**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);

}

}

**Program:**

**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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>mock-test</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.12.0</version>

<scope>test</scope>

</dependency>

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**Repository.java**

package com.example;

public interface Repository {

String getData();

}

**Service.java**

package com.example;

public class Service {

private Repository repository;

public Service(Repository repository) {

this.repository = repository;

}

public String processData() {

String data = repository.getData();

return "Processed " + data;

}

}

**ServiceTest.java**

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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);

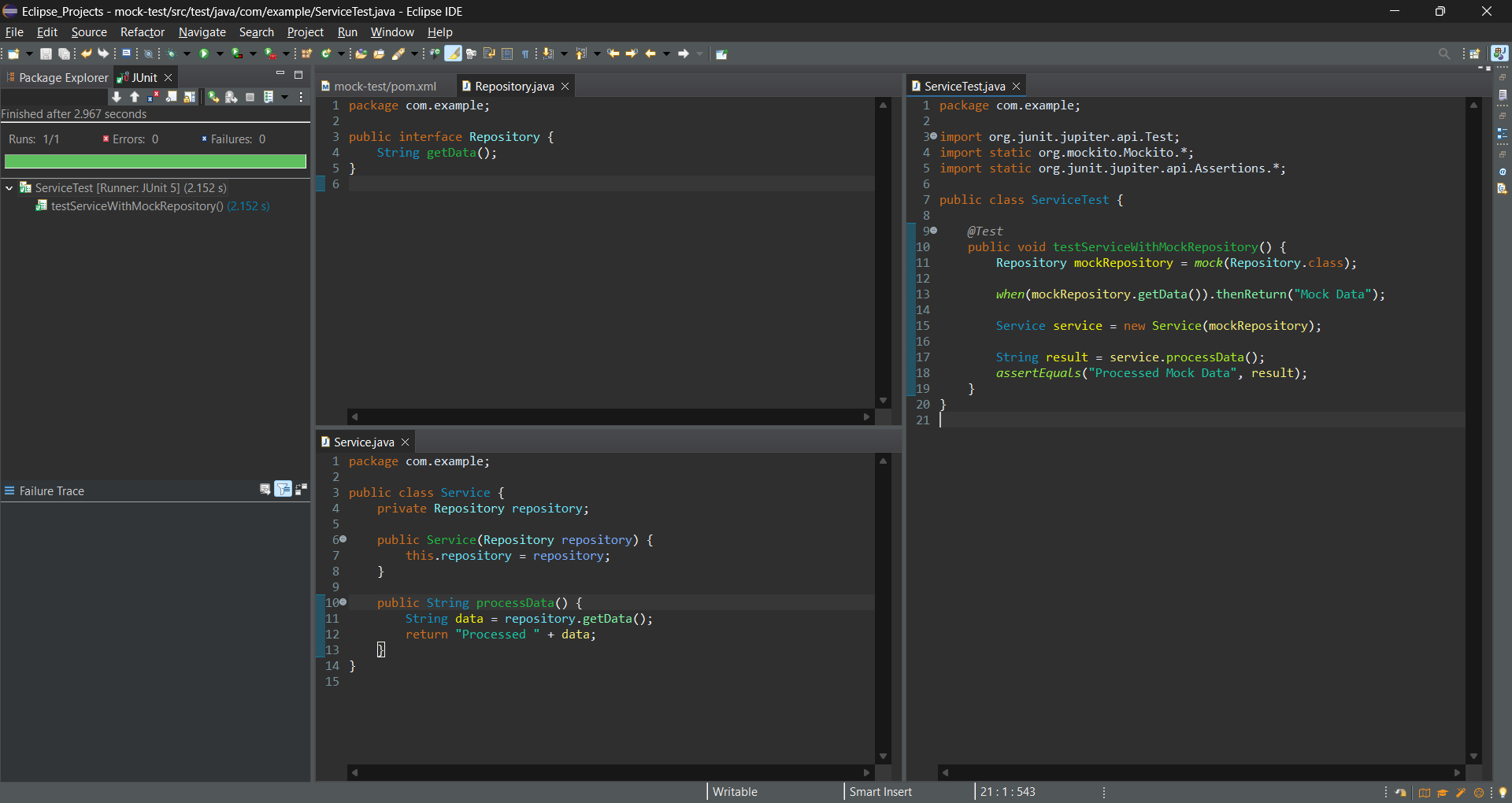
String result = service.processData();

*assertEquals*("Processed Mock Data", result);

}

}

**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);

}

}

**Program:**

**RestClient.java**

package com.example;

public interface RestClient {

String getResponse();

}

**ApiService.java**

package com.example;

public class ApiService {

private RestClient restClient;

public ApiService(RestClient restClient) {

this.restClient = restClient;

}

public String fetchData() {

String response = restClient.getResponse();

return "Fetched " + response;

}

}

**ApiServiceTest.java**

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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);

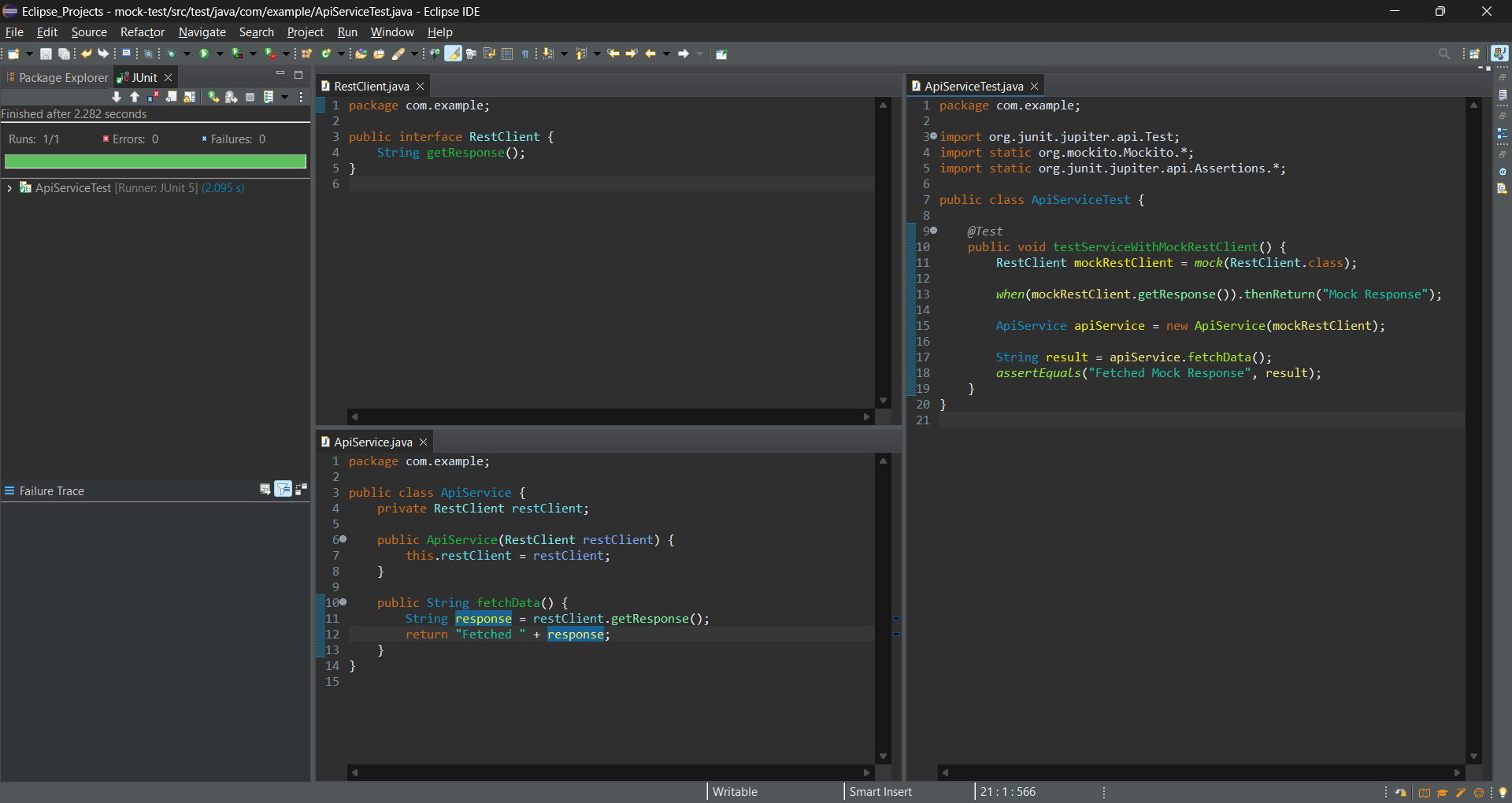
String result = apiService.fetchData();

*assertEquals*("Fetched Mock Response", result);

}

}

**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);

}

}

**Program:**

**FileReader.java**

package com.example;

public interface FileReader {

String read();

}

**FileWriter.java**

package com.example;

public interface FileWriter {

void write(String data);

}

**FileService.java**

package com.example;

public class FileService {

private FileReader fileReader;

private FileWriter fileWriter;

public FileService(FileReader fileReader, FileWriter fileWriter) {

this.fileReader = fileReader;

this.fileWriter = fileWriter;

}

public String processFile() {

String content = fileReader.read();

String processed = "Processed " + content;

fileWriter.write(processed);

return processed;

}

}

**FileServiceTest.java**

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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();

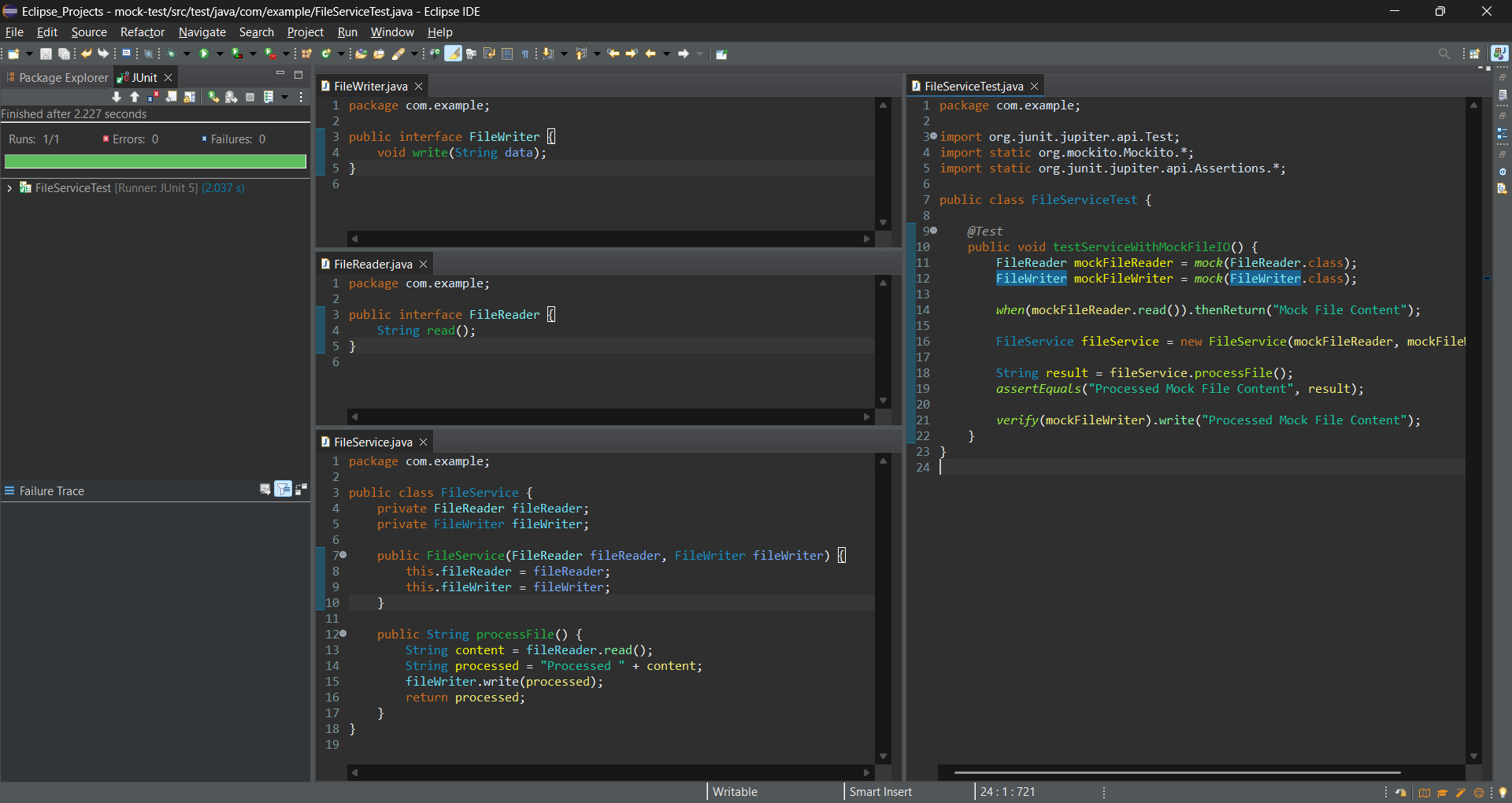
*assertEquals*("Processed Mock File Content", result);

*verify*(mockFileWriter).write("Processed Mock File Content");

}

}

**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);

}

}

**Program:**

**NetworkClient.java**

package com.example;

public interface NetworkClient {

String connect();

}

**NetworkService.java**

package com.example;

public class NetworkService {

private NetworkClient networkClient;

public NetworkService(NetworkClient networkClient) {

this.networkClient = networkClient;

}

public String connectToServer() {

String connection = networkClient.connect();

return "Connected to " + connection;

}

}

**NetworkServiceTest.java**

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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);

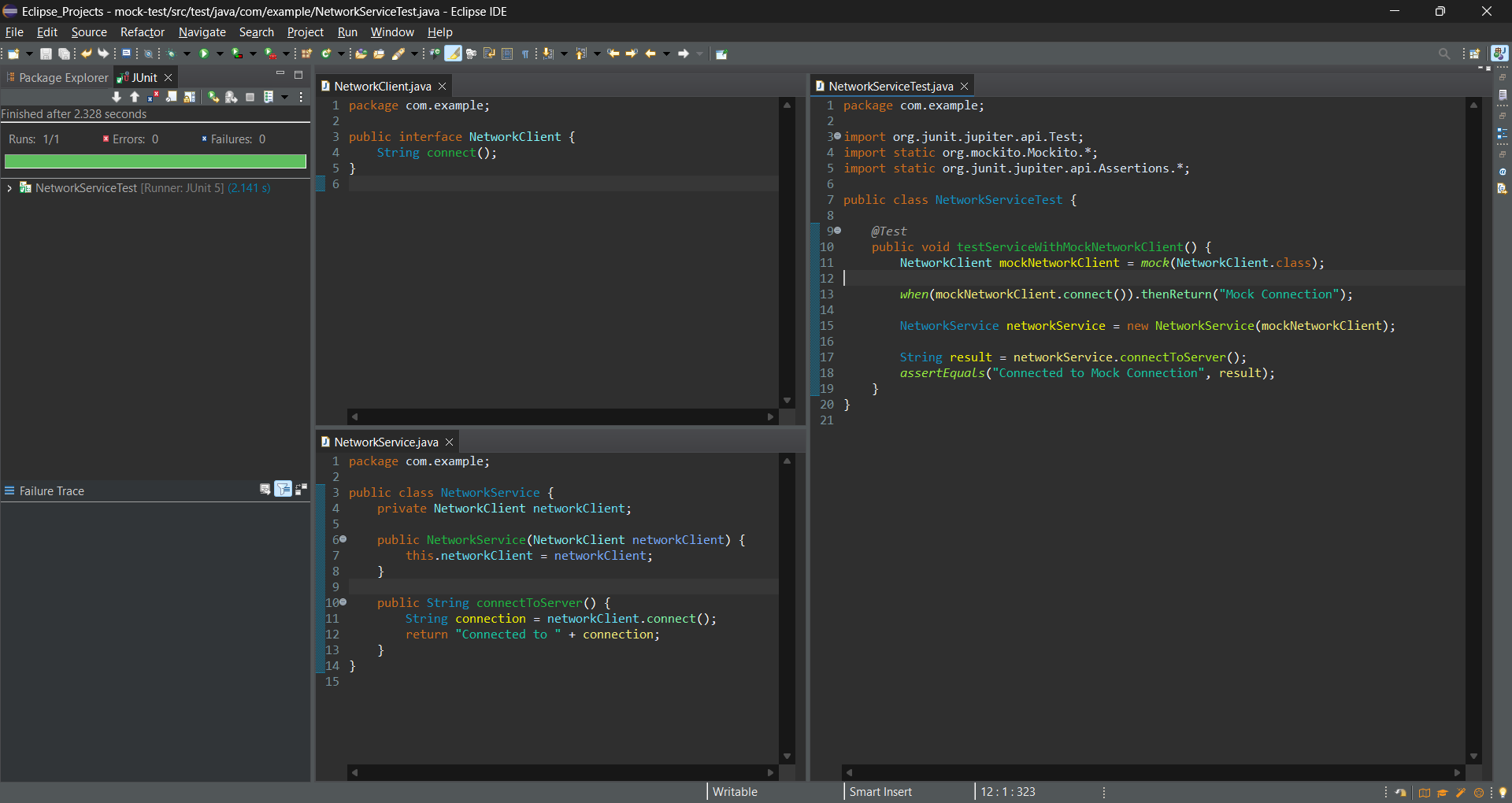
String result = networkService.connectToServer();

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

}

}

**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);

}

}

**Program:**

**Repository.java**

package com.example;

public interface Repository {

String getData();

}

**Service.java**

package com.example;

public class Service {

private Repository repository;

public Service(Repository repository) {

this.repository = repository;

}

public String processData() {

return "Processed " + repository.getData();

}

}

**MultiReturnServiceTest.java**

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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);

}

}

**Output:**

