Ruby Domain-Specific Language Workshop

Intro

Set expectations
Show how each step is on its own branch
Show how to run the tests

What is a Domain Specific Language?

Martin Fowler defines a DSL as:

"... a computer language that's targeted to a particular kind of problem, rather than a general purpose language that's aimed at any kind of software problem."

Examples:
CSS, regular expressions, Rails route file, Factory Bot, Retriever Data Mappers

Show CRM routes file and Retriever data mappers as examples

Blocks, Procs, and Lambdas

1. Show how map works in irb console
2. Write test for custom map function first (TDD)
3. Show example of map function with implicit block with yield
4. Refactor example to use explicit block
5. Couple minutes to have them code their own select method using explicit and implicit blocks with TDD
6. Show how to code the select method. Write test first and then implement.
7. Show how lambdas can be turned into blocks with &

Basic DSL

1. Explain motivation for Factory gem. We want a simple way to build different instances of a class for our tests.
2. Implement Bicycle(style, size, color, weight, gears)
3. Write a factory definition for bicycle in before block. Design by wishful thinking.
4. Write test for basic dsl (color: red, gears: 22, style: :road, etc.)
5. Stub out define method, hard code run method with OpenStruct to get tests to pass
6. Implement define method. Just create def, pass it options, and add to factories hash

7. Implement Definition(options, declarations). Implement add_attribute method.
8. Implement Declaration(name, value).
9. Finish implementation of define method using instance_eval
10. Implement run method on Definition. Explain dynamic assignment using send

Method Missing

1. Change add_attribute syntax in definition to more succint syntax. Now we get “undefined method ‘color’ error.
2. Refactor definition to use method_missing(name, *args, &block). Puts method_name and args and call super to show how we can get all the info we need.
3. Implement respond_to_missing?

Traits

Simple

1. Explain motivation for traits. Clunky to define separate factories for road and fixed gear bike.
2. Modify bicycle definition to use traits for road and fixie
3. Create stubbed trait method to avoid exceptions
4. Modify existing test to just test default attributes (size and color)
5. Create two new tests for road and fixed gear traits
6. Modify Factory and Definition run methods to take in single trait attribute. Modify run method to select only declaration with matching traits
   .select { |declaration| declaration.traitless? || declaration.in?(trait) }
7. Default attribute test is back to passing - now we need to figure out how to implement trait method. We need to create a temporary environment where all declarations are injected with the current trait. Give team 5 minutes to try to figure it out.

Definition Proxy

1. Definition is starting to have multiple responsibilities: it’s managing our DSL syntax, the list of declarations, the state of traits, and the construction of entities. Let’s extract the DSL logic and see how it cleans up traits as well.
2. Create DefinitionProxy(definition, options). Extract method_missing, add_attribute, and trait into definition proxy class. Get default attributes test passing first (comment out trait method)
3. Refactor traits method to use definition proxy. Explain reflection with self.class.new

Multiple traits

1. Write test showing how multiple traits can be specified (trait :blue)
2. Show how to use splat args and then give a couple minutes to try to implement on their own

3. Refactor to allow specification of multiple traits

Attribute Overrides

1. Write test for default attributes with overrides
2. Show how to pass hash using double splat. Puts overrides
3. Extract evaluate_declarations and evaluate_overrides methods
4. Write test for traits with overrides

After Hooks

1. Show desired syntax for after hooks

Exercise: Transient Attributes

Show desired API for transient attributes with rider_height example:

```
size { rider_height > 55 ? :large : :small }
color :red

transient do
  rider_height 55
end
```

Conclusion

1. What we learned:
    a. Implicit/explicit blocks and how to use them
    b. How to alter the context a block evaluates in using instance_eval
    c. How to hook into Ruby's method lookup chain using method_missing
    d. How to dynamically set attributes on objects using send
    e. How to write methods with dynamic arguments using splat and double splat
    f. How to acquire an object's class using reflection via self.class