Difference between Spy and Mock in Mockito

Overview

In Unit Test cases we can mock the object to be tested. This mocking is usually done using **mock**. But in scenarios mocking of object using **spy** is more beneficial. We generally use **mock when we have to completely mock the object behavior while using spy we will be spying or stubbing specific methods of it**. So mock achieves complete mocking while spy achieves partial mocking(Partial mocking can also be achieved using mock thenCallRealMethod(), when to use spy and thenCallRealMethod is covered in [next tutorial](https://www.javainuse.com/java/mockthenCallRealMethod).) . For Example, if there is a object with 3 methods such that first two methods are implemented and third method calls an external third party API and you have a test where you want to call all the three methods. Then if using Mock we will have to mock all the three methods. If we use Spy we will not mock all three but only the third method which is a call to an external API.

Lets Begin

We will implement the above mentioned scenario of an object having three method calls out of which one method calls an external third party API which we always have to mock. We will first implement this scenario using Mock and then later using Spy. We will now create the EmployeePaymentService class. It has three methods-

* getNoOfWorkingDays- Method call to get number of working days of an employee. All logic for this is implemented in this method itself.
* getSalaryPerDay-Method call to get number of daily salary of an employee. All logic for this is implemented in this method itself.
* processPay- Method to calculate the total monthly salary of an employee using his working days and daily salary. This logic is not implemented in the method but a third party API is called which returns the calculated salary.

package com.javainuse.service;

public class EmployeePaymentService {

/\*\*

\* @param empId employee id of Employee.

\* @return number of days the employee has worked.

\*/

public int getNoOfWorkingDays(String empId) {

// default value

int noOfWorkingDays = 0;

**// code for getting number of working Days. All logic for this is implemented**

**//in this method itself**

noOfWorkingDays = 25;

return noOfWorkingDays;

}

/\*\*

\* @param empId employee id of Employee.

\* @return salary per day of the employee.

\*/

public int getSalaryPerDay(String empId) {

// default value

int salaryPerDay = 0;

**// code for getting salary per day. All logic for this is implemented**

**//in this method itself.**

salaryPerDay = 1000;

return salaryPerDay;

}

/\*\*

\* This method is not calculated here but a third party method is called

\*

\* @param empId

\* @param empWrkingDays

\* @param empSalaryPerDay

\* @return

\*/

public int processPay(String empId, int empWrkingDays, int empSalaryPerDay) {

**// code for processing is not done internally here, but a third party**

**// external API call is made.**

**//third party API call**

// return thirdparty.getSalaryThirdPartyCall(empId,empWrkingDays,empSalaryPerDay);

// returning some default value

return 20000;

}

}

**Mockito Mock Example**   
We will **completely mock** the EmployeePaymentService class using **Mock** as follows-

package com.javainuse.service;

import org.testng.Assert;

import org.testng.annotations.BeforeMethod;

import org.testng.annotations.Test;

import org.mockito.Mock;

import org.mockito.MockitoAnnotations;

import static org.mockito.Matchers.anyInt;

import static org.mockito.Matchers.anyString;

import static org.mockito.Mockito.when;

public class EmployeePaymentServiceTest\_mock {

private int testWrkingDays=25;

private int testSalaryPerDay=1000;

private int testSalary=25000;

private String empId="emp100";

**//Mock class instance using mock**

**@Mock**

private EmployeePaymentService employeePaymentService;

/\*\*

\* Setup before test.

\*/

@BeforeMethod

public void beforeMethod() {

MockitoAnnotations.initMocks(this);

}

@Test

public void testGetSalary()

{

**//Tell mockito to mock all the three methods**

**when(employeePaymentService.getNoOfWorkingDays(anyString())).thenReturn(testWrkingDays);**

**when(employeePaymentService.getSalaryPerDay(anyString())).thenReturn(testSalaryPerDay);**

**when(employeePaymentService.processPay(anyString(), anyInt(), anyInt())).thenReturn(testSalary);**

**//Mock call**

int returnedWrkingDays=employeePaymentService.getNoOfWorkingDays(empId);

Assert.assertEquals(returnedWrkingDays, testWrkingDays);

**//Mock call**

int returnedSalaryPerDay=employeePaymentService.getSalaryPerDay(empId);

Assert.assertEquals(returnedSalaryPerDay, testSalaryPerDay);

**//Mock call**

int returnedSalary=employeePaymentService.processPay(empId,testWrkingDays,testSalaryPerDay);

Assert.assertEquals(returnedSalary, testSalary);

}

}

**Mockito Spy Example**   
We will **partially mock** the EmployeePaymentService class using Spy as follows-

package com.javainuse.service;

import org.testng.Assert;

import org.testng.annotations.BeforeMethod;

import org.testng.annotations.Test;

import org.mockito.MockitoAnnotations;

import org.mockito.Spy;

import static org.mockito.Matchers.anyInt;

import static org.mockito.Matchers.anyString;

import static org.mockito.Mockito.when;

public class EmployeePaymentServiceTest\_spy {

private int testWrkingDays = 25;

private int testSalaryPerDay = 1000;

private int testSalary = 25000;

private String empId = "emp100";

**//Mock class instance using Spy**

**@Spy**

private EmployeePaymentService employeePaymentService;

/\*\*

\* Setup before test.

\*/

@BeforeMethod

public void beforeMethod() {

employeePaymentService = new EmployeePaymentService();

MockitoAnnotations.initMocks(this);

}

@Test

public void testGetSalary() {

**//Tell mockito to mock only the processPay Method**

**when(employeePaymentService.processPay(anyString(), anyInt(), anyInt())).thenReturn(testSalary);**

**//Actual call made**

int returnedWrkingDays = employeePaymentService.getNoOfWorkingDays(empId);

Assert.assertEquals(returnedWrkingDays, testWrkingDays);

**//Actual call made**

int returnedSalaryPerDay = employeePaymentService.getSalaryPerDay(empId);

Assert.assertEquals(returnedSalaryPerDay, testSalaryPerDay);

**//Mock call**

int returnedSalary = employeePaymentService.processPay(empId, testWrkingDays, testSalaryPerDay);

Assert.assertEquals(returnedSalary, testSalary);

}

}

Download Source Code

Download it - [Java Mockito using Mock and Spy](https://www.javainuse.com/zip/java/test-mock-spy.rar)

What Next?

The next chapter explains the [difference between Spy and thenCallRealMethod](https://www.javainuse.com/java/mockthenCallRealMethod).

Difference between Spy and Mock thenCallRealMethod

Overview

In [previous tutorial](https://www.javainuse.com/java/mockSpy) we saw difference between mock and spy with example. But **partial mocking** for which spy is used can also be done using **mock thenCallRealMethod**. So when should we use spy and when the mock thenCallRealMethod.

Lets Begin

We will mock the EmployeePaymentService class defined in the [previous tutorial](https://www.javainuse.com/java/mockSpy).

package com.javainuse.service;

import org.testng.Assert;

import org.testng.annotations.BeforeMethod;

import org.testng.annotations.Test;

import org.mockito.Mock;

import org.mockito.MockitoAnnotations;

import static org.mockito.Matchers.anyInt;

import static org.mockito.Matchers.anyString;

import static org.mockito.Mockito.when;

public class EmployeePaymentServiceTest\_MockCallRealMethod {

private int testWrkingDays = 25;

private int testSalaryPerDay = 1000;

private int testSalary = 25000;

private String empId = "emp100";

**//Mock class instance using mock**

@Mock

private EmployeePaymentService employeePaymentService;

/\*\*

\* Setup before test.

\*/

@BeforeMethod

public void beforeMethod() {

MockitoAnnotations.initMocks(this);

}

@Test

public void testGetSalary() {

**//tell mockito to call actual methods instead of mocking**

when(employeePaymentService.getNoOfWorkingDays(anyString())).**thenCallRealMethod()**;

when(employeePaymentService.getSalaryPerDay(anyString())).**thenCallRealMethod()**;

**//tell mockito to mock the following call**

when(employeePaymentService.processPay(anyString(), anyInt(), anyInt())).thenReturn(testSalary);

**//Actual call as defined above**

int returnedWrkingDays = employeePaymentService.getNoOfWorkingDays(empId);

Assert.assertEquals(returnedWrkingDays, testWrkingDays);

**//Actual call as defined above**

int returnedSalaryPerDay = employeePaymentService.getSalaryPerDay(empId);

Assert.assertEquals(returnedSalaryPerDay, testSalaryPerDay);

**//Mock call as defined above**

int returnedSalary = employeePaymentService.processPay(empId, testWrkingDays, testSalaryPerDay);

Assert.assertEquals(returnedSalary, testSalary);

}

}

As we can see we called the actual implementation for getNoOfWorkingDays() and getSalaryPerDay() and mocked the processPay() method.**So similar partial mocking achieved using spy is also achieved using Mock thenCallRealMethod()**. Both can be used interchangeably. However there is major difference between the use of thenCallRealMethod and spy. When we use Mock the actual object instance is not created but bare-bones shell instance of the Class is created to track interactions. Whereas in case of spy we ourselves create the object instance to be used by spy. So using Mockito Spy guarantees that the real methods are called correctly.  
For example-

public class EmployeePaymentService {

EmployeePaymentService(

final NamedParameterJdbcTemplate namedParameterJdbcTemplate)

{

this.namedParameterJdbcTemplate = namedParameterJdbcTemplate;

}

public int getNoOfWorkingDays(String empId)

{

**//call some query**

namedParameterJdbcTemplate.query();

}

}

If for a unit test case we mock the above EmployeePaymentService using Mock as-  
**@Mock private EmployeePaymentService employeePaymentService;**  
and later call the actual method getNoOfWorkingDays using thenCallRealMethod as follows-  
**when(employeePaymentService.getNoOfWorkingDays(anyString())).thenCallRealMethod();  
int returnedWrkingDays = employeePaymentService.getNoOfWorkingDays(empId);**We get a **java.lang.NullPointerException** in the getNoOfWorkingDays method because the namedParameterJdbcTemplate has never been initialized. However if we were using the Mockito Spy this scenario would never arise because we would have ourselves created the EmployeePaymentService using constructor with namedJdbcTemplate.

Download Source Code

Download it - [Java Mockito using thenCallRealMethod](https://www.javainuse.com/zip/java/test-mock-spy.rar)