Ruby Metaprogramming

CSCI400

21 September 2017

Color Key

- Clickable URL link
- Write down an answer to this for class participation
- Just a comment – don't confuse with yellow

Overview

Metaprogramming

- Treating programs as data
  - Writing *programs that write programs* (!)
- Common application: domain-specific languages (DSL)

Familiar Example (1/2)

```
class Pikachu
    attr_accessor :name
    attr_reader :level

    def initialize(name, level)
        @name = name
        @level = level
    end
end
```

Familiar Example (2/2)

Let's replicate `attr_*`

Check out this Ruby source file

Quick Exercise

- Goal: Create small class with a few instance vars
- Start with source file on previous slide
  - Use to create getters/setters
- *Hint:* `require_relative` and `include`

`require` and `include`

Recall

- `require`
  - Like `include` in other languages (e.g. C++)
  - Runs another source file
  - Ensures its not `required` twice
- `include`
  - Imports modules for use as *mix-ins*

DSL Overview

Domain-Specific Language

- General-purpose language (GPL)
  - Used to solve problems in many domains
  - E.g. Python, Java, Ruby, C++, Haskell, ...
- Domain-specific languages
  - Used to solve problems within in *specific domain*

DSL Examples

Language	Domain
HTML	Web pages
Mathematica	Symbolic math
GraphViz	Drawing graphs
VHDL	Hardware description
YACC	Define parsers
Regular Expressions	Lexers
SQL	Relational-database queries

Internal vs. External DSL

- Internal DSL
  - AKA *embedded* DSL
  - DSL *within* a (more) general-purpose language
- External DSL (eDSL)
  - *Independent* of any other language
  - Compiled/interpreted (like a GPL)

Short post describing the distinction

Don't need to know the difference for exam

Quiz DSL Exercise

Source Files

Download

Goal

```
Question: Who was the first president of the USA?
1 - Danny Brown
2 - Friedrich Nietzsche
3 - George Washington
4 - Dan Harmon
Enter answer: 1

question 'Complete Airpline (1980) quote: "Don\'t call me _____."'
1 - in the morning
2 - Maeby Funke
3 - crazy
4 - Shirley
Enter answer: 4
You got 1 answers correct out of 2.
```

The Program File

Users create the questions – *We don't parse, we just execute*

```
question 'Who was the first president of the USA?'
wrong 'Danny Brown'
wrong 'Friedrich Nietzsche'
right 'George Washington'
wrong 'Dan Harmon'

question 'Complete Airpline (1980) quote: "Don\'t call me _____."'
wrong 'in the morning'
wrong 'Maeby Funke'
wrong 'crazy'
right 'Shirley'
```

`question`, `right`, and `wrong` are methods

Reading the Data

```
def question(text)
    puts "Just read a question: #{text}"
end
def right(text)
    puts "Just read a correct answer: #{text}"
end
def wrong(text)
    puts "Just read an incorrect answer: #{text}"
end

load 'questions.qm'
```

Source: questionsv1.rb

Responding to the DSL

- What should this quiz DSL really do?
  - *Doesn't* specify how to run a quiz
  - *Does* specify questions + answers – i.e. the program data
- `question` method
  - Create new entry in list of questions
- `right`, `wrong` methodS
  - Set the 'values' for that entry

Store the Questions

See: `Quiz` class in `quiz.rb`

Handle One Question

```
class Question
  def initialize(text)
    @text = text
    @answers = []
  end
  def add_answer(answer)
    @answers << answer
  end
end
```

Source: quiz.rb

What's an answer?

Need answer text and whether right/wrong

```
class Answer
  attr_reader :text, :correct
  def initialize(text, correct)
    @text = text
    @correct = correct
  end
end
```

Let's Load the Data

```
def question(text)
  Quiz.instance.add_question Question.new(text)
end

def right(text)
  Quiz.instance.last_question.add_answer Answer.new(text,true)
end

def wrong(text)
  Quiz.instance.last_question.add_answer Answer.new(text,false)
end
```

quiz.rb (not in a class)

Are We Loading the Data?

Check with `Minitest`: `questions_test.rb`

Now Let's Make it a Quiz (1/2)

```
class Quiz
  def run_quiz
    count=0
    @questions.each { |q| count += 1 if q.ask }
    puts "You got #{count} answers correct "\
         "out of #{@questions.size}."
  end
end
```

Now Let's Make it a Quiz (2/2)

```
class Question
  def ask
    puts ""
    puts "Question: #{@text}"
    @answers.size.times do |i|
      puts "#{i+1} - #{@answers[i].text}"
    end
    print "Enter answer: "
    answer = gets.to_i - 1
    return @answers[answer].correct
  end
end
```

Run It!

In `quiz_runner.rb`:

```
require './quiz.rb'
Quiz.instance.run_quiz
```