Time	~ 1 - 2 hours
Learning Goals	• Write rspec tests to implement all of the pending specs in the given files • Understand the test-drive development cycle

Writing good object-oriented tests is not always easy. A good test relies not only on the fact that the code is well-written, but that the test itself is well-written.

When you write *examples* (tests in RSpec are called examples), you should be consuming the public interface of your code.

What does this mean? In the simplest explanation, it means that your examples should test the inputs and outputs of the publicly defined methods for a given class. Internal variables and methods marked `private` or `protected` are off-limits. Your tests need to follow the same OOP rules as your objects.

In this challenge, you will be working with a feature-complete program. However, much of the application code is commented out and most of the specs are still pending.

Your task is to implement all of the pending specs. You will need to utilize the RSpec API including:

- RSpec expectations
- Equality matchers
- Type matchers

RSpec is designed to read very fluidly, but don't be fooled by the syntactic sugar it provides. It is still just Ruby.

Objectives

Start Challenge

Fork and clone the repo to your computer.

Orient yourself

Before you even begin figuring out how to write your specs, look at the existing specs. Take note of the structure and look at how the pieces fit together.

Then run the specs with

```
$ rspec cookie_spec.rb --format documentation --color
```

and study the output.

After you have oriented yourself, you should be able to provide at least an educated guess when answering the following questions:

- What does a `describe` block do?
- What does a `context` block do?
- How are `describe` blocks used differently from `context` blocks?
- What are the arguments for the `it` method?
- What does the `let` method do?
- How is `expect` used differently from `should`?

You don't need to know all of the answers now, but you should be able to generate a rough idea of how the syntax works from the provided code.

Take note of the structure of the specs. There are some real patterns at work here:

- A `describe` block for each *public method* of the class, using `#` to indicate an instance method.
- Use of `let` to define objects which will be used throughout the specs.
- Use of the `context` block to identify different environments and states
- Descriptions are written to form complete sentences which read as clear *specifications* or *expectations*, for example:

```
Cookie
  #status
    when unbaked
      is `:doughy`
```

Write the pending examples

Most of the specs in `cookie_spec.rb` are still *pending*, i.e. they have a description but are not actually testing anything.

Your task is implement each of the pending specs one at a time, make sure that they fail, and then uncomment the *minimum amount of code possible* to make them pass.

This is an important cycle to get used to for test driven development exercise:

1. Write a spec
2. Ensure that the spec *fails* when the relevant code is missing (or commented out)
3. Ensure that the spec *passes* when the relevant code is implemented

Use this cycle to fill out all of the pending examples in `cookie_spec.rb`.