**Hands On Lab 3**

**Mocking Final, Static and Private methods with PowerMock**

[1. Overview 1](#_Toc14703)

[2. Create a simple java maven project. 2](#_Toc19790)

[3. Adding Dependencies with Maven 2](#_Toc16127)

[4. Enabling PowerMock Annotations 3](#_Toc5446)

[5. Mocking final methods 3](#_Toc13113)

[6. Mocking static methods 6](#_Toc19496)

[7. Mocking private methods 7](#_Toc21364)

[8. Conclusion 9](#_Toc2040)

## **1. Overview**

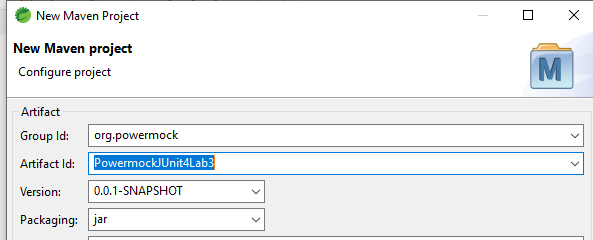
Mockito is a powerful, open-source Mocking framework in Java. The features it provides for unit-testing is inevitably unique and important, nonetheless, ease out a lot of work for developers while writing unit test cases.

While Mockito can help with virtually everything, there are some things it cannot do. Like stubbing or testing private, final or static methods. It needs much more power to write test cases for such methods which usually causes developers to write cumbersome code for these methods.

Here, PowerMockito comes to the rescue. ****PowerMockito is capable of testing private, final or static methods as it makes use of Java Reflection API****. Let’s see the demonstration in action while we study its uses.

Follow these  steps to create PowerMock project.

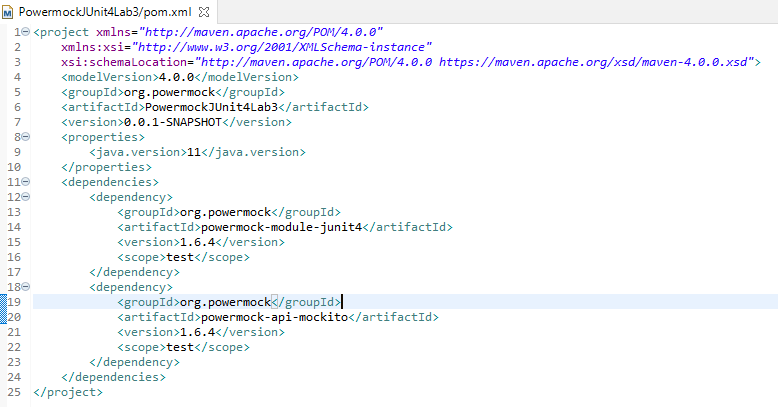
## **Create a simple java maven project.**



## **3. Adding Dependencies with Maven**

Add required dependencies to pom.xml

Your pom.xml will look like below:



## 

## **4. Enabling PowerMock Annotations**

Just like what we need to do with Mockito, we also need to enable the use of Annotations with PowerMockito. Much like Mockito, we make use of similar annotations, as shown here:

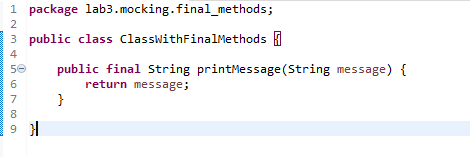
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | import org.junit.runner.RunWith;  import org.powermock.core.classloader.annotations.PrepareForTest;  import org.powermock.modules.junit4.PowerMockRunner;    @RunWith(PowerMockRunner.**class**)  @PrepareForTest(fullyQualifiedNames = "org.powermock.\*")  public **class** PowerTest { |

Let us look at each annotation we used above:

* @RunWith annotation is similar to what we did in Mockito. Instead of using Mockito, we will use a PowerMockRunner this time, which will enablePowerMockito APIs in the test.
* @PrepareForTest was provided a fullyQualifiedNames package with a wildcard. This informs PowerMockito which classes to prepare with Java Reflection API for testing.

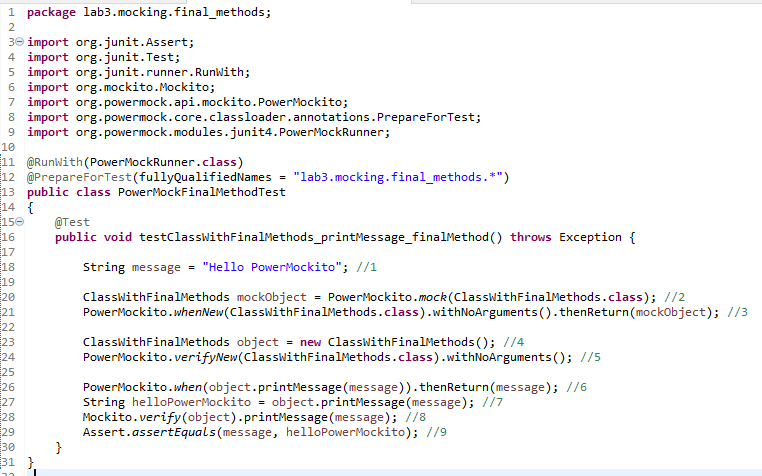
## **5. Mocking final methods**

Let’s start working with PowerMockito API by mocking final methods. To mock final methods, not much to the surprise, we should first define final methods. Here is an example of the model we will be testing. Create this class under src/main/java folder.



Simple enough, the method just returns the String which is passed to it. Now, it’s time to write our test.

Create this class under src/test/java folder.



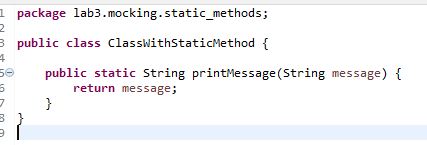
We have used class names while using methods so that no confusion arises when importing the classes. Let us see what happened in the above test:

1. We defined a generic String message which we will be using as a parameter and expectation.
2. We mock an instance of the system under test, ClassWithFinalMethods.
3. whenNew() method makes sure that whenever ****an instance of this class is made using the new keyword by invoking a no argument constructor****, this mock instance is returned instead of the real object.
4. We invoke the no argument constructor to make an instance of the system under test.
5. We verify that the no argument constructor was actually involved during the last step.
6. We set an expected String when the final method is called, using the String we defined in Step 1.
7. The final method printMessage(…) is invoked.
8. We verify that the final method was actually called.
9. Finally, we assert our expectations to the actual String returned to us.

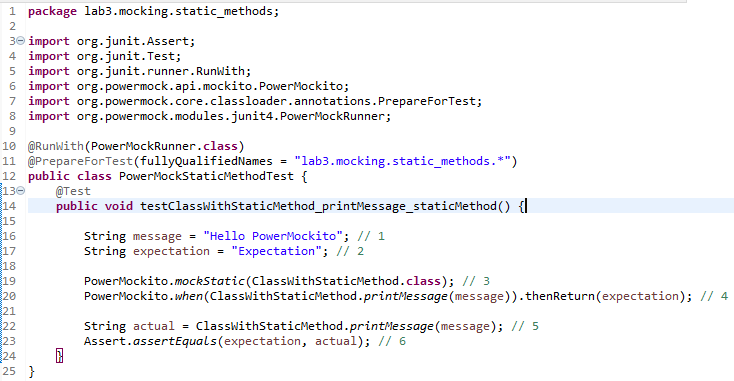
Now that was a lot actually. We used a simple example so that everything makes sense. As we have explicitly used class names with static methods, this test is ready to be run.

## **6. Mocking static methods**

We are through final methods now. It’s time to learn how static methods can be tested withPowerMockito. We make a new class named as ClassWithStaticMethod and add a new method, which is static in nature:



A similar method as earlier, the method just returns the String which is passed to it. Now, it’s time to write our test:



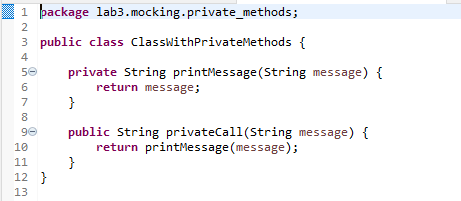
This test was smaller than earlier. Let us talk about each step we performed here:

1. We defined a generic String message which we will be using as a parameter.
2. Another generic String message, to be used as an expectation.
3. Prepare ClassWithStaticMethod for static method test.
4. Preparing expectations when the static method will be invoked.
5. Invoking the static method.
6. Verifying the expected and actual result.

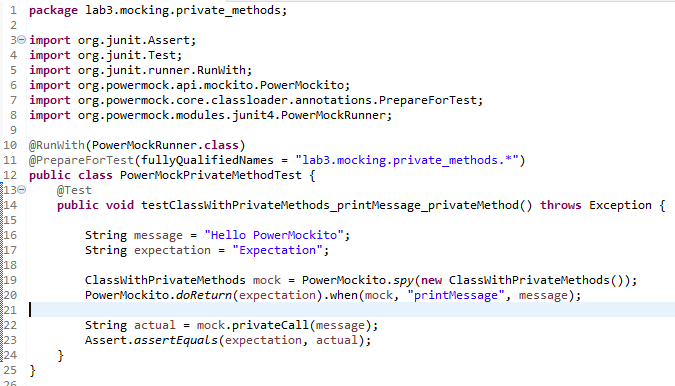
Testing static methods is quite simple actually. the key part is to call PowerMockito.mockStatic(…) so that PowerMockito API is enabled for the class.

## **7. Mocking private methods**

Down to the last bit, in this section, we will test private methods. This is another awesome usage of Reflection API in Java. First, we define our system under test:



Now, defining a public message which internally calls the private message is necessary. Let us see our test to see how it makes use of reflection API:



Here, few things to point out are:

* We start by creating a mock using Powermockito.spy(…) method.
* Next, we make use of Reflection API by providing method name as a String parameter to when(…) method.
* Finally, we invoke the public method which in turn invoked the private method and we verify our results using assertEquals(…) method.

## **8. Conclusion**

In this lab, we studied simple yet concise examples on how we can make use of PowerMockito API to stub and test private, final or static methods. This was a missing feature in Mockito, which was remedied by PowerMockito API.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*