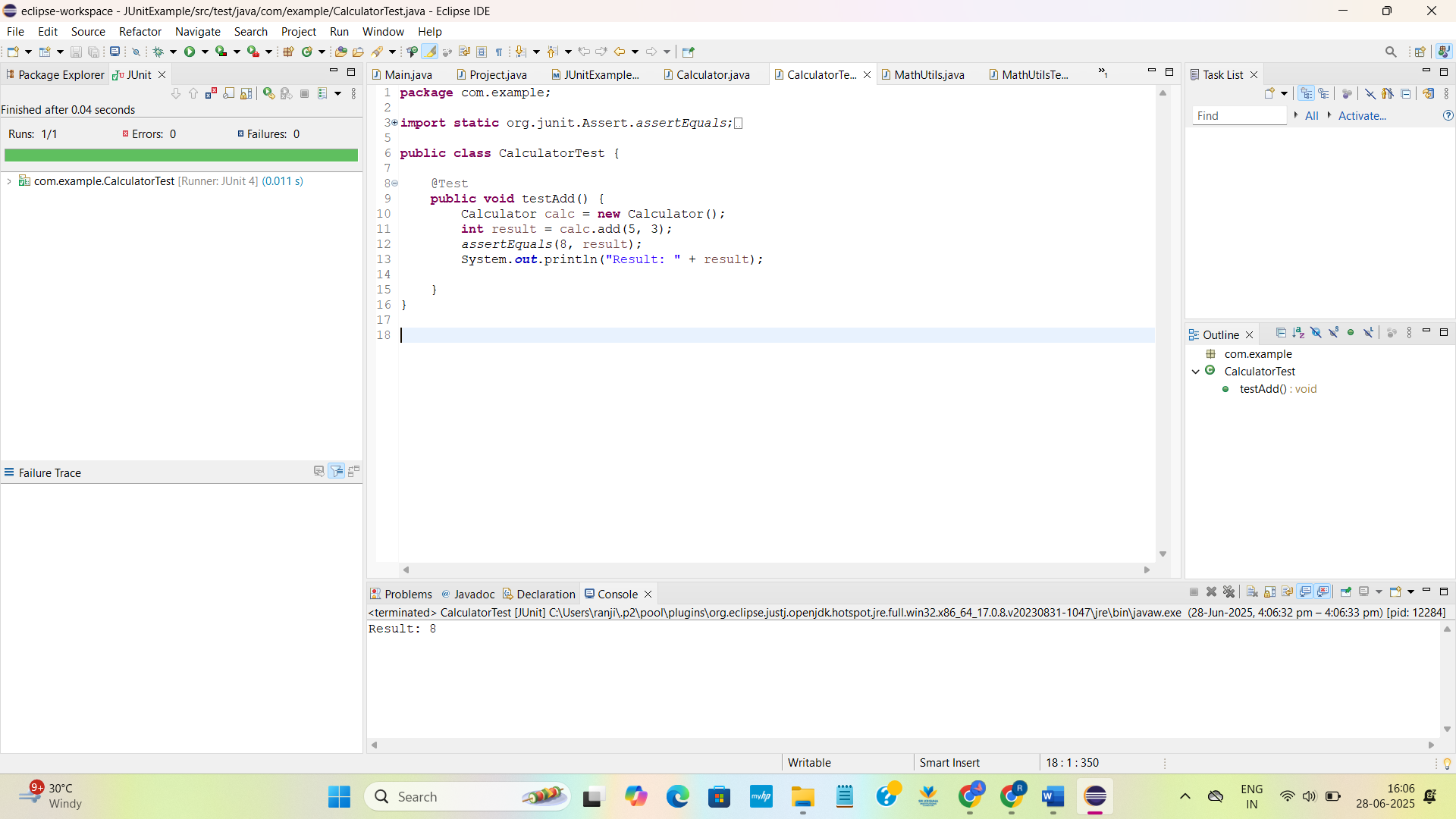
*assertEquals*(8, result);

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

}

}

**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:**

**AssertionsTest.java:**

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

@Test

public void testAssertions() {

assertEquals(5, 2 + 3);

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

assertTrue(5 > 3);

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

assertFalse(5 < 3);

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

assertNull(null);

System.out.println("assertNull successfullypassed ");

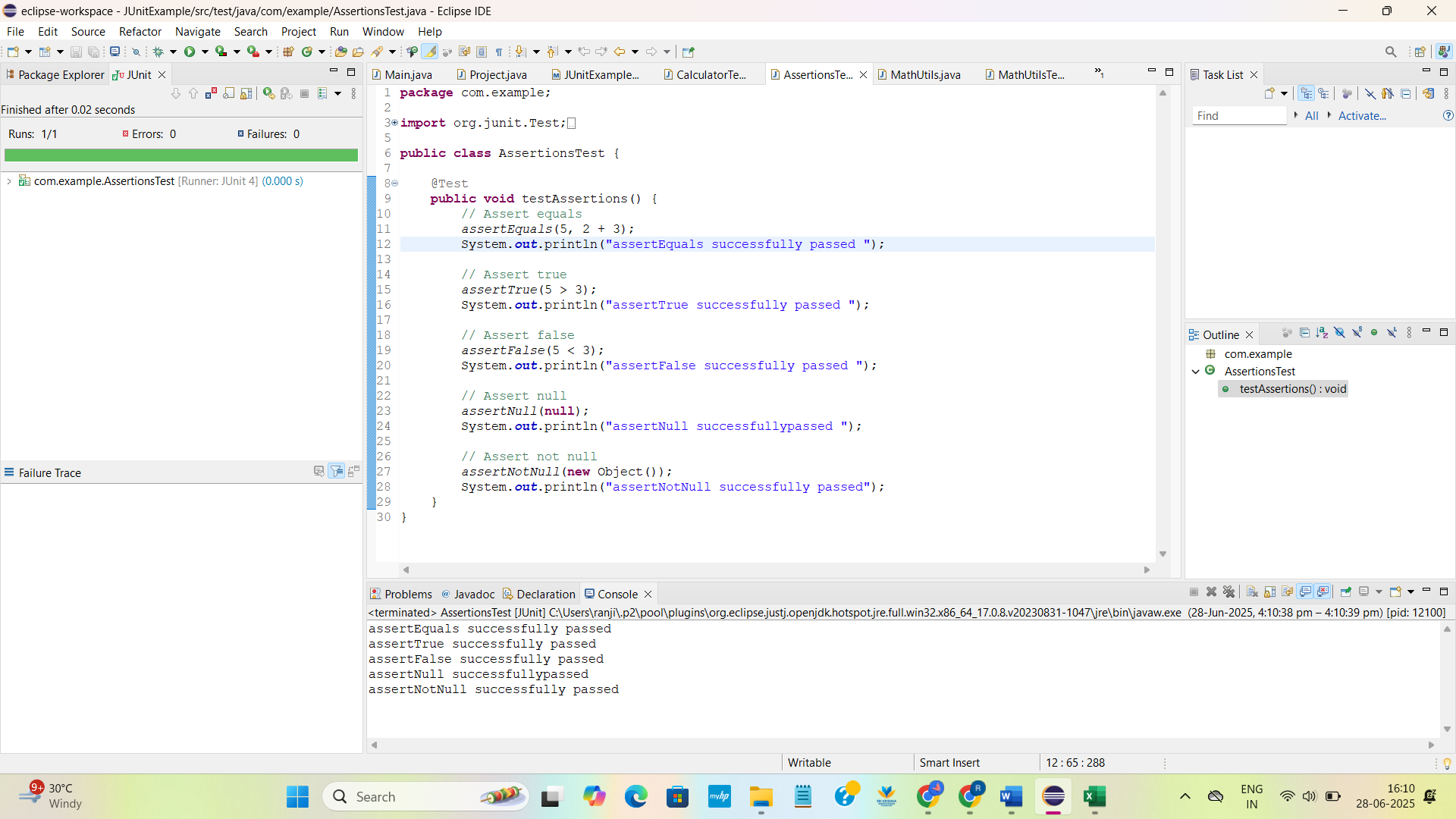
assertNotNull(new Object());

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

}

}

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:**

**MathUtils .java:**

package com.example;

public class MathUtils {

public int multiply(int a, int b) {

return a \* b;

}

}

**MathUtilsTest.java:**

package com.example;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.\*;

public class MathUtilsTest {

MathUtils mathUtils;

@Before

public void setUp() {

mathUtils = new MathUtils();

System.out.println("Setup done.");

}

@After

public void tearDown()

System.out.println("Teardown done.");

}

@Test

public void testMultiply() {

int a = 4;

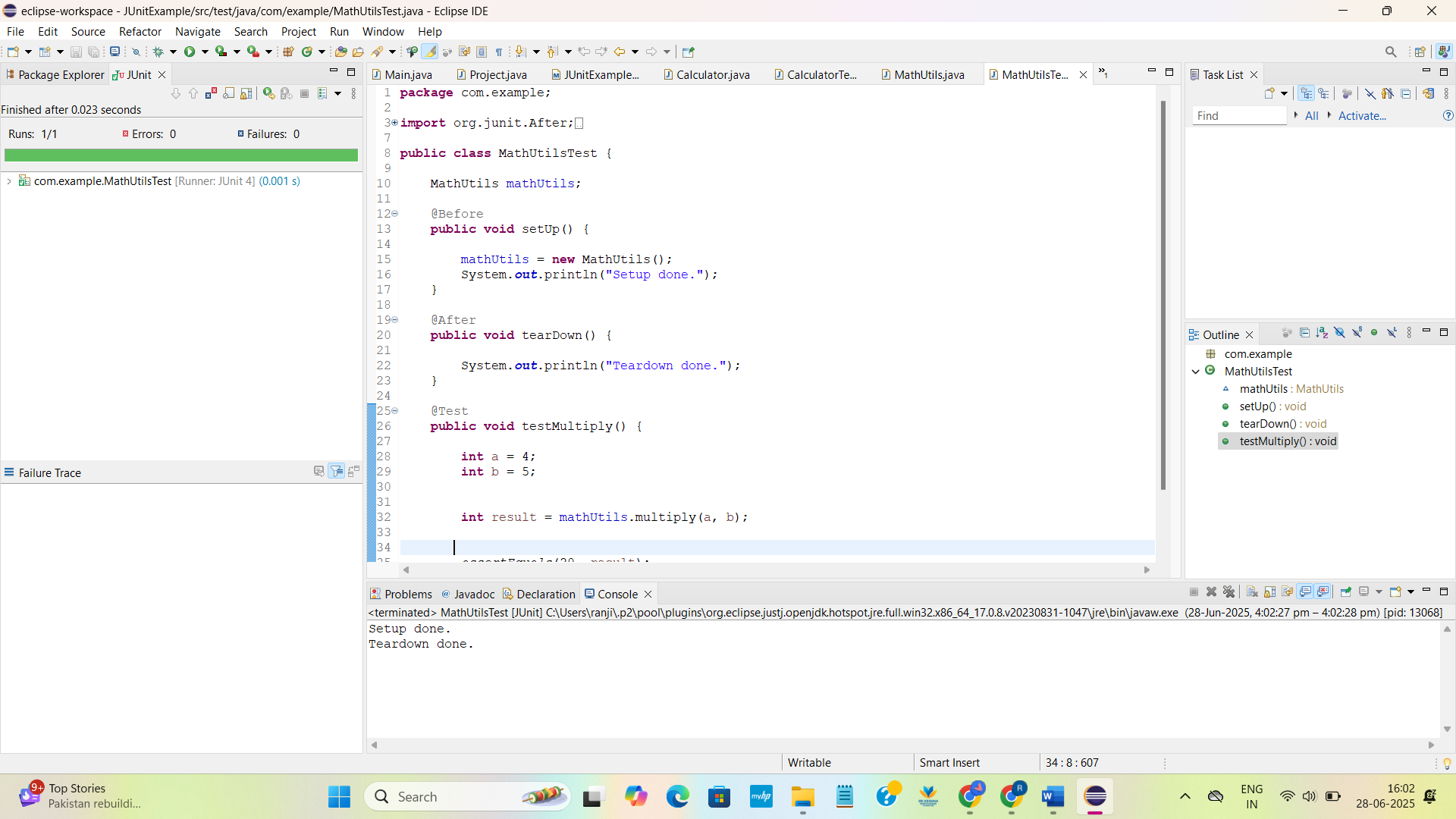
int b = 5;

int result = mathUtils.multiply(a, b);

assertEquals(20, result);

}}

**OUTPUT:**



**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:**

package com.examplee;

public interface ExternalApi {

String getData();

}

**MyService.java:**

package com.examplee;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java:**

package com.examplee;

import org.junit.jupiter.api.Test;

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

import static org.mockito.Mockito.\*;

public class MyServiceTest {

@Test

public void testExternalApi() {

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

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

MyService service = new MyService(mockApi);

String result = service.fetchData();

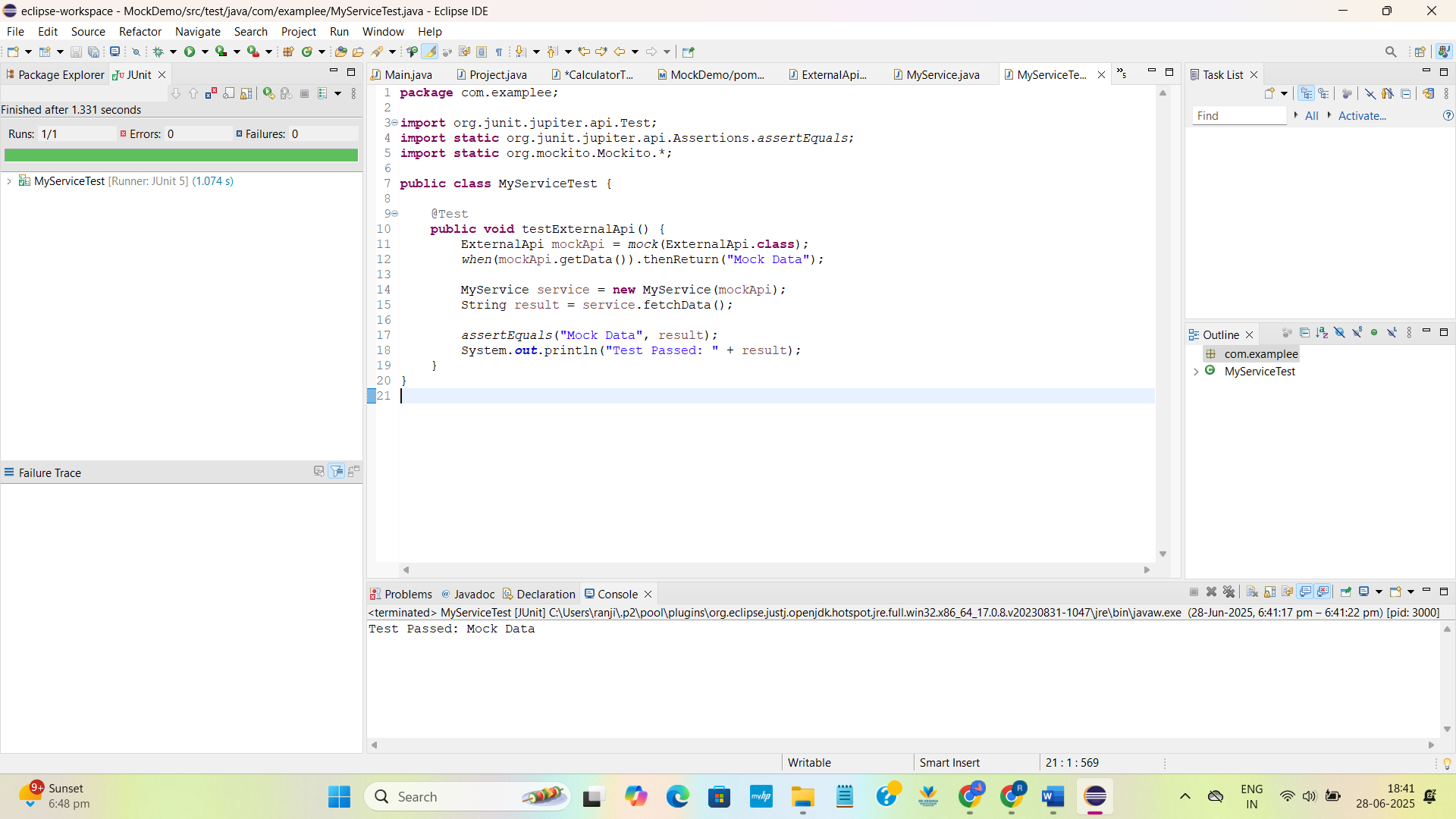
*assertEquals*("Mock Data", result);

System.*out*.println("Test Passed: " + result);

}

}

**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:**

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 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();

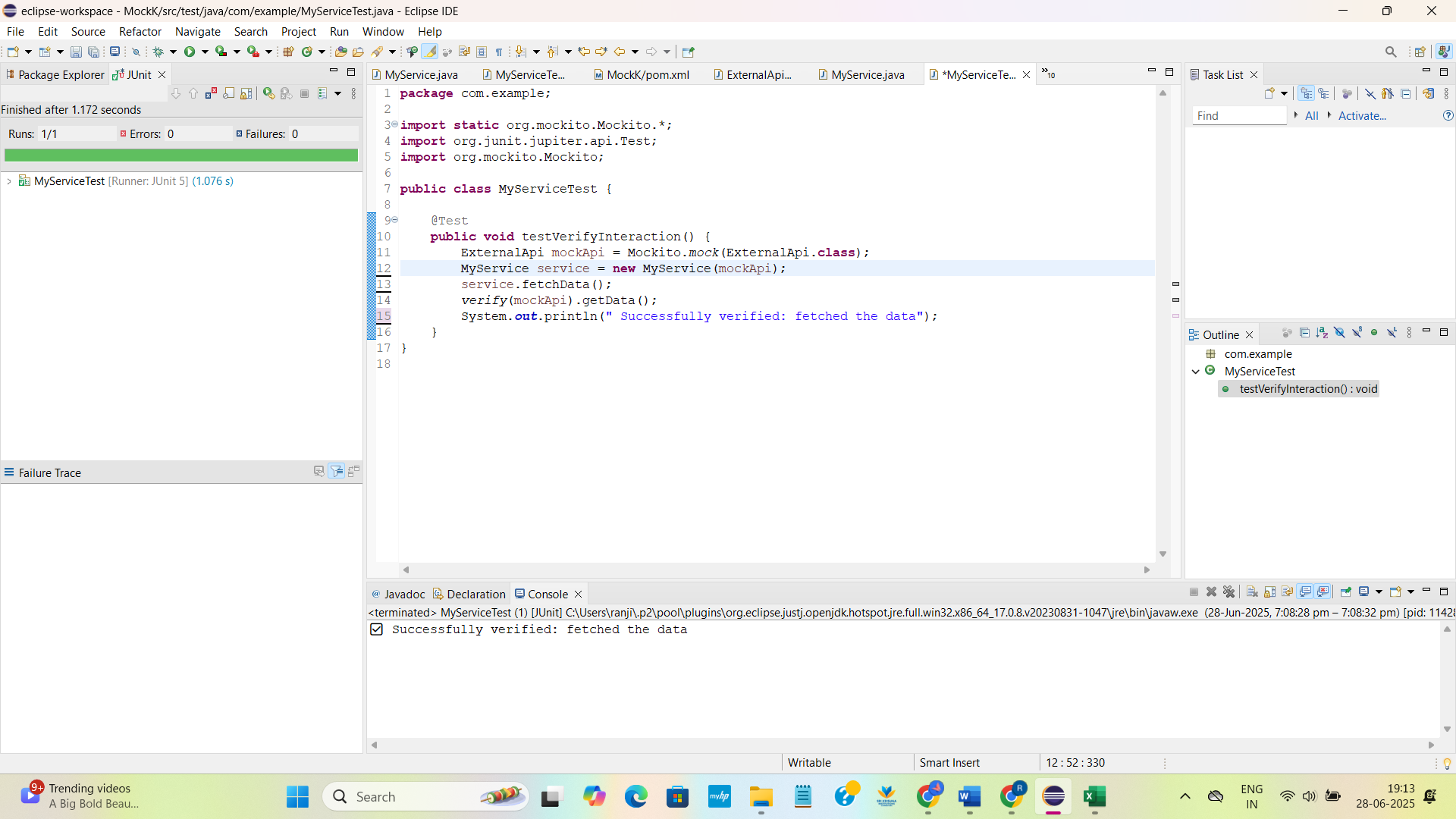
*verify*(mockApi).getData();

System.*out*.println(" Successfully verified: fetched the data");

}

}

**OUTPUT:**



**SLF4J logging framework**

**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>

2. 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:**

