Exercise 1: Control Structures

Exercise 3: Stored Procedures

Exercise 1: Setting Up JUnit

Exercise 3: Assertions in JUnit

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

Teardown Methods in JUnit

Exercise 1: Mocking and Stubbing

Exercise 2: Verifying Interactions

Exercise 1: Logging Error Messages and Warning Levels

**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.

DECLARE

CURSOR cur\_customers IS

SELECT customer\_id, interest\_rate

FROM loans

JOIN customers ON loans.customer\_id = customers.customer\_id

WHERE age > 60;

BEGIN

FOR rec IN cur\_customers LOOP

UPDATE loans

SET interest\_rate = interest\_rate - 1

WHERE customer\_id = rec.customer\_id;

DBMS\_OUTPUT.PUT\_LINE('Discount applied to Customer ID: ' || rec.customer\_id);

END LOOP;

END;

/

**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.

DECLARE

CURSOR cur\_customers IS

SELECT customer\_id, balance

FROM customers

WHERE balance > 10000;

BEGIN

FOR rec IN cur\_customers LOOP

UPDATE customers

SET IsVIP = 'TRUE'

WHERE customer\_id = rec.customer\_id;

DBMS\_OUTPUT.PUT\_LINE('Customer ID ' || rec.customer\_id || ' promoted to VIP.');

END LOOP;

END;

/

**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.

DECLARE

CURSOR cur\_due\_loans IS

SELECT c.customer\_id, c.name, 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;

BEGIN

FOR rec IN cur\_due\_loans LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan for Customer ' || rec.name ||

' (ID: ' || rec.customer\_id ||

') is due on ' || TO\_CHAR(rec.due\_date, 'DD-MON-YYYY'));

END LOOP;

END;

/

**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 OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE accounts

SET balance = balance + (balance \* 0.01)

WHERE account\_type = 'SAVINGS';

DBMS\_OUTPUT.PUT\_LINE('Monthly interest applied to all savings accounts.');

END;

/

**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(

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;

/

**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(

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) IS

v\_balance NUMBER;

BEGIN

-- Check source account balance

SELECT balance INTO v\_balance

FROM accounts

WHERE account\_id = p\_from\_account

FOR UPDATE;

IF v\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance in source account.');

END IF;

-- Deduct from source

UPDATE accounts

SET balance = balance - p\_amount

WHERE account\_id = p\_from\_account;

-- Add to destination

UPDATE accounts

SET balance = balance + p\_amount

WHERE account\_id = p\_to\_account;

DBMS\_OUTPUT.PUT\_LINE('Transferred ' || p\_amount ||

' from Account ' || p\_from\_account ||

' to Account ' || p\_to\_account);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20002, 'One or both accounts not found.');

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Unexpected error: ' || SQLERRM);

END;

/

**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>

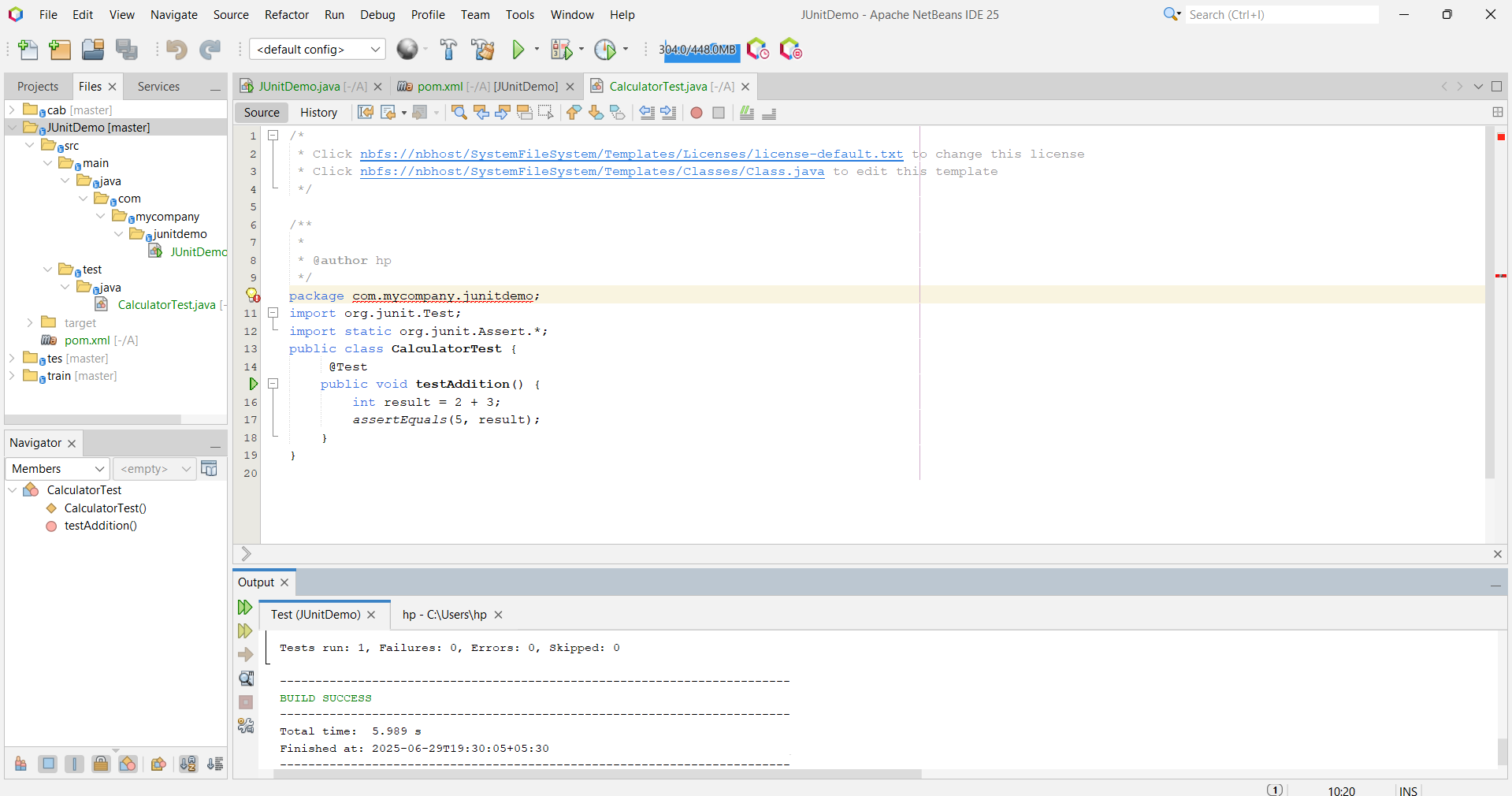
<artifactId>junit</artifactId>

<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:

Write tests using various JUnit assertions.

Solution Code:

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());

}

}

A screenshot of a computer

AI-generated content may be incorrect.

**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.

**SOLUTION:**

**1. AAA Pattern and Setup/Teardown with @Before and @After**

package com.example;

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

private Calculator calculator;

// Setup method: runs before each test

@Before

public void setUp() {

calculator = new Calculator(); // Arrange

System.out.println("Setup complete");

}

// Teardown method: runs after each test

@After

public void tearDown() {

calculator = null;

System.out.println("Teardown complete");

}

@Test

public void testAddition() {

// Arrange is already done in setUp()

// Act

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

// Assert

assertEquals("2 + 3 should be 5", 5, result);

}

@Test

public void testSubtraction() {

// Act

int result = calculator.subtract(5, 2);

// Assert

assertEquals("5 - 2 should be 3", 3, result);

}

}

**Calculator.java (class under test)**

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;

}

}

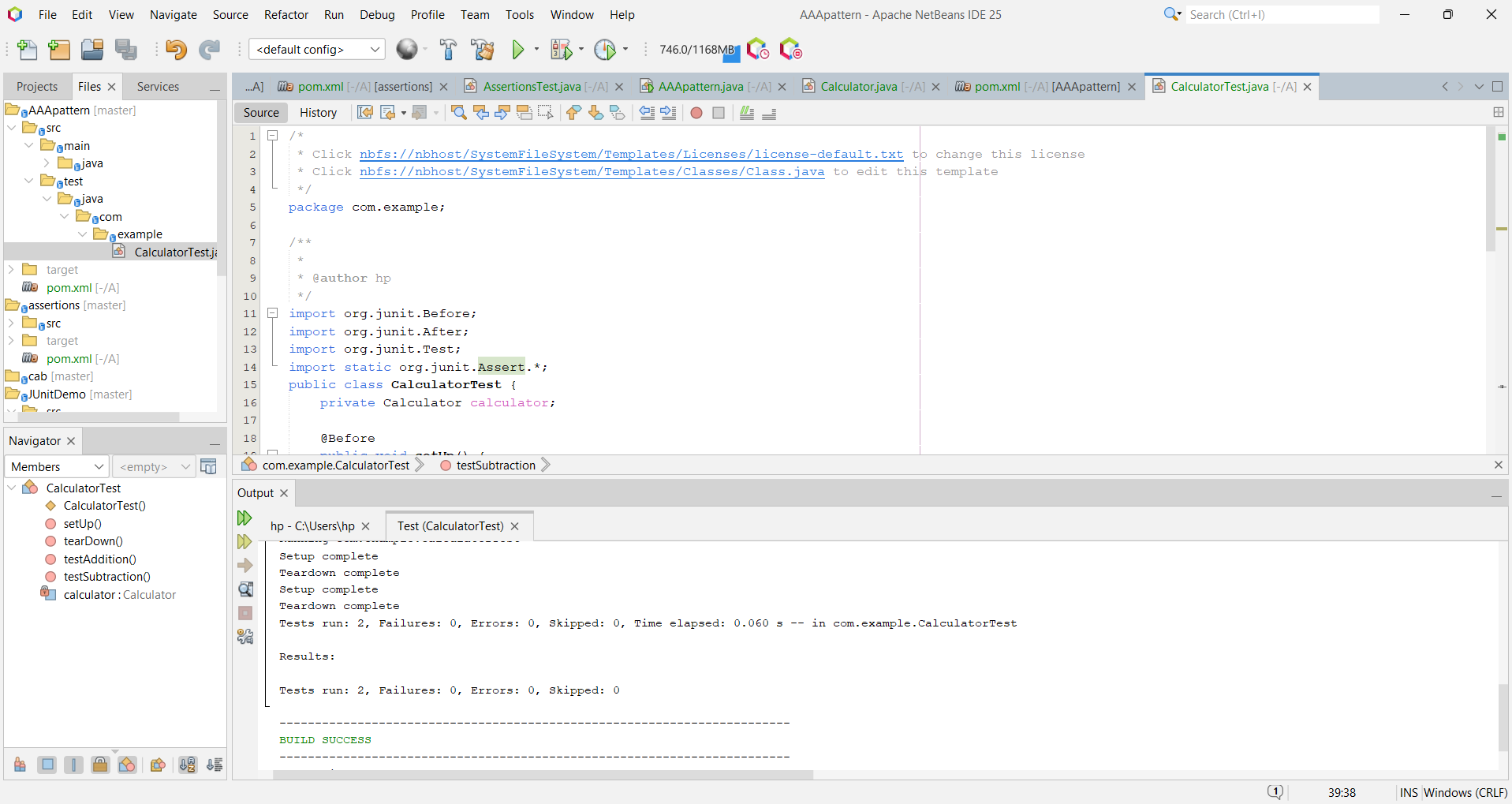
**Output Example in Console:**

Setup complete

Teardown complete

Setup complete

Teardown complete



**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;

import static org.junit.jupiter.api.Assertions.\*;

// Example service that uses an external API

public class MyServiceTest {

@Test

public void testExternalApi() {

// Step 1: Create a mock object for the external API

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

// Step 2: Stub the method to return predefined data

when(mockApi.getData()).thenReturn("Mock Data");

// Step 3: Inject the mock into the service and test the behavior

MyService service = new MyService(mockApi);

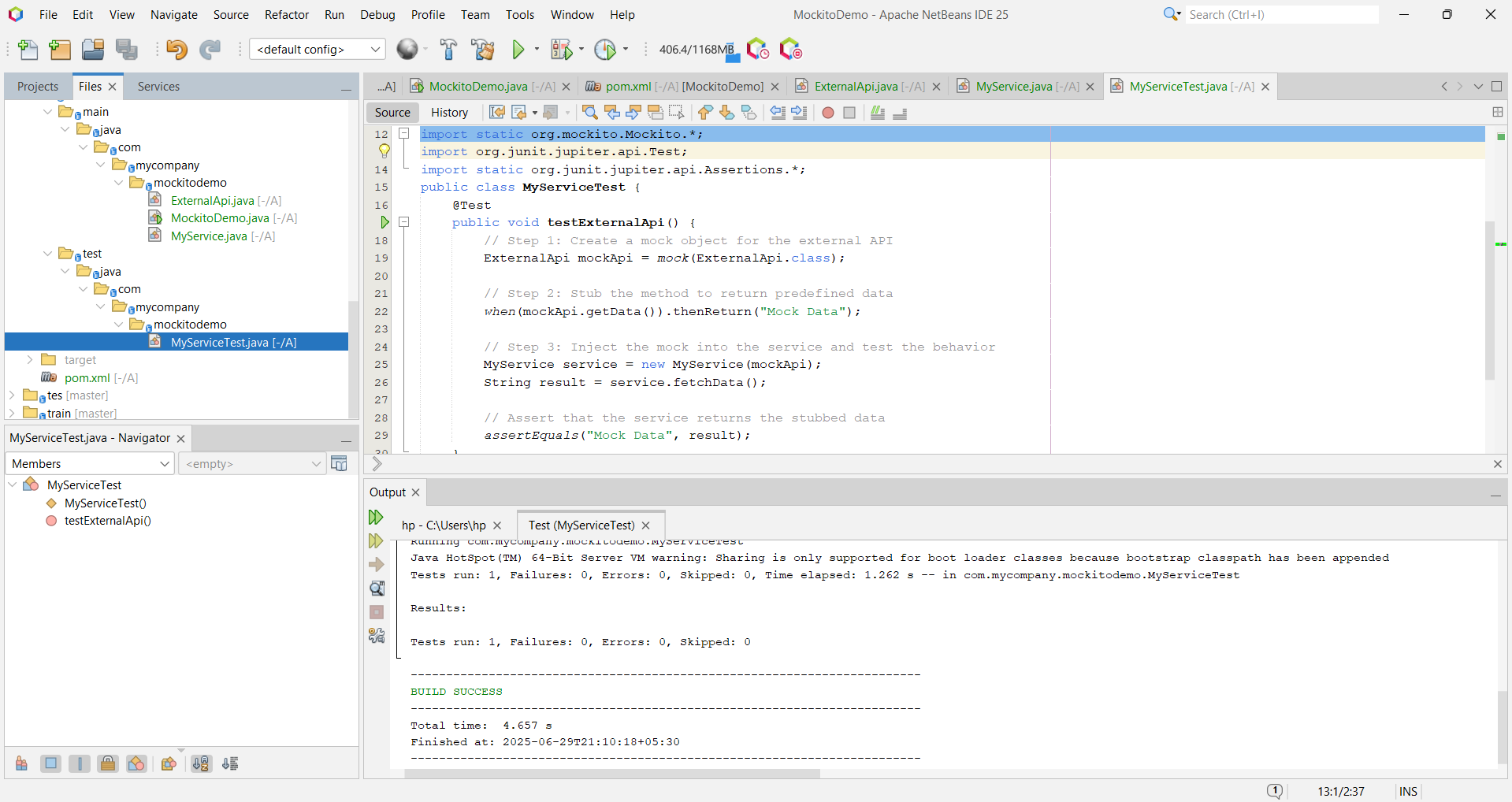
String result = service.fetchData();

// Assert that the service returns the stubbed data

assertEquals("Mock Data", result);

}

}



**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;

import static org.junit.jupiter.api.Assertions.\*;

// Test to verify method interaction using Mockito

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

// Step 1: Create a mock object for the external API

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

// Step 2: Inject the mock into the service

MyService service = new MyService(mockApi);

// Step 3: Call the method that should invoke the mock

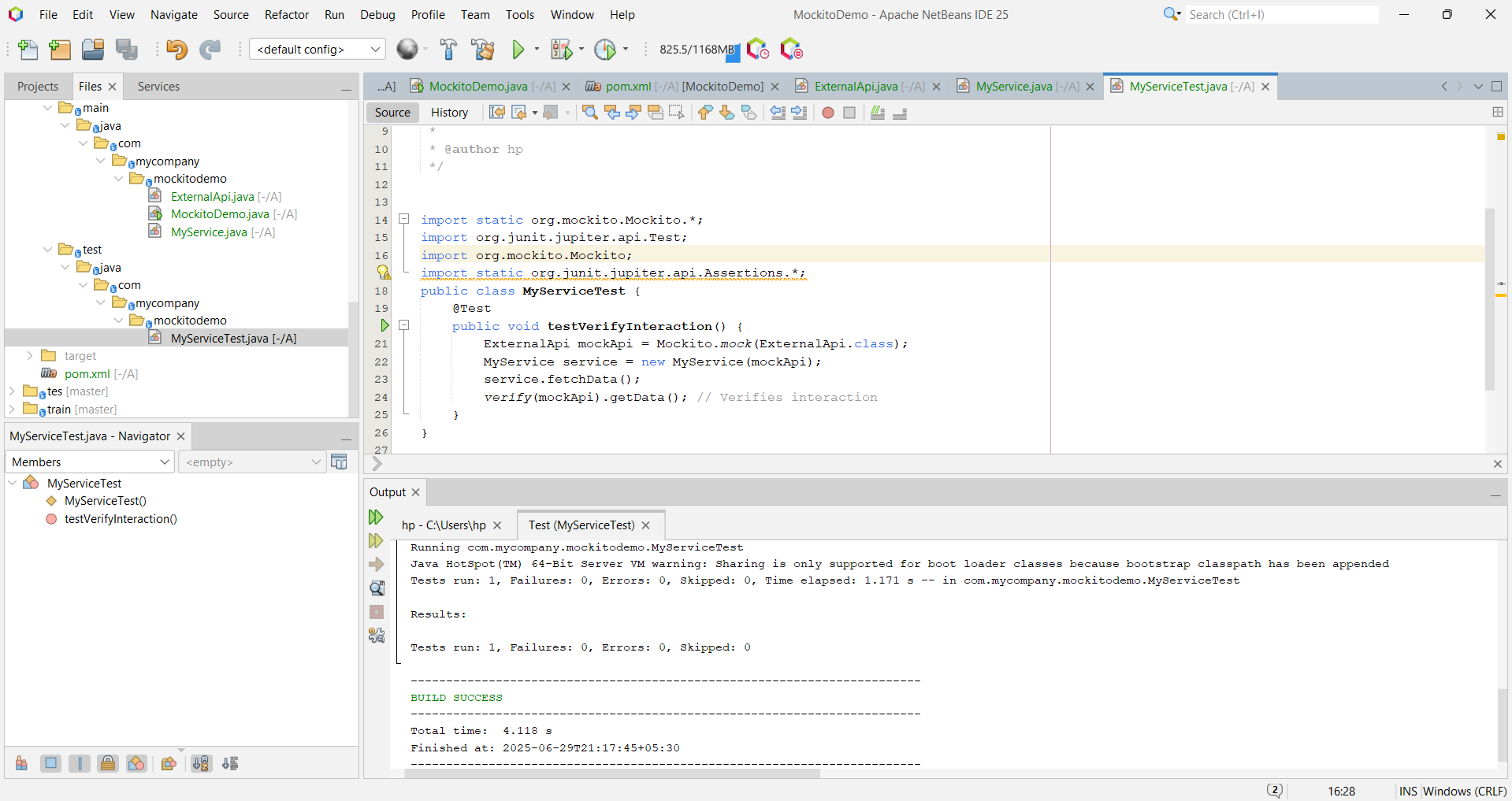
service.fetchData();

// Step 4: Verify that the expected method was called

verify(mockApi).getData();

}

}



**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>

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");

}

}

