Packaging Ruby Libraries with RubyGems

Dr Nic Williams
drnicwilliams.com

DR NIC ACADEMY

Easier to write

RubyGems than not to

Demo

- create class and use it
- save + run from cmd line
- extract classes to file + require
- show load path

$LOAD_PATH == $:

```
["/opt/local/lib/ruby/site_ruby/1.8",
 "/opt/local/lib/ruby/site_ruby/1.8/i686-darwin9.1.0",
 "/opt/local/lib/ruby/site_ruby",
 "/opt/local/lib/ruby/vendor_ruby/1.8",
"/opt/local/lib/ruby/vendor_ruby/1.8/i686-darwin9.1.0",
"/opt/local/lib/ruby/vendor_ruby",
"/opt/local/lib/ruby/1.8",
"/opt/local/lib/ruby/1.8/i686-darwin9.1.0",
"."]
```

files in current path (where app started)

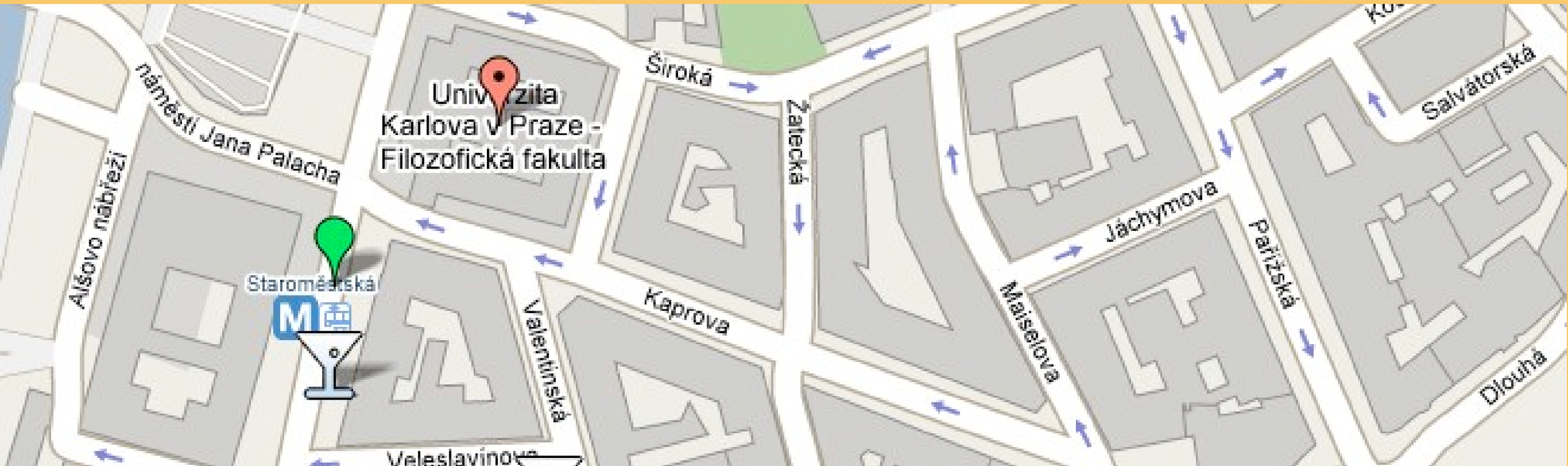
Relative require

```
require File.dirname(__FILE__) + "/person"
```

relative to THIS file

Update $LOAD_PATH

```
$:.unshift File.dirname(__FILE__)
require "person"
```



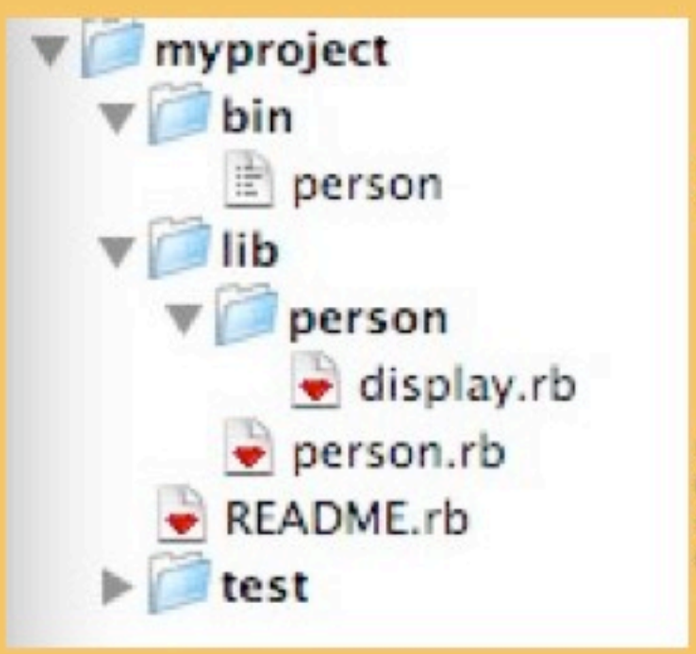
Demo

- break person.rb into person/*.rb files
- use relative requirements within person.rb
- add path to $LOAD_PATH and refactor requires

Where do tests go?

- /test or /spec folder
- then code needs own folder /lib
- executable code in own folder /bin
  - #!/usr/bin/env ruby
  - add /bin to env path
  - chmod +x each file

Good project structure



1. loads ../lib/person.rb
2. sets $LOAD_PATH
3. loads person/*.rb

Questions to ask

- How do I share my project?
- Where/how should other people install project?
- Inter-project dependencies?
- New project versions?

RubyGems

- Packaged in .gem file
- Installed into shared/central cache
- Version numbers
- Dependencies to other gems/versions
- Community download site - rubyforge.org

NewGem - scaffolding

- `gem install newgem`
- `newgem gem_name`
- creates scaffolding to new RubyGem project
- rake -T -- lots of helpers to build + deploy

Demo - porting library

- `newgem myproject` (or `newgem .`)
- `rake manifest:refresh`
- `rake install_gem`

Deployment to RubyForge

- create rubyforge account
- create rubyforge project
- rake deploy VERSION=X.Y.Z
  - website upload
  - gem/tar/zip upload

Native C extensions

- C is faster than Ruby
- External integration of existing C libraries

newgem - extconf generator

- script/generate extconf
- rake test
- autotest

Demo

- `script/generate extconf stack`
- `gem install ZenTest` (includes autotest)
- `autotest`

test/test_stack_extn.rb

```
require "test/unit"

$:.unshift File.dirname(__FILE__) + "/../ext/stack"
require "stack.so"

class TestStackExtn < Test::Unit::TestCase
  def test_init
    stack = Stack.new
    assert_equal([], stack.instance_variable_get("@arr"))
  end
end
```

ext/stack/stack.c

```c
VALUE cStack;
void Init_stack()
{
  cStack = rb_define_class("Stack", rb_cObject);
  rb_define_method(cStack, "initialize", t_init, 0);
}
```

ext/stack/stack.c cont...

```c
static VALUE t_init(VALUE self)
{
  VALUE arr;
  arr = rb_ary_new();
  rb_iv_set(self, "@arr", arr);
  return self;
}
```

test/test_stack_extn.rb

```
  def test_push
    stack = Stack.new
    stack.push 3
    stack.push 4
    assert_equal([3,4], stack.instance_variable_get("@arr"))
  end
```

ext/stack/stack.c cont...

```
static VALUE t_push(VALUE self, VALUE val)
{
  VALUE arr;
  arr = rb_iv_get(self, "@arr");
  rb_ary_push(arr, val);
  return arr;
}

VALUE cStack;
void Init_stack()
{
  cStack = rb_define_class("Stack", rb_cObject);
  rb_define_method(cStack, "initialize", t_init, 0);
  rb_define_method(cStack, "push", t_push, 1);
}
```

Build and install

- `rake manifest:refresh`
- `rake install_gem`
- “Building native extensions. This could take a while...”

The Pragmatic Programmers

C Extensions
p261+

Programming Ruby

The Pragmatic Programmers' Guide

Dave Thomas
with *Chad Fowler* and *Andy Hunt*

Second Edition, Includes Ruby 1.8