

WEEK 2:

PL/SQL_Exercises

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.

SOLUTION:

```
CREATE TABLE Customers (
    CustomerID NUMBER,
    Name VARCHAR2(50),
    Age NUMBER,
    Balance NUMBER,
    IsVIP VARCHAR2(5)
);
```

```
CREATE TABLE Loans (
    LoanID NUMBER,
    CustomerID NUMBER,
    InterestRate NUMBER,
    DueDate DATE
);
```

```
INSERT INTO Customers VALUES (1, 'Alice', 65, 12000, 'FALSE');
INSERT INTO Customers VALUES (2, 'Bob', 58, 9000, 'FALSE');
INSERT INTO Customers VALUES (3, 'Charlie', 70, 15000, 'FALSE');
```

```
INSERT INTO Loans VALUES (101, 1, 8.5, SYSDATE + 10);
INSERT INTO Loans VALUES (102, 2, 9.0, SYSDATE + 40);
INSERT INTO Loans VALUES (103, 3, 7.5, SYSDATE + 5);
```

```
COMMIT;
```

```
BEGIN
```

```
FOR customer_rec IN (
    SELECT CustomerID
    FROM Customers
    WHERE Age > 60
) LOOP
```

```
    UPDATE Loans
    SET InterestRate = InterestRate - 1
    WHERE CustomerID = customer_rec.CustomerID;
```

```
    DBMS_OUTPUT.PUT_LINE('Discount applied for Customer ID: ' || customer_rec.CustomerID);
```

```
END LOOP;
```

```
COMMIT;
```

```
END;
```

```
OUTPUT:
```

```

47   SELECT CustomerID
48   FROM Customers
49   WHERE Balance > 10000
50   ) LOOP
51   UPDATE Customers
52   SET IsVIP = 'TRUE'
53   WHERE CustomerID = vip_rec.CustomerID;
54
55   DBMS_OUTPUT.PUT_LINE('Customer ' || vip_rec.CustomerID || ' is now a VIP.');
56 END LOOP;
57 COMMIT;
58 /
59

```

Query result Script output DBMS output Explain Plan SQL history

Customer 1 is now a VIP.
Customer 3 is now a VIP.

PL/SQL procedure successfully completed.
Elapsed: 00:00:00.097

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.

```

BEGIN
  FOR vip_rec IN (
    SELECT CustomerID
    FROM Customers
    WHERE Balance > 10000
  ) LOOP
    UPDATE Customers
    SET IsVIP = 'TRUE'
    WHERE CustomerID = vip_rec.CustomerID;

    DBMS_OUTPUT.PUT_LINE('Customer ' || vip_rec.CustomerID || ' is now a VIP.');

  END LOOP;
  COMMIT;
END;

```

OUTPUT:

```

29   FROM Customers
30   WHERE Age > 60
31
32   LOOP
33
34     UPDATE Loans
35     SET InterestRate = InterestRate - 1
36     WHERE CustomerID = customer_rec.CustomerID;
37
38     DBMS_OUTPUT.PUT_LINE('Discount applied for Customer ID: ' || customer_rec.CustomerID);
39
40   END LOOP;
41
42   COMMIT;
43

```

Query result Script output DBMS output Explain Plan SQL history

Show more... ▾

Discount applied for Customer ID: 1
Discount applied for Customer ID: 3

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.000

- **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.

BEGIN

```

FOR loan_rec IN (
  SELECT LoanID, CustomerID, DueDate
  FROM Loans
  WHERE DueDate <= SYSDATE + 30
) LOOP
  DBMS_OUTPUT.PUT_LINE(
    'Reminder: Loan ID ' || loan_rec.LoanID ||
    ' for Customer ID ' || loan_rec.CustomerID ||
    ' is due on ' || TO_CHAR(loan_rec.DueDate, 'DD-MON-YYYY')
  );
END LOOP;
END;

```

```

 62 FOR loan_rec IN (
 63   SELECT LoanID, CustomerID, DueDate
 64   FROM Loans
 65   WHERE DueDate <= SYSDATE + 30
 66 ) LOOP
 67   DBMS_OUTPUT.PUT_LINE(
 68     'Reminder: Loan ID ' || loan_rec.LoanID ||
 69     ' for Customer ID ' || loan_rec.CustomerID ||
 70     ' is due on ' || TO_CHAR(loan_rec.DueDate, 'DD-MON-YYYY')
 71   );
 72 END LOOP;
 73 /
 74
 75
 76

```

Query result Script output DBMS output Explain Plan SQL history

Show more... Copy

```

Reminder: Loan ID 103 for Customer ID 3 is due on 30-JUN-2025
Reminder: Loan ID 101 for Customer ID 1 is due on 05-JUL-2025

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.013

```

About Oracle | Contact Us | Legal Notices | Terms and Conditions | Your Privacy Rights | Delete Your Live SQL Account | Cookie Preferences
Copyright © 2014, 2025 Oracle and/or its affiliates. All rights reserved.

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.

```

CREATE TABLE Accounts (
  AccountID NUMBER PRIMARY KEY,
  AccountHolder VARCHAR2(100),
  AccountType VARCHAR2(20), -- e.g., 'Savings', 'Current'
  Balance NUMBER(15, 2)
);

```

```

CREATE TABLE Employees (
  EmployeeID NUMBER PRIMARY KEY,
  Name VARCHAR2(100),
  DepartmentID NUMBER,
  Salary NUMBER(10, 2)
);

```

```
INSERT INTO Accounts VALUES (101, 'A', 'Savings', 10000);
INSERT INTO Accounts VALUES (102, 'B', 'Current', 8000);
INSERT INTO Accounts VALUES (103, 'C', 'Savings', 15000);
```

```
INSERT INTO Employees VALUES (1, 'June', 10, 50000);
INSERT INTO Employees VALUES (2, 'July', 20, 60000);
INSERT INTO Employees VALUES (3, 'Jan', 10, 55000);
```

```
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS
```

```
BEGIN
```

```
UPDATE Accounts
```

```
SET Balance = Balance + (Balance * 0.01)
```

```
WHERE AccountType = 'Savings';
```

```
COMMIT;
```

```
END;
```

```
OUTPUT:
```

The screenshot shows the Oracle Live SQL interface. The top navigation bar includes 'Live SQL', 'Worksheet', 'Library', '23ai', 'Return to Live SQL Classic', 'Help and Feedback', and 'Sign Out'. The left sidebar has 'Navigator' selected, showing 'My Schema' (Tables: ACCOUNTS, CUSTOMERS, EMPLOYEES, LOANS) and a search bar. The main area is a 'SQL Worksheet' containing the following code:

```
36    BONUS_PCT IN NUMBER
37  ) IS
38  BEGIN
39    UPDATE Employees
40    SET Salary = Salary + (Salary * bonus_pct)
41    WHERE DepartmentID = dept_id;
42
43    COMMIT;
44
45
46
```

The 'Script output' tab is selected, showing the compiled procedure:

```
BEGIN
  UPDATE Accounts
  SET Balance = Balance + (Balance * 0.01)...
```

Below the script output, it says 'Procedure PROCESSMONTHLYINTEREST compiled' and 'Elapsed: 00:00:00.019'.

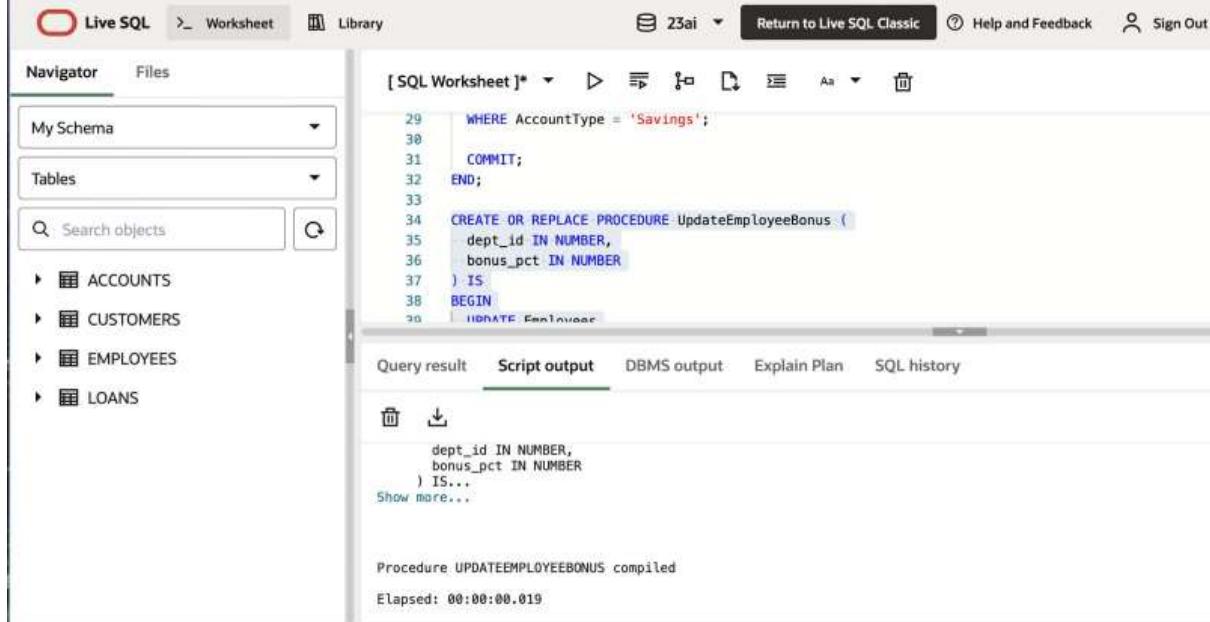
Scenario 2: The bank wants to implement a bonus scheme for employees based on their performance.

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.

```
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (
    dept_id IN NUMBER,
    bonus_pct IN NUMBER
) IS
BEGIN
    UPDATE Employees
    SET Salary = Salary + (Salary * bonus_pct)
    WHERE DepartmentID = dept_id;

    COMMIT;
END;
```

OUTPUT:



The screenshot shows the Oracle Live SQL interface. The top navigation bar includes 'Live SQL', 'Worksheet', 'Library', '23ai', 'Return to Live SQL Classic', 'Help and Feedback', and 'Sign Out'. The left sidebar has a 'Navigator' tab selected, showing 'My Schema' (Tables: ACCOUNTS, CUSTOMERS, EMPLOYEES, LOANS), a search bar, and a 'Script output' section. The main workspace is titled '[SQL Worksheet]*' and contains the following code:

```
29 WHERE AccountType = 'Savings';
30
31 COMMIT;
32 END;
33
34 CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (
35     dept_id IN NUMBER,
36     bonus_pct IN NUMBER
37 ) IS
38 BEGIN
39     UPDATE Employees
40     SET Salary = Salary + (Salary * bonus_pct)
41     WHERE DepartmentID = dept_id;

42     COMMIT;
43 END;
44
```

The 'Script output' tab is active, showing the results of the execution:

```
Procedure UPDATEEMPLOYEEBONUS compiled
Elapsed: 00:00:00.019
```

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.

```
CREATE OR REPLACE PROCEDURE TransferFunds (
    from_acc_id IN NUMBER,
    to_acc_id IN NUMBER,
    amount IN NUMBER
) IS
    insufficient_balance EXCEPTION;
BEGIN
    -- Check balance
    DECLARE
        current_balance NUMBER;
    BEGIN
        SELECT Balance INTO current_balance
        FROM Accounts
        WHERE AccountID = from_acc_id;

        IF current_balance < amount THEN
            RAISE insufficient_balance;
        END IF;
    END;

    -- Perform transfer
    UPDATE Accounts
    SET Balance = Balance - amount
    WHERE AccountID = from_acc_id;

    UPDATE Accounts
    SET Balance = Balance + amount
```

```

WHERE AccountID = to_acc_id;

COMMIT;

EXCEPTION
WHEN insufficient_balance THEN
ROLLBACK;

DBMS_OUTPUT.PUT_LINE('✖ Error: Insufficient balance.');

WHEN OTHERS THEN
ROLLBACK;

DBMS_OUTPUT.PUT_LINE('✖ Unexpected error: ' || SQLERRM);

END;

```

OUTPUT:

The screenshot shows the Oracle SQL Developer interface. The Navigator pane on the left lists the schema objects: My Schema, Tables, and a search bar. Below these are the tables: ACCOUNTS, CUSTOMERS, EMPLOYEES, and LOANS. The SQL Worksheet pane on the right contains the following PL/SQL code:

```

77  EXCEPTION
78    WHEN insufficient_balance THEN
79      ROLLBACK;
80      DBMS_OUTPUT.PUT_LINE('✖ Error: Insufficient balance.');
81    WHEN OTHERS THEN
82      ROLLBACK;
83      DBMS_OUTPUT.PUT_LINE('✖ Unexpected error: ' || SQLERRM);
84  END;
85
86

```

Below the code, the Script output tab is selected, showing the results of the compilation:

```

Procedure UPDATEEMPLOYEEBONUS compiled
Elapsed: 00:00:00.019

SQL> CREATE OR REPLACE PROCEDURE TransferFunds (
  from_acc_id IN NUMBER,
  to_acc_id IN NUMBER,
  amount IN NUMBER...
Show more...

Procedure TRANSFERFUND$ compiled
Elapsed: 00:00:00.020

```

JUnit Testing Exercises :

Exercise 1: Setting Up JUnit

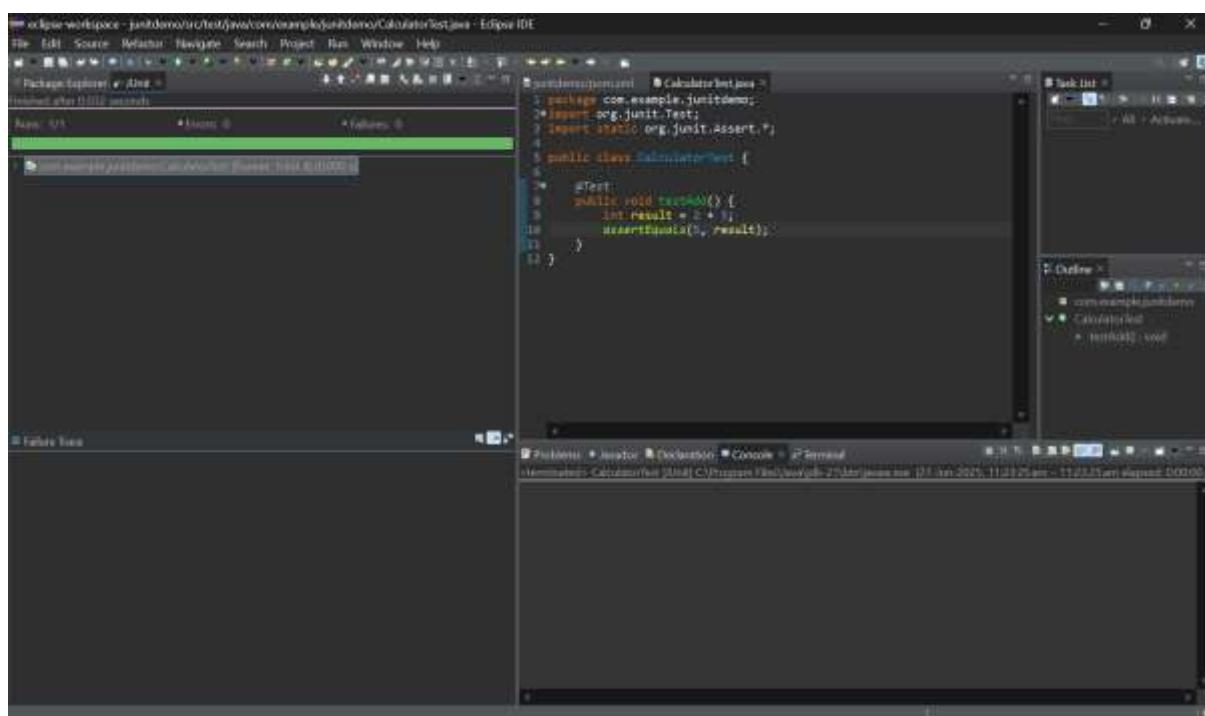
Scenario:

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.



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);

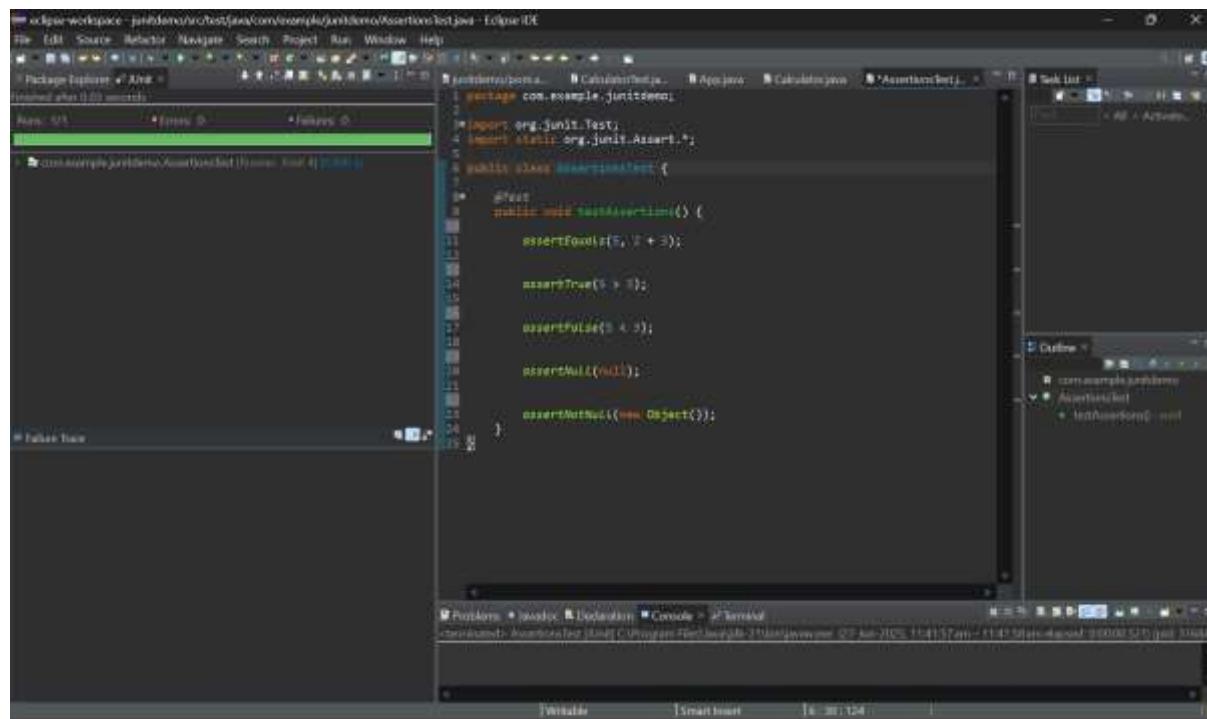
// Assert null assertNull(null);

// Assert not null

assertNotNull(new Object()); }

}

```



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.

eclipse-workspace - jndidemo [src/test/java/com/example/junitdemo/CalculatorTestJava - Eclipse IDE

File Edit Source Refactor Navigate Search Project Key Window Help

Package Explorer **Java** **Tasks** **Tempo** **Favorites**

Java (4) **Empty** **Favorites**

com.example.junitdemo.CalculatorTest (Running, 444 ms)

```
package com.example.junitdemo;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.*;

public class CalculatorTest {
    private Calculator calc;
    @Before
    public void setup() {
        System.out.println("Setting up calculator...");
        calc = new Calculator();
    }
    @After
    public void teardown() {
        System.out.println("Cleaning up calculator...");
    }
}
```

Problems **JavaDoc** **Declarator** **Console** **Terminal**

CalculatorTest (Initial Configuration Was Last Modified 21 hours ago on 07-Jun-2021, 11:46:11 AM - 2346 ms) [Run] [Stop]

Setting up calculator...
Cleaning up calculator...
Setting up calculator...
Cleaning up calculator...
Cleaning up calculator...

Failure Trace

Wizards | Smart Issues | Line 14 / 347

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;

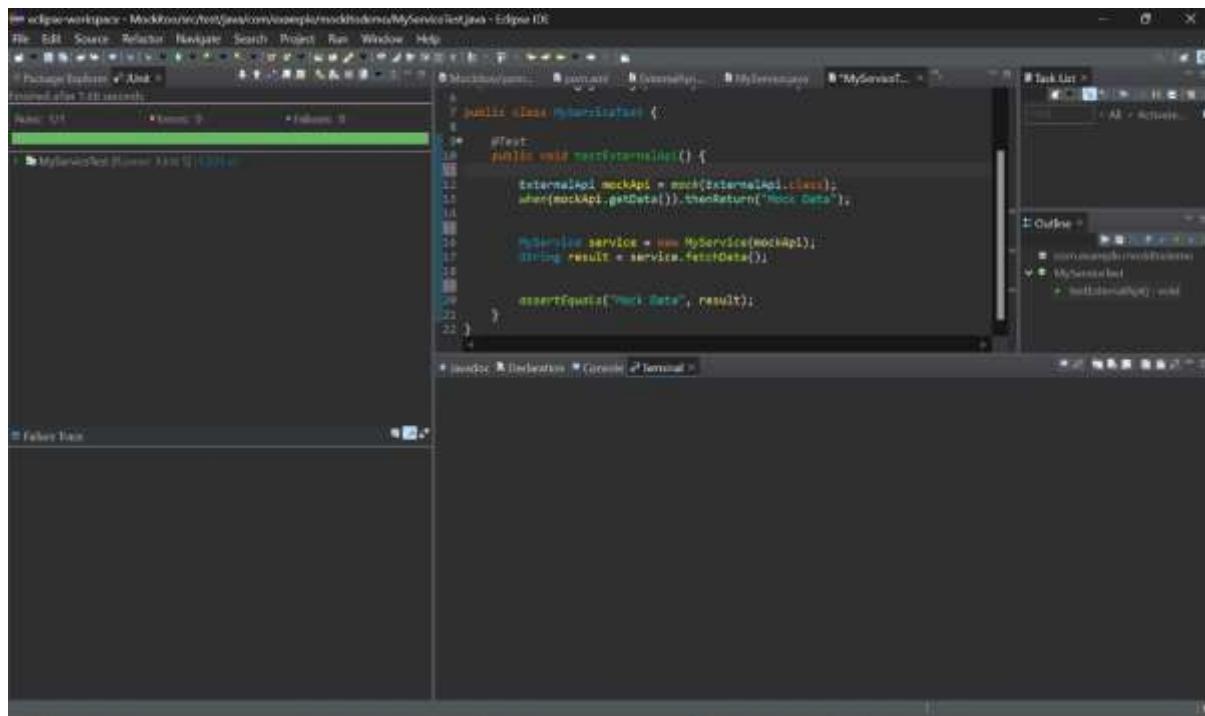
public class MyServiceTest { @Test

    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);
    }
}
```

OUTPUT:



The screenshot shows the Eclipse IDE interface with the code for `MyServiceTest.java` in the center editor window. The code is identical to the one provided above, demonstrating the use of Mockito to mock an `ExternalApi` and assert the return value of its `getData()` method. The Eclipse interface includes toolbars, menus, and various windows like Package Explorer, Task List, and Outline.

Exercise 2: Verifying Interactions

Scenario:

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

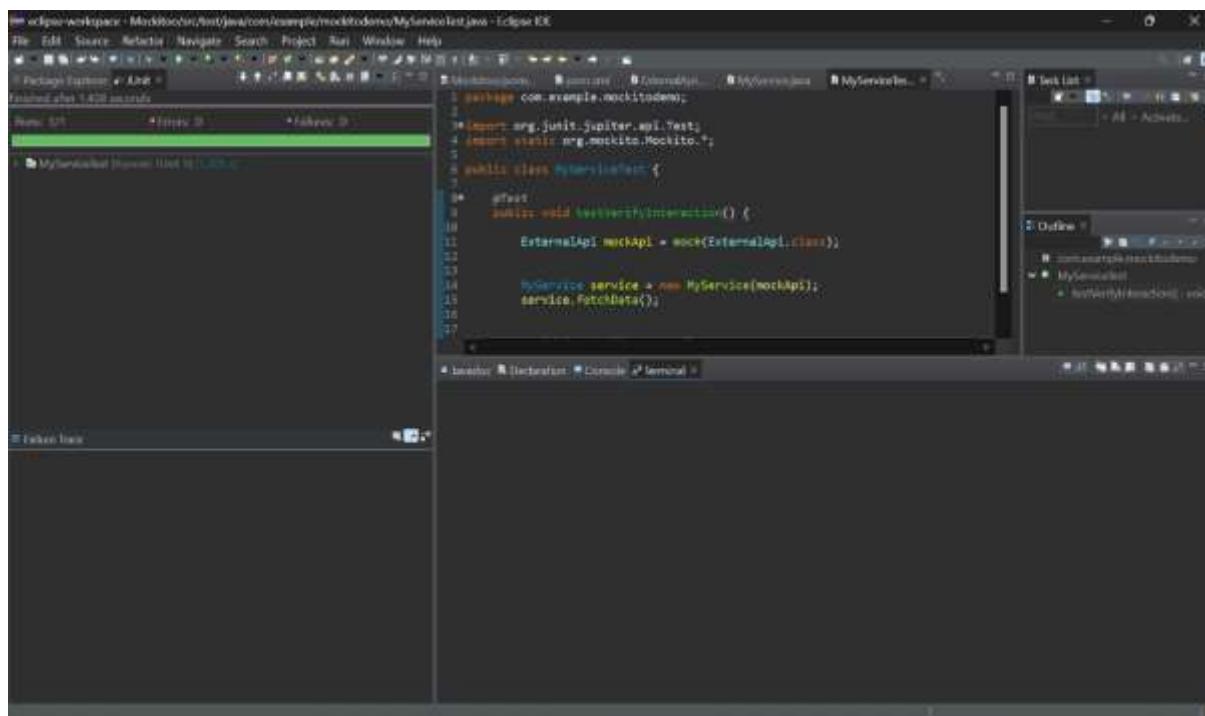
Steps:

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();  
    }  
}
```

OUTPUT:



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"); }

}
```

