**Mockito Hands-On Exercises**  
  
**Exercise 1:**

**Mocking and Stubbing**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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.geetha.mock</groupId>

<artifactId>MockitoDemoProject</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>Mockito Demo Project</name>

<dependencies>

<!-- ✅ JUnit 5 -->

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.10.2</version>

<scope>test</scope>

</dependency>

<!-- ✅ Mockito -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.11.0</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<!-- ✅ Plugin to run JUnit 5 tests -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.2.5</version>

</plugin>

</plugins>

</build>

</project>

ExternalApi.java:

**package** com.geetha.mock;

**public** **interface** ExternalApi {

String getData();

}

Myservice.java:

**package** com.geetha.mock;

**public** **class** MyService {

**private** ExternalApi api;

**public** MyService(ExternalApi api) {

**this**.api = api;

}

**public** String fetchData() {

**return** api.getData();

}

}

MyServiceTest.java:

**package** com.geetha.mock;

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

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

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

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

@Test

**public** **void** testExternalApiMocking() {

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

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

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

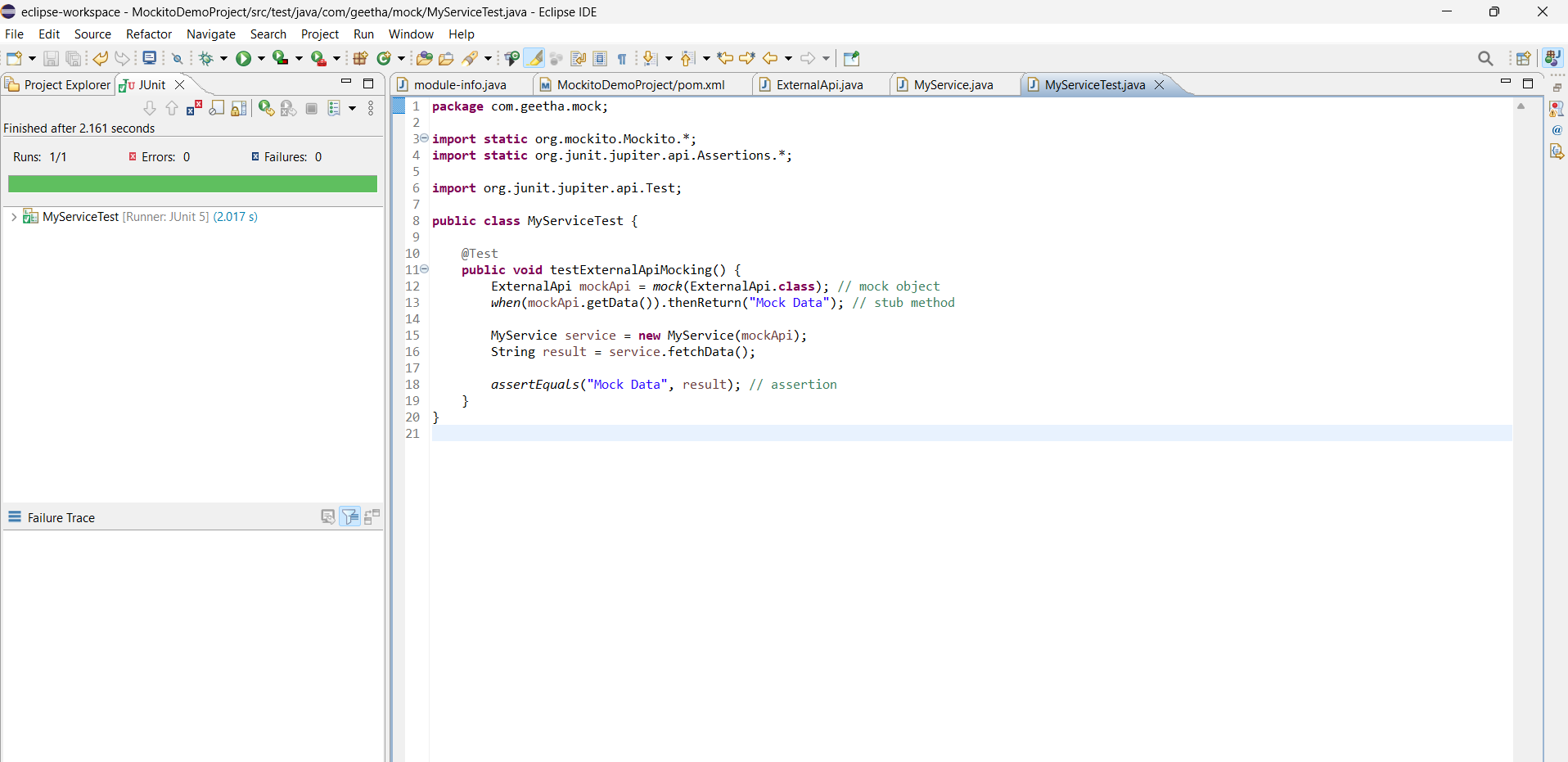
String result = service.fetchData();

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

}

}

**OUTPUT:**



**Exercise 2:**

**Verifying Interactions**

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

<modelVersion>4.0.0</modelVersion>

<groupId>com.geetha.mock</groupId>

<artifactId>MockitoDemoProject</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>Mockito Demo Project</name>

<dependencies>

<!-- ✅ JUnit 5 -->

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.10.2</version>

<scope>test</scope>

</dependency>

<!-- ✅ Mockito -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.11.0</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<!-- ✅ Plugin to run JUnit 5 tests -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.2.5</version>

</plugin>

</plugins>

</build>

</project>

ExternalApi.java:

**package** com.geetha.mock;

**public** **interface** ExternalApi {

String getData();

}

Myservice.java:

**package** com.geetha.mock;

**public** **class** MyService {

**private** ExternalApi api;

**public** MyService(ExternalApi api) {

**this**.api = api;

}

**public** String fetchData() {

**return** api.getData();

}

}

MyServiceTest.java:

**package** com.geetha.mock;

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

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

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

@Test

**public** **void** testVerifyMethodCalled() {

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

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

System.***out***.println("Calling service.fetchData()...");

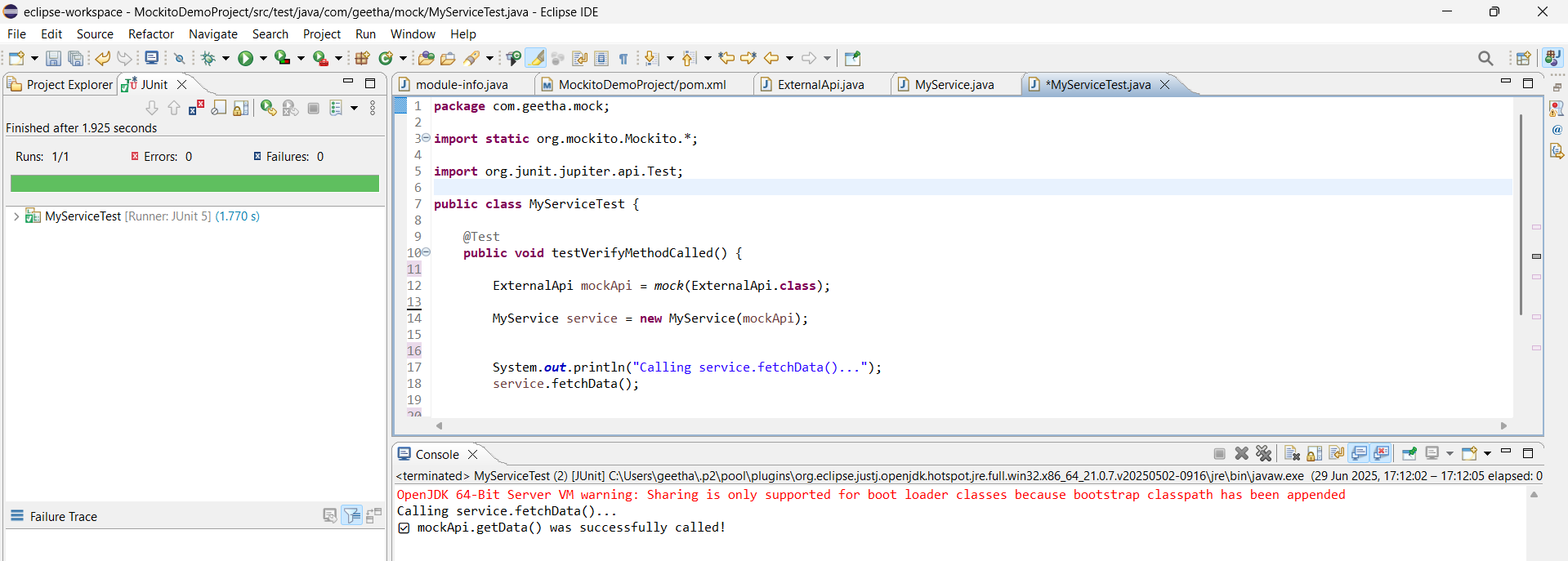
service.fetchData();

*verify*(mockApi).Data();

System.***out***.println("✅ mockApi.getData() was successfully called!");

}

}

**OUTPUT:**  


**Logging using SLJ4 framework**

**Exercise 1:**

**Logging Error Messages and Warning Levels**

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

<modelVersion>4.0.0</modelVersion>

<groupId>com.geetha.logging</groupId>

<artifactId>LoggingApp</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>LoggingApp</name>

<dependencies>

<!-- SLF4J API -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.30</version>

</dependency>

<!-- Logback (backend for SLF4J) -->

<dependency>

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

<artifactId>logback-classic</artifactId>

<version>1.2.3</version>

</dependency>

</dependencies>

</project>

LoggingExample.java:

**package** com.geetha.logging;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**public** **class** LoggingExample {

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

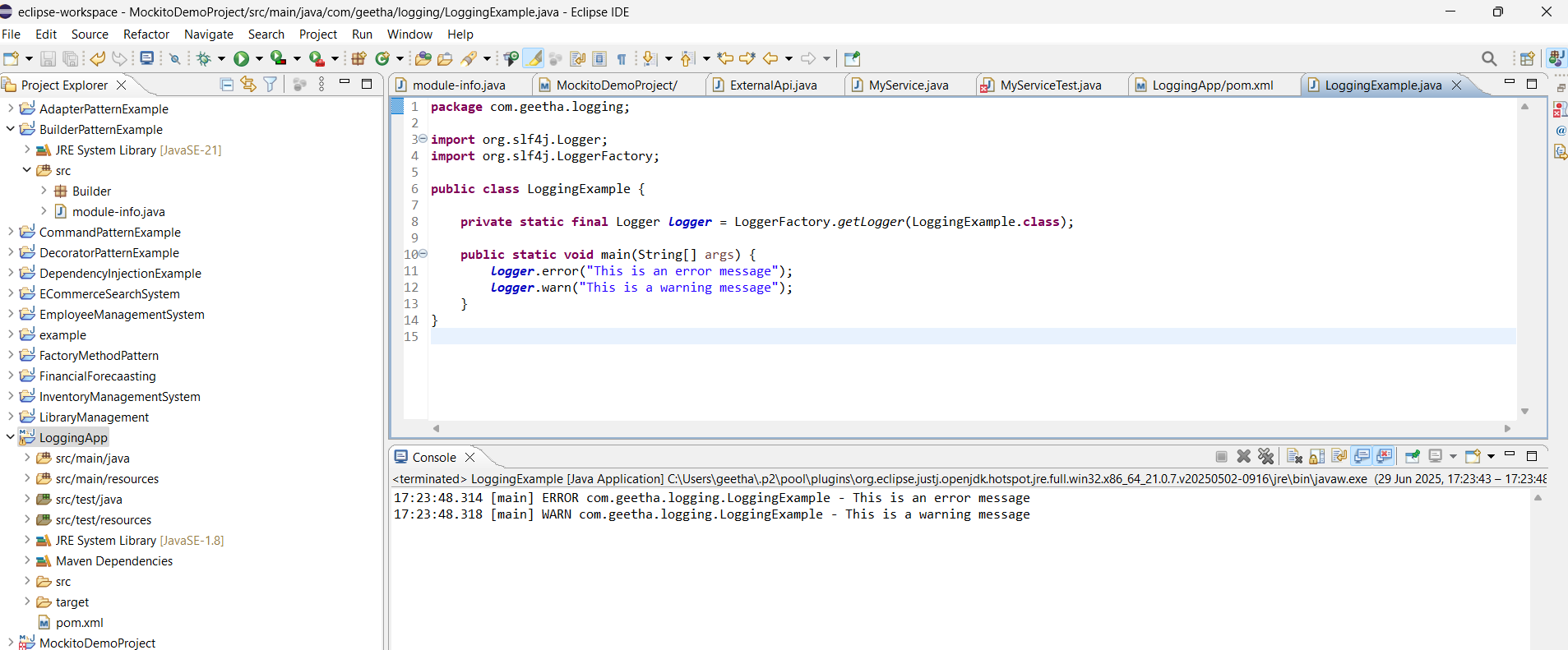
**public** **static** **void** main(String[] args) {

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

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

}

}

**OUTPUT:**  


**WEEK-2 CTS-MODULE**

**PL/SQL**

**Exercise 1  
Control Structures**

Before running a Scenario I created tables:

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);  
  
Inserting a values into tables:

-- Customers

INSERT INTO Customers VALUES (1, 'John Doe', TO\_DATE('1960-05-15', 'YYYY-MM-DD'), 15000, SYSDATE);

INSERT INTO Customers VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 9000, SYSDATE);

-- Loans

INSERT INTO Loans VALUES (1, 1, 5000, 5.0, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

INSERT INTO Loans VALUES (2, 2, 7000, 6.5, SYSDATE, SYSDATE + 15);

COMMIT;

**Scenario 1:**

Discount a loan interest

BEGIN

FOR rec IN (

SELECT l.LoanID, c.DOB

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

) LOOP

IF MONTHS\_BETWEEN(SYSDATE, rec.DOB) / 12 > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE LoanID = rec.LoanID;

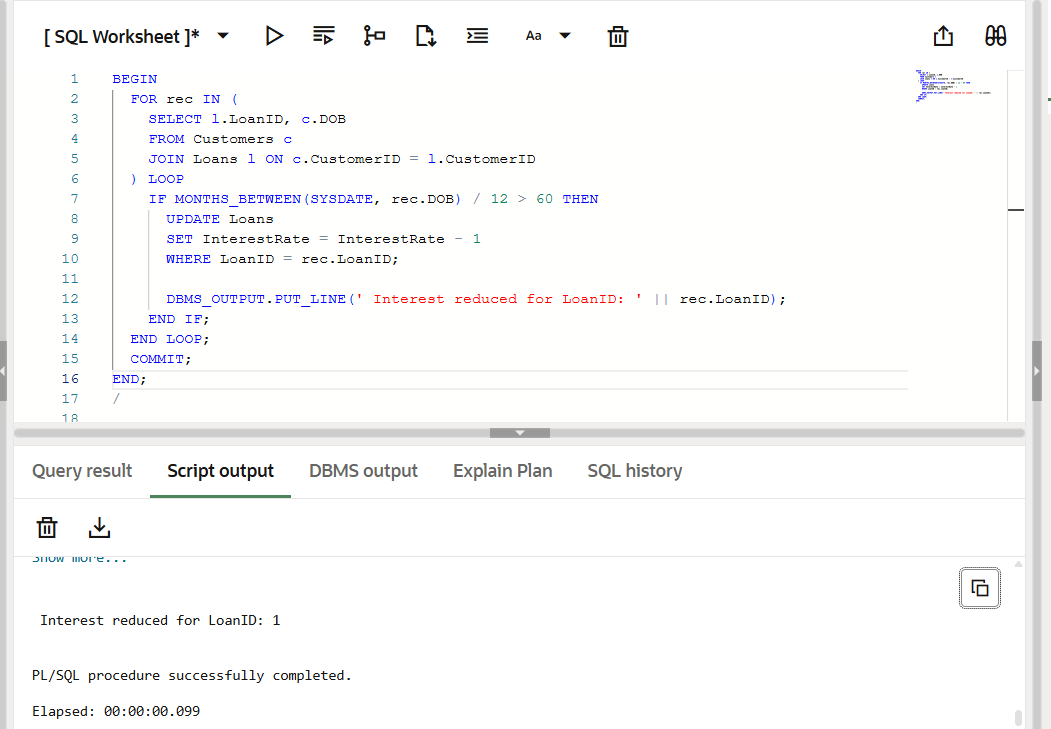
DBMS\_OUTPUT.PUT\_LINE(' Interest reduced for LoanID: ' || rec.LoanID);

END IF;

END LOOP;

COMMIT;

END;

**OUTPUT:**  


**Scenario 2:**  
  
customer an be promoted to VIP status

Before running a code ,alter the table using

ALTER TABLE Customers ADD IsVIP VARCHAR2(5);

BEGIN

  FOR rec IN (SELECT \* FROM Customers) LOOP

    IF rec.Balance > 10000 THEN

      UPDATE Customers SET IsVIP = 'TRUE' WHERE CustomerID = rec.CustomerID;

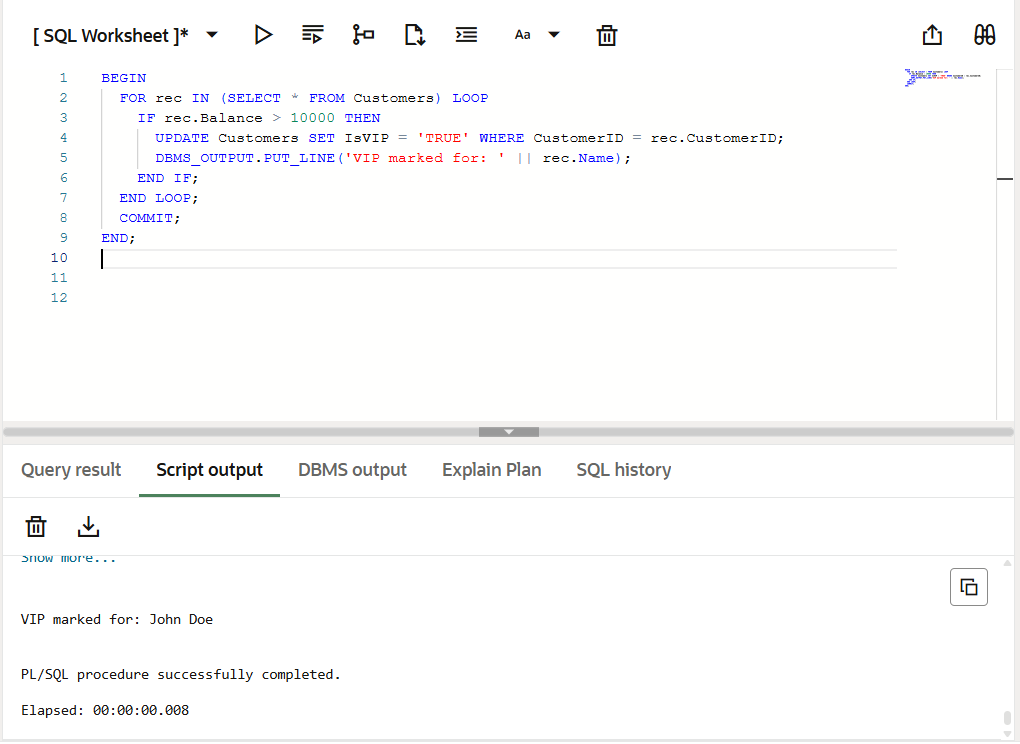
      DBMS\_OUTPUT.PUT\_LINE('VIP marked for: ' || rec.Name);

    END IF;

  END LOOP;

  COMMIT;

END;

**OUTPUT:**  
  
  
**Scenario 3:**

Banks sends a reminder to customers  
  
BEGIN

FOR rec IN (

SELECT c.Name, l.EndDate

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

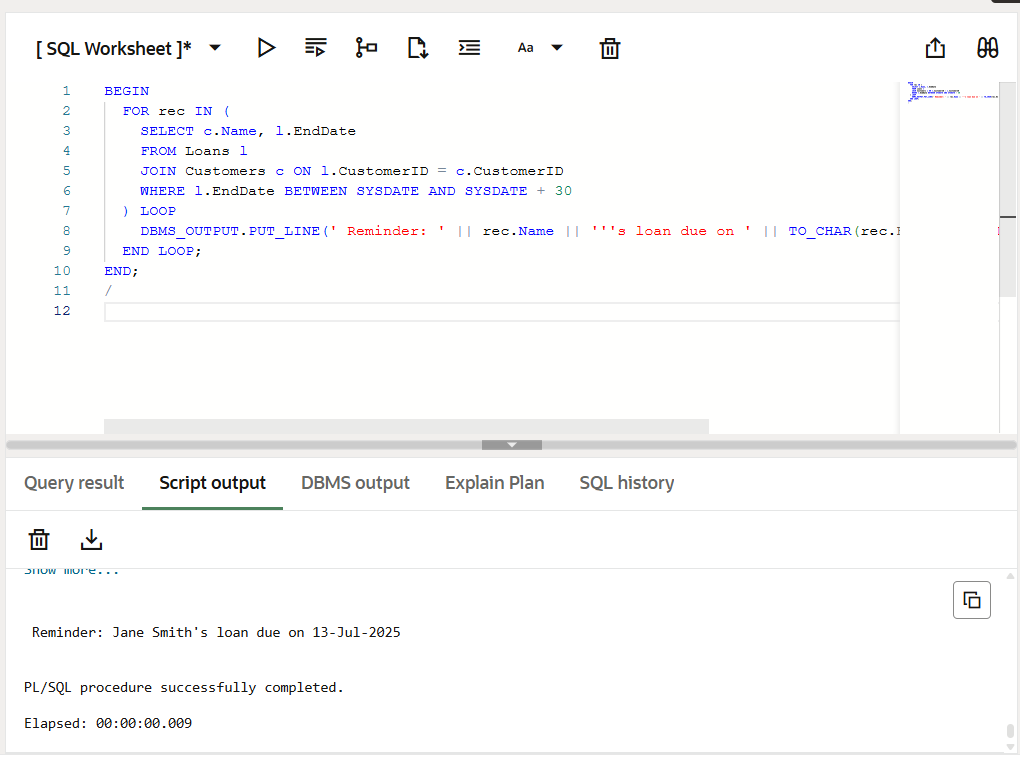
WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE(‘ Reminder: ' || rec.Name || '''s loan due on ' || TO\_CHAR(rec.EndDate, 'DD-Mon-YYYY'));

END LOOP;

END;

**OUTPUT:**  


**Exercise 3:**

**Stored Procedures**

Creating a tables:

-- Accounts table

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE

);

-- Employees table

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);  
  
Inserting a data:  
  
-- Sample Accounts

INSERT INTO Accounts VALUES (101, 1, 'Savings', 10000, SYSDATE);

INSERT INTO Accounts VALUES (102, 2, 'Current', 7000, SYSDATE);

-- Sample Employees

INSERT INTO Employees VALUES (201, 'Asha', 'Developer', 50000, 'IT', SYSDATE);

INSERT INTO Employees VALUES (202, 'Ravi', 'Tester', 40000, 'QA', SYSDATE);

COMMIT;  
  
**Scenario 1:**  
**Monthly Interest for savings account**:  
  
1.Creating a Stored procedure:

CREATE OR REPLACE PROCEDURE CalculateMonthlyInterest IS

BEGIN

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

UPDATE Accounts

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

LastModified = SYSDATE

WHERE AccountID = acc.AccountID;

DBMS\_OUTPUT.PUT\_LINE(' Interest added for AccountID: ' || acc.AccountID);

END LOOP;

COMMIT;

END;

2.Execute a procedure:

BEGIN

CalculateMonthlyInterest;

END;

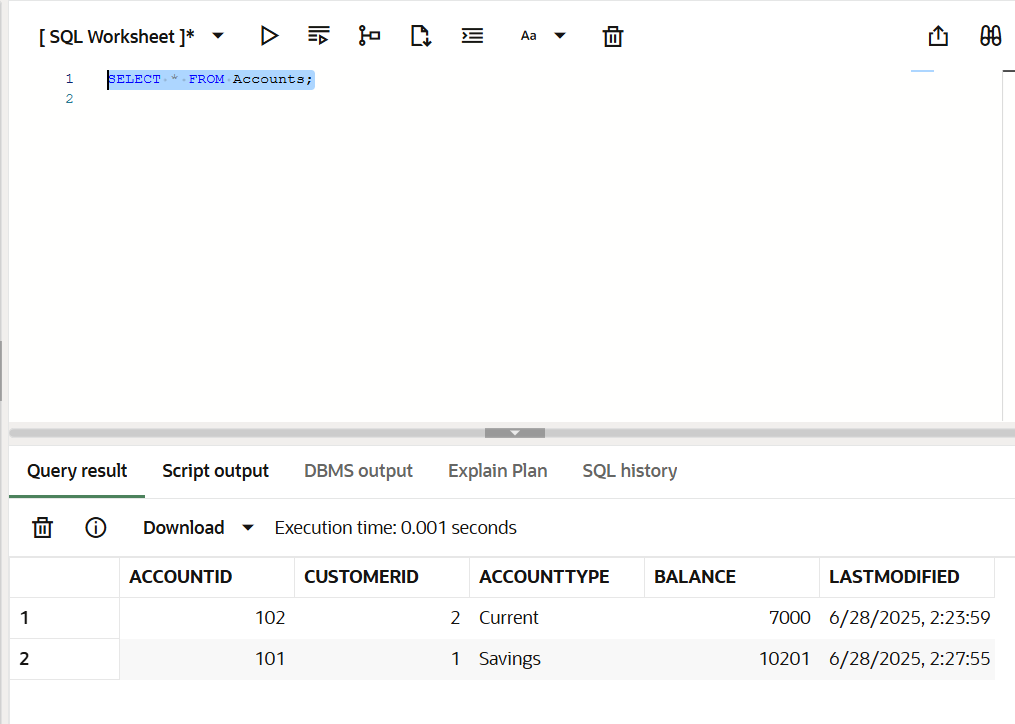
**Output:**

Interest added for AccountID: 101

3.View Balance :

SELECT \* FROM Accounts;

**OUTPUT:**



**Scenario 2:  
Employee Bonus By department**  
  
1.Creating a Stroed procedure:

CREATE OR REPLACE PROCEDURE GiveBonusToDepartment (

p\_dept IN VARCHAR2,

p\_bonus\_pct IN NUMBER

) IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* p\_bonus\_pct / 100),

HireDate = SYSDATE -- Just for update tracking

WHERE Department = p\_dept;

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

COMMIT;

END;

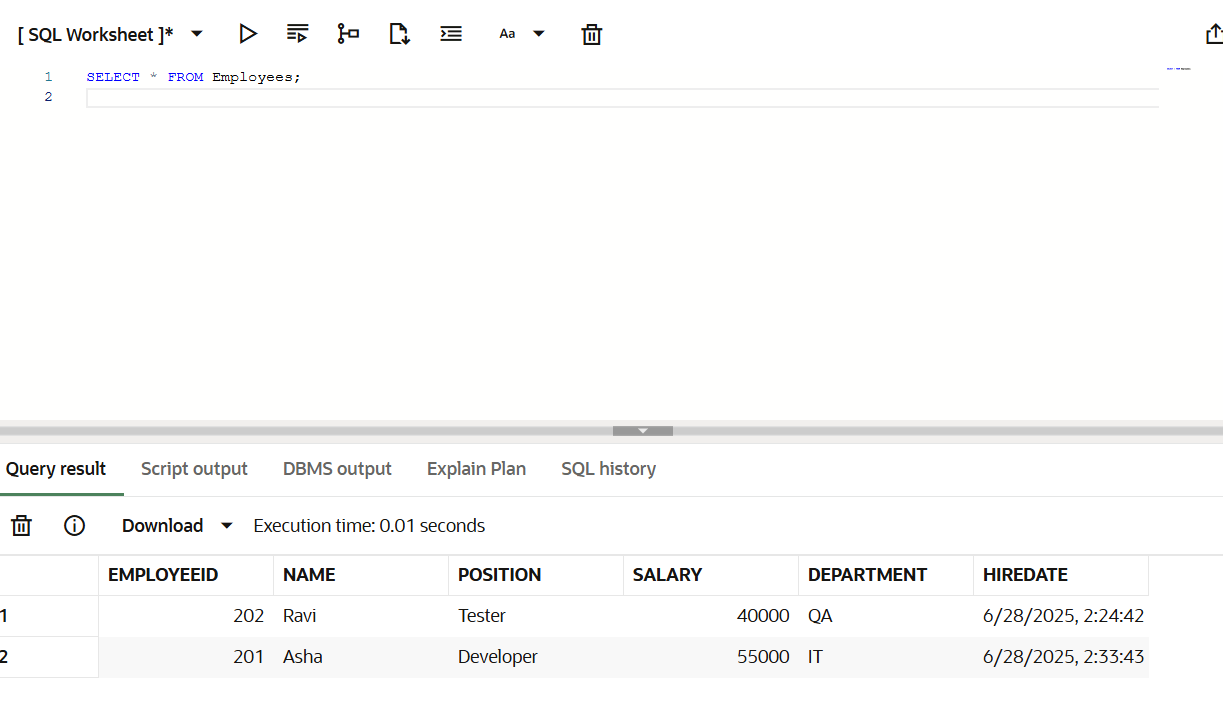
**Output:**  
Procedure GIVEBONUSTODEPARTMENT compiled  
  
2.Executing a procedure:  
  
BEGIN

GiveBonusToDepartment('IT', 10);

END;

**Output:**  
  
Bonus applied to department: IT

3.View the Updated details:

Updated Employees list:  
SELECT \* FROM Employees;  
  
**OUTPUT:**  


**Scenario 3:  
Transfer funds Between account safely**  
  
1.Creating a Stored Procedure:

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) IS

v\_balance NUMBER;

BEGIN

-- Check available balance

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_from\_account;

IF v\_balance < p\_amount THEN

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

ELSE

-- Debit from source

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_from\_account;

-- Credit to destination

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_to\_account;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE(' Transferred ' || p\_amount || ' from Account ' || p\_from\_account || ' to Account ' || p\_to\_account);

END IF;

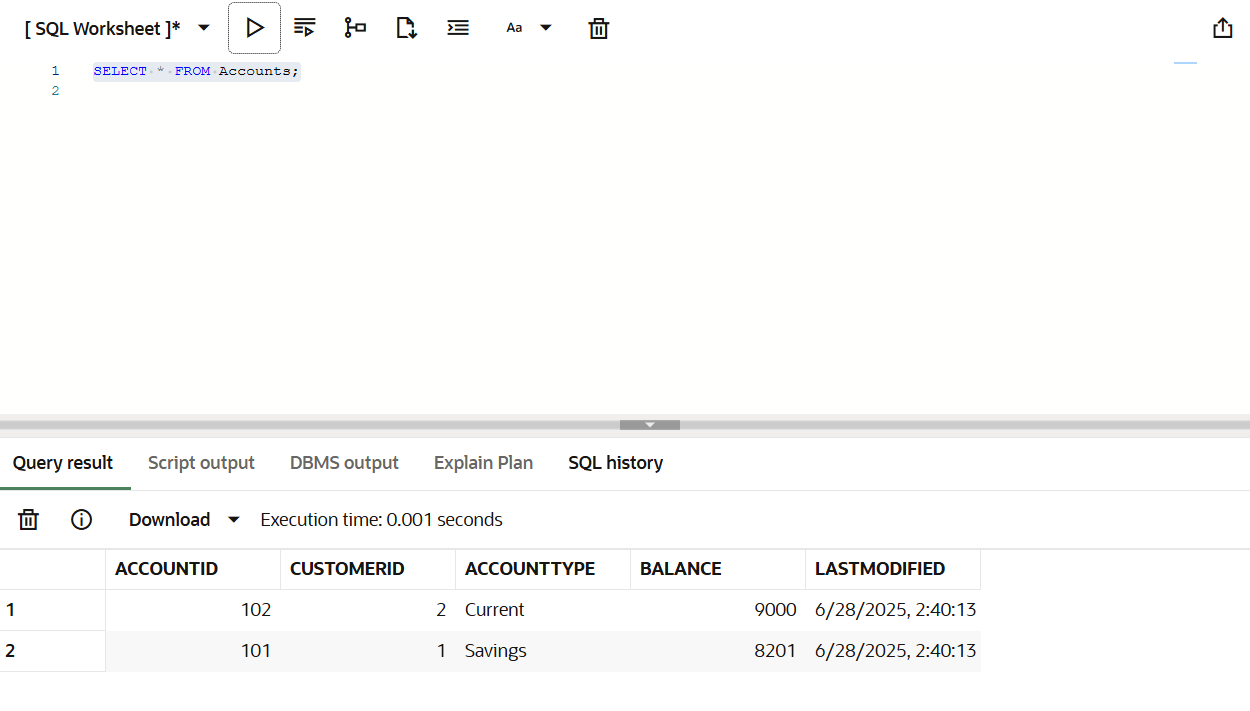
END;

**OUTPUT:**  
  
Procedure TRANSFERFUNDS compiled  
  
2.Executing a Procedure:

BEGIN

TransferFunds(101, 102, 2000);

END;

**OUTPUT:**  
  
Transferred 2000 from Account 101 to Account 102  
  
3.Verify Balances:  
  
SELECT \* FROM Accounts;  
  
**OUTPUT:**

**TDD using Junit5 and Mockito**

**Exercise 1:**

**Setting up Junit:**

JunitEx 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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.geetha</groupId>

<artifactId>JunitEx</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<!-- ✅ JUnit 4 Dependency -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**Simple test cases:**

SimpleMath.java:

**package** com.geetha.core;

**public** **class** SimpleMath {

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

**return** a + b;

}

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

**return** a \* b;

}

}

Simple MathTest.java:

**package** com.geetha.test;

**import** com.geetha.core.SimpleMath;

**import** org.junit.Test;

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

**public** **class** SimpleMathTest {

@Test

**public** **void** testAddition() {

SimpleMath math = **new** SimpleMath();

*assertEquals*(10, math.add(6, 4));

}

@Test

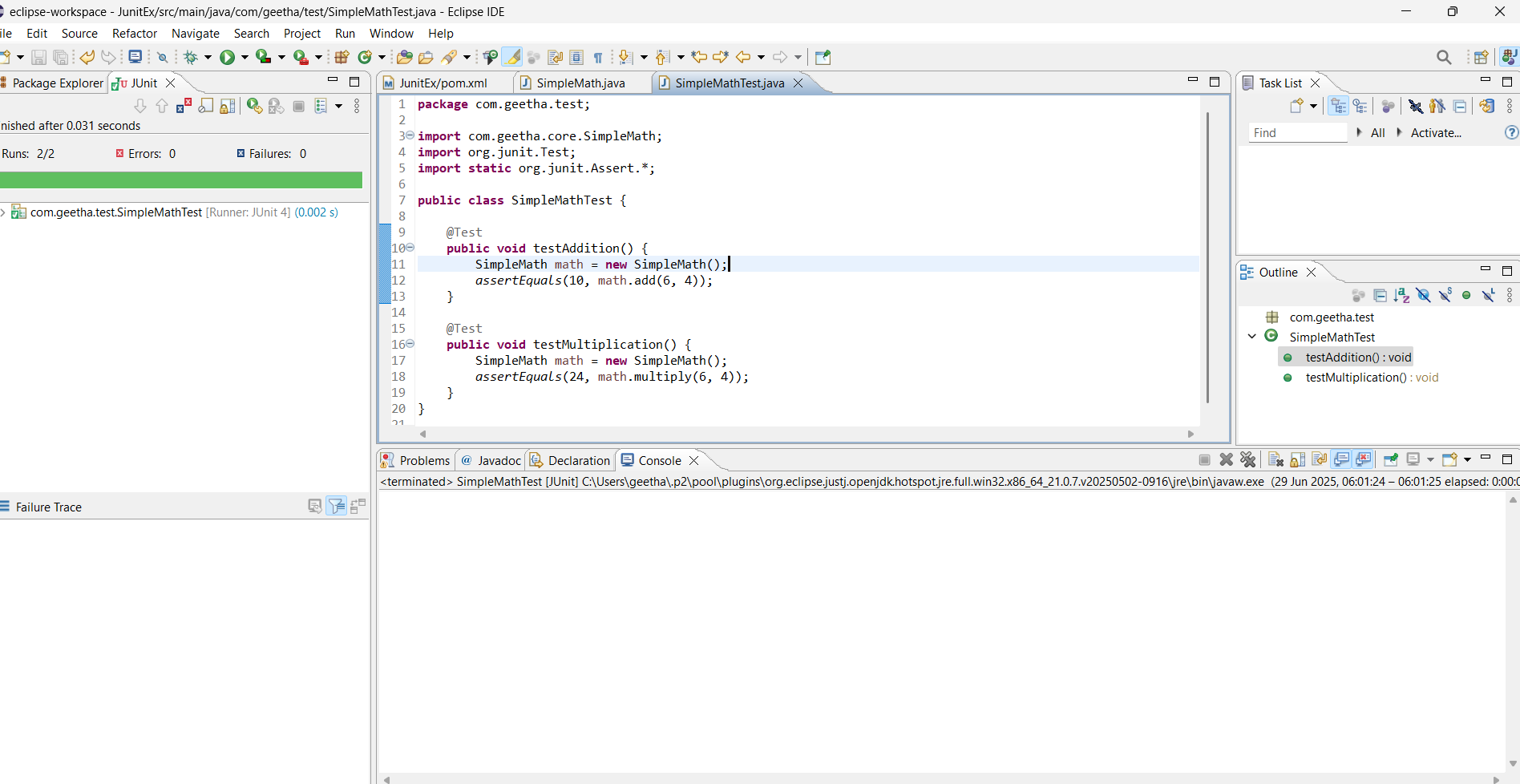
**public** **void** testMultiplication() {

SimpleMath math = **new** SimpleMath();

*assertEquals*(24, math.multiply(6, 4));

}

}

**output:**  


**Exercise 3:**

**Assertions in Junit:**

**package** com.geetha.test;

**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*(2 > 5);

// Assert Null

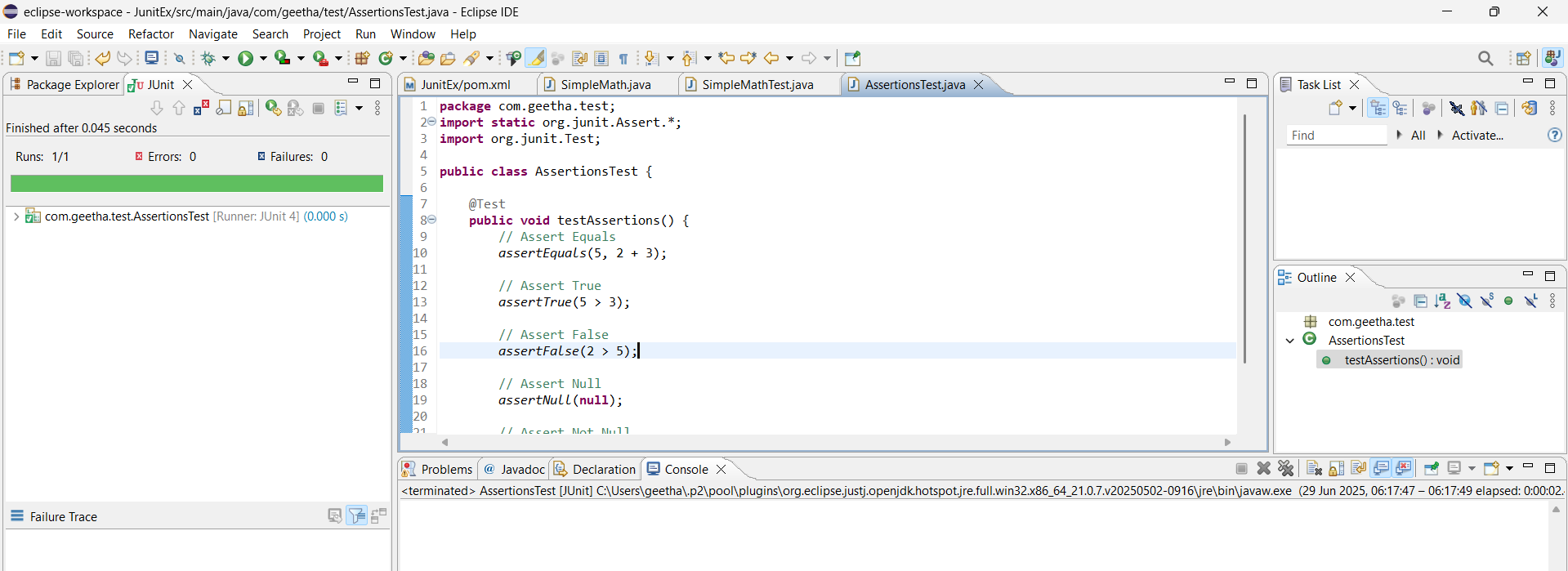
*assertNull*(**null**);

// Assert Not Null

*assertNotNull*(**new** Object());

}

}

**Output:**  


**Exercise 4:  
Arrange-Act-Asser Pattern,Test fixtures,Setup and teardown Methods in Junit:**

SimpleCalculator.Java:

**package** com.geetha.core;

**public** **class** SimpleCalculator {

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

{

**return** a+b;

}

**public** **int** mul(**int** a,**int** b)

{

**return** a\*b;

}

**public** **int** div(**int** a,**int** b)

{

**return** a/b;

}

**public** **int** sub(**int** a,**int** b)

{

**return** a-b;

}

}

SimpleCalculatorTest.java:

**package** com.geetha.test;

**import** com.geetha.core.SimpleCalculator;

**import** org.junit.Before;

**import** org.junit.After;

**import** org.junit.Test;

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

**public** **class** SimpleCalculatorTest {

**private** SimpleCalculator calculator;

@Before

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

calculator = **new** SimpleCalculator();

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

}

@After

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

calculator = **null**;

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

}

@Test

**public** **void** testAddition() {

**int** a = 10;

**int** b = 5;

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

*assertEquals*(15, result);

}

**public** **void** testDivision() {

**int** a = 20;

**int** b = 4;

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

*assertEquals*(5, result);

}

**public** **void** testMultiply()

{

**int** a=3;

**int** b=2;

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

*assertEquals*(6,result);

}

**public** **void** testSubtraction()

{

**int** a=9;

**int** b=2;

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

*assertEquals*(7,result);

}

}

**Output:**  
