Ruby master - Bug #13446
refinements with prepend for module has strange behavior

04/18/2017 01:17 AM - mtsfmfm (Fumiaki Matsushima)

| Status:          | Closed                                      |
| Priority:        | Normal                                      |
| Assignee:        | nobu (Nobuyoshi Nakada)                    |
| Target version:  | ruby 2.4.1p111 (2017-03-22 revision 58053) [x86_64-linux] |
| Backport:        | 2.2: UNKNOWN, 2.3: UNKNOWN, 2.4: UNKNOWN    |

Description
using Module.new {
  refine Enumerable do
    alias :orig_sum :sum
  end
}
module Enumerable
  def sum(*args)
    orig_sum(*args)
  end
end
class GenericEnumerable
  include Enumerable
  def each
  end
end

# GenericEnumerable.new.sum # if we uncomment this line, 'GenericEnumerable#sum' will work
Enumerable.prepend(Module.new) # if we comment out this line, 'GenericEnumerable#sum' will work
p GenericEnumerable.new.sum
# undefined method 'orig_sum' for #<GenericEnumerable:0x0000000127c120 @values=[1, 2, 3]> (NoMethodError)

Is this intentional?

Related issues:
Related to Ruby master - Bug #16242: Refinements method call to failed Closed

Associated revisions
Revision a0579f36 - 11/28/2019 10:57 AM - jeremyevans (Jeremy Evans)
Make prepending a refined module after inclusion not break refinements

After the previous commit, this was still broken. The reason it was broken is that a refined module that hasn’t been prepended to yet keeps the refined methods in the module's method table. When prepending, the module’s method table is moved to the origin iclass, and then the refined methods are moved from the method table to a new method table in the module itself.

Unfortunately, that means that if a class has included the module, prepending breaks the refinements, because when the methods are moved from the origin iclass method table to the module method table, they are removed from the method table from the iclass created when the module was included earlier.

Fix this by always creating an origin class when including a module that has any refinements, even if the refinements are not currently used. I wasn't sure the best way to do that.
The approach I choose was to use an object flag. The flag is
Fixes [Bug #13446]

History

#1 - 06/16/2017 02:28 AM - nobu (Nobuyoshi Nakada)
- Assignee set to nobu (Nobuyoshi Nakada)
- Status changed from Open to Assigned
- Description updated

#2 - 10/08/2019 11:20 PM - wanabe (_ wanabe)
- Related to Bug #16242: Refinements method call to failed added

#3 - 10/12/2019 11:51 PM - jeremyevans0 (Jeremy Evans)
Fixing this first requires fixing #16242, which allows including a module that uses prepend and is refined. However, the fix for #16242 does not fix the example in this case, as this prepends the included module after the module is included.

The reason this is still broken after the fix for #16242 is that a refined module that hasn't been prepended to yet keeps the refined methods in the module's method table. When prepending, the module's method table is moved to the origin iclass, and then the refined methods are moved from the method table to a new method table in the module itself.

Unfortunately, that means that if a class has included the module, prepending breaks the refinements, because when the methods are moved from the origin iclass method table to the module method table, they are removed from the method table from the iclass created when the module was included earlier.

Fix this by always creating an origin iclass when including a module that has any refinements, even if the refinements are not currently used (see https://github.com/ruby/ruby/pull/2550). I wasn't sure the best way to do that. The approach I choose was to use an object flag. The flag is set on the module when Module#refine is called, and if the flag is present when the module is included in another module or class, an origin iclass is created for the module.

#4 - 11/28/2019 10:57 AM - jeremyevans (Jeremy Evans)
- Status changed from Assigned to Closed

Applied in changeset git|a0579f3606561a74e323f6193b9504c06845236c.

Make prepending a refined module after inclusion not break refinements

After the previous commit, this was still broken. The reason it was broken is that a refined module that hasn't been prepended to yet keeps the refined methods in the module's method table. When prepending, the module's method table is moved to the origin iclass, and then the refined methods are moved from the method table to a new method table in the module itself.

Unfortunately, that means that if a class has included the module, prepending breaks the refinements, because when the methods are moved from the origin iclass method table to the module method table, they are removed from the method table from the iclass created when the module was included earlier.

Fix this by always creating an origin iclass when including a module that has any refinements, even if the refinements are not currently used. I wasn't sure the best way to do that.

The approach I choose was to use an object flag. The flag is set on the module when Module#refine is called, and if the flag is present when the module is included in another module or class, an origin iclass is created for the module.

Fixes [Bug #13446]