EL (Expression Language)

La especificación de JSP 2.0 introdujo Expression Language (EL), fue diseñado para ser más amigable que el manejo de scriptlets. Es similar a JavaScript y a XPath, de ahí que fue inspirado en ECMAScript que es la versión estándar de JavaScript.
EL no es un lenguaje de programación en sí mismo, su único propósito es:

- Referenciar objetos y sus propiedades.
- Escribir expresiones simples.

Las expresiones EL se pueden utilizar con tres valores de atributos distintos. En primer lugar, pueden aplicarse cuando un valor de atributo tiene una sola expresión. En segundo lugar, pueden utilizarse cuando el valor del atributo contiene una o más expresiones rodeadas o separadas por texto. Por último, pueden utilizarse si el valor del atributo solo contiene texto.

Expresiones EL	Implementación
Una sola expresión	<xyz.tag value="${expresión}"/>
Una o más expresiones	<xyz.tag value="texto${expresión}texto${expresión}"/>
Solo texto	<xyz.tag value="texto"/>

Como se dijo anteriormente, EL ha sido creado para reemplazar a los scriptlets.
Para poder deshabilitar el uso de scriptlets, se debe editar el archivo web.xml, y agregar la siguiente directiva:

```
<jsp-config>
   <jsp-property-group>
         <url-pattern>*.jsp</url-pattern>
         <scripting-invalid>true</scripting-invalid>
   </jsp-property-group>
</jsp-config>
```

A su vez, se puede pretender deshabilitar el uso de EL, para ello en el archivo web.xml se debe especificar:

```
<jsp-config>
   <jsp-property-group>
         <url-pattern>*.jsp</url-pattern>
         <el-ignored>true</el-ignored >
   </jsp-property-group>
</jsp-config>
```

Esta directiva puede directamente aplicarse a la página JSP incluyendo:

<%@page isELIgnored="true" %>

Las expresiones se componen de:

- *Identificadores.* Hay once identificadores reservados que corresponden a once objetos implícitos. El resto de identificadores sirven para crear variables.
- *Literales*. Son números, cadenas delimitadas por comillas simples o dobles, y los valores true, false, y null.
- *Operadores*. Permiten comparar y operar con identificadores y literales.
- *Operadores de acceso*. Se usan para referenciar propiedades de los objetos.

Operadores

Operadores matemáticos

Descripción	Símbolo	Expresión de ejemplo	Resultado del ejemplo
Multiplicación	*	${2*4}	8
División	/ o div	${5/2}	2.5
Resto de una división entera	% o mod	${5%2}	1
Suma	+	${2+2}	4
Resta	-	${7-2}	5

Operadores de comparación

Descripción	Símbolo	Expresión de ejemplo	Resultado del ejemplo

Igualdad	== o eq	${2 == '2'}	Verdadero
Desigualdad	!= o ne	${2 != 2}	Falso
Menor que	< o lt	${2 < 2}	Falso
Mayor que	> o gt	${3 > 2}	Verdadero
Menor o igual que	<= o le	${2 <= 2}	Verdadero
Mayor o igual que	>= o ge	${1 >= 2}	Falso

Operadores lógicos

Descripción	Símbolo	Expresión de ejemplo	Resultado del ejemplo
Negación	! o not	${!(2 == 2)}	Falso
Y	&& o and	${(2 == 2) && (2 >= 0)}	Verdadero
Ó	|| o or	${(2 == 2) || (2 != 2)}	Verdadero

Operadores de acceso

Para acceder a object.property se emplea: object['property']. Por ejemplo, ${param['login']}

Operador empty

El operador empty comprueba si una colección o cadena es vacía o nula. Por ejemplo, ${empty param.login}
Otra manera de hacerlo es usando la palabra clave null. Por ejemplo, ${param.login == null}

Objetos implícitos

Estos objetos no requieren ser declarados ya que se declaran automáticamente.

Objeto	Descripción
pageContext	Accede al objeto PageContext que da acceso a todos los espacios de nombre asociados con una página JSP.
pageScope	Map que contiene valores y nombres de atributos centrados en la página.
requestScope	Map que contiene valores y nombres de atributos centrados en la solicitud.
sessionScope	Map que contiene valores y nombres de atributos centrados en la sesión.
applicationScope	Map que contiene valores y nombres de atributos centrados en la aplicación.
param	Map que relaciona nombres de parámetros con valores de parámetro String.
paramValues	Map que relaciona nombres de parámetros con un componente de vector String de todos los valores de ese parámetro.
header	Map que contiene nombres de encabezado con valores de parámetro String.
headerValues	Map que contiene nombres de encabezado de un componente de vector String.
cookie	Map que contiene objetos cookie Web.
initParam	Map que alberga nombres de parámetros de inicialización de la página Web.

Funciones EL

Las funciones representan un modo de extender la funcionalidad del lenguaje de expresiones.
Una función EL es mapeada a un método **estático** de una clase Java. El mapeo se especifica dentro de un TLD (Tag Library Descriptor).
Una función en EL puede tomar cualquier número de parámetros. Estos deben ser declarados en el TLD.
Debe respetarse que los nombres de las funciones sean únicos, si no se generará un error al momento de traslación.

Dentro del TLD los parámetros y el tipo de retorno deben ser clases Java totalmente calificadas, por ejemplo, java.lang.String La sintaxis de la firma de una función EL (<function-signature>) es,

return_type static_function_name(parameter_1_type, ..., parameter_n_type)

Si la firma de la función definida en el TLD no aparea exactamente con la función en la clase Java, un error a tiempo de traslación ocurrirá.

Un TLD es un archivo XML que declara una librería de tags, describiendo la clase que implementa cada tag. Cada TLD puede describir cero o más funciones estáticas. A cada función se le asigna un nombre y método que implementa de la clase Java.

Veamos un ejemplo sencillo,

Clase Java

```
package ar.edu.ubp.das.util;

public class MyStringFunctions {

   public static String lower(String param) {
      return param.toLowerCase();
   }

   public static String upper(String param) {
      return param.toUpperCase();
   }

   public static String wordcap(String param) {
         StringBuilder string = new StringBuilder(param.length());
         String[]     words  = param.split("\\s");

         for(int i = 0, len = words.length; i < len; i++) {
           string.append(Character.toUpperCase(words[i].charAt(0)))
                 .append(words[i].substring(1))
                 .append(" ");
         }
         string.trimToSize();
         return string.toString();
   }

}
```

Archivo TLD

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
jsptaglibrary_2_1.xsd">
  <tlib-version>1.0</tlib-version>
  <short-name>myelfunctions</short-name>
  <uri>/WEB-INF/tlds/myELFunctions</uri>
  <function>
      <name>toLower</name>
      <function-class>ar.edu.ubp.das.util.MyStringFunctions</function-class>
      <function-signature>java.lang.String lower(java.lang.String)</function-signature>
  </function>
  <function>
      <name>toUpper</name>
      <function-class>ar.edu.ubp.das.util.MyStringFunctions</function-class>
      <function-signature>java.lang.String upper(java.lang.String)</function-signature>
  </function>
  <function>
      <name>toWordcap</name>
      <function-class>ar.edu.ubp.das.util.MyStringFunctions</function-class>
      <function-signature>java.lang.String wordcap(java.lang.String)</function-signature>
  </function>
</taglib>
```

Página JSP

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<%@taglib prefix="fn" uri="/WEB-INF/tlds/myELFunctions.tld"%>

<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Ejemplo</title>
   </head>
   <body>
```

```
      <b>${fn:toLower("iHOLA MUNDO!! :)")}</b><br/>
      <b>${fn:toUpper("ihola mundo!! :)")}</b><br/>
      <b>${fn:toWordcap("iHOLA MUNDO!! :)")}</b>
   </body>
</html>
```

Resultado

ihola mundo!! :)
iHOLA MUNDO!! :)
ihola Mundo!! :)

Funciones EL vs. Custom Tags

De acuerdo a las siguientes preguntas podemos determinar si es conveniente utilizar funciones EL o custom tags.

(1) ¿Se necesita conocimiento del ambiente? Sí, emplee tags. Un tag provee un fácil acceso al pageContext y otras variables, las funciones no, para accederlo deben pasarse como parámetros.
(2) ¿Se requiere comportamiento interactivo sobre el cuerpo del HTML? Sí, emplee tags.
(3) ¿Se quiere proveer un pequeño pedazo de código reusable que actúa con uno o más argumentos? Sí, emplee una función EL.
(4) ¿Se quiere reusar código Java existente? Sí, emplee una función EL.

JavaBeans

Un JavaBean es un modelo de componente de software reusable en lenguaje Java, que puede ser manipulado visualmente por una herramienta de programación. Cualquier clase Java que adhiere a ciertas convenciones en lo relativo a las propiedades, métodos y eventos puede ser un JavaBean.
Los componentes JavaBean son conocidos como beans.

Un bean es un componente, portable, visible o no, autocontenido y escrito en Java que cumple con las recomendaciones de la especificación JavaBean, en lo referente a los protocolos de comunicación y configuración del objeto generado.
Un bean es una clase Java que sigue determinadas convenciones, de manera que pueda ser empleada como un componente por herramientas de desarrollo, para la construcción de aplicaciones.

Características

(1) *Introspección*: Las herramientas de construcción de beans pueden descubrir las características de un bean (propiedades, métodos y eventos) y analizar cómo trabaja. La información de una clase, tales como nombre de los métodos, parámetros y tipos de retorno, pueden ser descubiertas por otras clases.
(2) *Propiedades*: Una propiedad es un atributo de un bean que puede ser leído o escrito por el cliente del bean, a través de métodos regulares cuyos nombres responden a los lineamientos de construcción de JavaBeans.
(3) *Customización*: Permite al desarrollador establecer y personalizar nuevas propiedades que gobiernan el comportamiento del bean.
(4) *Eventos*: Permite a un bean comunicarse con otros beans.
(5) *Persistencia*: Habilita al bean a salvar y recuperar su estado. Debe implementar la interface java.io.Serializable

Convenciones

(1) Un JavaBean es una clase cuyo constructor no tiene argumentos.
(2) No puede tener variables de instancias públicas, también conocidos como atributos de la clase.
(3) Los valores persistentes deberán ser accedidos a través de los métodos getter y setter, conocidos como métodos de acceso.
(4) Los nombres de los métodos getter o setter, se componen de la palabra get o set, más el nombre de la propiedad con su primera letra en mayúscula.
(5) Un método regular getter no tiene parámetros y retorna un valor del tipo de la propiedad.
(6) Un método setter tiene un único parámetro que es del tipo de la propiedad y tiene un retorno de tipo void.
(7) Una propiedad legible debe tener un método getter.
(8) Una propiedad escribible debe tener un método setter.
(9) Dependiendo de la combinación de métodos getter y setter, una propiedad es solo lectura, solo escritura o lectura y escritura.
(10) Una propiedad solo lectura no necesariamente debe coincidir con una variable de instancia. Puede ser un valor computado resultado de combinar varias variables de instancia.
(11) El tipo de una propiedad puede ser una clase, una interface o un tipo primitivo.
(12) Los beans pueden tener propiedades de múltiples valores representados por un vector de cualquier tipo. Esto es denominado una "propiedad indexada".

(13) Dos tipos de métodos de acceso pueden tener una propiedad indexada:
   a. Métodos de lectura y escritura de todo el vector.
   b. Métodos de lectura y escritura de un elemento del vector.
(14) En una propiedad booleana se puede emplear el método getter regular. Pero la recomendación es usar la palabra is combinada con el nombre de la propiedad con su primera letra en mayúscula.
(15) El método setter de una propiedad booleana sigue el patrón general.

Veamos algunos ejemplos,

```
package ar.edu.ubp.das.beans;

public class ProductoBean implements java.io.Serializable {

   private String nomProducto;
   private int cantidad;
   private float precioUnitario;

   public ProductoBean() {
   }

   public int getCantidad() {
      return cantidad;
   }

   public String getNomProducto() {
      return nomProducto;
   }

   public float getPrecioUnitario() {
      return precioUnitario;
   }

   public float getPrecioTotal() {
      return precioUnitario * cantidad;
   }

   public void setCantidad(int cantidad) {
      this.cantidad = cantidad;
   }

   public void setNomProducto(String nomProducto) {
      this.nomProducto = nomProducto;
   }

   public void setPrecioUnitario(float precioUnitario) {
      this.precioUnitario = precioUnitario;
   }

}
```

Tipos de beans

Existen dos tipos:

(1) Value beans: Encapsula toda la información acerca de una entidad, tal como un usuario, producto, etc.
(2) Utility beans: Efectúa alguna acción, tal como guardar información en la base de datos o enviar un e-mail. "Utility beans" pueden emplear "Value beans" como entrada o producirlos como resultado de una acción.

Empleo de beans

Para declarar el uso de una instancia bean en una página JSP se usa la directiva:

Sintaxis 1
<jsp:useBean id="name" scope="scope" Bean-Specification/>

Sintaxis 2
<jsp:useBean id="name" scope="scope" Bean-Specification>
Creation-Body

</jsp:useBean>

id: Nombre con el cual se referenciará el bean dentro del JSP. Atributo obligatorio.

scope: Ámbito de visibilidad del bean, entre ellos:

Ámbito	Duración
Page	El bean solo será válido dentro de la página JSP y será regenerado en cada nueva petición.
Request	El bean será válido a lo largo de la petición y estará disponible para incluirse o reenviarse a otras páginas JSP.
Session	El bean se asocia a una sesión en particular, la cual es responsable de su creación. El bean será válido mientras la sesión exista.
application	El bean es común a todas las sesiones y será válido hasta que la aplicación Web finalice.

Bean-Specification: Atributos adicionales que sirven para declarar el bean. Debe considerarse que el uso de los mismos es mutuamente excluyente, es decir, que existen determinadas combinaciones a respetar. Entre ellos encontramos:

Atributo	Descripción
class="className"	Clase Java que implementa al objeto.
type="typeName"	Especifica el tipo de la variable que identifica al bean. Este debe corresponderse con el nombre de la clase Java o superclase o interface que implementa la clase Java.
beanName="beanName"	El nombre del bean en caso de que esté serializado (persistencia), como lo suministraríamos en el método java.beans.Beans.instantiate().

Los atributos antes especificados deben aparecer en las siguientes combinaciones:

- class
- type
- class y type
- beanName y type

Creation-Body: Se ejecuta solo si el bean no existe y sirve para inicializar el mismo. El bean solo es creado si "no es encontrado dentro del scope establecido".

Para establecer un valor a una propiedad del bean, se usa:

<jsp:setProperty name="beanId" property="propertyName" Property-Specification />

name: Nombre con el cual se identifica el bean en la página JSP.

property: Nombre de la propiedad del bean a establecer su valor.

Property-Specification: Atributos adicionales cuyo uso es mutuamente excluyente. Entre ellos encontramos:

Atributo	Descripción
value="value"	Valor a establecer.
param="param"	Nombre del parámetro disponible en el HttpServletRequest cuyo valor será asignado a la propiedad.

Si no se especifica ninguno de estos atributos, se usa por defecto "param"

Para obtener el valor actual de una propiedad de un bean, se usa:

<jsp:getProperty name="beanId" property="propertyName" />

name: Nombre con el cual se identifica el bean en la página JSP.

property: Nombre de la propiedad del bean a establecer su valor.

Con todo lo antes expuesto veamos un ejemplo sencillo,

Clase Java

ar.edu.ubp.das.beans;

```
public class ContadorBean {

   private int contador;

   public ContadorBean() {
      this.contador = 0;
   }

   public void setContador(int c) {
      this.contador += c;
   }

   public int getContador() {
      return this.contador;
   }

}
```

Página JSP

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Beans</title>
   </head>
   <body>
      <jsp:useBean id="cont" class="ar.edu.ubp.das.beans.ContadorBean" scope="session"/>
      <jsp:setProperty name="cont" property="contador" value="1"/>

      <h2>Contador: <jsp:getProperty name="cont" property="contador"/></h2>
      <h2>Contador: ${cont.contador}</h2>

      <a href="index.jsp" target="_self">Volver a cargar</a>
   </body>
</html>
```

Resultado

Al estar declarado el bean con scope de "sesión", cada vez que se recargue la página se irá incrementando el contador.

Inicialización de un JavaBean

Existen dos posibilidades para inicializar un bean, veamos ejemplos para explicar cada una de ellas.

Clase Java

```
ar.edu.ubp.das.beans;

public class PersonaBean {

   private String apellido, nombre;

   public PersonaBean() { }

   public void setApellido(String a) {
      this.apellido = a;
   }

   public void setNombre(String n) {
      this.nombre = n;
   }

   public String getApellido() {
```

```
        return this.apellido;
    }

    public String getNombre() {
        return this.nombre;
    }

    public String getNombreCompleto() {
        return this.apellido + ", " + this.nombre;
    }

}
```

Posibilidad 1: Inicializar cada propiedad del bean de acuerdo al nombre de los parámetros del formulario procesado.

Página JSP 1: index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Beans</title>
    </head>
    <body>
        <form action="resultado.jsp" method="post">
            <p>Apellido: <input type="text" name="ape" maxlength="100" size="109"/></p>
            <p>Nombre: <input type="text" name="nom" maxlength="100" size="109"/></p>
            <input type="submit" value="Mostrar datos ingresados"/>
        </form>
    </body>
</html>
```

Página JSP 2: resultado.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Resultado</title>
    </head>
    <body>
        <jsp:useBean id="per" class="ar.edu.ubp.das.beans.PersonaBean" scope="request">
          <jsp:setProperty name="per" property="apellido" param="ape"/>
          <jsp:setProperty name="per" property="nombre" param="nom"/>
        </jsp:useBean>
        <h3>Par&aacute;metros Recibidos</h3>
        <h3>Directivas JSP</h3>
        <p><b>Apellido</b>: <jsp:getProperty name="per" property="apellido"/></p>
        <p><b>Nombre</b>: <jsp:getProperty name="per" property="nombre"/></p>
        <p><b>Nombre Completo</b>: <jsp:getProperty name="per" property="nombreCompleto" /></p>
        <h3>Scriptlets</h3>
        <p><b>Apellido</b>: <%= per.getApellido() %></p>
        <p><b>Nombre</b>: <%= per.getNombre() %></p>
        <p><b>Nombre Completo</b>: <%= per.getNombreCompleto() %></p>
        <h3>EL</h3>
        <p><b>Apellido</b>: ${per.apellido}</p>
        <p><b>Nombre</b>: ${per.nombre}</p>
        <p><b>Nombre Completo</b>: ${per.nombreCompleto}</p>
        <br>
        <a href="index.jsp">Volver</a>
    </body>
</html>
```

Posibilidad 2: Inicializar todas las propiedades del bean usando en el atributo "property" un asterisco (*). Esta posibilidad solo es válida si "todos" los parámetros del formulario procesado se denominan igual que las propiedades del bean.

Página JSP 1: index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Beans</title>
   </head>
   <body>
      <form action="resultado.jsp" method="post">
         <p>Apellido: <input type="text" name="apellido" maxlength="100" size="109"/></p>
         <p>Nombre: <input type="text" name="nombre" maxlength="100" size="109"/></p>
         <input type="submit" value="Mostrar datos ingresados"/>
      </form>
   </body>
</html>
```

Página JSP 2: resultado.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Resultado</title>
   </head>
   <body>
      <jsp:useBean id="per" class="ar.edu.ubp.das.beans.PersonaBean" scope="request">
        <jsp:setProperty name="per" property="*"/>
      </jsp:useBean>
      <h3>Par&aacute;metros Recibidos</h3>
      <h3>Directivas JSP</h3>
      <p><b>Apellido</b>: <jsp:getProperty name="per" property="apellido"/></p>
      <p><b>Nombre</b>: <jsp:getProperty name="per" property="nombre"/></p>
      <p><b>Nombre Completo</b>: <jsp:getProperty name="per" property="nombreCompleto" /></p>
      <h3>Scriptlets</h3>
      <p><b>Apellido</b>: <%= per.getApellido() %></p>
      <p><b>Nombre</b>: <%= per.getNombre() %></p>
      <p><b>Nombre Completo</b>: <%= per.getNombreCompleto() %></p>
      <h3>EL</h3>
      <p><b>Apellido</b>: ${per.apellido}</p>
      <p><b>Nombre</b>: ${per.nombre}</p>
      <p><b>Nombre Completo</b>: ${per.nombreCompleto}</p>
      <br>
      <a href="index.jsp">Volver</a>
   </body>
</html>
```

Veamos como trabajamos para inicializar un bean con propiedades indexadas.

Clase Java

```
ar.edu.ubp.das.beans;

public class CategoriasBean {

   private String[] categorias;

   public CategoriasBean() {
         this.categorias = new String[100];
   }
```

```
    public String getCategoria(int posicion) {
        return categorias.length > 0 ? categorias[posicion] : null;
    }

    public String[] getCategorias() {
        return this.categorias;
    }

    public void setCategoria(String categoria, int posicion) {
                this.categorias[posicion] = categoria;
    }

    public void setCategorias(String[] categorias) {
        this.categorias = categorias;
    }

}
```

Página JSP 1: index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Beans</title>
    </head>
    <body>
        <h1>Categor&iacute;as</h1>
        <form action="resultado.jsp" method="post">
            <p><input type="checkbox" id="A" name="categoria" value="A"/><label for="A">Categor&iacute;a
A</label></p>
            <p><input type="checkbox" id="B" name="categoria" value="B"/><label for="B">Categor&iacute;a
B</label></p>
            <p><input type="checkbox" id="C" name="categoria" value="C"/><label for="C">Categor&iacute;a
C</label></p>
            <p><input type="checkbox" id="D" name="categoria" value="D"/><label for="D">Categor&iacute;a
D</label></p>
            <p><input type="checkbox" id="E" name="categoria" value="E"/><label for="E">Categor&iacute;a
E</label></p>
            <input type="submit" value="Continuar..."/>
        </form>
    </body>
</html>
```

Página JSP 2: resultado.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Resultado</title>
    </head>
    <body>
        <h1>Categor&iacute;as Seleccionadas</h1>
        <jsp:useBean id="cat1" class="ar.edu.ubp.das.beans.CategoriasBean" scope="request">
            <%
            String[] c = request.getParameterValues("categoria");
             if(c != null) {
                for(int i = 0; i < c.length; i++) {
                   cat1.setCategoria(c[i], i);
                }
             }
            %>
```

```
      </jsp:useBean>
      <p>Primera Categor&iacute;a Elegida (Scriptlet): <%= cat1.getCategoria(0)%></p>

      <jsp:useBean id="cat2" class="src.CategoriasBean" scope="request">
         <jsp:setProperty name="cat2" property="categorias" value="${paramValues['categoria']}" />
      </jsp:useBean>
      <p>Primera Categor&iacute;a Elegida (EL): ${cat2.categorias[0]}</p>
      <a href="index.jsp">Volver</a>
   </body>
</html>
```