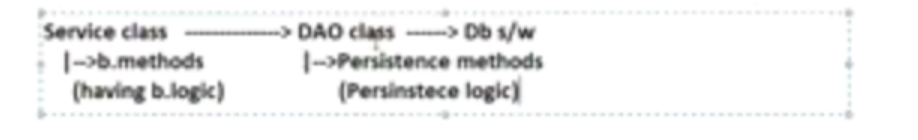
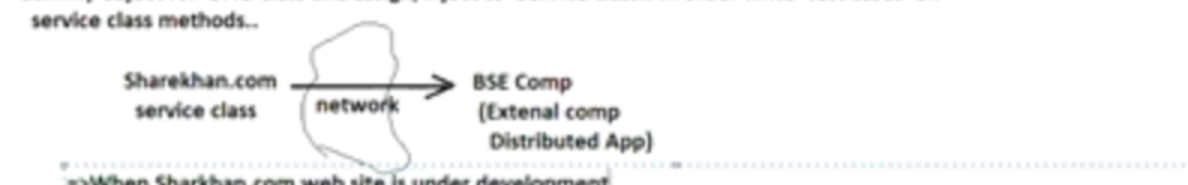
Mockito:

It is build on the top of Junit tool

It is given to perform unit testing by mocking the Local Dependent or external dependent objs.



Let us assume, DAO class coding is not completed. But service class coding is completed. We want to finish unit testing of service class. Then we need to create Mock obj/Fake obj/dummy obj for DAO class and assign or inject to service class in order to write test cases on service class methods.



When sharekhan.com website is under development. We cannot take subscription of BSE Comp because they charge huge money for that. Generally this subscription will be taken after hosting or releasing the sharekhan.com. Till that time we need to take mock BSE component and assign to Service class of sharekhan.com to perform unit testing.

Note: Mock objects are for temporary needs, mostly they will be used in the unit testing.

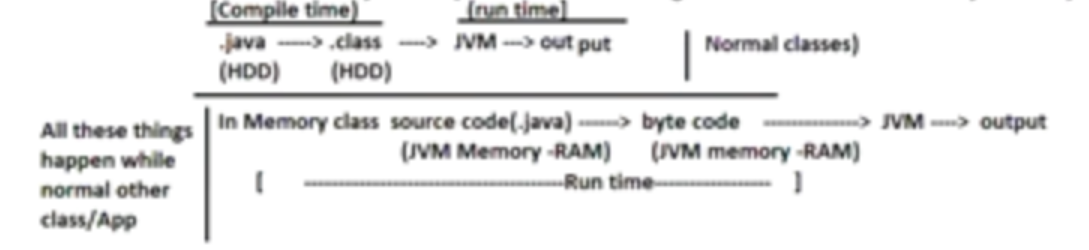
These mock objects created in test methods or test case class does not effect real code.

We can do these mocking in three ways:

1. Mock object/fake object(provides Temporary object)
2. Using Stubs obj(ame like mock object but, providing some functionality fors the methods of mock obj like specifying for certain input certain output should come)—some king of mock object but having some functionality to perform
3. Using Spy obj(it called partial mock object/half object that means..if you provide new functionality to method that will be used otherwise real object will be used)

Note:While working with spy object we will be having real object also

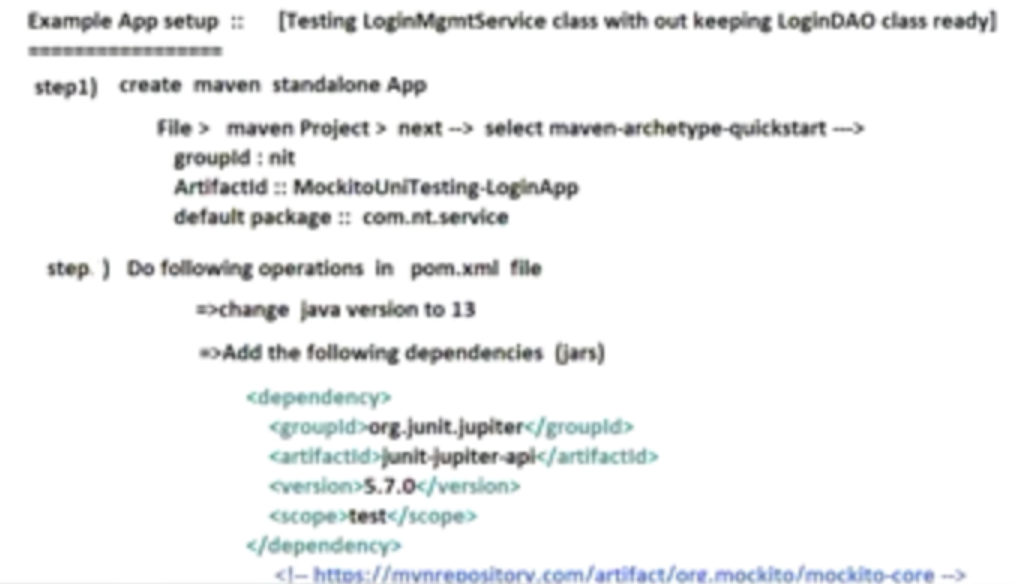
Instead of creating classes manually to prepare Mock,Stub and Spy objects. We can take mocking frameworks available in the market which are capable of generating such classes dynamically at runtime as in-memory classes(The classes that are generated in the JVM memory of RAM)



List of Mocking frameworks:

Mockito(Popular), JMockito, EasyMock, PowerMock etc..

Example application setup: Testing LoginMgmtService class without LoginDAO class ready



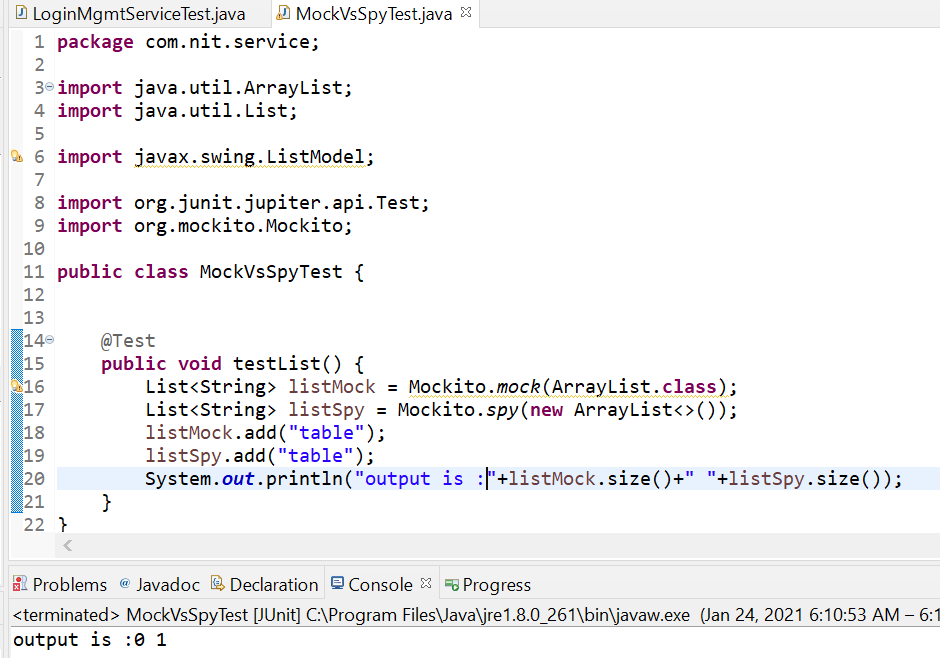


Step 3) Develop service interface, service class and DAO interface

Mock object with overridden functionality is stub.



When mocking is happened all the methods of that class implemented doing nothing.



**MockVsSpyTest.java:**

**package** com.nit.service;

**import** java.util.ArrayList;

**import** java.util.List;

**import** javax.swing.ListModel;

**import** org.junit.jupiter.api.Test;

**import** org.mockito.Mockito;

**public** **class** MockVsSpyTest {

@Test

**public** **void** testList() {

List<String> listMock = Mockito.*mock*(ArrayList.**class**);

List<String> listSpy = Mockito.*spy*(**new** ArrayList<>());

listMock.add("table");

listSpy.add("table");

System.***out***.println("output is : "+listMock.size()+" "+listSpy.size());

}

}

**Output:** *output is :0 1*

Mock object with overridden functionality is Stub.

**MockVsSpyTest.java:**

**package** com.nit.service;

**import** java.util.ArrayList;

**import** java.util.List;

**import** javax.swing.ListModel;

**import** org.junit.jupiter.api.Test;

**import** org.mockito.Mockito;

**public** **class** MockVsSpyTest {

@Test

**public** **void** testList() {

List<String> listMock = Mockito.*mock*(ArrayList.**class**); //Mock

List<String> listSpy = Mockito.*spy*(**new** ArrayList<>()); //Spy

listMock.add("table");

listSpy.add("table");

Mockito.*when*(listMock.size()).thenReturn(10);

System.***out***.println("output is : "+listMock.size()+" "+listSpy.size());

}

}

Output: output is : 10 1

***MockVsSpyTest.java:***

**package** com.nit.service;

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

**import** java.util.ArrayList;

**import** java.util.List;

**import** javax.swing.ListModel;

**import** org.junit.jupiter.api.Test;

**import** org.mockito.Mockito;

**public** **class** MockVsSpyTest {

@Test

**public** **void** testList() {

List<String> listMock = Mockito.*mock*(ArrayList.**class**); //Mock

List<String> listSpy = Mockito.*spy*(**new** ArrayList<>()); //Spy

// List<String> listSpy = Mockito.spy(ArrayList.class); //Spy

listMock.add("table");

Mockito.*when*(listMock.size()).thenReturn(10); //Stub on Mock obj

listSpy.add("table");

Mockito.*when*(listSpy.size()).thenReturn(10); //Stub on Spy obj

System.***out***.println("output is : "+listMock.size()+" "+listSpy.size());

}

}

***Output:***  *output is : 10 10 -🡪 If Mockito.when(..) statements are not commented*

***Output:***  *output is : 0 1 -🡪 If Mockito.when(..) statements are commented*

*Note: Spy objects are useful to check how many times methods are called?.. whether they are called or not because Spy object is always linked with real object.(for this use Mockito verify(-,-) method)*

***Mock VS Stub Vs Spy:***

Mock method will give mock object by creating In-Memory class implementing method with no functionalities.

If we want to provide functionality we need to go for Stub.

Spy creates mock object linked with real object. If we don’t provide stub functionality for the method it internally uses real object.

***Mockito Annotations:***

***@Mock 🡪 to generate mock object***

***@Spy 🡪 to generate spy object***

***@InjectMocks 🡪 to inject mock or spy objects to service class***

MockitoAnnotations.*openMocks*(**this**);

*Call above method in @before or constructor TestCase Class in order to activate Mockito annotations*

**public** **class** LoginMgmtServiceAnnoTest {

//@Mock gives mock object..@InjectMocks to this service class(loginService) other mock objects will be injected

@InjectMocks

**private** LoginMgmtServiceImple loginService;

@Mock

**private** ILoginDAO loginDAOMock;

// @Spy

// private ILoginDAO loginDAOSpy;

LoginMgmtServiceAnnoTest(){

MockitoAnnotations.*openMocks*(**this**);

}

}

To write stub functionality according to agile user stories/JIRA (given..when…then)

BDDMockito.*given*(loginDAOMock.authenticate("raja", "rani")).willReturn(1);

Or

Mockito.*when*(loginDAOMock.authenticate("raja", "rani")).thenReturn(1);

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------