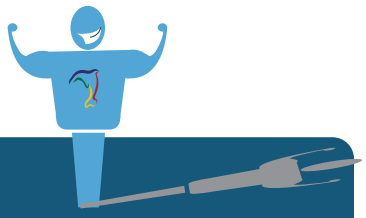
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

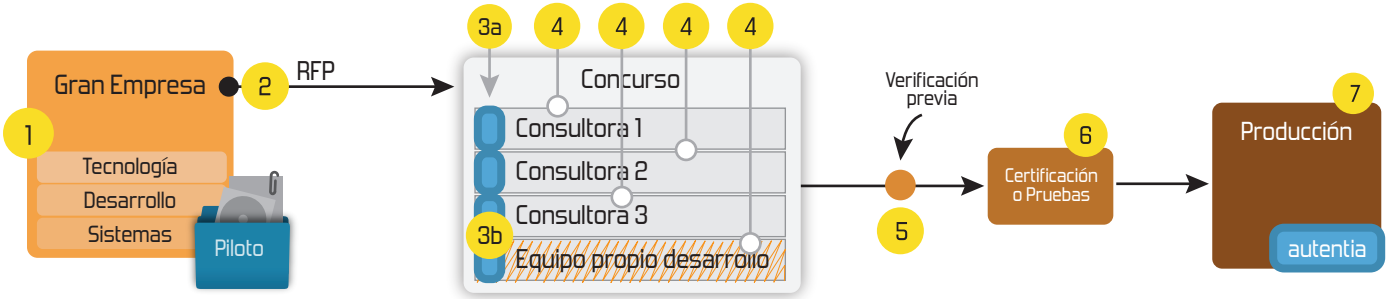
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

- Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery
- Gestor portales (Liferay)
- Gestor de contenidos (Alfresco)
- Aplicaciones híbridas
- Control de autenticación y acceso (Spring Security)
- UDDI
- Web Services
- Rest Services
- Social SSO
- SSO (Cas)
- JPA-Hibernate, MyBatis
- Motor de búsqueda empresarial (Solr)
- ETL (Talend)
- Dirección de Proyectos Informáticos.
- Metodologías ágiles
- Patrones de diseño
- TDD
- Tareas programadas (Quartz)
- Gestor documental (Alfresco)
- Inversión de control (Spring)
- BPM (jBPM o Bonita)
- Generación de informes (JasperReport)
- ESB (Open ESB)

Compartimos nuestro conociemiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Tutorial desarrollado por: Beatríz Bonilla

Puedes encontrarme en Autentia
Somos expertos en Java/J2EE
Contacta en info@autentia.com

Descargar este documento en formato PDF convValidJSF.pdf

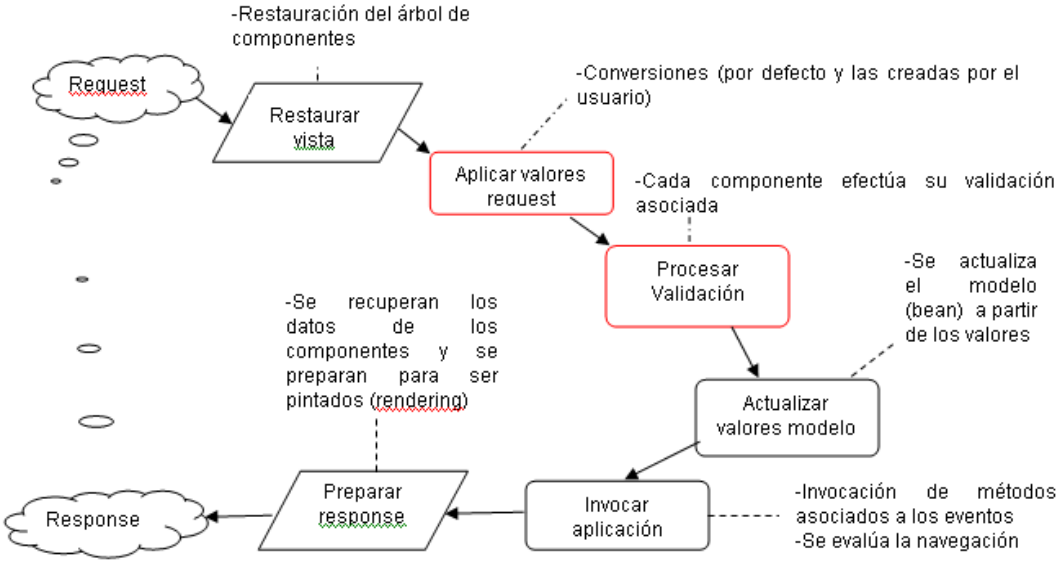
Firma en nuestro libro de Visitas

VALIDACIONES Y CONVERSIONES EN JSF

Introducción

En este tutorial veremos como JSF proporciona mecanismos para el formateo, conversión y vali los componentes; veremos como estos mecanismos pueden ser extendidos de forma sen adaptarlos a nuestras necesidades. La primera parte del tutorial estará dedicada al formateo y c y la segunda a la Validación.

Para entender en qué momento de la vida de una página JSF se producen las conversiones y va de los componentes del formulario, quizá nos sea de ayuda el siguiente diagrama:



JSF proporciona dos mecanismos separados que nos ayuda a validar los valores introducidc usuarios a la hora de submitir los formularios:

- El primer mecanismo es el de **conversión** y está definido por el interfaz *javax.faces.convert.Conv* múltiples implementaciones. **Las conversiones aseguran que el tipo de un dato introducido en un formul el correcto**, es decir, que el dato tipo cadena del formulario corresponde con el tipo JAVA es que está especificado en la propiedad correspondiente del bean. Los *Conversores* (implementa la interfaz *Converter*) son los componentes que se encargan de hacer estas transfor (cadena>Tipo JAVA y viceversa). JSF invoca a los *Conversores* antes de efectuar las validacic

lo tanto antes de aplicar los valores introducidos a las propiedades del bean. En el caso c dato tipo cadena no se corresponda con el tipo JAVA apropiado, el *Conversor* correspondient un *ConversionException* y el componente se marcará como invalidado.

- El segundo mecanismo es el de **validación** y está definido por la interfaz *javax.faces.validator.Vali* múltiples implementaciones. El proceso de validación se asegura que el dato introduc correspondiente componente es correcto según la lógica de la aplicación. El proceso de ocurre antes de que el FrameWork asigne los valores introducidos en el formulario a las pr del bean y justo después de que se hayan aplicado las conversiones, en el caso de que **validaciones aseguran que un dato introducido en un formulario JSF tenga un valor correcto.**

Los mensajes que lanzan tanto los Conversores como lo Validadores en el caso de excepción s mostrar en el formulario correspondiente a través de la etiqueta *messages*.

A continuación vamos a ver 4 casos prácticos:

- Un ejemplo de conversión utilizando un conversor estándar de JSF.
- Un ejemplo de conversión realizada por un conversor que creemos nosotros.
- Un ejemplo de validación utilizando un validador estándar de JSF.
- Un ejemplo de validación utilizando un validador creado por nosotros.

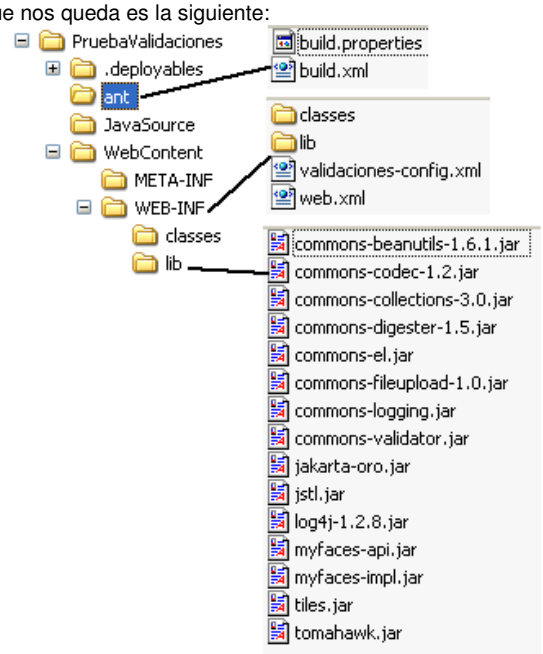
Herramientas utilizadas

- Sistema operativo: Windows XP Professional.
- Servidor Web: Apache Tomcat 5.5.9
- Entorno de desarrollo: Eclipse 3.1.1 con ExadelStudio-3[1].0.5

Preparamos el entorno

Seguiremos los siguientes pasos:

1. **Creamos un nuevo proyecto JSF en Eclipse, al que llamaremos *PruebaValidaciones*.** Indicaremos durante la creación que el entorn MyFaces 1.1.0. La estructura de directorios que nos queda es la siguiente:



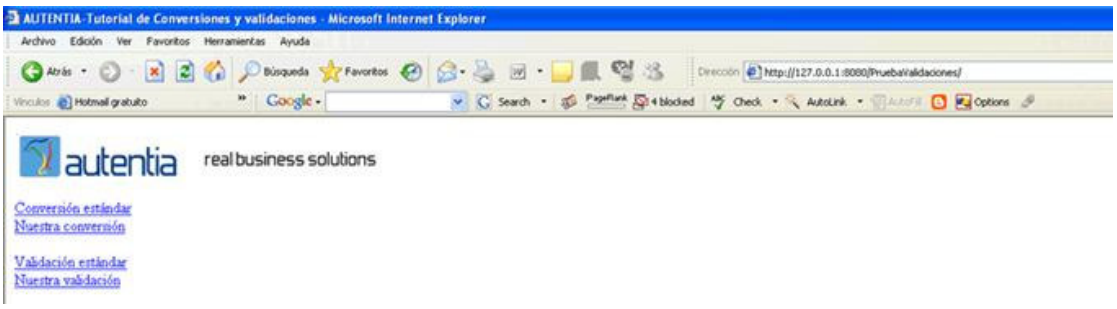
2. **Creamos la página *index.jsp* en el directorio *WebContent*:** redireccionará a la página *home.jsf*

```
<%@ taglib uri="http://myfaces.apache.org/extensions" prefix="t" %>
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>AUTENTIA- TUTORIAL CONVERSIONES-VALIDACIONES</head>
        <body>
                <jsp:forward page="home.jsf" />
        </body>
</html>
```

3. **Creamos la página *home.jsp* en el directorio *WebContent*:** esta página tendrá enlaces a los diferentes ejemplos que vamos a hacer:

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>
<html>
<head>
<title>AUTENTIA-Tutorial de Conversiones y validaciones</title>

</head>
<body>
<f:view>
  <h:panelGrid id="cabecera" columns="2" styleClass="cabecera"  >
       <t:graphicImage id="logo" url="img/demo_logo.gif" alt="AUTENTIA - Soluciones Reales para su
empresa"/>
          <f:verbatim>
               
          </f:verbatim>
  </h:panelGrid>
  <br/>
  <h:outputLink value="conversion_estandar.jsf"><f:verbatim>Conversión
estándar</f:verbatim></h:outputLink> <br/>
  <h:outputLink value="conversion_nuestra.jsf" ><f:verbatim>Nuestra
conversión</f:verbatim></h:outputLink> <br/>
   <br />
   <h:outputLink value="validacion_estandar.jsf" ><f:verbatim>Validación
estándar</f:verbatim></h:outputLink> <br/>
   <h:outputLink value="validacion_nuestra.jsf" ><f:verbatim>Nuestra
validación</f:verbatim></h:outputLink> <br/>
   <br />
</f:view>
</body>
</html>
```

Y si probamos lo poco que hemos hecho:



Los ejemplos que vamos a hacer a partir de este momento partirán de esta página.

Primer ejemplo: usando un conversor estándar

Antes de nada debemos tener en cuenta que los conversores estándar que proporciona JSF se dividen en conversores a tipos numéricos (*Numb*
y conversores a tipos Fecha (*DateTimeConverter*)

Existen dos formas de indicar que el valor de un determinado componente de nuestra página va a ser "convertido":

- Añadiendo el atributo *converter*="#{identificador_conversor}" al componente. Sólo los componentes del tipo *UIInput* (*inputText*, *in*
  inputHidden) y outputText aceptan esta propiedad. Un ejemplo sería el siguiente:

  ```
  <h:inputText converter="#{Double}"
  value="#{modifyInvoicePage.invoice.amount}"/>
  ```

  JSF deja a nuestra disposición los siguientes conversores estándar:

TIPO DE CONVERSOR	IDENTIFICADOR DEL CONVERSOR
BigDecimalConverter	BigDecimal
BigIntegerConverter	BigInteger
NumberConverter	Number
IntegerConverter	Integer

ShortConverter	Short
ByteConverter	Byte
CharacterConverter	Character
FloatConverter	Float
DoubleConverter	Double
BooleanConverter	Boolean
DateTimeConverter	DateTime

- Incluyendo *<f:convertNumber [lista_atributos]/>* o *<f:convertDateTime [lista_atributos]/>* dentro de la invocación al comp por ejemplo:

```
<h:inputText value="#{modifyInvoicePage.invoice.amount}">
<f:convertNumber type="currency"/>
</h:inputText>
```

O por ejemplo

```
<h:inputText id="invoiceDate"
value="#{modifyInvoicePage.invoice.invoiceDate}">
<f:convertDateTime pattern="M/d/yyyy"/>
</h:inputText>
```

Los atributos disponibles para tipos *number* son:

NOMBRE DEL ATRIBUTO	TIPO
currencyCode	String
currencySymbol	String
groupingUsed	boolean
integerOnly	boolean
locale	java.util.Locale
maxFractionDigits	int
maxIntegerDigits	Int
minFractionDigits	Int
minIntegerDigits	int
pattern	String
type	String

Y los disponibles para tipos *DateTime*:

NOMBRE DEL ATRIBUTO	TIPO
dateStyle	String
parseLocale	String o Locale
pattern	String
timeStyle	String
timeZone	String o TimeZone
type	String

1. Vamos a crear por fin el bean

La clase se llama *com.autentia.tutorialValidacion.GestionUsuariosBean.java,* cuelga del directorio *JavaSource* y de momento tiene la sig

```
package com.autentia.tutorialValidacion;
import java.math.BigDecimal;
import java.util.Date;
/**
 * Bean para probar conversiones y validaciones en JSF
 * @author AUTENTIA
 */
public class GestionUsuariosBean
{
        /** Nombre del usuario */
        private String nombre;
        /** Edad del usuario */
        private int edad;
        /** Fecha de nacimiento */
        private Date fechaNacimiento;
        /** Sueldo */
        private BigDecimal sueldo;
        /**
         * Constructor
```

```
	 */
	public GestionUsuariosBean(){}

	public int getEdad() {
		return edad;
	}
	public void setEdad(int edad) {
		this.edad = edad;
	}
	public Date getFechaNacimiento() {
		return fechaNacimiento;
	}
	public void setFechaNacimiento(Date fechaNacimiento) {
		this.fechaNacimiento = fechaNacimiento;
	}
	public String getNombre() {
		return nombre;
	}
	public void setNombre(String nombre) {
		this.nombre = nombre;
	}
	public BigDecimal getSueldo() {
		return sueldo;
	}
	public void setSueldo(BigDecimal sueldo) {
		this.sueldo = sueldo;
	}

}
```

Y lo registramos en el fichero descriptor de JSF (*WEB-INF/validaciones-config.xml*)

```
<?xml version="1.0"?>
<!DOCTYPE faces-config PUBLIC
 "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
 "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">

<faces-config>
	<managed-bean>
	 <description>Bean para probar conversiones/validaciones</description>
	 <managed-bean-name>gestionUsuariosBean</managed-bean-name>
	 <managed-bean-class>
		com.autentia.tutorialValidacion.GestionUsuariosBean
	 </managed-bean-class>
	 <managed-bean-scope>session</managed-bean-scope>
	</managed-bean>
</faces-config>
```

2. Creamos la página

La llamamos *conversion_estandar.jsp* y cuelga de *WebContent*:

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://myfaces.apache.org/extensions" prefix="t" %>

<html>
   <head>
	<title>AUTENTIA - TUTORIAL VALIDACIONES/CONVERSIONES</title>
	<link rel="stylesheet" type="text/css" href="css/estilos.css">
  </head>
  <body>
   <f:view>
   <h:form id="idUsuarios" name="gestionUsuariosBean">
   <h:messages id="messageList" styleClass="error" showSummary="true" showDetail="true" />
      <h:panelGrid columns="2" styleClass="gestionUsuariosFormTable"
                    headerClass="gestionUsuariosFormHeader"
                    footerClass="gestionUsuariosFormFooter"
                    columnClasses="gestionUsuariosFormLabels, gestionUsuariosFormInputs" width="600">
```

```
        <!-- Nombre -->
        <h:outputLabel for="login" value="Nombre"/>
        <h:panelGroup>
            <h:inputText id="nombre" styleClass="CajasTexto" size="30" maxlength="100"
                        value="#{gestionUsuariosBean.nombre}" />
        </h:panelGroup>
        <!-- Edad -->
        <h:outputLabel for="edad" value="Edad"/>
        <h:panelGroup>
         <h:inputText id="laEdad" converter="#{Integer}" styleClass="CajasTexto" size="30" maxlength="3"
                    value="#{gestionUsuariosBean.edad}" />
        </h:panelGroup>
        <!-- Fecha de nacimiento -->
        <h:outputLabel for="fechaNacimiento" value="Fecha de nacimiento"/>
        <h:panelGroup>
         <h:inputText id="fecha" styleClass="CajasTexto" size="30" maxlength="12"
                    value="#{gestionUsuariosBean.fechaNacimiento}">
           <f:convertDateTime pattern="dd/MM/yyyy"/>
        </h:inputText>
        </h:panelGroup>
        <!-- Sueldo -->
        <h:outputLabel for="sueldo" value="Sueldo"/>
        <h:panelGroup>
          <h:inputText converter="#{BigDecimal}" styleClass="CajasTexto" size="30" maxlength="15"
                    id="idSueldo"  value="#{gestionUsuariosBean.sueldo}" />
        </h:panelGroup>
         <h:panelGroup>
           <h:commandButton action="resultado_conversion" value="Validar" />
           <f:verbatim> </f:verbatim>
          </h:panelGroup>
                    </h:panelGrid>
    </h:form>
    </f:view>
  </body>
  </html>
```

Como vemos, es muy sencillo utilizar los conversores estándar y cualquier explicación sobra. Para que el botón de *Validar* funcione, debemos añ correspondiente regla de navegación en el fichero *WEB-INF/validaciones-config.xml*:

```
...
..
<navigation-rule>
        <from-view-id>/conversion_estandar.jsp</from-view-id>
        <navigation-case>
          <from-outcome>resultado_conversion</from-outcome>
          <to-view-id>/resultado_conversion.jsp</to-view-id>
          <redirect/>
        </navigation-case>
</navigation-rule>
...
...
```

Y para que el ejemplo nos quede chulo, creamos la página que dice que dice que todo ha ido bien y muestra los valores introducidos por el usuari

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://myfaces.apache.org/extensions" prefix="t" %>

<html>
        <head>
                <title>AUTENTIA - TUTORIAL VALIDACIONES/CONVERSIONES</title>
                <link rel="stylesheet" type="text/css" href="css/estilos.css">
        </head>
        <body>
        <f:view>
   <h:form id="idUsuarios" name="gestionUsuariosBean">
```

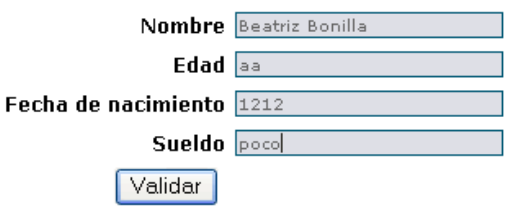
```
   <h:messages id="messageList" styleClass="error" showSummary="true" showDetail="true" />
      <h:panelGrid columns="2" styleClass="gestionUsuariosFormTable"
                  headerClass="gestionUsuariosFormHeader"
                  footerClass="gestionUsuariosFormFooter"
                  columnClasses="gestionUsuariosFormLabels, gestionUsuariosFormInputs"
width="600">
      <!-- Nombre -->
      <h:outputLabel for="login" value="Nombre"/>:
      <h:panelGroup>
          <h:outputText value="#{gestionUsuariosBean.nombre}"/>
      </h:panelGroup>
      <!-- Edad -->
      <h:outputLabel for="edad" value="Edad"/>
      <h:panelGroup>
       <h:outputText value="#{gestionUsuariosBean.edad}"/>
      </h:panelGroup>
      <!-- Fecha de nacimiento -->
      <h:outputLabel for="fechaNacimiento" value="Fecha de nacimiento"/>
      <h:panelGroup>
       <h:outputText value="#{gestionUsuariosBean.fechaNacimiento}"/>
      </h:panelGroup>
      <!-- Sueldo -->
      <h:outputLabel for="sueldo" value="Sueldo"/>
      <h:panelGroup>
       <h:outputText value="#{gestionUsuariosBean.sueldo}"/>
      </h:panelGroup>
       <h:panelGroup>
         <h:commandButton action="home" value="Volver" />
         <f:verbatim> </f:verbatim>
        </h:panelGroup>
       </h:panelGrid>
    </h:form>
   </f:view>
</body>
</html>
```

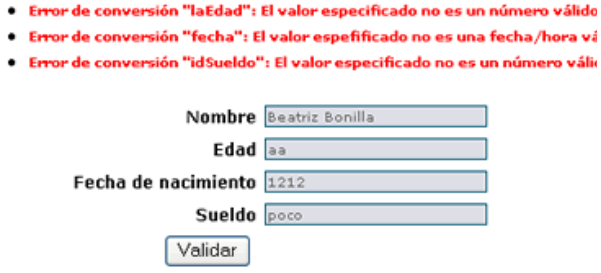
Por ultimo, para completar el ejemplo, añadimos la navegabilidad para el botón de *volver*, que retornará a la página *home.jsp*:

```
...
..
<navigation-rule>
        <from-view-id>/resultado_conversion.jsp</from-view-id>
        <navigation-case>
         <from-outcome>home</from-outcome>
         <to-view-id>/home.jsp</to-view-id>
         <redirect/>
        </navigation-case>
</navigation-rule>
...
...
```

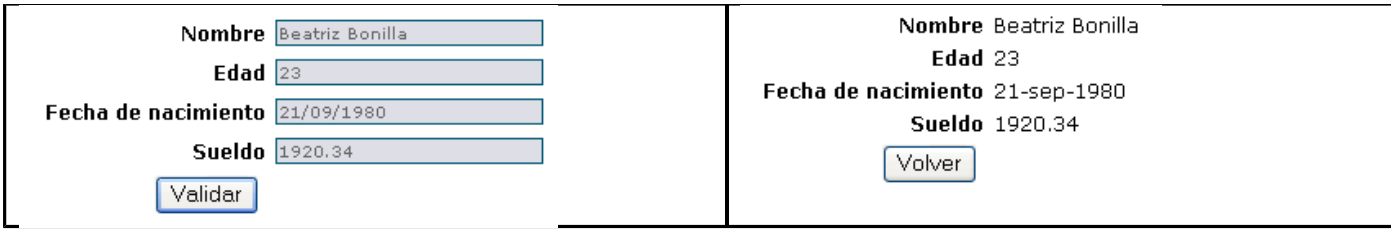
3. Probamos

Vamos a introducir todos los valores incorrectos y vemos cual es el resultado:





Y ahora con valores correctos (bueno, menos la edad y el sueldo, pero eso no hay forma que lo conozcan los conversores de JSF ;-)):



Segundo ejemplo: creando nuestro propio conversor

Para crear nuestro propio conversor, debemos seguir los siguientes pasos:

- Crear una clase que implemente la interfaz *javax.faces.convert.Converter*.
- Implementar el método *getAsObject*, que convierte el valor del componente que tengamos en la página (tipo cadena) a un ob JAVA que requiera la aplicación).
- Implementar el método *getAsString*, que convierte el objeto (tipo JAVA de nuestra aplicación) a tipo cadena (requerido por la
- Registrar el nuevo conversor en el contexto de JSF.
- Utilizarlo en la página insertando dentro del componente cuyo valor queramos convertir el tag *<f:converter>.*

1. Qué va hacer nuestro conversor

Va a desglosar un número de teléfono de tipo:

91-888 33 66

En dos partes:

Prefijo = 91

Número de teléfono= 888 33 66

El conversor lanzará excepciones si:

- No se introduce el guión para separar el prefijo del número
- Si el prefijo no consta de 2 números
- Si el número no consta de 7 números. Los espacios entre los números se ignorarán

2. Creamos el conversor

Como hemos indicado antes, debe implementar la interfaz *Converter* y los métodos *getAsObject y getAsString.* Vamos a llamarlo *Telefono* Para este ejemplo he utilizado la librería *commons-lang* de Yakarta que nos la podemos descargar desde la página http://jakarta.apache.org/site/downloads/downloads_commons-lang.cgi. La he incorporado al paquete *lib* y he actualizado el *clas* aplicación.

El método *getAsObject* va a devolver la instancia de una nueva clase *Telefono.java* que tiene la siguiente pinta:

```
package com.autentia.tutorialValidacion;
/**
 * Clase que representa un teléfono
 * @author AUTENTIA
 */
public class Telefono
{
	/** El prefijo */
	private String prefijo;
	/** El número de teléfono */
	private String numero;
	/**
	 * Constructor
	 */
	public Telefono(){}
	public String getNumero() {
		return numero;
	}
	public void setNumero(String numero) {
		this.numero = numero;
	}
	public String getPrefijo() {
		return prefijo;
	}
	public void setPrefijo(String prefijo) {
		this.prefijo = prefijo;
	}
	/**
	 * Devuelve el número de teléfono como un String
	 */
	public String toString()
	{
		StringBuffer elNumero = new StringBuffer();
```

```
                elNumero.append(prefijo);
                elNumero.append("-");
                elNumero.append(numero);
                return elNumero.toString();
        }
}
```

Y por fin la clase *TelefonoConverter*:

```
package com.autentia.tutorialValidacion;
import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.convert.Converter;
import javax.faces.convert.ConverterException;
import org.apache.commons.lang.StringUtils;

/**
 * Conversor de números de teléfono
 * @author AUTENTIA
 */

public class TelefonoConverter implements Converter
{
        /**
         * Devuelve la representación Telefono de JAVA a partir del valor de un componente JSF
         */
        public Object getAsObject(FacesContext context, UIComponent component, String value)
                                        throws ConverterException
        {
                Telefono telefono = null;
                if(!StringUtils.isEmpty(value))
                {
                        telefono = new Telefono();
                        String valores[] = StringUtils.split(value,"-");
                        if(valores.length!=2)
                                throw new ConverterException(new FacesMessage("El número de teléfono debe tene
el siguiente formato: 'pp-nnn nn nn', donde pp es el prefijo y nnn nn nn es el número"));
                        if(valores[0].length()!=2)
                                throw new ConverterException(new FacesMessage("El prefijo del número de teléfono
debe tener de tipo NN"));
                        String elNumero = StringUtils.remove(valores[1],' ');
                        if(elNumero.length()!=7)
                                throw new ConverterException(new FacesMessage("El número de teléfono debe ser
de tipo NNN NN NN"));
                        telefono.setPrefijo(valores[0]);
                        telefono.setNumero(valores[1]);
                }
                return telefono;
        }
        /**
         * Devuelve la representacion cadena de un teléfono
         */
        public String getAsString(FacesContext context, UIComponent component, Object value)
                                        throws ConverterException
        {
                return value.toString();
        }

}
```

3. Adaptamos el bean GestionUsuariosBean para probar nuestro conversor

Añadimos un nuevo atributo de tipo *Telefono* al bean:

```
...
...
/** Teléfono */
```

```
private Telefono telefono;
....
....
public Telefono getTelefono() {
        return telefono;
}
public void setTelefono(Telefono telefono) {
        this.telefono = telefono;
}
....
.....
```

4. Registramos el nuevo conversor en el contexto de JSF

Para ello editamos el fichero *WEB-INF/validaciones-config.xml* y ponemos lo siguiente:

```
        ............
         <converter>
                <converter-id>autentia.telefonoConverter</converter-id>

<converter-class>com.autentia.tutorialValidacion.TelefonoConverter</converter-class>
         </converter>
        ...............
```

5. Creamos la página *conversión_nuestra.jsp* que cuelga de WebContent:

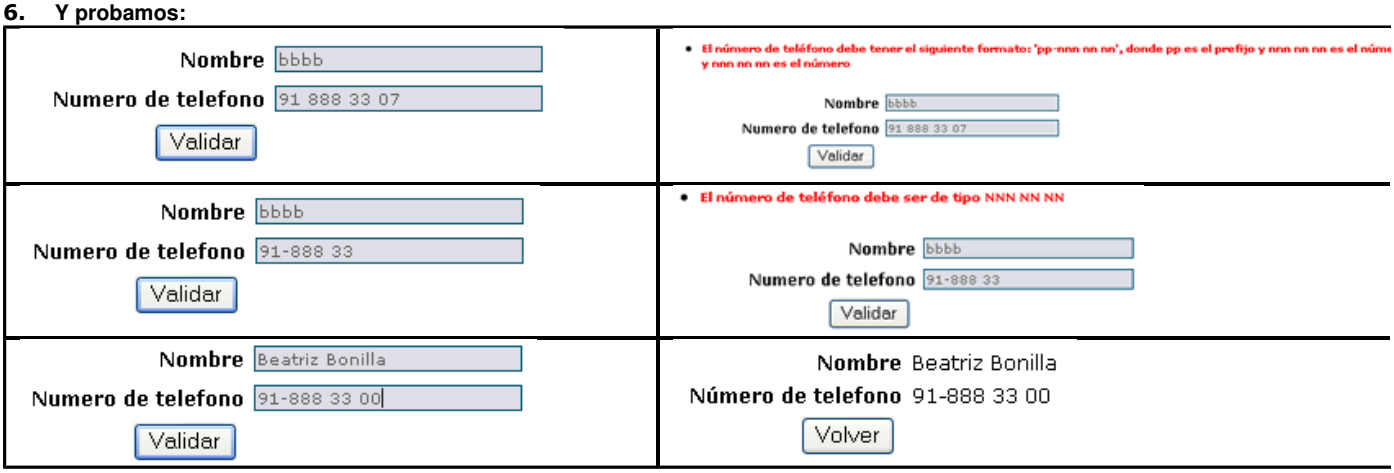
```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://myfaces.apache.org/extensions" prefix="t" %>

<html>
   <head>
                <title>AUTENTIA - TUTORIAL VALIDACIONES/CONVERSIONES</title>
                <link rel="stylesheet" type="text/css" href="css/estilos.css">
         </head>
         <body>
         <f:view>
   <h:form id="idUsuarios" name="gestionUsuariosBean">
   <h:messages id="messageList" styleClass="error" showSummary="true" showDetail="true" />
      <h:panelGrid columns="2" styleClass="gestionUsuariosFormTable"
                 headerClass="gestionUsuariosFormHeader"
                 footerClass="gestionUsuariosFormFooter"
                 columnClasses="gestionUsuariosFormLabels, gestionUsuariosFormInputs"
width="600">
      <!-- Nombre -->
      <h:outputLabel for="login" value="Nombre"/>
      <h:panelGroup>
          <h:inputText id="nombre" styleClass="CajasTexto" size="30" maxlength="100"
                     value="#{gestionUsuariosBean.nombre}" />
      </h:panelGroup>
      <!-- Numero telefono -->
      <h:outputLabel for="telefono" value="Numero de telefono"/>
      <h:panelGroup>
       <h:inputText id="telefono" styleClass="CajasTexto" size="30" maxlength="12"
                  value="#{gestionUsuariosBean.telefono}">
         <f:converter converterId="autentia.telefonoConverter" />
      </h:inputText>
      </h:panelGroup>
       <h:panelGroup>
         <h:commandButton action="resultado_conversion_nuestra" value="Validar" />
         <f:verbatim> </f:verbatim>
        </h:panelGroup>
                 </h:panelGrid>
         </h:form>
         </f:view>
</body>
```

```
</html>
```

Y también añadimos la navegabilidad necesaria para que, una vez convertido el valor, la aplicación vaya a la página de resultados (*resultado_conversion_nuestra*) y para que desde esta página de resultados podamos volver a la página home (*home*)

```
.........
<navigation-rule>
        <from-view-id>/conversion_nuestra.jsp</from-view-id>
        <navigation-case>
         <from-outcome>resultado_conversion_nuestra</from-outcome>
         <to-view-id>/resultado_conversion_nuestra.jsp</to-view-id>
         <redirect/>
        </navigation-case>
</navigation-rule>
<navigation-rule>
        <from-view-id>/resultado_conversion_nuestra.jsp</from-view-id>
        <navigation-case>
         <from-outcome>home</from-outcome>
         <to-view-id>/home.jsp</to-view-id>
         <redirect/>
        </navigation-case>
</navigation-rule>
...........
```

6. Y probamos:



Tercer ejemplo: utilizando validaciones estándar de JSF

En la primera parte de este tutorial hemos visto como los *Conversores* pueden cubrir gran parte de la funcionalidad nec validar que los valores introducidos por los usuarios en un formulario JSF sean correctos.

En adición, los *Validadores* pueden realizar validaciones más genéricas durantes la fase de Validación. Estos component implementan la interfaz *javax.faces.validator.Validator,* que contiene un único método *validate()*. JSF provee un conjun *Validadores* estándar que se describen en la tabla siguiente:

CLASE DE VALIDADOR	TAG JSF	ATRIBUTOS	DESCRIPCIÓN
DoubleRangeValidator	validateDoubleRange	minimum,maximum	Valida que el valor del co sea mayor que un mínim que un máximo. Este val tipo float
LengthValidator	validateLength	minimum,maximum	Valida que la longitud del componente esté entre u un máximo
LongRangeValidator	validateLongRange	minimum,maximum	Valida que el valor del co sea mayor que un mínim que un máximo. Este val tipo entero

Además, myFaces proporciona los siguientes *Validadores* como adición a los anteriores:

CLASE DE VALIDADOR	TAG JSF	ATRIBUTOS	DESCRIPCIÓN
EmailValidator	validateEmail		Valida que el valor del co tenga un formato tipo em

CreditCardValidator	validateCreditCard	none(true|false), amex(true|false), visa(true|false), mastercard(true|false), discover(true|false)	Valida que el valor del co corresponde con el núme siguientes tipos de tarjeta crédito: American Expres Discover, Mastercard, Vis
RegExprValidator	validateRegExpr	pattern	Valida que el valor del co se corresponde con el pat especificado como atribut
EqualValidator	validateEqual	for	Valida que el valor del co sea exactamente igual al componente cuyo *id* se co con el atributo for.

1. Preparamos el bean *GestionUsuariosBean* para el ejemplo

Vamos a efectuar las siguientes validaciones:

- Que la edad del usuario esté comprendida entre 5 y 100 años
- Que el nombre no tenga más de 40 caracteres
- Que el nuevo atributo *password* debe ser igual que el campo repetición de *password*

Por lo tanto, al bean sólo le falta un nuevo atributo *password*:

```
…
…
/** Password del usuario */
private String password;
public String getPassword() {
        return password;
}
public void setPassword(String password) {
        this.password = password;
}
…
…
```

2. Creamos la página

La llamaremos *validación_estandar.jsp* y colgará de *WebContent*:

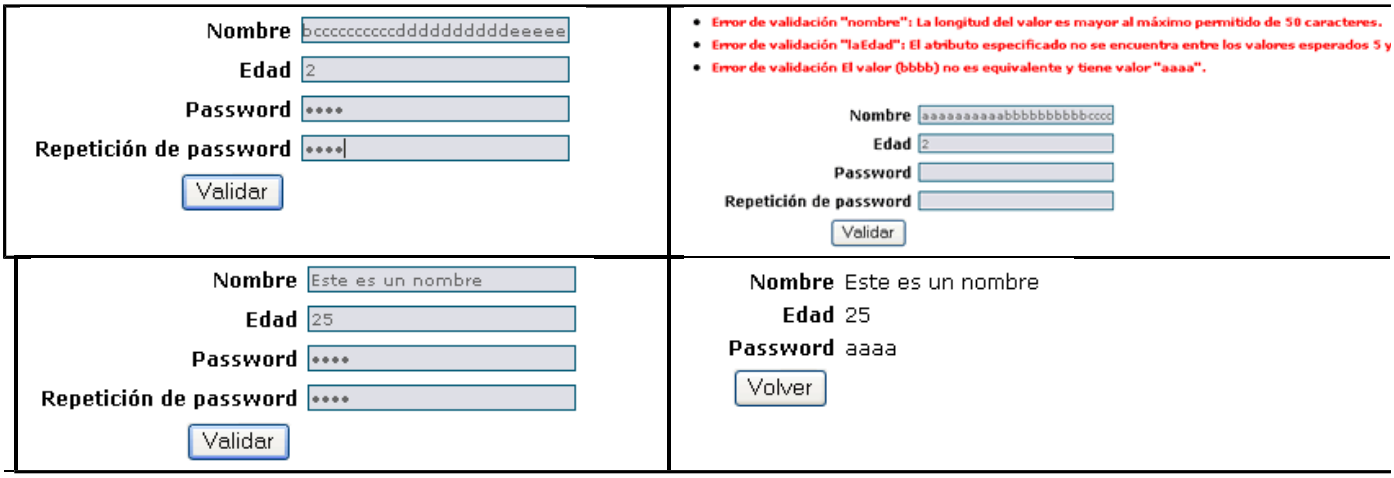
```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://myfaces.apache.org/extensions" prefix="t" %>

<html>
  <head>
            <title>AUTENTIA - TUTORIAL VALIDACIONES/CONVERSIONES</title>
            <link rel="stylesheet" type="text/css" href="css/estilos.css">
      </head>
      <body>
      <f:view>
  <h:form id="idUsuarios" name="gestionUsuariosBean">
  <h:messages id="messageList" styleClass="error" showSummary="true" showDetail="true" />
    <h:panelGrid columns="2" styleClass="gestionUsuariosFormTable"
            headerClass="gestionUsuariosFormHeader"
            footerClass="gestionUsuariosFormFooter"
            columnClasses="gestionUsuariosFormLabels, gestionUsuariosFormInputs"
width="600">
    <!-- Nombre -->
    <h:outputLabel for="login" value="Nombre"/>
    <h:panelGroup>
      <h:inputText id="nombre" styleClass="CajasTexto" size="30" maxlength="100"
              value="#{gestionUsuariosBean.nombre}">
          <f:validateLength maximum="50" />
                  </h:inputText>
    </h:panelGroup>
    <!-- Edad -->
    <h:outputLabel for="edad" value="Edad"/>
    <h:panelGroup>
     <h:inputText id="laEdad" converter="#{Integer}" styleClass="CajasTexto" size="30"
maxlength="3" value="#{gestionUsuariosBean.edad}">
```

```
        <f:validateLongRange maximum="100" minimum="5"/>
        </h:inputText>
       </h:panelGroup>
       <!-- Password -->
       <h:outputLabel for="password" value="Password"/>
       <h:panelGroup>
        <h:inputSecret id="password" styleClass="CajasTexto" size="30" maxlength="12"
value="#{gestionUsuariosBean.password}" />
       </h:panelGroup>
       <!-- Repetición de password -->
       <h:outputLabel for="password" value="Repetición de password"/>
       <h:panelGroup>
         <h:inputSecret id="passwordRepe" styleClass="CajasTexto" size="30" maxlength="12"
value="#{gestionUsuariosBean.password}">
           <t:validateEqual for="password" />
         </h:inputSecret>
       </h:panelGroup>
        <h:panelGroup>
          <h:commandButton action="resultado_validacion" value="Validar" />
          <f:verbatim> </f:verbatim>
         </h:panelGroup>
                 </h:panelGrid>
          </h:form>
          </f:view>
</body>
</html>
```

Añadimos la navegación en el fichero *WEB-INF/validaciones-config.xml* para que el botón *Validar* vaya al sitio adecuado:

```
...
...
<navigation-rule>
        <from-view-id>/validacion_estandar.jsp</from-view-id>
        <navigation-case>
         <from-outcome>resultado_validacion</from-outcome>
         <to-view-id>/resultado_validacion.jsp</to-view-id>
         <redirect/>
        </navigation-case>
</navigation-rule>
...
...
```

3. El resultado



Cuarto ejemplo: creando nuestro propio Validador

Para crear nuestro propio *Validador*, debemos seguir los siguientes pasos:

- Crear una clase que implemente la interfaz *javax.faces.validator.Validator*.
- Implementar el método *validate.*
- Registrar el nuevo validador en el contexto de JSF.

- Utilizarlo en la página insertando dentro del componente cuyo valor queramos validar el tag *<f:validator>.*

1. Qué va a validar nuestro Validador

Que el valor de un componente se corresponde un número de NIF válido.

El algoritmo que voy a utilizar para validar el NIF es el siguiente

- Compruebo que el valor a validar tiene una longitud igual a 9. Los primeros 8 caracteres deben ser números (corresponden al DNI) y el ı ser una letra (la del NIF)
- Almaceno la siguiente lista de letras en una variable:
  "TRWAGMYFPDXBNJZSQVHLCKE"
- Calculo el módulo entero de 23.
- Recupero de la lista de letras la que se encuentra en la posición resultado de efectuar el módulo entero de 23

2. Creamos el Validador

Como dijimos anteriormente, debe ser una implementación del interfaz *Validator*:

```
package com.autentia.tutorialValidacion;

import java.util.regex.Matcher;
import java.util.regex.Pattern;
import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.validator.Validator;
import javax.faces.validator.ValidatorException;

import org.apache.commons.lang.StringUtils;

/**
 * Validador de NIF
 * @author AUTENTIA
 */
public class NifValidator implements Validator
{
	/**
	 * Efectúa el proceso de validación
	 */
	public void validate(FacesContext context,
				UIComponent component, Object value) throws ValidatorException
	{
		// Si el valor es null, lo transformamos en un valor vacío
		String valor = StringUtils.defaultString((String)value);
		// el valor debe tener 9 posiciones, de las cuales las primeras deben ser dígitos y la última letra
		valor=valor.toUpperCase();
		Pattern mask =  Pattern.compile("[0-9]{8,8}[A-Z]");
		Matcher matcher = mask.matcher(valor);
		if(!matcher.matches())
			throw new ValidatorException(new FacesMessage("El componente " + component.getId() + "
				no contiene un NIF válido. Las 8 primeras posiciones deben ser numéricas"));
		String dni=valor.substring(0,8);
		String digitoControl = valor.substring(8,9);
		// Calculamos la letra de control
		String letras = "TRWAGMYFPDXBNJZSQVHLCKE";
		int posicion_modulo = Integer.parseInt(dni)%23;
		String digitoControlCalculado = letras.substring(posicion_modulo,posicion_modulo+1);
		if(!digitoControl.equalsIgnoreCase(digitoControlCalculado))
			throw new ValidatorException(new FacesMessage("El componente " + component.getId() +
								" no contiene un NIF válido"));
	}
}
```

3. Registramos el nuevo Validador en el contexto de JSF.

Para ello, editamos el fichero descriptor *WEB-INF/validaciones-config.xml* e insertamos las siguientes líneas:

```
...
<validator>
```

```
        <validator-id>autentia.nifValidator</validator-id>
                <validator-class>
            com.autentia.tutorialValidacion.NifValidator
        </validator-class>
</validator>
...
```

4. Creamos la página

La llamaremos *validacion_nuestra.jsp* y colgará de *WebContent*; su contenido es el siguiente:

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://myfaces.apache.org/extensions" prefix="t" %>

<html>
   <head>
                <title>AUTENTIA - TUTORIAL VALIDACIONES/CONVERSIONES</title>
                <link rel="stylesheet" type="text/css" href="css/estilos.css">
         </head>
         <body>
         <f:view>
   <h:form id="idUsuarios" name="gestionUsuariosBean">
   <h:messages id="messageList" styleClass="error" showSummary="true" showDetail="true" />
      <h:panelGrid columns="2" styleClass="gestionUsuariosFormTable"
                 headerClass="gestionUsuariosFormHeader"
                 footerClass="gestionUsuariosFormFooter"
                 columnClasses="gestionUsuariosFormLabels, gestionUsuariosFormInputs"
width="600">
      <!-- Nombre -->
      <h:outputLabel for="login" value="Nombre"/>
      <h:panelGroup>
          <h:inputText id="nombre" styleClass="CajasTexto" size="30" maxlength="100"
                     value="#{gestionUsuariosBean.nombre}">
                <f:validateLength maximum="50" />
                        </h:inputText>
      </h:panelGroup>
      <!-- Nif -->
      <h:outputLabel for="nif" value="Nif"/>
      <h:panelGroup>
       <h:inputText id="elNif" styleClass="CajasTexto" size="30" maxlength="10"
                     value="#{gestionUsuariosBean.nif}" required="true">
            <f:validator validatorId="autentia.nifValidator"/>
       </h:inputText>
      </h:panelGroup>

       <h:panelGroup>
         <h:commandButton action="resultado_validacion_nuestra" value="Validar" />
         <f:verbatim> </f:verbatim>
        </h:panelGroup>
                 </h:panelGrid>
         </h:form>
         </f:view>
</body>
</html>
```

Añadimos en *WEB-INF/validaciones-config.xml* la navegabilidad para el botón *Validar:*

```
...
<navigation-rule>
         <from-view-id>/validacion_nuestra.jsp</from-view-id>
         <navigation-case>
          <from-outcome>resultado_validacion_nuestra</from-outcome>
          <to-view-id>/resultado_validacion_nuestra.jsp</to-view-id>
```

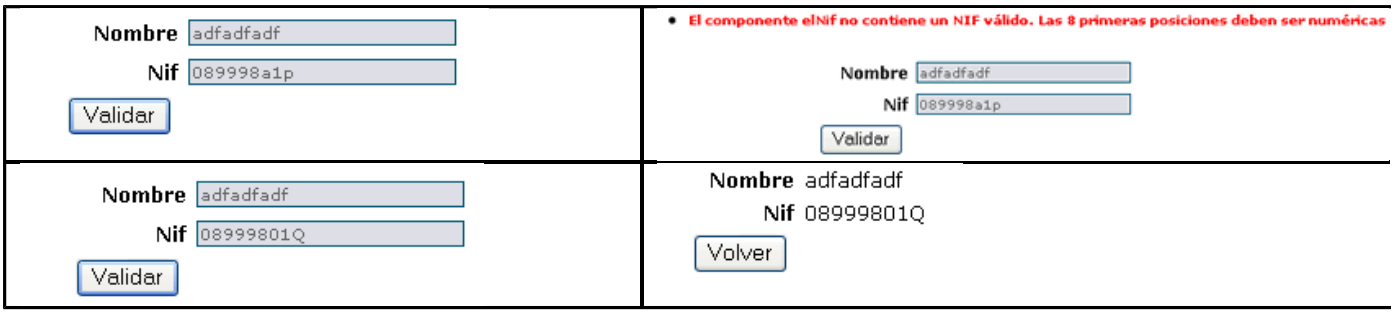
```
        <redirect/>
        </navigation-case>
</navigation-rule>
....
```

Y creamos la página de resultados *WebContent/resultado_validacion_nuestra.jsp:*

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://myfaces.apache.org/extensions" prefix="t" %>

<html>
  <head>
            <title>AUTENTIA - TUTORIAL VALIDACIONES/CONVERSIONES</title>
            <link rel="stylesheet" type="text/css" href="css/estilos.css">
      </head>
      <body>
      <f:view>
  <h:form id="idUsuarios" name="gestionUsuariosBean">
  <h:messages id="messageList" styleClass="error" showSummary="true" showDetail="true" />
    <h:panelGrid columns="2" styleClass="gestionUsuariosFormTable"
            headerClass="gestionUsuariosFormHeader"
            footerClass="gestionUsuariosFormFooter"
            columnClasses="gestionUsuariosFormLabels, gestionUsuariosFormInputs"
width="600">
    <!-- Nombre -->
    <h:outputLabel for="login" value="Nombre"/>:
    <h:panelGroup>
        <h:outputText value="#{gestionUsuariosBean.nombre}"/>
    </h:panelGroup>
    <!-- Nif -->
    <h:outputLabel for="nif" value="Nif"/>
    <h:panelGroup>
     <h:outputText value="#{gestionUsuariosBean.nif}"/>
    </h:panelGroup>
    <h:panelGroup>
      <h:commandButton action="home" value="Volver" />
      <f:verbatim> </f:verbatim>
     </h:panelGroup>
            </h:panelGrid>
      </h:form>
      </f:view>
</body>
</html>
```

5. Probamos

Desplegamos la aplicación en Tomcat y vemos cual es el resultado:



Si necesita ayuda al plantear sus soluciones tecnológicas no dude en contactar con nosotros a través de nuestra web www.autentia.com. Estar encantados de ayudarle en su problema poniendo a su disposición a nuestros mejores expertos a precios asequibles.

Otros Tutoriales Recomendados (También ver todos)

Nombre Corto	Descripción
Upload de ficheros en JSF	Os mostramos de una forma sencilla y guiada como crear una utilidad de upload de fiche utilizando JSF
Introducción a Struts Flow	Struts Flow es un módulo de extensión del conocido framework Struts, que facilita la implementación del flujo de páginas de una aplicación web
Activar el soporte SSL en Struts	Os mostramos las particularidades de uso y configuración de Struts para trabajar con SS
Manejar tablas de datos con JSF	En este tutorial os mostramos un ejemplo de utilización de la extension del componente DataTable, realizada por la implementación Tomahawk de MyFaces
Extender la validación en Struts	Os mostramos con un ejemplo como extender los mecanismos de validación en Struts, utilizando el framework Commons Validator
Upload de ficheros en Struts	En este tutorial os mostramos paso a paso como construir una sencilla aplicación de uplo de ficheros utilizando Struts
JSF y comparativa con Struts	Os mostramos los pasos necesarios para empezar a utilizar JSF (Java Server Faces) y su comparación / relación con Struts
Introducción a Spring Web Flow	Spring Web Flow es un módulo de extensión del framework Spring, que facilita la implementación del flujo de páginas de una aplicación web

Utilizando JSTL en JSF	Os mostramos como utilizar la librería estandar de etiquetas en JSF, implementando una sencilla aplicación web
Probando entornos para JSF	En este tutorial os mostramos con ejemplos como utliizar dos conocidos entornos de desarrollo para JSF: Exadel Studio y Sun Studio Creator