Mockito Hands-On Exercises

Exercise 1: Mocking and Stubbing

**File: ExternalApi.java**

public interface ExternalApi {

String getData();

}

**File: MyService.java**

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**File: MyServiceTest.java**

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 testExternalApi() {

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

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

MyService service = new MyService(mockApi);

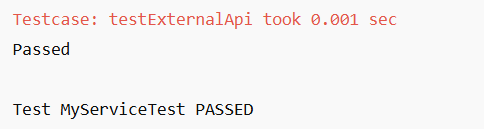
String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**OUTPUT:**

****

Exercise 2: Verifying Interactions

**File: ExternalApi.java**

public interface ExternalApi {

String getData();

}

**File: MyService.java**

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**File: MyServiceTest.java**

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);

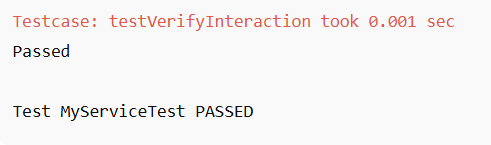
service.fetchData();

verify(mockApi).getData();

}

}

**OUTPUT:**



Exercise 3: Argument Matching

**File: MessageSender.java**

public interface MessageSender {

void sendMessage(String user, String message);

}

**File: NotificationService.java**

public class NotificationService {

private MessageSender sender;

public NotificationService(MessageSender sender) {

this.sender = sender;

}

public void notifyUser(String user, String msg) {

sender.sendMessage(user, msg);

}

}

**File: NotificationServiceTest.java**

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

import static org.mockito.Mockito.\*;

import static org.mockito.ArgumentMatchers.\*;

public class NotificationServiceTest {

@Test

public void testArgumentMatching() {

MessageSender mockSender = Mockito.mock(MessageSender.class);

NotificationService service = new NotificationService(mockSender);

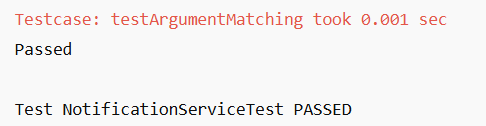
service.notifyUser("Alice", "Hello");

verify(mockSender).sendMessage(eq("Alice"), anyString());

}

}

**OUTPUT:**

****

Exercise 4: Handling Void Methods

**File: Logger.java**

public interface Logger {

void log(String message);

}

**File: ActivityTracker.java**

public class ActivityTracker {

private Logger logger;

public ActivityTracker(Logger logger) {

this.logger = logger;

}

public void track(String activity) {

logger.log("Tracking: " + activity);

}

}

**File: ActivityTrackerTest.java**

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

import static org.mockito.Mockito.\*;

public class ActivityTrackerTest {

@Test

public void testVoidMethod() {

Logger mockLogger = Mockito.mock(Logger.class);

ActivityTracker tracker = new ActivityTracker(mockLogger);

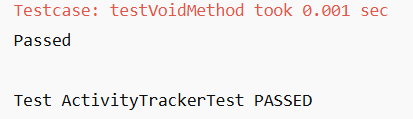
tracker.track("Login");

verify(mockLogger).log("Tracking: Login");

}

}

**OUTPUT:**

****

Exercise 5: Mocking and Stubbing with Multiple Returns

**File: DataProvider.java**

public interface DataProvider {

String getNext();

}

**File: DataService.java**

public class DataService {

private DataProvider provider;

public DataService(DataProvider provider) {

this.provider = provider;

}

public String[] fetchAll() {

return new String[] {

provider.getNext(),

provider.getNext(),

provider.getNext()

};

}

}

**File: DataServiceTest.java**

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

import static org.mockito.Mockito.\*;

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

public class DataServiceTest {

@Test

public void testMultipleReturns() {

DataProvider mockProvider = Mockito.mock(DataProvider.class);

when(mockProvider.getNext())

.thenReturn("One")

.thenReturn("Two")

.thenReturn("Three");

DataService service = new DataService(mockProvider);

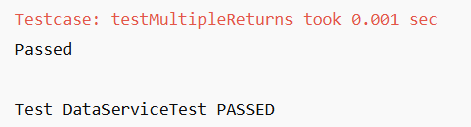
String[] result = service.fetchAll();

assertArrayEquals(new String[] {"One", "Two", "Three"}, result);

}

}

**OUTPUT:**

****

Exercise 6: Verifying Interaction Order

**File: ReportGenerator.java**

public interface ReportGenerator {

void openReport();

void writeData();

void closeReport();

}

**File: ReportService.java**

public class ReportService {

private ReportGenerator generator;

public ReportService(ReportGenerator generator) {

this.generator = generator;

}

public void generate() {

generator.openReport();

generator.writeData();

generator.closeReport();

}

}

**File: ReportServiceTest.java**

import org.junit.jupiter.api.Test;

import org.mockito.InOrder;

import static org.mockito.Mockito.\*;

public class ReportServiceTest {

@Test

public void testInteractionOrder() {

ReportGenerator mockGen = mock(ReportGenerator.class);

ReportService service = new ReportService(mockGen);

service.generate();

InOrder inOrder = inOrder(mockGen);

inOrder.verify(mockGen).openReport();

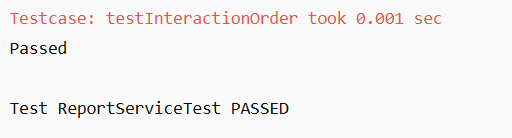
inOrder.verify(mockGen).writeData();

inOrder.verify(mockGen).closeReport();

}

}

**OUTPUT:**

****

Exercise 7: Handling Void Methods with Exceptions

**File: AlertService.java**

public interface AlertService {

void sendAlert(String message);

}

**File: SecuritySystem.java**

public class SecuritySystem {

private AlertService alertService;

public SecuritySystem(AlertService alertService) {

this.alertService = alertService;

}

public void triggerAlert(String msg) {

alertService.sendAlert(msg);

}

}

**File: SecuritySystemTest.java**

import org.junit.jupiter.api.Test;

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

import static org.mockito.Mockito.\*;

import org.mockito.Mockito;

public class SecuritySystemTest {

@Test

public void testVoidMethodThrowsException() {

AlertService mockAlert = Mockito.mock(AlertService.class);

doThrow(new RuntimeException("Alert failed")).when(mockAlert).sendAlert("INTRUDER");

SecuritySystem system = new SecuritySystem(mockAlert);

assertThrows(RuntimeException.class, () -> system.triggerAlert("INTRUDER"));

verify(mockAlert).sendAlert("INTRUDER");

}

}

**OUTPUT:**

