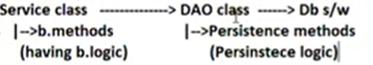
**1.Introduction:-**  The mockito is built on top of Junit Tool. The mockito is given to perform unit testing by mocking the local dependent or external dependent object.



Service class developer completed service class logic. He want to do unit test on service class and he want to go to another work. By the time, DAO class developer does not complete writing logic by his personal issues or by DB issues.

At this time, Service class developer creates mock object for DAO class to do unit testing on service class.

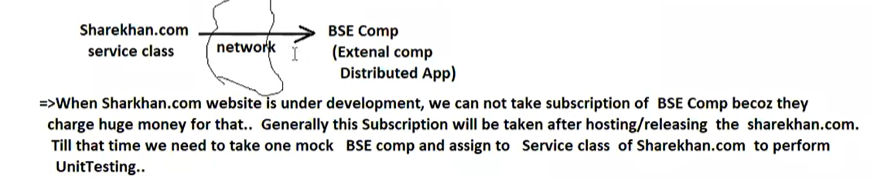
The mock objects are created in “test methods”.

**MOCK:**-The mock means temporary dummy or temporary duplicate or fake which is not real or original.

Ex:- 1interview which is conducted to avoid phobio in the candiate and which is not real interview But to look it is like real interview is called Mock interview.

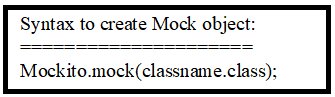
Ex:2 The exam which is not real exam but to look it is like real exam is called mock exam.

Real Time Example:-



We can do mocking in 3 ways:

1. Using Mock object/Fake obj



The mock method creates object for given class. This object have methods with empty body. That object will be returned.

1. Using Stub object :: It provides functionality for the method of mock object like specifying inputs and output should come)

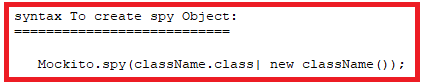


OR



1. Using spy object :: It is **partial mock object**. It provides new functionality to methods that will be used other wise real object methods functionality will be used.

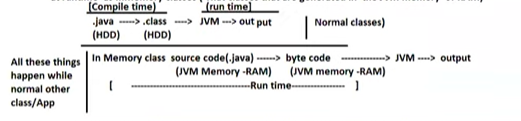
Note: while working with spy object, we should have real object also.



The Service class Developer has to do following activities **manually**:

* Defining Classes to prepare mock object,stub object and spy object
* Providing the functionality to methods of mock object
* …etc

Doing all above activites manually is complex process. The Service class developer need to use mocking frameworks available in the market which is capable to generate such classes dynamically as “InMemory classes”.



The List of mockito framework:

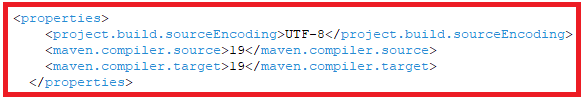
1. Mockito(Popular)
2. Jmocito
3. EasyMock
4. PowerMock
5. …etc.

**2. Example Application:: [Testing LoginMgmService class with out keeping LoginDAO class ready]**

Step1:- create maven standalone App

Step2:- Do following operations in pom.xml file

* Change the java version to current using version.



* Add Following dependencies (jars) . Get dependencies from mvnrepositary.com.

i)Junit5 Dependencies.

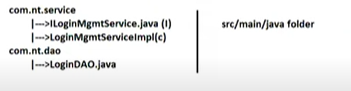
ii) Mockito dependencies



After Adding Update the maven project.

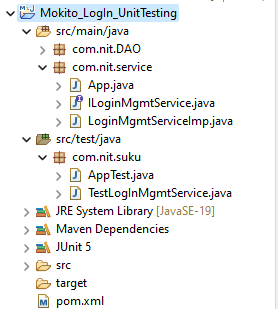
Projectname🡪rightclick🡪maven🡪update maven project.

Step3: develop the service interface, service class and DAO interface.



Project Directory structure:

===================



ILoginMgmtService.java

==================

**package** com.nit.service;

**public** **interface** ILoginMgmtService {

**public** **boolean** login(String uname,String pwd) ;

}

LogInMgmtServiceimp.java

=====================

**package** com.nit.service;

**import** com.nit.DAO.ILogInDao;

**public** **class** LoginMgmtServiceImp **implements** ILoginMgmtService{

**public** ILogInDao loginDao;

**public** LoginMgmtServiceImp(ILogInDao dao) {

**this**.loginDao=dao;

}

@Override

**public** **boolean** login(String uname, String pwd) {

**int** count=loginDao.Authenticate(uname, pwd);

**if** (count==1)

**return** **true**;

**else**

**return** **false**;

}

}

ILogInDao.java

===========

**package** com.nit.DAO;

**public** **interface** ILogInDao {

**public** **abstract** **int** Authenticate(String uname,String pwd);

}

TestLogInMgmtService.java

====================

**package** com.nit.suku;

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

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

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

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

**import** org.mockito.Mockito;

**import** com.nit.DAO.ILogInDao;

**import** com.nit.service.ILoginMgmtService;

**import** com.nit.service.LoginMgmtServiceImp;

@DisplayName("LoginTestClass-")

**public** **class** TestLogInMgmtService {

**private** **static** ILoginMgmtService *i1*;

**private** **static** ILogInDao *i2*;

@BeforeAll

**public** **static** **void** abc() {

//Create mock object for DAO class.

*i2*=Mockito.*mock*(ILogInDao.**class**);

*i1*=**new** LoginMgmtServiceImp(*i2*);

}

@Test

@DisplayName("Valid Credentials:")

**public** **void** testValidCredentials() {

//provide stub for the authenticate method of DAO object.

Mockito.*when*(*i2*.Authenticate("rama","sita")).thenReturn(1);

Assertions.*assertTrue*(*i1*.login("rama", "sita"));

}

@Test

@DisplayName("InValid Credentials:")

**public** **void** testInValidCredentials() {

//provide stub for the authenticate method of DAO object.

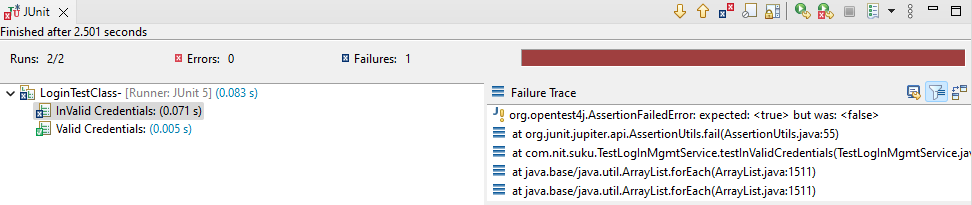
Mockito.*when*(*i2*.Authenticate("rama","sita")).thenReturn(1);

Assertions.*assertTrue*(*i1*.login("r", "s"));

}

}

Output:-



Example::3 Demonstrates spy object.

**package** com.nit.suku;

**import** java.util.ArrayList;

**import** java.util.List;

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

**import** org.mockito.Mockito;

**public** **class** UnitTestingWithSpyobject {

@Test

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

List<String>l1=Mockito.*spy*(ArrayList.**class**);

l1.add("suku");

System.***out***.println(l1.size());

List<String>l2=Mockito.*spy*(**new** ArrayList());

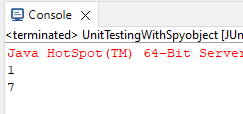
Mockito.*when*(l2.size()).thenReturn(7);

System.***out***.println(l2.size());

}

}

Output;



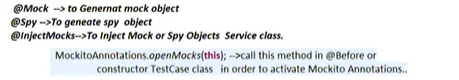
Explanation:

We did not provide functionality to size() method of First spy object. Therefore real object method logic has been used. We got 1.

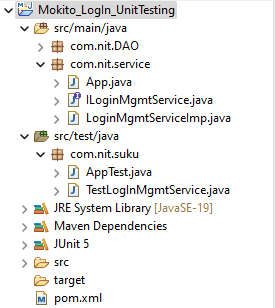
We provided functionality to size() method of second spy object. Therefore provided logic has been used instead of real object method logic. We got 7.

**4.Mockito With Objects:**

**===================**

****

**Example:**

****

ILoginMgmtService.java

==================

**package** com.nit.service;

**public** **interface** ILoginMgmtService {

**public** **boolean** login(String uname,String pwd) ;

}

LogInMgmtServiceimp.java

=====================

**package** com.nit.service;

**import** com.nit.DAO.ILogInDao;

**public** **class** LoginMgmtServiceImp **implements** ILoginMgmtService{

**public** ILogInDao loginDao;

**public** LoginMgmtServiceImp(ILogInDao dao) {

**this**.loginDao=dao;

}

@Override

**public** **boolean** login(String uname, String pwd) {

**int** count=loginDao.Authenticate(uname, pwd);

**if** (count==1)

**return** **true**;

**else**

**return** **false**;

}

}

ILogInDao.java

===========

**package** com.nit.DAO;

**public** **interface** ILogInDao {

**public** **abstract** **int** Authenticate(String uname,String pwd);

}

TestLogInMgmtService.java

====================

**package** com.nit.suku;

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

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

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

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

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

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

**import** org.mockito.InjectMocks;

**import** org.mockito.Mock;

**import** org.mockito.Mockito;

**import** org.mockito.MockitoAnnotations;

**import** com.nit.DAO.ILogInDao;

**import** com.nit.service.LoginMgmtServiceImp;

@DisplayName("LoginTestClass-")

@TestMethodOrder(value=MethodOrderer.OrderAnnotation.**class**)

**public** **class** TestLogInMgmtService {

@InjectMocks

**private** **static** LoginMgmtServiceImp *i1*;

@Mock

**private** **static** ILogInDao *i2*;

**public** TestLogInMgmtService() {

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

}

@Test

@DisplayName("Valid Credentials:")

@Order(1)

**public** **void** testValidCredentials() {

//provide stub for the authenticate method of DAO object.

Mockito.*when*(*i2*.Authenticate("rama","sita")).thenReturn(1);

Assertions.*assertTrue*(*i1*.login("rama", "sita"));

}

@Test

@DisplayName("InValid Credentials:")

@Order(2)

**public** **void** testInValidCredentials() {

//provide stub for the authenticate method of DAO object.

Mockito.*when*(*i2*.Authenticate("rama","sita")).thenReturn(1);

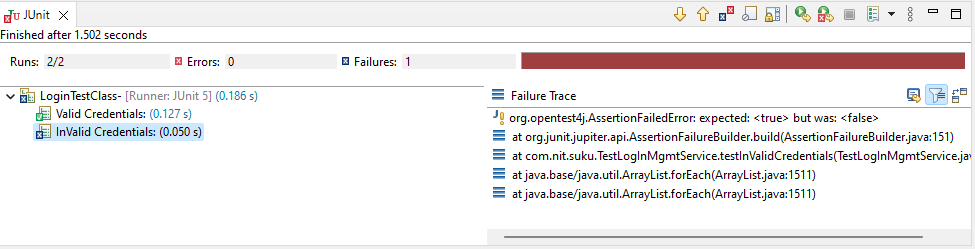
Assertions.*assertTrue*(*i1*.login("r", "s"));

}

}

**Output:**

**-----------**

****