Technical Explorations

by Keith R. Bennett

The Case for Nested Methods in Ruby

Nov 7, 2015 • keithrbennett

The wisdom of using local variables has been internalized in all of us from the beginning of our software careers. If we need a variable referring to data that is used only in a single method, we create a local variable for it.

Yet if it is logic to which we need to refer, we make it an instance method instead.

In my opinion, this is inconsistent and unfortunate. The result is a bloated set of instance methods that the reader must wade through to mentally parse the class. The fact that some of these methods are used only by one other method is never communicated by the code; the reader has to discover that for him/herself.

The number of possible interactions among the instance methods is one of many measures of our software's complexity. The number of possible instance method interactions is `(method_count * (method_count) - 1)`. Using this formula, a class with 10 methods will have a complexity of 90. If 4 of those methods are used by only 1 other method, and we could move them inside those methods, the complexity would plummet to 30 (`6 * (6 - 1)`), a third of the original amount!

While it is possible to extract subsets of these methods into new smaller classes, this is not always practical, especially in the case of methods called only by the constructor.

Fortunately, we do have lambdas in Ruby, so I will sometimes create lambdas inside methods for this purpose. However, lambdas are not as isolated as methods, in that they can access and modify local variables previously defined outside their scope. Furthermore, the lambdas can be passed elsewhere in the program and modify those locals from afar! So using methods would be cleaner and safer.

Another weakness of using lambdas for this purpose is that, unlike methods that are created at interpret time, lambdas are objects created at runtime – so if a method creating 2 lambdas is called a million times in a loop, you'll need to create and garbage collect another 2 million objects. (This can be circumvented by defining the lambdas as class constants or assigning them to instance variables, but then they might as well be instance methods.)

I realize that implementing this feature would be a substantial undertaking and may not be feasible at this time. That said, I think it would be useful to discuss this now so we might benefit from its implementation someday.

* * * *

(Much of this content is communicated in my talk on Ruby lambdas; slide show is at https://speakerdeck.com/keithrbennett/ruby-lambdas-functional-conf-bangalore-oct-2014 and YouTube video of the presentation at FunctionalConf in Bangalore at https://www.youtube.com/watch?v=hyRgf6Qc5pw.)

Published with GitHub Pages