This ticket is to propose to allow the private method to be called if its receiver is written as self.

The following Ruby program is to explain my idea.

class A
  private
def f
end
end

A.new.instance_eval do
  f()  # Okay, without receiver
  self.f  # Currently NoMethodError, but should be okay in my opinion
  self.itself.f  # NoMethodError anyway; the receiver is not written as self
end

This change will allow to call private accessor method like self.title=.

It also will make refactoring to make a public method private easier. Currently, such kind of refactoring may require to rename duplicated local variables or add () to method calls.

Related issues:
Has duplicate Ruby master - Feature #16123: Allow calling a private method wi...  Closed

Associated revisions
Revision 7fbd2f7c - 09/19/2019 05:20 PM - dylants (Dylan Thacker-Smith)
Allow calling a private method with self.

This makes it consistent with calling private attribute assignment methods, which currently is allowed (e.g. self.value =).

Calling a private method in this way can be useful when trying to assign the return value to a local variable with the same name.

[Feature #11297] [Feature #16123]

Revision d583df52 - 09/19/2019 05:21 PM - nobu (Nobuyoshi Nakada)
Added version guard

[Feature #11297] [Feature #16123]

Revision e6378cdc - 09/19/2019 05:21 PM - nobu (Nobuyoshi Nakada)
Allow calling a private accessor with self.

[Feature #11297] [Feature #16123]

Revision b80df6e8 - 09/19/2019 05:40 PM - nobu (Nobuyoshi Nakada)
Update NEWS and documents [ci skip]

[Feature #11297] [Feature #16123]
It changes the concept of private methods a little. It's OK to merge the patch if the document is updated at the same time.

Matz.

This gets rid of the current exception for private writer methods (self.foo = bar).

It also resolves the problems with private operator methods (self + bar) and compound assignments with private writers and/or private operators (self += bar, self.foo += bar, where either + or foo= or both are private). It removes pretty much all edge cases in one blow.

See also #9907 which would be simplified by this proposal. In particular, implementing the simple rule would make Charles Oliver Nutter's confusion go away (#9907-6, #9907-8), be consistent with Nobuyoshi Nakada's expectations (#9907-7) and alleviate Benoit Daloze's concerns about being decidable statically at parse time (#9907-9).

In fact, I believe that with this feature all of these should work:

```ruby
#!/usr/bin/env ruby
class Private
def doit
  self.foo = self
  self.foo ***= self
  self.foo *= self
  self.foo /= self
  self.foo %= self
  self.foo %= self
  self.foo <<= self
  self.foo >>= self
  self.foo &= self
  self.foo |= self
  self.foo ^= self
  self.foo &&= self
  self.foo ||= self
  self
  -self
  +self
  self ** self
  -self
  self * self
  self / self
  self % self
  self + self
  self - self
  self << self
  self >> self
  self & self
  self | self
  self ^ self
  self < self
  self <= self
  self > self
  self >= self
end
```

05/17/2020
self === self
self != self
self -= self
self <<= self
self[self, self] = self, self
(self, self)

end

private

attr_accessor :foo

def !(*args) p __method__, *args end
def -(*args) p __method__, *args end
def +=(*args) p __method__, *args end
def **(*args) p __method__, *args end
def -=(*args) p __method__, *args end
def *(*args) p __method__, *args end
def /(*args) p __method__, *args end
def %(*args) p __method__, *args end

def +(*args) p __method__, *args end
def -(*args) p __method__, *args end
def <<(*args) p __method__, *args end
def >>(*args) p __method__, *args end
def &(*args) p __method__, *args end
def |(*args) p __method__, *args end

def ^(*args) p __method__, *args end

def <=(*args) p __method__, *args end

def >=(*args) p __method__, *args end

def ==(args) p __method__, *args end

def ===(args) p __method__, *args end

def !=(*args) p __method__, *args end

def -=(*args) p __method__, *args end

def <=>(*args) p __method__, *args end

def [](*args) p __method__, *args end

def call(*args) p __method__, *args end
end

Private.new.doit

#4 - 09/19/2019 08:07 AM - matz (Yukihiro Matsumoto)
- Related to Feature #16123: Allow calling a private method with `self.` added

#5 - 09/19/2019 02:05 PM - nobu (Nobuyoshi Nakada)
- Related to deleted (Feature #16123: Allow calling a private method with `self.`)

#6 - 09/19/2019 02:06 PM - nobu (Nobuyoshi Nakada)
- Has duplicate Feature #16123: Allow calling a private method with `self.` added

#7 - 09/19/2019 06:25 PM - dylants (Dylan Thacker-Smith)
- Status changed from Open to Closed

Applied in changeset git|7fbd2f7cc247ee66e877ab3c88f0274834c6b6c7.

Allow calling a private method with self.

This makes it consistent with calling private attribute assignment methods, which currently is allowed (e.g. self.value =).

Calling a private method in this way can be useful when trying to assign the return value to a local variable with the same name.

[Feature #11297] [Feature #16123]