**WEEK2**

**Mockito**

NAME : SIDHARTH K

SUPERSET ID : 6430194

**Exercise 1: Parameterized Tests:**

**ExternalApi.java:**

public interface ExternalApi {

String getData();

}

**MyService.java:**

public class MyService {

private final ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData();

}

}

**Output:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Exercise 2: Verifying Interactions:**

**ExternalApi.java:**

A public interface ExternalApi {

String getData();

}

**MyServiceTest.java:**

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

import org.junit.jupiter.api.Test;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

ExternalApi mockApi = mock(ExternalApi.class);

MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData();

}

}

**MyService.java:**

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**Output:**

A screenshot of a computer error

AI-generated content may be incorrect.

**Exercise 3: Argument Matching:**

**ExternalApi.java:**

public interface ExternalApi {

void sendData(String data);

}

**MyService.java:**

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void processAndSend() {

api.sendData("Hello World");

}

}

**MyServiceTest.java:**

import static org.mockito.Mockito.\*;

import static org.mockito.ArgumentMatchers.\*;

import org.junit.jupiter.api.Test;

public class MyServiceTest {

@Test

public void testArgumentMatching() {

ExternalApi mockApi = mock(ExternalApi.class);

MyService service = new MyService(mockApi);

service.processAndSend();

verify(mockApi).sendData(eq("Hello World"));

}

}

**Output:**

A screenshot of a computer error message

AI-generated content may be incorrect.

**Exercise 4: Handling Void Methods:**

**ExternalApi.java:**

public interface ExternalApi {

void sendData(String data);

}

**MyService.java:**

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void process() {

api.sendData("Important Data");

}

}

**MyServiceTest.java:**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

public class MyServiceTest {

@Test

public void testVoidMethod() {

ExternalApi mockApi = mock(ExternalApi.class);

doNothing().when(mockApi).sendData(anyString());

MyService service = new MyService(mockApi);

service.process();

verify(mockApi).sendData(eq("Important Data"));

}

}

**Output:**

A screenshot of a computer error message

AI-generated content may be incorrect.

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

**ExternalApi.java:**

public interface ExternalApi {

String getData();

}

**MyService.java:**

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String[] fetchTwice() {

String first = api.getData();

String second = api.getData();

return new String[]{first, second};

}

}

**MyServiceTest.java:**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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

public class MyServiceTest {

@Test

public void testMultipleReturns() {

ExternalApi mockApi = mock(ExternalApi.class);

when(mockApi.getData())

.thenReturn("First Call")

.thenReturn("Second Call");

MyService service = new MyService(mockApi);

String[] results = service.fetchTwice();

assertEquals("First Call", results[0]);

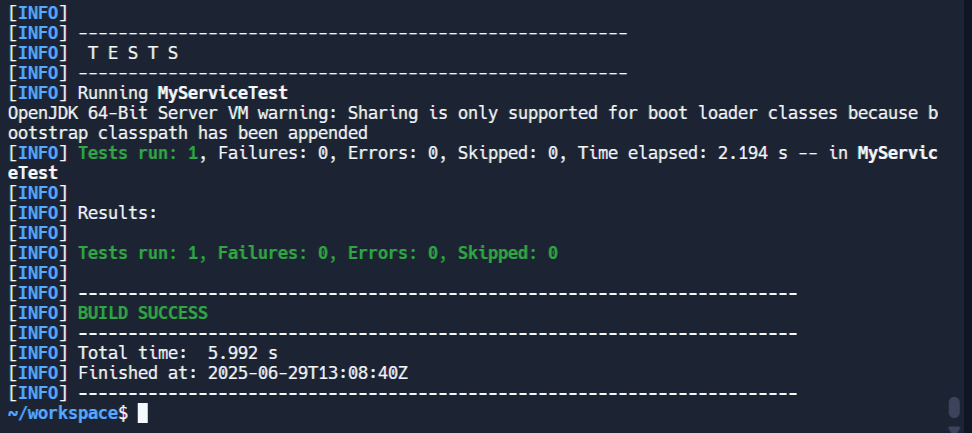
assertEquals("Second Call", results[1]);

verify(mockApi, times(2)).getData();

}

}

**Output:**



**Exercise 6: Verifying Interaction Order:**

**ExternalApi.java:**

public interface ExternalApi {

String getData();

}

**MyService.java:**

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String[] fetchTwice() {

String first = api.getData();

String second = api.getData();

return new String[]{first, second};

}

}

**MyServiceTest.java:**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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

public class MyServiceTest {

@Test

public void testVoidMethodThrowsException() {

ExternalApi mockApi = mock(ExternalApi.class);

doThrow(new RuntimeException("Send failed!"))

.when(mockApi)

.sendData(anyString());

MyService service = new MyService(mockApi);

RuntimeException thrown = assertThrows(

RuntimeException.class,

() -> service.process()

);

assertEquals("Send failed!", thrown.getMessage());

verify(mockApi).sendData(eq("Important Data"));

}

}

**Output:**

A screenshot of a computer error

AI-generated content may be incorrect.

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

**ExternalApi.java:**

public interface ExternalApi {

void sendData(String data);

}**MyService.java:**

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void process() {

api.sendData("Important Data");

}

}

**MyServiceTest.java:**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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

public class MyServiceTest {

@Test

public void testVoidMethodThrowsException() {

ExternalApi mockApi = mock(ExternalApi.class);

doThrow(new RuntimeException("Send failed!"))

.when(mockApi)

.sendData(anyString());

MyService service = new MyService(mockApi);

RuntimeException thrown = assertThrows(

RuntimeException.class,

() -> service.process()

);

assertEquals("Send failed!", thrown.getMessage());

verify(mockApi).sendData(eq("Important Data"));

}

}

**Output:**

A screenshot of a computer error

AI-generated content may be incorrect.