Ruby master - Feature #12715

Allow ruby hackers to omit having to specify class or module mandatory, if they know exactly what they want to do

08/30/2016 05:38 PM - shevegen (Robert A. Heiler)

Status:	Feedback	
Priority:	Normal	
Assignee:		
Target version:		

Description

Hello - I try to be somewhat short, as much as possible.

In ruby we have classes and modules. We can do module Foo; class Bar
and also class Foo; module Bar. Both is slightly different, also in
regards to errors you get when you try to extend either of these
two. The problem here is that, when we wish to add new behaviour to
a class or a module, we may have to know whether it is a class or a
module explicitely.

If we do not specify this correctly so, we may get an error like this
one here:

```
Bar is not a class (TypeError)
```

I believe, however had, that it should not be absolutely mandatory to
HAVE to know the type of a "class" or a "module", in particular not at
runtime when all you want to do is add new behaviour to an already
existing class or module. And if you know that class or module too.

I also believe that the current behaviour may not be changed for
several reasons; backwards compatibility; also a user may have
made a mistake or changed some class/module lateron, so we may
have to catch these errors too. I have run into this every now
and then e. g. when I had a standalone class in a project, but
lateron modified this class to become a subclass of another class;
then I also have to change the definitions for that class being
a subclass in other .rb files (I tend to create several .rb files
if my code becomes large, and some classes become really quite
huge in some of my projects; ~more than 200 lines of code is
already monster-sized in my opinion :)).

It would be nice if we could, however had, have another way to
modify the behaviour, in ruby.

I do not have a good suggestion here, so please, when you read
the following, keep in mind that I am aware that this is not
really perfect either - it just should serve as an illustration.

I will omit the "end"s to be more succinct.

```
module Foo; class Bar; def self.hi; puts 'hi from method hi()'
```

Ok, now I want to modify inner class Bar without having to care
whether it is a class or a module so I will use the word "modify"
as well:

```
modify Foo::Bar; def self.new_method; puts 'a new method!'
```

Here I would have used the (then new) keyword "modify".

I think the name somewhat fits. One could use other names, perhaps.

```
adapt   Foo::Bar; def self.new_method; puts 'a new method!'

mod     Foo::Bar; def self.new_method; puts 'a new method!'

change  Foo::Bar; def self.new_method; puts 'a new method!'

update  Foo::Bar; def self.new_method; puts 'a new method!'
```

Though I like modify as name more than the other variants.

I think that this probably has not a huge chance to be implemented,
most likely not for ruby 3.x; perhaps in the very distant future
towards ruby 4.x?

But I think the main reason why I actually wrote this suggestion
here, even though it probably does not have a huge chance to be
implemented, is mostly just to point out that I think I should not
have to be explicit about the class versus module dichotomy here when
all I want to do is really just add a class/module method (singleton
method?) to in particular already existing ruby classes/modules.

I do have to know whether it is a class or a module right now,
because in one way, it will work, in the other it will fail.

So my suggestion above would eliminate the possibility of the error,
simply by telling ruby "no matter if it is a class or a module,
simply add the code to it" and ruby can infer internally whether
it is a class or a module via a check.

I am not sure if any of this makes sense - I may have forgotten
some other things too - but thank you for reading it anyway!

History

#1 - 08/30/2016 05:42 PM - shevegen (Robert A. Heiler)

I forgot to add, in the above example:

```
modify Foo::Bar; def self.new_method; puts 'a new method!'
```

The current way would be either:

```
class Foo::Bar; def self.new_method; puts 'a new method!'
```

or

```
module Foo::Bar; def self.new_method; puts 'a new method!'
```

Which works perfectly fine if you know what inner Bar is, but
my point was about trying to not have to be this mandatory.
(An alternative would be to not fail on this, so both last
two ways would work; but as I wrote above, I assume that this
is not possible due to reasons stated above, hence why I
suggested a new keyword even though this may be unlikely to
be implemented due to several reasons - more keywords may also
make a language more complex, so ideally when possible I assume
that new keywords should be avoided.)

#2 - 08/30/2016 06:06 PM - jeremyevans0 (Jeremy Evans)

You can use class_eval/module_eval to do this in the examples provided, without caring if Foo::Bar is a class or module. For method definitions (including singleton method definitions), that should work fine. The only thing it doesn't handle is constant lookup or setup a new local variable scope. You can work around constant lookup/setting using self::. Example:

```
class Foo::Bar
  C = 1
end
```

becomes:

```
Foo::Bar.class_eval do
  self::C = 1
```

```
end
```

This is because class_eval doesn't setup a new CREF/local variable scope I believe (hopefully someone who knows more about the internals can correct me if I'm wrong). So the main need for this proposal would be a way in ruby to open up a new CREF/local variable scope for a given module/class.

I can see the pros and cons of supporting such a feature. If we do want to support it, I think we should allow module Foo::Bar to work if Foo::Bar is already defined as a class, but class Foo::Bar to not work if Foo::Bar is already defined as a module, with the logic that a class is a module but a module is not a class. This doesn't introduce a new keyword, and should not break any code that isn't specifically checking the the exception.

#3 - 08/30/2016 06:35 PM - nobu (Nobuyoshi Nakada)

- Description updated

I don't think this is acceptable, because it is possible already with class_eval/module_eval as Jeremy wrote, and new keyword like such common word breaks compatibility.
As for constant assignment, you can do it with const_set too.

#4 - 08/31/2016 01:21 AM - duerst (Martin Dürst)

I agree with Nobu that this is too minor an issue to introduce a new keyword.

Also, Robert wrote: "I do have to know whether it is a class or a module right now". But in my experience, that's usually easy to know. If you include it, it's a module. If you use new, it's a class. And so on. If you really don't know what it is, just adding some stuff to it may not be a good idea in the first place.

#5 - 08/31/2016 01:22 AM - duerst (Martin Dürst)

In addition, the error message is extremely straightforward, and the fix is extremely easy, so the benefit of this is low in that sense, too.

#6 - 11/25/2016 09:01 AM - matz (Yukihiro Matsumoto)

- Status changed from Open to Feedback