Time	~ 2 hours
Learning Goals	• Learn how to translate a pre-existing object model into that's easy for us to m • Learn how to write a simple scraper

The git repo for this challenge: https://github.com/NextAcademy/hackernews-objects (remember to fork and clone!)

We're going to write a simple Hacker News (HN) scraper. We have to be polite, though: if we're too aggressive we'll get the NEXT Academy network banned. We will also be building Ruby classes that represents a particular Hacker News comment thread.

Objectives

Save a HTML Page

First, we're going to save a specific post as a plain HTML file for us to practice on. As we're developing a scraper we'll be tempted to hammer the HN servers, which will almost certainly get everyone temporarily banned. We'll use the local HTML file to get the scraper working before we create a "live" version.

Note that this implies something about how your class should work: it shouldn't care *how* it gets the HTML.

Visit the Hacker News homepage and click through to a specific post. If you can't find one, use the A/B Testing Mistake post. You can save the HTML for this page somewhere using one of the two following ways:

1. Right click the page and select "view source." Copy and paste the HTML into Sublime and save the file.

2. Alternatively, open Terminal, make a `hn_scraper` directory, `cd` into it, and run the following command:

```
$ curl https://news.ycombinator.com/item?id=5003980 > post.html
```

This will create a `post.html` file which contains the HTML from the URL you entered. You're free to enter another URL.

Playing around with Nokogiri

You would have already dealt with Nokogiri in the previous challenge. For this challenge, try this from `irb`:

```
doc = Nokogiri::HTML(File.open('post.html'))
```

Note: Make sure you're in *the same directory as* *`post.html`*.

Here's an example of how you'd write a method that takes a Nokogiri document of a Hacker News thread as input and return an array of commentor usernames:

```
def extract_usernames(doc)
  doc.search('.comhead > a:first-child').map do |element|
    element.inner_text
  end
end
```

What do these other Nokogiri calls return?

```
doc.search('.subtext > span:first-child').map { |span| span.inner_text}


doc.search('.subtext > a:nth-child(3)').map {|link| link['href'] }


doc.search('.title > a:first-child').map { |link| link.inner_text}


doc.search('.title > a:first-child').map { |link| link['href']}


doc.search('.comment > font:first-child').map { |font| font.inner_text}
```

What is the data structure? Can you call Ruby methods on the returned data structure?

Make sure you open up the HTML page in your browser and poke around the source code to see how the page is structured. What do their tags actually look like? How are those tags represented in the Nokogiri searches?

Choose at least one other attribute to add for your comments class. What other comment-related data from the html page can you include? Figure out your own Nokogiri call to pull in the data.

Creating Your Object Model

We want two classes: `Post` and `Comment`. A post has many comments and each comment belongs to exactly one post. Let's build the `Post` class so it has the following attributes: `title`, `url`, `points`, and `item_id`, corresponding to the title on Hacker News, the URL the post points to, the number of points the post currently has, and the post's Hacker News item ID, respectively.

Additionally, create two instance methods:

1. `Post#comments` returns all the comments associated with a particular post
2. `Post#add_comment` takes a `Comment` object as its input and adds it to the comment list.

You'll have to design the `Comment` object yourself. What attributes and methods should it support and why?

We could go deeper and add, e.g., a `User` model, but we'll stop with `Post` and `Comment`.

Loading Hacker News Into Objects

We now need code which does the following:

1. Instantiates a `Post` object
2. Parses the Hacker News HTML
3. Creates a new `Comment` object for each comment in the HTML, adding it to the `Post` object in (1)

Ship it!

Command line + parsing the actual Hacker News

We're going to learn two new things: the basics of parsing command-line arguments and how to fetch HTML for a website using Ruby. We want to end up with a command-line program that works like this:

```
$ ruby hn_scraper.rb https://news.ycombinator.com/item?id=5003980
```

```
Post title: XXXXXX
Number of comments: XXXXX
... some other statistics we might be interested in -- your choice ...
```

First, read this blog post about command-line arguments in Ruby. How can you use `ARGV` to get the URL passed in to your Ruby script?

Second, read about Ruby's OpenURI module. By requiring `'open-uri'` at the top of your Ruby program, you can use open with a URL:

```
require 'open-uri'


html_file = open('http://www.ruby-doc.org/stdlib-1.9.3/libdoc/open-uri/rdoc/OpenURI.html')
puts html_file.read
```

This captures the HTML from that URL as a `StringIO` object, which NokoGiri accepts as an argument to `NokoGiri::HTML`.

Combine these two facts to let the user pass a URL into your program, parse the given Hacker News URL into objects, and print out some useful information for the user.