# Mocking Static Blocks in Java

Asked 16 years, 3 months ago  Modified 4 years, 2 months ago  Viewed 63k times

54

My motto for Java is "just because Java has static blocks, it doesn't mean that you should be using them." Jokes aside, there are a lot of tricks in Java that make testing a nightmare. Two of the most I hate are Anonymous Classes and Static Blocks. We have a lot of legacy code that make use of Static Blocks and these are one of the annoying points in our push in writing unit tests. Our goal is to be able to write unit tests for classes that depend on this static initialization with minimal code changes.

So far my suggestion to my colleagues is to move the body of the static block into a private static method and call it `staticInit`. This method can then be called from within the static block. For unit testing another class that depends on this class could easily mock `staticInit` with JMockit to not do anything. Let's see this in example.

```
public class ClassWithStaticInit {
  static {
    System.out.println("static initializer.");
  }
}
```

Will be changed to

```
public class ClassWithStaticInit {
  static {
    staticInit();
  }

  private static void staticInit() {
    System.out.println("static initialized.");
  }
}
```

So that we can do the following in a JUnit.

```
public class DependentClassTest {
  public static class MockClassWithStaticInit {
    public static void staticInit() {
    }
  }

  @BeforeClass
  public static void setUpBeforeClass() {
    Mockit.redefineMethods(ClassWithStaticInit.class,
MockClassWithStaticInit.class);
  }
}
```

However this solution also comes with its own problems. You can't run `DependentClassTest` and `ClassWithStaticInitTest` on the same JVM since you actually want the static block to run for `ClassWithStaticInitTest`.

What would be your way of accomplishing this task? Or any better, non-JMockit based solutions that you think would work cleaner?

java unit-testing mocking jmockit static-block

Share
Improve this question
Follow

edited Dec 21, 2019 at 14:40

Zoran Jankov
236 ● 2 ● 5 ● 18

asked Sep 14, 2008 at 5:31

Cem Catikkas
7,174 ● 4 ● 30 ● 33

Any idea on how to do this using Mokito framework? – saumilsdk Jul 26, 2020 at 17:56

## 10 Answers

Sorted by: Highest score (default)

65

PowerMock is another mock framework that extends EasyMock and Mockito. With PowerMock you can easily remove unwanted behavior from a class, for example a static initializer. In your example you simply add the following annotations to your JUnit test case:

```
@RunWith(PowerMockRunner.class)
@SuppressStaticInitializationFor("some.package.ClassWithStaticInit")
```

PowerMock does not use a Java agent and therefore does not require modification of the JVM startup parameters. You simple add the jar file and the above annotations.

Share Improve this answer Follow

answered Jan 28, 2009 at 19:40

Jan Kronquist
2,803 ● 25 ● 20

14

Occasionally, I find static initilizers in classes that my code depends on. If I cannot refactor the code, I use PowerMock's `@SuppressStaticInitializationFor` annotation to suppress the static initializer:

```
@RunWith(PowerMockRunner.class)
@SuppressStaticInitializationFor("com.example.ClassWithStaticInit")
public class ClassWithStaticInitTest {
```

```
    ClassWithStaticInit tested;

    @Before
    public void setUp() {
        tested = new ClassWithStaticInit();
    }

    @Test
    public void testSuppressStaticInitializer() {
        asserNotNull(tested);
    }

    // more tests...
}
```

Read more about suppressing unwanted behaviour.

Disclaimer: PowerMock is an open source project developed by two colleagues of mine.

Share Improve this answer Follow

answered Aug 30, 2011 at 11:20

matsev

33.7k ● 17 ● 125 ● 168

---

13

This is going to get into more "Advanced" JMockit. It turns out, you can redefine static initialization blocks in JMockit by creating a `public void $clinit()` method. So, instead of making this change

```
public class ClassWithStaticInit {
  static {
    staticInit();
  }

  private static void staticInit() {
    System.out.println("static initialized.");
  }
}
```

we might as well leave `ClassWithStaticInit` as is and do the following in the `MockClassWithStaticInit`:

```
public static class MockClassWithStaticInit {
  public void $clinit() {
  }
}
```

This will in fact allow us to not make any changes in the existing classes.

Share Improve this answer Follow

answered Sep 28, 2008 at 0:38

Cem Catikkas
7,174 ● 4 ● 30 ● 33

6

When I run into this problem, I usually do the same thing you describe, except I make the static method protected so I can invoke it manually. On top of this, I make sure that the method can be invoked multiple times without problems (otherwise it is no better than the static initializer as far as the tests go).

This works reasonably well, and I can actually test that the static initializer method does what I expect/want it to do. Sometimes it is just easiest to have some static initialization code, and it just isn't worth it to build an overly complex system to replace it.

When I use this mechanism, I make sure to document that the protected method is only exposed for testing purposes, with the hopes that it won't be used by other developers. This of course may not be a viable solution, for example if the class' interface is externally visible (either as a sub-component of some kind for other teams, or as a public framework). It is a simple solution to the problem though, and doesn't require a third party library to set up (which I like).

Share Improve this answer Follow

answered Sep 14, 2008 at 9:17

Mike Stone
44.6k ● 30 ● 114 ● 140

6

Sounds to me like you are treating a symptom: poor design with dependencies on static initialization. Maybe some refactoring is the real solution. It sounds like you've already done a little refactoring with your `staticInit()` function, but maybe that function needs to be called from the constructor, not from a static initializer. If you can do away with static initializers period, you will be better off. Only you can make this decision (*I can't see your codebase*) but some refactoring will definitely help.

As for mocking, I use EasyMock, but I have run into the same issue. Side effects of static initializers in legacy code make testing difficult. Our answer was to refactor out the static initializer.

Share Improve this answer Follow

answered Sep 14, 2008 at 5:55

Justin Standard
21.5k ● 22 ● 82 ● 90

You can't mock `static` or `private` methods with EasyMock. Moving the body of the static initializer is as far of a refactoring we can do for now. – Cem Catikkas Sep 14, 2008 at 18:26

If the class under test has the static init / private method, you want it to be called. No problem. But if it is the class being mocked, no problem for easy mock: it won't get called because

there is NO IMPLEMENTATION. You can safely mock the public interfaces as if the private stuff didn't exist. – Justin Standard Sep 16, 2008 at 4:47

3

You could write your test code in Groovy and easily mock the static method using metaprogramming.

```
Math.metaClass.'static'.max = { int a, int b ->
    a + b
}

Math.max 1, 2
```

If you can't use Groovy, you will really need to refactoring the code (maybe to inject something like a initializator).

Kind Regards

Share Improve this answer Follow

answered Sep 14, 2008 at 7:42

marcospereira
12.2k ● 3 ● 48 ● 53

I love Groovy but unfortunately all our test code must be in JUnit. – Cem Catikkas Sep 14, 2008 at 18:27

Cem, But you could write junit (even junit4) tests in groovy. ;-) Kind Regards – marcospereira Sep 15, 2008 at 4:04

1

I suppose you really want some kind of factory instead of the static initializer.

Some mix of a singleton and an abstract factory would probably be able to get you the same functionality as today, and with good testability, but that would add quite a lot of boiler-plate code, so it might be better to just try to refactor the static stuff away completely or if you could at least get away with some less complex solution.

Hard to tell if it´s possible without seeing your code though.

Share Improve this answer Follow

answered Sep 14, 2008 at 7:12

Henrik Gustafsson
53.9k ● 9 ● 50 ● 60

1

I'm not super knowledgeable in Mock frameworks so please correct me if I'm wrong but couldn't you possibly have two different Mock objects to cover the situations that you mention? Such as

```
public static class MockClassWithEmptyStaticInit {
  public static void staticInit() {
  }
}
```

and

```
public static class MockClassWithStaticInit {
  public static void staticInit() {
    System.out.println("static initialized.");
  }
}
```

Then you can use them in your different test cases

```
@BeforeClass
public static void setUpBeforeClass() {
  Mockit.redefineMethods(ClassWithStaticInit.class,
                         MockClassWithEmptyStaticInit.class);
}
```

and

```
@BeforeClass
public static void setUpBeforeClass() {
  Mockit.redefineMethods(ClassWithStaticInit.class,
                         MockClassWithStaticInit.class);
}
```

respectively.

Share Improve this answer Follow

answered Sep 14, 2008 at 14:55

martinatime

2,487 ● 1 ● 17 ● 23

`System.out.println("static initialized.");` is what the static initializer is doing in the first place. Why should we mock what it already is doing with the same thing?
– Cem Catikkas Sep 14, 2008 at 18:29

Cem, I think you missed my point. I'm trying to answer the concern "However this solution also comes with its own problems. You can't run DependentClassTest and ClassWithStaticInitTest on the same JVM since you actually want the static block to run for ClassWithStaticInitTest." – martinatime Sep 14, 2008 at 19:16

Basically, you have two different Mock Objects to Mock ClassWithStaticInit one for use with your DependentClassTest and the other for use with ClassWithStaticInitTest. – martinatime Sep 14, 2008 at 19:18

Well, there the problem is the static block is run only once per JVM. So you can either have the original static block run or your mock one. The `ClassWithStaticInit` depends on the fact that it has a static block. So for `ClassWithStaticInitTest you shouldn't really be mocking` ClassWithStaticInit` – Cem Catikkas Sep 14, 2008 at 20:47

1

You can use PowerMock to execute the private method call like:

```
ClassWithStaticInit staticInitClass = new ClassWithStaticInit()
Whitebox.invokeMethod(staticInitClass, "staticInit");
```

Share Improve this answer Follow

answered Oct 5, 2020 at 21:36

Sebastian Luna

496 ● 1 ● 7 ● 19

0

Not really an answer, but just wondering - isn't there any way to "reverse" the call to `Mockit.redefineMethods`?
If no such explicit method exists, shouldn't executing it again in the following fashion do the trick?

```
Mockit.redefineMethods(ClassWithStaticInit.class, ClassWithStaticInit.class);
```

If such a method exists, you could execute it in the class' `@AfterClass` method, and test `ClassWithStaticInitTest` with the "original" static initializer block, as if nothing has changed, from the same JVM.

This is just a hunch though, so I may be missing something.

Share Improve this answer Follow

answered Nov 16, 2015 at 14:17

KidCrippler

1,723 ● 2 ● 20 ● 35