Prohibit to pass a block singleton class

01/30/2019 07:19 AM - ko1 (Koichi Sasada)

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

Description
The following code works now:

```ruby
def foo
  class << Object.new
    yield
  end
end
```
```ruby
foo{ p :ok } #=> :ok
```

but I think this feature is very strange because local variables are not active in singleton class.

How about to prohibit this feature?

plan: warning at ruby 2.7 and prohibit it in ruby 3.

Associated revisions

Revision 0fc597f2 - 02/04/2019 07:10 AM - ko1 (Koichi Sasada)
check and show a warning for incorrect yield.

- compile.c (check_yield_place): this function check the yield location.
  - show a warning if yield in class syntax. [Feature #15575]
  - do strict check for toplevel yield. Without this patch, 1.times{ yield } in toplevel is valid-syntax (raise LocalJumpError at runtime) although toplevel simple yield is not valid syntax. This patch make them syntax error.

Revision 66999 - 02/04/2019 07:10 AM - ko1 (Koichi Sasada)
cHECK and show a warning for incorrect yield.

- compile.c (check_yield_place): this function check the yield location.
  - show a warning if yield in class syntax. [Feature #15575]
  - do strict check for toplevel yield. Without this patch, 1.times{ yield } in toplevel is valid-syntax (raise LocalJumpError at runtime) although toplevel simple yield is not valid syntax. This patch make them syntax error.

Revision df2b8c05 - 02/04/2019 08:13 AM - nobu (Nobuyoshi Nakada)
Show proper location for warning [Feature #15575]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@67001 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 67001 - 02/04/2019 08:13 AM - nobu (Nobuyoshi Nakada)
Show proper location for warning [Feature #15575]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@67001 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 81fb05db - 02/08/2019 02:54 AM - nobu (Nobuyoshi Nakada)
Suppress warning [Feature #15575]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@67034 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 67034 - 02/08/2019 02:54 AM - nobu (Nobuyoshi Nakada)
Suppress warning [Feature #15575]
Revision ee302f54 - 10/21/2019 04:06 PM - ko1 (Koichi Sasada)

add an NEWS entry about [Feature #15575]

History

#1 - 01/30/2019 12:17 PM - Eregon (Benoit Daloze)
Agreed.
IMHO class << expr should be no different than the normal class Name, but currently various things are allowed in that context such as return, yield, etc.

Constant resolution also becomes somewhat dynamically-scoped with class << expr which is also a weird exception.
So longer term maybe it would be good to deprecate class << entirely and use singleton_class.class_exec do instead.

#2 - 01/30/2019 12:24 PM - ko1 (Koichi Sasada)
Eregon (Benoit Daloze) wrote:

    Agreed.
    IMHO class << expr should be no different than the normal class Name, but currently various things are allowed in that context such as return, yield, etc.
    Constant resolution also becomes somewhat dynamically-scoped with class << expr which is also a weird exception.
    So longer term maybe it would be good to deprecate class << entirely and use singleton_class.class_exec do instead.

or make it syntax sugre of singleton_class.class_exec do?
Both are easy to explain the rule. Now, it is somewhat unclear.

#3 - 01/30/2019 12:27 PM - ko1 (Koichi Sasada)
Similar strange example (off-topic):

```ruby
1.times{
  class C
  break
  def undefined_method; end
  end
  p :unrechable
}
p [:ok, C.instance_methods(false)] #=> [:ok, []]
```

We can break from class syntax.
I hope nobody use it :(

#4 - 01/30/2019 12:28 PM - Eregon (Benoit Daloze)
ko1 (Koichi Sasada) wrote:

    or make it syntax sugre of singleton_class.class_exec do?

I think that would be confusing as class (like module and def) is a keyword which normally adds a new lexical scope and does not capture the parent local variables.

#5 - 01/30/2019 12:30 PM - Eregon (Benoit Daloze)
ko1 (Koichi Sasada) wrote:

    We can break from class syntax.
    I hope nobody use it :(

At least TruffleRuby doesn't implement it, and we had no bug report about this, so hopefully not used.
Agreed it should be deprecated/removed as it's confusing at best.

#6 - 01/31/2019 04:36 AM - alanwu (Alan Wu)
Another option would be to make everything use lexical scope, if the strangeness is indeed because locals are inaccessible inside class << self.
So imagine the following printing 1 2 3.

```ruby
foo = 1
class A
  bar = 2
  def hi
```
baz = 3
class << self
  p foo, bar, baz
end
end
A.new hi

Is it less strange if everything used the same scoping rule?

#7 - 01/31/2019 08:02 AM - matz (Yukihiro Matsumoto)
Agreed to prohibit. Disagree to change the scoping rule (alanwu (Alan Wu)).

Matz.

#8 - 02/04/2019 06:43 AM - ko1 (Koichi Sasada)
Matz, Thank you for confirmation.

I add a warning (without -w) like that:

```ruby
def foo
  class << Object.new
    yield
  end
end
foo{ p :ok } #=> :ok
```

test.rb: warning: `yield' in class syntax will not be supported from Ruby 3.0. [Feature #15575]

English correction is welcome :)

#9 - 02/04/2019 07:10 AM - ko1 (Koichi Sasada)
- Status changed from Open to Closed

Applied in changeset trunk|r66999.

check and show a warning for incorrect yield.

- compile.c (check_yield_place): this function check the yield location.
  - show a warning if yield in class syntax. [Feature #15575]
  - do strict check for toplevel yield. Without this patch, 1.times{ yield } in toplevel is valid-syntax (raise LocalJumpError at runtime) although toplevel simple yield is not valid syntax. This patch make them syntax error.