COGNIZANT

Digital Nurture 4.0

Deep Skilling - Java FSE

WEEK-2 HANDS ON

By Kaviya P

**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

**Repository.java**

**package** org.sample.my\_first\_maven;

**public** **interface** Repository {

String getData();

}

**Service.java**

**package** org.sample.my\_first\_maven;

**public** **class** Service {

**private** **final** Repository repository;

**public** Service(Repository repository) {

**this**.repository = repository;

}

**public** String processData() {

**return** "Processed " + repository.getData();

}

}

**ServiceTest.java**

**package** org.sample.my\_first\_maven;

**import** **static** org.junit.Assert.*assertEquals*;

**import** **static** org.mockito.Mockito.*mock*;

**import** **static** org.mockito.Mockito.*when*;

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

**public** **class** ServiceTest {

@Test

**public** **void** testServiceWithMockRepository() {

Repository mockRepository = *mock*(Repository.**class**);

*when*(mockRepository.getData()).thenReturn("Mock Data");

Service service = **new** Service(mockRepository);

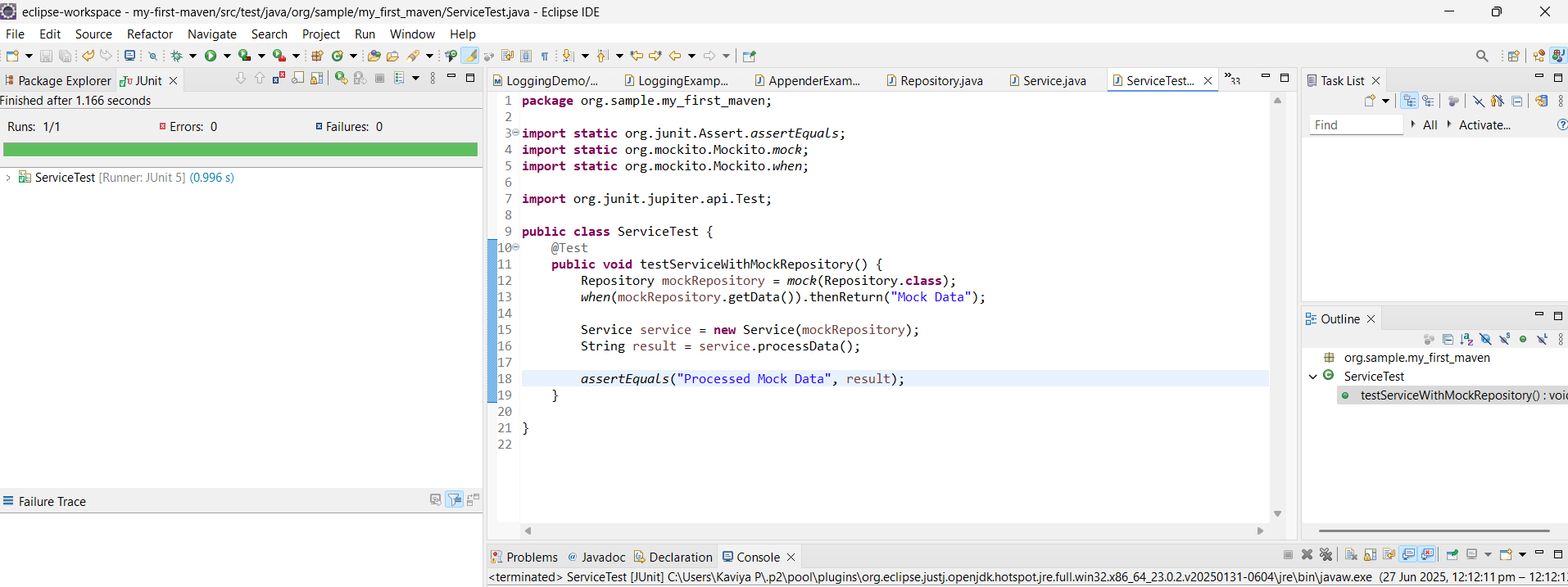
String result = service.processData();

*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.

**RestClient.java**

**package** org.sample.my\_first\_maven;

**public** **interface** RestClient {

String getResponse();

}

**ApiService.java**

**package** org.sample.my\_first\_maven;

**public** **class** ApiService {

**private** **final** RestClient restClient;

**public** ApiService(RestClient restClient) {

**this**.restClient = restClient;

}

**public** String fetchData() {

**return** "Fetched " + restClient.getResponse();

}

}

**ApiServiceTest.java**

**package** org.sample.my\_first\_maven;

**import** **static** org.junit.Assert.*assertEquals*;

**import** **static** org.mockito.Mockito.*mock*;

**import** **static** org.mockito.Mockito.*when*;

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

**public** **class** ApiServiceTest {

@Test

**public** **void** testServiceWithMockRestClient() {

RestClient mockRestClient = *mock*(RestClient.**class**);

*when*(mockRestClient.getResponse()).thenReturn("Mock Response");

ApiService apiService = **new** ApiService(mockRestClient);

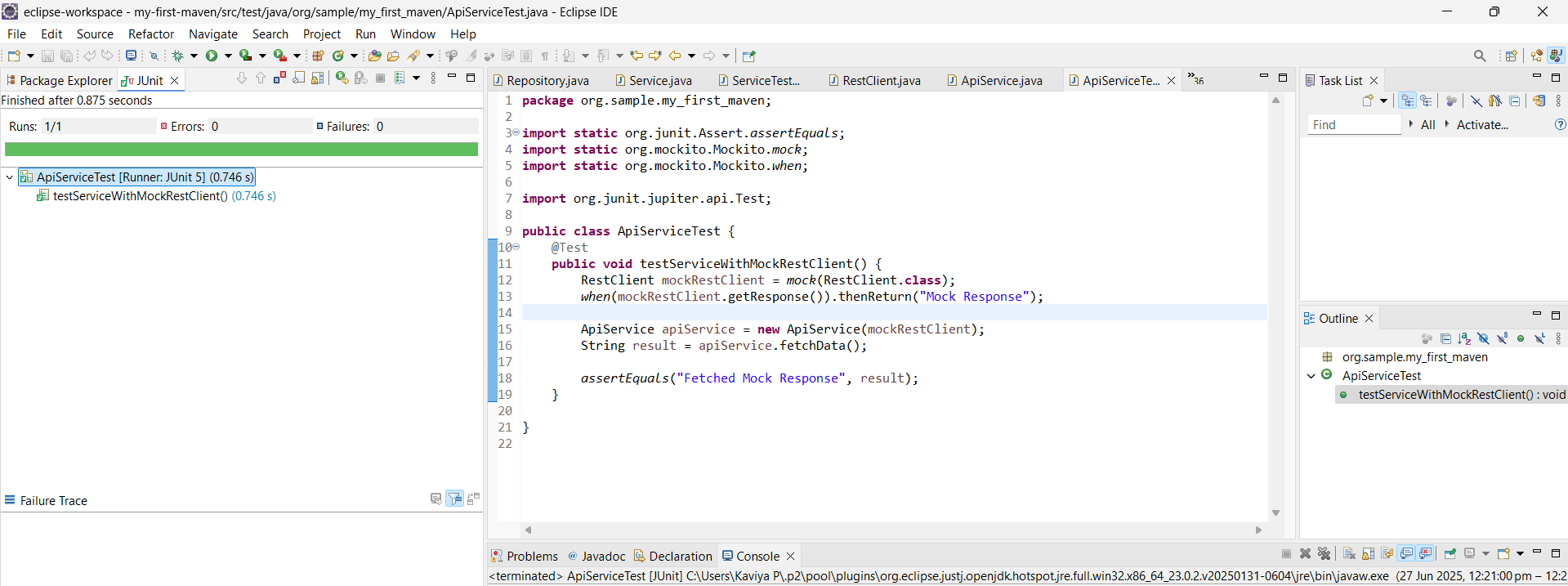
String result = apiService.fetchData();

*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.

**FileReader.java**

**package** org.sample.my\_first\_maven;

**public** **interface** FileReader {

String read();

}

**FileWriter.java**

**package** org.sample.my\_first\_maven;

**public** **interface** FileWriter {

**void** write(String data);

}

**FileService.java**

**package** org.sample.my\_first\_maven;

**public** **class** FileService {

**private** **final** FileReader fileReader;

**private** **final** FileWriter fileWriter;

**public** FileService(FileReader fileReader, FileWriter fileWriter) {

**this**.fileReader = fileReader;

**this**.fileWriter = fileWriter;

}

**public** String processFile() {

String content = fileReader.read();

fileWriter.write("Processed " + content);

**return** "Processed " + content;

}

}

**FileServiceTest.java**

**package** org.sample.my\_first\_maven;

**import** **static** org.junit.Assert.*assertEquals*;

**import** **static** org.mockito.Mockito.*mock*;

**import** **static** org.mockito.Mockito.*verify*;

**import** **static** org.mockito.Mockito.*when*;

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

**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();

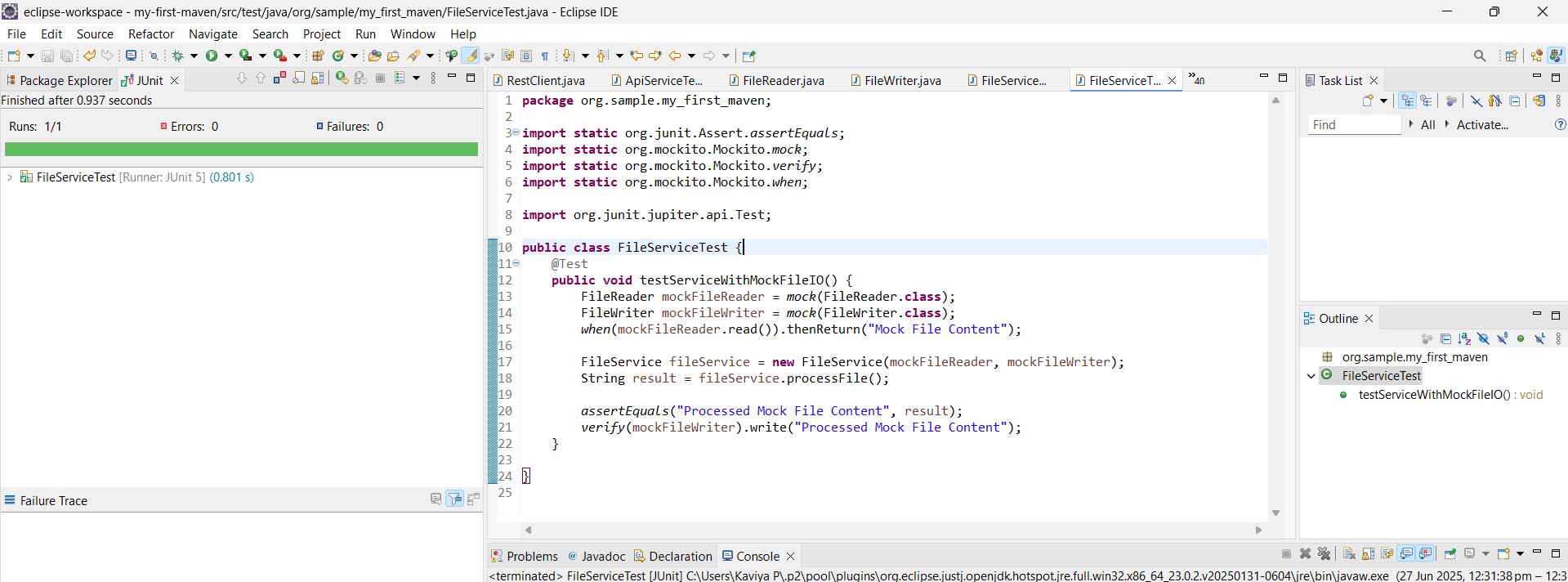
*assertEquals*("Processed Mock File Content", result);

*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.

**NetworkClient.java**

**package** org.sample.my\_first\_maven;

**public** **interface** NetworkClient {

String connect();

}

**NetworkService.java**

**package** org.sample.my\_first\_maven;

**public** **class** NetworkService {

**private** **final** NetworkClient networkClient;

**public** NetworkService(NetworkClient networkClient) {

**this**.networkClient = networkClient;

}

**public** String connectToServer() {

**return** "Connected to " + networkClient.connect();

}

}

**NetworkServiceTest.java**

**package** org.sample.my\_first\_maven;

**import** **static** org.junit.Assert.*assertEquals*;

**import** **static** org.mockito.Mockito.*mock*;

**import** **static** org.mockito.Mockito.*when*;

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

**public** **class** NetworkServiceTest {

@Test

**public** **void** testServiceWithMockNetworkClient() {

NetworkClient mockNetworkClient = *mock*(NetworkClient.**class**);

*when*(mockNetworkClient.connect()).thenReturn("Mock Connection");

NetworkService networkService = **new** NetworkService(mockNetworkClient);

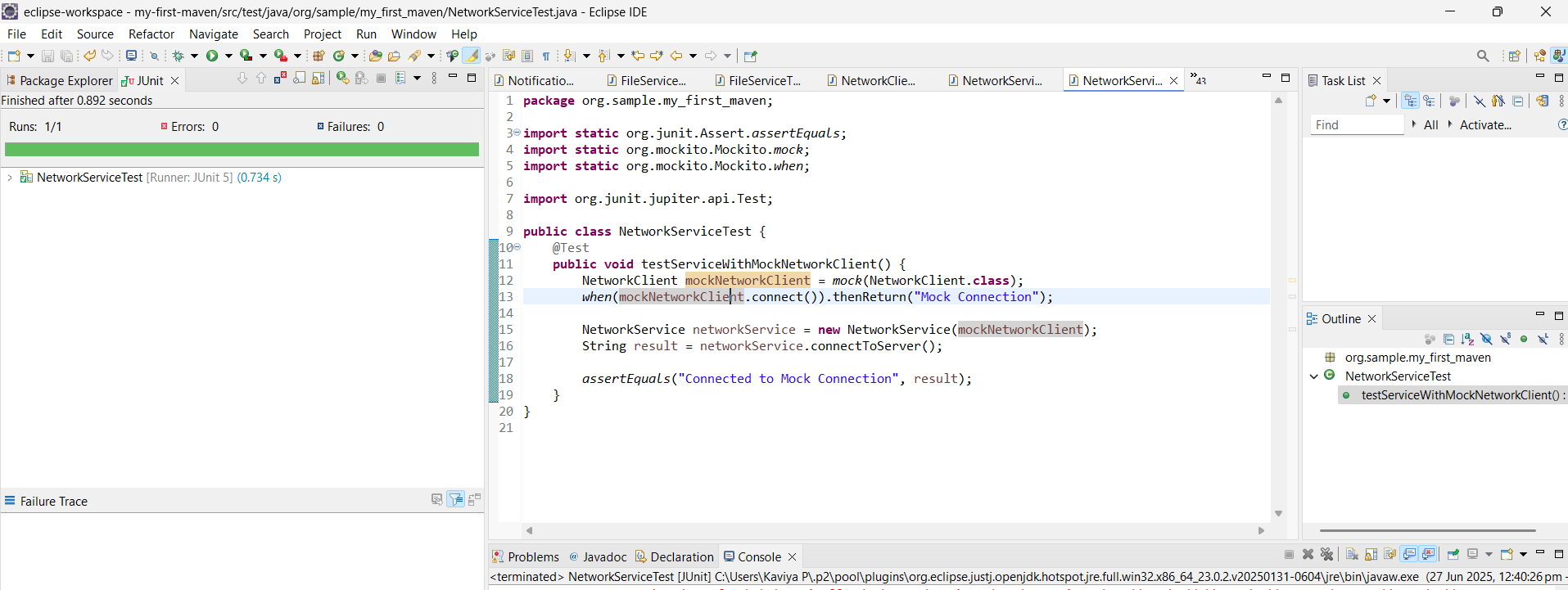
String result = networkService.connectToServer();

*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.

**MultiReturnServiceTest.java**

**package** org.sample.my\_first\_maven;

**import** **static** org.junit.Assert.*assertEquals*;

**import** **static** org.mockito.Mockito.*mock*;

**import** **static** org.mockito.Mockito.*when*;

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

**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 first = service.processData();

String second = service.processData();

*assertEquals*("Processed First Mock Data", first);

*assertEquals*("Processed Second Mock Data", second);

}

}

OUTPUT

