Class
Method < Object

Method objects are created by Object#method. They are associated with a particular object (not just with a class). They may be used to invoke the method within the object and as a block associated with an iterator. They may also be unbound from one object (creating an UnboundMethod) and bound to another.

```
def square(n)
  n*n
end
meth  = self.method(:square)

meth.call(9)                 # =>  81
[ 1, 2, 3 ].collect(&meth)   # =>  [1, 4, 9]
```

Instance methods

[ ]

meth[⟨ *args* ⟩*] → *object*

Synonym for Method.call.

==

meth== *other* → true or false

Returns true if *meth* is the same method as *other*.

```
def fred()
  puts "Hello"
end

alias bert fred   # =>  nil

m1 = method(:fred)
m2 = method(:bert)
m1 == m2          # =>  true
```

arity

meth.arity → *fixnum*

Returns an indication of the number of arguments accepted by a method. See Figure 27.2 on the next page. See also Method#parameters.

call

meth.call(⟨ *args* ⟩*) → *object*

Invokes the *meth* with the specified arguments, returning the method's return value.

```
m = 12.method("+")
m.call(3)    # =>  15
m.call(20)   # =>  32
```

eql?

meth.eql?(*other*) → true or false

Returns true if *meth* is the same method as *other*.

Figure 27.2. Method#arity in Action

Method#arity returns a non-negative integer for methods that take a fixed number of arguments. For Ruby methods that take a variable number of arguments, returns $-n-1$, where n is the number of required arguments. For methods written in C, returns -1 if the call takes a variable number of arguments.

```
class C
  def one;    end
  def two(a); end
  def three(*a);  end
  def four(a, b); end
  def five(a, b, *c);    end
  def six(a, b, *c, &d); end
end
c = C.new
c.method(:one).arity     # =>  0
c.method(:two).arity     # =>  1
c.method(:three).arity   # =>  -1
c.method(:four).arity    # =>  2
c.method(:five).arity    # =>  -3
c.method(:six).arity     # =>  -3

"cat".method(:size).arity      # =>  0
"cat".method(:replace).arity   # =>  1
"cat".method(:squeeze).arity   # =>  -1
"cat".method(:count).arity     # =>  -1
```

```
def fred()
  puts "Hello"
end

alias bert fred   # =>  nil

m1 = method(:fred)
m2 = method(:bert)
m1.eql?(m2)       # =>  true
```

name *meth*.name → *string*

1.9

Returns the name of the method *meth*.

```
method = "cat".method(:upcase)
method.name   # =>  :upcase
```

owner *meth*.owner → *module*

1.9

Returns the class or module in which *meth* is defined.

```
method = "cat".method(:upcase)
method.owner   # =>  String
```

receiver

meth.receiver → *obj*

1.9

Returns the object on which *meth* is defined.

```
method = "cat".method(:upcase)
method.receiver   # =>   "cat"
```

source_location

meth.source_location → [*filename*, *lineno*] or nil

1.9

Returns the source filename and line number where *meth* was defined or nil if self was not defined in Ruby source.

```
internal_method = "cat".method(:upcase)
internal_method.source_location   # =>   nil

require 'set'
set = Set.new
ruby_method = set.method(:clear)
ruby_method.source_location       # =>   ["/usr/lib/ruby/1.9.1/set.rb",
                                        114]
```

to_proc

meth.to_proc → *prc*

Returns a Proc object corresponding to this method. Because to_proc is called by the interpreter when passing block arguments, method objects may be used following an ampersand to pass a block to another method call. See the example at the start of this section.

unbind

meth.unbind → unbound_method

Dissociates *meth* from its current receiver. The resulting UnboundMethod can subsequently be bound to a new object of the same class (see UnboundMethod on page 724).