Ruby master - Bug #14744

Refinements modules have a superclass

05/08/2018 02:26 PM - Eregon (Benoit Daloze)

Status:	Open		
Priority:	Normal		
Assignee:	matz (Yukihiro Matsumoto)		
Target version:			
ruby -v:	ruby 2.5.1p57 (2018-03-29 revision 63029) [x86_64-linux]	**Backport:**	2.3: UNKNOWN, 2.4: UNKNOWN, 2.5: UNKNOWN

Description

```
$ ruby -e 'module M; refine Array do; p self; p self.class; p ancestors; end; end'
#<refinement:Array@M>
Module
[#<refinement:Array@M>, Array, Enumerable, Object, Kernel, BasicObject]
```

So the refinement module (self in the refine Array do block) is a Module, but looking at its ancestors it has Array as a "superclass".

Is that expected?
I thought modules can never have a superclass in Ruby.

History

#1 - 05/08/2018 08:42 PM - shevegen (Robert A. Heiler)

> I thought modules can never have a superclass in Ruby.

This may be a bug, I think; other modules do not seem to have anything but themselves if you call .ancestors() on them either.

I am not absolutely certain, but perhaps either way it could be mentioned in the documentation too, whether modules (including refinements) can lead to modules having superclasses.

#2 - 05/09/2018 03:57 AM - shugo (Shugo Maeda)

- Assignee set to matz (Yukihiro Matsumoto)

> So the refinement module (self in the refine Array do block) is a Module, but looking at its ancestors it has Array as a "superclass".
>
> Is that expected?

The superclass of a refinement module is for implementing super in refined methods.
However, it may be better to hidden from reflection APIs.

What do you think, Matz?

#3 - 05/09/2018 09:09 AM - Eregon (Benoit Daloze)

shugo (Shugo Maeda) wrote:

> The superclass of a refinement module is for implementing super in refined methods.
> However, it may be better to hidden from reflection APIs.

I see, thank you for the explanation.

I'm not sure it's good to hide though, as I noticed this interesting case by seeing code like:

```
module R
  refine Array do
```

```
    p instance_method(:sum)
  end
end
```

I think either super should be implemented differently to not rely on setting a Module's superclass,
or we leave it as it is, and have it visible with reflection APIs since anyway it behaves like there is a "superclass".

FWIW currently in TruffleRuby we do a slightly different lookup when looking for the super method of a refined method, and look at the active refinements to find the super method,
and then use an inline cache to avoid repeated lookups.