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.
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.

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

CREATE TABLE Customers (
  CustomerID NUMBER,
  Name VARCHAR2(100),
  Age NUMBER,
  LoanInterestRate NUMBER
);
/

BEGIN
  INSERT INTO Customers VALUES (1, 'Alice', 65, 8.5);
  INSERT INTO Customers VALUES (2, 'Bob', 45, 7.5);
  INSERT INTO Customers VALUES (3, 'Charlie', 70, 9.0);
  COMMIT;
END;
/

COLUMN CustomerID FORMAT 9999
COLUMN Name FORMAT A15
COLUMN Age FORMAT 99
COLUMN LoanInterestRate FORMAT 999.99

SELECT * FROM Customers;
/

BEGIN
  FOR cust IN (SELECT * FROM Customers) LOOP
    IF cust.Age > 60 THEN
      UPDATE Customers
      SET LoanInterestRate = LoanInterestRate - (LoanInterestRate
* 0.01)
      WHERE CustomerID = cust.CustomerID;
    END IF;
  END LOOP;
END;
/
```

```
COLUMN CustomerID FORMAT 9999
COLUMN Name FORMAT A15
COLUMN Age FORMAT 99
COLUMN LoanInterestRate FORMAT 999.99

-- Then select
SELECT * FROM Customers;
/
```

Output:

```
CUSTOMERID NAME            AGE LOANINTERESTRATE
---------- --------------- --- ----------------
         1 Alice            65             8.50
         2 Bob              45             7.50
         3 Charlie          70             9.00

CUSTOMERID NAME            AGE LOANINTERESTRATE
---------- --------------- --- ----------------
         1 Alice            65             8.42
         2 Bob              45             7.50
         3 Charlie          70             8.91
```

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
  EXECUTE IMMEDIATE 'DROP TABLE Customers';
EXCEPTION WHEN OTHERS THEN NULL;
END;
/
CREATE TABLE Customers (
  CustomerID NUMBER,
  Name VARCHAR2(100),
  Age NUMBER,
  Balance NUMBER,
  IsVIP CHAR(1)
);
/
```

```
BEGIN
  INSERT INTO Customers VALUES (1, 'Alice', 65, 9500, NULL);
  INSERT INTO Customers VALUES (2, 'Bob', 45, 12000, NULL);
  INSERT INTO Customers VALUES (3, 'Charlie', 70, 10200, NULL);
  INSERT INTO Customers VALUES (4, 'David', 38, 7000, NULL);
  COMMIT;
END;
/
COLUMN CustomerID FORMAT 9999
COLUMN Name FORMAT A15
COLUMN Age FORMAT 99
COLUMN Balance FORMAT 999999.99
COLUMN IsVIP FORMAT A5
-- Show table before update
SELECT * FROM Customers;
/
BEGIN
  FOR cust IN (SELECT * FROM Customers) LOOP
    IF cust.Balance > 10000 THEN
      UPDATE Customers
      SET IsVIP = 'Y'
      WHERE CustomerID = cust.CustomerID;
    ELSE
      UPDATE Customers
      SET IsVIP = 'N'
      WHERE CustomerID = cust.CustomerID;
    END IF;
  END LOOP;
END;
/
SELECT * FROM Customers;
/
```

Output:

```
CUSTOMERID NAME                AGE     BALANCE ISVIP
---------- ---------------- --- ---------- -----
         1 Alice               65     9500.00
         2 Bob                 45    12000.00
         3 Charlie             70    10200.00
         4 David               38     7000.00

CUSTOMERID NAME                AGE     BALANCE ISVIP
---------- ---------------- --- ---------- -----
         1 Alice               65     9500.00 N
         2 Bob                 45    12000.00 Y
         3 Charlie             70    10200.00 Y
         4 David               38     7000.00 N
```

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
  EXECUTE IMMEDIATE 'DROP TABLE Loans';
EXCEPTION WHEN OTHERS THEN NULL;
END;
/
CREATE TABLE Loans (
  LoanID NUMBER,
  CustomerName VARCHAR2(100),
  DueDate DATE
);
/
BEGIN
  INSERT INTO Loans VALUES (1, 'Alice', SYSDATE + 10);     --
within 30 days
  INSERT INTO Loans VALUES (2, 'Bob', SYSDATE + 35);       --
outside 30 days
  INSERT INTO Loans VALUES (3, 'Charlie', SYSDATE + 5);    --
within 30 days
  INSERT INTO Loans VALUES (4, 'David', SYSDATE - 2);      --
already overdue
  COMMIT;
END;
/
COLUMN LoanID FORMAT 9999
```

```
COLUMN CustomerName FORMAT A15
COLUMN DueDate FORMAT A20
SELECT * FROM Loans;
/
SELECT
  LoanID,
  CustomerName,
  TO_CHAR(DueDate, 'YYYY-MM-DD') AS DueDate
FROM Loans
WHERE DueDate BETWEEN SYSDATE AND SYSDATE + 30;
/
```

Output:

```
 LOANID CUSTOMERNAME     DUEDATE
 ------ ---------------- --------------------
      1 Alice            07-JUL-25
      2 Bob              01-AUG-25
      3 Charlie          02-JUL-25
      4 David            25-JUN-25

 LOANID CUSTOMERNAME     DUEDATE
 ------ ---------------- --------------------
      1 Alice            2025-07-07
      3 Charlie          2025-07-02
```

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.

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

CREATE TABLE SavingsAccounts (
  AccountID NUMBER,
  CustomerName VARCHAR2(100),
  Balance NUMBER
);
/

BEGIN
  INSERT INTO SavingsAccounts VALUES (1, 'Alice', 10000);
  INSERT INTO SavingsAccounts VALUES (2, 'Bob', 8500);
```

```
  INSERT INTO SavingsAccounts VALUES (3, 'Charlie', 15000);
  COMMIT;
END;
/

COLUMN AccountID FORMAT 9999
COLUMN CustomerName FORMAT A15
COLUMN Balance FORMAT 999999.99

SELECT * FROM SavingsAccounts;
/

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS
BEGIN
  UPDATE SavingsAccounts
  SET Balance = Balance + (Balance * 0.01);
END;
/

BEGIN
  ProcessMonthlyInterest;
END;
/

SELECT * FROM SavingsAccounts;
/
```

Output:

```
 ACCOUNTID CUSTOMERNAME         BALANCE
 --------- ---------------- -----------
         1 Alice                10000.00
         2 Bob                   8500.00
         3 Charlie              15000.00

 ACCOUNTID CUSTOMERNAME         BALANCE
 --------- ---------------- -----------
         1 Alice                10100.00
         2 Bob                   8585.00
         3 Charlie              15150.00
```

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.

```
BEGIN
  EXECUTE IMMEDIATE 'DROP TABLE Employees';
EXCEPTION WHEN OTHERS THEN NULL;
END;
/
CREATE TABLE Employees (
  EmpID NUMBER,
  Name VARCHAR2(100),
  Department VARCHAR2(50),
  Salary NUMBER
);
/
BEGIN
  INSERT INTO Employees VALUES (1, 'Alice', 'HR', 50000);
  INSERT INTO Employees VALUES (2, 'Bob', 'IT', 60000);
  INSERT INTO Employees VALUES (3, 'Charlie', 'IT', 70000);
  INSERT INTO Employees VALUES (4, 'David', 'Finance', 55000);
  COMMIT;
END;
/
COLUMN EmpID FORMAT 9999
COLUMN Name FORMAT A15
COLUMN Department FORMAT A10
COLUMN Salary FORMAT 999999.99
SELECT * FROM Employees;
/
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (
  deptName IN VARCHAR2,
  bonusPercent IN NUMBER
) IS
BEGIN
  UPDATE Employees
  SET Salary = Salary + (Salary * bonusPercent / 100)
  WHERE Department = deptName;
END;
/
BEGIN
  UpdateEmployeeBonus('IT', 10);
END;
/
SELECT * FROM Employees;
/
```

Output:

EMPID	NAME	DEPARTMENT	SALARY
1	Alice	HR	50000.00
2	Bob	IT	60000.00
3	Charlie	IT	70000.00
4	David	Finance	55000.00

EMPID	NAME	DEPARTMENT	SALARY
1	Alice	HR	50000.00
2	Bob	IT	66000.00
3	Charlie	IT	77000.00
4	David	Finance	55000.00

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.

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

CREATE TABLE Accounts (
  AccountID NUMBER,
  Name VARCHAR2(100),
  Balance NUMBER
);
/

BEGIN
  INSERT INTO Accounts VALUES (1, 'Alice', 10000);
  INSERT INTO Accounts VALUES (2, 'Bob', 5000);
  INSERT INTO Accounts VALUES (3, 'Charlie', 2000);
  COMMIT;
END;
/

COLUMN AccountID FORMAT 9999
COLUMN Name FORMAT A15
COLUMN Balance FORMAT 999999.99

SELECT * FROM Accounts;
```

```
/

CREATE OR REPLACE PROCEDURE TransferFunds (
  fromAccount IN NUMBER,
  toAccount IN NUMBER,
  amount IN NUMBER
) IS
  fromBalance NUMBER;
BEGIN
  SELECT Balance INTO fromBalance FROM Accounts WHERE AccountID =
fromAccount;

  IF fromBalance >= amount THEN
    UPDATE Accounts SET Balance = Balance - amount WHERE AccountID
= fromAccount;
    UPDATE Accounts SET Balance = Balance + amount WHERE AccountID
= toAccount;
  END IF;
END;
/

BEGIN
  TransferFunds(1, 2, 3000);
END;
/

SELECT * FROM Accounts;
/
```

Output:

ACCOUNTID	NAME	BALANCE
1	Alice	10000.00
2	Bob	5000.00
3	Charlie	2000.00

ACCOUNTID	NAME	BALANCE
1	Alice	7000.00
2	Bob	8000.00
3	Charlie	2000.00

JUnit Testing Exercises

Exercise 1:Setting up JUnit:

1. Created a new Java project in Eclipse IDE

2. Added JUnit dependency to your project.

pom.xml

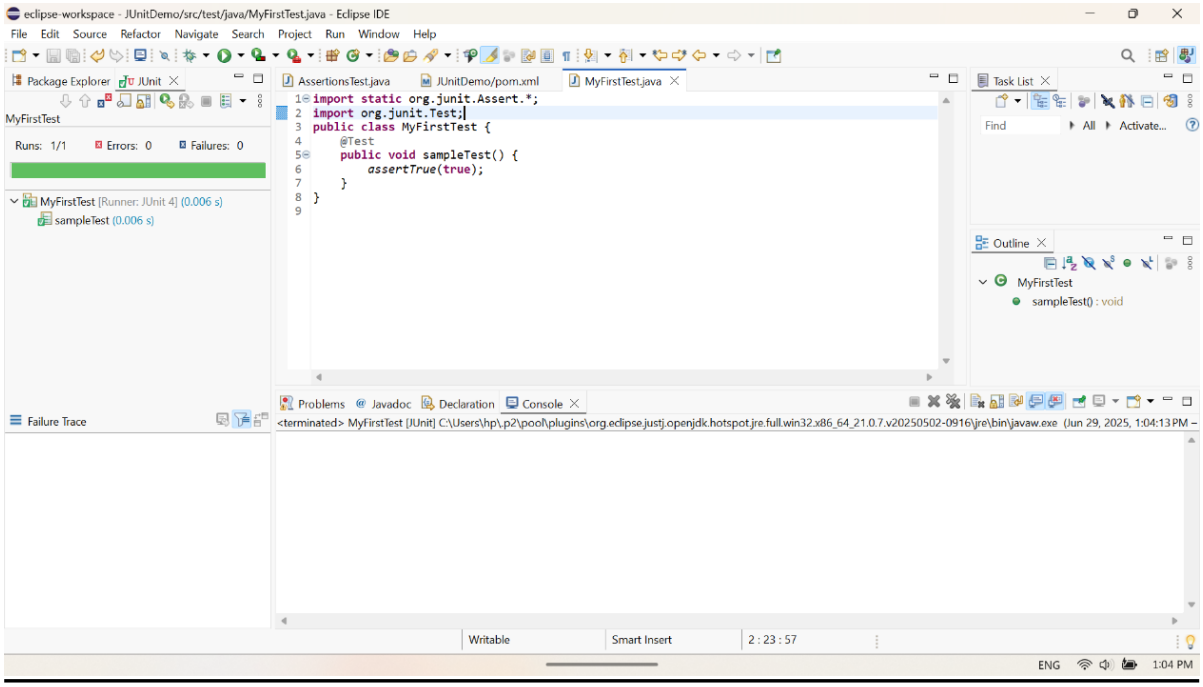
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://
maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>JUnitDemo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.13.2</version>
        <scope>test</scope>
    </dependency>
</dependencies>
</project>
```

3. Created a new test class in your project.

MyFirstTest.java

```
import static org.junit.Assert.*;
import org.junit.Test;
public class MyFirstTest {
     @Test
    public void sampleTest() {
        assertTrue(true);
     }
}
```

Output:

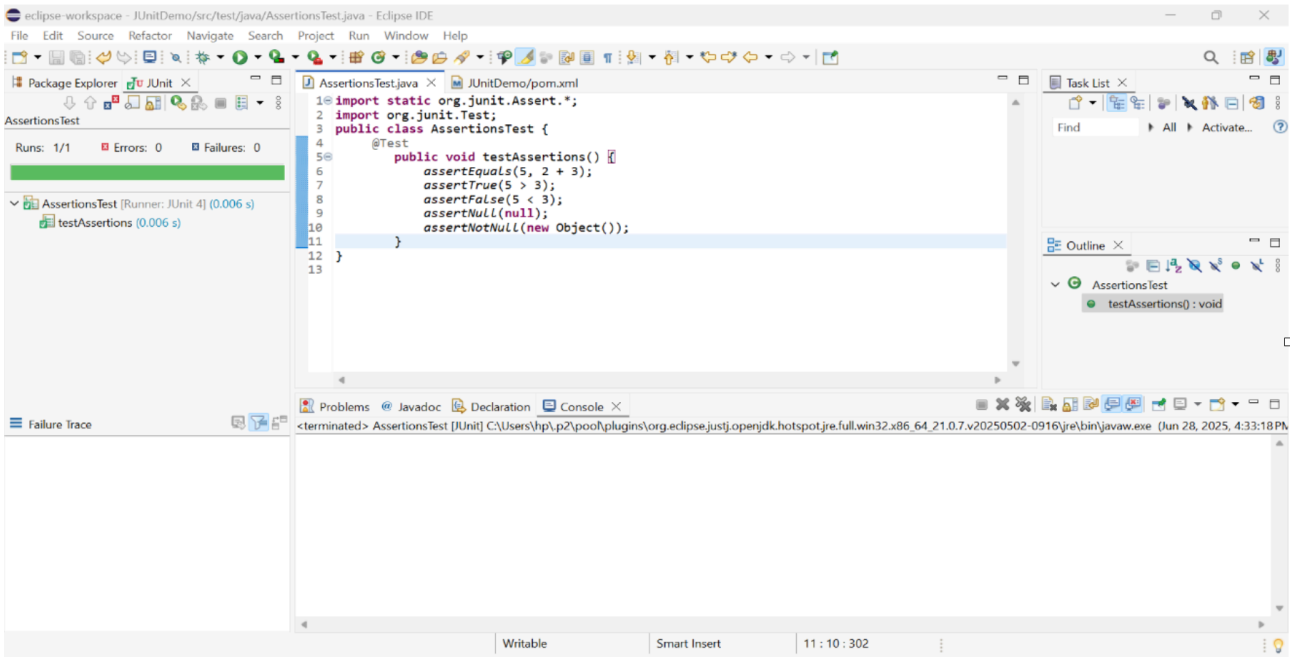


Exercise 3: Assertions in JUnit:

Write tests using various JUnit assertions.

```
import static org.junit.Assert.*;
import org.junit.Test;
public class AssertionsTest {
      @Test
         public void testAssertions() {
             assertEquals(5, 2 + 3);
             assertTrue(5 > 3);
             assertFalse(5 < 3);
             assertNull(null);
             assertNotNull(new Object());
         }
}
```

Output:



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

Calculator.java

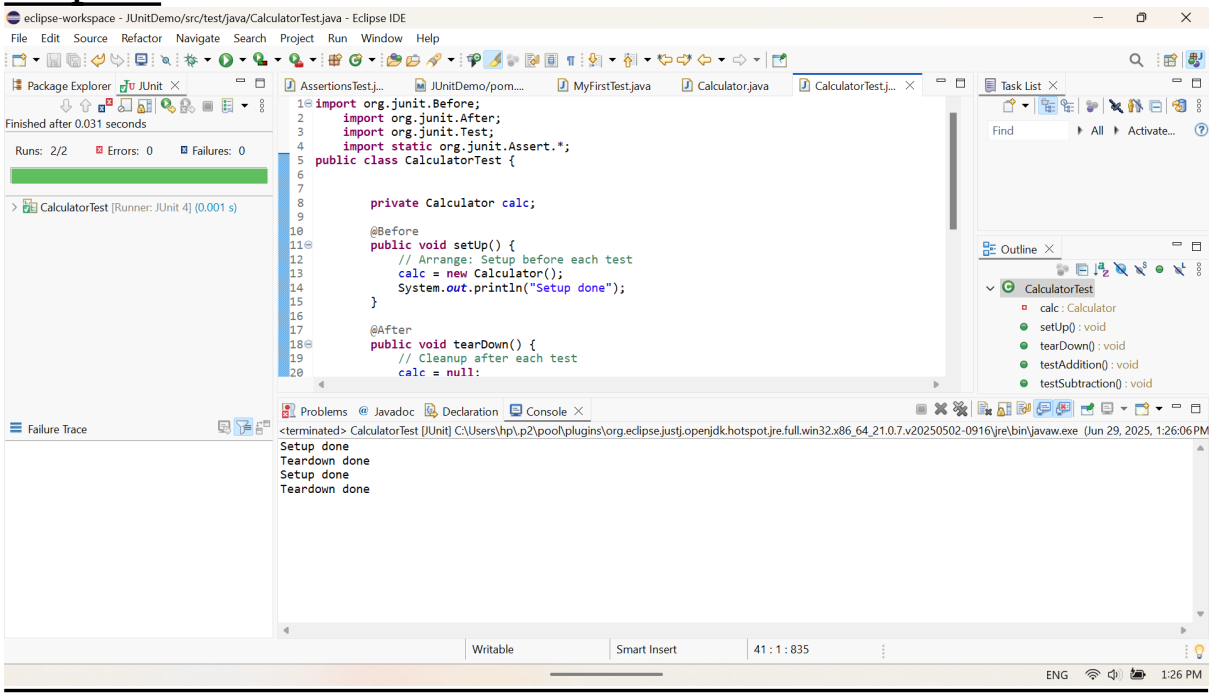
```
public class Calculator {

	public int add(int a, int b) {
	    return a + b;
	}

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


}
```

CalculatorTest.java

```
import org.junit.Before;
import org.junit.After;
import org.junit.Test;
import static org.junit.Assert.*;
public class CalculatorTest {

	private Calculator calc;

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

```
    @After
    public void tearDown() {
        // Cleanup after each test
        calc = null;
        System.out.println("Teardown done");
    }

    @Test
    public void testAddition() {
        // Act
        int result = calc.add(2, 3);

        // Assert
        assertEquals(5, result);
    }

    @Test
    public void testSubtraction() {
        int result = calc.subtract(10, 4);
        assertEquals(6, result);
    }
}
```

Output:



)Mockito

Created a Maven project named `MockitoDemo` in Eclipse and added JUnit 5 and Mockito dependencies in the `pom.xml` to enable unit testing and mocking.

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.example</groupId>
  <artifactId>JUnitDemo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.13.2</version>
        <scope>test</scope>
    </dependency>
</dependencies>
</project>
```

Exercise 1:Mocking and Stubbing:

1. Create a mock object for the external API.

   ### ExternalApi.java (Interface)

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

2. Stub the methods to return predefined values.

   ### MyService.java

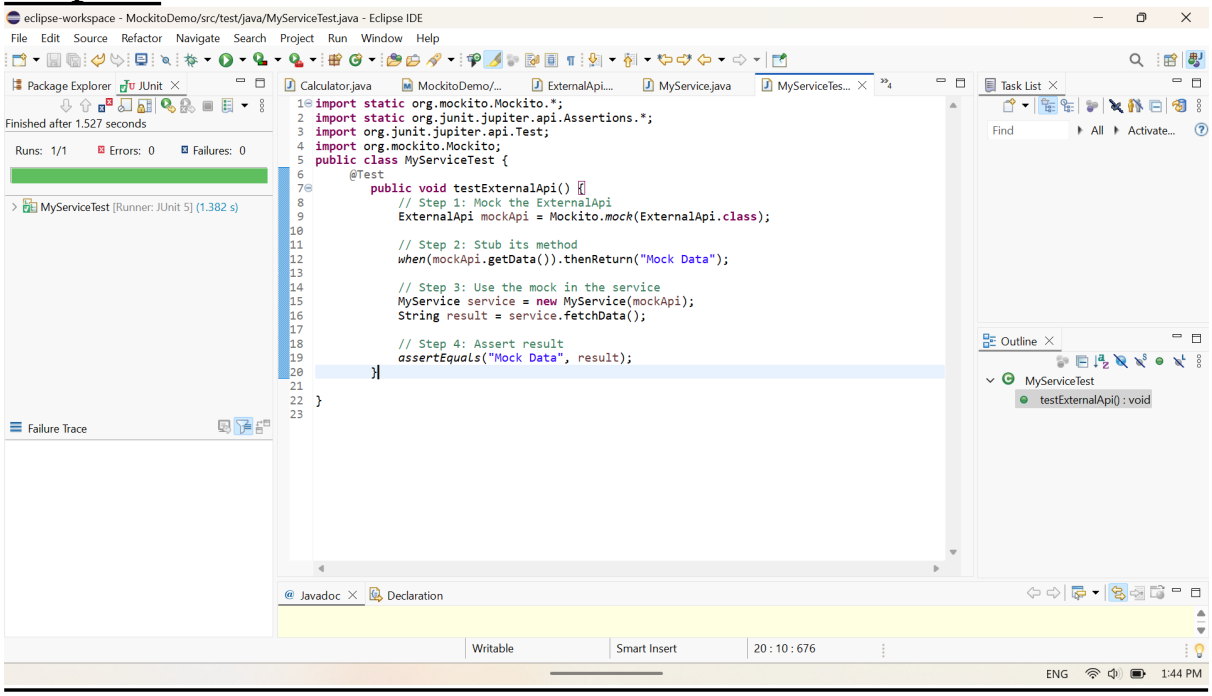
   ```
   public class MyService {
   private ExternalApi api;

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

   public String fetchData() {
     return api.getData();
   }

   }
   ```

3. Write a test case that uses the mock object.

MyServiceTest.java

```
import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
import org.mockito.Mockito;
public class MyServiceTest {
	@Test
	   public void testExternalApi() {
	       // Step 1: Mock the ExternalApi
	       ExternalApi mockApi = Mockito.mock(ExternalApi.class);

	       // Step 2: Stub its method
	       when(mockApi.getData()).thenReturn("Mock Data");

	       // Step 3: Use the mock in the service
	       MyService service = new MyService(mockApi);
	       String result = service.fetchData();

	       // Step 4: Assert result
	       assertEquals("Mock Data", result);
	   }

}
```

Output:



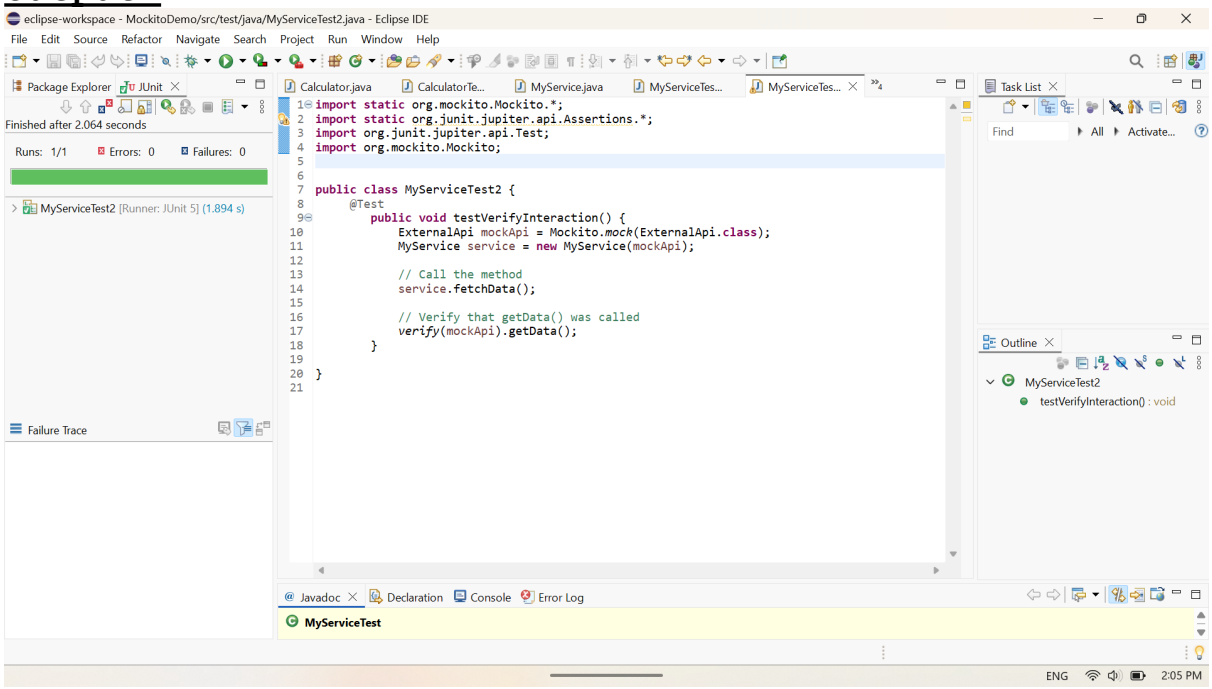
Exercise 2: Verifying Interactions:

MyServiceTest2.java

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


public class MyServiceTest2 {
```

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

            // Call the method
            service.fetchData();

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

}
```

Output:



1)SL4J Logging Exercises

Exercise 1:Logging error messages and Warning level:

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

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>
```

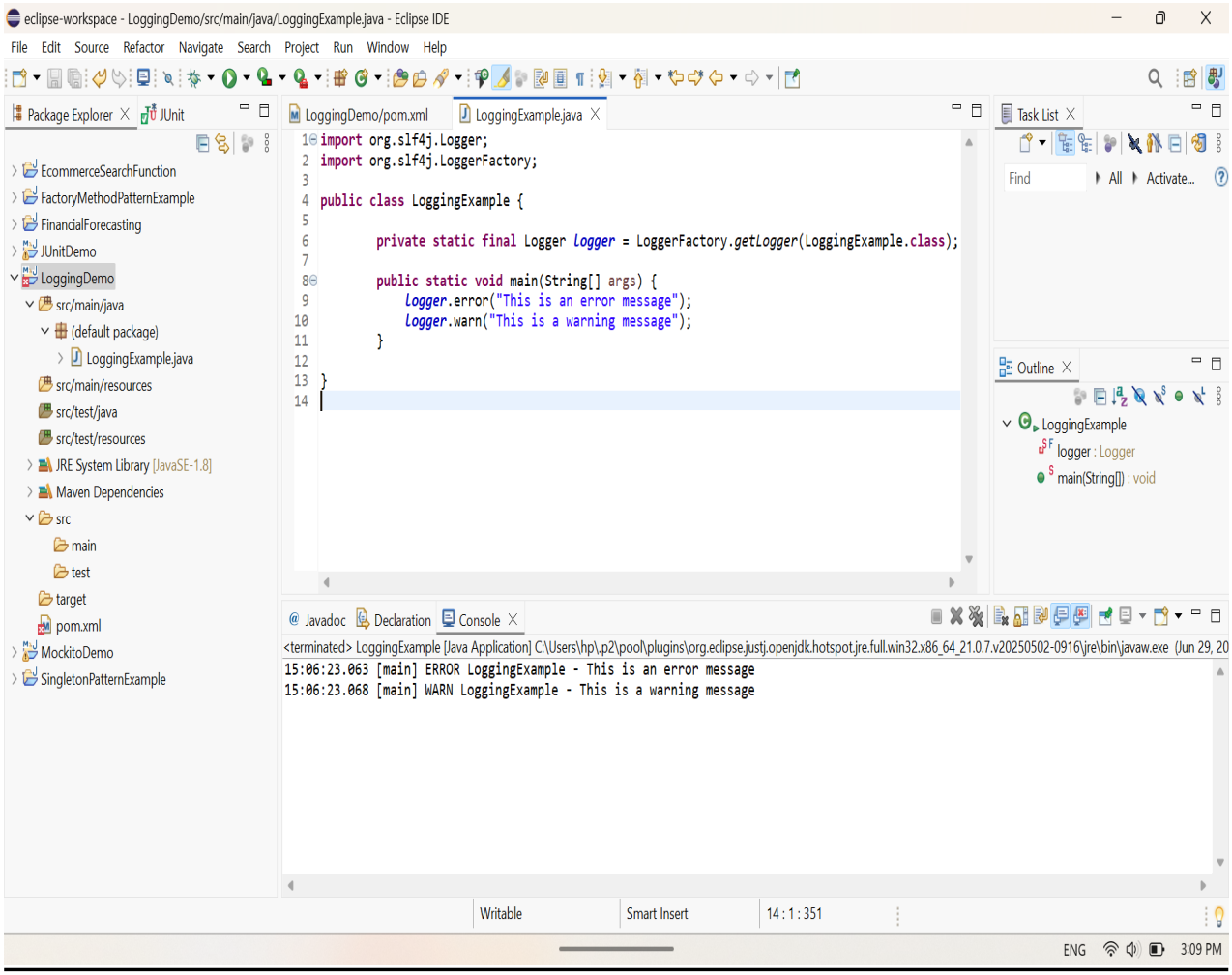
```
  <groupId>com.example</groupId>
  <artifactId>LoggingDemo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.30</version>
</dependency>
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>1.2.3</version>
</dependency>
</project>
```

2 . Create a Java class that uses SLF4J for logging:

LoggingExample.java

```
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:


eclipse-workspace - LoggingDemo/src/main/java/LoggingExample.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JUnit
EcommerceSearchFunction
FactoryMethodPatternExample
FinancialForecasting
JUnitDemo
LoggingDemo
src/main/java
(default package)
LoggingExample.java
src/main/resources
src/test/java
src/test/resources
JRE System Library [JavaSE-1.8]
Maven Dependencies
src
main
test
target
pom.xml
MockitoDemo
SingletonPatternExample
LoggingDemo/pom.xml
LoggingExample.java
1 import org.slf4j.Logger;
2 import org.slf4j.LoggerFactory;
3
4 public class LoggingExample {
5
6 private static final Logger logger = LoggerFactory.getLogger(LoggingExample.class);
7
8 public static void main(String[] args) {
9 logger.error("This is an error message");
10 logger.warn("This is a warning message");
11 }
12
13 }
14
Task List
Find
All
Activate...
Outline
LoggingExample
logger : Logger
main(String[]) : void
Javadoc
Declaration
Console
<terminated> LoggingExample [Java Application] C:\Users\hp\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v20250502-0916\jre\bin\javaw.exe (Jun 29, 20
15:06:23.063 [main] ERROR LoggingExample - This is an error message
15:06:23.068 [main] WARN LoggingExample - This is a warning message
Writable
Smart Insert
14 : 1 : 351
ENG
3:09 PM