UNIVERSIDAD DEL QUINDÍO
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

Información general	
Actualizado por:	Carlos Andrés Flórez Villarraga
Duración estimada en minutos:	60
Docente:	Christian Andrés Candela y Einer Zapata
Guía no.	17
Nombre de la guía:	Validaciones en JSF 2.3

Información de la Guía

OBJETIVO

Aprender a usar validaciones en JSF.

CONCEPTOS BÁSICOS

Manejo de Eclipse, Java, archivos de propiedades, xml y Glassfish.

CONTEXTUALIZACIÓN TEÓRICA

JSF (Java Server Faces) es un *framework* de presentación, usado para la construcción de interfaces web (páginas) que permitan la interacción con el usuario. JSF es un *framework* basado en componentes y hace uso del patrón de diseño MVC (Modelo Vista Controlador).

Uno de los principales objetivos de JSF es separar la presentación (diseño de la página) de la lógica asociada a dicha presentación. Se puede pensar en JSF como en la librería *swing*, con la diferencia que *swing* lo usamos para crear las ventanas de nuestras aplicaciones de escritorio, mientras que JSF nos permite la creación de páginas web dentro de nuestras aplicaciones empresariales. Teniendo en cuenta esto, es importante decir que al igual que en *Spring* (*framework* para el desarrollo de aplicaciones) en JSF se pueden crear componentes propios con el fin de reutilizarlos posteriormente, o se puede hacer uso de componentes de terceros.

JSF ha sido incluido dentro de la especificación JEE como su capa de presentación. Siendo JSF un estándar, posee múltiples implementaciones. De igual forma gracias a su inclusión como estándar, JSF está en continua evaluación y actualización.

Las validaciones al interior de JSF pueden realizarse de forma manual. Esto se logra usando programación defensiva en los métodos de procesamiento de los datos. Sin embargo, no necesariamente son la mejor alternativa. JSF incorpora herramientas para la validación de datos. La primera de ellas es un conjunto de etiquetas y atributos que permite establecer validaciones tales como rangos, tipos de datos y longitud de los mismos. En la versión JSF 2.3, además de las etiquetas, se permite establecer validaciones al hacer uso de anotaciones sobre los datos de los `ManageBean`, lo que permite independizar aún más la presentación de la lógica.

Si se desea que un determinado campo deba ser ingresado obligatoriamente por el usuario, basta con marcar el campo de entrada con el atributo **`required`**. JSF verifica que los campos marcados con **`required`** sean ingresados por el usuario antes de procesar alguna solicitud. Si lo quiere, se puede especificar el mensaje que será mostrado por el sistema al detectar que un campo no ha sido ingresado. Esto lo puede hacer adicionando

el atributo `requiredMessage` al campo de entrada. De igual forma puede personalizar otros mensaje de validación como por ejemplo `converterMessage`, para los mensajes de conversión de tipo, `validatorMessage`, para otro tipo de validación, como por ejemplo la longitud de una entrada.

Para la conversión de tipos se cuenta con los siguiente componentes estándar: `BigDecimalConverter`, `BigIntegerConverter`, `BooleanConverter`, `ByteConverter`, `CharacterConverter`, `DateTimeConverter`, `DoubleConverter`, `EnumConverter`, `FloatConverter`, `IntegerConverter`, `LongConverter`, `NumberConverter`, `ShortConverter`. Los convertidores pueden ser usados tanto en componentes de entrada como en componentes de salida de datos.

Ver: https://docs.oracle.com/javaee/7/tutorial/jsf-page-core001.htm

Para la validación de tipos se usan los siguientes componentes estándar: `validateBean`, `validateDoubleRange`, `validateLength`, `validateLongRange`, `validateRegEx` y `validateRequired`.

Ver: https://docs.oracle.com/javaee/7/tutorial/jsf-page-core003.htm

También es posible diseñar nuestros propios validadores. Una de las alternativas para esto son los métodos de validación personalizados. Consiste en el desarrollo de un método, el cual es invocado para verificar la validez de uno determinado campo. Para indicar que un campo debe hacer uso de un método de validación específico se usa el atributo **`validator`**. Los métodos de validación personalizados deben tener una firma como la siguiente:

```
public void funcionDeValidacion(FacesContext contexto, UIComponent componenteAValidar,
Object valor) throws ValidatorException {
        …
}
```

En caso de que el campo no sea válido el método debe generar una excepción `ValidatorException`.

Adicionalmente se pueden crear clases especializadas para la validación de valores. Dichas clases deben implementar la interfaz `javax.faces.validator.Validator` y ser anotadas con la anotación `@FacesValidator(value = "nombre")`. Donde nombre es el nombre del validador. Dichos validadores pueden ser usados desde la página usando la etiqueta `f:validator`.

Paralelo a las alternativas de validación presentadas, existe otro tipo de validación que se realiza al interior de los `ManagedBean`, la cual fue incorporada desde JSF 2.0. Este tipo de validación favorece la independencia del código y la presentación. A continuación se presenta una tabla con un conjunto de validaciones que pueden usarse en los atributos de los `ManageBean` en forma de anotaciones.

Anotación	Significado	Ejemplo
`@AssertFalse`	Verifica que el valor del atributo sea falso.	`@AssertFalse` `boolean isFalso;`
`@AssertTrue`	Verifica que el valor del atributo sea verdadero.	`@AssertTrue` `boolean isVerdadero;`
`@DecimalMax`	Permite establecer el número	`@DecimalMax("30.00")` `BigDecimal numeroDecimal;`

	máximo al cual puede llegar un atributo.	
`@DecimalMin`	Permite establecer el número mínimo al cual puede llegar un atributo.	`@DecimalMin("5.00")` `BigDecimal numeroDecimal;`
`@Digits`	Permite establecer el máximo número de dígitos, tanto en su parte entera como en su parte fraccionaria.	`@Digits(integer=6, fraction=2)` `BigDecimal numeroDecimal;`
`@Future`	Permite validar que la fecha asignada a un atributo sea una fecha futura.	`@Future` `Date fecha;`
`@Past`	Permite validar que la fecha asignada a un atributo sea una fecha pasada.	`@Past` `Date fecha;`
`@Max`	Permite establecer el valor máximo a ser asignado a un atributo.	`@Max(10)` `int numero;`
`@Min`	Permite establecer el valor mínimo a ser asignado a un atributo.	`@Min(5)` `int numero;`
`@NotNull`	Permite establecer que un determinado atributo no debe ser nulo.	`@NotNull` `String login;`
`@Null`	Permite establecer que un determinado atributo debe ser nulo.	`@Null` `String cadena;`
`@Pattern`	Permite establecer una expresión regular a ser cumplida por los valores asignados al atributo.	`@Pattern(regex="\\(\\d{3}\\)\\d{3}-\\d{4}")` `String numeroTelefono;`
`@Size`	Permite validar el tamaño mínimo y/o máximo de un atributo. Es aplicable a cadenas, arreglos, colecciones y mapas.	`@Size(min=2, max=240)` `String mensaje;`
`@Valid`	Indica que deben verificarse las validaciones contenidas por los atributos contenidos por el atributo al que se asigna la anotación.	`@Valid` `Usuario usuario;`

Ver: https://docs.oracle.com/javaee/7/tutorial/bean-validation.htm

PRECAUCIONES Y RECOMENDACIONES

No debe olvidar hacer uso de h:form en los campos de entrada de datos para sus formularios.

ARTEFACTOS

Se requiere tener instalado el JDK y un IDE para el desarrollo de aplicaciones (Eclipse JEE en su última versión), un servidor de aplicaciones que cumpla con las especificaciones de JEE, para esta práctica Glassfish.

EVALUACIÓN O RESULTADO

Se espera que el alumno pueda emplear satisfactoriamente diferentes herramientas de validación de datos proporcionadas por JSF.

Procedimiento

1. Para el desarrollo de esta guía necesitará una base de datos en MySQL, un proyecto de tipo Maven – pom, el cual contenga un módulo Maven con soporte para el uso de JPA, uno con soporte para EJB, uno para web y otro proyecto Maven configurado para la realización de pruebas. Además de una conexión a la base de datos para ser usada en la generación de las tablas.

2. Si no lo ha hecho, cree un `ManagedBean` con el nombre de `UsuarioBean`. Debe tener como atributos los que considere necesarios para registrar un usuario. No olvide generar los métodos `get` y `set`.

3. Cree una página para registrar un usuario (puede ser `registrarUsuario.xhtml`), o úsela si ya la tiene creada.

4. Al interior de la página cree un formulario que permita el ingreso de los datos necesarios para crear un usuario. Además, dentro del formulario agregue la etiqueta `<h:panelGrid>`. Debería verse similar a lo siguiente:

```
<h:form>
       <h:panelGrid columns="2" >
       </h:panelGrid>
</h:form>
```

5. Por cada uno de los campos de su `UsuarioBean` cree un campo de entrada así:

```
<h:outputLabel  for="cedula"  value="Cedula:" />
<h:inputText id="cedula" value="#{usuarioBean.cedula}" required="true"></h:inputText>
```

6. En el `ManageBean` cree un método que permita registrar un usuario en el sistema.

7. Adicione un botón que permita invocar un método creado en el `ManagedBean` para el registro.

```
<h:commandButton value="Enviar" action="#{usuarioBean.registrar}" />
```

   El botón debe ser adicionado inmediatamente después del cierre de la etiqueta `</h:panelGrid>`

8. En la página adicione la etiqueta `<h:messages /><br />` antes de la etiqueta del botón.

9. Despliegue y verifique los resultados.

10. Dentro de la etiqueta de entrada de texto de la cédula adicione una validación que verifique que la cédula no pueda tener valor de 0.

```
<h:inputText id="cedula" value="#{usuarioBean.cedula}" required="true">
    <f:validateLongRange minimum="1" />
</h:inputText>
```

Fíjese en el atributo **`required`**, indica que el campo es obligatorio. Con otro validador se puede verificar la longitud de los texto por ejemplo:

```
<h:inputText id="nombre" value="#{usuarioBean.nombre}">
    <f:validateLength minimum="4" maximum="50" />
</h:inputText>
```

También es posible hacer uso de anotaciones a nivel de los Bean para establecer validaciones, así como se mencionó anteriormente. Por ejemplo, si se desea que un determinado campo no sea `null`, se puede hacer uso de la anotación `@NotNull`. O en lugar de usar la etiqueta `validateLength` para controlar la longitud del texto puede usar la anotación `@Size` así:

```
@Size(min=4, max=50, message="Mensaje de validacion")
```

11. Adicione las validaciones para el teléfono, email y cedula, además las que considere necesarias.

12. Despliegue y verifique los resultados.

13. Investigue y de ser necesario use otras anotaciones para la validación de los atributos. Puede encontrar información en el libro *The Definitive Guide to JSF in Java EE 8*.

14. Si se desea es posible personalizar los mensajes de error de varias formas. Una de las opciones de personalización son los atributos (`requiredMessage` y `validatorMessage`) que pueden ser adicionados a las etiquetas `inputText` de igual forma que se adiciona el atributo **`required`**.

15. Despliegue y verifique los resultados.

16. También es posible mostrar los mensajes de error de forma separada, para ello se usa la etiqueta `message`, indicando a dicha etiqueta por medio del atributo `for` a quién pertenecen los mensajes que mostrará así: `<h:message for="cedula"/>`

17. Despliegue y verifique los resultados.

Para la próxima clase

Pruebe cambiar la apariencia de su página haciendo uso del atributo **`style`** en sus etiquetas. Si no lo conoce investigue cómo puede usarlo.