Ruby trunk - Feature #7519

Module Single Inheritance

12/06/2012 12:36 AM - trans (Thomas Sawyer)

Status:	Rejected	
Priority:	Normal	
Assignee:		
Target version:	Next Major	

Description

A limitation of modules is that they can not gain and augment the qualities of another module in the same manner that a class can of another class. They can use #include, but using #include to carry the behavior of one module into another is limited in that singleton methods are not available to it and also because of the well known Module Include Problem. So it occurs to me that modules could have their own inheritance chain.

For example:

```
module M
def self.foo
"foo"
end
def bar
"bar"
end
end

module N < M
def bar
super + "!"
end
end

N.foo #=> "foo"

class C
include N
end

C.new.bar  #=> "bar!"
```

I think it easy to think about in terms of classes being types of "nouns", and modules being types of "adjectives". So just as one "noun" can inherit the behavior of another "noun", so could one "adjective" inherit the behavior of another "adjective".

History

#1 - 12/06/2012 01:38 AM - matz (Yukihiro Matsumoto)

- Status changed from Open to Rejected

I think providing new inheritance system for modules is overkill for allowing module method inheritance.
It would make the role of modules in the language unclear.

If I were you, I'd make a proposal like another version of #include, or adding optional (keyword) argument to #include.

Matz.

#2 - 12/06/2012 03:21 AM - alexeymuranov (Alexey Muranov)

Maybe a solution would be to allow a second method table in modules, so that including a module would also add singleton methods to the base? I suggested it for classes here: #7250, but it can without any change work for modules. (It is different from just inheriting singleton methods like in class inheritance.)

However i would see nothing wrong with inheriting any object from any other object:

```
x = Object.new
```

```
def x.foo
"Foo"
end

object y < x
end

y.foo # => "Foo"
```

:)

P.S. OOP is object-oriented, not class-oriented :).

#3 - 12/12/2012 01:10 PM - trans (Thomas Sawyer)

=begin

> I think providing new inheritance system for modules is overkill for allowing module method inheritance.
> It would make the role of modules in the language unclear.

It's clear to me -- to be a pain in the butt ;)

I find the whole "def self.included(base); base.extend ClassMethods; end" to be about the worst anti-pattern I have ever seen. Modules are pretty well useless and the class method thing makes it that much worse. Even when I try to use them, in the end, they almost always end up getting factored out. (I'm not talking about namespaces, of course. For that they do their job.)

I actually thought you might like this particular suggestion b/c it keeps a strong stance on Single Inheritance. I really don't think it would have any effect whatsoever on what people perceive as the role of modules. That has everything to do with the lack new and nothing else.

> If I were you, I'd make a proposal like another version of #include, or adding optional (keyword) argument to #include.

I'm not so sure that is a good idea. A module should be an encapsulation of reusable behavior. It doesn't make sense to leave that to the "consumer". It would be like asking for a way to include a module but only including the methods that start with the letter s.

It I were to suggest anything along these lines it would be that one could specify which class-methods are visible or not. e.g.

```
module M
def self.a; "a"; end

visible
def self.b; "b"; end

end

class C
include M
end

C.a #=> error
C.b #=> "b"
```

However, I have my doubts that's really the best answer either. It adds more complexity to the language. And complexity is the enemy of productivity. I'd still tend to think it would be better if all class methods were visible, b/c one can easy tuck away methods that one did not want visible in another namespace. e.g.

```
module M
module S
def self.a; "a"; end
end

def self.b; "b"; end

end

class C
include M
end

C.a #=> error
C.b #=> "b"
```

It's a trade-off, of course, but the later is so much simpler it seems hard to justify any of the former language modifications.

But that's actually OT. Whether modules can have a (single) inheritance chain like classes is a separate question. Personally I very much like the symmetry.
=end