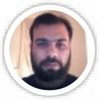
ABOUT MOHAMMAD MERAJ ZIA



I did my Engineering in Information Technology from IET, Lucknow, India. Currently doing MSc in Information Te
University. I have worked in Java/J2EE domain for the last 10 years. Have good understanding of Payment and Fi

# Mockito Mock Private Method Example with PowerMock

☐ Posted by: Mohammad Meraj Zia ☐ in Mockito ☐ April 11th, 2016

A unit test should test a class in isolation. Side effects from other
should be eliminated if possible. Mockito lets you write beautiful t
simple API. In this example we will learn how to mock a private m
technologies used in this example are Java 1.8, Eclipse Luna 4.4.2

## 1. Introduction

Mockito is a popular mocking framework which can be used in co
Mockito allows us to create and configure mock objects. Using Mo
development of tests for classes with external dependencies signi
create the mock objects manually or can use the mocking framew
EasyMock. jMock etc. Mock frameworks allow us to create mock o
and define their behavior. The classical example for a mock object
In production a real database is used, but for testing a mock object simulates the database and ensures that the test con
the same.

Mockito does not allow us to mock private methods but there are other tools which we can use to achieve this. According

Firstly, we are not dogmatic about mocking private methods. We just don't care about private methods because from the private methods don't exist. Here are a couple of reasons Mockito doesn't mock private methods:

1. It requires hacking of classloaders that is never bullet proof and it changes the API (you must use custom test runner, etc.).
2. It is very easy to work around – just change the visibility of method from private to package-protected (or protected).
3. It requires the team to spend time implementing & maintaining it. And it does not make sense given point (2) and a fa implemented in different tool (powermock).
4. Finally… Mocking private methods is a hint that there is something wrong with Object Oriented understanding. In OO roles) to collaborate, not methods. Forget about pascal & procedural code. Think in objects.

## 2. Creating a project

Below are the steps we need to take to create the project.

1. Open Eclipse. Go to File=>New=>Java Project. In the 'Project name' enter 'MockPrivateMethodExample'.
2. Eclipse will create a 'src' folder. Right click on the 'src' folder and choose New=>Package. In the 'Name' text-box enter 'com.javacodegeeks'. Click 'Finish'.
3. Right click on the package and choose New=>Class. Give the class name as MockPrivateMethodExample. Click 'Finish' default class with the given name.

### 2.1 Dependencies

For this example we need the below mentioned jars:

- junit-4.1.2
- mockito-all-1.10.19
- powermock-mockito-release-full-1.6.4-full
- javassist-3.12.1.GA

These jars can be downloaded from Maven repository. These are the latest (non-beta) versions available as per now. To a classpath right click on the project and choose Build Path=>Configure Build Path. The click on the 'Add External JARs' but hand side. Then go to the location where you have downloaded these jars. Then click ok.

Figure 1. Adding Dependencies.

## 3. Code

We will create a very simple Java class with two methods. The first method is ‘public’ which calls a private method.

*MockPrivateMethodExample.java*

```java
package com.javacodegeeks;

import java.util.Date;

/**
* Example class to test the mocking of private method.
* @author Meraj
*/
public class MockPrivateMethodExample {

  public String getDetails() {
    return "Mock private method example: " + iAmPrivate();
  }

  private String iAmPrivate() {
    return new Date().toString();
  }
}
```

Then we create a new class which will test this above class.

This class should be annotated with

```
@RunWith(PowerMockRunner.class)
```

annotation. When a class is annotated with

```
@RunWith
```

or extends a class annotated with

```
@RunWith
```

, JUnit will invoke the class it references to run the tests in that class instead of the runner built into JUnit.

We need another class level annotation for this example:

```
@PrepareForTest
```

. This annotation tells PowerMock to prepare certain classes for testing. Classes needed to be defined using this annotatic
that needs to be byte-code manipulated. This includes final classes, classes with final, private, static or native methods th
and also classes that should be return a mock object upon instantiation.

This annotation can be placed at both test classes and individual test methods. If placed on a class all test methods in thi
be handled by PowerMock (to allow for testability). To override this behavior for a single method just place a @PrepareFo
the specific test method. This is useful in situations where for example you'd like to modify class X in test method A but ir
want X to be left intact. In situations like this you place a @PrepareForTest on method B and exclude class X from the list

Sometimes you need to prepare inner classes for testing, this can be done by suppling the fully-qualified name of the inne
be mocked to the list. You can also prepare whole packages for test by using wildcards. The annotation should always be
@RunWith(PowerMockRunner.class) if using junit 4.x. The difference between this annotation and the @PrepareOnlyThisF
that this annotation modifies the specified classes and all its super classes whereas the @PrepareOnlyThisForTest annotat
the specified classes.

In the test class we will call the spy() method of org.powermock.api.mockito.PowerMockito by passing the reference to th
be tested:

```java
MockPrivateMethodExample spy = PowerMockito.spy(mockPrivateMethodExample);
```

Then we define what we want to do when this particular private method is called.

```java
PowerMockito.doReturn("Test").when(spy, {$methodName});
```

Here we are saying that return 'Test' when method ${methodName} is called.

Below is the full code for the test class:

*MockPrivateMethodTest.java*

```java
package com.javacodegeeks;

import org.junit.Assert;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.Mockito;
import org.powermock.api.mockito.PowerMockito;
import org.powermock.core.classloader.annotations.PrepareForTest;
import org.powermock.modules.junit4.PowerMockRunner;

@RunWith(PowerMockRunner.class)
@PrepareForTest(MockPrivateMethodExample.class)
public class MockPrivateMethodTest {

  private MockPrivateMethodExample mockPrivateMethodExample;

  // This is the name of the private method which we want to mock
  private static final String METHOD = "iAmPrivate";

  @Test
  public void testPrivateMethod() throws Exception {
    mockPrivateMethodExample = new MockPrivateMethodExample();

    MockPrivateMethodExample spy = PowerMockito.spy(mockPrivateMethodExample);
    PowerMockito.doReturn("Test").when(spy, METHOD);
    String value = spy.getDetails();

    Assert.assertEquals(value, "Mock private method example: Test");
    PowerMockito.verifyPrivate(spy, Mockito.times(1)).invoke(METHOD);
  }
}
```

## 4. Download the source file

This was an example of mocking a private method using PowerMock.

**Download**
You can download the full source code of this example here: **MockPrivateMethodExample**