Metaprogramming

Object

- state
- behavior

Me, Me, Me

First, understand self…

self

- `self` is the “current object”
  - Default receiver for method calls with no receiver
  - Place where instance variables are kept

- `self` always has a value

What Changes self?

- method call
- class definition

On Method Call

`receiver.method(params)`

- Switch `self` to `receiver`
- Look up `method` in `self`'s class (and up the chain)
- Invoke method

On Method Call

```
class Dave
  def return_self
    return self
  end
end

d = Dave.new

self          # => main
d.return_self  # => #<Dave:0x27a74>
self          # => main
```

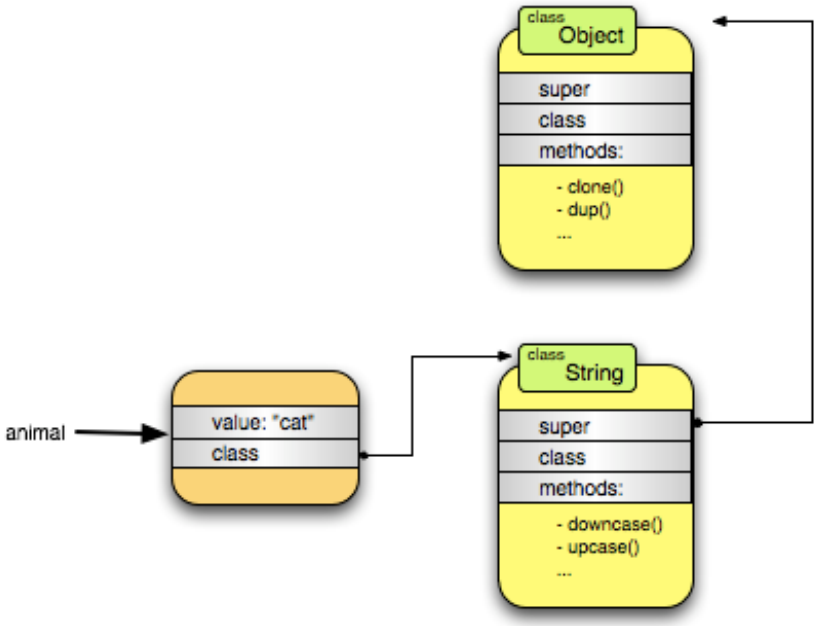
code/meta/meth_call_changes_self.rb

Digging Down…

```
animal = "cat"
puts animal.upcase
```

code/meta/animal.rb

Object = State + Behavior



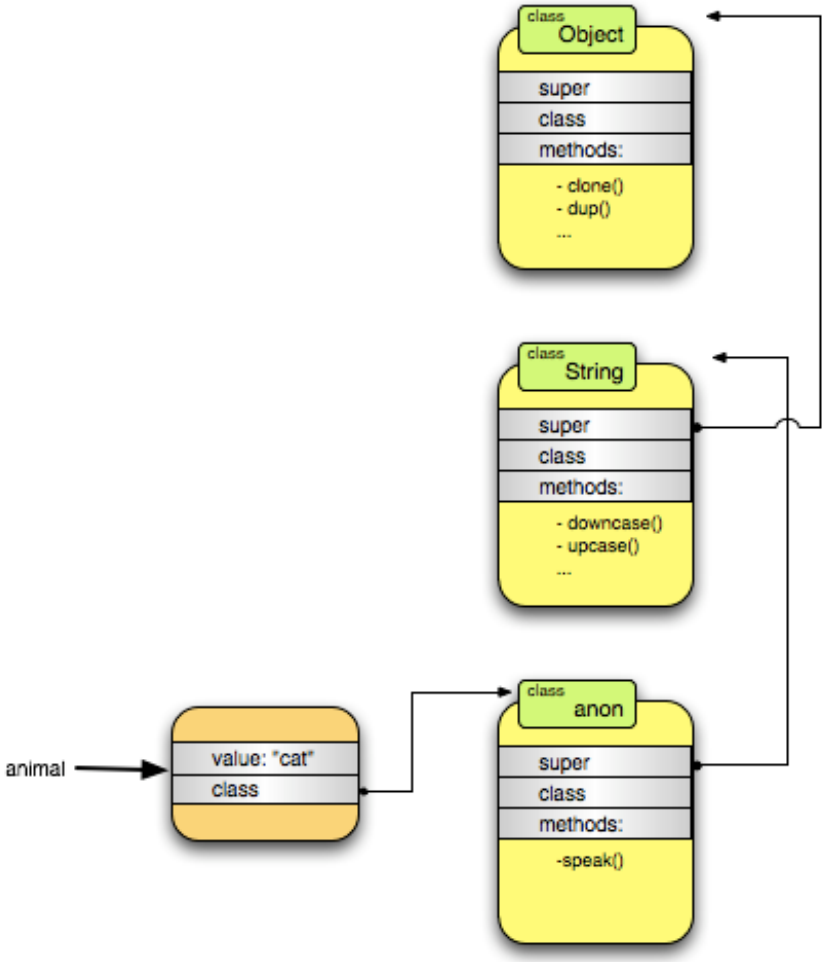
- Inheritance follows ‘super’ chain

Methods on an Object

```
animal = "cat"
def animal.speak
  puts "Miaow"
end
animal.speak
```

code/meta/singleton_animal.rb

Methods on an Object



Singleton Class

- aka: Metaclass, Eigenclass, Uniclass
- regular class, but hidden
- participates in inheritance

`self` and Class Definition

```
class Test
  puts "In the definition of class Test"
  puts "self = #{self}"
  puts "Class of self = #{self.class}"
end

puts "Top level, self=#{self}"
```

code/meta/class_self.rb

- Create (or find existing class object)
- Set `self` to class object
- Reset `self` at end

Classes and Instances

```
class MyClass
  @iv = 123
  puts "In class definition, @iv = #{@iv}"
  def self.class_method
    puts "In class method, @iv = #{@iv}"
  end
  def instance_method
   puts "In instance method, @iv = #{@iv}"  # inaccessible
 end
end

MyClass.class_method
MyClass.new.instance_method
```

code/meta/class_and_instances.rb

- `self` during class definition is the class object
- `MyClass` is a constant that is set to that object

Singleton Methods in a Class

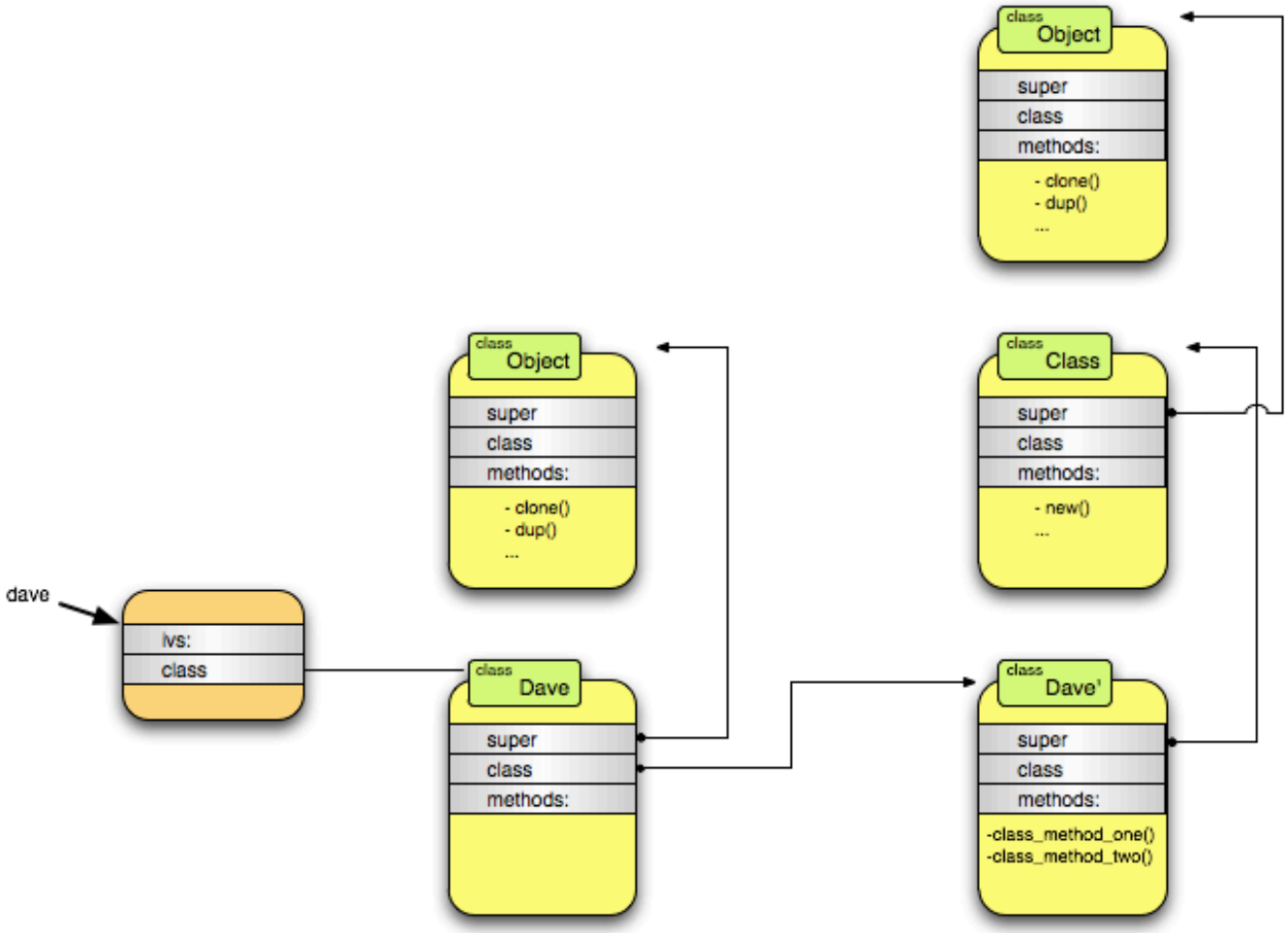
```
class Dave
  def self.class_method_one
    puts "In class method one"
  end
end

Dave.class_method_one
```

code/meta/class_singleton.rb

- There's no such thing as a class method

Methods on a Class



The Tricky Thing

Instance variables looked up in `self`

Methods looked up in `self.class`

The Tricky Thing (Part 2)

`def fred` is defined in current class

`def obj.fred` is defined in `obj`'s singleton class

Inheriting from Objects

- Opens the singleton class of an object
- Syntax is

`class << obj`

Inheriting from Objects

```
animal = "cat"
class << animal
  def speak
    puts "miaow"
  end
end

animal.speak # >> miaow

p animal.methods.sort # >> [ ... "sort_by", "speak", "split", ...]
```

code/meta/class_from_obj.rb

Inheriting from `self`

```
class MyClass
  class << self
    def class_method
      puts "Hi!"
    end
  end
end

MyClass.class_method   # >> Hi!
```

code/meta/class_from_self.rb

Inheriting from `self`

```
class Example
  @iv = 123
  class << self
    attr_reader :iv
  end
end

puts Example.iv
```

code/meta/class_iv.rb

Inheriting From Expressions

- The Ruby syntax is

`class` *name* < *expression*

- *expression* is anything that returns a `Class` object

Inheriting from Expressions

```
class Address < Struct.new(:street, :city, :zip, :state)
  def to_s
    "#{self.street}\n#{self.city}, #{self.state} #{self.zip}"
  end
end

add = Address.new("123 Main", "Springfield", "75123", "IL")

puts add.to_s  # >> 123 Main
               # >> Springfield, IL 75123
```

code/meta/class_from_struct.rb

Class Method Inheritance

```
class Parent
  def self.hello
    puts "hi from #{self}"
  end
  hello    # >> hi from Parent
end
class Child < Parent
  hello    # >> hi from Child
end
```

code/meta/class_methods_self.rb

Include

```
class Logger
  def log(msg)
    puts msg
  end
```

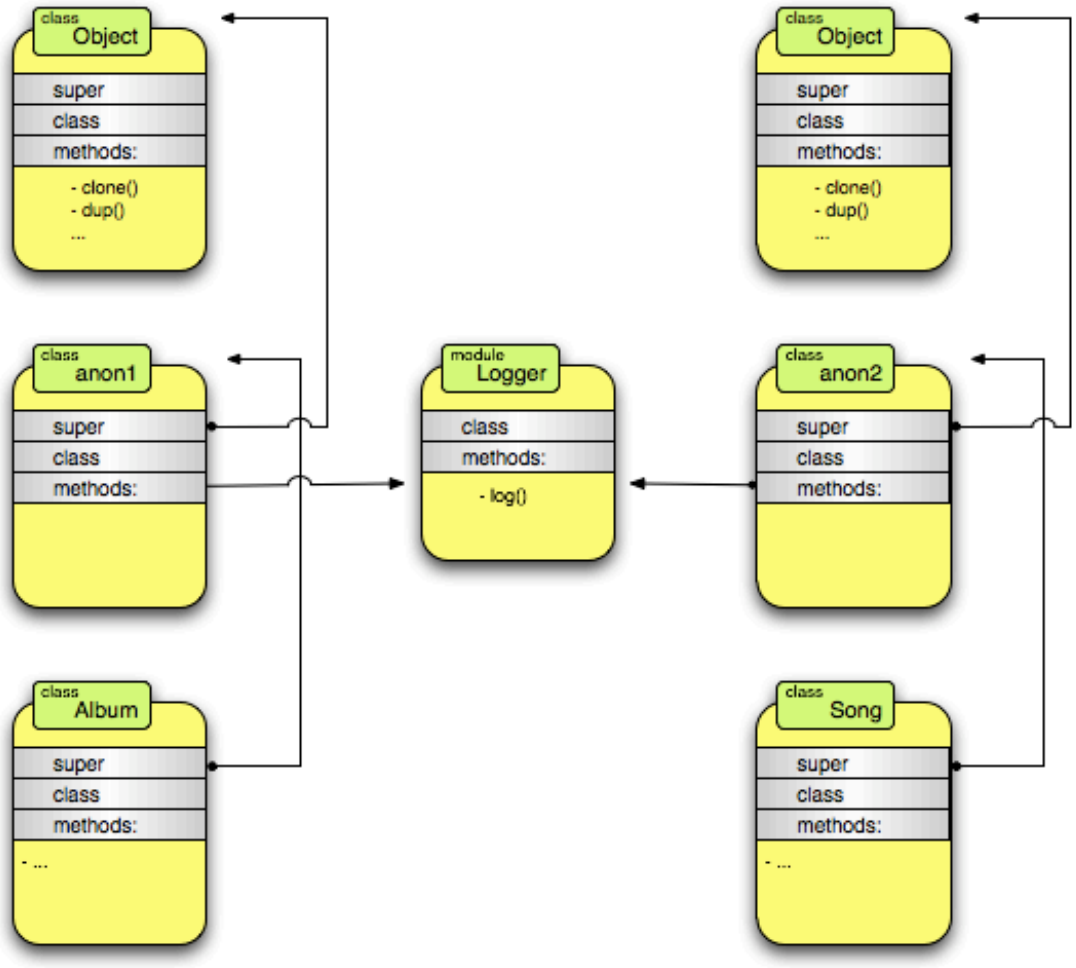
```
end

class Album
  include Logger
end

class Song
  include Logger
end
```

code/meta/include.rb

Including a Module



Include

- Creates an invisible class that proxies the module
- Makes that class the receiver's immediate parent

Extend

- Adds methods to a particular object

```
module Vocal
  def speak
    puts "#{self} says hello"
  end
end

animal = "cat"
animal.extend Vocal
animal.speak
```

code/meta/extend.rb

- Adds methods to the receiver's singleton class