

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:

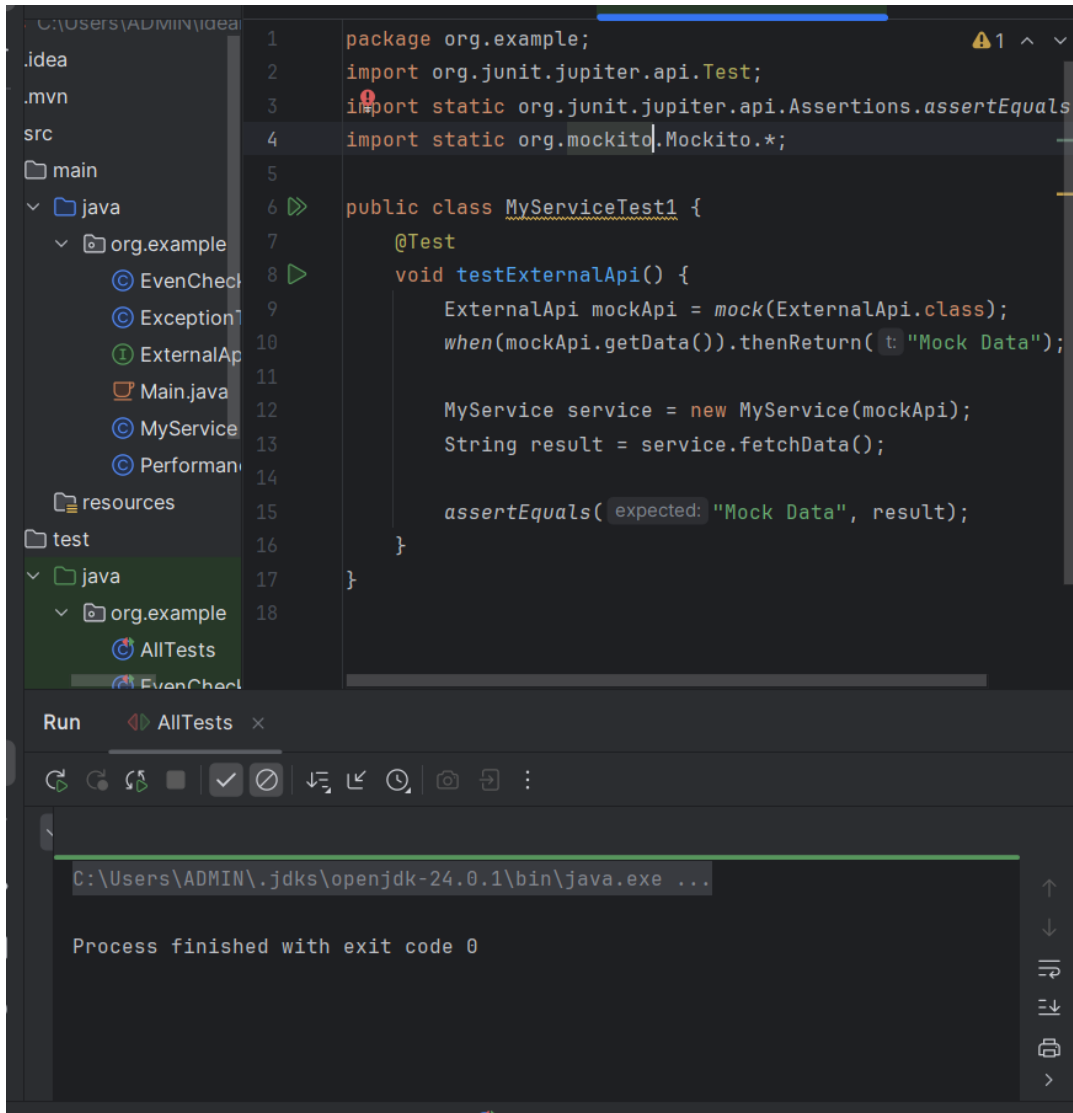
1. Create a mock object for the external API.
2. Stub the methods to return predefined values.
3. Write a test case that uses the mock object.

Solution Code:

```
import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test; import
org.mockito.Mockito;

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:



The screenshot shows an IDE with a project structure on the left and a code editor on the right. The project structure includes a `test` directory with a `java` subdirectory containing `org.example`, which has an `AllTests` class. The code editor displays the following Java code:

```
1 package org.example;
2 import org.junit.jupiter.api.Test;
3 import static org.junit.jupiter.api.Assertions.assertEquals;
4 import static org.mockito.Mockito.*;
5
6 public class MyServiceTest1 {
7     @Test
8     void testExternalApi() {
9         ExternalApi mockApi = mock(ExternalApi.class);
10        when(mockApi.getData()).thenReturn("Mock Data");
11
12        MyService service = new MyService(mockApi);
13        String result = service.fetchData();
14
15        assertEquals("Mock Data", result);
16    }
17 }
18
```

Below the code editor, the `Run` tab shows the command `C:\Users\ADMIN\.jdk\openjdk-24.0.1\bin\java.exe ...` and the message `Process finished with exit code 0`, indicating a successful test run.

Exercise 2: Verifying Interactions

Scenario:

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

Steps:

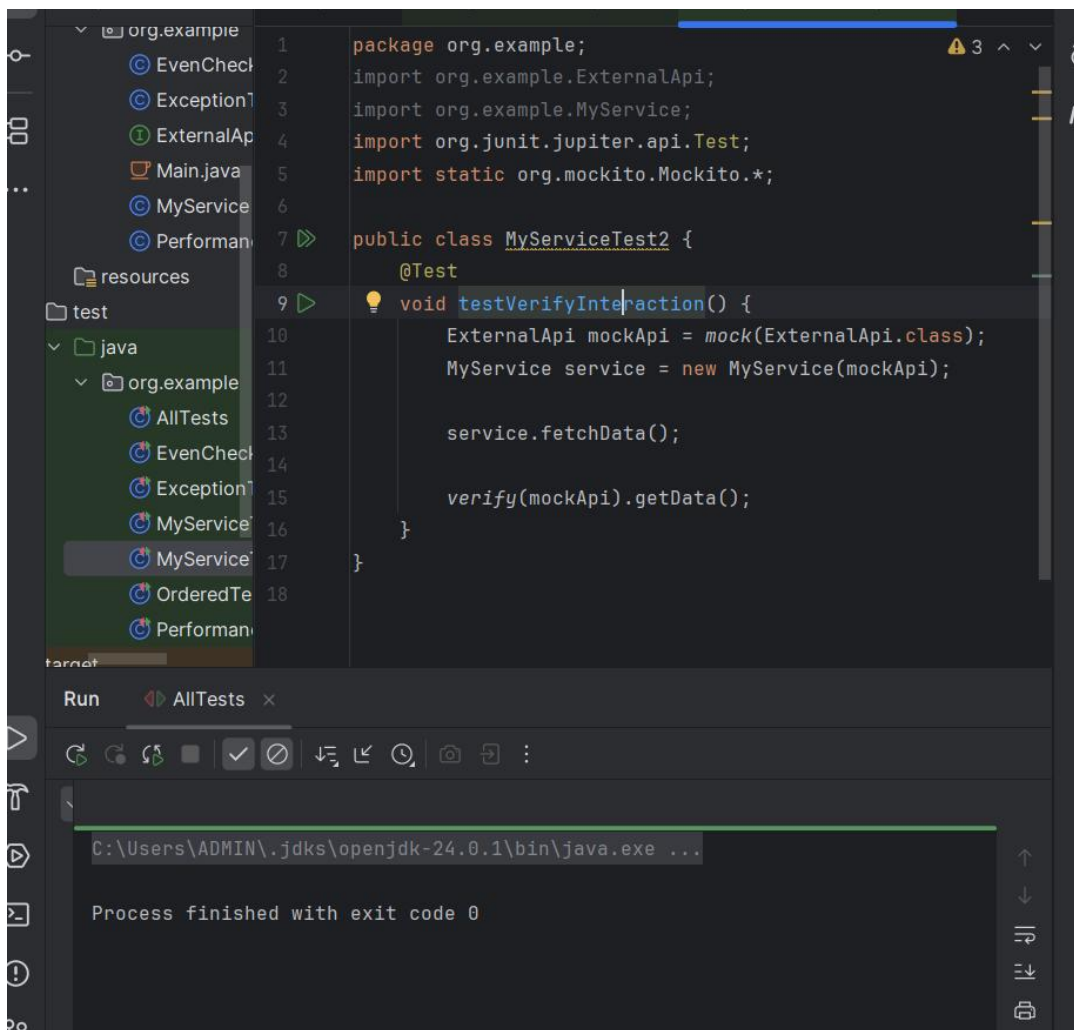
1. Create a mock object.
2. Call the method with specific arguments.
3. Verify the interaction.

Solution Code:

```
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:



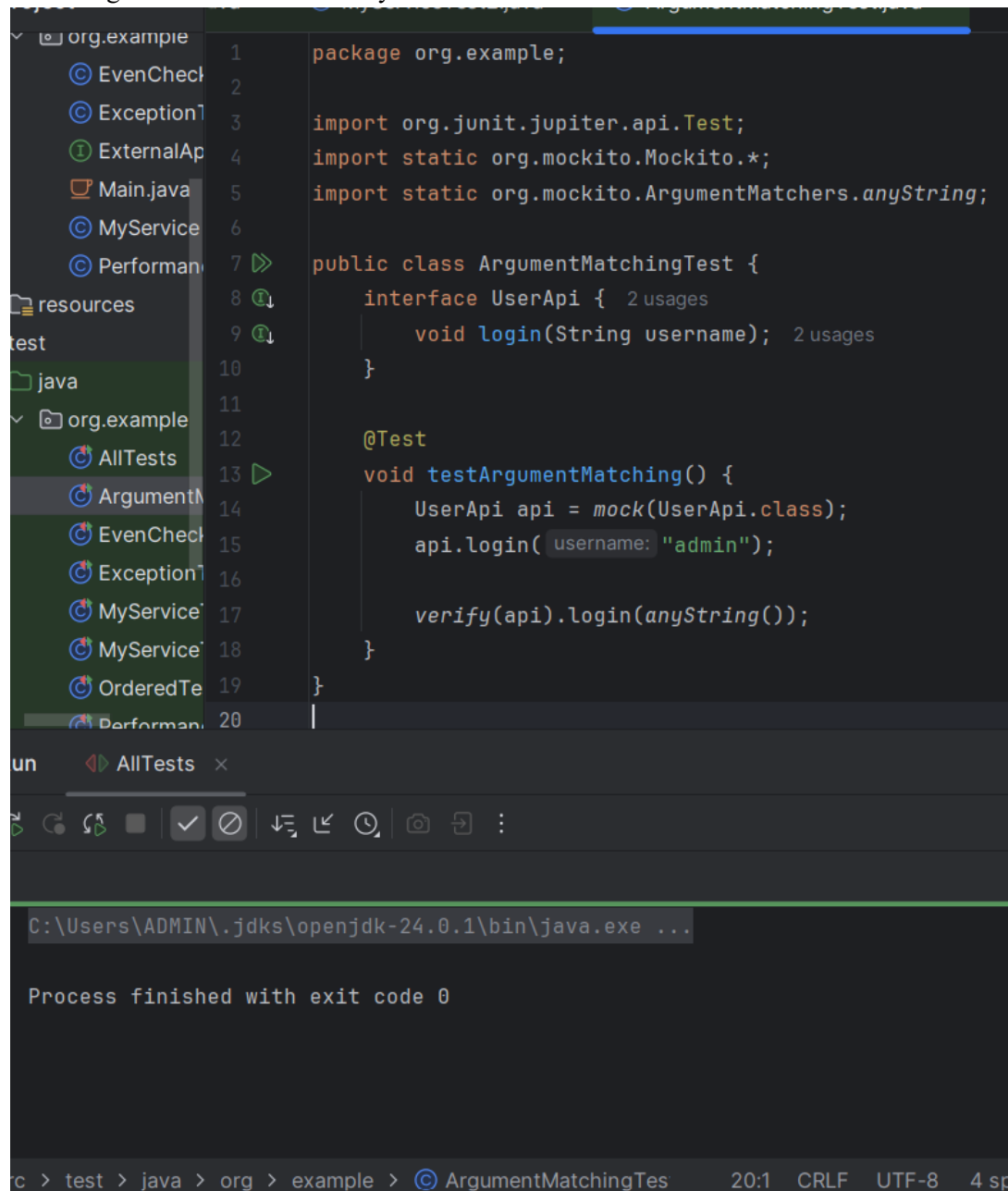
Exercise 3: Argument Matching

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.



The screenshot shows an IDE with a project named 'org.example'. The left sidebar displays a file tree with folders 'resources' and 'test', and a list of files including 'Main.java', 'MyService', 'Performance', 'AllTests', 'ArgumentM', 'EvenCheck', 'ExceptionT', 'MyService', 'MyService', 'OrderedTe', and 'Performan'. The main editor window shows the code for 'ArgumentMatchingTest.java'.

```
1 package org.example;
2
3 import org.junit.jupiter.api.Test;
4 import static org.mockito.Mockito.*;
5 import static org.mockito.ArgumentMatchers.anyString;
6
7 public class ArgumentMatchingTest {
8     interface UserApi { 2 usages
9         void login(String username); 2 usages
10    }
11
12    @Test
13    void testArgumentMatching() {
14        UserApi api = mock(UserApi.class);
15        api.login( username: "admin");
16
17        verify(api).login(anyString());
18    }
19 }
20
```

Below the code editor, the 'AllTests' test runner is shown. The output indicates that the process finished with exit code 0.

```
C:\Users\ADMIN\.jdk\openjdk-24.0.1\bin\java.exe ...
Process finished with exit code 0
```

The status bar at the bottom shows the file path 'c > test > java > org > example > ArgumentMatchingTes', the line number '20:1', the encoding 'CRLF', the character set 'UTF-8', and the file size '4 st'.

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.

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.

Exercise 6: Verifying Interaction Order

Scenario:

You need to ensure that methods are called in a specific order.

Steps:

1. Create a mock object.
2. Call the methods in a specific order.
3. Verify the interaction order.

Exercise 7: Handling Void Methods with Exceptions

Scenario:

You need to test a void method that throws an exception.

Steps:

1. Create a mock object.
2. Stub the void method to throw an exception.
3. Verify the interaction.