**DEEP NURTURE 4.0----WEEK-2**

**2.Pl/SQL Programming:**

**Exercise 1: Control Structures:**

**Accounts table:**

SELECT \* FROM user\_tables WHERE table\_name = 'ACCOUNTS';

**Testing-Table:**

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

Balance NUMBER(10, 2),

AccountType VARCHAR2(20),

LastModified DATE

);

**Procedure:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

v\_AccountID Accounts.AccountID%TYPE;

v\_Balance Accounts.Balance%TYPE;

BEGIN

FOR acc IN (

SELECT AccountID, Balance

FROM Accounts

WHERE AccountType = 'Savings'

)

LOOP

v\_AccountID := acc.AccountID;

v\_Balance := acc.Balance;

UPDATE Accounts

SET Balance = v\_Balance + (v\_Balance \* 0.01),

LastModified = SYSDATE

WHERE AccountID = v\_AccountID;

DBMS\_OUTPUT.PUT\_LINE('Interest applied to Account ID ' || v\_AccountID);

END LOOP;

END;

/

**Run the Procedure:**SET SERVEROUTPUT ON;

EXEC ProcessMonthlyInterest;

**OUTPUT:**

****

**Exercise 3: Stored Procedures**

**Replace Procedure:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR acc IN (

SELECT AccountID, Balance

FROM Accounts

WHERE AccountType = 'Savings'

)

LOOP

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01),

LastModified = SYSDATE

WHERE AccountID = acc.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Interest applied to Account ID ' || acc.AccountID);

END LOOP;

END;

/

BEGIN

ProcessMonthlyInterest;

END;

/

**Employees table:**

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Department VARCHAR2(50),

Salary NUMBER(10,2)

);  
**Data:**

INSERT INTO Employees VALUES (1, 'John', 'HR', 50000);

INSERT INTO Employees VALUES (2, 'Jane', 'HR', 60000);

INSERT INTO Employees VALUES (3, 'Mike', 'IT', 70000);

COMMIT;

**UpdateEmployeeBonus:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_Department IN VARCHAR2,

p\_BonusPercent IN NUMBER

) IS

v\_EmpID Employees.EmployeeID%TYPE;

v\_Salary Employees.Salary%TYPE;

BEGIN

FOR emp IN (

SELECT EmployeeID, Salary

FROM Employees

WHERE Department = p\_Department

)

LOOP

v\_EmpID := emp.EmployeeID;

v\_Salary := emp.Salary;

UPDATE Employees

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

WHERE EmployeeID = v\_EmpID;

DBMS\_OUTPUT.PUT\_LINE('Bonus applied to Employee ID ' || v\_EmpID);

END LOOP;

END;

/

BEGIN

UpdateEmployeeBonus('IT', 10);

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 balance in source account.');

END IF;

UPDATE Accounts

SET Balance = Balance - p\_Amount,

LastModified = SYSDATE

WHERE AccountID = p\_FromAccountID;

UPDATE Accounts

SET Balance = Balance + p\_Amount,

LastModified = SYSDATE

WHERE AccountID = p\_ToAccountID;

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

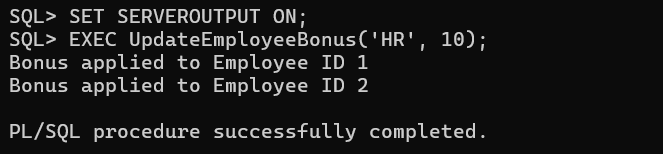
' from Account ID ' || p\_FromAccountID ||

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

END;

/

**OUTPUT:**



**2.JUnit\_Basic Testing Exercises:**

**Exercise 1: Setting Up Junit:**

**Pom.xml:**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>junit-example</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**Calculator.java:**

package com.java;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int divide(int a, int b) {

if (b == 0) {

throw new IllegalArgumentException("Cannot divide by zero");

}

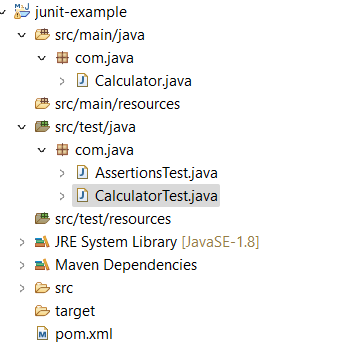
return a / b;

}

}

**CalculatorTest.java:**

Test class created

****

**Exercise 3: Assertions in Junit:**

**Assertiontest.java:**

package com.java;

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

@Test

public void testAssertions() {

assertEquals(5, 2 + 3);

assertTrue(5 > 3);

assertFalse(5 < 3);

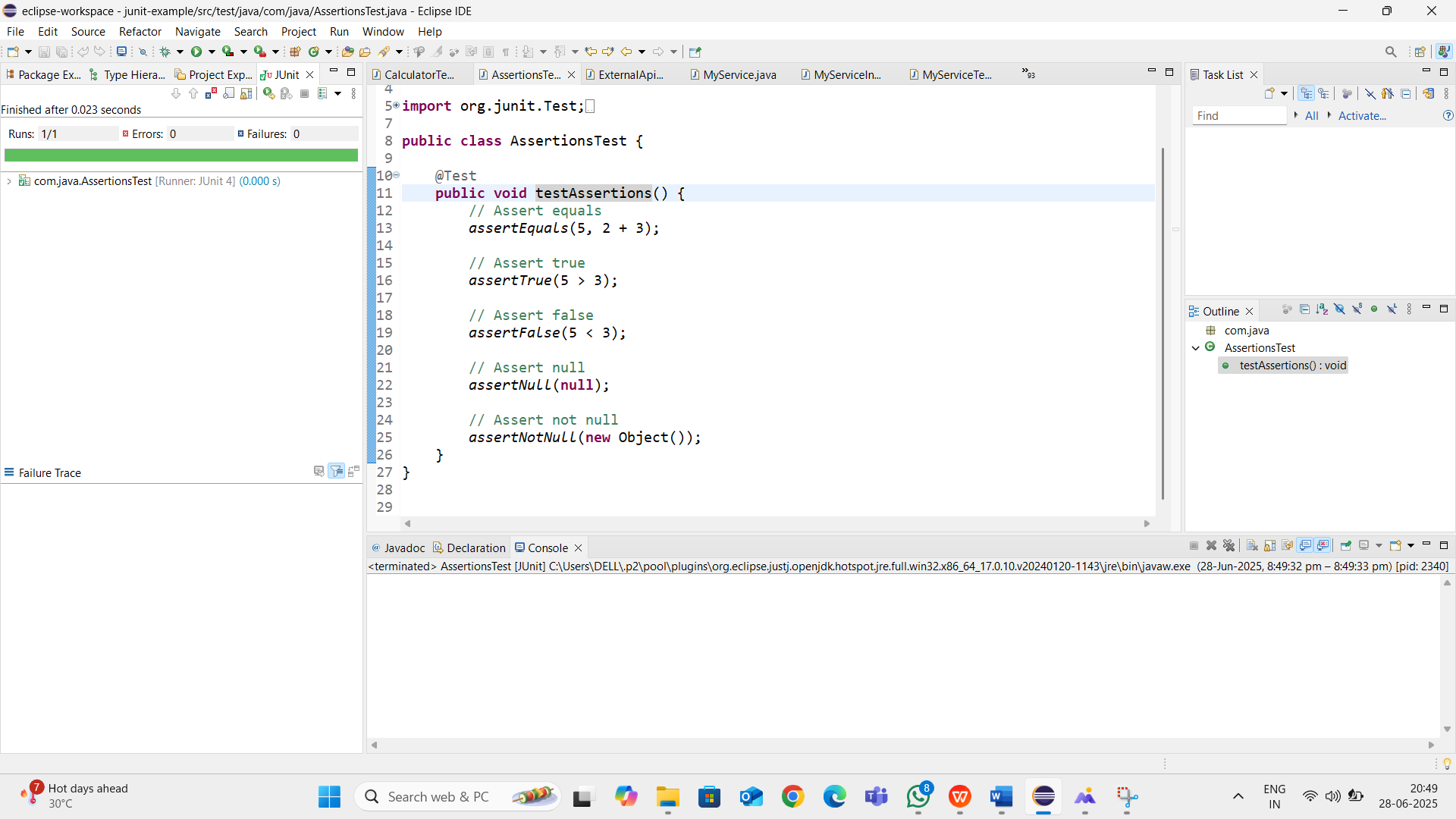
assertNull(null);

assertNotNull(new Object());

}

}

**Output:**



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

**CalculatorTest.java:**

package com.java;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

private Calculator calculator;

@Before

public void setUp() {

calculator = new Calculator(); // Arrange

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

}

@After

public void tearDown() {

calculator = null;

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

@Test

public void testAdd() {

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

assertEquals(5, result);

}

@Test

public void testDivide() {

int result = calculator.divide(10, 2);

assertEquals(5, result);

}

@Test(expected = IllegalArgumentException.class)

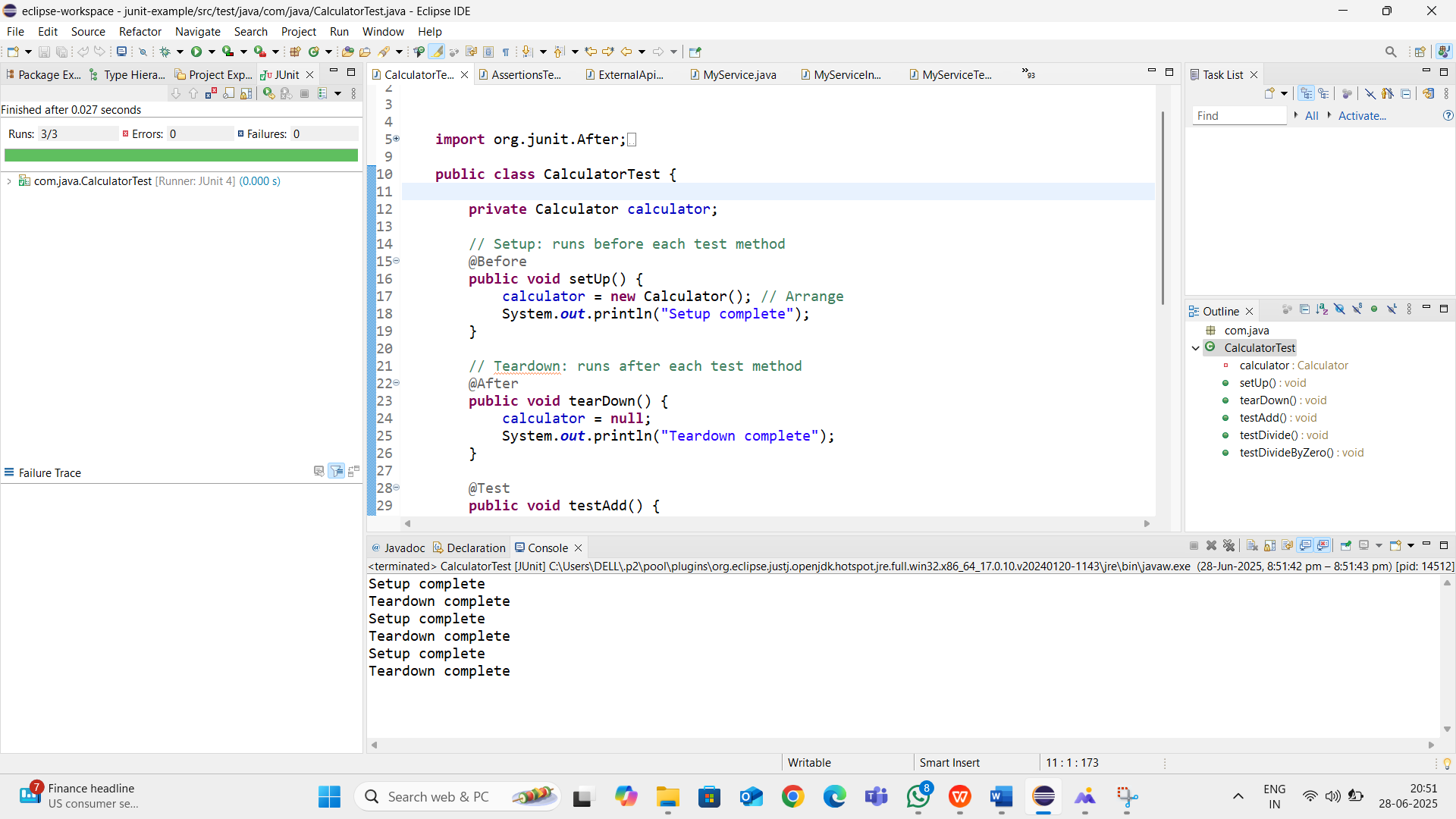
public void testDivideByZero() {

calculator.divide(10, 0);

}

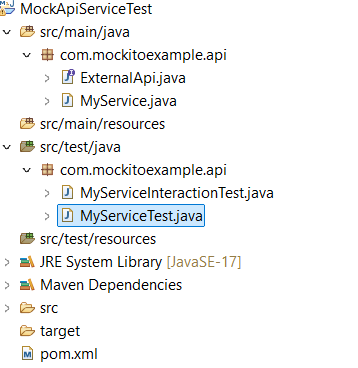
}

**OUTPUT:**



**3. Mockito exercises:**

**Exercise 1: Mocking and Stubbing:**



**Interfcae:**

**ExternalApi:**

package com.mockitoexample.api;

public interface ExternalApi {

String getData();

}

**MyService.java:**

package com.mockitoexample.api;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java:**

package com.mockitoexample.api;

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);

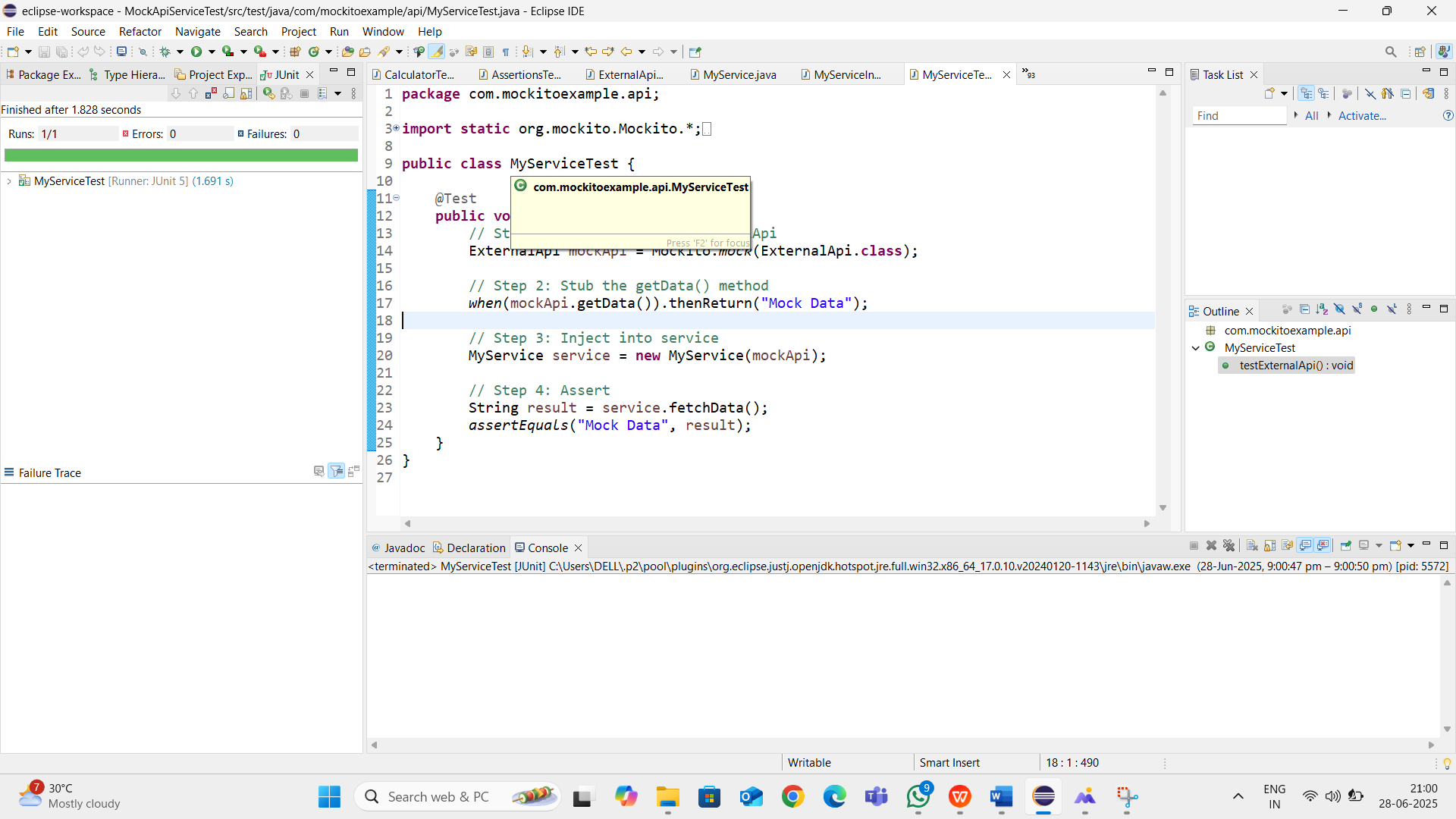
String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**OUTPUT:**



**Exercise 2: Verifying Interactions:**

**MyServiceInteractionTest.java:**

package com.mockitoexample.api;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceInteractionTest {

@Test

public void testVerifyInteraction() {

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

MyService service = new MyService(mockApi);

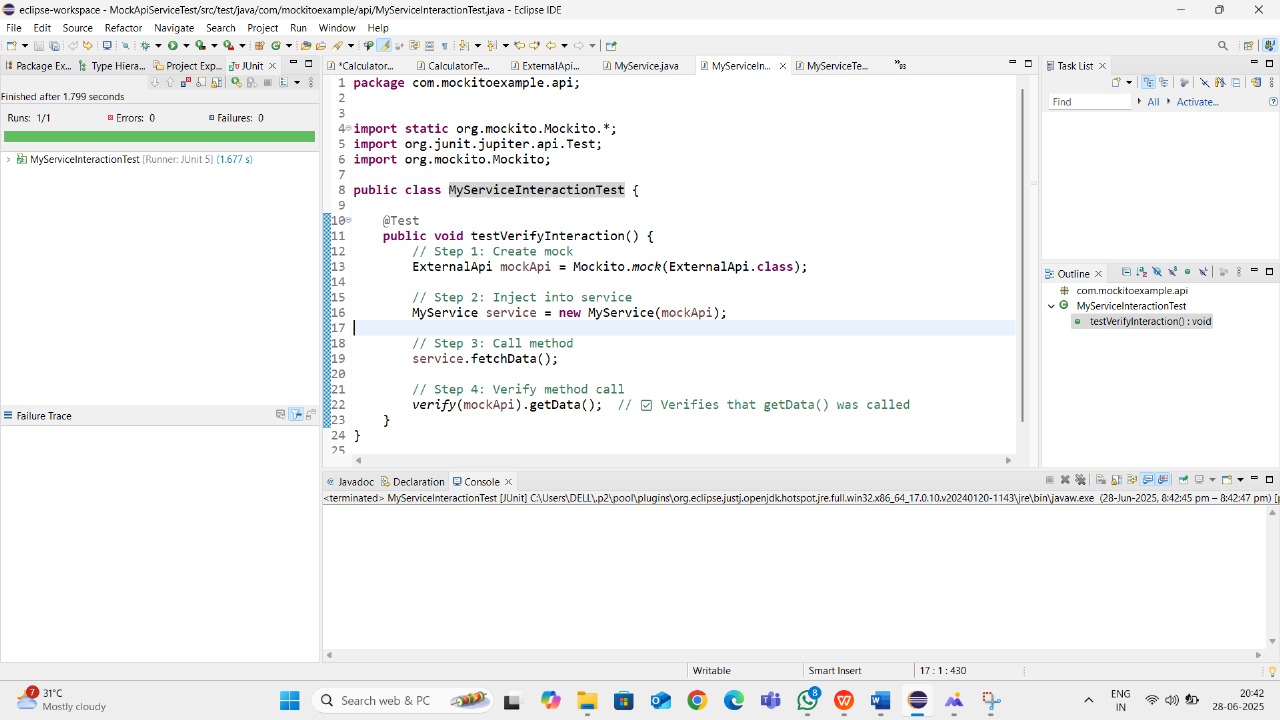
service.fetchData();

verify(mockApi).getData();

}

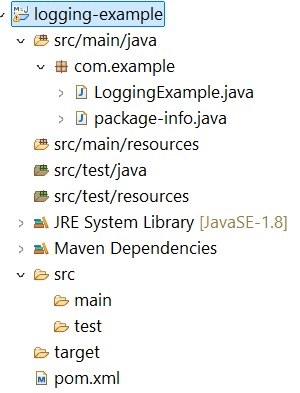
}

**Output:**



**6. SL4J Logging exercises:**

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



**Logging.Example.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:**

