Advanced Mockito Hands-On Exercise

Exercise 1: Mocking Databases and Repositories.

Test a service that interacts with a database repository.

Code :

Service.java

//Repository interface

**interface** Repository {

String getData();

}

//Service class that depends on Repository

**class** Service {

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

**public** Service(Repository repository) {

**this**.repository = repository;

}

**public** String processData() {

String data = repository.getData();

**return** "Processed " + data;

}

}

ServiceTest.java

//Complete test class

**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() {

// Create a mock repository using Mockito

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

// Stub the repository method to return predefined data

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

// Create service with mocked repository

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

// Test the service logic

String result = service.processData();

// Verify the result

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

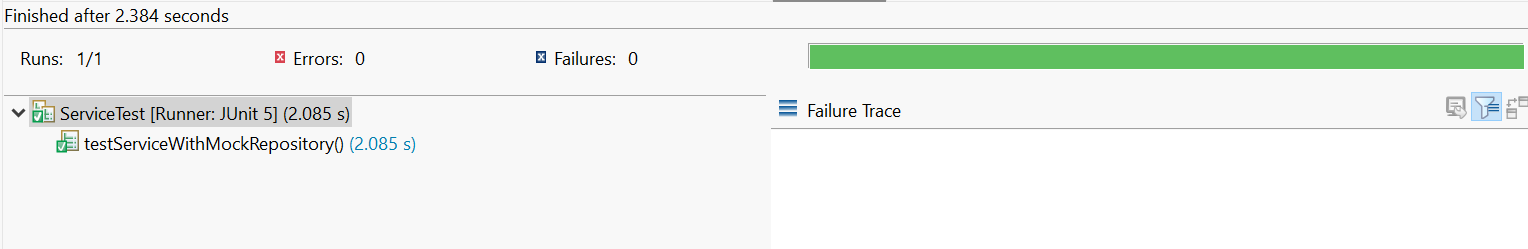
// Verify that getData() was called exactly once

*verify*(mockRepository, *times*(1)).getData();

}

}

Output :



Exercise 2: Mocking External Services (RESTful APIs).

Test a service that calls an external RESTful API.

Code :

ApiService.java

**package** mockprogram;

**interface** RestClient {

String getResponse();

}

**class** ApiService {

**private** **final** RestClient restclient;

**public** ApiService(RestClient restclient) {

**this**.restclient = restclient;

}

**public** String fetchData() {

String data = restclient.getResponse();

**return** "Fetched " + data;

}

}

ApiServiceTest.java

**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);

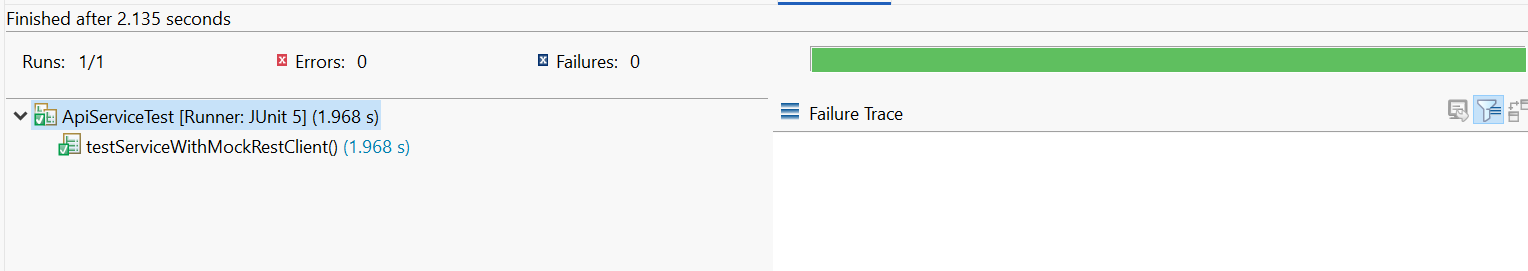
String result = apiService.fetchData();

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

}

}

Output :



Exercise 3: Mocking File I/O.

Test a service that reads from and writes to files.

Code :

FileService.java

**package** mockprogram;

**interface** FileReader {

String read();

}

**interface** FileWriter {

String write();

}

**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 data = filereader.read();

**return** "Processed mock " + data;

}

}

FileServiceTest.java

**package** mockprogram;

**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);

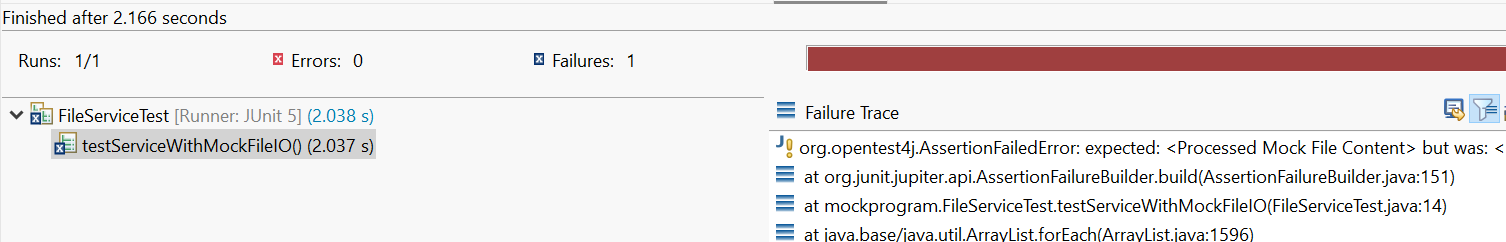
String result = fileService.processFile();

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

}

}

Output :



Exercise 4: Mocking Network Interactions.

Test a service that interacts with network resources.

Code :

NetworkService.java

**package** mockprogram;

**interface** NetworkClient {

String connect();

}

// NetworkService class that depends on NetworkClient

**class** NetworkService {

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

**public** NetworkService(NetworkClient networkClient) {

**this**.networkClient = networkClient;

}

**public** String connectToServer() {

String connection = networkClient.connect();

**return** "Connected to " + connection;

}

}

NetworkTest.java

**package** mockprogram;

**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);

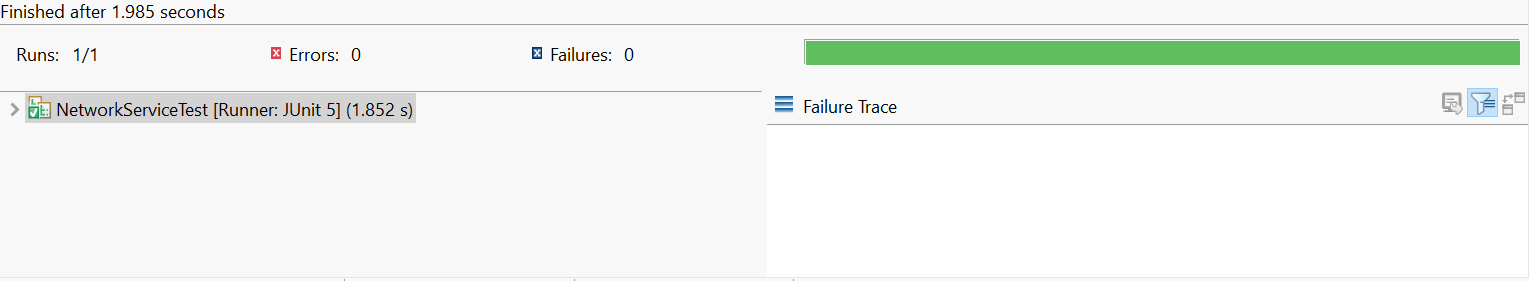
String result = networkService.connectToServer();

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

}

}

Output :



Exercise 5: Mocking Multiple Return Values.

Test a service that calls a method multiple times with different return values

Code :

Service.java

**package** mockprogram;

//NetworkClient interface

**interface** Repository {

String getData();

}

//NetworkService class that depends on NetworkClient

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

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

**public** Service (Repository repository) {

**this**.repository = repository;

}

**public** String processData() {

String data = repository.getData();

**return** "Processed " + data;

}

}

MultiReturnServiceTest.java

**package** mockprogram;

**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);

}

}

Output :

