Module 3-9

Data Validation

Collaborator Objects

Spring requires two collaborators to do form validation.

- A custom class of your own creation that maps to your form data.
- The class BindingResult, which saves the validation results.

Custom Class for the Model Attribute

The first component we need is a class, which contains attributes matching those of a form scripted in JSP. Consider the following example:

Name
Email Address
Phone Number (XXX)XXX-XXXX
Age
Sign Me Up!

```
public class MailingListSignUp {
        private String name;
        private String email;
        private String phone;
        private Integer age;

        // +getters and setters

}
```

Here we have a form, on the server side, note that we also have a class on the right with data members matching the form fields.

Custom Class for the Model Attribute

On the class, we can leverage Spring annotations to help us define what type of validation we want to eventually perform on each field.

```
public class MailingListSignUp {
        @NotBlank(message = "Name is required")
        private String name;

        @NotBlank(message = "Email address is required")
        @Email(message = "Please enter a valid email address")
        private String email;

        @Pattern(regexp = "^\\(\\d{3}\\)\\d{3}-\\d{4}$", message = "Invalid phone number")
        private String phone;

        @Min(value = 13, message = "You're too young")
        @Max(value = 150, message = "Get off my lawn")
        private Integer age;

        // Getters and Setters
}
```

The highlighted annotations provide ways to check for empty input, check for valid ranges or that the entry fits a specific format.

Custom Class for the Model Attribute

Here is a list of common validation annotations:

- **@NotBlank(“message”)**: Will check if a field is blank or not.
- **@Email(“message”)**: Verify in an input conforms to an email format.
- **@Min(value=<<x>>, message = “message”)**: A form must have a minimum input value, where <<x>> is that number.
- **@Max(value=<<y>>, message = “message”)**: A form must have a maximum input value, where <<y>> is that number.
- **@Pattern(regexp= “<<z>>” , message=“message”)**: A form’s value must conform to a regular expression, where <<z>> is that expression enclosed in double quotes.

Let’s define the class!

Server Code with Binding Result

- The BindingResult class allows us to manage any validation errors encountered.
- We can verify if any errors occurred by invoking the hasErrors() method, which is part of the BindingResult class.

Server Code with Binding Result

- Consider the following example, if a form sends a POST request to the endpoint /signUp, the following code executes:

```
@RequestMapping(path = "/signUp", method = RequestMethod.POST)
public String processMailingListForm(
@Valid @ModelAttribute MailingListSignUp signUp,
BindingResult result,
RedirectAttributes flash)
{
        flash.addFlashAttribute("signUp", signUp);

        if (result.hasErrors()) {
                flash.addFlashAttribute(BindingResult.MODEL_KEY_PREFIX + "signUp", result);
                return "redirect:/";
        }
…
}
```

If any of the validation rules on the form fail, we will go into this block.

Let’s write the server code!

JSP Setup

We now turn our attention to the view which needs to be configured a certain way in order for interact with the server code written so far:

- We will first need a new “form” tag library, which will allow us to create enhanced Spring forms (in red):

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
```

JSP Setup

An overview of these new tags are as follows:

- <form:form> … </form:form>: Defines the form, similar to <form></form> in HTML.
  -
- <form:input **path= “x”** />: Similar to <input></input>
  - The path attribute is responsible for binding the form element to a data member in the Java model class.
- <form:errors **path= “x” cssClass= “y”** />
  - It also contains a path attribute.
  - A CSS class is used to plug in a CSS class for the purposes of formatting error text.

JSP Setup

The main <form:form> tag should have a model attribute, which ties it to the the model attribute defined on the server side:

JSP Code:

```
<form:form method="POST" action="${mailingListSubmitUrl}" modelAttribute="signUp">
```

Server Code:

```
@RequestMapping(path = "/signUp", method = RequestMethod.POST)
public String processMailingListForm (
@Valid @ModelAttribute MailingListSignUp signUp,
BindingResult result,
RedirectAttributes flash)
{
….
```

JSP Setup Example

```
<body>

        <c:url var="mailingListSubmitUrl" value="/signUp"/>
        <form:form method="POST" action="${mailingListSubmitUrl}" modelAttribute="signUp">

                <div>
                        <label for="name">Name</label>
                        <form:input path="name"/>
                        <form:errors path="name" cssClass="error"/>
                </div>
                <div>
                        <label for="email">Email Address</label>
                        <form:input path="email"/>
                        <form:errors path="email" cssClass="error"/>
                </div>
                <div>
                        <label for="phone">Phone Number (XXX)XXX-XXXX</label>
                        <form:input path="phone"/>
                        <form:errors path="phone" cssClass="error"/>
                </div>
                <div>
                        <label for="age">Age</label>
                        <form:input path="age"/>
                        <form:errors path="age" cssClass="error"/>
                </div>
                <div>
                        <input type="submit" value="Sign Me Up!"/>
                </div>
        </form:form>
</body>
```

Overall, note that the structure of a form is not that dissimilar to a regular HTML form.

Let’s create the JSP’s