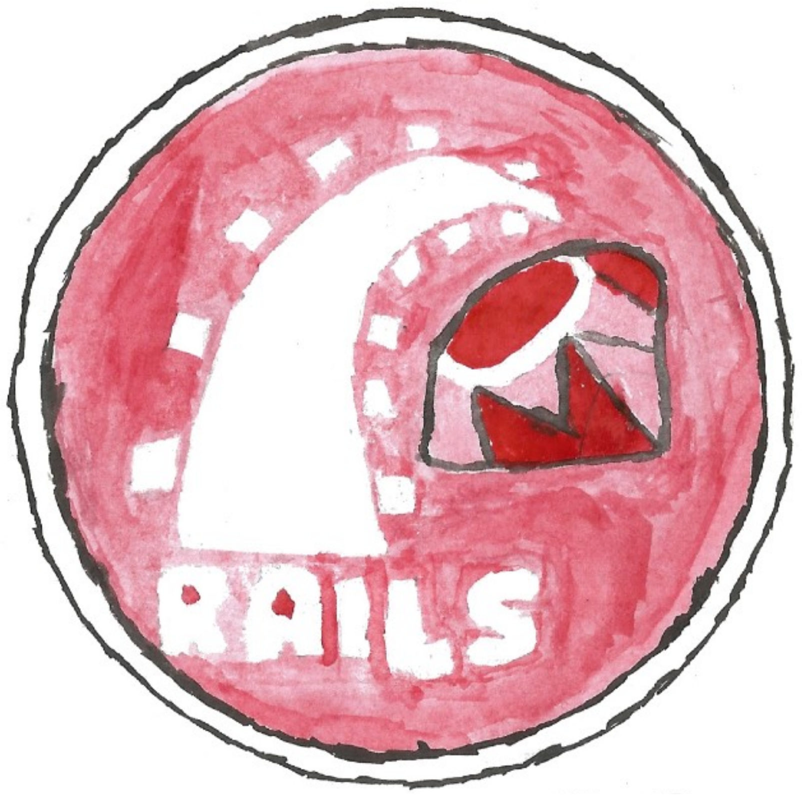
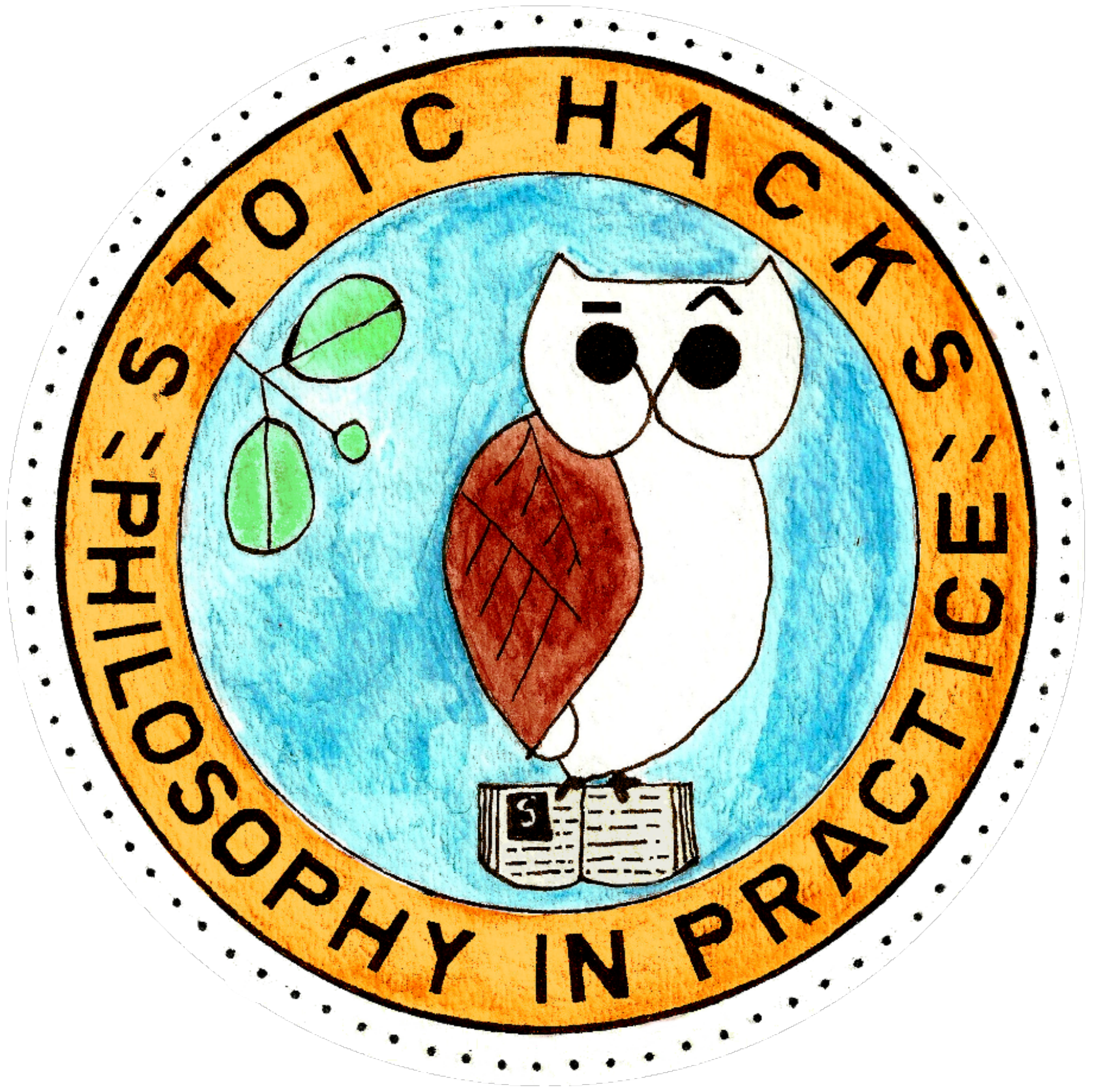


7 Lessons from Learning Ruby (again, sort of)

Eruditus Presentation #2
By Henry van Wagenberg

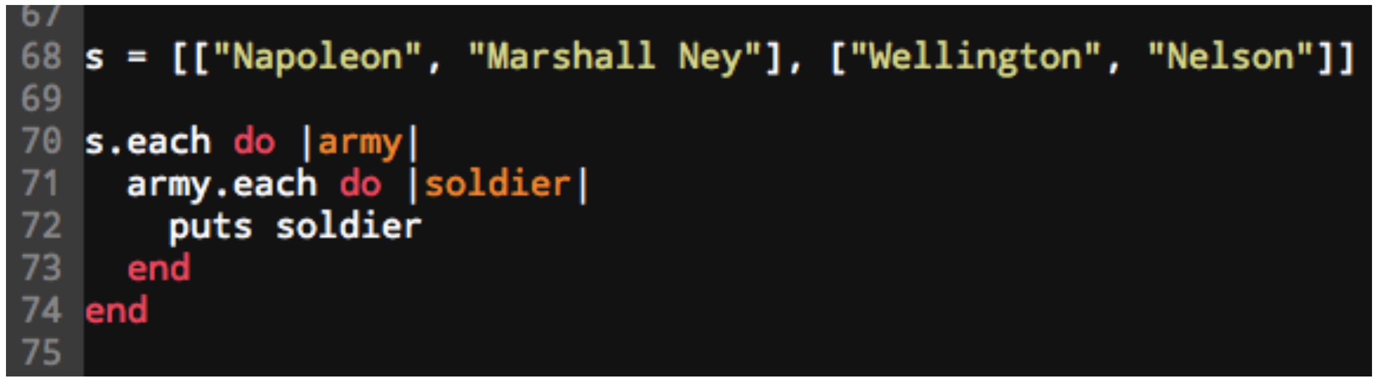
#1 Humility - and opening up to shame

- Be humble
- Learning happens when you admit to yourself and others what you don't know
- (I always think I know this lesson; then I encounter someone who is negative about this stuff)
- **Sources for Learning**
- *Design Patterns in Ruby*
- *Why's Guide to Ruby*
- *SitePoint's History of Ruby*
- *The Ruby Language FAQ*

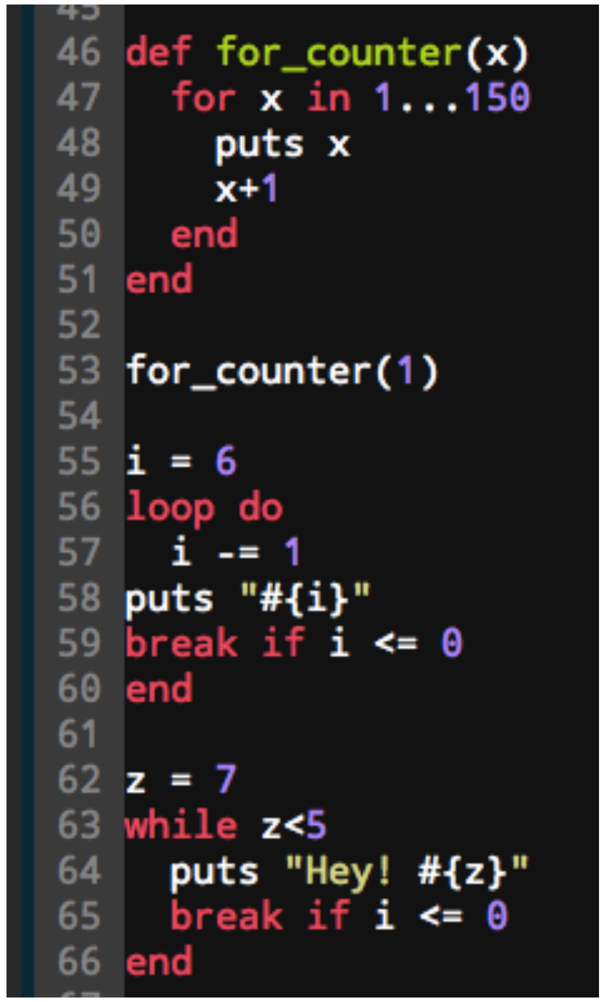


#2 .each and blocks

- .each's rules as a method are really just passing in a block to it: the pipes are the way an argument is passed into any block; do and end or { } are the way you write any block
- Difference between a loop and an iterator
- Ruby also has loops, that nobody uses
    - for, do, while



```ruby
s = [["Napoleon", "Marshall Ney"], ["Wellington", "Nelson"]]

s.each do |army|
  army.each do |soldier|
    puts soldier
  end
end

```

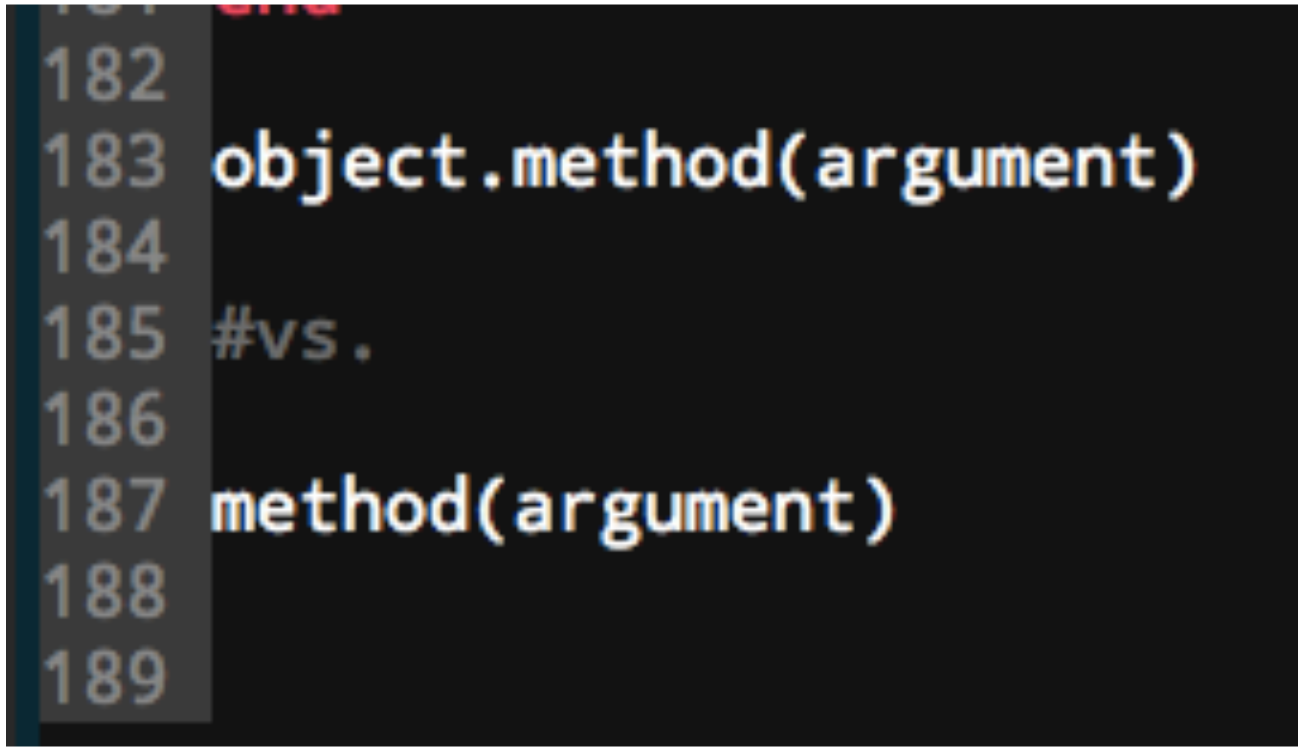


```ruby
def for_counter(x)
  for x in 1...150
    puts x
    x+1
  end
end

for_counter(1)

i = 6
loop do
  i -= 1
puts "#{i}"
break if i <= 0
end

z = 7
while z<5
  puts "Hey! #{z}"
  break if i <= 0
end
```

#4 When do you call a method with "." ? When do you call it simply by invoking its name?

You call methods with "." when the method is defined in a particular class and can only be invoked on a particular type of object.

You call methods simply by invoking their name when the method is either a kernel method (and thus works anywhere) such as **puts** or it's a method that you yourself (or someone) has defined in the wide world of the program -- without defining it inside of a particular object.



```

object.method(argument)

#vs.

method(argument)


```

#5 Gang of 4 Object-Oriented Patterns: in particular **Composition**

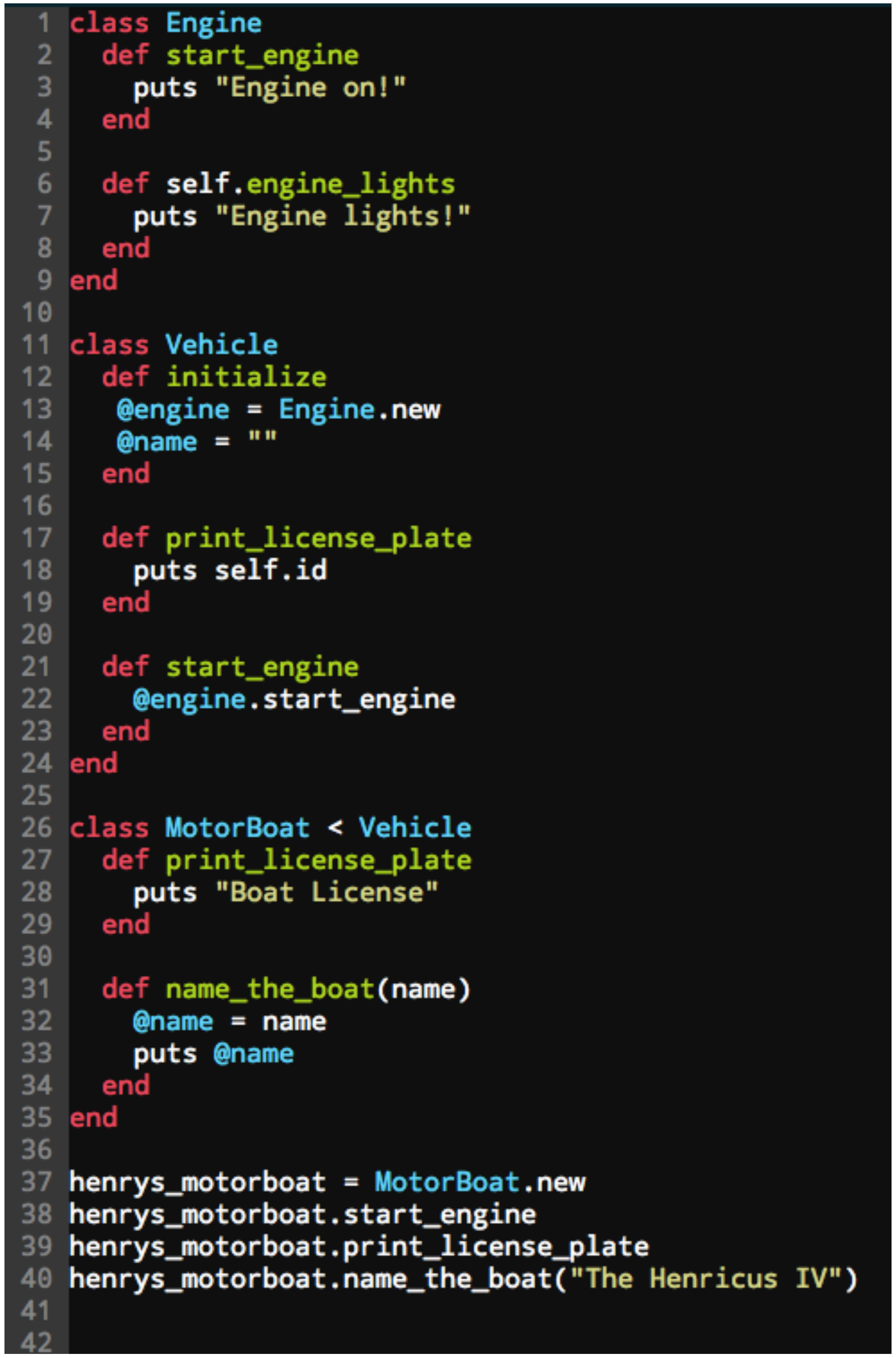
#1 Separate out the things that change from those that stay the same
Which aspects of your system design are likely to change, and which parts are more stable?

#2 Program to an interface, not to an implementation
How can you write your program to work for the most general type you can? How can you call a vehicle instead of a car? How can you call a moveable object instead of a vehicle?

#3 Prefer composition over inheritance
How can you reference the functionality & class you need via Composition instead of Inheritance?

#4 Delegate, delegate, delegate
How can you "pass the buck" by writing methods to call the "Composed" Class?

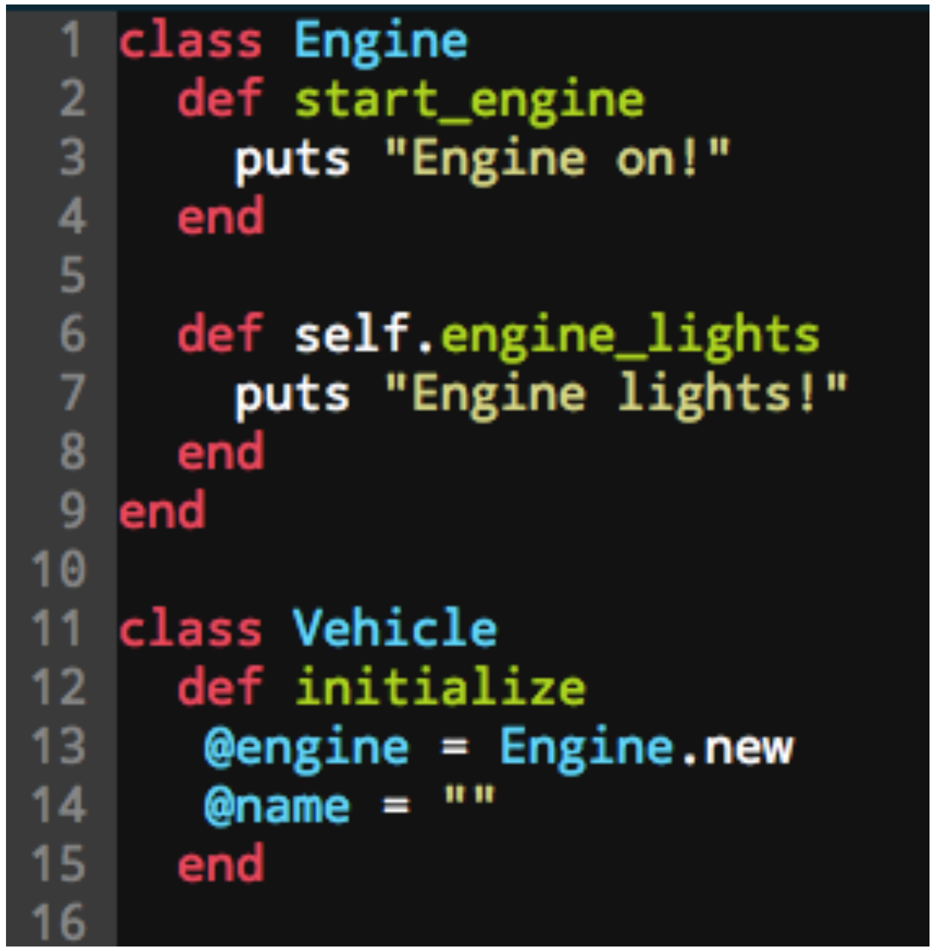
Bonus from *Design Patterns in Ruby*: #5 You ain't gonna need it (YAGNI).
Do you need to write this feature or design in flexibility ***right now****? If not - don't*



```ruby
class Engine
  def start_engine
    puts "Engine on!"
  end

  def self.engine_lights
    puts "Engine lights!"
  end
end

class Vehicle
  def initialize
   @engine = Engine.new
   @name = ""
  end

  def print_license_plate
    puts self.id
  end

  def start_engine
    @engine.start_engine
  end
end

class MotorBoat < Vehicle
  def print_license_plate
    puts "Boat License"
  end

  def name_the_boat(name)
    @name = name
    puts @name
  end
end

henrys_motorboat = MotorBoat.new
henrys_motorboat.start_engine
henrys_motorboat.print_license_plate
henrys_motorboat.name_the_boat("The Henricus IV")


```

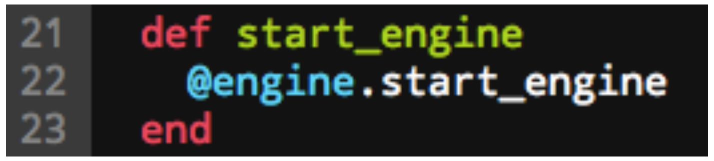
#5 Composition Cont'd

Composition is, in lieu of strict Inheritance, equipping your objects with *references* to other objects - namely, objects that supply the functionality that we need. Because the functionality is encapsulated in these other objects, we can call on it from whichever class needs that functionality. In short, we try to avoid saying that an object is a *kind of something* and instead say that it *has* something.



```ruby
class Engine
  def start_engine
    puts "Engine on!"
  end

  def self.engine_lights
    puts "Engine lights!"
  end
end

class Vehicle
  def initialize
   @engine = Engine.new
   @name = ""
  end

```

You can then use the principle of **Delegation** to access the referenced objects via your own active class. For example,



```ruby
  def start_engine
    @engine.start_engine
  end
```

What is the other -- **less effective** -- way to solve this problem, using **Inheritance**? Well, you might have a Movable Object Class, a Vehicle class and Car and Airplane classes beneath that - with the Engine methods inside "Vehicle", "def engine_stop" etc.

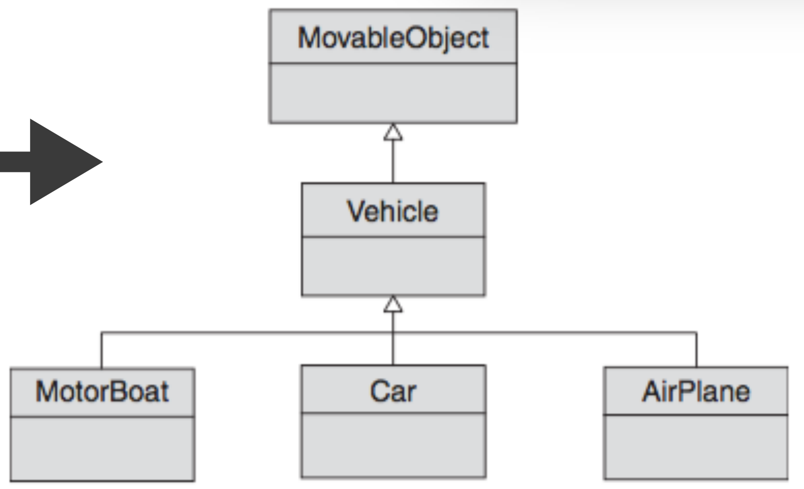


Figure 1-1 Getting the maximum mileage out of inheritance

#6 Polymorphism… But still not sure when to use it?

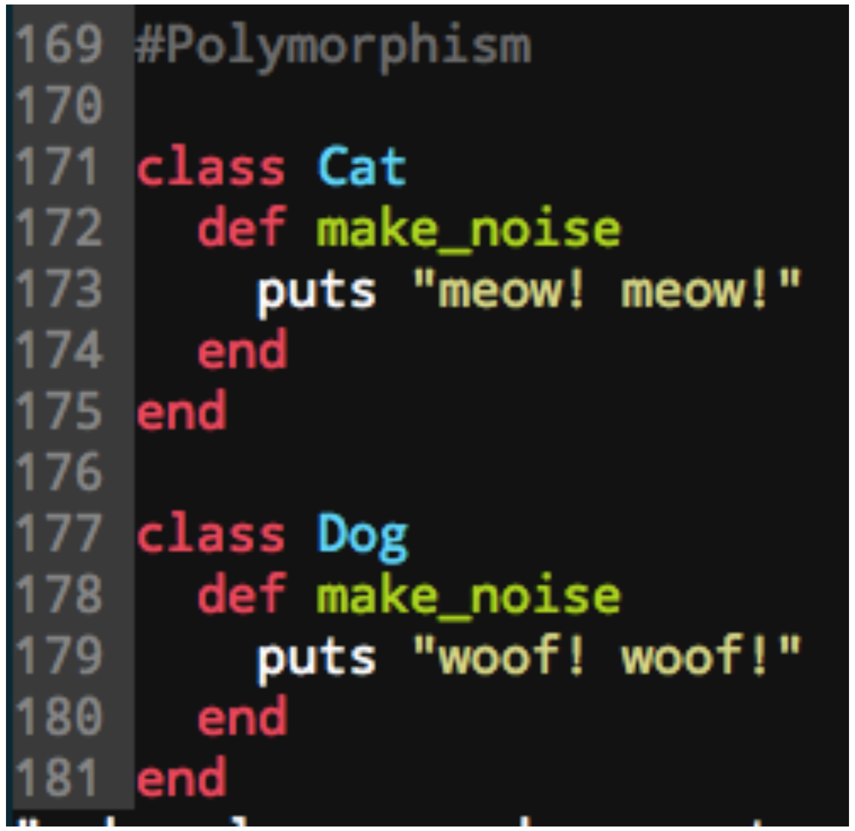
An example of Polymorphism in the non-programming world is a simple **button**. Pressing a button is always the same **interface**; what happens afterwards depends on what the button is wired up to do!

Polymorphism is defined by the dictionary as "the condition (ism) of occurring in several (poly) different forms (morphi)".

In Object-Oriented Programming Polymorphism is sending the **same message** to **different objects** and getting **different results**.

In the example here, I can call the **make_noise** method on both classes and get different results.

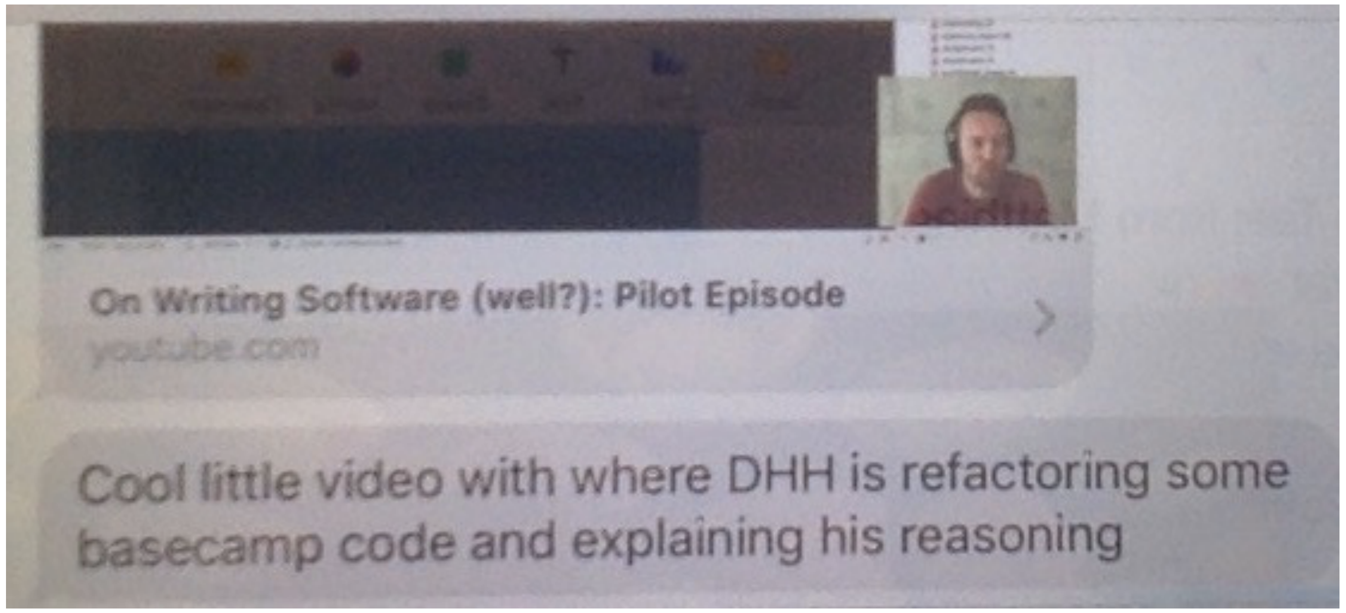
Polymorphism can be used in **combination** with **Inheritance**. For example, I can inherit and **overwite** a method from the class I inherited it from.



```
#Polymorphism

class Cat
  def make_noise
    puts "meow! meow!"
  end
end

class Dog
  def make_noise
    puts "woof! woof!"
  end
end
```

#7 DHH Video Talk from Matthias: it's not rocket science; move at a normal pace; give yourself the time and energy, and be thoughtful

- His example: maybe change **find_and_create** to **create_and_find** and then extract that out to Rails…

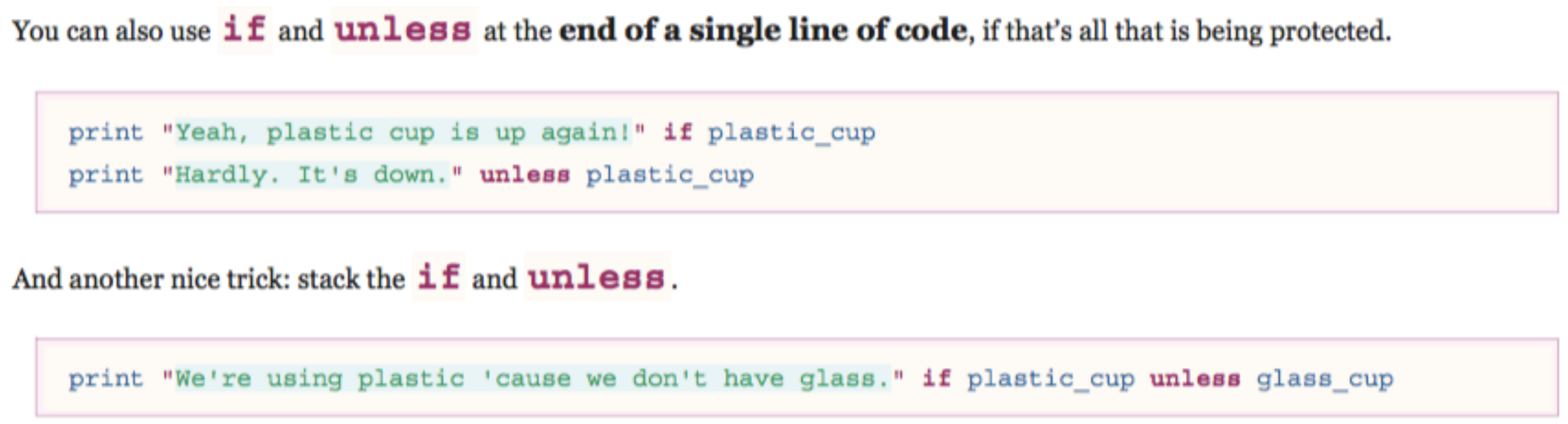


On Writing Software (well?): Pilot Episode
youtube.com

Cool little video with where DHH is refactoring some basecamp code and explaining his reasoning

Grab Bag of Reminders

- WEBrick - now included in the language itself!
- What happens when you call a method on a variable?You pass a **message** *from* the method *to* the variable object
- The defining syntax of a Regular Expression is two "slashes" surrounding the expression: / / e.g. /find this word/
- a = %w[ant bee cat dog elk] This is instead of a = ['ant', 'bee', 'cat', 'dog', 'elk']



You can also use `if` and `unless` at the **end of a single line of code**, if that's all that is being protected.

```
print "Yeah, plastic cup is up again!" if plastic_cup
print "Hardly. It's down." unless plastic_cup
```

And another nice trick: stack the `if` and `unless`.

```
print "We're using plastic 'cause we don't have glass." if plastic_cup unless glass_cup
```