

SUNY_Binghamton: CS445 Software Engineering (U)

- Getting Started Overview
- W0M:
 Overview and
 Introduction to
 Software
 Engineering
 (Week 0,
 Monday Aug.
 31)
- WOW:
 Beautifully
 Engineered
 Software, Plan
 & Document
 vs Agile (Week
 0, Wednesday
 Sept. 2)
- W1W: Introduction to Ruby (Week 1, Wednesday Sept. 9)
- W2W: More Ruby and Intro to BDD & TDD (Week 2, Wednesday Sept. 16)
- ▼ W3M: SaaS

HW 1-4: RUBY METAPROGRAMMING (100/100 points)

```
Specs: spec/attr_accessor_with_history_spec.rb
```

In lecture we saw how <code>attr_accessor</code> uses metaprogramming to create getters and setters for object attributes on the fly.

Define a method <code>attr_accessor_with_history</code> that provides the same functionality as <code>attr_accessor</code> but also tracks every value the attribute has ever had:

```
class Foo
  attr_accessor_with_history :bar
end

f = Foo.new
f.bar = 3  # => 3
f.bar = :wowzo # => :wowzo
f.bar = 'boo!' # => 'boo!'
f.bar_history # => [nil, 3, :wowzo]
```

(Calling bar_history before bar 's setter is ever called should return nil .)

History of instance variables should be maintained separately for each object instance. that is:

```
f = Foo.new
f.bar = 1 ; f.bar = 2
g = Foo.new
g.bar = 3 ; g.bar = 4
g.bar_history
```

then the last line should just return [nil,3], rather than [nil,1,3].

If you're interested in how the template works, the first thing to notice is that if we define <code>attr_accessor_with_history</code> in class <code>Class</code>, we can use it as in the snippet above. This is because a Ruby class like <code>Foo</code> or <code>String</code> is actually just an object of class <code>Class</code>. (If that makes your brain hurt, just don't worry about it for now. It'll come.)

Architecture and REST (Week 3, Monday Sept. 21)

ESaaS Ch. 2.1-2: The Web as a Client-Server System; TCP/IP intro (13:25)

ESaaS Ch. 2.3: HTML+CSS (9:33)

ESaaS Ch. 2.4: 3-tier sharednothing architecture & scaling (11:53)

ESaaS Ch. 2.5: Model-View-Controller (8:06)

Homework 1:
More Ruby (Due
Tues. 9/29 at
Midnight)
Homework 1 due Oct
06, 2015 at 05:00 UT®

W3.0M - Goals and Activities for Week 3, Monday

W3.2M -Background: Introduction to Git and HTML

W3.3M - Activities

W3.4M -Preparation for Monday, Sept. 28 The second thing to notice is that Ruby provides a method <code>class_eval</code> that takes a string and evaluates it in the context of the current class, that is, the class from which you're calling <code>attr_accessor_with_history</code>. This string will need to contain a method definition that implements a setter-with-history for the desired attribute <code>attr_name</code>.

HINTS:

- Don't forget that the very first time the attribute receives a value, its history array will have to be initialized.
- An attribute's initial value is always nil by default, so if
 foo_history is referenced before foo has ever been assigned,
 the correct answer is nil, but after the first assignment to foo,
 the correct value for foo_history would be [nil].
- Don't forget that instance variables are referred to as @bar within getters and setters, as Section 3.4 of the ESaaS textbook explains.
- Although the existing <code>attr_accessor</code> can handle multiple arguments (e.g. <code>attr_accessor:foo,:bar</code>), your version just needs to handle a single argument.
- Your implementation should be general enough to work in the context of any class and for attributes of any (legal) variable name
- Note that one powerful metaprogramming feature in Ruby is class_eval that can be called in the meta-class Class. class_eval can interpret a string on the fly to create some new code. In the example below, we define add_method() to the meta-class (and, through inheritance, to any class). When called, this method defines a new method that return 42 (notice how #{name} gets replaced with the parameter passed to add_method).

▶ W4M: SaaS

Architecture and REST (Week 4, Monday Sept. 28)

- W4W: Rails Intro (Week 4, Wednesday Sept. 30)
- W5M: Rails cont. (Week 5, Monday Oct.5)
- W5W: Enhancing SaaS with JavaScript (Week 5, Wednesday Oct. 7)
- W6M: Agile Methodology: Working with the Customer (Week 6, Monday Oct. 12)
- W6W: BDD
 with
 Cucumber and
 Capybara
 (Week 6,
 Wednesday
 Oct. 14)

```
class Class
  def add_method(name)
    class_eval %Q"
     def #{name}()
        42
     end
     "
  end
end

class MyClass
  add_method :my_method
end

mc = MyClass.new
puts mc.my_method # => 42
```

Browse... No files selected.

```
On Time
#attr accessor with history
  should remember history separately for each instance [30 points]
  when a symbol is passed [10 points]
    should define getter and setter [5 points]
   setter should return value set to [5 points]
   should work if getter used first [10 points]
   should work if setter used first [20 points]
    should remember values [10 points]
 when a string is passed [10 points]
   should define getter and setter [5 points]
    setter should return value set to [5 points]
    should work if getter used first [10 points]
   should work if setter used first [20 points]
    should remember values [10 points]
Finished in 0.01768 seconds
11 examples, 0 failures
```

SUBMIT URL TO PAIRING VIDEO (SCREENCAST)

(10 points possible)

Please submit the URL to an unlisted youtube video recording (screencast) of your pairing session on this assignment below.

- W7M: TDD with RSpec (Week 7, Monday Oct. 19)
- W7W: TDD with RSpec cont. and Review So Far (Week 7, Wednesday Oct. 21)
- W8M: Wrap Up and Assessment of Part 1 (Week 8, Monday, Oct. 26)
- W8W: ProjectPoster Session
- W9M: Introduction to Part 2 and Advanced Rails (Week 9, Monday Nov.
 2)
- W9W:
 Advanced Rails
 (Week7,
 Wednesday,
 Nov. 4)
- ▶ W10M:

?

If you are unable to access YouTube and/or G+, feel free to submit a link to a video hosted on some other service.

Note: we are hoping to see screencasts with screen sharing plus text chat, or even better, audio chat, but video from webcams are not required.

Refactoring & Legacy (Week 10, Monday Nov. 9)

- W10W: Refactoring & Legacy (Week 10, Nov. 11)
- W11M: Project Management (Week 11, Monday Nov. 16)
- W11W: Project Management (Week 11, Wednesday, Nov. 18)
- W12M: More Enhancing SaaS with Javascript (Week 12, Monday Nov. 23)
- W13M: Design Patterns for SaaS (Week 13, Monday Nov. 30)
- W13W: Design Patterns for SaaS (Week 13,

5 of 7

Wednesday, Dec. 2)

► W14M:

Practical DevOps:

Deployment,

Upgrades,

Performance,

Security (Week

14, Monday

Dec. 7)

▶ W14W:

Practical

DevOps:

Deployment,

Upgrades,

Performance,

Security (Week

14,

Wednesday

Dec. 9)

▶ W15M:

EarlyBird Project Demos (Monday, Dec. 14)

▶ W15W-W16T:

Project Demos and Final

Exam

(Wednesday-

Friday and

Monday and

Tuesday Dec.

16-18 and Dec

21-22)

Bonus Videos

© All Rights Reserved

7 of 7 10/05/2015 08:49 PM