Ruby Modules

Including, Extending, & Prepending

Ruby Modules

Just separate code.

Module

A `Module` is a collection of methods and constants. The methods in a module may be instance methods or module methods. Instance methods appear as methods in a class when the module is included, module methods do not. Conversely, module methods may be called without creating an encapsulating object, while instance methods may not. (See `Module#module_function`.)

Ruby Modules: Including

```ruby
module Toppings
 attr_accessor :cheese

 def add_cheese
   @cheese = 2
 end
end

class Menultem; end

class Sandwich < Menultem
 include Toppings
end

ham_and_cheese_sammy = Sandwich.new # => #<Sandwich:0x00007fe8510a0a28>
ham_and_cheese_sammy.add_cheese # => 2
ham_and_cheese_sammy.cheese # => 2
ham_and_cheese_sammy.class.ancestors # => [Sandwich, Toppings, Menultem, Object, Kernel, BasicObject]
```

Just separate code available to instances

Ruby Modules: extending

```ruby
module Toppings
 def toppings_available
   "mayo, lettuce, tomato"
 end
end

class MenuItem; end

class Sandwich < MenuItem
 extend Toppings
end

Sandwich.toppings_available # => "mayo, lettuce, tomato"
roast_beef_sammy.class.ancestors # => [Sandwich, MenuItem, Object, Kernel, BasicObject]
```

Just separate code available as class methods.

Ruby Modules: Extending While Including

```ruby
module Toppings
 attr_accessor :cheese

 def add_cheese
   @cheese = 2
 end

 module ClassMethods
   def toppings_available
     "mayo, lettuce, tomato"
   end
 end

 def self.included(base)
   base.extend(ClassMethods)
 end
end

class MenuItem; end

class Sandwich < MenuItem
 include Toppings
end

turkey_sammy = Sandwich.new # => #<Sandwich:0x00007fb672025578>
turkey_sammy.add_cheese # => 2
turkey_sammy.cheese # => 2
Sandwich.toppings_available # => "mayo, lettuce, tomato"
turkey_sammy.class.ancestors # => [Sandwich, Toppings, MenuItem, Object, Kernel, BasicObject]
```

Just separate code available as instance and class methods.

Ruby Modules: Prepending Instance Methods

```ruby
module PrependedToppings
 attr_accessor :cheese

 def add_cheese
   @cheese = 4
 end
end

module Toppings
 attr_accessor :cheese

 def add_cheese
   @cheese = 2
 end
end

class MenuItem; end

class Sandwich < MenuItem
 include Toppings
 prepend PrependedToppings
end

ham_and_cheese_sammy = Sandwich.new # => #<Sandwich:0x00007fe8510a0a28>
ham_and_cheese_sammy.add_cheese # => 4
ham_and_cheese_sammy.cheese # => 4
ham_and_cheese_sammy.class.ancestors # => [PrependedToppings, Sandwich, Toppings, MenuItem,
Object, Kernel, BasicObject]
```

Just separate code, but PREpended in the ancestor chain.

Ruby Modules: Prepending BOTH Instance & Class

```ruby
module PrependedToppings
 attr_accessor :cheese

 def add_cheese
  @cheese = 4
 end

 module ClassMethods
  def toppings_available
   "mayo, lettuce, tomato, pickles, mustard"
  end
 end

  def self.prepended(base)
   class << base
    prepend ClassMethods
   end
  end
end

module Toppings
 attr_accessor :cheese

 def add_cheese
  @cheese = 2
 end

 module ClassMethods
  def toppings_available
   "mayo, lettuce, tomato"
  end
 end

 def self.included(base)
  base.extend(ClassMethods)
 end
end

class MenuItem; end

class Sandwich < MenuItem
 include Toppings
 prepend PrependedToppings
```