Mock new object creation with PowerMock

automationrhapsody.com/mock-new-object-creation-powermock

By Lyudmil Latinov May 29, 2017

Post summary: How to control what objects are being instantiated when using PowerMock.

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

Mock new object creation

You might have a method which instantiates some object and works with it. This case could be very tricky to automate because you do not have any control over this newly created object. This is where PowerMock comes to help to allow you to control what object is being created by replacing it with an object you can control.

Code to test

Below is a simple method where a new object is being created inside a method that has to be unit tested.

```
1 public class PowerMockDemo {
2 public Point publicMethod() {
3 return new Point( 11 , 11 );
4 }
5 }
6
```

Unit test

What we want to achieve in the unit test is to control instantiation of new Point object so that it is replaced with an object we have control over. The first thing to do is to annotate unit test with ***@RunWith(PowerMockRunner.class)*** telling JUnit to use PowerMock runner and with ***@PrepareForTest(PowerMockDemo.class)*** telling PowerMock to get inside ***PowerMockDemo*** class and prepare it for mocking. Mocking is done with ***PowerMockito.whenNew(Point.class).withAnyArguments().thenReturn(mockPoint)***. It tells PowerMock when a new object from class ***Point*** is instantiated with whatever arguments to return ***mockPoint*** instead. It is possible to return different objects based on different arguments Point is created with ***withArguments()*** method. Full code is below:

```
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.hamcrest.CoreMatchers.is;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.mockito.Mockito.mock;
@RunWith (PowerMockRunner. class )
@PrepareForTest (PowerMockDemo. class )
public class PowerMockDemoTest {
private PowerMockDemo powerMockDemo;
@Before
public void setUp() {
powerMockDemo = new PowerMockDemo();
}
@Test
public void testMockNew() throws Exception {
Point mockPoint = mock(Point. class );
PowerMockito.whenNew(Point. class )
.withAnyArguments().thenReturn(mockPoint);
Point actualMockPoint = powerMockDemo.publicMethod();
assertThat(actualMockPoint, is(mockPoint));
}
}
```

Conclusion

PowerMock allows you to control want new objects are being created and replacing them with an object you have control over.

Related Posts

PowerMock examples and why better not to use them

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