ABOUT ROGER HUGHES

Using PowerMock to Mock Constructors

Posted by: Roger Hughes in Core Java September 19th, 2012 0 5 Views (*0 rating, 0 votes*)

In my opinion, one of the main benefits of dependency injection is that you can inject mock and/or stub objects into your
improve testability, increase test coverage and write better and more meaningful tests. There are those times, however, w
across some legacy code that doesn't use dependency injection and held together by composition rather than aggregatio

When this happens, you have three options:

1. Ignore the problem and not write any tests.
2. Refactor like mad, changing everything to use dependency injection.
3. Use PowerMock to mock constructors

Obviously, option 1 isn't a serious option, and although I'd recommend refactoring to move everything over to dependenc
takes time and you have to be pragmatic. That's where PowerMock comes in… this blog demonstrates how to use Power
constructor, which means that when your code calls new it doesn't create a real object, it creates a mock object.

To demonstrate this idea, the first thing we need is some classes to test, which are shown below.

```
public class AnyOldClass {

  public String someMethod() {
    return "someMethod";
  }

}
```

```
public class UsesNewToInstantiateClass {

  public String createThing() {

    AnyOldClass myclass = new AnyOldClass();

    String returnValue = myclass.someMethod();
    return returnValue;

  }

}
```

The first class, AnyOldClass, is the class that the code instantiates by calling new. In this example, as the name suggests,

The second class, the aptly named UsesNewToInstantiateClass, has one method, createThing(), which when called does a

```
AnyOldClass myclass = new AnyOldClass();
```

This is all pretty straight forward, so we'll move quickly on to the PowerMock assisted JUnit test:

```
import static org.easymock.EasyMock.expect;
import static org.junit.Assert.assertEquals;
import static org.powermock.api.easymock.PowerMock.expectNew;
import static org.powermock.api.easymock.PowerMock.replay;
import static org.powermock.api.easymock.PowerMock.verify;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.powermock.api.easymock.annotation.Mock;
import org.powermock.core.classloader.annotations.PrepareForTest;
import org.powermock.modules.junit4.PowerMockRunner;

@RunWith(PowerMockRunner.class)
@PrepareForTest(UsesNewToInstantiateClass.class)
public class MockConstructorTest {

  @Mock
  private AnyOldClass anyClass;

  private UsesNewToInstantiateClass instance;

  @Test
  public final void testMockConstructor() throws Exception {

    instance = new UsesNewToInstantiateClass();

    expectNew(AnyOldClass.class).andReturn(anyClass);

    final String expected = "MY_OTHER_RESULT";
    expect(anyClass.someMethod()).andReturn(expected);

    replay(AnyOldClass.class, anyClass);
    String result = instance.createThing();
    verify(AnyOldClass.class, anyClass);
    assertEquals(expected, result);

  }

}
```

Firstly, this class has the usual PowerMock additions of:

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

at the top of the file plus the creation of the anyOldClass mock object. The important line of code to consider is:

```
expectNew(AnyOldClass.class).andReturn(anyClass);
```

This line of code tells PowerMock to expect a call to new AnyOldClass() and return our anyClass mock object.

Also of interest are the calls to replay and verify. In the example above, they both have two arguments. The first, AnyOld
the expectNew(…) call above, whilst the second, anyClass refers to the straight forward mock call
expect(anyClass.someMethod()).andReturn(expected);.

There are those times when you should really let new do what it does: create a new object of the requested type. There i
that says you can over-isolate your code when testing and that mocking everything reduces the meaning and value of a t
right answer to this and it's a matter of choice.

It's fairly obvious that if your code accesses an external resource such as a database, then you'd either refactor and imple
PowerMock. If your code under test doesn't access any external resources, then it's more of a judgement call on how mu
too much? This perhaps needs some thought and may be the subject for another blog on anther day…

Reference: Using PowerMock to Mock Constructors from our JCG partner Roger Hughes at "Captain Debug's" Blog.

Tagged with: JUNIT POWERMOCK

(***0*** *rating,* ***0*** *votes*)

You need to be a registered member to rate this. Start the discussion 5 Views Tweet it!

LIKE THIS ARTICLE? READ MORE FROM JAVA CODE GEEKS