Ruby: Object-Oriented Concepts

Lecture 9
Classes

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

- Instantiation calls `initialize` method
  ```ruby
  f = LightBulb.new #=> <LightBulb:0x0000e71c2322e71c2322>
  f.on? #=> false
  ```
Visibility

- Instance variables are always private
  - Private to *object*, not class
- Methods can be private or public (default)

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

  def access_internals(other_bulb)
    inside # ok
    other_bulb.inside # no! inside is private
    self.inside  # no explicit recv'r allowed
  end
end
```
Getters/Setters

class LightBulb
  def initialize(color, state = false)
    @color = color # not visible outside object
    @state = state # not visible outside object
  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

```ruby
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_accessor :state
  attr_writer :size

  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 Integer
    def log2_of_cube # lg(self^3)
      (self**3).to_s(2).length - 1
    end
  end
  500.log2_of_cube #=> 26
  ```
Classes are Always Open (!)

- Existing methods can be redefined!
- When done with system code (libraries, core ...) called “monkey patching”
- Tempting, but... Just Don’t Do It
No Overloading

- Method identified by (symbol) name
  - No distinction based on number of arguments

- Approximation: default arguments
  ```ruby
def initialize(width, height = 10)
    @width = width
    @height = height
end
```

- Better alternative: trailing options hash
  ```ruby
  def initialize(width, options)
  end
  ```

- Modern alternative: default keyword args
  ```ruby
  def initialize(height: 10, width:)
  ```
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 on?
      @state
    end
    def flip_switch!
      @state = !@state
    end
  end
  ```
class LightBulb
    @state1  # class instance var
    def initialize
        @state2 = ...  # instance variable
        @@state3 = ...  # class variable
    end
    def bar  # instance method
        ...  # sees @state2, @@state3
    end
    def self.foo  # class method
        ...  # sees @state1, @@state3
    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
  ```ruby
  class LightBulb < Device
    def electrify(current, voltage)
      do_work
      super # with current and voltage
    end
  end
  ```
Modules

- Another container for definitions

```ruby
module Stockable
    MAX = 1000
    class Item ... end
    def self.inventory ... end # utility fn
    def order ... end
end
```

- Cannot, themselves, be instantiated

```ruby
s = Stockable.new # NoMethodError
i = Stockable::Item.new # ok
Stockable.inventory  # ok
Stockable.order      # NoMethodError
```
Modules as Namespaces

- Modules create independent namespaces
  - cf. packages in Java
- Access contents via scoping (::)
  - Math::PI #=> 3.141592653589793
  - Math::cos 0 #=> 1.0
  - widget = Stockable::Item.new
  - x = Stockable::inventory
  - Post < ActiveRecord::Base
  - BookController < ActionController::Base
- Style: use dot to invoke utility functions (ie module methods)
  - Math.cos 0 #=> 1.0
  - Stockable.inventory
Modules are Always Open

- Module contains several related classes
- Style: Each class should be in its own file
- So split module definition

```ruby
# game.rb
module Game
end

# game/card.rb
module Game
  class Card ... end
end

# game/player.rb
module Game
  class Player ... end
end
```
Modules as “Mixins”

- Another container for method definitions
  ```ruby
  module Stockable
    def order ... end
  end
  ```

- A module can be included in a class
  ```ruby
  class LightBulb < Device
    include Stockable, Comparable ...
  end
  ```

- Module's (instance) methods-vars are now (instance) methods-vars for class
  ```ruby
  bulb = LightBulb.new
  bulb.order        # from Stockable
  if bulb <= old_bulb # from Comparable
  ```
Requirements for Mixins

- Mixins often rely on certain aspects of classes into which they are included
- Example: Comparable methods use \(<\Rightarrow\)

```ruby
module Comparable
  def <(other) ... end
  def <=(other) ... end
end
```

- Enumerable methods use \(#each\)
- Recall *layering* in SW I/II
  - Class implements kernel methods
  - Module implements secondary methods
Software Engineering

- All the good principles of SW I/II apply
- Single point of control over change
  - Avoid magic numbers
- Client view: abstract state, contracts, invariants
- Implementers view: concrete rep, correspondence, invariants
- Checkstyle tool: e.g., rubocop
- Documentation (YARD or RDoc)
  - Notation for types: yardoc.org/types.html
    @param [String, #read] # either is ok
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 namespaces and mixins