Anotación @ModelAttribute e Hibernate Validator

Como hemos estado viendo, Spring nos provee de la anotación @RequestParam para enlazar los parámetros de un formulario a los parámetros de nuestro controlador.

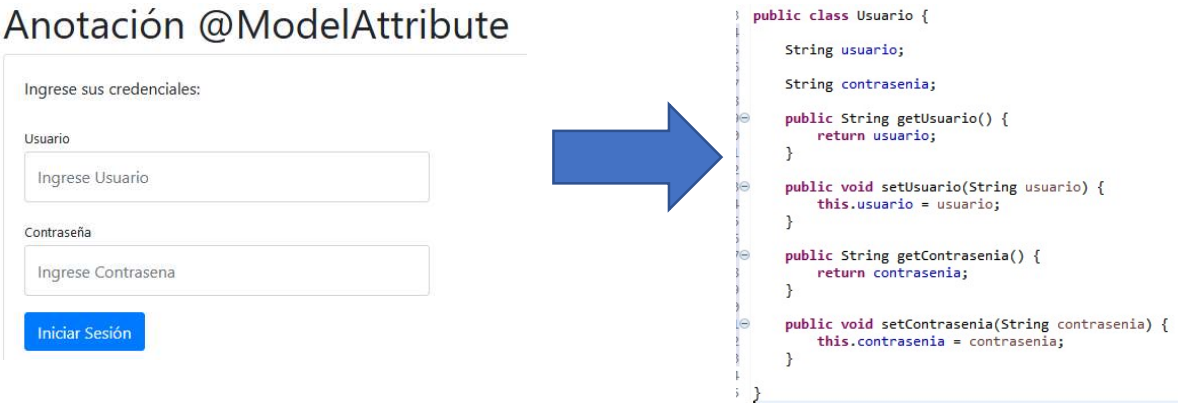
Sin embargo, muchas veces nos encontraremos con situaciones en que un formulario puede tener muchos campos, por lo cual sería poco desordenado tener un método con una cantidad considerable de parámetros.

Para poder lidiar con esta situación, Spring nos permite también enlazar una clase Java a todo un formulario HTML, de manera que en el método controlador ya no recibamos varios parámetros, sino uno solo, el cual es una instancia de esa clase.

Por ejemplo, utilizaremos el formulario de inicio de sesión y lo enlazaremos a una clase en Java que representará dicho formulario.

Para esto, crearemos en nuestro proyecto un nuevo paquete de nombre **com.uca.capas.modelo.domain**, y dentro crearemos una clase de nombre **Usuario.**

Como el formulario tiene dos campos usuario y contraseña, debemos crear estas dos variables en nuestra clase Usuario, que representarán dichos campos.



Ahora, ya tenemos la clase que representa dicho formulario, solo nos falta asociarlo al formulario HTML, para esto agregaremos las siguientes propiedades al formulario HTML:

```
<form method="post" th:action="@{/parametros1}" th:object="${usuario}">
    <div class="form-group"><label class="small mb-1" for="usuario">Usuario</label>
    <input class="form-control py-4" th:field="*{usuario}" id="usuario" type="text" placeholder="Ingrese Usuario
    <div class="form-group"><label class="small mb-1" for="password">Contrase&ntilde;a</label>
    <input class="form-control py-4" th:field="*{contrasenia}" id="password" type="password" placeholder="Ingres
    <div class="form-group"><input type="submit" class="btn btn-primary" value="Iniciar Sesi&oacute;n"></a></div
</form>
```

1. **Propiedad th:object**. Esta propiedad del **form** se utiliza para indicarle a Thymeleaf que utilice el objeto enviado desde el controlador para asociarlo al formulario. Se especifica el nombre del objeto que es enviado desde el controlador en el ModelAndView, en nuestro caso será una instancia de la clase Usuario.

```java
@RequestMapping("/index11")
public ModelAndView index11() {
    ModelAndView mav = new ModelAndView();
    mav.addObject("usuario", new Usuario());
    mav.setViewName("clases/clase11/index");
    return mav;
}
```

Como vemos, el controlador encargado de redirigir a la página que contiene el formulario lleva, como parte del modelo, una nueva instancia de la clase Usuario, el cual utilizará Spring para manejar el formulario. El nombre del objeto debe ser el mismo que se especificó en la propiedad **th:object.**

2. **Propiedad th:field**. Ahora bien, debemos especificarle a Spring a que propiedades del objeto queremos que asocie cada input, para ello utilizamos la propiedad **th:field**, que recibe como parámetro el nombre de la propiedad del objeto al que queremos asociar el input, de la forma **"*{nombre_propiedad}"**

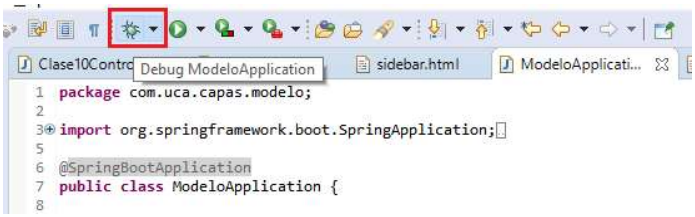
Con esto ya hemos asociado el formulario HTML con el objeto de tipo Usuario.

Ahora crearemos el método encargado de procesar la petición del formulario, con la diferencia que hoy no recibirá dos parámetros usuario y contraseña, sino un solo parámetro de tipo Usuario.

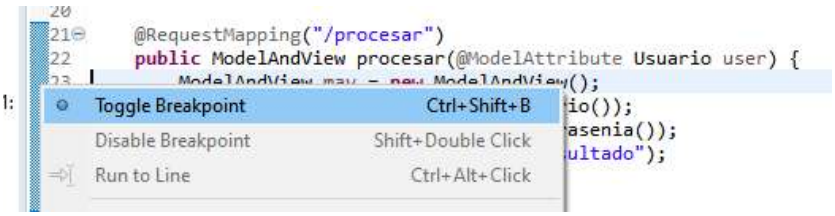
```java
@RequestMapping("/procesar")
public ModelAndView procesar(@ModelAttribute Usuario user) {
    ModelAndView mav = new ModelAndView();
    mav.addObject("user", user.getUsuario());
    mav.addObject("pass", user.getContrasenia());
    mav.setViewName("clases/clase11/resultado");
    return mav;
}
```

Efectivamente, el método recibe un parámetro de tipo Usuario, al cual le hemos puesto la anotación @ModelAttribute, con lo cual Spring enlazará los parámetros del formulario a cada variable de la clase.

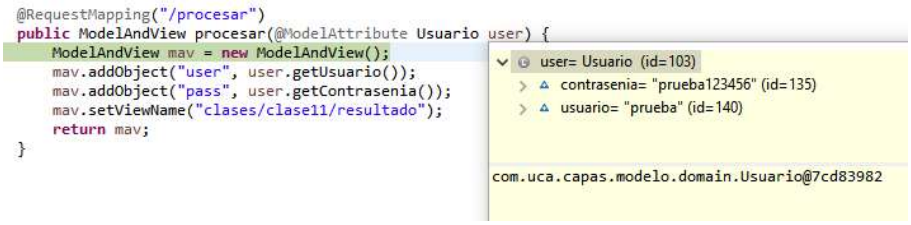
Ahora, para ver el objeto que recibe el controlador, iniciaremos el servidor en modo "debug", dando clic al siguiente botón ubicado en la barra de herramientas.



Ahora pondremos un Breakpoint en la clase Clase11Controller en el siguiente lugar dando clic derecho sobre la barra azul en la línea 23.



Al ejecutar el formulario, y dar clic al botón Iniciar sesión, la ejecución del programa parará en esa línea y podremos revisar los objetos:



Como vemos, el objeto **user** contiene en las propiedades contrasenia y usuario, los valores que fueron enviados por el formulario.

Validación de formularios con Hibernate Validator

Muchas veces debemos de realizar validaciones de datos a los campos de los formularios, puesto que son ingresados por los usuarios, los cuales están sujetos a cometer errores.

Dichas validaciones pueden ser:

- Longitud máxima o mínima de un campo
- Que cumpla cierto formato
- Que no vaya vacío
- Etc.

Estas validaciones las podemos realizar perfectamente de forma programática, sin embargo, existe una librería que nos permite realizar estas validaciones de manera sencilla, esto es, utilizando ciertas anotaciones sobre las propiedades de nuestra clase que representa al formulario (ModelAttribute).

Esta librería se llama hibernate-validator, y la utilizaremos para validar los datos que son ingresados por el usuario en el formulario.

Lo primero será agregar dicha librería a nuestro proyecto, a través de la siguiente dependencia en nuestro archivo pom.xml

```
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
</dependency>
```

Ahora, ya tendremos acceso a todas las anotaciones que servirán para realizar validaciones a nivel de dominio en las propiedades de la clase.

Por ejemplo, a la clase Usuario, agregaremos las siguientes validaciones a los campos usuario y contrasenia:

```
public class Usuario {

    @Size(min=1, max=25)
    String usuario;

    @NotEmpty
    String contrasenia;
```

En este caso hemos agregado dos validaciones por medio de dos anotaciones.

Las anotaciones se han puesto a las propiedades de la clase Usuario.

La primera es @Size, el cual tiene como propiedades min y max. Esto es la longitud mínima y la longitud máxima que puede tener.

La segunda es @NotEmpty, con la cual le decimos que este parámetro no debe venir vacío, no confundir con @NotNull

En la JSR 303 (documento de especificaciones) se mencionan las validaciones soportadas.

Ya hemos configurado las validaciones a nivel dominio. Ahora necesitamos decirle a Spring que queremos que ejecute dichas validaciones al momento de recibir el objeto en el controlador, y a la vez, incluiremos un objeto como parámetro en nuestro controlador, el cual utilizaremos para ver el resultado de la validación y ver los errores que hay en el objeto.

Para esto agregaremos la anotación @Valid al parámetro del objeto Usuario, y a la vez agregaremos un segundo parámetro de tipo **BindingResult**, el cual contendrá el resultado de la validación (y una colección de objetos de tipo Errors, donde estarán los errores encontrados).

```
@RequestMapping("/procesar2")
public ModelAndView procesar2(@Valid @ModelAttribute Usuario user, BindingResult result) {
    ModelAndView mav = new ModelAndView();
    mav.addObject("user", user.getUsuario());
    mav.addObject("pass", user.getContrasenia());
    mav.setViewName("clases/clase11/resultado");
    return mav;
}
```

Como vemos, ahora al objeto Usuario lo afectarán dos anotaciones, @Valid para validaciones de dominio, y @ModelAttribute para indicarle a Spring que utilice ese objeto para manejar el formulario HTML de la petición.

Ahora bien, Una de las cosas que se hacen cuando se encuentran errores es, precisamente, mostrárselas al usuario, En la página del formulario a la par de cada campo o en algún lugar que se estima conveniente.

Thymeleaf Nos permite pintar estos errores de una manera fácil utilizando un tag especial que se alimenta del objeto BindingResult para dicho fin.

Para esto vamos a verificar primero en el controlador si hay errores, y si lo hay, entonces redirigiremos nuevamente hacia la pantalla del formulario dónde se pintarán los errores encontrados a la par de cada campo.

Para saber si hay errores, utilizaremos el método **hasErrors** del objeto BindingResult, el cual devuelve **true** o **false** según hayan errores o no.

Si lo hay, redirigiremos a la misma pantalla del formulario, caso contrario, lo redirigiremos a la pantalla de siempre (resultado).

```java
@RequestMapping("/procesar2")
public ModelAndView procesar2(@Valid @ModelAttribute Usuario user, BindingResult result) {
    ModelAndView mav = new ModelAndView();

    if(result.hasErrors()) { //Hay errores, redirigimos a la pantalla del formulario
        mav.setViewName("clases/clase11/index");
    }
    else { //Si no hay, flujo normal
        mav.addObject("user", user.getUsuario());
        mav.addObject("pass", user.getContrasenia());
        mav.setViewName("clases/clase11/resultado");
    }
    return mav;
}
```

Ahora, debemos pintar los errores en el formulario, para esto utilizaremos la propiedad **th:errors**, el cual recibe la propiedad del objeto del cual queremos pintar los errores, quedando de esta manera:

```html
<form method="post" th:action="@{/procesar2}" th:object="${usuario}">
    <div class="form-group"><label class="small mb-1" for="usuario">Usuario</label>
        <input class="form-control py-4" th:field="*{usuario}" id="usuario" type="text" placeholder="Ingrese Usuari
        <span th:errors="*{usuario}" style="color: #FF0000"></span>
    </div>
    <div class="form-group"><label class="small mb-1" for="password">Contrase&ntilde;a</label>
        <input class="form-control py-4" th:field="*{contrasenia}" id="password" type="password" placeholder="Ingre
        <span th:errors="*{contrasenia}" style="color: #FF0000"></span>
    </div>
    <div class="form-group"><input type="submit" class="btn btn-primary" value="Iniciar Sesi&oacute;n"></a></div>
```

Le hemos agregado un estilo color rojo para que se note de una manera mas fácil.

Ahora, si ejecutamos el formulario y dejamos, por ejemplo, en blanco el campo usuario, al dar Iniciar Sesión nos mostrará el siguiente mensaje:

Recordando el flujo, al dar clic al botón “Iniciar Sesión”, la petición viaja hacia el controlador, y como el objeto ModelAttribute tiene la anotación @Valid, Spring ejecutará las validaciones de dominio y tendremos dicho resultado en el objeto BindingResult.

Luego, verificamos si hay errores con el método **hasErrors()** del objeto BindingResult, si lo hay entonces lo redirigiremos nuevamente a la pantalla del formulario, en el cual, debido a los tags **th:errors** que hemos puesto, Spring pintará automáticamente los errores que se encuentran en el objeto BindingResult para cada propiedad.

Los mensajes de error pueden ser personalizados, acorde a las necesidades del usuario, en el ejemplo anterior aparece **size must be between 1 and 25**, lo cambiaremos y le pondremos **El campo usuario debe tener una longitud entre 1 y 25 caracteres.**

Para esto, utilizaremos la propiedad **message** de las anotaciones de validación, que recibe como valor el mensaje a mostrar:

```
public class Usuario {

    @Size(min=1, max=25, message = "El campo usuario debe tener una longitud entre 1 y 25 caracteres.")
    String usuario;

    @NotEmpty(message = "El campo contrasenia no puede estar vacio.")
    String contrasenia;
```

Ahora, nos mostrará los siguientes mensajes: