WEEK-2

PL-SQL

Exercise 1: Control Structures

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

Question: Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

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

Question: Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.

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

Question: Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

Code:

```
 CREATE TABLE customers (
 cust_id   NUMBER PRIMARY KEY,
 age       NUMBER,
 balance   NUMBER,
 vip_flag  VARCHAR2(5)
);

CREATE TABLE loans (
 loan_id   NUMBER PRIMARY KEY,
 cust_id   NUMBER,
 int_rate  NUMBER,
 due_on    DATE,
 FOREIGN KEY (cust_id) REFERENCES customers(cust_id)
);
```

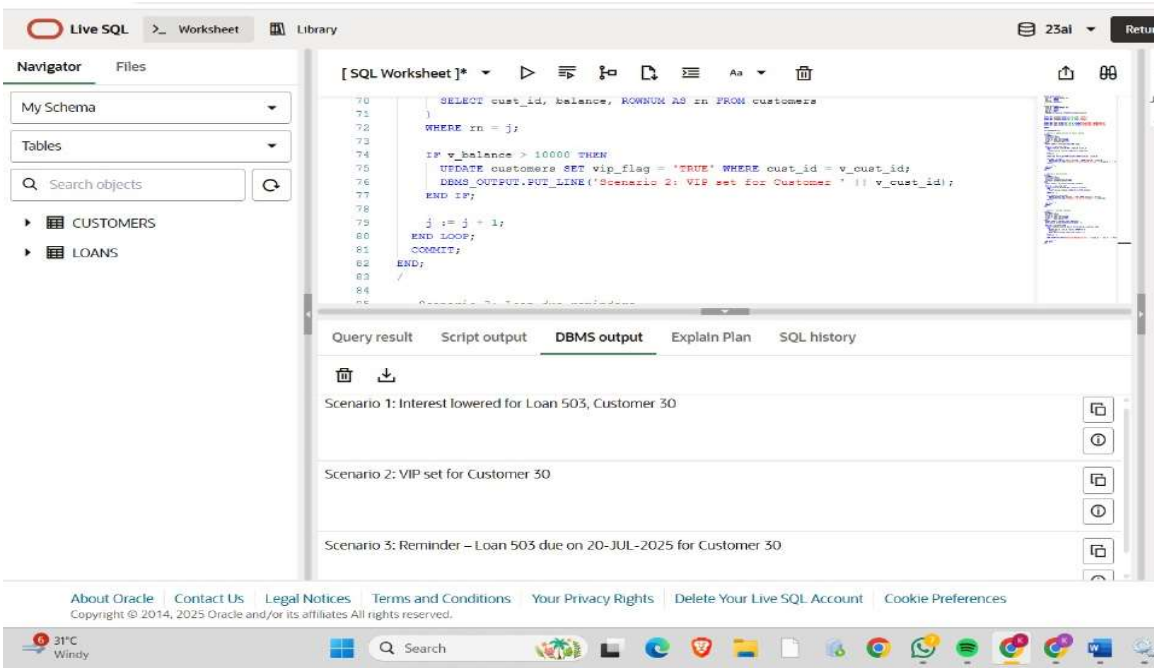
```
INSERT INTO customers VALUES (10, 68, 12500, 'FALSE');
INSERT INTO customers VALUES (20, 55, 9400, 'FALSE');
INSERT INTO customers VALUES (30, 72, 17800, 'FALSE');

INSERT INTO loans VALUES (501, 10, 11, TO_DATE('05-JUL-2025','DD-MON-YYYY'));
INSERT INTO loans VALUES (502, 20, 9, TO_DATE('15-AUG-2025','DD-MON-YYYY'));
INSERT INTO loans VALUES (503, 30, 10, TO_DATE('20-JUL-2025','DD-MON-YYYY'));

COMMIT;

SET SERVEROUTPUT ON;

-- Scenario 1: Reduce interest for senior citizens
DECLARE
  i NUMBER := 1;
  loan_total NUMBER;
  v_loan_id loans.loan_id%TYPE;
  v_cust_id loans.cust_id%TYPE;
  v_age     customers.age%TYPE;
BEGIN
  SELECT COUNT(*) INTO loan_total FROM loans;

  WHILE i <= loan_total LOOP
    SELECT loan_id, cust_id INTO v_loan_id, v_cust_id
    FROM (
      SELECT loan_id, cust_id, ROWNUM AS rn FROM loans
    )
    WHERE rn = i;

    SELECT age INTO v_age FROM customers WHERE cust_id = v_cust_id;
```

```
    IF v_age > 60 THEN
      UPDATE loans SET int_rate = int_rate - 1 WHERE loan_id = v_loan_id;
      DBMS_OUTPUT.PUT_LINE('Scenario 1: Interest lowered for Loan ' || v_loan_id || ', Customer ' ||
v_cust_id);
    END IF;

    i := i + 1;
  END LOOP;
  COMMIT;
END;
/

-- Scenario 2: Set VIP status
DECLARE
  j NUMBER := 1;
  cust_total NUMBER;
  v_cust_id customers.cust_id%TYPE;
  v_balance customers.balance%TYPE;
BEGIN
  SELECT COUNT(*) INTO cust_total FROM customers;

  WHILE j <= cust_total LOOP
    SELECT cust_id, balance INTO v_cust_id, v_balance
    FROM (
      SELECT cust_id, balance, ROWNUM AS rn FROM customers
    )
    WHERE rn = j;

    IF v_balance > 10000 THEN
      UPDATE customers SET vip_flag = 'TRUE' WHERE cust_id = v_cust_id;
      DBMS_OUTPUT.PUT_LINE('Scenario 2: VIP set for Customer ' || v_cust_id);
```

```
    END IF;

    j := j + 1;
  END LOOP;
  COMMIT;
END;
/

-- Scenario 3: Loan due reminders
DECLARE
  k NUMBER := 1;
  due_total NUMBER;
  v_loan_id loans.loan_id%TYPE;
  v_cust_id loans.cust_id%TYPE;
  v_due     loans.due_on%TYPE;
BEGIN
  SELECT COUNT(*) INTO due_total FROM loans
  WHERE due_on BETWEEN SYSDATE AND SYSDATE + 30;

  WHILE k <= due_total LOOP
    SELECT loan_id, cust_id, due_on INTO v_loan_id, v_cust_id, v_due
    FROM (
      SELECT loan_id, cust_id, due_on, ROWNUM AS rn
      FROM loans
      WHERE due_on BETWEEN SYSDATE AND SYSDATE + 30
    )
    WHERE rn = k;

    DBMS_OUTPUT.PUT_LINE('Scenario 3: Reminder – Loan ' || v_loan_id || ' due on ' ||
TO_CHAR(v_due, 'DD-MON-YYYY') || ' for Customer ' || v_cust_id);
```

```
    k := k + 1;
  END LOOP;
END;
```

OUTPUT:



Exercise 3: Stored Procedures

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

- **Question:** Write a stored procedure **ProcessMonthlyInterest** that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

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

- **Question:** Write a stored procedure **UpdateEmployeeBonus** that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

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

- **Question:** Write a stored procedure **TransferFunds** that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer.

Code :

```
-- Scenario 1: Process Monthly Interest
BEGIN
  EXECUTE IMMEDIATE 'DROP TABLE savings_accounts';
EXCEPTION WHEN OTHERS THEN NULL;
END;
/

CREATE TABLE savings_accounts (
  account_id     NUMBER PRIMARY KEY,
  customer_name  VARCHAR2(50),
  balance        NUMBER
);
```

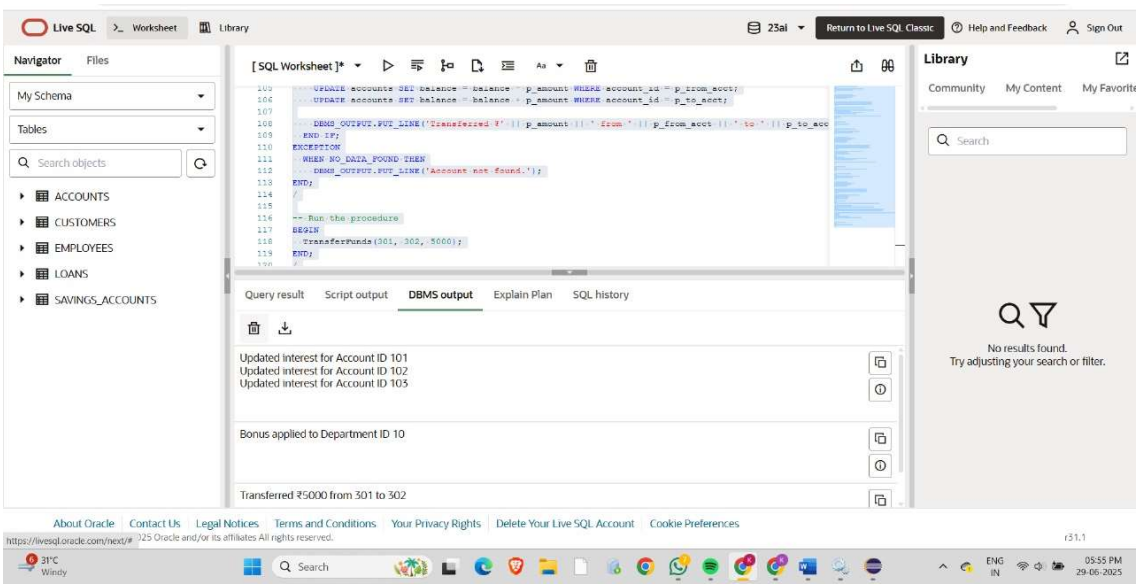
```
INSERT INTO savings_accounts VALUES (101, 'Ravi Kumar', 10000);
INSERT INTO savings_accounts VALUES (102, 'Anita Sharma', 15000);
INSERT INTO savings_accounts VALUES (103, 'Kiran Patel', 8000);
COMMIT;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS
BEGIN
  FOR acc IN (SELECT account_id, balance FROM savings_accounts) LOOP
    UPDATE savings_accounts
    SET balance = balance + (acc.balance * 0.01)
    WHERE account_id = acc.account_id;

    DBMS_OUTPUT.PUT_LINE('Updated interest for Account ID ' || acc.account_id);
  END LOOP;
END;
/

-- Run the procedure
BEGIN
  ProcessMonthlyInterest;
END;
/

-- Scenario 2: Update Employee Bonus
BEGIN
  EXECUTE IMMEDIATE 'DROP TABLE employees';
EXCEPTION WHEN OTHERS THEN NULL;
END;
/
```

```
CREATE TABLE employees (
  emp_id        NUMBER PRIMARY KEY,
  emp_name      VARCHAR2(50),
  department_id NUMBER,
  salary        NUMBER
);

INSERT INTO employees VALUES (201, 'Suresh Babu', 10, 40000);
INSERT INTO employees VALUES (202, 'Priya Raj', 10, 45000);
INSERT INTO employees VALUES (203, 'Vikram Desai', 20, 42000);
COMMIT;

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(
  p_dept_id IN NUMBER,
  p_bonus_percent IN NUMBER
) IS
BEGIN
  UPDATE employees
  SET salary = salary + (salary * p_bonus_percent / 100)
  WHERE department_id = p_dept_id;

  DBMS_OUTPUT.PUT_LINE('Bonus applied to Department ID ' || p_dept_id);
END;
/

-- Run the procedure
BEGIN
  UpdateEmployeeBonus(10, 10);  -- 10% bonus to dept 10
END;
/
```

```
-- Scenario 3: Transfer Funds
BEGIN
  EXECUTE IMMEDIATE 'DROP TABLE accounts';
EXCEPTION WHEN OTHERS THEN NULL;
END;
/

CREATE TABLE accounts (
  account_id    NUMBER PRIMARY KEY,
  holder_name   VARCHAR2(50),
  balance       NUMBER
);

INSERT INTO accounts VALUES (301, 'Meena Joshi', 20000);
INSERT INTO accounts VALUES (302, 'Arjun Singh', 10000);
INSERT INTO accounts VALUES (303, 'Divya Iyer', 30000);
COMMIT;

CREATE OR REPLACE PROCEDURE TransferFunds(
  p_from_acct IN NUMBER,
  p_to_acct   IN NUMBER,
  p_amount    IN NUMBER
) IS
  v_balance NUMBER;
BEGIN
  SELECT balance INTO v_balance FROM accounts WHERE account_id = p_from_acct;

  IF v_balance < p_amount THEN
    DBMS_OUTPUT.PUT_LINE('Insufficient balance in source account.');
  ELSE
    UPDATE accounts SET balance = balance - p_amount WHERE account_id = p_from_acct;
```

```
   UPDATE accounts SET balance = balance + p_amount WHERE account_id = p_to_acct;


   DBMS_OUTPUT.PUT_LINE('Transferred ₹' || p_amount || ' from ' || p_from_acct || ' to ' ||
p_to_acct);
  END IF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
   DBMS_OUTPUT.PUT_LINE('Account not found.');
END;
/


-- Run the procedure
BEGIN
  TransferFunds(301, 302, 5000);
END;
/
```

OUTPUT:



Mockito Hands-On Exercises

Exercise 1: Mocking and Stubbing

Scenario:

You need to test a service that depends on an external API. Use Mockito to mock the external API and stub its methods.

Steps:

1. Create a mock object for the external API.
2. Stub the methods to return predefined values.
3. Write a test case that uses the mock object.

Solution Code:

```
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);
}
```

CODE:

ExternalApi.java

```
package com.example.MockDemo;
public interface ExternalApi {
String getData();
}
```

Myservice.java

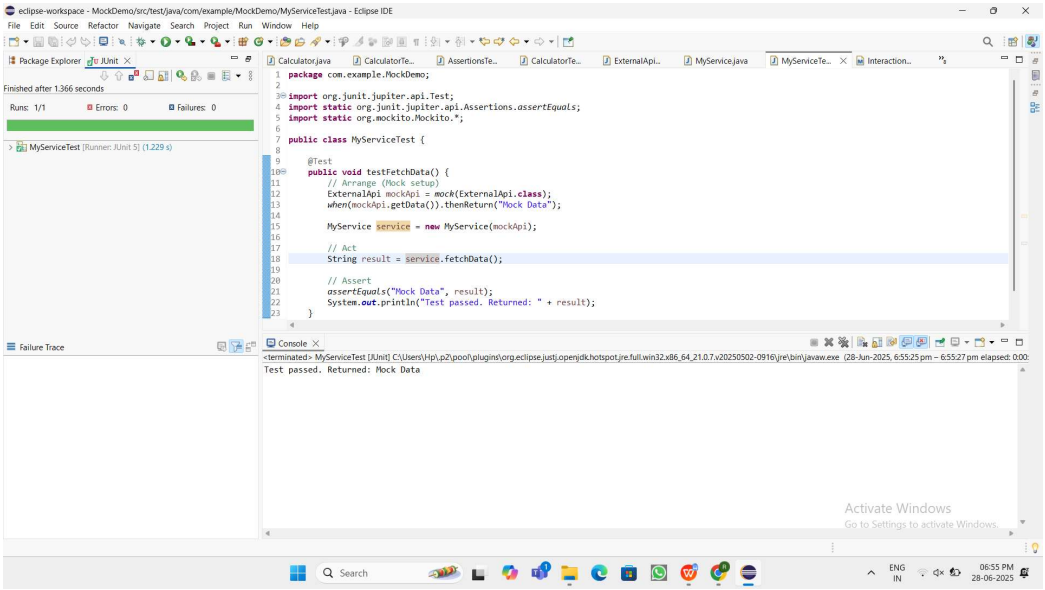
```
package com.example.MockDemo;

public class MyService {
private final ExternalApi externalApi;

public MyService(ExternalApi externalApi) {
this.externalApi = externalApi;
}
public String fetchData() {
return externalApi.getData();
}
}
```

MyServiceTest.java

```
package com.example.MockDemo;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.mockito.Mockito.*;

public class MyServiceTest {

@Test
public void testFetchData() {
// Arrange (Mock setup)
ExternalApi mockApi = mock(ExternalApi.class);
when(mockApi.getData()).thenReturn("Mock Data");
```

```
MyService service = new MyService(mockApi);
String result = service.fetchData();
assertEquals("Mock Data", result);
System.out.println("Test passed. Returned: " + result);
}
}
```

OUTPUT:



Exercise 2: Verifying Interactions

Scenario:

You need to ensure that a method is called with specific arguments.

Steps:

1. Create a mock object.
2. Call the method with specific arguments.
3. Verify the interaction.

Solution Code:

```
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();
```

CODE:

ExternalApi.java

```
package com.example.InteractionVerifier;

public interface ExternalApi {
String getData();
}
```

MyService.java

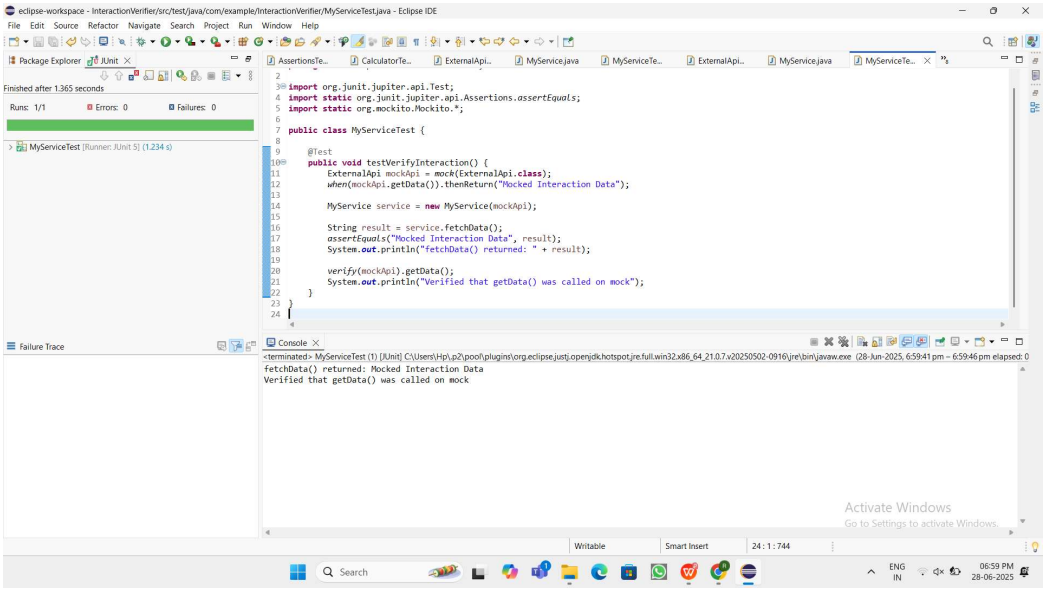
```
package com.example.InteractionVerifier;

public class MyService {
private ExternalApi externalApi;

public MyService(ExternalApi externalApi) {
this.externalApi = externalApi;
}

public String fetchData() {
return externalApi.getData();
}
}
```

MyServiceTest.java

```
package com.example.InteractionVerifier;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.mockito.Mockito.*;

public class MyServiceTest {

@Test
public void testVerifyInteraction() {
ExternalApi mockApi = mock(ExternalApi.class);
```

```
when(mockApi.getData()).thenReturn("Mocked Interaction Data");

MyService service = new MyService(mockApi);

String result = service.fetchData();
assertEquals("Mocked Interaction Data", result);
System.out.println("fetchData() returned: " + result);

verify(mockApi).getData();
System.out.println("Verified that getData() was called on mock");
}
}
```

OUTPUT:



JUnit Basic Testing

Exercise 1: Setting Up JUnit

Scenario:

You need to set up JUnit in your Java project to start writing unit tests.

Steps:

1. Create a new Java project in your IDE (e.g., IntelliJ IDEA, Eclipse).

2. Add JUnit dependency to your project. If you are using Maven, add the following to your

pom.xml:

```
<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>
```

3. Create a new test class in your project

CODE:

Calculator.java

```
package com.library.gui;


public class Calculator {


    public int add(int a, int b) {

        return a + b;
```

```
    }

    public int subtract(int a, int b) {
        return a - b;
    }

    public int multiply(int a, int b) {
        return a * b;
    }

    public int divide(int a, int b) {
        if (b == 0) {
            throw new ArithmeticException("Cannot divide by zero");
        }
        return a / b;
    }
}
```

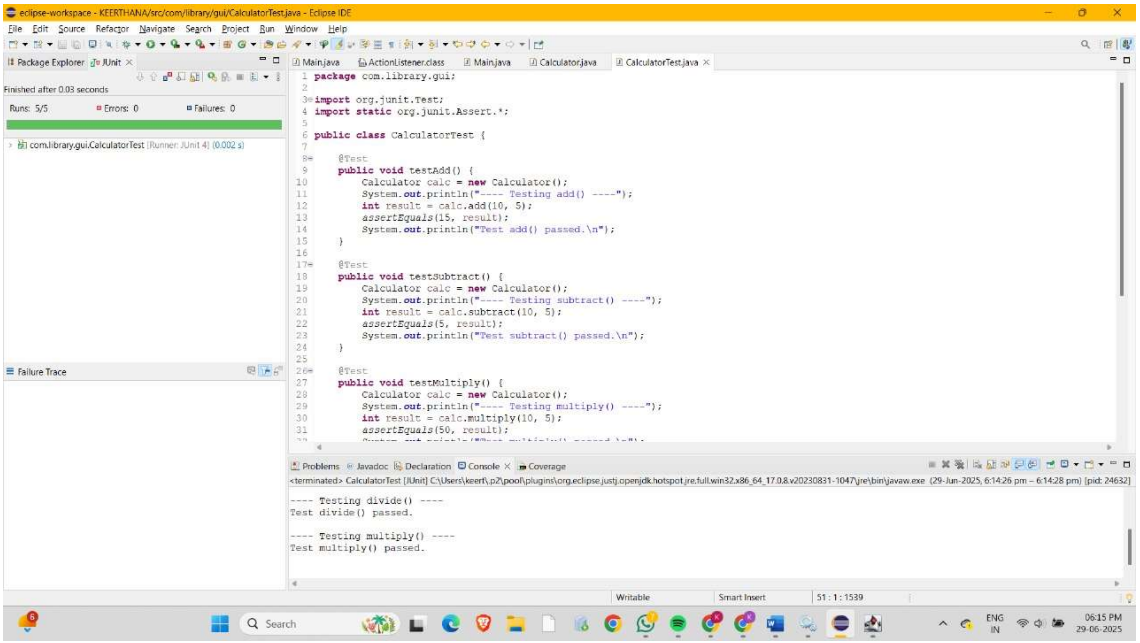
CalculatorTest.java

```
package com.library.gui;

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

public class CalculatorTest {

    @Test
```

```
public void testAdd() {
    Calculator calc = new Calculator();
    System.out.println("---- Testing add() ----");
    int result = calc.add(10, 5);
    assertEquals(15, result);
    System.out.println("Test add() passed.\n");
}

@Test
public void testSubtract() {
    Calculator calc = new Calculator();
    System.out.println("---- Testing subtract() ----");
    int result = calc.subtract(10, 5);
    assertEquals(5, result);
    System.out.println("Test subtract() passed.\n");
}

@Test
public void testMultiply() {
    Calculator calc = new Calculator();
    System.out.println("---- Testing multiply() ----");
    int result = calc.multiply(10, 5);
    assertEquals(50, result);
    System.out.println("Test multiply() passed.\n");
}
```

```
    @Test
    public void testDivide() {
        Calculator calc = new Calculator();
        System.out.println("---- Testing divide() ----");
        int result = calc.divide(10, 5);
        assertEquals(2, result);
        System.out.println("Test divide() passed.\n");
    }

    @Test(expected = ArithmeticException.class)
    public void testDivideByZero() {
        Calculator calc = new Calculator();
        System.out.println("---- Testing divide by zero ----");
        calc.divide(10, 0);  // Expected exception
    }
}
```

OUTPUT:



Exercise 3: Assertions in JUnit

Scenario:

You need to use different assertions in JUnit to validate your test results.

Steps: 1. Write tests using various JUnit assertions.

Solution Code:

```
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());
}
}
```

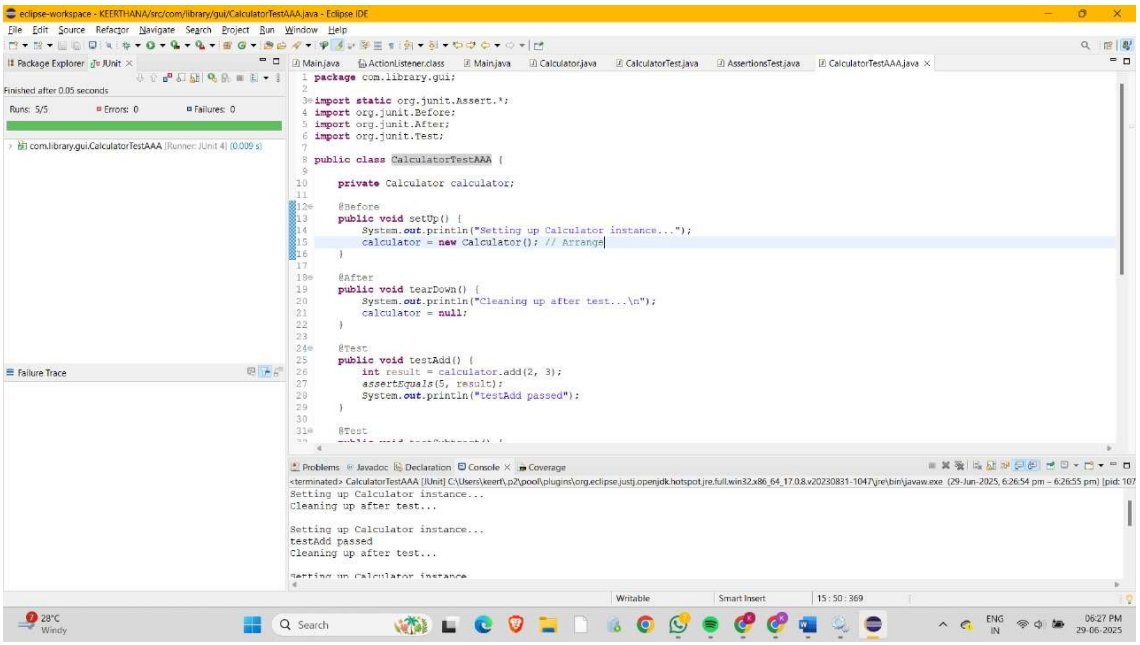
CODE:

AssertionsTest.java

```
package com.library.gui;

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

public class AssertionsTest {

    @Test
    public void testAssertions() {
        System.out.println("Running testAssertions...");

        // Assert equals
        assertEquals("Sum should be 5", 5, 2 + 3);
        System.out.println("assertEquals passed");

        // Assert true
        assertTrue("5 is greater than 3", 5 > 3);
```

```
        System.out.println("assertTrue passed");

        // Assert false
        assertFalse("5 is not less than 3", 5 < 3);
        System.out.println("assertFalse passed");

        // Assert null
        assertNull("Value should be null", null);
        System.out.println("assertNull passed");

        // Assert not null
        assertNotNull("Object should not be null", new Object());
        System.out.println("assertNotNull passed");
    }
}
```

OUTPUT:



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

Scenario:

You need to organize your tests using the Arrange-Act-Assert (AAA) pattern and use setup

and teardown methods.

Steps:

1. Write tests using the AAA pattern.

2. Use @Before and @After annotations for setup and teardown methods

CODE:

CalculatorTestAAA.java

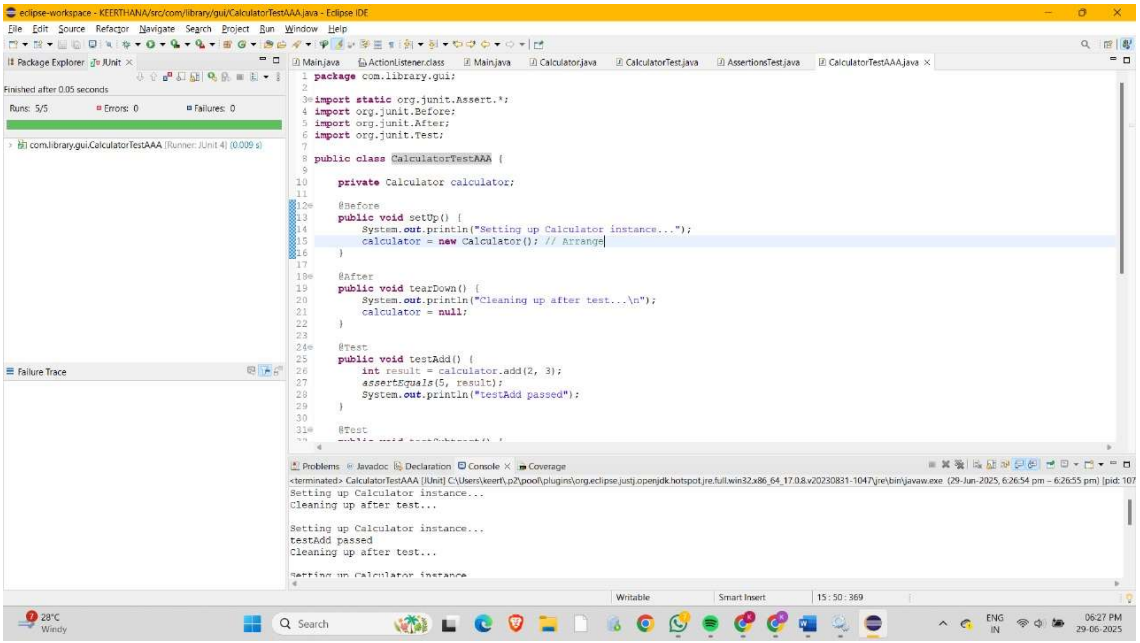
package com.library.gui;

```
import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.After;
import org.junit.Test;

public class CalculatorTestAAA {

    private Calculator calculator;

    @Before
    public void setUp() {
        System.out.println("Setting up Calculator instance...");
        calculator = new Calculator(); // Arrange
    }

    @After
    public void tearDown() {
        System.out.println("Cleaning up after test...\n");
        calculator = null;
    }

    @Test
    public void testAdd() {
        int result = calculator.add(2, 3);
        assertEquals(5, result);
        System.out.println("testAdd passed");
```

```
}

@Test
public void testSubtract() {
    int result = calculator.subtract(10, 4);
    assertEquals(6, result);
    System.out.println("testSubtract passed");
}

@Test
public void testMultiply() {
    int result = calculator.multiply(4, 3);
    assertEquals(12, result);
    System.out.println("testMultiply passed");
}

@Test
public void testDivide() {
    int result = calculator.divide(20, 5);
    assertEquals(4, result);
    System.out.println("testDivide passed");
}

@Test(expected = ArithmeticException.class)
public void testDivideByZero() {
    calculator.divide(10, 0);
```

```
    }

}
```

OUTPUT:


eclipse-workspace - KEERTHANA/src/com/library/gui/CalculatorTestAAA.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JUnit
Finished after 0.05 seconds
Runs: 5/5
Errors: 0
Failures: 0
com.library.gui.CalculatorTestAAA [Runner: JUnit 4] (0.009 s)
Failure Trace
Main.java
ActionListener.class
Main.java
Calculator.java
CalculatorTest.java
AssertionsTest.java
CalculatorTestAAA.java
package com.library.gui;
import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.After;
import org.junit.Test;
public class CalculatorTestAAA {
private Calculator calculator;
@Before
public void setUp() {
System.out.println("Setting up Calculator instance...");
calculator = new Calculator(); // Arrange
}
@After
public void tearDown() {
System.out.println("Cleaning up after test...\n");
calculator = null;
}
@Test
public void testAdd() {
int result = calculator.add(2, 3);
assertEquals(5, result);
System.out.println("testAdd passed");
}
@Test
Problems
Javadoc
Declaration
Console
Coverage
<terminated> CalculatorTestAAA [JUnit] C:\Users\keert\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.8.v20230831-1047\jre\bin\javaw.exe (29-Jun-2025, 6:26:54 pm – 6:26:55 pm) [pid: 107
Setting up Calculator instance...
Cleaning up after test...
Setting up Calculator instance...
testAdd passed
Cleaning up after test...
Setting up Calculator instance
Writable
Smart Insert
15 : 50 : 369
28°C
Windy
Search
ENG
IN
06:27 PM
29-06-2025

Logging using SLF4J

Exercise 1: Logging Error Messages and Warning Levels

Task: Write a Java application that demonstrates logging error messages and warning levels using SLF4J.

Step-by-Step Solution:

1. Add SLF4J and Logback dependencies to your `pom.xml` file:

```
<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>
```

2. Create a Java class that uses SLF4J for 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");
}
}
```

CODE:

LoggingExample.java

```
package com.example.LoggingDemonew;
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");
      logger.debug("This is a DEBUG message");
   }
}
```

OUTPUT: