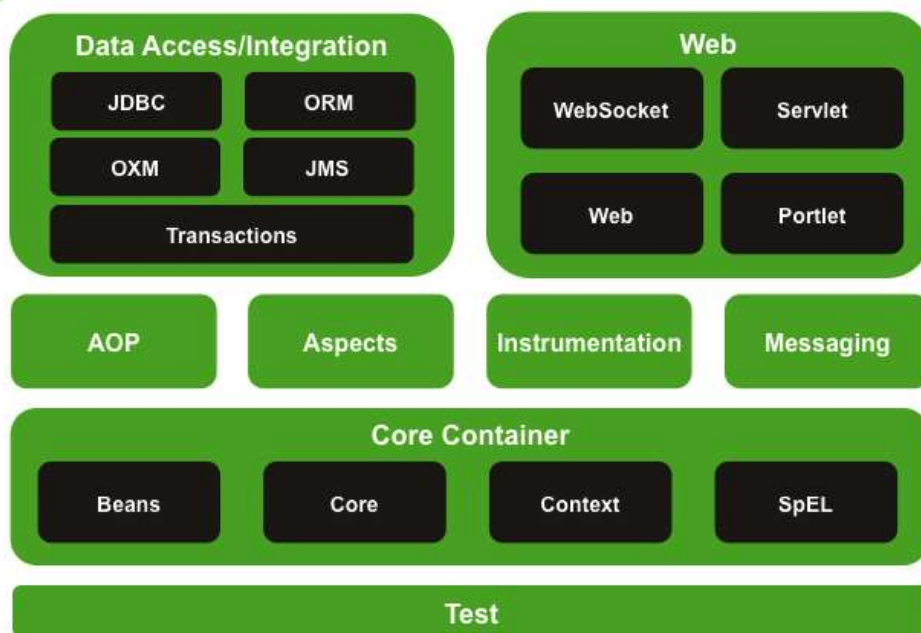


DESARROLLO WEB CON SPRING BOOT



Spring Framework Runtime



UNIDAD 06 SPRING MVC VALIDATION

Eric Gustavo Coronel Castillo
gcoronel@uni.edu.pe
INSTRUCTOR



CONTENIDO

CONTEXTO.....	3
FUNDAMENTOS.....	4
ANOTACIONES DE VALIDACIÓN	5
PROGRAMACIÓN EL FORMULARIO	8
CARGA DEL FORMULARIO	8
FORMULARIO CON JSP	8
FORMULARIO CON THYMELEAF	9
PAGINA DE RESULTADO	11
PROCESANDO EL FORMULARIO	12
PERSONALIZANDO LOS MENSAJES	13
ERRORES DE CONVERSIÓN	14
USANDO EXPRESIONES REGULARES.....	15
EJERCICIOS.....	16
EJERCICIO 1.....	16
EJERCICIO 2.....	16



CONTEXTO

VALIDACIÓN DE FORMULARIO

Nombre: El tamaño del nombre debe estar entre 2 y 30 caracteres.

Edad: El valor de la edad debe ser mínimo 18.

Correo: Debe ingresar una dirección de correo valida.

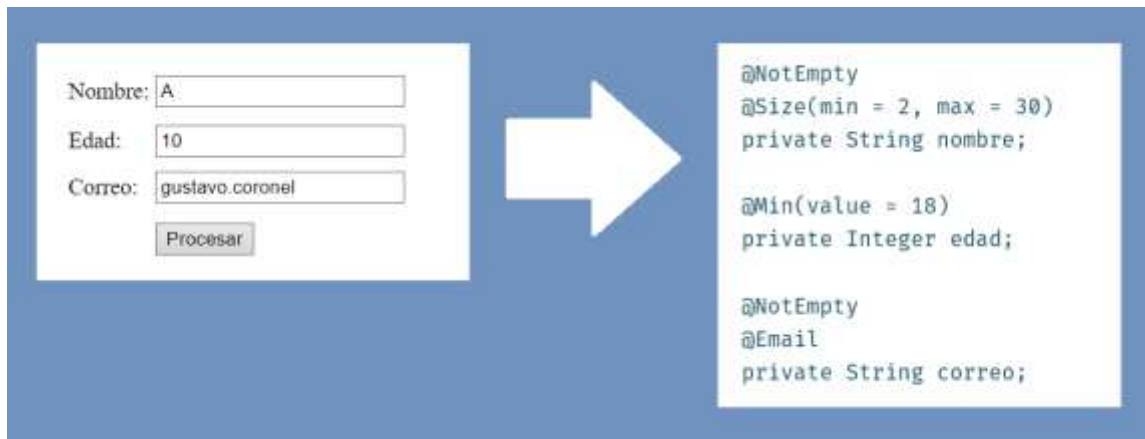
Uno de los problemas que debes resolver como programador, es la validación de los datos que el usuario ingresa, y mostrar un mensaje de error adecuado.

Para resolver este problema, Spring ofrece un API de validación que se encarga de todo el proceso de validación y generación de los mensajes de error en función de declaraciones que se realizan a nivel del bean.

Spring Validation es una de las características más utilizadas de Spring y está basado en JSR 303 (Bean Validation) para validar la información que ingresas en un formulario.

En el desarrollo de esta guía veras un ejemplo ilustrativo que lo podrás implementar con paginas JSP o utilizando THYMELEAF.

FUNDAMENTOS



El API de validación de Spring esta basado en anotaciones que se deben utilizar en los campos de los bean, estos deben tener relación con los campos del formulario.

El API maneja mensajes por defecto, pero también puedes personalizarlos, de tal manera que el usuario pueda recibir la mejor descripción del error que se ha producido.

Para que puedas utilizar esta API, lo primera que debes hacer es agregar la dependencia respectiva, tal como se ilustra a continuación:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

ANOTACIONES DE VALIDACIÓN

```
public class EmpleadoModel {  
  
    @NotEmpty  
    @Size(min = 2, max = 30)  
    private String nombre;  
  
    @NotNull  
    @Min(value = 18)  
    private Integer edad;  
  
    @NotEmpty  
    @Email  
    private String correo;  
  
    public EmpleadoModel() {  
    }  
  
    ■ ■ ■  
  
}
```

A continuación, tienes las anotaciones que puedes utilizar para configurar las validaciones de tus campos:

ANOTACIÓN	DESCRIPCIÓN
@NotNull	Valida que el valor de la propiedad anotada no sea nulo.
@AssertTrue	Valida que el valor de la propiedad anotada sea verdadero.
@Size	Valida que el valor de la propiedad anotada tenga un tamaño entre los atributos min y max ; se puede aplicar a las propiedades de cadena, colección, mapa y matriz.
@Min	Valida que la propiedad anotada tenga un valor no menor que el atributo value .
@Max	Valida que la propiedad anotada tenga un valor no mayor que el atributo value .
@Email	Valida que la propiedad anotada es una dirección de correo electrónico válida.

El API maneja mensajes por defecto, pero el atributo **message** es común a todas las anotaciones. En este atributo puedes indicar un mensaje personalizado que remplazara al mensaje por defecto y es el que se presentará cuando el valor de la propiedad respectiva falle en la validación.

A continuación, tienes algunas anotaciones adicionales que pueden utilizar:

ANOTACIÓN	DESCRIPCIÓN
@NotEmpty	Valida que la propiedad no sea nula ni esté vacía; se puede aplicar a valores de cadena, colección, mapa o matriz.
@NotBlank	Se puede aplicar solo a valores de texto y valida que la propiedad no sea nula ni espacios en blanco.
@Positive @PositiveOrZero	Se aplican a valores numéricos y validan que sean estrictamente positivos, o positivos incluyendo 0.
@Negative @NegativeOrZero	Se aplican a valores numéricos y validan que sean estrictamente negativos, o negativos incluyendo 0.
@Past @PastOrPresent	Validan que un valor de fecha está en el pasado o en el pasado, incluido el presente; se puede aplicar a los tipos de fecha, incluidos los agregados en Java 8.
@Future @FutureOrPresent	Validan que un valor de fecha está en el futuro, o en el futuro, incluido el presente.



A continuación, tienes un ejemplo ilustrativo:

```
public class EmpleadoModel {  
  
    @NotEmpty  
    @Size(min = 2, max = 30)  
    private String nombre;  
  
    @NotNull  
    @Min(value = 18)  
    private Integer edad;  
  
    @NotEmpty  
    @Email  
    private String correo;  
  
    public EmpleadoModel() {  
    }  
  
    // Se debe generar los métodos getters y setters  
  
}
```



PROGRAMACIÓN EL FORMULARIO

Carga del formulario

Debes programar el controlador para cargar el formulario, lo puedes programar con JSP o usando thymeleaf.

```
@GetMapping({"/","/home"})
public String home(@ModelAttribute("empleado") EmpleadoModel empleado) {

    return "home";

}
```

Formulario con JSP

En este caso, la página debe tener el nombre **home.jsp**.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>VALIDACIÓN DE FORMULARIO</title>
</head>
<body>
    <h1>VALIDACIÓN DE FORMULARIO</h1>
    <form:form modelAttribute="empleado" action="/verificar" method="post">
        <table>
            <tr>
                <td>Nombre:</td>
                <td><form:input id="nombre" path="nombre" /></td>
                <td style="color:red;"><form:errors path="nombre" /></td>
            </tr>
            <tr>
                <td>Edad:</td>
                <td><form:input id="edad" path="edad" /></td>
                <td style="color:red;"><form:errors path="edad" /></td>
            </tr>
        </table>
    </form>
</body>
```




```
<tr>
  <td>Correo:</td>
  <td><form:input id="correo" path="correo" /></td>
  <td style="color:red;"><form:errors path="correo" /></td>
</tr>
<tr>
  <td></td>
  <td><button type="submit">Procesar</button></td>
</tr>
</table>
</form:form>
</body>
</html>
```

Formulario con Thymeleaf

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>VALIDACIÓN DE FORMULARIO</title>
</head>
<body>
  <h1>VALIDACIÓN DE FORMULARIO</h1>
  <form th:action="@{/verificar}" th:object="${empleado}" method="post">
    <table>
      <tr>
        <td>Nombre:</td>
        <td><input type="text" th:field="*{nombre}" /></td>
        <td style="color: red;" th:if="${#fields.hasErrors('nombre')}}"
          th:errors="*{nombre}"></td>
      </tr>
      <tr>
        <td>Edad:</td>
        <td><input type="text" th:field="*{edad}" /></td>
        <td style="color: red;" th:if="${#fields.hasErrors('edad')}}"
          th:errors="*{edad}"></td>
      </tr>
      <tr>
        <td>Correo:</td>
        <td><input type="text" th:field="*{correo}" /></td>
        <td style="color: red;" th:if="${#fields.hasErrors('correo')}}"
          th:errors="*{correo}"></td>
      </tr>
    </table>
  </form>
```



```
<tr>
  <td></td>
  <td><button type="submit">Procesar</button></td>
</tr>
</table>
</form>
</body>
</html>
```



PAGINA DE RESULTADO

Se trata de una página simple y puede ser JSP o HTML según corresponda.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>RESULTADO</title>
</head>
<body>
  <div>
    <strong>¡Felicitaciones!</strong> Cumple con los requisitos para
    registrarte en este sitio.
  </div>
</body>
</html>
```



PROCESANDO EL FORMULARIO

En este caso debes programar el controlador para procesar el formulario.

```
@PostMapping("/verificar")
public String verificar(
    @Valid @ModelAttribute("empleado") EmpleadoModel empleado,
    BindingResult bindingResult) {

    if (bindingResult.hasErrors()) {
        return "home";
    }

    return "resultado";
}
```

En este caso, la anotación **@Valid** está indicando que se debe validar en objeto empleado, pero es necesario agregar un parámetro de tipo **BindingResult**, es en este parámetro donde se tendrán los mensajes de errores.

A continuación, tienes un ejemplo de como se verían los mensajes de error.

VALIDACIÓN DE FORMULARIO

Nombre:	<input type="text" value="G"/>	el tamaño debe estar entre 2 y 30
Edad:	<input type="text" value="10"/>	debe ser mayor que o igual a 18
Correo:	<input type="text" value="gustavo.coronel"/>	debe ser una dirección de correo electrónico con formato correcto
<input type="button" value="Procesar"/>		

PERSONALIZANDO LOS MENSAJES

Si quieres personalizar los mensajes de error, debes utilizar la propiedad **message** que tienen todas las anotaciones, tal como se ilustra a continuación.

```
public class EmpleadoModel {

    @NotEmpty(message = "El nombre no puede ser un valor nulo.")
    @Size(min = 2, max = 30,
        message = "El tamaño del nombre debe estar entre 2 y 30 caracteres.")
    private String nombre;

    @NotNull(message = "La edad no puede ser un valor nulo.")
    @Min(value = 18, message = "El valor de la edad debe ser mínimo 18.")
    private Integer edad;

    @NotEmpty(message = "El correo no puede ser un valor nulo.")
    @Email(message = "Debe ingresar una dirección de correo valida.")
    private String correo;

    public EmpleadoModel() {
    }

    // Implementar los métodos getters y setters.

}
```

Otra alternativa es crear los mensajes en un archivo de propiedades de nombre **messages.properties**, a continuación tienes un ejemplo de los mensajes:

```
# Mensajes para bean EmpleadoModel
NotEmpty.empleado.nombre=El nombre del empleado es obligatorio.
Size.empleado.nombre=El tamaño del nombre del empleado debe estar entre 2 y 30 caracteres.
NotNull.empleado.edad=La edad del empleado es obligatorio.
Min.empleado.edad=La edad del empleado de ser mínimo 18 años.
NotEmpty.empleado.correo=El correo del empleado es obligatorio.
Email.empleado.correo=El correo del empleado debe tener un formato valido.
```



ERRORES DE CONVERSIÓN

VALIDACIÓN DE FORMULARIO

Nombre:

Edad: Failed to convert property value of type java.lang.String to required type java.lang.Integer for property edad; nested exception is java.lang.NumberFormatException: For input string: "abc"

Correo:

En algunos casos puedes tener errores de conversión, como por ejemplo si en el campo Edad ingresas una cadena, para estos casos debes realizar una declaración en el archivo **messages.properties**.

Como se trata de un valor entero, puedes utilizar una de las siguientes opciones:

```
typeMismatch.java.lang.Integer=El campo {0} es un valor entero.
```

```
typeMismatch=El campo {0} tiene un formato incorrecto.
```

Si quieres hacer referencia al nombre del campo, por ejemplo, para el campo **edad** del objeto **empleado**, tienes estas opciones:

```
typeMismatch.empleado.edad=La edad del empleado es un valor entero.
```

```
typeMismatch.edad=La edad es un valor entero.
```



USANDO EXPRESIONES REGULARES

VALIDACIÓN DE FORMULARIO

Nombre:

Edad:

Correo: Correo mal formado

En algunos casos no encontraras una anotación de validación que cumpla exactamente los requerimientos que la aplicación necesita.

La validación Spring MVC te permite validar la entrada del usuario en una secuencia particular (es decir, expresión regular). La anotación **@Pattern** se utiliza para lograr la validación de expresiones regulares. Aquí, puedes proporcionar la expresión regular requerida para el atributo **regex** y pasarla con la anotación.

A continuación, tienes un ejemplo de como validarías el campo **email**.

```
@Pattern( regex = "\\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\\.[A-Z]{2,4}\\b",
          message = "Correo mal formado")
private String correo;
```

EJERCICIOS

En los siguientes ejercicios, los campos de entrada deben estar validados según la naturaleza del dato.

Ejercicio 1

Pacherres Delivery es una empresa de reparto de paquetes en la ciudad, el costo por kilómetro está en base al peso del paquete según el siguiente cuadro:

PESO EN KG	COSTO x KM
[1,5>	8 Soles
[5,10>	7 Soles
[10,∞>	6 Soles

Desarrollar un programa que permita calcular el importe que costaría a un cliente enviar un paquete.

Ejercicio 2

Desarrollar un programa para calcular el promedio de un estudiante, se sabe que son 4 notas de prácticas y se promedian las tres mejores, un examen parcial y un examen final.

También se debe determinar la condición del estudiante, si el promedio es mayor o igual que 14 está aprobado, caso contrario esta desaprobado.