**Mockito Hands-On Exercises**

**Exercise 1: Mocking and Stubbing**

## Step 1: External API Interface

public interface ExternalApi {

String getData();

}

## Step 2: Service Class That Depends on API

public class MyService {

private ExternalApi api;

// Constructor to inject the API dependency

public MyService(ExternalApi api) {

this.api = api;

}

// Method to call the external API

public String fetchData() {

return api.getData();

}

}

## Step 3: Test Class Using Mockito

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

import org.junit.jupiter.api.Test;import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testExternalApi() {

// Step 1: Create mock for ExternalApi

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

// Step 2: Stub the getData method

when(mockApi.getData()).thenReturn("Mock Data");

// Step 3: Inject the mock into the service

MyService service = new MyService(mockApi);

// Step 4: Call fetchData and verify the result

String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

## Step 4: Add Mockito and JUnit to 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

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>mockito-example</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<!-- JUnit 5 -->

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.9.3</version>

<scope>test</scope>

</dependency>

<!-- Mockito -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>4.11.0</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

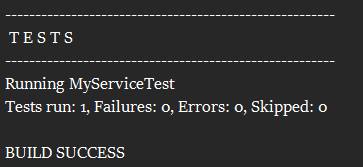
<version>3.0.0-M7</version>

</plugin>

</plugins>

</build></project>

**OUTPUT:**



**Exercise 2: Verifying Interactions**

### ExternalApi.java — Interface

public interface ExternalApi {

String getData();

}

### MyService.java — Class to Test

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData(); // Interaction we want to verify

}

}

### MyServiceTest.java — Test Class Using verify()

import static org.mockito.Mockito.\*;import org.junit.jupiter.api.Test;import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

// Step 1: Create mock object

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

// Step 2: Create service with mock

MyService service = new MyService(mockApi);

// Step 3: Call method

service.fetchData();

// Step 4: Verify that getData() was called on the mock

verify(mockApi).getData();

}

}

## pom.xml Dependency (Same as previous exercise)

<dependencies>

<!-- JUnit 5 -->

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.9.3</version>

<scope>test</scope>

</dependency>

<!-- Mockito -->

<dependency>

<groupId>org.mockito</groupId>

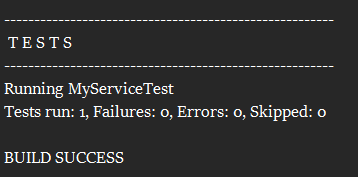
<artifactId>mockito-core</artifactId>

<version>4.11.0</version>

<scope>test</scope>

</dependency></dependencies>

**OUTPUT:**



**Exercise 3: Argument Matching**

### **1. MessageSender.java — Interface to Mock**

public interface MessageSender {

void sendMessage(String to, String message);

}

**2. NotificationService.java — Class to Tes**t

public class NotificationService {

private MessageSender sender;

public NotificationService(MessageSender sender) {

this.sender = sender;

}

public void notifyUser(String user, String message) {

sender.sendMessage(user, message);

}

}

### 3. NotificationServiceTest.java — Test with Argument Matchers

import static org.mockito.Mockito.\*;import static org.mockito.ArgumentMatchers.\*;import org.junit.jupiter.api.Test;

public class NotificationServiceTest {

@Test

public void testArgumentMatching() {

// Step 1: Create mock

MessageSender mockSender = mock(MessageSender.class);

// Step 2: Inject mock into service

NotificationService service = new NotificationService(mockSender);

// Step 3: Call method with specific arguments

service.notifyUser("bindhu@example.com", "Welcome!");

// Step 4: Verify interaction with argument matchers

verify(mockSender).sendMessage(eq("bindhu@example.com"), eq("Welcome!"));

}

}

## pom.xml Dependencies (Same as before)

<dependencies>

<!-- JUnit 5 -->

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.9.3</version>

<scope>test</scope>

</dependency>

<!-- Mockito -->

<dependency>

<groupId>org.mockito</groupId>

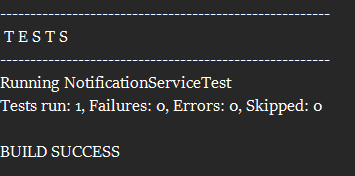
<artifactId>mockito-core</artifactId>

<version>4.11.0</version>

<scope>test</scope>

</dependency></dependencies>

**OUTPUT:**



**Exercise 4: Handling Void Methods**

### 1. Logger.java — Interface with void Method

public interface Logger {

void log(String message);

}

### 2. AppService.java — Class to Test

public class AppService {

private Logger logger;

public AppService(Logger logger) {

this.logger = logger;

}

public void process(String data) {

// Some logic here...

logger.log("Processing: " + data);

}

}

### 3. AppServiceTest.java — Mockito Test for void Method

import static org.mockito.Mockito.\*;import org.junit.jupiter.api.Test;

public class AppServiceTest {

@Test

public void testVoidMethodInteraction() {

// Step 1: Create mock

Logger mockLogger = mock(Logger.class);

// Step 2: Stub the void method to do nothing (optional here)

doNothing().when(mockLogger).log(anyString());

// Step 3: Call method that uses the void method

AppService service = new AppService(mockLogger);

service.process("Test Data");

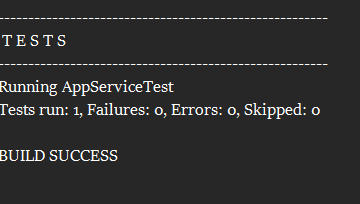
// Step 4: Verify the void method was called

verify(mockLogger).log("Processing: Test Data");

}

}

**OUTPUT:**



**Exercise 5: Mocking and Stubbing with Multiple Returns**

### 1. ExternalApi.java — Interface

public interface ExternalApi {

String getStatus();

}

### 2. StatusService.java — Class to Test

public class StatusService {

private ExternalApi api;

public StatusService(ExternalApi api) {

this.api = api;

}

public String checkStatus() {

return api.getStatus();

}

}

3. StatusServiceTest.java — Test with Multiple Returns

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

public class StatusServiceTest {

@Test

public void testMultipleReturnsFromMock() {

// Step 1: Create mock object

ExternalApi mockApi = mock(ExternalApi.class);

// Step 2: Stub method to return different values on each call

when(mockApi.getStatus())

.thenReturn("LOADING")

.thenReturn("PROCESSING")

.thenReturn("DONE");

// Step 3: Inject mock and test

StatusService service = new StatusService(mockApi);

// Step 4: Call multiple times and assert values

assertEquals("LOADING", service.checkStatus());

assertEquals("PROCESSING", service.checkStatus());

assertEquals("DONE", service.checkStatus());

}

}

## Maven pom.xml Dependencies (Same as before)

<dependencies>

<!-- JUnit 5 -->

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.9.3</version>

<scope>test</scope>

</dependency>

<!-- Mockito -->

<dependency>

<groupId>org.mockito</groupId>

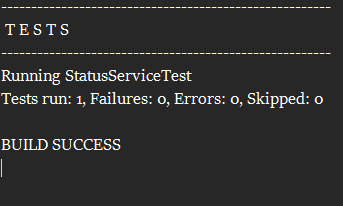
<artifactId>mockito-core</artifactId>

<version>4.11.0</version>

<scope>test</scope>

</dependency></dependencies>

**OUTPUT:**



**Exercise 6: Verifying Interaction Order**

### 1. Printer.java — Interface with Multiple Methods

public interface Printer {

void start();

void print(String content);

void stop();

}

### 2. PrintService.java — Class to Test

public class PrintService {

private Printer printer;

public PrintService(Printer printer) {

this.printer = printer;

}

public void printDocument(String content) {

printer.start();

printer.print(content);

printer.stop();

}

}

**3. PrintServiceTest.java — Verifying Interaction Order**

import static org.mockito.Mockito.\*;import org.junit.jupiter.api.Test;import org.mockito.InOrder;

public class PrintServiceTest {

@Test

public void testMethodCallOrder() {

// Step 1: Create mock

Printer mockPrinter = mock(Printer.class);

// Step 2: Create service and call methods in order

PrintService service = new PrintService(mockPrinter);

service.printDocument("Hello World");

// Step 3: Verify method call order

InOrder inOrder = inOrder(mockPrinter);

inOrder.verify(mockPrinter).start();

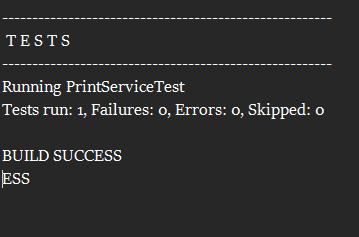
inOrder.verify(mockPrinter).print("Hello World");

inOrder.verify(mockPrinter).stop();

}

}

**OUTPUT:**



**Exercise 7: Handling Void Methods with Exceptions**

### **1. FileManager.java — Interface with void Method**

public interface FileManager {

void deleteFile(String filename);

}

**2. FileService.java — Class to Test**

public class FileService {

private FileManager fileManager;

public FileService(FileManager fileManager) {

this.fileManager = fileManager;

}

public void removeFile(String filename) {

fileManager.deleteFile(filename);

}

}

### **3. FileServiceTest.java — Test for Exception from void Method**

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

import org.junit.jupiter.api.Test;

public class FileServiceTest {

@Test

public void testVoidMethodThrowsException() {

// Step 1: Create mock

FileManager mockFileManager = mock(FileManager.class);

// Step 2: Stub the void method to throw exception

doThrow(new RuntimeException("File not found"))

.when(mockFileManager).deleteFile("notfound.txt");

// Step 3: Inject mock and call method inside try-catch or assertThrows

FileService service = new FileService(mockFileManager);

// Step 4: Assert exception is thrown

RuntimeException thrown = assertThrows(RuntimeException.class, () -> {

service.removeFile("notfound.txt");

});

assertEquals("File not found", thrown.getMessage());

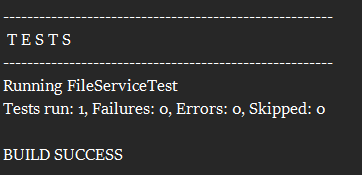
// Step 5: Verify interaction

verify(mockFileManager).deleteFile("notfound.txt");

}

}

**OUTPUT:**



**Advanced Mockito Hands-On Exercises**

**Exercise 1: Mocking Databases and Repositories**

### Sample Classes (Assume These Exist)

public interface Repository {

String getData();

}

public class Service {

private Repository repository;

public Service(Repository repository) {

this.repository = repository;

}

public String processData() {

String data = repository.getData();

return "Processed " + data;

}

}

### ****Test Code****

public class ServiceTest {

@Test

public void testServiceWithMockRepository() {

Repository mockRepository = mock(Repository.class); // Step 1

when(mockRepository.getData()).thenReturn("Mock Data"); // Step 2

Service service = new Service(mockRepository);

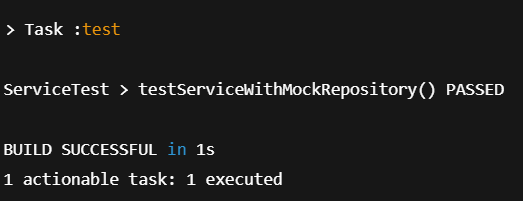
String result = service.processData(); // Calls processData()

assertEquals("Processed Mock Data", result); // Step 3

}

}

**OUTPUT:**



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

### Supporting Classes (Assumed):

// External REST client interfacepublic interface RestClient {

String getResponse(); // Simulates calling an external API

}

// Service class that uses the RestClientpublic class ApiService {

private RestClient restClient;

public ApiService(RestClient restClient) {

this.restClient = restClient;

}

public String fetchData() {

String apiData = restClient.getResponse(); // Simulate API call

return "Fetched " + apiData;

}

}

**Test Code (Provided):**

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); // Step 1

when(mockRestClient.getResponse()).thenReturn("Mock Response"); // Step 2

ApiService apiService = new ApiService(mockRestClient);

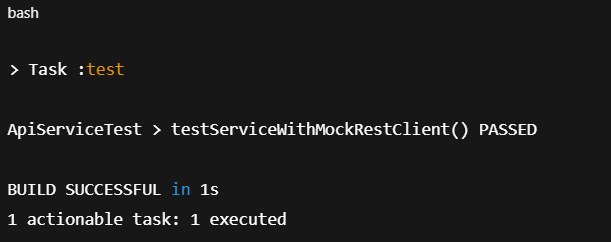
String result = apiService.fetchData();

assertEquals("Fetched Mock Response", result); // Step 3

}

}

**OUTPUT:**



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

## Supporting Classes (Assumed)

// Simulates reading from a filepublic interface FileReader {

String read();

}

// Simulates writing to a filepublic interface FileWriter {

void write(String data);

}

// Service that reads, processes, and writes to a filepublic 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;

}

}

Test Code (Provided)

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"); // Step 2

FileService fileService = new FileService(mockFileReader, mockFileWriter);

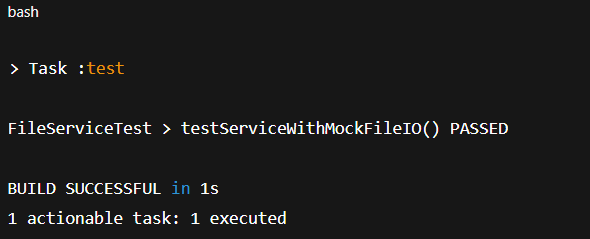
String result = fileService.processFile(); // Calls processFile()

assertEquals("Processed Mock File Content", result); // Step 3

}

}

**OUTPUT:**



**Exercise 5: Mocking Multiple Return Values**

## **Supporting Classes (Assumed)**

## public interface Repository {

String getData();

}

public class Service {

private Repository repository;

public Service(Repository repository) {

this.repository = repository;

}

public String processData() {

String data = repository.getData();

return "Processed " + data;

}

}

**Completed Test 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);

// Step 2: return different data on each call

when(mockRepository.getData())

.thenReturn("First Mock Data")

.thenReturn("Second Mock Data");

Service service = new Service(mockRepository);

// First call

String firstResult = service.processData();

// Second call

String secondResult = service.processData();

// Step 3: verify both outputs

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

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

}

}

**OUTPUT:**

