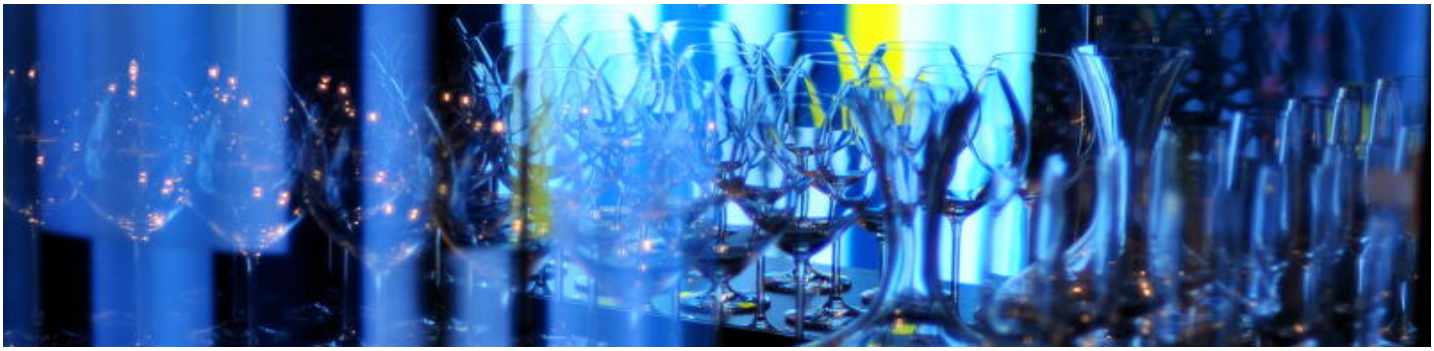
AndyPatterns



Easy Dependency Injection and Mocking in Ruby

Here is a simple technique where you can inject mock methods into a class that you are unit testing.

Given you have a class **classA**

```
class ClassA
  def hi
    "hi"
  end
end
```

You can replace the "hi" method from your unit test, without editing the source code of ClassA at all. You can inject mock methods - whatever.

```
def test_2
  ClassA.class_eval do # injecting into ClassA class
    def hi
      "hello"
    end
  end
  a = ClassA.new
  assert_equal "hello", a.hi
end
```

The trouble with **class_eval** is that it alters the class permanently, which is not so good for repeated unit testing where you may be injecting different things in different tests, or not injecting anything at all in some tests etc. I couldn't figure out how to restore a class in setup/teardown so....**instance_eval** is probably better as it only affects an instance. Viz:

```
def test_2
  a = ClassA.new
  a.instance_eval do # injecting into ClassA instance
    def hi
      "hello"
    end
  end
  assert_equal "hello", a.hi
end
```

The above injection (replacement of the 'hi' method with a different 'hi' method) does not affect the class, but the instance ONLY.

Source code

Comments

Posted by RB on Feb 17th, 2011

Your example and use-case looks good.

However, it might be nice to have a Ruby testing framework that completely resets your environment after each unit test. Then you could safely use class_eval and have the comfort of only thinking about one unit at a time.

Rails unit testing doesn't do environment resetting (afaik) (on the code level) - but it does do it on the database level. After each unit-test, the database transaction is rolled-back, so each unit-test is not dependent or influenced by other unit tests.

There will also be other unit-testing situations where you will have to use class_eval. For example, you may be testing a method that builds many objects of a class.

Or, arguably, if using instance_eval, you would have to assume too much about how the code you are testing is written. In testing, all you want to do it test the interface: the high-level view from the outside. You shouldn't be able to know if the tested code is making new instances of classes containing the method you want to inject or fake-out. Eg a method in an object could make a new instance of its own class, without you knowing!

-RB

Posted by Jason on Feb 20th, 2011

I wonder what the best mocking tool is for Ruby and how it would fit into the above dependency injection scheme. Perhaps this technique is simple enough so you don't need a mocking framework - just create the mocks as regular classes inside the unit test and away you go.