ABOUT MOHAMMAD MERAJ ZIA

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

PowerMockito Constructor Example

Posted by: Mohammad Meraj Zia in PowerMockito May 11th, 2016 0 176 Views (*0 rating, 0 votes*)

A unit test should test a class in isolation. Side effects from other classes or the system should be eliminated if possible. Mockito lets you write beautiful tests with a clean & simple API. In this example we will learn how to mock constructor using PowerMock. PowerMockito extends Mockito functionality with several new features such as mocking static and private methods and more. Tools and 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 conjunction with JUnit. Mockito allows us to create and configure mock objects. Using Mockito simplifies the development of tests for classes with external dependencies significantly. We can create the mock objects manually or can use the mocking framewors like Mockito, EasyMock. jMock etc. Mock frameworks allow us to create mock objects at runtime and define their behavior. The classical example for a mock object is a data provider. In production a real database is used, but for testing a mock object simulates the database and ensures that the test conditions are always the same.

PowerMock provides a class called

```
PowerMockito
```

for creating mock/object/class and initiating verification, and expectations, everything else you can still use Mockito to setup and verify expectation (e.g.

```
times()
```

,

```
anyInt()
```

). All usages require

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

and

```
@PrepareForTest
```

annotated at class level.

2. Creating a project

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

- Open Eclipse. Go to File=>New=>Java Project. In the 'Project name' enter 'PowerMockConstructorExample'.

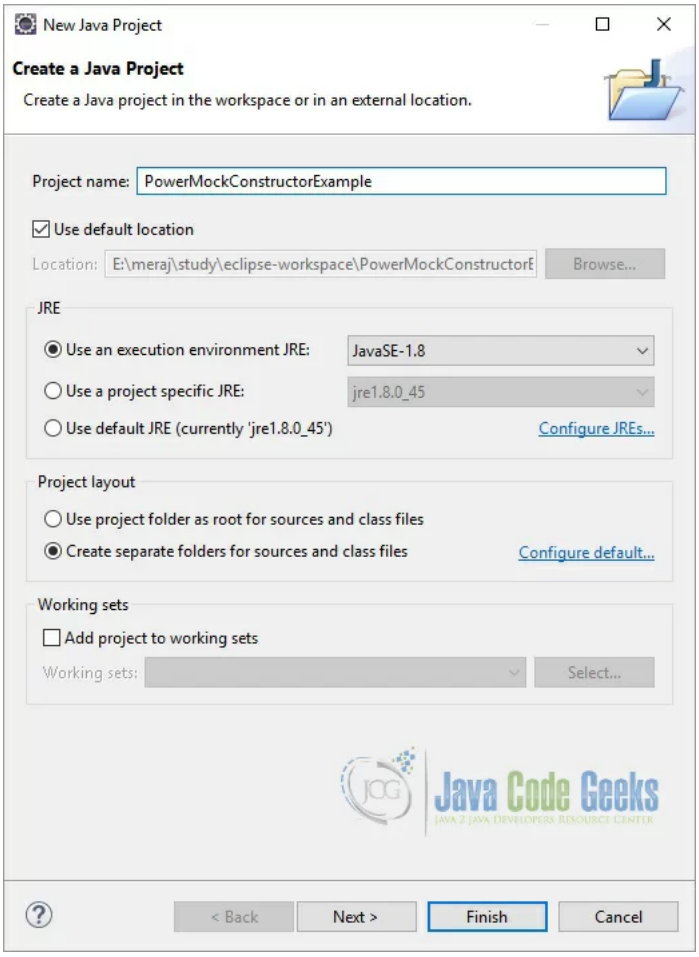

Figure 1. Create Java Project

- 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'.

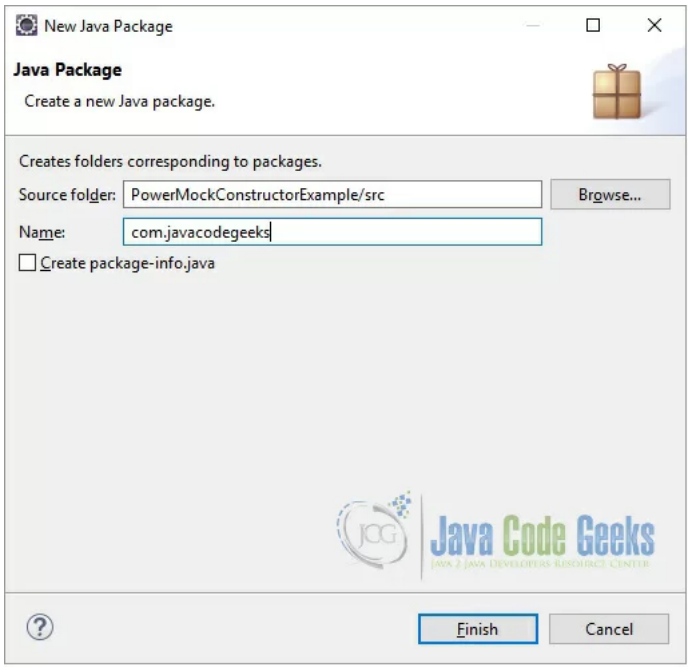

Figure 2. New Java Package

- Right click on the package and choose New=>Class. Give the class name as PowerMockConstructorExample. Click 'Finish'. Eclipse will create a default class with the given name.

Figure 3. New Java Class

2.1 Dependencies

For this example we need the below mentioned jars:

- cglib-nodep-3.2.2.jar
- easymock-3.4.jar
- hamcrest-all-1.3.jar
- javassist-3.12.1.GA.jar
- junit-4.12.jar
- objenesis-2.2.jar
- powermock-api-easymock-1.6.5.jar
- powermock-mockito-release-full-1.6.4-full.jar

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

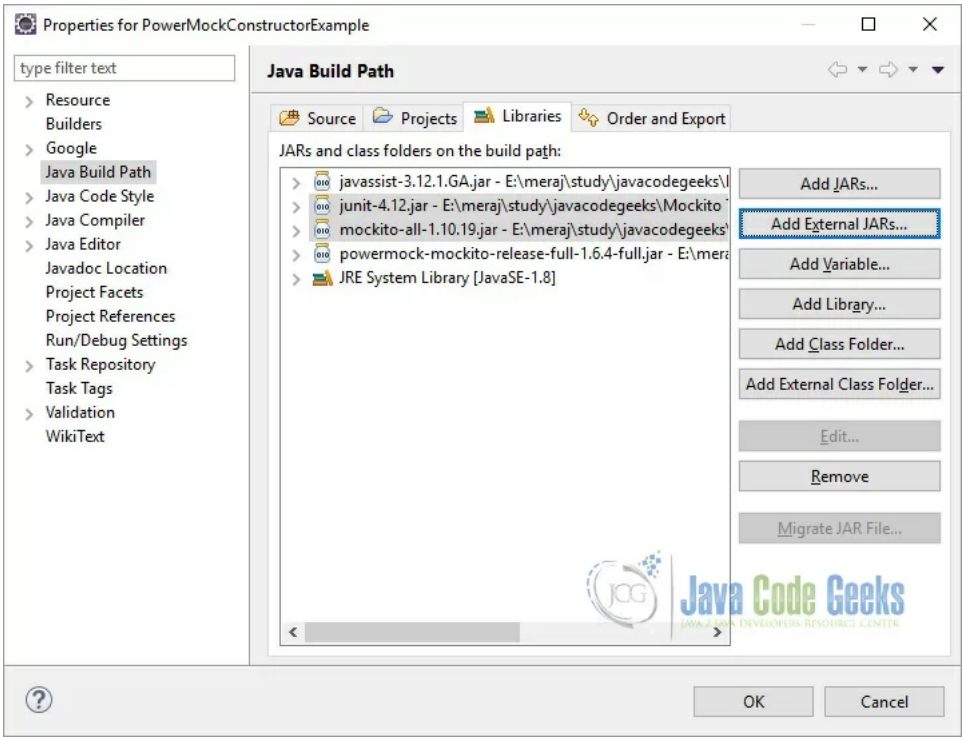

Figure 4. Dependencies

3. Code

First we will see a very simple example of how we can mock a constructor using PowerMock. First we will create a very basic class with just one method.

SimpleClass.java

```
01 package com.javacodegeeks;
02
03 import java.util.Calendar;
04
05 public class SimpleClass {
06
07   @SuppressWarnings("deprecation")
08   public String getMeCurrentDateAsString() {
09     return Calendar.getInstance().getTime().toGMTString();
10   }
11 }
```

Now we will create another class which will initialize SimpleClass and will call getMeCurrentDateAsString() method of this class.

PowerMockConstructorExample.java

```
01 package com.javacodegeeks;
02
03 public class PowerMockConstructorExample {
04
05   public String getMeSimpleObject() {
06     SimpleClass simpleClass = new SimpleClass(); // Create instance
07     String returnValue = simpleClass.getMeCurrentDateAsString();
08     return returnValue;
09   }
10 }
```

Now we will see the test class.

PowerMockConstructorExampleTest.java

```
01 package com.javacodegeeks;
02
03 import static org.easymock.EasyMock.expect;
04 import static org.powermock.api.easymock.PowerMock.expectNew;
05 import static org.powermock.api.easymock.PowerMock.replay;
06
07 import org.junit.Test;
08 import org.junit.runner.RunWith;
09 import org.powermock.api.easymock.annotation.Mock;
10 import org.powermock.core.classloader.annotations.PrepareForTest;
11 import org.powermock.modules.junit4.PowerMockRunner;
12 import static org.powermock.api.easymock.PowerMock.verify;
13 import static org.junit.Assert.assertEquals;
14
15 @RunWith(PowerMockRunner.class)
16 @PrepareForTest(PowerMockConstructorExample.class)
17 public class PowerMockConstructorExampleTest {
```

```
18
19    @Mock private SimpleClass mockSimpleClass;
20
21    private PowerMockConstructorExample instance;
22
23    @Test
24    public void testMockConstructor() throws Exception {
25      instance = new PowerMockConstructorExample();
26      expectNew(SimpleClass.class).andReturn(mockSimpleClass);
27
28      expect(mockSimpleClass.getMeCurrentDateAsString()).andReturn("Mock Result");
29
30      replay(SimpleClass.class, mockSimpleClass);
31      String value = instance.getMeSimpleObject();
32      verify(SimpleClass.class, mockSimpleClass);
33      assertEquals("Mock Result", value);
34    }
35  }
```

Few this needs to be noted for this class. This class is annotated with

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

. 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.

This class is also annotated with

```
@PrepareForTest(PowerMockConstructorExample.class)
```

. This annotation tells PowerMock to prepare certain classes for testing. Classes needed to be defined using this annotation are typically those that needs to be byte-code manipulated. This includes final classes, classes with final, private, static or native methods that should be mocked 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 this test class will be handled by PowerMock (to allow for testability). To override this behavior for a single method just place a

```
@PrepareForTest
```

annotation on the specific test method. This is useful in situations where for example you'd like to modify class X in test method A but in test method B you want X to be left intact.

In situations like this you place a

```
@PrepareForTest
```

on method B and exclude class X from the

```
value()
```

list. You can also prepare whole packages for test by using wildcards:

```
@PrepareForTest("com.mypackage.*")
```

. The annotation should always be combined with the

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

if using junit 4.x.

We use the expectNew() method of PowerMock to mock any new instance of the given class (in our case SimpleClass).

```
1  expectNew(SimpleClass.class).andReturn(mockSimpleClass);
```

Allows specifying expectations on new invocations. For example you might want to throw an exception or return a mock. Note that you must replay the class when using this method since this behavior is part of the class mock.

4. Download the source file

This was an example of mocking constructor using PowerMockito.

Download
You can download the full source code of this example here: **Power Mock Constructor Example**


(No Ratings Yet) Start the discussion 176 Views Tweet it!