Mockito

Mock means sample or duplicate or proxy or trail or alternate or dummy

Mockito

=======

(It is built on the top of Junit Tool)

=>It is given to perform unit testing by mocking the Local Dependent or extenal Dependent objs..

usecase1::

Service class

|-->b.methods

(having b.logic)

---> DAO class ------> Db s/w

-->Persistence methods (Persinstece logic)

class

Let us assume DAO class coding is not completed, But Service coding is completed and

to

we want finish unit testing of service class.. Then we need to create Mock obj/Fake obj/ dummy object for DAO class and assign/inject to Service class.. In order write test cases on service class methods..

Service classs :: JAva class with b.nethods having b.logic DAO class:: JAva class with Persistence logic

(CURD operations on DB table)

usecase2:

Sharekhan.com

service class

network

BSE Comp (Extenal comp Distributed App)

BSE: Bombay Stock Exchage

=>When Sharkhan.com website is under development, we can not take subscription of BSE Comp becoz they charge huge money for that.. Generally this Subscription will be taken after hosting/releasing the sharekhan.com. Till that time we need to take one mock BSE comp and assign to Service class of Sharekhan.com to perform UnitTesting..

note:: Mock objs are for temporary needs, mostly they will be used in the Unit Testing... Thse mock objs created in test methods or Test case class

We can do this mocking in 3 ways

a) Using Mock obj/Fake obj (Provides Temporary object)

but not in the real

code like

service, DAO, controller classes

b) Using Stub obj (Providing some Functionality for the methods of mock obj like specifying for certain inputs certain output should come) c) Using Spy object (It is called Partial Mock obj/ Half mock obj that means

if u provide few functionlity to method that will be used.. otherwise

real object functionality will be used)

we

note:: While working with Spy object will be having real object also.

=> Instead of creating classes manually to prepare Mock, Stub and Spy objects.. we can use mocking frameworks available in the market whic are capable generate such classes dynamically proxy at runtime as InMemory classes (The classes that are generated in the JVM memory of RAM) [Compile time)

.java -----> .class (HDD)

(run time]

Stub, Spy objects are extension of Mock object

note:: Defining the functionality for the methods of Mock obj is called making the Mock obj as Stud object.

note:: Spy Object is a mock object that acts as wrapper around the Real object where the new functionality of the method will execute if defined otherwise the old functionality will execute from the real obj

This class source code and byte is permanent becoz they allobcate memory in HDD (Hard Disk Drive) --> JVM output (JVM memory -RAM) (JVM Memory)

(HDD)

JVM ---> out put (JVM Memory of RAM)

Normal classes)

------>

(JVM Memory -RAM)

---Run time-------

In Memory class source code(.java) byte de

All these things

happen while

normal other class/App

[

List of Mocking Frameworks::

]

This class comes when the execution of the App begins and This class ends when the execution of the App completes (Every thing happens in JVM memory of the RAM)

All the Mocking frameworks are given on the top of Junit Framework

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

Example App setup :: [Testing LoginMgmtService class with out keeping LoginDAO class ready]

======

==========

step1) create maven standalone App

File > maven Project > next --> select maven-archetype-quickstart ---> groupId : nit

ArtifactId :: MockitoUniTesting-LoginApp

default package :: com.nt.service

step2) Do following operations in pom.xml file

=>change java version to 1 7

=>Add the following dependencies (jars)

We can do Mockito programming i.e creating Mock, stub and spy objects

in the test case classes of Junit in two approaches

a) Using Programatic Approach (The Mock objects will be created through Java code of Mockito API)

b) Using Annotation Approach (The Mock objects will be created through annotations) (Best)

```
<dependency>
<groupId>org.junit.jupiter</groupId>
<artifactId>junit-jupiter-api</artifactId>
<version>5.7.0</version>
<scope>test</scope>
</dependency>
<!-- https://mvnrepository.com/artifact/org.mockito/mockito-core -->
<dependency>
<groupId>org.mockito</groupId>
<artifactId>mockito-core</artifactId>
<version>3.6.28</version>
<scope>test</scope>
</dependency>
```

impl

step3) develop service interface, service class and DAO interface

@Test

com.nt.service

|--->|LoginMgmtService.java (1)

|--->LoginMgmtServiceImpl(c)

com.nt.dao

Enduser -------> LoginServiceMgmt ----------> LoginDAO -----> DB s/w

(Though the real LoginDAO class

is not ready, we should complete unit testing on LoginServiceMgmt

class with support Mock /Stub obj)

Step4) Develop Mickito based Test classes

step5) Run Tests...

src/main/java folder

Refer GIT for complete Application..

|--->LoginDAO.java

loginDAOMock=Mockito.mock(ILoginDAO.class); // mock(-) generates InMemory class implementing

// ILoginDAO(1) having null method definitations for authenticate(-,-) nethod

=> Mock obj is the just Proxy Obj and not linked with any real object

=> Spy Obj is the Mock obj that acts as the wrapper around the real obj

=> if u provide dummy functionality for the methods of Mock obj or Spy object then they becom stub object

public void testList() {

}

List<String> listMock-Mockito.mock(ArrayList.class); //Mock //List<String> listSpy-Mockito.spy(ArrayList.class); //Spy (or) List<String> listSpy=Mockito.spy(new ArrayList()); //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(listMock.size()+" "+listSpy.size());

10 0

10 if Mockito.when(...) statemetns are not commented

1

if Mockito.when(...) statemetns are

commented

note:: Spy objects are useful to check how many time methods are called?.. wheather they are called or not.. becoz Spy object is always linked with real object.. (for this use Mockito.verify(-,-) method

@Test

public void testRegisterWithSpy() {

ILoginDAO loginDAOSpy=Mockito.spy(ILoginDAO.class);

ILoginMgmtService loginSerivce=new LoginMgmtServiceImpl(loginDAOSpy);

loginSerivce.registerUser("raja", "admin");

loginSerivce.registerUser("suresh", "visitor");

loginSerivce.registerUser("jani", "");

Mockito.verify(loginDAOSpy,Mockito.times(1)).addUser("raja", "admin"); // check the above register method calls addUser(-,-) is called for 1 tome Mockito.verify(loginDAOSpy,Mockito.times(0)).addUser("suresh", "visitor");

Mockito.verify(login DAOSpy,Mockito.never()).addUser("jani", "");

internally or not for the inputs "raja", "admin"

}

Mockito Annotations

==========

@Mock --> to Genernat mock object

@Spy -->To geneate spy object

@InjectMocks-->To Inject Mock or Spy Objects Service class.

MockitoAnnotations.openMocks(this); -->call this method in @Before or constructo TestCase class in order to activate Mockito Annotations.. note:: For @Mock object if we provide dummy functionality for the methods //Test case class

public class LoginMgmtServiceTestsAnno {

@InjectMocks //create the service class object private LoginMgmtServiceImpl loginService;

@Mock // create DAO class mock object

private ILoginDAO loginDAOMock;

/*@Spy

then the @Mock obj becomes Stub Object

note: Annotations are given for MetaData, where we perform code and about code configurations to additional instrutions or inputs or information to compilers /JRES / containers/frameworks/servers and etc..

As of now, java Programming is preferring two concepts for meta data operations (for code about Code operations) a) xml (outdated, it is separate language to learn)

b) annotations (latest and doing well so best)

}

private ILoginDAO loginDAOSpy;*/

public LoginMgmtServiceTestsAnno() {

MockitoAnnotations.openMocks(this);

}

Initializes objects annotated with Mockito annotations for given testClass: @org.mockito.Mock, @Spy, @Captor, @InjectMocks

To write stub functionalify according agile user stories/JIRA user stories (given.. when.. then..)

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

⇓ ***equal to***

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

note::

user stories of agile or JIRA can be used as the base to create testcases (@Test methods) in TestCase class

Example App Code

MockitoProj01-LoginApp

#src/main/java

com.nt.dao

> ILoginDAO.java

com.nt.service

>LoginMgmtService.java

> LoginMgmtServiceImpl.java

src/test/java

com.nt.tests

› Login TestsWithMockito.java

> Login TestsWithMockitoAnnotations.java

> Mock StubVsSpyTest.java

> JRE System Library [JavaSE-17]

>junit-jupiter-api-5.10.0.jar - C:\Users\NATARAJ\.m2\repository\org\junit\jupite

✓

Maven Dependencies

>

opentest4j-1.3.0.jar - C:\Users\NATARAJ\.m2\repository\org\opentest4j\opente

>

>

junit-platform-commons-1.10.0.jar - C:\Users\NATARAJ\.m2\repository\org\ju

apiguardian-api-1.1.2.jar - C:\Users\NATARAJ\.m2\repository\org\apiguardian

> mockito-core-5.5.0.jar - C:\Users\NATARAJ\.m2\repository\org\mockito\mock

> byte-buddy-1.14.6.jar - C:\Users\NATARAJ\.m2\repository\net\bytebuddy\byte

>

byte-buddy-agent-1.14.6.jar - C:\Users\NATARAJ\.m2\repository\net\bytebudc

> objenesis-3.3.jar - C:\Users\NATARAJ\.m2\repository\org\objenesis\objenesis\:

> src

> target

pom.xml

```
//ILoginDAO.java
package com.nt.dao;
public interface ILoginDAO {
public int authenticate(String username, String password);
public String addUser(String username, String role);
}
//ILoginMgmtService.java
package com.nt.service;
public interface ILoginMgmtService {
public boolean login(String user,String pwd);
public String registerUser(String user,String role);
}
//service Impl class
package com.nt.service;
import com.nt.dao.ILoginDAO;
public class Login MgmtServiceImpl implements ILoginMgmtService {
private ILoginDAO dao;
public LoginMgmtServiceImpl(|LoginDAO dao) {
```

```
this.dao=dao;
}
@Override
public boolean login(String user, String pwd) {
if(user.equalsIgnoreCase("") || pwd.equalsIgnoreCase(""))
throw new IllegalArgumentException("Invalid inputs");
//use DAO
int count=dao.authenticate(user, pwd);
artturn falsenl
else
return true;
}
@Override
public String registerUser(String user, String role) {
if(!user.equals("") && !role.equals("")) {
dao.addUser(user, role);
//use DAO
return "user added";
}
else
return "user not added";
}
}
package com.nt.tests;
import static org.junit.jupiter.api.Assertions.assertFalse; import static org.junit.jupiter.api.Assertions.assertThrows;
import static org.junit.jupiter.api.Assertions.assertTrue;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import org.mockito.Mockito;
import com.nt.dao.ILoginDAO;
import com.nt.service.I LoginMgmtService;
import com.nt.service.Login MgmtServiceImpl;
// Mockito in Programatic Approach
public class Login Tests With Mockito {
private static ILoginMgmtService loginService;
private static ILoginDAO login DAO;
```

```
@BeforeAll
public static void setUp() {
// Mock the LoginDAO
loginDAO=Mockito.mock(ILoginDAO.class);
//create Service class obj
System.out.println("Mock object class nanme::"+loginDAO.getClass());
loginService=new LoginMgmtServiceImpl(loginDAO);
}
@AfterAll
public static void clearDown() {
}
@Test
loginDAO=null;
loginService=null;
public void test Login With Valid Credentials() {
// provide some funcationality for autheticate method on the mock DAO object (Stub object)
Mockito.when(loginDAO.authenticate("raja", "rani")).thenReturn(1);
// perform testing
}.
assertTrue(loginService.login("raja", "rani"));
@Test
public void testLoginWithInValid Credentials() {
// provide some funcationality for autheticate method on the mock DAO object (Stub object)
Mockito.when(loginDAO.authenticate("raja", "rani1")).thenReturn(0);
assertFalse(loginService.login("raja", "rani1"));
// perform testing
}
@Test
public void testLoginWithNoCrendentials() {
}
assertThrows(IllegalArgumentException.class, ()->loginService.login("",""));
@Test
public void testRegisterUser() {
loginService.registerUser("raja", "rani");
loginService.registerUser("anil", "hyd");
loginService.registerUser("chari", "");
Mockito.verify(login Dao, Mockito.times(1)).addUser("raja", "rani");
```

```
Mockito.verify(login Dao, Mockito.times(1)).addUser("anil", "hyd");
}
Mockito.verify(login Dao, Mockito.times(0)).addUser("chari", "");
package com.nt.tests;
import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertThrows;
import static org.junit.jupiter.api.Assertions.assertTrue;
import org.junit.jupiter.api.Test;
import org.mockito.Inject Mocks;
import org.mockito.Mock;
import org.mockito.Mockito;
import org.mockito.MockitoAnnotations;
import com.nt.dao.ILogin DAO;
import com.nt.service.ILoginMgmtService;
import com.nt.service.LoginMgmtServiceImpl;
//Mockito in Programatic Approach
public class Login TestsWithMockitoAnnotations
{
@InjectMocks // (creates the Service class obj) and also inject mockDAO class obj
private static LoginMgmtServiceImpl loginService;
@Mock // creates the MockDAO class obj
private ILoginDAO loginDAO;
public Login TestsWithMockitoAnnotations() {
MockitoAnnotations.openMocks(this);
Keeps the current setup ready for the
}
annotations
public void testLogin With ValidCredentials() {
// provide some funcationality for autheticate method on the mock DAO object (Stub object)
Mockito.when(loginDAO.authenticate("raja", "rani")).thenReturn(1);
// perform testing
assertTrue(loginService.login("raja", "rani"));
}
@Test
public void testLoginWithInValid Credentials() {
// provide some funcationality for autheticate method on the mock DAO object (Stub object)
Mockito.when(login DAO.authenticate("raja", "rani1")).thenReturn(0);
```

```
// perform testing
assertFalse(loginService.login("raja", "rani1"));
}
@Test
public void testLoginWithNoCrendentials() {
}
assertThrows(IllegalArgumentException.class, ()->loginService.login("",""));
@Test
public void testRegisterUserWithSpy() {
//create Spy Object
}
ILoginDAO loginDAOSpy=Mockito.spy(ILoginDAO.class);
// create Service class object
ILoginMgmtService loginSerivce=new LoginMgmtServiceImpl(login DAOSpy);
//invoke the methods
loginSerivce.registerUser("raja", "admin");
loginSerivce.registerUser("suresh", "customer");
loginSerivce.registerUser("jani", "");
//Check weather addUser(-,-) is called expected no.of times or not
Mockito.verify(loginDAOSpy, Mockito.times(1)).addUser("raja", "admin");
Mockito.verify(login DAOSpy, Mockito.times(1)).addUser("suresh", "customer");
//Mockito.verify((bogin DAC SpyMotioneret)dsker(äjahi";');");
Mockito.verify(loginDAOSpy,Mockito.times(0)).addUser("jani", "");
}
package com.nt.tests;
import java.util.ArrayList;
import java.util.List;
import org.junit.jupiter.api.Test;
import org.mockito.Mockito;
public class Mock_StubVsSpyTest {
@Test
public void testList() {
List<String> listMock-Mockito.mock(ArrayList.class); //Mock
List<String> listSpy-Mockito.spy(ArrayList.class); //Spy (or) //List<String> listSpy-Mockito.spy(new
ArrayList()); //Spy (or)
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

//test the size method

System.out.println(listMock.size()+" "+listSpy.size());

giving

note:: Mock/Stub Object can not use real functionality for the methods if we not do provide mock functionalify the for methods is commented have eg:: if Mickito.when(-) on listMock obj.. the listMock.size() is Othough we added one item "table" to mock object beocz mock/stub object can not real funcationality of the size() when the mock functionality is not provided

use

note:: Spy Object can use the real functionality for the methods if we do not do provide mock functionality the for same methods

eg:: if Mickito.when(-) on listSpy obj is commented.. then listSpy.size() gives 1 becoz we have added one item "table" to mock object i.e spy object using real functionality

Example App on Mockito annotations

=====

==========

DAO Interface, Service Interface and Service Impl classes are same as previous App

MockitoProj01

> src/main/java

✓src/test/java

com.nt.serivce

AppTest.java

→ → AppTest

AppTest.java

=========

package com.nt.serivce;

D$ login Dao

D$ loginService

• setupOnce(): void

tearDownOnce() : void

testArrayListForMockVsStul

• testLoginWithInValidCreder

• testLoginWithNoCredential

⚫ testLoginWithValidCredenti

⚫ testRegisterUser(): void

> JRE System Library [JavaSE-21]

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertThrows;
import java.util.ArrayList;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.Test;
import org.mockito.Inject Mocks;
import org.mockito.Mock;
import org.mockito.Mockito;
› Maven Dependencies
> src
>
target
Mpom.xml
Mockito Annotations are
import org.mockito.MockitoAnnotations;
import org.mockito.Spy;
import com.nt.dao.ILoginDAO;
/**
* Unit test for simple App.
*/
public class AppTest {
@Inject Mocks
========
@Mock --->creates the Mock /proxy object based on given interface
@Spy --> creates the Spy object by taking the given real object as wrapper
object
@InjectMocks -> Injects Mock object to given object
What
is mocking?
private static LoginMgmtServiceImpl loginService;
@Mock
private static ILoginDAO login Dao;
@Mock
private ArrayList<String> listMock;
@Spy
private ArrayList<String> listSpy;
public AppTest() {
```

```
// Activate Annotations driven Mocking
MockitoAnnotations.openMocks(this);
@Test
public void testLogin With ValidCredentials() {
//provide functionality for the Mock obj method and make the obj as the Stud object
Mockito.when(loginDao.authenticate("raja", "rani")).thenReturn(1);
//invoke the method
String actualResult=loginService.login("raja", "rani");
assertEquals("Valid Credentials", actualResult);
}
@Test
public void testLoginWithInValid Credentials() {
//provide functionality for the Mock obj method and make the obj as the Stub object
Mockito.when(login Dao.authenticate("raja", "rani11")).thenReturn(0);
//invoke the method
String actualResult=loginService.login("raja", "rani11");
assertEquals("InValid Credentials", actualResult);
}
@Test
public void testLoginWithNoCredentials() {
}
assertThrows(IllegalArgumentException.class, ()->{loginService.login("","",""); });
@Test
public void testRegisterUser() {
}
loginService.registerUser("raja", "rani");
loginService.registerUser("anil", "hyd");
loginService.registerUser("chari", "");
Mockito.verify(login Dao, Mockito.times(1)).addUser("raja", "rani");
Mockito.verify(login Dao, Mockito.times(1)).addUser("anil", "hyd");
Mockito.verify(login Dao, Mockito.times(0)).addUser("chari", "");
@Test
public void testArrayListForMockVsStubVsSpy() {
//add times to mock, spy objects
}
listMock.add("table");
listSpy.add("table");
```

// define dummy functionality for for Mock, Spy objects (Stub objects)

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

//Mockito.when(listSpy.size()).thenReturn(10);

System.out.println(listMock.size()+"......." + listSpy.size());

@AfterAll

public static void tearDownOnce() {

loginDao=null;

loginService=null;

}

What is the need of creating mock objects in testing process?

Explain different use-cases where the mocking is required in real projects?

List of different mocking frameworks available in java ?

what is difference b/w Mock obj, Stub Object and Spy Object?

Explain different API methods (java statements) to create Mock, Stub and spy objects

Explain different API annotations to create Mock, Stub and spy objects?

What is use of @Mock and @Spy Annotations?

What is difference b/w Mock object and Stub object?

what is the difference b/w stub object and spy objects?

How to define functionality for Mock object to make it as Student obj?

How many approaches are the for Mockito Programming?

What is use of @Inject Mock annotation?

What is use of calling MockitoAnnotations.openMock(-) method?

How to define functionality on Mock object in Agile/JIRA user-story style?

What is the link b/w Junit tool and Mockito tool?

Explicitly setting up instrumentation for inline mocking (Java 21+)

==========

==========

Starting from Java 21, the JDK restricts the ability of libraries to attach a Java agent to their own JVM. As a result, the inline-mock-maker might not be able to mockito-core as a Java agent, and it may be more appropriate to choose a different approach depending on your project constraints.

function without an explicit setup to enable instrumentation, and the JVM will always display a warning. The following are examples about how to set up

step1) place these plugins in pom.xml file

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-dependency-plugin</artifactId>

<executions>

<execution>

<goals>

<goal>properties</goal>

</goals>

</execution>

</executions>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.5.2</version>

<configuration>

<argLine>

-javaagent:${org.mockito:mockito-core:jar}

-Xshare:off

</argLine>

</configuration>

</plugin>

step2) Add this dependency in pom.xml

<!-- https://mvnrepository.com/artifact/org.mockito/mockito-inline -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-inline</artifactId>

<version>5.2.0</version>

<scope>test</scope> </dependency>