The Ruby Object Model

a presentation for the Nashville Software School

24 June, 2013

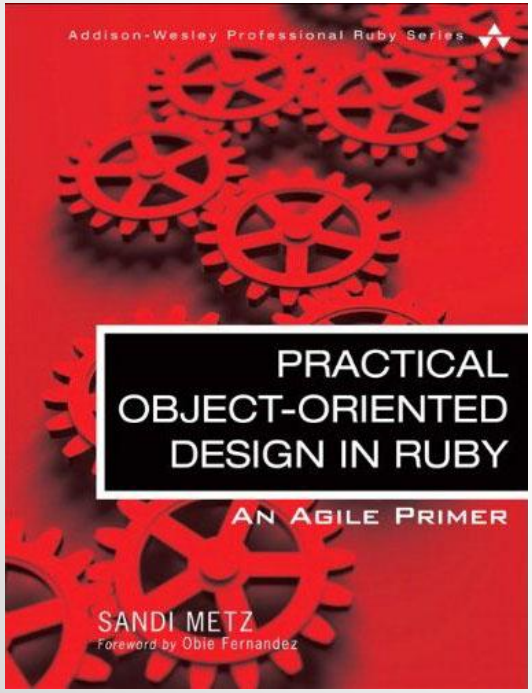
WHOAMI?

- Max Beizer => jr. developer @ Centresource
- NSS Cohort One graduate
- maxbeizer on:
  - twitter
  - github
  - irc

Credit Where Credit is Due...

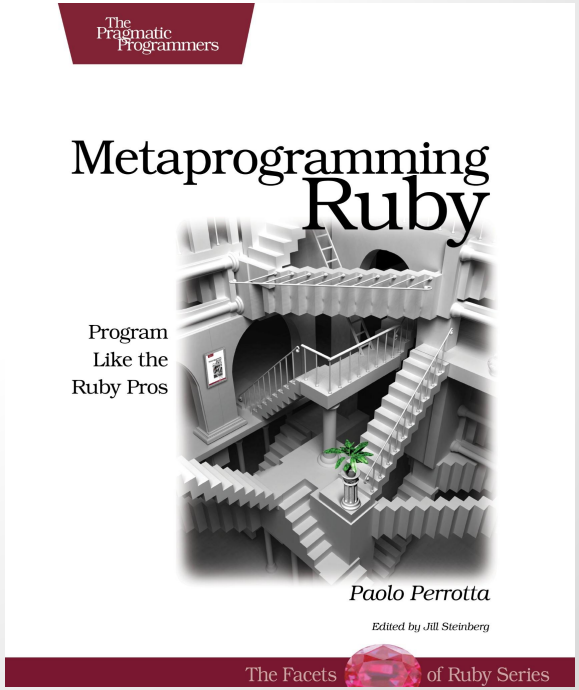
Practical Object-Oriented Design in Ruby

Sandi Metz



Metaprogramming Ruby

Paolo Perrotta



Credit Where Credit is Due...

elizabrocksoftware.com

Ruby and Objects

In Ruby, everything is an object.*

* except when it isn't

```
$irb
1.9.3p362 :001 > "Hello, World!"
 => "Hello, World!"
1.9.3p362 :002 > _.class
 => String
1.9.3p362 :003 > String.superclass
 => Object
1.9.3p362 :004 > Object.superclass
 => BasicObject
1.9.3p362 :005 >
```

see also: ancestors

Every Object "Returns to the Source"

BasicObject

Why do I care?

```
$irb
1.9.3p362 :001 > salutations = "Hello, World!"
 => "Hello, World!"
1.9.3p362 :002 > salutations.foo
NoMethodError: undefined method `foo' for "Hello, World!":String
        from (irb):2
        from /Users/maxbeizer/.rvm/rubies/ruby-1.9.3-p362/bin/irb:16:in `<main>'
1.9.3p362 :003 > salutations.length
 => 13
1.9.3p362 :004 > salutations.reverse
 => "!dlroW ,olleH"
```

Why do I care?

```
$irb
1.9.3p362 :001 > salutations = "Hello, World!"
 => "Hello, World!"
1.9.3p362 :002 > salutations.foo
NoMethodError: undefined method `foo' for "Hello, World!":String
        from (irb):2
        from /Users/maxbeizer/.rvm/rubies/ruby-1.9.3-p362/bin/irb:16:in `<main>'
1.9.3p362 :003 > salutations.length
 => 13
1.9.3p362 :004 > salutations.reverse
 => "!dlroW ,olleH"
```

Why does salutations respond to length and reverse, but not foo?

Remember:

```
$irb
1.9.3p362 :001 > "Hello, World!"
 => "Hello, World!"
1.9.3p362 :002 > _.class
 => String
1.9.3p362 :003 > String.superclass
 => Object
1.9.3p362 :004 > Object.superclass
 => BasicObject
1.9.3p362 :005 >
```

ruby-doc.org -- String

#length
#lines
#ljust
#lstrip
#lstrip!
#match
#next
#next!
#oct
#ord
#partition
#prepend
#replace
#reverse
#reverse!

ruby-doc.org -- String

#length
#lines
#ljust
#lstrip
#lstrip!
#match
#next
#next!
#oct
#ord
#partition
#prepend
#replace
#reverse
#reverse!

no love for foo

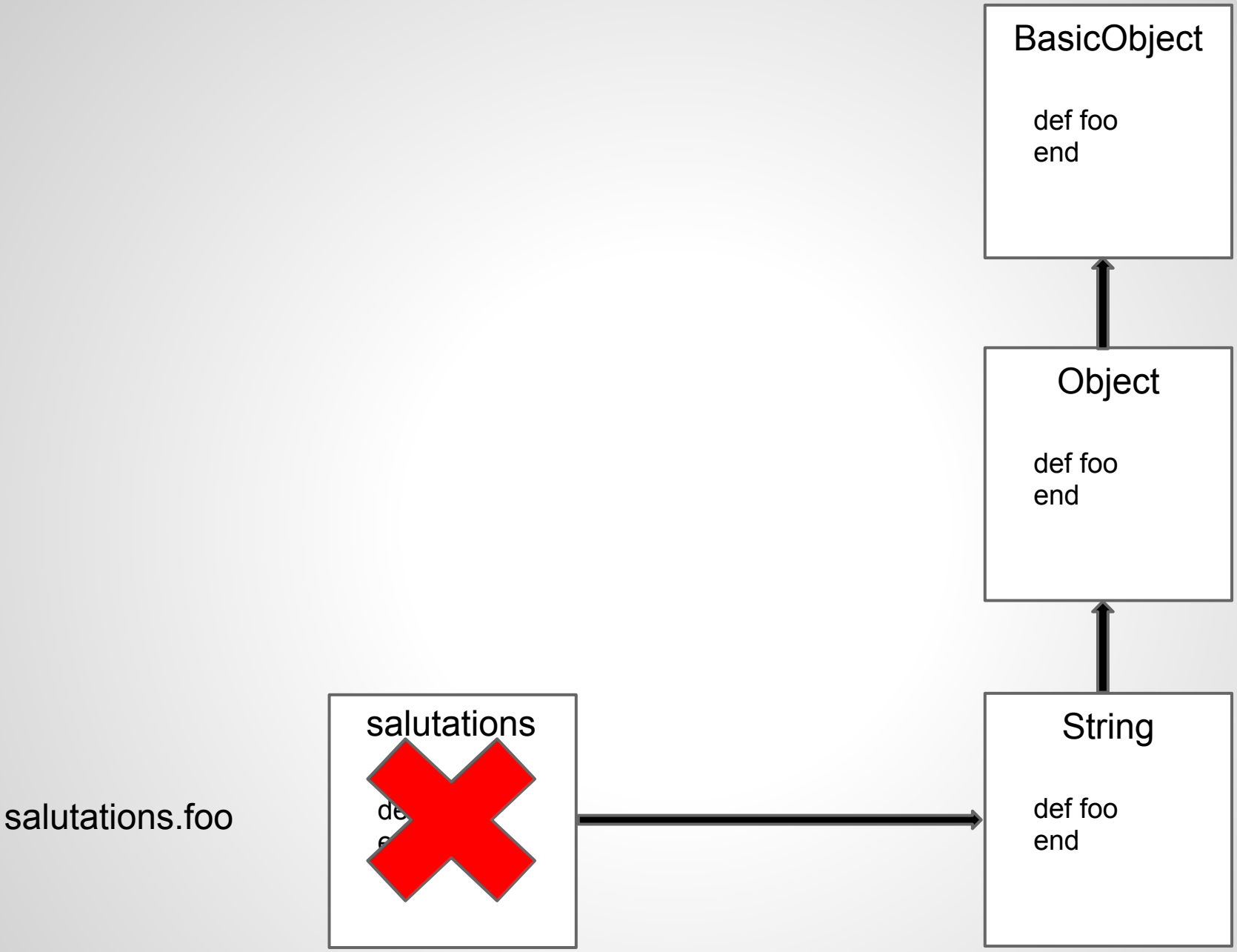
:(

salutations.foo

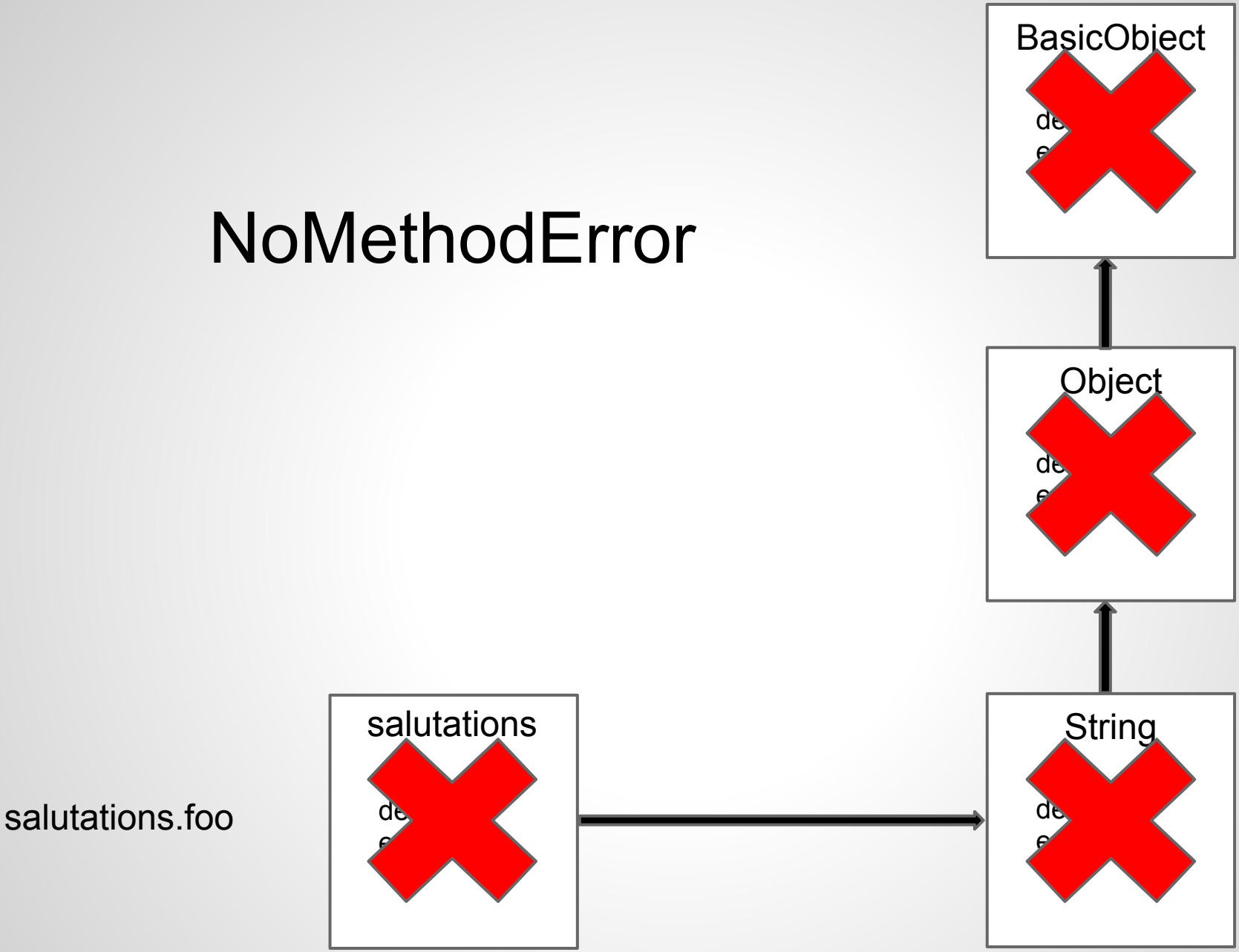
salutations

```
def foo
end
```

salutations.foo


BasicObject
def foo
end
Object
def foo
end
salutations
String
def foo
end
salutations.foo

NoMethodError



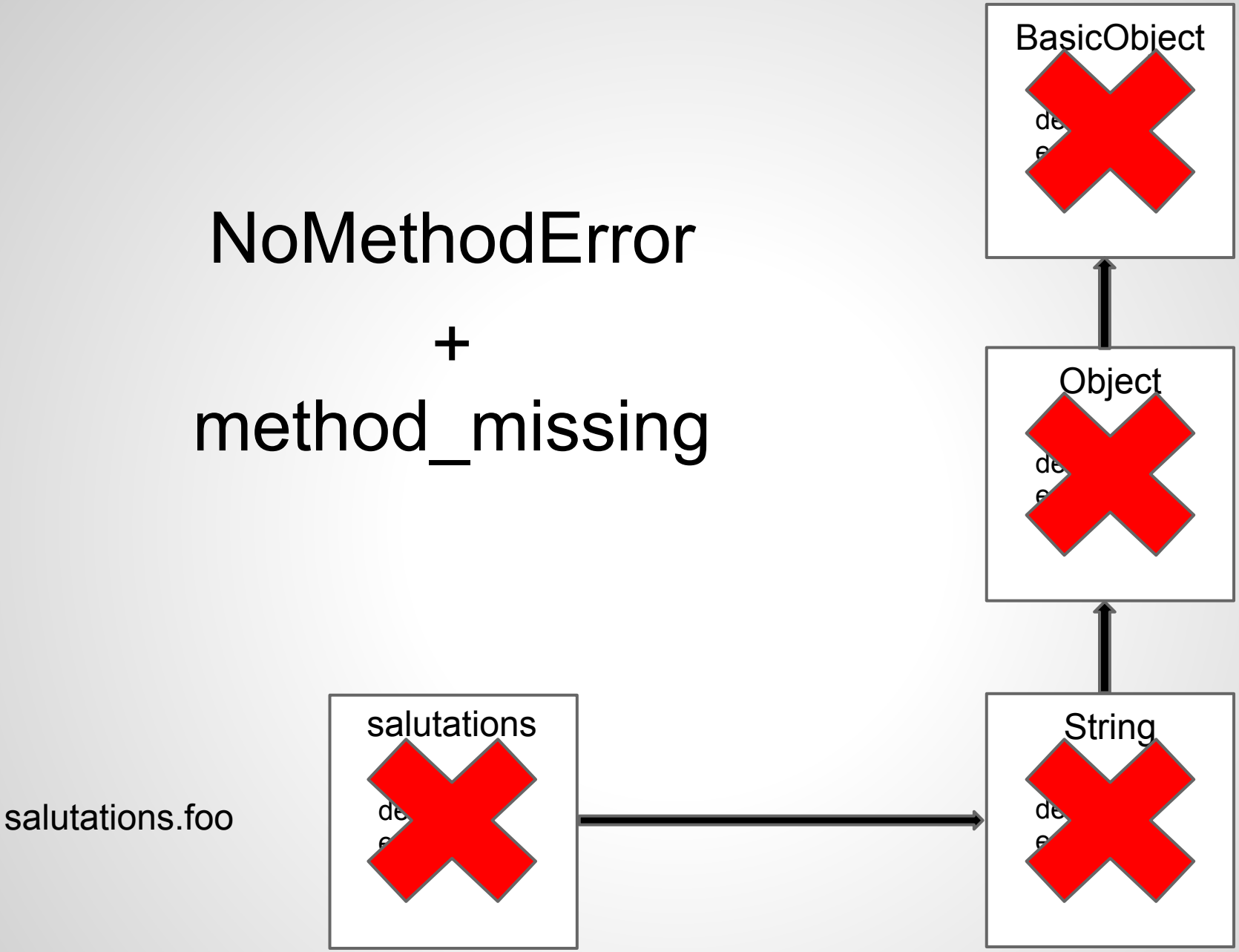
Impress Your Friends:

Lookup what method_missing does.

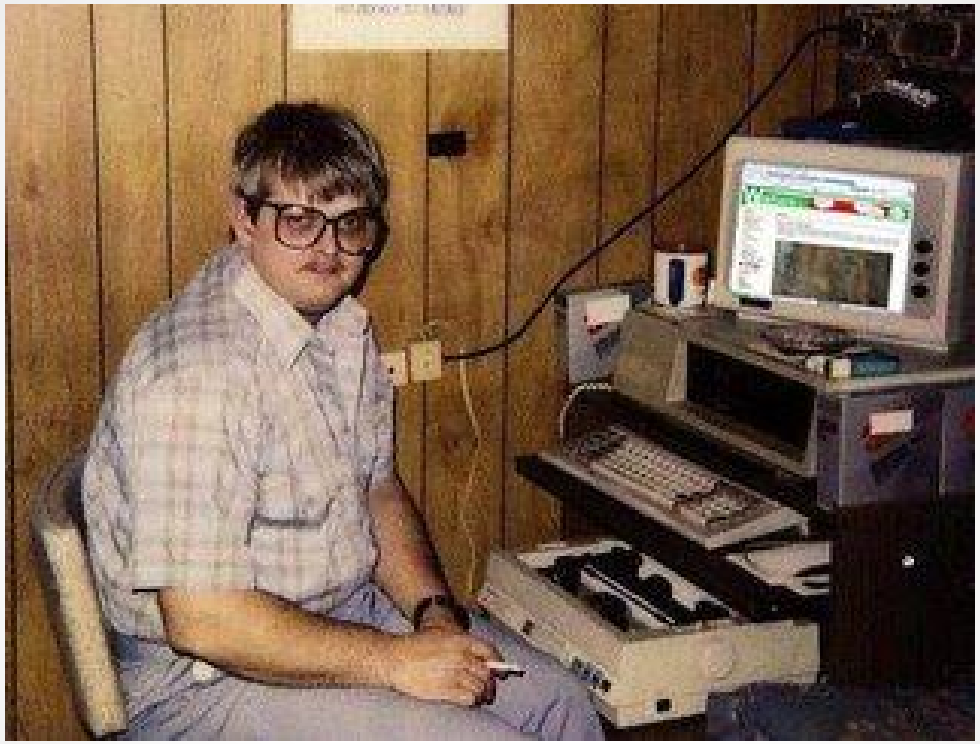
NoMethodError

+

method_missing



This Guy Says:



"Doesn't this make Ruby soooo much slower than #{my favorite compiled language}?"

My Retort:

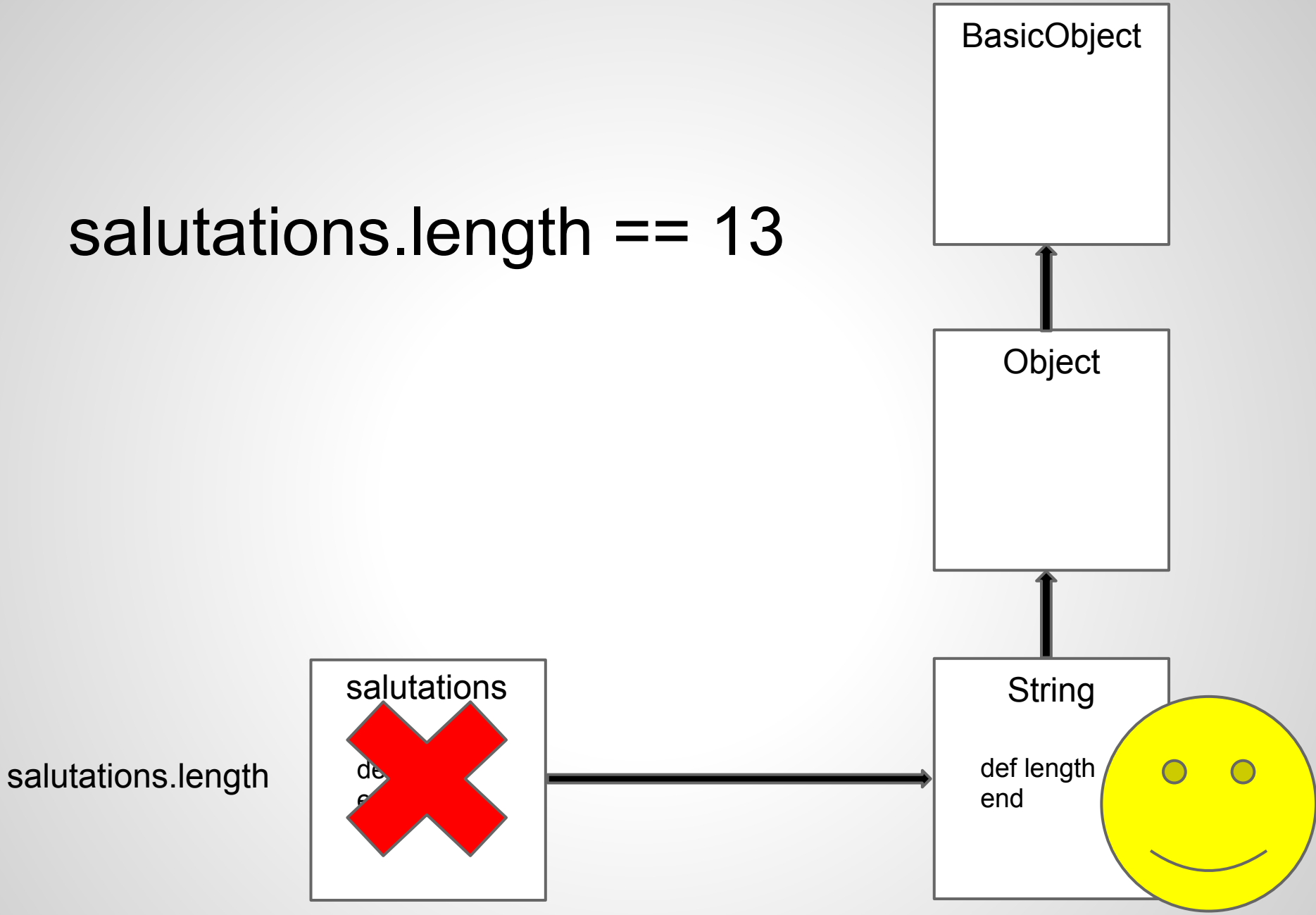
Moving on...

salutations.length

salutations

```
def length
end
```

salutations.length == 13



Uncle Max's Story Time...

Inheritance

or how I learned to stop worrying and love method lookup

A Simple Ruby Class: You

```ruby
class NssStudent
  attr_accessor :name

  def initialize(name)
    @name = name
  end
end
```

It's Alive!!!!!1

```
 git:[master]$irb
1.9.3-p362 :001 > load 'example.rb'
 => true
1.9.3-p362 :002 > dane = NssStudent.new("Dane")
 => #<NssStudent:0x007fada31e3310 @name="Dane">
1.9.3-p362 :003 > dane.name
 => "Dane"
1.9.3-p362 :004 > lisa = NssStudent.new("Lisa")
 => #<NssStudent:0x007fada31cee10 @name="Lisa">
1.9.3-p362 :005 > lisa.name
 => "Lisa"
```

you == NSS Student

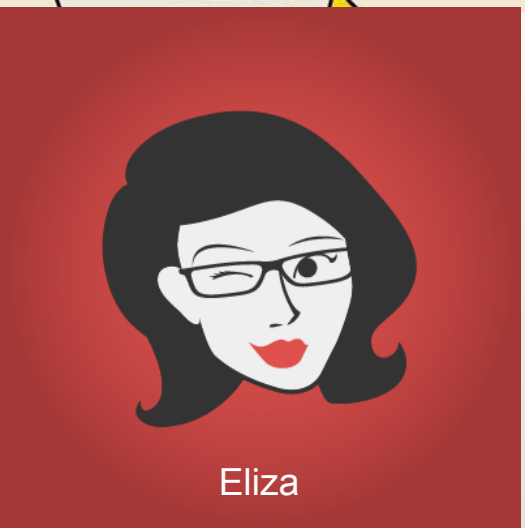
me != NSS Student

eliza != NSS Student

you == NSS Student

me == junior dev

eliza == accomplished professional



We All Have:

- name
- experience
- job(?)

Naïve Implementation

(A.K.A. my middle names)

D.R.Y. ?

```ruby
class NssStudent
  attr_reader :name, :job, :experience

  def initialize(name, job, experience)
    @name = name
    @job = job
    @experience = experience
  end
end

class JrDev
  attr_reader :name, :job, :experience

  def initialize(name, job, experience)
    @name = name
    @job = job
    @experience = experience
  end
end

class AccomplishedProfessional
  attr_reader :name, :job, :experience

  def initialize(name, job, experience)
    @name = name
    @job = job
    @experience = experience
  end
end
```

Naïve Implementation

(A.K.A. my middle names)

Sopping wet

```
class NssStudent
  attr_reader :name, :job, :experience

  def initialize(name, job, experience)
    @name = name
    @job = job
    @experience = experience
  end
end

class JrDev
  attr_reader :name, :job, :experience

  def initialize(name, job, experience)
    @name = name
    @job = job
    @experience = experience
  end
end

class AccomplishedProfessional
  attr_reader :name, :job, :experience

  def initialize(name, job, experience)
    @name = name
    @job = job
    @experience = experience
  end
end
```

Naïve Implementation

(A.K.A. my middle names)

What if the requirements changed and "experience" were henceforth to be known as "awesome_points" ...?

What about specialization? Why have three classes that do the same thing?

```ruby
class NssStudent
  attr_reader :name, :job, :experience

  def initialize(name, job, experience)
    @name = name
    @job = job
    @experience = experience
  end
end

class JrDev
  attr_reader :name, :job, :experience

  def initialize(name, job, experience)
    @name = name
    @job = job
    @experience = experience
  end
end

class AccomplishedProfessional
  attr_reader :name, :job, :experience

  def initialize(name, job, experience)
    @name = name
    @job = job
    @experience = experience
  end
end
```

Inheritance Is All About Commonality

- Share common methods, attributes to keep it DRY
- Generalization vs. Specialization
- Abstract vs. Concrete

use the object model/lookup

```ruby
class NssStudent
  attr_reader :name, :job, :experience

  def initialize(name, job, experience)
    @name = name
    @job = job
    @experience = experience
  end

  def bang_out_code
    "stack overflow copy and paste"
  end

  def learn_a_lot
    "give me knowledge!!!"
  end
end

class JrDev
  attr_reader :name, :job, :experience

  def initialize(name, job, experience)
    @name = name
    @job = job
    @experience = experience
  end

  def bang_out_code
    "ask somebody, then stack overflow copy and paste"
  end

  def bumble_about
    "what was that thing I learned at NSS?"
  end
end
```

```ruby

class AccomplishedProfessional
  attr_reader :name, :job, :experience

  def initialize(name, job, experience)
    @name = name
    @job = job
    @experience = experience
  end

  def bang_out_code
    "make miracles happen"
  end

  def make_the_big_bucks
    "mucho " * @experience + "dinero"
  end
end
```

```ruby
class NssStudent
  attr_reader :name, :job, :experience

  def initialize(name, job, experience)
    @name = name
    @job = job
    @experience = experience
  end

  def bang_out_code
    "stack overflow copy and paste"
  end

  def learn_a_lot
    "give me knowledge!!!"
  end
end

class JrDev
  attr_reader :name, :job, :experience

  def initialize(name, job, experience)
    @name = name
    @job = job
    @experience = experience
  end

  def bang_out_code
    "ask somebody, then stack overflow copy and paste"
  end

  def bumble_about
    "what was that thing I learned at NSS?"
  end
end
```

```ruby

class AccomplishedProfessional
  attr_reader :name, :job, :experience

  def initialize(name, job, experience)
    @name = name
    @job = job
    @experience = experience
  end

  def bang_out_code
    "make miracles happen"
  end

  def make_the_big_bucks
    "mucho " * @experience + "dinero"
  end
end
```

Specialization

Different Methods

Same Method Name, Different Result

```ruby
class NssStudent
  attr_reader :name, :job, :experience

  def initialize(name, job, experience)
    @name = name
    @job = job
    @experience = experience
  end

  def bang_out_code
    "stack overflow copy and paste"
  end

  def learn_a_lot
    "give me knowledge!!!"
  end
end

class JrDev
  attr_reader :name, :job, :experience

  def initialize(name, job, experience)
    @name = name
    @job = job
    @experience = experience
  end

  def bang_out_code
    "ask somebody, then stack overflow copy and paste"
  end

  def bumble_about
    "what was that thing I learned at NSS?"
  end
end
```

```ruby

class AccomplishedProfessional
  attr_reader :name, :job, :experience

  def initialize(name, job, experience)
    @name = name
    @job = job
    @experience = experience
  end

  def bang_out_code
    "make miracles happen"
  end

  def make_the_big_bucks
    "mucho " * @experience + "dinero"
  end
end
```

Generalization

NssStudent is a ...?

JrDev is a ...?

AccomplishedProfessional is a ...?

```ruby
class Developer
  attr_reader :name, :job, :experience

  def initialize(name, job, experience)
    @name = name
    @job = job
    @experience = experience
  end

  def bang_out_code
    "stack overflow copy and paste"
  end
end

class NssStudent < Developer
  def learn_a_lot
    "give me knowledge!!!"
  end
end

class JrDev < Developer
  def bang_out_code
    "ask somebody, then " + super
  end

  def bumble_about
    "what was that thing I learned at NSS?"
  end
end

class AccomplishedProfessional < Developer
  def bang_out_code
    "make miracles happen"
  end

  def make_the_big_bucks
    "mucho " * @experience + "dinero"
  end
end
```

it fits!!!!!!!1

win

```ruby
class Developer
  attr_reader :name, :job, :experience

  def initialize(name, job, experience)
    @name = name
    @job = job
    @experience = experience
  end

  def bang_out_code
    "stack overflow copy and paste"
  end
end

class NssStudent < Developer
  def learn_a_lot
    "give me knowledge!!!"
  end
end

class JrDev < Developer
  def bang_out_code
    "ask somebody, then " + super
  end

  def bumble_about
    "what was that thing I learned at NSS?"
  end
end

class AccomplishedProfessional < Developer
  def bang_out_code
    "make miracles happen"
  end

  def make_the_big_bucks
    "mucho " * @experience + "dinero"
  end
end
```

shared, generalized code extracted/abstracted

specialized code

call to super invokes the method in the super class

overrides the superclass

The Abstract Class in Inheritance

part of the lookup chain

generally never to be instantiated on its own

Inheritance: is it right for you?

Is it right for your problem set?

Ask yourself: is this an *isa?*

Inheritance: is it right for you?

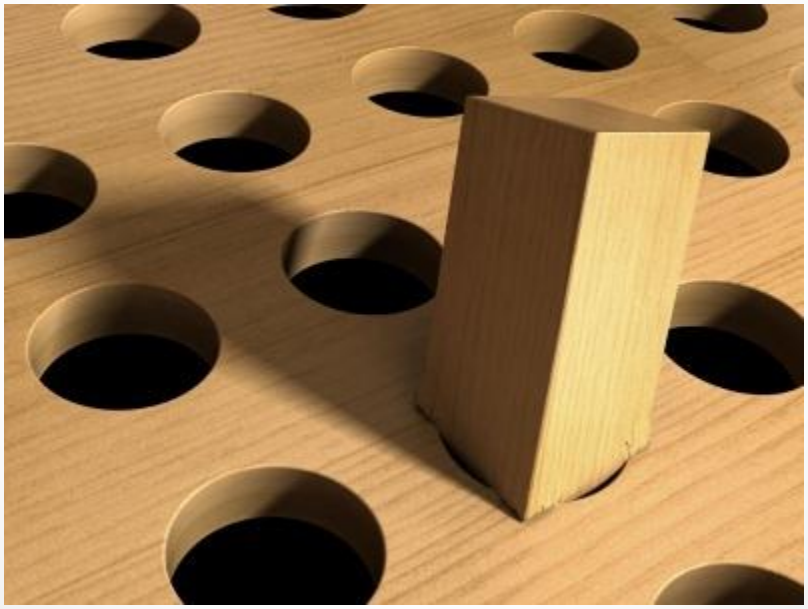
Is it right for your problem set?

developer-or

caveat ~~emptor~~

Inheritance: is it right for you?

Is it right for your problem set?



put another way: not everything is a nail

Rails-Colored Glasses

Way back when I was first learning Ruby

(pause for laughter)

Modules
were all like:

Nowadays

Modules are all like:

SoftWhere Co. App.

```ruby
class Executive < ActiveRecord::Base
  #heaps of big important stuff omitted
  def attend_meeting_at(meeting_place)
    self.location = meeting_place
  end

  def answer_email
    self.frustration_level += 1
  end

  def speak_in_front_of_large_groups(size)
    self.confidence_level = size / 1
  end
end

class MiddleManagement < ActiveRecord::Base
  #lots of mgmt stuff omitted
  def attend_meeting_at(meeting_place)
    self.location = meeting_place
  end

  def answer_email
    self.frustration_level += 1
  end

  def speak_in_front_of_large_groups(size)
    self.confidence_level = size / 2
  end
end
```

```ruby
class Developer < ActiveRecord::Base
  #general nerdery omitted
  def attend_meeting_at(meeting_place)
    self.location = meeting_place
  end

  def answer_email
    self.frustration_level += 1
  end

  def speak_in_front_of_large_groups(size)
    self.confidence_level = size / 100
  end
end
```

SoftWhere Co. App.

SoftWhere Co. App.

```ruby
class Executive < ActiveRecord::Base
  #heaps of big important stuff omitted
  include Officeable

  def amount_of_unease
    1
  end
end

class MiddleManagement < ActiveRecord::Base
  #lots of mgmt stuff omitted
  include Officeable

  def amount_of_unease
    2
  end
end

class Developer < ActiveRecord::Base
  #general nerdery omitted
  include Officeable

  def amount_of_unease
    100
  end
end
```

```ruby
#lib/officeable.rb
module Officeable
  def attend_meeting_at(meeting_place)
    self.location = meeting_place
  end

  def answer_email
    self.frustration_level += 1
  end

  def speak_in_front_of_large_groups(size)
    self.confidence_level = size / amount_of_unease
  end
end
```

SoftWhere Co. App.

```ruby
class Executive < ActiveRecord::Base
  #heaps of big important stuff omitted
  include Officeable

  def amount_of_unease
    1
  end
end

class MiddleManagement < ActiveRecord::Base
  #lots of mgmt stuff omitted
  include Officeable

  def amount_of_unease
    2
  end
end

class Developer < ActiveRecord::Base
  #general nerdery omitted
  include Officeable

  def amount_of_unease
    100
  end
end
```

```ruby
#lib/officeable.rb
module Officeable
  def attend_meeting_at(meeting_place)
    self.location = meeting_place
  end

  def answer_email
    self.frustration_level += 1
  end

  def speak_in_front_of_large_groups(size)
    self.confidence_level = size / amount_of_unease
  end
end
```

Officeable's methods become instance methods for the Executive, MiddleManagement, and Developer models.

win

SoftWhere Co. App.

```ruby
class Executive < ActiveRecord::Base
  #heaps of big important stuff omitted
  include Officeable

  def amount_of_unease
    1
  end
end

class MiddleManagement < ActiveRecord::Base
  #lots of mgmt stuff omitted
  include Officeable

  def amount_of_unease
    2
  end
end

class Developer < ActiveRecord::Base
  #general nerdery omitted
  include Officeable

  def amount_of_unease
    100
  end
end
```

```ruby
#lib/officeable.rb
module Officeable
  def attend_meeting_at(meeting_place)
    self.location = meeting_place
  end

  def answer_email
    self.frustration_level += 1
  end

  def speak_in_front_of_large_groups(size)
    self.confidence_level = size / amount_of_unease
  end
end
```

All three methods have a single, authoritative place where they live. That's D.R.Y.

win

SoftWhere Co. App.

```ruby
#lib/officeable.rb
module Officeable
  def attend_meeting_at(meeting_place)
    self.location = meeting_place
  end

  def answer_email
    self.frustration_level += 1
  end

  def speak_in_front_of_large_groups(size)
    self.confidence_level = size / amount_of_unease
  end
end
```

What if I create a new class that needs to include Officeable but I forget about or don't know about 'amount_of_unease' ???

not so much

SoftWhere Co. App.

```ruby
#lib/officeable.rb
module Officeable
  def attend_meeting_at(meeting_place)
    self.location = meeting_place
  end

  def answer_email
    self.frustration_level += 1
  end

  def speak_in_front_of_large_groups(size)
    self.confidence_level = size / amount_of_unease
  end

  def amount_of_unease
    raise TemplateError, 'The Officeble module requires the' +
                         'including class to define an' +
                         'amount_of_unease method'
  end
end
```

The next developer receives a helpful error message. Novel concept.

win

SoftWhere Co. App.

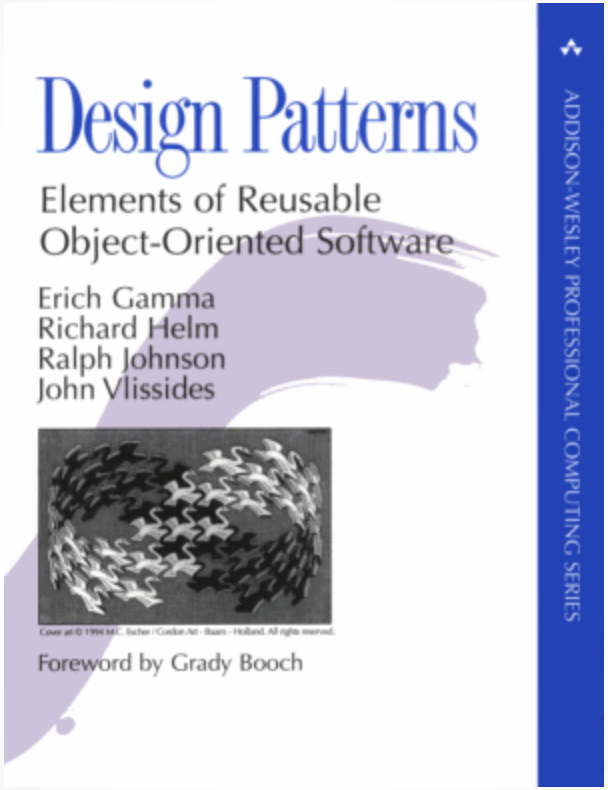
```ruby
#lib/officeable.rb
module Officeable
  def attend_meeting_at(meeting_place)
    self.location = meeting_place
  end

  def answer_email
    self.frustration_level += 1
  end

  def speak_in_front_of_large_groups(size)
    self.confidence_level = size / amount_of_unease
  end
end
```

Side note: you'll probably want to add a guard to make sure amount_of_unease is not zero.

A Rails Template Pattern



Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides

Make sure `lib` is in the load path

```
#config/application.rb
config.autoload_paths += %W(#{config.root}/extras)
```

image credits:

- http://www.newgre.org/admissions/applying-doctoral-programs-it%E2%80%99s-match/attachment/square-peg-in-a-round-hole/
- amazon.com
- elizabrocksoftware.com
- http://www.hsxdude.com/
- http://pragdave.pragprog.com/pragdave/2007/05/rails_is_love.html
- http://creepypasta.wikia.com/wiki/File:Creepy-van.jpg
- http://juliasetssail.blogspot.com/2010_04_01_archive.html
- http://knowyourmeme.com/memes/haters-gonna-hate
- http://blog.ausweb.com.au/system-administrator-appreciation-day/
- http://worldtruth.tv/philosophy-the-matrix-return-to-the-source/