WEEK 2 PLSQL EXCERCISES

1.Control Structures

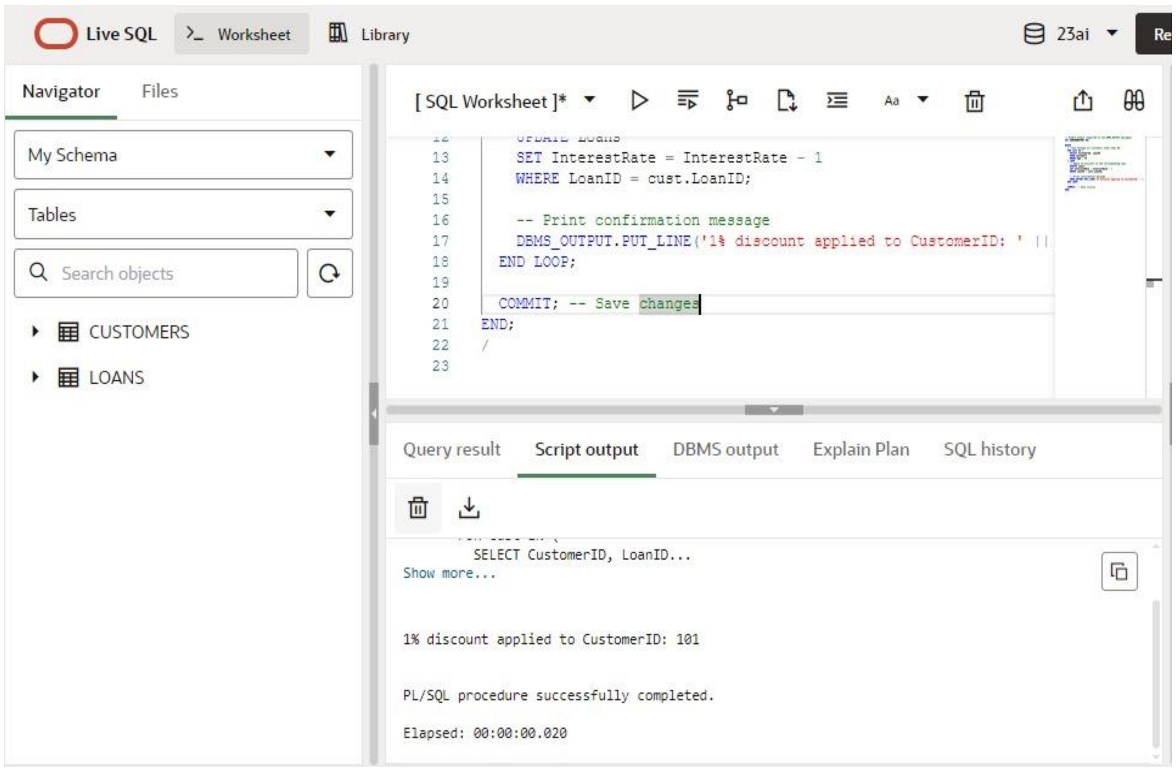
Scenario 1: The bank wants to apply a discount to loan interest rates for customers above 60 years old.

- **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.

-- Enable output (required to see DBMS_OUTPUT messages)

CODE:

```
SET SERVEROUTPUT ON;
BEGIN
  -- Loop through all customers older than 60
  FOR cust IN (
    SELECT CustomerID, LoanID
    FROM Customers
    WHERE Age > 60
  ) LOOP
    -- Apply 1% discount to the corresponding loan
    UPDATE Loans
    SET InterestRate = InterestRate - 1
    WHERE LoanID = cust.LoanID;

    -- Print confirmation message
    DBMS_OUTPUT.PUT_LINE('1% discount applied to CustomerID: ' || cust.CustomerID);
  END LOOP;

  COMMIT; -- Save changes
END;
/
```



```
    UPDATE Loans
    SET InterestRate = InterestRate - 1
    WHERE LoanID = cust.LoanID;

    -- Print confirmation message
    DBMS_OUTPUT.PUT_LINE('1% discount applied to CustomerID: ' ||
  END LOOP;

  COMMIT; -- Save changes
END;
/
```

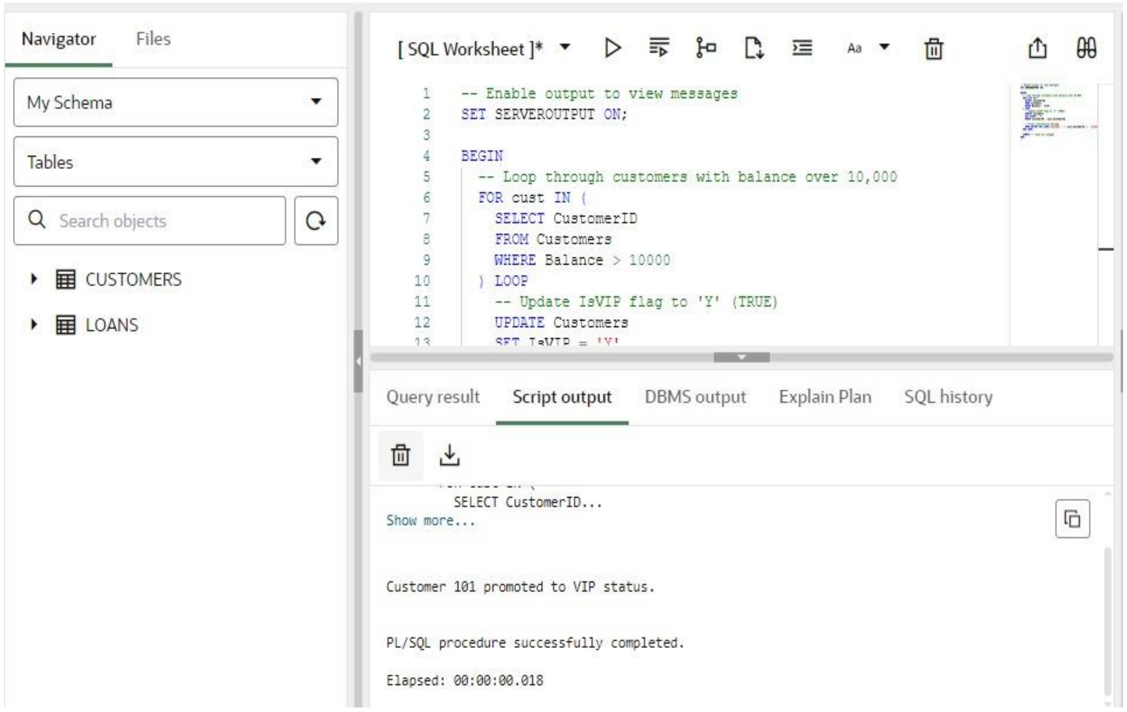
```
    SELECT CustomerID, LoanID...
Show more...

1% discount applied to CustomerID: 101

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.020
```

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.

CODE:

```
-- Enable output to view messages
SET SERVEROUTPUT ON;

BEGIN
  -- Loop through customers with balance over 10,000
  FOR cust IN (
    SELECT CustomerID
    FROM Customers
    WHERE Balance > 10000
  ) LOOP
    -- Update IsVIP flag to 'Y' (TRUE)
    UPDATE Customers
    SET IsVIP = 'Y'
    WHERE CustomerID = cust.CustomerID;

    -- Print confirmation message
    DBMS_OUTPUT.PUT_LINE('Customer ' || cust.CustomerID || ' promoted to VIP status.');
  END LOOP;
  COMMIT; -- Save all changes
END;
/
```



```
-- Enable output to view messages
SET SERVEROUTPUT ON;

BEGIN
  -- Loop through customers with balance over 10,000
  FOR cust IN (
    SELECT CustomerID
    FROM Customers
    WHERE Balance > 10000
  ) LOOP
    -- Update IsVIP flag to 'Y' (TRUE)
    UPDATE Customers
```

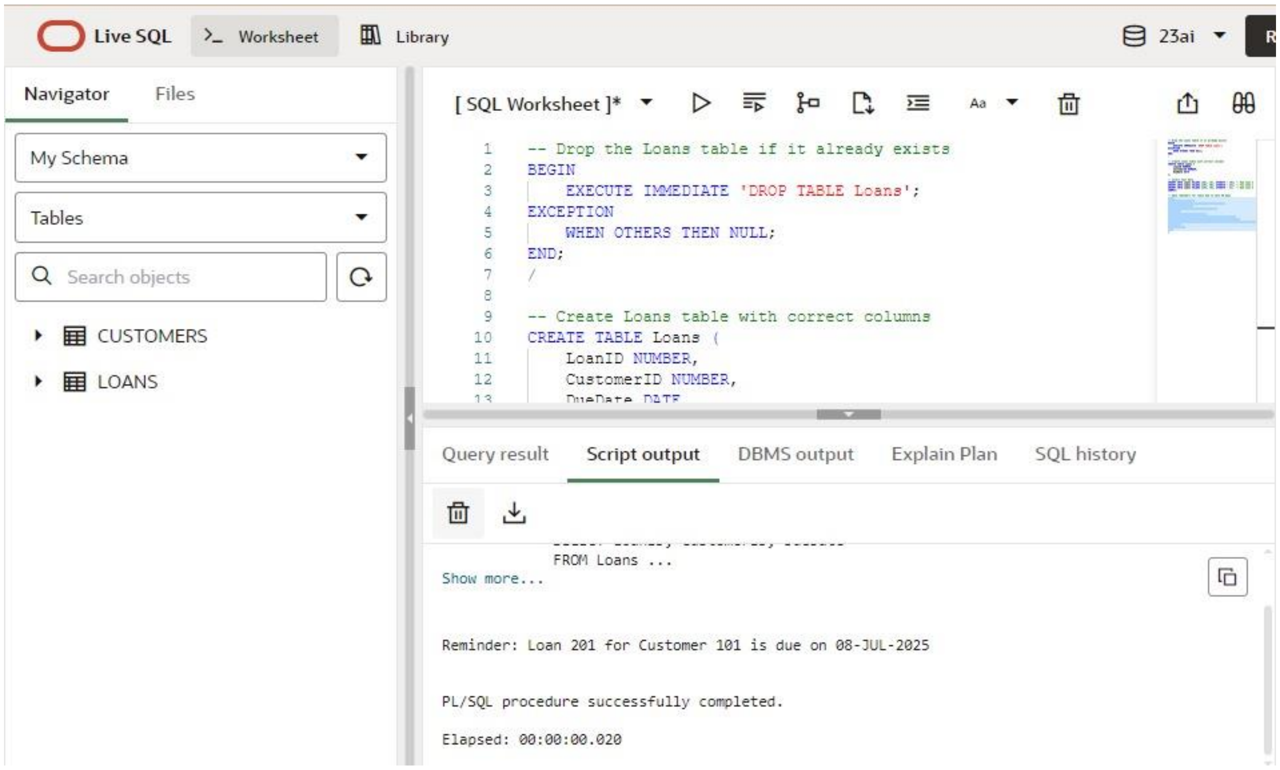
```
    SELECT CustomerID...
Show more...


Customer 101 promoted to VIP status.


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.018
```

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:

```
-- Drop the Loans table if it already exists
BEGIN
  EXECUTE IMMEDIATE 'DROP TABLE Loans';
EXCEPTION
  WHEN OTHERS THEN NULL;
END;
/

-- Create Loans table with correct columns
CREATE TABLE Loans (
  LoanID NUMBER,
  CustomerID NUMBER,
  DueDate DATE
);

-- Insert test data
INSERT INTO Loans VALUES (201, 101, SYSDATE + 10); -- Due soon
INSERT INTO Loans VALUES (202, 102, SYSDATE + 35); -- Too late
INSERT INTO Loans VALUES (203, 103, SYSDATE + 5);  -- Due soon
COMMIT;

-- Show reminders for loans due in next 30 days
BEGIN
  FOR due_rec IN (
    SELECT LoanID, CustomerID, DueDate
    FROM Loans
    WHERE DueDate <= SYSDATE + 30
  ) LOOP
    DBMS_OUTPUT.PUT_LINE(
      'Reminder: Loan ' || due_rec.LoanID ||
      ' for Customer ' || due_rec.CustomerID ||
      ' is due on ' || TO_CHAR(due_rec.DueDate, 'DD-MON-YYYY')
    );
  END LOOP;
END;
/
```



```
-- Drop the Loans table if it already exists
BEGIN
    EXECUTE IMMEDIATE 'DROP TABLE Loans';
EXCEPTION
    WHEN OTHERS THEN NULL;
END;
/

-- Create Loans table with correct columns
CREATE TABLE Loans (
    LoanID NUMBER,
    CustomerID NUMBER,
```

```
            FROM Loans ...
Show more...


Reminder: Loan 201 for Customer 101 is due on 08-JUL-2025


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.020
```

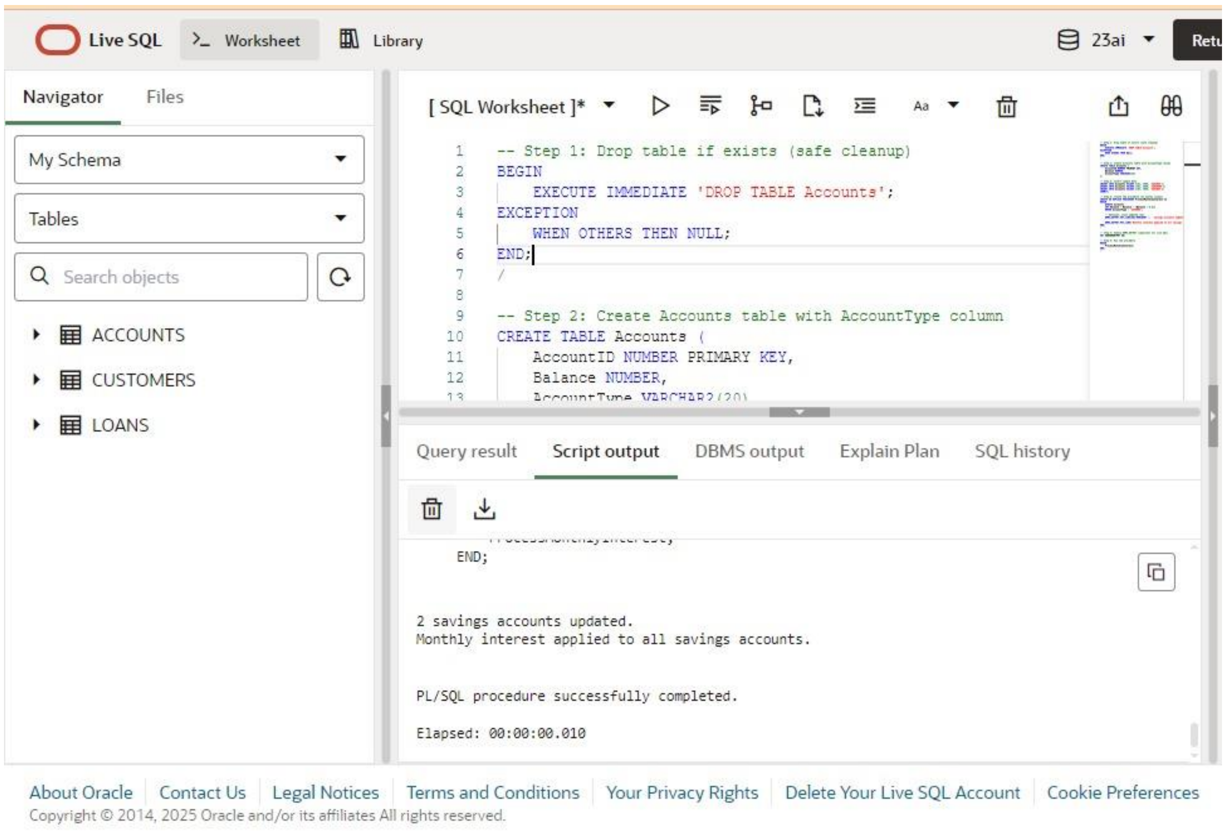
2. Stored Procedures

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.

CODE:

```
BEGIN
   EXECUTE IMMEDIATE 'DROP TABLE Accounts';
EXCEPTION
   WHEN OTHERS THEN NULL;
END;
/
CREATE TABLE Accounts (
   AccountID NUMBER PRIMARY KEY,
   Balance NUMBER,
   AccountType VARCHAR2(20)
);
INSERT INTO Accounts VALUES (101, 1000, 'SAVINGS');
INSERT INTO Accounts VALUES (102, 2000, 'CHECKING');
INSERT INTO Accounts VALUES (103, 3000, 'SAVINGS');
COMMIT;
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS
BEGIN
   UPDATE Accounts
   SET Balance = Balance + (Balance * 0.01)
   WHERE AccountType = 'SAVINGS';
   -- Optional: count updated rows
   DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT || ' savings accounts updated.');
   DBMS_OUTPUT.PUT_LINE('Monthly interest applied to all savings accounts.');
END;
/
SET SERVEROUTPUT ON;

BEGIN
   ProcessMonthlyInterest;
END;
/
```



```
-- Step 1: Drop table if exists (safe cleanup)
BEGIN
    EXECUTE IMMEDIATE 'DROP TABLE Accounts';
EXCEPTION
    WHEN OTHERS THEN NULL;
END;
/

-- Step 2: Create Accounts table with AccountType column
CREATE TABLE Accounts (
    AccountID NUMBER PRIMARY KEY,
    Balance NUMBER,
```

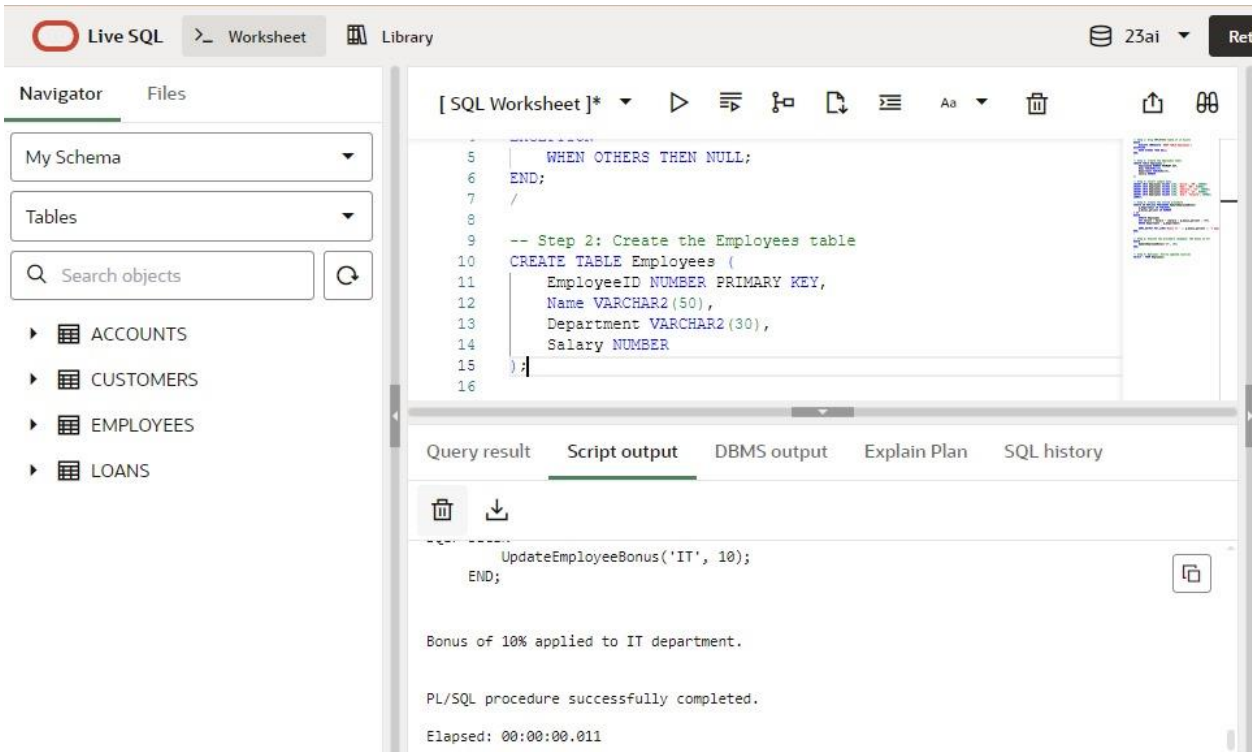
```
    END;


2 savings accounts updated.
Monthly interest applied to all savings accounts.


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.010
```

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.

CODE:

```
BEGIN
   EXECUTE IMMEDIATE 'DROP TABLE Employees';
EXCEPTION
   WHEN OTHERS THEN NULL;
END;
/

CREATE TABLE Employees (
   EmployeeID NUMBER PRIMARY KEY,
   Name VARCHAR2(50),
   Department VARCHAR2(30),
   Salary NUMBER
);

INSERT INTO Employees VALUES (101, 'Alice', 'HR', 40000);
INSERT INTO Employees VALUES (102, 'Bob', 'HR', 45000);
INSERT INTO Employees VALUES (103, 'Charlie', 'IT', 60000);
INSERT INTO Employees VALUES (104, 'David', 'IT', 65000);
INSERT INTO Employees VALUES (105, 'Eva', 'Finance', 50000);
COMMIT;
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(
   p_department IN VARCHAR2,
   p_bonus_percent IN NUMBER
) IS
BEGIN
   UPDATE Employees
   SET Salary = Salary + (Salary * p_bonus_percent / 100)
   WHERE Department = p_department;

   DBMS_OUTPUT.PUT_LINE('Bonus of ' || p_bonus_percent || '% applied to ' ||
p_department || ' department.');
END;
/
BEGIN
   UpdateEmployeeBonus('IT', 10);
END;
/
SELECT * FROM Employees;
```



```
    WHEN OTHERS THEN NULL;
END;
/

-- Step 2: Create the Employees table
CREATE TABLE Employees (
    EmployeeID NUMBER PRIMARY KEY,
    Name VARCHAR2(50),
    Department VARCHAR2(30),
    Salary NUMBER
);
```

```
        UpdateEmployeeBonus('IT', 10);
    END;


Bonus of 10% applied to IT department.


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.011
```

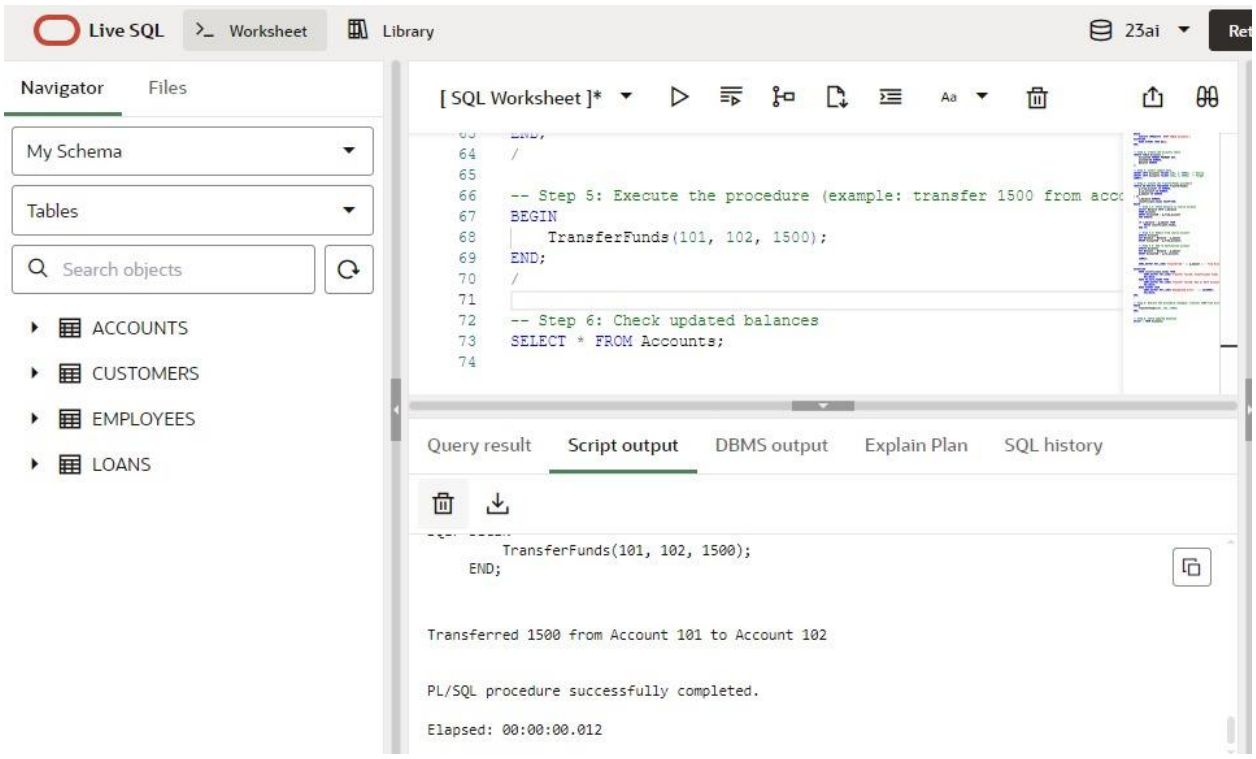
Scenario 3: Customers should be able to transfer funds between their accounts.

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:

```
BEGIN
   EXECUTE IMMEDIATE 'DROP TABLE Accounts';
EXCEPTION
   WHEN OTHERS THEN NULL;
END;
/
CREATE TABLE Accounts (
   AccountID NUMBER PRIMARY KEY,
   CustomerID NUMBER,
   Balance NUMBER
);
INSERT INTO Accounts VALUES (101, 1, 5000);  -- Source
INSERT INTO Accounts VALUES (102, 1, 2000);  -- Target
COMMIT;
CREATE OR REPLACE PROCEDURE TransferFunds(
   p_from_account IN NUMBER,
   p_to_account IN NUMBER,
   p_amount IN NUMBER
) IS
   v_balance NUMBER;
   insufficient_funds EXCEPTION;
BEGIN
   SELECT Balance INTO v_balance
   FROM Accounts
   WHERE AccountID = p_from_account
   FOR UPDATE;
   IF v_balance < p_amount THEN
      RAISE insufficient_funds;
   END IF;
   UPDATE Accounts
   SET Balance = Balance - p_amount
   WHERE AccountID = p_from_account;
   UPDATE Accounts
   SET Balance = Balance + p_amount
   WHERE AccountID = p_to_account;
   COMMIT;
   DBMS_OUTPUT.PUT_LINE('Transferred ' || p_amount || ' from Account ' || p_from_account || ' to
Account ' || p_to_account);
EXCEPTION
   WHEN insufficient_funds THEN
      DBMS_OUTPUT.PUT_LINE('Transfer failed: Insufficient funds in Account ' || p_from_account);
      ROLLBACK;
   WHEN NO_DATA_FOUND THEN
```

```
        DBMS_OUTPUT.PUT_LINE('Transfer failed: One or both accounts do not exist.');
        ROLLBACK;
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Unexpected error: ' || SQLERRM);
        ROLLBACK;
END;
/
BEGIN
    TransferFunds(101, 102, 1500);
END;
/
SELECT * FROM Accounts;
```



```
/

-- Step 5: Execute the procedure (example: transfer 1500 from acco
BEGIN
    TransferFunds(101, 102, 1500);
END;
/

-- Step 6: Check updated balances
SELECT * FROM Accounts;
```

```
        TransferFunds(101, 102, 1500);
    END;


Transferred 1500 from Account 101 to Account 102


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.012
```

WEEK 2 JUNIT BASIC TESTING EXERCISE

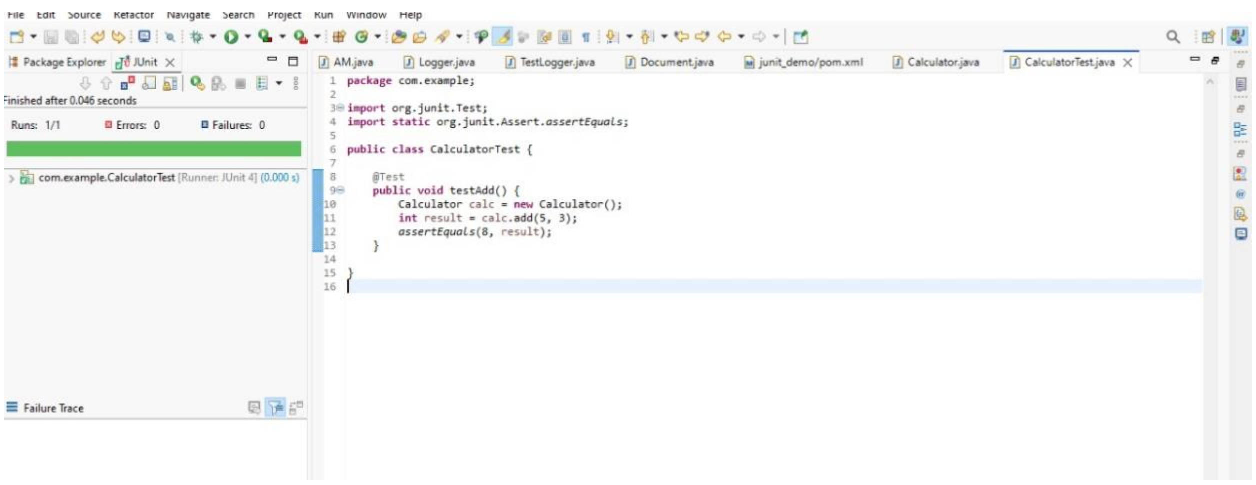
1. Setting Up JUnit Scenario: You need to set up JUnit in your Java project to start writing unit tests.

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:
3. Create a new test class in your project.

CODE:

```
package com.example;
public class Calculator {
   public int add(int a, int b) {
      return a + b;
   }
}
package com.example;
import org.junit.Test;
import static org.junit.Assert.assertEquals;
public class CalculatorTest {
   @Test
   public void testAdd() {
      Calculator calc = new Calculator();
      int result = calc.add(5, 3);
      assertEquals(8, result);
   }

}
```



```
Finished after 0.046 seconds
Runs: 1/1    Errors: 0    Failures: 0
com.example.CalculatorTest [Runner: JUnit 4] (0.000 s)
```

```java
package com.example;

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

public class CalculatorTest {

    @Test
    public void testAdd() {
        Calculator calc = new Calculator();
        int result = calc.add(5, 3);
        assertEquals(8, result);
    }

}
```

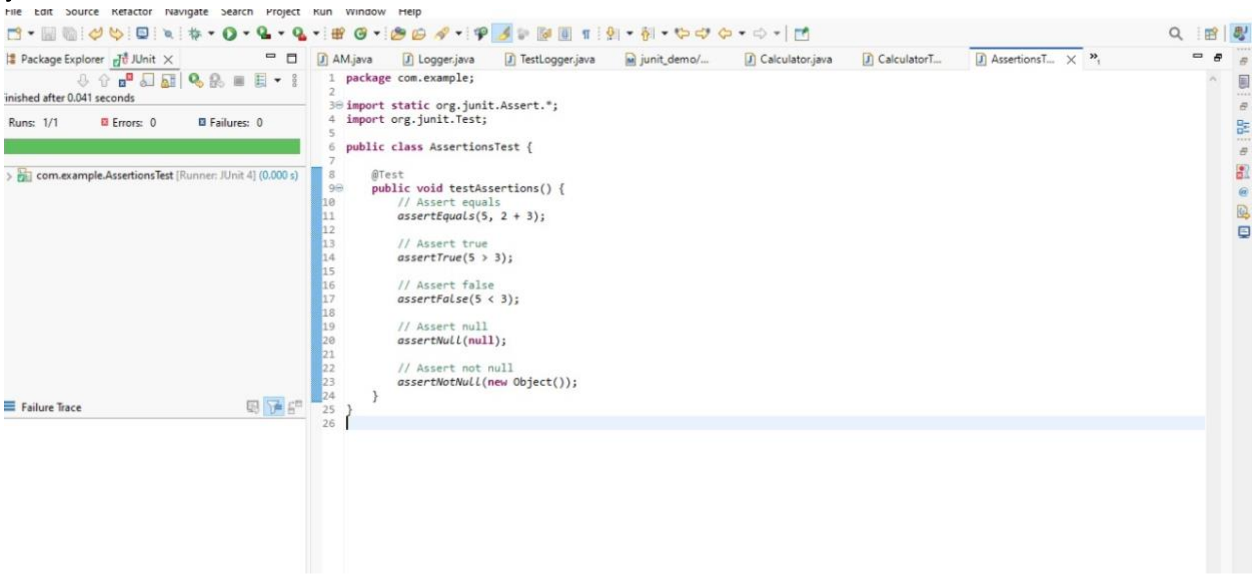
2. Assertions in JUnit Scenario: You need to use different assertions in JUnit to validate your test results.

Steps:

1. Write tests using various JUnit assertions.

   CODE:

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



```
inished after 0.041 seconds
Runs: 1/1    Errors: 0    Failures: 0
com.example.AssertionsTest [Runner: JUnit 4] (0.000 s)
```

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

3. Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in JUnit

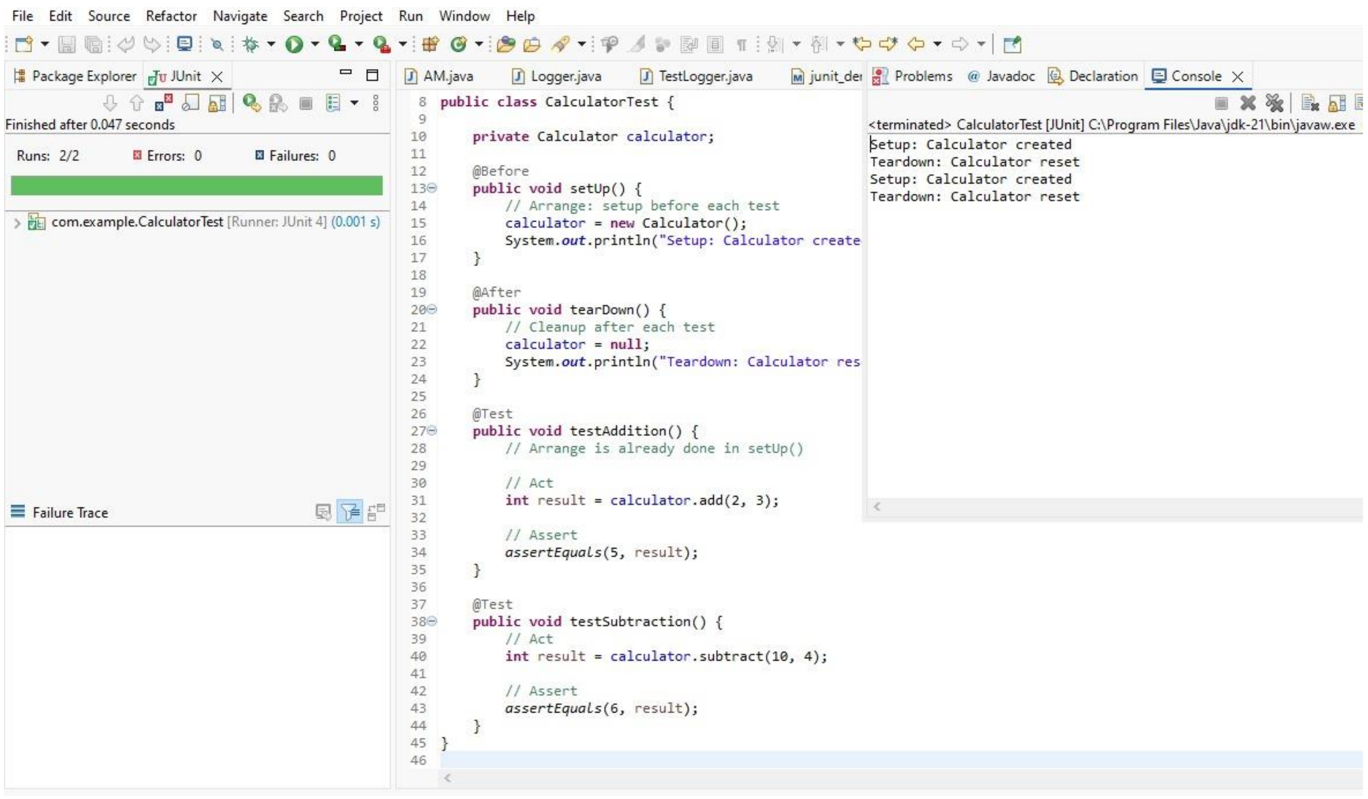
Scenario: You need to organize your tests using the Arrange-Act-Assert (AAA) pattern and use setup and teardown methods.

Steps:

1. Write tests using the AAA pattern.
2. Use @Before and @After annotations for setup and teardown methods.

CODE:

```java
package com.example;
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }
    public int subtract(int a, int b) {
        return a - b;
    }
}
package com.example;
import static org.junit.Assert.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
public class CalculatorTest {
   private Calculator calculator;
   @Before
   public void setUp() {
      // Arrange: setup before each test
      calculator = new Calculator();
      System.out.println("Setup: Calculator created");
   }
   @After
   public void tearDown() {
      // Cleanup after each test
      calculator = null;
      System.out.println("Teardown: Calculator reset");
   }
   @Test
   public void testAddition() {
      // Arrange is already done in setUp()
      // Act
      int result = calculator.add(2, 3);
      // Assert
      assertEquals(5, result);
   }
   @Test
   public void testSubtraction() {
      // Act
      int result = calculator.subtract(10, 4);

      // Assert
      assertEquals(6, result);
   }
}
```



Finished after 0.047 seconds

Runs: 2/2 Errors: 0 Failures: 0

com.example.CalculatorTest [Runner: JUnit 4] (0.001 s)

```java
public class CalculatorTest {

    private Calculator calculator;

    @Before
    public void setUp() {
        // Arrange: setup before each test
        calculator = new Calculator();
        System.out.println("Setup: Calculator create
    }

    @After
    public void tearDown() {
        // Cleanup after each test
        calculator = null;
        System.out.println("Teardown: Calculator res
    }

    @Test
    public void testAddition() {
        // Arrange is already done in setUp()

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

        // Assert
        assertEquals(5, result);
    }

    @Test
    public void testSubtraction() {
        // Act
        int result = calculator.subtract(10, 4);

        // Assert
        assertEquals(6, result);
    }
}
```

<terminated> CalculatorTest [JUnit] C:\Program Files\Java\jdk-21\bin\javaw.exe

```
Setup: Calculator created
Teardown: Calculator reset
Setup: Calculator created
Teardown: Calculator reset
```

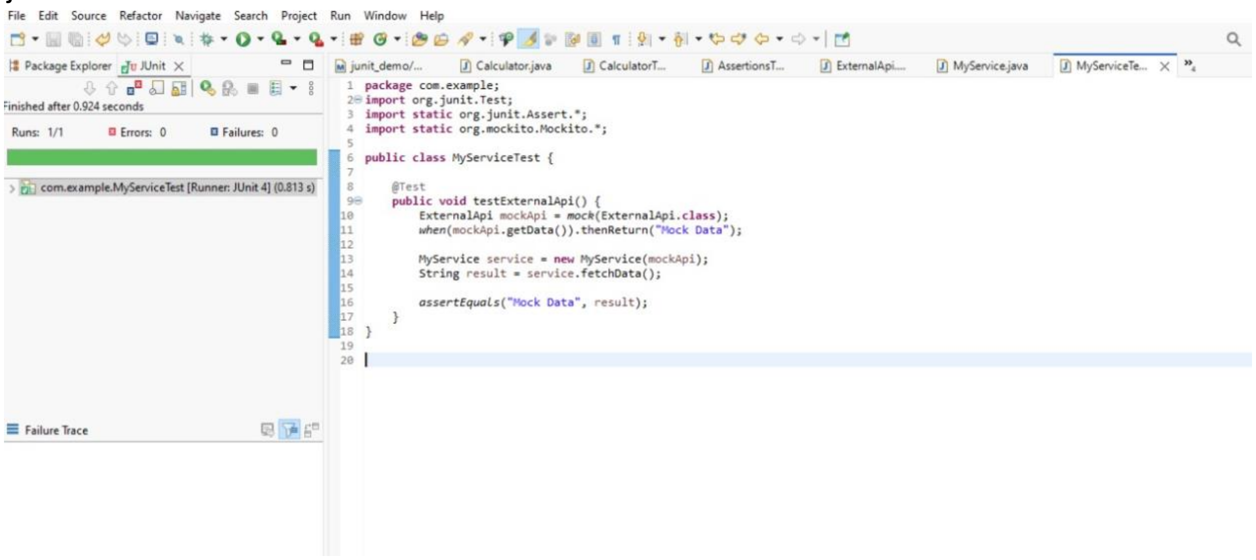
WEEK 2 MOCKITO EXERCISES

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.

CODE:

```
package com.example;
public interface ExternalApi {
   String getData();
}
package com.example;
public class MyService {
   private ExternalApi api;
   public MyService(ExternalApi api) {
      this.api = api;
   }
   public String fetchData() {
      return api.getData();
   }
}
package com.example;
import org.junit.Test;
import static org.junit.Assert.*;
import static org.mockito.Mockito.*;
public class MyServiceTest {
   @Test
   public void testExternalApi() {
      ExternalApi mockApi = mock(ExternalApi.class);
      when(mockApi.getData()).thenReturn("Mock Data");
      MyService service = new MyService(mockApi);
      String result = service.fetchData();
```

```
        assertEquals("Mock Data", result);
    }
}
```



Finished after 0.924 seconds

Runs: 1/1 Errors: 0 Failures: 0

com.example.MyServiceTest [Runner: JUnit 4] (0.813 s)

```
package com.example;
import org.junit.Test;
import static org.junit.Assert.*;
import static org.mockito.Mockito.*;

public class MyServiceTest {

    @Test
    public void testExternalApi() {
        ExternalApi mockApi = mock(ExternalApi.class);
        when(mockApi.getData()).thenReturn("Mock Data");

        MyService service = new MyService(mockApi);
        String result = service.fetchData();

        assertEquals("Mock Data", result);
    }
}
```

2.Verifying Interactions Scenario: You need to ensure that a method is called with specific arguments. Steps:

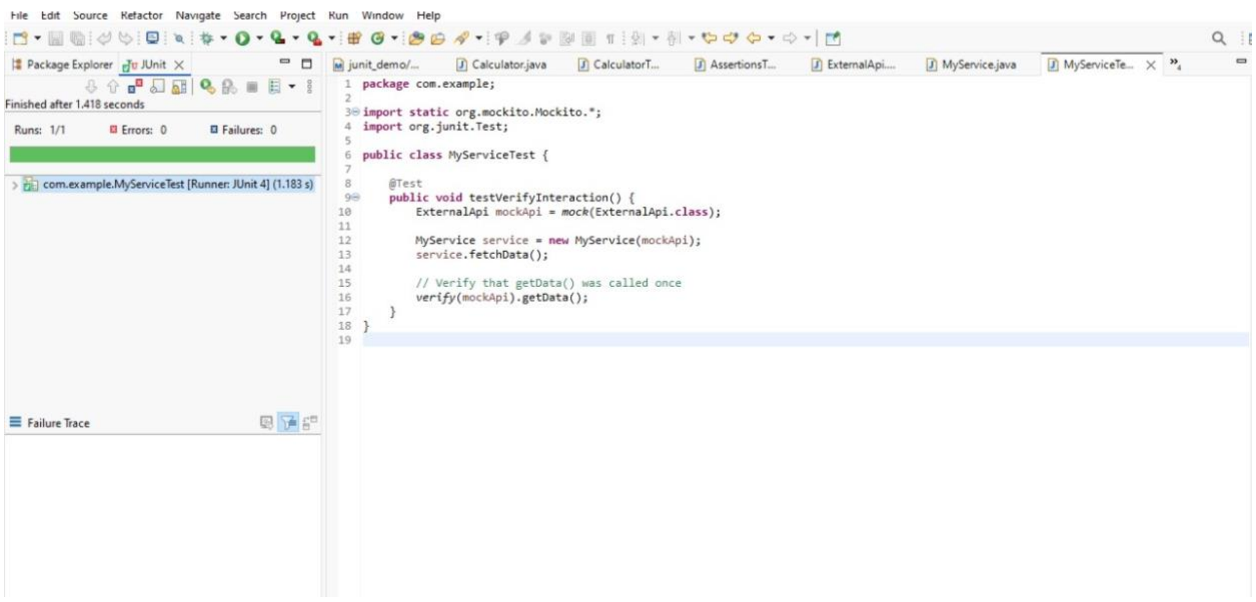
1. Create a mock object.
2. Call the method with specific arguments.
3. Verify the interaction.

package com.example;

CODE:

```
import static org.mockito.Mockito.*;
import org.junit.Test;
public class MyServiceTest {
    @Test
    public void testVerifyInteraction() {
        ExternalApi mockApi = mock(ExternalApi.class);
        MyService service = new MyService(mockApi);
        service.fetchData();
        // Verify that getData() was called once
        verify(mockApi).getData();
    }
}
package com.example;
```

```java
public interface ExternalApi {
    String getData();
}
package com.example;
public class MyService {
    private ExternalApi api;
    public MyService(ExternalApi api) {
        this.api = api;
    }
    public String fetchData() {
        return api.getData();
    }
}
```



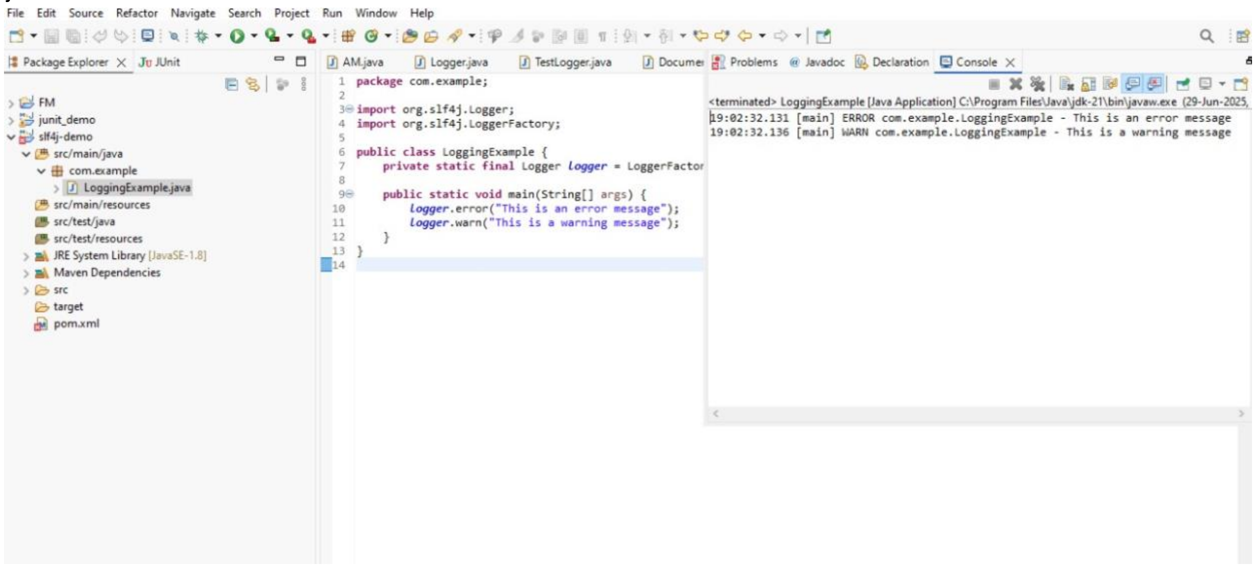
```
Finished after 1.418 seconds
Runs: 1/1    Errors: 0    Failures: 0
com.example.MyServiceTest [Runner: JUnit 4] (1.183 s)
```

```java
package com.example;

import static org.mockito.Mockito.*;
import org.junit.Test;

public class MyServiceTest {

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

        MyService service = new MyService(mockApi);
        service.fetchData();

        // Verify that getData() was called once
        verify(mockApi).getData();
    }
}
```

WEEK 2 SL4J LOGGING EXERCISES

1. Logging Error Messages and Warning Levels Task: Write a Java application that demonstrates logging error messages and warning levels using SLF4J.
   CODE:

```
package com.example;
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");
  }
}
```



```
package com.example;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class LoggingExample {
    private static final Logger logger = LoggerFactor

    public static void main(String[] args) {
        logger.error("This is an error message");
        logger.warn("This is a warning message");
    }
}
```

```
<terminated> LoggingExample [Java Application] C:\Program Files\Java\jdk-21\bin\javaw.exe (29-Jun-2025,
19:02:32.131 [main] ERROR com.example.LoggingExample - This is an error message
19:02:32.136 [main] WARN com.example.LoggingExample - This is a warning message
```