**Week2\_PLSQL\_Exercises\_HandsOn**

**Exercise 1: Control Structures**

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

CustomerName VARCHAR2(50),

Age NUMBER,

Balance NUMBER,

IsVIP VARCHAR2(5)

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

InterestRate NUMBER(5,2),

DueDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Customers VALUES (1, 'John Doe', 65, 15000, 'FALSE');

INSERT INTO Customers VALUES (2, 'Jane Smith', 45, 9000, 'FALSE');

INSERT INTO Customers VALUES (3, 'Alan Rich', 70, 11000, 'FALSE');

INSERT INTO Loans VALUES (101, 1, 9.5, SYSDATE + 10); -- Due in 10 days

INSERT INTO Loans VALUES (102, 2, 10.0, SYSDATE + 40); -- Due in 40 days

INSERT INTO Loans VALUES (103, 3, 11.0, SYSDATE + 20); -- Due in 20 days

COMMIT;

**Scenario 1 - Apply 1% Discount to Interest Rates (Age > 60)**

**Code:**

BEGIN

FOR cust IN (

SELECT c.CustomerID, l.LoanID, l.InterestRate

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

WHERE c.Age > 60

) LOOP

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE LoanID = cust.LoanID;

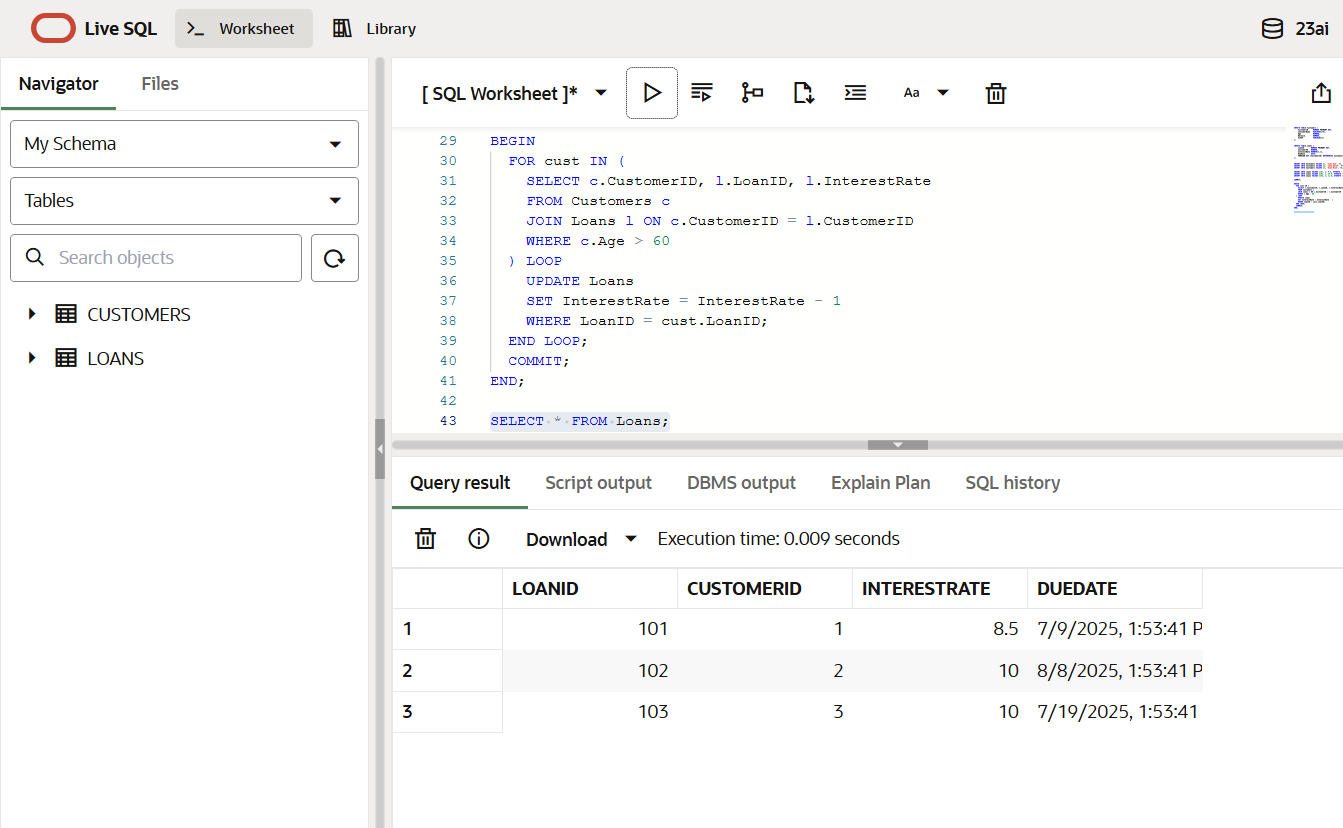
END LOOP;

COMMIT;

END;

SELECT \* FROM Loans;

**Output:**

****

**Scenario 2 - Promote Customers with Balance > 10000 to VIP**

**Code:**

BEGIN

FOR cust IN (

SELECT CustomerID FROM Customers WHERE Balance > 10000

) LOOP

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = cust.CustomerID;

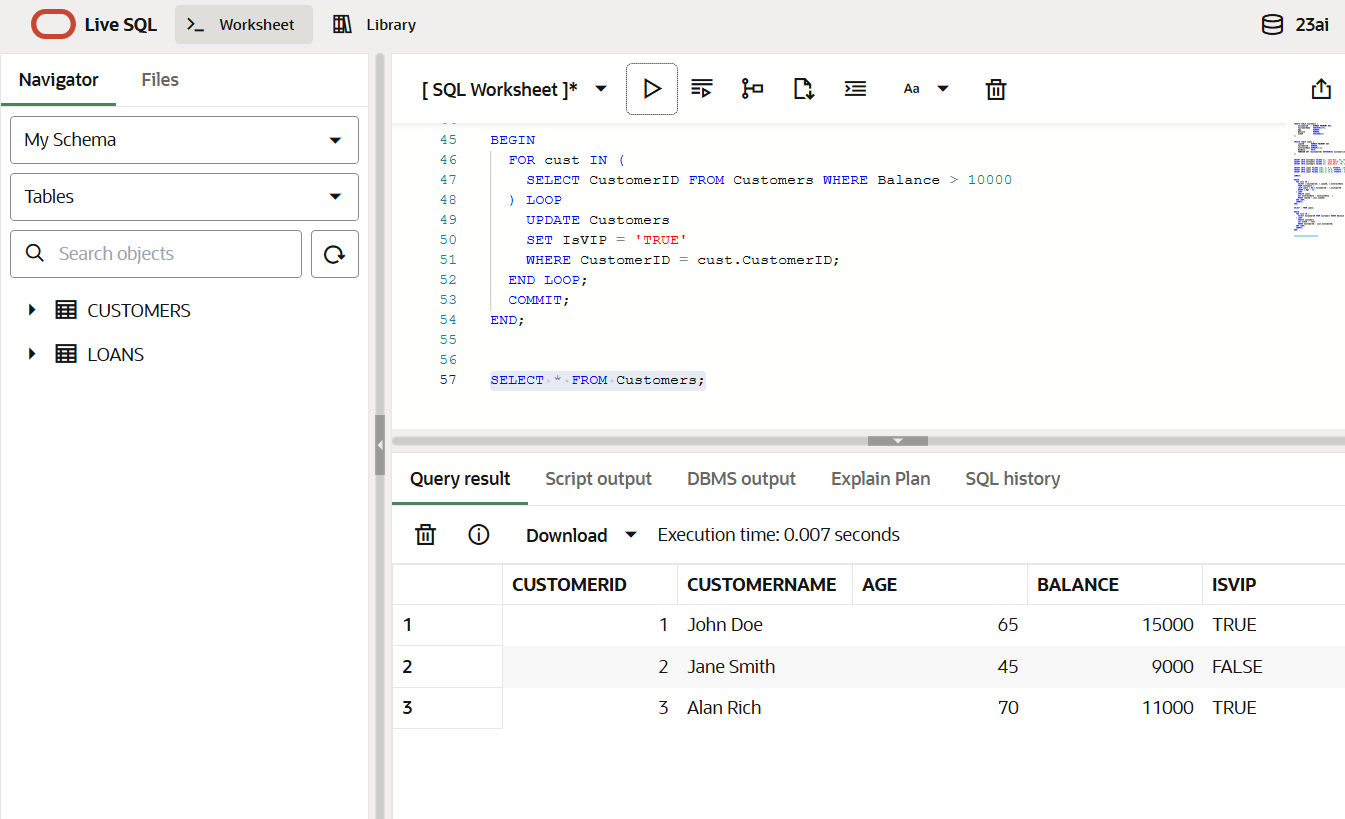
END LOOP;

COMMIT;

END;

SELECT \* FROM Customers;

**Output:**

****

**Scenario 3 - Print Reminders for Loans Due Within 30 Days**

**Code:**

BEGIN

FOR loan\_rec IN (

SELECT l.LoanID, l.CustomerID, l.DueDate, c.CustomerName

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.DueDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || loan\_rec.LoanID ||

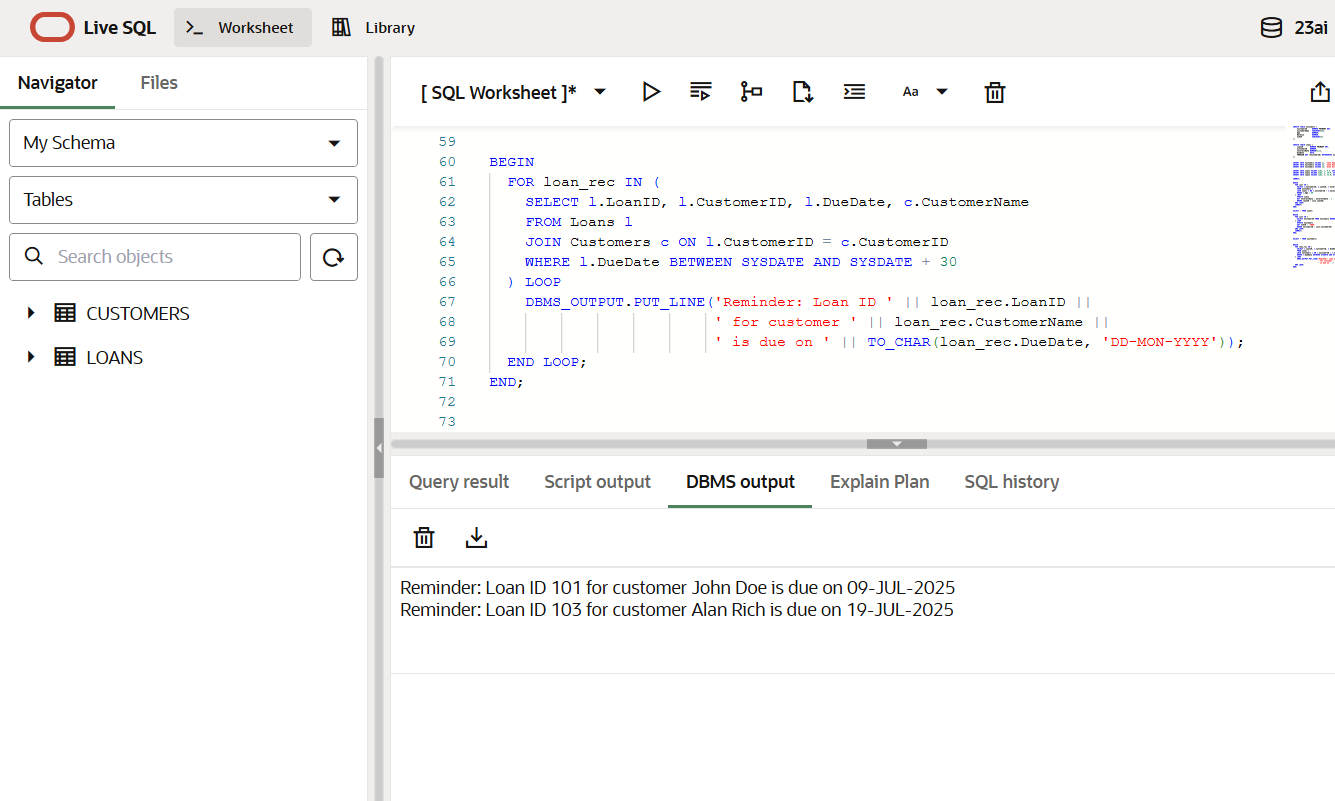
' for customer ' || loan\_rec.CustomerName ||

' is due on ' || TO\_CHAR(loan\_rec.DueDate, 'DD-MON-YYYY'));

END LOOP;

END;

**Output:**

****

**Exercise 3: Stored Procedures**

CREATE TABLE Accounts (

    AccountID      NUMBER PRIMARY KEY,

    CustomerID     NUMBER,

    AccountType    VARCHAR2(20),

    Balance        NUMBER(10,2)

);

CREATE TABLE Employees (

    EmployeeID     NUMBER PRIMARY KEY,

    Name           VARCHAR2(50),

    Department     VARCHAR2(30),

    Salary         NUMBER(10,2)

);

INSERT INTO Accounts VALUES (1, 100, 'Savings', 5000);

INSERT INTO Accounts VALUES (2, 101, 'Savings', 7000);

INSERT INTO Accounts VALUES (3, 102, 'Current', 3000);

INSERT INTO Employees VALUES (1, 'Alice', 'HR', 60000);

INSERT INTO Employees VALUES (2, 'Bob', 'IT', 80000);

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

COMMIT;

**Scenario 1: Process Monthly Interest for Savings Accounts**

**Code:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

  FOR acc IN (SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'Savings') LOOP

    UPDATE Accounts

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

    WHERE AccountID = acc.AccountID;

  END LOOP;

  COMMIT;

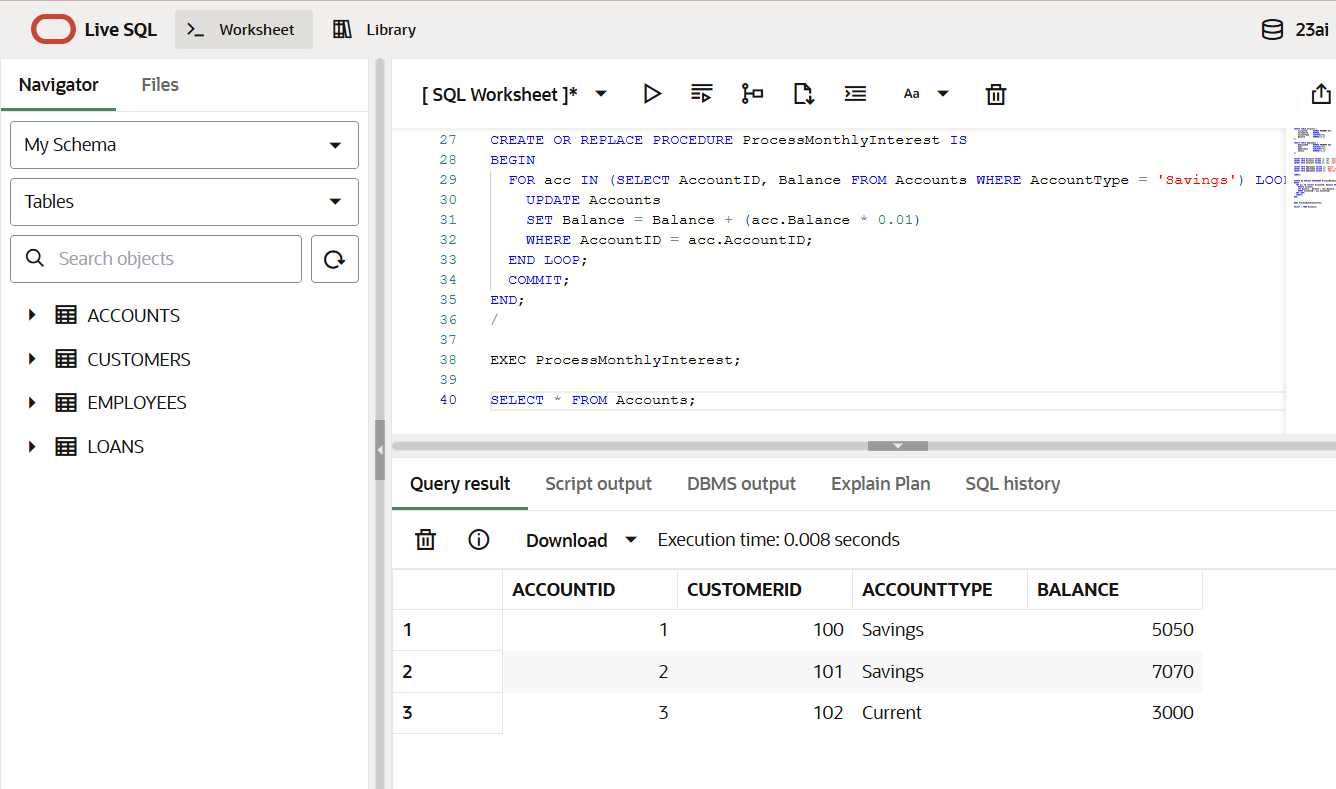
END;

/

EXEC ProcessMonthlyInterest;

SELECT \* FROM Accounts;

**Output:**

****

**Scenario 2: Update Employee Bonus by Department**

**Code:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

dept\_name IN VARCHAR2,

bonus\_pct IN NUMBER

) IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* bonus\_pct / 100)

WHERE Department = dept\_name;

COMMIT;

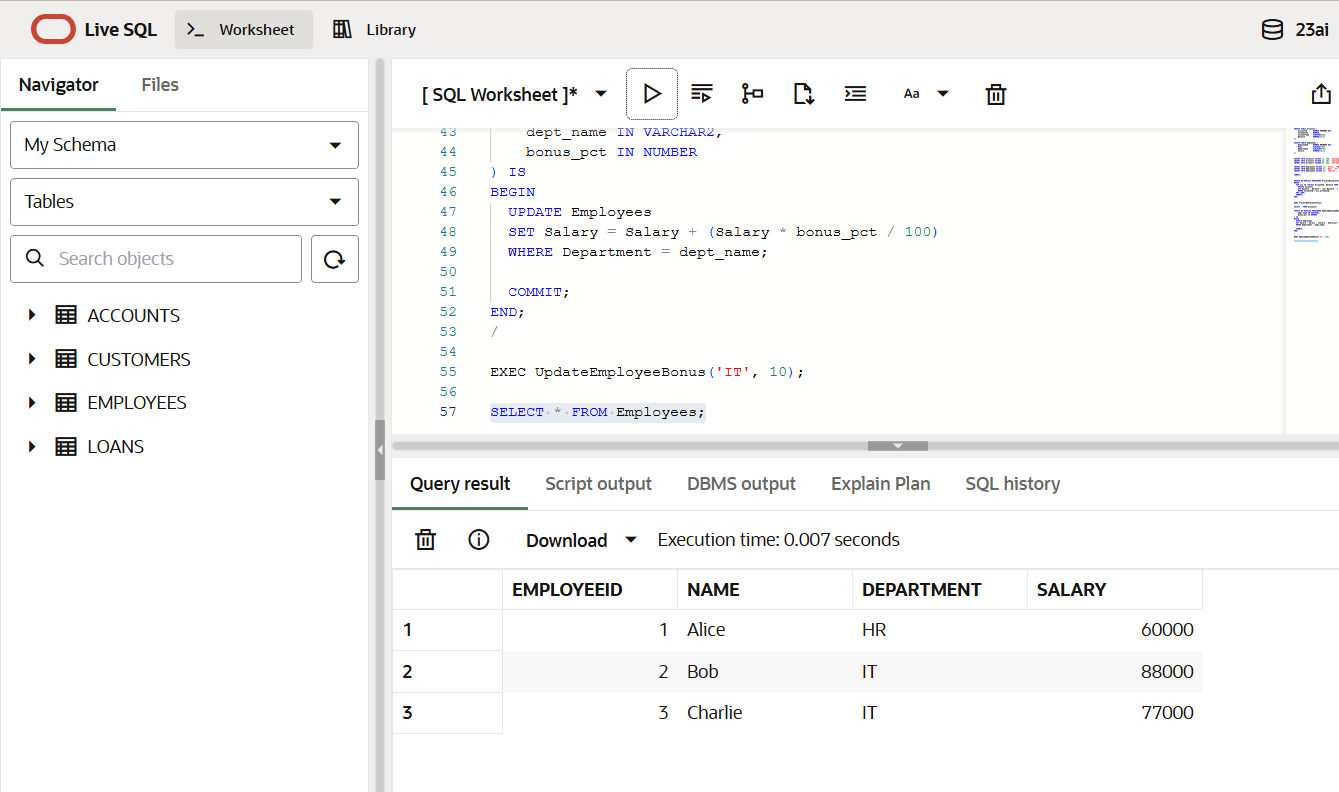
END;

/

EXEC UpdateEmployeeBonus('IT', 10);

SELECT \* FROM Employees;

**Output:**

****

**Scenario 3: Transfer Funds Between Accounts**

**Code:**

CREATE OR REPLACE PROCEDURE TransferFunds (

from\_acc IN NUMBER,

to\_acc IN NUMBER,

amount IN NUMBER

) IS

from\_balance NUMBER;

BEGIN

SELECT Balance INTO from\_balance FROM Accounts WHERE AccountID = from\_acc FOR UPDATE;

IF from\_balance < amount THEN

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

ELSE

UPDATE Accounts SET Balance = Balance - amount WHERE AccountID = from\_acc;

UPDATE Accounts SET Balance = Balance + amount WHERE AccountID = to\_acc;

COMMIT;

END IF;

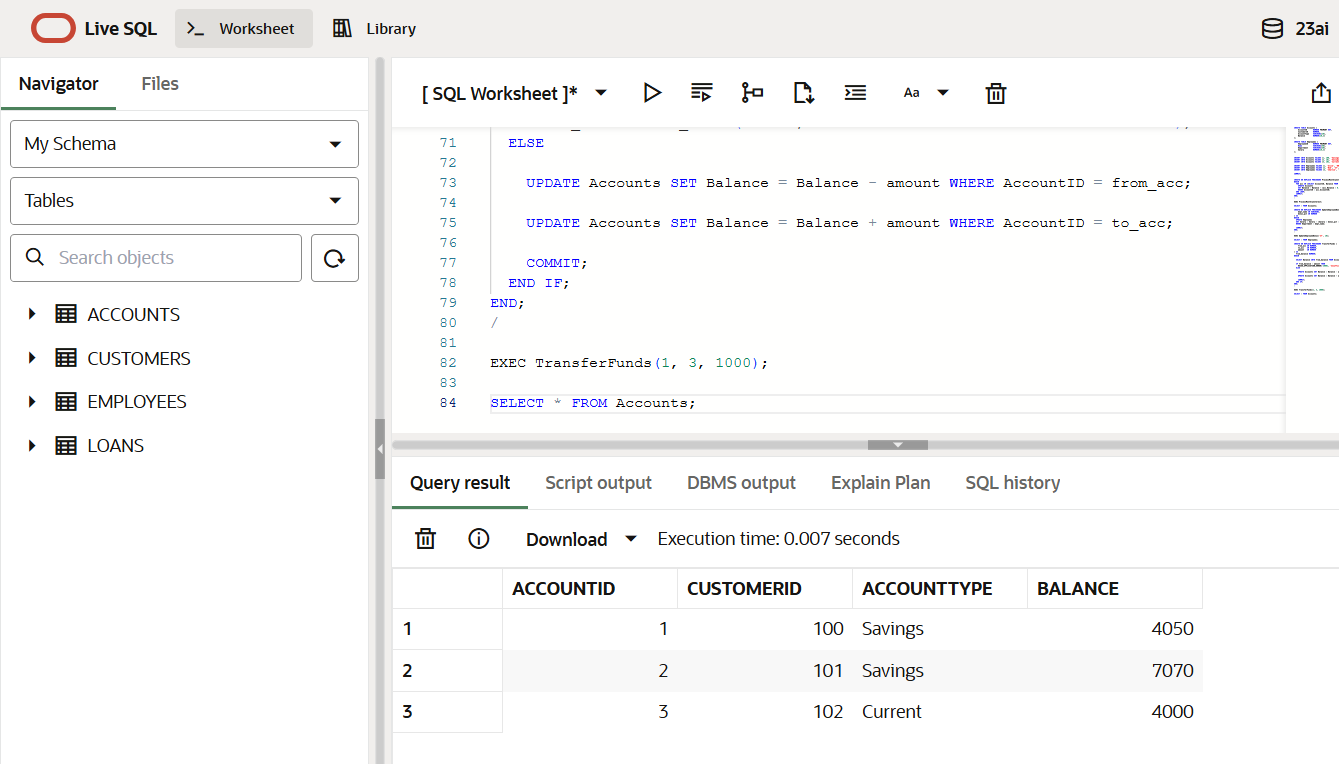
END;

/

EXEC TransferFunds(1, 3, 1000);

SELECT \* FROM Accounts;

**Output:**



**Week2\_JUnit\_Basic Testing Exercises\_HandsOn**

**Exercise 1: Setting Up JUnit**

**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.\*;

public class CalculatorTest {

@Test

public void testAdd() {

Calculator calc = new Calculator();

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

assertEquals(5, result);

}

}

pom.xml:

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

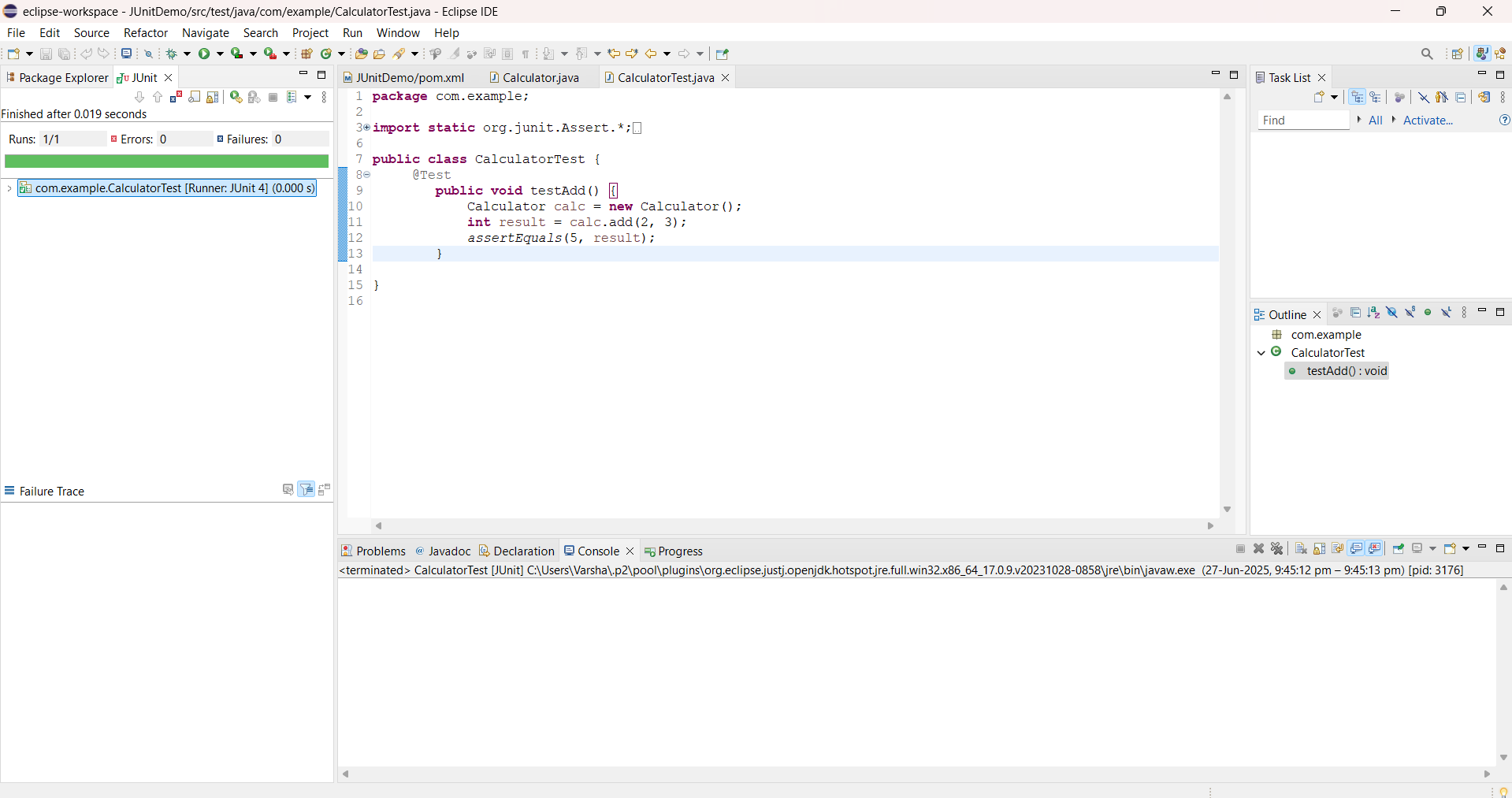
<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

**Output:**

****

**Exercise 3: Assertions in JUnit**

**Code:**

AssertionsTest.java:

package com.example;

import static org.junit.Assert.\*;

import org.junit.Test;

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());

}

}

pom.xml:

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

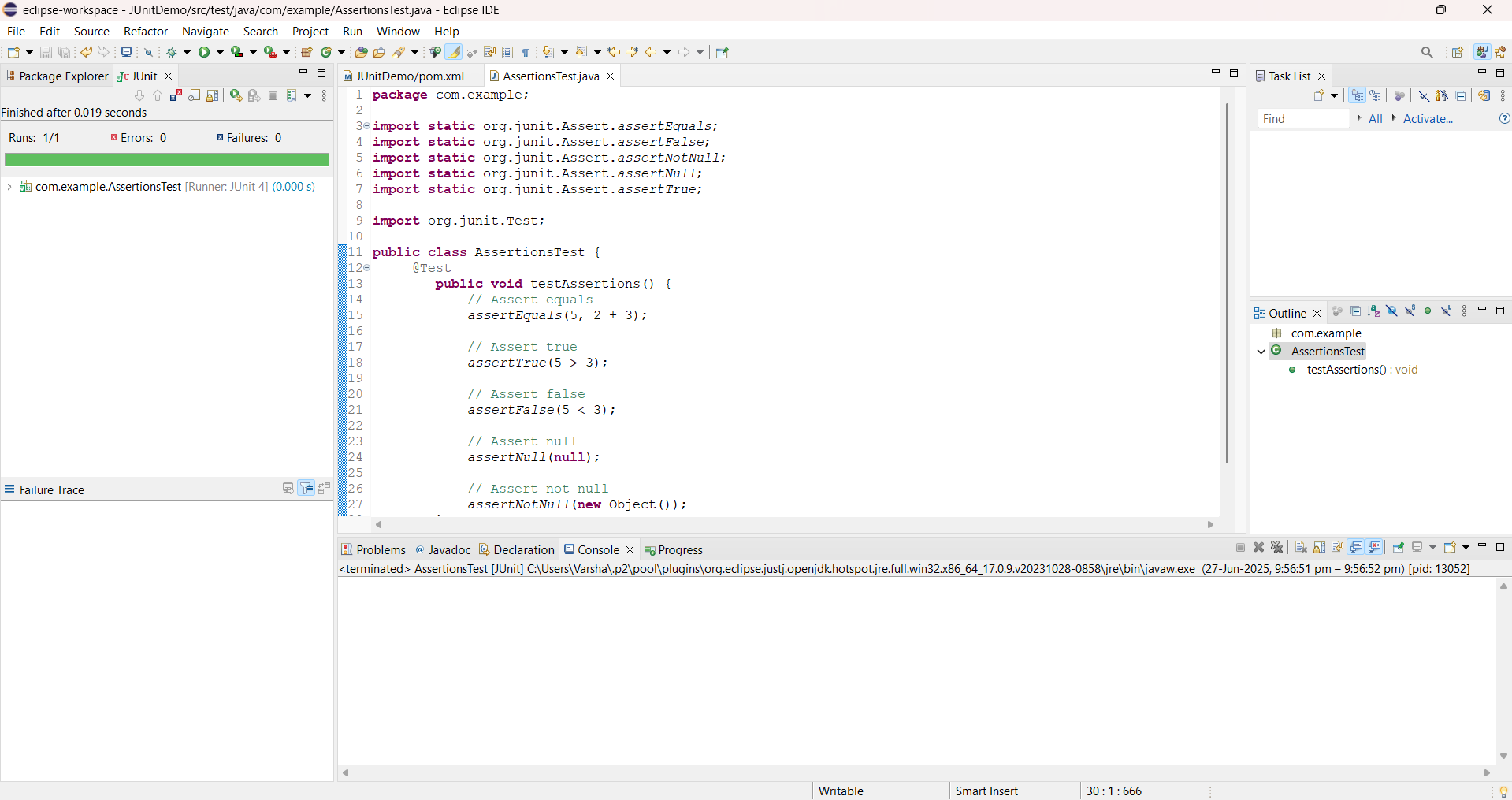
<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

**Output:**

****

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

**Code:**

Calculator.java:

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int multiply(int a, int b) {

return a \* b;

}

}

CalculatorTest.java:

package com.example.test;

import static org.junit.Assert.\*;

import org.junit.\*;

import com.example.Calculator;

public class CalculatorTest {

private Calculator calculator;

@Before

public void setUp() {

// Arrange: Setup test fixture

calculator = new Calculator();

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

}

@After

public void tearDown() {

// Cleanup

System.out.println("Cleaning up after test");

}

@Test

public void testAddition() {

// Act

int result = calculator.add(10, 5);

// Assert

assertEquals(15, result);

}

@Test

public void testMultiplication() {

// Act

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

// Assert

assertEquals(12, result);

}

}

pom.xml:

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

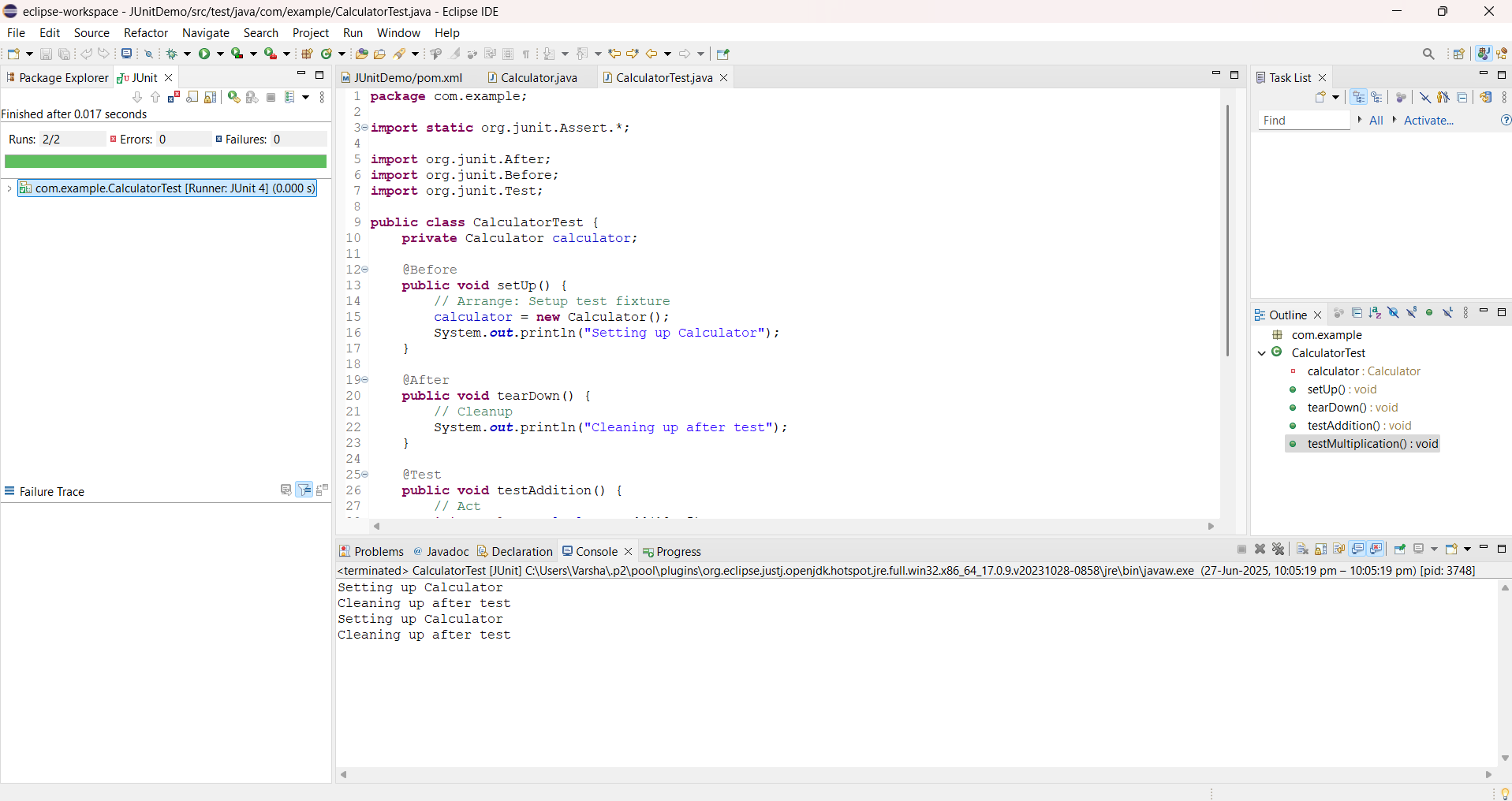
<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

**Output:**

****

**Week2\_Mockito Exercises\_HandsOn**

**Exercise 1: Mocking and Stubbing**

**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.junit.Assert.*assertEquals*;

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);

}

}

pom.xml:

<dependencies>

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

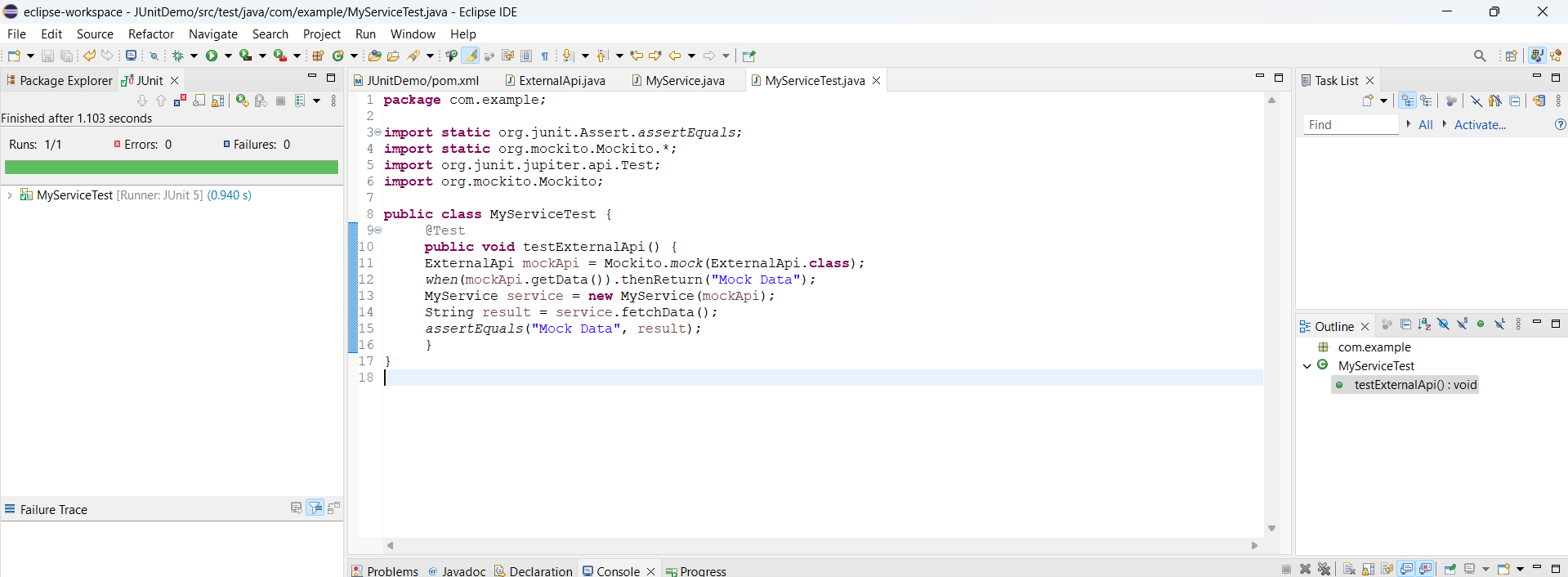
<version>5.11.0</version>

<scope>test</scope>

</dependency>

</dependencies>

**Output:**

****

**Exercise 2: Verifying Interactions**

**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();

}

}

pom.xml:

<dependencies>

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

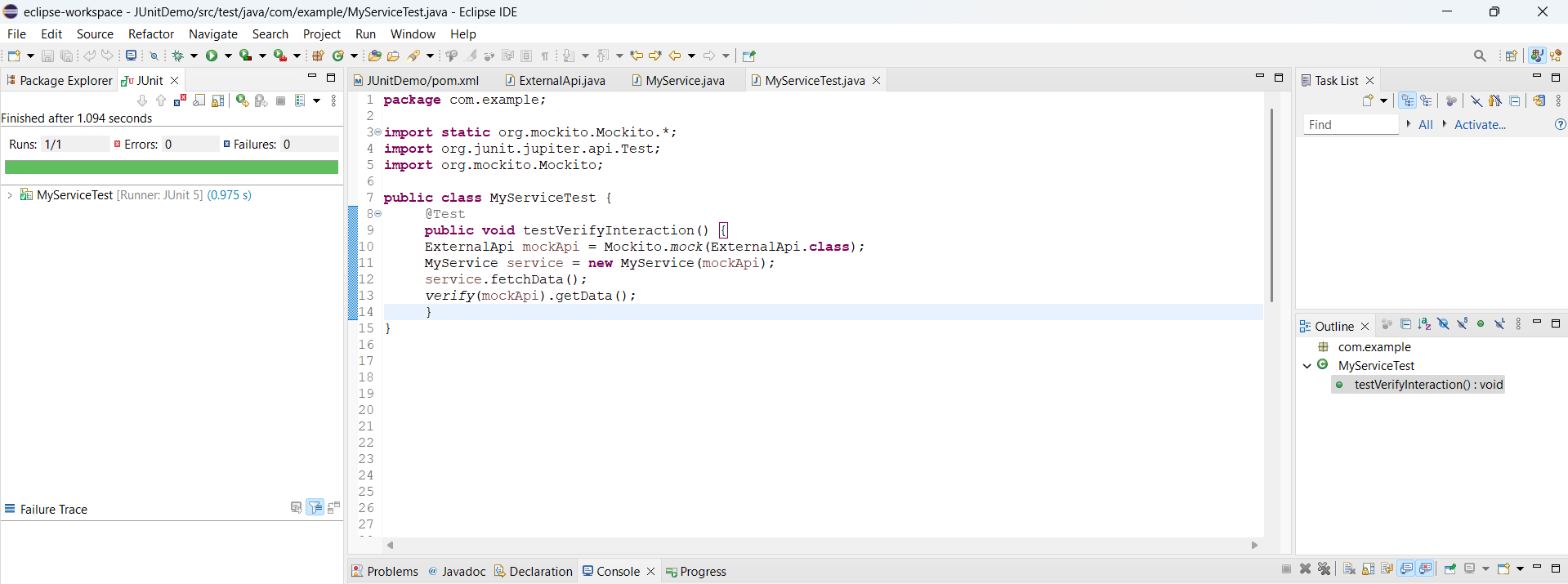
<version>5.11.0</version>

<scope>test</scope>

</dependency>

</dependencies>

**Output:**

****

**Week2\_SL4J Logging Exercises\_HandsOn**

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

**Code:**

LoggingExample.java:

package com.example;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LoggingExample {

private static final 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");

}

}

pom.xml:

<dependencies>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.30</version>

</dependency>

<dependency>

<groupId>ch.qos.logback</groupId>

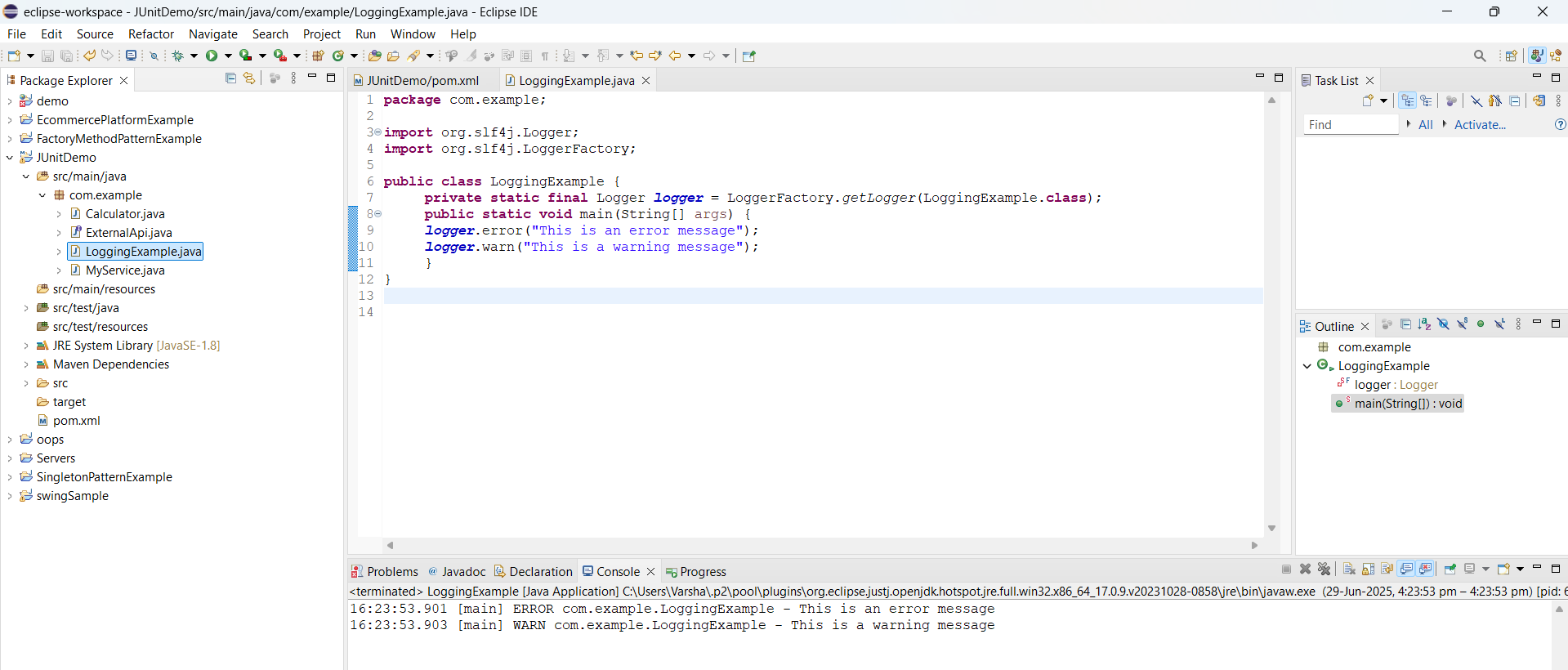
<artifactId>logback-classic</artifactId>

<version>1.2.3</version>

</dependency>

</dependencies>

**Output:**

****