Metaprogramming in Ruby

Created By Kristen Mills

What is Metaprogramming?

The Basics

Symbols

- Kind of like strings
- Immutable
- Not garbage collected

Blocks

- That do…end thing
  - or curly braces if on one line
- One of the 3 types of closures

Monkey Patching

Monkey Patching by example

```ruby
class String
  def my_method
    "my_method is a cool method"
  end
end

'abc'.my_method # => "my_method is a cool method"
```

Refinements

- Allows you to monkey patch in a given scope

Aliasing method

- alias method_1, method_2
- alias_method :method_1, :method_2

Defining Methods

- def method_name(params)
- define_method(name, &block)
- define_singleton_method(name, &block)

Instance Variables

- instance_variable_set(name, value)
- instance_variable_get(name)

attr_accessor

```ruby
def attr_accessor(*args)
  args.each do |arg|
    define_method(arg) do
      instance_variable_get(:"@#{arg}")
    end
    define_method(:"#{arg}=") do |value|
      instance_variable_set(:"@#{arg}", value)
    end
  end
end
```

Eigenclasses!

Wait, where does that method go?

Accessing the Eigenclass

```
eigenclass = class << obj

class << an_object
  # your code here
end
```

Before you ask

- Yes, eigenclasses have Eigenclasses

Calling Methods

- send(name, *args)

Removing Methods

- remove_method(name)
- undef_method(name)

Rails Black Magic

(aka Method Missing)

Method Missing

- method_missing(method, *args, &block)

Domain Specific Languages

Evaluating Strings/Blocks

- eval(string)
- instance_eval(string)
- instance_eval(&block)
- instance_exec(*args, &block)
- yield(*args)
- call(*args)

Vending Machine

```ruby
finite initial: :idle do

  before :idle do
    @current_money = 0
  end

  after :accepting do
    puts "Current amount in machine: $%.2f" % @current_money
  end

  @money.each do |event_name, amount|
    event :"insert_#{event_name}" do
      before do
        puts "Adding #{event_name}"
        add_money (amount)
      end
      go from: :idle, to: :accepting
      go from: :accepting, to: :accepting
    end
  end

  @products.each do |event_name, price|
    event :"buy_#{event_name}" do
      before { puts "Buying #{event_name}" }
      go from: :accepting, to: :vending, if: lambda { @current_money >= price }
      after do
        @current_money -= price
      end
    end
  end

  event :complete_vend do
```

add_event

```ruby
def add_event(event_name, &block)
  # Some other stuff happens before here
  @class.send(:define_method, :"can_#{event_name}?") do
    event.transitions.key? current_state.name
  end

  @class.send(:define_method, :"#{event_name}") do
    if event.transitions.key? current_state.name

      transition = event.transitions[current_state.name]
      unless transition.condition.nil? or self.instance_exec(&transition.condition)
        raise Error.new('Does not meet the transition condition')
      end
      new_state = states[event.transitions[current_state.name].to]

      event.callbacks[:before].each do |callback|
        self.instance_eval &callback
      end
      # More callbacks happen here
      @current_state = new_state
      # More Callbacks happe here

      event.callbacks[:after].each do |callback|
        self.instance_eval &callback
      end
      self
    else
      raise Error.new 'Invalid Transition'
    end
  end
 end
```

Any Questions?

Thanks!