Ruby: Object-Oriented Concepts

Lecture 8
Classes

- Classes have methods and variables
  ```ruby
  class LightBulb
    #name with CamelCase
    def initialize  #special method name
      @state = false #@ means "instance variable"
    end
    def is_on?
      @state  #name with snake_case
    end
    def flip_switch!
      @state = !@state
    end
  end
  end
  ```

- Instantiation calls initialize method
  ```ruby
  f = LightBulb.new #=> <LightBulb:0x2322 @state=false>
  f.is_on? #=> false
  ```
Visibility

- Instance variables are never visible outside of the object
- Default visibility for methods is public
  - Can be made private (to object, not class)

```ruby
class LightBulb
  private def inside
    ...
  end

  def accessInternals (other_bulb)
    inside  # ok
    other_bulb.inside  # no! inside is private
    self.inside  # no explicit receiver allowed
  end
end
```
Getters/Setters

class LightBulb
    def initialize (color, state = false)
        @color = color #not visible outside
        @state = state #not visible outside
    end
    def color
        @color
    end
    def state
        @state;
    end
    def state= (value)
        @state = value
    end
end
Attributes

class LightBulb
  def initialize (color, state = false)
    @color = color
    @state = state
  end
  def color
    @color
  end

  attr_accessor :state  # name is a symbol
end
Attributes

class LightBulb
  def initialize (color, state = false)
    @color = color
    @state = state
  end

  attr_reader :color

  attr_accessor :state

end
Attributes

class LightBulb
  attr_reader :color
  attr_writer :size
  attr_accessor :state

  def initialize (color, state = false)
    @color = color
    @state = state
    @size = 0
  end
end
Classes Are Always Open

- A class can always be extended
  ```ruby
class Street
  def construction ... end
end
...

class Street
  def repave ... end #Street now has 2 methods
end
```

- Applies to core classes too
  ```ruby
class Fixnum
  def log2_of_cube #lg(self^3)
    (self**3).to_s(2).length - 1
  end
end

500.log2_of_cube #=> 26
```
No Overloading

- Method identified by (symbol) name
  - No distinction based on number of arguments
- To mimic overloading, use default arguments (and sometimes splat)

```ruby
class Rectangle
  def initialize (width, height = 10, *colors)
    @width = width
    @height = height
    @border = colors[0] if colors[0]
    @fill = colors[1] if colors[1]
  end
end
```
A Class is an Object Instance too

- Even classes are objects, created by `:new`
  ```ruby
  LightBulb = Class.new do #class LightBulb
    def initialize
      @state = false
    end
    def is_on?
      @state
    end
    def flip_switch!
      @state = !@state
    end
  end
  ```
Instance, Class, Class Instance

class LightBulb
    @state1    #class instance variable
    def initialize
        @state2 = ...    #instance variable
        @@state3 = ...    #class variable
    end
    def bar    #instance method
        ...
    end
    def self.foo    #class method
        ...
    end
end
Inheritance

- Single inheritance between classes
  ```ruby
  class LightBulb < Device
  ...
  end
  ```
- Default superclass is Object (which inherits from BasicObject)
- Super calls parent's method
  - No args means forward all args
    ```ruby
    def electrify (current, voltage)
      do_work
      super #with current and voltage
    end
    end
    ```
Modules

- Modules provide mixins
  ```ruby
  class LightBulb < Device
    include Enumerable, Stockable ...
  end
  ```
  - Enumerable methods call `#each`, so `LightBulb` must provide this method
  - Recall layering secondary and kernel ops

- Modules provide namespaces
  - Can not, themselves, be instantiated
    ```ruby
    module Stockable
      class Item ...
      def self.inventory ...
    end
    end
    ```
  - Access via scoping (`::`)
    ```ruby
    class Widget < Stockable::Item
    x = Stockable::inventory
    ```
Summary

- Classes as blueprints for objects
  - Contain methods and variables
  - Public vs private visibility of methods
  - Attributes for automatic getters/setters
- Metaprogramming
  - Classes are objects too
  - "Class instance" variables
- Single inheritance
- Modules are mixins and namespaces