**Mockito Hands-On Exercises**

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

**Code:**

EXTERNALAPI.java

**public** **interface** ExternalApi {

**String** getData();

**String** getData(**String** type);

**void** saveData(**String** data);

}

MYSERVICES.java

**public** **class** MyService {

**private** **ExternalApi** api;

**public** MyService(**ExternalApi** api) {

        this.api **=** api;

    }

**public** **String** fetchData() {

**return** api.getData();

    }

**public** **String** fetchTypedData(**String** type) {

**return** api.getData(type);

    }

**public** **void** save(**String** data) {

        api.saveData(data);

    }

}

MYSERVICETEST.java

**import** **org.junit.jupiter.api.Test**;

**import** **org.mockito.InOrder**;

**import** **org.mockito.Mockito**;

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

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

**public** **class** MyServiceTest {

***Exercise 1: Mocking and Stubbing***

    @**Test**

**public** **void** testMockingAndStubbing() {

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

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

**MyService** service **=** **new** MyService(mockApi);

        assertEquals("Mock Data", service.fetchData());

    }

***Exercise 2: Verifying Interactions***

    @**Test**

**public** **void** testVerifyInteraction() {

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

**MyService** service **=** **new** MyService(mockApi);

        service.fetchData();

        verify(mockApi).getData();

    }

***Exercise 3: Argument Matching***

    @**Test**

**public** **void** testArgumentMatching() {

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

        when(mockApi.getData(anyString())).thenReturn("Matched");

**MyService** service **=** **new** MyService(mockApi);

        service.fetchTypedData("example");

        verify(mockApi).getData(eq("example"));

    }

***Exercise 4: Handling Void Methods***

    @**Test**

**public** **void** testVoidMethod() {

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

**MyService** service **=** **new** MyService(mockApi);

        service.save("Data");

        verify(mockApi).saveData("Data");

    }

***Exercise 5: Multiple Return Values***

    @**Test**

**public** **void** testMultipleReturns() {

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

        when(mockApi.getData())

            .thenReturn("First")

            .thenReturn("Second");

**MyService** service **=** **new** MyService(mockApi);

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

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

    }

***Exercise 6: Verifying Interaction Order***

    @**Test**

**public** **void** testInteractionOrder() {

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

**MyService** service **=** **new** MyService(mockApi);

        service.fetchData();

        service.save("Ordered");

**InOrder** inOrder **=** inOrder(mockApi);

        inOrder.verify(mockApi).getData();

        inOrder.verify(mockApi).saveData("Ordered");

    }

***Exercise 7: Void Method with Exception***

    @**Test**

**public** **void** testVoidMethodException() {

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

        doThrow(**new** RuntimeException("Boom")).when(mockApi).saveData("fail");

**MyService** service **=** **new** MyService(mockApi);

        assertThrows(RuntimeException.class, () **->** service.save("fail"));

        verify(mockApi).saveData("fail");

    }

}

**OUTPUT:**



