**WEEK – 2**

**PL/SQL Programming**

**Exercise 1:** Control Structures

**Scenario 1:** Apply 1% Discount for Customers Above 60

BEGIN

FOR i IN (SELECT \* FROM Loans) LOOP

DECLARE

v\_dob DATE;

v\_age NUMBER;

BEGIN

SELECT DOB INTO v\_dob FROM Customers WHERE CustomerID = i.CustomerID;

v\_age := FLOOR(MONTHS\_BETWEEN(SYSDATE, v\_dob) / 12);

IF v\_age > 60 THEN

UPDATE Loans SET InterestRate = InterestRate - 1 WHERE LoanID = i.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Applied discount to Loan ID: ' || i.LoanID);

END IF;

END;

END LOOP;

END;

/

**Output:**

Applied discount to Loan ID: 101

Applied discount to Loan ID: 103

**Scenario 2: Promote Customers to VIP with Balance > 10,000**

BEGIN

FOR i IN (SELECT \* FROM Customers) LOOP

IF i.Balance > 10000 THEN

UPDATE Customers SET IsVIP = 'Y' WHERE CustomerID = i.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('VIP status given to Customer ID: ' || i.CustomerID);

END IF;

END LOOP;

END;

/

**Output:**VIP status given to Customer ID: 2

VIP status given to Customer ID: 4

**Scenario 3:** Reminders for Loans Due in 30 Days

BEGIN

FOR i IN (SELECT \* FROM Loans) LOOP

IF i.EndDate BETWEEN SYSDATE AND (SYSDATE + 30) THEN

DECLARE

v\_name VARCHAR2(100);

BEGIN

SELECT Name INTO v\_name FROM Customers WHERE CustomerID = i.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Reminder for ' || v\_name || ': Loan ID ' || i.LoanID || ' due on ' || TO\_CHAR(i.EndDate, 'DD-MON-YYYY'));

END;

END IF;

END LOOP;

END;

/

**Output:**

Reminder for Young Rich: Loan ID 104 due on [DD-MON-YYYY]

Reminder for Middle Joe: Loan ID 105 due on [DD-MON-YYYY]

**Exercise 3: Stored Procedures**

**Scenario 1:** Process Monthly Interest for Savings Accounts

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

FOR acc IN (SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'Savings') LOOP

UPDATE Accounts

SET Balance = Balance + (acc.Balance \* 0.01),

LastModified = SYSDATE

WHERE AccountID = acc.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Interest applied to Account ID: ' || acc.AccountID);

END LOOP;

END;

/

**Output:**

Interest applied to Account ID: 1

Interest applied to Account ID: 2

Interest applied to Account ID: 4

**Scenario 2: Add Bonus to Employees in a Department**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percent IN NUMBER

) AS

BEGIN

FOR emp IN (SELECT EmployeeID, Salary FROM Employees WHERE Department = p\_department) LOOP

UPDATE Employees

SET Salary = Salary + (emp.Salary \* (p\_bonus\_percent / 100))

WHERE EmployeeID = emp.EmployeeID;

DBMS\_OUTPUT.PUT\_LINE('Bonus applied to Employee ID: ' || emp.EmployeeID);

END LOOP;

END;

/

**Output:**

Bonus applied to Employee ID: 2

Bonus applied to Employee ID: 3

Bonus applied to Employee ID: 5

**Scenario 3: Transfer Funds Between Accounts**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) AS

v\_balance NUMBER;

BEGIN

-- Get balance of source account

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_from\_account FOR UPDATE;

IF v\_balance < p\_amount THEN

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: insufficient balance in Account ID: ' || p\_from\_account);

RETURN;

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_from\_account;

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_to\_account;

DBMS\_OUTPUT.PUT\_LINE('Transferred ' || p\_amount || ' from Account ID ' || p\_from\_account || ' to Account ID ' || p\_to\_account);

END;

/

**Output:**Transferred 500 from Account ID 1 to Account ID 2

**Junit**

**Exercise 1:** Setting up Junit

Using Eclipse IDE to test and importing the Junit library instead of dependencies.

**Exercise 2:** Writing basic Junit Tests

**Arthematic.java**

package example;

public class Arthematic {

public int addition(int a, int b) {

return a + b;

}

}

**ArthematicTest.java**

package example;

import static org.junit.Assert.\*;

import org.junit.Test;

public class ArthematicTest {

@Test

public void test() {

Arthematic ar = new Arthematic();

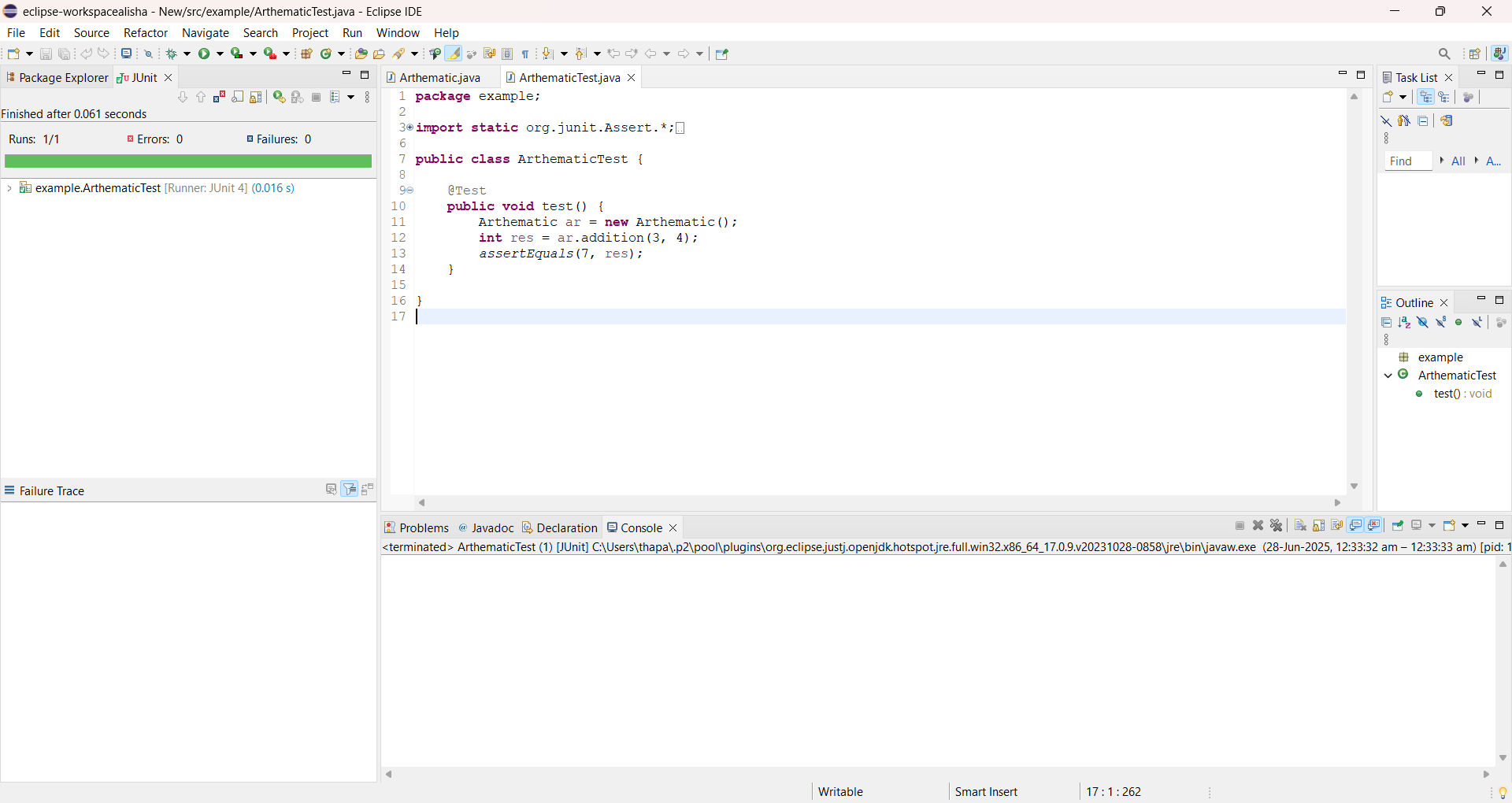
int res = ar.addition(3, 4);

assertEquals(7, res);

}

}

**Output:**

****

**Exercise 3:** Assertions in Junit

**Solution Code:**

package example;

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);

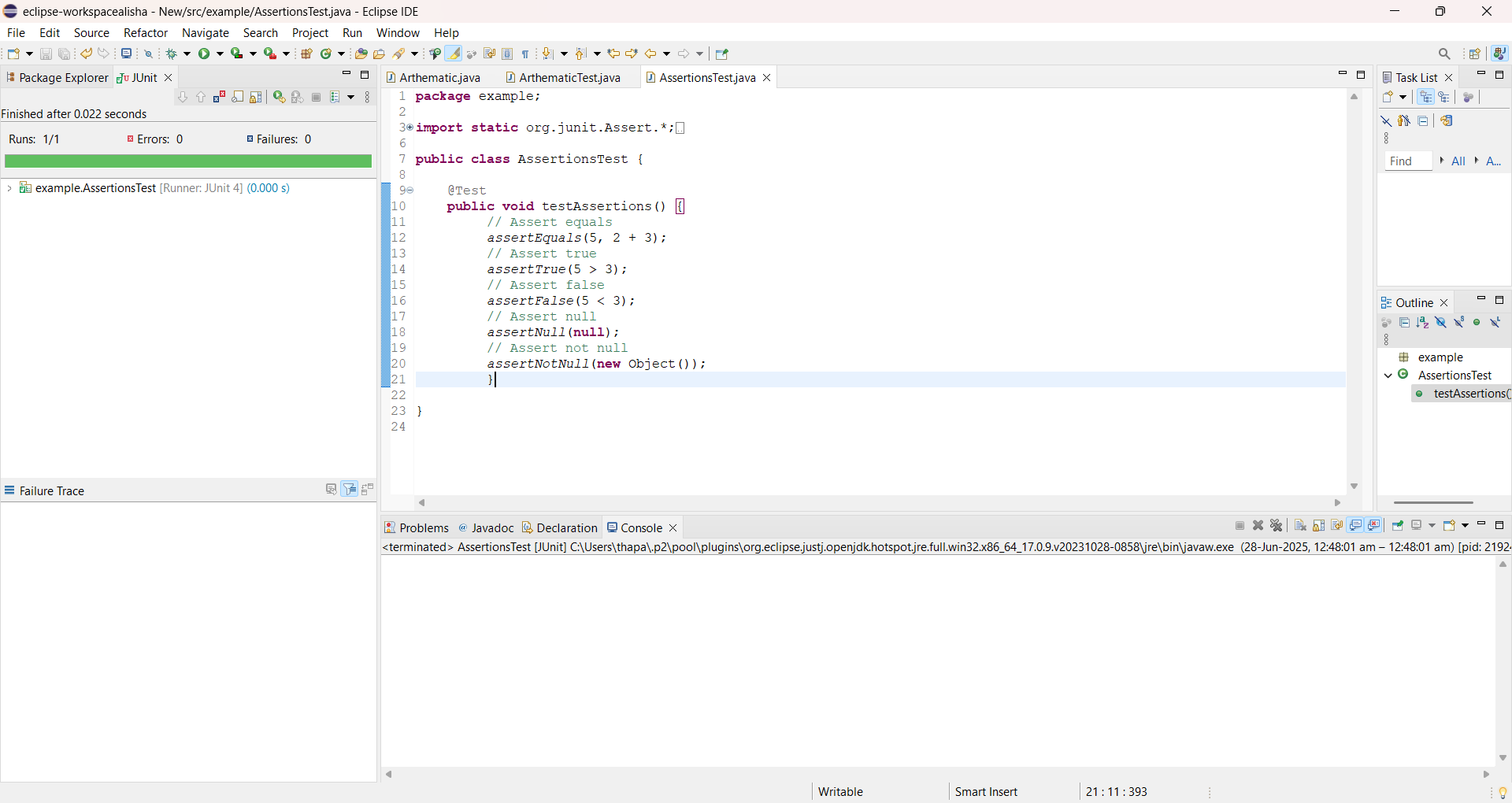
assertNull(null);

assertNotNull(new Object());

}

}

**Output:**

****

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

**AaaPatterns.java**

package patterns;

import org.junit.After;

import org.junit.AfterClass;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.Test;

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

public class AaaPattern {

private int num1;

private int num2;

@BeforeClass

public static void setupBeforeAll() {

System.*out*.println("Before all tests");

}

@AfterClass

public static void cleanupAfterAll() {

System.*out*.println("After all tests");

}

@Before

public void setupBeforeEach() {

System.*out*.println("Before each test");

num1 = 2;

num2 = 3;

}

@After

public void cleanupAfterEach() {

System.*out*.println("After each test");

} @Test

public void testAddTwoNumbers() {

System.*out*.println("Test: Adding two numbers");

int result = num1 + num2;

*assertEquals*("The sum should be 5", 5, result);

}

@Test

public void testMultiplyTwoNumbers() {

System.*out*.println("Test: Multiplying two numbers");

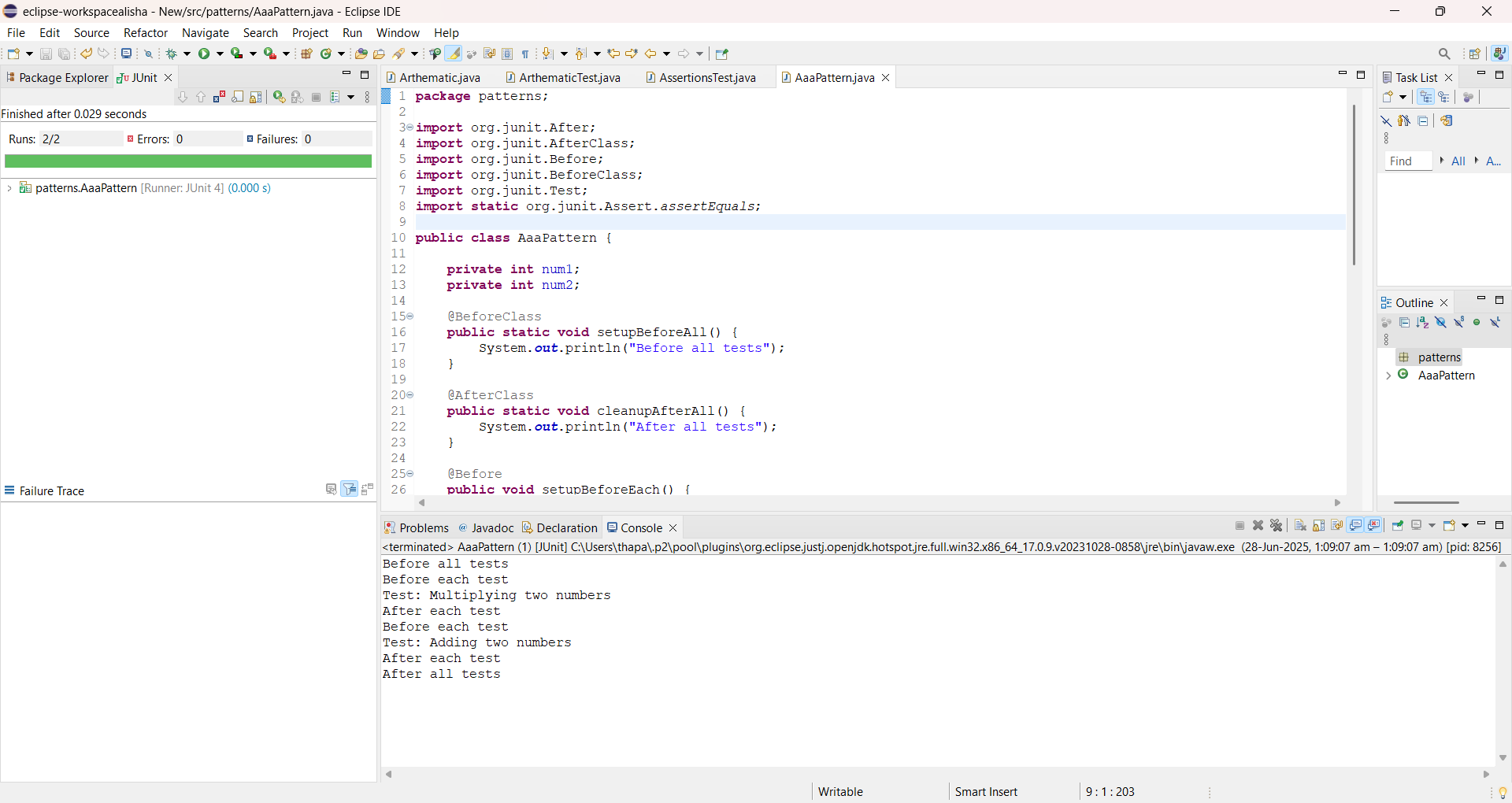
int result = num1 \* num2;

*assertEquals*("The product should be 6", 6, result);

}

}

**Output:**

****

**Mockito**

**Exercise 1: Mocking and Stubbing**

ExternalApi.java

package com.example.mockito\_test;

public interface ExternalApi {

String getData();

}

MyService.java

package com.example.mockito\_test;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

MockStub.java

package com.example.mockito\_test;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class MockStub {

@Test

public void testExternalApi() {

ExternalApi mockApi = mock(ExternalApi.class);

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

MyService service = new MyService(mockApi);

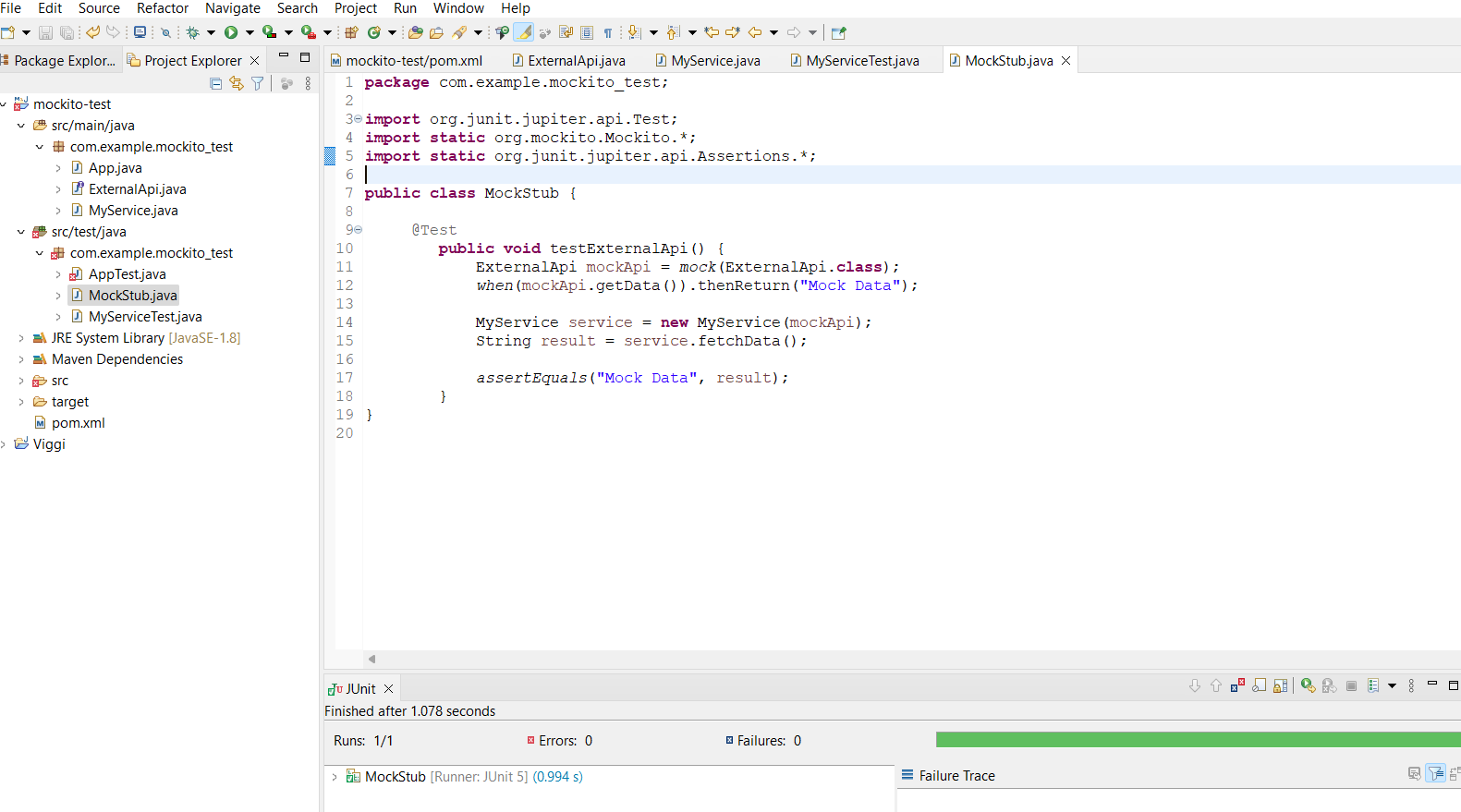
String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**Output:**



**Exercise 2:** **Verifying Interactions**

ExternalApi.java

package com.example.mockito\_test;

public interface ExternalApi {

String getData();

}

MyService.java

package com.example.mockito\_test;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

MyServiceTest.java

package com.example.mockito\_test;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

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

MyService service = new MyService(mockApi);

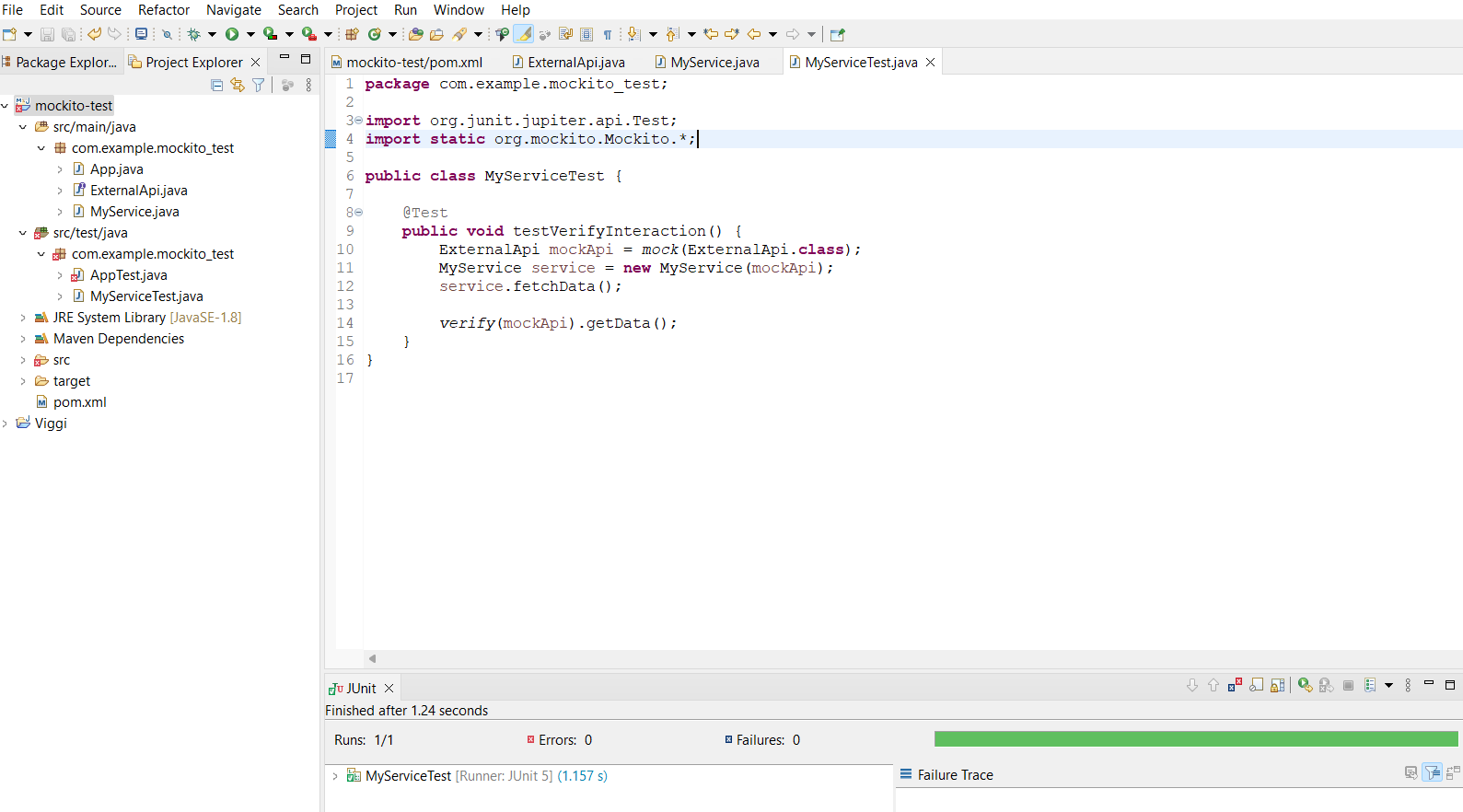
service.fetchData();

*verify*(mockApi).getData();

}

}

**Output:**



**SLF4J logging Framework**

**Exercise1: Logging Error Messages and Warning Levels Task**

**LoggingExample.java**

package com.example.mockito\_test;

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:**

