**3. Mockito exercises**

**Exercise 1: Mocking and Stubbing**

**CODE**

***File name: ExternalApi.java***

package com.example.mock;

public interface ExternalApi {

String getData();

}

***File name: MyService.java***

package com.example.mock;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

***File name: MyServiceTest.java***

package com.example.mock;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class MyServiceTest {

@Test

public void testExternalApi() {

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

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

MyService service = new MyService(mockApi);

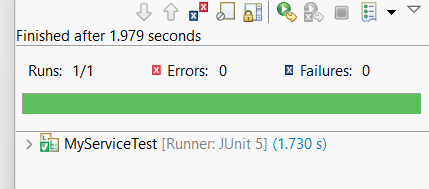
String result = service.fetchData();

*assertEquals*("Mock Data", result); // ✅ Should pass

}

}

**OUTPUT**



**Exercise 2: Verifying Interactions**

**CODE**

***File name: ExternalApi.java***

package com.example.mock;

public interface ExternalApi {

String getData();

}

***File name: MyService.java***

package com.example.mock;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

***File name: MyServiceTest.java***

package com.example.mock;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

// 1. Create mock object

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

// 2. Stub the method (optional for this test)

*when*(mockApi.getData()).thenReturn("Mocked Response");

// 3. Inject into MyService

MyService service = new MyService(mockApi);

// 4. Call the method

String result = service.fetchData();

// 5. Verify interaction

*verify*(mockApi).getData(); // This confirms getData() was called once

// 6. (Optional) Check result

*assertEquals*("Mocked Response", result);

// 7. Print output (for clarity)

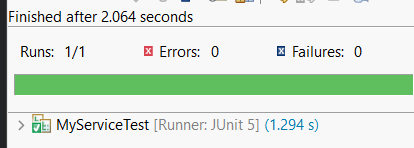
System.*out*.println("Verified: getData() was called, and result = " + result);

}

}

**OUTPUT**





**Exercise 3: Argument Matching**

**CODE**

***File name: UserRepository.java***

package com.example.mock;

public interface UserRepository {

void saveUser(String name);

}

***File name: UserService.java***

package com.example.mock;

public class UserService {

private UserRepository repository;

public UserService(UserRepository repository) {

this.repository = repository;

}

public void register(String name) {

if (name != null && !name.isEmpty()) {

repository.saveUser(name);

}

}

}

***File name: UserServiceTest.java***

package com.example.mock;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

import static org.mockito.ArgumentMatchers.\*;

public class UserServiceTest {

@Test

public void testRegisterWithExactArgument() {

// 1. Create mock

UserRepository mockRepo = *mock*(UserRepository.class);

// 2. Inject into service

UserService service = new UserService(mockRepo);

// 3. Call method

service.register("Alice");

// 4. Verify exact match

*verify*(mockRepo).saveUser("Alice");

// 5. Optional: verify only once

*verify*(mockRepo, *times*(1)).saveUser(*eq*("Alice"));

System.*out*.println("Verified: saveUser(\"Alice\") was called once");

}

@Test

public void testRegisterWithAnyString() {

UserRepository mockRepo = *mock*(UserRepository.class);

UserService service = new UserService(mockRepo);

service.register("Bob");

// Match any string (not necessarily "Bob")

*verify*(mockRepo).saveUser(*anyString*());

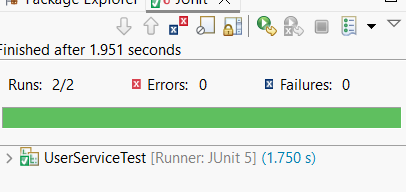
System.*out*.println(" Verified: saveUser() was called with any string");

}

}

**OUTPUT**

****



**Exercise 4: Handling Void Methods**

**CODE**

***File name: Notifier.java***

package com.example.mock;

public interface Notifier {

void sendNotification(String message);

}

***File name: AlertService.java***

package com.example.mock;

public class AlertService {

private Notifier notifier;

public AlertService(Notifier notifier) {

this.notifier = notifier;

}

public void alertUser(String message) {

notifier.sendNotification(message);

}

}

***File name: AlertServiceTest.java***

package com.example.mock;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class AlertServiceTest {

@Test

public void testVoidMethodCalled() {

// 1. Create mock

Notifier mockNotifier = *mock*(Notifier.class);

// 2. Inject into service

AlertService service = new AlertService(mockNotifier);

// 3. Call method

service.alertUser("Low Battery");

// 4. Verify interaction

*verify*(mockNotifier).sendNotification("Low Battery");

System.*out*.println(" Verified: sendNotification(\"Low Battery\") was called.");

}

@Test

public void testStubVoidMethod() {

// 1. Create mock

Notifier mockNotifier = *mock*(Notifier.class);

// 2. Stub the void method to do nothing (optional)

*doNothing*().when(mockNotifier).sendNotification(*anyString*());

// 3. Inject and call

AlertService service = new AlertService(mockNotifier);

service.alertUser("Warning!");

// 4. Verify

*verify*(mockNotifier).sendNotification(*eq*("Warning!"));

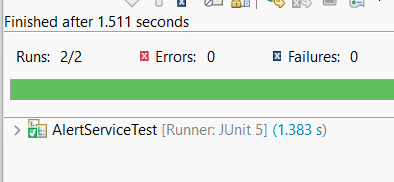
System.*out*.println(" Verified: sendNotification(\"Warning!\") stubbed and called.");

}

}

**OUTPUT**





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

**CODE**

**File name: ExternalApi.java**

package com.example.mock;

public interface ExternalApi {

String getData();

}

***File name: MyService.java***

package com.example.mock;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

***File name: MyserviceTest.java***

package com.example.mock;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class MyServiceTest {

@Test

public void testMultipleReturns() {

// 1. Create mock

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

// 2. Stub method to return multiple values

*when*(mockApi.getData()).thenReturn("First Call").thenReturn("Second Call").thenReturn("Third Call");

// 3. Inject into service

MyService service = new MyService(mockApi);

// 4. Call method multiple times and assert

*assertEquals*("First Call", service.fetchData());

*assertEquals*("Second Call", service.fetchData());

*assertEquals*("Third Call", service.fetchData());

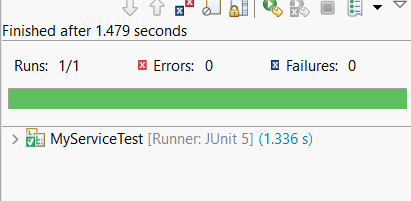
System.*out*.println("First, second, and third calls returned different responses as expected.");

}

}

**OUTPUT**





**Exercise 6: Verifying Interaction Order**

**CODE**

***File name: ExternalApi.java***

package com.example.mock;

public interface ExternalApi {

String stepOne();

String stepTwo();

String stepThree();

}

***File name: MyService.java***

package com.example.mock;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void performSteps() {

api.stepOne();

api.stepTwo();

api.stepThree();

}

}

***File name: MyServiceTest.java***

package com.example.mock;

import org.junit.jupiter.api.Test;

import org.mockito.InOrder;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

@Test

public void testVerifyInteractionOrder() {

// 1. Create mock

ExternalApi mockApi = mock(ExternalApi.class);

// 2. Inject into service

MyService service = new MyService(mockApi);

// 3. Call method

service.performSteps();

// 4. Create InOrder verifier

InOrder inOrder = inOrder(mockApi);

// 5. Verify call order

inOrder.verify(mockApi).stepOne();

inOrder.verify(mockApi).stepTwo();

inOrder.verify(mockApi).stepThree();

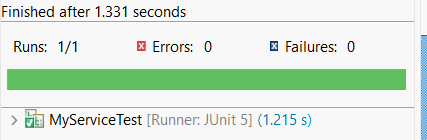
System.out.println("Verified: stepOne(), stepTwo(), and stepThree() were called in correct order.");

}

}

**OUTPUT**





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

**CODE**

***File name: Notifier.java***

package com.example.mock;

public interface Notifier {

void send(String message);

}

***File name: AlertService.java***

package com.example.mock;

public class AlertService {

private Notifier notifier;

public AlertService(Notifier notifier) {

this.notifier = notifier;

}

public void triggerAlert(String msg) {

notifier.send(msg);

}

}

***File name: AlertServiceTest.java***

package com.example.mock;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class AlertServiceTest {

@Test

public void testVoidMethodThrowsException() {

// 1. Create mock

Notifier mockNotifier = *mock*(Notifier.class);

// 2. Stub the void method to throw exception

*doThrow*(new RuntimeException("Failed to send")).when(mockNotifier).send("Danger");

// 3. Inject into AlertService

AlertService service = new AlertService(mockNotifier);

// 4. Assert exception is thrown

*assertThrows*(RuntimeException.class, () -> {

service.triggerAlert("Danger");

});

// 5. Verify interaction

*verify*(mockNotifier).send("Danger");

System.*out*.println("Verified: send(\"Danger\") threw exception and was called.");

}

}

**OUTPUT**



