**WEEK 2:**

**Exercise 1: Control Structures**

**Scenario 1: The bank wants to apply a discount to loan interest rates for customers above 60 years old.**

* **CODE:**

**Step 1:To create table SeniorCustomers :**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE SeniorCustomers';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

CREATE TABLE SeniorCustomers (

customer\_id INT PRIMARY KEY,

name VARCHAR2(100),

age INT,

loan\_interest\_rate NUMBER(5,2)

);

INSERT INTO SeniorCustomers VALUES (1, 'Ravi', 65, 9.0);

INSERT INTO SeniorCustomers VALUES (2, 'Asha', 58, 8.5);

INSERT INTO SeniorCustomers VALUES (3, 'Vikram', 70, 10.0);

COMMIT;  
  
**Step 2: Discount for Senior Customers**

BEGIN

FOR cust\_rec IN (

SELECT customer\_id

FROM SeniorCustomers

WHERE age > 60

) LOOP

UPDATE SeniorCustomers

SET loan\_interest\_rate = loan\_interest\_rate - 1

WHERE customer\_id = cust\_rec.customer\_id;

END LOOP;

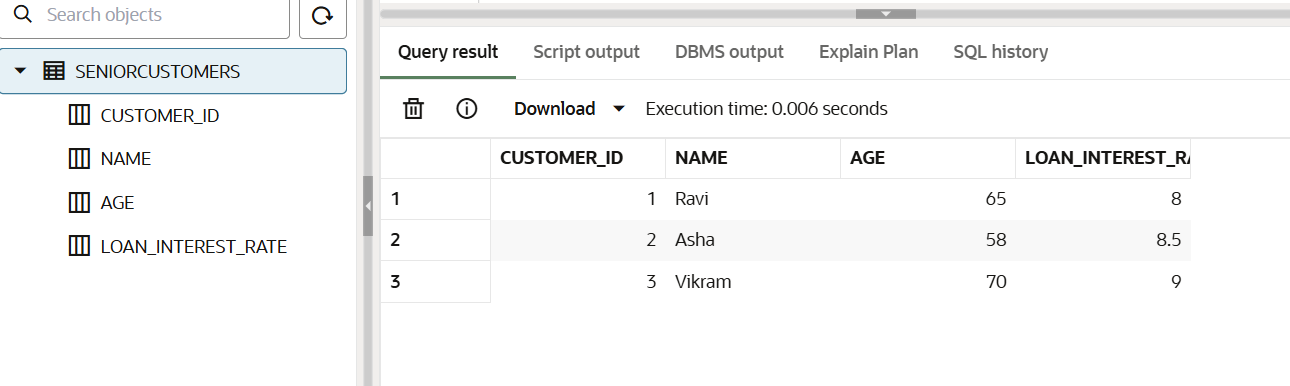
COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Interest discount applied.');

END;

/

* **OUTPUT:**



**Scenario 2: A customer can be promoted to VIP status based on their balance.**

* **CODE:**

Step 1: To create table Customers:

-- Drop the table if it exists

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

-- Create Customers table

CREATE TABLE Customers (

customer\_id INT PRIMARY KEY,

name VARCHAR2(100),

balance NUMBER(10,2),

IsVIP VARCHAR2(5) DEFAULT 'FALSE'

);

-- Insert sample data

INSERT INTO Customers VALUES (1, 'Neha', 9000, 'FALSE');

INSERT INTO Customers VALUES (2, 'Rahul', 15000, 'FALSE');

INSERT INTO Customers VALUES (3, 'Sonia', 25000, 'FALSE');

INSERT INTO Customers VALUES (4, 'Aman', 7000, 'FALSE');

COMMIT;

Step 2: PL/SQL Block to Promote to VIP

BEGIN

FOR cust\_rec IN (

SELECT customer\_id

FROM Customers

WHERE balance > 10000

) LOOP

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE customer\_id = cust\_rec.customer\_id;

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('🏅 VIP status updated.');

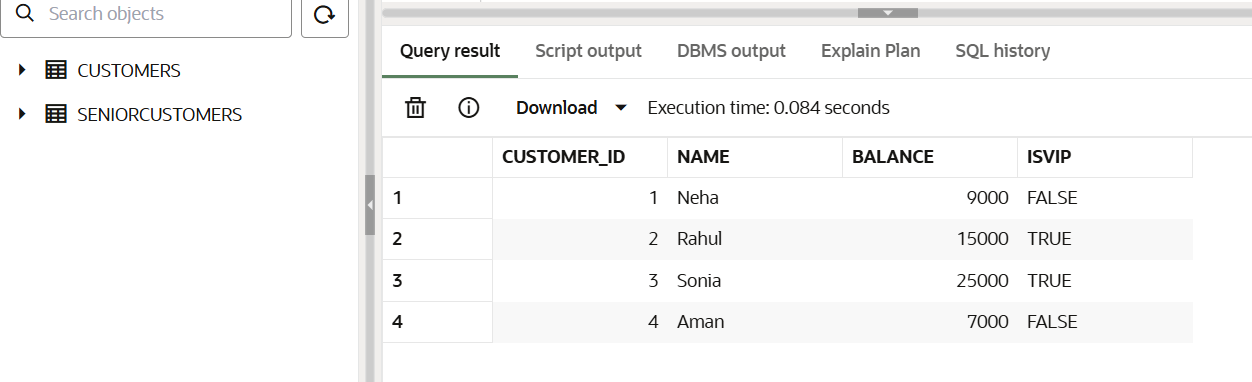
END;

/

SELECT customer\_id, name, balance, IsVIP

FROM Customers;

* **OUTPUT:**



**Scenario 3: The bank wants to send reminders to customers whose loans are due within the next 30 days.**

**CODE:**

**Step 1: To create table LoanAccounts and LoanCustomers**

CREATE TABLE LoanCustomers (

    customer\_id INT PRIMARY KEY,

    name VARCHAR2(100)

);

CREATE TABLE LoanAccounts (

    loan\_id INT PRIMARY KEY,

    customer\_id INT,

    due\_date DATE,

    FOREIGN KEY (customer\_id) REFERENCES LoanCustomers(customer\_id)

);

INSERT INTO LoanCustomers VALUES (1, 'Karan');

INSERT INTO LoanCustomers VALUES (2, 'Meena');

INSERT INTO LoanCustomers VALUES (3, 'Ajay');

INSERT INTO LoanAccounts VALUES (101, 1, SYSDATE + 15);

INSERT INTO LoanAccounts VALUES (102, 2, SYSDATE + 40);

INSERT INTO LoanAccounts VALUES (103, 3, SYSDATE + 5);

COMMIT;  
  
Step 2: Loan Due Reminder for Next 30 Days

BEGIN

FOR loan\_rec IN (

SELECT l.loan\_id, c.name, l.due\_date

FROM LoanAccounts l

JOIN LoanCustomers c ON l.customer\_id = c.customer\_id

WHERE l.due\_date <= SYSDATE + 30

) LOOP

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

' for ' || loan\_rec.name ||

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

END LOOP;

END;

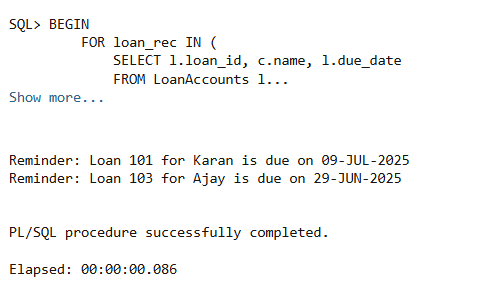
/

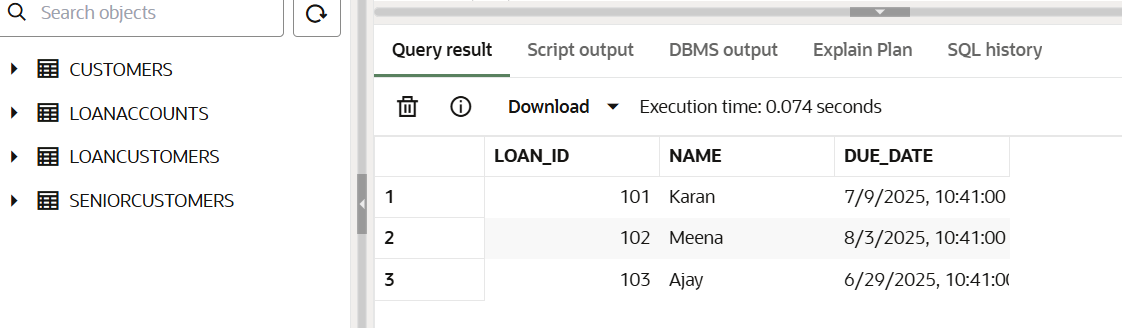
SELECT l.loan\_id, c.name, l.due\_date

FROM LoanAccounts l

JOIN LoanCustomers c ON l.customer\_id = c.customer\_id;

* **OUTPUT:**

****



**Exercise 3: Stored Procedures**

**Scenario 1: The bank needs to process monthly interest for all savings accounts.**

* **CODE:**

Step 1: Create Table – SavingsAccounts

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE SavingsAccounts';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE SavingsAccounts (

account\_id INT PRIMARY KEY,

customer\_name VARCHAR2(100),

balance NUMBER(10, 2)

);

-- Sample data

INSERT INTO SavingsAccounts VALUES (1, 'Anjali', 5000);

INSERT INTO SavingsAccounts VALUES (2, 'Raj', 8000);

INSERT INTO SavingsAccounts VALUES (3, 'Tina', 10000);

COMMIT;

Step 2: Stored Procedure – ProcessMonthlyInterest

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE SavingsAccounts

SET balance = balance + (balance \* 0.01);

DBMS\_OUTPUT.PUT\_LINE('Monthly interest of 1% applied to all savings accounts.');

END;

/

Step 3: Run Procedure

sql

CopyEdit

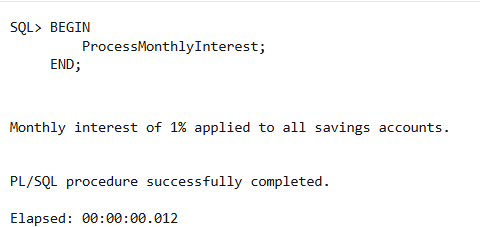
BEGIN

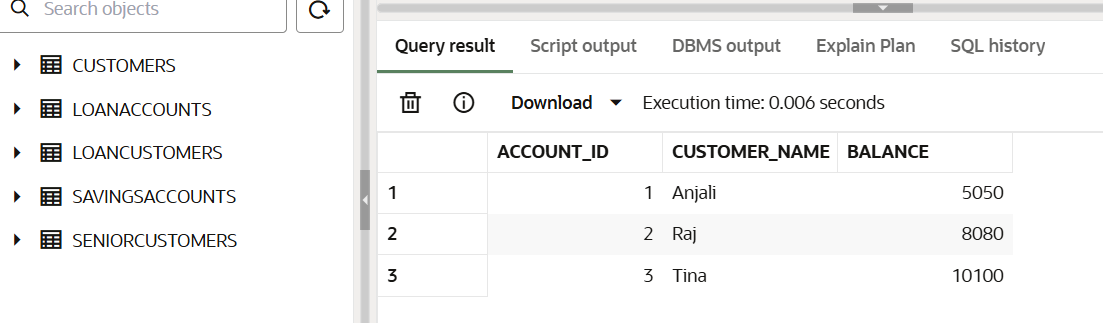
ProcessMonthlyInterest;

END;

SELECT \* FROM SavingsAccounts;

* **OUTPUT:**





**Scenario 2:** The bank wants to implement a bonus scheme for employees based on their performance.

* **CODE:**

**Step 1: Create Table – Employees**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Employees (

emp\_id INT PRIMARY KEY,

name VARCHAR2(100),

department VARCHAR2(50),

salary NUMBER(10,2)

);

-- Sample data

INSERT INTO Employees VALUES (101, 'Amit', 'Sales', 30000);

INSERT INTO Employees VALUES (102, 'Sneha', 'IT', 40000);

INSERT INTO Employees VALUES (103, 'Vijay', 'Sales', 35000);

COMMIT;

**Step 2: Stored Procedure – UpdateEmployeeBonus**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

dept\_name IN VARCHAR2,

bonus\_percent IN NUMBER

) IS

BEGIN

UPDATE Employees

SET salary = salary + (salary \* bonus\_percent / 100)

WHERE department = dept\_name;

DBMS\_OUTPUT.PUT\_LINE('Bonus of ' || bonus\_percent || '% applied to ' || dept\_name || ' department.');

END;

/

**Step 3: Run Procedure**

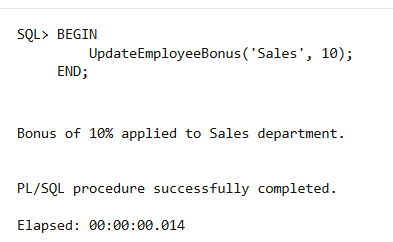
BEGIN

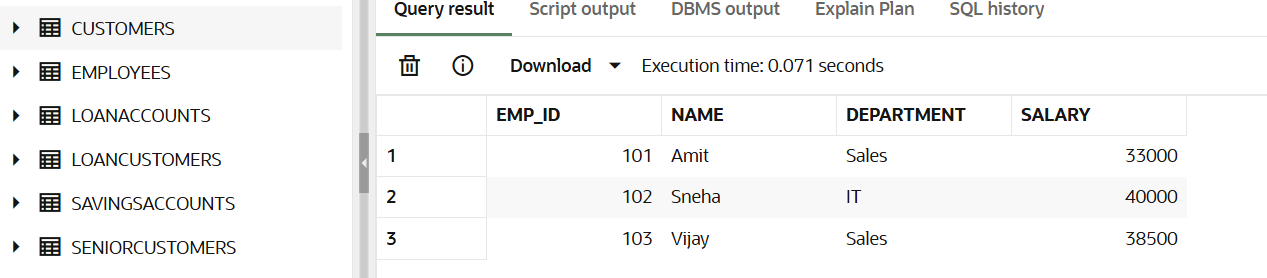
UpdateEmployeeBonus('Sales', 10);

END;

SELECT \* FROM Employees;

* **OUTPUT:**

****



**Scenario 3: Customers should be able to transfer funds between their accounts.**

* **CODE:**

**Step 1: Create Table – BankAccounts**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE BankAccounts';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE BankAccounts (

account\_id INT PRIMARY KEY,

holder\_name VARCHAR2(100),

balance NUMBER(10,2)

);

-- Sample data

INSERT INTO BankAccounts VALUES (201, 'Kiran', 7000);

INSERT INTO BankAccounts VALUES (202, 'Meera', 6000);

COMMIT;

**Step 2: Stored Procedure – TransferFunds**

CREATE OR REPLACE PROCEDURE TransferFunds (

from\_acc IN INT,

to\_acc IN INT,

amount IN NUMBER

) IS

from\_balance NUMBER;

BEGIN

-- Check balance

SELECT balance INTO from\_balance FROM BankAccounts WHERE account\_id = from\_acc;

IF from\_balance < amount THEN

DBMS\_OUTPUT.PUT\_LINE('Insufficient balance. Transfer aborted.');

ELSE

UPDATE BankAccounts SET balance = balance - amount WHERE account\_id = from\_acc;

UPDATE BankAccounts SET balance = balance + amount WHERE account\_id = to\_acc;

DBMS\_OUTPUT.PUT\_LINE('Transfer of ' || amount || ' completed from account ' || from\_acc || ' to ' || to\_acc);

END IF;

END;

/

**Step 3: Run Procedure**

BEGIN

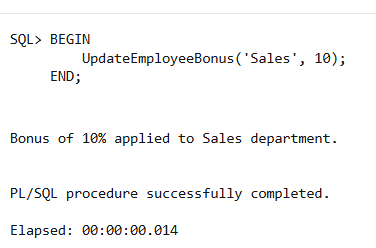
TransferFunds(201, 202, 2000);

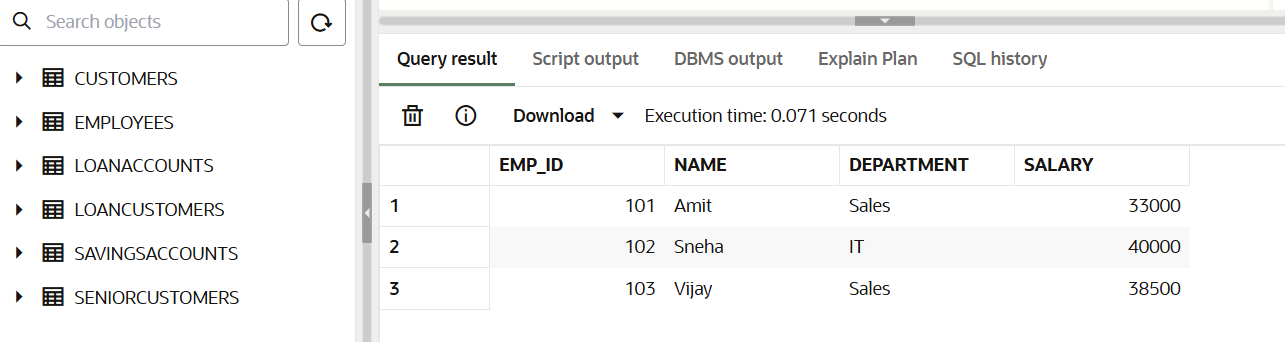
END;

/

SELECT \* FROM BankAccounts;

* **OUTPUT:**





**JUnit Testing Exercises**

**Setting Up JUnit**

* **CODE:**

1. Calculator.java

package com.example;

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class CalculatorTest {

@Test

public void testMultiply() {

Calculator calc = new Calculator();

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

assertEquals(12, result);

}

}

1. CalculatorTest.java

package com.example;

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class CalculatorTest {

@Test

public void testMultiply() {

Calculator calc = new Calculator();

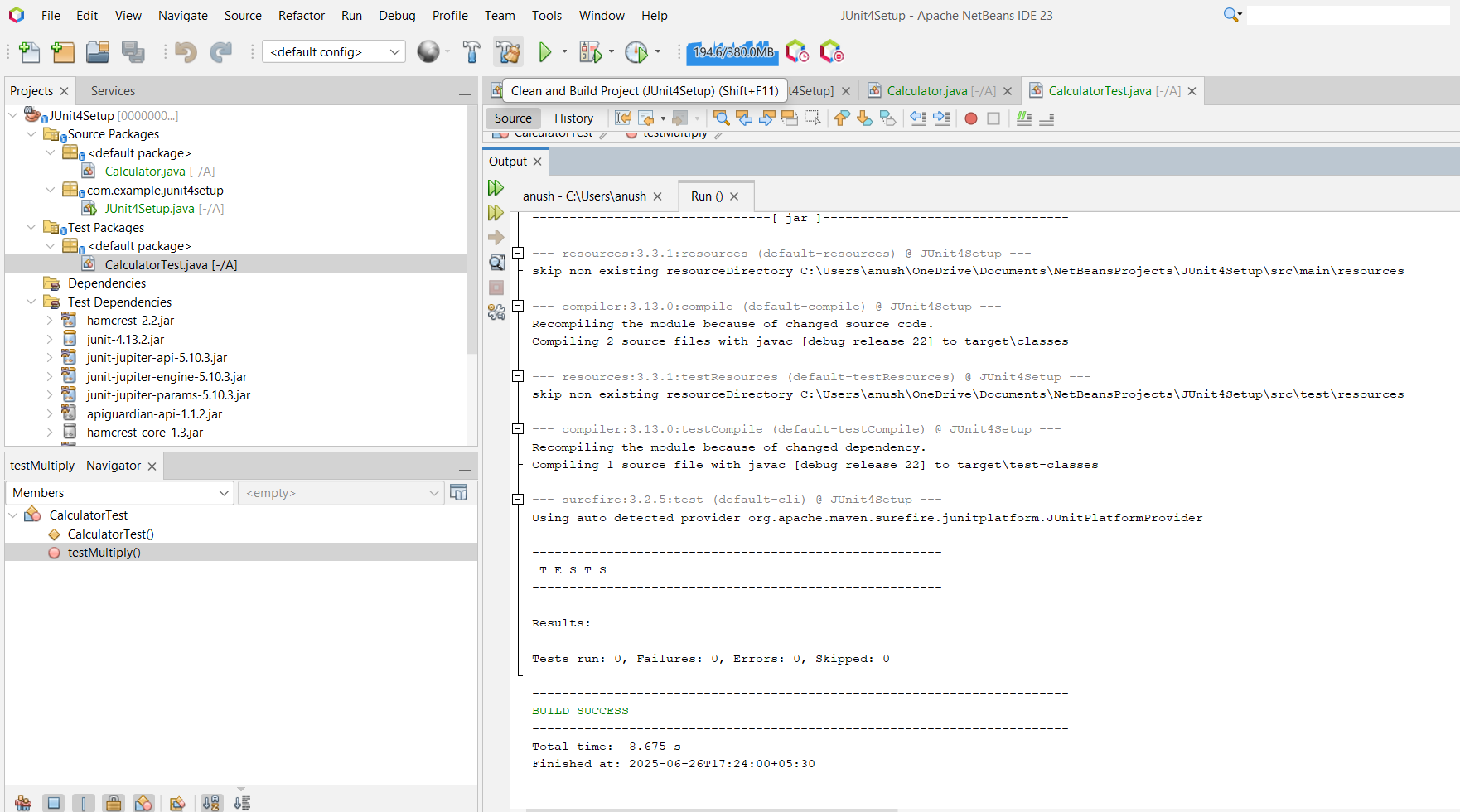
int result = calc.multiply(3, 4);

assertEquals(12, result);

}

}

* **OUTPUT**



**Exercise 3: Assertions in JUnit**

* **CODE:**

package com.example.junit4setup;

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionTest {

@Test

public void testAssertions() {

// Assert equals

assertEquals(5, 2 + 3);

// Assert true

assertTrue(5 > 3);

// Assert false

assertFalse(5 < 3);

// Assert null

assertNull(null);

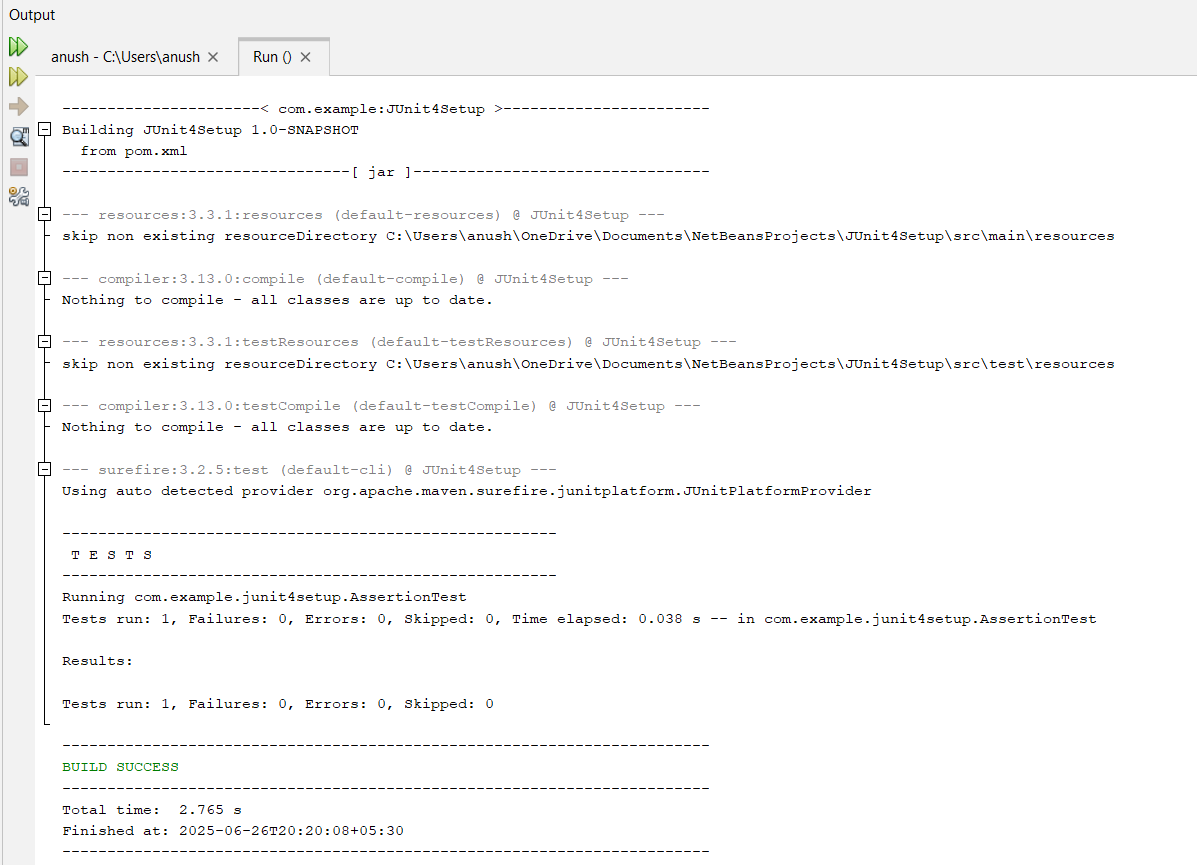
// Assert not null

assertNotNull(new Object());

}

}

* **OUTPUT:**

****

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

* **CODE:**

1. **Calculator.java**

package com.example.junit4setup;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

**2.CalculatorTest.java**

package com.example.junit4setup;

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();

System.out.println("setUp() - Calculator instance created");

}

@After

public void tearDown() {

calculator = null;

System.out.println(" tearDown() - Calculator instance set to null");

}

@Test

public void testAdd\_PositiveNumbers() {

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

assertEquals(8, result);

}

@Test

public void testAdd\_NegativeNumbers() {

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

assertEquals(-5, result);

}

@Test

public void testAdd\_MixedNumbers() {

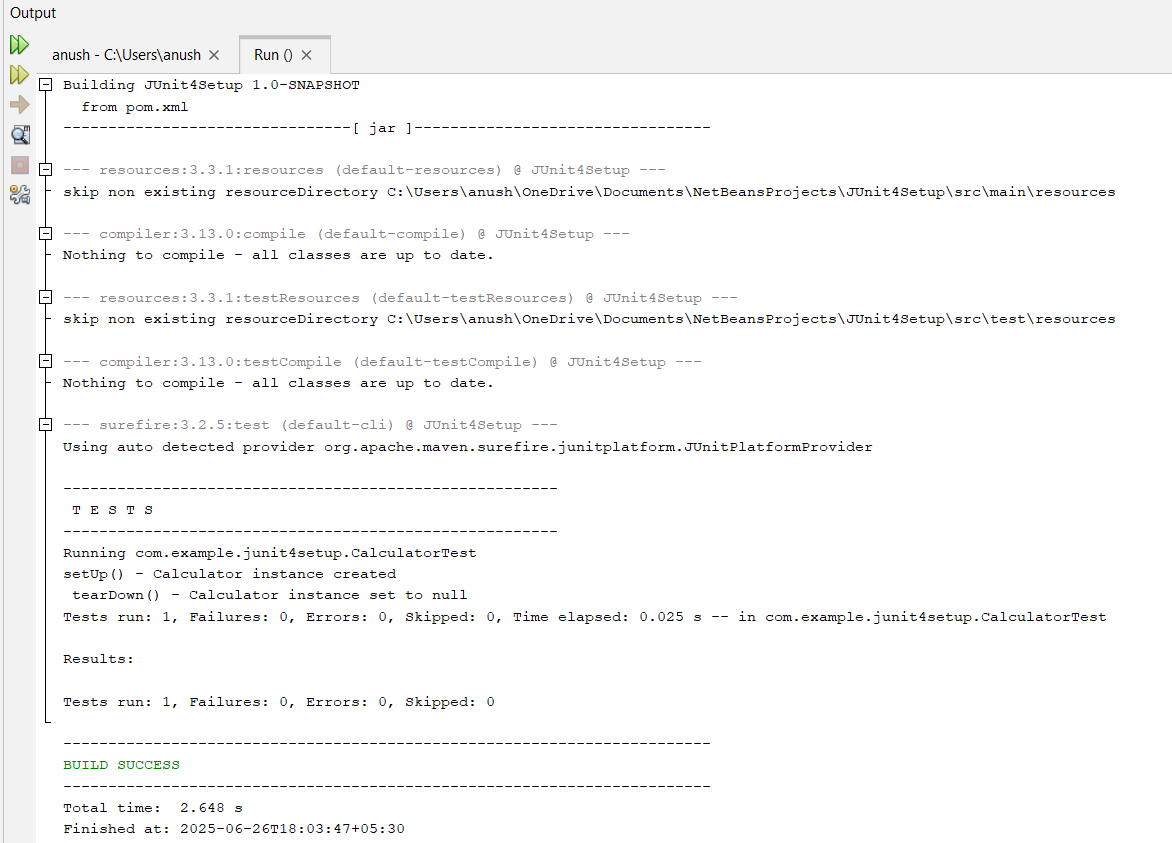
int result = calculator.add(-2, 5);

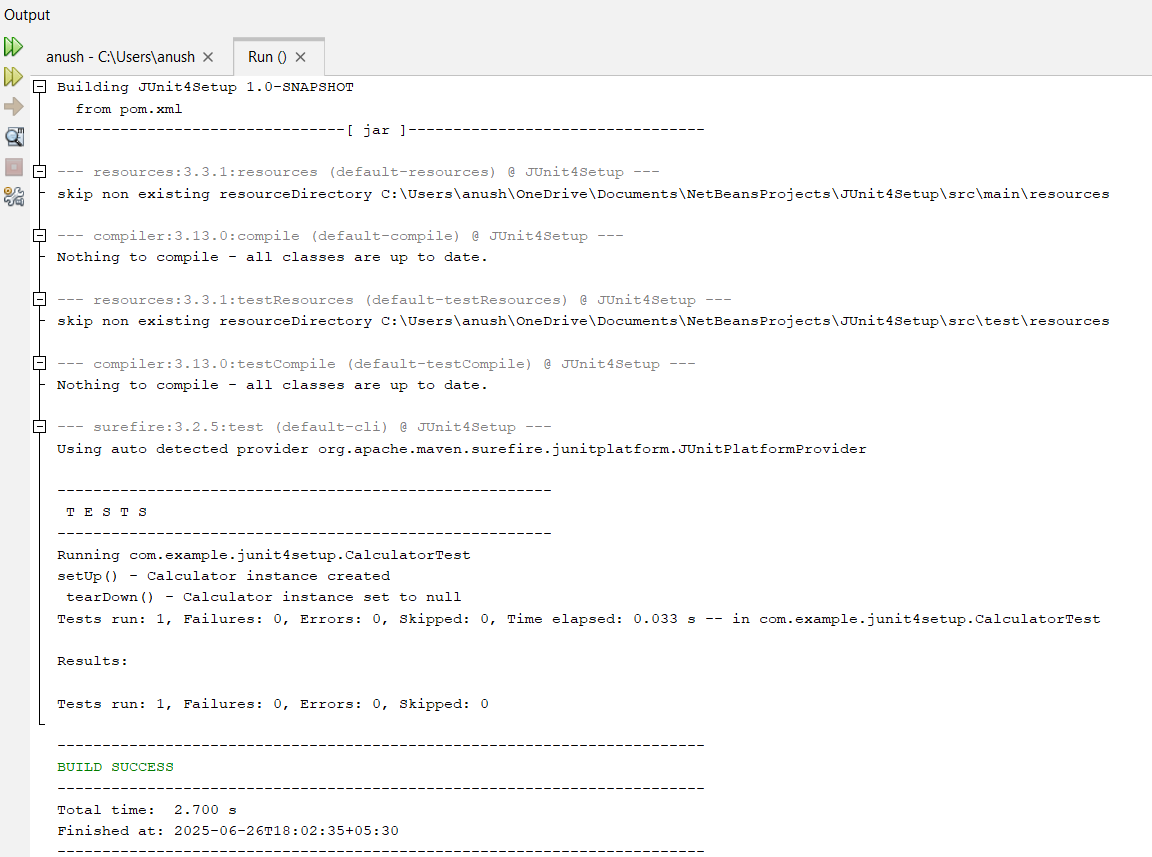
assertEquals(3, result);

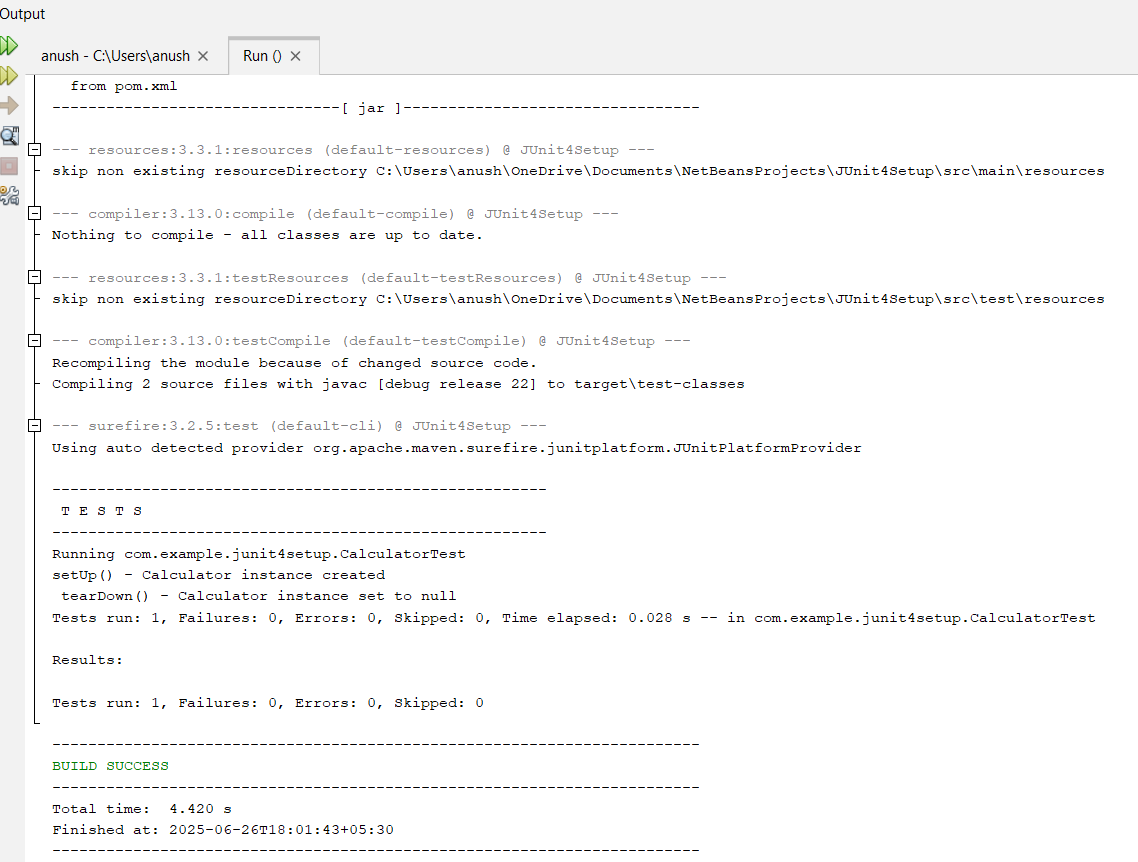
}

}

* **OUTPUT:**

****





**Mockito Hands-On Exercises**

**Exercise 1: Mocking and Stubbing**

* **CODE:**

1. **ExternalApi.java**

package com.example.junit4setup;

public interface ExternalApi {

String getData();

}

1. **MyService.java**

package com.example.junit4setup;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}public String fetchData() {

return api.getData();

}

}

1. **MyServiceTest**

package com.example.mockdemo;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

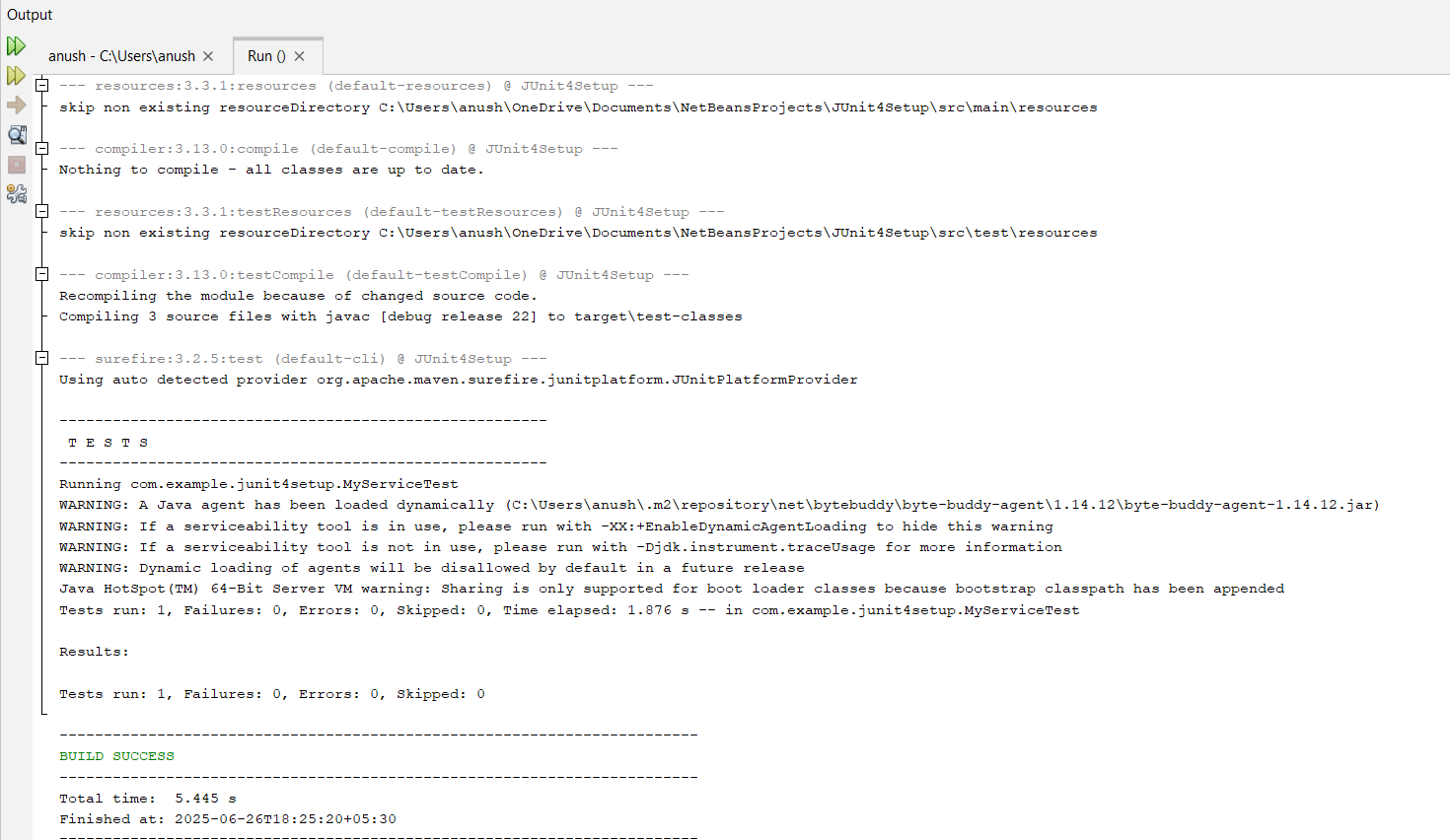
public String fetchData() {

return api.getData();

}

}

* **OUTPUT:**

****

**Exercise 2: Verifying Interactions**

* **CODE:**

**MyServiceTest.java**package com.example.junit4setup;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

// Step 1: Create a mock object

ExternalApi mockApi = mock(ExternalApi.class);

// Step 2: Inject the mock into the service

MyService service = new MyService(mockApi);

// Step 3: Call the method

service.fetchData();

// Step 4: Verify that getData() was called on the mock

verify(mockApi).getData();

}

}

* **OUTPUT:**



**Logging using SLF4J Exercise 1:**

**Logging Error Messages and Warning Levels**

* **CODE:**

**LoggingExample.java**

package com.example.junit4setup;

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");

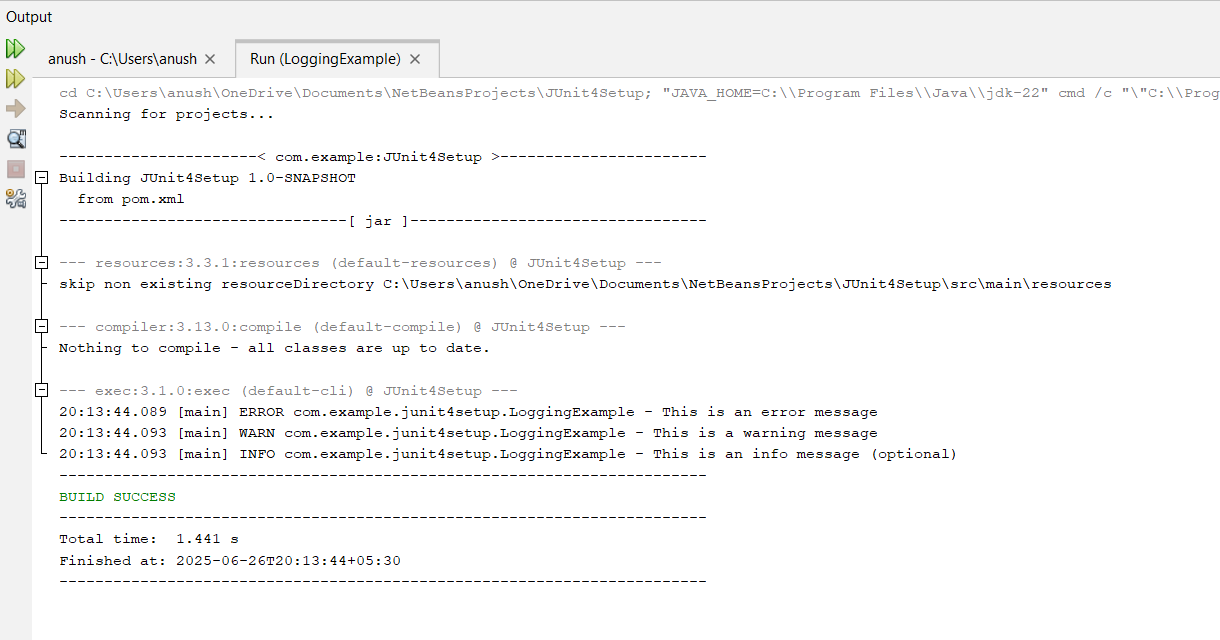
logger.warn("️This is a warning message");

logger.info("This is an info message (optional)");

}

}

* **OUTPUT:**

****