Mockito framework - For Beginners

1.Overview of Mockito framework

Mockito is used to mock java classes for testing purposes.

Let us understand this with an example. Suppose you write an explanation which will need to connect to DB and update tables and also hit an http server, both of which are either cumbersome to set-up or are managed by someone else. Even then, you need to be able to test your application. Mockito, or other mocking frameworks like easymock come to your rescue.

When you mock a class, you are actually creating a dummy or mock of that class, and then re-defining method to bypass the actual db connection/http connection and instead return dummy responses or objects which you are expecting the db layer/http server to return anyways. If your application works fine with these dummy objects, then when the actual db layer/http server are plugged in, the application will work as expected.

Mocking frameworks are generally used with unit testing apis like junit to test applications modularly. Check the tutorials for junit4 and junit3.

2.Versions of Mockitoapi

The following are the major releases of mockito.

Versions of Mockitoapi

The following are the major releases of mockito.

Download Version	**Java compatibility**	**Release Date**
mockito-1.9.0	1.2 and later	Dec 2011
mockito-1.8.5	1.2 and later	May 2010
mockito-1.8.4	1.2 and later	Mar 2010
mockito-1.8.3	1.2 and later	Mar 2010
mockito-1.8.2	1.2 and later	Dec 2009

mockito-1.8.1	1.2 and later	Nov 2009
mockito-1.8.0	1.2 and later	Jul 2009
mockito-1.7	1.2 and later	Jan 2009
mockito-1.6	1.2 and later	Oct 2008
mockito-1.5	1.2 and later	Jul 2008
mockito-1.4	1.2 and later	Jun 2008
mockito-1.3	1.2 and later	Apr 2008
mockito-1.2	1.2 and later	Mar 2008
mockito-1.1	1.2 and later	Feb 2008
mockito-1.0	1.2 and later	Feb 2008
mockito-0.91	1.2 and later	Feb 2008
mockito-0.9	1.2 and later	Jan 2008

3.Requirements checklist : Artefacts needed to start using Mockitoapi

For using mockito, the only requirement is to have mockito-all-<version>.jar in the classpath.

Since, mocking is usually done for unit tests, you will usually also use a unit test api like JUnit, TestNG etc.

4.How to create a mock object

This example shows how to create a dummy or mock for an object.

A mock object of Calendar class is created by using the method mock(...) of classorg.mockito.Mockito.

The usage of mock objects is explained in examples on stubbing method in the next few pages.

```
package com.techfundaes.mockitoBag;

```

```java
import java.util.Calendar;

import static org.mockito.Mockito.*;

public class CreateMocks
{
    public static void main(String[] args)
    {
        Calendar mockedCalendar = mock(Calendar.class);
        when(mockedCalendar.get(Calendar.YEAR)).thenReturn(2020);

        System.out.println(mockedCalendar.get(Calendar.YEAR));
    }
}

```

How to stub method with hardcoded arguments
This example explains how to use mock objects by stubbing method behaviour.

Here, a mock object of java.util.List class is hard-wired to return a string object "target" whenever the get method is invoked on it with argument 0.

```java
package com.techfundaes.mockitoBag;

import static org.mockito.Mockito.*;

import java.util.List;

public class CustomiizeMethodBehaviour
```

```
{
    public static void main(String[] args)
    {
        List myMockedList = mock(List.class);
        when(myMockedList .get(0)).thenReturn("target");

        System.out.println(myMockedList .get(0));
    }
}
```

5.How to stub method to accept any argument

The previous example mocked a class with hardcoded argument. However mockito is quite powerful. You can use methods like anyInt() of class org.mockito.Mockito to set the mock objects behaviour when it is called with any integer as argument.

Also, notice the how the behaviour of method isEmpty() is mocked below.

```
package com.techfundaes.mockitoBag;

import static org.mockito.Mockito.*;

import java.util.ArrayList;
import java.util.List;

public class CustomizeMethodBehaviourAdvanced
{
    public static void main(String[] args)
11.    {
12.        List myMockedList = mock(ArrayList.class);
13.        when(myMockedList.get(anyInt())).thenReturn(5);
14.        when(myMockedList.isEmpty()).thenReturn(false);
15.
16.        System.out.println(myMockedList.get(1));
17.        System.out.println(myMockedList.isEmpty());
18.    }
19.}
20.
```

6.How to stub method for sequential calls

The example below shows the mock objects behaviour when its method is stubbed multiple times.

In line no. 11, the mock object myMockedList is asked to return String "target" when get(0) is called on it. Then it is asked to return String "others" on next call.

The resulting behaviour is that on first call of get(0), the string target is returned and from then onwards on all subsequent calls of get(0), the string others is returned.

Aslo, notice an alternate way of doing this as shown in line no. 12.

```
1.  package com.techfundaes.mockitoBag;
2.
3.  import static org.mockito.Mockito.*;
4.  import java.util.List;
5.
6.  public class SequentialMethodCalls
```

```
7.  {
8.      public static void main(String[] args)
9.      {
10.         List myMockedList = mock(List.class);
11.         when(myMockedList.get(0)).thenReturn("target").thenReturn("others");
12.         //when(myMockedList.get(0)).thenReturn("target", "others", "more");
13.
14.         System.out.println(myMockedList.get(0));
15.         System.out.println(myMockedList.get(0));
16.         System.out.println(myMockedList.get(0));
17.     }
18. }
19.
```

7.How to stub method to throw exception

A mocked object can also be asked to throw an exception when particular methods are called on it.

In the example below, the mock object is stubbed to throw NullPointerException when the method get(..) is called on it in line no. 14. The method used for this is thenThrow(..) of class org.mockito.Mockito.

If we need to throws exception when a method whose return type is void is called (eg. List.clear()), then we can use the alternate way of throwing exception , i.e. doThrow(..) of class org.mockito.Mockito as shown in line no. 15.

```
1.  package com.techfundaes.mockitoBag;
2.
3.  import static org.mockito.Mockito.*;
```

```

import java.util.ArrayList;
import java.util.List;

public class ThrowException
{
    public static void main(String[] args)
    {
        List myMockedList = mock(ArrayList.class);

        when(myMockedList.get(anyInt())).thenThrow(new NullPointerException());
        doThrow(new RuntimeException()).when(myMockedList).clear();

        System.out.println(myMockedList.get(1));
        myMockedList.clear();
    }
}
```

8.Verify stubbed method - Basic

Mockito provides api to verify whether the stubbed method was called by the application or not.

Addiionally, it can verify the number of times a method was calledas shown in the next page.

```
package com.techfundaes.mockitoBag;

```

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

import java.util.List;

public class VerifyMethodCalls
{
    public static void main(String[] args)
    {
        List myMockedList = mock(List.class);

        myMockedList.get(0);
        myMockedList.clear();

        verify(myMockedList).get(0);
        verify(myMockedList).clear();
    }
}
```

9.Verify stubbed method – Frequency

Mockito provides api to verify whetehr the stubbed methods were called a given number of time, at least n times , at most n times etc using methods times(...), never(), atLeast(), atMost() of class org.mockito.Mockito as shown below.

```
package com.techfundaes.mockitoBag;

import static org.mockito.Mockito.*;
import java.util.List;

public class VerfifyMethodCallCount
{
    public static void main(String[] args)
    {
        List myMockedList = mock(List.class);
        myMockedList.clear();
        myMockedList.get(0);
        myMockedList.get(1);
        myMockedList.add("a");
        myMockedList.add("b");
        myMockedList.add("c");
        verify(myMockedList).clear();
        verify(myMockedList, times(1)).clear();
        verify(myMockedList, times(2)).get(anyInt());
        verify(myMockedList, times(3)).add(anyObject());
        verify(myMockedList, never()).remove(anyObject());
        verify(myMockedList, atLeast(2)).add(anyObject());
        verify(myMockedList, atMost(1)).clear();
    }
}
```