**Mockito Hands-On Exercises (Contains both Mandatory and Additional Exercises)**

**Exercise 1: Mocking and Stubbing (Mandatory Exercise)**

**Scenario:**

**You need to test a service that depends on an external API. Use Mockito to mock the external API and stub its methods.**

**Steps:**

1. **Create a mock object for the external API.**
2. **Stub the methods to return predefined values.**
3. **Write a test case that uses the mock object.**

**MyServiceTest.java (Given Solution Code)**

package E1\_MockingAndStubbing;

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

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

*@Test*

public void testExternalApi() {

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

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

MyService service = new MyService(mockApi);

String result = service.fetchData();

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

}

}

**ExternalApi.java**

package E1\_MockingAndStubbing;

public interface ExternalApi {

String getData();

}

**MyService.java**

package E1\_MockingAndStubbing;

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

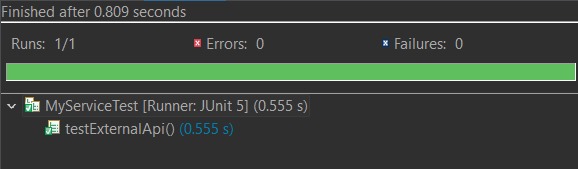
public String fetchData() {

return api.getData();

}

}

**Output:**



**Exercise 2: Verifying Interactions (Mandatory Exercise)**

**Scenario:**

**You need to ensure that a method is called with specific arguments.**

**Steps:**

1. **Create a mock object.**
2. **Call the method with specific arguments.**
3. **Verify the interaction.**

**MyServiceTest.java (Given Solution Code)**

package E2\_VerifyingInteractions;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

*@Test*

public void testVerifyInteraction() {

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

MyService service = new MyService(mockApi);

service.fetchData();

*verify*(mockApi).getData();

}

}

**ExternalApi.java**

package E2\_VerifyingInteractions;

public interface ExternalApi {

String getData();

}

**MyService.java**

package E2\_VerifyingInteractions;

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

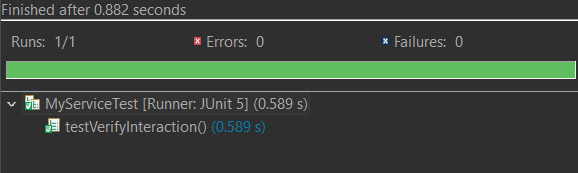
public String fetchData() {

return api.getData();

}

}

**Output:**



**Exercise 3: Argument Matching (Additional Exercise)**

**Scenario:**

**You need to verify that a method is called with specific arguments.**

**Steps:**

1. **Create a mock object.**
2. **Call the method with specific arguments.**
3. **Use argument matchers to verify the interaction.**

**MyServiceTest.java**

package E3\_ArgumentMatching;

import static org.mockito.Mockito.\*;

import static org.mockito.ArgumentMatchers.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

*@Test*

public void testArgumentMatching() {

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

MyService service = new MyService(mockApi);

service.updateData();

*verify*(mockApi).sendData(*eq*("user123"), *eq*(42));

}

}

**ExternalApi.java**

package E3\_ArgumentMatching;

public interface ExternalApi {

void sendData(String key, int value);

}

**MyService.java**

package E3\_ArgumentMatching;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

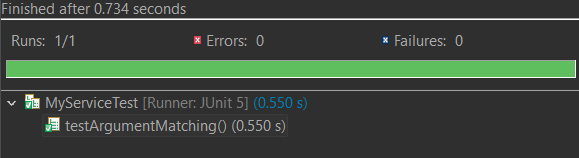
public void updateData() {

api.sendData("user123", 42);

}

}

**Output:**



**Exercise 4: Handling Void Methods (Additional Exercise)**

**Scenario:**

**You need to test a void method that performs some action.**

**Steps:**

1. **Create a mock object.**
2. **Stub the void method.**
3. **Verify the interaction.**

**MyServiceTest.java**

package E4\_HandlingVoidMethods;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

*@Test*

public void testVoidMethod() {

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

*doNothing*().when(mockApi).logEvent(*anyString*());

MyService service = new MyService(mockApi);

service.process();

*verify*(mockApi).logEvent("Process started");

}

}

**ExternalApi.java**

package E4\_HandlingVoidMethods;

public interface ExternalApi {

void logEvent(String message);

}

**MyService.java**

package E4\_HandlingVoidMethods;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

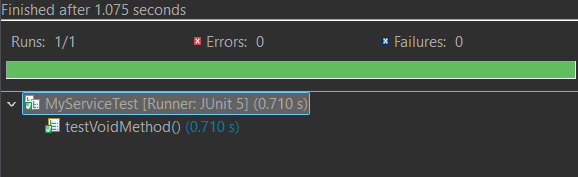
public void process() {

api.logEvent("Process started");

}

}

**Output:**



**Exercise 5: Mocking and Stubbing with Multiple Returns (Additional Exercise)**

**Scenario:**

**You need to test a service that depends on an external API with multiple return values.**

**Steps:**

1. **Create a mock object for the external API.**
2. **Stub the methods to return different values on consecutive calls.**
3. **Write a test case that uses the mock object.**

**MyServiceTest.java**

package E5\_MockingAndStubbingMultipleRuns;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

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

public class MyServiceTest {

*@Test*

public void testMultipleReturns() {

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

*when*(mockApi.getStatus())

.thenReturn("Pending")

.thenReturn("Processing")

.thenReturn("Completed");

MyService service = new MyService(mockApi);

String[] result = service.checkStatusMultipleTimes();

*assertArrayEquals*(new String[] {"Pending", "Processing", "Completed"}, result);

}

}

**ExternalApi.java**

package E5\_MockingAndStubbingMultipleRuns;

public interface ExternalApi {

String getStatus();

}

**MyService.java**

package E5\_MockingAndStubbingMultipleRuns;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String[] checkStatusMultipleTimes() {

return new String[] {

api.getStatus(),

api.getStatus(),

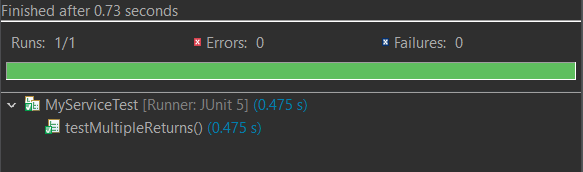
api.getStatus()

};

}

}

**Output:**



**Exercise 6: Verifying Interaction Order (Additional Exercise)**

**Scenario:**

**You need to ensure that methods are called in a specific order.**

**Steps:**

1. **Create a mock object.**
2. **Call the methods in a specific order.**
3. **Verify the interaction order.**

**MyServiceTest.java**

package E6\_VerifyingInteractionsOrder;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.InOrder;

import org.mockito.Mockito;

public class MyServiceTest {

*@Test*

public void testInteractionOrder() {

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

MyService service = new MyService(mockApi);

service.execute();

InOrder inOrder = *inOrder*(mockApi);

inOrder.verify(mockApi).connect();

inOrder.verify(mockApi).fetchData();

inOrder.verify(mockApi).disconnect();

}

}

**ExternalApi.java**

package E6\_VerifyingInteractionsOrder;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void execute() {

api.connect();

api.fetchData();

api.disconnect();

}

}

**MyService.java**

package E6\_VerifyingInteractionsOrder;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void execute() {

api.connect();

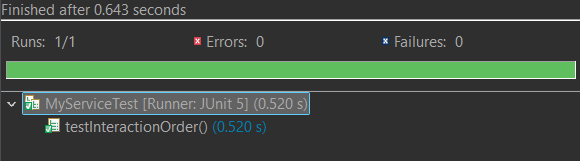
api.fetchData();

api.disconnect();

}

}

**Output:**



**Exercise 7: Handling Void Methods with Exceptions (Additional Exercise)**

**Scenario:**

**You need to test a void method that throws an exception.**

**Steps:**

1. **Create a mock object.**
2. **Stub the void method to throw an exception.**
3. **Verify the interaction.**

**MyServiceTest.java**

package E7\_HandlingVoidMethodsWithException;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

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

public class MyServiceTest {

*@Test*

public void testVoidMethodThrowsException() {

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

*doThrow*(new RuntimeException("Deletion failed")).when(mockApi).deleteData("123");

MyService service = new MyService(mockApi);

*assertThrows*(RuntimeException.class, () -> service.remove("123"));

*verify*(mockApi).deleteData("123");

}

}

**ExternalApi.java**

package E7\_HandlingVoidMethodsWithException;

public interface ExternalApi {

void deleteData(String id);

}

**MyService.java**

package E7\_HandlingVoidMethodsWithException;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void remove(String id) {

api.deleteData(id);

}

}

**Output:**

