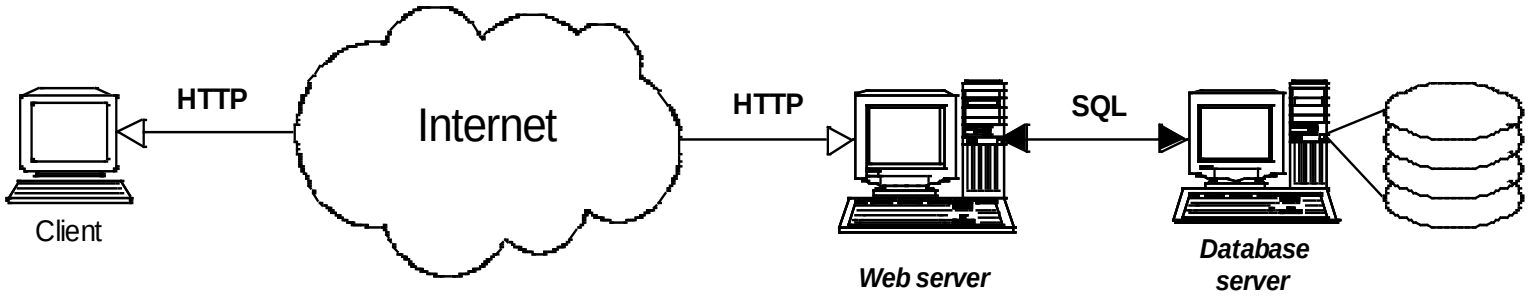
JEE : LE TIERS PRÉSENTATION

LE CONTEUR DE SERVLET

SERVEURS WEB À CONTENU DYNAMIQUE

- Architecture multi-tiers
  - un serveur Web
  - un serveur d'application
  - une base de données
- Motivations de J2EE
  - Faciliter la programmation d'application web



REQUÊTE À UN SERVEUR WEB

- Envoi de paramètre

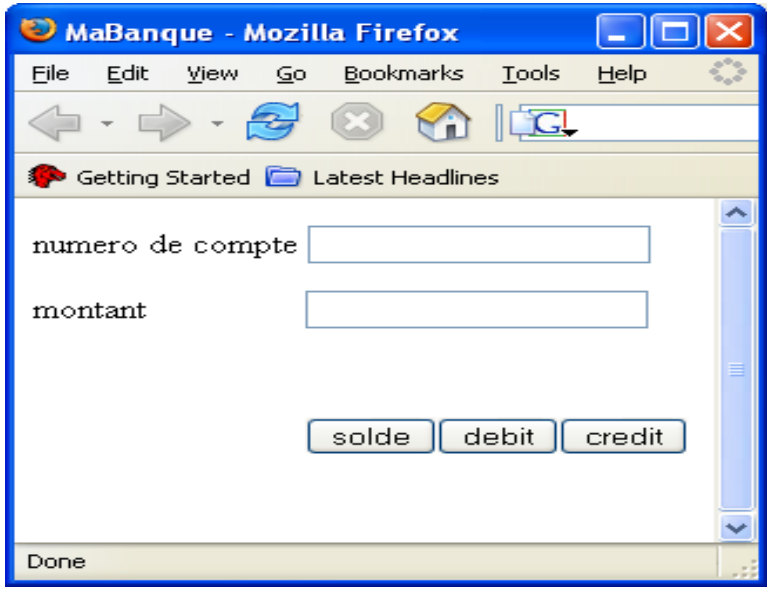
  champ1=valeur1&champ2=valeur2...

- Les requêtes GET et POST
  - GET : paramètres inclus dans l'URL

    http://nom_du_serveur/cgi-bin/script.cgi?champ1=valeur1&champ2=valeur2...
    - Limitation à 255 catactères, visible ...
  - POST : paramètres inclus dans les entêtes HTTP
  - Utilisation de formulaires (pour interactions)

- Reception des paramètres
  - GET : variable d'environnement QUERY_STRING
  - POST : STDIN, et variable d'environnement CONTENT_LENGTH

FORMULAIRE

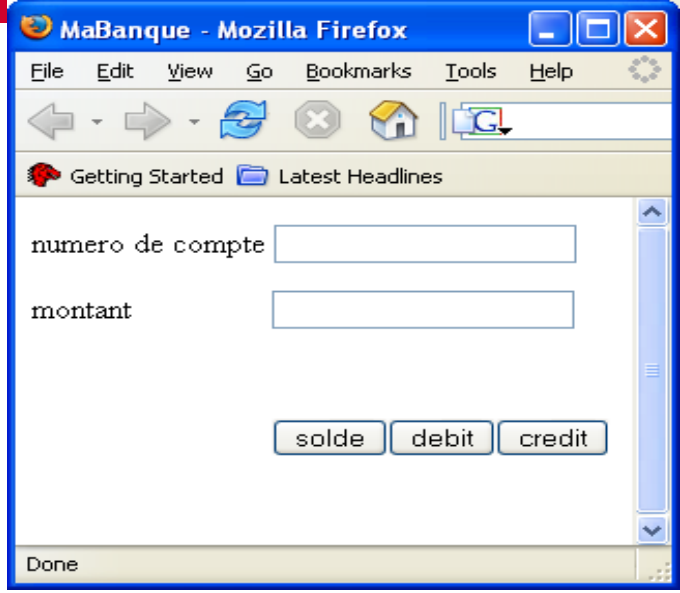



```
<html>
    <head><title>MaBanque</title></head>
    <body>
        <form method="post"  action="/servlet/BanqueAccount">
            <p>numero de compte<input type="text" name= "num"></p>
            <p>montant<input type="text" name= "val"></p>
            <p><input type="submit" name="operation" value="solde">
                <input type="submit" name="operation" value="debit">
                <input type="submit" name="operation" value="credit"></p>
        </form>
    </body>
</html>
```

LES SERVLETS

- Programme serveur écrit en java
  - S'exécute dans le même processus
- Container de servlet
  - Serveurs JAVA : Java Web Server ou Tomcat
    - Font aussi serveur web
  - Serveur Plug-in de Apache

EXEMPLE (1/3)




```
<html>
    <head><title>MaBanque</title></head>
    <body>
        <form method="post"  action="/servlet/BanqueAccount">
            <p>numero de compte<input type="text" name= "num"></p>
            <p>montant<input type="text" name= "val"></p>
            <p><input type="submit" name="operation" value="solde">
                <input type="submit" name="operation" value="debit">
                <input type="submit" name="operation" value="credit"></p>
        </form>
    </body>
</html>
```

EXEMPLE 2/3

```
public class BanqueAccount extends HttpServlet {
 Connection conn = DriverManager.getConnection(url, user, password);

 protected void doPost(HttpServletRequest req,
                       HttpServletResponse resp) throws ServletException, IOException {

   String op = req.getParameter("operation");
   int num = Integer.parseInt(req.getParameter("num"));
   int val = Integer.parseInt(req.getParameter("val"));
   resp.setContentType("text/html");                // type mime
   PrintWriter out = resp.getWriter();         // pour faire des println

    // recupere les informations du compte à partir de son numero
    String selectStatement = "SELECT * FROM account WHERE num = ?";
    PreparedStatement ps = conn.prepareStatement(selectStatement);
    ps.setInt(1, num);
    ResultSet res = ps.executeQuery();
```

EXEMPLE 3/3

```
if (res.next())
if (op.equals("debit")) {
    int ob = res.getInt("balance");
    String updateStatement = "UPDATE account SET balance=? WHERE num=?";
    ps = conn.prepareStatement(updateStatement);
    ps.setInt(1, ob-val);
    ps.setInt(2, num);
    ps.executeUpdate();
} else if (op.equals("credit")) …
…
    // renvoie une page dynamique de confirmation
    out.println("<html><body>");
    out.println("<p> " + res.getString("name") + "</p>");
    out.println("<p> Your account has been updated.</p>");
    out.println("</body></html>");
    out.close();
   }
}
```

SESSION

- Notion de session
  - Une requête dépend du résultat des requêtes précédentes
  - Ex : caddie
- Création de session
  - HttpSession **HttpServletRequest.getSession**();
  - HttpSession **HttpServletRequest.getSession**(boolean create);

HTTPSESSION

Object getAttribute(String name)

Enumeration getAttributeNames()

Long getCreationTime()

String getId()

int getMaxInactiveInterval()

void invalidate()

void removeAttribute(String name)

void setAttribute(String name, Object value)

void setMaxInactiveInterval(int interval)

…

PACKAGING D'UNE SERVLET

- Un répertoire par application
  - Pages web (html)
  - Répertoire "WEB-INF"
    - Répertoire "classes" : les classes des servlets
    - "web.xml" : descripteur des servlets

```
<web-app>
  <servlet>
    <servlet-name>myBanque</servlet-name>
    <servlet-class>BanqueAccount</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>myBanque</servlet-name>
    <url-pattern>/URL_Banque</url-pattern>
  </servlet-mapping>
```

Ou Annotation @WebServlet(“/URL_Banque/”) sur la classe de la servlet.

INSTALLATION DANS TOMCAT

- Création d'un fichier WAR (jar)
- Copie dans $CATALINA_HOME/webapps
- Le fichier WAR est expansé par le conteneur de servlet au moment du déploiement.

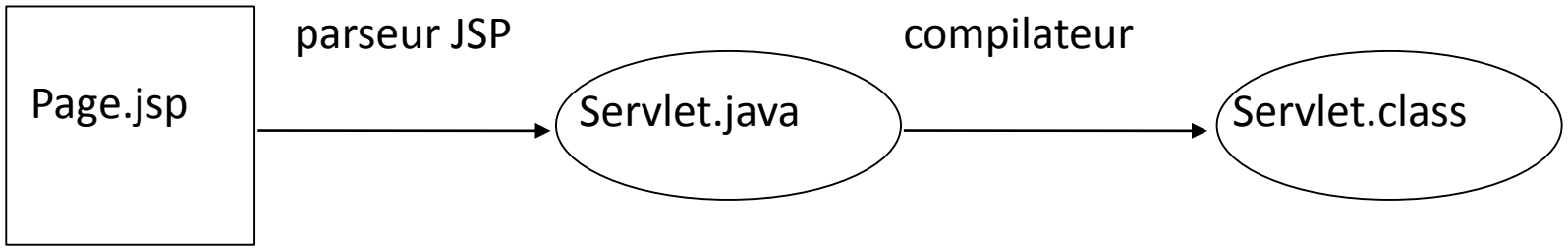
SERVLET
BILAN

- Facile à programmer
  - Programmation en Java et API simple
  - Mais pas aussi simple qu'un script comme PHP
- Mieux adapté que les scripts à des traitements plus complexes
  - Interface JDBC pour accès à différentes BD SQL
  - Extension plus facile (liaison dynamique de Java) que les extensions d'un langage de script
- Le traitement des données récupérées de la BD peuvent être lourds (d'où les EJBs)

LE CONTEUR WEB : LES JSP

JSP (JAVA SERVER PAGE)

- Langage de script (proche de Java)
  - Générer des pages dynamiques
  - Intégré dans des pages web
  - Compilé dynamiquement en servlet
- Intégration avec des classes Java



UN PETIT EXEMPLE

```
<%@ page language="Java" %>
<html>
<head>
<title>First.jsp</title>
</head>
<body>
<h1>Nombres de 1 à 10</h1>

<% for(int i=1; i<=10; i++) {
        out.println(i + "<br>");
    }
%>
</body>
</html>
```

LES SCRIPTS JAVA

- ***Du code Java : <% code Java %>***
- ***Des évaluations d'expression : <%= expression %>***
- ***Des variables prédéfinies***

```
<%@ page language="Java" %>
<html><head><title>First.jsp</title>
</head><body>
<h1>Nombres de 1 à 10</h1>
<% for(int i=1; i<=10; i++) {  %>
        <%= i %> <br/>
<%  } %>
</body>
</html>
```

Variables prédéfinies

HttpServletRequest request
HttpServletResponse response
HttpSession session
PrintWriter out
ServletConfig config
…

LES ACTIONS

- Permet de condenser des traitements qui seraient longs à écrire en Java
- **<jsp:usebean id="nomAttribut" class="package.classe" scope="portéeAttribut/>**
- Importe un attribut si il existe, le crée sinon

```
<jsp:usebean id="personne" class="testjsp.Personne" scope="session/>
```

équivaut à

```
<% testjsp.Personne personne = (testjsp.Personne)session.getAttribute("personne");
    if (personne == null) {
        personne = new testjsp.Personne();
        session.setAttribute("personne", personne);
    }
%>
```

LES ACTIONS

- **<jsp:setProperty name="nomAttr" property="nomProp"/>**
- S'utilise en complément de useBean
- La classe Java doit être un javaBean (set<AttName>(), get<AttName>())
- Initialise un javaBean à partir des paramètres de formulaire

```
<jsp:usebean id="personne" class="testjsp.Personne"
    scope="session/>
<jsp:setProperty name="personne" property="nom" />
```

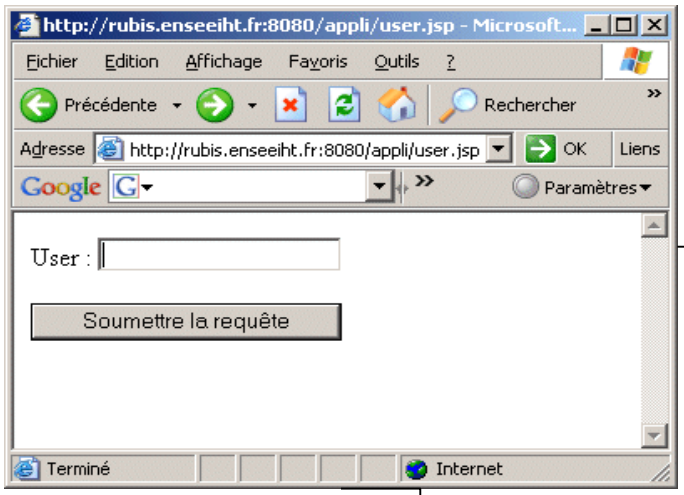
équivaut à

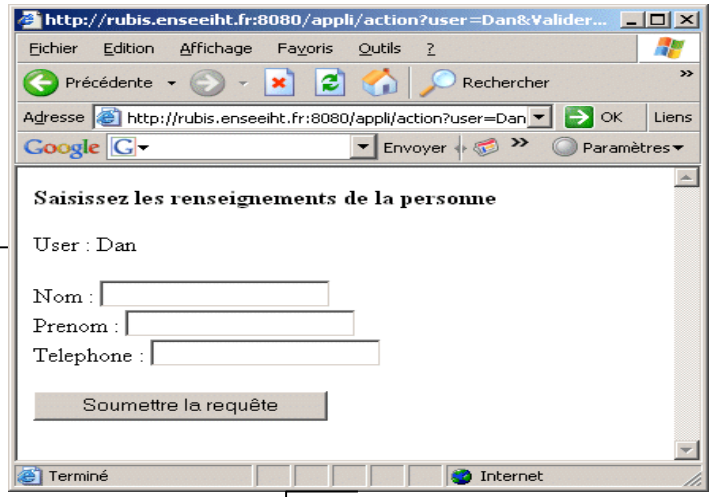
```
<jsp:usebean id="personne" class="testjsp.Personne"
    scope="session/>
<%  if (request.getParameter("nom") != null)
        personne.setNom(request.getParameter("nom"));
%>
```

LES ACTIONS

- **<jsp:forward page="page2.jsp" />**
  - Transfère le contrôle à une autre page JSP
  - Peut prendre des paramètres avec jsp:param

EXEMPLE : UN ANNUAIRE





EXEMPLE : ARCHITECTURE

- Une page JSP pour chaque écran
  - user.jsp, personne.jsp, listeuser.jsp
- Une servlet qui aiguille les requêtes vers les pages
- Deux objets Java (beans) pour gérer les données
  - Personne.java, ListePersonne.java

EXEMPLE : USER.JSP

```
<%@ page language="java" %>

<html>

<body>

<form action="action" method="post">
        User : <input type="text" name="user"/><br/><br/>
        <input type="submit" name="Valider"/>
        <input type="hidden" name="formulaire" value='user'/>

</form>

</body>

</html>
```

EXEMPLE : PERSONNE.JSP

```
<%@ page language="java" %>
<html>
<body>
<jsp:useBean id="user" class="mvc.Personne" scope="session"/>
<b> Saisissez les renseignements de la personne</b> <br/><br/>
User : <%= user.user %><br/>
<form action="action" method="post">
        Nom : <input type="text" name="nom"/><br/>
        Prenom : <input type="text" name="prenom"/><br/>
        Telephone : <input type="text" name="telephone"/><br/><br/>
        <input type="submit" name="Valider"/>
        <input type="hidden" name="formulaire" value="personne"/>
</form>
</body>
</html>
```

EXEMPLE : LISTEUSER.JSP

```
<%@ page language="java" import="java.util.*, mvc.* "%>

<html>

<body>

<b> Liste des users enregistres </b> <br/><br/>

<table border="2">

<th>User</th><th>Nom</th><th>Prenom</th><th>Telephone</th>

<jsp:useBean id="listeuser" class="mvc.ListePersonne" scope="application"/>

</table>
```

EXEMPLE : LISTEPERSONNE.JSP

```
<%
      for(Personne personne : listeuser.values()) {

%>
            <tr>
                  <td><%= personne.user %></td>
                  <td><%= personne.nom %></td>
                  <td><%= personne.prenom %></td>
                  <td><%= personne.telephone %></td>
            </tr>

<%  }  %>

</body>

</html>
```

EXEMPLE : ACTION.JAVA (SERVLET)

```
public class Action extends HttpServlet {

  protected void doPost(HttpServletRequest request, HttpServletResponse response)

                              throws ServletException, IOException {


    String formulaire = request.getParameter("formulaire");

    HttpSession session = request.getSession(); /* data de session */

    ServletContext context = getServletContext(); /*data independante de session */


    if (formulaire == null) {

      RequestDispatcher disp = request.getRequestDispatcher("user.jsp");

      disp.forward(request, response);

    }
```

EXEMPLE : ACTION.JAVA (SERVLET)

```
if (formulaire.equals("user")) {
        Personne personne = new Personne();
        personne.user = request.getParameter("user");
        session.setAttribute("user", personne);
        ListePersonne listeUser =  (ListePersonne)context.getAttribute("listeuser");
        if (listeUser == null) {
          listeUser = new ListePersonne();
          context.setAttribute("listeuser", listeUser);
         }
        if (listeUser.isRegistered(personne.user)) {
          RequestDispatcher disp = request.getRequestDispatcher("listeuser.jsp");
          disp.forward(request, response);
         } else {
           RequestDispatcher disp = request.getRequestDispatcher("personne.jsp");
           disp.forward(request, response);
         }
}
```

EXEMPLE : ACTION.JAVA (SERVLET)

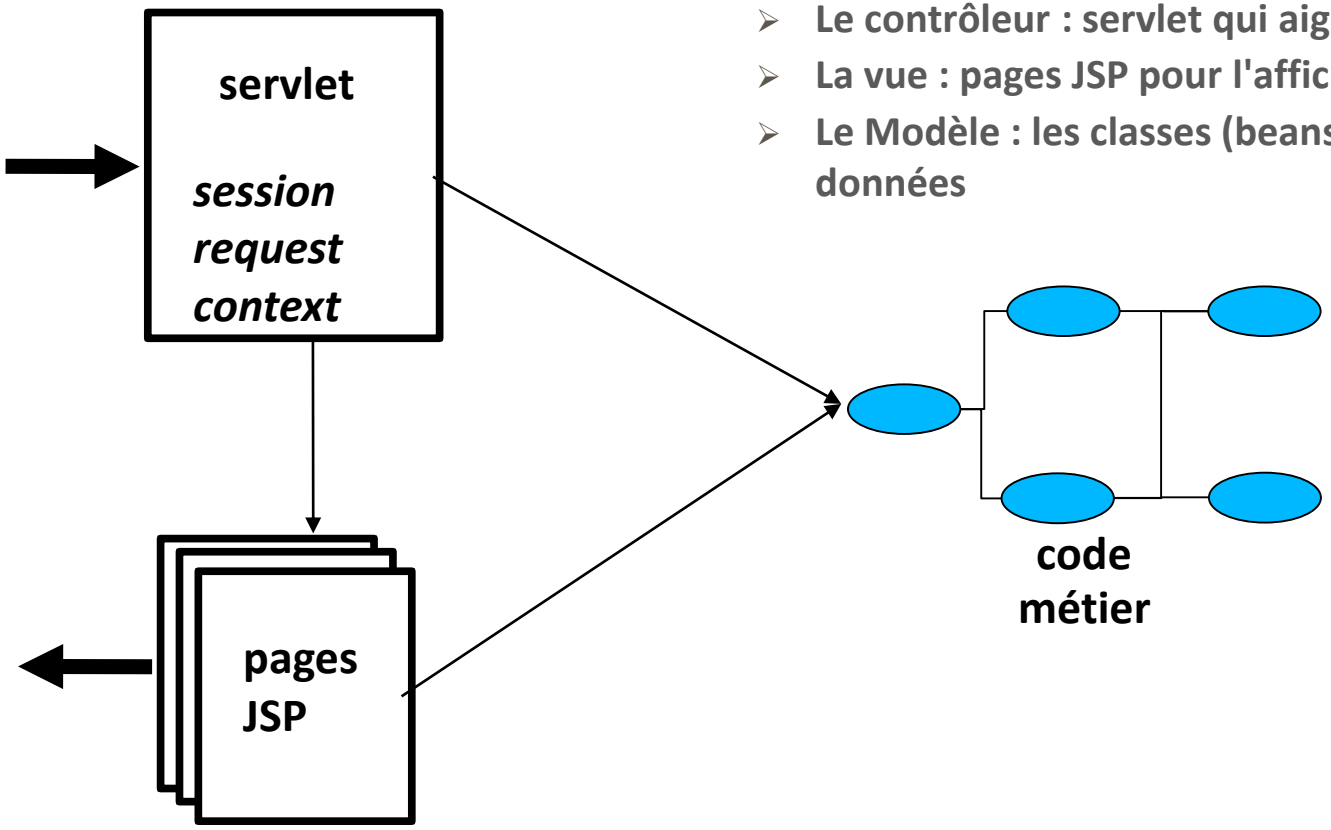
```
if (formulaire.equals("personne")) {
        Personne personne = (Personne)session.getAttribute("user");
        personne.nom = request.getParameter("nom");
        personne.prenom = request.getParameter("prenom");
        personne.telephone = request.getParameter("telephone");

        ListePersonne listeUser =  (ListePersonne)context.getAttribute("listeuser");
       listeUser.liste.put(personne.user, personne);

        RequestDispatcher disp = request.getRequestDispatcher("listeuser.jsp");
        disp.forward(request, response);
    }
  }
}
```

EXEMPLE : LES BEANS

```
public class Personne {
        public String user, nom, prenom, telephone;

}



public class ListePersonne {
        public Map liste = new HashMap();
        .......
        public boolean isRegistered(String user) {
            for(Personne personne : liste.values()) {
                if (personne.user.equals(user)) return true;
            }
            return false;
        }

}
```

MODÈLE MVC

- Model View Controler
- Séparation entre
  - Le contrôleur : servlet qui aiguille les requêtes
  - La vue : pages JSP pour l'affichage à l'écran
  - Le Modèle : les classes (beans) qui traitent les données



JSP - BILAN

- Présentation
  - Sous forme de pages HTML
  - Programmation en Java dans les pages
- Code métier
  - Sous forme de Servlet
  - Échange de données avec les pages sous forme de javaBeans
- Modèle MVC donne une séparation claire entre
  - Présentation (page JSP)
  - Contrôle (servlet)
  - Métier (programmes Java)
- Le traitement des données récupérées de la BD peuvent être lourds (d'où les EJBs)

JAVA SERVER FACES

*Réussir la transformation. Ensemble.

Delivering Transformation. Together.

sopra steria

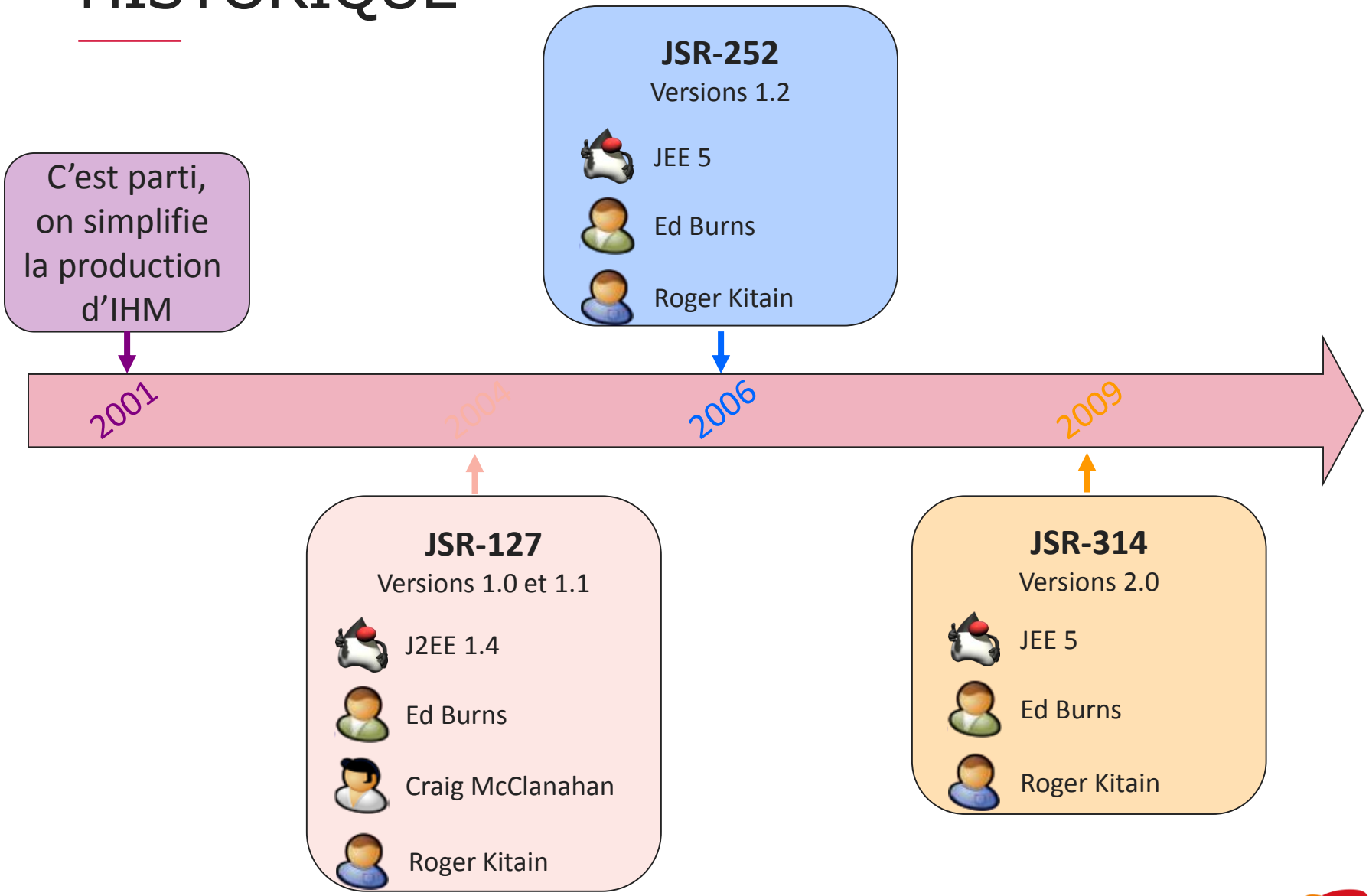
QU'EST-CE QUE JSF ?

- JSF : **J**ava **S**erver **F**aces

  - Framework de présentation pour les applications Web en Java :
    - librairie de composants graphiques,
    - fonctionnalités gravitant autour de ces composants,

  - Définie au sein d'une Java Specification Request (JSR) émise par la Java Community Process (JCP).

HISTORIQUE


C'est parti, on simplifie la production d'IHM
2001
JSR-252
Versions 1.2
JEE 5
Ed Burns
Roger Kitain
2006
2004
JSR-127
Versions 1.0 et 1.1
J2EE 1.4
Ed Burns
Craig McClanahan
Roger Kitain
2009
JSR-314
Versions 2.0
JEE 5
Ed Burns
Roger Kitain

LES IMPLÉMENTATIONS

- Les implémentations doivent respecter les spécifications décrites par les JSR-314.

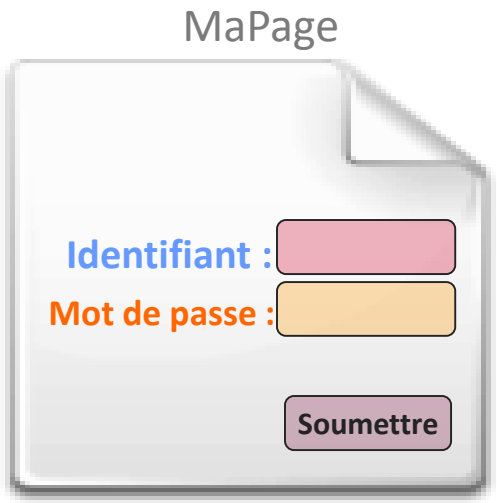
- Sun – Mojarra
  - https://javaserverfaces.dev.java.net/
- Apache – MyFaces
  - http://myfaces.apache.org/

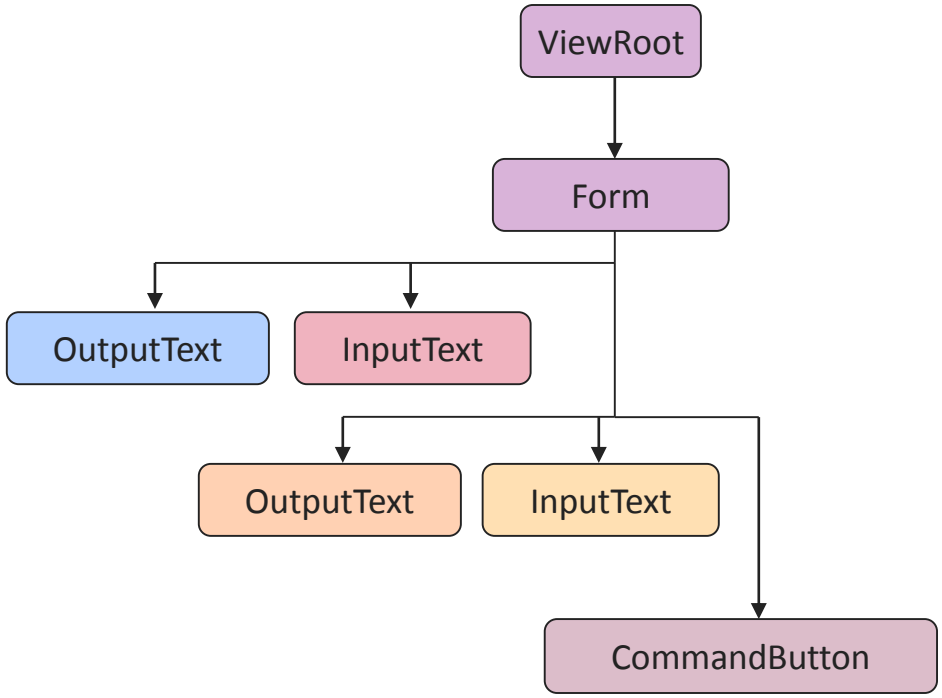
QUOI DE PLUS QUE LES AUTRES ?

- Le concept novateur :
  - Modèle évènementiel.
  - Approche « composant ».
- Mais aussi :
  - Respect du concept M.V.C. (Model/View/Controller),
  - Permet de générer autre chose que du HTML (XML, WML, XUL, ….),
  - Propose des librairies de composants graphiques facilement « *surchargeables* »,
  - Permet de créer ses propres composants,
  - Ajax ready

QUOI DE PLUS QUE LES AUTRES ? SUITE…

- Nouvelle vision
  - Représentation de la page sous forme d'arbre des composants, et accessible via le contexte de l'application.



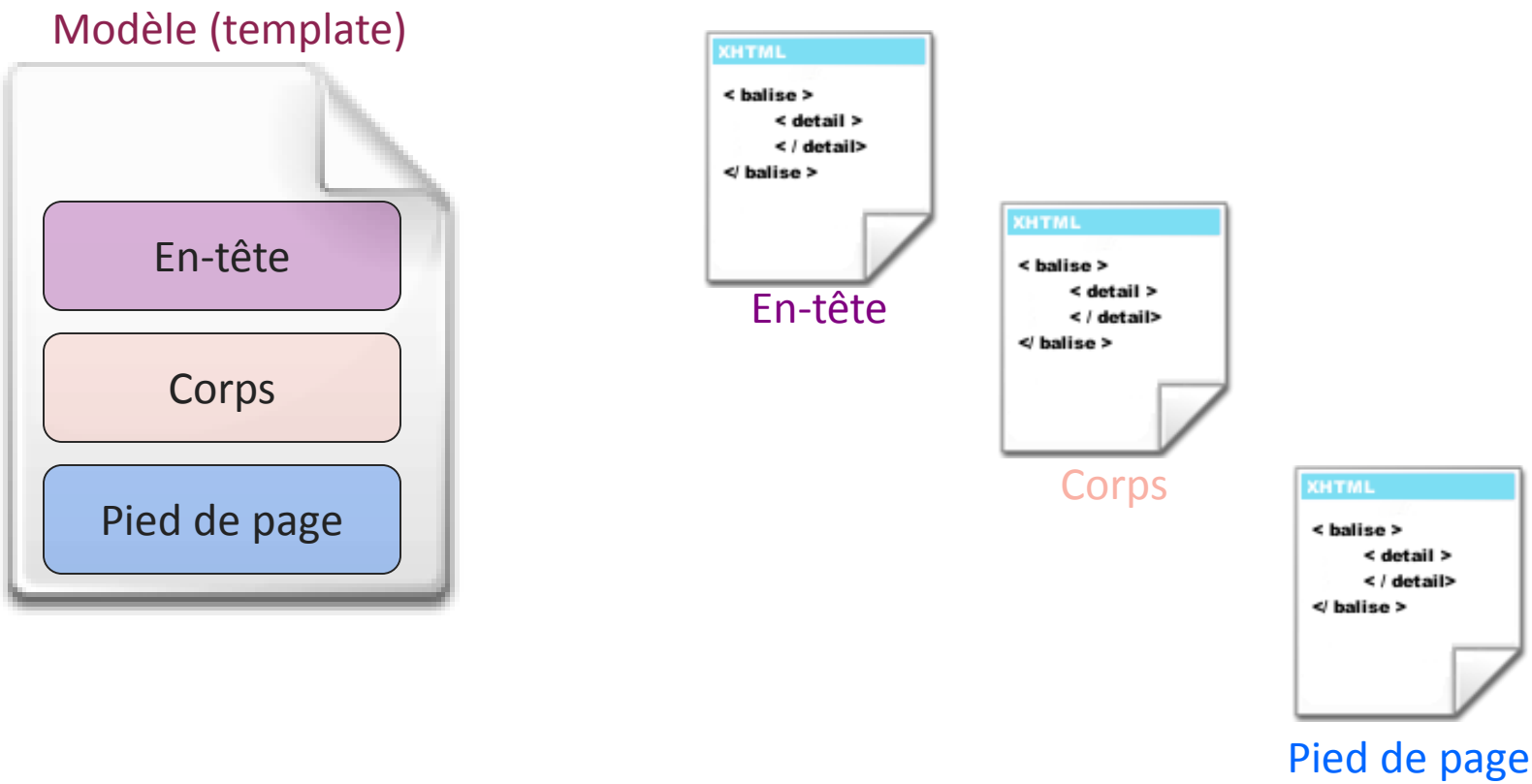


IHM

- Plusieurs technologies possibles pour l'écriture des pages :
  - JSP
  - XHTML
- Ensemble de balises JSF constituant la page :
  - Composants graphiques,
  - Composants de conversion,
  - Composants de validation,
- Templating (modélisation) de pages avec Facelets
  - Inclus dans JSF

IHM : TEMPLATE DE PAGE - FACELETS

- Facelets est un framework de composition de pages ou de composants.



IHM : TEMPLATE DE PAGE (SUITE…)

```
<f:view>
    <!-- EN-TÊTE DE PAGE -->
    <div id="entetePage">
        <ui:include src="../commun/entetePage.xhtml" />
    </div>
    <!-- FIN D'EN-TÊTE DE PAGE -->

    <!-- COMPOSITION DE LA PAGE -->
    <div id="corpsPage">

        <!-- BANDEAU DES ERREURS -->
        <div id="messages">
            <h:messages globalOnly="true" showDetail="false" showSummary="true"
                        errorClass="messageErreur"
                        infoClass="messageInfo" />
        </div>

        <ui:insert name="corpsPage">CORPS DE LA PAGE</ui:insert>

    </div>
    <!-- FIN DE LA COMPOSITION DE LA PAGE -->

    <!-- PIED DE PAGE -->
    <div id="piedPage">
        <ui:include src="../commun/piedPage.xhtml" />
    </div>
    <!-- FIN DE PIED DE PAGE -->
</f:view>
```

Inclusion de page

Insertion de page

Inclusion de page

IHM : EXEMPLE DE PAGE

```
<!-- login.xhtml -->
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:ui="http://java.sun.com/jsf/facelets">
    <h:body>
        <ui:composition template="/pages/templates/template.xhtml">
            <ui:define name="corpsPage">
                <h:form id="loginForm">
                    <h:outputLabel styleClass="labelObligatoire" id="identifiantLabel" for="identifiant">
                        <h:outputText value="#{defaultBundle['login.identifiant.lib']}"/> :
                    </h:outputLabel>
                    <h:inputText required="true" id="identifiant" value="#{loginBean.identifiant}"
                                 label="#{defaultBundle['login.identifiant.lib']}"
                                 title="#{defaultBundle['login.identifiant.info']}">
                        <f:validateRequired for="identifiant"/>
                    </h:inputText>
                    <h:message showSummary="true" showDetail="false" id="identifiantErreur"
                               for="identifiant"
                               styleClass="erreurSaisie"/>
                </h:form>
            </ui:define>
        </ui:composition>
    </h:body>
</html>
```

Déclaration des librairies

Template de page (Facelets)

Ressources

Composants JSF

Lien JavaBean

IHM : LES COMPOSANTS GRAPHIQUES

h:inputText ma saisie

h:outputLink Google

h:selectOneMenu Label1 (Label1, Label2, Label3)

h:inputSecret ••••••••••

h:inputTextarea
ma saisie sur
plusieurs lignes

h:selectOneRadio Label10 Label20 Label30

h:outputText NormandyJug

h:selectOneListbox Label100 Label200 Label300

h:dataTable

Column1	Column2	Column3
Cellule01	Cellule02	Cellule03
Cellule11	Cellule12	Cellule13
Cellule21	Cellule22	Cellule23
Cellule31	Cellule32	Cellule33

h:graphicImage

IHM : COMPOSANTS ADDITIONNELS

- Il existe des librairies supplémentaires proposant des composants supplémentaires.
  - Compléments des composants de base,
  - Menu
  - Onglet
  - Treeview
  - Calendrier
  - …
- MyFaces Tomahawk

- ICEfaces

- JBoss RichFaces

MANAGEDBEAN

- C’est un JavaBean géré par JSF.

- Permet de faire le lien entre l’IHM et le code métier de l’application.
  - Doit contenir
    - Des accesseurs et des mutateurs pour champs de l’IHM.
    - Constructeur vide

- Définition
  - au sein du fichier faces-config.xml
  - ou par le biais d’annotations.
    - @ManagedBean

DESSINE MOI UN MANAGEDBEAN

```java
@ManagedBean(name="loginBean")
@SessionScoped
public class LoginBean implements Serializable {

    /**
     * Identifiant de l'utilisateur
     */
    private String identifiant;

    /**
     * Constructeur
     */
    public LoginBean() {
        super();
    }

    /**
     * Accesseur pour l'attribut identifiant
     *
     * @return Retourne la valeur de l'attribut identifiant.
     */
    public String getIdentifiant() {
        return this.identifiant;
    }

    /**
     * Mutateur de l'attribut identifiant
     *
     * @param _identifiant Valeur associée à l'attribut identifiant.
     */
    public void setIdentifiant(String _identifiant) {
        this.identifiant = _identifiant;
    }
```

Annotations de paramétrage

Attribut relatif au champ de saisie de l'IHM

Accesseur du champ de saisie

Mutateur du champ de saisie

LA CONFIGURATION : FACES-CONFIG.XML

```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config 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-facesconfig_2_0.xsd"
        version="2.0">

    <application>
        <message-bundle>/presentation/messages</message-bundle>
        <locale-config>
            <supported-locale>en</supported-locale>
            <supported-locale>fr</supported-locale>
        </locale-config>
    </application>

    <navigation-rule>
        <from-view-id>/login.xhtml</from-view-id>
        <navigation-case>
            <from-outcome>succes</from-outcome>
            <to-view-id>/WEB-INF/pages/accueil/accueil.xhtml</to-view-id>
        </navigation-case>
        <navigation-case>
            <from-outcome>erreur</from-outcome>
            <to-view-id>/login.xhtml</to-view-id>
        </navigation-case>
    </navigation-rule>

</faces-config>
```

Fichier de ressources par défaut

I18N

Navigation

LA CONFIGURATION : WEB.XML

```xml
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    id="WebApp_ID" version="2.5">

    <!-- Nom de l'application -->
    <display-name>HelloNormandyJug</display-name>

    <!-- Utiliser les fichiers *.xhtml -->
    <context-param>
        <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
        <param-value>.xhtml</param-value>
    </context-param>

    <!-- Servlet JSF -->
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <!-- Mapping de Servlet JSF -->
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <!-- URL prises en compte -->
        <url-pattern>*.jsf</url-pattern>
    </servlet-mapping>

    <!-- Point d'entrée -->
    <welcome-file-list>
        <welcome-file>login.jsf</welcome-file>
    </welcome-file-list>

</web-app>
```

Nom de l'application

Extension des pages

Servlet utilisée

Mapping de la servlet

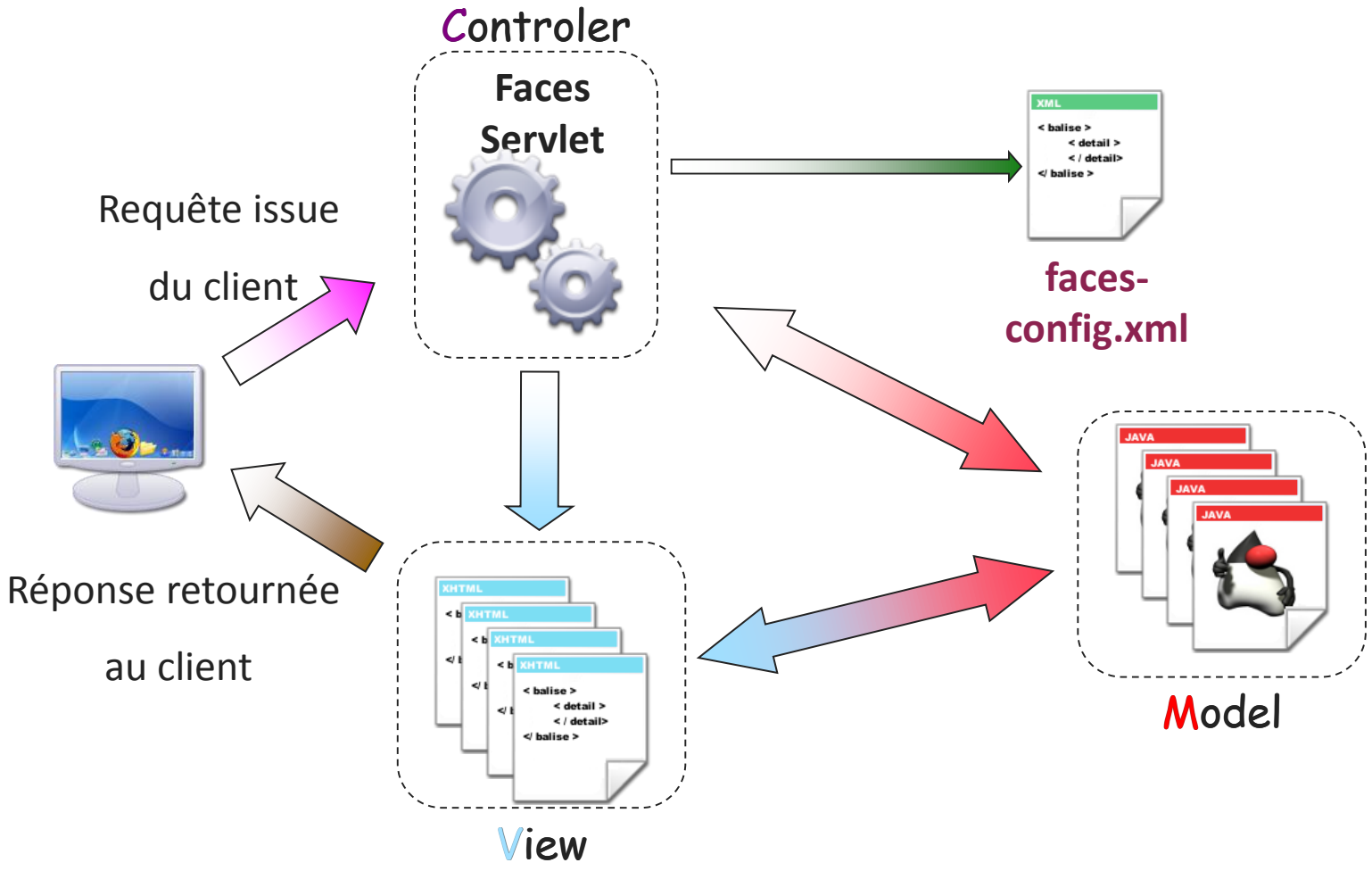
Page d'accueil

WEB.XML (SUITE)

L'ajout d'une balise « security-constraint » permet d'interdire l'accès direct à une page JSF. Toutes les requêtes doivent passer par la Servlet contrôleur de JSF.

```
...
<security-constraint>

    <web-resource-collection>
        <web-resource-name>no-direct-access</web-resource-name>
        <url-pattern>*.xhtml</ url-pattern >
    </web-resource-collection>

    <auth-constraint>
        <description>accès direct interdit !</description>
    </auth-constraint>

</security-constraint>
...
```


RESPECT DU CONCEPT M.V.C.



LES CONVERTISSEURS

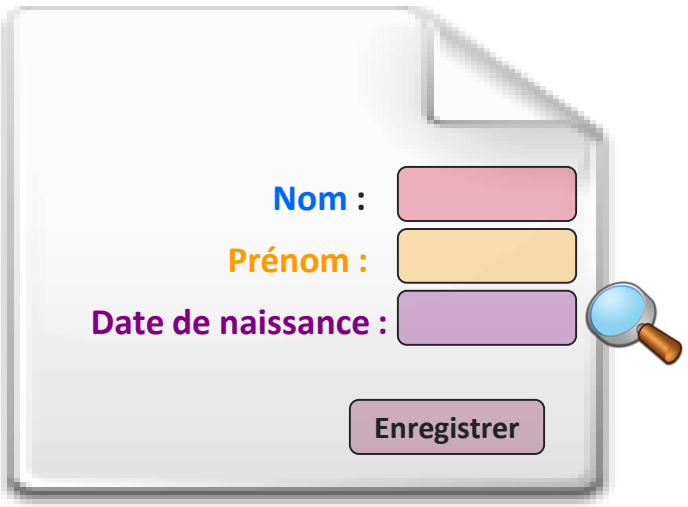
- Permet la conversion des données :
  - IHM vers ManagedBean,
  - ManagedBean vers IHM.

- Exemples de convertisseurs :
  - Conversion de date,
  - Conversion de nombre.

- Il est facile de créer son propre convertisseur.

LES CONVERTISSEURS

MaPage



PersonneBean

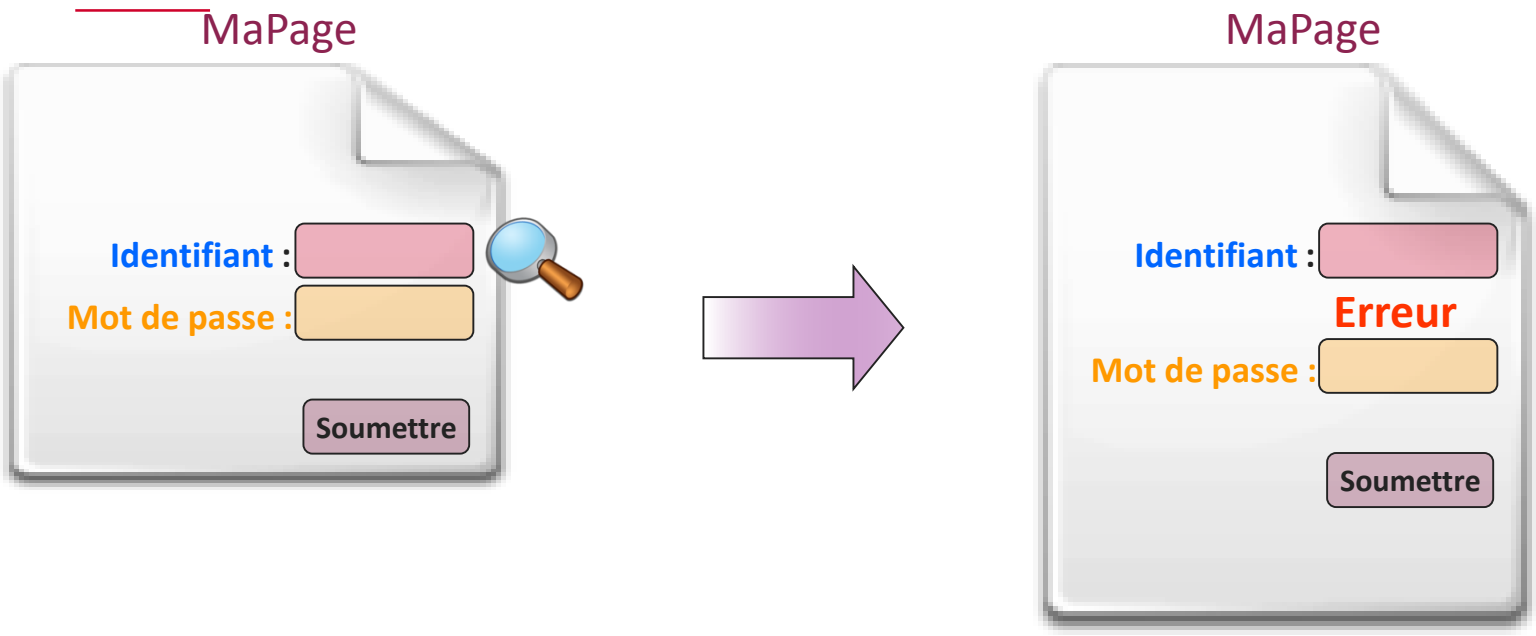
```
/**
 * Nom de l'utilisateur
 */
private String nom;

/**
 * Prénom de l'utilisateur
 */
private String prenom;

/**
 * Date de naissance de l'utilisateur
 */
private Date dateNaissance;
```

```
<h:inputText id="dateNaissance" value="#{personneBean.dateNaissance}">
    <f:convertDateTime for="dateNaissance" type="date" pattern="dd/MM/yyyy"/>
</h:inputText>
```

LES VALIDATEURS

- Vérifier la validité des données converties.
- Applicable sur l'ensemble des composants de saisies.
- Exemples de validateurs :
  - valider la présence de saisie,
  - valider que la saisie est conforme à une plage de valeurs,
  - valider le format de saisie (expression régulière)
  - valider la longueur de la saisie,
  - …

LES VALIDATEURS




```
<h:inputText required="true" id="identifiant" value="#{loginBean.identifiant}"
             label="#{defaultBundle['login.identifiant.lib']}"
             title="#{defaultBundle['login.identifiant.info']}">
    <f:validateRequired for="identifiant"/>
</h:inputText>
<h:message showSummary="true" showDetail="false" id="identifiantErreur"
           for="identifiant"
           styleClass="erreurSaisie"/>
```

LE RENDU

- Les composants JSF peuvent être transcrits en HTML, XML, WML… en fonction de la cible.

- Ceci est possible par le biais de « *Renderer* ».

- Les « *Renderers* » sont des classes Java :
  - récupérant les attributs des composants,
  - transcrivant le composant en fonction du format souhaité.

LE RENDU

maPage.xhtml

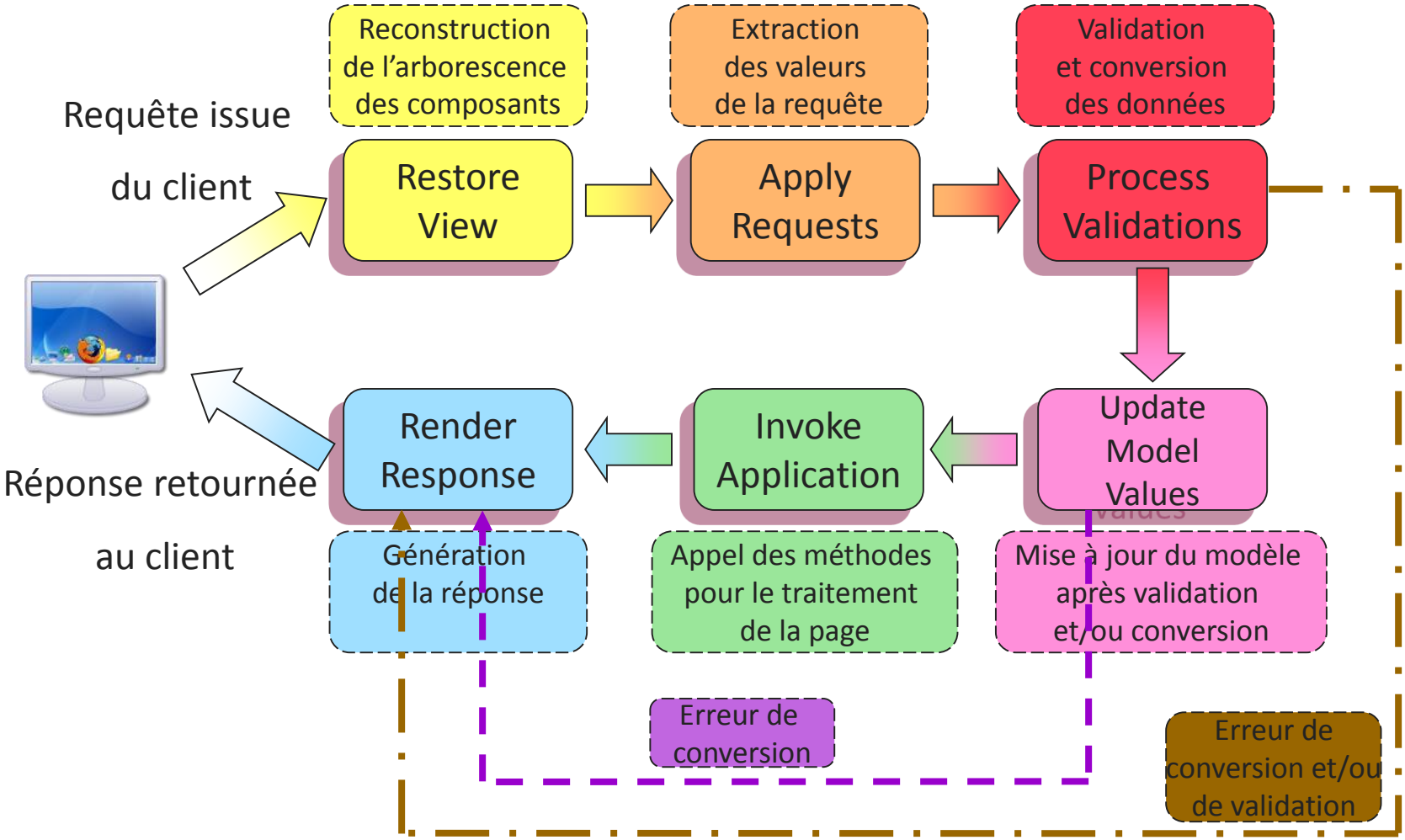
```
<h:inputText id="dateNaissance" value="#{personneBean.dateNaissance}">
    <f:convertDateTime for="dateNaissance" type="date" pattern="dd/MM/yyyy"/>
</h:inputText>
```

HTMLInputTextRenderer.java

```
<input id="personneForm:dateNaissance" type="text" name="perssoneForm:dateNaissance" />
```

maPage.html

LE CYCLE DE VIE

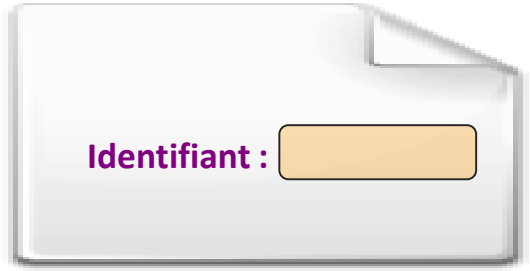


CRÉATION DE COMPOSANTS.

- Qu'est-ce que c'est ?
  - Assemblage de plusieurs composants de base.
  - Création de ses propres composants.
- A quoi ça sert ?
  - Faciliter et uniformiser de développement d'IHM.
  - Répondre aux besoins de l'utilisateur final.
- We can do it !
  - Dans la plupart des cas, pas besoin de connaître un langage *obscur* pour confectionner son propre composant.
    - Java,
    - Xhtml + Facelets,
    - HTML, Javascript, JSTL si nécessaire.

DÉFINIR SON COMPOSANT

```
<!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://java.sun.com/jsf/html"
      xmlns:composite="http://java.sun.com/jsf/composite">
    <head>
        <title>Champ de saisie avec son label</title>
    </head>
    <body>

        <composite:interface>
            <composite:attribute name="id" required="true"/>
            <composite:attribute name="label" required="false"/>
            <composite:attribute name="styleClassLabel" required="false"/>
            <composite:attribute name="value" required="false"/>
            <composite:attribute name="size" required="false"/>
            <composite:attribute name="maxlength" required="false"/>
            <composite:attribute name="tabindex" required="false"/>
            <composite:attribute name="title" required="true"/>
        </composite:interface>

        <composite:implementation>

            <h:outputLabel id="#{cc.attrs.id}Label" for="#{cc.attrs.id}"
                           value="#{cc.attrs.label}" styleClass="#{cc.attrs.styleClassLabel}"/> :

            <h:inputText id="#{cc.attrs.id}" value="#{cc.attrs.value}"
                         title="#{cc.attrs.title}" tabindex="#{cc.attrs.tabindex}"
                         maxlength="#{cc.attrs.maxlength}" size="#{cc.attrs.size}"/>

        </composite:implementation>
    </body>
</html>
```



Attributs

UTILISER SON COMPOSANT

```
<normandyJug:labelInput id="monChampDeSaisie"
                        label="#{defaultBundle['monComposant.monChampDeSaisie.lib']}"
                        styleClassLabel="label"
                        value="#{monComposantsBean.monChampDeSaisie}"
                        size="15" maxlength="10"
                        tabindex="1"
                        title="#{defaultBundle['monComposant.monChampDeSaisie.info']}"/>
```

COMPOSANT : LES VARIABLES UTILES

- “cc” est une variable prédéfinie
  - = “composite component”
  - Référence le composant racine
- Propriété principale “attrs”
  - Contient les attributs transmis au composant lors de l’utilisation• Main page: <someDir:someFile message="test"/>
  - Exemple d’utilisation :

    <h2>Message: #{cc.attrs.message}</h2>
- D’autres attributs utiles :
  - parent, children, clientId

ET AJAX ?

- Les implémentations JSF2 supportent nativement AJAX.
- Les librairies supplémentaires proposent des compléments :
  - MyFaces 

  - ICEfaces 

  - JBoss Richfaces 

JSF AJAX

- Objectifs:
  - Faire des actions en ascynchrone
- Solution : la balise <f:ajax …>
- Exemple :
  - Le composant id1 sera mise à jour sur le click du bouton.

```
<h:commandButton … action="…">
<f:ajax render="id1"/>
</h:commandButton>
<h:outputText … value="#{…}" id="id1"/>
```

- Les attributs :
  - render : Les éléments à réafficher
  - execute : Les éléments à envoyer aux serveurs
  - event : L'événement sur lequel il faut réagir (keyup, blur …)
  - onevent : une fonction javascript à exécuter lors de l'événement
  - Pour render et execute, des valeurs spéciales : @this, @form, @none, @all

MANAGED BEANS ++

- Définition du nom de la variable
  - Au niveau de la classe: @ManagedBean(name="toto")
  - Utilisation de la variable dans le JSF: #{toto.firstName}
- Définition du scope (durée de vie)
  - @RequestScoped : la requête HTTP (défaut)
  - @SessionScoped : la session de l'utilisateur. Le bean doit être serializable.
  - @ApplicationScoped : Partagé par tous les utilisateurs. Attention à la concurrence d'accès !
  - @ViewScoped : Durée de vie de la page. Le bean doit être serializable.
  - @CustomScoped(value="#{someMap}") Contrôler par le développeur
  - @NoneScoped : Utile quand le bean est référencé par d'autres bean.
  - Placé par convention après le @ManagedBean :

```
@ManagedBean
@SessionScoped
public class SomePojo { … }
```

MANAGED BEANS & LES FORMULAIRES

- Les text box

  <h:inputText value="#{customer.firstName}"/>

  - Appel du getter pour l'affichage du formulaire
  - Appel du setter lors de la soumission du formulaire

- Les combo box

  - Dans la JSF:

    *<h:selectOneMenu value="#{bean.**userChoice**}">*

    *<f:selectItems value="#{bean.**listOfOptions**}"/>*

    </h:selectOneMenu>

  - Dans le managed bean :

    List<SelectItem> getListOfOptions() { …}

    String getUserChoise() {…}

WARNING 1 : INJECTION DE DÉPENDANCES

- L'injection de dépendances est réalisée une seule fois au moment de construction d'un bean. De manière générale un bean ne peut référencer qu'un bean de scope plus long.

- Il est donc interdit par exemple pour un bean de scope session de référencer un bean de scope request.
  - Cette règle est valable quelle que soit la méthode d'injection:

- par l'annotation (@ManagedProperty)

- Dans le fichier faces-config.xml

```
<managed-property>
    <property-name>email</property-name>
    <value>#{initializerBean.defaultEmail}</value>
</managed-property>
```

WARNING 2 : INJECTION DE DÉPENDANCES

- L'injection de dépendance est réalisé par JSF après la construction de la classe (logique) …
- Ne pas invoquer de méthodes sur les DAOs ou services métier injectés, dans le code du constructeur : ils sont encore null.
  - Ce mécanisme serait utile pour initialiser une seule fois des combos par exemple, en attachant ensuite le bean en ViewScope.
  - Solution 1 : utiliser l'event "preRenderView" et utiliser un boolean pour détecter le premier event reçu. Car "preRenderView est émis à

```
<f:event type="preRenderView"  listener="#{driverController.prepare}" />
<f:metadata>
    <f:viewParam name="id" value="#{driverController.id}" />
</f:metadata>
```

  - Solution2 : utiliser le principe de Factory avec une durée de vie liée à un scope (pattern classique de Seam framework)

JSF NAVIGATION

- 4 modes de fonctionnement :
  - Static : directement le nom de la page

<h:commandButton value="Show Results" action="page-result"/>

  - Appelle d'une méthode pour calculer le nom de la page destination (sans l'extension)

<h:commandButton value="Show Results" action="#{simpleController.doNavigation}"/>

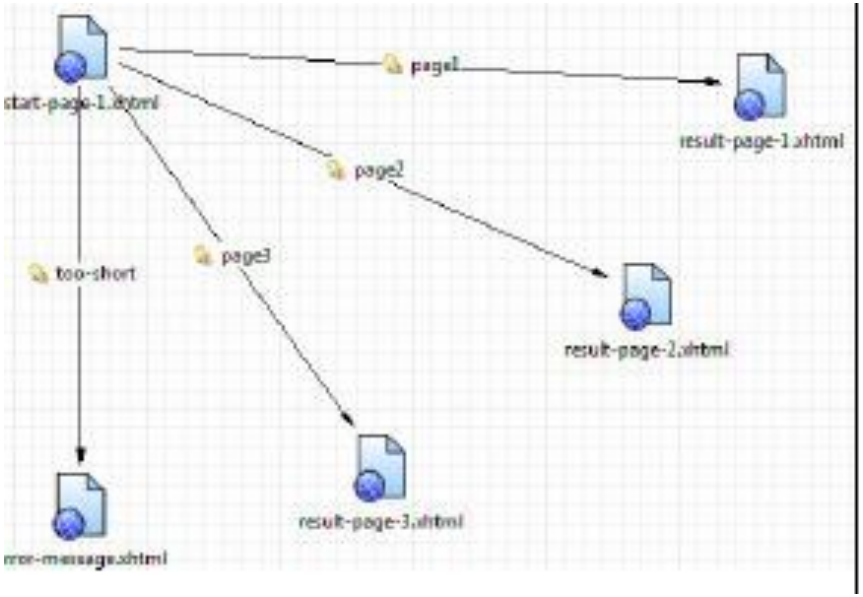
  - Appelle d'une méthode pour calculer une chaine de caractère

<h:commandButton value="Show Results" action="#{simpleController.doNavigation}"/>

  - Définition de règles de navigation dans le fichier faces-config.xml

JSF NAVIGATION

- **Définition de règles de navigation dans le fichier faces-config.xml**
  - *from-view-id*
  - *from-outcome*
  - *to-view-id*



- **Intérêts :**
  - Meilleure maîtrise du projet
    - information centralisée
    - Outil Graphique
  - Facilité d'adaptation
  - Permet d'utiliser des * pour simplifier l'expression

Faces-config.xml

```
<?xml version="1.0"?>
    <faces-config …>
      <navigation-rule>
       <from-view-id>/start-page-1.xhtml</from-view-id>
       <navigation-case>
           <from-outcome>too-short</from-outcome>
           <to-view-id>/error-message.xhtml</to-view-id>
       </navigation-case>
       <navigation-case>
           <from-outcome>page1</from-outcome>
           <to-view-id>/result-page-1.xhtml</to-view-id>
       </navigation-case>
     </navigation-rule>
  </faces-config>
```

JSF NAVIGATION ++

- Navigation Conditionnel

```
<navigation-case>
    <from-outcome>success</from-outcome>
    <if>#{user.returnVisitor}</if>
    <to-view-id>/welcome-back.xhtml</to-view-id>
</navigation-case>
```

- To-Id dynamique

```
<navigation-rule>
    <from-view-id>/exam-question.xhtml</from-view-id>
  <navigation-case>
      <to-view-id>#{exam.nextQuestionPage}</to-view-id>
  </navigation-case>
</navigation-rule>
```

JSF PARAMÈTRE DE PAGE JSF

- Problème: Comment récupérer les paramètres sur une page JSF ?
- Solution: Utiliser f:viewParam pour affecter un champ d'un managed bean. Exemple :

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html">
    <f:metadata>
   <f:viewParam name="param1" value="#{mybean.prop1}"/>
</f:metadata>
```

Le paramètre « param1 », s'il est non null sera affecté au champ prop1 du managed bean « mybean »

- Attribut de configuraiton avancées :
  - required : indique si le paramètre est requis (true|false)
  - requiredMessage : le message d'erreur lorsque le paramètre est manquant
  - Id: l'identifiant du paramètre

JSF PARAMÈTRE DE PAGE JSF

- Comment envoyer les paramètres à une page JSF ?

- Trois situations :
  - Depuis un lien :
    <h:link outcome="blah?param1=val1&param2=val2" value="Click here to go to blah page"/>

  - Depuis un bouton :
    <h:button outcome="blah?param1=val1&param2=val2" value="Click here to go to blah page"/>

  - Depuis une page non JSF:
    <form action="blah">
    <input type="text" name="param1"/>
    <input type="text" name="param2"/> …
    </form>

JSF PARAMÈTRE DE PAGE JSF

- Redirection d'une requête POST en GET
- Objectif: Pouvoir mettre les liens et URLs dans ses marques pages
- URL GET contient tous les paramètres
- Lors de la redirection (navigation) on peut transmettre les paramètres
  ➔ Ajouter « redirect=true&includeViewParams=true » dans l'URL
- Attention à ne pas exposer des données secrètes comme un mot de passe.

JSF DATATABLE

- Objectif: Affichage de d'une liste de données sous forme de tableau

```
<h:dataTable var="someVar" value="#{someCollection}" border="…">
        <h:column>#{someVar.property1}</h:column>
        <h:column>#{someVar.property2} </h:column>
</h:dataTable>
```

- Les attributs du datatable
  - styleClass, captionClass, headerClass, rowClasses, columnClasses
- Entête de colonne :

```
<f:facet name="header">First Heading</f:facet>
```

JSF EXPRESSION

- *#{employee.firstName}*
  Appelle de getFirstName sur le bean employee.
- *<h:inputText value="#{employee.firstName}"/>*
  - Lors de l'affichage du formulaire : Remplit le formulaire
  - Lors de la soumission du formulaire : Valide la données et remplit le bean
- Acces à des éléments d'une collection, tableau, List, Map:
  #{someBean.someProperty[indexOrKey]}
  Exemple:
  - *#{person.friends[2]}*
  - *#{employee.addresses[0].zip}*
- Navigation en profondeur: #{var.prop1.prop2.prop3}
- *<h:commandButton value="Button Label" action="#{employee.processEmployee}"/>*
  Appel de la methode processEmployee sur le click du bouton

JSF EXPRESSION : VARIABLES PRÉDÉFINIES

- facesContext: exemple: *#{facesContext.externalContext.remoteUser}*
- param: les paramètres de la requetes, exemple: *#{param.custID}*
- Header: l'entête de la requête,
  exemple: *#{header["Accept-Encoding"]}*
- cookie: Cookie object, exemple: *#{cookie["userCookie"].value}*
- request, session:
  exemple: *#{request.queryString}, #{session.id}*
- initParam : paramètres d'initialisation
- requestScope, sessionScope, applicationScope, etc.

JSF EXPRESSION

- Opérateur:
  - Arithmetic: *+ - * / div % mod*
  - Relational: *== eq != ne < lt > gt <= le >= ge*
  - Logical: *&& and || or ! Not*
  - *Empty*:
    - Vrai si null, String vide, tableau/collection/map vide
    - Faux sinon
- Test : *#{someCondition ? simpleVal1 : simpleVal2}*
- Des paramètres d'une méthode
  - Exemple: *#{someBean.someMethod(arg1, arg2)}*

JSF I18N

Objectifs: Rendre les applications web multi-langue

- **Utilisation de fichier .properties**
  - Un fichier par langue
    - Toto_fr.properties
    - Toto_en.properties
  - Clé = nom de la propriété
  - Valeur = le texte à afficher dans la langue
  - WEB-INF/classess

WEB-INF/**ressource/Toto**_fr.properties

```
p1=Bonjour
p2=Il y a {0} livres et {1} auteurs.
```

WEB-INF/**ressource/Toto**_en.properties

```
p1=Hello
p2=There is {0} books and {1} authors.
```

- **Déclaration dans le fichier faces-config.xml**

```
<application>
 <resource-bundle>
  <base-name>ressource.Toto</base-name>
  <var>someVar</var>
 </resource-bundle>
</application>
```

// Définition de la variable someVar

- **Utilisation dans une JSF**

```
//Utilisation de la langue définie sur le navigateur
<f:view locale="#{facesContext.externalContext
                                    .requestLocale}">
....
// Utilisation de la variable pour accéder aux propriétés
<h:outputFormat value="#{someVar.p2}">
    <f:param value="5"/>
    <f:param value="#{mybean.nbAuthors"/>
</h:outputFormat>
```

JSF VALIDATION

- **Objectifs:**
  - **Vérifier le format des données d'un formulaire**
  - **Réafficher le formulaire avec les messages d'erreurs**
- **3 tactiques :**
  - **Manuelle**
  - **Implicite automatique**
  - **Validation personnalisée**

JSF TP VALIDATION : MANUELLE

- **Vérification des valeurs dans les setter ou la méthode action**
- **Génération de message FacesContext.addMessage**
- **Renvoie de null pour réafficher le formulaire**
- **Exemple :**

Contrôle et génération des messages dans le bean :

```
public String doAction() {
    FacesContext context = FacesContext.getCurrentInstance();
    if (getLogin().equals("")) {
        context.addMessage(" login" new FacesMessage("Login required"));
    }
    …
    if (context.getMessageList().size() > 0) {
        return null;
    } else {
        …
    }
}
```

Affichage du champ et du message :

```
<h:inputText id="login"…/><h:message for="login"/>
```

JSF VALIDATION : IMPLICITE AUTOMATIQUE

- **Les champs du bean sont typés (type primitif)**
- **La JSF déclare les messages d'erreurs**
- **Exemple :**

Affichage du champ et du message :

```
<h:inputText value="#{mybean.stuffDuration}"
        required="true"
        requiredMessage="You must enter a duration"
        converterMessage="Duration must be a whole number"
        id="duration"/>
<h:message for="duration" styleClass="error"/>
```

JSF VALIDATION : LES VALIDATEURS/CONVERTISSEURS

- f:validateLength :
  - minimum,
  - maximum
- f:validateLongRange :
  - minimum,
  - maximum
- f:validateDoubleRange :
  - minimum,
  - maximum
- f:validateRegex :
  - pattern

- **f:convertNumber :**
  - **currencyCode, currencySymbol**
  - **groupingUsed**
  - **integerOnly**
  - **Locale**
  - **max(min)FractionDigits**
  - **max(min)IntegerDigits**
  - **pattern (ala DecimalFormat)**
  - **type: number, currency, percentage**
- **f:convertDateTime**
  - **type: date, time, both**
  - **dateStyle, timeStyle : default, short, medium, long, full**
  - **pattern (ala SimpleDateFormat)**
  - **Locale**
  - **timeZone**

```
<h:inputText value="#{bidBean2.userID}"
        required="true"
        requiredMessage="You must enter a user ID"
        validatorMessage="ID must be 5 or 6 chars"
        id="userID">
<f:validateLength minimum="5" maximum="6"/>
```


JSF VALIDATION : PERSONNALISÉE

- **Définir sa propre méthode de validation**
- **La JSF déclare les messages d'erreurs**
- **Exemple :**

Affichage du champ et du message :

```
<h:inputText id="someID" validator="#{someBean.someMethod}"/>
<h:message for="someID"/>
```

Méthode de valiadation dans le bean :

```
public void validateBidAmount(FacesContext context,
        UIComponent componentToValidate,
        Object value) throws ValidatorException {

        double bidAmount = ((Double)value).doubleValue();
        double previousHighestBid = currentHighestBid();
        if (bidAmount <= previousHighestBid) {
                FacesMessage message = new FacesMessage("Bid must be higher than current "
                        + "highest bid ($" + previousHighestBid + ").");
                throw new ValidatorException(message);
        }
}
```

JSF BOUCLE

- La syntaxe :

```
<ui:repeat var="someVar" value="#{someBean.someCollection}">
        <someHTML>
        #{someVar.someProperty}
        </someHTML>
</ui:repeat>
```

- Ne fonctionne pas avec les types primitifs (int, byte) mais avec les types Objet (Integer, Byte, Author)
- On peut imbriquer les boucle.
- D'autres attributs :
  - varStatus = nom d'une variable pour accéder au status
    - begin, end, step: int
    - index: int
    - first/last: boolean
    - even/odd: boolean

JSF EVENT

- Deux catégories d ’événement :
  - Les ActionListener qui soumettent le formulaire
    - *<h:commandButton actionListener="..." immediate="true"/>*
    - actionListener: le nom d’une méthode
    - immediate: boolean indiquant si la méthode s’exécute avant la validation
    - La méthode appelée :

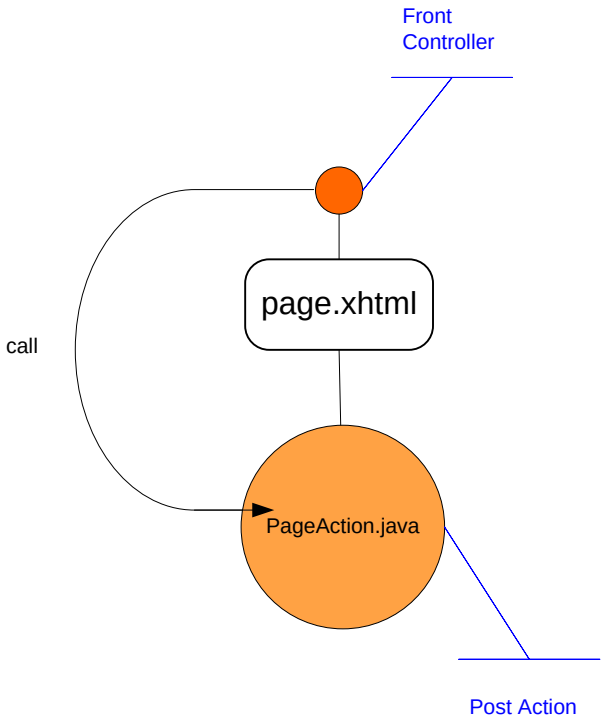
```
public void someMethod(ActionEvent event) {
doSomeSideEffects();
}
```

  - Les ValueChangeListener qui ne soumettent pas le formulaires
    - *<h:selectOneMenu valueChangeListener=" "...>*
    - *<h:selectBooleanCheckbox.../>*
    - *<h:selectOneRadio .../>*
    - *<h:inputText .../>*
    - *Besoin d’appeler onclick="submit()" ou onchange="submit()"*
    - Exemple :

```
<h:selectBooleanCheckbox value="#{formSettings.checked}"
valueChangeListener="#{formSettings.swapLocale2}"
onclick="submit()" immediate="true"/>
```

PRINCIPE PAGE ⇔ CONTROLLER

- Le principe consiste a intégrer une page à seul contrôleur. Chaque page / IHM doit être pensé comme un bloc autonome
- Ce principe permet de gérer avec moins de soucis la navigation entre pages, il suffit de définir la navigation d'un coté et la logique métier continuera (en général) de fonctionner !



ASSEMBLAGE DES PAGES