Mockito/PowerMock

...

Presented by: Bello Muhammad

Contents

Mockito?

Mockito is an open source testing framework for Java that allows the creation of **test double**

objects in automated unit tests for the purpose of **test-driven** development or **behavior-driven** development.

Test Double?

Test Double is a generic term used by **Gerard Meszaros** to mean any case where you replace a production object for testing purposes.

- Dummy objects
- Fake objects
- Stubs objects
- Spies objects
- Mocks objects

Why Mock?

Most of the time the code we write have dependencies. Often, delegates some work to other methods in other classes.

Dependency is Bad

Independence is Light

Mockito Installation:

There are different ways of adding Mockito into your project, below are some of the common ways.

1. Download the jar file and at it to our project path.
2. Add the dependency to the build automation tool that you are using in your project. E.g.

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

// Gradle Dependency
testCompile “org.mockito:mockito-core:1.9.5“

// Ivy Dependency
<dependency org=”org.mockito” name=”mockito-core” rev=”1.9.5” conf=”test->default”/>
```

Using Mockito

After adding the dependencies in our project what next?

How to Inject Mocks?

Assuming we want to test a class in our application, we first create and inject the dependencies using either annotations or Mockito's static method **mock().**

```
private EtcBaseDao etcBaseDao;
private EtcBaseService etcBaseService;

@Before
public void setUp() {
   etcBaseDao = Mockito.mock(EtcBaseDao.class);
   etcBaseService = new EtcBaseService(etcBaseDao);
}
```

Using @Mock and @InjectMocks

There are two commonly used annotations:

1. @Mock
2. @InjectMocks

```
@Mock
private EtcBaseDao etcBaseDao;

@InjectMocks
private EtcBaseService etcBaseService;

@Before
public void setUp() throws Exception {
   MockitoAnnotations.initMocks(this);
}
```

Tired of Using initMocks(this)?

Using runners from Mockito or from Spring Framework will give you automatic validation and initMocks.

- MockitoJUnitRunner
- MockitoJUnit4Runner

```
@Runwith(MockitoJUnit4Runner.class)
public class MockitoJUnit4RunnerTest {
        @Mock
        private EtcBaseDao etcBaseDao;

        @InjectMocks
        private EtcBaseService etcBaseService;
}
```

Stubbing Method's Return Value

The ability to return a test double as value when a method is called is called **Stubbing**. Using Mockito's **when()** with **thenReturn()** you can specify how and what a method should return.

The pattern:

- Arrange
- Act
- assert

Mocking Methods with Mockito

After creating and injecting your mock, you should then tell Mockito how to behave when certain method are invoked.

```
Mockito.when(instanceName.methodName(methodArguments)).thenReturn(true);
```

```
Mockito.when(etcBaseService.save(etcBase)).thenReturn(etcBase);
```

We can also use matchers as shown below.

```
Mockito.when(etcBaseService.save(Mockito.any(EtcBase.class))).thenReturn(etcBase);
```

We can't mix matcher (this will fail)

```
Mockito.when(etcBaseService.getByIdAndDate(branchId, Mockito.any(Date.class)))
        .thenReturn(etcBase);
```

thenReturn doReturn Answer
doAnswer thenAnswer

WHEN TO USE WHAT?

assertThat assertEquals

You should use thenReturn() or doReturn()

when you already know the return value at the time you mock the method call.

You should use Answer() or doAnswer()

when you need to do additional things when a mocked method is invoked.

You should use assertThat() or assertEquals()

There is no much difference between must of Hamcrest matchers and junit asserts, only that hamcrest tends to given more information when there is an error with the test method.

Throwing Exception from a Method

We can use the JUnit expected to make sure that a method throw exception when it's called.

@Test(expected = ExceptionName.class)

```
@Test(expected = ExceptionClassName.class)
public void theNameOfOurTestMethod() {

        // Do all your stuff here.
}
```

com.etcbase.mockito.service.MockitoExpectedTest.java

Mocking Void Methods with Mockito

With Mockito we can use doAnswer() to mock a void method, doThrow() to throw an exception from a void method. The following example illustrate that.

```
Mockito.doAnswer(new Answer<Void>() {
 @Override
public Void answer(InvocationOnMock invocationOnMock)
                                    throws Throwable {
                return null;
  }}).when(etcBaseDao).delete(Mockito.any(Long.class));
```

com.etcbase.mockito.service.MockitoVoidTest.java

Using Verify with Mockito

Apart from asserting that the return values are valid, we can also verify that a given method is called on a given mock object during test execution, most especially when the method under test is a Void method.

There are two types of verify methods:

- One that takes the mock object only and
  verify(theService).theMethod(...);

- One that takes mock object and verification mode
  verify(theService, times(1)).theMethod(...);

com.etcbase.mockito.service.MockitoWithVerifyTest.java

Using ArgumentCaptor

The argumentcaptor allows you to capture any argument that is passed into a mock method. Mockito.ArgumentCaptor.

Usage:

```
@Captor
Private ArgumentCaptor<EtcBase> etcBaseArgumentCaptor;
```

com.etcbase.mockito.service.MockitoArgumentCaptorTest.java

Mockito Spy

Sometimes we do want to interact with the real service and verify that it was invoked, that is where Mockito spy is at your back.

Usage:

```
@Spy  // instead of using @Mock we use @Spy
private EtcBaseDao etcBaseDao;
```

Using MockitoJUnitRunner

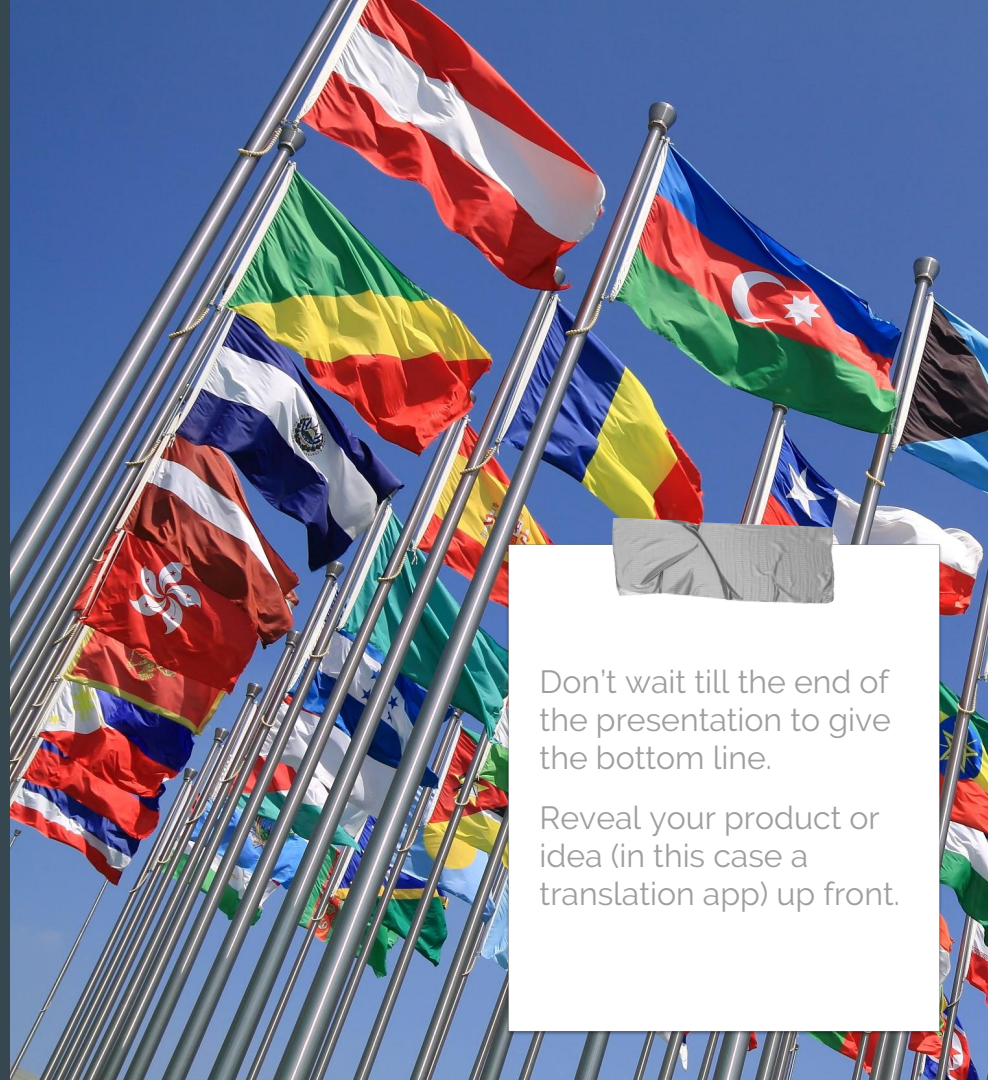
Are you tired of getting meaning less error? Or verification errors? Again..., initMocks. With Mockito MockitoJUnitRunner give you get automatic validation and automatic initMocks().

```
@Test
public void successAutomaticValidation() {
        when(myMock.method1());

        // Do something here

         // Error is reported here
        verify(myMock).method2();
}
```

The Google Translate app can repeat anything you say in up to

NINETY LANGUAGES

from German and Japanese to Czech and Zulu



Don't wait till the end of the presentation to give the bottom line.

Reveal your product or idea (in this case a translation app) up front.

Questions?