**EXERCISE 1: CONTROL STRUCTURES**

**CODE**

SET SERVEROUTPUT ON;

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE loans';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE customers';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE customers (

cust\_id NUMBER PRIMARY KEY,

age NUMBER,

balance NUMBER,

vip\_flag VARCHAR2(5)

);

CREATE TABLE loans (

loan\_id NUMBER PRIMARY KEY,

cust\_id NUMBER,

int\_rate NUMBER,

due\_on DATE,

FOREIGN KEY (cust\_id) REFERENCES customers(cust\_id)

);

INSERT INTO customers VALUES (1, 65, 12000, 'FALSE');

INSERT INTO customers VALUES (2, 45, 8000, 'FALSE');

INSERT INTO customers VALUES (3, 70, 15000, 'FALSE');

INSERT INTO loans VALUES (101, 1, 10, TO\_DATE('04-JUL-2025','DD-MON-YYYY'));

INSERT INTO loans VALUES (102, 2, 9, TO\_DATE('01-SEP-2025','DD-MON-YYYY'));

INSERT INTO loans VALUES (103, 3, 8, TO\_DATE('29-JUN-2025','DD-MON-YYYY'));

COMMIT;

BEGIN

FOR loan\_rec IN (

SELECT l.loan\_id, l.cust\_id, l.int\_rate

FROM loans l

JOIN customers c ON l.cust\_id = c.cust\_id

WHERE c.age > 60

)

LOOP

UPDATE loans

SET int\_rate = int\_rate - 1

WHERE loan\_id = loan\_rec.loan\_id;

DBMS\_OUTPUT.PUT\_LINE(

'Scenario 1: 1% interest discount applied on Loan ' || loan\_rec.loan\_id ||

' (Customer ID ' || loan\_rec.cust\_id || ')'

);

END LOOP;

FOR cust\_rec IN (

SELECT cust\_id, balance FROM customers

WHERE balance > 10000

)

LOOP

UPDATE customers

SET vip\_flag = 'TRUE'

WHERE cust\_id = cust\_rec.cust\_id;

DBMS\_OUTPUT.PUT\_LINE(

' Scenario 2: VIP status set for Customer ' || cust\_rec.cust\_id ||

' (Balance: $' || cust\_rec.balance || ')'

);

END LOOP;

FOR due\_rec IN (

SELECT loan\_id, cust\_id, due\_on

FROM loans

WHERE due\_on BETWEEN SYSDATE AND SYSDATE + 30

)

LOOP

DBMS\_OUTPUT.PUT\_LINE(

'Scenario 3: Reminder - Loan ' || due\_rec.loan\_id ||

' for Customer ' || due\_rec.cust\_id ||

' is due on ' || TO\_CHAR(due\_rec.due\_on, 'DD-MON-YYYY')

);

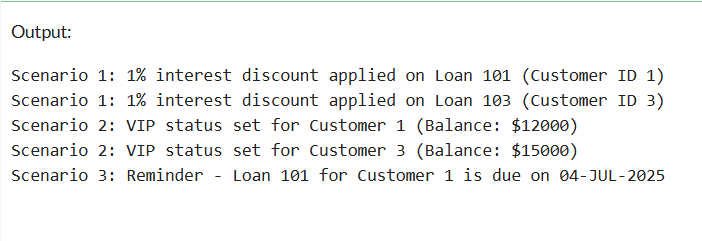
END LOOP;

COMMIT;

END;

/

**OUTPUT**

****

**EXERCISE 3:STORED PROCEEDURES**

**CODE**

SET SERVEROUTPUT ON;

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(20), -- Increased size here

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (2, 2, 'Checking', 1500, SYSDATE);

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

COMMIT;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

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('Applied interest to Account ID: ' || acc.AccountID);

END LOOP;

COMMIT;

END;

/

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_department IN VARCHAR2,

p\_bonus\_pct IN NUMBER

) IS

v\_count NUMBER;

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* (p\_bonus\_pct / 100))

WHERE Department = p\_department;

v\_count := SQL%ROWCOUNT;

DBMS\_OUTPUT.PUT\_LINE('Bonus applied to ' || v\_count || ' employee(s) in department: ' || p\_department);

COMMIT;

END;

/

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) IS

v\_balance NUMBER;

v\_new\_id NUMBER;

BEGIN

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

IF v\_balance < p\_amount THEN

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

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;

SELECT NVL(MAX(TransactionID), 0) + 1 INTO v\_new\_id FROM Transactions;

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (v\_new\_id, p\_from\_account, SYSDATE, p\_amount, 'Transfer-Out');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (v\_new\_id + 1, p\_to\_account, SYSDATE, p\_amount, 'Transfer-In');

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

COMMIT;

END;

/

BEGIN

ProcessMonthlyInterest;

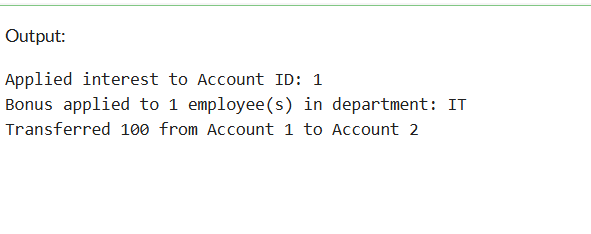
UpdateEmployeeBonus('IT', 10);

TransferFunds(1, 2, 100);

END;

/

**OUTPUT**



**Exercise 1: Setting Up Junit**

**CODE**

**PasswordValidator.java**

package com.example;

public class PasswordValidator {

public boolean isValid(String password) {

if (password == null || password.length() < 8) return false;

boolean hasUpper = false, hasLower = false, hasDigit = false;

for (char ch : password.toCharArray()) {

if (Character.isUpperCase(ch)) hasUpper = true;

else if (Character.isLowerCase(ch)) hasLower = true;

else if (Character.isDigit(ch)) hasDigit = true;

}

return hasUpper && hasLower && hasDigit;

}

}

**PasswordValidatorTest.java**

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class PasswordValidatorTest {

PasswordValidator validator = new PasswordValidator();

@Test

public void testValidPassword() {

String password = "Strong123";

boolean result = validator.isValid(password);

System.out.println("Test Valid Password: " + result);

assertTrue(result);

}

@Test

public void testShortPassword() {

String password = "S1mple";

boolean result = validator.isValid(password);

System.out.println("Test Short Password: " + result);

assertFalse(result);

}

@Test

public void testNoUppercase() {

String password = "strong123";

boolean result = validator.isValid(password);

System.out.println("Test No Uppercase: " + result);

assertFalse(result);

}

@Test

public void testNoLowercase() {

String password = "STRONG123";

boolean result = validator.isValid(password);

System.out.println("Test No Lowercase: " + result);

assertFalse(result);

}

@Test

public void testNoDigits() {

String password = "StrongPwd";

boolean result = validator.isValid(password);

System.out.println("Test No Digits: " + result);

assertFalse(result);

}

@Test

public void testNullPassword() {

String password = null;

boolean result = validator.isValid(password);

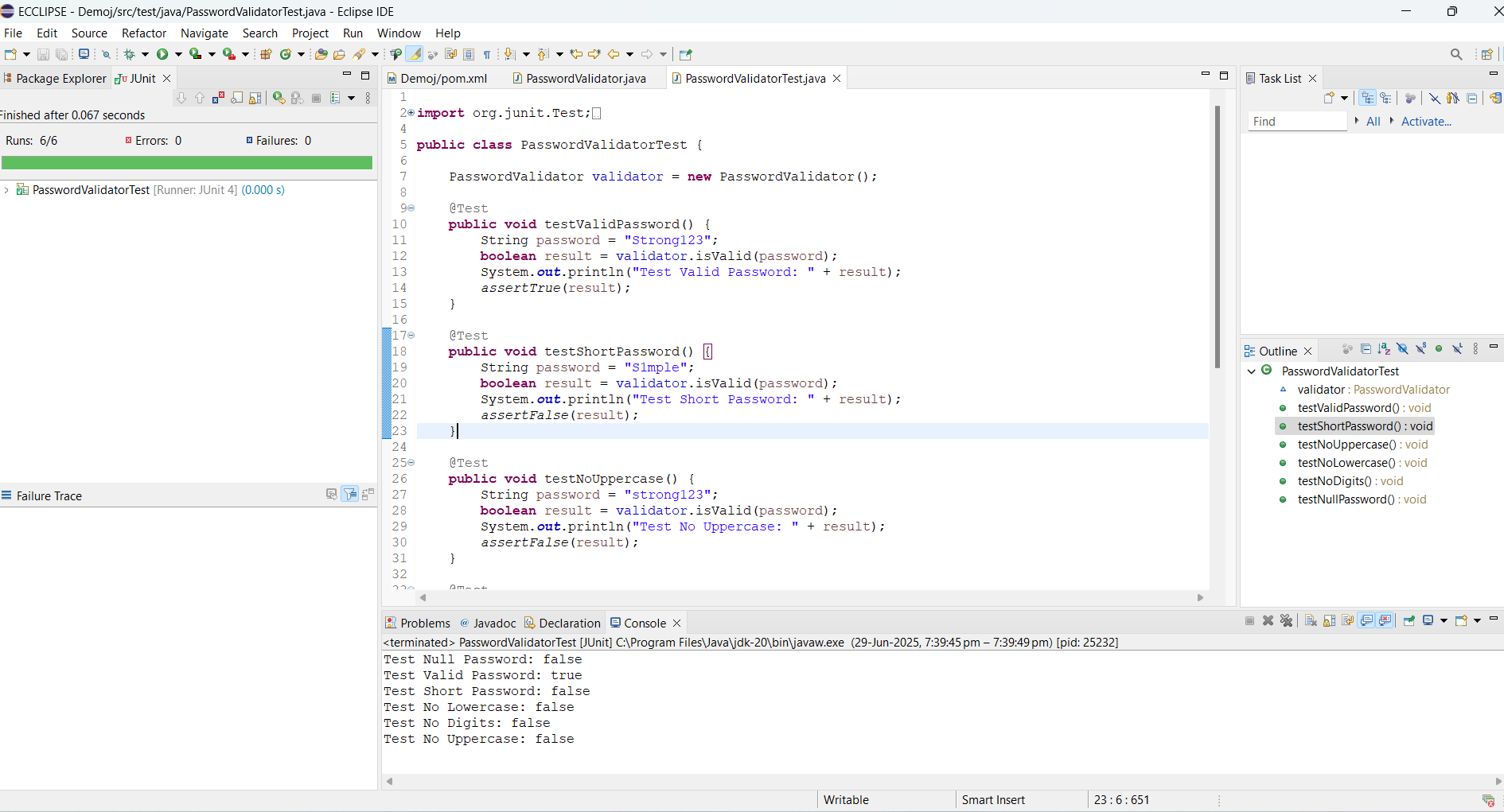
System.out.println("Test Null Password: " + result);

assertFalse(result);

}

}

**OUTPUT**

****

**Exercise 3: Assertions in JUnit**

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

@Test

public void testAssertions() {

*assertEquals*(5, 2 + 3);

System.*out*.println("assertEquals passed");

*assertTrue*(5 > 3);

System.*out*.println("assertTrue passed");

*assertFalse*(5 < 3);

System.*out*.println("assertFalse passed");

*assertNull*(null);

System.*out*.println("assertNull passed");

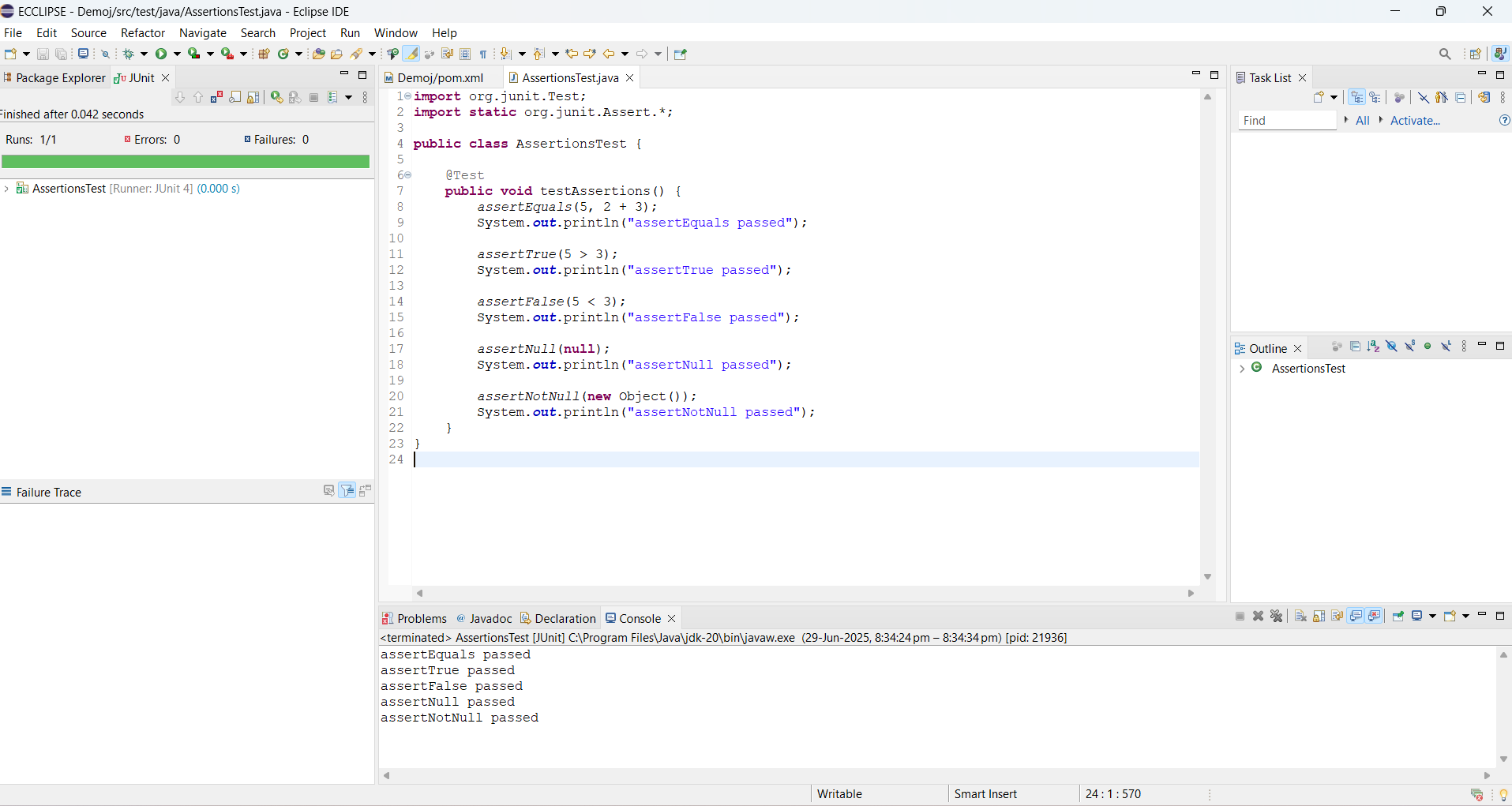
*assertNotNull*(new Object());

System.*out*.println("assertNotNull passed");

}

}

**OUTPUT**



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

**CODE**

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class PasswordValidatorTest {

private PasswordValidator validator;

// Setup method (Arrange)

@Before

public void setUp() {

validator = new PasswordValidator();

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

}

// Teardown method

@After

public void tearDown() {

validator = null;

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

}

// Test case: Valid password

@Test

public void testValidPassword() {

// Arrange done in setUp

// Act

boolean result = validator.isValid("Abcd1234");

// Assert

assertTrue(result);

System.out.println("testValidPassword passed.");

}

// Test case: Password too short

@Test

public void testShortPassword() {

boolean result = validator.isValid("Ab1");

assertFalse(result);

System.out.println("testShortPassword passed.");

}

// Test case: No digits

@Test

public void testPasswordWithoutDigits() {

boolean result = validator.isValid("Password");

assertFalse(result);

System.out.println("testPasswordWithoutDigits passed.");

}

// Test case: No letters

@Test

public void testPasswordWithoutLetters() {

boolean result = validator.isValid("12345678");

assertFalse(result);

System.out.println("testPasswordWithoutLetters passed.");

}

// Test case: Null password

@Test

public void testNullPassword() {

boolean result = validator.isValid(null);

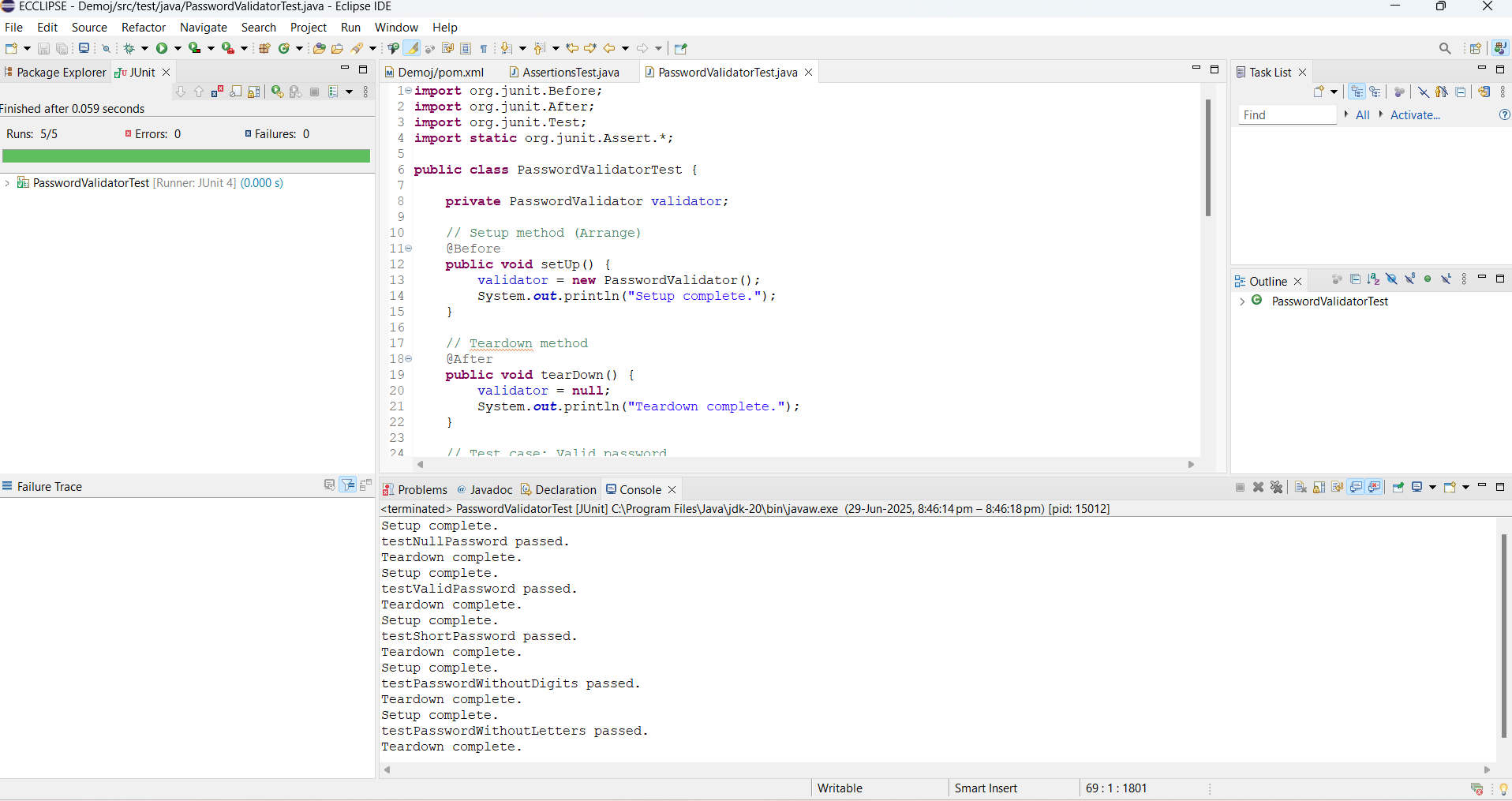
assertFalse(result);

System.out.println("testNullPassword passed.");

}

}

**OUTPUT**

****

**Mockito Hands-On Exercises**

**Exercise 1: Mocking and Stubbing**

**CODE**

**ExternalApi.java**

public interface ExternalApi {

String getData();

}

**Myservice.java**

public class MyService {

private ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData();

}

}

**MyServiceTest.java**

import org.junit.Test;

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

import static org.mockito.Mockito.\*;

public class MyServiceTest {

@Test

public void testFetchData() {

// Arrange (Mock setup)

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

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

MyService service = new MyService(mockApi);

String result = service.fetchData();

// Assert

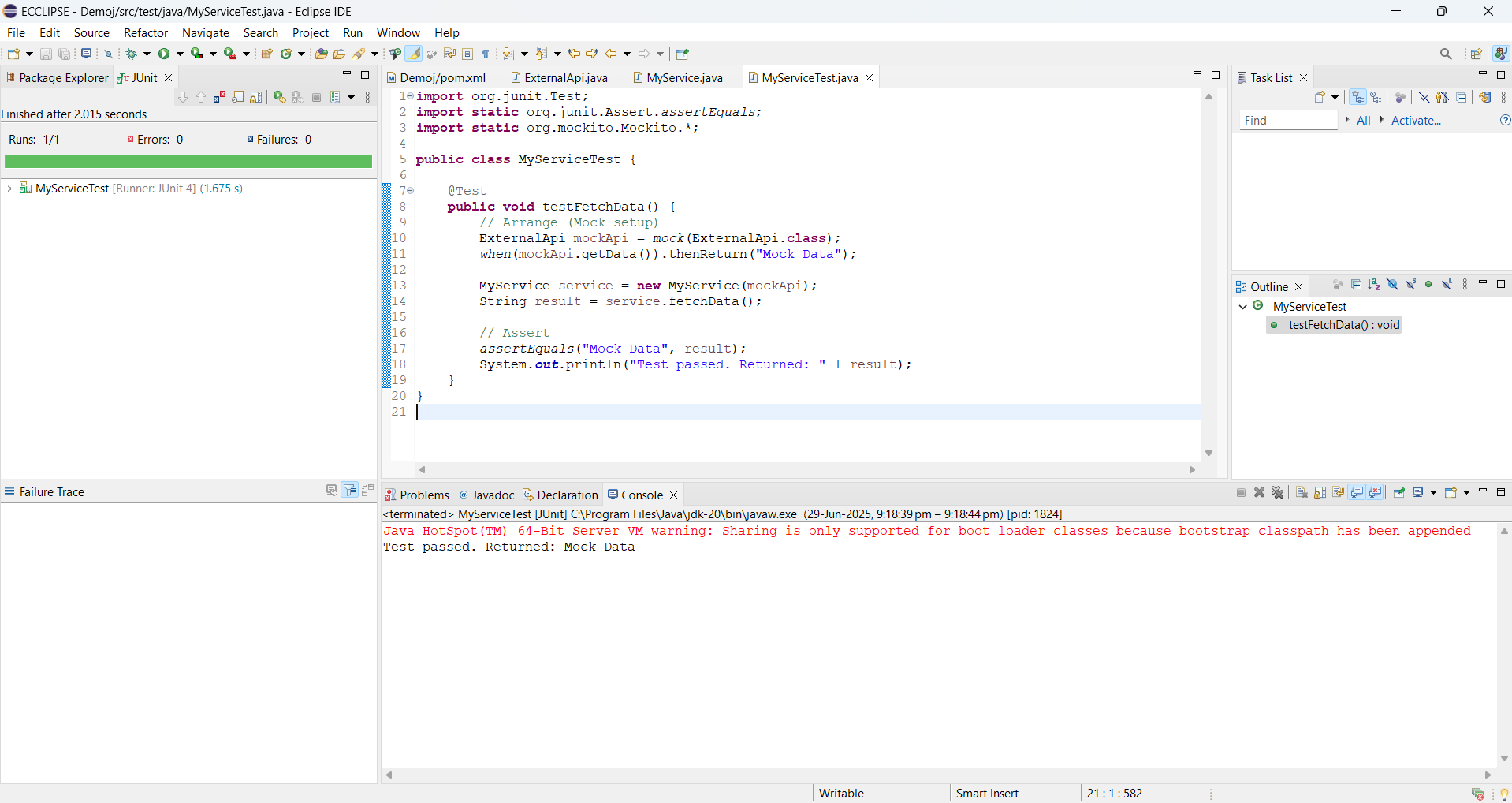
*assertEquals*("Mock Data", result);

System.*out*.println("Test passed. Returned: " + result);

}

}

**OUTPUT**



**Exercise 2: Verifying Interactions**

**CODE**

**ExternalApi.java**

public interface ExternalApi {

String getData();

}

**Myservice.java**

public class MyService {

private ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData();

}

}

**MyServiceTest.java**

import org.junit.Test;

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

import static org.mockito.Mockito.\*;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

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

*when*(mockApi.getData()).thenReturn("Mocked Interaction Data");

MyService service = new MyService(mockApi);

String result = service.fetchData();

*assertEquals*("Mocked Interaction Data", result);

System.*out*.println("fetchData() returned: " + result);

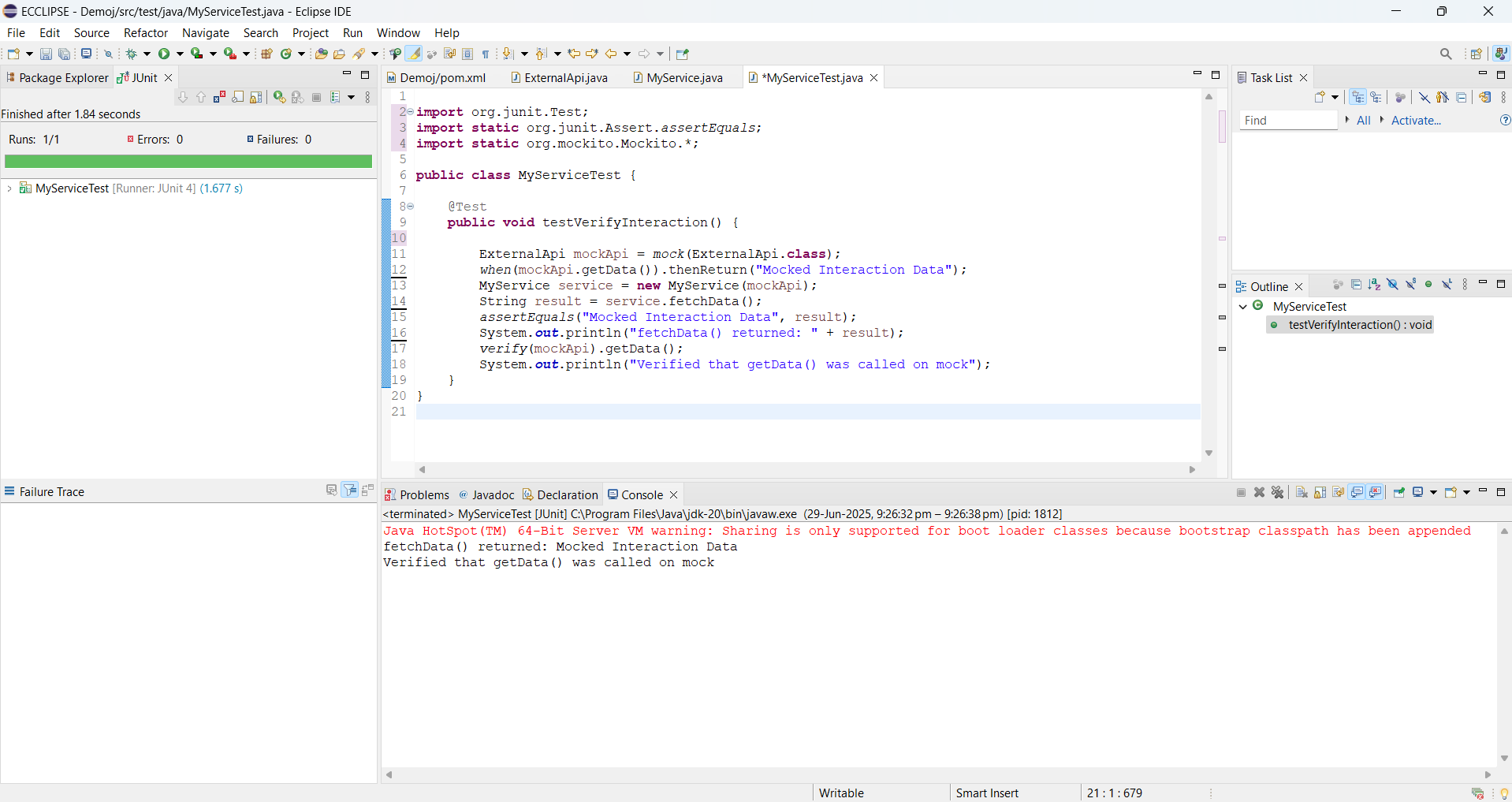
*verify*(mockApi).getData();

System.*out*.println("Verified that getData() was called on mock");

}

}

**OUTPUT**

****

**Logging using SLF4J**

**Exercise 1: Logging Error Messages and Warning Levels**

**CODE**

package com.example.LoggingDemonew;

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");

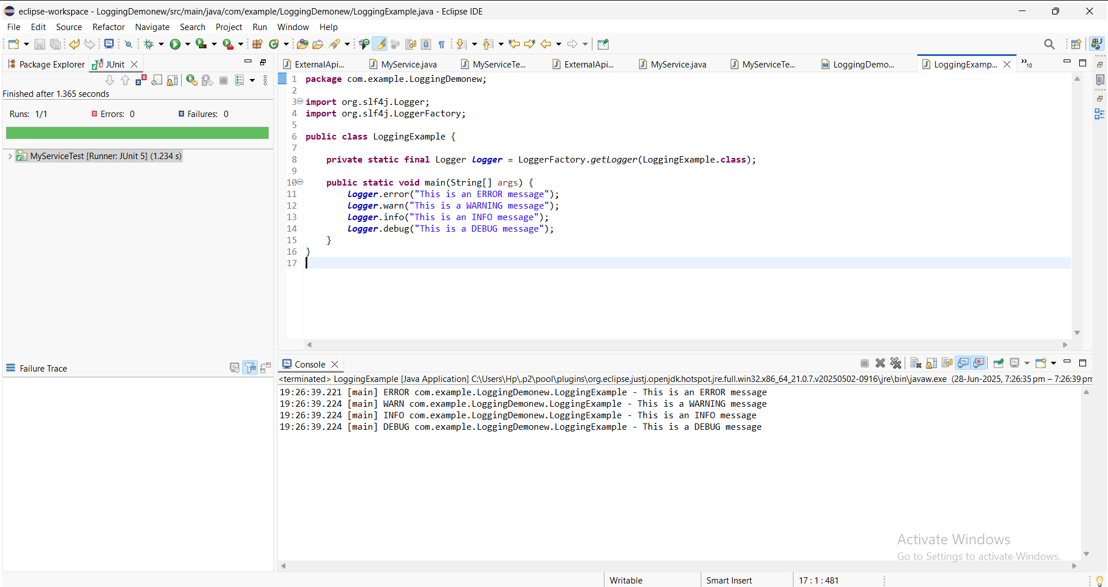
logger.info("This is an INFO message");

logger.debug("This is a DEBUG message");

}

}

**OUTPUT:**

****