



Institut Informatique  
Claude Chappe

# Master STS Mention Informatique Parcours ISI

Programmation distribuée  
M1 / 178EN003

## C4.2 – Application Web JSF 2.2

Thierry Lemeunier  
[thierry.lemeunier@univ-lemans.fr](mailto:thierry.lemeunier@univ-lemans.fr)  
[www-lium.univ-lemans.fr/~lemeunie](http://www-lium.univ-lemans.fr/~lemeunie)

# Plan du cours

- Communication par composant distribué :
  - Application Web JSF 2.2
    - Des Servlets à JSP à JSF
    - Principes de JSF
    - Cycle de vie d'une JSF
    - Les expressions UEL
    - Les *managed beans* dans JSF
    - Composants standards JSF
    - Navigation entre pages et flots de pages JSF
    - Convertisseurs et validateurs de composants
    - Listeners de composants
    - Les fichiers ressources
    - HTML5, JavaScript et Ajax
    - Les templates et les composites
    - Les fichiers de déploiement et de configuration

# Plan du cours

- Communication par composant distribué :
  - Application Web JSF 2.2
    - Des Servlets à JSP à JSF
    - Principes de JSF
    - Cycle de vie d'une JSF
    - Les expressions UEL
    - Les *managed beans* dans JSF
    - Composants standards JSF
    - Navigation entre pages et flots de pages JSF
    - Convertisseurs et validateurs de composants
    - Listeners de composants
    - Les fichiers ressources
    - HTML5, JavaScript et Ajax
    - Les templates et les composites
    - Les fichiers de déploiement et de configuration

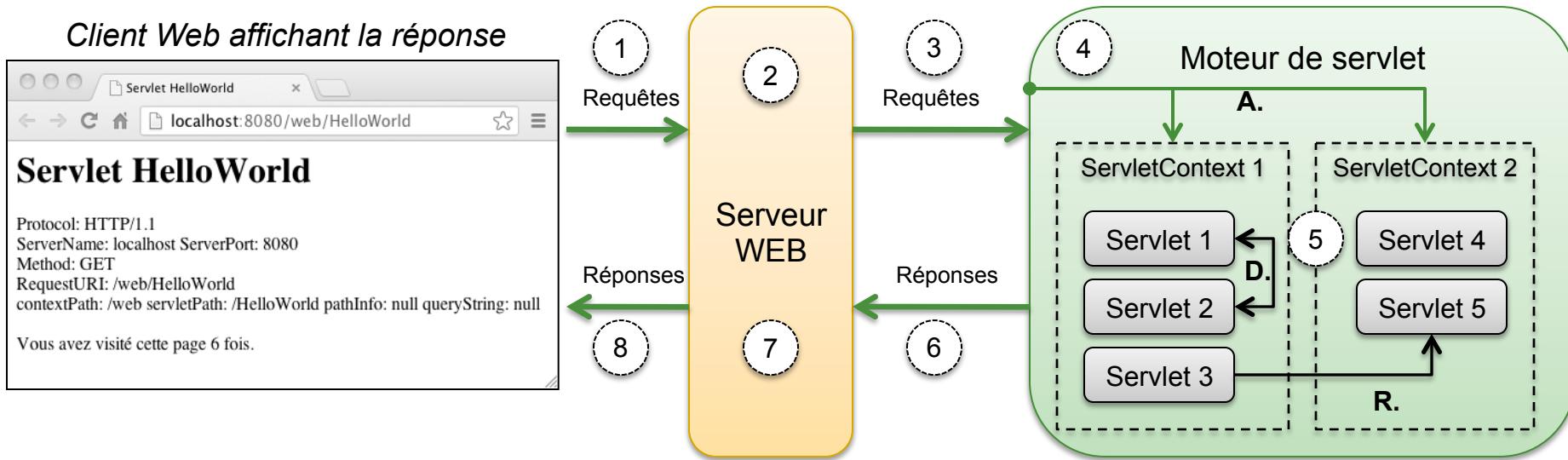
# Des Servlets à JSP à JSF (1 / 12)

## ■ Les servlets

- Principes :
  - Créer des flux HTML dynamiques (ou tout autre format géré par le client)
  - Classe Java qui étend `javax.servlet.http.HttpServlet` ou `javax.servlet.GenericServlet`
    - Une servlet a accès à toutes les API Java
    - Possibilités : partage d'information entre servlets ; filtrage des réponses et des requêtes (compression/décompression ; cryptage/décryptage ; etc.) ; ajout de listener sur des événements du cycle de vie ; exécution asynchrone ; gestion de la sécurité ; gestion des cookies, sessions...
    - Exécution dans un conteneur de servlet (lien entre la servlet et le serveur web)
    - Un conteneur de servlet est aussi appelé un moteur de servlet
  - Le conteneur de servlet gère le cycle de vie de la servlet :
    - Une servlet peut être chargée automatiquement lors du démarrage du serveur web ou lors de la première requête du client.
    - Une servlet est chargée et initialisée qu'une seule fois ; ensuite elle reste instanciée
    - Pour des serveurs web à forte audience, le conteneur gère un pool de servlets
    - Le conteneur reçoit la requête du client et sélectionne la servlet qui aura à la traiter
    - Généralement le conteneur exécute chaque requête à une servlet dans un thread séparé (une servlet doit donc être conçue pour fonctionner en environnement multithread)
    - Le conteneur de servlet open source le plus utilisé : Apache Tomcat
- Versions :
  - Servlet 3.1, Mai 2013, Java EE 7, Java SE 7
  - Servlet 2.2, Aout 1999, J2EE 1.2, J2SE 1.2
  - Servlet 1.0, Juin 1997

# Des Servlets à JSP à JSF (2/12)

## ■ Cycle de vie d'une servlet (dans le cas de HTTP) :



A. : association d'URL et de servlets  
D. : délégation d'une autre servlet  
R. : redirection vers une autre servlet

- 1 : Requête HTTP du client  
2 : Redirection vers le moteur de servlet  
3 : Réception de la requête redirigée par le serveur web  
4 : Sélection de la servlet selon l'URL (contextPath + servletPath + pathInfo)

Si aucune instance de la servlet existe, chargement puis instanciation (appel de la méthode *init*)

- 5 : Instanciation des objets *HttpServletRequest* et *HttpServletResponse* par le moteur de servlet  
Exécution de la servlet sélectionnée avec ces deux objets en argument (appel de la méthode *service*)

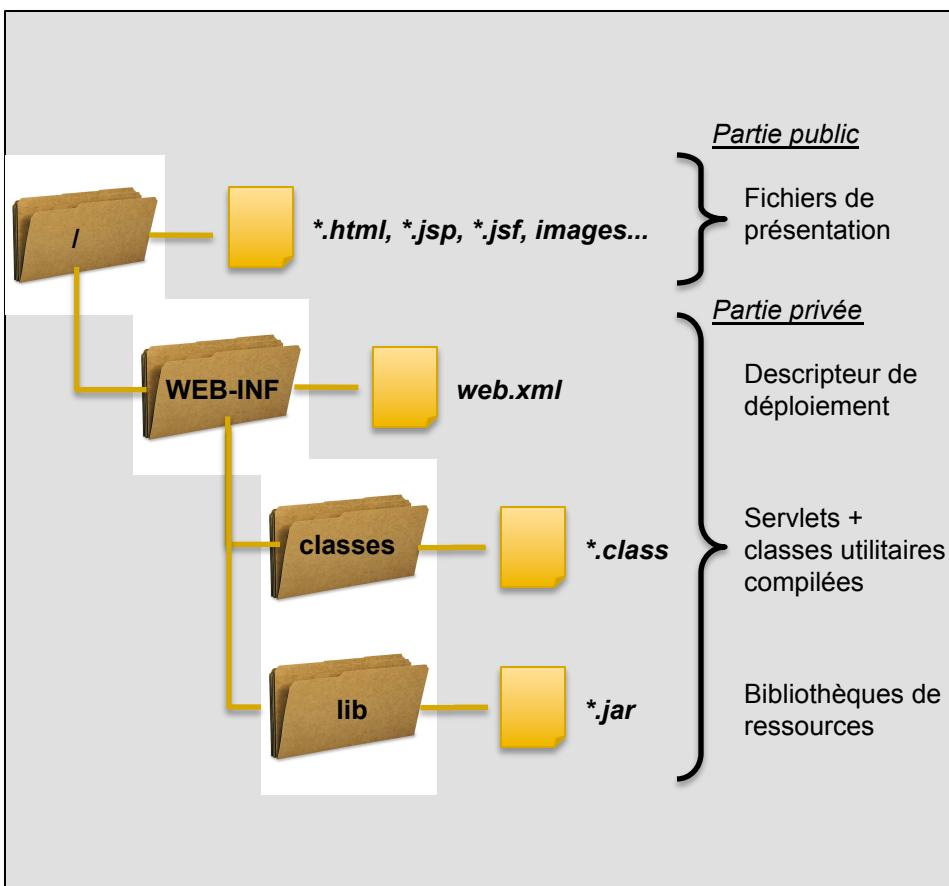
- 6 : Envoie de la réponse au serveur web  
7 : Redirection de la réponse vers le client  
8 : Réception de la réponse du serveur par le client

Quand le moteur de servlet veut arrêter une servlet, il appelle d'abord la méthode *destroy* de la servlet.

# Des Servlets à JSP à JSF (3/12)

## ■ Exemple de servlet :

Architecture d'une application Web Java EE



Fichier de déploiement : `web.xml`

```
<?xml version="1.0" encoding="UTF-8"?>  
  
<web-app version="3.0"  
    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-app_3_0.xsd">  
  
    <display-name>Test d'une servlet Java</display-name>  
  
    <servlet>  
        <servlet-name>HelloWorld</servlet-name>  
        <servlet-class>testservlet.HelloWorld</servlet-class>  
    </servlet>  
  
    <servlet-mapping>  
        <servlet-name>HelloWorld</servlet-name>  
        <url-pattern>/HelloWorld</url-pattern>  
    </servlet-mapping>  
  
    <session-config>  
        <session-timeout>30</session-timeout>  
    </session-config>  
  
</web-app>
```

Annotations explain specific elements of the `web.xml` file:

- Fichier XML**: Points to the XML declaration at the top.
- Version + Espaces de nom**: Points to the XML version and namespace declarations.
- Nom de l'application**: Points to the `display-name` element.
- Déclaration d'une servlet**: Points to the `servlet` and `servlet-mapping` declarations.
- ServletPath**: Points to the `url-pattern` element.
- Durée de validité d'une session**: Points to the `session-timeout` element.

# Des Servlets à JSP à JSF (4/12)

```
Servlet HelloWorld hérite de HttpServlet  
import ...  
  
public class HelloWorld extends HttpServlet {  
  
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
  
        response.setContentType("text/html;charset=UTF-8");  
  
        PrintWriter out = response.getWriter();  
        try {  
            out.println("<html>");  
            out.println("<head><title>Servlet HelloWorld</title></head>");  
            out.println("<body>");  
            out.println("<h1>Servlet HelloWorld" + "</h1>");  
            ...  
            out.println("RequestURI: " + request.getRequestURI() + "<br/>");  
            out.print(" contextPath: " + request.getContextPath());  
            out.print(" servletPath: " + request.getServletPath());  
            out.print(" pathInfo: " + request.getPathInfo());  
            out.println(" queryString: " + request.getQueryString());  
  
            HttpSession session = request.getSession();  
            Integer count = (Integer) session.getAttribute("count");  
            if (count == null) count = new Integer(1); else count = new Integer(count.intValue() + 1);  
            session.setAttribute("count", count);  
            out.println("<br/><br/>Vous avez visité cette page " + count + " fois.");  
  
            out.println("</body>");  
            out.println("</html>");  
        } finally { out.close(); }  
    }  
}
```

Méthode **doGet** avec un objet requête et un objet réponse

Modification entête HTTP de la réponse

Obtention d'un flux d'écriture puis écriture de balises HTML dans le flux

Accès à la session actuelle puis création ou modification d'un attribut de session

# Des Servlets à JSP à JSF (5/12)

## ■ JavaServer Page (JSP) :

### □ Principes :

- Avoir moins de lourdeur que les servlets et être plus orienté vers l'écriture d'IHM
- Lors d'une requête sur une JSP, la page est transformée en une servlet qui est compilée à la volée puis exécutée par le moteur de servlet
- Une servlet d'une page JSP implémente l'interface `javax.servlet.jsp.HttpJspPage` ou l'interface `javax.servlet.jsp.JspPage`
- Formalisme proche du HTML avec des balises spécifiques de traitement des données
  - Page JSP = document texte (HTML, XML, etc.) + éléments JSP
  - Eléments JSP :
    - Ils sont écrits **soit** sous forme de balises (syntaxe JSP ou XML) **soit** sous forme d'expressions EL (Expression Language) d'accès en lecture aux propriétés d'objets
    - Ils sont transformés en lignes de code Java qui sont incluses dans la servlet
- Possibilité d'utiliser des classes utilitaires du type JavaBean
- Possibilité de définir ces propres balises dans des bibliothèques
- Accès à des objets gérés par le moteur : `application`, `session`, `request`, `response`, `out`, etc.
- JSTL (*JavaServer Pages Standard Tag Library*) : bibliothèques de balises standards

### □ Limites :

- JSP n'interdit pas de mélanger la partie IHM et la partie métier
- JSP est trop "HTTP" et pas assez IHM : manque d'interactivité (pas de support d'Ajax)

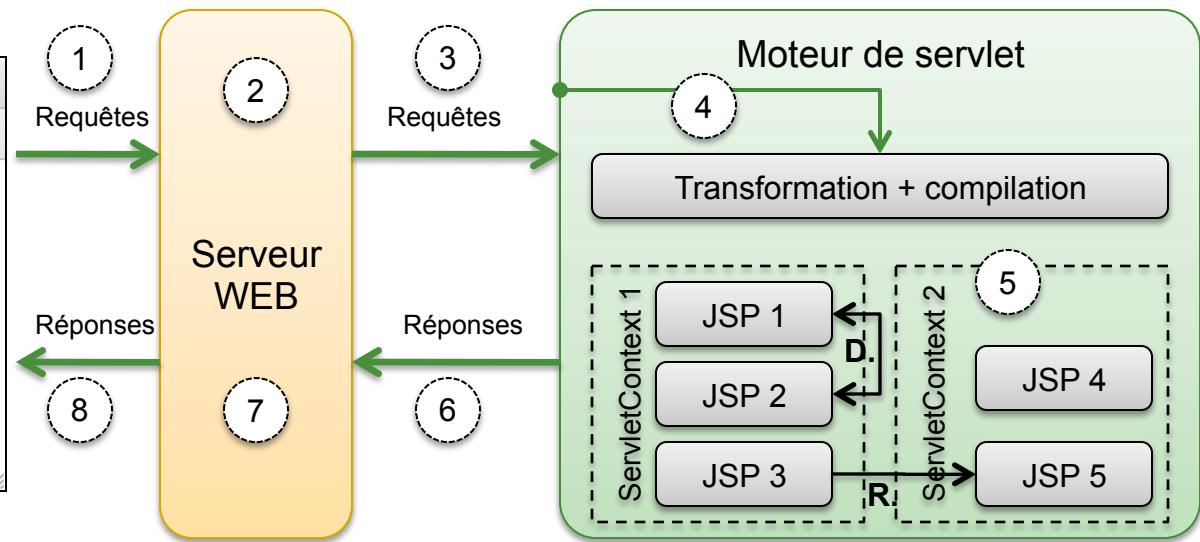
### □ Versions :

- JSP 2.3, Juin 2013, Java EE 7
- JSP 2.0, Novembre 2003, J2EE 1.4
- JSP 1.0, Décembre 1999, J2EE 1.2

# Des Servlets à JSP à JSF (6/12)

## ■ Cycle de vie d'une JSP (dans le cas de HTTP) :

*Client Web affichant la réponse*



- 1 : Requête HTTP du client
  - 2 : Redirection vers le moteur de servlet
  - 3 : Réception de la requête redirigée par le serveur web
  - 4 : Vérification que la servlet de la page JSP est plus ancienne que la page JSP ou que la servlet n'existe pas  
Si c'est le cas, transformation de la page JSP en classe servlet puis compilation de la classe  
Chargement du fichier .class puis instantiation (appel de la méthode `jspInit`)
  - 5 : Instanciation des objets `HttpServletRequest` et `HttpServletResponse` par le moteur de servlet  
Exécution de la servlet avec ces deux objets en argument (appel de la méthode `_jspService`)
  - 6 : Envoie de la réponse au serveur web
  - 7 : Redirection de la réponse vers le client
  - 8 : Réception de la réponse du serveur par le client
- Quand le moteur de servlet veut arrêter une servlet, il appelle d'abord la méthode `jspDestroy` de la servlet.

# Des Servlets à JSP à JSF (7/12) – JSPHelloWorld.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
  <head>
    <title>JSP Hello World</title>
  </head>
  <body>
    <h1>JSP Hello World</h1>

    Protocol: <%= request.getProtocol() %><br/>
    ServerName: <%= request.getServerName() %>
    ServerPort: <%= request.getServerPort() %><br/>
    Method: <%= request.getMethod() %><br/>
    RequestURI: <%= request.getRequestURI() %><br/>
    contextPath: <%= request.getContextPath() %>
    &nbsp;servletPath: <%= request.getServletPath() %>
    &nbsp;pathInfo: <%= request.getPathInfo() %>
    &nbsp;queryString: <%= request.getQueryString() %>

    <%
      Integer count = (Integer) session.getAttribute("count");
    %>

    <c:if test="${empty count}">
      <% count = new Integer(0); %>
    </c:if>

    <%
      count = new Integer(count.intValue() + 1);
      session.setAttribute("count", count);
    %>

    <br/><br/>Vous avez visité cette page ${count} fois
  </body>
</html>
```

Type de contenu et encodage de la page

Utilisation de la bibliothèque *core* de JSTL

Fichier = Balises HTML +  
Balises JSP

Utilisation de l'objet implicite *request*  
dans des balises d'expression JSP

Déclaration d'une variable *count* dans  
la méthode *\_jspService* à partir de  
l'objet implicite *session*

Balise JSTL *<c:if>* + expression EL \${...}

Modification de la variable *count*

Modification de la variable *count* et  
sauvegarde dans la session courante

Accès en lecture de la variable *count*  
dans une expression EL

# Des Servlets à JSP à JSF (8/12)

## ■ Le langage des pages JSP :

Les balises	
<u>Balise de directive</u> : contrôle de la génération de la servlet <code>&lt;%@ ... %&gt;</code>	Inclusion : <code>&lt;%@ include file="un autre fichier jsp" %&gt;</code> Bibliothèque : <code>&lt;%@ taglib prefix="c" uri="adresse" %&gt;</code>
	Page : attributs liés à la page - importation : <code>&lt;%@ page import="java.util.*" %&gt;</code> - encodage : <code>&lt;%@ page pageEncoding="UTF-8" %&gt;</code> - contenu : <code>&lt;%@ page contentType="text/plain" %&gt;</code> - décl. page d'erreur : <code>&lt;%@ page errorPage="error.jsp" %&gt;</code> - déf. page d'erreur : <code>&lt;%@ page isErrorPage=true %&gt;</code>
<u>Balise de commentaire</u> <code>&lt;%-- blabla --%&gt;</code>	<code>&lt;%-- Commentaire pour les développeurs --%&gt;</code>
<u>Balise de déclaration</u> : insérer du code dans la servlet (possibilité de redéfinir des méthodes <i>jspInit</i> et <i>jspDestroy</i> ) <code>&lt;%! ... %&gt;</code>	<code>&lt;%! private int count = 0; private int incrementCount() { return count++; } %&gt;</code>
<u>Balise de scriptlet</u> : bloc de code placé dans <i>_jspService</i> (accès attributs / méthodes déclarés et objets implicites) <code>&lt;% ... %&gt;</code>	<code>&lt;% for (int i = 0; i &lt; 5 ; i++) { %&gt; HelloWorld&lt;br/&gt; &lt;% incrementCount(); %&gt;</code>
<u>Balise d'expression</u> : évalue une expression et renvoie sa valeur sous forme de String <code>&lt;%= ... %&gt;</code>	<code>&lt;% String[] noms = {"mickey", "donald"}; for (int i = 0 ; i &lt; noms.length ; i++) { %&gt; Le &lt;%= i %&gt; ème nom est &lt;%= noms[i] %&gt; &lt;% } %&gt;</code>
<b>Les expressions EL : \${ ... }</b> Expressions écrites en EL ( <i>Expression Language</i> ) évaluées immédiatement à l'exécution de la JSP (accès en lecture uniquement)	Votre nom est \${ customer.name } Vous avez visité \${ count } fois cette page

# Des Servlets à JSP à JSF (9/12) – Aperçu de JSTL

Domaine	Espace de nom et préfixe		Exemples
Core (attributs, structures de contrôle, URL...)	http://java.sun.com/jsp/jstl/core	c	<pre>&lt;%-- Création d'un attribut de session --%&gt; &lt;c:set var="pays" value="\${param.pays}" scope="session" /&gt; &lt;%-- Suppression d'un attribut d'application --%&gt; &lt;c:remove var="color" scope="application" /&gt; &lt;%-- Table de choix --%&gt; &lt;c:choose&gt;     &lt;c:when test="\${customer.category == 'trial'}" &gt;         ...     &lt;/c:when&gt;     &lt;c:otherwise&gt;... &lt;/c:otherwise&gt; &lt;/c:choose&gt;</pre>
XML (parser, parcours, transformation XSLT...)	http://java.sun.com/jsp/jstl/xml	x	<pre>&lt;%-- Charge un fichier (nom = paramètre d'initialisation) --%&gt; &lt;c:import url="\${initParam.booksURL}" var="xml" /&gt; &lt;%-- Parse le fichier ; résultat → un attribut d'application) --%&gt; &lt;x:parse doc="\${xml}" var="booklist" scope="application" /&gt;</pre>
I18N (internationalisation)	http://java.sun.com/jsp/jstl/fmt	fmt	<pre>&lt;%-- Affichage d'un message selon la langue sélectionnée --%&gt; &lt;h3&gt;&lt;fmt:message key="Choose"/&gt;&lt;/h3&gt; &lt;%-- Formatage d'une valeur numérique selon la langue --%&gt; &lt;fmt:formatNumber value="\${book.price}" type="currency"/&gt;</pre>
Database	http://java.sun.com/jsp/jstl/sql	sql	<pre>&lt;%-- Sélection de la source de données (nom JNDI) --%&gt; &lt;sql:setDataSource dataSource="jdbc/BookDB" /&gt; &lt;%-- Requête dans la base --%&gt; &lt;c:set var="bid" value="\${param.bid}" /&gt; &lt;sql:query var="books" &gt;     select * FROM PUBLIC.books where id = ?     &lt;sql:param value="\${bid}" /&gt; &lt;/sql:query&gt;</pre>
Functions (chaînes de caractères)	http://java.sun.com/jsp/jstl/functions	fn	<pre>&lt;c:if test="\${fn:length(param.username) &gt; 0}" &gt; ... &lt;/c:if&gt;</pre>

# Des Servlets à JSP à JSF (10/12) – JavaBean

- JavaBean = classe Java publique non abstraite non finale [implémente *Serializable*] + conventions :
  - Un constructeur par défaut (sans argument)
  - Un ensemble de propriétés accessibles :
    - Soit en lecture / écriture
    - Soit en lecture seule
    - Soit en écriture seule
  - Propriété nommée *<property>* de type *<PropertyClass>*
    - Soit une seule valeur Soit des valeurs indexés (List, Array, ...)
    - Accès en lecture : *public PropertyClass getProperty() { ... }*
    - Accès en écriture : *public void setProperty(PropertyClass property) { ... }*

```
public class Product {  
    ...  
    String name;          // Nom du produit  
    String description;  // Description du produit  
    ...  
    public String getName() { return name; }  
    public void setName(String name) { this.name = name; }  
  
    public String getDescription() { return description; }  
    ...  
}
```

JavaBean Product

Propriété *name* accessible  
en lecture / écriture

Propriété *description*  
accessible  
en lecture seule

# Des Servlets à JSP à JSF (11/12) – JavaBean (suite)

## ■ Utilisation d'un JavaBean dans une page JSP

### □ Déclaration du JavaBean :

```
<jsp:useBean id="beanName" class="package_name.class_name"  
            scope="scope" />
```

Scope : durée de l'existence de l'instance

scope ::= *application* | *session* | *page* | *request*

### □ Accès aux propriétés du JavaBean :

#### ■ Accès en écriture :

```
<jsp:setProperty name="beanName" property="propName"  
                  value="string constant | expression"  
                  | param="paramName" />
```

ou

```
<% monBean.setProperty(valeur); %>
```

#### ■ Accès en lecture :

```
<jsp:getProperty name="beanName" property="propName" />
```

ou

```
 ${ beanName.propName }
```

ou

```
<%= beanName.getProperty() %>
```

# Des Servlets à JSP à JSF (12/12)

- Exemple de nécessité d'avoir une logique d'affichage directement dans la page JSP :

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
```

Déclaration de l'utilisation de la bibliothèque core de JSTL

```
<%@ page import="controller.CatalogController" %>
<%@ page import="entity.catalog.Product" %>
```

Importation des classes utiles

```
<TABLE>
  <jsp:useBean id="catalog" class="CatalogController" scope="request"/>
  <c:forEach var="product" items="${catalog.products}">
    <TR>
      <TD>
        <A href=""><c:out value="${product.name}" /></A>
        <BR><c:out value="${product.description}" />
      </TD>
    </TR>
  </c:forEach>
</TABLE>
```

Utilisation d'un JavaBean

Boucle de parcours d'une liste (propriété de *CatalogController*)

Accès aux propriétés du JavaBean Product  
Affichage des valeurs des propriétés

# Plan du cours

- Communication par composant distribué :
  - Application Web JSF 2.2
    - ✓ Des Servlets à JSP à JSF
    - Principes de JSF
    - Cycle de vie d'une JSF
    - Les expressions UEL
    - Les *managed beans* dans JSF
    - Composants standards JSF
    - Navigation entre pages et flots de pages JSF
    - Convertisseurs et validateurs de composants
    - Listeners de composants
    - Les fichiers ressources
    - HTML5, JavaScript et Ajax
    - Les templates et les composites
    - Les fichiers de déploiement et de configuration

# Principes de JSF (1/6) – Objectifs

- Intentions de JavaServer Faces (JSF) :
  - Proposer un véritable framework orienté IHM pour les applications Web
  - Améliorer la conception et l'interactivité par rapport à JSP
  - Depuis Java EE 7, JSP est considéré comme une technologie « *deprecated* »
- Contenus de JSF :
  - Framework Web MVC de composants graphiques gérés par événements
  - Page JSF = page XHTML + balises JSF (+ certaines balises JSTL)
  - Le cycle de vie d'une page JSF tient compte de listeners d'événements, de convertisseurs et de validateurs de données
  - Possibilité de définir la navigation entre pages et des flots (plusieurs pages liées)
  - Possibilité de définir des templates (structures communes réutilisables)
  - Possibilité d'utiliser des composants JavaBean
  - Possibilité d'utiliser nativement HTML5, JavaScript et Ajax
  - Possibilité de définir ses propres composants réutilisables (pas abordé ici !)
- Versions :
  - JSF 2.2, Juin 2013, Java EE 7
  - JSF 2.0, Décembre 2009, Java EE 6
  - JSF 1.1, Novembre 2003, J2EE 1.4

# Principes de JSF (2/6) – JSFHelloWorld.xhtml

```
<?xml version='1.0'encoding='UTF-8'?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:h="http://xmlns.jcp.org/jsf/html">
<h:head>
 <title>JSF Hello World</title>
</h:head>
<h:body>
 <h:outputText value="JSF Hello World" style="font-size: 130%;"/><br/><br/>
 <h:outputLabel value="Protocol: "/><h:outputText value="#{request.protocol}"/><br/>
 <h:outputLabel value="ServerName: "/><h:outputText value="#{request.serverName}"/>
 <h:outputLabel value="ServerPort: "/><h:outputText value="#{request.serverPort}"/><br/>
 <h:outputLabel value="Method: "/><h:outputText value="#{request.method}"/><br/>
 <h:outputLabel value="RequestURI: "/><h:outputText value="#{request.requestURI}"/><br/>
 <h:outputLabel value="contextPath: "/><h:outputText value="#{request.contextPath}"/>
 <h:outputLabel value=" servletPath: "/><h:outputText value="#{request.servletPath}"/>
 <h:outputLabel value=" pathInfo: "/><h:outputText value="#{request.pathInfo}"/>
 <h:outputLabel value=" queryString: "/><h:outputText value="#{request.queryString}"/><br/><br/>
 <h:outputText value="Vous avez visité cette page #{myBean.count} fois"/>
</h:body>
</html>
```

Fichier JSFHelloWorld.xhtml

Format XML

Déclaration au moteur de l'utilisation de balises

Utilisation de balises JSF

Accès à l'objet implicite *request* via des expressions EL

Affichage par le client du rendu de la vue

Référence sur une instance nommée *myBean* (nom par défaut) de la classe MyBean

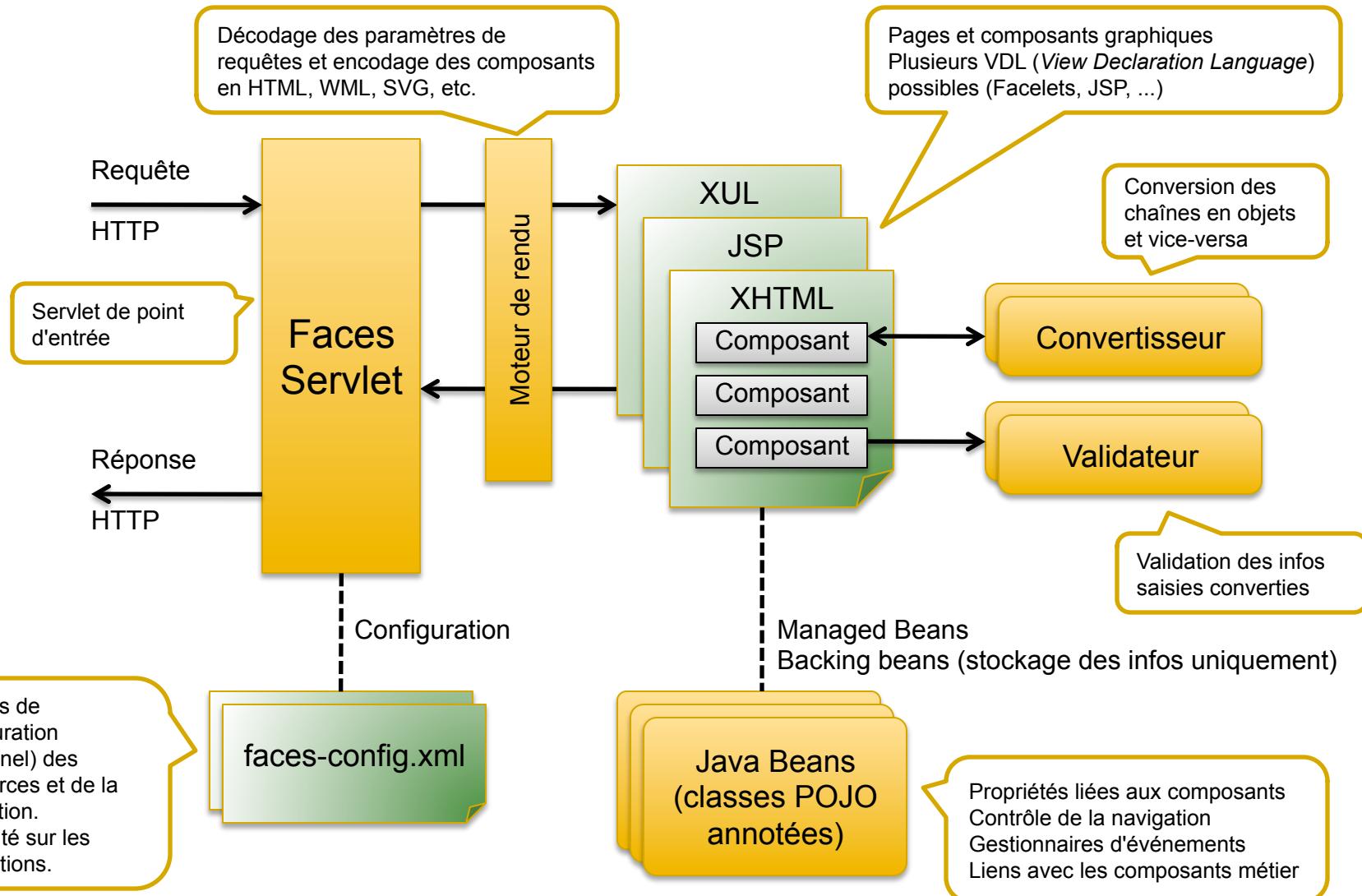
```
import ...
@ManagedBean
@SessionScoped
public class MyBean implements Serializable {
    private int count;
    public MyBean() { this.count = 0; }
    public int getCount() { return this.count++; }
}
```

Annotations évitant la déclaration dans un fichier de configuration

Classe Java Bean pour gérer des propriétés de session



# Principes de JSF (3/6) – Architecture fonctionnelle



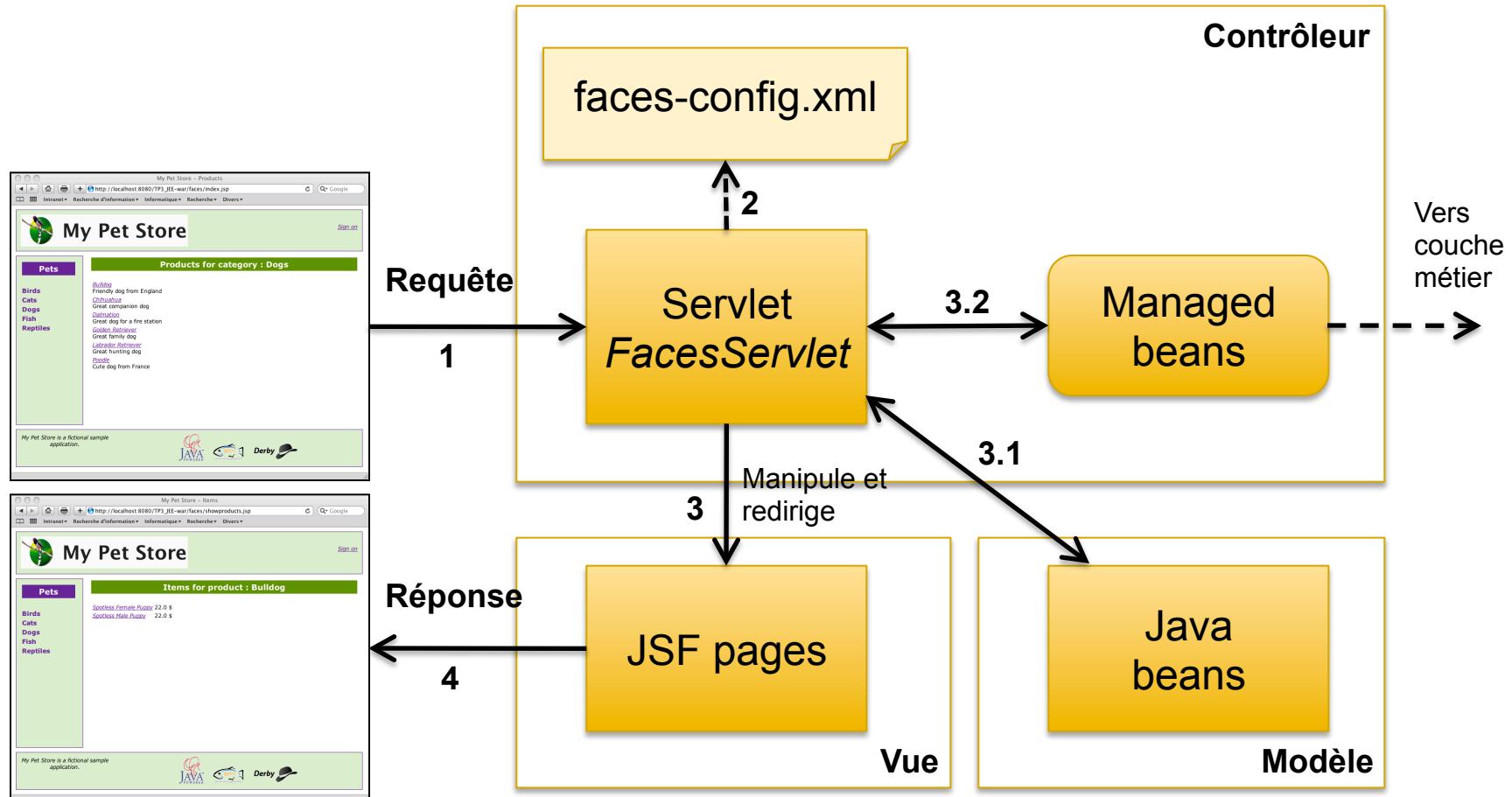
# Principes de JSF (4/6) – Facelets

- *Facelets* est le VDL (*View Declaration Language*) du framework JSF 2.2
- JSP est considéré comme déprécié car il ne supporte pas toutes les possibilités de JSF 2.x
- *Facelets* est constitué d'un ensemble de balises à utiliser dans un document XHTML
- *Facelets* supporte certaines balises JSTL

Bibliothèque	URI + préfixe		Exemple
Balises pour les templates	http://xmlns.jcp.org/jsf/facelets	ui	<ui:composition>
Balises pour les composites	http://xmlns.jcp.org/jsf/composite	composite	<composite:interface>
Balises pour les composants JSF standards	http://xmlns.jcp.org/jsf/html	h	<h:head> <h:outputText>
Balises pour personnaliser les composants	http://xmlns.jcp.org/jsf/core	f	<f:actionListener> <f:selectItem> <f:ajax>
Balises pour le support de HTML5	http://xmlns.jcp.org/jsf http://xmlns.jcp.org/jsf/passthrough	jsf p	<jsf:id> <p:type>
Balises principales de JSTL	http://xmlns.jcp.org/jsp/jstl/core	c	<c:forEach>
Balises des fonctions JSTL	http://xmlns.jcp.org/jsp/jstl/linebreakfunctions	fn	<fn:toUpperCase>

# Principes de JSF (5/6) – MVC

## ■ JSF est basé sur le pattern MVC

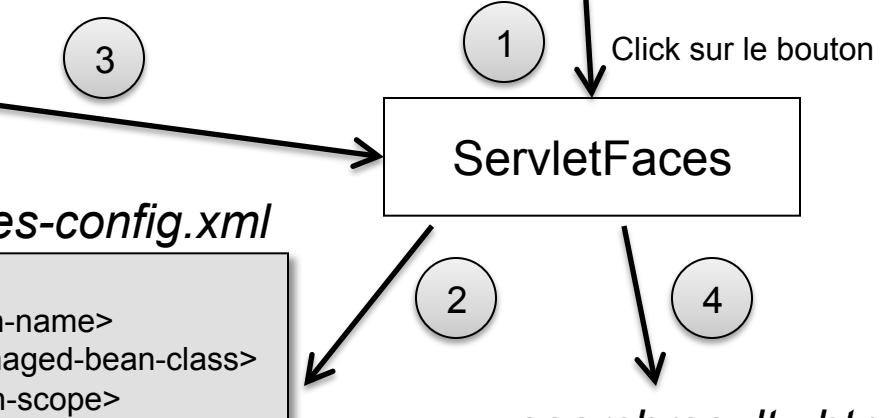
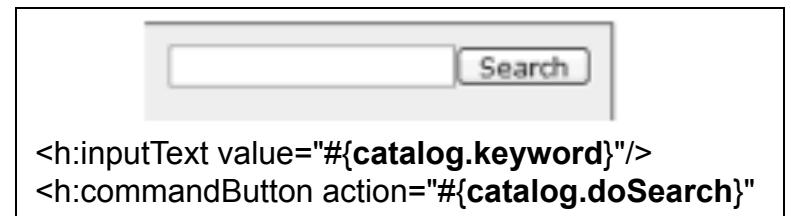


# Principes de JSF (6/6) – MVC (suite)

```
public class CatalogController {  
    private String keyword;  
    private List<Item> items;  
    @EJB private CatalogBeanLocal catalogBean;  
  
    public String doSearch() {  
        items = catalogBean.searchItems(keyword);  
        return "items.found";  
    }  
    ...  
}
```

```
<managed-bean>  
    <managed-bean-name>catalog</managed-bean-name>  
    <managed-bean-class>CatalogController</managed-bean-class>  
    <managed-bean-scope>session</managed-bean-scope>  
</managed-bean>  
<navigation-rule>  
    <from-view-id>*</from-view-id>  
    <navigation-case>  
        <from-outcome>items.found</from-outcome>  
        <to-view-id>/searchresult.xhtml</to-view-id>  
    </navigation-case>  
</navigation-rule>
```

```
<<stateless bean>>  
    CatalogBean  
  
+ searchItems(keyword : String) : String
```

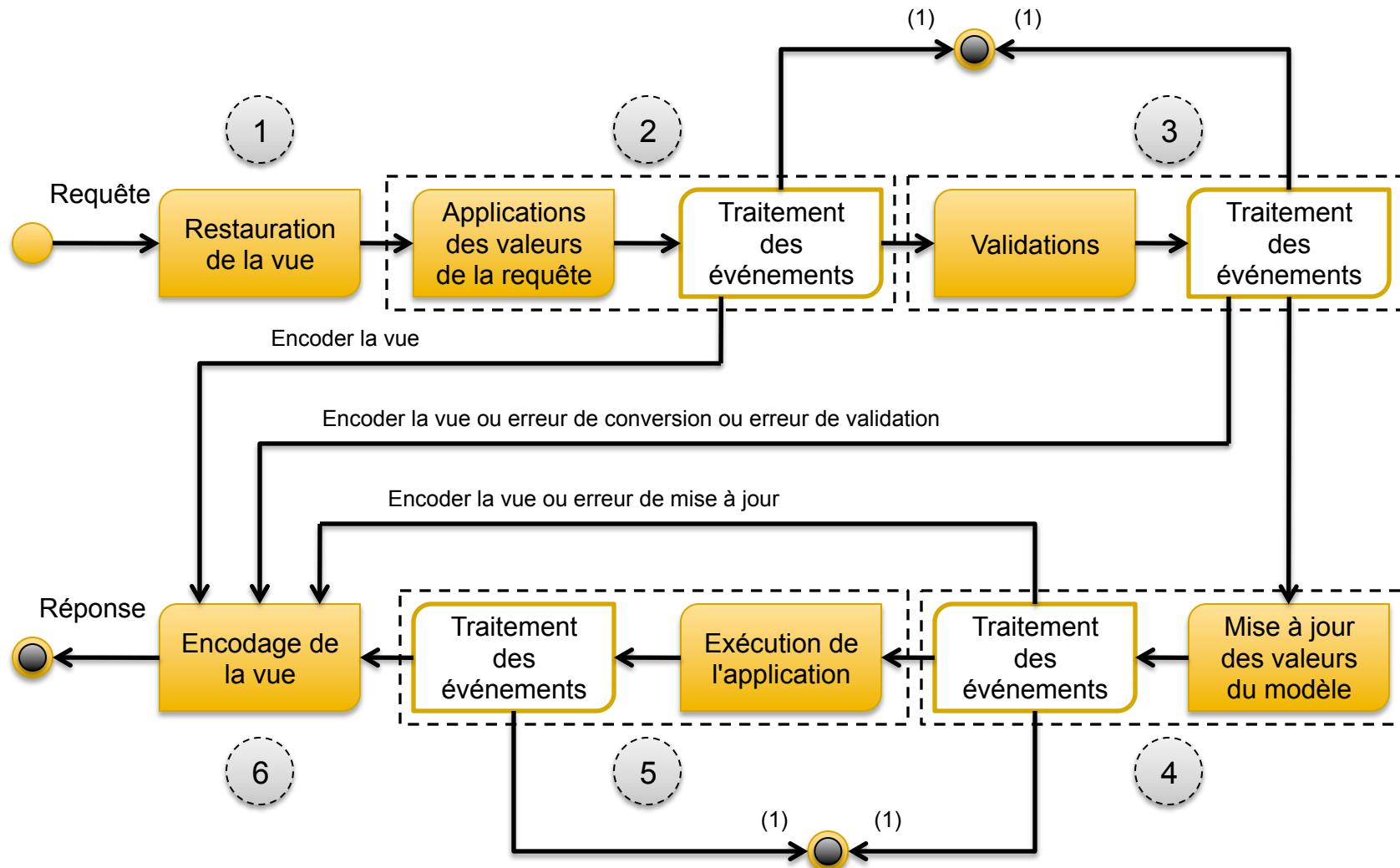


<h: dataTable value="#{catalog.items}">

# Plan du cours

- Communication par composant distribué :
  - Application Web JSF 2.2
    - ✓ Des Servlets à JSP à JSF
    - ✓ Principes de JSF
    - Cycle de vie d'une JSF
    - Les expressions UEL
    - Les *managed beans* dans JSF
    - Composants standards JSF
    - Navigation entre pages et flots de pages JSF
    - Convertisseurs et validateurs de composants
    - Listeners de composants
    - Les fichiers ressources
    - HTML5, JavaScript et Ajax
    - Les templates et les composites
    - Les fichiers de déploiement et de configuration

# Cycle de vie d'une requête (1/4) – Les 6 phases



(1) Redirection ou accès à une vue sans composant JSF ou à une ressource

# Cycle de vie d'une requête (2/4)

- Les types de requêtes :
  - Requête initiale : première requête à une page JSF (cycle court : P1 puis P6)
  - Requête *postback* : traitement d'un formulaire ou lien d'une page JSF (cycle complet)
  - Requête partielle (Ajax) : traitement asynchrone (cycle complet adapté)
- Chaque requête est encapsulée dans un objet de la classe *javax.faces.context.FacesContext* manipulé durant le cycle de vie.  
Il permet notamment :
  - Accès aux composants de la vue (et modification si nécessaire)
  - Accès aux différents niveaux de contexte (application, session...)
  - Ajout de message d'erreur dans la vue
- Le traitement des événements générés pendant une phase est effectué à la fin de la phase en question sauf pour les événements liés aux actions (clics sur un bouton ou un lien) traités uniquement en phase 5.
- En cas d'erreur (erreur de conversion par exemple) ou d'exception durant une des phases du cycle, le traitement passe directement en phase de rendu
- Les fins prématurées du cycle (sans phase de rendu) sont :
  - Soit à cause d'une réponse ne contenant pas de composant JSF (Servlet, JSP pure,...)
  - Soit à cause d'une requête à une ressource (image, fichier CSS ou JavaScript, ...)
  - Soit à cause d'une redirection vers une autre application

# Cycle de vie d'une requête (3/4)

- Phase 1 : restauration de la vue
  - Vue = arborescence des composants JSF (racine : `javax.faces.component.UlviewRoot`)
  - En cas de requête initiale, création d'une nouvelle vue
  - En cas de requête *postback* ou partielle, restauration de l'ancienne vue
- Phase 2 : applications des valeurs de la requête
  - Mettre à jour à jour les valeurs locales des composants de la vue par décodage des paramètres de la requête (valeur locale = valeur soumise)
  - Les composants de type bouton ou lien activés génèrent un événement traité en phase 5
  - Tout composant ayant dans sa balise l'attribut *immediate=true* entraîne :
    - La conversion et sa validation des valeurs saisies dès cette phase (si composant éditable)
    - Le traitement des événements liés au composant dès cette phase
  - Possibilité d'aller directement à la phase de rendu
  - Possibilité de finir le cycle (cas d'une réponse non-JSF par exemple)
- Phase 3 : validations
  - Appel des convertisseurs puis des validateurs sur les valeurs locales des composants
  - Toute erreur de conversion entraîne l'ajout d'un message d'erreur dans la vue
  - Toute erreur de validation entraîne l'ajout d'un message d'erreur dans la vue
  - En cas de succès, stockage de la conversion dans la valeur des composants et génération d'événements traités en fin de phase
  - Possibilité d'aller directement à la phase de rendu (erreur de conversion par exemple)
  - Possibilité de finir le cycle (cas d'une réponse non-JSF par exemple)

# Cycle de vie d'une requête (4/4)

- Phase 4 : mises à jour des valeurs du modèle
  - Modifications des propriétés des Java Beans liées aux attributs *value* des balises des composants
  - Possibilité d'aller directement à la phase de rendu
  - Possibilité de finir le cycle directement
- Phase 5 : exécution de l'application
  - Prise en compte des événements d'application : exécution des méthodes des Java Beans liées aux attributs *action* des balises des composants
  - L'exécution de ces méthodes peut affecter la navigation
  - Possibilité d'aller directement à la phase de rendu
  - Possibilité de finir le cycle directement
- Phase 6 : encodage de la vue
  - Rendu des composants dans une réponse (HTML, XML, etc.)
  - En cas d'erreur, intégration des messages d'erreurs de conversion ou de validation sur la page à l'origine de la requête et rendu de la même page
  - Sauvegarde de l'état de la vue (pour une éventuelle restauration)

# Plan du cours

- Communication par composant distribué :
  - Application Web JSF 2.2
    - ✓ Des Servlets à JSP à JSF
    - ✓ Principes de JSF
    - ✓ Cycle de vie d'une JSF
    - Les expressions UEL
    - Les *managed beans* dans JSF
    - Composants standards JSF
    - Navigation entre pages et flots de pages JSF
    - Convertisseurs et validateurs de composants
    - Listeners de composants
    - Les fichiers ressources
    - HTML5, JavaScript et Ajax
    - Les templates et les composites
    - Les fichiers de déploiement et de configuration

# UEL (1/3) – Unified Expression Language

- Les expressions UEL permettent d'accéder :
  - Aux Java Beans et ainsi à leurs propriétés et à leurs méthodes
  - Aux objets implicites (voir tableau ci-après) mais avec accès en lecture seule
- UEL : *Unified Expression Language*
  - Langage des expressions de JSP :
    - Evaluation immédiate (en phase de rendu uniquement) et en lecture seul
    - Ex. : \${ product.name } // Java Beans *product* et sa propriété *name*
  - + Langage des expressions de JSF :
    - Evaluation immédiate (en cas de requête initiale) ou différée (selon la phase du cycle de vie en cas de requête *postback*)
    - Accès en lecture et en écriture + appel de méthodes
    - Ex. : #{ product.doTotal } // Java Beans *product* et sa méthode *doTotal*
  - Possibilité de combiner les deux types d'expressions

Balise  
JSTL

Balises  
JSF

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="h" uri="http://java.sun.com/jsf/html"%>
...
<c:forEach var="product" items="${products}">
    <h:inputText value="#{product.name}" />
    <h:outputText value="#{product.doTotal}" />
</c:forEach>
```

Page JSP avec  
composants JSF

Expressions  
JSP + JSF

# UEL (2/3) – Opérateurs – Utilisations

- UEL permet d'utiliser les opérateurs suivants :
  - Arithmétiques : + - \* / (**div**) % (**mod**)
  - Relationnels : == (**eq**) != (**neq**) < (**let**) > (**gt**) <= (**le**) >= (**ge**)
  - Logiques : && (**and**) || (**or**) ! (**not**) true false
  - Autres : (...) empty [...] . instanceof null
  - Exemples :
    - **#{ 2 lt 3}** équivalent à **#{ 2 < 3 }** // Retourne un booléen
    - **#{ empty book }** // Retourne true si l'instance est **null**
    - **#{ book.isbn }** équivalent à **#{ book["isbn"] }** // Accès à la propriété ISBN
    - **#{ customer.address["street"] }** // Accès à la propriété street
    - **#{ product.doTotal }** // Appel d'une méthode sans paramètre
    - **#{ book.buy('EURO') }** // Appel d'une méthode avec paramètre
    - **du texte \${ expr } encore du texte** // Construction d'une chaîne
    - **<some:tag value="du#{ expr }#{ expr }texte#{ expr }"/>** // Idem
    - **#{ sessionScope.name }** // Accès à l'attribut *name* via un objet implicite
- Les expressions UEL sont utilisables :
  - dans les pages JSP ou JSF (dans des balises JSTL, JSF, HTML)
  - dans les fichiers de configuration de type *faces-config.xml*
  - dans les annotations des Java Beans
  - dans les fichiers CSS chargé par le composant JSF *h:outputStylesheet*
  - dans les scripts JavaScript écrits dans une page JSF. Par exemple :  
`<script type="text/javascript">var myvariable= #{bean.property};</script>`

# UEL (3/3) – Tableau des objets implicites JSF

application	Instance de la classe ServletContext (accès au moteur de Servlet)
applicationScope	Map des valeurs des attributs du scope application (clé : nom de l'attribut)
cookie	Map des valeurs des cookies de la requête courante (clé : nom du cookie)
facesContext	Instance de la classe FacesContext (utilisé dans le cycle de vie de la requête)
header	Map des valeurs des entêtes HTTP de la requête courante (clé : nom de l'entête)
headerValues	Map des valeurs (sous forme de tableaux de chaînes) des entêtes HTTP de la requête courante (clé : nom de l'entête)
initParam	Map des valeurs des paramètres d'initialisation de l'application (clé : nom du paramètre)
param	Map des valeurs des paramètres de la requête courante (clé : nom du paramètre)
paramValues	Map des valeurs (sous forme de tableaux de chaînes) des paramètres (clé : nom du paramètre)
request	Instance de HttpServletRequest
requestScope	Map des valeurs des attributs du scope requête (clé : nom de l'attribut)
session	Instance de HttpSession (représente la session en cours avec un client)
sessionScope	Map des valeurs des attributs du scope session (clé : nom de l'attribut)
view	Instance de UIViewRoot (racine de l'arborescence des composants de la vue courante)
viewScope	Map des valeurs des attributs du scope view (clé : nom de l'attribut)
component	Instance du composant courant
cc	Instance du composant composite courant de plus au niveau
flowScope	Map des valeurs des attributs du scope flow (clé : nom de l'attribut)
resource	Transforme un identifiant d'une ressource en chemin. Par exemple : value="#{resource['img:flowers/rose.jpeg']}"

# Plan du cours

- Communication par composant distribué :
  - Application Web JSF 2.2
    - ✓ Des Servlets à JSP à JSF
    - ✓ Principes de JSF
    - ✓ Cycle de vie d'une JSF
    - ✓ Les expressions UEL
    - **Les *managed beans* dans JSF**
  - Composants standards JSF
  - Navigation entre pages et flots de pages JSF
  - Convertisseurs et validateurs de composants
  - Listeners de composants
  - Les fichiers ressources
  - HTML5, JavaScript et Ajax
  - Les templates et les composites
  - Les fichiers de déploiement et de configuration

# Les *managed beans* dans JSF (1/4) – Utilisation

- Principes :
  - Classe POJO (*Plain Old Java Object*) (cf. partie JSP) annotée avec l'annotation `@ManagedBean`
  - *Managed beans* : instances gérées par le conteneur de servlet accessibles dans les différents scopes (par défaut `@RequestScope`)
  - *Backing beans* : Java Bean spécifique utilisé pour la mémorisation seule
  - Toutes les annotations peuvent être remplacées par des balises dans un fichier de configuration de ressource du type `faces-config.xml`
- Utilisation des Java Beans avec JSF :
  - Les propriétés et les méthodes sont accessibles via des expressions EL
  - Propriétés utilisées dans les balises des composants : permet la mémorisation et l'accès aux valeurs saisies par l'utilisateur (après conversion et validation)
  - Contrôle de la navigation : permet de choisir et naviguer vers la nouvelle vue compte tenu des informations saisies et de la logique métier
  - Liens avec les composants métier : permet d'accéder à la logique métier
  - Gestionnaires d'événements : permet de réagir aux actions de l'utilisateur sur l'IHM ou à différents moments du cycle de vie
  - Une propriété d'un bean peut référencé un composant de la vue afin d'y accéder et de le modifier facilement (par exemple en modifiant sa visibilité)

# Les *managed beans* dans JSF (2/4) – Déclaration

- **@ManagedBean** : 2 paramètres
  - *name* : indique le nom des instances. Par défaut c'est le nom de la classe commençant par une minuscule.
  - *eager* : indique si le *bean* est instancié dès le démarrage de l'application. Par défaut il vaut *false*.
- Les portées :
  - *@ApplicationScoped* : *bean* commun à toute l'application ; période d'accès = durée de l'application
  - *@SessionScoped* : *bean* commun à un même client ; période d'accès = durée de la session
  - *@FlowScoped* : *bean* commun à un flot de pages ; période d'accès = pas de navigation en dehors du flot
  - *@ViewScoped* : *bean* commun à une même page ; période d'accès = pas de navigation vers une autre page
  - *@RequestScoped* : *bean* commun à une requête ; période d'accès = cycle de traitement de la requête
  - *@NoneScoped* : *bean* non utilisable dans une page ; période d'accès = durée d'utilisation par d'autres *beans*
- Cycle de vie :
  - *@ManagedProperty* : initialisation d'une valeur de propriété à l'instanciation
  - *@PostConstruct* : méthode annotée appelée après l'instanciation du *bean*
  - *@PreDestroy* : méthode annotée appelée avant la suppression du *bean*

```
@ManagedBean(name = "myManagedBean", eager = true)
@SessionScoped
public class BookController {

    @ManagedProperty(value = "#{initController.defaultUser}")
    private User user;

    private Book defaultBook;

    @PostConstruct
    private void init() {
        defaultBook = new Book("default title", 0, "default description");
    }

    // Constructeurs, getters, setters
}
```

Déclaration d'un *managed bean* nommé *myManagedBean* chargé avec l'application et accessible durant une même session

Initialisation à partir de la propriété d'un autre *bean* nommé *initController*

Initialisation explicite dans une méthode *@PostConstruct*

# Les *managed beans* dans JSF (3/4) – Référenciation

```
<h:inputText  
    value="#{cashier.name}"  
    binding="#{cashier.comp}" />
```

```
@ManagedBean  
public class Cashier {  
    private String name; // Name  
    private UIInput comp; // UIComponent  
  
    public String getName() { return name; }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public void setComp(UIInput comp) {  
        this.comp = comp;  
    }  
  
    public String submit() {  
        comp.setRendered(false);  
        return "submitForm.xhtml";  
    }  
}
```

```
<h:commandButton  
    value="#{bundle.Submit}"  
    action="#{cashier.submit}" />
```

- Référencer une propriété d'un *managed bean*
  - Attribut *value* d'une balise :
    - Une expression EL référence le *bean* et sa propriété
    - Le composant peut être éditable (la propriété nécessite un setter) ou pas (la propriété nécessite un getter)
  - Attribut *binding* d'une balise : la propriété référence le composant lui-même qui peut être modifié par le *bean*
  
- Référencer une méthode d'un *managed bean*
  - Attribut *action* d'une balise : la méthode retourne une valeur de navigation (String)
  - Attribut *actionListener* d'une balise : la méthode réceptionne un événement du type *ActionEvent* (retourne void)
  - Attribut *validator* d'une balise : la méthode valide la valeur saisie ou lève une exception (retourne void)
  - Attribut *valueChangeListener* d'une balise : la méthode réceptionne un événement de type *ValueChangeEvent* lorsque la valeur du composant a été modifiée (retourne void)

# Les *managed beans* dans JSF (4/4) – Description

- Les *managed beans* peuvent être déclarés dans un fichier de configuration des ressources

```
<managed-bean eager="true">
    <managed-bean-name>addressBean</managed-bean-name>
    <managed-bean-class>com.example.mybeans.AddressBean</managed-bean-class>
    <managed-bean-scope>none</managed-bean-scope>
    <managed-property>
        <property-name>street</property-name>
        <null-value/>
    <managed-property>
</managed-bean>

<managed-bean eager="true">
    <managed-bean-name>customer</managed-bean-name>
    <managed-bean-class>CustomerBean</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
    <managed-property>
        <property-name>areaCode</property-name>
        <value>#{initParam.defaultAreaCode}</value>
    </managed-property>
    <managed-property>
        <property-name>mailingAddress</property-name>
        <value>#{addressBean}</value>
    </managed-property>
    <managed-property>
        <property-name>options</property-name>
        <map-entries>
            <key-class>int</key-class>
            <value-class>Boolean</value-class>
            <map-entry>
                <key>1</key>
                <value>true</value>
            </map-entry>
        </map-entries>
    </managed-property>
</managed-bean>
```

The diagram illustrates the annotations for managed beans in the XML code. Brackets on the right side group annotations for each managed bean. The first bracket groups annotations for the addressBean managed bean. The second bracket groups annotations for the customer managed bean. The third bracket groups annotations for the areaCode property of the customer managed bean. The fourth bracket groups annotations for the mailingAddress property of the customer managed bean. The fifth bracket groups annotations for the options property of the customer managed bean. The sixth bracket groups annotations for the entire managed-bean element.

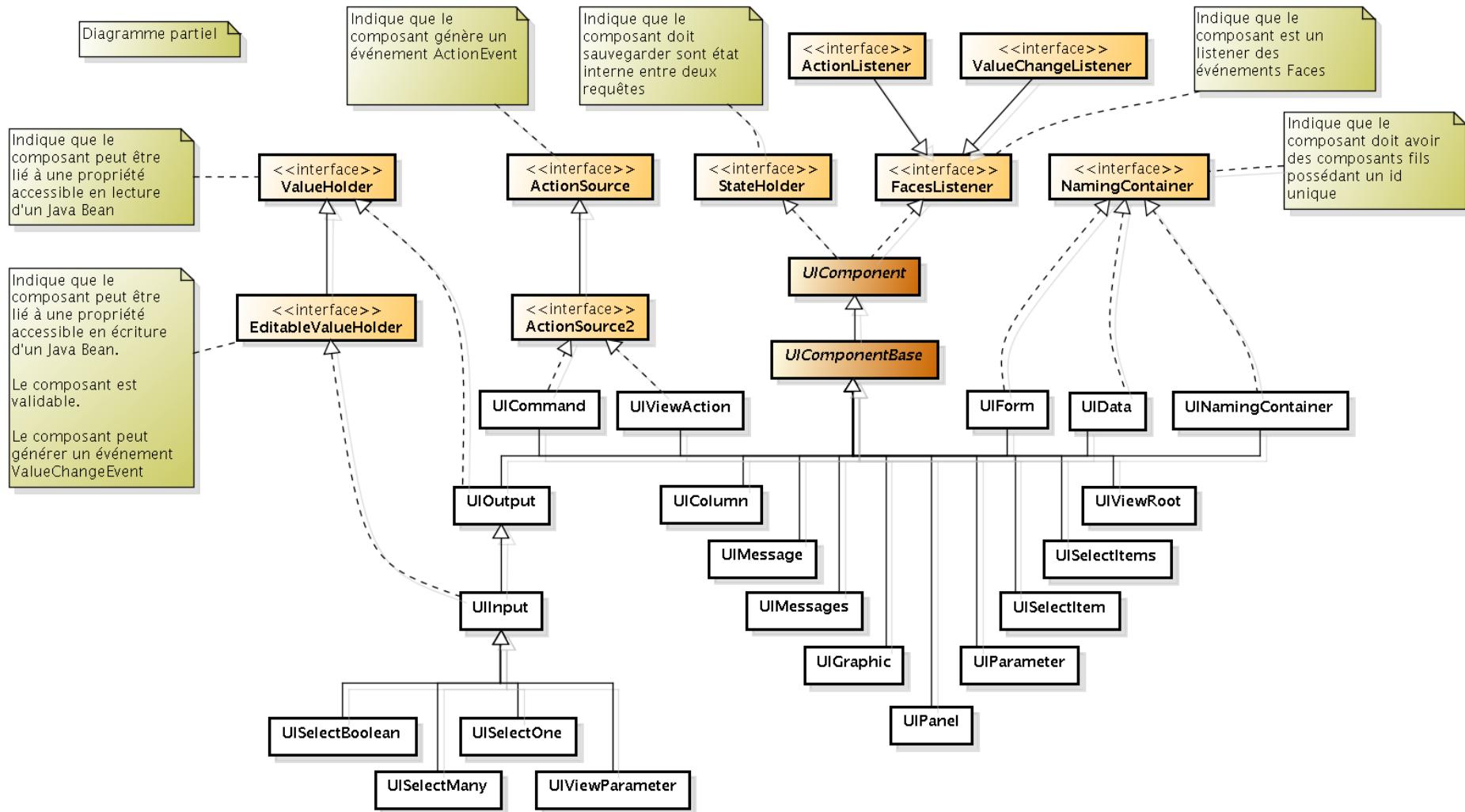
- Managed bean**
  - créé au démarrage
  - nommé *adressBean*
  - de type *AdressBean*
  - ayant aucun scope
  - et une propriété *street* à *null* par défaut
- Managed bean**
  - créé au démarrage
  - nommé *customer*
  - de type *CustomerBean*
  - ayant un scope niveau requête
- Propriété nommée *areaCode* ayant pour valeur initiale un paramètre de l'application
- Propriété nommée *mailingAddress* ayant pour valeur initiale un autre Java Bean nommé *adresBean*
- Propriété nommée *options* ayant pour valeur initiale une map (clé de type int et valeur de type Boolean) avec une seule entrée décrite
- Il est également possible de déclarer des propriétés de type List ou Array (avec *<list-entries>* et des balises *<value>* incluses)

# Plan du cours

- Communication par composant distribué :
  - Application Web JSF 2.2
    - ✓ Des Servlets à JSP à JSF
    - ✓ Principes de JSF
    - ✓ Cycle de vie d'une JSF
    - ✓ Les expressions UEL
    - ✓ Les *managed beans* dans JSF
    - Composants standards JSF
      - Navigation entre pages et flots de pages JSF
      - Convertisseurs et validateurs de composants
      - Listeners de composants
      - Les fichiers ressources
      - HTML5, JavaScript et Ajax
      - Les templates et les composites
      - Les fichiers de déploiement et de configuration

# Composants standards (1/11) – Hiérarchie

- Tous les composants JSF héritent de `javax.faces.component.UIComponent`
- La vue est constituée d'une arborescence de composants dont la racine est de type `UIViewRoot`



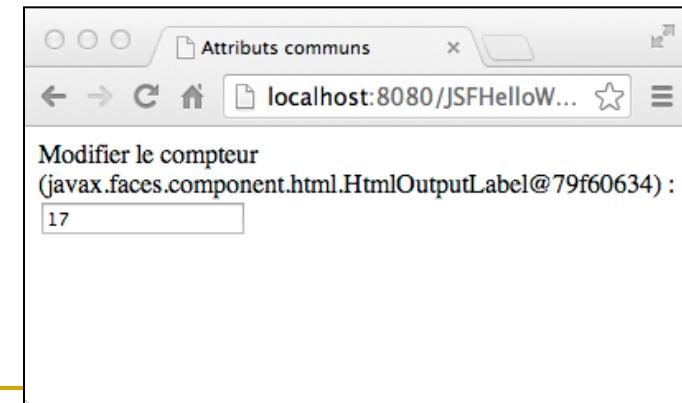
# Composants standards (2/11) – Attributs communs

binding	Référence dans une expression EL une propriété d'un <i>bean</i> et lie cette propriété à ce composant. La propriété et le composant doivent être du même type. Cela permet d'accéder au composant depuis le code java du <i>bean</i> .
id	Identifiant unique du composant (une chaîne)
immediate	Indique si les événements, la conversion et la validation associés à ce composant sont appliqués dès la phase 2 du cycle de traitement d'une requête
rendered	Indique une condition booléenne pour effectuer le rendu ou pas de ce composant (par défaut à true)
style	Précise directement un style CSS dans le composant
styleClass	Précise une classe CSS à utiliser pour ce composant
value	Indique la valeur du composant sous forme d'expression EL Permet de lier une propriété d'un <i>bean</i> à ce composant

```

<?xml version='1.0'encoding='UTF-8'?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
  <h:head>
    <title>Attributs communs</title>
  </h:head>
  <h:body>
    <h:form>
      <h:outputLabel id="MonLabel"
                     binding="#{myBean.CLabel}"
                     rendered="#{myBean.count > 10}"
                     style="font-size: 16px;"
                     value="Modifier le compteur (#{myBean.CLabel}) :"/>
      <h:inputText id="MonChamps"
                   immediate="true"
                   value="#{myBean.count}" />
    </h:form>
  </h:body>
</html>

```



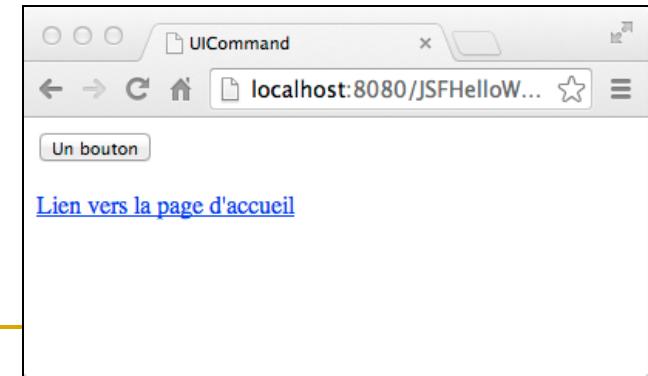
# Composants standards (3/11) – Les commandes

- Ils permettent à l'utilisateur de déclencher une action en activant le composant par un clic
- Balises et attributs spécifiques principaux :

<h:commandButton>	Rendu comme un bouton	<i>action</i> : chaîne de navigation ou expression EL référençant une méthode d'un <i>bean</i> retournant une chaîne de navigation
<h:commandLink>	Rendu comme un lien	<i>actionListener</i> : expression EL référençant une méthode d'un <i>bean</i> de gestion d'un événement de type <i>ActionEvent</i>

```
<?xml version='1.0'encoding='UTF-8'?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>UICommand</title>
    </h:head>
    <h:body>
        <h:form>
            <h:commandButton value="Un bouton"
                             action="JSFHelloWorld"
                             actionListener="#{myBean.doButton}" />
            <br/><br/>
            <h:commandLink value="Lien vers la page d'accueil"
                           action="JSFHelloWorld" />
        </h:form>
    </h:body>
</html>
```

```
@ManagedBean
@SessionScoped
public class MyBean implements Serializable {
    ...
    public void doButton(ActionEvent event) {
        System.out.println("Bouton activé");
        System.out.println(event.getPhaseId().getName()); // Phase 5
    }
}
```

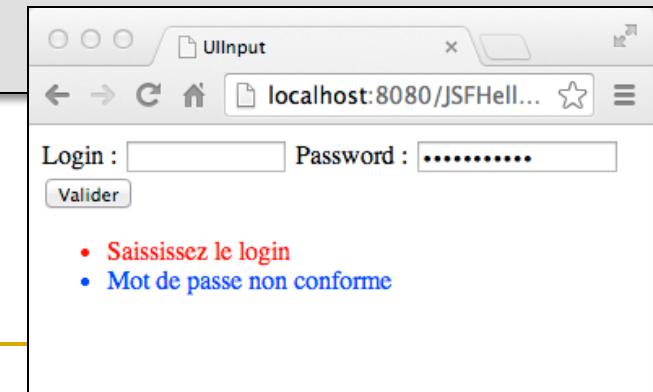


# Composants standards (4/11) – Les champs de saisie

- Ils affichent leur valeur courante (saisie auparavant ou initialisée) et l'utilisateur peut la saisir
- Balises et attributs spécifiques principaux :

<h:inputHidden>	Champ caché	<p><i>converter</i> : référence sur un convertisseur</p> <p><i>converterMessage</i> : message affiché en cas d'erreur de conversion</p> <p><i>label</i> : nom à utiliser pour identifier le composant pour les messages d'erreur</p> <p><i>required</i> : valeur booléenne pour indiquer l'obligation de saisir une chaîne</p> <p><i>requiredMessage</i> : message en cas de non saisie d'un champ obligatoire</p> <p><i>validator</i> : référence sur un validateur</p> <p><i>validatorMessage</i> : message affiché en cas d'erreur de validation</p> <p><i>valueChangeListener</i> : référence sur une méthode de gestion lors de la modification de la valeur du champ</p> <p><i>size</i> : taille par défaut du composant</p> <p><i>maxlength</i> : nombre de caractères saisissables maximum</p>
<h:inputSecret>	Rendu comme un champ de saisie d'une chaîne masquée (saisie d'un mot de passe par exemple)	
<h:inputText>	Rendu comme un champ de saisie d'une chaîne en claire	
<h:inputArea>	Rendu comme une zone de saisie d'un texte	

```
<?xml version='1.0'encoding='UTF-8'?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
  <h:head>
    <title>UIInput</title>
  </h:head>
  <h:body>
    <h:form>
      Login : <h:inputText value="#{myBean.login}"
                           required="true"
                           requiredMessage="Saisissez le login"
                           maxlength="15"
                           size="15"/>
      Password : <h:inputSecret value="#{myBean.psswd}"
                                required="true"
                                redisplay="true"
                                requiredMessage="Saisissez le mot de passe"
                                validator="#{myBean.doValidatePsswd}" />
      <h:commandButton value="Valider" action="JSFHelloWorld"/>
    </h:form>
  </h:body>
</html>
```



# Composants standards (5/11) – Les textes et liens

- Ils affichent leur valeur courante (non éditable) indiquée dans la balise ou liée à une propriété
- Balises :

<h:outputFormat>	Rendu comme un texte littéral avec construction du texte
<h:outputLabel>	Rendu comme un label
<h:outputText>	Rendu comme un texte littéral
<h:outputLink>	Rendu comme un lien (sans génération d'événement)
<h:link>	Idem que le précédent mais indique un lien vers une vue JSF exclusivement

```
<?xml version='1.0'encoding='UTF-8'?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
  <h:head>
    <title>UIOutput</title>
  </h:head>
  <h:body>
    <h:outputFormat value="Bonjour, {0} ! Vous avez visité {1} fois cette page.">
      <f:param value="#{myBean.login}" />
      <f:param value="#{myBean.count}" />
    </h:outputFormat>
    <br/>
    <h:outputLabel value="Un label" style="font-family: sans-serif;" /><br/>
    <h:outputText value="Un texte pas très intéressant"/><br/>
    <h:outputLink title="Lien vers l'accueil" value="/JSFHelloWorld.xhtml">
      Vers JSFHelloWorld
    </h:outputLink>
  </h:body>
</html>
```

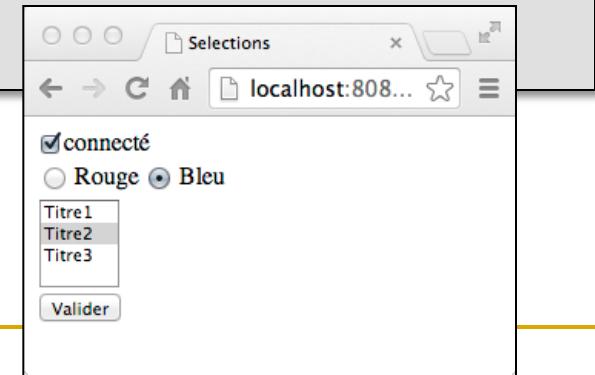


# Composants standards (6/11) – Les sélections

- Ils permettent de sélectionner une ou plusieurs valeurs parmi celles proposées
- Les valeurs peuvent être données « en dur » ou provenir de propriétés de beans
- Balises :

<h:selectManyCheckbox>	Rendu comme une liste de cases à cocher Plusieurs peuvent être sélectionnées
<h:selectManyListBox>	Rendu comme une liste à choix multiples
<h:selectManyMenu>	Rendu comme une liste déroulante de valeurs Plusieurs peuvent être sélectionnées
<h:selectOneListBox>	Rendu comme une liste à plusieurs valeurs mais une seule peut être sélectionnée
<h:selectOneMenu>	Rendu comme une liste déroulante à choix unique N'affiche qu'une seule valeur à la fois
<h:selectOneRadio>	Rendu comme une liste de boutons radio
<h:selectBooleanCheckbox>	Rendu sous forme de case à cocher

```
<?xml version='1.0'encoding='UTF-8'?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
 xmlns:h="http://xmlns.jcp.org/jsf/html"
 xmlns:f="http://xmlns.jcp.org/jsf/core">
<h:head>
 <title>Selections</title>
</h:head>
<h:body>
<h:form>
 <h:selectBooleanCheckbox value="#{myBean.isLogin}" />
 <h:outputLabel value="connecté"/><br/>
 <h:selectOneRadio value="#{myBean.couleur}" />
 <f:selectItem itemValue="Rouge" itemLabel="Rouge"/>
 <f:selectItem itemValue="Bleu" itemLabel="Bleu"/>
</h:selectOneRadio>
 <h:selectOneListbox value="#{myBean.select}" />
 <f:selectItems value="#{myBean.articles}" var="art"
 itemValue="#{art.ident}" itemLabel="#{art.titre}" />
</h:selectOneListbox>
<br/><h:commandButton value="Valider"/>
</h:form>
</h:body>
</html>
```

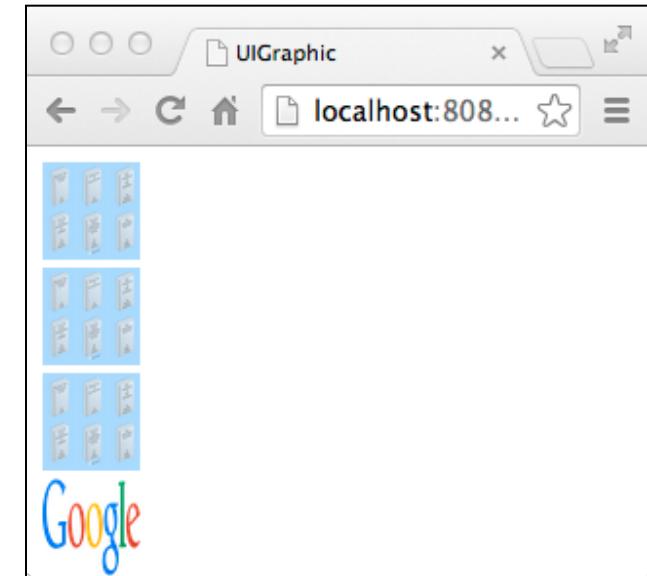


# Composants standards (7/11) – Les images

- Un composant JSF permet de charger et d'afficher des ressources de type image
- Balise et principaux attributs spécifiques :

<h:graphicImage>	Rendu comme une image	<i>height</i> : hauteur en pixel <i>width</i> : largeur en pixel <i>url</i> : URI de la ressource (alias de <i>value</i> ) <i>name</i> : nom de la ressource <i>library</i> : nom de la librairie contenant la ressource <i>alt</i> : nom alternatif de la ressource <i>usemap</i> : identifiant d'une carte de zones cliquables
------------------	-----------------------	--

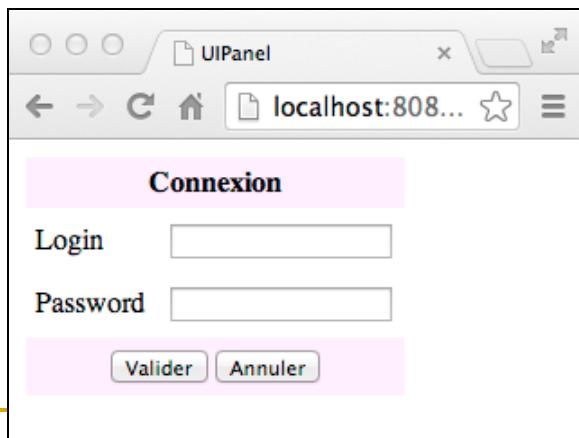
```
<?xml version='1.0'encoding="UTF-8'?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
<h:head>
  <title>UIGraphic</title>
</h:head>
<h:body>
  <h:graphicImage url="/resources/images/book_all.jpg"
                  height="50" width="50"/>
  <br/>
  <h:graphicImage name="book_all.jpg" library="images" alt="Une image"
                  height="50" width="50"/>
  <br/>
  <h:graphicImage value="#{resource['images:book_all.jpg']}"
                  height="50" width="50"/>
  <br/>
  <h:graphicImage value="http://www.google.fr/images/srpr/logo11w.png"
                  height="50" width="50"/>
</h:body>
</html>
```



# Composants standards (8/11) – Les grilles

- Ils permettent de structurer la vue en disposant les composants
- Balises :

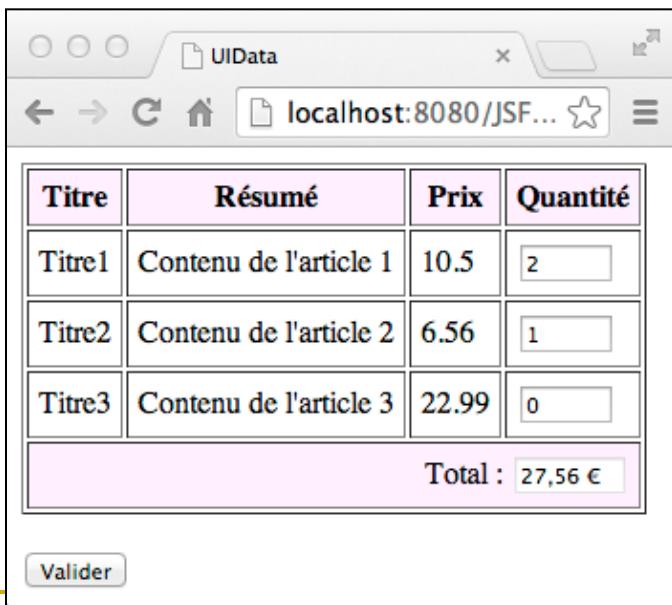
<h:panelGroup>	Groupe plusieurs composants sous un même parent
<h:panelGrid>	Rendu comme une table (mais pas de modèle de données)  <i>columns</i> : indique le nombre de colonne <i>footerClass</i> : classe CSS appliquée au pied <i>headerClass</i> : classe CSS appliquée à l'entête <i>columnClasses</i> : liste des classes CSS appliquées aux colonnes <i>rowClasses</i> : liste des classes CSS appliquées aux lignes



```
<?xml version='1.0'encoding='UTF-8'?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
  <h:head>
    <title>UIPanel</title>
    <h:outputStylesheet name="style.css" library="style"/>
  </h:head>
  <h:body>
    <h:form>
      <b><h:panelGrid columns="2 ">
        <h:column headerClass="list-header">
          <h:outputText value="Connexion au site" />
        </h:column>
        <h:column footerClass="list-footer">
          <h:outputText value="Une description de la grille" />
        </h:column>
      </h:panelGrid></b>
      <h:panelGroup>
        <h:panelGrid columns="2">
          <h:column headerClass="list-header">
            <h:outputText value="Connexion" />
          </h:column>
          <h:column footerClass="list-footer">
            <h:outputText value="Connexion" />
          </h:column>
        </h:panelGrid>
        <h:panelGrid columns="2">
          <h:column headerClass="list-header">
            <h:outputText value="Login" />
          </h:column>
          <h:column footerClass="list-footer">
            <h:inputText value="#{myBean.login}" required="true" />
          </h:column>
        </h:panelGrid>
        <h:panelGrid columns="2">
          <h:column headerClass="list-header">
            <h:outputText value="Password" />
          </h:column>
          <h:column footerClass="list-footer">
            <h:inputSecret value="#{myBean.psswd}" required="true" />
          </h:column>
        </h:panelGrid>
      </h:panelGroup>
      <h:panelGrid columns="2">
        <h:column headerClass="list-header">
          <h:commandButton value="Valider" />
        </h:column>
        <h:column footerClass="list-footer">
          <h:commandButton value="Annuler" type="reset" />
        </h:column>
      </h:panelGrid>
    </h:panelGrid>
  </h:form>
  </h:body>
</html>
```

# Composants standards (9/11) – Les tableaux

<h:dataTable>	Rendu comme un tableau à partir d'un modèle de données sous-jacent (un tableau, une liste, une table...)	<p><i>var</i> : variable d'itération des données de l'attribut <i>value</i></p> <p><i>border</i> : épaisseur de la bordure</p> <p><i>first</i> : numéro de la première ligne à afficher</p> <p><i>rows</i> : nombre de lignes à afficher</p> <p><i>captionClass</i> : classe du titre</p> <p><i>captionStyle</i> : style CSS du titre</p> <p><i>footerClass</i> : classe CSS du pied</p> <p><i>headerClass</i> : classe CSS de l'entête</p> <p><i>columnClasses</i> : liste des classes CSS appliquées aux colonnes</p> <p><i>rowClasses</i> : liste des classes CSS appliquées aux lignes</p> <p><i>width</i> : largeur du tableau</p> <p><i>summary</i> : résumé du tableau</p>
---------------	--	---

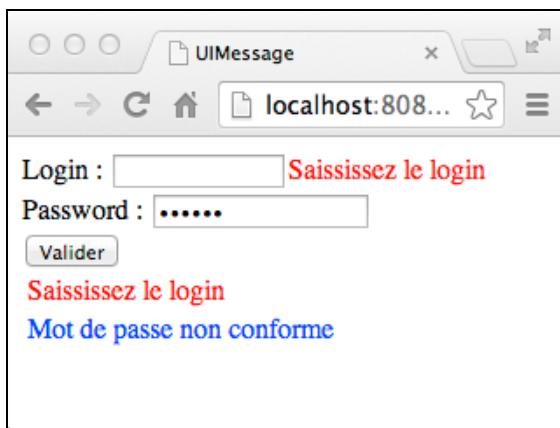


```
...
<h:dataTable value="#{myBean.articles}" var="art" border="1"
    headerClass="list-header" footerClass="list-footer "
    cellpadding="5" cellspacing="2">
    <h:column>
        <f:facet name="header">
            <h:outputText value="Titre"/>
        </f:facet>
        <h:outputText value="#{art.titre}"/>
    </h:column>
    <h:column>
        <f:facet name="header">
            <h:outputText value="Résumé"/>
        </f:facet>
        <h:outputText value="#{art.contenu}"/>
    </h:column>
    <h:column>
        <f:facet name="header">
            <h:outputText value="Prix"/>
        </f:facet>
        <h:outputText value="#{art.prix}"/>
    </h:column>
    <h:column>
        <f:facet name="header">
            <h:outputText value="Quantité"/>
        </f:facet>
        <h:inputText value="#{art.quantite}" size="5"/>
    </h:column>
    <f:facet name="footer">
        <h:panelGroup>
            <h:outputText value="Total : "/>
            <h:inputText value="#{myBean.total}" readonly="true" size="7">
                <f:convertNumber currencySymbol="€" type="currency" />
            </h:inputText>
        </h:panelGroup>
    </f:facet>
</h:dataTable>
...
```

# Composants standards (10/11) – Les messages

- Ils permettent d'afficher des messages automatiques (par défauts ou précisés) ou programmés
- Balises :

<h:message>	Zone de notification pour un composant. Rendu comme du texte	<i>for</i> : indique l'identifiant du composant source du message <i>infoStyle</i> , <i>warnStyle</i> , <i>errorStyle</i> , <i>fatalStyle</i> : style CSS selon le niveau de严重性 <i>infoClass</i> , <i>warnClass</i> , <i>errorClass</i> , <i>fatalClass</i> : classe CSS selon le niveau de严重性 <i>showDetail</i> , <i>showSummary</i> : affiche le message détaillé, affiche le message résumé
<h:messages>	Zone de notification globale. Rendu comme un tableau ou une liste	



```
...
<h:form>
    Login : <h:inputText id="textlogin" value="#{myBean.login}"
                    required="true" requiredMessage="Saississez le login"/>
    <h:message for="textlogin" warnStyle="color:blue;" errorStyle="color:red;"/>
    <br/>
    Password : <h:inputSecret value="#{myBean.psswd}">
                    required="true" requiredMessage="Saississez le mot de passe"
                    validator="#{myBean.doValidatePsswd}"/>
    <br/>
    <h:commandButton value="Valider" action="JSFHelloWorld"/>
    <br/>
    <h:messages showDetail="true" showSummary="false"
                errorStyle="color:red;" warnStyle="color:blue;"/>
</h:form>
...
```

```
public void doValidatePsswd(FacesContext ctx, UIComponent c, Object obj) {
    String passwd = (String)obj;
    if (passwd.length() < 15) {
        ((UIInput)c).setValid(false);
        FacesMessage message = new FacesMessage(FacesMessage.SEVERITY_WARN, "Problème de saisie", "Mot de passe non conforme");
        ctx.addMessage(c.getClientId(ctx), message);
    }
}
```

# Composants standards (11/11) – Divers balises

<h:head> <h:body>	Déclaration de la zone d'entêtes et du corps de la vue. Balises utilisées principalement pour la réallocation de ressources
<h:form>	Déclaration d'un formulaire pour poster les informations saisies
<f:param>	Paramètre pour le composant <i>h:outputFormat</i>  Ajout d'un paramètre à la liste des paramètres de la requête postée
<f:attribute>	Ajoute un attribut à un composant. L'attribut peut être un attribut normal de la balise ou n'importe quel autre attribut.
<f:setPropertyActionListener>	Ajoute un listener standard qui sauvegarde une valeur dans une propriété d'un <i>backing bean</i> quand le formulaire est soumis
<f:facet>	Ajoute à un composant parent un composant spécifique Une <i>facet</i> ne peut contenir qu'un seul fils (cf. les grilles et les tableaux)
<f:metadata> <f:viewParam> <f:viewAction>	Balises utiles pour gérer les paramètres d'une requête GET (non abordé dans ce cours)

```
...
public void attrListener(ActionEvent event) {
    couleur = (String)event.getComponent().getAttributes().get("couleur");
}
...
```

```
<?xml version='1.0'encoding='UTF-8'?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
<h:head>
    <title>Divers balises</title>
</h:head>
<h:body>
    <h:form>
        <h:outputLabel value="Votre nom : "/>
        <h:inputText value="#{myBean.login}" />
        <br/><br/>
        <h:commandButton actionListener="#{myBean.attrListener}">
            <f:attribute name="value" value="Soumission"/>
            <f:attribute name="couleur" value="verte"/>
            <f:param name="country" value="France"/>
            <f:setPropertyActionListener
                target="#{myBean.psswd}" value="le mot de passe" />
        </h:commandButton>
        <br/><br/>
        <h:outputText value="Attribut précédent : #{myBean.couleur}" />
    </h:form>
</h:body>
</html>
```



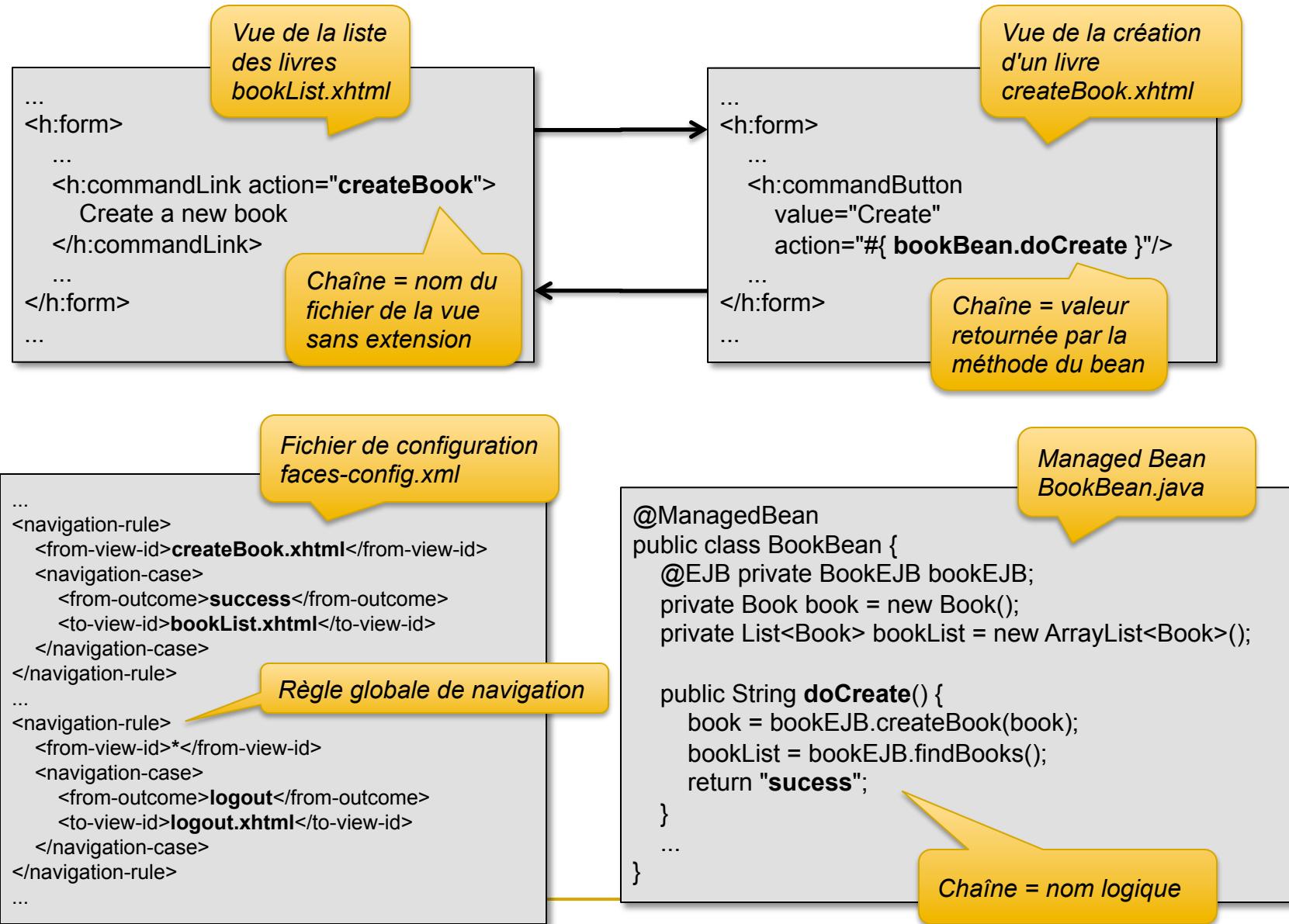
# Plan du cours

- Communication par composant distribué :
  - Application Web JSF 2.2
    - ✓ Des Servlets à JSP à JSF
    - ✓ Principes de JSF
    - ✓ Cycle de vie d'une JSF
    - ✓ Les expressions UEL
    - ✓ Les *managed beans* dans JSF
    - ✓ Composants standards JSF
    - Navigation entre pages et flots de pages JSF
    - Convertisseurs et validateurs de composants
    - Listeners de composants
    - Les fichiers ressources
    - HTML5, JavaScript et Ajax
    - Les templates et les composites
    - Les fichiers de déploiement et de configuration

# Navigation dans une application JSF (1/2)

- La navigation consiste pour la servlet de contrôle *ServletFaces* à choisir la vue suivante de la vue actuelle qui sera rendue au client en réponse à sa requête
- La navigation peut se faire de trois manières différentes :
  - Via des règles de navigation dans un fichier de configuration ; c'est la seule manière d'avoir des règles globales
  - Via des chaînes de navigation directement dans les balises des composants JSF des vues
  - Via des méthodes des *managed bean* qui retournent une chaîne de navigation, ce qui permet de mettre en œuvre une logique de navigation dynamique
- On peut mélanger dans une même application ces trois manières de naviguer
- Les chaînes de navigation peuvent être :
  - Soit des noms des fichiers des vues (avec ou sans extension)
  - Soit des noms logiques

# Navigation dans une application JSF (2/2)

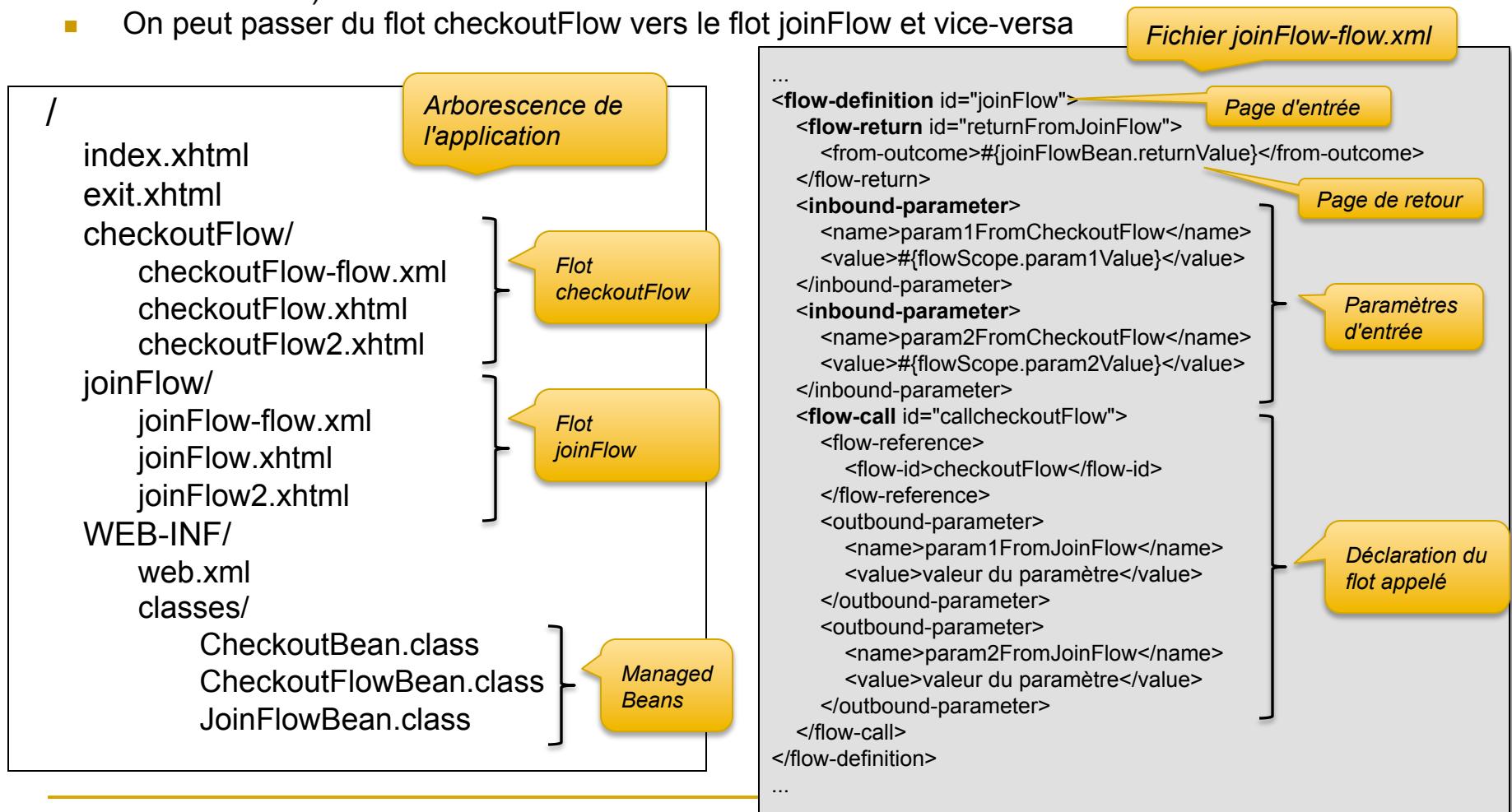


## Flot de pages JSF (1 / 3)

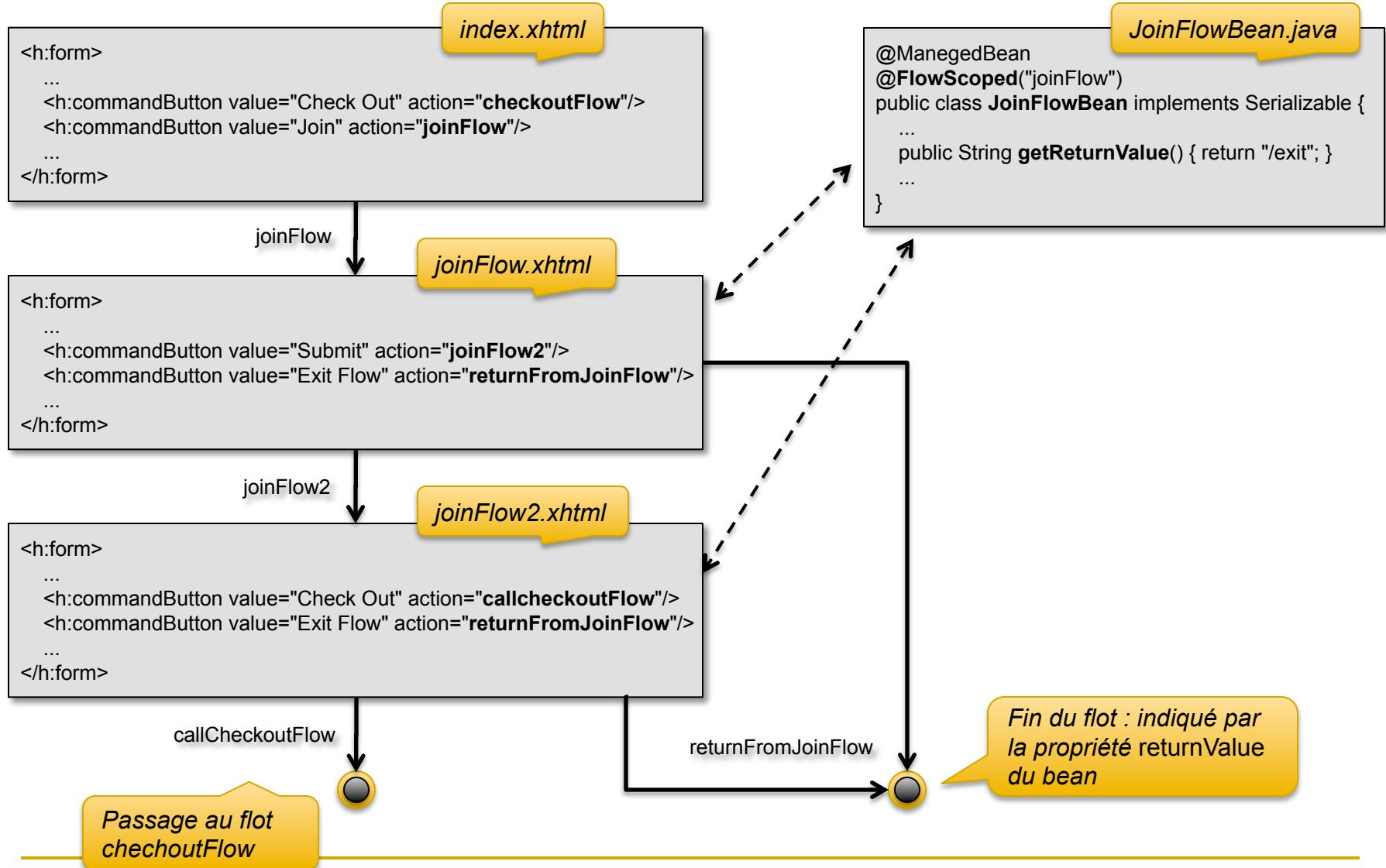
- Un flot est un ensemble de pages liées fonctionnellement (par exemple un flot pour ouvrir un compte sur un site marchand ou pour finaliser l'achat d'articles d'un caddie)
- Les pages du flot partagent un même scope *FlowScope* durant lequel des *managed beans* sont accessibles
- Les flots sont définis dans des fichiers de configuration ou par programmation :
  - On définit les pages d'entrée et de sorties
  - Les éventuels paramètres d'entrée
  - Les appels des autres flots

# Flot de pages JSF (2/3) – Exemple

- Une application possède deux flots :
  - joinFlow : flot d'inscription au site défini dans le fichier joinFlow-flow.xml
  - checkoutFlow : flot de finalisation des achats d'un caddie virtuel (fichier checkoutFlow-flow.xml)
- On peut passer du flot checkoutFlow vers le flot joinFlow et vice-versa



# Flot de pages JSF (3/3) – Exemple du flot joinFlow



# Plan du cours

- Communication par composant distribué :
  - Application Web JSF 2.2
    - ✓ Des Servlets à JSP à JSF
    - ✓ Principes de JSF
    - ✓ Cycle de vie d'une JSF
    - ✓ Les expressions UEL
    - ✓ Les *managed beans* dans JSF
    - ✓ Composants standards JSF
    - ✓ Navigation entre pages et flots de pages JSF
    - Convertisseurs et validateurs de composants
    - Listeners de composants
    - Les fichiers ressources
    - HTML5, JavaScript et Ajax
    - Les templates et les composites
    - Les fichiers de déploiement et de configuration

# Les convertisseurs (1/3)

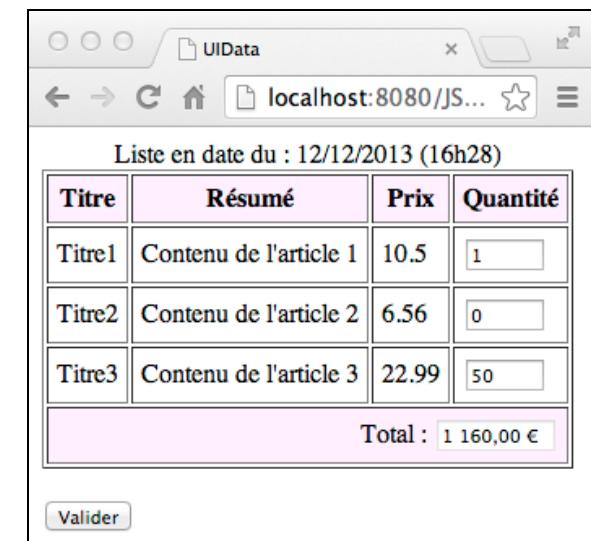
- Les convertisseurs sont utilisés dans deux cas :
  - Lors du traitement d'une requête, pour transformer les paramètres (chaîne de caractère) en objets Java correspondants aux propriétés des Java Beans
  - Lors du rendu de la vue, pour transformer les objets Java en chaîne de caractère qui sont incorporées dans la réponse du serveur
- Des convertisseurs standards (package `javax.faces.convert`) existent pour les nombres et les dates ; pour les autres types, il faut fournir ses propres convertisseurs
- En cas d'erreur de conversion une exception `ConverterException` est levée et la même page est retournée avec un message incorporé (qui peut être modifiée via l'attribut `converterMessage`)
- `DateTimeConverter` et `NumberConverter` peuvent être personnalisés notamment grâce à l'attribut `binding` permettant d'accéder au convertisseur depuis une propriété d'un Java Bean

BigDecimalConverter	Convertit un String en <code>java.math.BigDecimal</code> et vice versa
BigIntegerConverter	Convertit un String en <code>java.math.BigInteger</code> et vice versa
BooleanConverter	Convertit un String en Boolean (et <code>boolean</code> ) et vice versa
ByteConverter	Convertit un String en Byte (et <code>byte</code> ) et vice versa
CharacterConverter	Convertit un String en Character (et <code>char</code> ) et vice versa
DateTimeConverter	Convertit un String en <code>java.util.Date</code> et vice versa
DoubleConverter	Convertit un String en Double (et <code>double</code> ) et vice versa
EnumConverter	Convertit un String en Enum (et <code>enum</code> ) et vice versa
FloatConverter	Convertit un String en Float (et <code>float</code> ) et vice versa
IntegerConverter	Convertit un String en Integer (et <code>int</code> ) et vice versa
LongConverter	Convertit un String en Long (et <code>long</code> ) et vice versa
NumberConverter	Convertit un String en classe abstraite <code>java.lang.Number</code> et vice versa
ShortConverter	Convertit un String en Short (et <code>short</code> ) et vice versa

# Les convertisseurs (2/3)

<h:convertNumber>	<i>type</i> : interprétation de la valeur ( <i>number</i> , <i>currency</i> ou <i>percent</i> ) <i>currencySymbol</i> : symbole affiché lors que le nombre est de type <i>currency</i> <i>currencyCode</i> : code ISO 4217 du symbole pour un nombre de type <i>currency</i> <i>maxFractionDigits</i> , <i>maxIntegerDigits</i> : nombre maximum de chiffre après et avant la virgule <i>minFractionDigits</i> , <i>minIntegerDigits</i> : nombre minimum de chiffre après et avant la virgule <i>locale</i> : indique le code de la langue (fr, us, en, etc.) <i>pattern</i> : format du nombre à respecter
<h:convertDateTime>	<i>type</i> : interprétation de la valeur ( <i>date</i> , <i>time</i> , <i>both</i> ) <i>locale</i> : indique le code de la langue <i>pattern</i> : format du nombre à respecter <i>timeStyle</i> : style d'affichage du temps ( <i>default</i> , <i>short</i> , <i>medium</i> , <i>long</i> , <i>full</i> ) <i>dateStyle</i> : style d'affichage de la date ( <i>default</i> , <i>short</i> , <i>medium</i> , <i>long</i> , <i>full</i> ) <i>timeZone</i> : localisation du temps

```
...  
<h:form>  
  <h:dataTable value="#{myBean.articles}" ...>  
    <f:facet name="caption">  
      Liste en date du :  
      <h:outputText value="#{myBean.today()}">  
        <f:convertDateTime type="both" locale="fr" pattern="dd/MM/YYYY (HH'h'mm)" />  
      </h:outputText>  
    </f:facet>  
    ...  
    <f:facet name="footer">  
      <h:panelGroup>  
        <h:outputText value="Total : "/>  
        <h:inputText value="#{myBean.total}" readonly="true" size="10">  
          <f:convertNumber type="currency" currencyCode="EUR" maxFractionDigits="2" />  
        </h:inputText>  
      </h:panelGroup>  
    </f:facet>  
  </dataTable>  
</h:form>  
...
```



The screenshot shows a web browser window with the URL `localhost:8080/JSP/UIData`. The page displays a table titled "Liste en date du : 12/12/2013 (16h28)". The table has four columns: Titre, Résumé, Prix, and Quantité. The data rows are:

Titre	Résumé	Prix	Quantité
Titre1	Contenu de l'article 1	10.5	1
Titre2	Contenu de l'article 2	6.56	0
Titre3	Contenu de l'article 3	22.99	50

Below the table, a summary row shows "Total : 1 160,00 €". At the bottom of the page is a "Valider" button.

# Les convertisseurs (3/3)

- Pour définir son propre convertisseur, il faut une
  - classe annotée `@FacesConverter` ou déclarée dans un fichier de configuration
  - et qui implémente l'interface `javax.faces.convert.Converter`

```
...
@FacesConverter(value="dollarConverter")
public class DollarConverter implements Converter {

    @Override
    public Object getAsObject(FacesContext context, UIComponent component, String value) {
        return value;
    }

    @Override
    public String getAsString(FacesContext context, UIComponent component, Object value) {
        float amountInEuros = Float.parseFloat(value.toString());
        double amountInDollars = amountInEuros * 1.374;

        NumberFormat f = NumberFormat.getInstance(Locale.US);
        if (f instanceof DecimalFormat) {
            ((DecimalFormat) f).applyPattern("#####,##0.##");
        }
        return f.format(amountInDollars);
    }
}
```



```
...
Prix en euros :
<h:outputText value="#{myBean.article.prix}">
    <f:convertNumber type="currency" currencySymbol="€" locale="fr"/>
</h:outputText><br/>
Prix en dollars (1) :
<h:outputText value="#{myBean.article.prix}"><br/>
    <f:converter converterId="dollarConverter"/>
</h:outputText><br/>
Prix en dollars (2) :
<h:outputText value="#{myBean.article.prix}" converter="dollarConverter"/>
...
```

# Les validateurs (1 / 5)

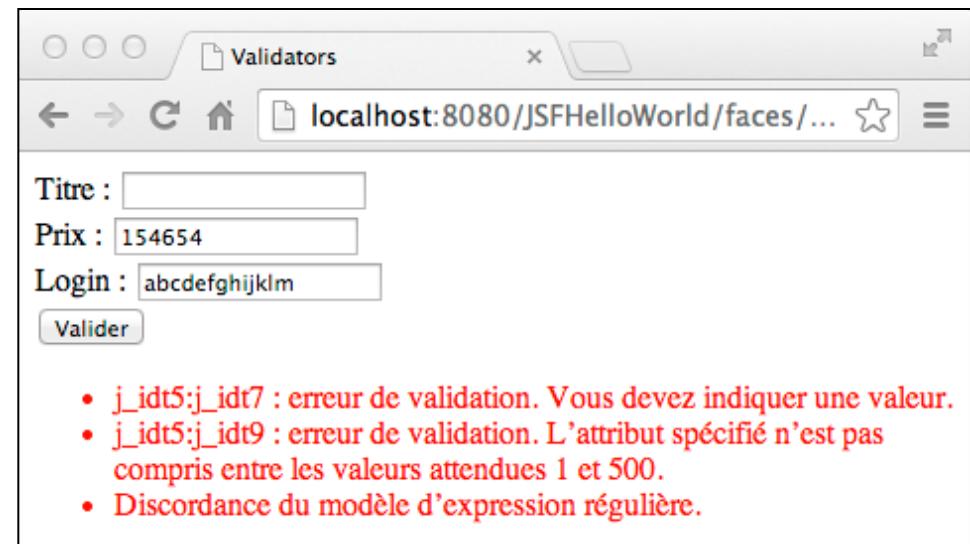
- La validation des informations saisies avant la mise à jour du modèle des données se fait côté serveur par des validateurs standards ou créés sur mesure
- En cas d'erreur de validation une exception *ValidatorException* est levée et la même page est retournée avec un message incorporée qui peut être modifiée via l'attribut *validatorMessage*
- L'attribut *required* d'un composant *UIInput* indique que la valeur est obligatoire. Si elle n'est pas saisie cela entraîne une erreur de validation
- L'attribut *binding* d'un validateur permet de référencer une propriété d'un *Java Bean* du même type que le validateur et ainsi d'accéder au validateur depuis le code java
- Tableau des validateurs standards :

DoubleRangeValidator	Compare la valeur du composant aux valeurs minimales et maximales indiquées (de type double)
LongRangeValidator	Compare la valeur du composant aux valeurs minimales et maximales indiquées (de type long)
LengthValidator	Teste le nombre de caractères de la valeur textuelle du composant
RegexValidator	Compare la valeur du composant à une expression régulière

# Les validateurs (2/5) – Validateurs standards

DoubleRangeValidator	<i>minimum</i> : valeur minimale admise <i>maximum</i> : valeur maximale admise
LengthValidator	
LongRangeValidator	
RegexValidator	<i>pattern</i> : format admis (cf. le package <i>java.util.regex</i> )

```
...  
<h:form>  
    Titre :  
        <h:inputText value="#{myBean.article.titre}" required="true">  
            <f:validateLength minimum="2" maximum="20"/>  
        </h:inputText>  
        <br/>  
    Prix :  
        <h:inputText value="#{myBean.article.prix}">  
            <f:validateDoubleRange minimum="1" maximum="500"/>  
        </h:inputText>  
        <br/>  
    Login :  
        <h:inputText value="#{myBean.login}">  
            <f:validateRegex pattern="(^[a-zA-Z]{6,10}$)" />  
        </h:inputText>  
        <br/>  
        <h:commandButton value="Valider"/>  
</h:form>  
...
```



The screenshot shows a web browser window with the URL `localhost:8080/JSFHelloWorld/faces/...`. The page contains three input fields: 'Titre' (empty), 'Prix' (value '154654'), and 'Login' (value 'abcdefghijklm'). Below the inputs is a 'Valider' button. To the right of the browser window, a list of validation errors is displayed in red text:

- j\_idt5:j\_idt7 : erreur de validation. Vous devez indiquer une valeur.
- j\_idt5:j\_idt9 : erreur de validation. L'attribut spécifié n'est pas compris entre les valeurs attendues 1 et 500.
- Discordance du modèle d'expression régulière.

# Les validateurs (3/5) – Validateurs sur mesure (1/3)

- Il est possible de définir ses propres contraintes de validations :

- 1) Soit en implémentant une méthode d'un *managed bean* qui effectue la validation
- 2) Soit en créant un validateur spécifique (classe Java) en implémentant l'interface *Validator* et en l'annotant `@FacesValidator` ou en le déclarant dans un fichier de configuration
- 3) Soit en annotant des attributs, des méthodes ou des classes : c'est la validation par *Java Beans (Bean Validation)*

Méthode 1 : méthode  
d'un managed bean

```
...  
public void validateNumberRange(FacesContext context, UIComponent toValidate, Object value) {  
    int input = (Integer) value; // Property of type int  
    if (input < minimum || input > maximum) { // Property constraint  
        ((UIInput) toValidate).setValid(false); // Invalidate component  
        FacesMessage message = new FacesMessage("Invalid guess"); // Message to show  
        context.addMessage(toValidate.getClientId(context), message); // Add message to the view  
    }  
}  
...
```

Référencement de la  
méthode

```
...  
<h:inputText id="inputGuess"  
    value="#{userNumberBean.userNumber}"  
    required="true"  
    size="3"  
    validator="#{userNumberBean.validateNumberRange}">  
</h:inputText>  
...
```

# Les validateurs (4/5) – Validateurs sur mesure (2/3)

Méthode 2 : créer un validateur spécifique

```
...
@FacesValidator(value = "isbnValidator")
public class IsbnValidator implements Validator {
    private Pattern pattern;
    private Matcher matcher;
    @Override
    public void validate(FacesContext context, UIComponent component, Object value) {
        String componentName = value.toString();
        pattern = Pattern.compile("(?=\\d{13})");
        matcher = pattern.matcher(componentName);
        if (!matcher.find()) {
            String message = MessageFormat.format("{0} is not a valid isbn format", componentName);
            FacesMessage facesMessage = new FacesMessage(SEVERITY_ERROR, message, message);
            throw new ValidatorException(facesMessage);
        }
    }
}
```

Annotation pour déclarer un validateur

Méthode de l'interface Validator

Motif à respecter pour les numéros ISBN

Lever une exception

Création d'un message

Utilisation du validateur spécifique

```
...
<h:inputText value="#{book.isbn}" validator="isbnValidator"/>

// ou

<h:inputText value="#{book.isbn}">
    <f:validator validatorId="isbnValidator" />
</h:inputText>
```

# Les validateurs (5/5) – Validateurs sur mesure (3/3)

Méthode 3 : validation  
par Java Bean

@AssertFalse	La valeur de l'attribut ou de la propriété doit être <i>false</i>	@AssertFalse boolean isUnsupported;
@AssertTrue	La valeur de l'attribut ou de la propriété doit être <i>true</i>	@AssertTrue boolean isActive;
@DecimalMax	La valeur de l'attribut ou de la propriété doit être un décimal inférieur ou égal à la valeur annotée	@DecimalMax("30.00") BigDecimal discount;
@DecimalMin	La valeur de l'attribut ou de la propriété doit être décimal supérieur ou égal à la valeur annotée	@DecimalMin("5.00") BigDecimal discount;
@Digits	La valeur de l'attribut ou de la propriété doit avoir un n chiffres avant la virgule et m chiffres après	@Digits(integer=6, fraction=2) BigDecimal price;
@Future	La valeur de l'attribut ou de la propriété doit être une date dans le futur	@Future Date eventDate;
@Max	La valeur de l'attribut ou de la propriété doit être un entier inférieur ou égal à la valeur annotée	@Max(10) int quantity;
@Min	La valeur de l'attribut ou de la propriété doit être un entier supérieur ou égal à la valeur annotée	@Min(5) int quantity;
@NotNull	La valeur de l'attribut ou de la propriété ne doit pas être <i>null</i>	@NotNull String username;
@Null	La valeur de l'attribut ou de la propriété doit être <i>null</i>	@Null String unusedString;
@Past	La valeur de l'attribut ou de la propriété doit être une date dans le passé	@Past Date birthday;
@Pattern	La valeur de l'attribut ou de la propriété doit matcher avec le motif	@Pattern(regexp="^\\((?\\d{3})\\)?[- ]?(\\d{3})[- ]?(\\d{4})\$") protected String mobilePhone; // Format (xxx) xxx xxxx
@Size	La valeur de l'attribut ou de la propriété doit avoir une taille comprise dans l'intervalle donné	@Size(min=2, max=240) String briefMessage;

# Plan du cours

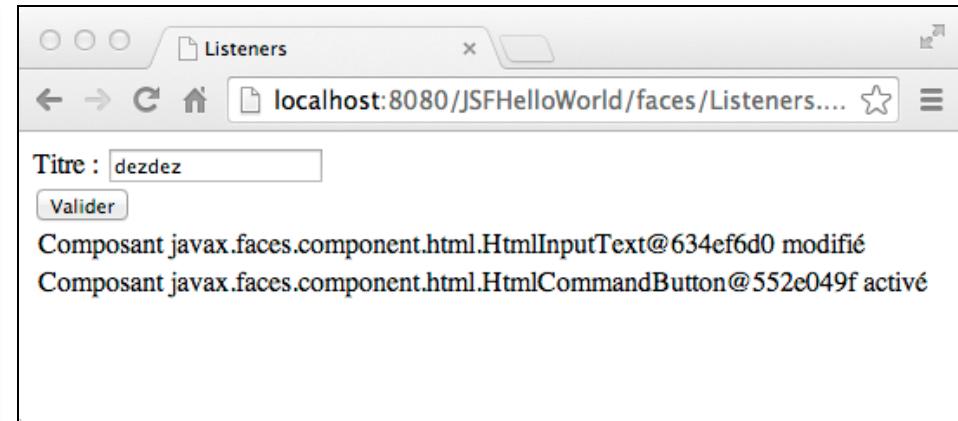
- Communication par composant distribué :
  - Application Web JSF 2.2
    - ✓ Des Servlets à JSP à JSF
    - ✓ Principes de JSF
    - ✓ Cycle de vie d'une JSF
    - ✓ Les expressions UEL
    - ✓ Les *managed beans* dans JSF
    - ✓ Composants standards JSF
    - ✓ Navigation entre pages et flots de pages JSF
    - ✓ Convertisseurs et validateurs de composants
    - **Listeners de composants**
    - Les fichiers ressources
    - HTML5, JavaScript et Ajax
    - Les templates et les composites
    - Les fichiers de déploiement et de configuration

# Listeners de composants (1/2)

- On s'intéresse ici aux listeners d'événements liés à l'application :
  - Les *action events* (`javax.faces.event.ActionEvent`) : les événements générés par un composant activé (qui implémente l'interface `ActionSource`)
  - Les *value-change events* (`javax.faces.event.ValueChangeEvent`) : les événements générés par un composant dont la valeur a changé après validation (implémente l'interface `EditValueHolder`)
- Les événements sont traités (cf. cycle de vie d'une requête) :
  - Pour les *action events* : en phase 5
  - Pour les *value-change events* : en phase 3
  - Sauf dans le cas où l'attribut *immediate* vaut *true* : l'événement est traité en phase 2
- Les objets événements permettent d'accéder au composant à la source de l'événement
- Il y a deux méthodes pour enregistrer un listener :
  - Soit définir une classe Java spécifique et la liée à un composant en utilisant une balise incluse `<f:valueChangeListener>` ou `<f:actionListener>`
  - Soit en implémentant une méthode d'un *managed bean* et la lier au composant avec son attribut `valueChangeListener` ou son attribut `actionListener`

# Listeners de composants (2/2)

```
...  
<h:form>  
    Titre :  
    <h:inputText value="#{myBean.article.titre}"  
                 valueChangeListener="#{myBean.titreListener}"/>  
    <br/>  
    <h:commandButton value="Valider">  
        <f:actionListener type="ValiderListener"/>  
    </h:commandButton>  
</h:form>  
<h:messages layout="table" showDetail="true" showSummary="false"/>  
...
```



```
...  
public void titreListener(ValueChangeEvent event) throws AbortProcessingException {  
    if (event.getNewValue() != null) {  
        FacesContext context = FacesContext.getCurrentInstance(); // Accès au contexte de la requête  
        FacesMessage message = new FacesMessage(FacesMessage.SEVERITY_INFO, "Info", "Composant " + event.getComponent() + " modifié");  
        context.getExternalContext().getSessionMap().put("titre", event.getNewValue()); // Sauvegarde de la valeur dans la session  
        context.addMessage(event.getComponent().getClientId(), message); // Ajout du message dans la vue  
    }  
}
```

Méthode du Managed Bean  
MyBean

```
...  
public class ValiderListener implements ActionListener {  
    @Override  
    public void processAction(ActionEvent event) throws AbortProcessingException {  
        FacesContext context = FacesContext.getCurrentInstance(); // Accès au contexte de la requête  
        FacesMessage message = new FacesMessage(FacesMessage.SEVERITY_INFO, "Info", "Composant " + event.getComponent() + " activé");  
        context.addMessage(event.getComponent().getClientId(), message); // Ajout du message dans la vue  
    }  
}
```

Classe implémentant  
ActionListener

# Plan du cours

- Communication par composant distribué :
  - Application Web JSF 2.2
    - ✓ Des Servlets à JSP à JSF
    - ✓ Principes de JSF
    - ✓ Cycle de vie d'une JSF
    - ✓ Les expressions UEL
    - ✓ Les *managed beans* dans JSF
    - ✓ Composants standards JSF
    - ✓ Navigation entre pages et flots de pages JSF
    - ✓ Convertisseurs et validateurs de composants
    - ✓ Listeners de composants
    - Les fichiers ressources
  - HTML5, JavaScript et Ajax
  - Les templates et les composites
  - Les fichiers de déploiement et de configuration

# Les fichiers ressources (1/3)

- Dans JSF, une ressource est un élément utilisé côté client et considéré comme statique par le serveur (non exécuté par le serveur) :
  - Une image
  - Un fichier de style CSS
  - Un fichier de script JavaScript
- Une ressource peut être interne (même contexte que l'application) ou externe (autre endroit)
- Une ressource interne est un fichier situé dans un répertoire de l'application :
  - dans le répertoire *resources/* placé soit avec les pages xHTML soit dans le répertoire *META-INF/*
  - avec un identifiant unique (= chemin complet)

resources / <identifiant\_ressource>

ou

META-INF / resources / <identifiant\_ressource>

- L'identifiant est de la forme :

[ locale / ] [ nomBibliothèque / ] [ versionBibliothèque / ] nomRessource [ / versionRessource ]

- Exemples d'identifiants :

book1.gif  
en/book2.gif  
en\_us/book3.gif  
en/myLibrary/book4.gif  
myLibrary/book5.gif  
myLibrary/1\_0/book6.gif  
myLibrary/1\_0/book7.gif/2\_3.gif

# Les fichiers ressources (2/3)

- Accès aux ressources :
  - La ressource correspondant à la locale du client est automatiquement choisie selon les informations transmises par la requête
  - Les expressions EL sont possibles dans les fichiers CSS et dans les scripts JavaScript écrits dans une page JSF
  - Pour les fichiers CSS et les fichiers JavaScript, il est possible d'indiquer l'endroit (attribut *target ::= head | body | form*) de la vue où ils seront insérés (rendu par le composant)
- Exemples :

```
<h:graphicImage value="book2.gif" />
<h:graphicImage value="book4.gif" library="myLibrary" />
<h:graphicImage value="#{resource['book2.gif']}' />
<h:graphicImage value="#{resource['myLibrary:book4.gif']}' />
```

Ici, on considère que la locale est en

Il est nécessaire d'indiquer le nom de la bibliothèque désirée

```
<h:outputStylesheet library="css" name="default.css"/>
```

Chargement d'une feuille de style CSS

```
<h:form>
  <h:outputScript name="myscript.js" library="mylibrary" target="head"/>
</h:form>
```

Chargement d'un fichier de script de type JavaScript rendu dans l'en-tête

# Les fichiers ressources (3/3) – Les textes localisés

- Les textes localisés sont des ressources pour l'application mais côté serveur
- Ils sont définis dans des fichiers *properties* et déclarés dans un fichier de configuration
- Ils sont utilisés dans des expressions EL ou les messages d'erreurs personnalisés
- Ils sont aussi accessibles dans le code Java (*java.util.ResourceBundle*)
- Ils sont constitués de lignes de la forme **clé=valeur** de sorte qu'une même clé correspond à plusieurs valeurs localisées de plusieurs fichiers ressources
- Au développement, ces fichiers sont dans les sources (*src/*)
- Au déploiement, ces fichiers sont placés dans la partie privée de l'application (*WEB-INF/classes*)

```
...  
<application>  
  <resource-bundle>  
    <base-name>messages.Messages</base-name>  
    <var>bundle</var>  
  </resource-bundle>  
  <locale-config>  
    <default-locale>en</default-locale>  
    <supported-locale>fr</supported-locale>  
  </locale-config>  
</application>  
...
```

Deux locales supportées

Détail du fichier de configuration  
*faces-config.xml*

messages/Messages.properties

```
Login=Login :  
Password=Password :  
LoginRequiredMessage=Login required  
Validate=Validate
```

messages/Messages\_fr.properties

```
Login=Nom :  
Password=Mot de passe :  
LoginRequiredMessage=Nom nécessaire  
Validate=Valider
```

Utilisation des textes localisés

```
...  
<h:outputText value="#{bundle.Login}" />  
<h:inputText value="#{myBean.login}"  
  required="true"  
  requiredMessage="#{bundle.LoginRequiredMessage}" />  
<h:commandButton value="#{bundle.Validate}" />  
...
```

# Plan du cours

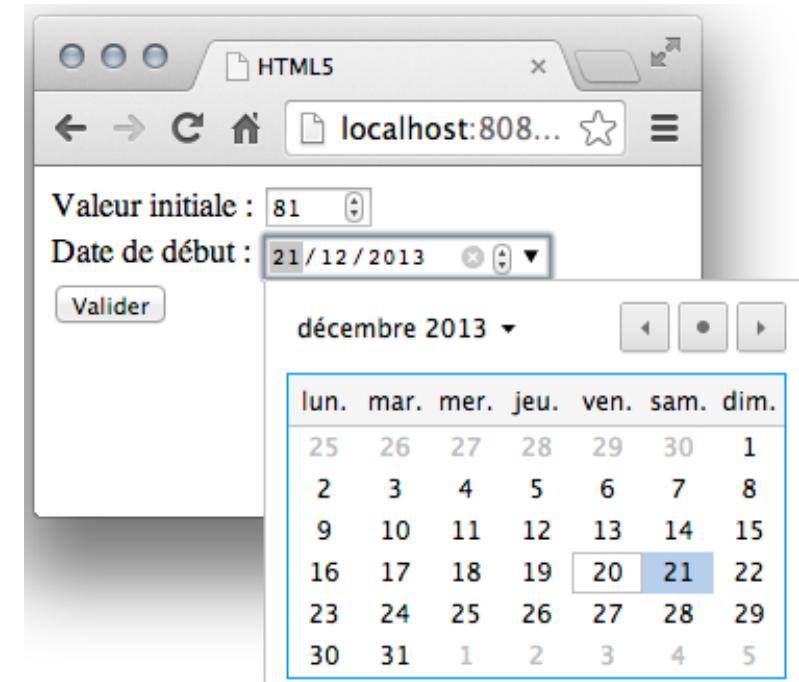
- Communication par composant distribué :
  - Application Web JSF 2.2
    - ✓ Des Servlets à JSP à JSF
    - ✓ Principes de JSF
    - ✓ Cycle de vie d'une JSF
    - ✓ Les expressions UEL
    - ✓ Les *managed beans* dans JSF
    - ✓ Composants standards JSF
    - ✓ Navigation entre pages et flots de pages JSF
    - ✓ Convertisseurs et validateurs de composants
    - ✓ Listeners de composants
    - ✓ Les fichiers ressources
    - HTML5, JavaScript et Ajax
    - Les templates et les composites
    - Les fichiers de déploiement et de configuration

# HTML5 et JSF (1/3)

- HTML5 est intégré de deux manières :
  - Possibilité d'utiliser des attributs HTML5 dans des composants JSF
  - Prise en charge des balises HTML5 par le moteur de servlet en traitant ces balises comme des composants JSF
- Exemple :

```
<?xml version='1.0'encoding='UTF-8'?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:jsf="http://xmlns.jcp.org/jsf"
      xmlns:p="http://xmlns.jcp.org/jsf/passthrough">
  <h:head>
    <title>HTML5</title>
  </h:head>
  <h:body>
    <h:form>
      <h:outputLabel value="Valeur initiale : "/>
      <h:inputText id="price" p:type="number"
                   value="#{myBean.count}"
                   p:min="80" p:max="120" p:step="20"
                   p:title="Donnez une valeur : 80, 100 ou 120."/>
      <br/>
      <h:outputLabel value="Date de début : "/>
      <input jsf:id="date" type="date" value="#{myBean.date}"/>
      <br/>
      <h:commandButton value="Valider"/>
    </h:form>
  </h:body>
</html>
```

*Attributs HTML5  
dans un  
composant JSF*



*Balise HTML5  
traitée comme un  
composant JSF*

# HTML5 et JSF (2/3) – Intégration HTML5 → JSF

Balise HTML5	Attribut pour sélectionner le composant JSF utilisé en équivalence	Composant JSF équivalent
a	jsf:action jsf:actionListener	h:commandLink
	jsf:value	h:outputLink
	jsf:outcome	h:link
body		h:body
button		h:commandButton
	jsf:outcome	h:button
form		h:form
head		h:head
img		h:graphicImage
input	type="button"	h:commandButton
	type="checkbox"	h:selectBooleanCheckbox
	type="color"   "date"   "datetime"   "datetime-local"   "email"   "month"   "number"   "range"   "search"   "time"   "url"   "week"	h:inputText
	type="file"	h:inputFile
	type="hidden"	h:inputHidden
	type="password"	h:inputSecret
	type="reset"   "submit"	h:commandButton
	type="**"	h:inputText
label		h:outputLabel
link		h:outputStylesheet
script		h:outputScript
select	multiple="**"	h:selectManyListbox
		h:selectOneListbox
textarea		h:inputTextArea

# HTML5 et JSF (3/3) – Intégration JSF → HTML5

- Le principe est d'accéder aux attributs HTML5 dans un composant JSF sans traitement de ces attributs par le moteur de servlet
  - 1<sup>ère</sup> manière : utiliser la librairie *jsf/passthrough*
  - 2<sup>ème</sup> manière : utiliser la balise *<f:passThroughAttribute>* pour ajouter un attribut
  - 3<sup>ème</sup> manière : utiliser la balise *<f:passThroughAttributes>* pour ajouter plusieurs attributs

...  
<html ...  
  xmlns:p="http://xmlns.jcp.org/jsf/passthrough">  
...  
  <h:inputText id="nights" p:type="number" value="#{bean.nights}"  
    p:min="1" p:max="30" p:required="required"  
    p:title="Enter a number between 1 and 30 inclusive." />  
...  
</html>

Attributs HTML5 dans  
un composant JSF

1<sup>ère</sup> manière

...  
  <h:inputText value="#{user.email}">  
    <f:passThroughAttribute name="type" value="email" />  
  </h:inputText>  
...

2<sup>ème</sup> manière  
name : nom de l'attribut HTML5  
value : valeur de l'attribut HTML5

...  
  <h:inputText value="#{bean.nights}">  
    <f:passThroughAttributes value="#{bean.nameValuePairs}" />  
  </h:inputText>  
...

3<sup>ème</sup> manière

```
...  
private Map<String, Object> nameValuePairs;  
...  
public Bean() {  
    this.nameValuePairs = new HashMap<>();  
    this.nameValuePairs.put("type", "number");  
    this.nameValuePairs.put("min", "1");  
    this.nameValuePairs.put("max", "30");  
    this.nameValuePairs.put("required", "required");  
    this.nameValuePairs.put("title", "Enter a number between 1 and 4 inclusive.");  
}  
...
```

# JavaScript et JSF (1 / 2)

- Le code JavaScript côté client permet une plus grande interactivité grâce à la manipulation dynamique du DOM (*Document Object Model*)
- Il est possible d'incorporer du code JavaScript :
  - Soit avec la balise <script> de xHTML
  - Soit avec la balise <h:outputScript> de Facelets
- Les fonctions JavaScript incorporées sont référencées avec l'un des attributs suivants : *onblur*, *onchange*, *onclick*, *ondblclick*, *onfocus*, *onkeydown*, *onkeypress*, *onkeyup*, *onmousedown*, *onmousemove*, *onmouseout*, *onmouseover*, *onmouseup*, *onselect*
- Attention, car les identifiants sont modifiés au moment du rendu pour intégrer l'identifiant du formulaire

# JavaScript et JSF (2/2) – Exemple

```
...  
<h:head>  
    <title>JavaScript</title>  
    <script type="text/javascript">  
        function testValue() {  
            var idform = document.forms[0].id; // L'identifiant du formulaire  
            var label = document.getElementById(idform + ":" + "#{bundle.idTextLogin}");  
            if (label.value == "") {  
                document.getElementById(idform + ":" + "idErreur").style.display = "inline";  
            }  
        }  
    </script>  
</h:head>  
<h:body>  
    <h:form>  
        <h:outputScript name="monscript2.js" library="script" target="head"/>  
        <h:outputLabel value="Nom :"/>  
        <h:inputText id="#{bundle.idTextLogin}" value="#{myBean.login}"  
                    onmouseover="JavaScript:testValue();"/>  
        <h:outputLabel id="idErreur" value="Nom obligatoire" style="display: none; color: Red;">  
        <br /><h:commandButton value="#{bundle.Validate}" />  
    </h:form>  
</h:body>  
...
```

Code JavaScript incorporé  
Possibilité d'utiliser des expressions EL

L'identifiant rendu incorpore  
l'identifiant du formulaire

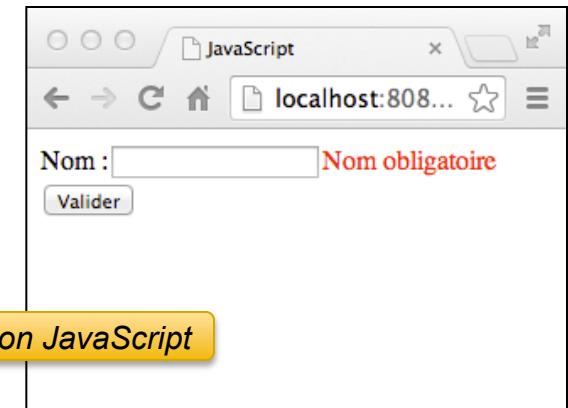
Code JavaScript incorporé  
comme ressource

Référence à une fonction JavaScript

Ce label est inclus dans le rendu  
mais ne sera pas affiché au départ

```
...  
<form id="j_idt5" name="j_idt5" method="post" action="/JSFHelloWorld/faces/JavaScript.xhtml">  
    <label>Nom :</label>  
    ...  
    <input id="j_idt5:login" type="text" name="j_idt5:login" value="" onmouseover="JavaScript:testValue();"/>  
    <label id="j_idt5:idErreur" style="display:none; color: Red;">Nom obligatoire</label>  
    <br /><input type="submit" name="j_idt5:j_idt9" value="Valider" />  
    ...  
</form>  
...
```

Détail du rendu de la page JSF



# Ajax et JSF (1/2)

- Ajax (*Asynchronous JavaScript and XML*) est l'utilisation de l'objet XMLHttpRequest de JavaScript qui permet de programmer des requêtes (synchrone ou asynchrone) au serveur et mettre à jour le DOM selon la réponse
- Avec JSF, une requête Ajax est asynchrone. Elle suit le même cycle de vie qu'une requête normale mais le traitement est restreint à certains composants à actualiser et à rendre
- Ajax est pris en charge avec la bibliothèque jsf.js :
  - Elle est incorporée automatiquement quand on utilise la balise `<f:ajax>`. Cette balise peut être incluse dans un composant ou inclure un ou plusieurs composants JSF (elle s'applique à chacun de ces composants).

Balise `<f:ajax>` incluse dans `<h:commandButton>`  
La requête est déclenchée uniquement par ce composant sur l'événement onclick

```
<h:commandButton id="submit" value="Submit">  
  <f:ajax event="click" execute="userNo" render="result"/>  
</h:commandButton>
```

```
<f:ajax event="click" render="@all">  
  <h:form>  
    <h:inputText value="#{user.name}" />  
    <h:commandButton>  
      <f:ajax event="mouseover"/>  
    </h:commandButton>  
  </h:form>  
</f:ajax>
```

Balise `<f:ajax>` qui inclut plusieurs composants  
La requête est déclenchée pour chacun d'eux sur l'événement onclick sauf pour le bouton (événement onmouseover)

- Elle doit être explicitement incorporée avec le composant `<h:outputScript name="jsf.js" library="javax.faces"/>` quand on utilise la fonction `jsf.ajax.request` (non abordé dans ce cours !)

# Ajax et JSF (2/2) – Balise <f:ajax> et ses attributs

*event* : Type de l'événement déclencheur de la requête Ajax

Par défaut vaut *action* pour les composants qui implémentent ActionSource

Par défaut vaut *valueChange* pour les composants qui implémentent EditableValueHolder

Les valeurs possibles sont : blur, change, click, dblclick, focus, etc. (événements JavaScript habituels sans le *on*)

*execute* : Liste des identifiants des composants à exécuter par le serveur JSF (@this par défaut).

@all : tous les composants

@form : les composants du formulaire

@none : aucun composant

@this : le composant déclencheur

*render* : Liste des identifiants de composants à rendre (@none par défaut)

*listener* : Nom de la méthode d'un listener d'événement de type javax.faces.event.AjaxBehaviorEvent appelée en phase 5 du cycle de vie.

*onevent* : Nom de la fonction JavaScript qui gère l'état et la valeur de la réponse à la requête en cours.

La fonction est appelée 3 fois selon que la requête commence, finie et est un succès. (voir documentation pour plus de détail !)

*onerror* : Nom de la fonction JavaScript qui gère les erreurs de la requête en cours. (voir documentation pour plus de détail !)

*Un listener est exécuté par JSF à chaque changement de valeur du composant.  
Il ajoute un message d'erreur si nécessaire.*

```
...  
<h:form>  
    <h:outputLabel value="Nouveau titre : "/>  
    <h:inputText valueChangeListener="#{myBean.titreListener2}"  
        id="idTitreArticle" value="#{myBean.article.titre}">  
        <f:ajax event="change" execute="idTitreArticle" render="idMessage"/>  
    </h:inputText>  
    <h:message id="idMessage" for="idTitreArticle"  
        showDetail="true" showSummary="false"  
        style="color: red;"/>  
    <br/>  
    <h:outputLabel value="Nouveau contenu : "/><br/>  
    <h:inputTextarea value="#{myBean.article.contenu}" /><br/>  
    <h:commandButton value="Valider"/>  
</h:form>  
...
```

*Une requête Ajax est envoyée dès que la valeur est modifiée (ce qui appelle le listener).*



# Plan du cours

- Communication par composant distribué :
  - Application Web JSF 2.2
    - ✓ Des Servlets à JSP à JSF
    - ✓ Principes de JSF
    - ✓ Cycle de vie d'une JSF
    - ✓ Les expressions UEL
    - ✓ Les *managed beans* dans JSF
    - ✓ Composants standards JSF
    - ✓ Navigation entre pages et flots de pages JSF
    - ✓ Convertisseurs et validateurs de composants
    - ✓ Listeners de composants
    - ✓ Les fichiers ressources
    - ✓ HTML5, JavaScript et Ajax
    - Les templates et les composites
    - Les fichiers de déploiement et de configuration

# Présentation (très) rapide des templates (1/2)

- Les balises de *templating* permettent une réutilisation de contenu et une standardisation des pages d'une même application
- Par exemple, un template est une page JSF servant de modèle pour d'autres pages calquées sur la structure définie par le template
- Par exemple, une composition permet de structurer plusieurs balises ou d'appliquer un template mais ne définit pas un véritable composant (pas de convertisseur / validateur / listener attachés à la composition)

<ui:insert>	Définit un point d'insertion dans un template dans lequel on pourra ensuite insérer un contenu placé dans un marqueur <ui:define>
<ui:composition>	Création d'un "composant" utilisant éventuellement un template. Le contenu extérieur est ignoré. Un même template peut être utilisé par plusieurs compositions
<ui:define>	Définit un contenu qui sera inséré dans l'élément <ui:insert> correspondant du template
<ui:decorate>	Idem que <ui:composition> mais garde le contenu extérieur
<ui:component>	Idem que <ui:composition> mais ajout d'un <i>UIComponent</i> à la vue
<ui:include>	Inclusion d'un contenu commun à plusieurs pages de l'application
<ui:repeat>	Répétition (alternative à la balise <c:foreach> de JSTL)
<ui:fragment>	Idem que <ui:component> mais garde le contenu extérieur
<ui:debug>	Affiche l'arbre des composants à la vue pour débogage
<ui:param>	Passage de paramètres aux fichiers inclus
<ui:remove>	Suppression du contenu de la page

Voir documentation pour plus de détail !

# Présentation (très) rapide des templates (2/2)

## ■ Définition d'un template et son utilisation :

Fichier template layout.xhtml

```
...  
<html xmlns="http://www.w3.org/1999/xhtml"  
      xmlns:ui="http://java.sun.com/jsf/facelets"  
      xml:lang="fr" lang="fr">  
<head>  
  <title>  
    <ui:insert name="title">Default title</ui:insert>  
  </title>  
</head>  
<body>  
  <h1><ui:insert name="title">Default title</ui:insert></h1>  
  <hr/>  
  <ui:insert name="content">Default content</ui:insert>  
  <hr/>  
</body>  
</html>
```

Utilisation du template

```
...  
<html xmlns="http://www.w3.org/1999/xhtml"  
      xmlns:ui="http://java.sun.com/jsf/facelets"  
      xml:lang="fr" lang="fr">  
<ui:composition template="layout.xhtml">  
  <ui:define name="title">Liste des articles</ui:define>  
  <ui:define name="content">  
    <h:dataTable value="#{myBean.articles}" var="art">  
      <h:column>  
        <f:facet name="header">  
          <h:outputText value="Titre"/>  
        </f:facet>  
        <h:outputText value="#{art.titre}"/>  
      </h:column>  
      <h:column>  
        <f:facet name="header">  
          <h:outputText value="Résumé"/>  
        </f:facet>  
        <h:outputText value="#{art.contenu}"/>  
      </h:column>  
    </ui:define>  
</ui:composition>  
</html>
```

Les balises `<ui:insert>` définissent les parties ajustables du template.

Les autres balises ne peuvent pas être modifiées et sont communes

L'utilisation du template permet de populer (de manière optionnelle) les parties ajustables avec des balises `<ui:define>` qui répondent aux balises `<ui:insert>`

# Présentation (très) rapide des composites (1 / 2)

- Un composite est la définition d'un composant JSF sur mesure et réutilisable sans besoin d'écrire du code Java
- Un composite est un véritable composant JSF avec support des convertisseurs, des validateurs et des listeners
- Un composite est un fichier xHTML stocké comme une ressource : `resources/nom_bibliothèque/nom_composite.xhtml`

<code>&lt;composite:interface&gt;</code>	Déclaration de l'interface publique (le "contrat") d'un composite
<code>&lt;composite:implementation&gt;</code>	Définition de l'implémentation d'un composite
<code>&lt;composite:attribute&gt;</code>	Déclaration d'un attribut pouvant être fourni à une instance du composite. Une balise <code>&lt;composite:interface&gt;</code> peut contenir plusieurs attributs.
<code>&lt;composite:facet&gt;</code>	Déclaration de la possibilité d'utiliser une <i>facet</i> (personnalisation du composite) Une balise <code>&lt;composite:interface&gt;</code> peut contenir plusieurs <i>facets</i> .
<code>&lt;composite:insertFacet&gt;</code>	Balise utilisée dans la partie implémentation. La <i>facet</i> insérée sera représentée dans le composite.
<code>&lt;composite:insertChildren&gt;</code>	Balise utilisée dans la partie implémentation. Tous les composants fils ou les templates seront gardés et insérés dans le composite.
<code>&lt;composite:valueHolder&gt;</code>	Balise utilisée dans la partie interface. Le composite implémente l'interface <i>ValueHolder</i> .
<code>&lt;composite:editableValueHolder&gt;</code>	Balise utilisée dans la partie interface. Le composite implémente l'interface <i>EditableValueHolder</i> .
<code>&lt;composite:actionSource&gt;</code>	Balise utilisée dans la partie interface. Le composite implémente l'interface <i>ActionSource</i> .

# Présentation (très) rapide des composites (2/2)

Fichier resources/cpste/modifyarticle.xhtml

Le composite porte le nom du fichier

Il utilise la bibliothèque composite de JSF

La partie interface déclare deux attributs obligatoires et une source d'un événement de type ActionEvent

```
...  
<html xmlns="http://www.w3.org/1999/xhtml"  
      xmlns:h="http://xmlns.jcp.org/jsf/html"  
      xmlns:composite="http://xmlns.jcp.org/jsf/composite">  
  
<composite:interface>  
    <composite:attribute name="title" required="true"/>  
    <composite:attribute name="content" required="true"/>  
    <composite:actionSource name="button" targets="form:bton"/>  
</composite:interface>  
  
<composite:implementation>  
    <h1>Modification d'un article</h1>  
    <h:form id="form">  
        <h:panelGrid columns="2">  
            <h:outputLabel value="Titre :"/>  
            <h:inputText value="#{cc.attrs.title}" />  
            <h:outputLabel value="Contenu :"/>  
            <h:inputTextarea value="#{cc.attrs.content}" />  
            <h:commandButton id="bton" value="Valider"/>  
        </h:panelGrid>  
    </h:form>  
</composite:implementation>  
  
</html>
```

La partie implémentation définit le composite :

- Les différentes balises JSF ou HTML
- Les attributs utilisées via l'objet cc (compositeComponent) et sa propriété attrs

Fichier utilisant le composite modifyarticle

Il utilise la bibliothèque cpste

L'utilisation du composite impose de donner les valeurs de ses deux attributs. On donne le nom d'une classe listener pour la source button (attribut for)

```
...  
<html xmlns="http://www.w3.org/1999/xhtml"  
      xmlns:h="http://xmlns.jcp.org/jsf/html"  
      xmlns:ma="http://xmlns.jcp.org/jsf/composite/cpste"  
      xmlns:f="http://xmlns.jcp.org/jsf/core">  
  
    <h:body>  
        <ma:modifyarticle title="#{myBean.articles[0].titre}"  
                           content="#{myBean.articles[0].contenu}">  
            <f:actionListener for="button" type="ValiderListener"/>  
        </ma:modifyarticle>  
        <h:messages showDetail="true" showSummary="false"/>  
    </h:body>  
</html>
```



# Plan du cours

- Communication par composant distribué :
  - Application Web JSF 2.2
    - ✓ Des Servlets à JSP à JSF
    - ✓ Principes de JSF
    - ✓ Cycle de vie d'une JSF
    - ✓ Les expressions UEL
    - ✓ Les *managed beans* dans JSF
    - ✓ Composants standards JSF
    - ✓ Navigation entre pages et flots de pages JSF
    - ✓ Convertisseurs et validateurs de composants
    - ✓ Listeners de composants
    - ✓ Les fichiers ressources
    - ✓ HTML5, JavaScript et Ajax
    - ✓ Les templates et les composites
    - Les fichiers de déploiement et de configuration

# Fichier de configuration de JSF

- La configuration d'une application peut être indiquée par un ou plusieurs fichiers de configuration
- Les fichiers de configurations sont nécessaire dans deux cas :
  - Pour spécifier des éléments tels que les ressources de textes localisés ou des règles de navigation
  - Pour surcharger les annotations des *managed beans* pour le déploiement de l'application (les fichiers ont priorités sur les annotations)
- Les fichiers peuvent se trouver dans les endroits suivants :
  - Le répertoire /WEB-INF de l'application
  - Le répertoire /META-INF de n'importe quel fichier jar se trouvant dans /WEB-INF/lib
  - Les répertoires indiqués par le paramètre javax.faces.application.CONFIG\_FILES du fichier de déploiement
- Pour démarrer une application, l'objet *FacesServlet* crée une instance unique de *javax.faces.application.Application* et la configure avec les informations des fichiers de configuration trouvés
- Format :
  - Fichier XML contenant la balise racine *<faces-config>*
  - Le nom du fichier doit finir par : *faces-config.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<faces-config version="2.2"
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd">
...
</faces-config>
```

*Fichier main-faces-config.xml (JSF 2.2)*

# Packaging et fichier de déploiement et (1 / 2)

- Pour être facilement déployée, une application JSF peut être compressée dans une fichier WAR (format jar avec l'extension .war)
- Une application JSF doit obligatoirement contenir un fichier de déploiement nommé web.xml (format XML)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1"
  xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
...
</web-app>
```

Fichier web.xml (Servlet 3.1)

- Exemple de structure d'un fichier WAR :

/	documents xHTML ou HTML
/resources	
/image	→ répertoire des images
/script	→ répertoire des documents JavaScript
/style	→ répertoire des documents CSS
/comp	→ répertoire des composants composites
/WEB-INF	
/web.xml	→ fichier de déploiement (obligatoire)
/faces-config.xml	→ fichier de configuration (optionnel)
/glassfish-web.xml	→ fichier de configuration spécifique au serveur d'application GlassFish
/lib	→ répertoire des fichiers jar (autres composants, autres classes Java, etc.)
/classes	→ répertoire des fichiers JavaBeans et autres fichiers ou classes utilitaires

# Packaging et fichier de déploiement et (2/2)

- Le fichier de déploiement doit inclure les spécifications suivantes :
  - La classe servlet de traitement des requêtes d'une page JSF (généralement c'est *FacesServlet* mais on peut définir sa propre servlet)
  - Le chemin des pages traitées par la servlet ( *servlet mapping*)
  - Le chemin du ou des fichiers de configuration s'ils sont dans un endroit non-conventionnel
- Le fichier de déploiement peut inclure les spécifications suivantes :
  - Les pages d'entrée de l'application
  - Lieu de sauvegarde de l'état des composants : sur le serveur (par défaut) ou sur le client
  - Encryptage et compression des sauvegardes sur le client
  - Restriction d'accès aux pages JSF
  - Validation XML des documents
  - Spécification du statut de l'application (en développement, en test, en exploitation)

```
...  
  <servlet>  
    <servlet-name>FacesServlet</servlet-name>  
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>  
  </servlet>  
  <servlet-mapping>  
    <servlet-name>FacesServlet</servlet-name>  
    <url-pattern>/faces/*</url-pattern>  
  </servlet-mapping>  
  <welcome-file-list>  
    <welcome-file>JSFHelloWorld.xhtml</welcome-file>  
  </welcome-file-list>  
  <context-param>  
    <param-name>javax.faces.PROJECT_STAGE</param-name>  
    <param-value>Production</param-value>  
  </context-param>  
...
```

*La servlet de traitement des pages JSF*

*Chemin des pages JSF traitées par la servlet*

*Liste des pages d'entrée de l'application*

*Statut de l'application*