ABOUT MOHAMMAD MERAJ ZIA

I did my Engineering in Information Technology from IET, Lucknow, India. Currently doing MSc in Information Technology from Derby University. I have worked in Java/J2EE domain for the last 10 years. Have good understanding of Payment and Finance domains.

Mockito Mock Database Connection Example

Posted by: Mohammad Meraj Zia in Mockito August 31st, 2016

A unit test should test a class in isolation. Side effects from other classes or the system should be eliminated if possible. Mockito lets you write beautiful tests with a clean & simple API. In this example we will learn how to write a simple test case using Mockito. Tools and technologies used in this example are Java 1.8, Eclipse Luna 4.4.2

1. Introduction

Mockito is a popular mocking framework which can be used in conjunction with JUnit. Mockito allows us to create and configure mock objects. Using Mockito simplifies the development of tests for classes with external dependencies significantly. We can create the mock objects manually or we can use the mocking framewors like Mockito, EasyMock. jMock etc. Mock frameworks allow us to create mock objects at runtime and define their behavior. The classical example for a mock object is a data provider. In production, a real database is used, but for testing a mock object simulates the database and ensures that the test conditions are always the same.

2. Creating a project

Below are the steps required to create the project.

- Open Eclipse. Go to File=>New=>Java Project. In the 'Project name' enter 'MockitoMockDatabaseConnection'.

Figure 1. New Java Project

- Eclipse will create a 'src' folder. Right click on the 'src' folder and choose New=>Package. In the 'Name' text-box enter 'com.javacodegeeks'. Click 'Finish'.

Figure 2. New Java Package

- Right click on the package and choose New=>Class. Give the class name and click 'Finish'. Eclipse will create a default class with the given name.

2.1 Declaring mockito dependency

For this example we need the junit and mockito jars. These jars can be downloaded from Maven repository. We are using 'junit-4.12.jar' and 'mockito-all-1.10.19.jar'. There are the latests versions available as per now. To add these jars in the classpath right click on the project and choose Build Path=>Configure Build Path. Then click on the 'Add External JARs' button on the right hand side. Then go to the location where you have downloaded these jars and click ok.

If you are using Gradle you can do:

```
1 repositories { jcenter() }
```

```
dependencies { testCompile 'org.mockito:mockito-core:1.+' }
```

3. Code

There are two ways which we can use to mock the database connection. The first one is by mocking the

```
java.sql
```

classes itself and the second way is by mocking the Data Access Objects (DAO) classes which talks to the database. First we will see how we can mock the

```
java.sql
```

classes directly.

First we will create a class which will be responsible for connecting to the database and running the queries. All the Service/DAO classes will talk to this class. We will define two methods in this class. The first method will be responsible for creating the database session:

```
Class.forName("com.mysql.jdbc.Driver");
dbConnection = DriverManager.getConnection("jdbc:mysql://localhost:6666/jcg", "root",
"password");
```

The second method will be responsible for running the query.

DBConnection.java

```
package com.javacodegeeks;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBConnection {

  private Connection dbConnection;

  public void getDBConnection() throws ClassNotFoundException, SQLException {
    Class.forName("com.mysql.jdbc.Driver");
    dbConnection = DriverManager.getConnection("jdbc:mysql://localhost:6666/jcg", "root",
"password");
  }

  public int executeQuery(String query) throws ClassNotFoundException, SQLException {
    return dbConnection.createStatement().executeUpdate(query);
  }
}
```

Now we will write the test and see how we can make use of Mockito to mock the database connection.

DBConnectionTest.java

```
package com.javacodegeeks;

import java.sql.Connection;
import java.sql.Statement;

import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.Mockito;
import org.mockito.MockitoAnnotations;

public class DBConnectionTest {

  @InjectMocks private DBConnection dbConnection;
  @Mock private Connection mockConnection;
  @Mock private Statement mockStatement;

  @Before
  public void setUp() {
    MockitoAnnotations.initMocks(this);
  }

  @Test
  public void testMockDBConnection() throws Exception {
    Mockito.when(mockConnection.createStatement()).thenReturn(mockStatement);
    Mockito.when(mockConnection.createStatement().executeUpdate(Mockito.any())).thenReturn(1);
    int value = dbConnection.executeQuery("");
    Assert.assertEquals(value, 1);
    Mockito.verify(mockConnection.createStatement(), Mockito.times(1));
  }
}
```

Here we have annotated the DBConnection class with

```
@InjectMocks
```

annotation. This annotation marks a field on which injection need to be performed. The

```
Connection
```

and

`Statement`

classes of

`java.sql`

package are annotated with

`@Mock`

. In the setUp method we will call the

`initMocks()`

method. This Initializes objects annotated with Mockito annotations for given test class. Will have mocked the call to the

`executeUpdate()`

method by using the Mockito's

`when()`

method as below:

```
1 Mockito.when(mockConnection.createStatement().executeUpdate(Mockito.any())).thenReturn(1);
```

Now we will see how to mock DAO classes. First we will define the DAO class. This class will has just the method which always throws

`UnsupportedOperationException`

MyDao.java

```
1 package com.javacodegeeks;
2
3 public class MyDao {
4
5   public MyEntity findById(long id) {
6     throw new UnsupportedOperationException();
7   }
8 }
```

Now we will define the Entity class which this method in DAO returns:

MyEntity.java

```
01 package com.javacodegeeks;
02
03 public class MyEntity {
04
05   private String firstName;
06   private String surname;
07
08   public String getFirstName() {
09     return firstName;
10   }
11
12   public void setFirstName(String firstName) {
13     this.firstName = firstName;
14   }
15
16   public String getSurname() {
17     return surname;
18   }
19
20   public void setSurname(String surname) {
21     this.surname = surname;
22   }
23 }
```

Now we will define the Service class which has the reference to this DAO:

MyService.java

```
01 package com.javacodegeeks;
02
03 public class MyService {
04
05   private MyDao myDao;
06
07   public MyService(MyDao myDao) {
08     this.myDao = myDao;
09   }
10
11   public MyEntity findById(long id) {
12     return myDao.findById(id);
13   }
14 }
```

Now we will create a test class which will mock the MyDao class. In the first test we will verify that when we call the method of the service class (which in turn calls the DAO) the mock object has been called. We will do this by making use of the

`verify()`

method of the Mockito class.

```
MyService myService = new MyService(myDao);
myService.findById(1L);
Mockito.verify(myDao).findById(1L);
```

In the second test we will create an entity object and will verify the results as below:

```
MyService myService = new MyService(myDao);
Mockito.when(myDao.findById(1L)).thenReturn(createTestEntity());
MyEntity actual = myService.findById(1L);
Assert.assertEquals("My first name", actual.getFirstName());
Assert.assertEquals("My surname", actual.getSurname());
Mockito.verify(myDao).findById(1L);
```

MyServiceTest.java

```
package com.javacodegeeks;

import org.junit.Assert;
import org.junit.Rule;
import org.junit.Test;
import org.mockito.Mock;
import org.mockito.Mockito;
import org.mockito.MockitoAnnotations;
import org.mockito.junit.MockitoJUnit;
import org.mockito.junit.MockitoRule;

public class MyServiceTest {

  @Mock private MyDao myDao;

  @Rule public MockitoRule rule = MockitoJUnit.rule();

  @Test
  public void testFindById() {
    MockitoAnnotations.initMocks(this);
    MyService myService = new MyService(myDao);
    myService.findById(1L);
    Mockito.verify(myDao).findById(1L);
}

  @Test
  public void test() {
    MyService myService = new MyService(myDao);
    Mockito.when(myDao.findById(1L)).thenReturn(createTestEntity());
    MyEntity actual = myService.findById(1L);
    Assert.assertEquals("My first name", actual.getFirstName());
    Assert.assertEquals("My surname", actual.getSurname());
    Mockito.verify(myDao).findById(1L);
}

  private MyEntity createTestEntity() {
    MyEntity myEntity = new MyEntity();
    myEntity.setFirstName("My first name");
    myEntity.setSurname("My surname");
    return myEntity;
  }
}
```

4. Download the source file

This was an example of mocking database connection using Mockito.

Download
You can download the full source code of this example here: **MockitoMockDatabaseConnection**

Tagged with: MOCKITO