The Polite Programmer's Guide to Ruby Etiquette

Creating Software that doesn't break other Software

Moderating our show is ...

Mr. Manners

(A.K.A Ed Sumerfield)

Introducing our Hosts ...

Chris Nelson

Jim Weirich

Method Missing

Dear Polite Programmers,

I have some code that adds awesome magic methods, but somehow it breaks my object. This seems terribly impolite. What can I do to improve my code?

-- Black Magic Programmer

Dear Polite Programmers,

I have some code that adds awesome magic methods, but somehow it breaks my object. This seems terribly impolite. What can I do to improve my code?

-- Black Magic Programmer

```
require 'stereo'

class BlackMagicStereo < Stereo
  def method_missing(name, *args)
    if sym.to_s =~ /^play_(\w+)$/
      system("open songs/#{$1}.m4a")
    end
  end
end

stereo = BlackMagicStereo.new
stereo.play_black_magic_woman    # THIS WORKS
stereo.play("Magic Bus")         # THIS FAILS
```

```
require 'delegate'
require 'juke_box'

class Stereo < Delegator
  def initialize
    @real_stereo = JukeBox.new
  end
  def __getobj__
    @real_stereo
  end
end
```

```
require 'stereo'

class BlackMagicStereo < Stereo
  def method_missing(name, *args)
    if sym.to_s =~ /^play_(\w+)$/
      system("open songs/#{$1}.m4a")
    end
  end
end

stereo = BlackMagicStereo.new
stereo.play_black_magic_woman    # THIS WORKS
stereo.play("Magic Bus")         # THIS FAILS
```

```ruby
require 'stereo'

class BlackMagicStereo < Stereo
  def method_missing(name, *args)
    if sym.to_s =~ /^play_(\w+)$/
      system("open songs/#{$1}.m4a")
    else
      super
    end
  end
end

stereo = BlackMagicStereo.new
stereo.play_black_magic_woman    # THIS WORKS
stereo.play("Magic Bus")         # THIS FAILS
```

Hook Methods

- method_missing
- const_missing
- append_features
- method_added
- inherited

The Polite Programmer ...

- Always calls super with method missing
  - (this applies to any hook method)

Respond to

Dear Polite Programmers,

Thanks for the help with method missing. However, I have another question. Why doesn't the following work?

```
if stereo.respond_to?("play_black_magic_woman")
  stereo.play_black_magic_woman
end
```

-- Black Magic Programmer

Dear Polite Programmers,

Thanks for the help with method missing. However, I have another question. Why doesn't the following work?

```
if stereo.respond_to?("play_black_magic_woman")
  stereo.play_black_magic_woman
end
```

-- Black Magic Programmer

```
class BlackMagicStereo < Stereo
  def method_missing(name, *args)
    if sym.to_s =~ /^play_(\w+)$/
      system("open #{$1}.m4a")
    else
      super
    end
  end
end
```

```
class BlackMagicStereo < Stereo
  def method_missing(name, *args)
    if sym.to_s =~ /^play_(\w+)$/
      system("open #{$1}.m4a")
    else
      super
    end
  end

  def respond_to?(name)
    sym.to_s =~ /^play_/ || super
  end
end
```

The Polite Programmer ...

- Always pairs an appropriate respond_to? method with every method_missing.
- Don't forget to delegate to super where appropriate!

More Method Missing

Dear Polite Programmers,

I'm trying to extend my friend's BlackMagicStereo class to handle video. Can you tell me what I'm doing wrong?

-- Gandalf the Grey

Dear Polite Programmers,

I'm trying to extend my friend's BlackMagicStereo class to handle video. Can you tell me what I'm doing wrong?

-- Gandalf the Grey

```
require 'black_magic'

class BlackMagicStereo
  def method_missing(name, *args, &block)
    if sym.to_s =~ /^show_(\w+)$/
      system("open videos/#{$1}.mpg")
    else
      super
    end
  end
end
```

```
stereo = BlackMagicStereo.new

stereo.show_return_of_the_king      # WORKS
stereo.play_earendil_was_a_mariner # FAILS
```

```
require 'black_magic'

class BlackMagicStereo
  def method_missing(name, *args, &block)
    if sym.to_s =~ /^show_(\w+)$/
      system("open videos/#{$1}.mpg")
    else
      super
    end
  end
end
```

```
require 'black_magic'

class BlackMagicStereo
  alias method_missing_without_show \
    method_missing
  def method_missing(name, *args, &block)
    if sym.to_s =~ /^show_(\w+)$/
      system("open videos/#{$1}.mpg")
    else
      method_missing_without_show(
        name, *args, &block)
    end
  end
end
```

The Polite Programmer ...

- Delegates to an aliased version of the original method_missing when reopening an existing class.

Alternative ...

```
require 'black_magic'

module GreyMagic
  def method_missing(name, *args, &block)
    if sym.to_s =~ /^show_(\w+)$/
      system("open videos/#{$1}.mpg")
    else
      super
    end
  end
end

class BlackMagicStereo
  include GreyMagic
end
```

The Polite Programmer ...

- Attempts to intrude into existing classes as little as possible.
- Uses a module to mix in behavior to an existing class.

Hook Rules Summary:

- On classes that you own:
  - Delegate vertically to super
- On classes that you do not own (i.e. are extending)
  - Delegate horizontally to an aliased version of the hook method

Dependencies

Dear Polite Programmers,

I've packaged my awesome library as a gem (named the "happy_words" gem). But when users install my gem and try to run it, they get "no such file to load -- re ".

Is there some point of etiquette that I am missing?

-- Clueless (but Happy) in Gemland

Dear Polite Programmers,

I've packaged my awesome library as a gem (named the "happy_words" gem). But when users install my gem and try to run it, they get "no such file to load -- re ".

Is there some point of etiquette that I am missing?

-- Clueless (but Happy) in Gemland

```
$ gem install pkg/happy_words-0.0.1.gem
Successfully installed happy_words-0.0.1
1 gem installed
$
$ irb
>> require 'happy_words'
LoadError: no such file to load -- re
from <internal:lib/rubygems/custom_require>:
29:in `require'
>>
```

```
require 'rake/gempackagetask'
SPEC = Gem::Specification.new do |s|
  s.name = 'happy_words'
  s.version = '0.0.1'
  s.summary = "Find Happy Words in Text"
  s.files = PKG_FILES.to_a
  s.require_path = 'lib'
  ...
end

Rake::GemPackageTask.new(SPEC) do |pkg|
end
```

```
require 'rake/gempackagetask'
SPEC = Gem::Specification.new do |s|
  s.name = 'happy_words'
  s.version = '0.0.1'
  s.summary = "Find Happy Words in Text"
  s.files = PKG_FILES.to_a
  s.require_path = 'lib'
  s.add_dependency('re', '~> 0.0.1')
  ...
end

Rake::GemPackageTask.new(SPEC) do |pkg|
end
```

```
require 'rake/gempackagetask'
SPEC = Gem::Specification.new do |s|
  s.name = 'happy_words'
  s.version = '0.0.1'
  s.summary = "Find Happy Words in Text"
  s.files = PKG_FILES.to_a
  s.require_path = 'lib'
  s.add_dependency('re', '~> 0.0.1')
  s.add_development_dependency('rspec', '>= 2.0.0')
  s.add_development_dependency('flexmock')
  ...
end

Rake::GemPackageTask.new(SPEC) do |pkg|
end
```

```
gem install happy_words --development
```

The Polite Programmer ...

- Declares his dependencies in the Gem specification.
- Declares his development dependencies appropriately.

Privacy - Part I

Dear Polite Programmers,

I'm using the gem 'license-to-kill' (version 0.0.7) for managing Unix processes. There is a private method named 'smoke_screen' that I would like to call. Is it polite to use 'send' to get around the private?

-- Bond, Jill Bond

Dear Polite Programmers,

I'm using the gem 'license-to-kill' (version 0.0.7) for managing Unix processes. There is a private method named 'smoke_screen' that I would like to call. Is it polite to use 'send' to get around the private?

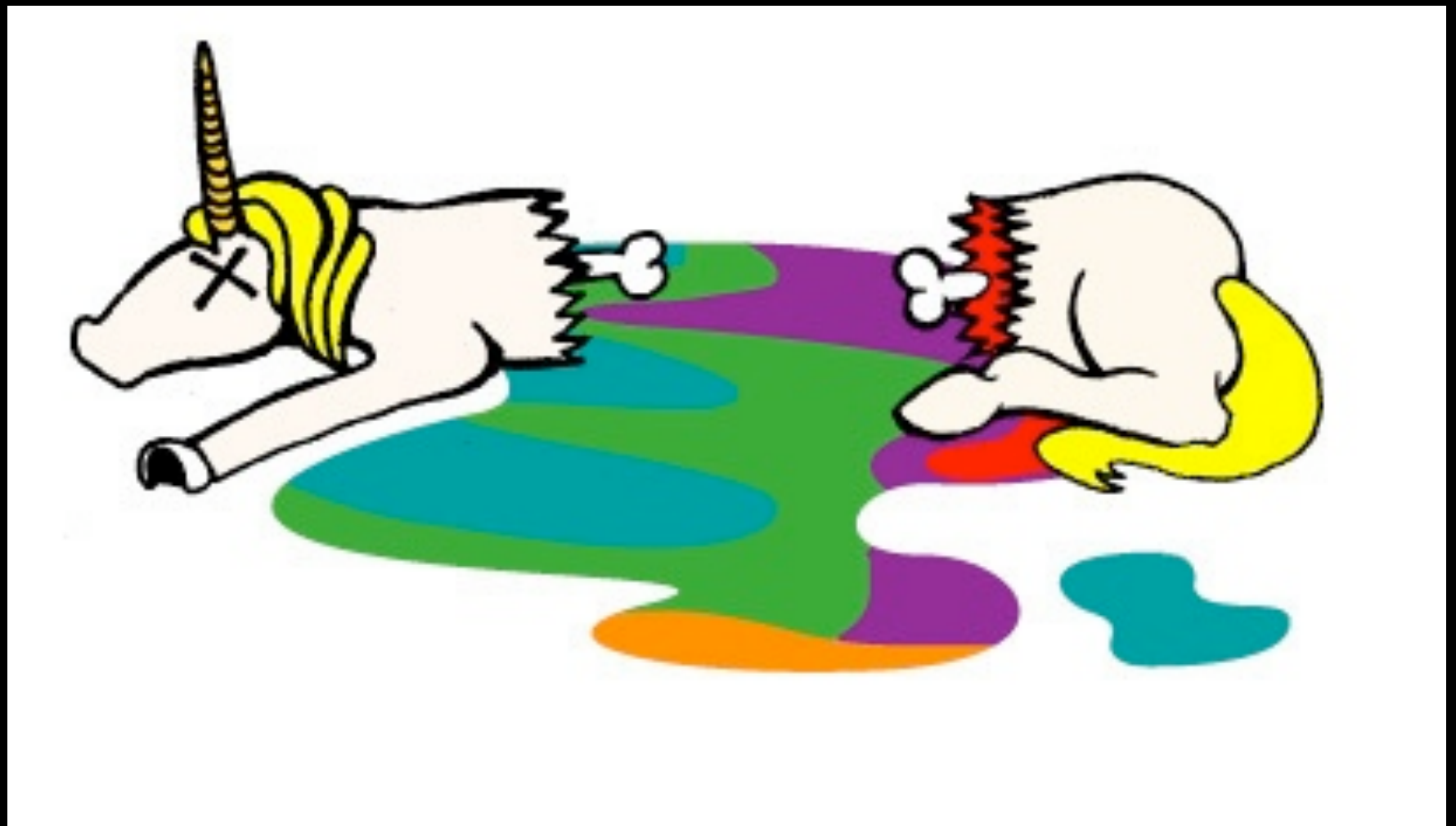
-- Bond, Jill Bond

```
license = LicenseToKill.new

license.send(:smoke_screen)
```

Jim W: Move it to another class and test that class
Testing private controller methods via send
makes controller tests WAY too brittle.

Scott B: it also kills unicorns
so - congratulations. now they're extinct.



The Polite Programmer ... Prudent

- Doesn't violate private space unless they have a strong reason to do so.

Privacy - Part II

Dear Polite Programmers,

My pair programming partner and I have a disagreement that we hope you can resolve. I like to mark as many methods as possible as private. My partner prefers to never use private. Please help us before the situation comes to blows.

-- Mr. and Mrs. Smith

Dear Polite Programmers,

My pair programming partner and I have a disagreement that we hope you can resolve. I like to mark as many methods as possible as private. My partner prefers to never use private. Please help us before the situation comes to blows.

-- Mr. and Mrs. Smith

```
class BlackMagicStereo < Stereo
  def method_missing(name, *args)
    if name.to_s =~ /^play_(\w+)$/
      system("open songs/#{$1}.m4a")
    else
      super
    end
  end

  def respond_to?(name)
    name.to_s =~ /^play_/ || super
  end
end
```

```
class BlackMagicStereo < Stereo
  def method_missing(name, *args)
    if name.to_s =~ /^play_(\w+)$/
      system("open songs/#{$1}.m4a")
    else
      super
    end
  end

  def respond_to?(name)
    name.to_s =~ /^play_/ || super
  end
end
```

Duplication

```
class BlackMagicStereo < Stereo
  def method_missing(name, *args)
    song = extract_song_name(name)
    if song
      system("open songs/#{song}.m4a")
    else
      super
    end
  end

  def respond_to?(name)
    extract_song_name(name)
  end

  ...
```

```
  ...

  private

  def extract_song_name(name)
    md = (name.to_s =~ /^play_(\w+)$/)
    md && md[1]
  end
end
```

```
  ...

  private

  def extract_song_name(name) # :nodoc:
    md = (name.to_s =~ /^play_(\w+)$/)
    md && md[1]
  end
end
```

Use Private to indicate ...

- Methods that are implementation details, not interface
- Methods that are subject to change in the future.

```
module BlackMagic
  module Internal
    class AnImplementationDetail # :nodoc:
      # This class is an internal
      # implemenation detail.
      # Clients should not use it.
    end
  end
end
```

The Polite Programmer ...

- Clearly communicates what is public API methods and what is reserved for internal use.
  - private/nodoc for methods
  - internal namespace/nodoc for classes, modules and constants
- Carefully considers what should be public API

Monkey Patching

Dear Polite Programmers,

My friend, who programs in Python and likes yellow hats, tells me that Monkey Patching is evil and should never be done. As a rubyist, I love to Monkey Patch. I'm curious, are there any rules about the etiquette of Monkey Patching?

-- George

Dear Polite Programmers,

My friend, who programs in Python and likes yellow hats, tells me that Monkey Patching is evil and should never be done. As a rubyist, I love to Monkey Patch. I'm curious, are there any rules about the etiquette of Monkey Patching?

-- George

Two Scenarios:

- Adding Behavior
- Replacing Behavior

```
class Monkey
  def climb
    puts "Climbing Rope"
  end
end
```

Adding Behavior

```
# Open existing class Monkey:

class Monkey
  def swing
    puts "Swinging on a Rope"
  end
end
```

Adding Behavior

```
# Open existing class Monkey (Ruby 1.8 version):

class Monkey
  fail "swing already defined on monkey" if
    instance_methods.include?("swing")

  def swing
    puts "Swinging on a Rope"
  end
end
```

Adding Behavior

```
# Open existing class Monkey (Both 1.8 and 1.9):

class Monkey
  fail "swing already defined on monkey" if
    instance_methods.map { |n| n.to_s }.
    include?(:swing)

  def swing
    puts "Swinging on a Rope"
  end
end
```

Replacing Behavior

```
# Open existing class Monkey

class Monkey
  alias climb_without_redirect climb
  def climb
    old_stdout = $stdout
    $stdout = open("climb_output.txt", "w")
    climb_without_redirect
  ensure
    $stdout = old_stdout
  end
end
```

Don't Break Contracts!

The Polite Programmer ...

- Monkey patches only when regular programming techniques are inadequate
- Checks for existing methods when adding new methods
- Delegates horizontally to aliased method when wrapping existing methods
- Does *NOT* break contracts when wrapping

Specifying Protocols

Dear Polite Programmers,

I have a robot fighting arena gem which uses Robot objects provided by clients of my library. What is the most polite way to explain to the library users what methods the Robot needs to respond to?

-- Rachim Sachem

Dear Polite Programmers,

I have a robot fighting arena gem which uses Robot objects provided by clients of my library. What is the most polite way to explain to the library users what methods the Robot needs to respond to?

-- Rachim Sachem

Client's Responsibility

Library Code

Client *provides* the Robot

Library *uses* the Robot

RSpec Version ...

basic_robot_spec.rb

```
require 'spec_helper'
require 'botwars/basic_robot'

shared_examples_for "a basic robot" do
  it "is created" do
    robot.should_not be_nil
  end

  context "when starting the game" do ... end
  context "when running the game" do ... end
  context "when dead" do ... end
end
```

jim_bot_spec.rb

```
require 'spec_helper'
require 'botwars/jim_bot'
require 'botwars/basic_robot_spec'

describe JimBot do
  it_should_behave_like "a basic robot"

  let(:robot) { JimBot.new }
end
```

jim_bot_spec.rb

```
require 'spec_helper'
require 'botwars/jim_bot'
require 'botwars/basic_robot_spec'

describe JimBot do
  it_should_behave_like "a basic robot"

  let(:robot) { JimBot.new }
end
```

Test::Unit Version ...

basic_robot_tests.rb

```
module BasicRobotTests
  def test_created
    assert_not_nil robot
  end

  def test_can_start_game
    ...
  end
  ...
end
```

jim_bot_test.rb

```
require 'botwars/jim_bot'
require 'botwars/basic_robot_tests'

class JimBotTest < Test::Unit::TestCase
  include BasicRobotTests

  def robot
    @robot ||= JimBot.new
  end
end
```

jim_bot_test.rb

```
require 'botwars/jim_bot'
require 'botwars/basic_robot_tests'

class JimBotTest < Test::Unit::TestCase
  include BasicRobotTests

  def robot
    @robot ||= JimBot.new
  end
end
```

The Polite Programmer ...

- Provides a sample test case to verify client-provided code
  - A mix-in module in Test::Unit
  - Shared examples in RSpec
- The ActiveModel Lint code is a good exemplar

Namespaces

Dear Polite Programmers,

My brother Darryl likes to put everything at the top global level. My other brother Darryl does the same, but prefixes his class names and constants with his initials (e.g. OBD_xxx). What is the polite way to handle this sibling rivalry?

-- Larry

Dear Polite Programmers,

My brother Darryl likes to put everything at the top global level. My other brother Darryl does the same, but prefixes his class names and constants with his initials (e.g. OBD_xxx). What is the polite way to handle this sibling rivalry?

-- Larry

Class / Module Name Conflicts

lib/happy_words.rb

```
module HappyWords
  ...
end

require 'happy_words/patterns'
```

lib/happy_words/patterns.rb

```
module HappyWords
  class Patterns
    ...
  end
end
```

lib/happy_words.rb

```
module HappyWords
   ...
end

require 'happy_words/patterns'
```

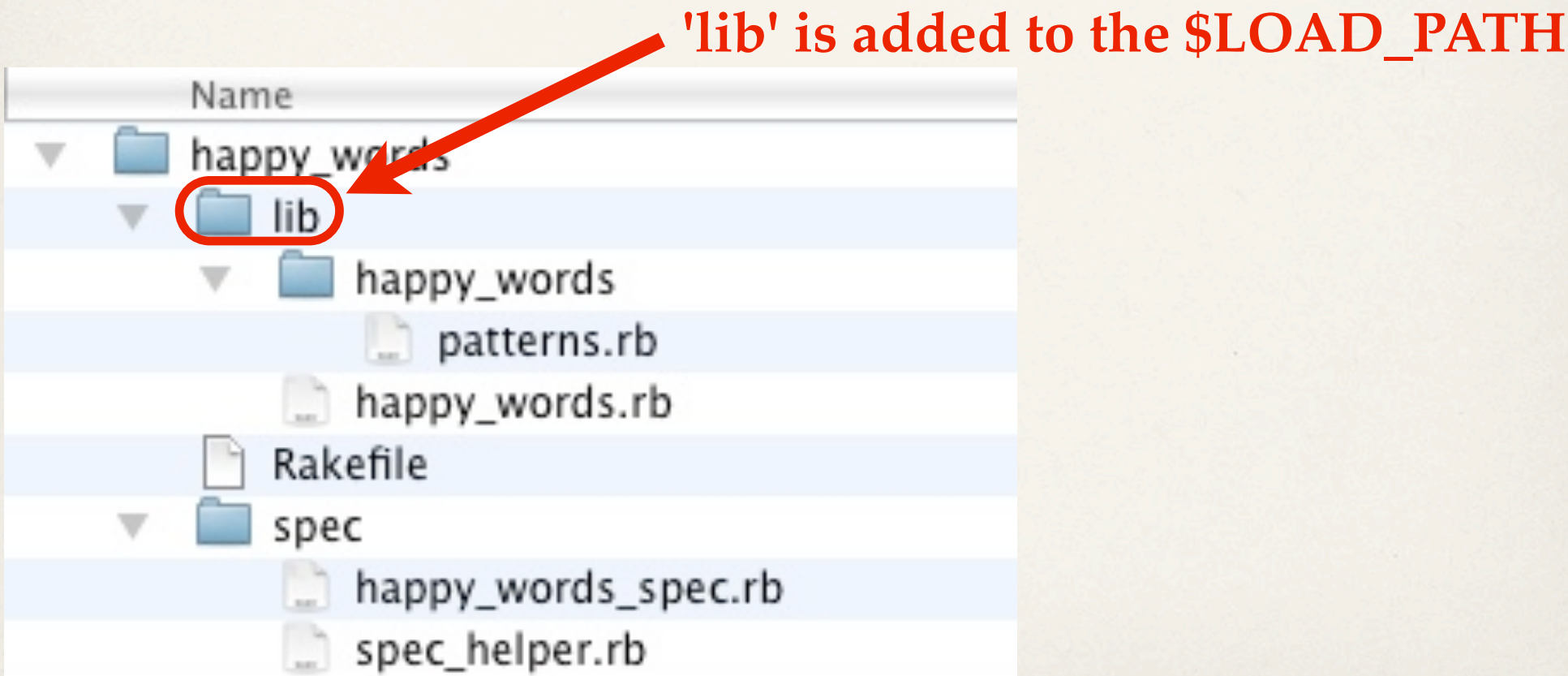
lib/happy_words/patterns.rb

```
module HappyWords
   class Patterns
      ...
   end
end
```

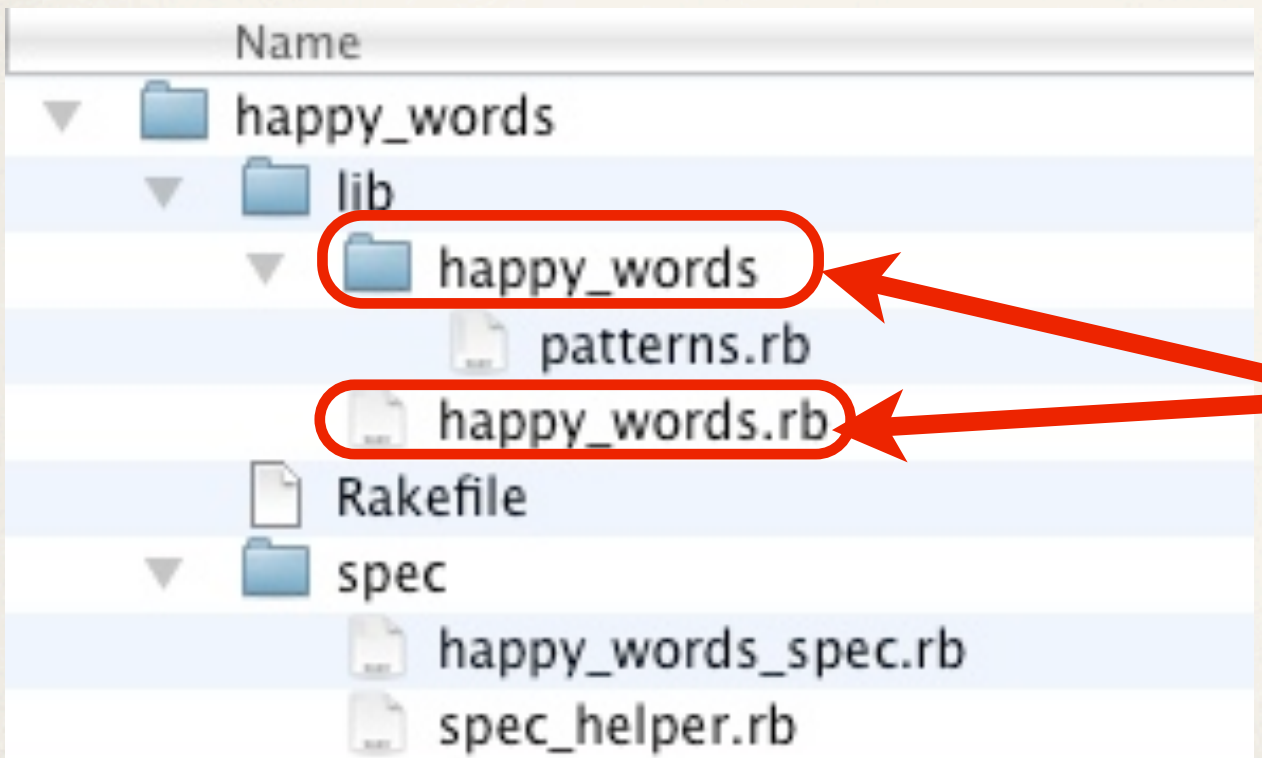
Everything is in the HappyWords namespace.

File Name Conflicts

Project: Happy Words



Project: Happy Words



All files in 'lib' must be unique across all libraries

Gem Name Conflicts

- Check rubygems.org for gem name conflicts

Keep Names Consistent

Ruby Names: HappyWords

File Names: happy_words

Gem Names: ???

Survey of RubyGems.org

Dashes (no underscores):	5046
Underscores (no dashes):	3792
Both underscores and dashes:	529
CamelCase:	417

Gem Name Suggestion

- Multi-Words: Use underscore
  - e.g. happy_words
- Related parent project: Use dash
  - e.g. rspec-given, zlib-crc32_combine

The Polite Programmer ...

- Uses namespaces consistently
  - Top Level Name Space: CamelCase
  - Library Dirs: snake_case
  - Gem Name:
    - project_name
    - (or parent-project_name)

For More Polite Programming...

http://politeprogrammer.com

Twitter:

@ thepolitepgmr

License

https://github.com/jimweirich/polite_programmer_presentation

With the Voice Talents of:

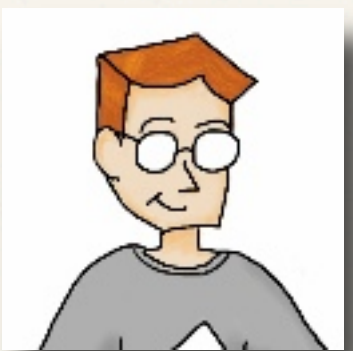
- Glen Vanderburg
- James Edward Grey II
- Brandon Dimcheff
- Kelly Fowler
- Jay McGaffigan
- Jamis Buck
- Doug Alcorn
- Joe O'Brien

The Polite Programmer is brought to you by:

EdgeCase
software artisans
Partner / Build / Teach

gaslightsoftware

Jim Weirich



Ed Sumerfield

Chris Nelson