DSL in Ruby through Metaprogramming and Pattern Matching

Wilhelm Chung

What are we going to talk about?

- What are domain specific languages
- Internal DSL example: Dwemthy's Array
- External DSL example: Pattern Matching for Logo

The Right Level of Abstraction

Common DSLs

- Regular expressions
- CSS
- HTML
- SQL

Internal DSLs

“I wish I could make the language do [this]”

Dwemthy's Array

A Dungeon Master wants to create Creatures for his dungeon

TeethDeer from Dwemthy's Array

```
class TeethDeer < Creature
  life 655
  strength 2
  charisma 44
end
```

Spawn the TeethDeer

```
td = TeethDeer.new

td.life() # 655
td.boost_life() # @life += 10

td.strength() # 2
td.boost_strength() # @strength += 10
```

TeethDeer with bombs

```
class TeethDeer < Creature
  traits :bombs
  life 10
  strength 2
  charisma 44
  bombs 3
end
```

Spawn TeethDeer with bombs

```
td = TeethDeer.new
td.bombs() # <= 3
td.boost_bombs() # @bombs += 10
```

Metaprogramming

Creature Class

```
class Creature
  def self.traits(*names)
    # metaprogramming
  end
end
```

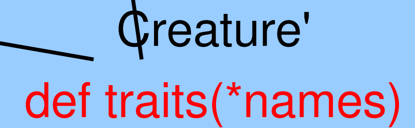
Creature Class

```
class Creature
  def self.traits(*names)
    names.each do |trait|
      # define_method(trait, num) in singleton class
      # define_method(trait)
      # define_method(boost_trait)
    end
  end
end
```

```
def self.traits(*names)
  names.each do |trait|
    # define_method(trait, num) in singleton class
    # define_method(trait)
    # define_method(boost_trait)
  end
end
```

Creature

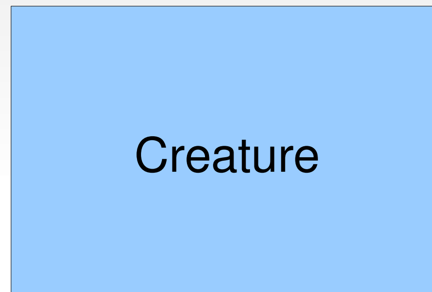
Creature'

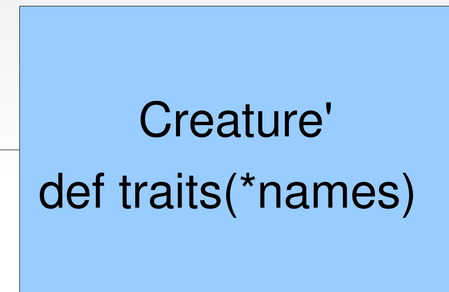


Inherit from Creature Class

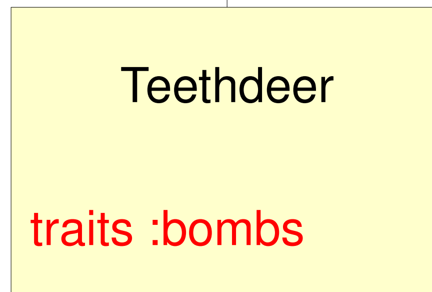
```
class TeethDeer < Creature
  traits :bombs
end
```

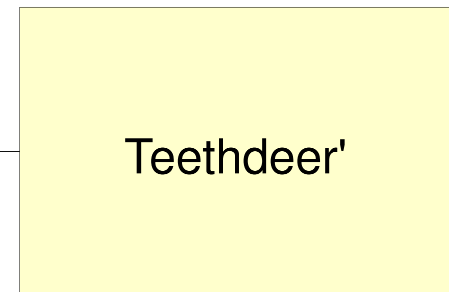
```
def self.traits(*names)
  names.each do |trait|
    # define_method(trait, num) in singleton class
    # define_method(trait)
    # define_method(boost_trait)
  end
end
```



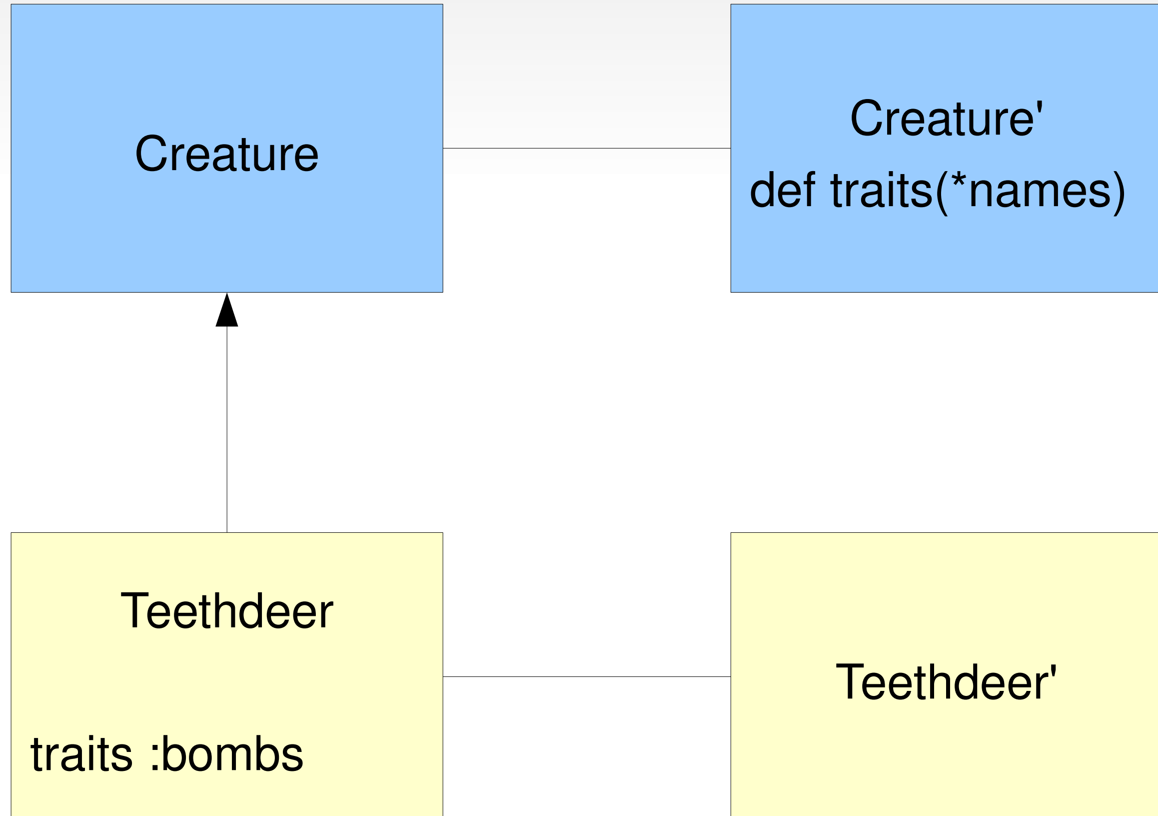


Inherit from Creature when making Teethdeer





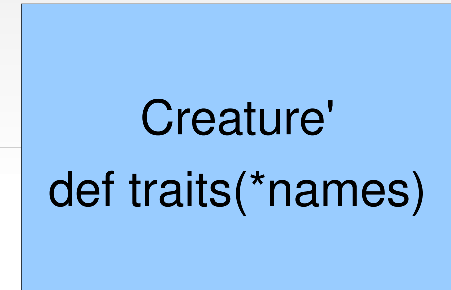
```
def self.traits(*names)
  names.each do |trait|
    # define_method(trait, num) in singleton class
    # define_method(trait)
    # define_method(boost_trait)
  end
end
```

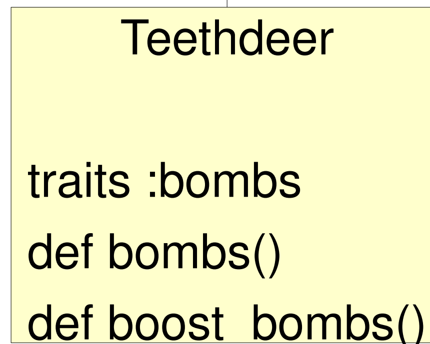


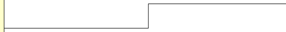
Q: The call to traits in TeethDeer puts bombs() and boost_bombs() in Creature or Teethdeer?

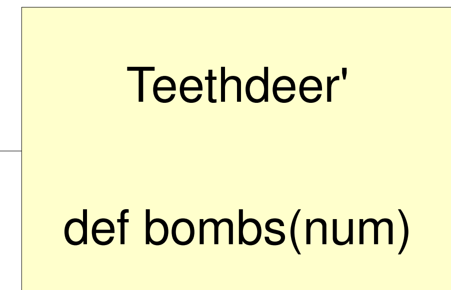
```
def self.traits(*names)
  names.each do |trait|
    # define_method(trait, num) in singleton class
    # define_method(trait)
    # define_method(boost_trait)
  end
end
```

A: Teethdeer









Rails Associations are a DSL

```
class Website < ActiveRecord::Base
  has_many :votes
  has_many :voters, :through => :votes
end

link = Website.find(:first)
link.votes
link.voters
```

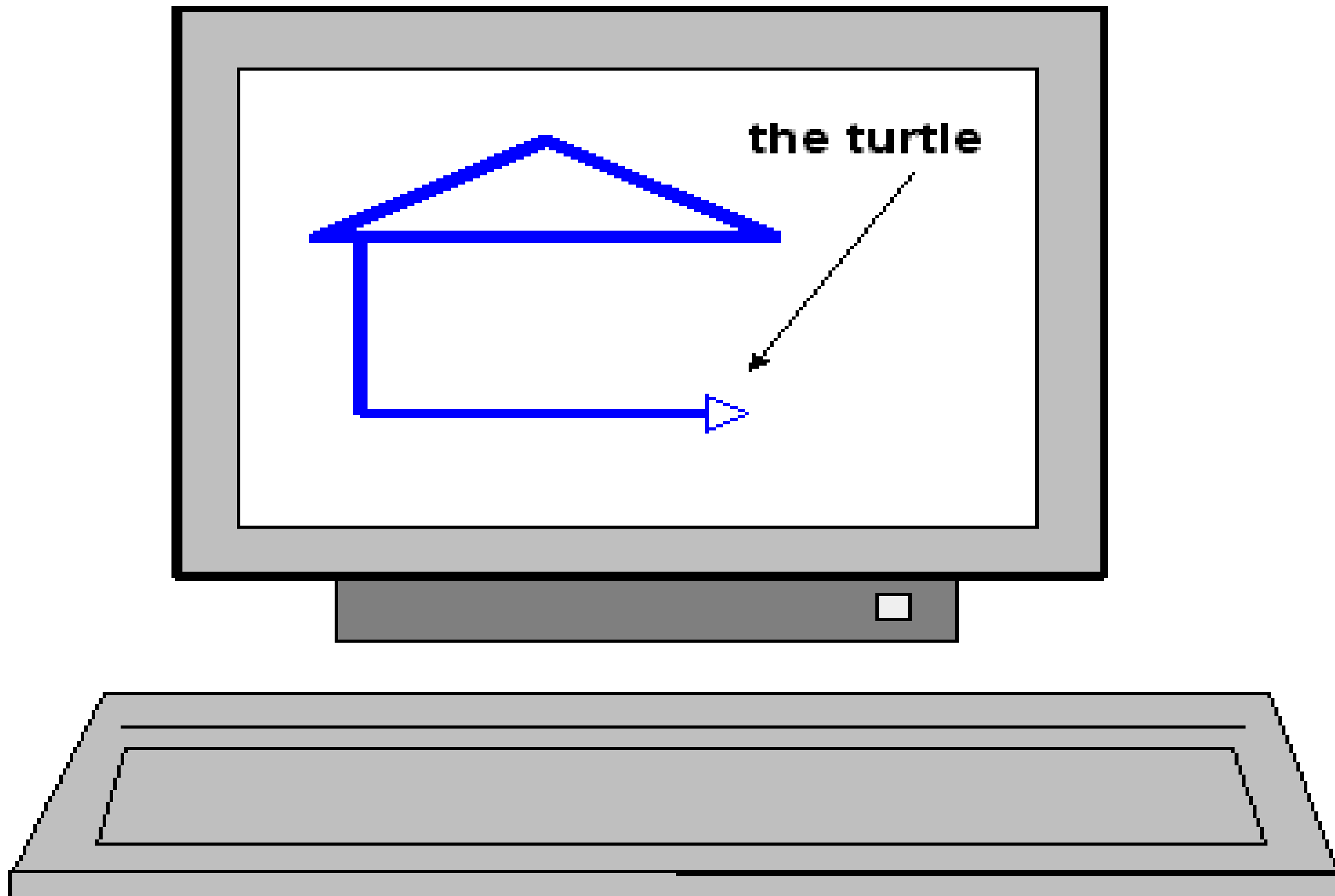
Rails Plugins are a DSL

```
class Website < ActiveRecord::Base
  acts_as_votable
end

link = Website.find(:first)
link.votes
link.voters
```

External DSL

Written in an external language and transformed by an interpreter or compiler

Logo and Turtles



Logo class

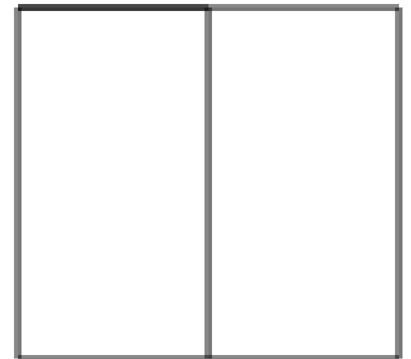
```
class Logo
  def render; end
  def turn(angle); end
  def move(dist); end
  def eval(str); end
end
```

Logo without DSL

```
logo = Logo.new

logo.move(100)

logo.turn(90)

logo.move(100)
```

Logo with DSL

```
logo = Logo.new
logo.eval [:repeat, 4, [:forward, 100, :right, 90],
  :forward, 50, :right, 90, :forward, 100]
puts logo.render
```



Tangent: Fibonacci

fib(0) # 1

fib(1) # 1

fib(2) # fib(1) + fib(0) = 1 + 1 = 2

fib(3) # fib(2) + fib(1) = 2 + 1 = 3

fib(4) # fib(3) + fib(2) = 3 + 2 = 5

fib(5) # fib(4) + fib(3) = 5 + 3 = 8

Tangent: Fibonacci

```
def fib(n)
  return 1 if n == 0 || n == 1
  return fib(n-1) + fib(n-2)
end
```

Pattern Matching in Erlang

```
fib(0) -> 1;
fib(1) -> 1;
fib(N) -> fib(N-1) + fib(N-2).


fib(1). # <= 1
fib(5). # <= 8
```

Pattern Matching in Ruby

```
require 'rubygems'
require 'multi'

multi(:fib, 0)       { 1 }
multi(:fib, 1)       { 1 }
multi(:fib, Integer) { |n| fib(n-1) + fib(n-2) }

fib(1) # <= 1
fib(5)  # <= 8
```

Back to Logo: Parser with Pattern Matching

Let's try building a parser using pattern matching

Move forward

```
run([:forward, 100])

amulti(:run, :forward, Numeric) {|sym, f, rest|
  move(f)
  run(rest)
}
amulti(:run) { }
```

Turn Right

```
run([:forward, 100, :right, 15, :forward 100])

amulti(:run, :forward, Numeric) {|sym, f, rest|
  move(f)
  run(rest)
}
amulti(:run, :right, Numeric)   {|sym, r, rest|
  turn(r)
  run(rest)
}
```

Repeat

```
run([:repeat, 4, [:forward, 100, :right, 15]])

amulti(:run, :repeat, Numeric, Array) do |sym, i, code, rest|
  i.to_i.times{ run(code) }
  run(rest)
end
```

All together

```
amulti(:run, :right, Numeric)   {|sym, r, rest| turn(r) ; run(rest)  }
amulti(:run, :left, Numeric)    {|sym, l, rest| turn(-l) ; run(rest) }
amulti(:run, :forward, Numeric) {|sym, f, rest| move(f) ; run(rest)  }
amulti(:run, :repeat, Numeric, Array) do |sym, i, code, rest|
  i.to_i.times{ run(code) }
  run(rest)
end
amulti(:run) { }
```

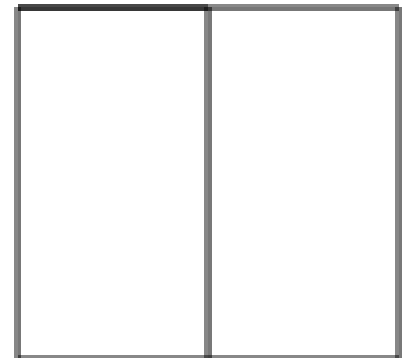
Glossed over a slight detail

```
logo = Logo.new
logo.eval [:repeat, 4, [:forward, 100, :right, 90],
  :forward, 50, :right, 90, :forward, 100]
puts logo.render
```

We used arrays. What if you wanted to eval strings from source files?

Logo with S-expressions

```
logo = Logo.new
logo.eval %q{
  (repeat 4 (forward 100 right 90)
  forward 50 right 90 forward 100)
}
puts logo.render
```



Parsing S-expressions

parse("(repeat 4 (forward 100 right 90)
forward 50 right 90 forward 100)")

[:repeat, 4, [:forward, 100, :right, 90],
:forward, 50, :right, 90, :forward, 100]

Building an S-expression Parser

```
smulti(:parse, /\s+/)    {|c, rest| parse(rest)                    }
smulti(:parse, /\(/)     {|c, rest| @res = ListParser.new(rest)     }
smulti(:parse, /\"/)     {|c, rest| @res = StringParser.new(rest)   }
smulti(:parse, NumberRE) {|c, rest| @res = Number.new(rest, c) }
smulti(:parse, SymbolRE) {|c, rest| @res = Symbol.new(rest, c) }
```

Parsing steps

- “(repeat 4 (forward 100 right 90) forward 50 right 90 forward 100)”
- [:repeat, 4, [:forward, 100, :right, 90], :forward, 50, :right, 90, :forward, 100]
  - logo.move(100); logo.turn(90); logo.move(100)...etc

Links

- http://whytheluckystiff.net/articles/seeingMetaclasses
- http://poignantguide.net/dwemthy/
- http://www.artima.com/rubycs/articles/patterns_sexp