`Prepend` should prepend a module before the class

I see

```ruby
module M; end
module A; end
class B; include A; end
A.prepend M
B.prepend M
p B.ancestors
```
gives [B, M, A, Object, Kernel, BasicObject] now. It used to be [M, B, A, Object, Kernel, BasicObject].

I think it should be prepended to class B. Probably it should be [M, B, M, A, Object, Kernel, BasicObject].

The reason behind this change may be duplication removing, but considering the following code, it is OK to duplicate prepend modules.

```ruby
module M; end
class A; end
class B<A; end
A.prepend M
B.prepend M
p B.ancestors #=> [M, B, M, A, Object, Kernel, BasicObject]
```

Am I missing something?

Matz.

Related issues:
Related to Ruby master - Bug #7844: include/prepend satisfiable module dependencies are not satisfied... Closed

Associated revisions
Revision e0909454 - 01/15/2021 04:43 AM - jeremyevans (Jeremy Evans)
Make Module#prepend affect ancestor chain even if argument already included in receiver

Previously, if a class included a module and then prepended the same module, the prepend had no effect. This changes the behavior so that the prepend has an effect unless the module is already prepended the receiver.

While here, rename the origin_seen variable in include_modules_at, since it is misleading. The variable tracks whether c has been seen, not whether the origin of klass has been.

Fixes [Bug #17423]

History
#1 - 12/22/2020 01:58 AM - matz (Yukihiro Matsumoto)
- Related to Bug #7844: include/prepend satisfiable module dependencies are not satisfied added
This stems from the change that Module#prepend (and Module#include) now affect classes already including the receiver. You get the same behavior for Ruby 2.0-3.0 if you prepend M to A before including A in B:

```ruby
module M
end
module A
end
A.prepend M
class B; include A; end
B.prepend M
p B.ancestors
```

The difference for your original code is shown by B.ancestors before and after the prepend of M:

```ruby
module M
end
module A
end
class B;
include A;
end
A.send :prepend, M
p B.ancestors
B.send :prepend, M
p B.ancestors
```

Output:

```
$ ruby30 t1.rb
[B, M, A, Object, Kernel, BasicObject]
[B, M, A, Object, Kernel, BasicObject]
```

```
$ ruby27 t1.rb
[B, A, Object, Kernel, BasicObject]
[M, B, A, Object, Kernel, BasicObject]
```

Ruby has never had include add a module to the ancestor chain if it already exists in the ancestor chain of the receiver, and has never had prepend add a module to the ancestor chain if it already exists in the ancestor chain before the superclass. Prior to Ruby 2.3, Ruby wouldn't prepend the module if it was anywhere in the ancestor chain.

I'm not opposed to allow prepend to ignore the existing ancestor chain and always prepend the module to it. However, it seems to be a very risky change to make at the current time. Maybe we can try after the release of 3.0?

---

This is a serious breakage, in fact, breaks Rails. So I believe we need to change this soon. But:

- We need more time to investigate
- We don't have much time before the release
- Rails made a patch to address this issue

Thus fix this after the 3.0 release.

Matz.

---

Based on the decision made at the developer meeting (https://github.com/ruby/dev-meeting-log/blob/master/DevelopersMeeting20210113Japan.md#bug-17423-prepend-should-prepend-a-module-before-the-class-jeremyevans0), I've added a pull request that will make Module#prepend insert a module in the ancestor chain even if the receiver already includes the module: https://github.com/ruby/ruby/pull/4072

---

Applied in changeset git:09094546a19d6b62b3e21d0b061b103cf21f760.

Make Module#prepend affect ancestor chain even if argument already included in receiver

Previously, if a class included a module and then prepended the same module, the prepend had no effect. This changes the behavior so that the prepend has an effect unless the module is already prepended the receiver.

While here, rename the origin_seen variable in include_modules_at, since it is misleading. The variable tracks whether c has been seen, not whether the origin of kclass has been.
Fixes [Bug #17423]