

# Classes vs Modules

small bags and big bags

# Classes

**i has two ansers.**



IGANHASCHEEBURGER.COM 🍷 🍷

# Class

- inherit from Module
- contain methods
- inherit
- instantiate

# Inherit From Module

```
p Class.ancestors # => [Class, Module, Object, Kernel]
```

# Contain Methods

```
class Person
  def name
    "Phil Collins"
  end

  def birthday
    Time.now
  end
end
```

# Inherit

```
class Employee < Person
  def salary
    1_000_000
  end
end
```

# Instantiate

```
class Employee < Person
  def salary
    1_000_000
  end
end
```

```
Employee.new
```

# Module

- inherits from Object
- contain methods
- contain classes
- contain modules
- cannot be instantiated
- **include**-able

# Inherits From Object

```
p Module.ancestors # => [Module, Object, Kernel]
```

# Contain Methods

```
module EltonJohn
  def your_song
    ...
  end

  def levon
    ...
  end
end
```

# Contain Classes

```
module MusicRecordings
  class Tape
    ...
  end

  class CompactDisc
    ...
  end

  class Vinyl
    ...
  end
end
```

```
MusicRecordings::Tape.new
```

# Contain Modules

```
module AwesomeArtists  
  module Whitesnake  
    class CD; end  
  end  
end
```

```
  module Ratt  
    class Tape; end  
  end  
end
```

```
AwesomeArtists::Ratt::Tape.new
```

# Cannot Instantiate

```
module AwesomeArtists
  module Whitesnake
    class CD; end
  end
  module Ratt
    class Tape; end
  end
end
```

```
AwesomeArtists.new
```

```
$ ruby test.rb
```

```
test.rb:10: undefined method `new' for AwesomeArtists:Module (NoMethodError)
```

# Include-able

```
class Artist
  include Comparable

  attr_accessor :name

  def <=>(other)
    return -1 if name == 'Whitesnake'
    name <=> other.name
  end
end
```

# Cat Break!



# Metaclasses

Artist:  
name  
aptitude

(singleton)  
rocks?

artist  
(instance)



```
class Artist
  attr_accessor :name, :aptitude
end
```

```
artist = Artist.new
```

```
class << artist
  def rocks?; true; end
end
```

```
class Artist
  attr_accessor :name, :aptitude
  def metaclass
    class << self; self; end
  end
end

artist = Artist.new

artist.metaclass.instance_eval do
  define_method(:rocks?) do
    true
  end
end

p artist.singleton_methods
```

singleton\_methods

# Hooks

# Hooks



# Hooks

- `Module#const_missing`
- `Module#included`
- `Module#extended`
- `Module#method_added`
- `Object#singleton_method_added`
- `Class#inherited`

# Module#const\_missing

```
class PhilCollins
  EASY_LOVER = 20
  class << self
    def const_missing(constant)
      puts "Doh.  Couldn't find #{constant}"
      10
    end
  end
end

def sing!
  puts "SUSSUDIO      = #{SUSSUDIO}"
  puts "EASY_LOVER   = #{EASY_LOVER}"
end

puts PhilCollins::SUSSUDIO
singer = PhilCollins.new
singer.sing!
```

# Module#included

```
module Banana
  class << self
    def included(klass)
      puts "I was included by #{klass}"
    end
  end

  def peel!
    puts "peeling peeling"
  end
end

class Monkey
  include Banana
end

Monkey.new.peel!
```

# Module#extended

```
module Banana
  class << self
    def extended(klass)
      puts "I was extended by #{klass}"
    end
  end

  def peel!
    puts "peeling peeling"
  end
end

class Monkey
  extend Banana
end

Monkey.peel!
```

# Module#method\_added

```
class PhilCollins
  class << self
    def method_added(method_name)
      puts "The method #{method_name} was added to #{self}"
    end
  end

  def sussudio; end
  def easy_lover; end
end
```

# Object#singleton\_method\_added

```
class Rachmaninov
  def singleton_method_added(method_name)
    puts "Quit being so sneaky!"
  end
end

composer = Rachmaninov.new
class << composer
  def compose!
  end
end
```

# Class#inherited

```
module Bach
  class JohannAmbrosius
    def self.inherited(klass)
      puts "My son, #{klass} was born!"
    end
  end

  class JohannSebastian < JohannAmbrosius
  end
end
```

# Hooks in the Wild!

# const\_missing

```
class Object
  class << self
    def const_missing(klass)
      require klass.to_s.downcase
      const_get(klass)
    end
  end
end

puts MyModel
```

# included and extended

```
module Validations
  def self.included(klass)
    klass.extend ClassMethods
  end

  module ClassMethods
    def validate_blah
    end
  end
end

class MyModel
  include Validations

  validate_blah
end
```

# One More Hook!

my personal favorite!

method\_missing

`Kernel#method_missing(symbol [, *args])`

```
class PirateShip
  def method_missing(symbol, *args)
    puts "#{symbol} was called"
  end
end

ship = PirateShip.new
ship.hello_world
```

# Domain Specific Languages

DSL's are:

# Languages

Specific

to

a

# Domain

# Types of DSL

- External
- Internal

# External DSLs

# Internal DSLs

# Examples

- rake
- Active Record
- rspec
- Builder

# rake

```
desc 'Run the test suite. Use FILTER to add to the command line.'
```

```
task :test do
```

```
  run_tests
```

```
end
```

  

```
desc 'Show which test files fail when run alone.'
```

```
task :test_deps do
```

```
  tests = Dir["test/**/test_*.rb"] + Dir["test/**/*_test.rb"]
```

  

```
  tests.each do |test|
```

```
    if not system "ruby -Ibin:lib:test #{test} &> /dev/null" then
```

```
      puts "Dependency Issues: #{test}"
```

```
    end
```

```
  end
```

```
end
```

# ActiveRecord

```
class People < ActiveRecord::Base
  acts_as_something
  validates_presence_of :name
  validates_numericality_of :age
end
```

# Rake Style

```
class MiniDsl
  def initialize(&block)
    instance_eval(&block)
    @current_task = nil
    @tasks = []
  end

  def desc(description)
    current_task.desc = description
  end
  ...
end

MiniDsl.new do
  desc "Some description"
  ...
end
```

# Active Record Style

```
class PhilCollins < AwesomeRecord
  attr_accessor :sussudio
  validates_presence_of :sussudio
end
```

```
phil = PhilCollins.new
puts phil.valid?
phil.sussudio = true
puts phil.valid?
```

# (An) Implementation

```
class AwesomeRecord
  class << self
    def validates_presence_of(field)
      self.class_eval %{
        def validate_#{field}
          if !respond_to?(:#{field}) || send(:#{field}) == nil
            return false
          end
          return true
        end
      }
    end
  end

  def valid?
    methods.find_all { |x| x =~ /^validate_/ }.all? do |method|
      send(method)
    end
  end
end
```