Uncle John's Bathroom Reader

Reusable RSpec 1

by Speedy Spec

You've been hacking away with RSpec now for a couple of weeks, and we're all pretty impressed with your enthusiasm. Unfortunately as more examples have been written, code has been duplicated, so let's talk about code reuse.

RSpec has multiple options for reusing code in your examples, in this issue we'll discuss two: **shared contexts** for reusing set up code and **custom matchers** for reusing validation code.

You may have noticed that some of your specs have gained a lot of repetitive **`before`** and **`let`** blocks such as the **`AccountState`** mocking in issue **#02**. To clean this up we can define a shared context to make it reusable:

```
shared_context "premium account state" do
  let(:state) do
    mock('AccountState Paid Mock', :paid? => true)
  end
  before :each do
    # set up some initial state
  end
end
```

Now that we've defined this shared context, we can use it in multiple places by referring to it with the string we used to define it:

```
describe '#premium?' do
  include_context "premium account state"
  it { should be_premium }
end
```

Custom matchers are more powerful than most other forms of code reuse in RSpec, so writing them is slightly more involved. Let's write a matcher to ensure that our **`User`** object is a valid premium user:

```
RSpec::Matchers.define :be_valid_premium do
  match do |user|  # This block returning false fails the matcher
    (user.created_at > 0) && (user.activated?) && (user.premium?)
  end
end
```

We can then use this in an example just like any of the built in matchers:

```
describe User do
  describe '#register!' do
    subject { User.new.register! }
    it { should be_valid_premium }
  end
end
```

Like most forms of abstraction, both shared contexts and custom matchers must be used judiciously or else examples might become difficult to follow. When in doubt, ask a colleague, that's what they're there for!

That's it for this issue of **Uncle John's Bathroom Reader**, now go wash your hands.

`Shared Context`:
A block containing a number of RSpec keywords (**`before`**, **`let`**, etc) containing set up code which can be reused by referring to it by name.

`Shared Example`:
Block containing examples (**`it`** blocks) to be reused across multiple specs. Especially useful for testing inheritance hierarchies and mixins.

`Helper Method`:
Method defined in a module which can be included in an example group (**`describe`**, **`context`**) to encapsulate common logic.

`Custom Matcher`:
User-defined matcher which can fit into RSpec's existing matcher pipeline. Can abstract common verification logic and provide useful error messages when the matcher fails.

Bathroom Reader News

The week of Dec. 24th - Dec. 28th there will be no new **Uncle John's Bathroom Reader**. If you will be in the office that week, don't forget your smart phone.

Corrections:

In the last issue (#02) we used the incorrect method for making a mock's response throw; **`and_raise`** will raise the argument, whereas **`and_throw`** will throw it, e.g.:

```
m = mock('Mock Object 4')
m.stub(:throwit).and_throw(:failboat)
```

Docs on writing custom matchers: https://github.com/dchelimsky/rspec/wiki/custom-matchers

Collection of tips and tricks for writing better specs: http://betterspecs.org/

Send comments and suggestions for future issues to: **ujbr@lookout.com**