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

Specs: `spec/attr_accessor_with_history_spec.rb`

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

Define a method `attr_accessor_with_history` that provides the same functionality as `attr_accessor` 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 `attr_accessor_with_history` in class `Class`, we can use it as in the snippet above. This is because a Ruby class like `Foo` or `String` is actually just an object of class `Class`. (If that makes your brain hurt, just don't worry about it for now. It'll come.)

The second thing to notice is that Ruby provides a method `class_eval` that takes a string and evaluates it in the context of the current class, that is, the class from which you're calling `attr_accessor_with_history`. This string will need to contain a method definition that implements a setter-with-history for the desired attribute `attr_name`.

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 `attr_accessor` can handle multiple arguments (e.g. `attr_accessor :foo, :bar`), 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).

```
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: Project Poster 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,

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