Ruby Reflection

CSCI400

16 September 2017

Color Key

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

Reflection

a.k.a. *introspection*

Program can examine/modify its own state

- Set variables
- Call methods
- Add new methods
- Define new objects

Objects getting existential. . .

- What class am I?
  - `o.class`
- Am I an X? (synonyms)
  - `o.is_a?`
  - `o.kind_of?`
- What else can I do?
  - `o.methods`
- Who's my parent?
  - `o.superclass`
- Can I do this?
  - `o.respond_to?`
- What are my variables?
  - `o.local_variables`
  - `o.global_variables`
  - `o.instance_variables`

Design Philosophy

As to why both is_a? and kind_of? exist: I suppose it's part of Ruby's design philosophy. Python would say there should only be one way to do something; Ruby often has synonymous methods so you can use the one that sounds better. It's a matter of preference. It may partly be due to Japanese influence: I'm told that they will use a different word for the same number depending on the sentence in order to make it sound nicer. Matz may have carried that idea into his language design.

- Some guy on StackOverflow

Source

Example

```
class Cat
  def initialize(name, age)
    @name = name
    @age = age
  end
  def purr
    puts "purrr"
  end
```

Example

```ruby
class Cat
  def initialize(name, age)
    @name = name
    @age = age
  end
  def purr
    puts "purrr"
  end
end
```

```ruby
cat = Cat.new("Fluffy", 6)
puts cat.class
puts "Cat? #{cat.is_a? Cat}"
puts "String? #{cat.is_a? String}"
puts "Kind of cat? #{cat.kind_of? Cat}"
```

Example 2

```
class Cat
  def show_local
    x = 5
    local_variables.each do |var|
      puts var
    end
  end
end
```

Example 2

```
class Cat
  def show_local
    x = 5
    local_variables.each do |var|
      puts var
    end
  end
```

```
cat.instance_variables.each do |var|
  puts var
end
cat.show_local
```

Instance Variable Manipulation

Look up: `instance_variable_set`, `instance_variable_get`

Why might we do this?

```
class MyTest < Minitest::Test
  test_something
    # ...
  end
end
```

`test_something` gets run automatically, but how?

Ok, why else?

- Consider a game program
  - User has powerful game piece
  - Rolls die, is able to spawn new object of same type
- How can we create a new instance of that class?
  - `player.class.new(params)`

Sounds pretty useful!

- What is reflection, + why is it useful?
  - Skim up to `dump` method example (3rd answer)
- Briefly describe something cool you can do with reflection

Cool, anything else?

- Ruby on Rails & Active Record
- Object-relational mapper (ORM)
  - Database table:
    - Rows = objects
    - Columns = instance vars
- Objects act as *interface* to database

Simple example

Open and run: message_framework.rb

Another use of Reflection

- Given an *arbitrary* object. . .
  1. Prompt user for input
  2. Modify object using that input

In-Class Challenge

Use reflection + user input to create class instance

```
Enter the information for your Pikachu
Enter the name: Sparky
Enter the level: 5
Sparky is a level 5 Pikachu
```

Start with this

More Challenge Details

- *Write a function*[*] that. . .
  - Accepts an object
  - *Uses reflection* to prompt user for input
  - *Uses reflection* to store user input in object
- Can verify with `puts <object>`

[*] Not a class method

Questions to consider...

- What other languages have reflection?
- Is it used for different purposes depending on the language?