Ruby master - Feature #14365

irreconcilable ancestor chain ordering expectations should perhaps produce an error

01/16/2018 05:34 PM - bughit (bug hit)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

```
module Module1
  def foo
    puts 'Module1#foo'
    super
  end
end

module Module2
  include Module1

  def foo
    puts 'Module2#foo'
    super
  end
end

class SuperClass1
  def foo
    puts "SuperClass1#foo"
  end
end

class SubClass1 < SuperClass1
  include Module2
  include Module1

  def foo
    puts "SubClass1#foo"
    super
  end
end

p SubClass1.ancestors

SubClass1.new.foo
```

If you look at SubClass1 in isolation, the expectation is that Module1 will be in front of Module2
If you look at Module2 in isolation, the expectation is that Module2 will be in front of Module1

these can not be reconciled, Module2 wins, but would it not be better if this were an error? Forcing you to write:

```
class SubClass1 < SuperClass1
  include Module1
  include Module2

  def foo
    puts "SubClass1#foo"
    super
  end
end
```

History

#1 - 01/16/2018 09:34 PM - shevegen (Robert A. Heiler)

> would it not be better if this were an error?

I have not yet reached the build-up chain that you described above,
but I think the biggest issue in your report so far is the focus
on an error.

Why should this be an error? The syntax is valid and ruby does not
make automatic assumptions about what a user would/should want
preferentially per se.

I assume it would better be filed under Misc or Feature since it
is more a behaviour change than a "bug".

To the issue of module versus class - it is largely a deliberate
design decision by matz. We can discuss which way may be better,
module, class, both, inheritance versus composition. Classes and
modules are very similar to one another and also do something
similar. To me the distinction is not necessarily absolutely
logical because they seem to be very much alike.

However had, I think that the main "modus operandi" lies, and
should lie, in class-based inheritance and hierarchies, even
if that is less flexible. (The "tree of life" in biology, for
example, could not too easily be built up in class-based
ruby inheritance, due to the species concept not being a
first-order citizen - see exchange of genetic material between
bacteria but also "higher" organisms such as parasites and
the host organism. The tree of life was a more useful concept
decades ago in old traditional biology than it is in modern
biology.)

It took me some time but in the code I wrote in the last 3-4
years, I usually prefer to go this way:

(**1**) Toplevel constant is a module, such as "module Foobar".

(**2**) Usually if the project is large, I have "class Base" which
is the base class for other classes in the project. I may
also have a "class Base" in a file called prototype.rb sometimes,
largely to avoid circular dependency warnings and to make the
build-up easier too (I may file a new issue about this eventually,
in regards to circular dependencies, to make ruby be less noisy
in regards to circular warning problems, and I may also suggest a
new or different way to require files in projects, in particular
large projects... see also what Hiroshi Shibata suggested for
passing a hash to, I think, require - but for the time being, I
just wanted to mention how I approach the situation).

(**3**) I may have toplevel module-methods such as
Foobar.do_something().

(**4**) Most of the other code I use and write goes into classes,
which can sometimes be called via such module-methods too,
like Foobar.create_custom_directories and so forth.

Additionally my setup is in general very simple and linear.

I almost never have complicated setups like in your case
with 4 different hierarchies.

IF your suggestion is meant to have a ruby hacker run ruby
or ruby code to give a warning or a helpful message, such
as in the spirit like the did-you-mean gem or "wrong
indentation" and such, then I have nothing against the
suggestion. That would be fine, IMO, if a ruby hacker wants
to enforce a stricter mode where ruby operates with/in.

But I would be against making it an "error" because just
as you could reason that it is an error/bug, others could
reason that it is a feature and flexibility situation.

Last code addition wins.

#2 - 08/13/2019 05:18 AM - jeremyevans0 (Jeremy Evans)

- Backport deleted (2.3: UNKNOWN, 2.4: UNKNOWN, 2.5: UNKNOWN)

- ruby -v deleted (ruby 2.5.0p0 (2017-12-25 revision 61468) [x86_64-linux])

- Tracker changed from Bug to Feature