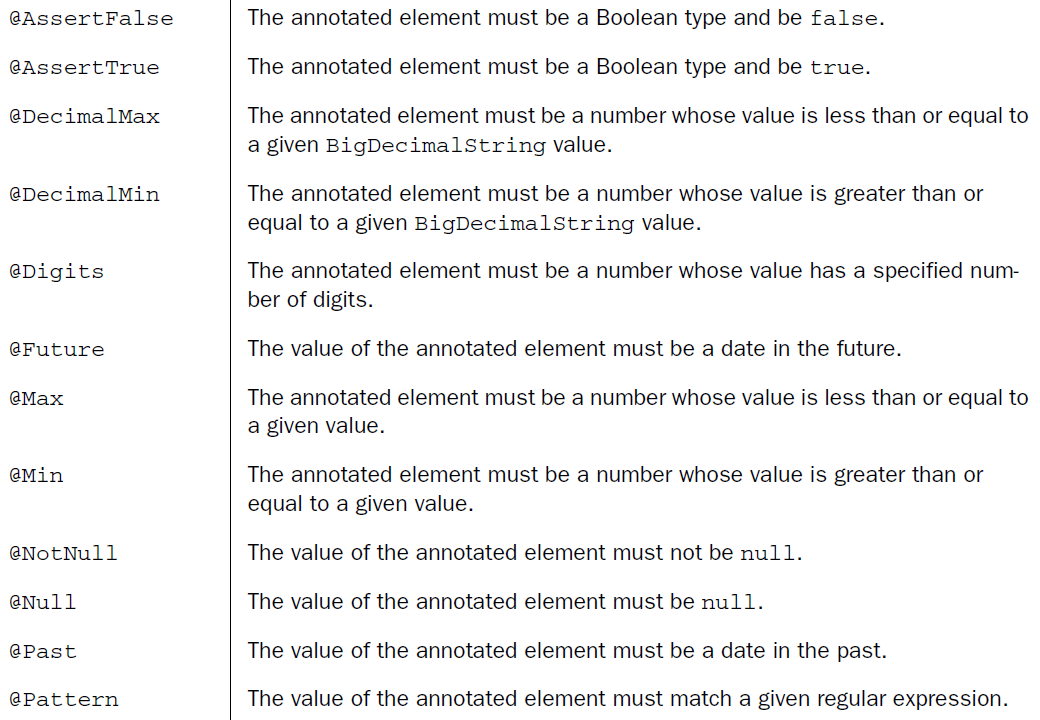
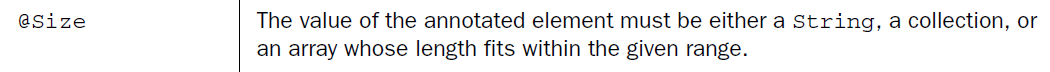
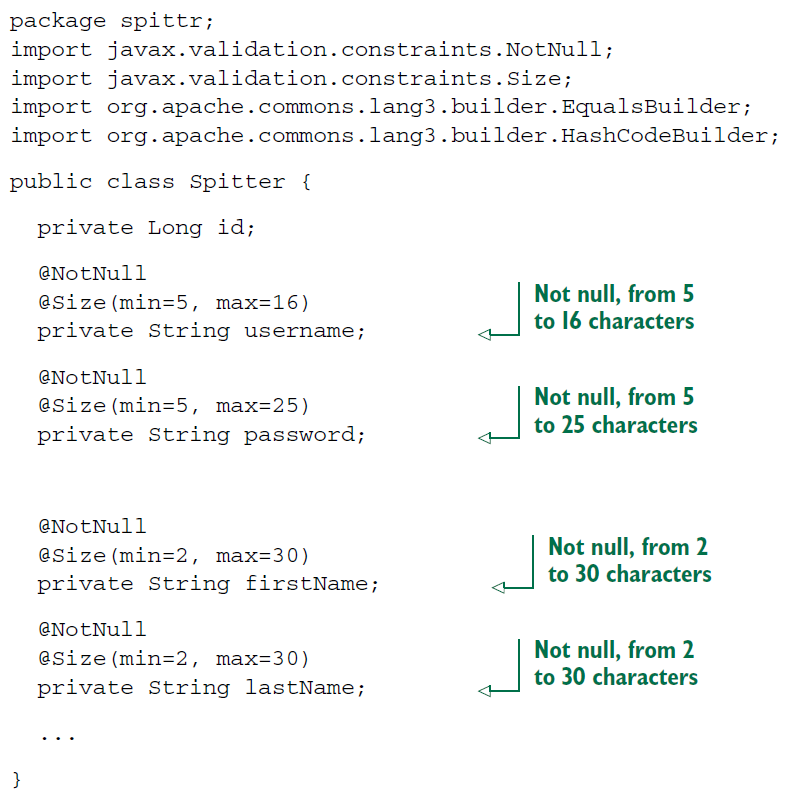
***Validating forms***What will happen if the form doesn’t send a username or password parameter? Or what if the firstName or lastName value is empty or too long? Let’s look at how to add validation to the form submission to prevent inconsistencies in the data presented.

* If the user were to leave the username or password field empty when submitting the form, it could result in the creation of a new Spitter object whose username and password were empty Strings. At the very least, this is odd behavior. But left unchecked, it could present a security concern where anyone could sign in to the application by submitting an empty login form.
* Also, you should take steps to prevent the user from submitting an empty firstName and/or lastName in an effort to maintain some level of anonymity. And it’s probably a good idea to limit the length of the values given in those fields, keeping them at a reasonable size and avoiding misuse of the fields.
* One way to handle validation, albeit naive, is to add code to the processRegistration() method to check for invalid values and send the user back to the registration form unless the data is valid. It’s a short method, so tossing in a few extra if statements won’t do much harm. Right?
* Rather than litter your handler methods with validation logic, however, you can take advantage of Spring’s support for the Java Validation API (a.k.a. JSR-303). Starting with Spring 3.0, Spring supports the Java Validation API in Spring MVC. No extra configuration is required to make Java Validation work in Spring MVC. You just need to make sure an implementation of the Java API, such as Hibernate Validator, is in the project’s classpath.
* The Java Validation API defines several annotations that you can put on properties to place constraints on the values of those properties. All of these annotations are in the javax.validation.constraints package.

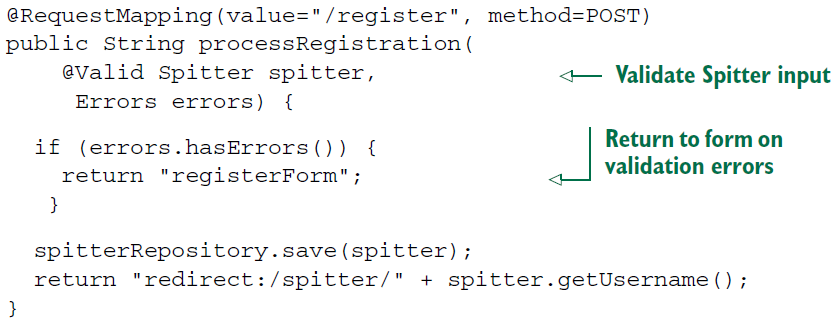




* Java Validation API implementations may provide additional validation annotations. And it’s also possible to define your own constraints. But for our purposes, we’ll focus on a couple of the core constraint validations from the table.
* Thinking over the constraints you need to apply to the fields in Spitter, it seems you’ll probably need the @NotNull and @Size annotations. All you need to do is toss those annotations around on the properties of Spitter. The next listing shows Spitter with its properties annotated for validation.



* All the properties of Spitter are now annotated with @NotNull to ensure that they aren’t left null. Similarly, the @Size annotation is placed on the properties to constrain them between minimum and maximum lengths. What this means in the Spittr application is that the user must completely fill out the registration form with values that fit within the size constraints.
* Now that you have annotated Spitter with validation constraints, you need to change the processRegistration() method to apply validation. The new validationenabled processRegistration() is shown next.



* A lot has changed since the original processRegistration().The Spitter parameter is now annotated with @Valid to indicate to Spring that the command object has validation constraints that should be enforced.
* Just having validation constraints on the Spitter’s properties won’t prevent the form from being submitted. Even if the user fails to fill in a field on the form or gives a value whose length exceeds the maximum length, the processRegistration()method will still be called. This gives you a chance to deal with the validation problems however you see fit in processRegistration().
* If there are any validation errors, they’re available in the Errors object that you’re now asking for as a parameter to processRegistration(). (Note that it’s important that the Errors parameter immediately follow the @Valid-annotated parameter that’s being validated.) The first thing processRegistration() does is call Errors.hasErrors() to check for any errors.
* If there are errors, Errors.hasErrors() returns registerForm, the view name for the registration form. This will take the user’s browser back to the registration form so they can correct any problems and try again. For now, the blank form will be displayed, but in the next chapter, you’ll adapt the form to show the values that were originally submitted and communicate validation problems to the user.
* If there are no errors, the Spitter is saved via the repository, and the controller redirects to the profile page as before.
* Thinking over the constraints you need to apply to the fields in Spitter, it seems you’ll probably need the @NotNull and @Size annotations. All you need to do is toss those annotations around on the properties of Spitter.

