Eigenclasses, Singletons, What?

Christopher Mark Gore

cgore.com

Monday, June 8, AD 2015

Everything is a Thing

```ruby
12.class # Fixnum
"Hi there".class # String
2.71828.class # Float
[1, 2, 3].class # Array
{age: 36, name: "Chris"}.class # Hash
:foo.class # Symbol
```

We Can Insist on the "Right" Thing

We'll explain the difference between the concepts "instance of," "is a," and "kind of" in a few slides.

```
1  if not k.class == Fixnum
2    raise ILoveOnlyNumbers
3  end
4  if not m.instance_of? Fixnum
5    raise ILoveOnlyNumbers
6  end
7  if not n.is_a? Fixnum
8    raise ILoveOnlyNumbers
9  end
10 if not o.kind_of? Fixnum
11   raise ILoveOnlyNumbers
12 end
```

We Can Do Different Things to Different Things

```
1 if n.class == Fixnum
2   puts "Whole␣Numbers␣FTW!!!"
3 elsif n.class == Float
4   puts "More␣Accuracy␣is␣MORE!!!"
5 else
6   raise ILoveOnlyNumbers
7 end
```

Even Classes are Things!

This is one of the more unique aspects of Ruby. In a lot of other object-oriented programming languages, classes are "special" somehow, and you can't treat them the same way as everything else.

```
1 :foo.class # Symbol
2 :foo.class.class # Class
3 :foo.class.class.class # Class
```

We'll see how that's actually really nice and useful in a bit.

We Can Make Our Own Classes

```ruby
class Cat
  def initialize color
    @color = color
    @lives = 9
  end
  def die
    @lives = @lives - 1
  end
  def alive?
    @lives >= 1
  end
  def eat_tuna
    puts "Purrr!"
  end
end
lucky = Cat.new "black"
lucky.alive? # true
```

Classes Have Methods

And those methods let our class instances do things.

```
1  class ATM
2    def initialize amount
3      @amount = amount
4    end
5    def deposit account, amount
6      @amount = @amount + amount
7      account.deposit amount
8    end
9    def withdraw accout, amount
10     raise GoAwayPeasant if amount > account.amount
11     raise RunOnTheBank if amount > @amount
12     account.withdraw amount
13     @amount = @amount - amount
14   end
15 end
```

Instance Variables

Instance variables are written with an at-sign like `@foo`, and are per-instance.

```
1  class Person
2    attr_reader :age, name
3    def initialize(age, name)
4      @age = age
5      @name = name
6    end
7  end
8  john = Person.new 12, "John␣Jacobson"
9  fred = Person.new 44, "Fred␣Fellows"
10 john.age # 12
11 fred.age # 44
12 john == fred # false
```

Getters and Setters

```ruby
class Car
  attr_accessor :color
  attr_reader :mileage
  def initialize(color, mileage)
    @color, @mileage = color, mileage
  end
  def drive miles
    @mileage = @mileage + miles
  end
end
c = Car.new 4, :blue
c.drive 92_000
c.mileage # 92004
c.color = :green
c.mileage = 123 # NoMethodError: undefined method `mileage='
```

Class Variables

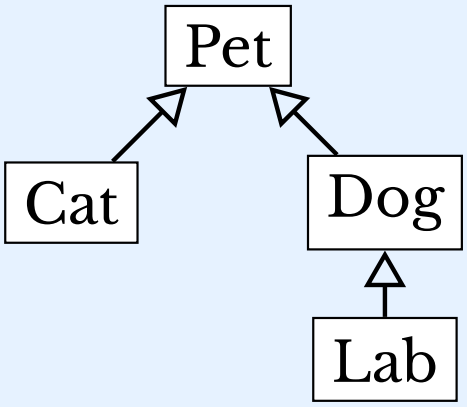
Class variables are written with two at-signs like `@@foo`, and are per-class. They are almost never what you actually want.

```
1  class People
2    @@population = 0
3    attr_reader :name, :age
4    def initialize(name, age)
5      @@population = @@population + 1
6      @alive = true
7      @name, @age = name, age
8    end
9    def die
10     @alive = false
11     @@population = @@population - 1
12   end
13 end
```

Class Methods

```
1  class Person
2    attr_reader :name, :born, :died
3    def initialize(name, born, died)
4      @name, @born, @died = name, born, died
5    end
6    class << self
7      def from_s s # String like "John Doe (1912 - 1990)"
8        name, born, died = s.scan(/(.*)\((\d+) - (\d+)\)/)[0]
9        Person.new name, born.to_i, died.to_i
10     end
11   end
12 end
13 george = Person.from_s "George␣Washington␣(1732␣-␣1739)"
14 ben = Person.from_s "Benjamin␣Franklin␣(1706␣-␣1790)"
```

Inheritance




```
1 class Pet
2 end
3 class Cat < Pet
4 end
5 class Dog < Pet
6 end
7 class Lab < Dog
8 end
```

“Kind Of” versus “Instance Of”

Class Instance Variables

```
1  class Foo
2    @bar = 0
3    attr_accessor :bar
4    def initialize(bar)
5      @bar = bar
6    end
7    class << self
8
9    end
10 end
```

Mixins

Questions?