**PL/SQL programming**

**Exercise 1: Control Structures**

**Scenario 1**

**//Table Creation:**

CREATE TABLE Customers (

C\_id NUMBER PRIMARY KEY,

C\_Name VARCHAR2(50),

C\_Age NUMBER,

loan\_interest NUMBER

);

**//Sample Data:**

INSERT INTO Customers VALUES (1, 'Gnaneswara', 21, 10.5);

INSERT INTO Customers VALUES (2, 'Nani', 20, 9.5);

INSERT INTO Customers VALUES (3, 'siri', 30, 11.0);

**Solution:**

BEGIN

FOR cust\_rec IN (SELECT c\_id, age FROM customers) LOOP

IF cust\_rec.Age > 60 THEN

UPDATE Customers

SET loan\_interest = loan\_interest - 1

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

DBMS\_OUTPUT.PUT\_LINE('1% discount applied to customer ID: ' || cust\_rec.customer\_id);

END IF;

END LOOP;

END;

/

**Scenario 2**

**//Table Creation:**

CREATE TABLE Customers (

c\_id NUMBER PRIMARY KEY,

name VARCHAR2(50),

balance NUMBER,

IsVIP CHAR(1)

);

**//Sample Data:**

INSERT INTO Customers VALUES (1, 'Gnaneswara', 10000,’F’);

INSERT INTO Customers VALUES (2, 'Nani', 5000,’F’);

INSERT INTO Customers VALUES (3, 'siri', 20000,’F’);

**Solution:**

BEGIN

FOR cust\_rec IN (SELECT c\_id, balance FROM customers) LOOP

IF cust\_rec.balance > 10000 THEN

UPDATE customers

SET IsVIP = 'T'

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

DBMS\_OUTPUT.PUT\_LINE('Customer ID ' || cust\_rec.c\_id || ' promoted to VIP.');

END IF;

END LOOP;

END;

/

**Scenario 3**

**//Table Creation:**

CREATE TABLE loans (

loan\_id NUMBER PRIMARY KEY,

c\_name VARCHAR2(50),

due\_date DATE

);

**//Sample Data:**

INSERT INTO Customers VALUES (1, 'Gnaneswara', SYSDATE + 10);

INSERT INTO Customers VALUES (2, 'Nani', SYSDATE + 20);

INSERT INTO Customers VALUES (3, 'siri', SYSDATE + 30);

**Solution:**

BEGIN

FOR loan\_rec IN (

SELECT c\_name, due\_date

FROM loans

WHERE due\_date BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: ' || loan\_rec.customer\_name ||

', your loan is due on ' || TO\_CHAR(loan\_rec.due\_date, 'DD-Mon-YYYY'));

END LOOP;

END;

/

**Exercise 3: Stored Procedures**

**Scenario 1**

**Solution:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR acc\_rec IN (SELECT account\_id, balance FROM savings\_accounts) LOOP

UPDATE savings\_accounts

SET balance = balance + (acc\_rec.balance \* 0.01)

WHERE account\_id = acc\_rec.account\_id;

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

END LOOP;

END;

/

BEGIN

ProcessMonthlyInterest;

END;

/

**Scenario 2**

**Solution:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percent IN NUMBER

) IS

BEGIN

UPDATE employees

SET salary = salary + (salary \* p\_bonus\_percent / 100)

WHERE department = p\_department;

DBMS\_OUTPUT.PUT\_LINE('Bonus of ' || p\_bonus\_percent || '% applied to department: ' || p\_department);

END;

/

BEGIN

UpdateEmployeeBonus('Sales', 10);

END;

/

**Scenario 3**

**//Table Creation:**

CREATE TABLE accounts (

Account\_id NUMBER PRIMARY KEY,

customer\_name VARCHAR2(50),

balance NUMBER

);

**//Sample Data:**

INSERT INTO Customers VALUES (1, 'Gnaneswara', 20000);

INSERT INTO Customers VALUES (2, 'Nani', 10000);

INSERT INTO Customers VALUES (3, 'siri', 15000);

**Solution:**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) IS

v\_balance NUMBER;

BEGIN

-- Get source account balance

SELECT balance INTO v\_balance

FROM accounts

WHERE account\_id = p\_from\_account;

-- Check if sufficient balance

IF v\_balance < p\_amount THEN

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: Insufficient balance in account ' || p\_from\_account);

ELSE

-- 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('₹' || p\_amount || ' transferred from account ' || p\_from\_account ||

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

END IF;

END;

/

BEGIN

TransferFunds(101, 102, 5000);

END;

/

**TDD using JUnit5 and Mockito**

**Exercise 1: Setting Up Junit**

**Pom.xml:**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.Junit5</groupId>

<artifactId>Junit5\_Basics</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>Junit5\_Basics</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<maven.compiler.source>24</maven.compiler.source>

<maven.compiler.target>24</maven.compiler.target>

<junit.jupiter.version>5.10.0</junit.jupiter.version>

</properties>

<dependencies>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter</artifactId>

<version>${junit.jupiter.version}</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.1.2</version>

<configuration>

<useModulePath>false</useModulePath>

</configuration>

</plugin>

</plugins>

</build>

</project>

**Calculator.java:**

**package** com.Junit5.Juint5\_Basics;

**public** **class** Calculator {

**public** **int** add(**int** a,**int** b) {

**return** a+b;

}

}

**CalculatorTest.java:**

package com.Junit5.Juint5\_Basics;

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

import org.junit.jupiter.api.Test;

class CalculatorTest {

Calculator c=new Calculator();

@Test

public void test() {

}

}

**Exercise 3: Assertions in Junit**

**Code:**

package com.Junit5.Juint5\_Basics;

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

import org.junit.jupiter.api.Test;

public class AssertionsTest {

@Test

public void Test1() {

int actual=2+3;

int expected=5;

*assertEquals*(expected,actual);

}

@Test

public void Test2() {

int actual=2-3;

int expected=5;

*assertNotEquals*(expected,actual);

}

@Test

public void Test3() {

*assertTrue*(2>1);

}

@Test

public void Test4() {

*assertFalse*(1>2);

}

@Test

public void Test5() {

String s=null;

*assertNull*(s);

}

@Test

public void Test6() {

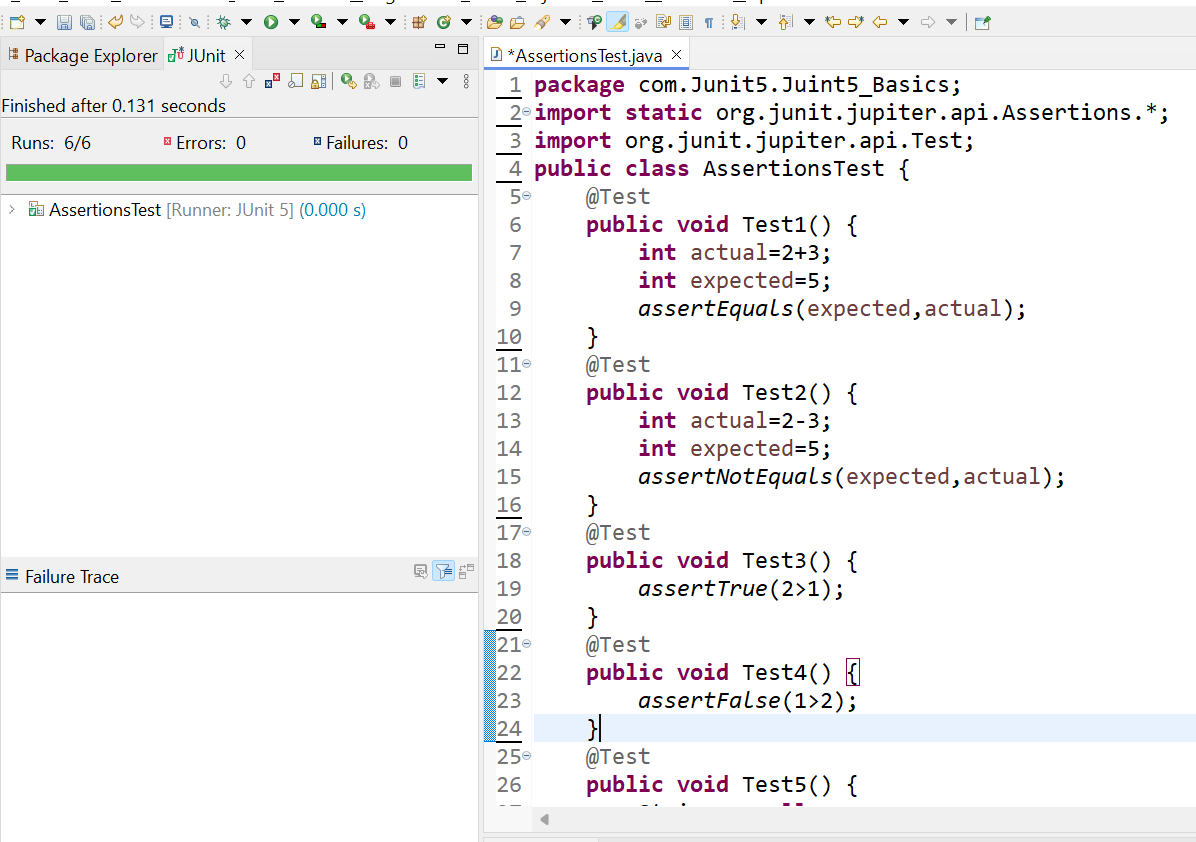
String s="Junit";

*assertNotNull*(s);

}

}

**Output:**

****

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

**Squares.java:**

package com.Junit5.Juint5\_Basics;

public class Squares {

public int square(int a){

return a\*a;

}

}

**SquaresTest.java:**

package com.Junit5.Juint5\_Basics;

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

import org.junit.jupiter.api.AfterEach;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

class SquaresTest {

Squares sq;

@BeforeEach

void setUp() {

sq = new Squares();

System.*out*.println("Setup done");

}

@Test

void testSquare() {

int result = sq.square(5);

*assertEquals*(25, result);

}

@AfterEach

void tearDown() {

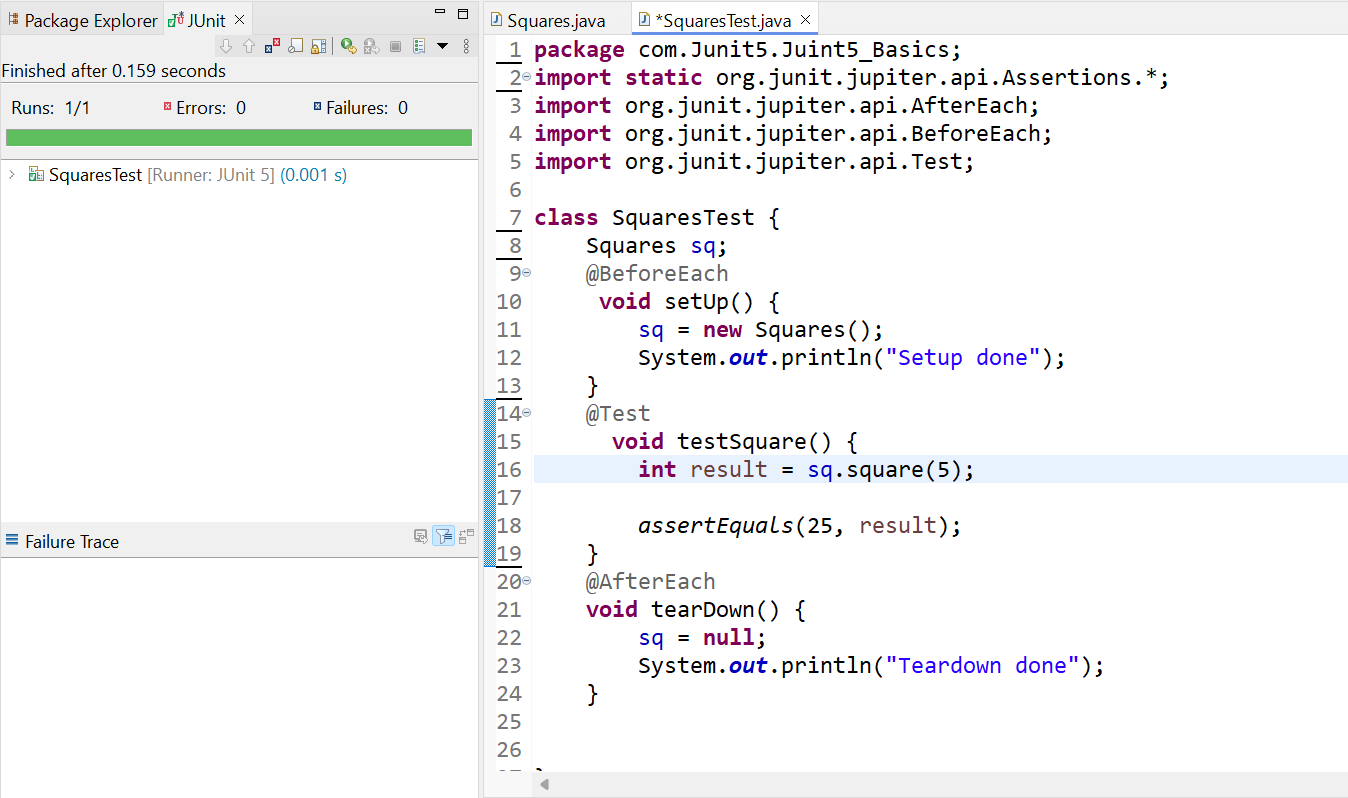
sq = null;

System.*out*.println("Teardown done");

}

}

**Output:**

****

**Exercise 1: Mocking and Stubbing**

**WeatherAPI.java**

package com.weather.WeatherApp;

public interface WeatherAPI {

String getWeather(String city);

}

**WeatherService.java**

package com.weather.WeatherApp;

public class WeatherService {

private WeatherAPI weatherAPI;

public WeatherService(WeatherAPI weatherAPI) {

this.weatherAPI = weatherAPI;

}

public String fetchWeather(String city) {

return weatherAPI.getWeather(city);

}

}

**WeatherServiceTest.java**

package com.weather.test;

import com.weather.WeatherApp.WeatherAPI;

import com.weather.WeatherApp.WeatherService;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class WeatherServiceTest {

private WeatherAPI mockWeatherAPI;

private WeatherService weatherService;

@BeforeEach

public void setup() {

mockWeatherAPI = *mock*(WeatherAPI.class);

*when*(mockWeatherAPI.getWeather("Delhi")).thenReturn("Sunny");

*when*(mockWeatherAPI.getWeather("Mumbai")).thenReturn("Rainy");

*when*(mockWeatherAPI.getWeather("Chennai")).thenReturn("Humid");

*when*(mockWeatherAPI.getWeather("Kolkata")).thenReturn("Cloudy");

*when*(mockWeatherAPI.getWeather("Hyderabad")).thenReturn("Thunderstorms");

weatherService = new WeatherService(mockWeatherAPI);

}

@Test

public void testFetchWeatherDelhi() {

String result = weatherService.fetchWeather("Delhi");

System.*out*.println("Weather in Delhi: " + result);

*assertEquals*("Sunny", result);

}

@Test

public void testFetchWeatherMumbai() {

String result = weatherService.fetchWeather("Mumbai");

System.*out*.println("Weather in Mumbai: " + result);

*assertEquals*("Rainy", result);

}

@Test

public void testFetchWeatherChennai() {

String result = weatherService.fetchWeather("Chennai");

System.*out*.println("Weather in Chennai: " + result);

*assertEquals*("Humid", result);

}

@Test

public void testFetchWeatherKolkata() {

String result = weatherService.fetchWeather("Kolkata");

System.*out*.println("Weather in Kolkata: " + result);

*assertEquals*("Cloudy", result);

}

@Test

public void testFetchWeatherHyderabad() {

String result = weatherService.fetchWeather("Hyderabad");

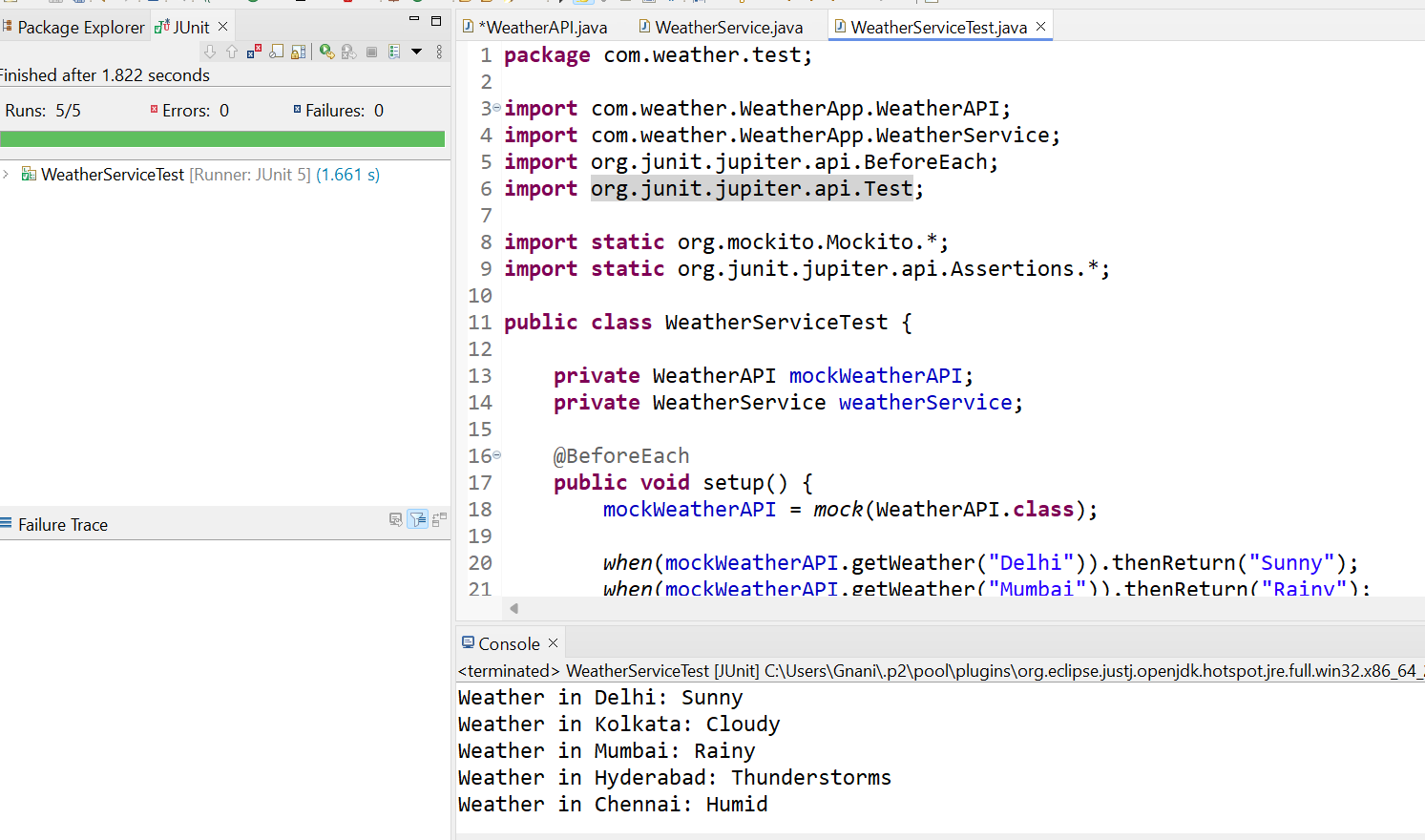
System.*out*.println("Weather in Hyderabad: " + result);

*assertEquals*("Thunderstorms", result);

}

}

**Output:**

****

**Exercise 2: Verifying Interactions**

**Notifier.java:**

package com.verify.interaction;

public interface Notifier {

void sendEmail(String email);

}

**UserService.java:**

package com.verify.interaction;

public class UserService {

private Notifier notifier;

public UserService(Notifier notifier) {

this.notifier = notifier;

}

public void registerUser(String email) {

notifier.sendEmail(email);

}

}

**UserServiceTest.java:**

package com.verify.test;

import com.verify.interaction.Notifier;

import com.verify.interaction.UserService;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class UserServiceTest {

@Test

public void testSendEmailCalledWithCorrectArgument() {

Notifier mockNotifier = *mock*(Notifier.class);

UserService userService = new UserService(mockNotifier);

userService.registerUser("test@example.com");

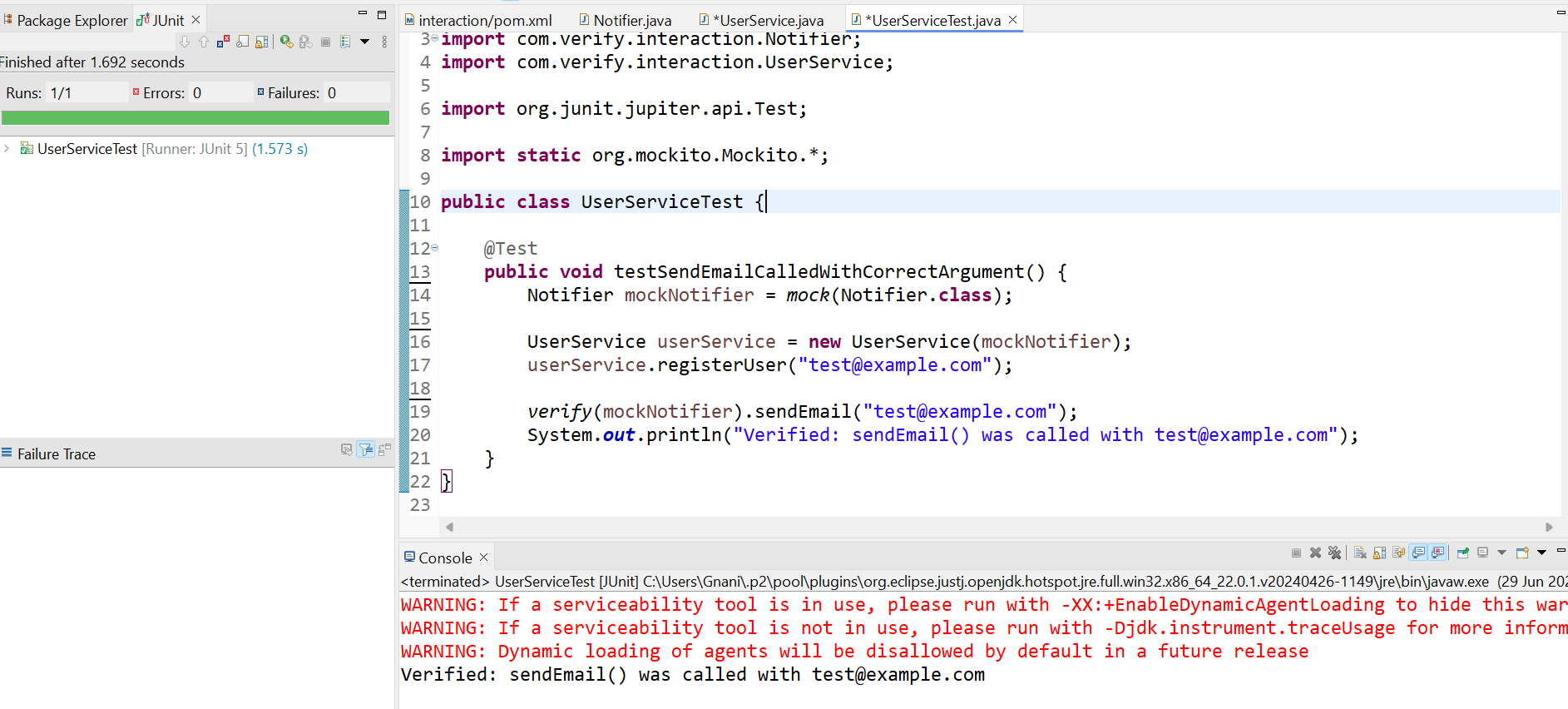
*verify*(mockNotifier).sendEmail("test@example.com");

System.*out*.println("Verified: sendEmail() was called with test@example.com");

}

}

**Output:**

****

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

**AppLogger.java:**

package com.logging.logdemo;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class AppLogger {

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

public static void main(String[] args) {

*logger*.info("Application started");

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

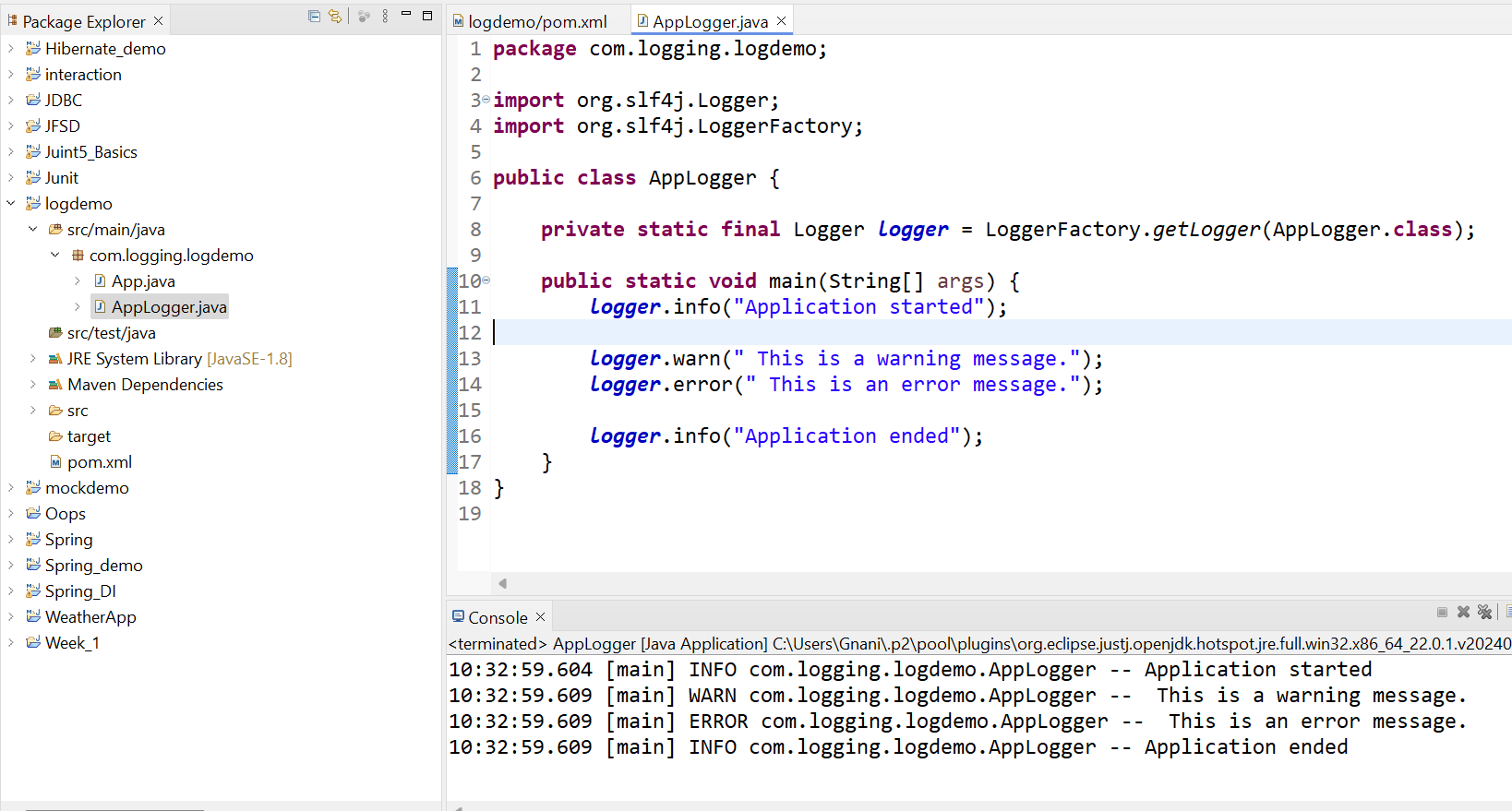
*logger*.error(" This is an error message.");

*logger*.info("Application ended");

}

}

**Output:**

****