**PLSQL Exercise 1: Control Structures**

SET SERVEROUTPUT ON;

DECLARE

num NUMBER := 5;

i NUMBER := 1;

BEGIN

-- IF-ELSE control

IF num > 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Positive Number');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Non-positive Number');

END IF;

-- LOOP

DBMS\_OUTPUT.PUT\_LINE('Printing numbers from 1 to ' || num);

WHILE i <= num LOOP

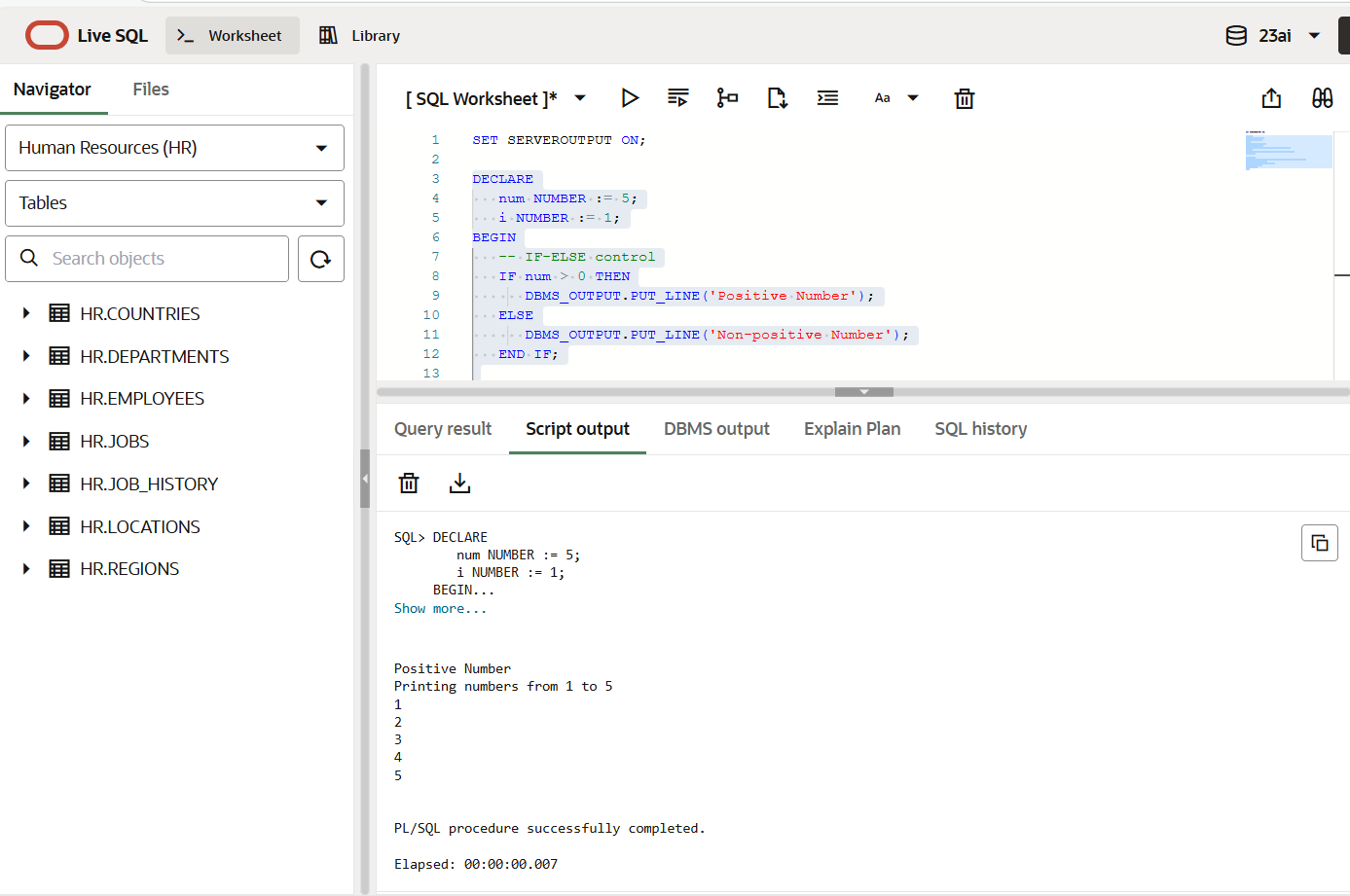
DBMS\_OUTPUT.PUT\_LINE(i);

i := i + 1;

END LOOP;

END;

**Output**



**PLSQL Exercise 3: Stored Procedures**

USE kmp;

DELIMITER //

CREATE PROCEDURE Show\_Square(IN num INT)

BEGIN

DECLARE result INT;

SET result = num \* num;

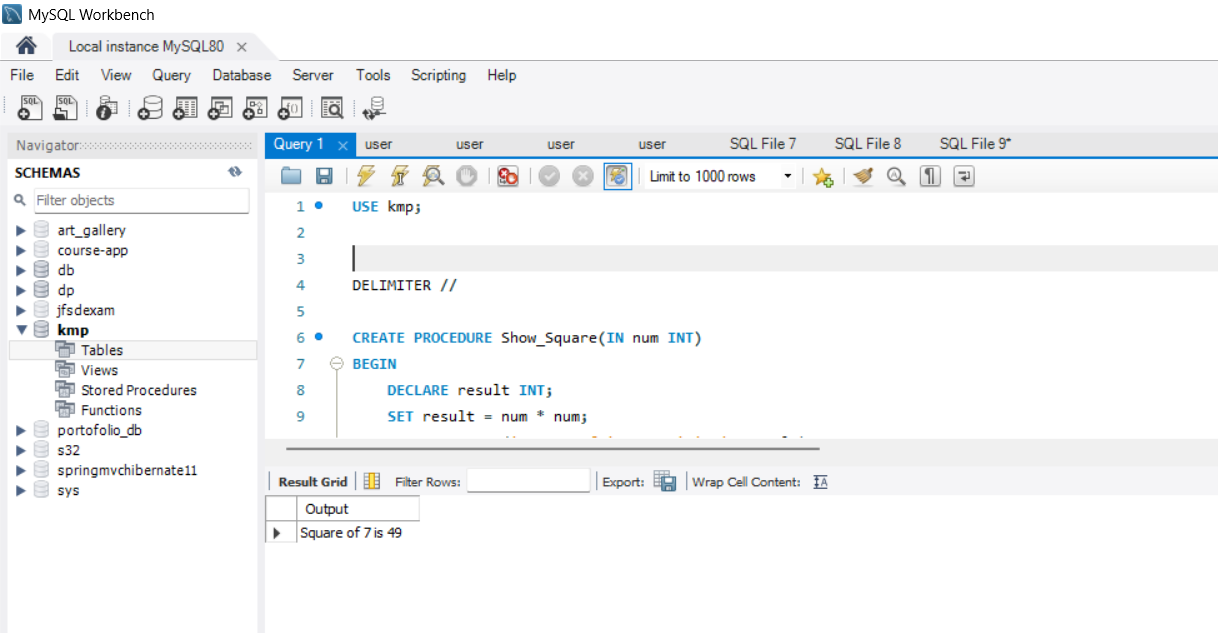
SELECT CONCAT('Square of ', num, ' is ', result) AS Output;

END;

//

DELIMITER ;

CALL Show\_Square(7);



**PART 2: JUnit EXERCISES**

**1>Teardown methods in junit**

package com.example;

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

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

public class CalculatorTest {

@BeforeEach

void setup() {

System.out.println("Setup before each test");

}

@Test

void testAddition() {

assertEquals(5, 2 + 3);

}

@AfterEach

void teardown() {

System.out.println("Teardown after each test");

}

}

package com.example;

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

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

public class CalculatorTest {

@BeforeEach

void setup() {

System.out.println("Setup before each test");

}

@Test

void testAddition() {

assertEquals(5, 2 + 3);

}

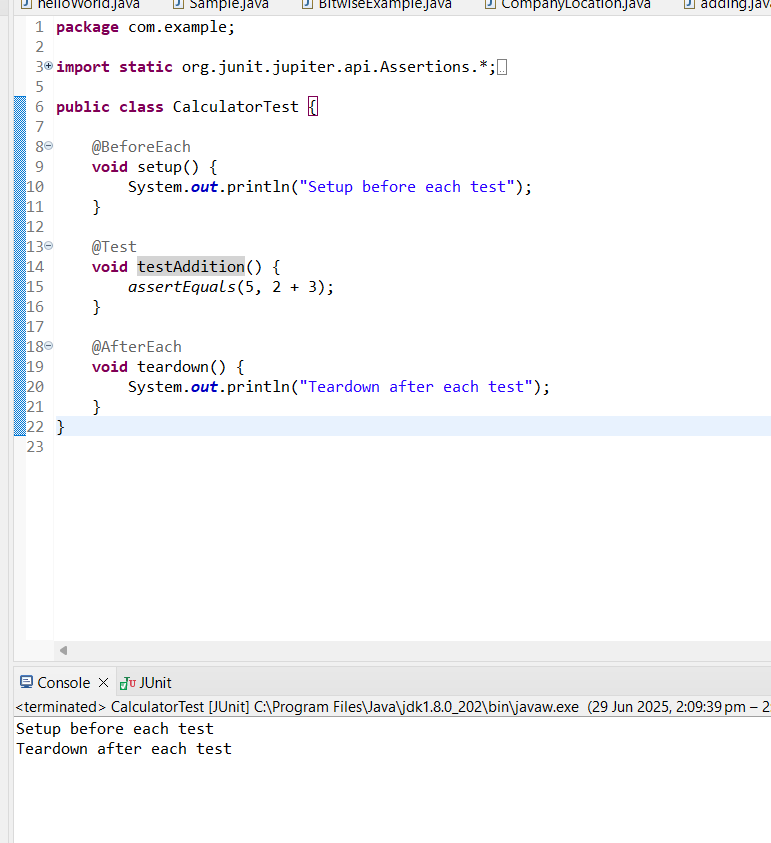
@AfterEach

void teardown() {

System.out.println("Teardown after each test");

}

}



A screenshot of a computer

AI-generated content may be incorrect.

**2>Assertions in junit**

package com.example;

import org.junit.jupiter.api.Test;

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

public class CalculatorTest {

Calculator calc = new Calculator();

@Test

void testAdd() {

assertEquals(5, calc.add(2, 3), "Addition should be correct");

}

@Test

void testSubtract() {

assertEquals(1, calc.subtract(4, 3), "Subtraction should be correct");

}

@Test

void testDivide() {

assertEquals(2, calc.divide(10, 5), "Division should be correct");

}

@Test

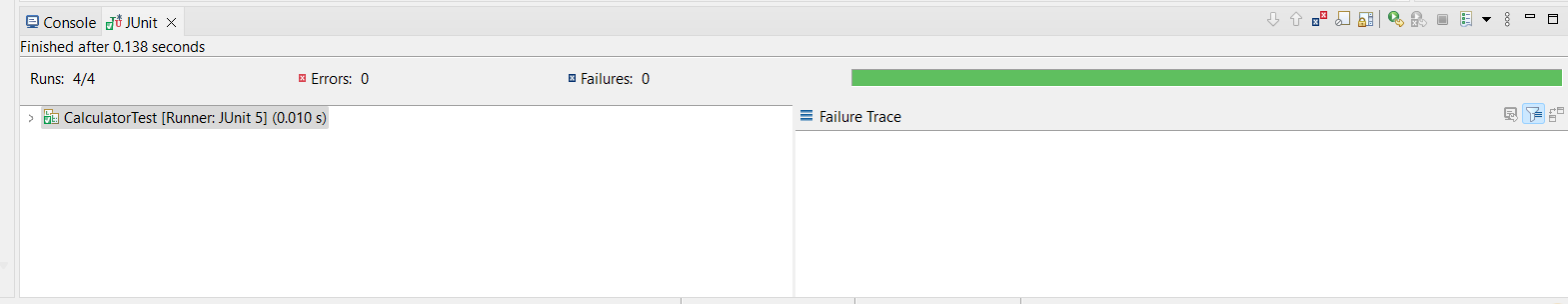
void testDivideByZeroThrowsException() {

Exception exception = assertThrows(ArithmeticException.class, () -> calc.divide(10, 0));

assertEquals("Division by zero", exception.getMessage());

}

}



**4>Arrange-Act-Assert(AAA)**

package com.example;

import org.junit.jupiter.api.Test;

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

public class CalculatorTest {

Calculator calc = new Calculator();

@Test

void testAdd() {

assertEquals(5, calc.add(2, 3), "Addition should be correct");

}

@Test

void testSubtract() {

assertEquals(1, calc.subtract(4, 3), "Subtraction should be correct");

}

@Test

void testDivide() {

assertEquals(2, calc.divide(10, 5), "Division should be correct");

}

@Test

void testDivideByZeroThrowsException() {

Exception exception = assertThrows(ArithmeticException.class, () -> calc.divide(10, 0));

assertEquals("Division by zero", exception.getMessage());

}

}

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;

}

public int divide(int a, int b) {

if (b == 0) {

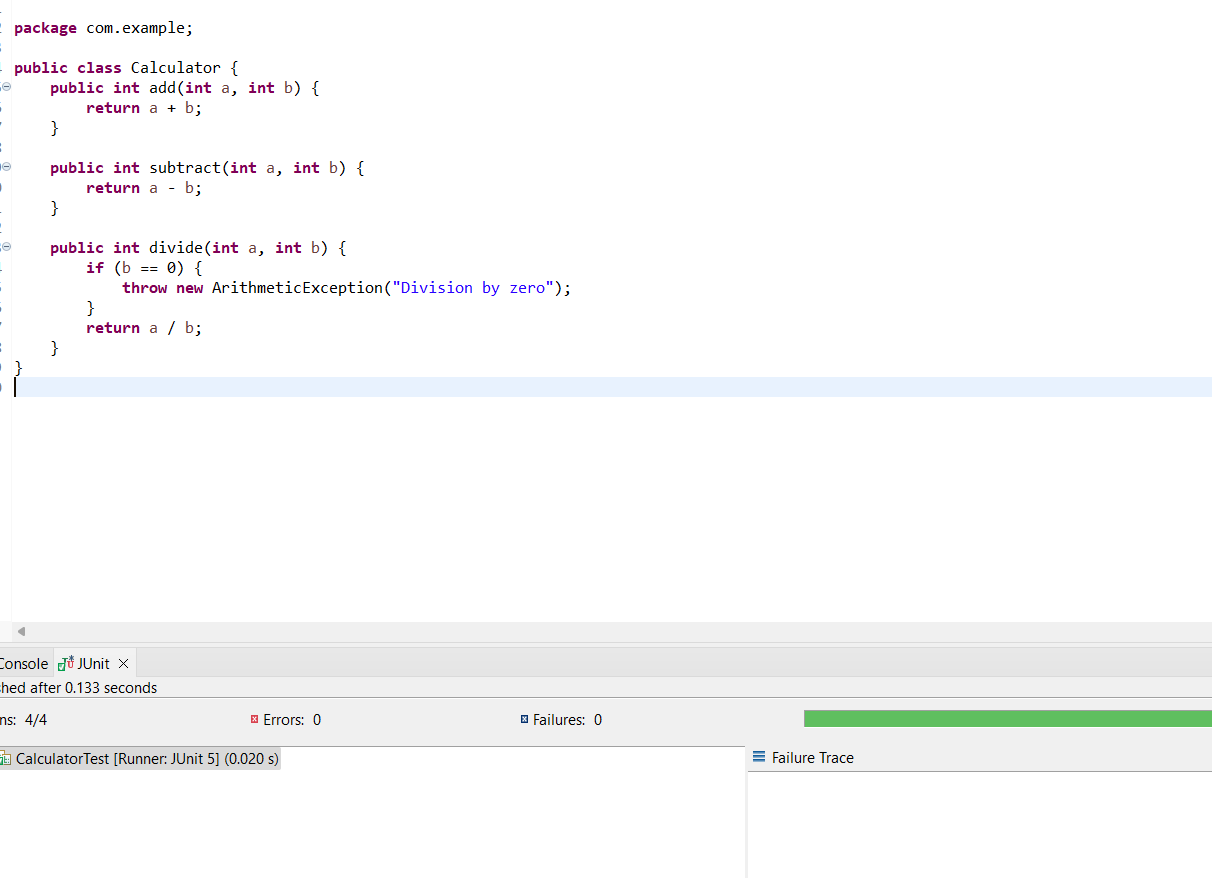
throw new ArithmeticException("Division by zero");

}

return a / b;

}

}



**PART 3:Mockito exercises**

**Mocking and stubbing**

package com.example;

import org.junit.jupiter.api.Test;

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

import static org.mockito.Mockito.\*;

public class UserServiceTest {

@Test

void testGetUserEmail\_Stubbing() {

UserRepository mockRepo = mock(UserRepository.class);

when(mockRepo.findEmailById(1)).thenReturn("john@example.com");

UserService service = new UserService(mockRepo);

String email = service.getUserEmail(1);

assertEquals("john@example.com", email);

}

}

package com.example;

public class UserService {

private UserRepository repository;

public UserService(UserRepository repository) {

this.repository = repository;

}

public String getUserEmail(int userId) {

return repository.findEmailById(userId);

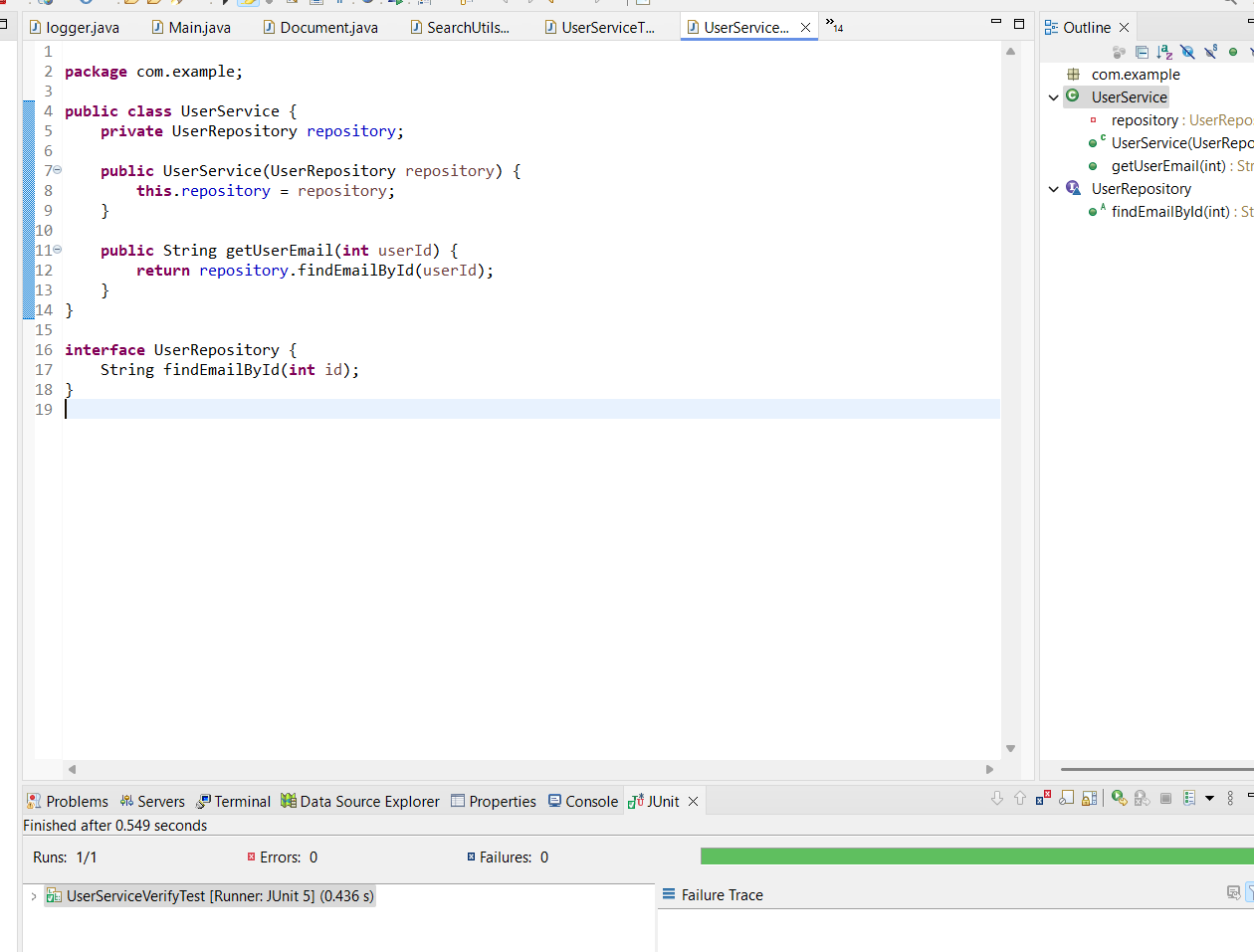
}

}

interface UserRepository {

String findEmailById(int id);

}



**2>mockito-verifying-interactions**

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class UserServiceVerifyTest {

@Test

void testVerifyFindEmailCalled() {

// Arrange

UserRepository mockRepo = mock(UserRepository.class);

UserService service = new UserService(mockRepo);

// Act

service.getUserEmail(1);

// Assert (Verify Interaction)

verify(mockRepo).findEmailById(1);

}

}

package com.example;

public class UserService {

private UserRepository repository;

public UserService(UserRepository repository) {

this.repository = repository;

}

public String getUserEmail(int userId) {

return repository.findEmailById(userId);

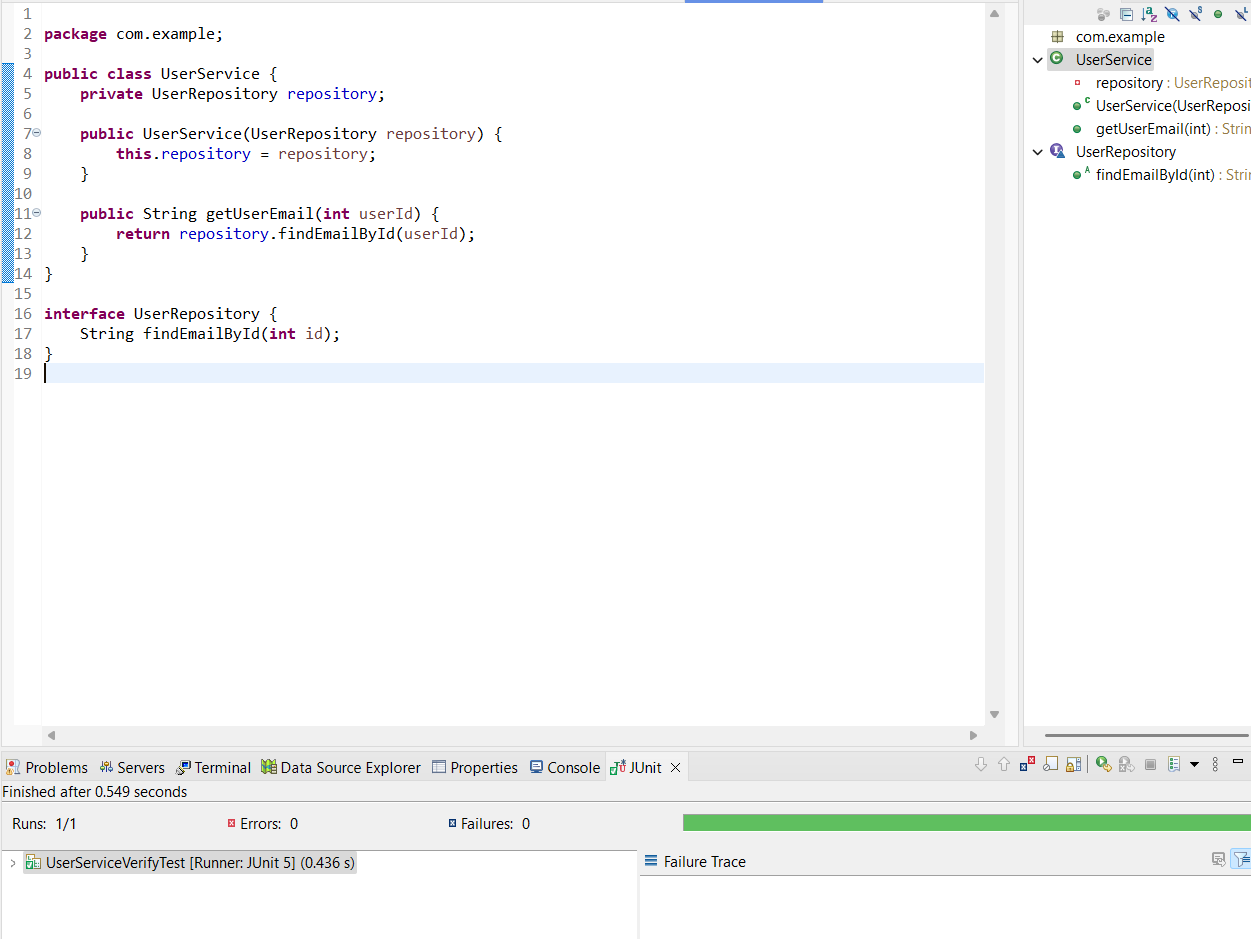
}

}

interface UserRepository {

String findEmailById(int id);

}



**PART 4: SLF4J Logging**

**Exercise 1: Logging Error and Warning**

package com.example;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LoggerExample {

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

public static void main(String[] args) {

logger.info("Application started.");

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

logger.error("This is an error message.");

}

}

