Advanced Mockito Hands-On Exercises Exercise 1: Mocking Databases and Repositories Repository.java

##### package com.example.AdvMockitoEx1;

public interface Repository { String getData();

}

##### Service.java

package com.example.AdvMockitoEx1;

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.AdvMockitoEx1; import static org.mockito.Mockito.\*;

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

public class ServiceTest { @Test

public void testServiceWithMockRepository() {

// Step 1: Create mock Repository

Repository mockRepository = mock(Repository.class);

// Step 2: Stub getData() to return "Mock Data" when(mockRepository.getData()).thenReturn("Mock Data");

// Step 3: Inject mock into Service

Service service = new Service(mockRepository);

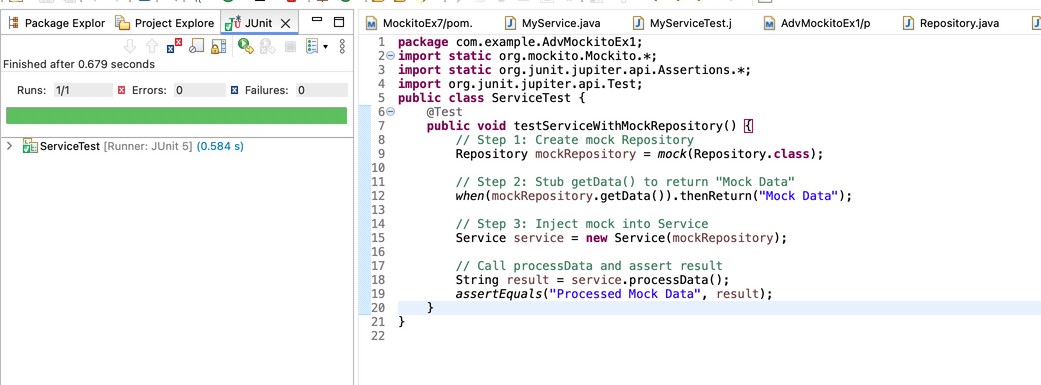
// Call processData and assert result String result = service.processData();

assertEquals("Processed Mock Data", result);

}

}

# output



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

##### ApiService.java

package com.example.AdvMockitoEx2;

public class ApiService {

private flnal RestClient restClient;

public ApiService(RestClient restClient) {

this.restClient = restClient;

}

public String fetchData() {

String response = restClient.getResponse();

return "Fetched " + response;

}

}

#### RestClient.java

package com.example.AdvMockitoEx2;

public interface RestClient { String getResponse();

}

### ApiServiceTest.java

package com.example.AdvMockitoEx2; import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test; public class ApiServiceTest {

@Test

public void testServiceWithMockRestClient() {

// Step 1: Create mock REST client

RestClient mockRestClient = mock(RestClient.class);

// Step 2: Stub method

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

// Step 3: Inject into service

ApiService apiService = new ApiService(mockRestClient);

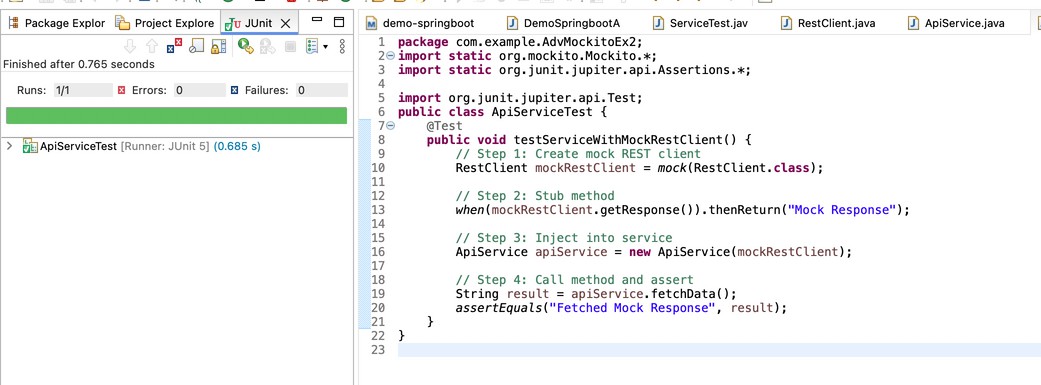
// Step 4: Call method and assert String result = apiService.fetchData();

assertEquals("Fetched Mock Response", result);

}

}

# output



## Exercise 3: Mocking File I/O

#### FileReader.java

package com.example.AdvMockitoEx3;

public interface FileReader { String read();

}

#### FileService .java

package com.example.AdvMockitoEx3;

public class FileService {

private flnal FileReader fileReader;

private flnal 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;

}

}

#### FileWriter .java

package com.example.AdvMockitoEx3;

public class FileWriter {

void write(String content);

}

#### FileServiceTest.java

package com.example.AdvMockitoEx3; 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() {

// Step 1: Mock the interfaces

FileReader mockFileReader = mock(FileReader.class); FileWriter mockFileWriter = mock(FileWriter.class);

// Step 2: Stub the fileReader

when(mockFileReader.read()).thenReturn("Mock File Content");

// Step 3: Inject into FileService

FileService fileService = new FileService(mockFileReader, mockFileWriter);

// Step 4: Run test

String result = fileService.processFile();

// Step 5: Assertions

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

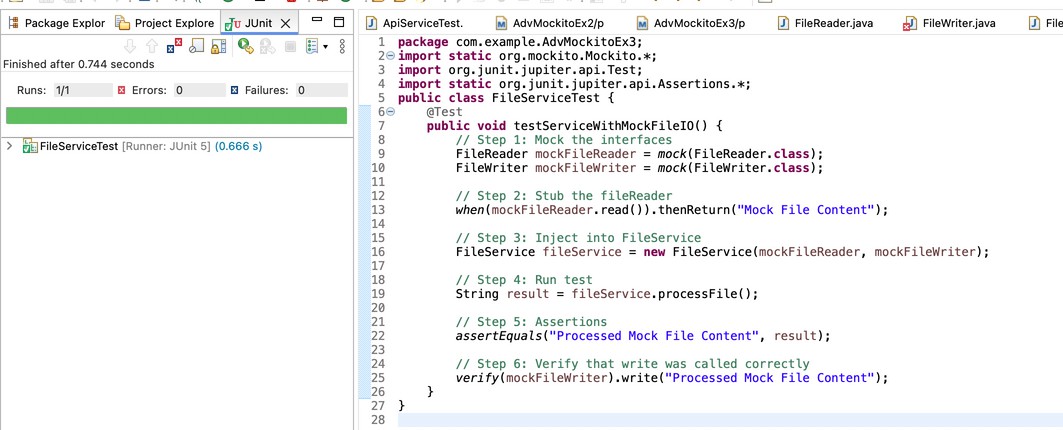
// Step 6: Verify that write was called correctly

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

}

}

# output



Exercise 4: Mocking Network Interactions

##### NetworkClient.java

package com.example.AdvMockitoEx4;

public interface NetworkClient { String connect();

}

#### NetworkService .java

package com.example.AdvMockitoEx4;

public class NetworkService {

private flnal 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.AdvMockitoEx4; import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test; public class NetworkServiceTest {

@Test

public void testServiceWithMockNetworkClient() {

// Step 1: Mock the NetworkClient

NetworkClient mockNetworkClient = mock(NetworkClient.class);

// Step 2: Stub the connect() method

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

// Step 3: Inject mock into NetworkService

NetworkService networkService = new NetworkService(mockNetworkClient);

// Step 4: Call the method and assert

String result = networkService.connectToServer();

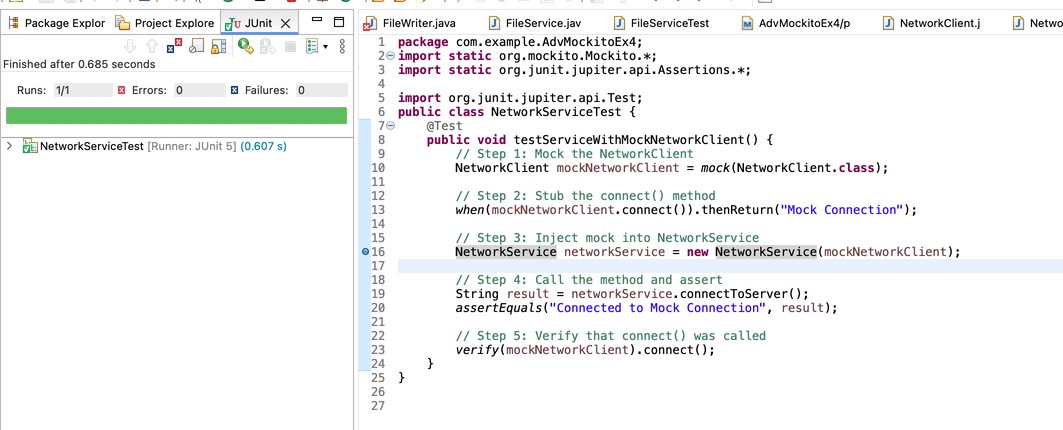
assertEquals("Connected to Mock Connection", result);

// Step 5: Verify that connect() was called verify(mockNetworkClient).connect();

}

}

# output



Exercise 5: Mocking Multiple Return Values

#### Repository.java

package com.example.AdvMockitoEx5;

public interface Repository { String getData();

}

#### Service .java

package com.example.AdvMockitoEx5;

public class Service {

private flnal Repository repository;

public Service(Repository repository) {

this.repository = repository;

}

public String processData() {

return "Processed " + repository.getData();

}

}

#### MultiReturnServiceTest.java

package com.example.AdvMockitoEx5; import static org.mockito.Mockito.\*;

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

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 firstResult = service.processData(); String secondResult = service.processData();

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

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

}

}

output

