

Eigen What Now?

EigenClasses and Metaprogramming in Ruby

Eigenclasses

... Why do we care?

Include some methods in an obj? # 1

```
# attempt 1
module MyModule
  def my_method
    puts :in_my_method
  end
end

class << self
  ...

  self.class_eval do
    include MyModule
  end
end
```

Include some methods in an obj? # 2

```
# attempt 2
module MyModule
  def my_method
    puts :in_my_method
  end
end

class << self
  ...

  self.instance_eval do
    include MyModule
  end
end
```

Include some methods in an obj? # 3

```
# attempt 3
module MyModule
  def my_method
    puts :in_my_method
  end
end

class << self
  ...

  self.instance_eval do
    extend MyModule
  end
end
```

Include some methods in an obj? # 4

```
# attempt 4
module MyModule
  def self.my_method
    puts :in_my_method
  end
end

class << self
  ...

  self.class_eval do
    extend MyModule
  end
end
```



What was I not getting?

`class_eval` and
`instance_eval`:

- Eigenclasses
- Class Scope
- What is `self`

`class << self:`

- Eigenclasses

`include` and `extend`:

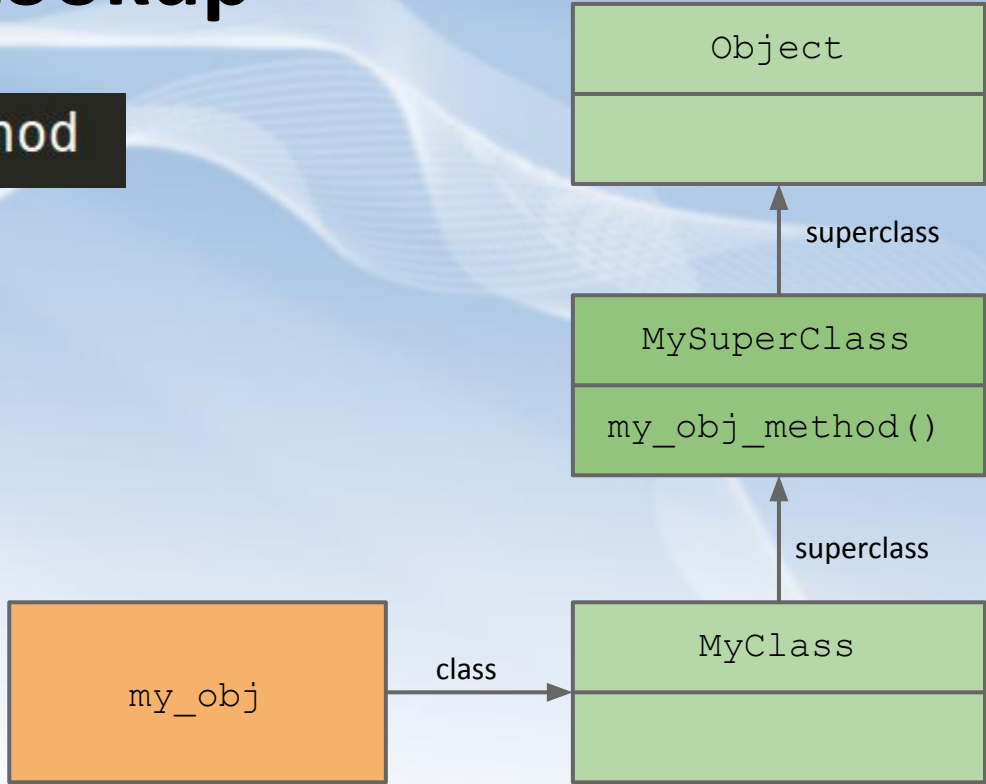
- Eigenclasses

Eigenclasses

```
class MyClass
  class << self
    def my_method
      puts :my_method
    end
  end
end
```

Ruby Method Lookup

```
my_obj.my_obj_method
```



Singleton Methods

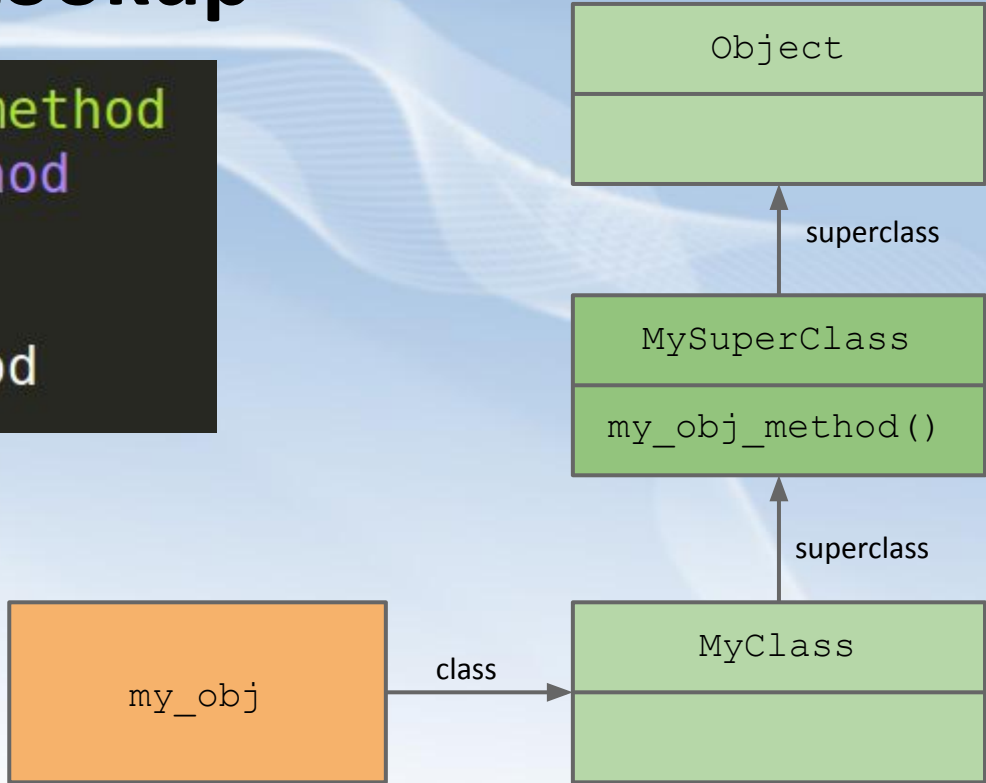
```
def my_obj.my_obj_method  
  puts :my_obj_method  
end
```

```
my_obj.my_obj_method
```

Ruby Method Lookup

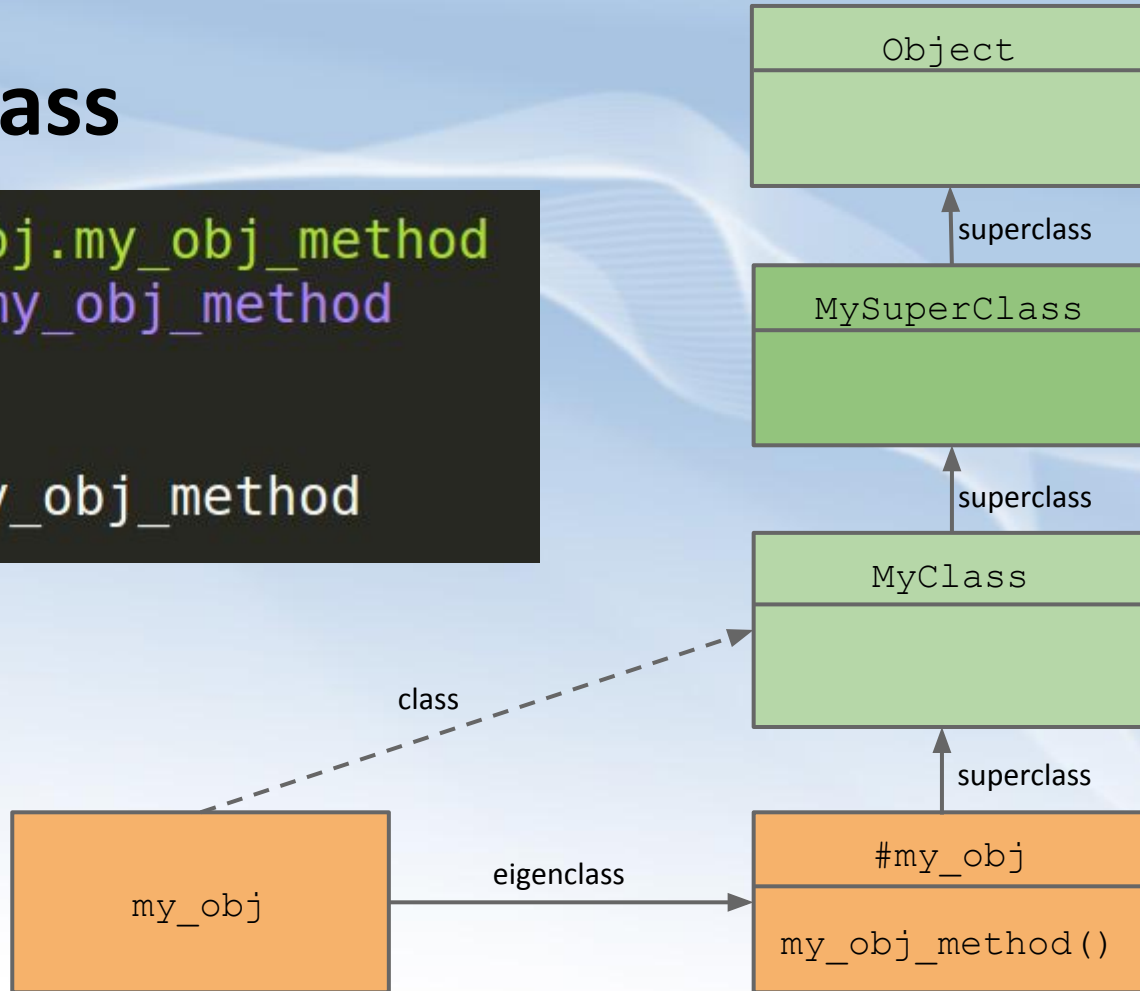
```
def my_obj.my_obj_method  
  puts :my_obj_method  
end
```

```
my_obj.my_obj_method
```



Eigenclass

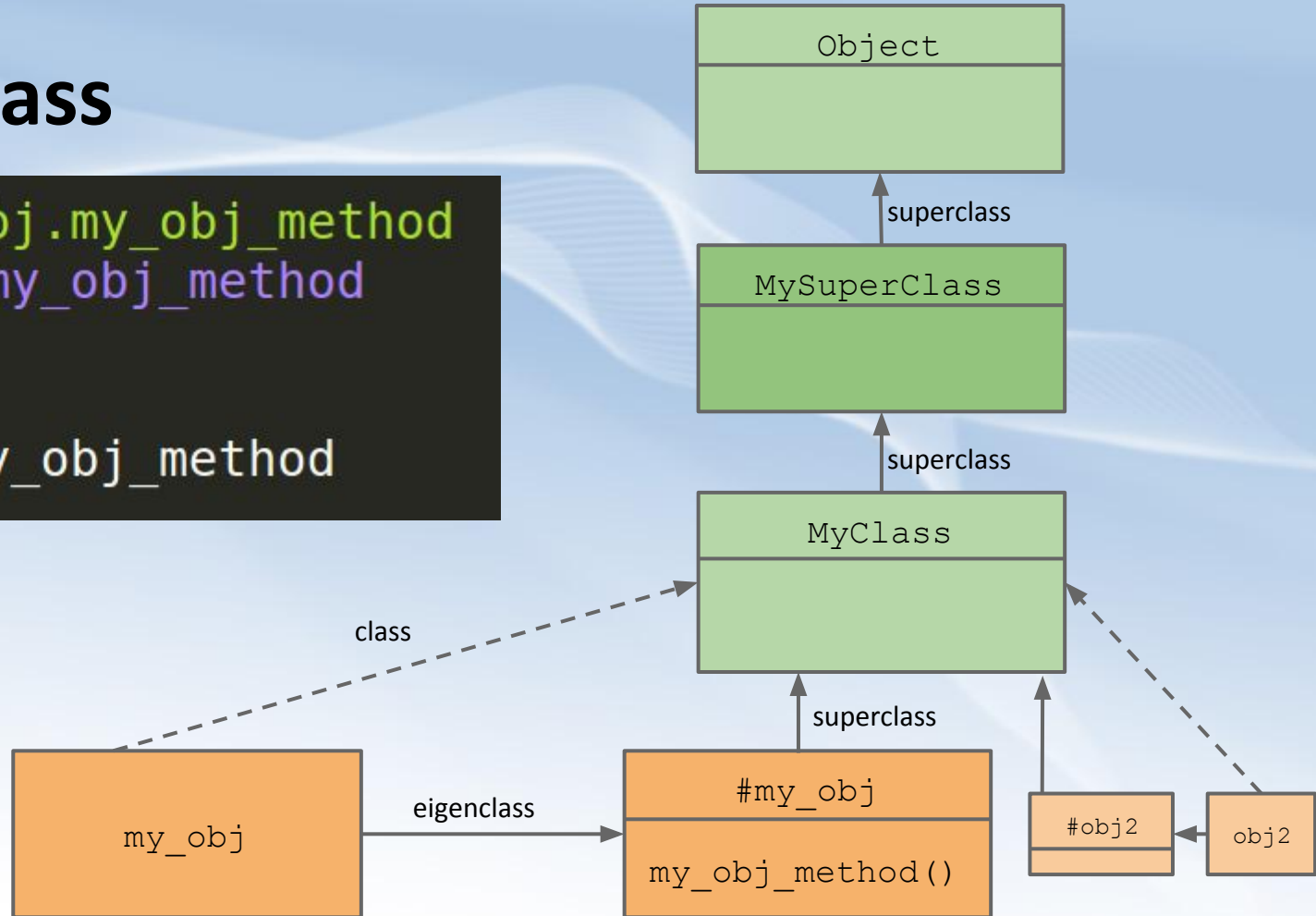
```
def my_obj.my_obj_method  
  puts :my_obj_method  
end  
  
my_obj.my_obj_method
```



Eigenclass

```
def my_obj.my_obj_method  
  puts :my_obj_method  
end
```

```
my_obj.my_obj_method
```



Inside the Eigenclass

```
class << my_obj
  def my_obj_method
    puts :my_obj_method
  end
end
```

```
eigenclass = class << my_obj
  self
end
```

```
eigenclass # #<Class:#<MyClass:0x00000000bfdcf40>>
eigenclass.instance_methods # includes :my_obj_method
eigenclass.superclass # MyClass
```

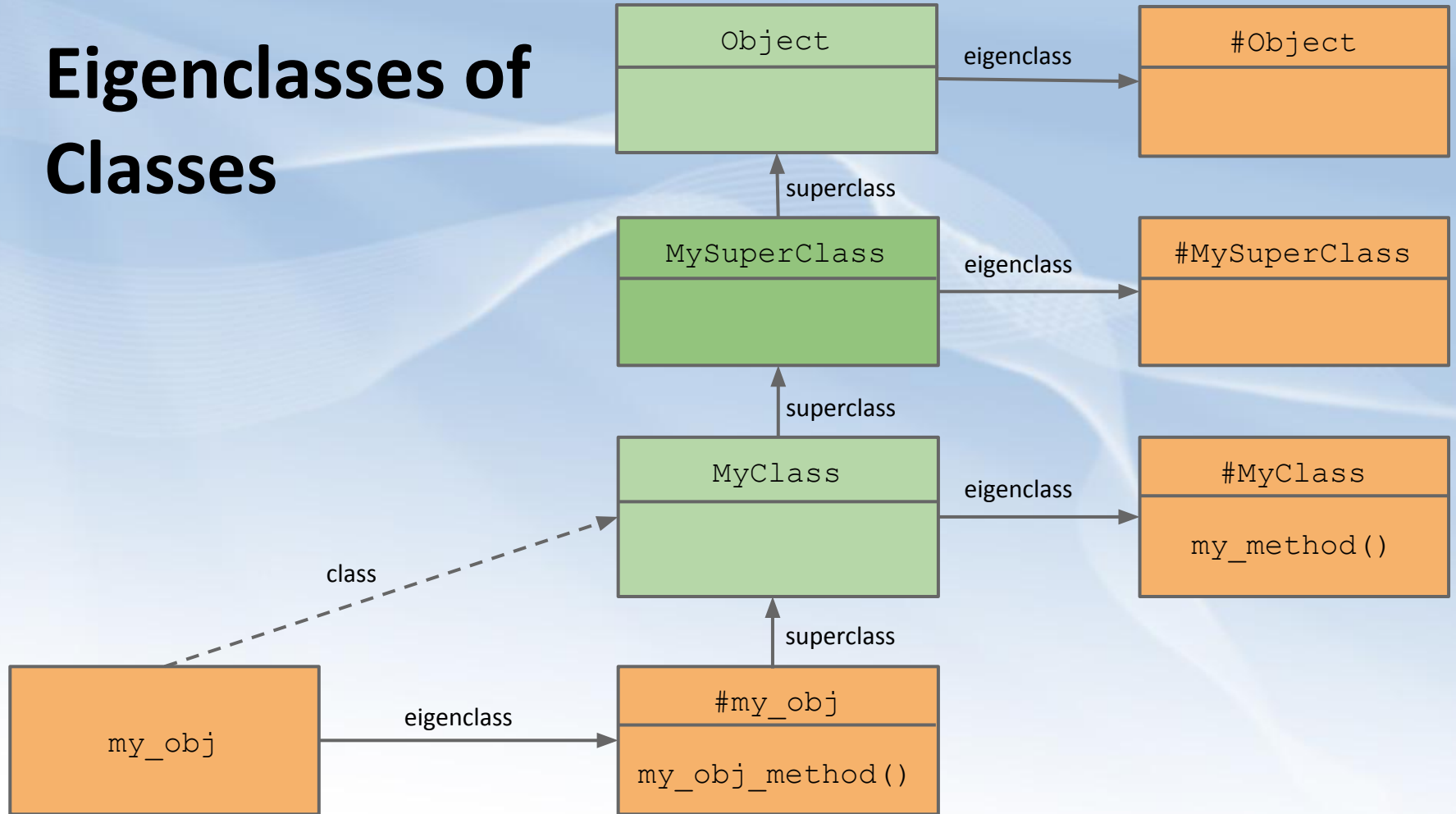
Class Methods = Singleton Methods

```
def MyClass.my_method  
  puts :my_method  
end
```

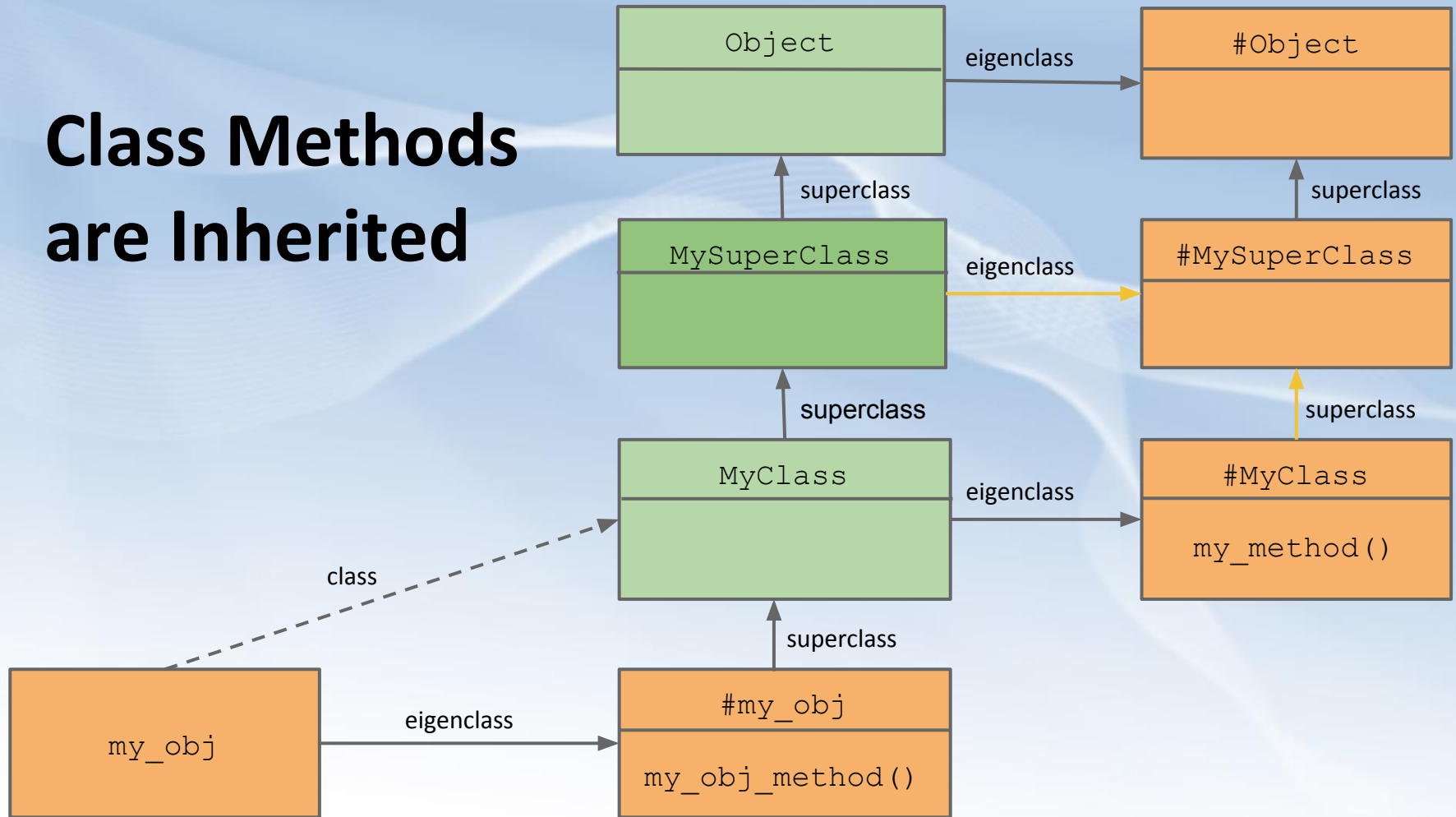
```
class MyClass  
  def self.my_method  
    puts :my_method  
  end  
end
```

```
class MyClass  
  class << self  
    def my_method  
      puts :my_method  
    end  
  end  
end
```

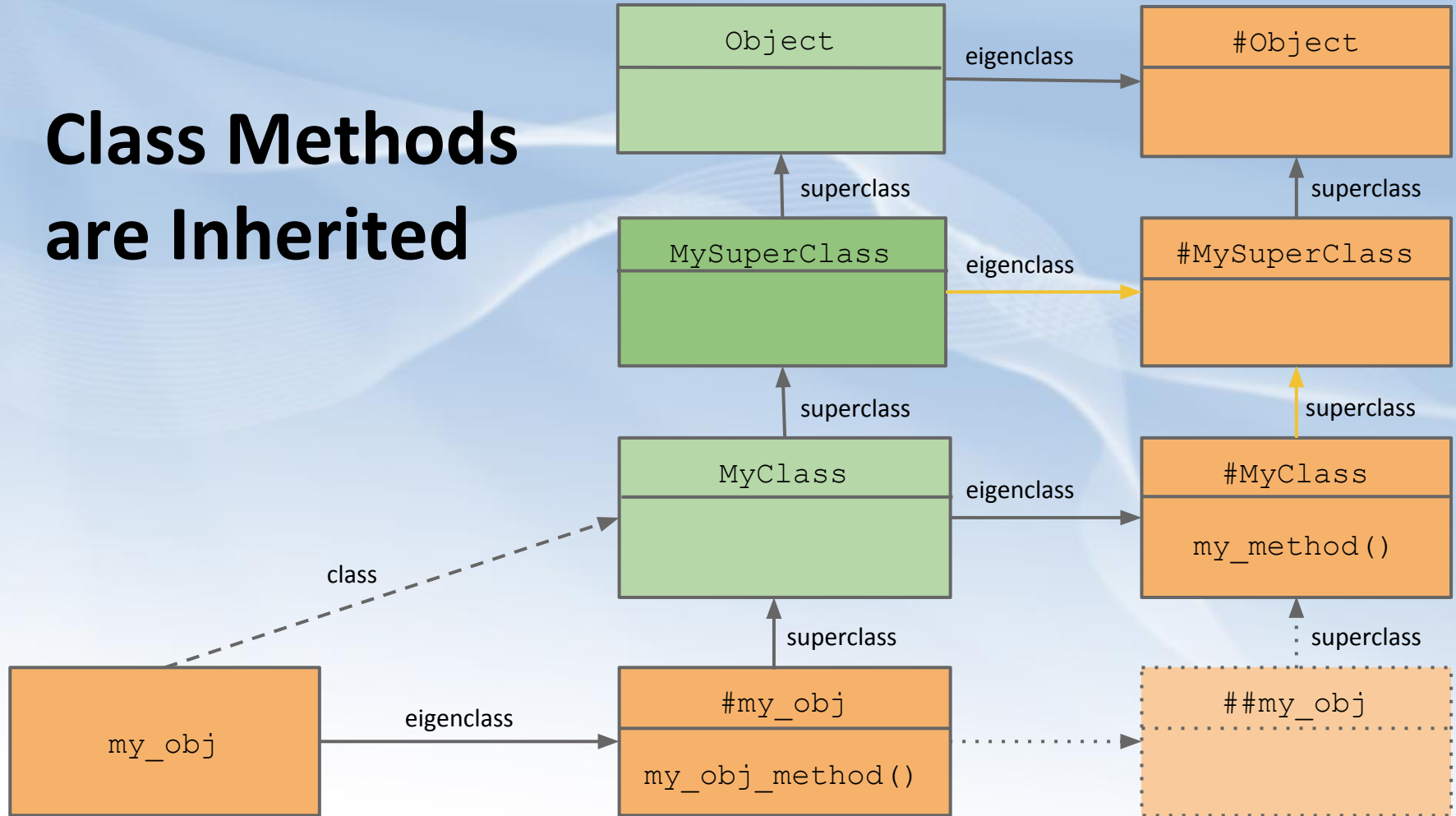

Eigenclasses of Classes



Class Methods are Inherited



Class Methods are Inherited



A close-up photograph of Donald Trump speaking at a podium. He is wearing a dark suit, white shirt, and blue tie. He has a stern, slightly aggressive expression on his face and is pointing his right index finger upwards. The background is a blue curtain. A microphone is visible in the foreground.

HERE'S WHAT YOU CAN DO WITH YOUR CLASS

class_eval and instance_eval

obj.class/instance_eval do

	self	Class Scope
class_eval	the class	the class
instance_eval	receiver (obj)	Eigenclass of receiver

end

No Surprises

```
# instance method
MyClass.class_eval do
  def bye
    puts :bye
  end
end

# class method
MyClass.class_eval do
  def self.later
    puts :later
  end
end
```

```
m = MyClass.new

# singleton method
m.instance_eval do
  def hello
    puts :hello
  end
end

# singleton method
m.instance_eval do
  def self.hi
    puts :hi
  end
end
```

On a Class

```
MyClass.instance_eval do
```

	<code>self</code>	Class Scope
<code>class_eval</code>	the class <code>MyClass</code>	the class <code>MyClass</code>
<code>instance_eval</code>	receiver (<code>MyClass</code>)	Eigenclass of receiver

```
end
```

On an Instance

```
m = MyClass.new  
m.class_eval do
```

	<code>self</code>	Class Scope
<code>class_eval</code>	the class	the class
<code>instance_eval</code>	receiver (<code>m</code>)	Eigenclass of receiver

```
end
```


THE CLASS?

On an Instance

```
m = MyClass.new
```

```
m.class_eval do
```

	<code>self</code>	Class Scope
<code>class_eval</code>	the class Eigenclass of <code>m</code>	the class Eigenclass of <code>m</code>
<code>instance_eval</code>	receiver (<code>m</code>)	Eigenclass of receiver

```
end
```

include VS extend

- `include MyModule`: Includes the *instance methods* of `MyModule` in the receiver *class*
- `extend MyModule`: Includes the *instance methods* of `MyModule` in the *Eigenclass* of the receiver

include VS extend

```
module MyModule
  def self.my_module_method
    puts :my_module_method
  end
end
```

```
  def my_method
    puts :my_method
  end
end
```

```
class MyClass
  include MyModule # instance methods => instance methods
  extend MyModule # instance methods => class methods
end
```

More Reading



@code_kitten



epdejager



The
Pragmatic
Programmers

Metaprogramming Ruby

Program
Like the
Ruby Pros



Paolo Perrotta

Edited by Jill Steinberg