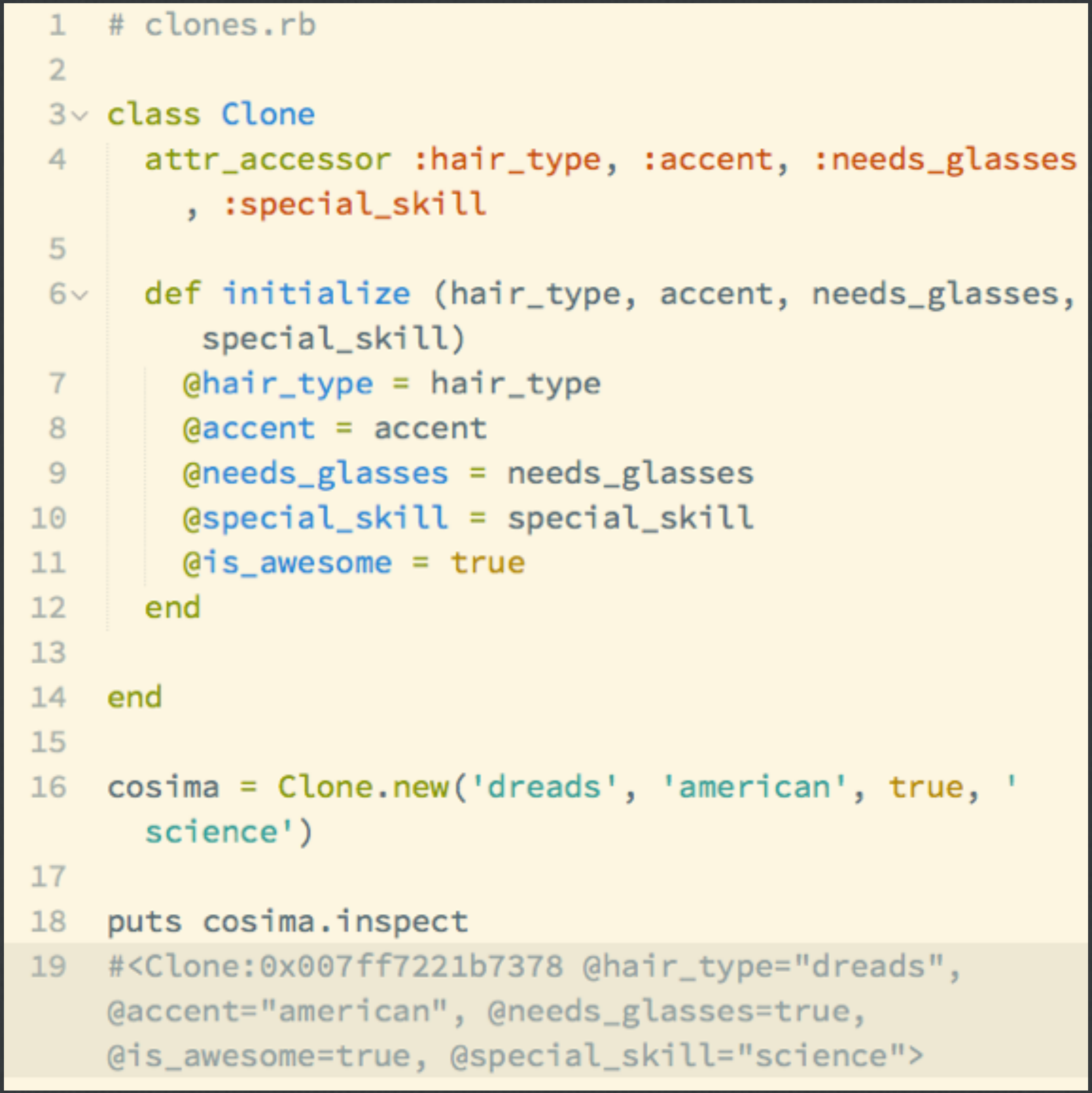
Inheriting traits and behavior

Classes

quick recap

BEDROOM
FAMILY ROOM
STAIR
KITCHEN
SECOND FLOOR
FIRST FLOOR
UTILITIES
EXISTING PORCH
GARAGE



```
# clones.rb

class Clone
  attr_accessor :hair_type, :accent, :needs_glasses
    , :special_skill

  def initialize (hair_type, accent, needs_glasses,
     special_skill)
    @hair_type = hair_type
    @accent = accent
    @needs_glasses = needs_glasses
    @special_skill = special_skill
    @is_awesome = true
  end

end

cosima = Clone.new('dreads', 'american', true, '
  science')

puts cosima.inspect
#<Clone:0x007ff7221b7378 @hair_type="dreads",
@accent="american", @needs_glasses=true,
@is_awesome=true, @special_skill="science">
```

Quick lab

- With a partner, create a new directory called animals in YOUR class folder
- in animals/lib, create a Dog and Cat class in separate ruby files.
- in animals/, create a main.rb file.
- Give the dog and cat three attributes relevant to what they actually are, and an appropriate to_s method
- Give them each a talk method that makes them talk the way dogs and cats talk.
- In main.rb, require dog and cat and instantiate three of each and put them in a dogs and cats array.
- Loop over the arrays and make them all talk!

Think of a ball

This could get out of hand

Imagine that you have a complex Ruby application that has a User class with lots of methods and attributes.

This could get out of hand

If we want to make an Admin class that can do everything a user can do, but can also do some extra stuff, we don’t have to rewrite everything from the User class.

This could get out of hand

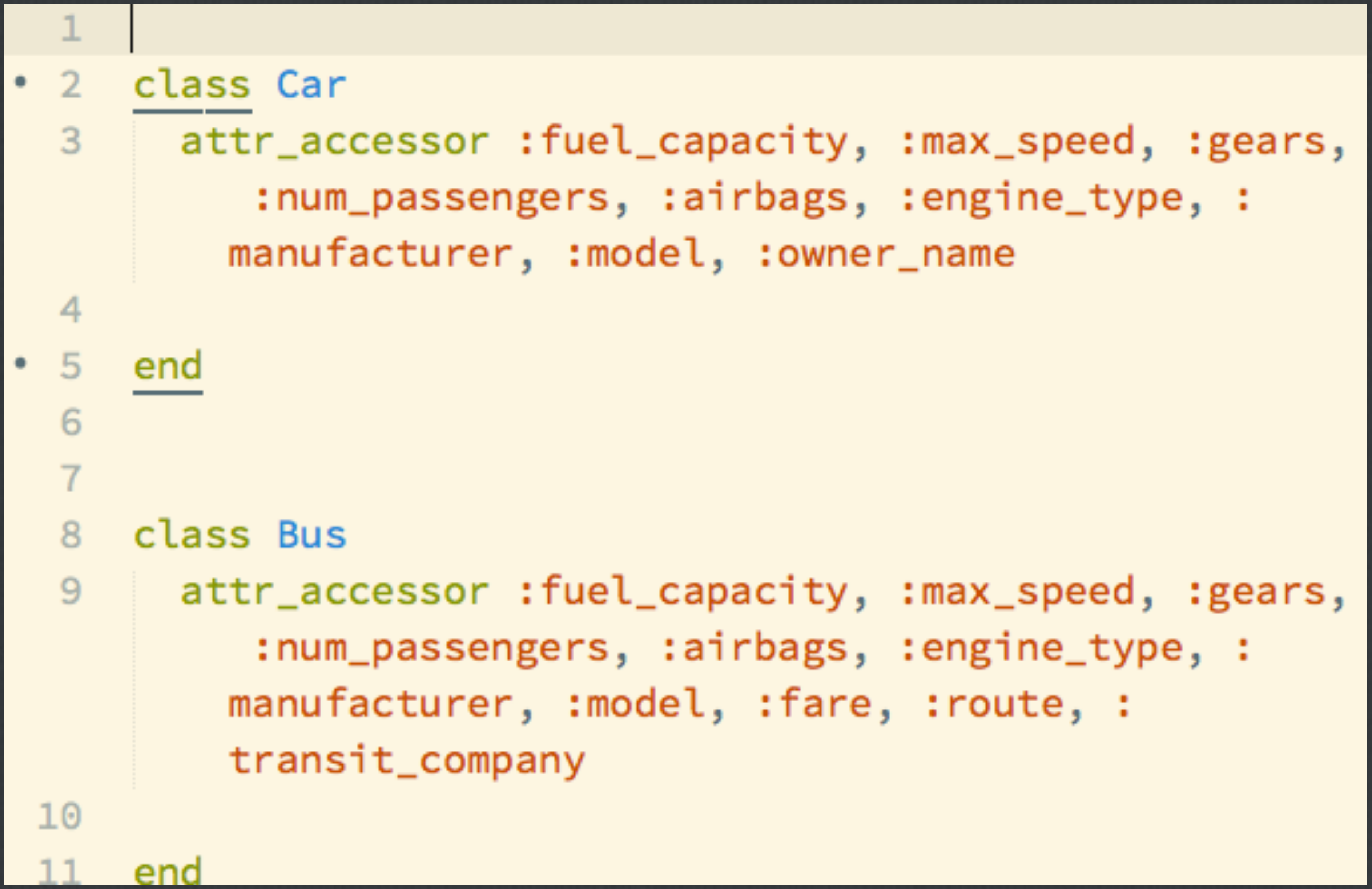
If we want to make an Admin class that can do everything a user can do, but can also do some extra stuff, we don't have to rewrite everything from the User class.

That would not be very elegant or easy to maintain.

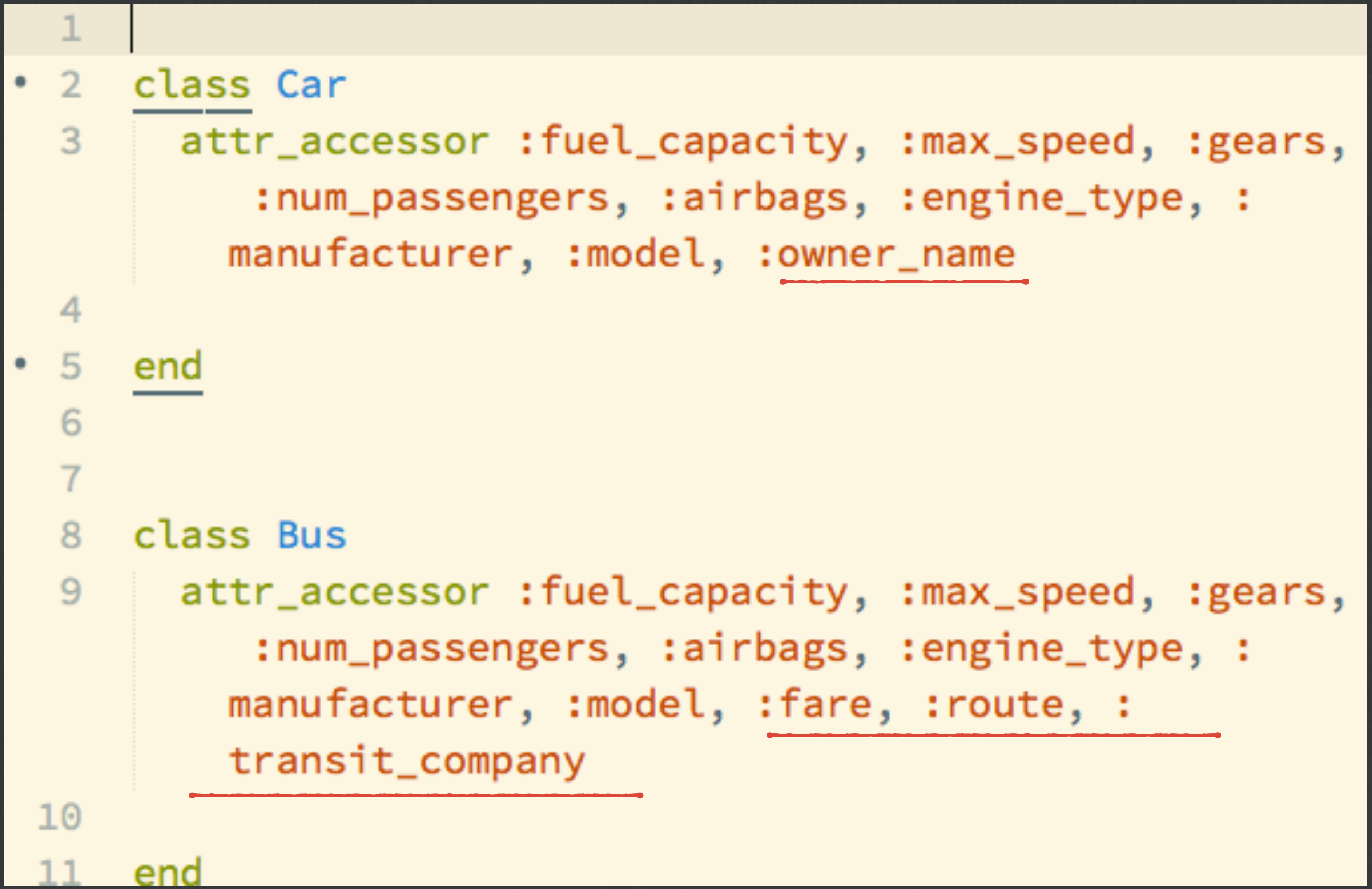
Rather than clones

What are some things that inherit traits from another?

For instance, a car and a bus share a lot in common before they diverge

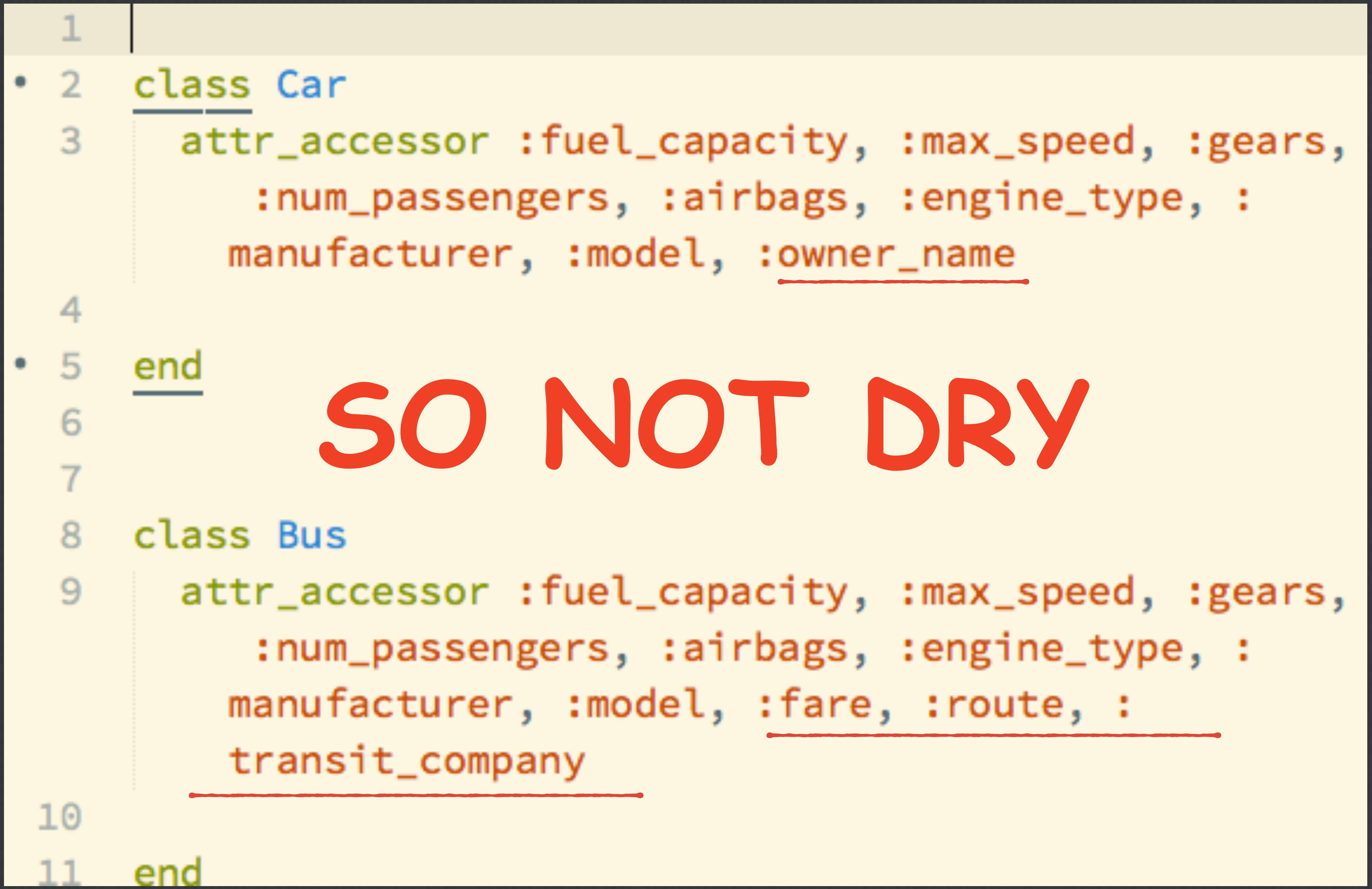


```

class Car
  attr_accessor :fuel_capacity, :max_speed, :gears,
     :num_passengers, :airbags, :engine_type, :
    manufacturer, :model, :owner_name

end


class Bus
  attr_accessor :fuel_capacity, :max_speed, :gears,
     :num_passengers, :airbags, :engine_type, :
    manufacturer, :model, :fare, :route, :
    transit_company

end
```



```ruby

class Car
  attr_accessor :fuel_capacity, :max_speed, :gears, :num_passengers, :airbags, :engine_type, :manufacturer, :model, :owner_name

end


class Bus
  attr_accessor :fuel_capacity, :max_speed, :gears, :num_passengers, :airbags, :engine_type, :manufacturer, :model, :fare, :route, :transit_company

end
```

only a few differences



```
class Car
  attr_accessor :fuel_capacity, :max_speed, :gears,
     :num_passengers, :airbags, :engine_type, :
    manufacturer, :model, :owner_name

end


class Bus
  attr_accessor :fuel_capacity, :max_speed, :gears,
     :num_passengers, :airbags, :engine_type, :
    manufacturer, :model, :fare, :route, :
    transit_company

end
```

SO NOT DRY

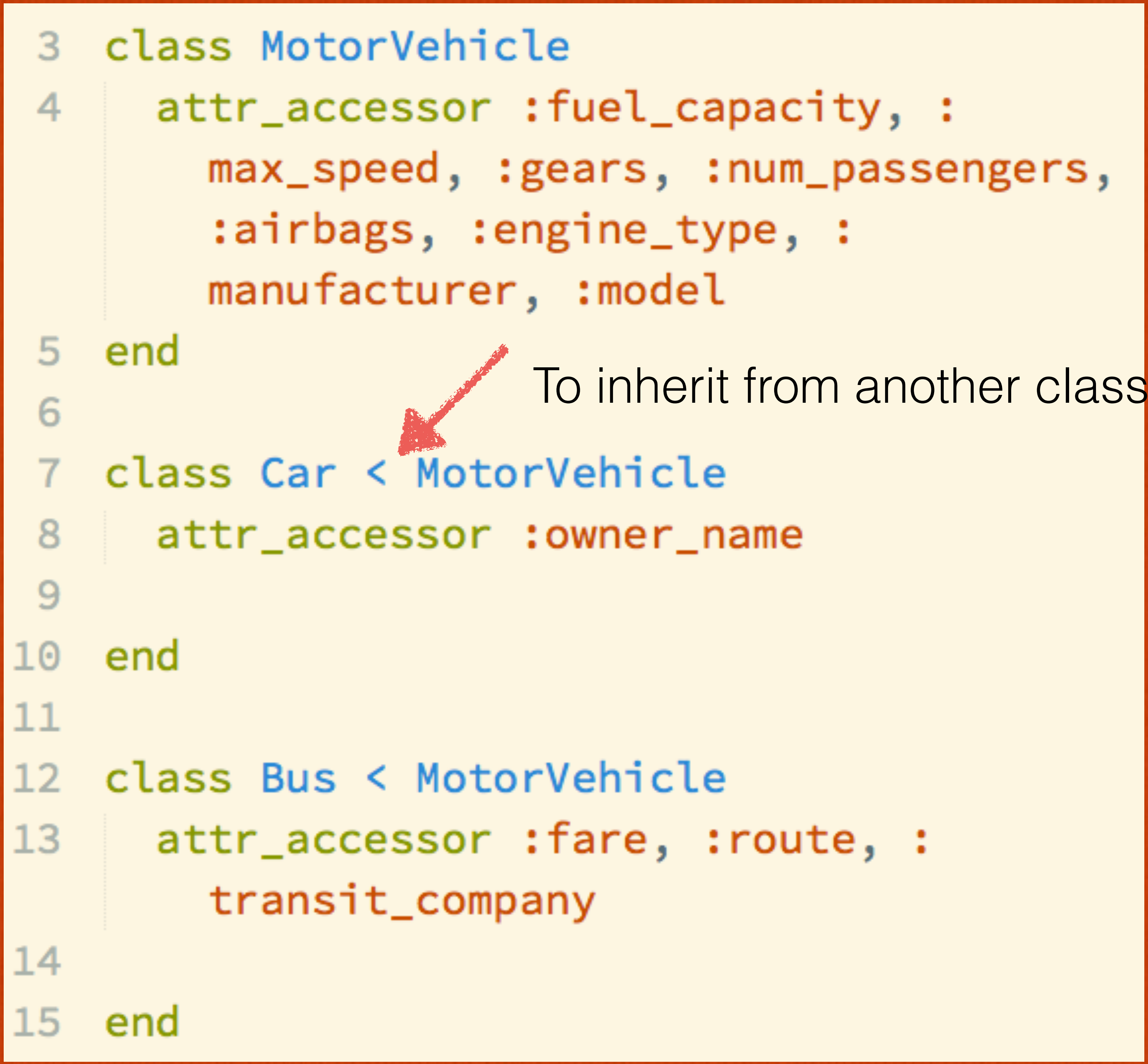
only a few differences

INSTRUCTOR

Classes don’t just dictate behavior and traits for objects, they can pass them on to different classes

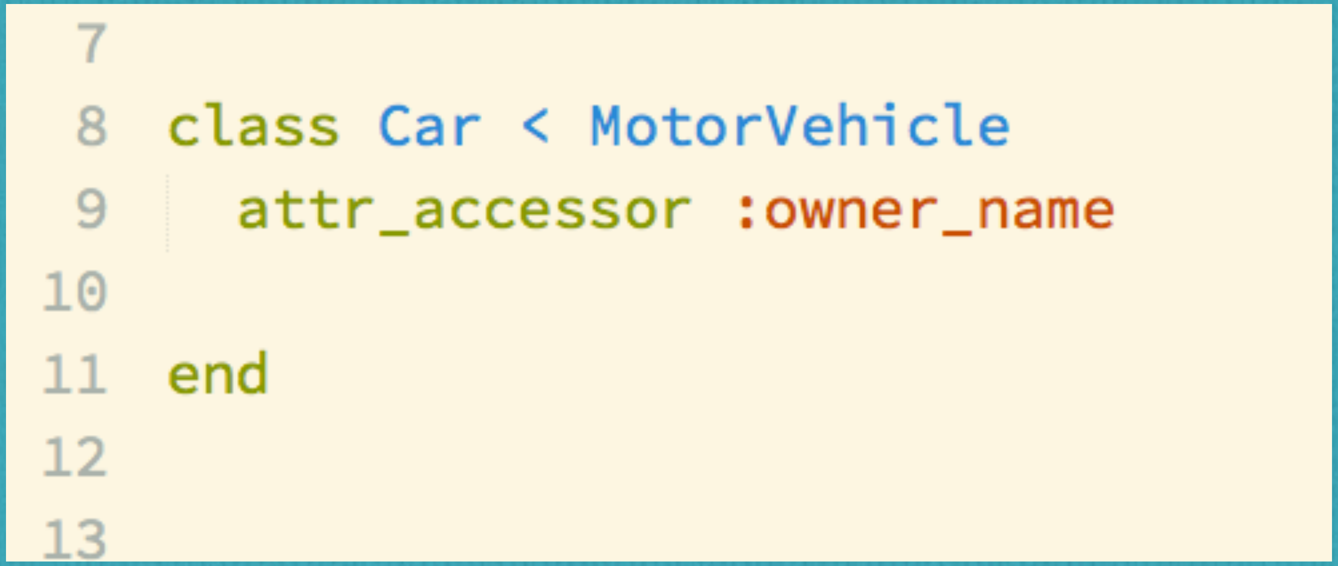
<

That's how classes inherit from other classes



```ruby
class MotorVehicle
  attr_accessor :fuel_capacity, :
    max_speed, :gears, :num_passengers,
    :airbags, :engine_type, :
    manufacturer, :model
end

class Car < MotorVehicle
  attr_accessor :owner_name

end

class Bus < MotorVehicle
  attr_accessor :fare, :route, :
    transit_company

end
```

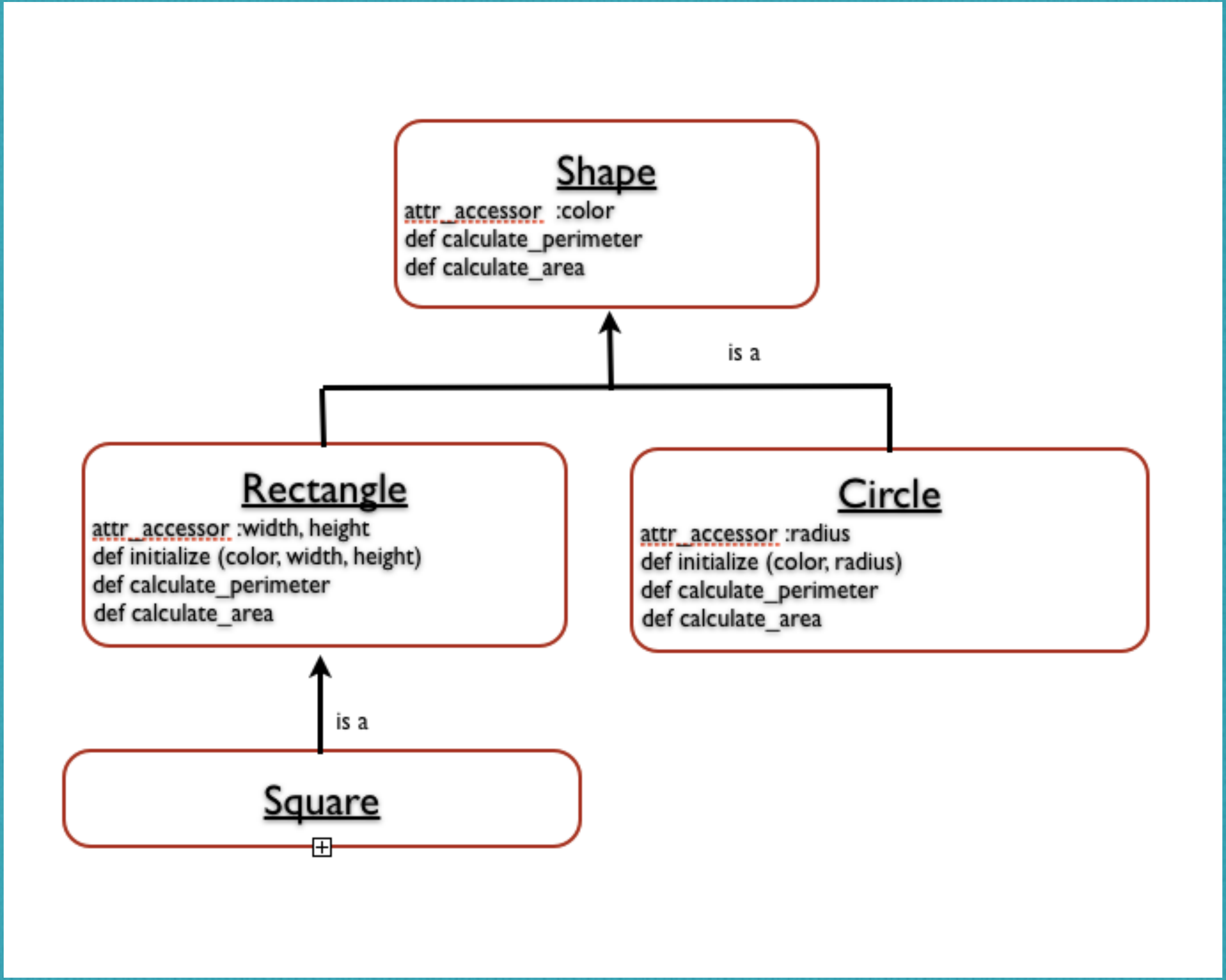
To inherit from another class, use “<“



```ruby

class Car < MotorVehicle
  attr_accessor :owner_name

end


```

This could read like:

I, Motor Vehicle, being of sound mind, do solemnly give thee, Car, all of my attributes and methods in perpetuity.


Shape
attr_accessor :color
def calculate_perimeter
def calculate_area
is a
Rectangle
attr_accessor :width, height
def initialize (color, width, height)
def calculate_perimeter
def calculate_area
Circle
attr_accessor :radius
def initialize (color, radius)
def calculate_perimeter
def calculate_area
is a
Square

Hot take

Let's refactor the animals from before, make them both inherit from Animal

Give Animal attributes (vertebrae, num_legs, your call) and set them as defaults in Cat and Dog

5 minutes please!

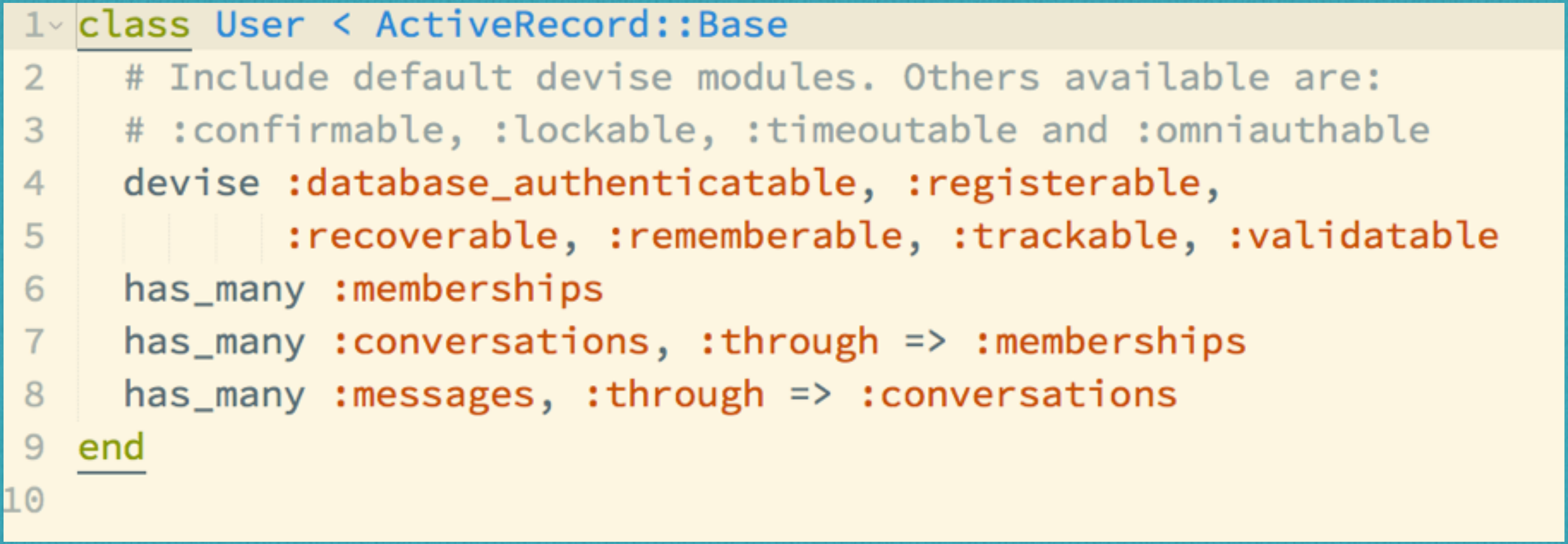
Code along: a bank app

- We are going to make a small app that has a class for Savings accounts, but might have different types of accounts as well.
- Since these two types of bank accounts share a lot of behavior, we will try to DRY it up with inheritance

Make your bank more sophisticated

- Let's extend on the bank app with Checking accounts
- Checking accounts should be able to charge you monthly, but bear no interest. No minimums either
- If you finish early, think about how you can give all Accounts the ability to transfer money from one account to another

How this is relevant to Rails



```ruby
class User < ActiveRecord::Base
  # Include default devise modules. Others available are:
  # :confirmable, :lockable, :timeoutable and :omniauthable
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :trackable, :validatable
  has_many :memberships
  has_many :conversations, :through => :memberships
  has_many :messages, :through => :conversations
end

```

Wrapping up

- What is a class?

Wrapping up

- What is a class?
- What operator lets classes inherit from one another?

Wrapping up

- What is a class?
- What operator lets classes inherit from one another?
- What are the benefits of inheritance?

Wrapping up

- What is a class?
- What operator lets classes inherit from one another?
- What are the benefits of inheritance?
- How can a child class override its parent class?

Default image that comes with Keynote

More new stuff

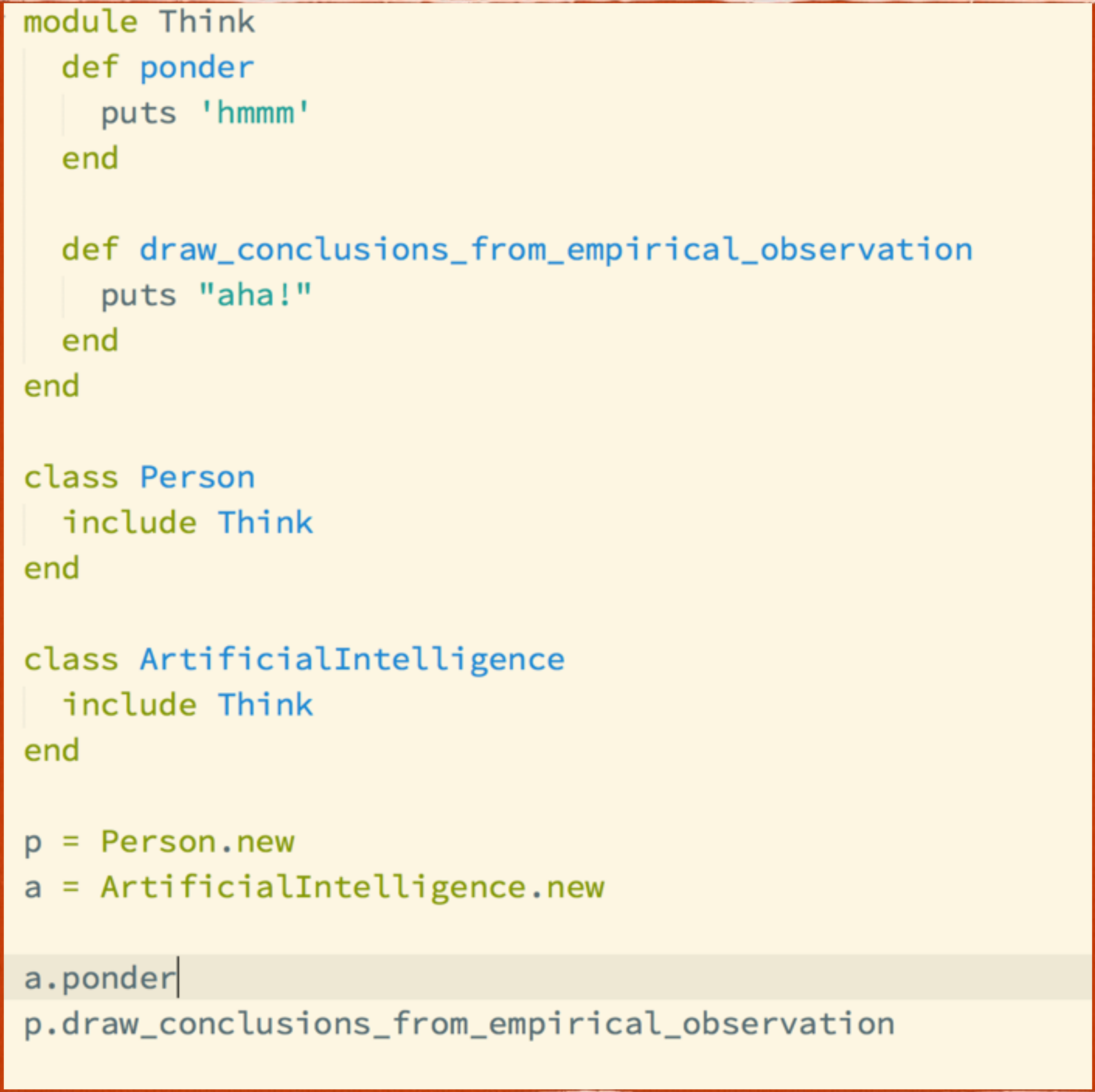
- Classes can inherit from each other.
- When you inherit from another class, you gain <u>everything</u> about it. Minus the name.
    - This includes methods, attributes, class methods
- This is helpful for sharing functionality between classes, but sometimes you want two classes to share only some behavior.

Modules

- Modules are what allow you to share only some behavior. You include them in classes. You can’t make instances of modules
- They start with
    - module MyModuleName
- Similar to
    - class MyClassName

Using them

- To compose a class with a module, you use the “include” method
- So, given a module called “Human”…
  - include Human
- “include” goes near your attr_accessor (if you have one)
- It should be inside the class



```ruby
module Think
  def ponder
    puts 'hmmm'
  end

  def draw_conclusions_from_empirical_observation
    puts "aha!"
  end
end

class Person
  include Think
end

class ArtificialIntelligence
  include Think
end

p = Person.new
a = ArtificialIntelligence.new

a.ponder
p.draw_conclusions_from_empirical_observation
```