ABOUT RAM MOKKAPATY

Ram holds a master's degree in Machine Design from IT B.H.U. His expertise lies in test driven development and re-factoring. He is passionate about open source technologies and actively blogs on various java and open-source technologies like spring. He works as a principal Engineer in the logistics domain.

Powermock – Mockito Integration Example

Posted by: Ram Mokkapaty in Mockito April 1st, 2015

Most of the mocking frameworks in Java, including Mockito, cannot mock static methods or final classes. If we come across a situation where we need to test these components, we won't be able to unless we re-factor the code and make them testable. For example:

1. Making private methods packaged or protected
2. Avoiding static methods

But re-factoring at the cost of good design may not always be the right solution.
In such scenarios, it makes sense to use a testing framework like Powermock which allows us to mock even the static, final and private methods.
Good thing about Powermock is that it doesn't re-invent the testing framework and in fact enhances the testing frameworks like Easymock and Mockito.

In this article, we will see an integration example of Powermock and Mockito but first let's do the setup.

Below are my setup details:

- I am using Maven – the build tool
- Eclipse as the IDE, version Luna 4.4.1.
- JUnit is my testing framework.
- Add Mockito and PowerMockito dependencies to our

```
pom.xml
```

.

1. Dependencies in pom.xml

Our dependencies consist of:

1. `junit`
2. `mockito-core`
3. `powermock-api-mockito`
4. `powermock-module-junit4`

pom.xml:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.javacodegeeks.testng.maven</groupId>
    <artifactId>testngMaven</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.11</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.mockito</groupId>
            <artifactId>mockito-core</artifactId>
            <version>2.0.5-beta</version>
        </dependency>
        <dependency>
            <groupId>org.powermock</groupId>
            <artifactId>powermock-api-mockito</artifactId>
            <version>1.6.2</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.powermock</groupId>
            <artifactId>powermock-module-junit4</artifactId>
            <version>1.6.2</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

2. System Under Test (SUT)

Our system under test is a system called

`SomeSystem`

which owns some services. A service is defined by

`Service`

interface which has couple of methods

`getName()`

and

`start()`

. If the start of the service is successful it will return 1 else 0.

One can add a

`Service`

to the

`SomeSystem`

using

`add(service)`

method. Our

`SubSystem`

has a

```
start()
```

method which will start the services it contains. On start of each service,

```
ServiceListener
```

is notified of the success or failure of the service.

SomeSystem:

```
01 package com.javacodegeeks.mockito;
02
03 import java.util.ArrayList;
04 import java.util.List;
05
06 public class SomeSystem {
07     private List services = new ArrayList();
08     private ServiceListener serviceListener;
09     private List events = new ArrayList();
10
11     public void start() {
12         for (Service service : services) {
13             boolean success = startServiceStaticWay(service) > 0;
14             notifyServiceListener(serviceListener, service, success);
15             addEvent(service, success);
16         }
17     }
18
19     private void addEvent(Service service, boolean success) {
20         events.add(getEvent(service.getName(), success));
21     }
22
23     private String getEvent(String serviceName, boolean success) {
24         return serviceName + (success ? "started" : "failed");
25     }
26
27     public static void notifyServiceListener(ServiceListener serviceListener,
28             Service service, boolean success) {
29         if (serviceListener != null) {
30             if (success) {
31                 serviceListener.onSuccess(service);
32             } else {
33                 serviceListener.onFailure(service);
34             }
35         }
36     }
37
38     public void add(Service someService) {
39         services.add(someService);
40     }
41
42     public static int startServiceStaticWay(Service service) {
43         int returnCode = service.start();
44         return returnCode;
45     }
46
47     public void setServiceListener(ServiceListener serviceListener) {
48         this.serviceListener = serviceListener;
49     }
50
51     public List getEvents() {
52         return events;
53     }
54 }
```

Service:

```
1 package com.javacodegeeks.mockito;
2
3 public interface Service {
4     String getName();
5     int start();
6 }
```

ServiceListener:

```
1 package com.javacodegeeks.mockito;
2
3 public interface ServiceListener {
4     void onSuccess(Service service);
5     void onFailure(Service service);
6 }
```

3. Integrate PowerMockito and Mockito

In

```
setupMock()
```

, we will set up our system. We will create mock objects for

```
Service
```

and

`ServiceListener`

using

`Mockito.mock. B`

oth are interfaces and we don't have the actual implementations ready. Since

`SomeSystem`

is our SUT, we will create a spy object of it so that later we can stub some of its behavior.

Now let's come to our first test

`startSystem`

:

1. We will stub

   `service.start()`

   using PowerMockito so that it returns 1.
2. Next, we start the system calling

   `system.start()`

3. Finally, we will verify the behavior using Mockito's

   `verify()`

   API

```
1 Mockito.verify(serviceListener).onSuccess(service);
```

Notice that we stub using PowerMockito but verify using Mockito. This shows that Powermock doesn't re-invent the wheel rather enhances the existing testing frameworks.

PowerMockitoIntegrationExample:

```
01 package com.javacodegeeks.mockito;
02
03 import org.junit.Before;
04 import org.junit.Test;
05 import org.junit.runner.RunWith;
06 import org.mockito.Mockito;
07 import org.powermock.api.mockito.PowerMockito;
08 import org.powermock.modules.junit4.PowerMockRunner;
09
10
11 @RunWith(PowerMockRunner.class)
12 public class PowerMockitoIntegrationExample {
13     private Service service;
14     private SomeSystem system;
15     private ServiceListener serviceListener;
16
17     @Before
18     public void setupMock() {
19         // Mock
20         service = Mockito.mock(Service.class);
21         serviceListener = Mockito.mock(ServiceListener.class);
22
23         system = Mockito.spy(new SomeSystem());
24         system.add(service);
25         system.setServiceListener(serviceListener);
26     }
27
28     @Test
29     public void startSystem() {
30         // Stub using Mockito and PowerMockito
31         p("Stub using PowerMockito. service.start() should return 1 as we want start of the
service to be successful");
32         PowerMockito.when(service.start()).thenReturn(1);
33
34         // Run
35         p("Start the system, should start the services in turn");
36         system.start();
37
38         // Verify using Mockito
39         p("Verify using Mockito that service started successfuly");
40         Mockito.verify(serviceListener).onSuccess(service);
41
42         p("Verifed. Service started successfully");
43     }
44
45     private void p(String s) {
46         System.out.println(s);
47     }
48 }
```

Output:

```
1 Stub using PowerMockito. service.start() should return 1 as we want start of the service to be
```

```
successful
Start the system, should start the services in turn
Verify using Mockito that service started successfuly
Verifed. Service started successfully
```

4. Mocking Static Method

The use of static methods goes against the Object Oriented concepts but in real world we still use a lot of static methods and there are times when it makes sense to use static methods. Nevertheless, the ability to mock static methods may come handy to us. In this example, we will stub a static non-void method.

In the beginning of test class you will notice

```
@RunWith
```

annotation that contains

```
PowerMockRunner.class
```

as value. This statement tells JUnit to execute the test using

```
PowerMockRunner
```

.

You may also see annotation

```
@PrepareForTest
```

which takes the class to be mocked. This is required when we want to mock final classes or methods which either final, private, static or native.

We will use

```
PowerMockito.mockStatic
```

statement which takes in the class to be mocked. It tells PowerMockito to mock all the static methods. We then stub the static method's behavior.

For example, in

```
stubStaticNonVoidMethod
```

, we stub

```
SomeSystem.startServiceStaticWay
```

to return 1.

```
PowerMockito.when(SomeSystem.startServiceStaticWay(service)).thenReturn(1);
```

PowerMockitoStaticMethodExample:

```
package com.javacodegeeks.mockito;

import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.Mockito;
import org.powermock.api.mockito.PowerMockito;
import org.powermock.modules.junit4.PowerMockRunner;


@RunWith(PowerMockRunner.class)
public class PowerMockitoStaticMethodExample {
    private Service service;
    private SomeSystem system;
    private ServiceListener serviceListener;

    @Before
    public void setupMock() {
        // Mock
        service = Mockito.mock(Service.class);
        serviceListener = Mockito.mock(ServiceListener.class);

        system = new SomeSystem();
        //system = Mockito.spy(new SomeSystem());
        system.add(service);
        system.setServiceListener(serviceListener);
    }

    @Test
    public void stubStaticNonVoidMethod() {
        // Stub static method startServiceStatic to start successfully
        p("Call mockStatic SomeSystem.class to enable static mocking");
        PowerMockito.mockStatic(SomeSystem.class);

        p("Stub static method startServiceStaticWay to return 1");
        PowerMockito.when(SomeSystem.startServiceStaticWay(service))
                .thenReturn(1);

        // Run
        p("Start the system, should start the services in turn");
```

```
        system.start();

        // Verify success
        p("Verify using Mockito that service started successfuly");
        Mockito.verify(serviceListener).onSuccess(service);

        // Stub static method startServiceStatic to fail
        p("Stub static method startServiceStaticWay to return 0");
        PowerMockito.when(SomeSystem.startServiceStaticWay(service))
                .thenReturn(0);

        // Run
        p("Start the system again");
        system.start();

        // Verify failure
        p("Verify using Mockito that service has failed");
        Mockito.verify(serviceListener).onFailure(service);
    }

    private void p(String s) {
        System.out.println(s);
    }
}
```

Output:

```
Call mockStatic SomeSystem.class to enable static mocking
Stub static method startServiceStaticWay to return 1
Start the system, should start the services in turn
Verify using Mockito that service started successfuly
Stub static method startServiceStaticWay to return 0
Start the system again
Verify using Mockito that service has failed
```

5. Mocking static void Method

In this example, we will mock a void static method. The first step would be to call

```
PowerMockito.mockStatic
```

similar to the static non-void method. Since a void method doesn't return anything, the earlier way of mocking static methods won't work here.

```
PowerMockito.doNothing().when(SomeSystem.class);
```

Next, we will stub the behavior. After stubbing, we will call the static method on which it applies.

```
SomeSystem.notifyServiceListener(serviceListener, service, true);
```

We will follow similar style for verifying a static void method.

```
PowerMockito.verifyStatic();
SomeSystem.startServiceStaticWay(service);
```

PowerMockitoStaticVoidMethodExample:

```
package com.javacodegeeks.mockito;

import org.junit.Before;
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)
public class PowerMockitoStaticVoidMethodExample {
    private Service service;
    private SomeSystem system;
    private ServiceListener serviceListener;

    @Before
    public void setupMock() {
        service = Mockito.mock(Service.class);
        serviceListener = Mockito.mock(ServiceListener.class);

        system = new SomeSystem();
        system.add(service);
        system.setServiceListener(serviceListener);
    }

    @PrepareForTest({ SomeSystem.class })
    @Test
    public void stubStaticVoidMethod() {
        p("Call mockStatic SomeSystem.class to enable static mocking");
        PowerMockito.mockStatic(SomeSystem.class);

        p("Stub static void method SomeSystem.notifyServiceListener to do nothing");
        PowerMockito.doNothing().when(SomeSystem.class);
        SomeSystem.notifyServiceListener(serviceListener, service, true);

        p("Stub using PowerMockito. service.start() should return 1 as we want start of the
```

```
service to be successful");
        PowerMockito.when(service.start()).thenReturn(1);

        p("Start the system");
        system.start();

        p("Verify static method startServiceStaticWay(service) is called");
        PowerMockito.verifyStatic();
        SomeSystem.startServiceStaticWay(service);

        p("Verify serviceListener.onSuccess(service) is not called as notifyServiceListener is
stubbed to do nothing");
        Mockito.verify(serviceListener, Mockito.never()).onSuccess(service);
    }

    private void p(String s) {
        System.out.println(s);
    }
}
```

Output:

```
Call mockStatic SomeSystem.class to enable static mocking
Stub static void method SomeSystem.notifyServiceListener to do nothing
Stub using PowerMockito. service.start() should return 1 as we want start of the service to be
successful
Start the system
Verify static method startServiceStaticWay(service) is called
Verify serviceListener.onSuccess(service) is not called as notifyServiceListener is stubbed to do
nothing
```

6. Subbing Private Method

Using PowerMockito we can stub as well as verify private methods. In this example, I will show you how to stub a private method.

Our private method

```
addEvent
```

adds an event to the list. The event will tell us know whether a service started successfully or failed. Since we can't access the private method, we will have to pass the SUT object, private method name along with the method arguments to

```
PowerMockito.doNothing().when()
```

method.

In test case

```
stubPrivateMethodAddEvent
```

, we stub

```
addEvent
```

to do nothing.

```
PowerMockito.doNothing().when(system, "addEvent", service, true)
```

In test case

```
stubPrivateMethodGetEventString
```

, we stub

```
getEvent
```

to return some hardcoded string.

```
PowerMockito.when(system, "getEvent", serviceA, true).thenReturn(serviceA_is_successful);
```

PowerMockitoStubPrivateMethodExample:

```
package com.javacodegeeks.mockito;


import org.junit.Before;
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;
import org.junit.Assert;


@PrepareForTest({ SomeSystem.class })
@RunWith(PowerMockRunner.class)
public class PowerMockitoStubPrivateMethodExample {
    private Service service;
    private SomeSystem system;
    private ServiceListener serviceListener;

    @Before
```

```
22      public void setupMock() {
23          // Mock
24          service = Mockito.mock(Service.class);
25          serviceListener = Mockito.mock(ServiceListener.class);
26
27          system = PowerMockito.spy(new SomeSystem());
28          system.add(service);
29          system.setServiceListener(serviceListener);
30      }
31
32      @Test
33      public void stubPrivateMethodAddEvent() throws Exception {
34          p("Stub using PowerMockito. service.start() should return 1 as we want start of the
    service to be successful");
35          PowerMockito.when(service.start()).thenReturn(1);
36
37          p("Stub service name to return serviceA");
38          Mockito.when(service.getName()).thenReturn("serviceA");
39
40          p("Stub private addEvent to do nothing");
41          PowerMockito.doNothing().when(system, "addEvent", service, true);
42
43          p("Start the system, should start the services in turn");
44          system.start();
45
46          p("Since we have stubbed addEvent, assert that system.getEvents() is empty");
47          Assert.assertTrue(system.getEvents().isEmpty());
48      }
49
50      @Test
51      public void stubPrivateMethodGetEventString() throws Exception {
52          final String serviceA = "serviceA";
53          final String serviceA_is_successful = serviceA + " is successful";
54          p("Stub using PowerMockito. service.start() should return 1 as we want start of the
    service to be successful");
55          PowerMockito.when(service.start()).thenReturn(1);
56
57          p("Stub service name to return serviceA");
58          Mockito.when(service.getName()).thenReturn(serviceA);
59
60          p("Stub private addEvent to do nothing");
61          PowerMockito.when(system, "getEvent", serviceA, true).thenReturn(serviceA_is_successful);
62
63          p("Start the system, should start the services in turn");
64          system.start();
65
66          p("Since we have stubbed getEvent, assert that system.getEvents() contains the event
    string");
67          Assert.assertTrue(!system.getEvents().isEmpty());
68          Assert.assertEquals(serviceA_is_successful, system.getEvents().get(0));
69          System.out.println(system.getEvents());
70      }
71
72      private void p(String s) {
73          System.out.println(s);
74      }
75  }
```

In

```
stubPrivateMethodAddEvent
```

, since we have stubbed

```
addEvent
```

to do nothing, no events will added to the list.

In

```
stubPrivateMethodGetEventString
```

, we confirm that the event string we have returned is found in the events.

Output:

```
01  Test stubPrivateMethodAddEvent:
02  Stub using PowerMockito. service.start() should return 1 as we want start of the service to be
    successful
03  Stub service name to return serviceA
04  Stub private addEvent to do nothing
05  Start the system, should start the services in turn
06  Since we have stubbed addEvent, assert that system.getEvents() is empty
07
08  Test stubPrivateMethodGetEventString:
09  Stub using PowerMockito. service.start() should return 1 as we want start of the service to be
    successful
10  Stub service name to return serviceA
11  Stub private addEvent to do nothing
12  Start the system, should start the services in turn
13  Since we have stubbed getEvent, assert that system.getEvents() contains the event string
14  [serviceA is successful]
```

7. Verifying Private Method

Verification is similar to stubbing and PowerMockito allows us to verify even the private methods. The name of the method is passed to the

```
PowerMockito.verifyPrivate
```

along with its arguments.

```
PowerMockito.verifyPrivate(system).invoke("addEvent", new Object[] { service, true });
```

PowerMockitoVerifyPrivateMethodExample:

```
package com.javacodegeeks.mockito;


import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.Mockito;
import org.powermock.api.mockito.PowerMockito;
import org.powermock.modules.junit4.PowerMockRunner;

@RunWith(PowerMockRunner.class)
public class PowerMockitoVerifyPrivateMethodExample {
    private Service service;
    private SomeSystem system;
    private ServiceListener serviceListener;

    @Before
    public void setupMock() {
        // Mock
        service = Mockito.mock(Service.class);
        serviceListener = Mockito.mock(ServiceListener.class);

        system = Mockito.spy(new SomeSystem());
        system.add(service);
        system.setServiceListener(serviceListener);
    }

    @Test
    public void verifyPrivateMethods() throws Exception {
        p("Stub using PowerMockito. service.start() should return 1 as we want start of the
service to be successful");
        PowerMockito.when(service.start()).thenReturn(1);

        p("Stub service name to return serviceA");
        Mockito.when(service.getName()).thenReturn("serviceA");

        p("Start the system, should start the services in turn");
        system.start();

        p("Verify private method addEvent(service, true) is called");
        PowerMockito.verifyPrivate(system).invoke("addEvent",
                new Object[] { service, true });
        p("Verified private method is called");
    }

    private void p(String s) {
        System.out.println(s);
    }
}
```

Output:

```
Stub using PowerMockito. service.start() should return 1 as we want start of the service to be
successful
Stub service name to return serviceA
Start the system, should start the services in turn
Verify private method addEvent(service, true) is called
Verified private method is called
```

8. Download Source Code

This example was about PowerMockito and Mockito integration.

Download
You can download the full source code of this example here: **powerMockitoIntegration.zip**

Tagged with: POWERMOCK