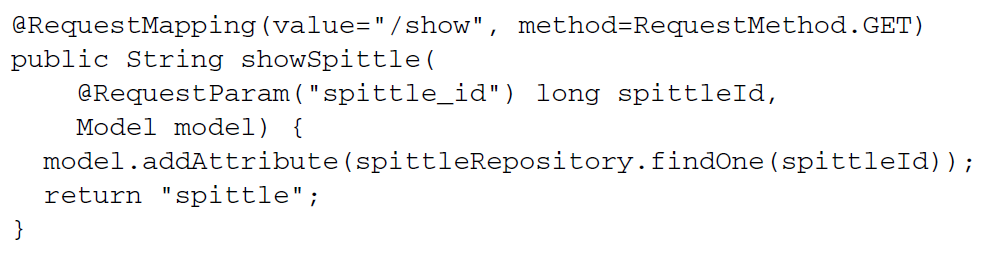
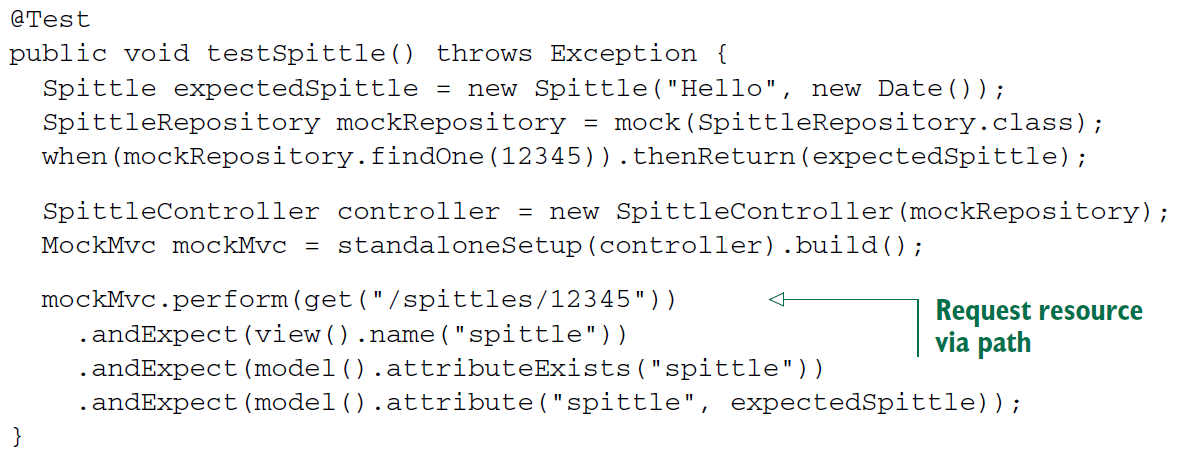
***Taking input via path parameters***

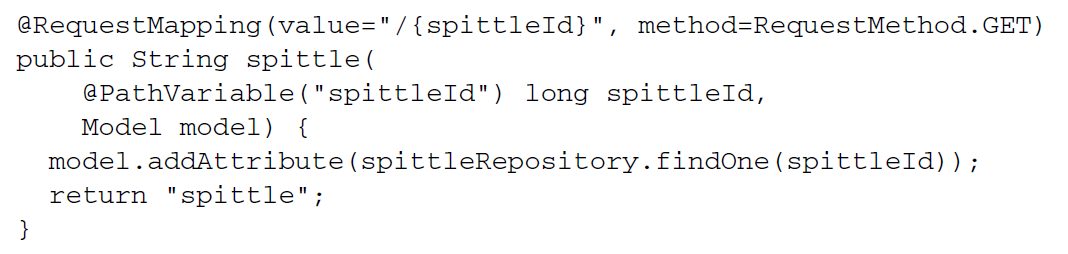
Let’s say your application needs to support the display of a single Spittle, given its ID. One option you have is to write a handler method that accepts the ID as a query parameter using @RequestParam:



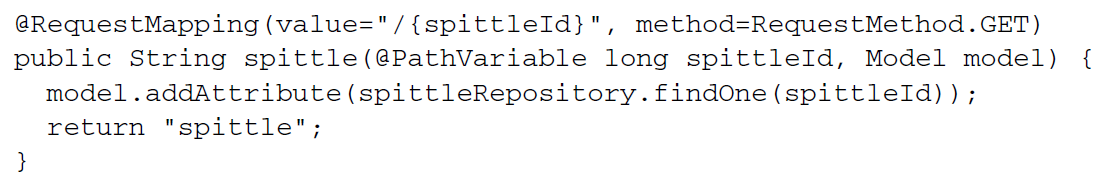
* This handler method would handle requests such as /spittles/show?spittle\_id=12345. Although this could be made to work, it’s not ideal from a resource-orientation perspective.
* Ideally, the resource being identified (the Spittle) would be identified by the URL path, not by query parameters. As a general rule, query parameters should not be used to identify a resource. A GET request for /spittles/12345 is better than one for spittles/show?spittle\_id=12345. The former identifies a resource to be retrieved. The latter describes an operation with a parameter—essentially RPC over HTTP
* With the goal of resource-oriented controllers in mind, let’s capture this requirement in a test. The following listing shows a new test method to assert resourceoriented request handling in SpittleController.



* The most important part of the test is in the last few lines, where it performs a GET request for /spittles/12345 and asserts that the view name is spittle and that the expected Spittle object is placed in the model. Because you haven’t yet implemented the handler method for that kind of request, the request will fail. But you can fix that by adding a new method to SpittleController.
* Up to this point, all of your controller methods have been mapped (via @RequestMapping) to a statically defined path. But if you’re going to make this test pass, you’ll need to write an @RequestMapping that has a variable portion of the path that represents the Spittle ID.
* To accommodate these path variables, Spring MVC allows for placeholders in a @RequestMapping path. The placeholders are names surrounded by curly braces ({ and }). Although all the other parts of the path need to match exactly for the request to be handled, the placeholder can carry any value.
* Here’s a handler method that uses placeholders to accept a Spittle ID as part of the path:



* For example, it can handle requests for /spittles/12345, the path being tested for in above testing.
* As you can see, spittle() has a spittleId parameter that is annotated with @PathVariable("spittleId").This indicates that whatever value is at the placeholder position in the request path will be passed into the handler method’s spittleId parameter. If the request is a GET request for /spittles/54321, then 54321 will be passed in as the value of spittleId.
* Notice that the phrase spittleId is repeated a few times in the example: in the @RequestMapping path, as the value attribute of @PathVariable, and again as a method parameter name. Because the method parameter name happens to be the same as the placeholder name, you can optionally omit the value parameter on @PathVariable.



* If no value attribute is given for @PathVariable, it assumes the placeholder’s name is the same as the method parameter name. This can make the code a little cleaner by not duplicating the placeholder name any more than necessary. But be cautioned: if you decide to rename the parameter, you must also change the placeholder name to match.
* The spittle() method will pass the parameter along to the findOne() method on the SpittleRepository to find a single Spittle object and will add that Spittle to the model. The model key will be spittle, inferred by the type passed in to addAttribute().
* The data in the Spittle object can then be rendered in the view by referring to the request attribute whose key is spittle (the same as the model key). Here’s a snippet of a JSP view that renders the Spittle:

