**WEEK - 2**

**PL SQL Programming Hands On**

**Exercise 1: Control Structures**

**Scenario 1:**

**Code:**

SET SERVEROUTPUT ON;

DECLARE

v\_customer\_id Customers.CustomerID%TYPE;

v\_dob Customers.DOB%TYPE;

v\_age NUMBER;

BEGIN

FOR cust IN (

SELECT CustomerID, DOB FROM Customers

) LOOP

v\_customer\_id := cust.CustomerID;

v\_dob := cust.DOB;

v\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, v\_dob) / 12);

IF v\_age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = v\_customer\_id;

DBMS\_OUTPUT.PUT\_LINE('1% interest discount applied for customer ' || v\_customer\_id || ' (age: ' || v\_age || ')');

END IF;

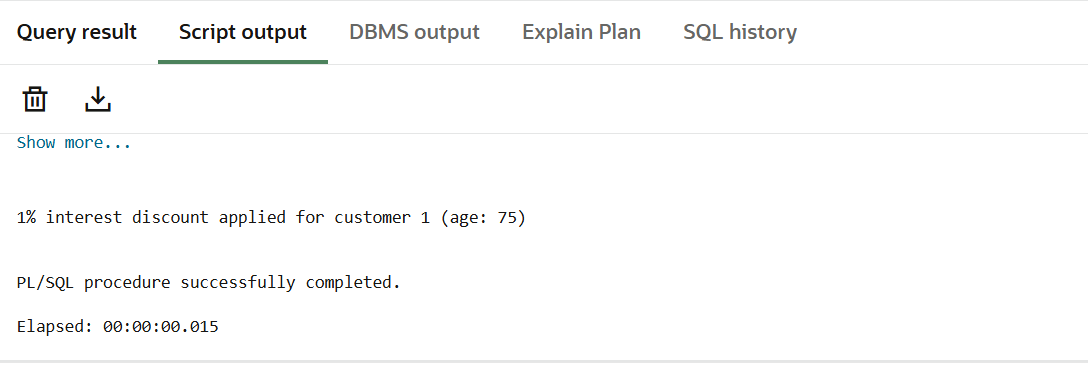
END LOOP;

COMMIT;

END;

/

**Output:**



**Scenario 2:**

**Code:**

SET SERVEROUTPUT ON;

BEGIN

FOR cust IN (

SELECT CustomerID, Balance FROM Customers

) LOOP

IF cust.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 1

WHERE CustomerID = cust.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Customer ' || cust.CustomerID || ' promoted to VIP.');

END IF;

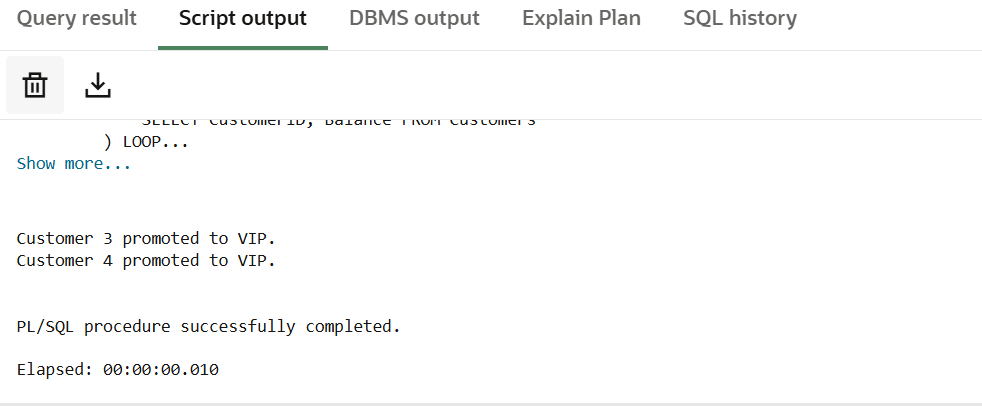
END LOOP;

COMMIT;

END;

/

**Output:**



**Scenario 3:**

**Code:**

SET SERVEROUTPUT ON;

BEGIN

FOR loan\_rec IN (

SELECT l.loanid, l.customerid, l.loanamount, l.enddate

FROM LOANS l

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

)

LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Dear Customer' ||

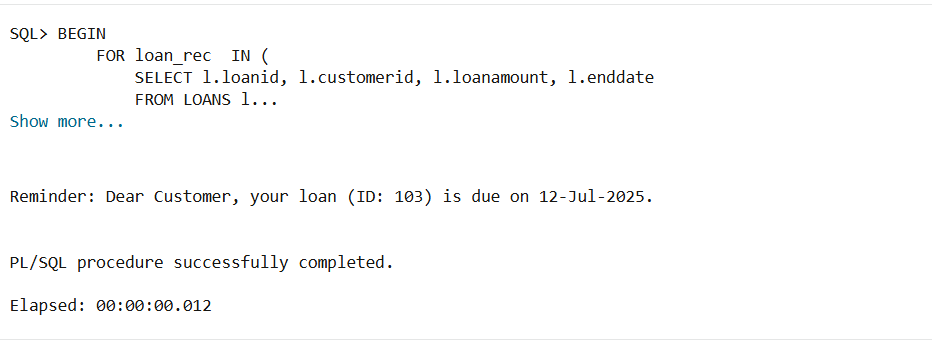
', your loan (ID: ' || loan\_rec.LoanID || ') is due on ' || TO\_CHAR(loan\_rec.EndDate, 'DD-Mon-YYYY') || '.');

END LOOP;

END;

/

**Output:**



**Exercise 3: Stored Procedures**

**Scenario 1:**

**Code:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

BEGIN

UPDATE ACCOUNTS

SET BALANCE=BALANCE + BALANCE\*0.1,

lastmodified=sysdate

WHERE ACCOUNTTYPE='Savings';

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Monthly interest processed for all savings accounts.');

END;

/

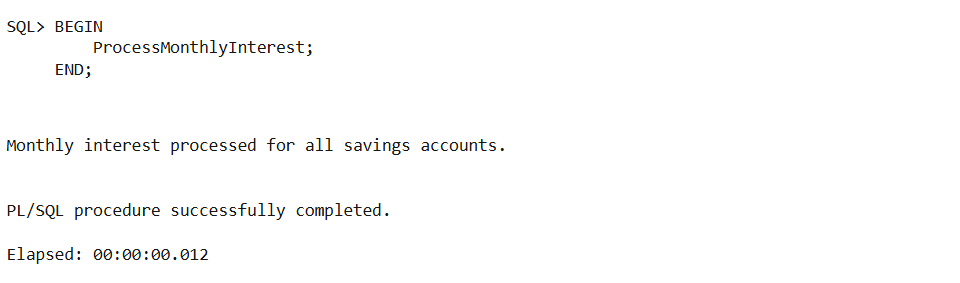
BEGIN

ProcessMonthlyInterest;

END;

/

**Output:**



**Scenario 2:**

**Code**:

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_dept IN employees.department%type,

p\_bonus IN NUMBER

)

IS

BEGIN

UPDATE EMPLOYEES

SET SALARY=SALARY + SALARY\*p\_bonus

WHERE DEPARTMENT=p\_dept;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Bonus of ' || p\_bonus || '% applied to department: ' || p\_dept);

END;

/

SET SERVEROUTPUT ON;

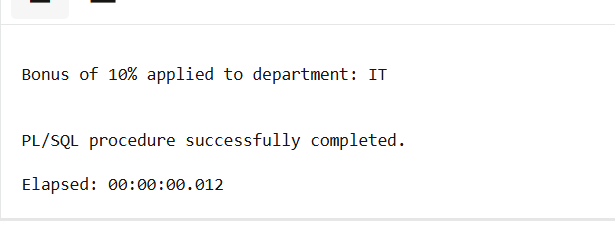
BEGIN

UpdateEmployeeBonus('IT', 10); -- Gives 10% bonus to IT department

END;

/

**Output:**



**Scenario 3:**

**Code:**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

)

IS

v\_balance NUMBER;

BEGIN

-- Get and lock source account

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_from\_account\_id

FOR UPDATE;

IF v\_balance < p\_amount THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient balance in source account.');

RETURN;

END IF;

-- Deduct from source account

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_from\_account\_id;

-- Credit to destination account

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_to\_account\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer of ₹' || p\_amount ||

' from Account ' || p\_from\_account\_id ||

' to Account ' || p\_to\_account\_id || ' successful.');

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: One or both account IDs not found.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Unexpected error: ' || SQLERRM);

ROLLBACK;

END;

/

SET SERVEROUTPUT ON;

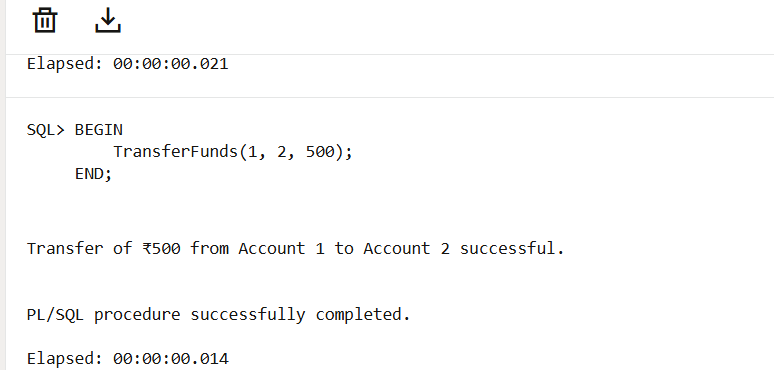
BEGIN

TransferFunds(1, 2, 500);

END;

/

**Output**:



**TDD using JUnit and Mockito, SLF4J Logging Hands- On**

**J Unit Testing Exercises:**

**Exercise 1: Setting Up JUnit**

Code:

**package** com.example.test;

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

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

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

@Test

**void** testAddition() {

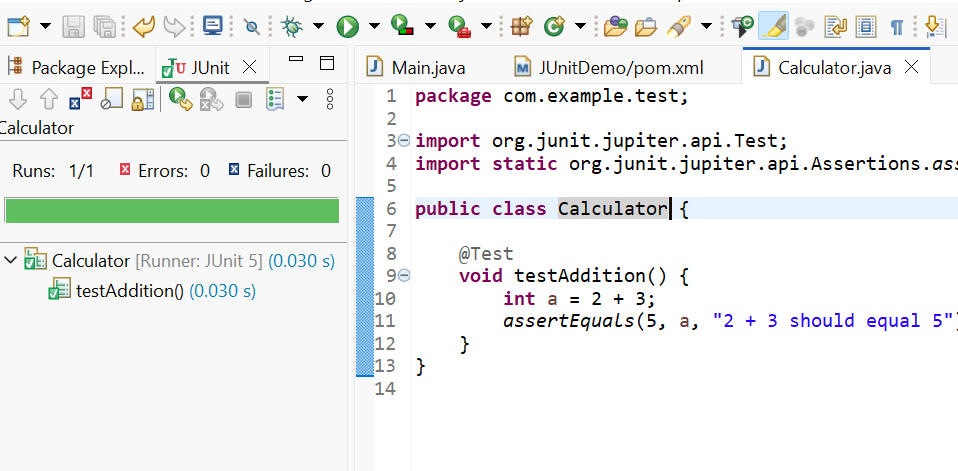
**int** a = 2 + 3;

*assertEquals*(5, a, "2 + 3 should equal 5");

}

}

Output:



**Exercise 2: Writing Basic JUnit Tests**

Code:

Calculator.java

**package** com.example.JUnitDemo;

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

// Add two numbers

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

**return** a + b;

}

// Subtract two numbers

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

**return** a - b;

}

// Multiply two numbers

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

**return** a \* b;

}

}

CalculatorTest.java

**package** com.example.test;

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

**import** com.example.JUnitDemo.Calculator;

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

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

Calculator calculator = **new** Calculator();

@Test

**void** testAdd() {

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

*assertEquals*(5, result, "2 + 3 should equal 5");

}

@Test

**void** testSubtract() {

**int** result = calculator.subtract(5, 2);

*assertEquals*(3, result, "5 - 2 should equal 3");

}

@Test

**void** testMultiply() {

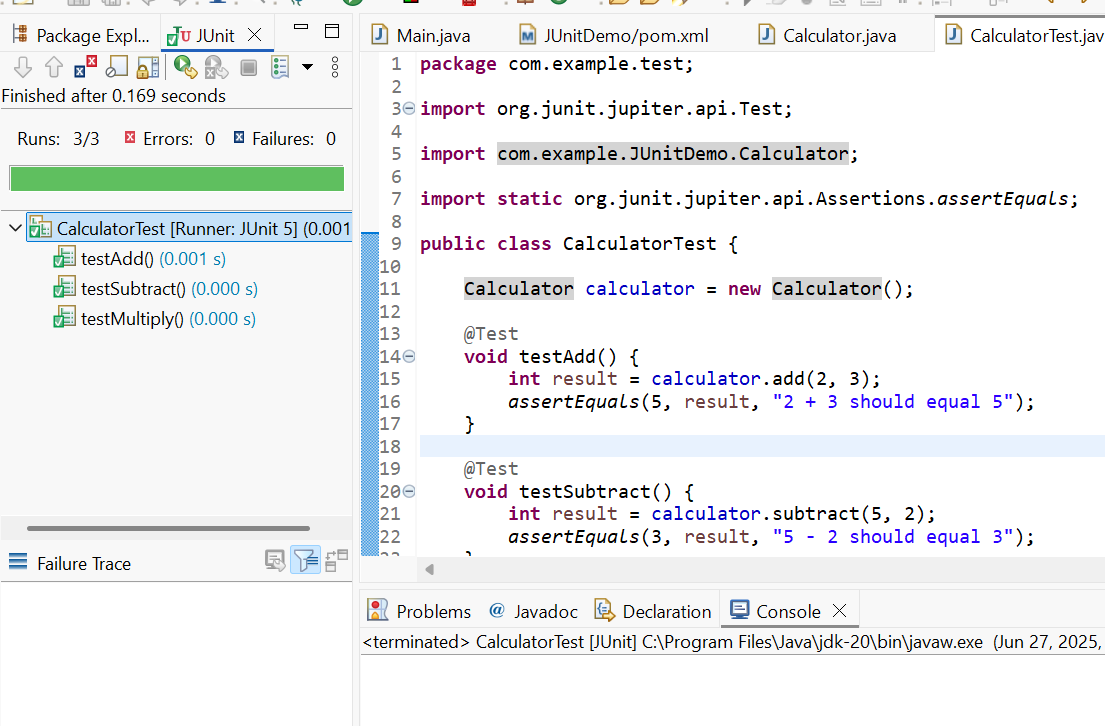
**int** result = calculator.multiply(4, 3);

*assertEquals*(12, result, "4 \* 3 should equal 12");

}

}

Output:



**Exercise 3: Assertions in JUnit**

Code:

AssertionsTest.java

**package** com.example.test;

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

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

**public** **class** AssertionsTest {

@Test

**public** **void** testAssertions() {

// Assert equals

*assertEquals*(5, 2 + 3, "2 + 3 should equal 5");

// Assert true

*assertTrue*(5 > 3, "5 is greater than 3");

// Assert false

*assertFalse*(5 < 3, "5 is not less than 3");

// Assert null

Object obj = **null**;

*assertNull*(obj, "Object should be null");

// Assert not null

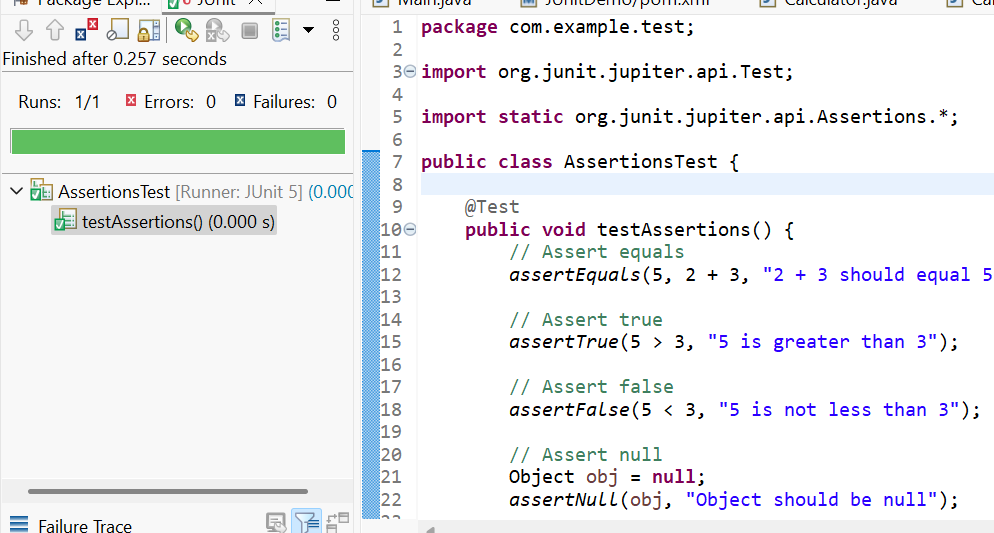
Object obj2 = **new** Object();

*assertNotNull*(obj2, "Object should not be null");

}

}

Output:



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

Code:

**package** com.example.test;

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

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

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

**import** com.example.JUnitDemo.\*;

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

**public** **class** AAATest {

**private** Calculator calculator;

@BeforeEach

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

System.***out***.println("Running setup...");

calculator = **new** Calculator();

}

@AfterEach

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

System.***out***.println("Running teardown...");

calculator = **null**; }

@Test

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

// Arrange

**int** a = 2;

**int** b = 3;

// Act

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

// Assert

*assertEquals*(5, result, "2 + 3 should equal 5");

}

@Test

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

// Arrange

**int** a = 5;

**int** b = 2;

// Act

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

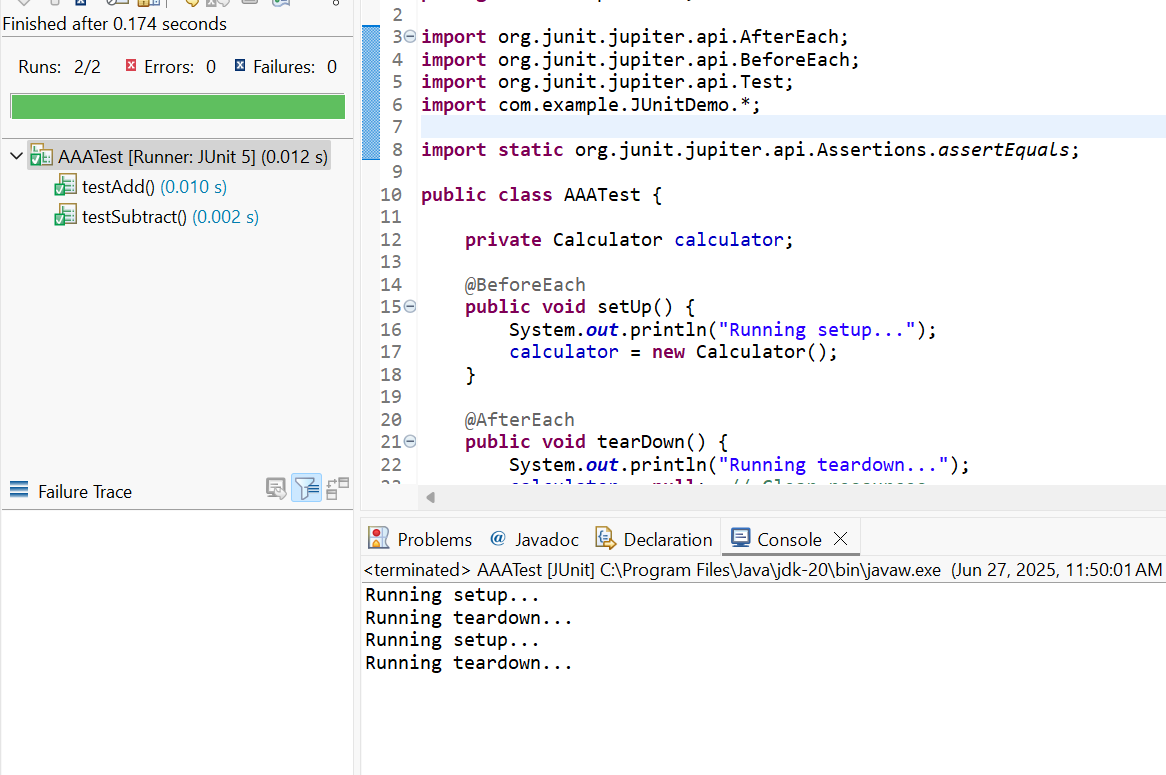
// Assert

*assertEquals*(3, result, "5 - 2 should equal 3");

}

}

Output:



**Mockito Exercises:**

**Exercise 1: Mocking and Stubbing**

**Code:**

**package** com.example.test;

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

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

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

**import** com.example.JUnitDemo.ExternalAPI;

**import** com.example.JUnitDemo.MyService;

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

@Test

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

// Arrange: Create mock

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

// Stub: define behavior

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

// Act: use service with mock

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

String result = service.fetchData();

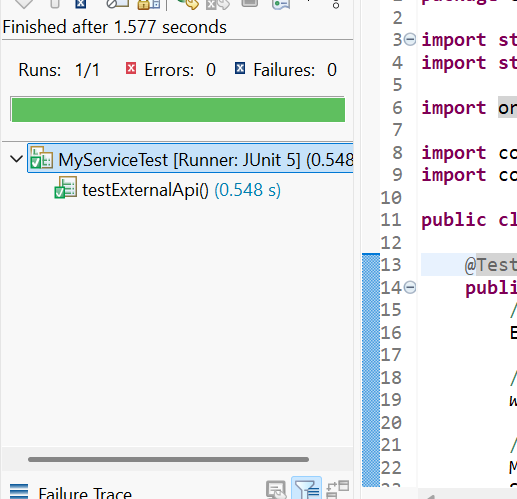
// Assert: check result

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

}

}

**Output:**



**Exercise 2: Verifying Interactions**

**Code:**

**package** com.example.test;

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

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

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

**import** com.example.JUnitDemo.ExternalAPI;

**import** com.example.JUnitDemo.MyService;

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

@Test

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

// Arrange: Create mock

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

// Stub: define behavior

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

// Act: use service with mock

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

String result = service.fetchData();

*verify*(mockApi).getData();

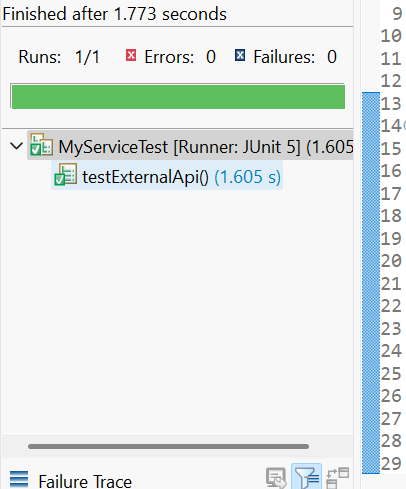
// Assert: check result

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

}

}

**Output:**



**Logging using SLF4J Exercises:**

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

**Code:**

**package** com.example.test;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**public** **class** App {

// Create Logger instance

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

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

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

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

***logger***.info("This is an INFO message");

***logger***.debug("This is a DEBUG message");

}

}

**Output:**

