Mockito Framework

By

Anand Kulkarni

anand.kulkarni1@zensar.com

Table of Content

Introduction

- Mocking is a process used in unit testing when the unit being tested has external dependencies.
- The purpose of mocking is to isolate and focus on the code being tested and not on the behavior or state of external dependencies.

Need of Mocking

There are several scenarios where we should use mocks:

- When we want to test a component that depends on other component, but which is not yet developed.
- When the real component performs slow operations, usual with dealing with database connections or other intense disk read/write operations.
- When there are infrastructure concerns that would make impossible the testing. This is similar in same way to the first scenario described when, for example, our development connects to a database, but the server where is hosted is not configured or accessible for some reason.

Mockito framework

- *Mockito is a Java-based open source mocking framework used for unit testing of Java application.*
- *The main purpose of using the Mockito framework is to simplify the development of a test by mocking external dependencies and use them in the test code.*
- *Mockito is a Mocking Framework that makes it very easy to create mocks for the classes and interfaces with which your class under test interacts.*

Mockito installation

Create a Maven project in STS & add below dependencies inside pom.xml –

```
<dependency>
   <groupId>junit</groupId>
   <artifactId>junit</artifactId>
   <version>4.13.2</version>
   <scope>test</scope>
</dependency>
<dependency>
   <groupId>org.mockito</groupId>
   <artifactId>mockito-all</artifactId>
   <version>1.9.5</version>
   <scope>test</scope>
</dependency>
```

Mockito APIs

1. Mockito is an open source framework & its APIs you can find at https://site.mockito.org/javadoc/current/index.html?org/mockito/Mockito.html
2. API defines root package called ‘org.mockito’ along with several sub packages.
3. The most important class is org.mockito.Mockito because all major functions have been defined inside the same class.

Simple Mockito demo

```
public interface CityService {
        List<String> findCitiesByCountries(String country);
}
public class CityServiceImpl implements CityService {
        public List<String> findCitiesByCountries(String country) {
                List<String> cities = new ArrayList<String>();
                cities = executeSQLQuery("SELECT * FROM CITIES WHERE country='"
+ country + "'");
                return cities;
        }
}
```

Simple Mockito demo continue..

```
public class BusinessApp {
        CityService cityService; //dependency

        public List<String> retrieveCityListByCountry(String country) {
                List<String> cities =  this.cityService.findCitiesByCountries(country);
                return cities;
        }
}
```

Simple Mockito demo continue..

```
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.when;
public class BusinessAppMockitoTest {
        @Test
        public void testRetrieveCities() {
                CityService cityService = mock(CityService.class); //Mocking
        when(cityService.findCitiesByCountries("INDIA")).thenReturn(Arrays.asList("Delhi
", "Mumbai", "Kolkatta", "Chennai"));
                BusinessApp businessApp = new BusinessApp(cityService);
                assertEquals(4, businessApp.retrieveCityListByCountry("INDIA").size());
        }
}
```

Mockito methods

Mockito class defines several static methods those are useful in unit testing. Let us discuss few important methods.

- *mock()*

The mock() is a basic method that takes single parameter of the class to be mocked. For example,

```
MyList listMock = mock(MyList.class);
```

Mockito methods continue…

- *when()*

The when() method is used to stub a method so that it will return fixed value irrespective of its parameters. For example,

```
when(listMock.get(2)).thenReturn("ABC");
```

Mockito methods continue…

- *verify()*

Mockito can ensure whether a mock method is being called with required arguments or not. It is done using the verify() method. For example,

```
//test the add functionality

Assert.assertEquals(calcService.add(10, 20),30);

//verify call to calcService is made or not with same arguments.

verify(calcService).add(10, 20);
```

Mockito methods continue…

- *spy()*

When spy() is called, the actual method of real object is called where as in mock() method, you always call mocked method.

//create a spy on actual object

calcService = spy(calculator);

//perform operation on real object & test the add functionality

Assert.assertEquals(calcService.add(20, 10),30);

Mockito methods continue…

- *reset()*

The Mockito reset() method is used to reset the mocks. It is mainly used for working with the container injected mocks. Usually, the reset() method results in a lengthy code and poor tests. It's better to create new mocks rather than using reset() method. That is why the reset() method is rarely used in testing.

```
when(calcService.add(20.0,10.0)).thenReturn(30.0);

//reset mock

reset(calcService);

Assert.assertEquals(calcService.add(20, 10),30); //FALSE
```

Mockito methods continue…

- *doThrow()*

Mockito doThrow() method will throw specified RuntimeException when meets a condition. In the below example, when we call clear() method on mockedList, it will throw RuntimeException.

```
doThrow(new RuntimeException()).when(mockedList).clear();
```

Mockito methods continue…

- *doCallRealMethod()*

We use doCallRealMethod() when we want to call the real implementation of a method. For example,

```
Foo mock = mock(Foo.class);

doCallRealMethod().when(mock).someVoidMethod();

// this will call the real implementation of Foo.someVoidMethod()

mock.someVoidMethod();
```

Mockito methods continue…

- *doNothing()*

As the name indicates, doNothing() method will not perform any action if the given condition is met. In the below example, if we call clear() method on mockedList, it won't do anything.

```
doNothing().when(mockedList).clear();
```

Mockito methods continue…

- *doReturn()*

The doReturn() method is called before when() i.e.

doReturn(true).when(cityService.isValidCity("xxx"));

We have two options, doReturn-when or when-thenReturn. Both methods are almost same except thenReturn() has type safety. Hence, developers prefer when-thenReturn instead of doReturn-when.

Mockito methods continue…

- *never()*

The never() method is used to confirm that specific method is never called on the object. Suppose, in an application, we load user's profile only if login is successful then we can use never() to confirm loadUserProfile() method is never called in case login fails.

```
loginService.login("james", "wrong_password");

Mockito.verify(loginService, never()).loadUserProfile(any());
```

Mockito methods continue…

- *timeout()*

Mockito provides a special Timeout option to test if a method is called within stipulated time frame.

//passes if add() is called within 100 ms.

verify(calcService,timeout(100)).add(20,10);

Thank you!!