Code Reading Lab: Sinatra

Part 1: Testing

Pull down Sinatra with git:

```
git clone git://github.com/sinatra/sinatra.git
```

Guidelines

- Try not to read the documentation, even the README, during these code reading exercises.
- Documentation can lie, but code generally doesn't.
- Ask us anything.

We're going to be looking at some of the tests first. Start by looking around in the *test/* directory. Here are some questions to get you started:

Questions

1. What testing framework does Sinatra use? How would you figure this out, in general?
2. As an example of Sinatra's style of testing, look in *test/base_test.rb*. Why is the test code interleaved with the class definitions? What benefit does this confer?
3. Some of the tests have assertions like `assert ok?` or `assert redirect?`. How does this work?
4. Look in *test/contest.rb*. This defines some helper methods that the Sinatra test uses. What does the `self.context` method do, under the hood? Why?
5. Do you see any potential issues with using the class hierarchy to define contexts in this way? Any potential interactions with the testing framework?
6. Routes can take conditions. How does code like this work? You'll have to start digging into the framework's code to track this one down.

```
user_agent(/Foo/) # sets a condition on the following route
get '/foo' do
  'foo'
end

get '/foo' do
  'not foo'
end
```

In addition to reviewing these questions, please note anything you see in the code that interests you, or anything you have questions about. We'll be taking some time soon to share things we found through looking at the code and discuss any points of confusion along the way.

Code Reading Lab: Sinatra

Part 2: Routing and Request Processing

We continue our look at Sinatra's code by digging into the inner workings of the framework. Most of Sinatra's actual code is in *lib/sinatra/base.rb*, while a few bits of plumbing and setup are in the remaining files.

Questions

1. Sinatra applications are ultimately just classes, and they can be inherited from. Many of the settings, such as Sinatra options and routes, are passed from superclass to subclass. How does this work?
2. How do instance variables get passed from the controller code into the views, as in the following example?

```
get "/" do
  @text = "Hello, world!"
  haml %{%p= @text}
end
```

3. What is required to write an extension and register it with Sinatra? Can you describe the extension API just by reading Sinatra's code? (Extensions are registered using the `register` method -- start your search there.)
4. What is the purpose of this bit of code? It seems fairly convoluted. (You may have to know a bit about Rack to understand the purpose of this code.)

```
def body(value=nil, &block)
  if block_given?
    def block.each ; yield call ; end
    response.body = block
  else
    response.body = value
  end
end
```

Code Reading Lab: Sinatra

Part 3: Plumbing

At some level, this is the most fundamental part of a web framework -- getting it to actually serve requests. Sinatra's job is made a lot easier, because it is built on the Rack webserver interface. This also makes it compatible with many web servers without having to write an interface for all of them. Here are some more questions to guide you through the code that actually makes Sinatra tick:

Questions

1. A Sinatra application can be fairly complex, and can take advantage of Sinatra's modular and object-oriented design, but applications can also be extremely simple. Here's an example:

   ```
   # hello_world.rb
   require 'rubygems'
   require 'sinatra'
   get('/') { "Hello, world!" }
   ```

   There appears to be nothing here telling Sinatra to start a server. How does it know to start one up?
2. When the above application is run, what actually happens, at the modular Sinatra level? In other words, where and how is the actual Sinatra application created?
3. How is the following irb session functionally different from the code in #1? Why doesn't it start a server? It's the exact same code.

   ```
   $ irb -rubygems
   irb(main):001:0> require 'sinatra'
   => true
   irb(main):002:0> get('/') { "Hello, World" }
   => [/^\/$/, [], [], #<Proc:0x02cb27d8@c:/Ruby/lib/ruby/gems/1.8/gems/
   sinatra-0.9.4/lib/sinatra/base.rb:779>]
   irb(main):003:0> exit
   $
   ```

4. Sinatra supports many deployment options: Thin, Mongrel, FastCGI, Passenger, etc. When you start up a Sinatra server, how does it know which application server to use?
5. Why does `Sinatra::Base#call` (*lib/sinatra/base.rb*:360) dup itself and then delegate to `#call!` ?