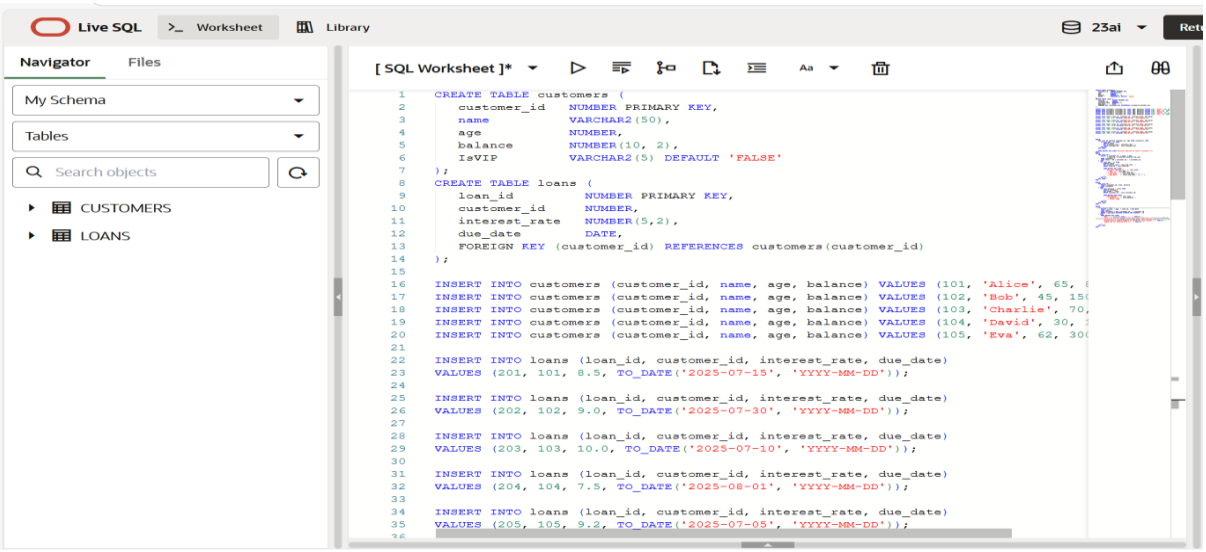
WEEK – 2

Exercise 1: Control Structures

Table created and inserted the value



i) Scenario 1:

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.

[SQL Worksheet]*

```oracle_plsql_block


BEGIN
   FOR cust IN (
      SELECT c.customer_id, c.name, c.age,
             l.loan_id, l.interest_rate AS old_rate
      FROM customers c
      JOIN loans l ON c.customer_id = l.customer_id
   ) LOOP
      IF cust.age > 60 THEN
         UPDATE loans
         SET interest_rate = cust.old_rate - 1
         WHERE loan_id = cust.loan_id;

         DBMS_OUTPUT.PUT_LINE(
            'Discount applied to: ' || cust.name ||
            ' | Age: ' || cust.age ||
            ' | Loan ID: ' || cust.loan_id ||
            ' | Old Rate: ' || cust.old_rate || '%' ||
            ' | New Rate: ' || (cust.old_rate - 1) || '%'
         );
      END IF;
   END LOOP;
END;
```

Output:

Query result | **Script output** | DBMS output | Explain Plan | SQL history

```
Discount applied to: Alice | Age: 65 | Loan ID: 201 | Old Rate: 6.5% | New Rate: 5.5%
Discount applied to: Charlie | Age: 70 | Loan ID: 203 | Old Rate: 8% | New Rate: 7%
Discount applied to: Eva | Age: 62 | Loan ID: 205 | Old Rate: 7.2% | New Rate: 6.2%


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.121
```

ii) Scenario 2:

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.

[SQL Worksheet]*

```
BEGIN
   FOR cust IN (
      SELECT customer_id, name, balance
      FROM customers
   ) LOOP
      IF cust.balance > 10000 THEN
         UPDATE customers
         SET IsVIP = 'TRUE'
         WHERE customer_id = cust.customer_id;

         DBMS_OUTPUT.PUT_LINE(
            'VIP updated: ' || cust.name ||
            ' | Balance: $' || cust.balance ||
            ' | IsVIP: TRUE'
         );
      END IF;
   END LOOP;
END;
```

Output :

Query result | **Script output** | DBMS output | Explain Plan | SQL history

```
VIP updated: Eva | Balance: $30000 | IsVIP: TRUE
VIP updated: Bob | Balance: $15000 | IsVIP: TRUE
VIP updated: David | Balance: $12000 | IsVIP: TRUE


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.014
```

ii) Scenario 3:

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

[SQL Worksheet]*

```oracle_plsql
BEGIN
   FOR loan_rec IN (
      SELECT c.name, c.age, l.loan_id, l.due_date
      FROM loans l
      JOIN customers c ON l.customer_id = c.customer_id
      WHERE l.due_date BETWEEN SYSDATE AND SYSDATE + 30
   ) LOOP
      DBMS_OUTPUT.PUT_LINE(
         'Dear ' || loan_rec.name || ',' || CHR(10) ||
         'This is a gentle reminder that your loan (Loan ID: ' || loan_rec.loan_id || ')' || CH
         'is due on ' || TO_CHAR(loan_rec.due_date, 'Day, DD-Mon-YYYY') || '.' || CHR(10) ||
         'Please ensure timely payment to avoid any late charges.' || CHR(10) ||
         'Thank you for banking with us!' || CHR(10) ||
         '---'
      );
   END LOOP;
END;
```

Output:

Query result | **Script output** | DBMS output | Explain Plan | SQL history

```
Dear Alice,
This is a gentle reminder that your loan (Loan ID: 201)
is due on Tuesday  , 15-Jul-2025.
Please ensure timely payment to avoid any late charges.
Thank you for banking with us!
---
Dear Charlie,
This is a gentle reminder that your loan (Loan ID: 203)
is due on Thursday , 10-Jul-2025.
Please ensure timely payment to avoid any late charges.
Thank you for banking with us!
---
Dear Eva,
This is a gentle reminder that your loan (Loan ID: 205)
is due on Saturday , 05-Jul-2025.
Please ensure timely payment to avoid any late charges.
Thank you for banking with us!
---


PL/SQL procedure successfully completed.
```

Exercise 3: Stored Procedures

Table Created and Row inserted

[SQL Worksheet]*

```sql

CREATE TABLE accounts (
    account_id      NUMBER PRIMARY KEY,
    customer_name   VARCHAR2(50),
    account_type    VARCHAR2(20),
    balance         NUMBER(10, 2)
);

INSERT INTO accounts VALUES (1001, 'Alice', 'Savings', 5000);
INSERT INTO accounts VALUES (1002, 'Bob', 'Savings', 15000);
INSERT INTO accounts VALUES (1003, 'Charlie', 'Current', 7000);
INSERT INTO accounts VALUES (1004, 'David', 'Savings', 8000);
COMMIT;

CREATE TABLE employees (
    emp_id        NUMBER PRIMARY KEY,
    name          VARCHAR2(50),
    department    VARCHAR2(30),
    salary        NUMBER(10,2)
);

INSERT INTO employees VALUES (1, 'Anjali', 'HR', 40000);
INSERT INTO employees VALUES (2, 'Ravi', 'IT', 60000);
INSERT INTO employees VALUES (3, 'Sneha', 'IT', 65000);
INSERT INTO employees VALUES (4, 'Vikas', 'Finance', 55000);
COMMIT;

```

i)Scenario 1:

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.

[SQL Worksheet]*

```sql

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest
IS
    v_interest NUMBER;
BEGIN
    FOR acc IN (SELECT account_id, customer_name, balance FROM accounts WHERE account_type = 'Sa
        v_interest := acc.balance * 0.01;

        UPDATE accounts
        SET balance = balance + v_interest
        WHERE account_id = acc.account_id;

        DBMS_OUTPUT.PUT_LINE('Interest applied to ' || acc.customer_name ||
                             ' | Interest: $' || v_interest ||
                             ' | New Balance: $' || (acc.balance + v_interest));
    END LOOP;
END;

BEGIN
    ProcessMonthlyInterest;
END;
```

Output

Query result | Script output | DBMS output | Explain Plan | SQL history

```
SQL> BEGIN
        ProcessMonthlyInterest;
     END;


Interest applied to Alice | Interest: $50 | New Balance: $5050
Interest applied to Bob | Interest: $150 | New Balance: $15150
Interest applied to David | Interest: $80 | New Balance: $8080


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.016
```

ii)Scenario 2:

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

[SQL Worksheet]*

```
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (
   p_department IN VARCHAR2,
   p_bonus_pct  IN NUMBER
)
IS
   v_bonus NUMBER;
BEGIN
   FOR emp IN (
      SELECT emp_id, name, salary
      FROM employees
      WHERE department = p_department
   ) LOOP
      v_bonus := ROUND(emp.salary * (p_bonus_pct / 100), 2);

      UPDATE employees
      SET salary = salary + v_bonus
      WHERE emp_id = emp.emp_id;

      DBMS_OUTPUT.PUT_LINE(
         'Bonus added for: ' || RPAD(emp.name, 10) ||
         '| Bonus Amount: ' || TO_CHAR(v_bonus, '99999.99') ||
         '| New Salary: ' || TO_CHAR(emp.salary + v_bonus, '999999.99')
      );
   END LOOP;
END;
/
```

Output

```
Bonus added for: Alice Johnson | Bonus Amount:  6000.00 | New Salary: 76000.00
Bonus added for: Bob Brown     | Bonus Amount:  6500.00 | New Salary: 66500.00
```

iii)Scenario 3:

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.

[SQL Worksheet]*

```oracle_plsql
CREATE OR REPLACE PROCEDURE TransferFunds (
    p_from_account IN NUMBER,
    p_to_account   IN NUMBER,
    p_amount       IN NUMBER
)
IS
    v_from_balance  NUMBER;
    v_to_balance    NUMBER;
    v_from_name     VARCHAR2(50);
    v_to_name       VARCHAR2(50);
BEGIN

    SELECT balance, customer_name INTO v_from_balance, v_from_name
    FROM accounts WHERE account_id = p_from_account;

    SELECT balance, customer_name INTO v_to_balance, v_to_name
    FROM accounts WHERE account_id = p_to_account;


    IF v_from_balance < p_amount THEN
        DBMS_OUTPUT.PUT_LINE('Transfer failed: ' || v_from_name || ' (Account ' || p_from_account
                             ') has insufficient funds. Available: ₹' || TO_CHAR(v_from_balance,
    ELSE

        UPDATE accounts
        SET balance = balance - p_amount
        WHERE account_id = p_from_account;

        UPDATE accounts
        SET balance = balance + p_amount
        WHERE account_id = p_to_account;

        DBMS_OUTPUT.PUT_LINE('Transfer Successful!');
        DBMS_OUTPUT.PUT_LINE('Amount Transferred: ₹' || TO_CHAR(p_amount, '99999.99'));
        DBMS_OUTPUT.PUT_LINE('From: ' || v_from_name || ' (Account ' || p_from_account || ')');
```

[SQL Worksheet]*

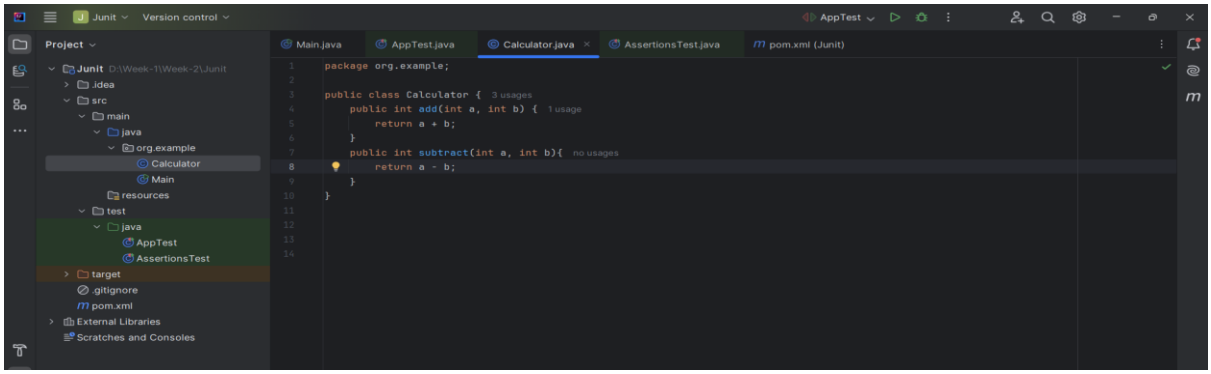
```oracle_plsql
    IF v_from_balance < p_amount THEN
        DBMS_OUTPUT.PUT_LINE('Transfer failed: ' || v_from_name || ' (Account ' || p_from_account
                             ') has insufficient funds. Available: ₹' || TO_CHAR(v_from_balance,
    ELSE

        UPDATE accounts
        SET balance = balance - p_amount
        WHERE account_id = p_from_account;

        UPDATE accounts
        SET balance = balance + p_amount
        WHERE account_id = p_to_account;

        DBMS_OUTPUT.PUT_LINE('Transfer Successful!');
        DBMS_OUTPUT.PUT_LINE('Amount Transferred: ₹' || TO_CHAR(p_amount, '99999.99'));
        DBMS_OUTPUT.PUT_LINE('From: ' || v_from_name || ' (Account ' || p_from_account || ')');
        DBMS_OUTPUT.PUT_LINE('To:   ' || v_to_name   || ' (Account ' || p_to_account || ')');


        DBMS_OUTPUT.PUT_LINE('Updated Balance - ' || v_from_name || ': ' || TO_CHAR(v_from_balanc
        DBMS_OUTPUT.PUT_LINE('Updated Balance - ' || v_to_name   || ': ' || TO_CHAR(v_to_balance
    END IF;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Transfer failed: One or both account IDs not found.');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An unexpected error occurred: ' || SQLERRM);
END;
/

BEGIN
    TransferFunds(1002, 1003, 2000);
END;
```

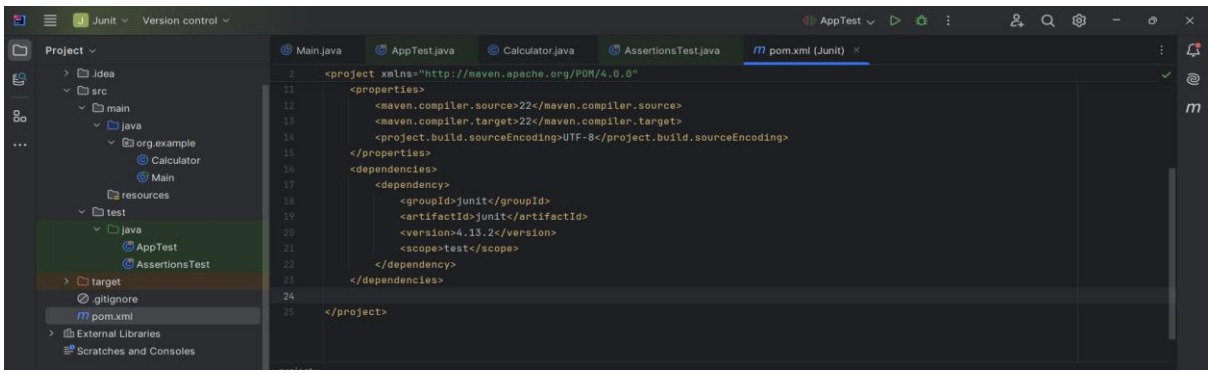
Output

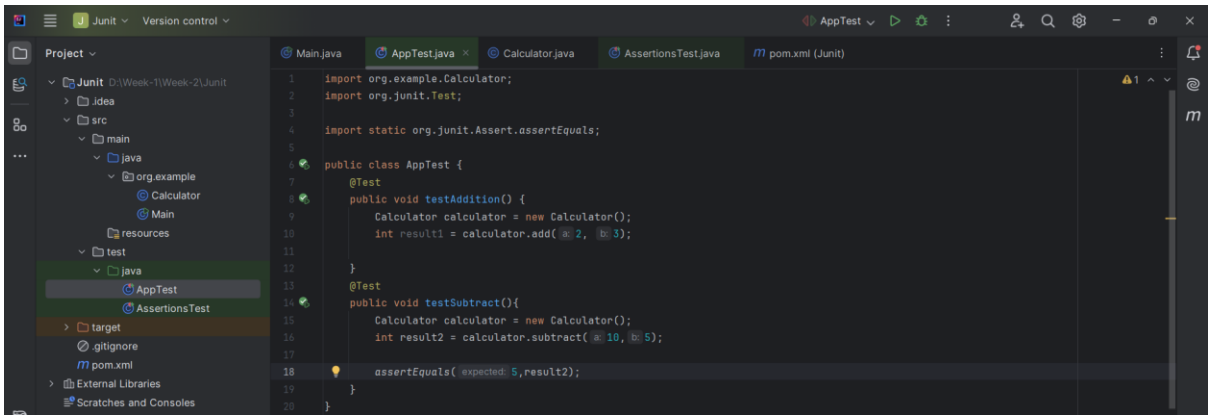
Query result | **Script output** | DBMS output | Explain Plan | SQL history

```
Elapsed: 00:00:00.019

SQL> BEGIN
        TransferFunds(1002, 1003, 2000); -- Bob to Charlie
     END;


Transfer Successful!
Amount Transferred: ₹  2000.00
From: Bob (Account 1002)
To:   Charlie (Account 1003)
Updated Balance - Bob:  11150.00
Updated Balance - Charlie:  11000.00


PL/SQL procedure successfully completed.
```

Exercise 1: Setting Up Junit

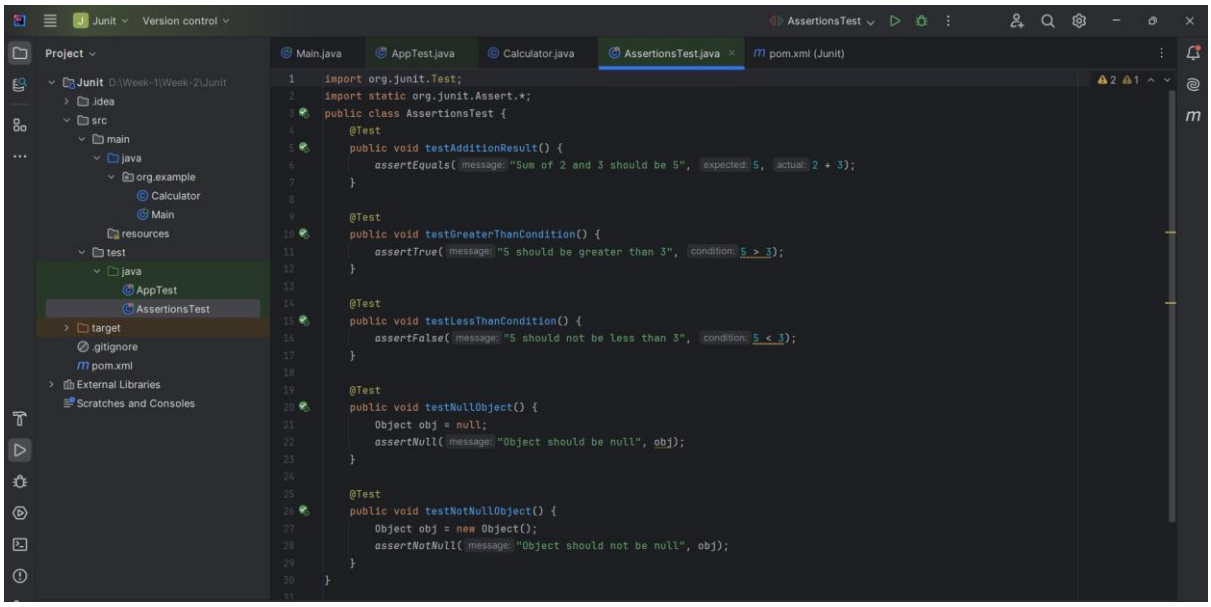






Output:

Exercise 3: Assertions in Junit



Output:

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

```java
public class CalculatorTest {
    public void setUp() {
        System.out.println("Setting up Calculator");
        Calculator calculator = new Calculator();
    }
    @After
    public void tearDown() {
        System.out.println("Cleaning up Calculator");
        Calculator calculator = null;
    }
    @Test
    public void testAddition() {
        Calculator calculator = new Calculator();
        int a = 10;
        int b = 20;
        int result = calculator.add(a, b);
        assertEquals( message: "10 + 20 should be 30",  expected: 30, result);
    }
    @Test
    public void testSubtraction() {
        Calculator calculator = new Calculator();
        int a = 50;
        int b = 20;
        int result = calculator.subtract(a, b);
        assertEquals( message: "50 - 20 should be 30",  expected: 30, result);
    }
    @Test
    public void testMultiply() {
        Calculator calculator = new Calculator();
        int a = 50;
        int b = 20;
```

```java
public class CalculatorTest {
    @Test
    public void testAddition() {
        Calculator calculator = new Calculator();
        int a = 10;
        int b = 20;
        int result = calculator.add(a, b);
        assertEquals( message: "10 + 20 should be 30",  expected: 30, result);
    }
    @Test
    public void testSubtraction() {
        Calculator calculator = new Calculator();
        int a = 50;
        int b = 20;
        int result = calculator.subtract(a, b);
        assertEquals( message: "50 - 20 should be 30",  expected: 30, result);
    }
    @Test
    public void testMultiply() {
        Calculator calculator = new Calculator();
        int a = 50;
        int b = 20;
        int result = calculator.multiply(a, b);
        assertEquals( message: "50 * 20 should be 1000",  expected: 100, result);
    }
}
```

Output:

```
Run   CalculatorTest
CalculatorTest                21 ms   Tests failed: 1, passed: 2 of 3 tests – 21 ms
  testAddition                 9 ms   "C:\Program Files\Java\jdk-22\bin\java.exe" -ea -Didea.test.cyclic.buffer.size=1048576 "-javaagent:D:\1Installed Files\IntelliJ IDEA Commun
  testMultiply                12 ms   Setting up Calculator
  testSubtraction              0 ms   Cleaning up Calculator
                                      Setting up Calculator
                                      Cleaning up Calculator

                                      java.lang.AssertionError: 50 * 20 should be 1000
                                      Expected :100
                                      Actual   :1000
                                      <Click to see difference>

                                      <1 internal line>
                                          at org.junit.Assert.failNotEquals(Assert.java:835) <1 internal line>
                                          at CalculatorTest.testMultiply(CalculatorTest.java:59) <27 internal lines>

                                      Setting up Calculator
                                      Cleaning up Calculator

                                      Process finished with exit code -1
```

Exercise 1: Mocking and Stubbing

1.ExternalApi Interface

```java
package org.example;

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

2.MyService Class

```java
package org.example;

public class MyService {
    private ExternalApi externalApi;

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

    public String fetchData() {
        System.out.println("Fetching data from mock API");
        return externalApi.getData();
    }
}
```

3.MyServiceTest

```java
import static org.junit.Assert.assertEquals;
import static org.mockito.Mockito.*;

public class MyServiceTest {

    @Test
    public void testExternalApi() {
        System.out.println("Test started...");
        ExternalApi mockApi = mock(ExternalApi.class);
        when(mockApi.getData()).thenReturn("Mock Data");
        MyService service = new MyService(mockApi);
        String result = service.fetchData();
        System.out.println("Result: " + result);
        assertEquals("Mock Data", result);
        verify(mockApi).getData();
        System.out.println("Test finished.");
    }
}
```

Output

```
Run  MyServiceTest.testExternalApi
MyServiceTest  1 sec 72 ms
  testExternalApi  1 sec 72 ms
Tests passed: 1 of 1 test – 1 sec 72 ms
Java HotSpot(TM) 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended
Fetching data from mock API
Result: Mock Data
Test finished.

Process finished with exit code 0
```

Exercise 2: Verifying Interactions

1.ExternalApi

```java
package org.example;

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

2.MyService Class

```java
package org.example;

public class MyService2 {
    private ExternalApi externalApi;
    public MyService2(ExternalApi externalApi) {
        this.externalApi = externalApi;
    }
    public String fetchData() {
        return externalApi.getDataById("12345");
    }
}
```

3.MyServiceTest

```java
public class MyServiceTest2 {
    @Test
    public void testVerifyInteractionWithArgs() {
        ExternalApi mockApi = mock(ExternalApi.class);
        MyService2 service = new MyService2(mockApi);
        service.fetchData();
        verify(mockApi).getDataById("12345");
        System.out.println();
        System.out.println();
        System.out.println();
        System.out.println();
        System.out.println("Verified: getDataById() was called.");
        System.out.println("Test Passed.");
    }
}
```

Output

```
Verified: getDataById() was called.
Test Passed.

Process finished with exit code 0
```

Exercise 1: Logging Error Messages and Warning Levels

1) Pom.xlm

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    <dependencies>
        <dependency>
        </dependency>
        <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) LoggingExample

```
package org.example;

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

public class LoggingExample {
    private static final Logger logger = LoggerFactory.getLogger(LoggingExample.class);  2 usages

    public static void main(String[] args) {
        System.out.println("Starting application...");

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

        System.out.println("Finished logging.");
    }
}
```

Output

```
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:D:\1Installed Files\IntelliJ IDEA Community Edition 2024.1.3\lib\idea_rt.jar=53651:D:\1Installed Files\IntelliJ IDEA Com
Starting application...
21:07:13.541 [main] ERROR org.example.LoggingExample - This is an error message
21:07:13.545 [main] WARN org.example.LoggingExample - This is a warning message
Finished logging.

Process finished with exit code 0
```