Part 2: Mockito to fully mock the DAO layer

Posted on September 9, 2015 by Arulkumaran Kumaraswamipillai

This extends Part 1: Unit testing with JUnit, Mockito & Spring by mocking the DAO layer with the Mockito framework.

Step 1: Service and DAO layer interfaces and implementations

Service Layer

```
1 package com.mytutorial;
2
3 public interface SimpleService {
4     abstract String processUser(int id);
5 }
```

```
1  package com.mytutorial;
2
3  import javax.inject.Inject;
4  import javax.inject.Named;
5
6  @Named
7  public class SimpleServiceImpl implements Simple
8
9      @Inject
```

600+ Full Stack Java/JEE Interview Q&As ♥Free ♦FAQs

open all | close all

- Ice Breaker Interview
- Core Java Interview C
- JEE Interview Q&A (3
- Pressed for time? Jav
- SQL, XML, UML, JSC
- Hadoop & BigData Int
- Java Architecture Inte
- Scala Interview Q&As
- Spring, Hibernate, & M
  - Spring (18)
  - Hibernate (13)
  - AngularJS (2)
  - Git & SVN (6)
  - JMeter (2)
  - JSF (2)
  - Maven (3)
- Testing & Profiling/Sa
  - Automation Testing
  - Code Coverage (2)
  - Code Quality (2)
  - jvisualvm profiling (
  - Performance Testir

```
10      SimpleDao dao;
11
12      public String processUser(int id) {
13          String name = dao.getNameById(id);
14          return "Hello " + name;
15      }
16  }
```

DAO Layer

This layer will be mocked by Mockito in the next step. This means it simply creates a skeletal instance of the Class "SimpleDao" , which gets entirely instrumented to track interactions with it.

```
1  package com.mytutorial;
2
3  public interface SimpleDao {
4      abstract String getNameById(int id);
5  }
```

```
1  package com.mytutorial;
2
3  import javax.inject.Named;
4
5  @Named
6  public class SimpleDaoImpl implements SimpleDao
7
8      public String getNameById(int id) {
9          if(id == 1) {
10             return "John";
11         }
12         else if (id == 2) {
13             return "Peter";
14         }
15         else {
16             throw new IllegalArgumentException()
17         }
18     }
19 }
```

Step 2: Now the JUnit class that mocks the Dao layer by returning "Sam" as the value. This test only tests the service layer method "processUser", which is a **unit**. It checks if "Hello " gets prefixed to the given user. This can be done two ways.

With MockitoJUnitRunner

```
1  package com.mytutorial;
```


16 Technical Key Areas

```
2
3  import org.junit.Assert;
4  import org.junit.Test;
5  import org.junit.runner.RunWith;
6  import org.mockito.InjectMocks;
7  import org.mockito.Mock;
8  import org.mockito.Mockito;
9  import org.mockito.runners.MockitoJUnitRunner;
10
11 @RunWith(MockitoJUnitRunner.class)
12 public class SimpleTest {
13
14     @InjectMocks
15     SimpleService service = new SimpleServiceImp
16
17     @Mock
18     SimpleDao daoMock;
19
20     @Test
21     public void testProcessUser() {
22         //invoke this mocked Dao that always ret
23         Mockito.when(daoMock.getNameById(Mockito
24
25         String processUser = service.processUser
26
27         Assert.assertEquals("Hello Sam", process
28         //verifies if getNameById is called
29         Mockito.verify(daoMock).getNameById(Mock
30         //verifies if getNameById is called once
31         Mockito.verify(daoMock, Mockito.times(1)
32     }
33 }
34
35
```

With SpringJUnit4ClassRunner.class

```
1  package com.mytutorial;
2
3  import javax.inject.Inject;
4
5  import org.junit.Assert;
6  import org.junit.Before;
7  import org.junit.Test;
8  import org.junit.runner.RunWith;
9  import org.mockito.InjectMocks;
10 import org.mockito.Mock;
11 import org.mockito.Mockito;
12 import org.mockito.MockitoAnnotations;
13 import org.springframework.test.context.ContextC
14 import org.springframework.test.context.junit4.S
15
16 @ContextConfiguration(classes = { AppConfig.clas
17 @RunWith(SpringJUnit4ClassRunner.class)
18 public class SimpleTest {
19
20     @Inject
21     @InjectMocks
22     SimpleService service;
23
24     @Mock
```

```
25      SimpleDao daoMock;
26
27      @Before
28      public void setUp() {
29          MockitoAnnotations.initMocks(this);
30      }
31
32      @Test
33      public void testProcessUser(){
34          //invoke this mocked Dao that always ret
35          Mockito.when(daoMock.getNameById(Mockito
36
37          String processUser = service.processUser
38
39          Assert.assertEquals("Hello Sam", process
40          //verifies if getNameById is called
41          Mockito.verify(daoMock).getNameById(Mock
42          //verifies if getNameById is called once
43          Mockito.verify(daoMock, Mockito.times(1)
44      }
45  }
```

Note:

With Mockito you can verify if the method “getNameById” has been invoked on the “daoMock” and also how many times been invoked. This is what separates a mock object from a stub. You can also have other verify options like

```
1  Mockito.verifyNoMoreInteractions(daoMock);
```

In this example, “daoMock” was fully mocked. You can also partially mock and object with “Mockito.spy”, which will be covered in the next part.

Mock Vs Spy difference?

When Mockito creates a mock, it does it from the class type (i.e. SimpleDao), and NOT from an actual instance. The mock simply creates a skeletal instance of the Class, which gets entirely instrumented to track interactions with it. On the other hand, the **spy** wraps an existing instance.

Popular Member Posts

♦ 11 Spring boot interview questions & answers

906 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

816 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

427 views

18 Java scenarios based interview Questions and Answers

409 views

♦ 7 Java debugging interview questions & answers

324 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

312 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

304 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

301 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

251 views

♦ Object equals Vs == and pass by reference Vs value

234 views

Bio | Latest Posts

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java

interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

‹ Part 1: Unit testing with JUnit, Mockito & Spring

Dozer bean mapping tutorial ›

Posted in JUnit Mockito Spring, member-paid

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

☀ Turn readers of your Java CV go from “Blah blah” to “Wow”? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category

© Disclaimer

The contents in this Java-Success are copy righted. The author has the right to correct or enhance the current content without any prior notice.

These are general advice only, and one needs to take his/her own circumstances into consideration. The author will not be held liable for any damages caused or alleged to be caused either directly or indirectly by these materials and resources. Any trademarked names or labels used in this blog remain the property of their respective trademark owners. No guarantees are made regarding the accuracy or usefulness of content, though I do make an effort to be accurate. Links to external sites do not imply endorsement of the linked-to sites.

↑