**PL SQL PROGRAMMING**

**Exercise 1: Control Structures**

**Scenario 1: Apply Interest Rate Discount for Senior Citizens**

BEGIN

FOR cust IN (SELECT CustomerID, LoanID, InterestRate, Age

FROM Customers

JOIN Loans ON Customers.CustomerID = Loans.CustomerID)

LOOP

IF cust.Age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE LoanID = cust.LoanID;

END IF;

END LOOP;

COMMIT;

END;

**Scenario 2: Promote Customers to VIP Based on Balance**

BEGIN

FOR cust IN (SELECT CustomerID, Balance

FROM Customers)

LOOP

IF cust.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = cust.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

**Scenario 3: Send Loan Due Reminders Within 30 Days**

DECLARE

v\_due\_date DATE := SYSDATE + 30;

BEGIN

FOR loan\_rec IN (SELECT LoanID, CustomerID, DueDate

FROM Loans

WHERE DueDate <= v\_due\_date AND DueDate >= SYSDATE)

LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || loan\_rec.LoanID ||

' for Customer ID ' || loan\_rec.CustomerID ||

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

END LOOP;

END;

**Exercise 3: Stored Procedures**

**Scenario 1: ProcessMonthlyInterest for Savings Accounts**

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)

WHERE AccountID = acc.AccountID;

END LOOP;

COMMIT;

END;

**Scenario 2: UpdateEmployeeBonus Based on Department**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_DepartmentID IN NUMBER,

p\_BonusPercent IN NUMBER

) IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* p\_BonusPercent / 100)

WHERE DepartmentID = p\_DepartmentID;

COMMIT;

END;

**Scenario 3: TransferFunds Between Accounts**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_FromAccountID IN NUMBER,

p\_ToAccountID IN NUMBER,

p\_Amount IN NUMBER

) IS

v\_balance NUMBER;

BEGIN

-- Get the balance of the source account

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_FromAccountID

FOR UPDATE;

-- Check for sufficient balance

IF v\_balance < p\_Amount THEN

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

END IF;

-- Deduct from source

UPDATE Accounts

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_FromAccountID;

-- Add to destination

UPDATE Accounts

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_ToAccountID;

COMMIT;

END;

**JUnit Testing Exercises**

**Exercise 1: Setting Up Junit**

In IntelliJ IDEA:

* File → New → Project → Select Maven → Name your project → Finish.
* **Add JUnit Dependency in pom.xml**
* Open pom.xml and add this inside <dependencies>:

**<**dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

**Create a Java Class**

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

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));

}

}

**Exercise 3: Assertions in JUnit**

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

public void testAssertions() {

// Assert equals

assertEquals(5, 2 + 3);

// Assert true

assertTrue(5 > 3);

// Assert false

assertFalse(5 < 3);

// Assert null

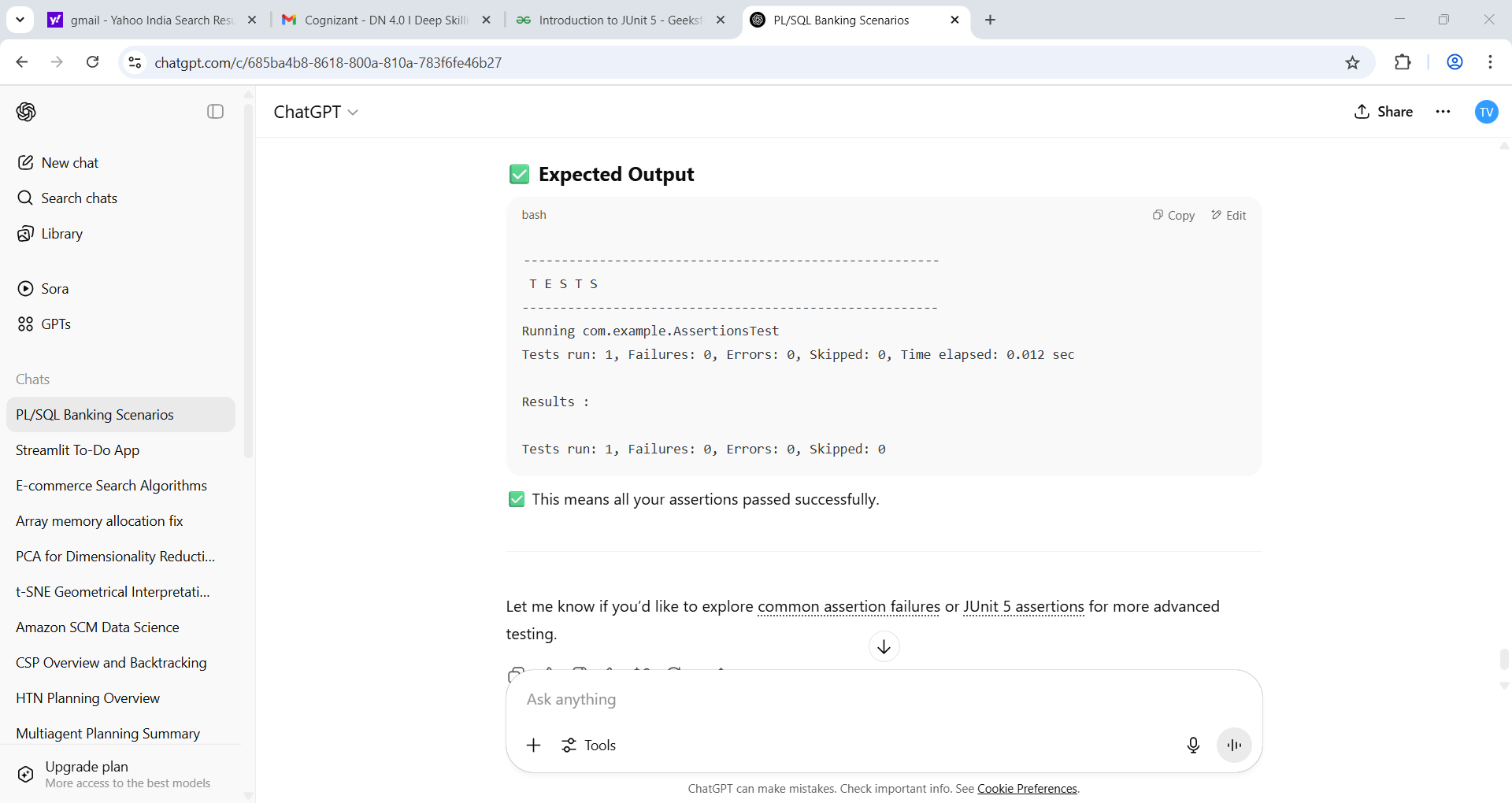
assertNull(null);

// Assert not null

assertNotNull(new Object());

}

}



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

**Teardown Methods in JUnit**

**Step 1: Calculator Class**

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int divide(int a, int b) {

return a / b;

}

}

**Test Class Using AAA and Setup/Teardown**

package com.example;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

private Calculator calc;

**@Before**

public void setUp() {

// Arrange - setup before each test

calc = new Calculator();

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

}

**@After**

public void tearDown() {

// Teardown - cleanup after each test

calc = null;

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

}

**@Test**

public void testAdd() {

// Act

int result = calc.add(2, 3);

// Assert

assertEquals(5, result);

}

**@Test**

public void testDivide() {

// Act

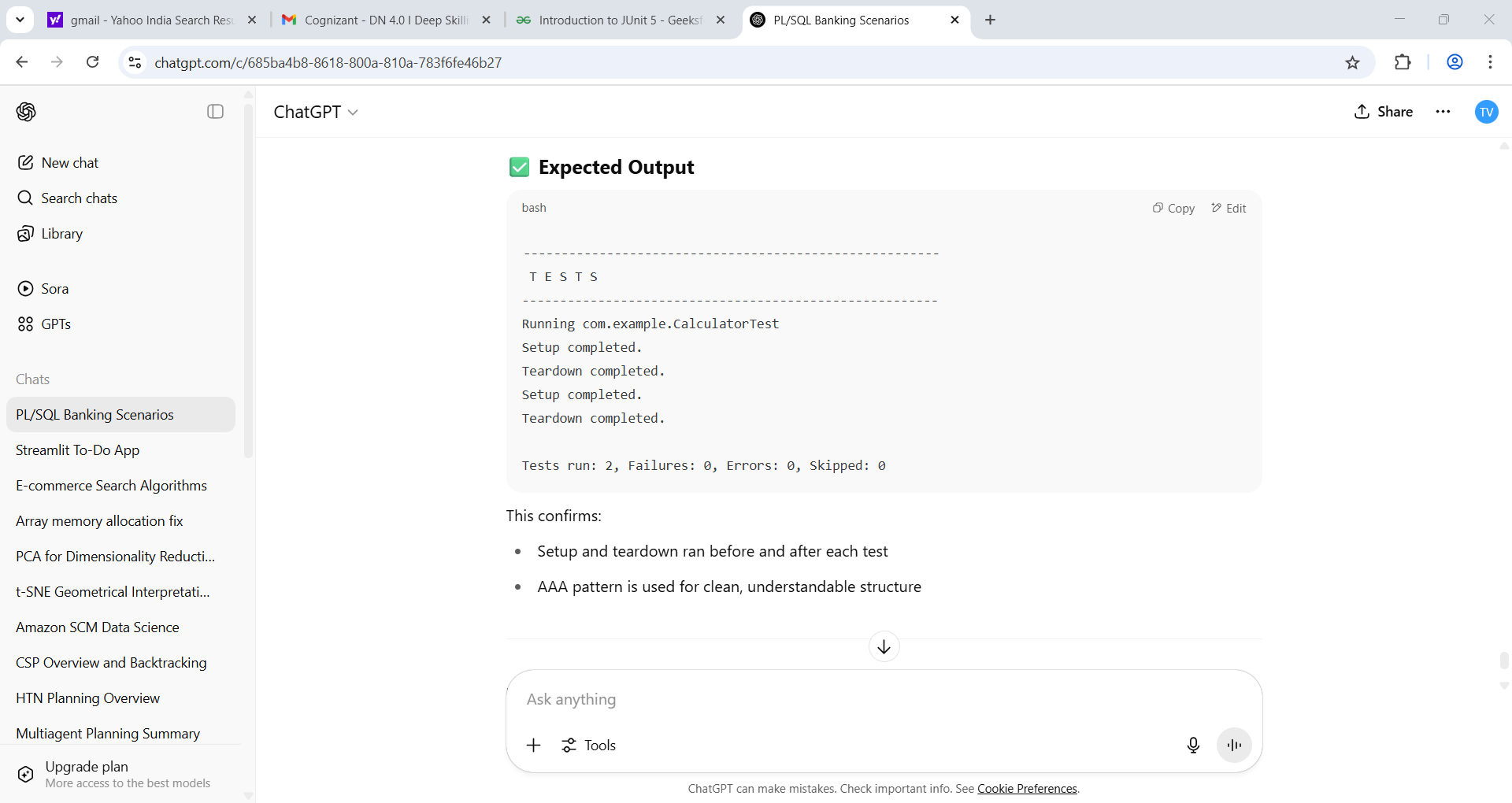
int result = calc.divide(10, 2);

// Assert

assertEquals(5, result);

}

}



**MOCKITO EXERCISES**

**Exercise 1: Mocking and Stubbing**

**Create Interfaces and Service**

// ExternalApi.java

public interface ExternalApi {

String getData();

}

// MyService.java

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**Test Class Using Mockito**

// MyServiceTest.java

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

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

public class MyServiceTest {

@Test

public void testExternalApi() {

// Step 1: Create mock object

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

// Step 2: Stub method to return predefined value

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

// Step 3: Inject mock into service

MyService service = new MyService(mockApi);

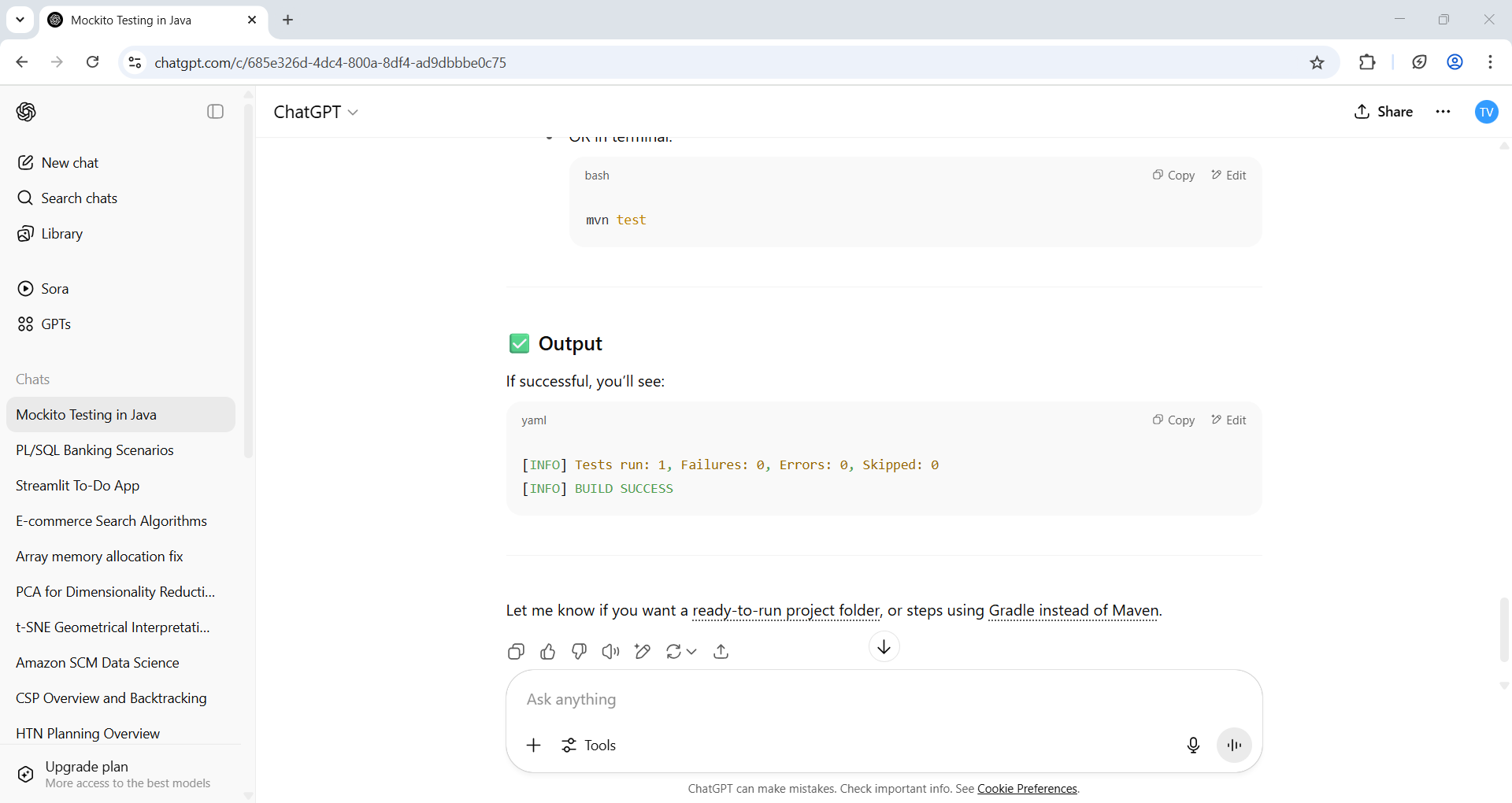
// Step 4: Execute and assert

String result = service.fetchData();

assertEquals("Mock Data", result);

}

}



**Exercise 2: Verifying Interactions**

**External Interface**

// ExternalApi.java

public interface ExternalApi {

String getData();

}

**Service Class Using ExternalApi**

// MyService.java

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData(); // This interaction will be verified

}

**}**

**Test Class for Verifying Interaction**

// MyServiceTest.java

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

// Step 1: Create mock

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

// Step 2: Inject into service

MyService service = new MyService(mockApi);

// Step 3: Call the method

service.fetchData();

// Step 4: Verify interaction

verify(mockApi).getData(); // Verifies if getData() was called exactly once

}

}

