***WEEK 2***

***Advanced Mockito Hands-On Exercises***

**Exercise 1: Mocking Databases and Repositories**

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

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

import org.junit.jupiter.api.Test;

public class ServiceTest {

*@Test*

public void testServiceWithMockRepository() {

// 1. Create mock repository

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

// 2. Stub method to return mock data

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

// 3. Inject mock into service

Service service = new Service(mockRepository);

// 4. Call service method

String result = service.processData();

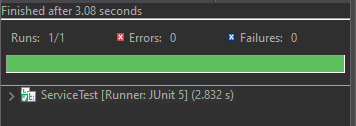
// 5. Assert expected result

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

}

}

**OUTPUT**



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

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

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

import org.junit.jupiter.api.Test;

public class ApiServiceTest {

*@Test*

public void testServiceWithMockRestClient() {

// 1. Create mock REST client

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

// 2. Stub REST method

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

// 3. Inject mock into service

ApiService apiService = new ApiService(mockRestClient);

// 4. Call service method

String result = apiService.fetchData();

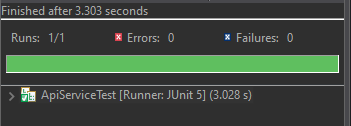
// 5. Assert result

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

}

}

**OUTPUT**

****

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

**FileReader.java**

package com.example;

public interface FileReader {

String read();

}

**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 processedContent = "Processed " + content;

fileWriter.write(processedContent);

return processedContent;

}

}

**FileWriter.java**

package com.example;

public interface FileWriter {

void write(String content);

}

**FileServiceTest.java**

package com.example;

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

public class FileServiceTest {

*@Test*

public void testServiceWithMockFileIO() {

// 1. Create mocks

FileReader mockFileReader = *mock*(FileReader.class);

FileWriter mockFileWriter = *mock*(FileWriter.class);

// 2. Stub read method

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

// 3. Inject mocks into service

FileService fileService = new FileService(mockFileReader, mockFileWriter);

// 4. Call method

String result = fileService.processFile();

// 5. Assert result

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

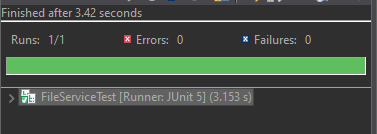
// 6. Verify write was called with correct processed content

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

}

}

**OUTPUT**

****

**Exercise 4: Mocking Network Interactions**

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

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

import org.junit.jupiter.api.Test;

public class NetworkServiceTest {

*@Test*

public void testServiceWithMockNetworkClient() {

// 1. Create mock

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

// 2. Stub the method

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

// 3. Inject mock into service

NetworkService networkService = new NetworkService(mockNetworkClient);

// 4. Call method and assert result

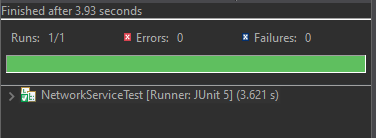
String result = networkService.connectToServer();

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

}

}

**OUTPUT**

****

**Exercise 5: Mocking Multiple Return Values**

**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;

}

}

**MultiReturnServiceTest.java**

package com.example;

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() {

// 1. Create mock

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

// 2. Stub method with multiple return values

*when*(mockRepository.getData())

.thenReturn("First Mock Data")

.thenReturn("Second Mock Data");

// 3. Inject mock into service

Service service = new Service(mockRepository);

// 4. Call processData() twice

String firstResult = service.processData();

String secondResult = service.processData();

// 5. Assertions

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

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

}

}

**OUTPUT**

