Metaprogramming in Ruby

1.6.2 Problem 2

This example has been adapted from Hal Fulton's article "An Exercise in Metaprogramming with Ruby".

Suppose we have two CSV (comma-separated values) files with a descriptive header at the top, as follows:

File: **location.txt**

```
name,country
"Matz", "USA"
"Fabio Akita", "Brazil"
"Peter Cooper", "UK"
```

File: **twitter.txt**

```
twitterid,url
"AkitaOnRails","http://www.akitaonrails.com/"
"peterc","http://www.petercooper.co.uk/"
```

Let us start by writing a class and storing it in a file **datawrapper.rb**. We'll call our class **DataWrapper** and also define a *class method* called **wrap** which will take a filename as a parameter and build a class from it. The first line of the above two text files have a comma-separated list of attribute names. Furthermore, we want to treat the file as an array of data items, reading it into an array of objects.

```
# file: datawrapper.rb
class DataWrapper
   def self.wrap(file_name)
     data = File.new(file_name)
     header = data.gets.chomp
     data.close
     puts header # => name,country
     # in the end we return the class name
  end
end
```

Now let's start writing a small program **testdatawrapper.rb** that uses the above. Let's read our **location.txt** file.

```
#testdatawrapper.rb
require 'datawrapper'
DataWrapper.wrap("location.txt")
```

Coming back to our **datawrapper.rb** program, let's create a new class and give it a suitable name:

```
# file: datawrapper.rb
class DataWrapper
  def self.wrap(file_name)
    data = File.new(file_name)
    header = data.gets.chomp
    data.close
    class_name = File.basename(file_name, ".txt").capitalize
    klass = Object.const_set(class_name, Class.new)
    klass # we return the class name
  end
end
```

The variable **klass** refers to our new class. If the file was called **location.txt**, the class will be named **Location**.

Let us run our modified program **testdatawrapper.rb**.

```
#testdatawrapper.rb
require 'datawrapper'
data = DataWrapper.wrap("location.txt") # Capture return value
puts data # => Location
```

Now, let's start to add attributes to it. The first line of data is a list of names. Let's turn it into a simple array of strings by splitting on the comma character. The modified **datawrapper.rb** program is:

```
# file: datawrapper.rb
class DataWrapper
  def self.wrap(file_name)
    data = File.new(file_name)
    header = data.gets.chomp
    data.close
    class_name = File.basename(file_name, ".txt").capitalize
```

```
    klass = Object.const_set(class_name, Class.new)
    # get attribute names
    names = header.split(",")
    p names # => ["name", "country"]
    klass # we return the class name
  end
end
```

Now we can use **class_eval** in the context of our new class **klass**. At the same time, we'll define an **initialize** method. Also, we shall write a **to_s** method so that we can use **puts**; and let's also **alias** that to **inspect** for convenience. The modified **datawrapper.rb** program is:

```
# file: datawrapper.rb
class DataWrapper
  def self.wrap(file_name)
    data = File.new(file_name)
    header = data.gets.chomp
    data.close
    class_name = File.basename(file_name, ".txt").capitalize
    klass = Object.const_set(class_name, Class.new)
    # get attribute names
    names = header.split(",")
    klass.class_eval do
      attr_accessor *names
      define_method(:initialize) do |*values|
        names.each_with_index do |name, i|
          instance_variable_set("@"+name, values[i])
        end
      end
      define_method(:to_s) do
        str = "<#{self.class}:"
        names.each {|name| str << " #{name}=#{self.send(name)}
        str + ">"
      end
      alias_method :inspect, :to_s
    end
    klass # we return the class name
  end
end
```

Next, we write a class-level method that does a **read** of an entire file and returns an array of objects representing its contents. Because it's a class method, we're just adding a singleton onto an object **klass** which happens to be a class. The modified **datawrapper.rb** program is:

```
# file: datawrapper.rb
class DataWrapper
```

```
  def self.wrap(file_name)
    data = File.new(file_name)
    header = data.gets.chomp
    data.close
    class_name = File.basename(file_name, ".txt").capitalize
    klass = Object.const_set(class_name, Class.new)
    # get attribute names
    names = header.split(",")
    klass.class_eval do
      attr_accessor *names
      define_method(:initialize) do |*values|
        names.each_with_index do |name, i|
          instance_variable_set("@"+name, values[i])
        end
      end
      define_method(:to_s) do
        str = "<#{self.class}:"
        names.each {|name| str << " #{name}=#{self.send(name)}
        str + ">"
      end
      alias_method :inspect, :to_s
    end
    def klass.read
      array = []
      data = File.new(self.to_s.downcase+".txt")
      data.gets  # throw away header
      data.each do |line|
        line.chomp!
        values = eval("[#{line}]")
        array << self.new(*values)
      end
      data.close
      array
    end
    klass # we return the class name
  end
end
```

Let us now modify our program **testdatawrapper.rb** and test out the program **datawrapper.rb**.

```
#testdatawrapper.rb
require 'datawrapper'
klass = DataWrapper.wrap("location.txt")
list = klass.read
list.each do |location|
  puts("#{location.name} is from the #{location.country}")
end
```

Now let's look at a totally different data file i.e. **twitter.txt**. Our **testdatawrapper.rb** is as follows:

```
#testdatawrapper.rb
```

```
require 'datawrapper'
klass = DataWrapper.wrap("twitter.txt")
list = klass.read
list.each do |twitter|
  puts("#{twitter.twitterid}'s site is #{twitter.url}")
end
```

Even if we add another field to the data file, none of our code in the program **datawrapper.rb** would have to change. This is a an exercise and an example of the kind of metaprogramming that Ruby allows.

1.7 Scope

References

- An Exercise in Metaprogramming with Ruby.
- Metaprogramming Ruby - Author: Paolo Perrotta.
- Metaprogramming in Ruby: It's All About the Self.
- Programming Ruby 1.9 - Author: Dave Thomas.
- Seeing Metaclasses Clearly.
- The Book Of Ruby - Author: Huw Collingbourne.
- The Ruby Object Model and Metaprogramming screencasts with Dave Thomas.
- Understanding Ruby Singleton Classes.

<Home | Prev

Note: The material in these study notes is drawn primarily from the above references. Our acknowledgment and thanks to all of them.
This page was last updated on 16th Dec. 2009.