**Advanced Mockito Hands-On Exercises**

**Exercise 1: Mocking Databases and Repositories You need to test a service that interacts with a database repository.**

Steps: 1. Create a mock repository using Mockito.

2. Stub the repository methods to return predefined data.

3. Write a test to verify the service logic using the mocked repository.

Solution Code:

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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

public class ServiceTest {

@Test public void testServiceWithMockRepository() {

Repository mockRepository = mock(Repository.class); when(mockRepository.getData()).thenReturn("Mock Data");

Service service = new Service(mockRepository);

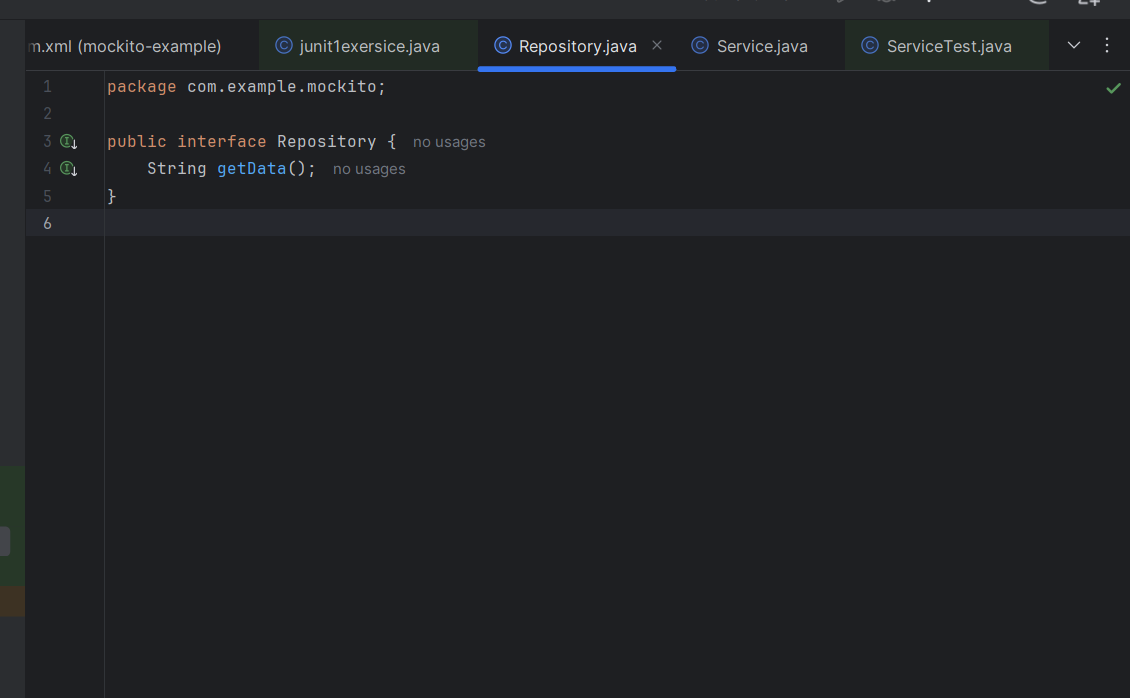
String result = service.processData();

assertEquals("Processed Mock Data", result);

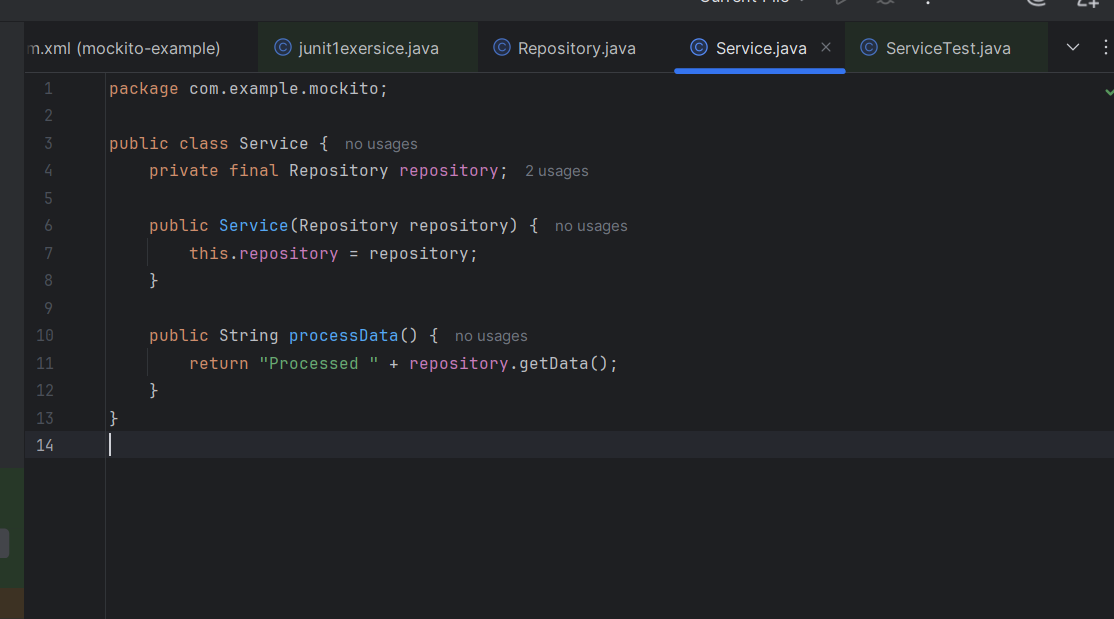
} }

**SOLUTION:**

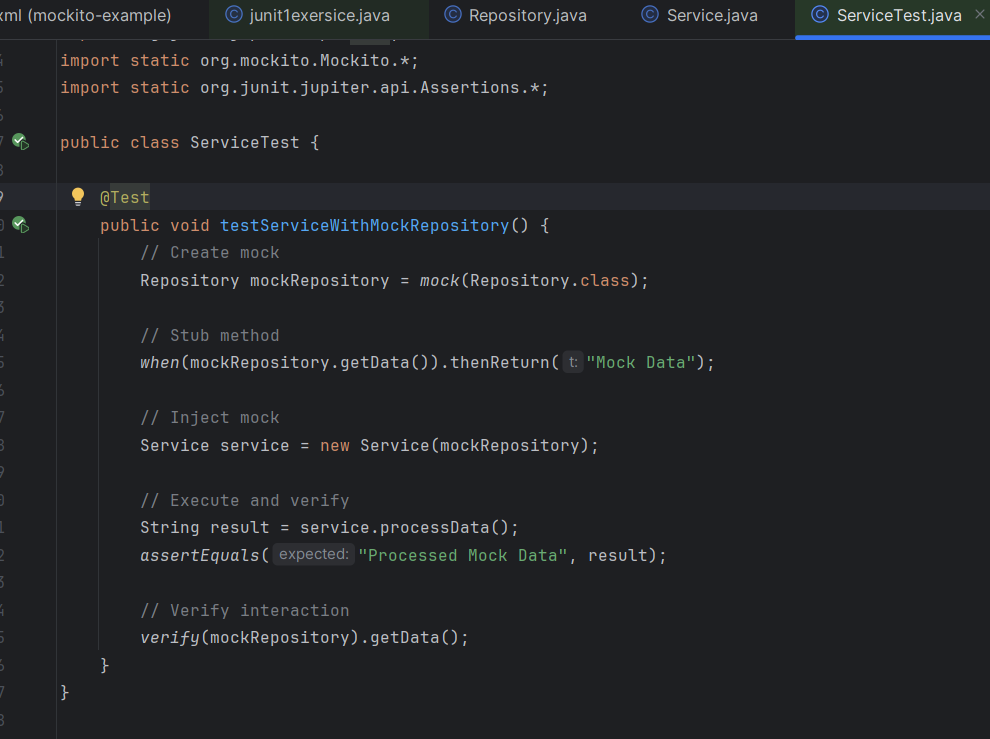
**Repository.java:**

****

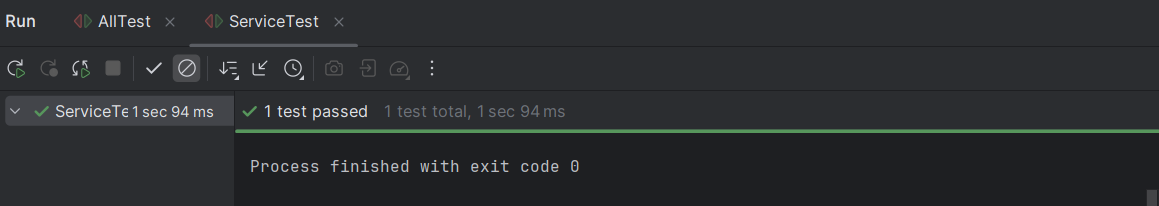
**Service.java:**

****

**ServiceTest.java:**

****

**OUTPUT:**

****

**Exercise 2: Mocking External Services (RESTful APIs)**

**You need to test a service that calls an external RESTful API.**

Steps: 1. Create a mock REST client using Mockito.

2. Stub the REST client methods to return predefined responses.

3. Write a test to verify the service logic using the mocked REST client.

Solution Code:

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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

public class ApiServiceTest {

@Test public void testServiceWithMockRestClient() {

RestClient mockRestClient = mock(RestClient.class); when(mockRestClient.getResponse()).thenReturn("Mock Response");

ApiService apiService = new ApiService(mockRestClient);

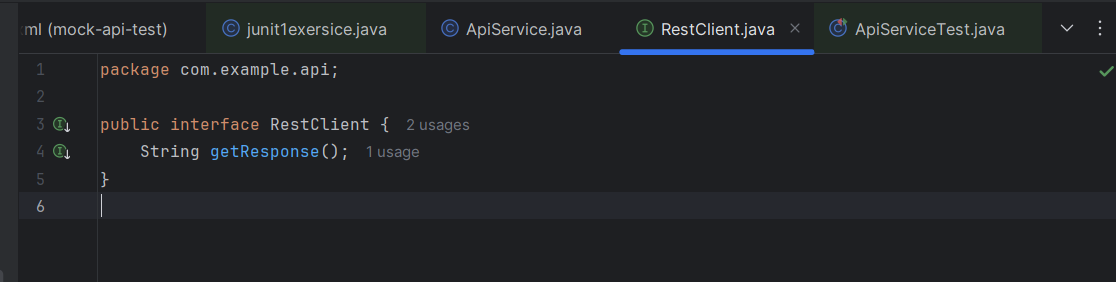
String result = apiService.fetchData();

assertEquals("Fetched Mock Response", result);

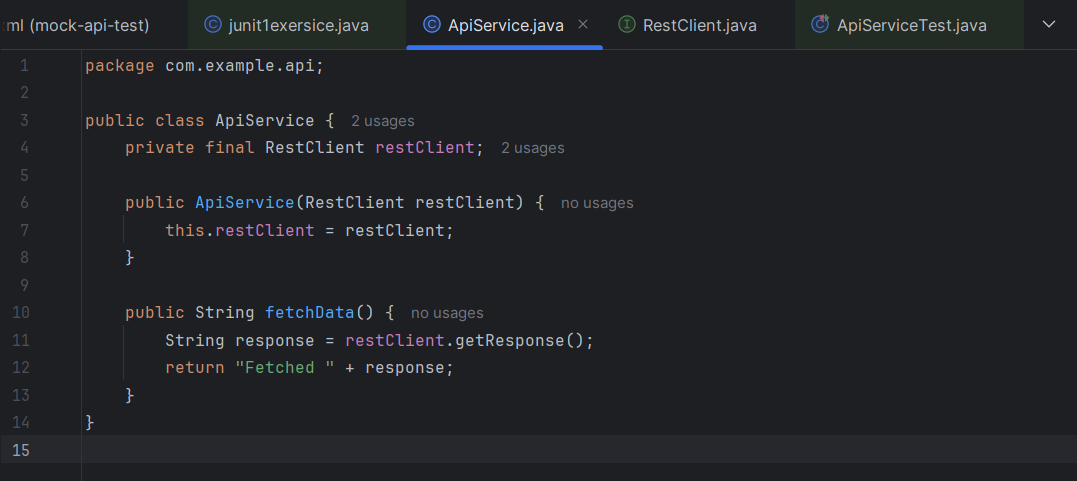
} }

**SOLUTION:**

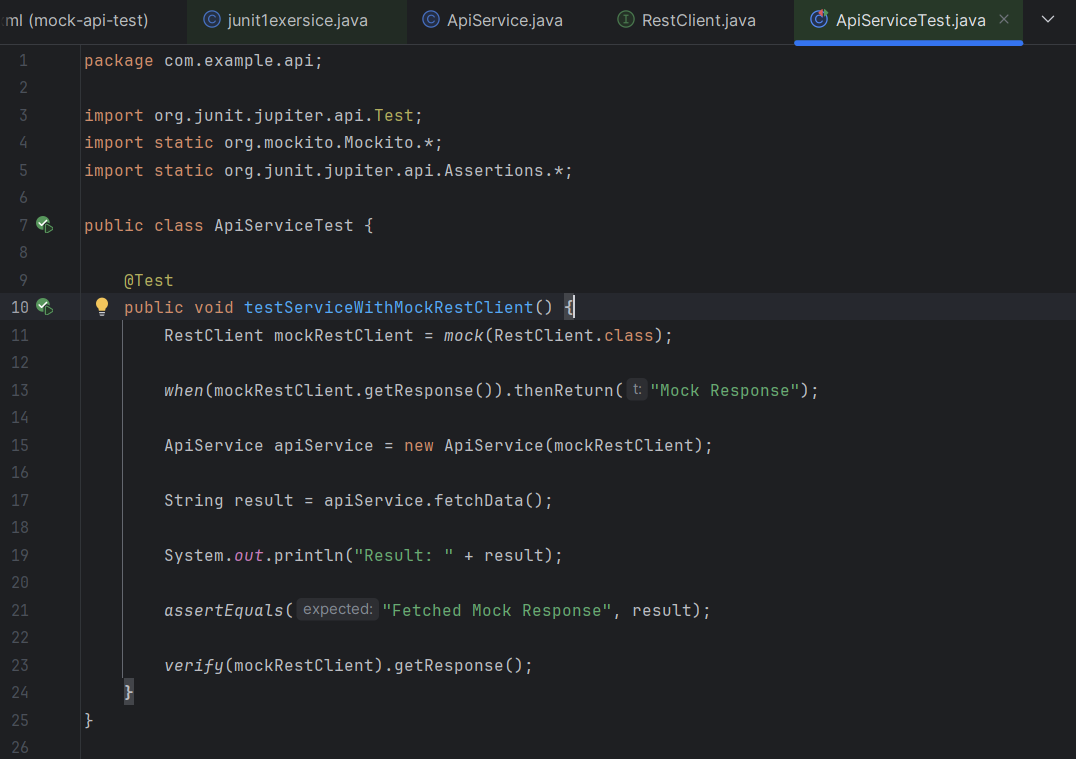
**Restlient.java:**

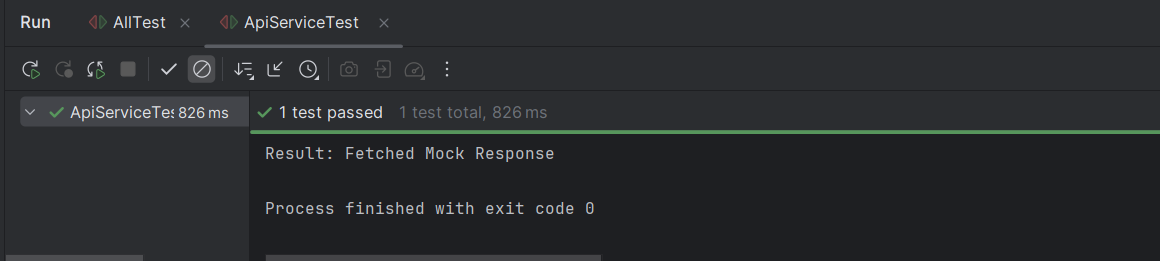
****

**ApiService.java:**

****

**ApiServiceTest.java:**

****

**OUTPUT:**

**Exercise 3: Mocking File I/O**

**You need to test a service that reads from and writes to files.**

Steps: 1. Create a mock file reader and writer using Mockito.

2. Stub the file reader and writer methods to simulate file operations.

3. Write a test to verify the service logic using the mocked file reader and writer.

Solution Code:

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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

public class FileServiceTest {

@Test public void testServiceWithMockFileIO() {

FileReader mockFileReader = mock(FileReader.class);

FileWriter mockFileWriter = mock(FileWriter.class); when(mockFileReader.read()).thenReturn("Mock File Content");

FileService fileService = new FileService(mockFileReader, mockFileWriter);

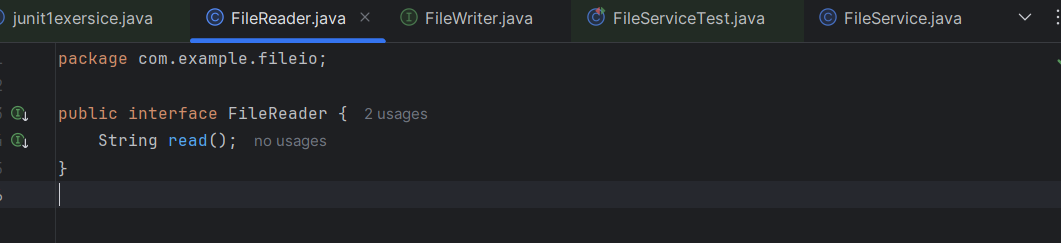
String result = fileService.processFile();

assertEquals("Processed Mock File Content", result);

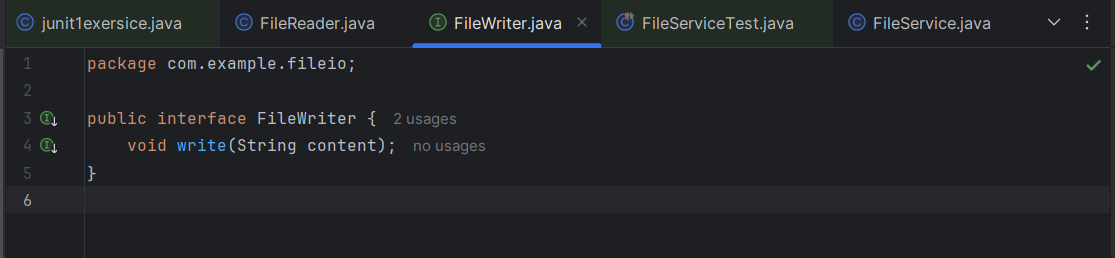
} }

**SOLUTION:**

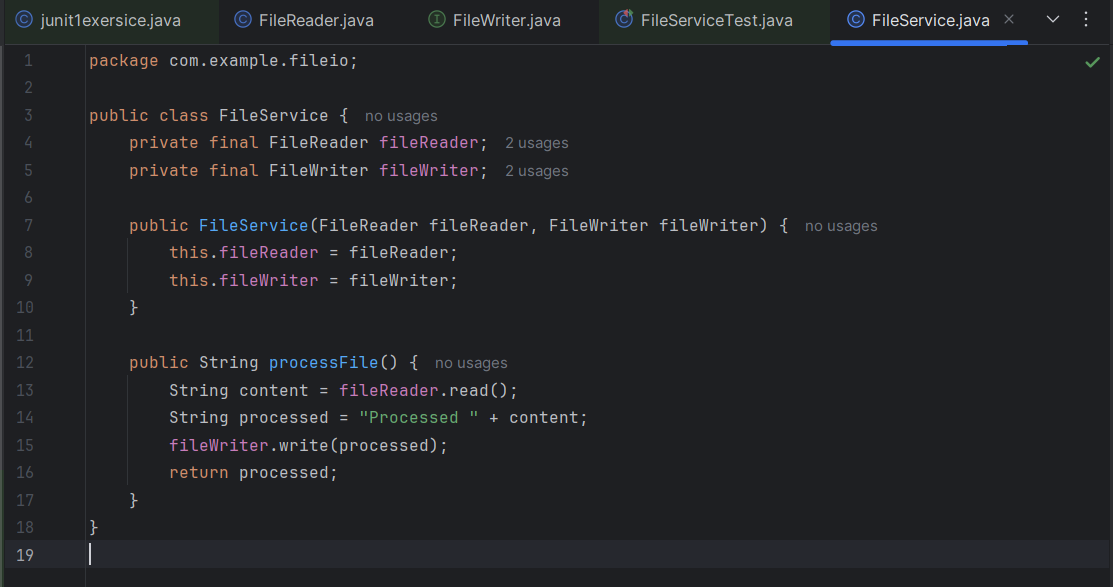
**FileReader.java:**

****

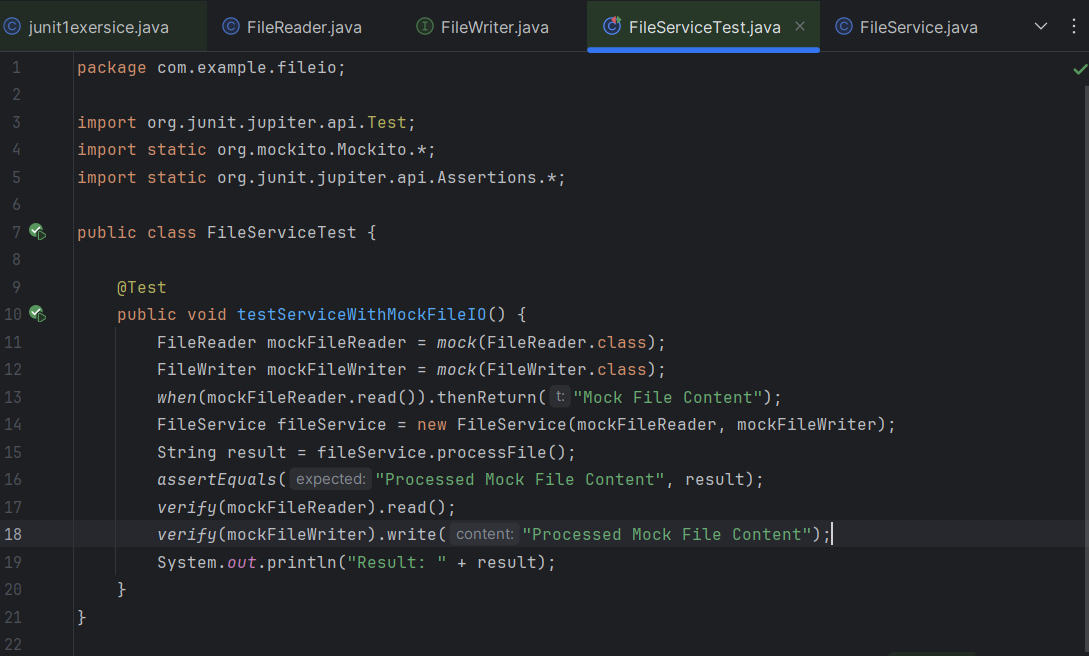
**FileWriter.java:**

****

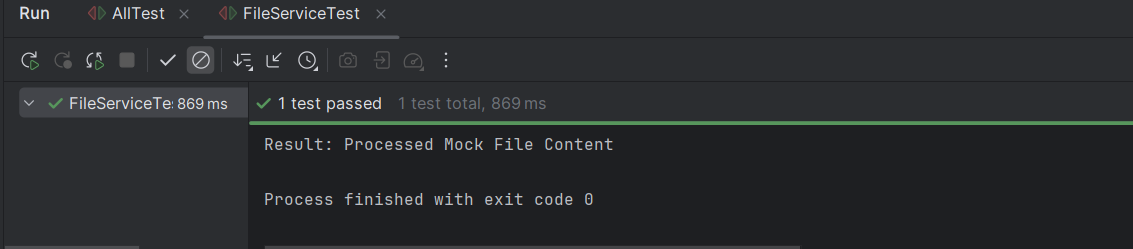
**FileService.java:**

****

**FileServiceTest.java:**

****

**OUTPUT:**

****

**Exercise 4: Mocking Network Interactions You need to test a service that interacts with network resources.**

Steps: 4. 1. Create a mock network client using Mockito.

5. 2. Stub the network client methods to simulate network interactions.

6. 3. Write a test to verify the service logic using the mocked network client.

Solution Code:

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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

public class NetworkServiceTest {

@Test public void testServiceWithMockNetworkClient() {

NetworkClient mockNetworkClient = mock(NetworkClient.class); when(mockNetworkClient.connect()).thenReturn("Mock Connection");

NetworkService networkService = new NetworkService(mockNetworkClient);

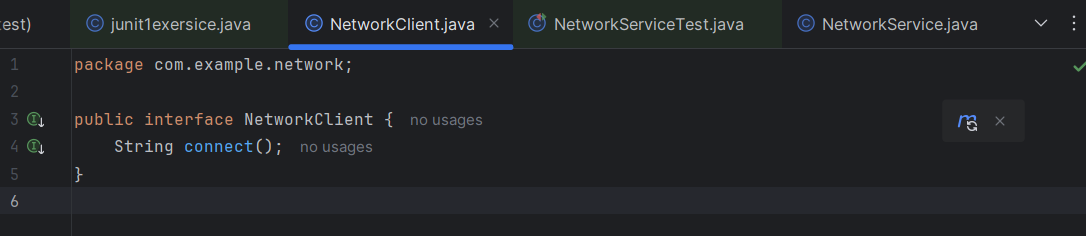
String result = networkService.connectToServer();

assertEquals("Connected to Mock Connection", result);

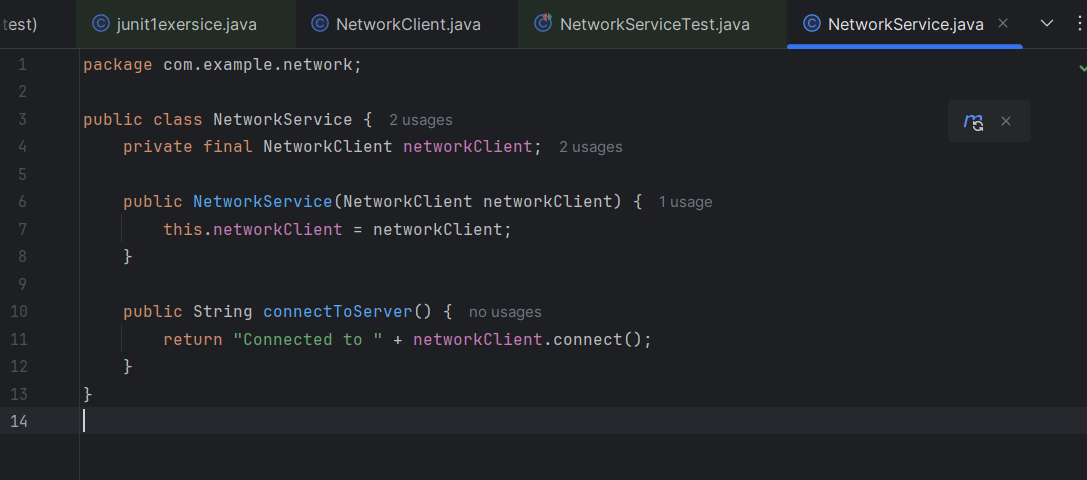
} }

**SOLUTION:**

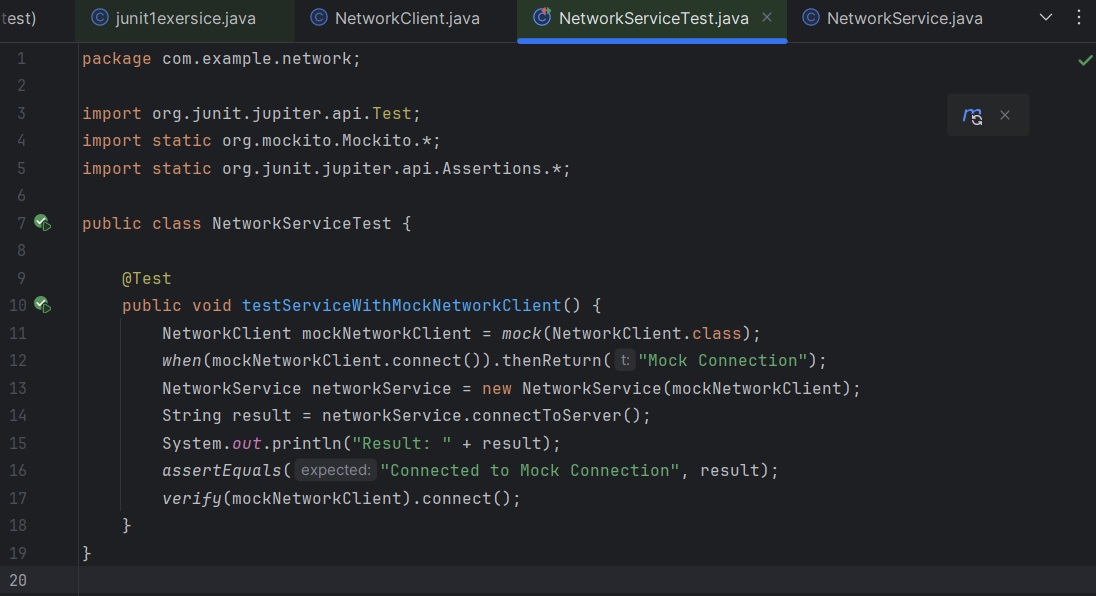
**NetworkClient.java:**

****

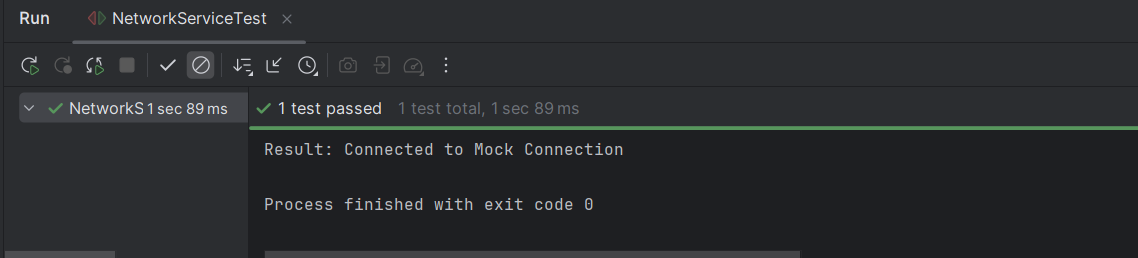
**NetworkService.java:**

****

**NetworkClientTest.java:**

****

**OUTPUT:**

****

**Exercise 5: Mocking Multiple Return Values**

**You need to test a service that calls a method multiple times with different return values. Steps:**

1. Create a mock object using Mockito.

2. Stub the method to return different values on consecutive calls.

3. Write a test to verify the service logic using the mocked object.

Solution Code:

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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

public class MultiReturnServiceTest {

@Test public void testServiceWithMultipleReturnValues() {

Repository mockRepository = mock(Repository.class);

when(mockRepository.getData()) .thenReturn("First Mock Data") .thenReturn("Second Mock Data");

Service service = new Service(mockRepository);

String firstResult = service.processData();

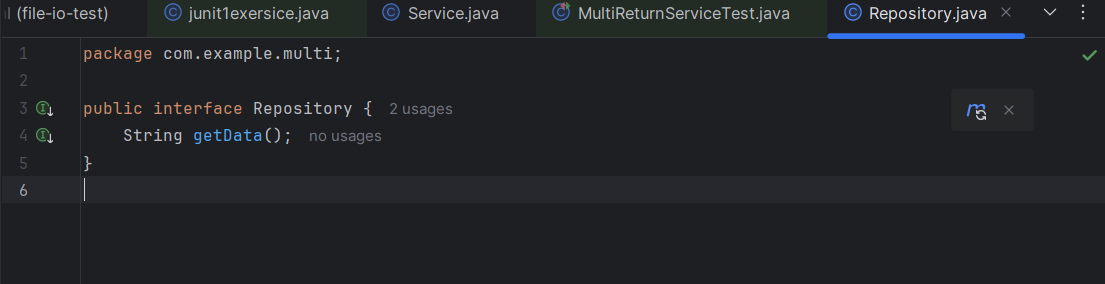
String secondResult = service.processData();

assertEquals("Processed First Mock Data", firstResult); assertEquals("Processed Second Mock Data", secondResult);

} }

**SOLUTION:**

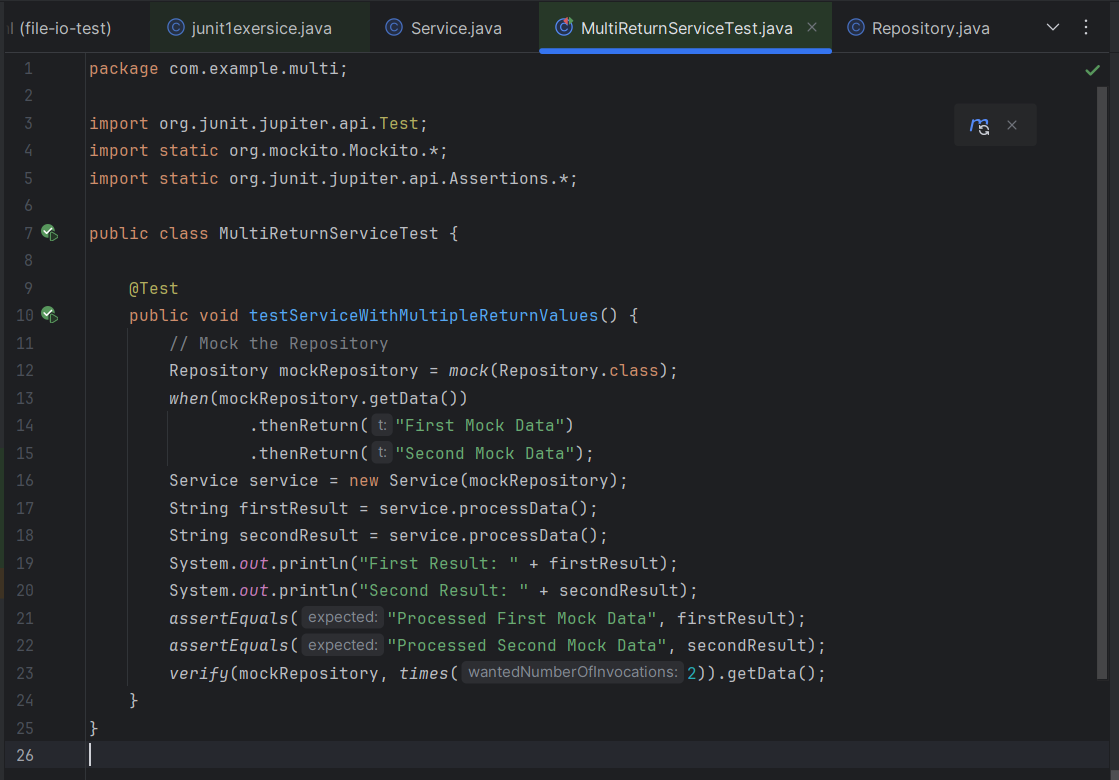
**Repository.java:**

****

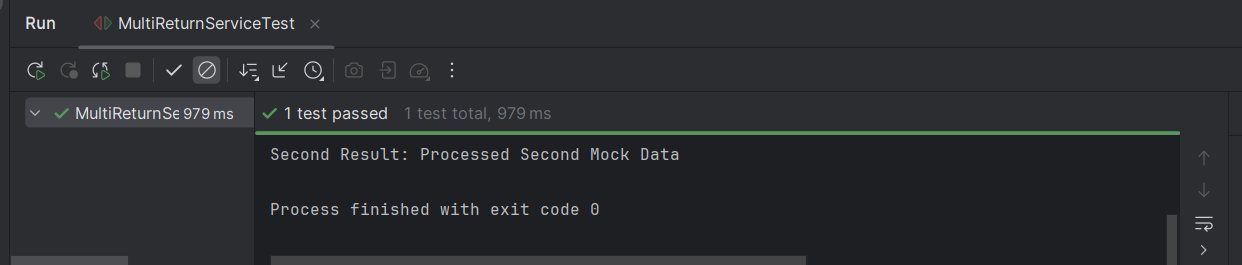
**Service.java:**

****

**MultiReturnServiceTest.java:**

****

**OUTPUT:**

****