**Mockito Hands-On Exercises**

**Exercise 1: Mocking and Stubbing**

public interface ExternalApi {

/\*\*

\* Retrieves data from external source

\* @return data as string

\*/

String getData();

/\*\*

\* Retrieves data by ID

\* @param id identifier

\* @return data for the given ID

\*/

String getDataById(String id);

/\*\*

\* Saves data to external source

\* @param data data to save

\* @return success status

\*/

boolean saveData(String data);

/\*\*

\* Checks if the external service is available

\* @return true if available, false otherwise

\*/

boolean isServiceAvailable();

}

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

**private** **final** ExternalApi externalApi;

/\*\*

\* Constructor with dependency injection

\* **@param** externalApi the external API dependency

\*/

**public** MyService(ExternalApi externalApi) {

**this**.externalApi = externalApi;

}

/\*\*

\* Fetches data from external API

\* **@return** the data retrieved from external API

\*/

**public** String fetchData() {

**return** externalApi.getData();

}

/\*\*

\* Fetches data by ID with error handling

\* **@param** id the identifier

\* **@return** the data or error message

\*/

**public** String fetchDataById(String id) {

**if** (id == **null** || id.trim().isEmpty()) {

**return** "Invalid ID";

}

**if** (!externalApi.isServiceAvailable()) {

**return** "Service unavailable";

}

**return** externalApi.getDataById(id);

}

/\*\*

\* Processes and saves data

\* **@param** rawData raw data to process

\* **@return** success message or error

\*/

**public** String processAndSaveData(String rawData) {

**if** (rawData == **null** || rawData.trim().isEmpty()) {

**return** "No data to process";

}

String processedData = rawData.toUpperCase();

**boolean** saved = externalApi.saveData(processedData);

**return** saved ? "Data saved successfully" : "Failed to save data";

}

/\*\*

\* Gets data with fallback mechanism

\* **@param** id the identifier

\* **@return** data or fallback value

\*/

**public** String getDataWithFallback(String id) {

**try** {

**if** (externalApi.isServiceAvailable()) {

String data = externalApi.getDataById(id);

**return** data != **null** ? data : "Default Data";

} **else** {

**return** "Fallback Data - Service Unavailable";

}

} **catch** (Exception e) {

**return** "Error occurred: " + e.getMessage();

}

}

}

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

**import** **static** org.junit.Assert.\*;

**import** org.junit.\*;

**import** org.mockito.Mock;

**import** org.mockito.MockitoAnnotations;

/\*\*

\* Test class demonstrating Mockito mocking and stubbing

\* Shows different ways to create mocks and stub method behaviors

\*/

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

@Mock

**private** ExternalApi mockExternalApi;

**private** MyService myService;

@Before

**public** **void** setUp() {

MockitoAnnotations.initMocks(**this**);

myService = **new** MyService(mockExternalApi);

}

@After

**public** **void** tearDown() {

myService = **null**;

}

@BeforeClass

**public** **static** **void** setUpClass() {

System.***out***.println("=== Starting Mockito Tests ===");

}

@AfterClass

**public** **static** **void** tearDownClass() {

System.***out***.println("=== Finished Mockito Tests ===");

}

@Test

**public** **void** testFetchData\_BasicMocking() {

String expectedData = "Mock Data";

*when*(mockExternalApi.getData()).thenReturn(expectedData);

String result = myService.fetchData();

*assertEquals*("Should return mocked data", expectedData, result);

*verify*(mockExternalApi).getData();

}

@Test

**public** **void** testFetchDataById\_ValidId() {

String testId = "123";

String expectedData = "Data for ID: 123";

*when*(mockExternalApi.isServiceAvailable()).thenReturn(**true**);

*when*(mockExternalApi.getDataById(testId)).thenReturn(expectedData);

String result = myService.fetchDataById(testId);

*assertEquals*("Should return data for valid ID", expectedData, result);

*verify*(mockExternalApi).isServiceAvailable();

*verify*(mockExternalApi).getDataById(testId);

}

@Test

**public** **void** testFetchDataById\_InvalidId() {

String invalidId = "";

String result = myService.fetchDataById(invalidId);

*assertEquals*("Should return error for invalid ID", "Invalid ID", result);

*verify*(mockExternalApi, *never*()).getDataById(*anyString*());

}

@Test

**public** **void** testFetchDataById\_ServiceUnavailable() {

String testId = "123";

*when*(mockExternalApi.isServiceAvailable()).thenReturn(**false**);

String result = myService.fetchDataById(testId);

*assertEquals*("Should return service unavailable message",

"Service unavailable", result);

*verify*(mockExternalApi).isServiceAvailable();

*verify*(mockExternalApi, *never*()).getDataById(*anyString*());

}

@Test

**public** **void** testProcessAndSaveData\_Success() {

String rawData = "test data";

String expectedProcessedData = "TEST DATA";

*when*(mockExternalApi.saveData(expectedProcessedData)).thenReturn(**true**);

String result = myService.processAndSaveData(rawData);

*assertEquals*("Should return success message",

"Data saved successfully", result);

*verify*(mockExternalApi).saveData(expectedProcessedData);

}

@Test

**public** **void** testProcessAndSaveData\_Failure() {

String rawData = "test data";

String expectedProcessedData = "TEST DATA";

*when*(mockExternalApi.saveData(expectedProcessedData)).thenReturn(**false**);

String result = myService.processAndSaveData(rawData);

*assertEquals*("Should return failure message",

"Failed to save data", result);

*verify*(mockExternalApi).saveData(expectedProcessedData);

}

@Test

**public** **void** testProcessAndSaveData\_EmptyData() {

String emptyData = "";

String result = myService.processAndSaveData(emptyData);

*assertEquals*("Should return no data message",

"No data to process", result);

*verify*(mockExternalApi, *never*()).saveData(*anyString*());

}

@Test

**public** **void** testGetDataWithFallback\_ServiceAvailable() {

String testId = "456";

String expectedData = "Real Data";

*when*(mockExternalApi.isServiceAvailable()).thenReturn(**true**);

*when*(mockExternalApi.getDataById(testId)).thenReturn(expectedData);

String result = myService.getDataWithFallback(testId);

*assertEquals*("Should return real data when service is available",

expectedData, result);

}

@Test

**public** **void** testGetDataWithFallback\_ServiceUnavailable() {

String testId = "456";

*when*(mockExternalApi.isServiceAvailable()).thenReturn(**false**);

String result = myService.getDataWithFallback(testId);

*assertEquals*("Should return fallback message when service unavailable",

"Fallback Data - Service Unavailable", result);

*verify*(mockExternalApi, *never*()).getDataById(*anyString*());

}

@Test

**public** **void** testGetDataWithFallback\_NullResponse() {

String testId = "789";

*when*(mockExternalApi.isServiceAvailable()).thenReturn(**true**);

*when*(mockExternalApi.getDataById(testId)).thenReturn(**null**);

String result = myService.getDataWithFallback(testId);

*assertEquals*("Should return default data when API returns null",

"Default Data", result);

}

@Test

**public** **void** testAlternativeWayToCreateMock() {

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

MyService alternativeService = **new** MyService(alternativeMock);

String testData = "Alternative Mock Data";

*when*(alternativeMock.getData()).thenReturn(testData);

String result = alternativeService.fetchData();

*assertEquals*("Alternative mock should work the same way",

testData, result);

}

@Test

**public** **void** testVerifyInteractions() {

String testId = "verify-test";

*when*(mockExternalApi.isServiceAvailable()).thenReturn(**true**);

*when*(mockExternalApi.getDataById(testId)).thenReturn("Test Data");

myService.fetchDataById(testId);

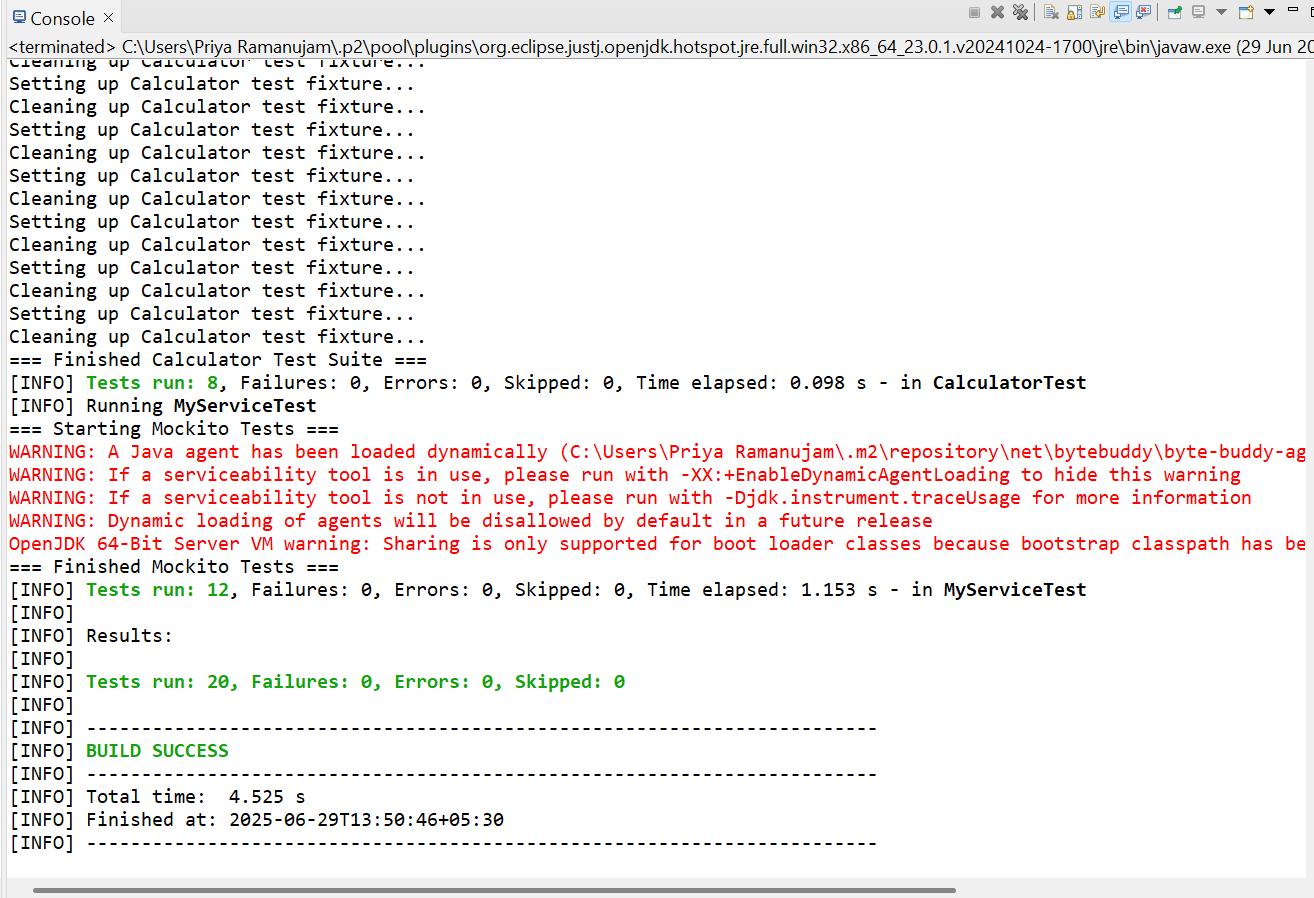
*verify*(mockExternalApi, *times*(1)).isServiceAvailable();

*verify*(mockExternalApi, *times*(1)).getDataById(testId);

*verify*(mockExternalApi, *never*()).saveData(*anyString*());

}

}



**Exercise 2: Verifying Interactions**

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

**import** **static** org.junit.Assert.\*;

**import** org.junit.\*;

**import** org.mockito.Mock;

**import** org.mockito.MockitoAnnotations;

**import** org.mockito.ArgumentCaptor;

**import** org.mockito.InOrder;

/\*\*

\* Test class demonstrating various types of interaction verification with Mockito

\* Shows how to verify method calls, arguments, call counts, and call order

\*/

**public** **class** InteractionVerificationTest {

@Mock

**private** ExternalApi mockExternalApi;

**private** MyService myService;

@Before

**public** **void** setUp() {

MockitoAnnotations.initMocks(**this**);

myService = **new** MyService(mockExternalApi);

}

@BeforeClass

**public** **static** **void** setUpClass() {

System.***out***.println("=== Starting Interaction Verification Tests ===");

}

@AfterClass

**public** **static** **void** tearDownClass() {

System.***out***.println("=== Finished Interaction Verification Tests ===");

}

@Test

**public** **void** testBasicMethodCallVerification() {

*when*(mockExternalApi.getData()).thenReturn("Test Data");

String result = myService.fetchData();

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

*verify*(mockExternalApi).getData();

}

@Test

**public** **void** testMethodNotCalled() {

String invalidId = "";

String result = myService.fetchDataById(invalidId);

*assertEquals*("Invalid ID", result);

*verify*(mockExternalApi, *never*()).getDataById(*anyString*());

*verify*(mockExternalApi, *never*()).isServiceAvailable();

}

@Test

**public** **void** testVerifyWithSpecificArguments() {

String specificId = "USER123";

*when*(mockExternalApi.isServiceAvailable()).thenReturn(**true**);

*when*(mockExternalApi.getDataById(specificId)).thenReturn("User Data");

myService.fetchDataById(specificId);

*verify*(mockExternalApi).getDataById("USER123");

*verify*(mockExternalApi).getDataById(*eq*("USER123"));

}

@Test

**public** **void** testVerifyWithArgumentMatchers() {

*when*(mockExternalApi.isServiceAvailable()).thenReturn(**true**);

*when*(mockExternalApi.getDataById(*anyString*())).thenReturn("Some Data");

myService.fetchDataById("ANY\_ID");

*verify*(mockExternalApi).getDataById(*anyString*());

*verify*(mockExternalApi).getDataById(*startsWith*("ANY"));

*verify*(mockExternalApi).getDataById(*matches*(".\*\_ID"));

}

@Test

**public** **void** testVerifyMultipleArgumentsWithMatchers() {

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

*when*(mockApi.saveData(*anyString*())).thenReturn(**true**);

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

service.processAndSaveData("test input");

*verify*(mockApi).saveData("TEST INPUT");

}

@Test

**public** **void** testVerifyCallCounts() {

*when*(mockExternalApi.getData()).thenReturn("Data");

myService.fetchData();

myService.fetchData();

myService.fetchData();

*verify*(mockExternalApi, *times*(3)).getData();

*verify*(mockExternalApi, *atLeast*(2)).getData();

*verify*(mockExternalApi, *atMost*(5)).getData();

*verify*(mockExternalApi, *atLeastOnce*()).getData();

}

@Test

**public** **void** testVerifyNoMoreInteractions() {

*when*(mockExternalApi.getData()).thenReturn("Data");

myService.fetchData();

*verify*(mockExternalApi).getData();

*verifyNoMoreInteractions*(mockExternalApi);

}

@Test

**public** **void** testVerifyCallOrder() {

*when*(mockExternalApi.isServiceAvailable()).thenReturn(**true**);

*when*(mockExternalApi.getDataById("123")).thenReturn("Test Data");

myService.fetchDataById("123");

InOrder inOrder = *inOrder*(mockExternalApi);

inOrder.verify(mockExternalApi).isServiceAvailable();

inOrder.verify(mockExternalApi).getDataById("123");

}

@Test

**public** **void** testVerifyMultipleCallsInOrder() {

*when*(mockExternalApi.isServiceAvailable()).thenReturn(**true**);

*when*(mockExternalApi.getDataById(*anyString*())).thenReturn("Data");

*when*(mockExternalApi.saveData(*anyString*())).thenReturn(**true**);

myService.fetchDataById("user1");

myService.fetchDataById("user2");

myService.processAndSaveData("some data");

InOrder inOrder = *inOrder*(mockExternalApi);

inOrder.verify(mockExternalApi).isServiceAvailable();

inOrder.verify(mockExternalApi).getDataById("user1");

inOrder.verify(mockExternalApi).isServiceAvailable();

inOrder.verify(mockExternalApi).getDataById("user2");

inOrder.verify(mockExternalApi).saveData("SOME DATA");

}

@Test

**public** **void** testArgumentCapture() {

ArgumentCaptor<String> argumentCaptor = ArgumentCaptor.*forClass*(String.**class**);

*when*(mockExternalApi.saveData(*anyString*())).thenReturn(**true**);

myService.processAndSaveData("input data");

*verify*(mockExternalApi).saveData(argumentCaptor.capture());

String capturedArgument = argumentCaptor.getValue();

*assertEquals*("Should capture the processed (uppercase) data",

"INPUT DATA", capturedArgument);

}

@Test

**public** **void** testMultipleArgumentCapture() {

ArgumentCaptor<String> argumentCaptor = ArgumentCaptor.*forClass*(String.**class**);

*when*(mockExternalApi.saveData(*anyString*())).thenReturn(**true**);

myService.processAndSaveData("first");

myService.processAndSaveData("second");

myService.processAndSaveData("third");

*verify*(mockExternalApi, *times*(3)).saveData(argumentCaptor.capture());

java.util.List<String> capturedArguments = argumentCaptor.getAllValues();

*assertEquals*("Should capture all three arguments", 3, capturedArguments.size());

*assertEquals*("FIRST", capturedArguments.get(0));

*assertEquals*("SECOND", capturedArguments.get(1));

*assertEquals*("THIRD", capturedArguments.get(2));

}

@Test

**public** **void** testVerifyWithTimeout() {

*when*(mockExternalApi.getData()).thenReturn("Data");

myService.fetchData();

*verify*(mockExternalApi, *timeout*(1000)).getData();

}

@Test

**public** **void** testComplexVerificationScenario() {

*when*(mockExternalApi.isServiceAvailable()).thenReturn(**true**);

*when*(mockExternalApi.getDataById(*anyString*())).thenReturn("Data");

*when*(mockExternalApi.saveData(*anyString*())).thenReturn(**true**);

myService.fetchDataById("user1");

myService.getDataWithFallback("user2");

myService.processAndSaveData("business data");

*verify*(mockExternalApi, *times*(2)).isServiceAvailable();

*verify*(mockExternalApi).getDataById("user1");

*verify*(mockExternalApi).getDataById("user2");

*verify*(mockExternalApi).saveData("BUSINESS DATA");

InOrder inOrder = *inOrder*(mockExternalApi);

inOrder.verify(mockExternalApi).getDataById("user1");

inOrder.verify(mockExternalApi).getDataById("user2");

inOrder.verify(mockExternalApi).saveData("BUSINESS DATA");

*verify*(mockExternalApi, *never*()).saveData("business data");

}

@Test

**public** **void** testVerificationFailureExample() {

*when*(mockExternalApi.getData()).thenReturn("Data");

myService.fetchData();

*verify*(mockExternalApi, *times*(1)).getData();

}

@Test

**public** **void** testCustomArgumentMatchers() {

*when*(mockExternalApi.saveData(*anyString*())).thenReturn(**true**);

myService.processAndSaveData("test123");

*verify*(mockExternalApi).saveData(*argThat*(data ->

data != **null** &&

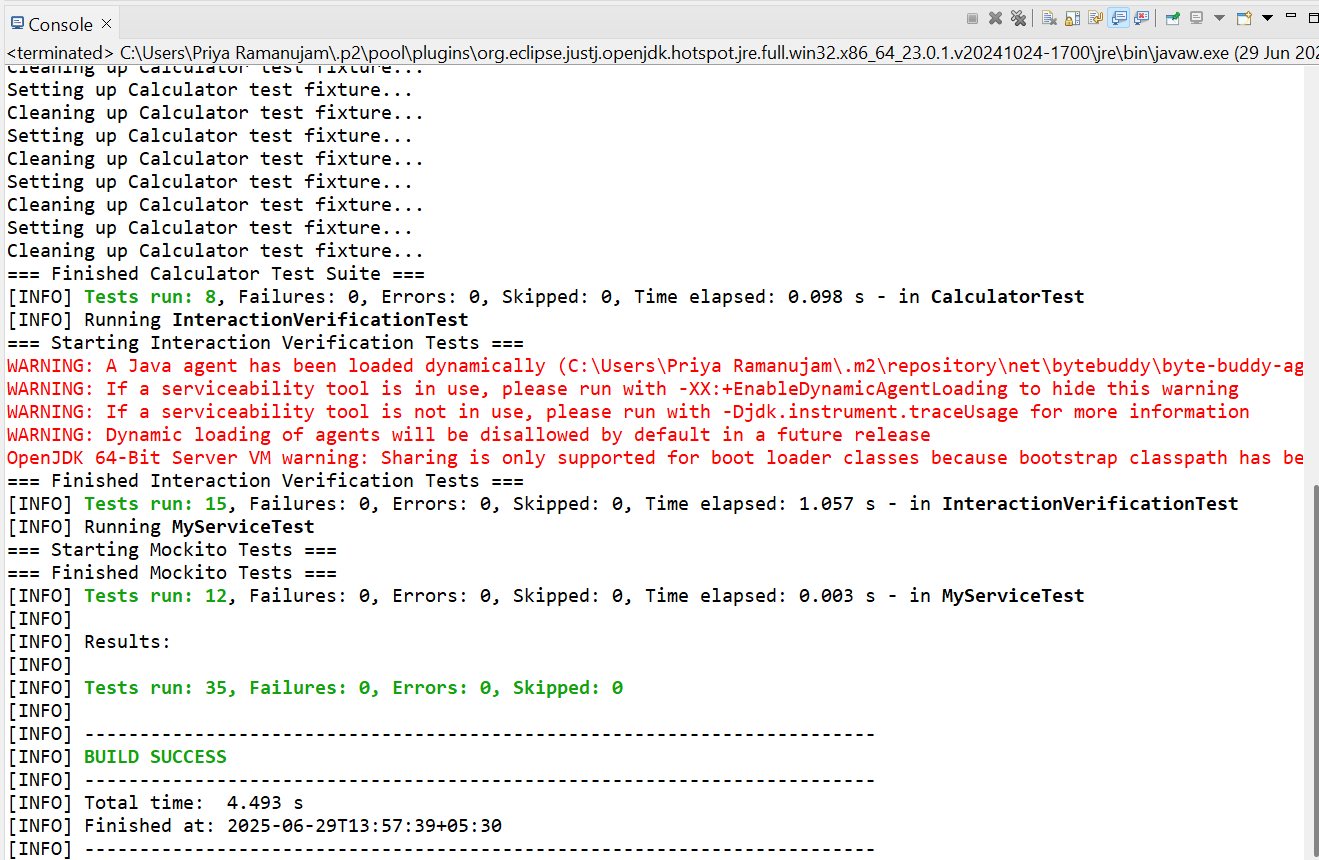
data.length() > 5 &&

data.contains("TEST")

));

}

}

****