Mockito Hands-On Exercises

**Exercise 1: Mocking and Stubbing**

**Scenario:**

You need to test a service that depends on an external API. Use Mockito to mock the

external API and stub its methods.

**Steps:**

1. Create a mock object for the external API.

2. Stub the methods to return predefined values.

3. Write a test case that uses the mock object.

**Solution:**

**ExternalApi.java**

package excercise1;

public interface ExternalApi {

String getData();

}

**Service.java**

package excercise1;

public class Service {

private ExternalApi api;

public Service(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**ServiceTest.java**

import excercise1.\*;

import org.junit.Test;

import org.mockito.Mockito;

import static org.junit.Assert.assertEquals;

import static org.mockito.Mockito.\*;

public class ServiceTest {

@Test

public void testExternalApi() {

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

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

Service service = new Service(api);

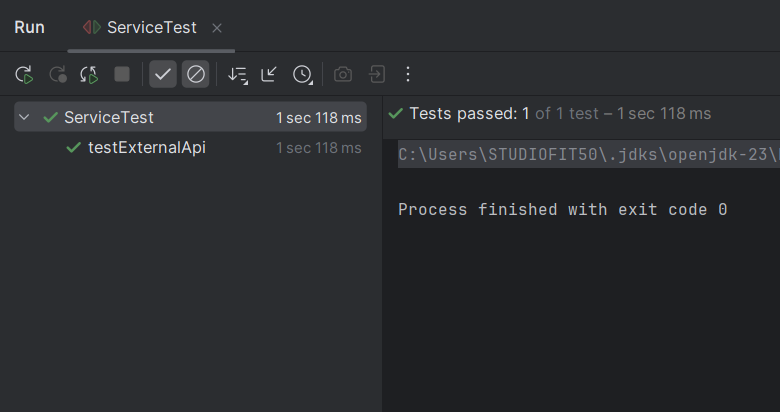
String data = service.fetchData();

assertEquals("Mock Data", data);

}

}

**Output:**



**Exercise 2: Verifying Interactions**

**Scenario:**

You need to ensure that a method is called with specific arguments.

**Steps:**

1. Create a mock object.

2. Call the method with specific arguments.

3. Verify the interaction.

**Solution:**

**ServiceTest.java**

import excercise1.\*;

import org.junit.Test;

import org.mockito.Mockito;

import static org.junit.Assert.assertEquals;

import static org.mockito.Mockito.\*;

public class ServiceTest {

@Test

public void testVerifyInteraction() {

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

Service service = new Service(mockApi);

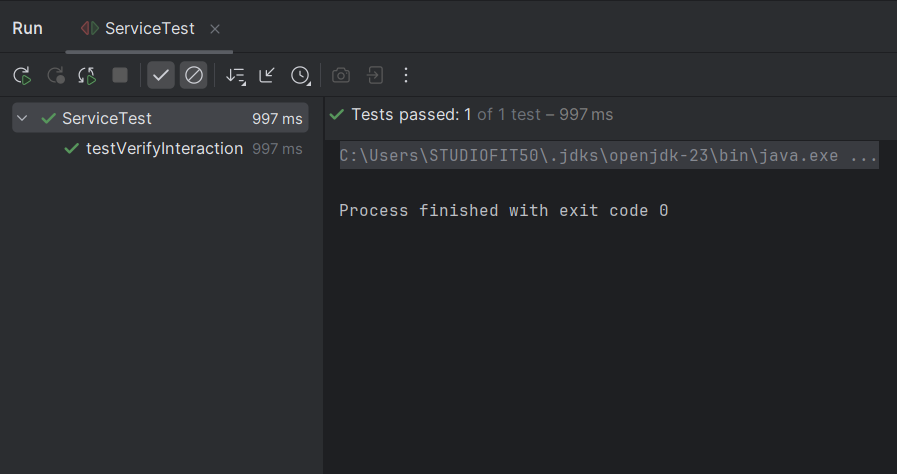
service.fetchData();

verify(mockApi).getData();

}

}

**Output:**



**Exercise 3: Argument Matching**

**Scenario:**

You need to verify that a method is called with specific arguments.

**Steps:**

1. Create a mock object.

2. Call the method with specific arguments.

3. Use argument matchers to verify the interaction.

**Solution:**

**Notifier.java**

package excercise3;

public interface Notifier {

void send(String message);

}

**UserService.java**

package excercise3;

public class UserService {

private Notifier notifier;

public UserService(Notifier notifier) {

this.notifier = notifier;

}

public void registerUser(String username) {

notifier.send("Welcome " + username);

}

}

**UserServiceTest.java**

import static org.mockito.Mockito.\*;

import excercise3.\*;

import org.junit.Test;

public class UserServiceTest {

@Test

public void testSendCalled() {

Notifier mockNotifier = mock(Notifier.class);

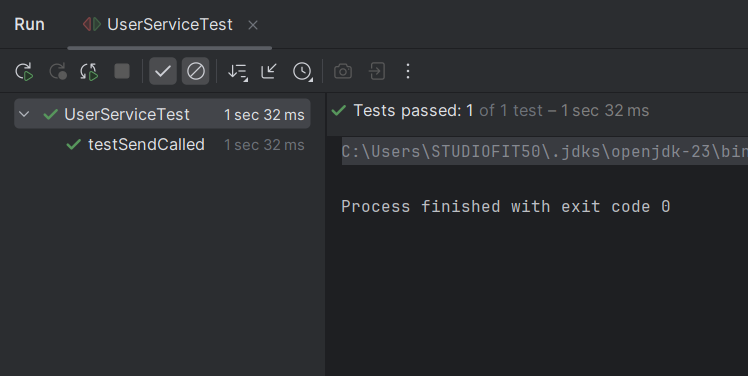
UserService userService = new UserService(mockNotifier);

userService.registerUser("Aysha");

verify(mockNotifier).send("Welcome Aysha");

}

}

**Output:**  


**Exercise 4: Handling Void Methods**

**Scenario:**

You need to test a void method that performs some action.

**Steps:**

1. Create a mock object.

2. Stub the void method.

3. Verify the interaction.

**Solution:**

**Logger.java**

package excercise4;

public interface Logger {

void log(String message);

}

**Processor.java**

package excercise4;

public class Processor {

Logger logger;

public Processor(Logger logger) {

this.logger = logger;

}

public void process(String task) {

logger.log("Processed: " + task);

}

}

**ProcessorTest.java**

import static org.mockito.Mockito.\*;

import excercise4.Logger;

import excercise4.Processor;

import org.junit.jupiter.api.Test;

public class ProcessorTest {

@Test

public void testVoidMethodLogging() {

Logger mockLogger = mock(Logger.class);

doNothing().when(mockLogger).log(anyString());

Processor processor = new Processor(mockLogger);

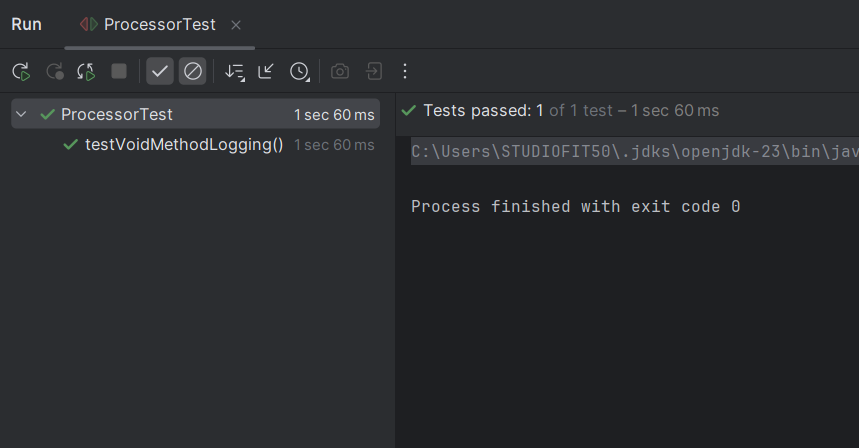
processor.process("Task A");

verify(mockLogger).log("Processed: Task A");

}

}

**Output:**



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

**Scenario:**

You need to test a service that depends on an external API with multiple return values.

**Steps:**

1. Create a mock object for the external API. 2. Stub the methods to return different values on consecutive calls.

3. Write a test case that uses the mock object.

**Solution:**

**QuoteApi.java**

package excercise5;

public interface QuoteApi {

public String getQuote();

}

**QuoteService.java**

package excercise5;

public class QuoteService {

private QuoteApi api;

public QuoteService(QuoteApi api) {

this.api = api;

}

public String[] fetchThreeQuotes() {

return new String[] {

api.getQuote(),

api.getQuote(),

api.getQuote()

};

}

}

**QuoteServiceTest.java**

import static org.mockito.Mockito.\*;

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

import excercise5.QuoteApi;

import excercise5.QuoteService;

import org.junit.jupiter.api.Test;

public class QuoteServiceTest {

@Test

public void testMultipleReturnsFromMock() {

QuoteApi mockApi = mock(QuoteApi.class);

when(mockApi.getQuote()).thenReturn(

"Quote 1", "Quote 2", "Quote 3"

);

QuoteService service = new QuoteService(mockApi);

String[] result = service.fetchThreeQuotes();

assertEquals("Quote 1", result[0]);

assertEquals("Quote 2", result[1]);

assertEquals("Quote 3", result[2]);

}

}

**Output:**  
  
