***Accepting request input***

* Some web applications are read-only. Humans poke about on the website in their web browser, reading whatever content the server sends to the browser. The good news is that it doesn’t have to be that way. Many web applications give the user an opportunity to chime in and send data back to the server. Without this capability, the web would be a very different place.
* Spring MVC provides several ways that a client can pass data into a controller’s handler method. These include

- Query parameters

- Form parameters

- Path variables

***Taking query parameters***

One thing that your Spittr application will need to do is display a paged list of spittles. As it is, SpittleController only displays the most recent spittles; it offers no way to page back through the history of the spittles that have been written. If you’re going to let users go through spittle history a page at a time, you’ll need to offer a way for them to pass in parameters that determine which set of spittles to display.

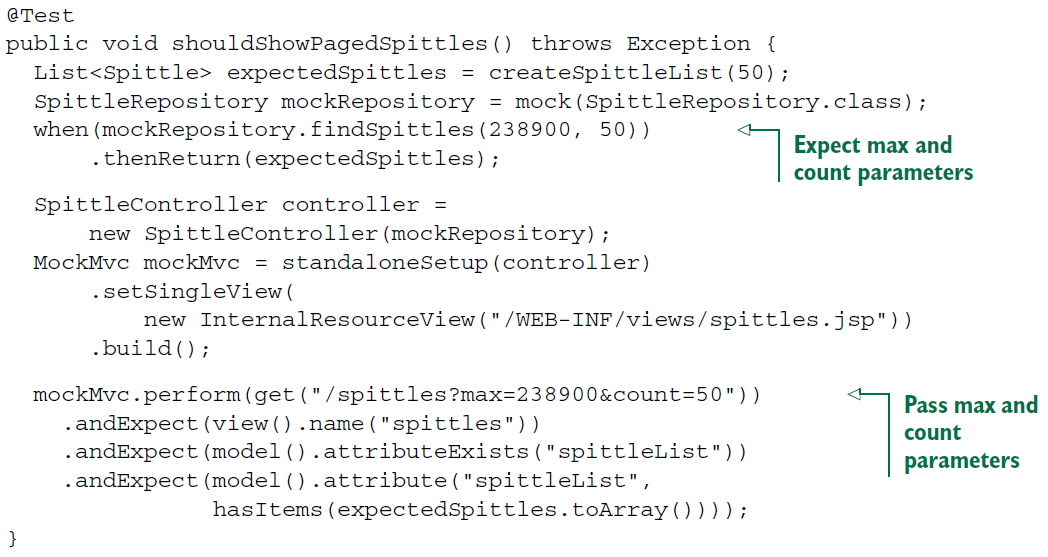
In deciding how to do this, consider that if you’re viewing a page of spittles, it’s ordered with the most recent spittle first. Therefore, the first spittle on the next page should have an ID that is *before* the ID of the last spittle on the current page. So, in order to display the next page of spittles, you should be able to pass in a spittle ID that is just less than the ID of the last spittle on the current page. You can also pass in a parameter saying how many spittles to display.

To implement this paging solution, you’ll need to write a handler method that accepts the following:

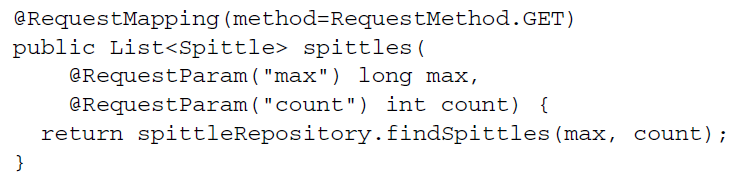
 A before parameter (which indicates the ID of the Spittle that all Spittle objects in the results are before)

 A count parameter (which indicates how many spittles to include in the result) To achieve this, let’s replace the spittles()method you created in listing 5.10 with a new spittles() method that works with the before and count parameters. You’ll start by adding a test to reflect the functionality you want to see from the new

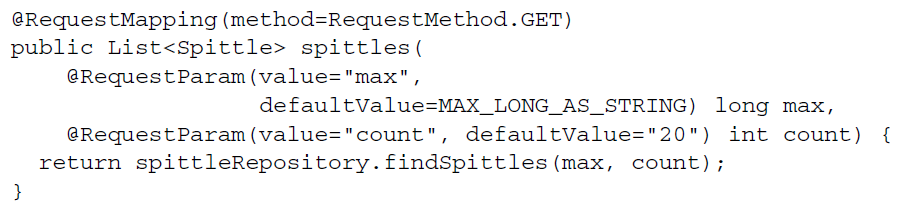
spittles()method.



* The key difference between this test method and the one in listing with no parameter (/spittles) is that it performs a GET request against /spittles, passing in values for the max and count parameters. This tests the handler method when those parameters are present; the other test method tests for when those parameters are absent. With both tests in place, you can be assured that no matter what changes you make to the controller, it will still be able to handle both kinds of requests:



* If the handler method in SpittleController is going to handle requests with or without the max and count parameters, you’ll need to change it to accept those parameters. But still default to Long.MAX\_VALUE and 20 if those parameters are absent on the request. The defaultValue attribute of @RequestParam will do the trick:



* Now, if the max parameter isn’t specified, it will default to the maximum value of Long. Because query parameters are always of type String, the defaultValue attribute requires a String value. Therefore, Long.MAX\_VALUE won’t work. Instead, you can capture Long.MAX\_VALUE in a String constant named MAX\_LONG\_AS\_STRING:



* Even though the defaultValue is given as a String, it will be converted to a Long when bound to the method’s max parameter.
* The count parameter will default to 20 if the request doesn’t have a count parameter.
* Query parameters are a common way to pass information to a controller in a request. Another way that’s popular, especially in a discussion of building resource oriented controllers, is to pass parameters as part of the request path. Let’s see how to use path variables to take input as part of the request path.