Mock static methods in JUnit with PowerMock example

automationrhapsody.com/mock-static-methods-junit-powermock-example

By Lyudmil Latinov October 30, 2016

Post summary: Examples how to mock static methods in JUnit tests with PowerMock.

This post is part of PowerMock series examples. The code shown in examples below is available in GitHub java-samples/junit repository.

In Mock JUnit tests with Mockito example post, I have shown how and why to use Mockito java mocking framework to create good unit tests. There are several things that Mockito is not supporting, but one of them is mocking of static methods. It is not that common to encounter such situation is real life, but the moment you encounter it Mockito is not able to solve the task. This is where PowerMock comes to the rescue.

PowerMock

PowerMock is a framework that extends other mock libraries giving them more powerful capabilities. PowerMock uses a custom classloader and bytecode manipulation to enable mocking of static methods, constructors, final classes and methods, private methods, removal of static initializers and more.

Example class for unit test

We are going to unit test a class called***LocatorService*** that internally uses a static method from utility class ***Utils***. Method ***randomDistance(int distance)*** in ***Utils*** is returning random variable, hence it has no predictable behavior and the only way to test it is by mocking it:

```
public class LocatorService {
public Point generatePointWithinDistance(Point point, int distance) {
return new Point(point.getX() + Utils.randomDistance(distance),
point.getY() + Utils.randomDistance(distance));
}
}

```

And Utils class is:

```
import java.util.Random;
public final class Utils {
private static final Random RAND = new Random();
private Utils() {
}
public static int randomDistance( int distance) {
return RAND.nextInt(distance + distance) - distance;
}
}
```

Nota bene: it is good code design practice to make utility classes final and with a private constructor.

Using PowerMock

In order to use PowerMock two things has to be done:

1. Import PowerMock into the project
2. Annotate unit test class
3. Mock the static class

Import PowerMock into the project

In case of using Maven import statement is:

```
< dependency >
< groupId >org.powermock</ groupId >
< artifactId >powermock-module-junit4</ artifactId >
< version >1.6.5</ version >
< scope >test</ scope >
</ dependency >
< dependency >
< groupId >org.powermock</ groupId >
< artifactId >powermock-api-mockito</ artifactId >
< version >1.6.5</ version >
< scope >test</ scope >
</ dependency >
```

Nota bene: there is a possibility of version mismatch between PowerMock and Mockito. I've received: ***java.lang.NoSuchMethodError: org.mockito.mock.MockCreationSettings.isUsingConstructor()Z*** exception when using PowerMock 1.6.5 with Mockito 1.9.5, so I had to upgrade to Mockito 1.10.19.

Annotate JUnit test class

Two annotations are needed. One is to run unit test with PowerMockRunner: ***@RunWith(PowerMockRunner.class)***. Other is to prepare Utils class for testing: ***@PrepareForTest({Utils.class})***. The final code is:

```java
import org.junit.runner.RunWith;
import org.powermock.core.classloader.annotations.PrepareForTest;
import org.powermock.modules.junit4.PowerMockRunner;
@RunWith (PowerMockRunner. class )
@PrepareForTest ({Utils. class })
public class LocatorServiceTest {
}

```

Mock static class

Explicit mocking to static class should be made in order to be able to use standard Mockito when().thenReturn() construction:

```java
int distance = 111 ;
PowerMockito.mockStatic(Utils. class );
when(Utils.randomDistance(anyInt())).thenReturn(distance);
```

Putting it all together

Final JUnit test class is shown below. The code in tests verifies logic in***LocatorService***, if a point is given then new point is returned by adding random to its X and Y coordinates. By removing the random element with mocking code can be tested with specific values.

```
package com.automationrhapsody.junit;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.powermock.api.mockito.PowerMockito;
import org.powermock.core.classloader.annotations.PrepareForTest;
import org.powermock.modules.junit4.PowerMockRunner;
import static org.junit.Assert.assertTrue;
import static org.mockito.Matchers.anyInt;
import static org.mockito.Mockito.when;
@RunWith (PowerMockRunner. class )
@PrepareForTest ({Utils. class })
public class LocatorServiceTest {
private LocatorService locatorServiceUnderTest;
@Before
public void setUp() {
PowerMockito.mockStatic(Utils. class );
locatorServiceUnderTest = new LocatorService();
}
@Test
public void testGeneratePointWithinDistance() {
int distance = 111 ;
when(Utils.randomDistance(anyInt())).thenReturn(distance);
Point input = new Point( 11 , 11 );
Point expected = new Point(input.getX() + distance,
input.getY() + distance);
assertTrue(arePointsEqual(expected,
locatorServiceUnderTest.generatePointWithinDistance(input, 1 )));
}
public static boolean arePointsEqual(Point p1, Point p2) {
return p1.getX() == p2.getX()
&& p1.getY() == p2.getY();
}
}
```

Conclusion

PowerMock is a powerful addition to standard mocking libraries as Mockito. Using it has some specifics, but once you understand them it is easy and fun to use it. Keep in mind that if you encounter a need to use PowerMock that can mean that code under test is not well designed. In my experience, it is possible to have very good unit tests with more than 85% coverage without any PowerMock usage. Still, there are some exceptional cases where PowerMock can be put in operation.

Related Posts

- PowerMock examples and why better not to use them
- Verify static method was called with PowerMock
- Mock JUnit tests with Mockito example

Category: Java, Unit testing | Tags: JUnit, Mockito, PowerMock