UW Ruby Programming 110

Winter 2015

Michael Cohen

Lecture 5

Feb 5, 2015

Lecture 5

1. Test Driven Development
2. Web Scraping
3. Method Missing
4. Meta Programming
5. Assignments

Section 1
Test Driven Development

Section 1: TDD

Philosophy

Write tests first
to help evolve your design

Benefits

1. Validate implementation conforms to spec
2. Enable safe refactoring

Section 1: TDD

Types of tests

1. Unit
2. Functional
3. Integration

Section 1: TDD

Tools

- TestUnit
- RSpec
- MiniTest

Section 1: TDD

MiniTest

```
require 'minitest/autorun'

class TestMyClass < Minitest::Test
  def setup
    ...
  end

  def test_thing1
    ...
  end

  def test_thing2
    ...
  end
end
```

MiniTest

```
assert something
assert_nil obj
assert_empty obj
assert_equal value, obj
assert_instance_of <class>, obj
assert_kind_of <class>, obj
```

MiniTest

```
refute something
refute_nil obj
refute_empty obj
refute_equal value, obj
refute_instance_of <class>, obj
refute_kind_of <class>, obj
```

Section 1: TDD

Demo

Section 2

Web Scraping

Section 2: Web Scraping

Overview

1. read document over http
2. convert document to objects
3. manipulate objects
4. render output

Section 2: Web Scraping

Examples

1. Find links on Wikipedia article.
2. Render article summary for RSS feed.

Section 2: Web Scraping

Example 1: Wikipedia

```
# read document:

require 'open-uri'

url = 'http://en.wikipedia.org/wiki/Ruby'
html = open(url)
```

Example 1: Wikipedia

```
# convert to objects:

require 'nokogiri'

doc = Nokogiri::HTML(html)
```

Section 2: Web Scraping

Example 1: Wikipedia

```
# manipulate objects:

# use css selectors to find all <a> tags:
anchors = doc.css('a')
anchors.length
hrefs = anchors.map {|node| node.attribute 'href'}
```

Section 2: Web Scraping

Example 2: RSS Feed

```
# read document:

require 'open-uri'

url = 'http://scripting.com/rss.xml'
rss = open(url)
```

Example 2: RSS Feed

```
# convert to objects:

require 'nokogiri'
```

```
rss_doc = Nokogiri::XML(rss)
```

Section 2: Web Scraping

Example 2: RSS Feed

```
# manipulate objects:

# use xpath to find all items:
item_nodes = rss_doc.xpath('/rss/channel/item')
item_nodes.length

items = item_nodes.map do |node|
  node.elements.reduce({}) do |item, el|
    item[el.name] = el.content
    item
  end
end
```

Section 3

Method Missing

Method Missing

`method_missing` **is invoked when a message is sent to an object, and there is no method defined to respond to the message.**

Method Missing

```
class MethodMissingDemo
   def method_missing(method, *args)
      puts "method_missing: #{method}"
   end
end

obj = MethodMissingDemo.new
obj.random_method
```

Section 3: Method Missing

LoggingProxy

```
class LoggingProxy
  def initialize(target)
    @target = target
  end
  def method_missing(method, *args)
    puts "calling #{method}"
    @target.send method, *args
  end
  def respond_to?(method)
    @target.respond_to? method
  end
end
```

Section 4

Meta Programming

Section 4: Meta Programming

Definition

Meta-programming:
program that writes a program.

Trivial Example

```
class Trivial
   if DEBUG
      def log(msg); puts msg; end
   else
      def log(msg); end
   end
end
```

Real Example

```
class Article
  attr :title, :author, :published_on, :content
end
```

What is `attr`?

It's not a language construct, it's just a method call that writes code.

Let's see how that's done.

attr

`attr` creates a getter and a setter:

```
attr :title
```

becomes:

```
def title
  @title
end
def title=(new_title)
  @title = new_title
end
```

attr

How do we implement that?

Let's try to build our own version, `prop`

```
class IceCream
  prop :flavor, :temperature
end
```

Section 4: Meta Programming

`prop` is a method

We want

```
prop :flavor
```

to become:

```
def flavor
  @flavor
end
def flavor=(new_flavor)
  @flavor = new_flavor
end
```

`prop` is a method

```
def prop(prop_name)
   # ???
end
```

`prop` is a method

Something like this:

```
def prop(prop_name)
  def <prop_name>
    @<prop_name>
  end
  def <prop_name>=(value)
    @<prop_name> = value
  end
end
```

define_method

We need `define_method`:

```
def prop(prop_name)
  define_method prop_name do
    # ??? @<prop_name>
  end
  define_method "#{prop_name}=" do |value|
    # ??? @<prop_name> = value
  end
end
```

instance*variable*get/set

We need `instance_variable_get/set`:

```
def prop(prop_name)
  define_method prop_name do
    instance_variable_get "@#{prop_name}"
  end
  define_method "#{prop_name}=" do |value|
    instance_variable_set "@#{prop_name}", value
  end
end
```

multiple properties

We want to support multiple properties:

```
def prop(*prop_names)
  prop_names.each do |prop_name|
    define_method prop_name do
      instance_variable_get "@#{prop_name}"
    end
    define_method "#{prop_name}=" do |new_value|
      instance_variable_set "@#{prop_name}", new_value
    end
  end
end
```

Section 4: Meta Programming

`prop`

Where does this belong?

Section 4: Meta Programming

add to Class

It adds behavior to Class:

```
class Class
  def prop(*prop_names)
    prop_names.each do |prop_name|
      define_method prop_name do
        instance_variable_get "@#{prop_name}"
      end
      define_method "#{prop_name}=" do |new_value|
        instance_variable_set "@#{prop_name}", new_value
      end
    end
  end
end
```

Section 4: Meta Programming

use it

```ruby
class Article
  prop :title, :author, :published_on, :content
end
```

```ruby
article = Article.new
article.title = "Ruby Meta-programming Secrets"
article.title      #=> "Ruby Meta-programming Secrets"
```

Section 4: Meta Programming

Dynamic Object

Another example of meta-programming: create a dynamic object

Hash vs. Object

A hash can have arbitrary properties, but you need to use [] to access them:

```
h = {}
h[:flavor] = "chocolate"
h[:flavor]                  #=> "chocolate"
h.flavor                    #=> ERROR
h[:temperature]             #=> nil
```

Hash vs. Object

An object has fixed properties/attributes, but you can use dot notation:

```
class FixedProperty
  prop :flavor
end

fp = FixedProperty.new
fp.flavor = "vanilla"
fp.flavor                    #=> "vanilla"
fp.temperature               #=> ERROR
```

Section 4: Meta Programming

Hash vs. Object

Can we combine these - arbitrary properties with dot notation?

The secret is `method_missing`.

Prop with Method Missing

Let's combine `prop` with `method_missing` to give us an object that can dynamically add properties using dot notation.

Section 4: Meta Programming

Prop with Method Missing

```
class DynamicObject
  def method_missing(method, *args)
    # does method exist yet?
    unless respond_to? method
      # property doesn't exist yet, need to create it:
      prop_name = method[-1] == "=" ? method.to_s.chop.to_sym : method
      self.class.prop prop_name
    end

    # now invoke the method:
    send method, *args
  end
end
```

DynamicObject

```
obj = DynamicObject.new

obj.flavor = "vanilla"
obj.flavor                  #=> "vanilla"
obj.temperature             #=> nil
obj.temperature = "cold"    #=> "cold"
obj.texture = "smooth"      #=> "smooth"
```

Section 5

Assignment #5

Section 5: Assignment #5

Problem 1

```ruby
# implement prop_reader, write MiniTest unit tests

# expected results:
class PropReader
  prop_reader :flavor
  def initialize(flavor)
    @flavor = flavor
  end
end

obj = PropReader.new "spicy"
obj.respond_to? :flavor      #=> true
obj.respond_to? :"flavor="   #=> false
obj.flavor                   #=> "spicy"
```