Ruby trunk - Feature #8839

Class and module should return the class or module that was opened

08/31/2013 02:57 AM - headius (Charles Nutter)

Status:	Assigned	
Priority:	Normal	
Assignee:	matz (Yukihiro Matsumoto)	
Target version:		

Description

With the change for https://bugs.ruby-lang.org/issues/3753, "def" forms now return the symbolic name of the method defined. Because class and module bodies just return the last expression in their bodies, this means they will now usually end up returning a symbol for the last method defined. This does not seem useful or correct.

I think class and module should return a reference to the class or module just opened. This would make the return value useful and consistent.

Related issues:

Related to Ruby trunk - Feature #11905: Change the 'class' keyword to return ...	**Closed**

History

#1 - 08/31/2013 03:34 AM - marcandre (Marc-Andre Lafortune)

When thinking of potential incompatibilities, the only case I could think of where I'd ever used the result of class or module was:

```
class << foo
  self
end
```

Ironically, the proposed change would not introduce an incompatibility in this case :-)

#2 - 08/31/2013 06:26 PM - Anonymous

+1

#3 - 09/26/2013 06:41 AM - headius (Charles Nutter)

So...if there's no objections, can we get this into 2.1? I think with the new def return value change it really needs to happen.

#4 - 09/26/2013 06:43 AM - enebo (Thomas Enebo)

+1

#5 - 09/26/2013 11:57 AM - nobu (Nobuyoshi Nakada)

No, class and def are evaluated in different timings.
This proposal makes no sense.

#6 - 09/26/2013 12:03 PM - charliesome (Charlie Somerville)

nobu: I don't understand what you mean by "evaluated in different timings". Could you please explain?

I think this proposal is a good idea. For example, this makes no sense:

```
class A
  def foo
  end
end # => :foo
```

But this makes more sense:

```
class A
  def foo
  end
end # => A
```

#7 - 09/26/2013 08:21 PM - mame (Yusuke Endoh)

- Status changed from Open to Assigned

- Assignee set to matz (Yukihiro Matsumoto)

- Target version set to 2.6

How useful is this proposal?
I think we should not change anything without consideration of use case.

--
Yusuke Endoh mame@tsg.ne.jp

#8 - 09/27/2013 04:22 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

I've been wondering the same thing since I saw this ticket being created...

#9 - 09/27/2013 04:22 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

Actually, I don't understand even why returning a symbol from method definition is useful...

#10 - 09/27/2013 07:29 PM - headius (Charles Nutter)

mame (Yusuke Endoh) wrote:

> How useful is this proposal?
> I think we should not change anything without consideration of use case.

One use:

```
my_class = class Foo
  ...
end
```

We can get the reference to a class being created immediately without adding "self" at the end. It also brings some equivalence with my_class = Class.new.

Another:

```
class Foo
  def self.init
    @foo = Foo.new
  end
end.init
```

The use cases I can think of are all fairly subtle, but I think they're valid.

Ultimately, the biggest reason I think this should happen is because class Foo; def bar; end; end returning :bar in 2.1 makes very little sense.

#11 - 09/27/2013 09:11 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

I see now. I really tried a few times to get something like your second example to work:

```
class MyProcessor
 ...
end.new(params).process
```

I'd usually use such pattern in small scripts since methods are not hoisting like functions declarations in JavaScript and I prefer to write code top-down, so I enclose them in a class.

#12 - 09/27/2013 11:33 PM - mame (Yusuke Endoh)

headius (Charles Nutter) wrote:

> One use:
>
> ```
> my_class = class Foo
>   ...
> end
> ```
>
> We can get the reference to a class being created immediately without adding "self" at the end.

I fail to see why it needs to be a local variable.
Why don't you use Foo instead of my_class?

> Another:

```
class Foo
  def self.init
    @foo = Foo.new
  end
end.init
```

The use cases I can think of are all fairly subtle, but I think they're valid.

It is very arguable if the new idiom should be encouraged.
Personally, I don't like to see such a code. Rather, I prefer:

```
class Foo
  def self.init
    @foo = Foo.new
  end
  self.init
end
```

or even:

```
class Foo
  @@foo = Foo.new
end
```

Ultimately, the biggest reason I think this should happen is because class Foo; def bar; end; end returning :bar in 2.1 makes very little sense.

IMHO, most of casual users do not care too much for a return value.
For example, we often see the following type of code:

```
def foo
  internal_ary = []
  # ... building internal_ary ...
  internal_ary.each do |elem|
    # ... using elem ...
  end
end
```

Obviously, the author of this method has no intent to return internal_ary.
But, few people writes a code that returns nil explicitly to hide the value.

```
def foo
  internal_ary = []
  # ...
  internal_ary.each do |elem|
    # ...
  end
  nil
end
```

--
Yusuke Endoh mame@tsg.ne.jp

#13 - 09/28/2013 03:39 AM - headius (Charles Nutter)

mame (Yusuke Endoh) wrote:

> headius (Charles Nutter) wrote:
>
> > One use:
> >
> > ```
> > my_class = class Foo
> >   ...
> > end
> > ```
> >
> > We can get the reference to a class being created immediately without adding "self" at the end.
>
> I fail to see why it needs to be a local variable.
> Why don't you use Foo instead of my_class?

How about this:

```
meta = class << self; end
```

> It is very arguable if the new idiom should be encouraged.
> Personally, I don't like to see such a code. Rather, I prefer:
>
> ```
> class Foo
>   def self.init
>     @foo = Foo.new
>   end
>   self.init
> end
> ```
>
> or even:
>
> ```
> class Foo
>   @@foo = Foo.new
> end
> ```

I respect your opinion, but I fail to see why your way is better than mine.

> > Ultimately, the biggest reason I think this should happen is because class Foo; def bar; end; end returning :bar in 2.1 makes very little sense.
>
> IMHO, most of casual users do not care too much for a return value.

Users not caring about return value is not a good reason to return a nonsensical value. Returning the last method name defined in a class body makes no sense. Returning the class that was just defined may not make sense to you, but it makes sense to me and several others who have commented here.

#14 - 09/28/2013 04:20 AM - robertgleeson (Robert Gleeson)

I agree that returning the class or module makes sense (to me).
I'd also like to see "def foo" return a (Unbound)Method instead of a Symbol.
it seems like that'd also make more sense (not to derail this conversation). A (Unbound)Method is much more useful.
I saw headius mention you can say:

```
class Foo
  instance_method def foo
    end # => UnboundMethod.
end
```

...but seems like a nice default.

+1 to the proposal.

#15 - 09/28/2013 08:41 AM - mame (Yusuke Endoh)

headius (Charles Nutter) wrote:

> mame (Yusuke Endoh) wrote:
>
> > headius (Charles Nutter) wrote:
> >
> > > One use:
> > >
> > > ```
> > > my_class = class Foo
> > >   ...
> > > end
> > > ```
> > >
> > > We can get the reference to a class being created immediately without adding "self" at the end.
> >
> > I fail to see why it needs to be a local variable.
> > Why don't you use Foo instead of my_class?
>
> How about this:
>
> ```
> meta = class << self; end
> ```

Use:

```
meta = self.singleton_class
```

> It is very arguable if the new idiom should be encouraged.
> Personally, I don't like to see such a code. Rather, I prefer:
>
> ```
> class Foo
>   def self.init
>     @foo = Foo.new
>   end
>   self.init
> end
> ```
>
> or even:
>
> ```
> class Foo
>   @@foo = Foo.new
> end
> ```

I respect your opinion, but I fail to see why your way is better than mine.

Your way is too easy to overlook ".init" because a method call is not usually expected there.

> Ultimately, the biggest reason I think this should happen is because class Foo; def bar; end; end returning :bar in 2.1 makes very little sense.

> IMHO, most of casual users do not care too much for a return value.

Users not caring about return value is not a good reason to return a nonsensical value.

"It makes no sense" is not a good reason to change anything.
Rather, it is consistent and compatible to just return the last value.
We need better reason to break compatibility, I think.

This is my personal opinion, of course.

--
Yusuke Endoh mame@tsg.ne.jp

#16 - 09/28/2013 03:39 PM - shugo (Shugo Maeda)

I did a quick hack to try this proposal and found that some test failed with it:

https://gist.github.com/shugo/6739085

For example, bootstraptest/test_block.rb uses the last value of a class definition as follows:

```
assert_equal 'ok', %q{
  STDERR.reopen(STDOUT)
  class C
    define_method(:foo) do |&block|
      block.call if block
    end
    result = "ng"
    new.foo() {result = "ok"}
    result
  end
}
```

If class definitions are changed to return the defined class, there'll be no way to get values created in class definitions except using global variables or constants, etc.
I'm not sure whether there's any use case of the current behavior other than testing purposes, but please consider there might be some users of the current behavior.

#17 - 09/30/2013 10:20 PM - headius (Charles Nutter)

shugo (Shugo Maeda) wrote:

> I did a quick hack to try this proposal and found that some test failed with it:
>
> https://gist.github.com/shugo/6739085

> For example, bootstraptest/test_block.rb uses the last value of a class definition as follows:

The case given is rather contrived; I have never seen code in the wild use last expression return from a class.

> If class definitions are changed to return the defined class, there'll be no way to get values created in class definitions except using global variables or constants, etc.

This is a fair point, I suppose, but I still see more reasons to make the return value consistent than leave it as is and have classes suddenly returning a symbol for the last defined method. Most folks probably don't even know about the return value since if you're just doing "def" as last expression it has always been nil.

#18 - 09/30/2013 11:53 PM - avdi (Avdi Grimm)

On Sat, Sep 28, 2013 at 2:39 AM, shugo (Shugo Maeda)
redmine@ruby-lang.orgwrote:

> For example, bootstraptest/test_block.rb uses the last value of a class
> definition as follows:

The only time I've ever used the return value of a class definition it's
been to get the class itself, by making self the last statement in the
class definition.

That said, if you really wanted to preserve the ability to return something
other from a class definition, could you make break <SOME_VALUE> override
the return, as it does in blocks?

--
Avdi Grimm
http://avdi.org

I only check email twice a day. to reach me sooner, go to
http://awayfind.com/avdi

#19 - 10/01/2013 09:37 AM - headius (Charles Nutter)

avdi (Avdi Grimm) wrote:

> That said, if you really wanted to preserve the ability to return something
> other from a class definition, could you make break <SOME_VALUE> override
> the return, as it does in blocks?

That wouldn't be backward-compatible with anyone expecting last expression, but it's an excellent idea to address Shugo's concern.

#20 - 11/07/2013 05:34 PM - jballanc (Joshua Ballanco)

Just to throw my 2¢ in...

I think the main benefit to returning a symbol from def is that it enables the use of method decorators. Similarly, I would be in favor of returning the class defined from class so that we could also build class decorators. A trivial example:

```
def enumerable(klass)
  klass.send(:include, Enumerable)
  klass
end

enumerable
class Foo
  #...
end
```

#21 - 11/08/2013 04:04 AM - headius (Charles Nutter)

jballanc (Joshua Ballanco) wrote:

> Just to throw my 2¢ in...
>
> I think the main benefit to returning a symbol from def is that it enables the use of method decorators. Similarly, I would be in favor of returning the class defined from class so that we could also build class decorators. A trivial example:
>
> ```
> def enumerable(klass)
> ```

```
    klass.send(:include, Enumerable)
    klass
  end

  enumerable
  class Foo
    #...
  end
```

Clever! Though I don't think that would parse the way you want. This would work though, obviously:

```
enumerable class Foo ...
```

#22 - 06/16/2014 09:05 PM - alexeymuranov (Alexey Muranov)

+1. I do not want to share my silly code where i would use it, but currently it may look something like that:

```
def MyClass
  # ...
  self
end.define_more_methods(x, y, z).freeze
```

#23 - 04/19/2015 10:46 AM - sawa (Tsuyoshi Sawada)

I think this proposal would be useful in cases like this:

```
class Foo
  prepend module Bar
    def baz
      ...
    end
  end
end
```

#24 - 06/11/2015 12:13 PM - cesario (Franck Verrot)

Shugo Maeda wrote:

> If class definitions are changed to return the defined class, there'll be no way to get values created in class definitions except using global variables or constants, etc.
> I'm not sure whether there's any use case of the current behavior other than testing purposes, but please consider there might be some users of the current behavior.

AFAICT, only one value (arguably it can be a complex object but I've never seen that kind of trick before) can be returned, so you'd still have to use globals/constants anyway. Am I mistaking here?

On a general level, it is surprising that class ...; end and Class.new don't return the same thing while def and define_method, something needs to be done to class to mimic this logic (even if I don't understand why we don't return a Method object today, like Rubinius does). Teaching Ruby, I'm saying class and Class.new are equivalent, but they're not with regards to this.

As a side-note, I'd like to present another use case. I've built a quick gem that loads Ruby code with rb_eval_string_protect and make this possible:

```
# foo.rb
require 'rb_import'
module Foo
  Bar = import("./my_class.rb")
end

f = import("./my_class.rb")

puts f.new.say_hello == Foo::Bar.new.say_hello # => true
puts f == Foo::Bar # => false

# my_class.rb
Class.new do
  def say_hello
    puts "Hello from my_class.rb"
  end
end
```

But in my_class.rb, I can't simply use a regular class definition "class Foo", I either got to use one of those

```
Class.new {}
```

```
class Foo
  ...
  self
end

class Foo
  ...
end
Foo
```

#25 - 12/28/2015 08:08 AM - nobu (Nobuyoshi Nakada)

- Related to Feature #11905: Change the 'class' keyword to return a symbol added

#26 - 12/28/2015 08:36 AM - yuki24 (Yuki Nishijima)

I find myself repeating writing something like below (which, of course, doesn't work at this moment):

```
private_constant class Person
  ...
end
```

Although it would need to return a symbol, it would be great if we could do the above.