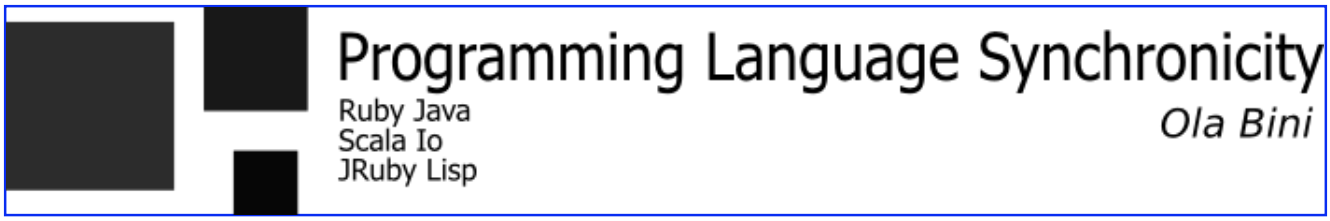

September 18th, 2008

Don’t overuse instance_eval and instance_exec

Not sure how known this antipattern is. I’ve seen some libraries that are very good at not doing it, but I’m also seeing lots of Ruby code that does use it without reason.

What I’m talking about here is the common usage of using instance_eval on a block to make it possible to use other methods inside of that specific block. If you want to do stuff based on method_missing inside of a block, this is the way people generally use.

So what’s the problem with it? Well, the problem is that blocks are generally closures. And you expect them to actually be full closures. And it’s not obvious from the point where you write the block that that block might not be a full closure. That’s what happens when you use instance_eval: you reset the self of that block into something else - this means that the block is still a closure over all local variables outside the block, but NOT for method calls. I don’t even know if constant lookup is changed or not.

Using instance_eval changes the rules for the language in a way that is not obvious when reading a block. You need to think an extra step to figure out exactly why a method call that you can lexically see around the block can actually not be called from inside of the block.

There are ways around instance_eval usage in a library - you can always assign self to a local variable outside of the block, and then call methods on that local variable. How ugly does that sound?

In almost all cases, if a block needs to handle method calls on a specific object, it should send that object in to the block as an argument. Take a look at routes in Rails - they could have been defined with instance_eval, but they’re not. There is no reason to use instance_eval for this case. Rails send in a route instead. File.open doesn’t use instance_eval with the block. It could, but instead it sends in the file. This is because there is no need to use instance_eval, and sending in the object as an argument gives the definition more power.

This doesn’t mean instance_eval should never be used. That’s not true, it’s a hugely useful feature, but it should definitely be used with good taste. If you’re unsure when to use it, **don’t**!

By Ola Bini | In: Uncategorized | tags: good taste, instance_eval, instance_exec, ruby. | #

4 Comments, Comment or Ping

1. Wayne Conrad

   Thank you for the sanity. The novice programmer is not to be feared. It’s the intermediate programmer, who knows the advanced tools but not when they should be applied.

   September 18th, 2008

2.  Dan Yoder

   I have this funny feeling I helped inspire this article. =)

   Functor 0.5.0 (currently on github, soon to be on RubyForge) has dropped instance_eval for the new block and passes self instead.

   A great example of when instance_eval semantics are used correctly is in the Module and Class constructors. Obviously, there it makes sense, since the initialization is likely to be things like defining methods.

   Another example where I think it is legit is when Functor uses instance_exec to execute a functor as a method.

   Thanks again for your feedback.

   September 22nd, 2008

3.  _why

   Constant lookup isn’t altered by instance_eval. They get resolved under the block’s original scope. This is one of those things where I don’t know what would follow the Principle of Least Surprise because it’s all surprising, in a way.

   October 8th, 2008

4.  Adrian Kuhn

   If you consider constant lookup, as just-another-method being sent (see Gilad’s Newspeak), then rebinding it to the new self is the least surprise.

   October 8th, 2008

Reply to “Don’t overuse instance_eval and instance_exec”

« NewSpeak at JavaZone
Taste of Eliza »