WEEK 2  
Topic 1: PL/SQL Programming

Exercise 1: Control Structures

Code :  
  
DECLARE

marks NUMBER := 87;

BEGIN

IF marks >= 90 THEN

DBMS\_OUTPUT.PUT\_LINE('Grade: A');

ELSIF marks >= 80 THEN

DBMS\_OUTPUT.PUT\_LINE('Grade: B');

ELSIF marks >= 70 THEN

DBMS\_OUTPUT.PUT\_LINE('Grade: C');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Grade: D or below');

END IF;

-- Alternative using CASE

CASE

WHEN marks >= 90 THEN DBMS\_OUTPUT.PUT\_LINE('Excellent');

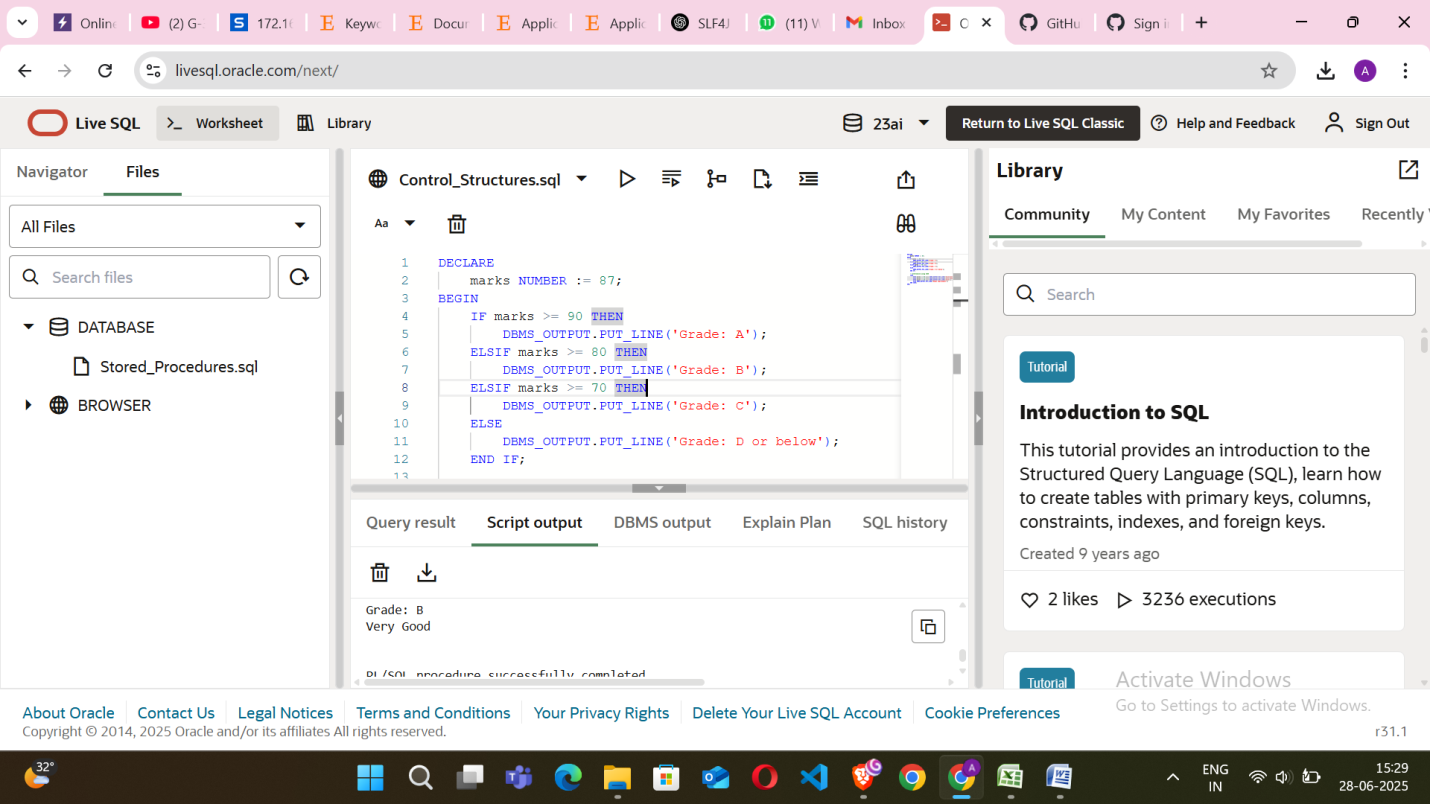
WHEN marks >= 75 THEN DBMS\_OUTPUT.PUT\_LINE('Very Good');

WHEN marks >= 60 THEN DBMS\_OUTPUT.PUT\_LINE('Good');

ELSE DBMS\_OUTPUT.PUT\_LINE('Needs Improvement');

END CASE;

END;

O/P :   
  


Exercise 2: Stored Procedure  
Code :  
  
CREATE OR REPLACE PROCEDURE get\_employee\_details (

p\_empid IN emp.emp\_id%TYPE,

p\_salary OUT emp.salary%TYPE,

p\_job OUT emp.job\_title%TYPE

) AS

BEGIN

SELECT salary, job\_title

INTO p\_salary, p\_job

FROM emp

WHERE emp\_id = p\_empid;

END;  
  
- - CALLING PROCEDURE

DECLARE

v\_salary emp.salary%TYPE;

v\_job emp.job\_title%TYPE;

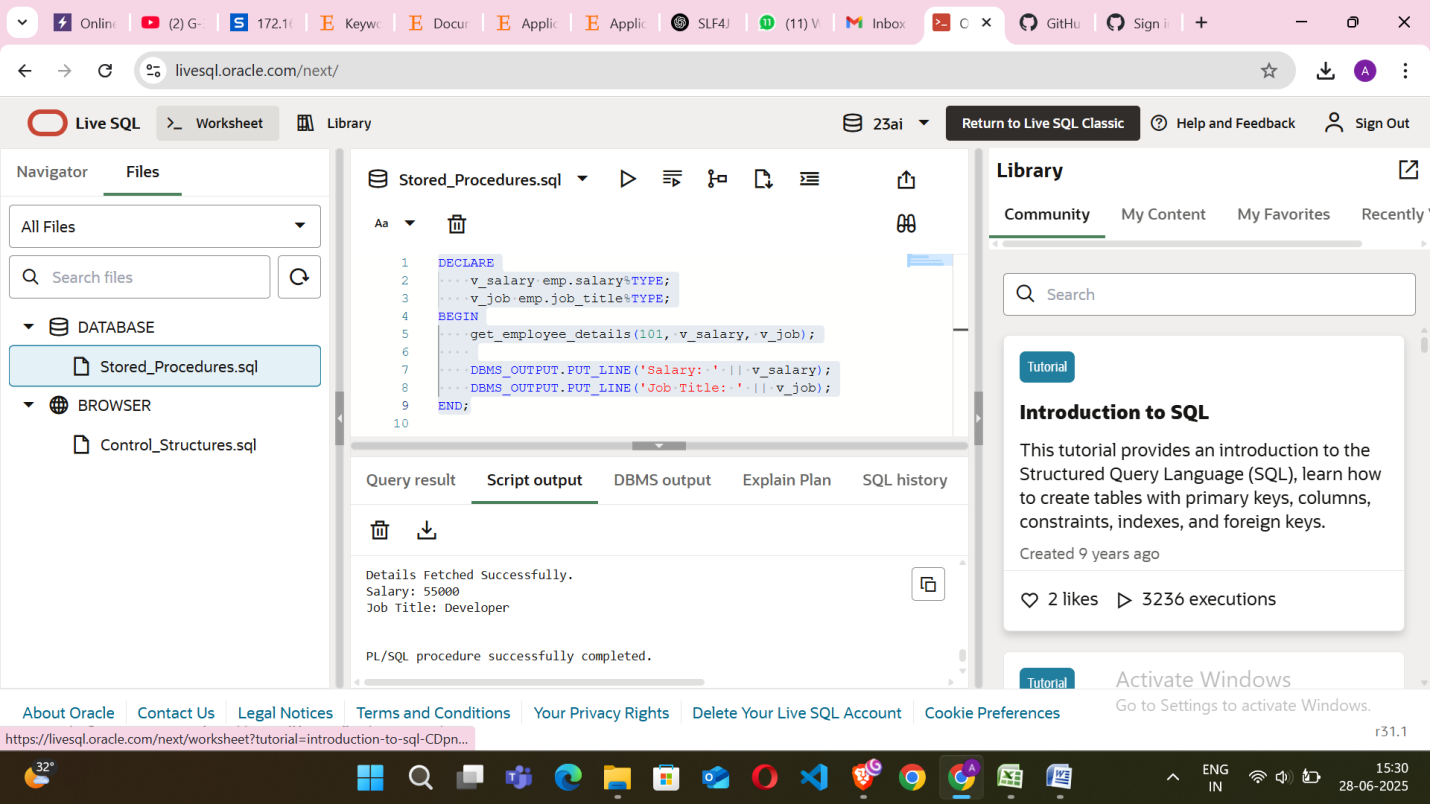
BEGIN

get\_employee\_details(101, v\_salary, v\_job);

DBMS\_OUTPUT.PUT\_LINE('Salary: ' || v\_salary);

DBMS\_OUTPUT.PUT\_LINE('Job Title: ' || v\_job);

END;

O/P:  
  


Topic 2: TDD using JUnit5 & Mockito,SLF4J Logging Framework

Exercise 1: Setting Up JUnit  
  
Code :  
  
// Calculator.java  
  
package Exercise1;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public static void main(String[] args) {

Calculator c = new Calculator();

System.out.println(c.add(10, 20));

}

}

// CalculatorTest.java  
  
package Exercise1;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

@Test

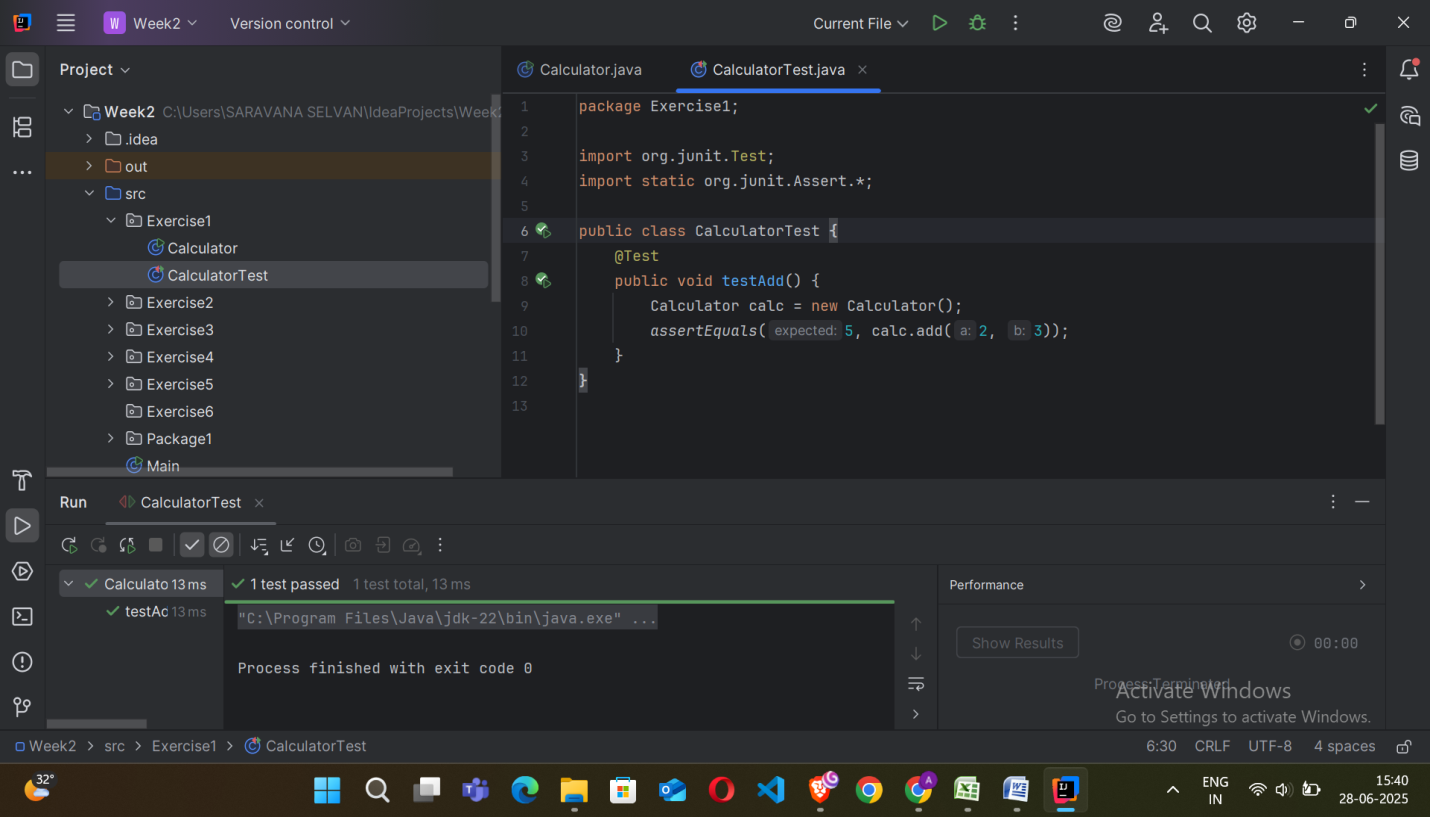
public void testAdd() {

Calculator calc = new Calculator();

assertEquals(5, calc.add(2, 3));

}

}

O/P:  
  
  
Exercise 2: Assertions in JUnit  
  
Code :   
  
//Greeter.java  
  
package Exercise2;

public class Greeter {

public String greet(String name) {

return "Hello, " + name;

}

public static void main(String[] args) {

Greeter greeter = new Greeter();

String result = greeter.greet("Alice");

System.out.println(result);

}

}  
  
// GreeterTest.java  
  
package Exercise2;

import org.junit.jupiter.api.Test;

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

public class GreeterTest {

@Test

void testGreet() {

Greeter g = new Greeter();

String result = g.greet("Alice");

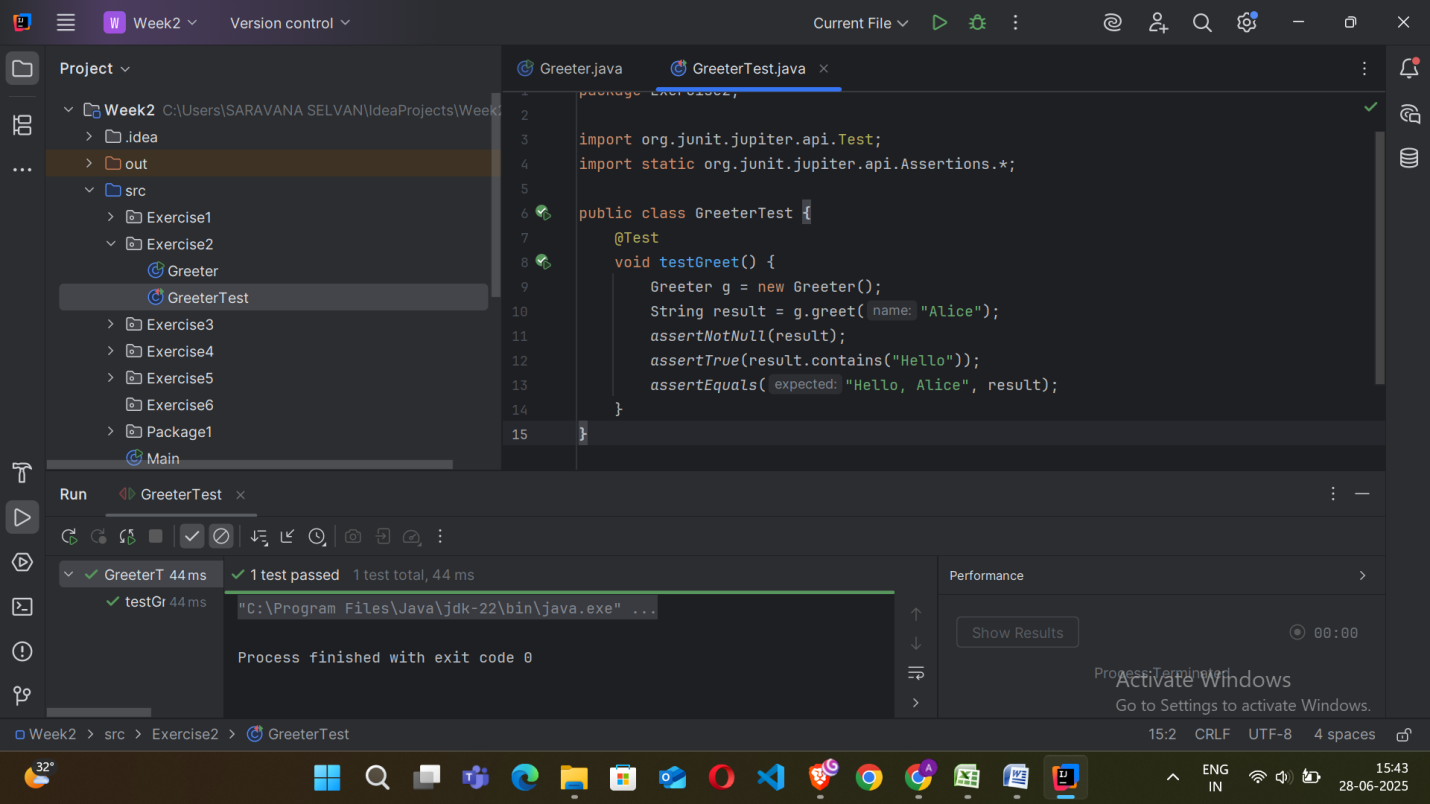
assertNotNull(result);

assertTrue(result.contains("Hello"));

assertEquals("Hello, Alice", result);

}

}

O/P:  
  
  
  
Exercise 3: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in JUnit  
  
Code :   
  
//BankAccount.java  
  
package Exercise3;

public class BankAccount {

private int balance = 0;

public void deposit(int amount) {

balance += amount;

}

public void withdraw(int amount) {

balance -= amount;

}

public int getBalance() {

return balance;

}

public static void main(String[] args) {

BankAccount account = new BankAccount();

account.deposit(100);

account.withdraw(40);

System.out.println("Balance after withdraw: " + account.getBalance());

account.deposit(50);

System.out.println("Balance after deposit: " + account.getBalance());

account = null;

}

}

//BankAccountTest.java

package Exercise3;

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

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

public class BankAccountTest {

BankAccount account;

@BeforeEach

void setUp() {

account = new BankAccount();

account.deposit(100);

}

@AfterEach

void tearDown() {

account = null;

}

@Test

void testWithdraw() {

account.withdraw(40);

assertEquals(60, account.getBalance());

}

@Test

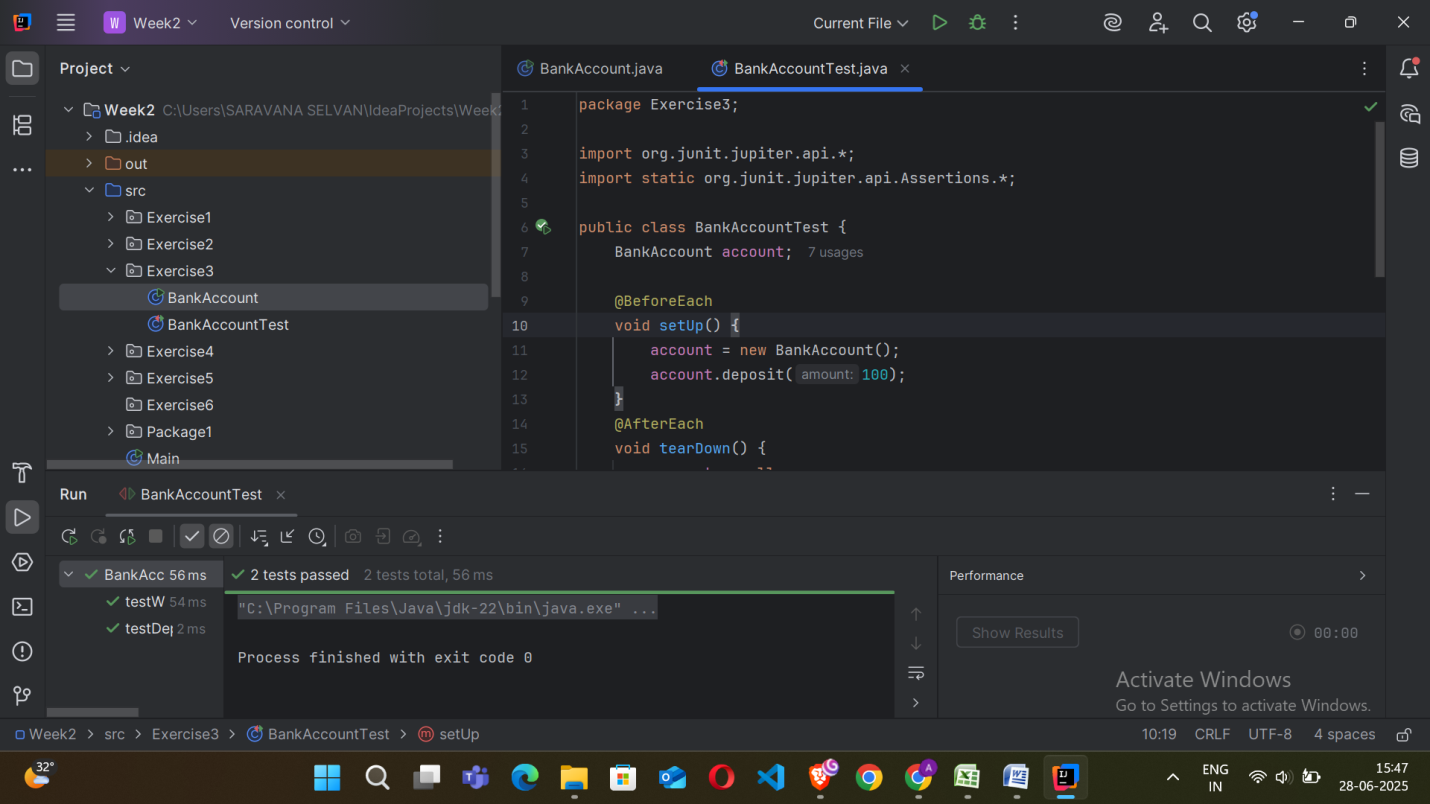
void testDeposit() {

account.deposit(50);

assertEquals(150, account.getBalance());

}

}

O/P:  
  


Exercise 4: Mocking and Stubbing  
  
Code :

//UserService.java  
  
package com.example.ex2;

import org.junit.jupiter.api.Test;

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

import static org.mockito.Mockito.\*;

class UserServiceTest {

@Test

void testRegisterUser\_MockingAndStubbing() {

EmailService emailService = mock(EmailService.class);

when(emailService.sendEmail("john@example.com", "Welcome!")).thenReturn(true);

UserService userService = new UserService(emailService);

boolean result = userService.registerUser("john@example.com");

assertTrue(result);

}

}  
  
//EmailService.java  
  
package com.example.ex2;

public interface EmailService {

boolean sendEmail(String to, String content);

}

//UserServiceTest.java  
  
package com.example.ex2;

import org.junit.jupiter.api.Test;

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

import static org.mockito.Mockito.\*;

class UserServiceTest {

@Test

void testRegisterUser\_MockingAndStubbing() {

EmailService emailService = mock(EmailService.class);

when(emailService.sendEmail("john@example.com", "Welcome!")).thenReturn(true);

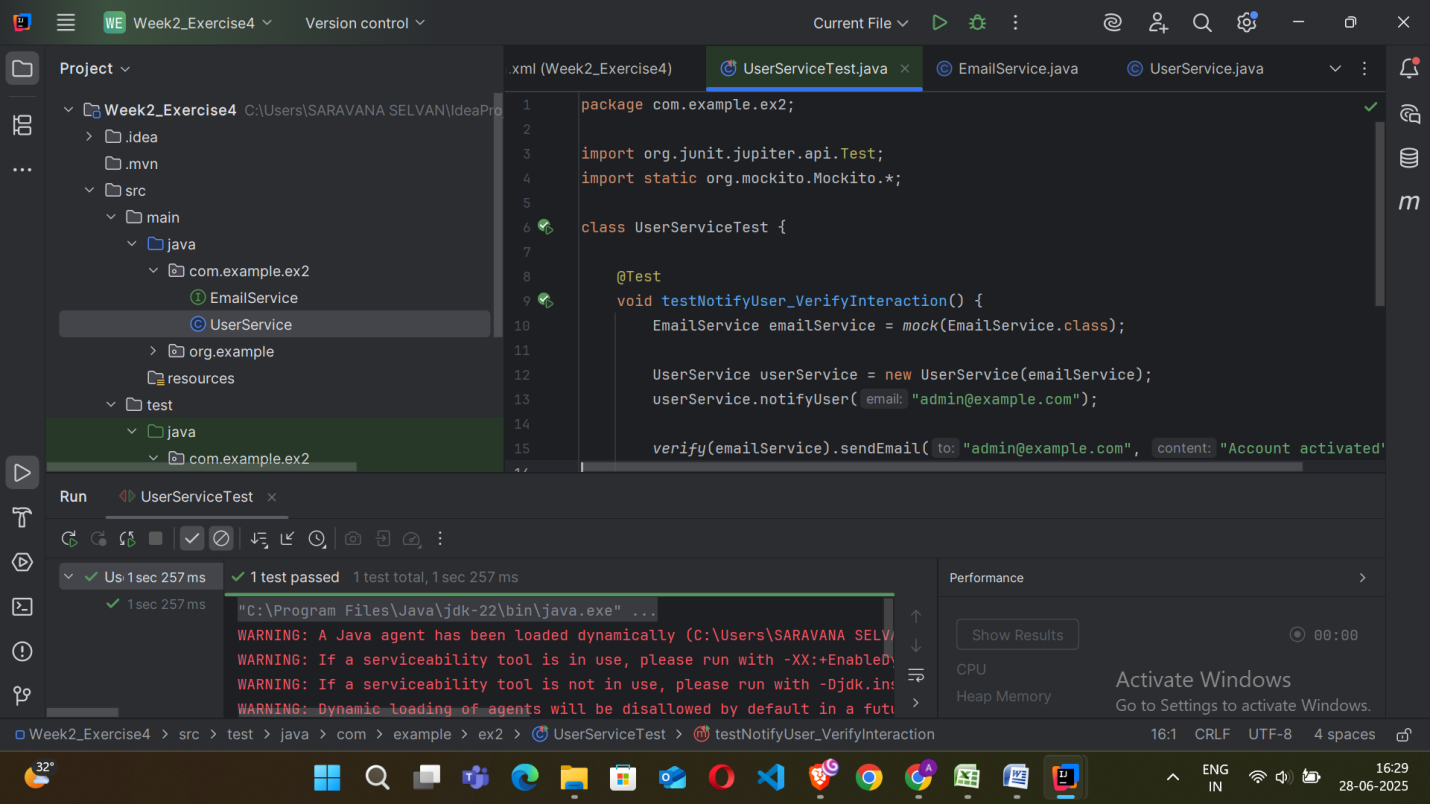
UserService userService = new UserService(emailService);

boolean result = userService.registerUser("john@example.com");

assertTrue(result);

}

}

O/P:  
  


Exercise 5: Verifying Interactions  
  
Code :   
  
//UserService.java  
  
package com.example.ex2;

public class UserService {

private final EmailService emailService;

public UserService(EmailService emailService) {

this.emailService = emailService;

}

public void notifyUser(String email) {

emailService.sendEmail(email, "Account activated");

}

}  
  
//EmailService.java  
  
package com.example.ex2;

public interface EmailService {

boolean sendEmail(String to, String content);

}  
  
//UserServiceTest.java  
  
package com.example.ex2;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

class UserServiceTest {

@Test

void testNotifyUser\_VerifyInteraction() {

EmailService emailService = mock(EmailService.class);

UserService userService = new UserService(emailService);

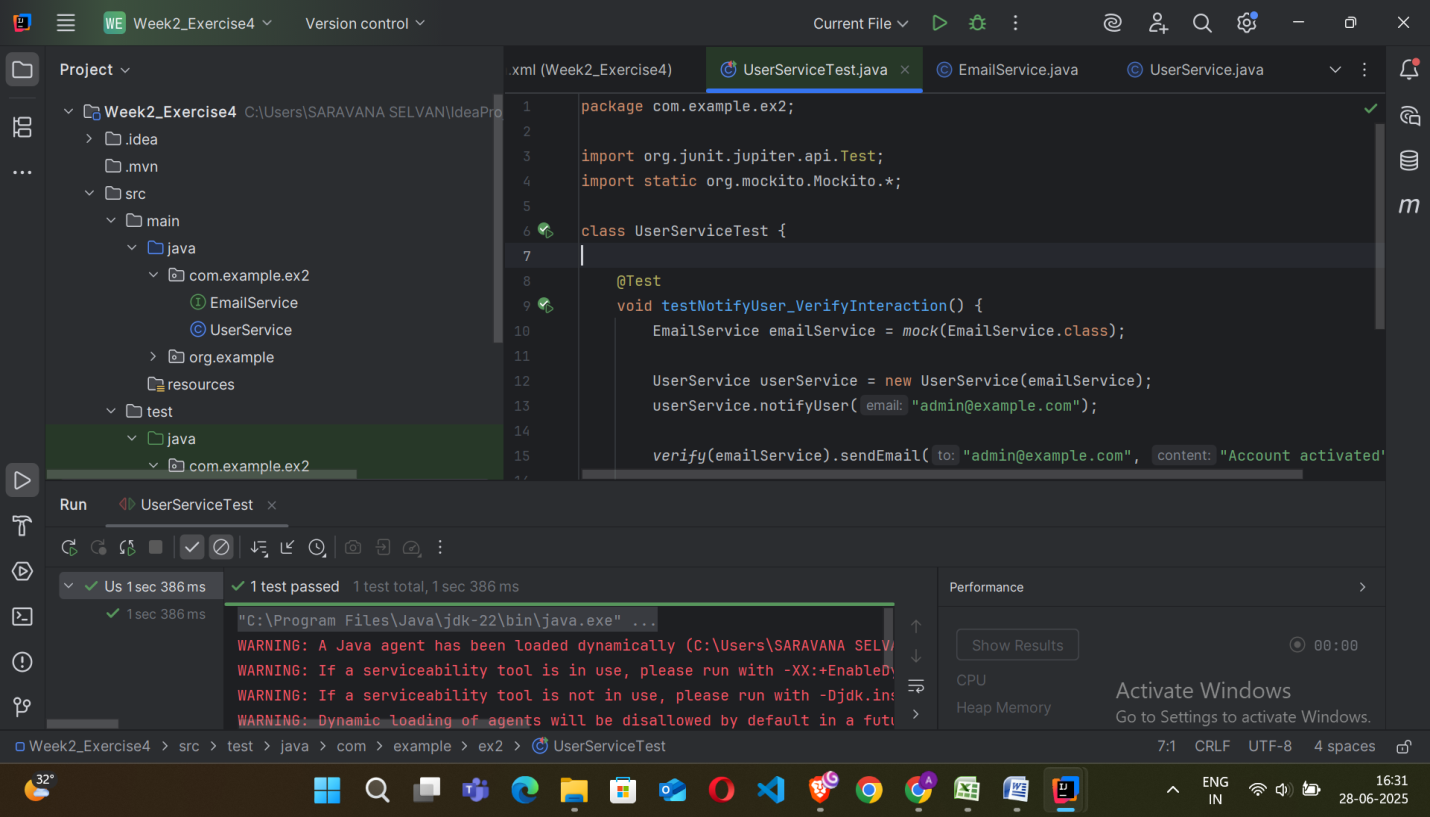
userService.notifyUser("admin@example.com");

verify(emailService).sendEmail("admin@example.com", "Account activated");

verifyNoMoreInteractions(emailService);

}

}

O/P:  
  


Exercise 6: Logging Error Messages and Warning Levels  
  
Code :   
  
import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LoggingDemo {

private static final Logger logger =LoggerFactory.getLogger(LoggingDemo.class);

public static void main(String[] args) {

logger.info("Application started.");

int speed = 120;

if (speed > 100) {

logger.warn("Over-speeding! Current speed: {} km/h", speed);

}

try {

int x = 10 / 0;

} catch (Exception e) {

logger.error("Exception occurred: Division by zero", e);

}

logger.info("Application ended.");

}

}

O/P:  
  
