

CC68J APLICACIONES EMPRESARIALES CON JEE

JAVA CUSTOM TAGS

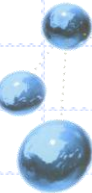
Creando tags personalizados

Profesores:

■ Andrés Farías

Objetivos

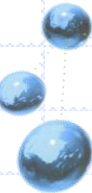
- Introducción General.
- Taglibs en detalle:
 - ✓ Tag Handlers.
 - ✓ Atributos de un TagLib.
 - ✓ Iteración en un TagLib.





¿Para qué querer tags propios?

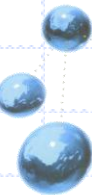
INTRODUCCIÓN



Introducción

Problemas con JSPs

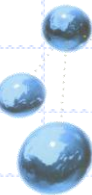
- *JavaBeans* nos permiten separar la parte de presentación de una página JSP de la implementación de una regla de negocio
 - ✓ Sin embargo, sólo 3 elementos acción en JSP se pueden usar para acceder a un bean:
 - ◆ `jsp:useBean`
 - ◆ `jsp:getProperty`
 - ◆ `jsp:setProperty`
 - ✓ Por tanto a menudo tenemos que incluir código en un JSP
 - ✓ JSP (1.1 en adelante) define *Custom Tags* que pueden ser usadas para definir acciones propietarias, de manera que en nuestro JSP únicamente tenemos código de marcado



Introducción

Características

- **Custom tags** (etiquetas personalizadas) tienen acceso a todos los objetos disponibles en una JSP.
 - ✓ Pueden ser personalizadas usando atributos.
- Como los JavaBeans se centran en reusabilidad de código.
- Los custom tags son extensiones de JSP definidas por el usuario.
- Las librerías de etiquetas de JSP son:
 - ✓ Creadas por Programadores, expertos en acceder a datos y servicios, y son
 - ✓ Usadas por Diseñadores de aplicaciones web especialistas en presentación.



Ejemplo Simple: Time Now!

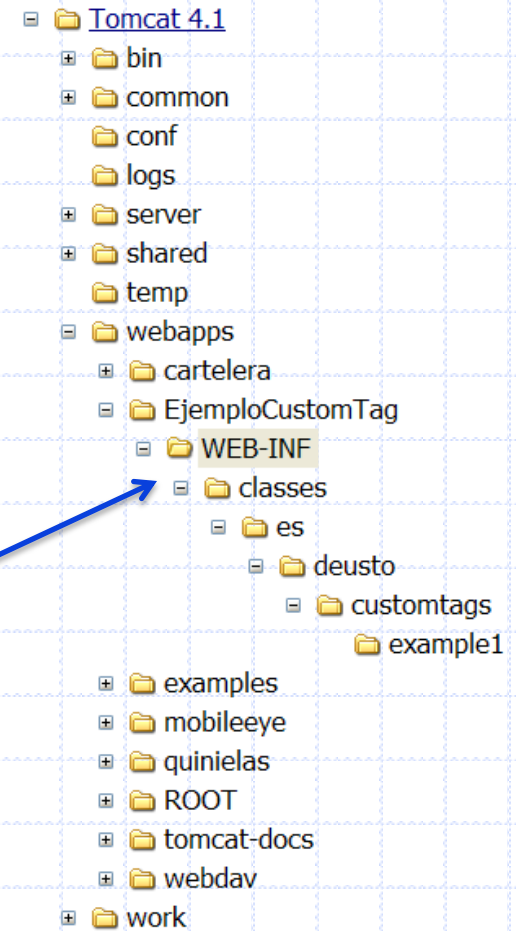
Objetivo y contexto

■ Objetivo:

- ✓ construir una etiqueta personalizada que envía el siguiente string al browser: “Hoy es: ” seguido por la fecha y hora del sistema.

■ Ubicación:

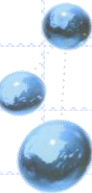
- ✓ En el directorio clases de nuestra aplicación web.
- ✓ Se declara luego en descriptor de despliegue **web.xml**.



TagLib simple: Time Now!

Un tag-library paso a paso

1. Crear un fichero TLD con nombre **hello.tld**, y guardarlo en **WEB-INF**
2. Escribir la clase **HelloWorldTag.java**.
3. Modificar el fichero **web.xml** añadiendo un elemento **<taglib>**
4. Crear una JSP (**SimplePage.jsp**) que utiliza la etiqueta personalizada definida.
5. Desplegar nuevamente la aplicación si ya estaba desplegada en el contenedor de servlets.



TagLib simple: Time now!

Paso 1: creación del taglib

- Crear un fichero TLD con nombre **timenow.tld**, y guardarlo en el directorio **WEB-INF/tlds**

Debe estar en el directorio WEB-INF, no necesariamente en una sub-carpeta.

```
<taglib ... version="2.0">
```

```
<description>Despliega la fecha y hora actual</description>
```

```
<display-name>Time Now!</display-name>
```

```
<tlib-version>1.0</tlib-version>
```

```
<short-name>timenow</short-name>
```

```
<uri>timenow</uri>
```

```
<tag>
```

```
<display-name>now</display-name>
```

```
<name>now</name>
```

```
<tag-class>cl.uchile.cc68j.taglibs.TimeNowCT</tag-class>
```

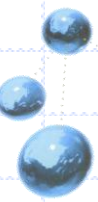
```
<body-content>empty</body-content>
```

```
</tag>
```

```
</taglib>
```

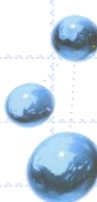
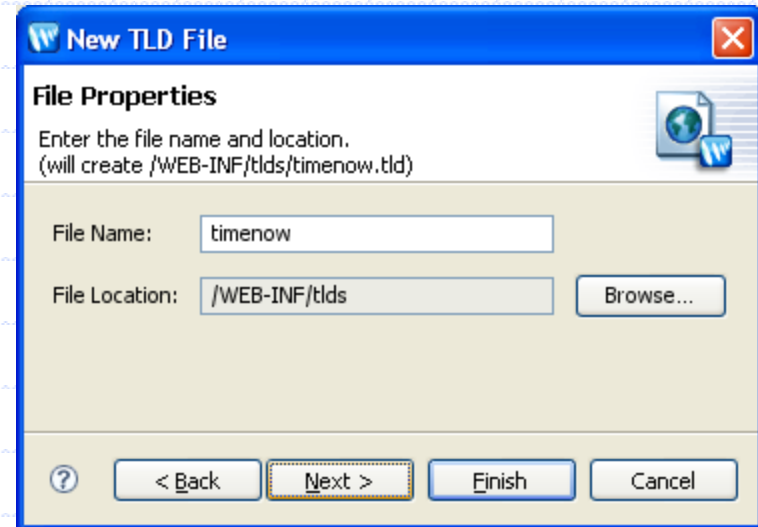
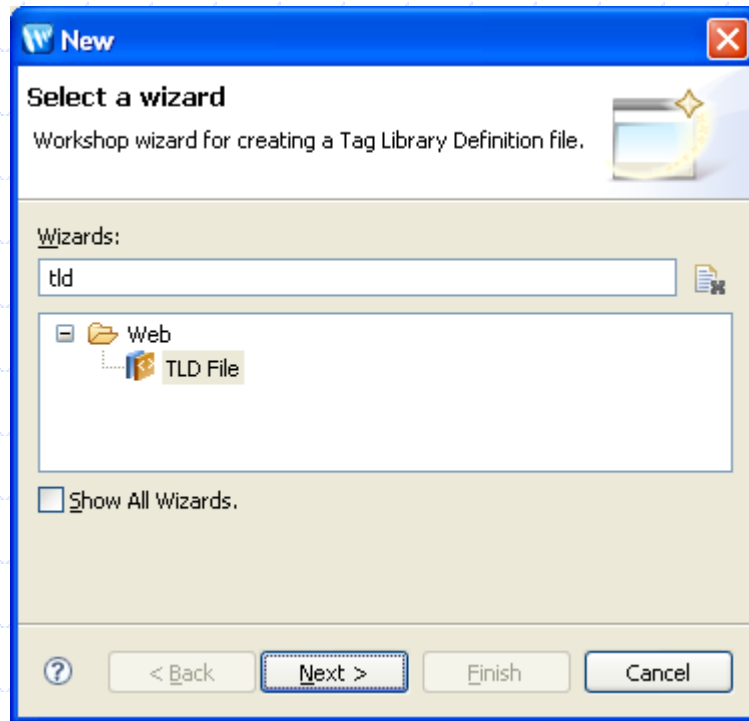
Información general de la librería de etiquetas personalizadas.

Información sobre el tag (la etiqueta) de la librería.



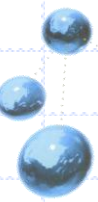
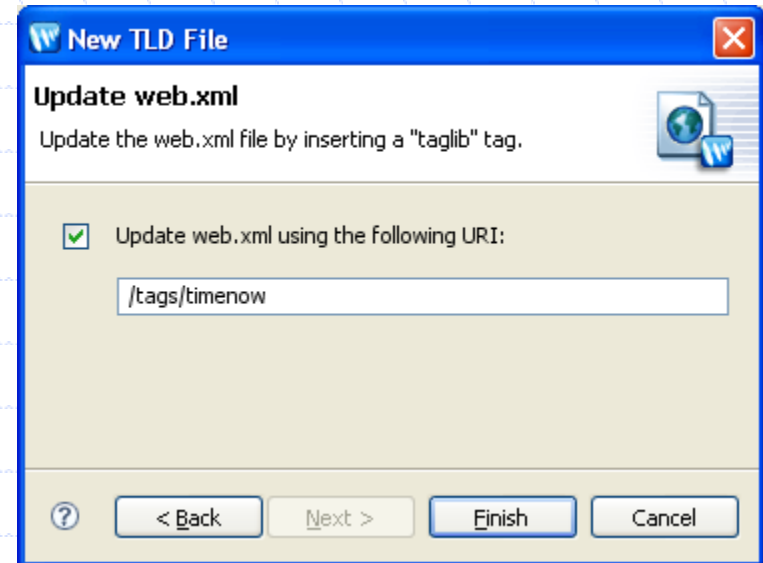
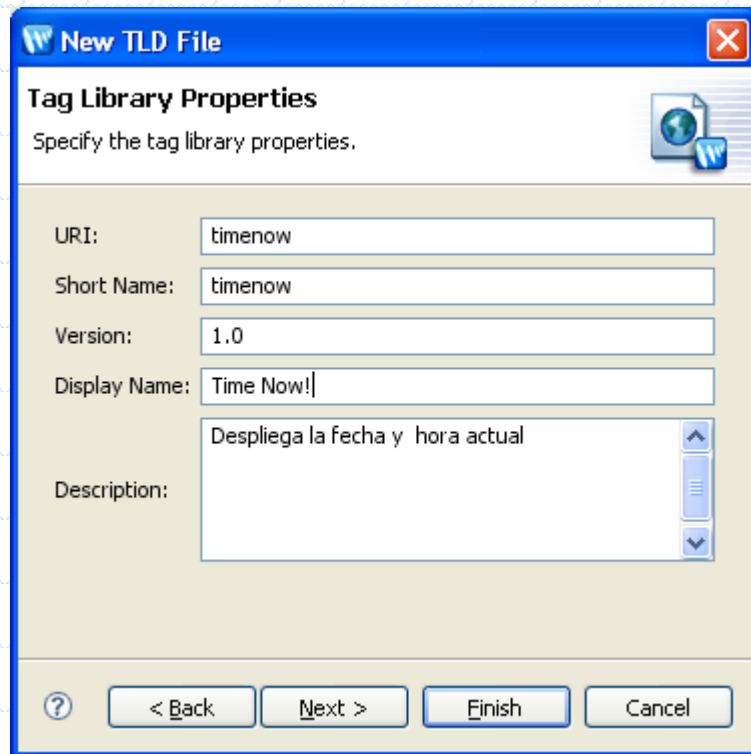
TagLib simple: Time now!

Paso 1: creación del taglib con el asistente (1/3)



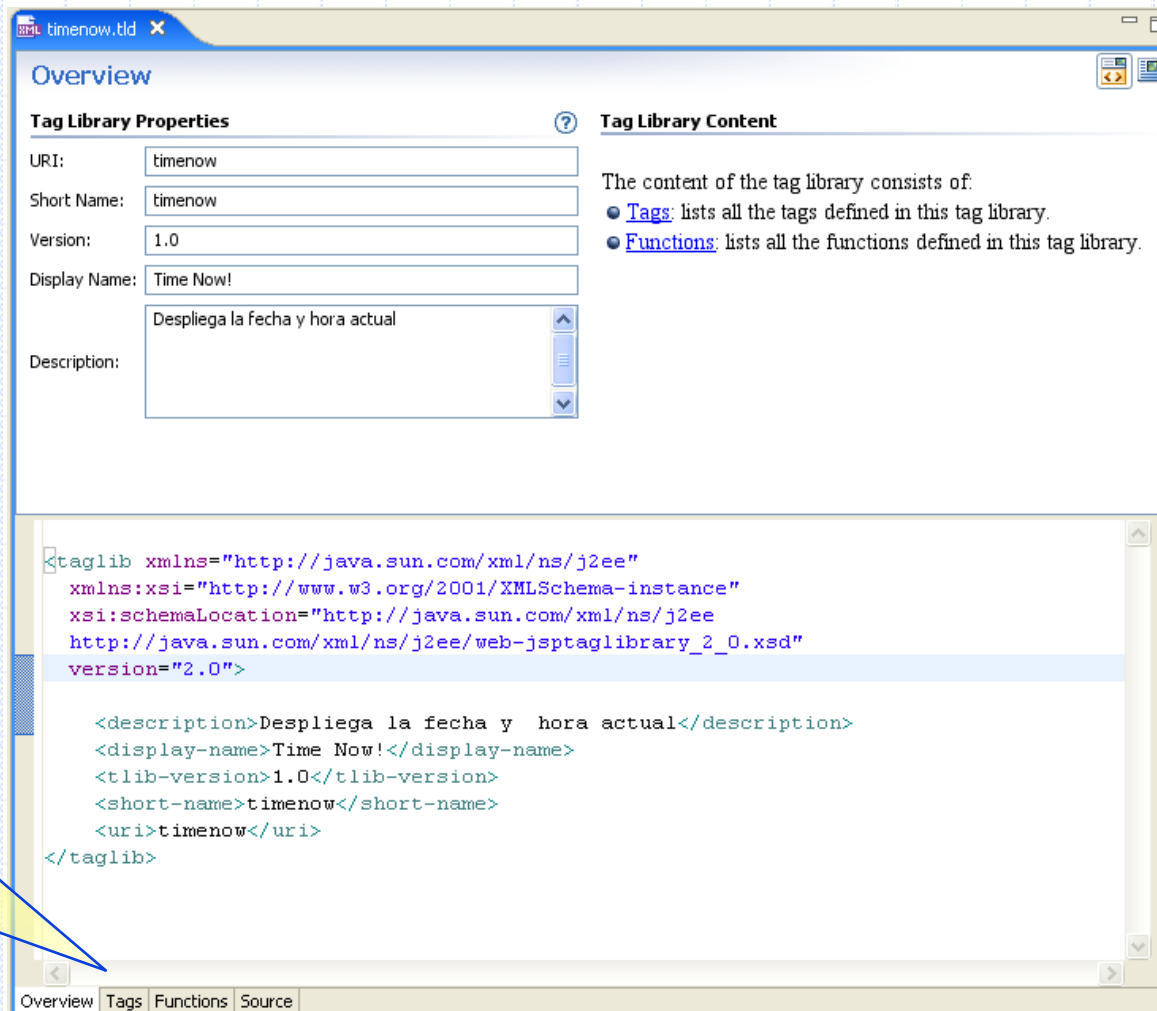
TagLib simple: Time now!

Paso 1: creación del taglib con el asistente (2/3)



TagLib simple: Time now!

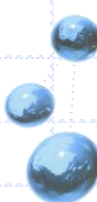
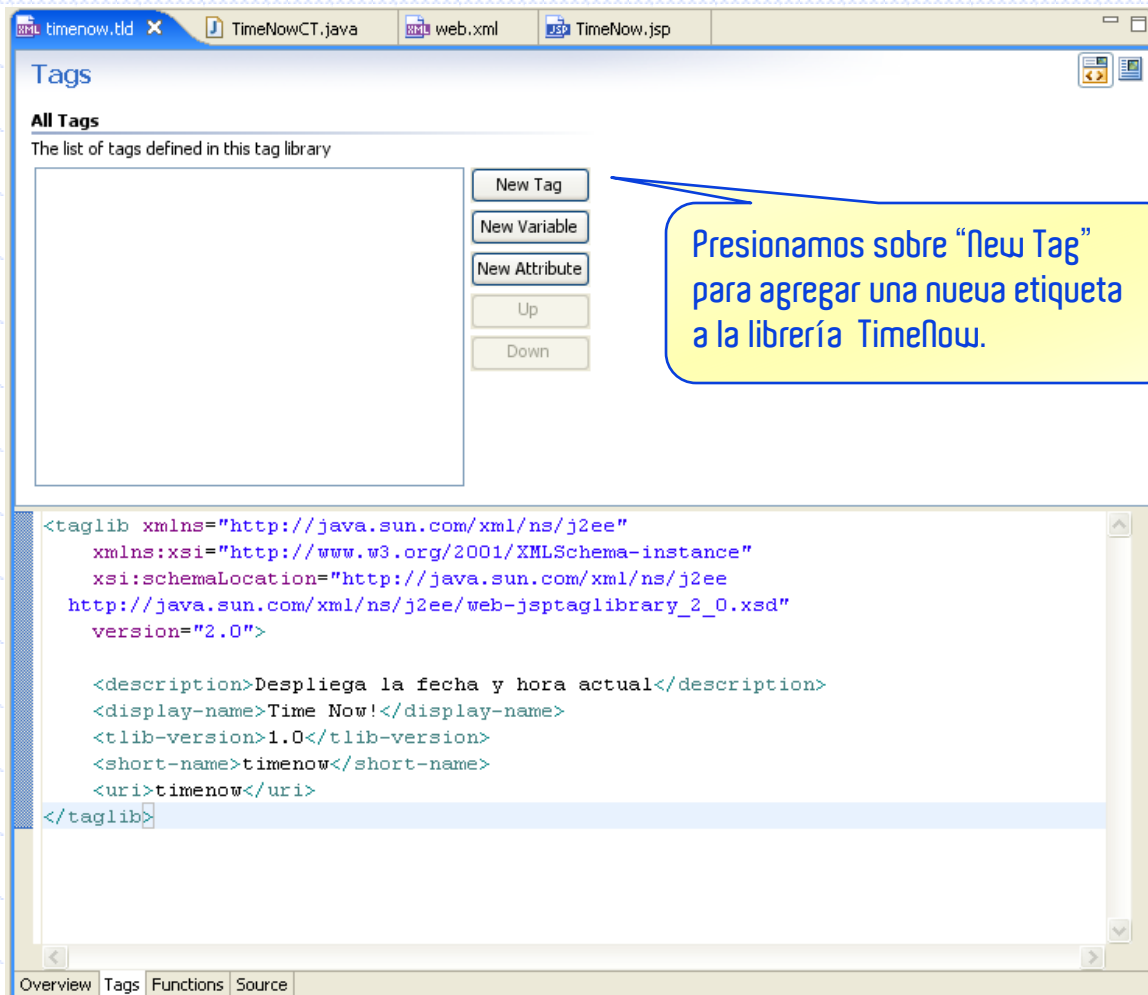
Paso 1: creación del taglib con el asistente (3/3)



Presionamos
sobre "Tags"
para ver la lista
de tags y agregar
uno.

TagLib simple: Time now!

Paso 1: creación del tag con el asistente (1/3)



TagLib simple: Time now!

Paso 1: creación del tag con el asistente (2/3)

Nombre de la clase que implementa la etiqueta.

Meramente documental

Ejemplo de uso.

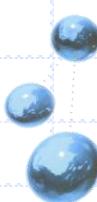
The 'Tag Tag' dialog box is titled 'Tag Tag' and contains a 'Tag Properties' section. It has several input fields and buttons:

- Name:** A text field containing 'now'.
- Tag Class:** A text field containing 'cl.uchile.cc68j.taglibs.TimeNowCT'.
- Tei Class:** An empty text field.
- Body Content:** A dropdown menu with 'empty' selected.
- Display Name:** A text field containing 'Now!'.
- Description:** A text area containing 'Despliega la fecha y hora actual'.
- Example:** A text area containing '<tn:now />|'.

At the bottom of the dialog are three buttons: 'Help', 'OK', and 'Cancel'.

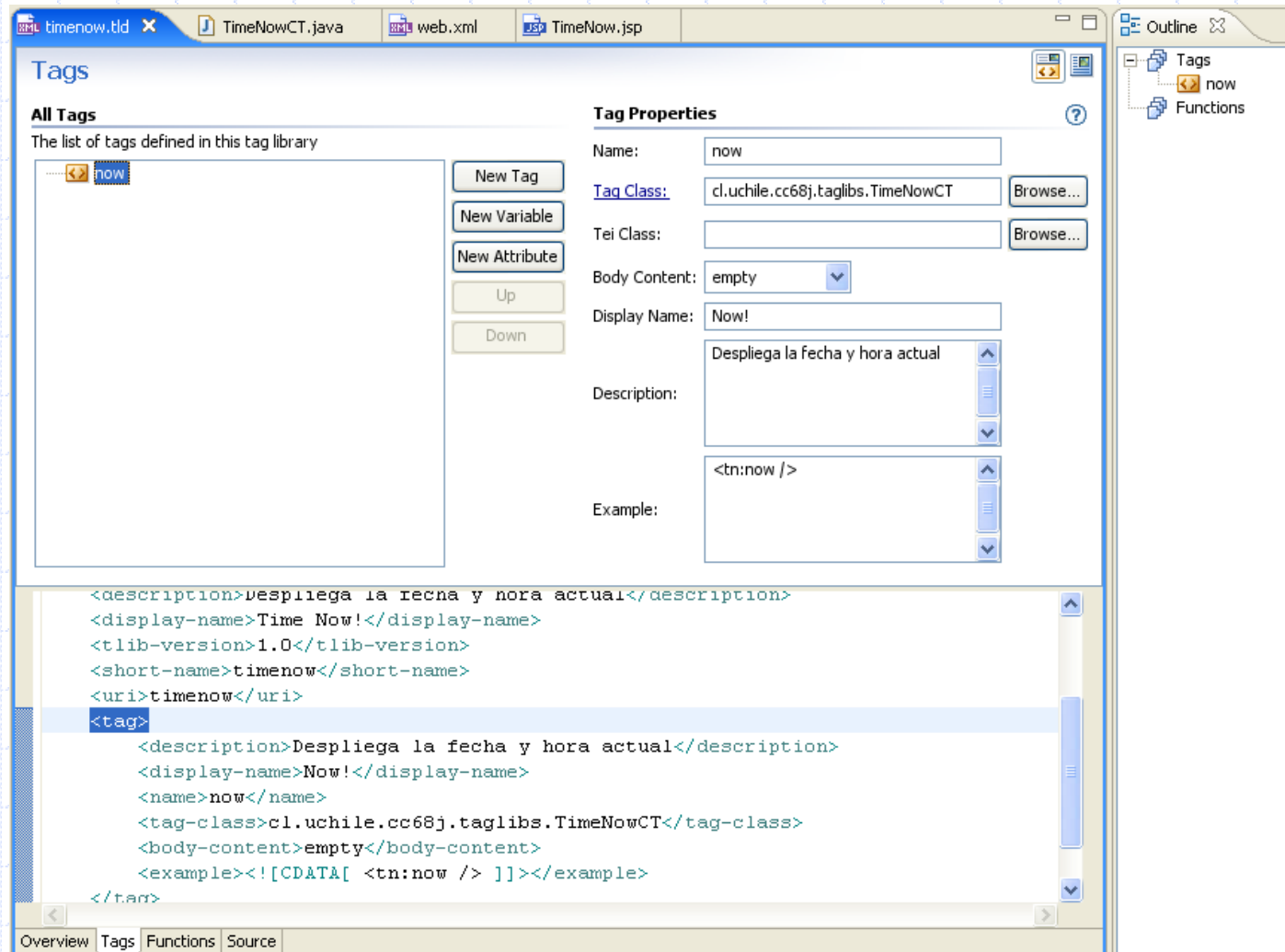
Nombre de la etiqueta personalizada.

Tipo de contenido considerado en el tag, en este caso ninguno..



TagLib simple: Time now!

Paso 1: creación del tag con el asistente (3/3)



TagLib simple: Time now!

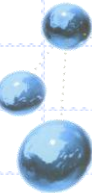
Paso 2: creación de la clase del taglib

```
package cl.uchile.cc68j.taglibs;
import java.util.Date;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.TagSupport;

public class TimeNowCT extends TagSupport {

    private static final long serialVersionUID = 1L;

    @Override
    public int doEndTag() throws JspException {
        JspWriter out = pageContext.getOut();
        try {
            out.println("La hora actual es: " + (new Date()));
        } catch (Exception e) {
            System.out.println("Problemas con el IO.");
        }
        return super.doEndTag();
    }
}
```



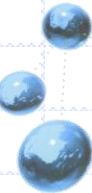
TagLib simple: Time now!

Paso 3: modificar web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <display-name>TagLibs</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>

  ...
  <jsp-config>
    <taglib>
      <taglib-uri>/tags/timenow</taglib-uri>
      <taglib-location>/WEB-INF/tlds/timenow.tld</taglib-location>
    </taglib>
  </jsp-config>
</web-app>
```

La declaración se realiza dentro del elemento `<jsp-config>` y con el elemento `<taglib>`.



TagLib simple: Time now!

Paso 3: modificar web.xml

Web Application Deployment Descriptor

Deployment Descriptor Elements
Deployment descriptor elements defined in this web.xml

- Context Params
- Error Pages
- Filters
- Listeners
- Mime Mappings
- Servlets
- Tag Libraries
 - /tags/timenow**
- JSP Property Groups
- Welcome Files

Up
Down

Tag Library Properties

Taglib Uri:

Taglib Location:

El asistente de taglibs (TLD) agrega el taglib automáticamente al web.xml.

```
<web-app>  
  <jsp-config>  
    <taglib>  
      <taglib-uri>/tags/timenow</taglib-uri>  
      <taglib-location>/WEB-INF/tlds/timenow.tld</taglib-location>  
    </taglib>  
  </jsp-config>  
</web-app>
```

Deployment Descriptor Source

TagLib simple: Time now!

Paso 4: crear la JSP

Declaración del TagLib a utilizar y el prefijo con que se utiliza.

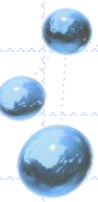
```
<%@ taglib uri="/tags/timenow" prefix="timenow" %>
<html>
  <head><title>Time Now!</title></head>
  <body>
    La fecha actual es: <timenow:now />
  </body>
</html>
```

Mismo prefijo que el declarado.

El tag siendo utilizado en la página JSP.

- Se puede eliminar el rol de **web.xml** escribiendo el nombre y localización del TLD directamente en la página JSP → tenemos que modificar todos los JSP si cambiamos el nombre del TLD

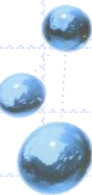
```
<%@ taglib uri="/WEB-INF/taglibs/timenow.tld"
  prefix="timenow"%>
<timenow:now/>
```



Descriptor de la Librería de Etiquetas

Tag Library Descriptor

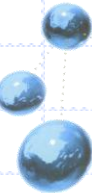
- Fichero XML que define una librería de tags y sus etiquetas.
- Consta de:
 - ✓ Prólogo XML como todo documento XML
 - ✓ El elemento raíz **<taglib>**, que tiene como sub-elementos:
 - ◆ **tlib-version** → versión de la librería de etiquetas
 - ◆ **jsp-version** → la versión de JSP, actualmente 2.0
 - ◆ **shortname** → nombre corto para la librería
 - ◆ **description** → información para documentación
 - ◆ **uri** → enlace a más información sobre tag library
 - ◆ **tag** → elemento más importante, puede haber varios y tiene sub-elementos



Descriptor de la Librería de Etiquetas

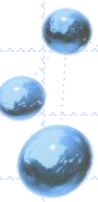
Elemento <tag>

- Puede tener los siguientes sub-elementos (**name** and **tag-class** son los únicos obligatorios):
 - ✓ **name** → identificador de la etiqueta
 - ✓ **tag-class** → nombre de la clase completo que realiza el procesamiento de esta etiqueta
 - ✓ **tei-class** → clase de ayuda de esta etiqueta
 - ✓ **body-content** → el tipo de cuerpo de una etiqueta:
 - ◆ **empty** → no hay cuerpo
 - ◆ **JSP** → cuerpo es un conjunto de elementos
 - ◆ **tagdependent** → cuerpo debe ser interpretado por la etiqueta
 - ✓ **description**
 - ✓ **attribute**: cero o más atributos que puede tener tres subelementos:
 - ◆ **name** (obligatorio)
 - ◆ **required**: **true** o **false** (valor por defecto false)
 - ◆ **rtexprvalue**: determina si el valor de este atributo se puede determinar en tiempo de ejecución.



Uso de TagLibs en JSP

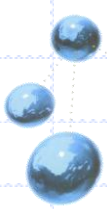
- La directiva **taglib** en un JSP tiene el siguiente formato:
 - ✓ `<%@taglib uri="tagLibraryURI" prefix="tagPrefix" %>`
 - ✓ **uri** especifica una dirección absoluta o relativa que identifica el descriptor de la librería de etiquetas asociado con el prefijo
 - ✓ **prefix**: atributo que se precederá a las acciones personalizadas
- Después de usar la directiva **taglib** es posible usar una etiqueta personalizada con los siguientes formatos:
 - ✓ `<prefix:tagName/>`
 - ✓ `<prefix:tagName>body</prefix:tagName>`
 - ✓ Para pasar atributos al manejador de la etiqueta hay que usar: `attributeName="attributeNameValue"`
- Ejemplo:
 - ✓ `<math:power number="12" power="13"/>`





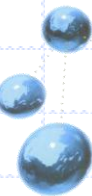
Un administrador de las etiquetas personalizadas

TAG HANDLERS



La API de Etiquetas Personalizadas

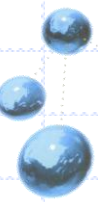
- Paquete `javax.servlet.jsp.tagext`
- Un manejador de etiquetas (**Tag Handler**) es una clase que está ligada a una etiqueta personalizada y es invocada cada vez que el contenedor de JSPs encuentra la etiqueta.
- En JSP 1.2, la clase `javax.servlet.jsp.tagext` tiene 4 interfaces y 12 clases
 - ✓ Los dos interfaces más importantes son **Tag** y **BodyTag**
 - ✓ Estos interfaces dictan el ciclo de vida de un manejador de etiquetas



Manejadores de etiquetas

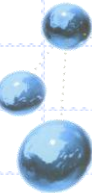
Tag Handlers

- Un manejador de etiquetas tiene acceso a un API a través del cual se puede comunicar con una página JSP.
- El punto de entrada a la API es el objeto [`javax.servlet.jsp.PageContext`](#), a través del cual el **TagHandler** puede recuperar todos los otros objetos implícitos (**request**, **session**, y **application**) accesibles desde una página JSP.
- Los objetos implícitos pueden tener atributos con nombre asociados con ellos. Tales atributos son accesibles usando métodos **[set|get]Attribute**.
- Si la etiqueta está anidada, un **Tag Handler** también tiene acceso al **Tag Handler** (llamado *parent*) de la etiqueta padre.
- Un conjunto de clases **Tag Handler** relacionadas constituyen una tag library y son empaquetadas como un fichero JAR.



El interfaz Tag

- Este interfaz define los siguientes métodos:
 - ✓ doStartTag
 - ✓ doEndTag
 - ✓ getParent
 - ✓ setParent
 - ✓ setPageContext
 - ✓ release

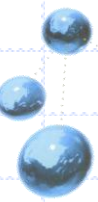


Tag Handlers

Ciclo de vida de un Tag Handler

(1/2)

- El contenedor JSP llama a los métodos de un handler en un orden específico.
- 1. Obtiene una instancia del pool de **TagHandlers** o crea uno nuevo. Después llama a **setPageContext** que representa a la página donde se encontró la custom tag.
✓ `public void setPageContext(PageContext pageContext).`
- 2. El contenedor de JSP llama a **setParent**, pasándole el objeto **Tag** que representa a la etiqueta padre de la etiqueta procesada
✓ `public void setParent(Tag parent).`
- 3. El contenedor de JSP asigna todos los atributos de la etiqueta, usando métodos `get` y `set` como en JavaBeans. Si se encuentra un atributo **temperatura** llamará a **setTemperatura**.

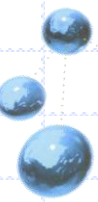


Tag Handlers

Ciclo de vida de un Tag Handler

(2/2)

4. El contenedor llama a **doStartTag**, que puede devolver **Tag.SKIP_BODY** (el contenedor no debe procesar el cuerpo de la etiqueta) o **Tag.EVAL_BODY_INCLUDE** (el contenido del cuerpo deberá ser procesado).
✓ `public int doStartTag() throws JspException`
5. El El contenedor llama a **doEndTag**, que devuelve bien **Tag.SKIP_PAGE** (no procesar el resto del JSP) o **Tag.EVAL_PAGE** (procesar el resto del JSP)
✓ `public int doEndTag throws JspException`
6. Finalmente, el contenedor de JSPs llama al método **release()**, donde se pueden liberar recurso (cerrar conexiones)
✓ `public void release()`
7. El contenedor devuelve la instancia del manejador de etiquetas al pool para uso futuro.

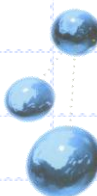
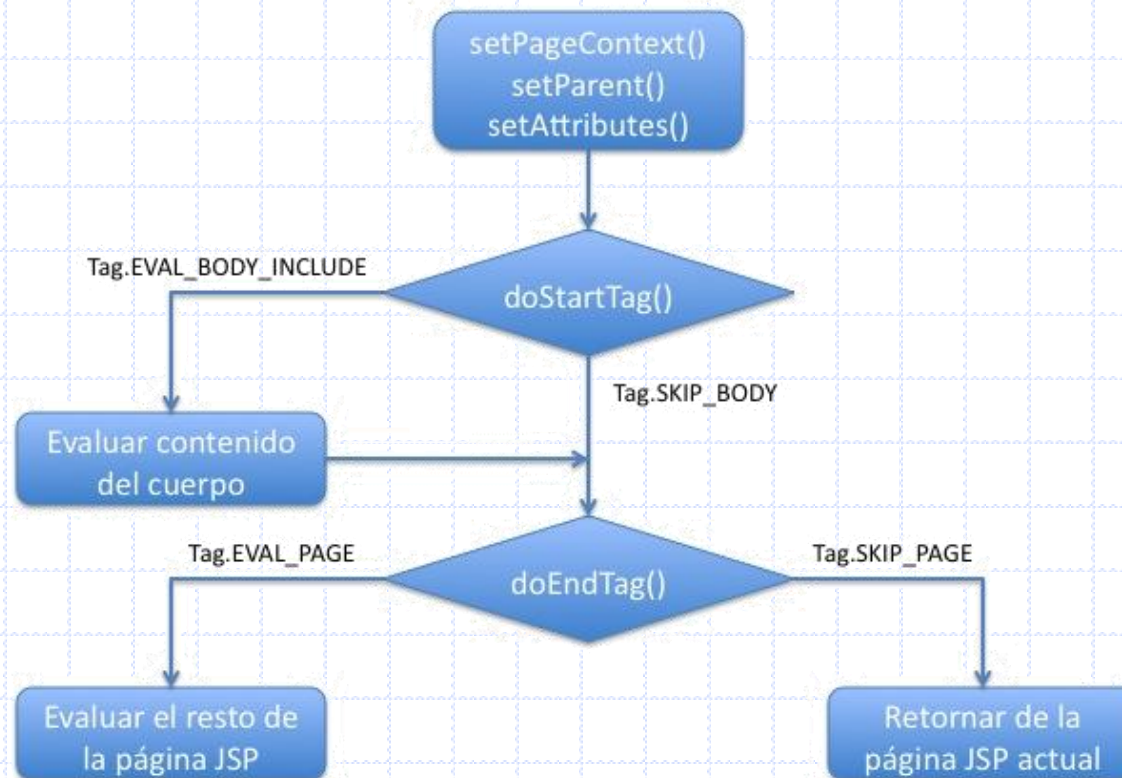


Tag Handlers

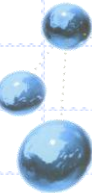
Ciclo de vida de un Tag Handler

(3/2)

Flujo interfaz javax.servlet.jsp.tagext.Tag



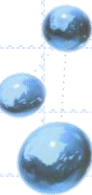
ATRIBUTOS DE UN TAG



Atributos de un TagLib

Aprendiendo a través de un ejemplo

- Objetivo:
 - ✓ Etiqueta personalizada para calcular el doble de un número
- El manejador debe definir un método **setNumber** para asignar un atributo al valor.



Atributos de un TagLib

La clase del Taglib

```
package cl.uchile.cc68j.taglibs;

import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class DoublerTag implements Tag{

    private int number;
    private PageContext pageContext;

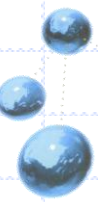
    public void setNumber(int number) {
        this.number = number;
    }
    public void setParent(Tag t) {
    }
    public void
    setPageContext(PageContext p) {
        pageContext = p;
    }

    public Tag getParent() {
        return null;
    }
}
```

```
public int doStartTag() {
    try {
        JspWriter out =
        pageContext.getOut();
        out.println("El Doble de " +
        number + " es " + (2 * number));
    }
    catch (Exception e) {
        System.out.println("ERROR");
    }
    return EVAL_BODY_INCLUDE;
}

public int doEndTag() throws
JspException {
    return EVAL_PAGE;
}

public void release() {
}
```



Atributos de un TagLib

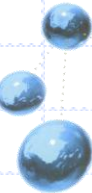
La JSP y el TLD

- La página JSP que llama a DoublerTag es:

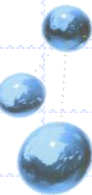
```
<%@ taglib uri="/WEB-INF/taglib.tld" prefix="miTag"%>  
<miTag:doubleTag number="12"/>
```

- El fichero TLD es:

```
<taglib>  
  <tlib-version>1.0</tlib-version>  
  <jsp-version>2.0</jsp-version>  
  <short-name/>  
  <tag>  
    <name>doubleTag</name>  
    <tag-class>cl.uchile.teaching.ej2.DoublerTag</tag-class>  
    <attribute>  
      <name>number</name>  
      <required>true</required>  
    </attribute>  
  </tag>  
</taglib>
```



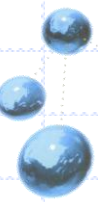
ITERACIÓN EN TAGS



Iteraciones en Tags

El interfaz IterationTag

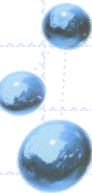
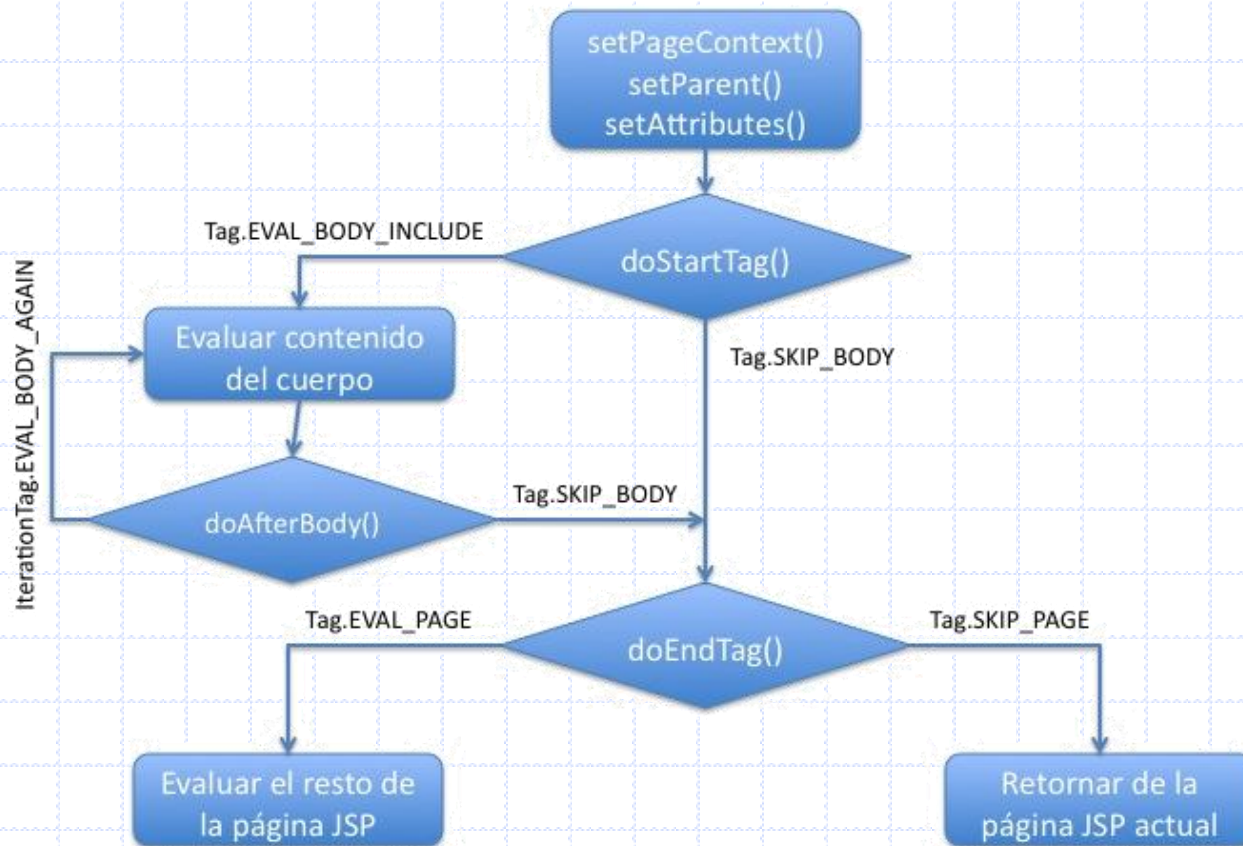
- Extiende la interfaz **Tag** añadiendo el método **doAfterBody**, que puede retornar los siguientes valores:
 - ✓ **Tag.SKIP_BODY**: el cuerpo se ignora y el contenedor llama al método **doEndTag()**.
 - ✓ **IterationTag.EVAL_BODY_AGAIN**: el método **doAfterBody** es llamado de nuevo, permitiendo implementar la iteración del tag.
- Mantiene el mismo ciclo de vida del Tag Handler, salvo por el ciclo introducido por el método **doAfterBody()**.



Iteraciones en Tags

Ciclo de vida del IterationTag

Flujo interfaz javax.servlet.jsp.tagext.IterationTag



Ejemplo de una Iteración en un Tag

La potencia de un número

(1/3)

```
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class PowerTag implements IterationTag {

    private PageContext pageContext;
    private int number;
    private int power;
    private int counter = 0;
    private int result = 1;

    public void setNumber(int number) {
        System.out.println("number: " + number);
        this.number = number;
    }

    public int getNumber() {
        return this.number;
    }
}
```

...

```
public void setPower(int power) {
    System.out.println("power: " + power);
    this.power = power;
}

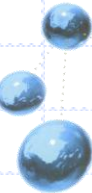
public int getPower() {
    return this.power;
}

public void setParent(Tag t) {
}

public void setPageContext(PageContext p) {
    System.out.println("setPageContext");
    pageContext = p;
}

public void release() {
    System.out.println("doAfterBody");
}

public Tag getParent() {
    return null;
}
```



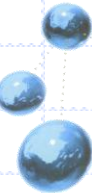
Ejemplo de una Iteración en un Tag

La potencia de un número

(2/3)

```
public int doStartTag() {  
  
    System.out.println("doStartTag");  
    return EVAL_BODY_INCLUDE;  
}  
  
public int doAfterBody() {  
  
    System.out.println("doAfterBody")  
    ;  
  
    counter++;  
    result *= number;  
    if (counter >= power)  
        return SKIP_BODY;  
    else  
        return EVAL_BODY_AGAIN;  
}  
}
```

```
public int doEndTag() throws  
    JspException {  
  
    System.out.println("doEndTag");  
    try {  
        JspWriter out =  
            pageContext.getOut();  
        out.println(number + "^"  
            + power + "=" + result);  
        this.counter = 0;  
        this.result = 1;  
    }  
    catch (Exception e) {  
    }  
    return EVAL_PAGE;  
}  
}
```



Ejemplo de una Iteración en un Tag

La potencia de un número

(3/3)

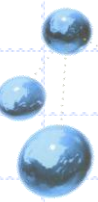
JSP para evaluar código

```
<%@ taglib uri="/myTLD"
    prefix="easy"%>
<easy:myTag number="2"
    power="3">.</easy:myTag>
```

El elemento `<rtexprvalue>` indica si el atributo puede contener scriptlets, siendo evaluado dinámicamente (valor true). El valor por defecto es false.

Archivo TLD

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD
    JSP Tag Library 1.2//EN"
    "http://java.sun.com/dtd/web-
    jsptaglibrary_1_2.dtd">
<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>myTag</short-name>
  <tag>
    <name>myTag</name>
    <tag-
      class>es.deusto.customtags.PowerTag</tag-class>
    <body-content>tagdependent</body-content>
    <attribute>
      <name>number</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
    <attribute>
      <name>power</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
  </tag>
</taglib>
```





Manipulando el contenido de una etiqueta personalizada

CONTENIDO DE UN TAGLIB



Contenido de un TagLib

Clases involucradas

- Una etiqueta personalizada puede tener un cuerpo:

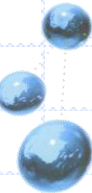
```
<%@ taglib uri="/myTLD" prefix="x"%>
```

```
<x:theTag>
```

```
    This is the body content
```

```
</x:theTag>
```

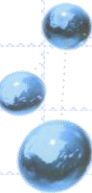
- Para manipular el cuerpo de una etiqueta es necesario utilizar las clases
 - ✓ BodyTag y
 - ✓ BodyContent



Contenido de un TagLib

La clase BodyTag

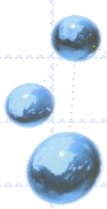
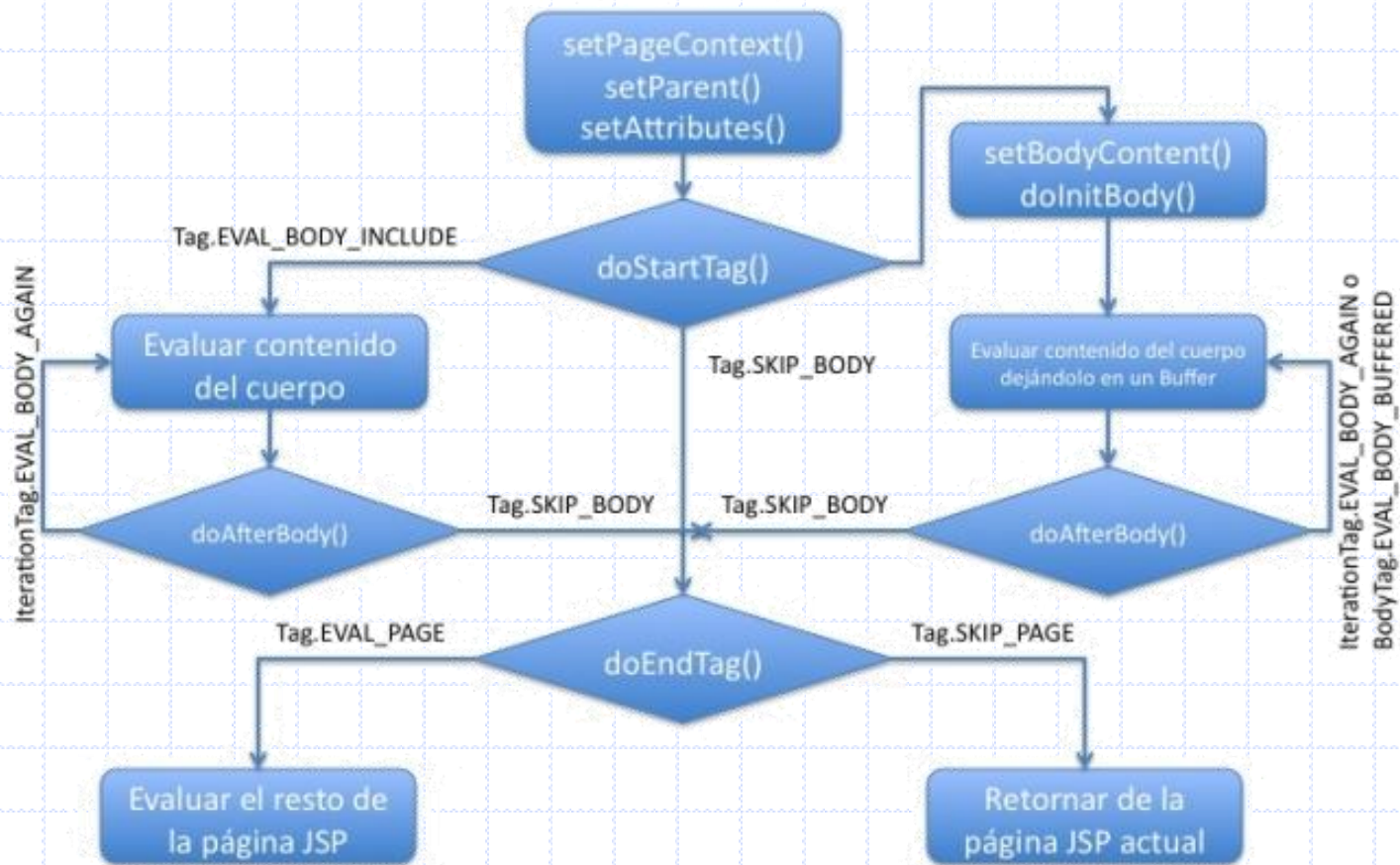
- Extiende la clase **IterationTag** con 2 métodos.
- Tiene un ciclo de vida similar a **IterationTag**, sin embargo:
 - ✓ **doStartTag** puede devolver **SKIP_BODY**, **EVAL_BODY_INCLUDE** (el cuerpo se evalúa como en **IterationTag**) o **EVAL_BODY_BUFFERED** (un objeto de tipo **BodyContent** es creado al cuerpo de la etiqueta personalizada).
- Los métodos:
 - ✓ **public void setBodyContent(BodyContent bodyContent)**
 - ◆ Llamado después de **doStartTag**, seguido de **doInitBody**. No se invoca si:
 - La custom tag no tiene cuerpo
 - La custom tag tiene cuerpo pero **doStartTag** devuelve **SKIP_BODY** o **EVAL_BODY_INCLUDE**
 - ✓ **public void doInitBody() throws java.servlet.jsp.JspException**
 - ◆ No se invoca si se cumple alguna de las mismas condiciones que para **setBodyContent**



Iteraciones en Tags

Ciclo de vida del IterationTag

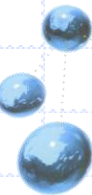
Flujo interfaz javax.servlet.jsp.tagext.BodyTag



Contenido de un TagLib

La clase BodyContent

- Representa el cuerpo de una etiqueta personalizada
- Con este objeto se puede reemplazar el contenido del elemento.
- Ejercicios de ejemplo:
 - ✓ Codificar el contenido HTML de una etiqueta personalizada y visualizar su contenido en el navegador.
 - ✓ Capitalizar la primera letra de cada palabra del contenido del tag.



Ejemplo BodyContent

Interpretación de un texto HTML

(1/3)

```
public class EncoderTag implements
```

```
BodyTag {
```

```
    PageContext pageContext;
```

```
    BodyContent bodyContent;
```

```
    /* Encode an HTML tag so it will be
    displayed as it is on the browser.
    Particularly, this method searches
    the passed in String and replace
    every occurrence of the following
    character:
```

```
    * '<' with "&lt;";
```

```
    * '>' with "&gt;";
```

```
    * '&' with "&amp;";
```

```
    * '/' with "&quot;";
```

```
    * ' ' with "&nbsp;"; */
```

```
private String encodeHtmlTag(String tag) {
```

```
    if (tag == null) return null;
```

```
    int length = tag.length();
```

```
    StringBuffer encodedTag = new
    StringBuffer(2 * length);
```

```
    for (int i = 0; i < length; i++) {
```

```
        char c = tag.charAt(i);
```

```
        if (c == '<')
```

```
            encodedTag.append("&lt;");
```

```
        else if (c == '>')
```

```
            encodedTag.append("&gt;");
```

```
        else if (c == '&')
```

```
            encodedTag.append("&amp;");
```

```
        else if (c == '"')
```

```
            encodedTag.append("&quot;");
```

```
        else if (c == ' ')
```

```
            encodedTag.append("&nbsp;");
```

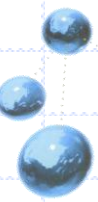
```
        else
```

```
            encodedTag.append(c);
```

```
    }
```

```
    return encodedTag.toString();
```

```
}
```



Ejemplo BodyContent

Interpretación de un texto HTML

(2/3)

```
public void setPageContext(PageContext p) {
    pageContext = p;
}

public int doStartTag() {
    return EVAL_BODY_BUFFERED;
}

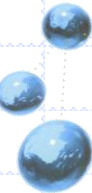
public void setBodyContent(BodyContent
bodyContent) {
    this.bodyContent = bodyContent;
}

public void doInitBody() {
}

public int doEndTag() throws JspException {
    return EVAL_PAGE;
}
```

```
public int doAfterBody() {
    String content = bodyContent.getString();
    try {
        JspWriter out =
            bodyContent.getEnclosingWriter();
        out.print(encodeHtmlTag(content));
    }
    catch (Exception e) {}

    return SKIP_BODY;
}
```



Ejemplo BodyContent

Interpretación de un texto HTML

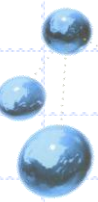
(3/3)

- Ejemplo JSP que usa **EncoderTag**:

```
<%@ taglib uri="/myTLD" prefix="easy"%>
<strings:uppercase>Un simple ejemplo.</ strings:uppercase>
```

- TLD file:

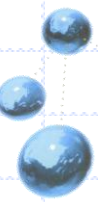
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
    "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">
<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name/>
  <tag>
    <name>encoderTag</name>
    <tag-class>cl.uchile.cc68j.taglibs.bodycontent.EncoderTag</tag-class>
    <body-content>tagdependent</body-content>
  </tag>
</taglib>
```



Clases de ayuda

Helper Classes

- Se denominan **Helper classes** a aquellas clases que son provistas por una implementación dada y que implementan las interfaces **Tag**, **IterationTag** y **BodyTag**:
 - ✓ `public class TagSupport implements IterationTag, Serializable`
 - ✓ `public class BodyTagSupport extends TagSupport implements Bodytag`
- Utilizar estas clases permite preocuparse por el código del tag y no re-escribir los métodos que poseen una implementación típica (`setPageContext`, `setBodyContent`, etc.). Ahora sólo es necesario sobre-escribir los métodos que quieran utilizarse en el procesamiento de librerías de tags personalizadas.

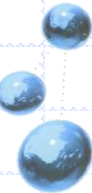


Ejemplo BodyTagSupport

Capitalización

(1/2)

```
public class CapitalizerTag extends BodyTagSupport {  
  
    public int doAfterBody() {  
        String content = bodyContent.getString();  
        try{  
            JspWriter out = bodyContent.getEnclosingWriter();  
            out.print(content.toUpperCase());  
        }  
        catch(Exception e) { e.printStackTrace(); }  
        return SKIP_BODY;  
    }  
}
```



Ejemplo BodyTagSupport

Capitalización

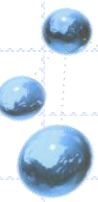
(2/2)

- JSP que utiliza CapitalizerTag:

```
<%@ taglib uri="/myTLD" prefix="easy"%>  
<strings:capitalizer>este es un buen ejemplo</strings:capitalizer>
```

- Fichero TLD:

```
<?xml version="1.0" encoding="ISO-8859-1"?>  
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"  
    "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">  
<taglib>  
  <tlib-version>1.0</tlib-version>  
  <jsp-version>1.2</jsp-version>  
  <short-name/>  
  <tag>  
    <name>capitalizer</name>  
    <tag-class>cl.uchile.cc68j.taglibs.bodycontent.CapitalizerTag</tag-class>  
    <body-content>tagdependent</body-content>  
  </tag>  
</taglib>
```

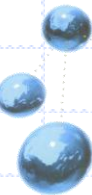


Etiquetas anidadas

- Cuando dos o más etiquetas personalizadas están anidadas es posible obtener una referencia a la clase padre a través de **findAncestorWithClass**

- Ejemplo:

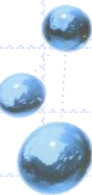
```
OuterTag1 parent1 = (OuterTag1)  
    findAncestorWithClass(this, OuterTag.class);
```





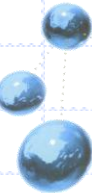
Clases de ayuda para Tag Libs

TAG EXTRA INFO



Variables de script

- La clase **Tag Extra Info** (TEI) se usa para permitir la creación de variables de script.
- Hay que definir en la clase **TagExtraInfo** el método **getVariableInfo()**.
- Este método crea un arreglo de objetos **VariableInfo**
 - ✓ Se creará uno de estos objetos por cada variable a definir, especificándose:
 - ◆ Nombre variable
 - ◆ Clase de la variable
 - ◆ Boolean indicando si habrá que crear una nueva variable
 - ◆ Scope de la variable:
 - **AT_BEGIN**: variable disponible en interior etiqueta y el resto del JSP
 - **NESTED**: variable disponible en el interior de la etiqueta
 - **AT_END**: variable disponible en el resto del JSP



Ejemplo de iterador

La clase del TagLib

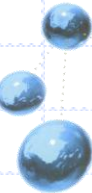
```
package cl.uchile.cc68j.tags;

import java.util.Collection;
import java.util.Iterator;

import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.BodyTagSupport;

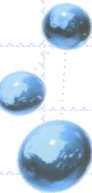
public class IteratorTag
    extends BodyTagSupport {

    private Collection collection;
    private Iterator iterator;
    private PageContext pageContext;
```



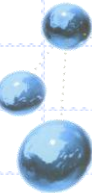
Ejemplo: definir un iterador II

```
public void setCollection(Collection collection) {  
    this.collection = collection;  
}  
  
public int doStartTag() throws JspException {  
    return collection.size() > 0 ? EVAL_BODY_BUFFERED : SKIP_BODY;  
}  
  
public void doInitBody() throws JspException {  
    iterator = collection.iterator();  
    this.pageContext.setAttribute("item", iterator.next());  
}  
...
```



Ejemplo: definir un iterador III

```
public int doAfterBody() throws JspException {
    if (iterator == null) {
        iterator = collection.iterator();
    }
    if (iterator.hasNext()) {
        this.pageContext.setAttribute("item", iterator.next());
        return EVAL_BODY_AGAIN;
    }
    else {
        try {
            getBodyContent().writeOut(getPreviousOut());
        }
        catch (java.io.IOException e) {
            throw new JspException(e.getMessage());
        }
        return SKIP_BODY;
    }
}
```

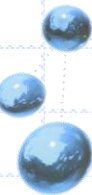


Ejemplo: definir un iterador IV

- Definir un objeto TagExtraInfo

```
import java.util.Collection;
import java.util.Iterator;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class IteratorTagInfo extends TagExtraInfo {
    public VariableInfo[] getVariableInfo(TagData data) {
        return new VariableInfo[]
        {
            new VariableInfo("item",
                            "java.lang.Object",
                            true,
                            VariableInfo.AT_BEGIN)
        };
    }
}
```



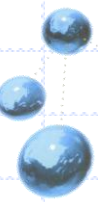
Ejemplo: definir un iterador V

```
<taglib>
  <tlib-version>1.0</tlib-version><jsp-version>1.2</jsp-version>
  <tag>
    <name>iterate</name>
    <tag-class>es.deusto.customtags.IteratorTag</tag-class>
    <tei-class>es.deusto.customtags.IteratorTagInfo</tei-class>

    <body-content>JSP</body-content>

    <variable>
      <name-given>item</name-given>
      <name-from-attribute>item</name-from-attribute>
      <variable-class>java.lang.String</variable-class>
      <declare>true</declare> <scope>AT_BEGIN</scope>
    </variable>

    <attribute>
      <name>id</name><required>true</required><rtextprvalue>true</rtextprvalue>
    </attribute>
    <attribute>
      <name>collection</name><required>true</required><rtextprvalue>true</rtextprvalue>
    </attribute>
  </tag>
</taglib>
```



Ejemplo: definir un iterador VI

```
<html><head><title>Un Iterator</title></head>
```

```
<%@ taglib uri="/myTLD" prefix="it"%>
```

```
<body>
```

```
<%
```

```
java.util.Vector vector = new java.util.Vector();
```

```
vector.addElement("one");
```

```
vector.addElement("two");
```

```
vector.addElement("three");
```

```
vector.addElement("four");
```

```
%>
```

Iterating over <%= vector %> ...

```
<p>
```

```
<it:iterate collection="<%=vector%>">
```

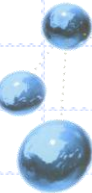
```
    Item: <%= item %><br>
```

```
</it:iterate>
```

```
</p>
```

```
</body>
```

```
</html>
```





PREGUNTAS?

