Ruby - Feature #18832

Do not have class/module keywords consider ancestors of Object

06/15/2022 08:13 PM - fxn (Xavier Noria)

Status:	Closed	
Priority:	Normal	
Assignee:		
Target version:		

Description

The following code:

```
module M
  class C
  end
end

include M

p Object.const_defined?(:C, false)

class C < String # (1)
end
```

prints false, as expected, but then raises superclass mismatch for class C (TypeError) at (1).

I believe this is a bug, because Object itself does not have a C constant, so (1) should just work, and the superclasse of M::C should be irrelevant.

Associated revisions

Revision 12ac8971a394118a57640299f654e46e763093fa - 07/21/2022 03:28 PM - jeremyevans (Jeremy Evans)

Do not have class/module keywords look up ancestors of Object

Fixes case where Object includes a module that defines a constant,
then using class/module keyword to define the same constant on
Object itself.

Implements [Feature #18832]

Revision 12ac8971a394118a57640299f654e46e763093fa - 07/21/2022 03:28 PM - jeremyevans (Jeremy Evans)

Do not have class/module keywords look up ancestors of Object

Fixes case where Object includes a module that defines a constant,
then using class/module keyword to define the same constant on
Object itself.

Implements [Feature #18832]

Revision 12ac8971 - 07/21/2022 03:28 PM - jeremyevans (Jeremy Evans)

Do not have class/module keywords look up ancestors of Object

Fixes case where Object includes a module that defines a constant,
then using class/module keyword to define the same constant on
Object itself.

Implements [Feature #18832]

History

#1 - 06/15/2022 08:23 PM - jeremyevans0 (Jeremy Evans)

I don't think this is a bug. p Object.const_defined?(:C, false) explicitly says you don't want to look into ancestors. However class C at top level will look into ancestors of Object (even class Object::C will). I don't believe Ruby has syntax for a class opening that doesn't look into ancestors. You would have to do something like: Object::C = Class.new(String). Changing class C < String to ignore Object::C (i.e. M::C) is likely to break backwards compatibility significantly.

#2 - 06/15/2022 08:47 PM - fxn (Xavier Noria)

There is something weird here somewhere.

In general, Ruby does not check the ancestors as far as I can tell. Take the same situation with non-empty nesting:

```
module M
  class C
  end
end

module N
  include M

  class C < String
  end
end
```

That one does not raise, and M::C and N::C are different class objects.

This property is what allows you to be anywhere defining a class or module and be sure you are creating regardless of what the ancestors have (think about a chain you don't control like inheriting from ActiveRecord::Base.

As you know, the class and module keyword have their own constant lookup (to decide whether to create or reopen), and they look only in the first element of the nesting without checking its ancestor chain.

Why isn't the top-level acting the same way?

#3 - 06/15/2022 09:01 PM - fxn (Xavier Noria)

Even more, if you set an autoload for C in Object, the autoload is triggered. Which is also consistent with "I have not found the constant in my lookup, but let's trigger the autoload to get it defined hopefully". This is also consistent. (And the autoload raises because it hits what the original example shows.)

To me, this smells like a bug related to some quirk of the top-level object that is leaking.

#4 - 06/15/2022 09:08 PM - jeremyevans0 (Jeremy Evans)

fxn (Xavier Noria) wrote in #note-3:

> Even more, if you set an autoload for C in Object, the autoload is triggered. Which is also consistent with "I have not found the constant in my lookup, but let's trigger the autoload to get it defined hopefully". This is also consistent. (And the autoload raises because it hits what the original example shows.)
>
> To me, this smells like a bug related to some quirk of the top-level object that is leaking.

I agree that it would probably be best for things to be consistent. Note that this is not specifically related to top-level, but behavior of Object specifically, since it happens not just at top-level:

```
module M
  class C
  end
end

include M

p Object.const_defined?(:C, false)

class Object
  class C < String # still superclass mismatch
  end
end
```

Object is treated specially during the lookup, that's probably where the difference comes from.

#5 - 06/15/2022 09:49 PM - fxn (Xavier Noria)

You're right, it happens too if the first element of the nesting is Object.

For context, this happened in a Rails application that had app/models/comment.rb, which worked regularly fine. However, in a .rake file the project had include REXML at the top-level and in Rake tasks the application could no longer define the Comment model due to the superclass mismatch. REXML documents such top-level include. This is debatable, but cannot distract from discussing the consistency of the pure lookup logic.

If Object does not have Comment, I expect class Comment; end to define a new class, as in any other "namespace".

#6 - 06/16/2022 05:13 PM - Eregon (Benoit Daloze)

I agree we shouldn't look in modules prepended/included in Object, only look on the lexical parent itself (not its ancestors).
This is one of these inconsistent "deep constant lookup" exceptions (the other I remember is "deepMethodSearch" used for alias and public/protected/private), I think we should fix it.

@fxn I think it'd be useful if you show a Ruby code snippet of the real-world problem to motivate the change.

FWIW, the logic in TruffleRuby to deal with that special case:
https://github.com/oracle/truffleruby/blob/c99ec964fb51f8364c6919467e898db6969f4058/src/main/java/org/truffleruby/language/objects/LookupForExistingModuleNode.java#L63
https://github.com/oracle/truffleruby/blob/c99ec964fb51f8364c6919467e898db6969f4058/src/main/java/org/truffleruby/core/module/ModuleOperations.java#L202

#7 - 06/17/2022 08:58 AM - fxn (Xavier Noria)

@Eregon (Benoit Daloze) sure. The real-world problem that originated this ticket can be found here. It is the one I briefly described above.

#8 - 06/17/2022 09:49 AM - Eregon (Benoit Daloze)

@fxn Yes, but that is too long. What I think is useful for the dev meeting is a snippet of a few lines, which represents and reproduces the real issue (and Ruby code is often much clearer than a textual description of it).
e.g. I find all the examples with M/C above too synthetic and hard to judge if it matters.

#9 - 06/17/2022 10:00 AM - fxn (Xavier Noria)

@Eregon (Benoit Daloze) Is this one better suited?

```
require "active_record"
require "rexml"

include REXML

class Comment < ActiveRecord::Base # superclass mismatch for class Comment (TypeError)
end
```

That is the gist of what happens in the ticket.

#10 - 06/17/2022 10:01 AM - fxn (Xavier Noria)

It is super important to highlight that the problem is not the behavior isolated, but that this is unexpected because it does not work like this for any other cref.

#11 - 06/17/2022 10:08 AM - fxn (Xavier Noria)

A good exercise is to try to document this (all lookups should be documented). Something like:

> In the class/module keywords, the constant is checked in the cref. If present, the object is retrieved. If it is a class/module respectively, reopened, otherwise, error condition. If not present, a corresponding object is created and assigned to the constant in the cref. In the case of Object, ancestors are checked also and the object reopened if found because... WHY?

Does the team believe this documentation makes sense?

#12 - 06/17/2022 10:45 AM - Eregon (Benoit Daloze)

Yes, I think that's a great example why we should fix this.

Anyone should still be able to define a Comment (defined as a constant of Object) class with just class Comment at the top-level, even if with REXML::Comment + include REXML.

BTW, I thought class Object::Comment < String; end might be a workaround but it does not (as Jeremy found out as well above):

```
$ ruby -v -rexml -e 'include REXML; class Object::Comment < String; end'
ruby 3.0.3p157 (2021-11-24 revision 3fb7d2cadc) [x86_64-linux]
-e:1:in `<main>': superclass mismatch for class Comment (TypeError)
```

This is also very surprising (the intention is to define Object::Comment with a superclass of Object):

```
$ ruby -v -rexml -e 'include REXML; class Comment; end; p Comment.ancestors'
ruby 3.0.3p157 (2021-11-24 revision 3fb7d2cadc) [x86_64-linux]
[REXML::Comment, Comparable, REXML::Child, REXML::Node, Object, REXML, PP::ObjectMixin, Kernel, BasicObject]
```

#13 - 06/20/2022 11:59 PM - jeremyevans0 (Jeremy Evans)

- Tracker changed from Bug to Feature

- Subject changed from Suspicious superclass mismatch to Do not have class/module keywords consider ancestors of Object

- Backport deleted (2.7: UNKNOWN, 3.0: UNKNOWN, 3.1: UNKNOWN)

As I initially expected, this is not a bug. Having class/module keywords consider ancestors of Object is by design, with explicit code for just this purpose, and there is both a test and spec for the current behavior. Switching to feature request.

That being said, I do think it would be best to change the behavior, so I submitted a pull request to do so: https://github.com/ruby/ruby/pull/6048 . I'll add this as a topic for the next developer meeting.

#14 - 06/21/2022 05:50 AM - fxn (Xavier Noria)

Awesome, thanks Jeremy.

#15 - 07/21/2022 11:49 AM - mame (Yusuke Endoh)

We discussed this ticket at the dev meeting.

@matz (Yukihiro Matsumoto) said he wanted to try the change, i.e., include M; class C; end at the toplevel should define ::C instead of reopening M::C. He may revisit the change if a compatibility issue is found.

#16 - 07/21/2022 03:28 PM - jeremyevans (Jeremy Evans)

- Status changed from Open to Closed

Applied in changeset git|12ac8971a394118a57640299f654e46e763093fa.

Do not have class/module keywords look up ancestors of Object

Fixes case where Object includes a module that defines a constant,
then using class/module keyword to define the same constant on
Object itself.

Implements [Feature #18832]