I have the following code:

```ruby
module M1
  CONST = 'CONST'.freeze
end

module M2
  def self.included(base)
    base.include M1
    base.include InstanceMethods
  end

  module InstanceMethods
    def const?(value)
      M1::CONST == value
    end
  end
end

class C
  include M2
end

c = C.new

C.new.const?('CONST') # => true
```

M2 module "inherits" M1 module using included hook. This code works, but I prefer making constants private. However, when I add `private_constant :CONST` to M1 then the code doesn't work anymore since we cannot refer M1::CONST.

### Workaround

One way to solve this is to define method that just returns the private constant.

```ruby
# Refined version of M1 with `const` method
module M1
  CONST = 'CONST'.freeze

  private
  private_constant :CONST

  def const
    CONST
  end
end

# Refined version of M2, using `const` method inherited from M1
module M2
  def self.included(base)
    base.include M1
    base.include InstanceMethods
  end

  module InstanceMethods
    def const
      M1::CONST
    end
  end
end
```
This works anyway, but defining private method for every private constant could be cumbersome.

**Solution**

So my suggestion here is to add with_private_method option to private_constant method. The auto-generated private method has the same name as the given constant and just returns it. So we can do the following:

```ruby
module M1
  CONST = 'CONST'.freeze
  private_constant :CONST, with_private_method: true
end
```

Now we can access CONST with const method. This makes it possible to refer private constants through private methods more easily, that helps some code like my example.

**History**

#1 - 07/29/2022 11:26 AM - mame (Yusuke Endoh)

How about calling include M1 in M2::InstanceMethods?

```ruby
module M1
  CONST = 'CONST'.freeze
  private_constant :CONST
end

module M2
  def self.included(base)
    base.include InstanceMethods
  end

  module InstanceMethods
    include M1
    def const?(value)
      CONST == value
    end
  end
end

class C
  include M2
end
C.new.const?('CONST') # => true
```

#2 - 07/29/2022 02:41 PM - okuramasafumi (Masafumi OKURA)

In a real world code I'm facing, M1 contains lots of other methods so including it in InstanceMethods causes confusion.

I pushed my code on GitHub.

https://github.com/okuramasafumi/alba/blob/jsonapi/lib/alba/resource.rb#L299-L301
https://github.com/okuramasafumi/alba/blob/jsonapi/lib/alba/isonapi.rb#L30-L43

#3 - 07/29/2022 02:49 PM - jeremyevans0 (Jeremy Evans)

okuramasafumi (Masafumi OKURA) wrote in #note-2:

In a real world code I'm facing, M1 contains lots of other methods so including it in InstanceMethods causes confusion.

You should be able to work around this by making sure the M1 namespace surrounds the method in question, so unqualified constant lookup works:

```ruby
module M1
  CONST = 'CONST'.freeze
  private_constant :CONST
end
```
module M2
  def self.included(base)
    base.include M1
    base.include InstanceMethods
  end

module ::M1
  module ::M2::InstanceMethods
    def const?(value)
      CONST == value
    end
  end
end
end

class C
  include M2
end

C.new.const?('CONST') # => true

I don't think this need is common enough to support in private_constant. If you really need a private method defined that returns the constant value, you can always define it manually.