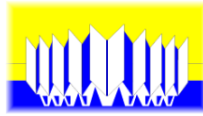


Faculté des Sciences de Tunis, Tunisie

JSF 2.0

Framework de présentation de la plateforme JEE



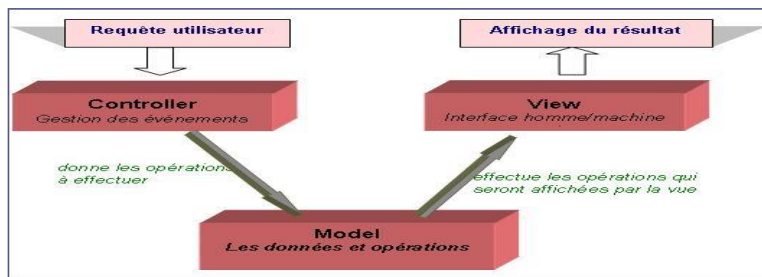
Présenté par : Nourhène ALAYA

Sommaire

- I. Le modèle MVC2
- II. Architecture du Framework JSF
- III. Fonctionnement de JSF
 - 1. Faces-config.xml
 - 2. Les classes Managed Bean
 - 3. Les page Web avec Facelet
 - 4. Les expressions de langage
 - 5. Les règles de navigation
 - 6. Composants graphiques
 - 7. Les Backing Bean
 - 8. Fichier de propriétés et internationalisation
 - 9. Le Templating

Rappel MVC

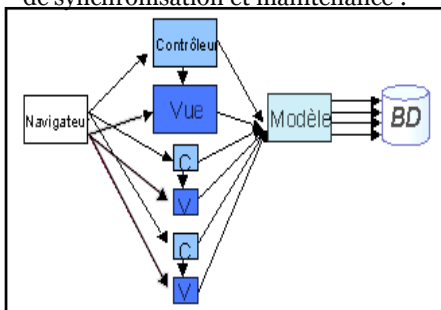
- Le **Model-View-Controller** (MVC) a été développé à la fin des années 70. Il a été conçu à l'origine pour permettre de gérer les interactions entre une interface graphique et un utilisateur sur les premiers ordinateurs possédant un système à base de fenêtres.
- Il s'appuie essentiellement sur la séparation en 2 couches verticales regroupant d'un côté les objets métiers (Modèle) et de l'autre les objets IHM, ces derniers étant eux mêmes regroupés en objets charges de l'acquisition d'informations, en provenance de l'utilisateur (contrôleur) et en objets charges de la restitution d'informations vers l'utilisateur (Vue).
- Le Modèle devrait être toujours indépendant : **ne Jamais mélanger code du Modèle avec code GUI!**



MVC1 vs MVC2

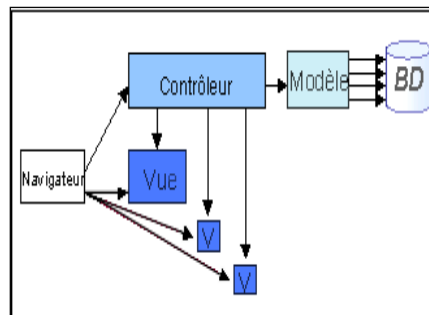
• MVC1

- La séparation en 2 couches verticales.
 - Les **objets métiers** (Modèle)
 - Les **objets IHM** (Contrôleur, Vue)
 - Plusieurs contrôleurs**
- Composants séparés. Mais...Problème de synchronisation et maintenance !



• MVC 2

- hérite des propriétés du modèle MVC
- Le navigateur interagit avec **un seul composant, le contrôleur**, l'architecture est simplifiée.
- Il garantit l'unicité du point d'entrée de l'application



MVC1 vs MVC2

- Dans l'architecture MVC 2, il n'existe plus qu'un seul et unique contrôleur réceptionnant toutes les requêtes Web du clientes.
- Le contrôleur unique devient le point **d'entrée exclusif** de l'application.
- Il devient alors très aisé de **centraliser** la gestion des accès, des droits, des statistiques ou de toute autre fonctionnalité transverse.

=> Simplification de l'architecture de l'application

=> Meilleur maintenance du code.

Framework intégrant MVC2

- ❑ Si vous voulez implémenter le pattern MVC2 correctement, il y aura beaucoup de code administratif

- Le traitement du Request , la gestion d'erreurs , le mapping de la vue, etc

⇒ **Difficile à maintenir, d'où le besoin d'utiliser un Framework.**

- ❑ **Qu'est ce qu'un Framework ? => Framework = squelette d'application.**

- Ensemble de classes et mécanismes associés à une architecture logicielle qui fournissent un ou plusieurs services techniques ou métiers aux applications qui s'appuient dessus
- Construite sur une infrastructure commune pour fournir un point de départ
- Aide les développeurs à se concentrer sur ce que l'application a besoin de faire au lieu de comment elle va le fera.

- ❑ Plusieurs Framework qui utilisent le pattern MVC2

- **Struts , Barracuda , Hammock** (disparu) , **Tapestry , Java Server Faces** (le nouveau standard Java)

Framework de MVC2

Java Server Faces (JSF)

<http://www.oracle.com/technetwork/java/javase/javaserverfaces-139869.html>

<http://www.coreservlets.com/JSF-Tutorial/jsf2/>

Introduction: Evolution des Framework Web

- Servlet (1997)
- JSP Procédurale (1999)
- Frameworks de composition
 - **Tapestry (2000)**
- Frameworks à base d'action
 - **Struts (2001)**
- Frameworks à base de composant
 - **JavaServer Faces (2004)**

Introduction: Java Server Faces

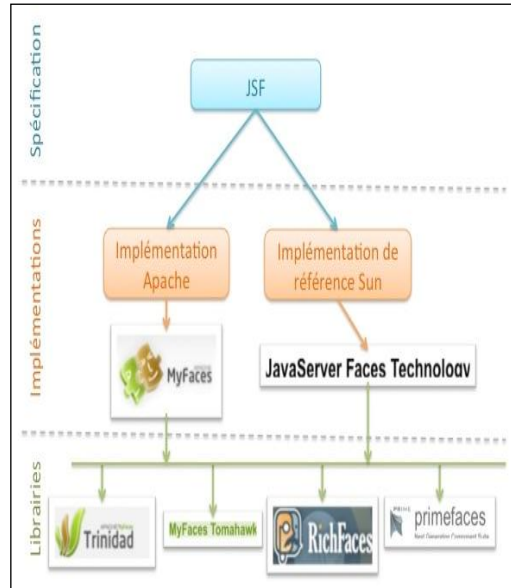
- Java Server Faces est un Framework développé par Sun pour la création des interfaces utilisateur coté serveur pour les applications web.
- JSF est basé sur l'architecture MVC2.
- JSF est construit sur un modèle de développement **orienté événement**.
- JSF offre un modèle d'architecture extensible en terme de **composants**.
- **Pourquoi JSF?**
 - **JSF permet d'accroître la productivité des développeurs dans la conception d'interfaces graphiques d'applications Web exécutés coté serveur grâce à la notion de composants graphiques.**
 - **Ressemble aux WebForms de la plateforme .Net de Microsoft.**

Qu'apporte JSF ?

- Java Server Faces permet
 - Une **séparation** de la couche présentation des autres couches (MVC 2)
 - Un ensemble de **composants graphiques** riches et réutilisables
 - Une liaison simple entre les actions côté client de l'utilisateur (**event listener**) et le code Java côté serveur
 - Création de nouveaux **composants graphiques**
 - JSF peut être utilisé pour générer autre chose que du HTML (XUL, XML, WML, XHTML ...)

Les implémentations de JSF

- ❑ Java Server Faces est une **spécification** : il est donc nécessaire d'obtenir une implémentation de la part d'un tiers.
- ❑ Plusieurs implémentations commerciales ou libres sont disponibles, notamment:
 - l'implémentation de référence de **Sun (JSF)**
<http://www.oracle.com/technetwork/java/javace/javaserverfaces-139869.html>
 - l'implémentation **MyFaces** qui est devenu un projet du groupe Apache.
<http://myfaces.apache.org/>



Librairies de composants graphiques pour les pages JSF

- Oracle ADF Faces
- ICEFaces
- **JBoss RichFaces**
- **JBoss Ajax4JSF**
- Apache Tomahawk
- Apache Sandbox
- Apache Trinidad
- Primefaces
- **And many more ...**

ORACLE



Outils de développement JSF

- Eclipse JSF Tools (WTP 2.0)
- IBM RAD
- NetBeans
- Sun Java Studio Creator
- RHDS / Exadel
- MyEclipse IDE
- Oracle JDeveloper
- BEA Workshop for JSF
- **Eclipse JBoss Tools**



NetBeans



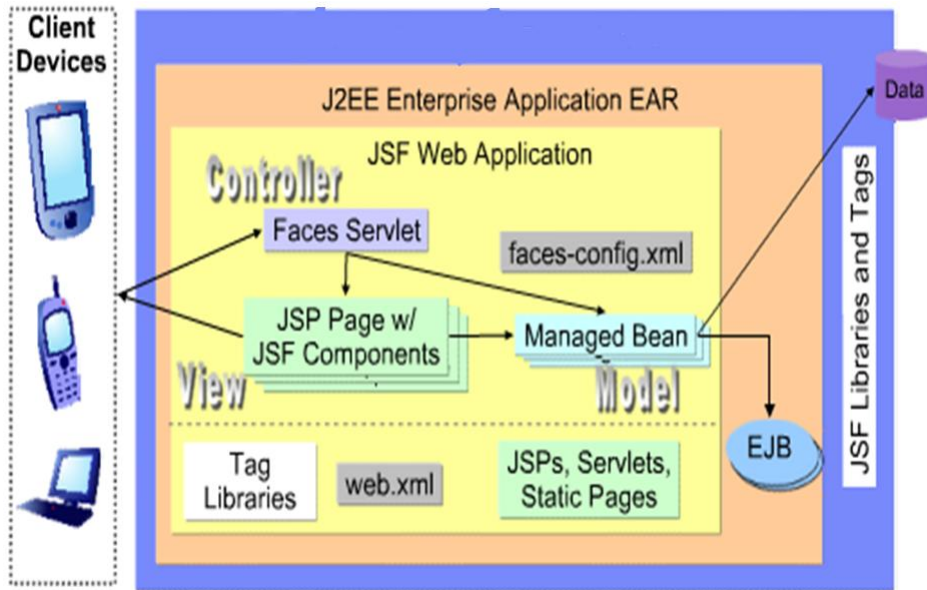
ORACLE 10g
JDEVELOPER



Composants de l'architecture JSF

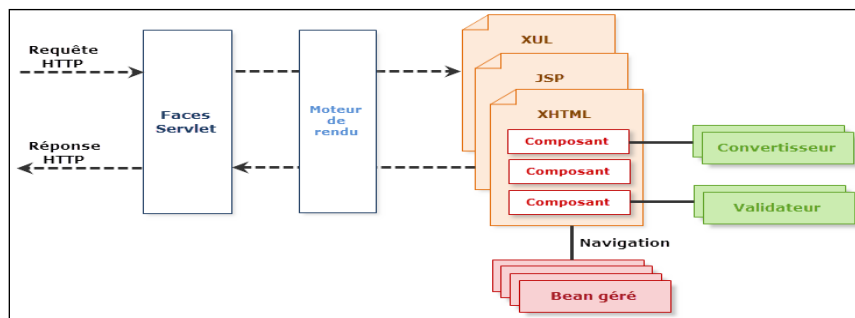
- **Le contrôleur (Faces Servlet) :**
 - Servlet principal de l'application qui sert de contrôleur. Déjà implémenté dans le Framework. Toutes les requêtes de l'utilisateur passent systématiquement par ce Servlet, qui les examine et appelle les différentes actions correspondantes. Ce contrôleur sera déclaré dans le Web.xml et configuré dans le **fichier faces-config.xml**
- **La vue: pages web (JSP, JSF, XHTML) :**
 - D'autre type de vue existe, comme **WML** pour les dispositifs mobiles. La version **JSF 2.0** utilise les **Facelets**. Les **Facelets** sont formées d'une arborescence de composants **UI** (également appelés widgets ou contrôles).
- **Le modèle (Managed Bean/ Backing Bean) :**
 - Classes Java spécialisées qui synchronise les valeurs avec les composants UI, accède au logique métier et gère la navigation entre les pages.
- **faces-config.xml**
 - Fichier de configuration de l'application définissant les règles de navigation et les différents Managed Bean utilisés.

Architecture de JSF avec MVC2

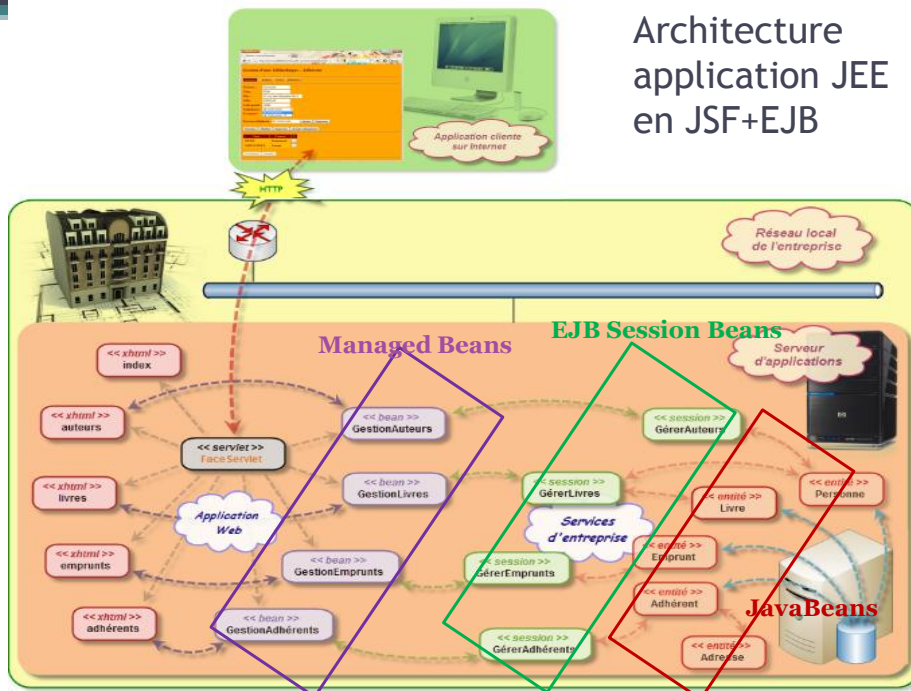


Composants de l'architecture JSF

- Outre les composants MVC, JSF est composé de:
 - **Moteur de rendu (Renderer):** décode la requête de l'utilisateur pour initialiser les valeurs du composant et encode la réponse pour créer une représentation du composant que le client pourra comprendre et afficher.
 - **Convertisseurs et validateurs:** Le protocole HTTP est un protocole uniquement textuel, donc nous aurons besoin de valider les champs de saisie textuelle et de les convertir vers les autres types d'Objets.



17



18

Les étapes de développements avec JSF

1. Configurer le fichier **web.xml** afin de déclarer le Faces Servlet
2. Développer les objets du modèle qui détiennent les données (**Managed Bean ou les Backing Bean**)
3. Déclarer les Managed Bean dans le fichier de configuration de l'application **faces-config.xml**
4. Créer des pages en utilisant les composants d'interface utilisateur UI et les tagLib de JSF
5. Définir les règles de navigation entre les pages dans faces-config.xml

1. La configuration JSF dans le Web.xml

- Exemple basique d'un Web.xml avec l'implémentation Sun de JSF

```
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <display-name>Test JSF</display-name>
  <description>Application de tests avec JSF</description>
  <context-param>
    <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
    <param-value>client</param-value>
  </context-param>

  <!-- Servlet faisant office de controleur-->
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup> 1 </load-on-startup>
  </servlet>

  <!--Le mapping de la servlet -->
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.faces</url-pattern>
  </servlet-mapping>
</web-app>
```

1. Configuration de JSF dans le web.xml

- Identifier la servlet principale : **javax.faces.webapp.FacesServlet**
- Spécifier le nom et le chemin du fichier de configuration
 - Nom du paramètre : **javax.faces.application.CONFIG_FILES**
 - Exemple : /WEB-INF/faces-config.xml
- Spécifie où l'état de l'application doit être sauvé
 - Nom du paramètre : **javax.faces.STATE_SAVING_METHOD**
 - Valeurs possibles : **client** ou **server**
- La classe Servlet principale est le point d'entrée d'une application JSF. On peut pas appeler directement une page web en utilisant l'extension de son fichier (.jsp ou .xhtml) sans avoir utiliser l'«url-pattern» du Servlet principal. Ainsi, l'utilisation de préfixe ou suffixe déclaré dans le web.xml (url-pattern) devient obligatoire. On trouve plusieurs manières de déclencher des ressources JSF:
 - Préfixe /faces/ (<http://localhost/myAppli/faces/index.jsp>)
 - Suffixes *.jsf (<http://localhost/myAppl/index.jsf>)
 - ou *.faces (<http://localhost/myAppl/index.faces>)

```
<context-param>
  <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
  <param-value>.jsf</param-value>
</context-param>
```

1. La configuration de JSF dans le Web.xml

- Le démarrage d'une application directement avec une page par défaut utilisant JSF ne fonctionne pas correctement. Il est préférable d'utiliser une page (index.jsp), qui va effectuer une redirection vers la page d'accueil de l'application d'une manière invisible à l'utilisateur.
- Exemple dans un projet dont le suffixe du Servlet est «.jsf»
 - Dans «index.jsp» il suffit d'écrire la ligne suivante :

```
<% response.sendRedirect("welcome.jsf"); %>
```

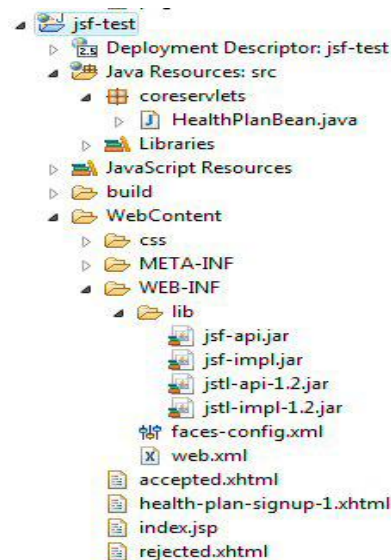
Ou bien

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<html>
<head>
<meta http-equiv="refresh" content="0; url=login.jsf" />
</head>
<body>
</body>
</html>
```

La page index.jsp ne sera pas affiché, il aura une redirection silencieuse vers login.jsf

1. La configuration de JSF dans le Web.xml

- Pour exploiter et déployer une application WEB avec JSF il faut:
 - des librairies JSF (Apache ou Sun) et JSTL
 - configurer le fichier web.xml selon la version de JSF



2. Le fichier de configuration faces-config.xml

- La description de l'ensemble des balises peut être trouvée: <http://www.horstmann.com/corejsf/faces-config.html>
- Un exemple basique de fichier faces-config.xml, que nous allons enrichir fur à mesure de ce cours:

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

<!DOCTYPE faces-config PUBLIC
"-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.1//EN"
"http://java.sun.com/dtd/web-facesconfig_1_1.dtd">

<faces-config>
  <navigation-rule>
    ...
  </navigation-rule>
  <managed-bean>
    ...
  </managed-bean>
</faces-config>
```

2. Le fichier de configuration faces-config.xml

- Le fichier gérant la logique de l'application web s'appelle par défaut **faces-config.xml**
- Il est placé dans le répertoire WEB-INF au même niveau que web.xml. Il décrit essentiellement six principaux éléments :
 - les Beans managés **<managed-bean>**
 - les règles de navigation **<navigation-rule>**
 - les ressources éventuelles suite à des messages **<message-bundle>**
 - la configuration de la localisation **<resource-bundle>**
 - la configuration des Validateurs et des Convertisseurs **<validator>** et **<converter>**
 - d'autres éléments liés à des nouveaux composants JSF **<render-kit>**
- Le fichier de configuration est un fichier XML décrit par une DTD. La balise de départ est **<faces-config>**

3. Les classes Managed Bean

- Un Bean géré est un Bean dont la vie est gérée par JSF et déclaré dans le fichier de configuration faces-config.xml
- Les Beans gérés de JSF n'héritent pas d'une classe particulière. Il permettra de réaliser
 - l'affichage des données provenant de la couche métier
 - le stockage des valeurs d'un formulaire
 - la validation des valeurs
 - l'émission des clés de navigation reçues par faces-config.xml
- Il se compose de :
 - Ensemble des attributs d'instance (private), qui correspondent à chaque zone de saisie du formulaire d'entrée
 - Les accesseurs de ses attributs.
 - Les méthodes d'action qui retournent une chaîne (la clé de navigation). Ces méthodes seront mentionnées comme l'action suite au click à un bouton présent dans le formulaire d'entrée.

3. Les classes Managed Bean

- Squelette d'une classe Managed Bean :

```
public class SomeBean {
    private String someProperty;

    public String getSomeProperty() { ... }

    public void setSomeProperty() { ... }

    public String actionControllerMethod() {
        ...
    }

    // Other methods
}
```

❑ Il faut respecter les règles suivantes lors de l'écriture d'une classe Managed Bean:

- Pas de déclaration de constructeur
- Pas d'attributs public
- Si un attribut est de type **boolean**, sa méthode d'accès 'getXXX' doit être écrite sous forme de '**isXXX**'. (eg. **isMarried()**)

4. Les classes Managed Bean

- Il faut déclarer un Bean managé dans le fichier de configuration de JSF à l'aide de la balise `<managed-bean>`
- Trois éléments essentiels sont à préciser:
 - `<managed-bean-name>` définit un nom qui servira d'étiquette quand le Bean sera exploité dans les pages JSP
 - `<managed-bean-class>` déclare le nom de la classe de type `package.class`
 - `<managed-bean-scope>` précise le type de scope utilisé pour le Bean (`none`, `application`, `session`, `request`)

```
<managed-bean>
  <managed-bean-name>MyBean</managed-bean-name>
  <managed-bean-class>mypackage.MyFirstBean</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
...
```

faces-config.xml

4. Exemple d'un Managed Bean qui correspond à un formulaire

```
package com.fst.JSF;
public class ConnectBean {
    private String email="user@mail" ;
    private String password="" ;
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public String connect(){
        if (email.isEmpty() || !email.contains("@") ) {
            return "Rejected";
        }
        if (password.isEmpty() || !password.equals("cpoo")) {
            return "Rejected";
        }
        return "Accepted";
    }
}
```

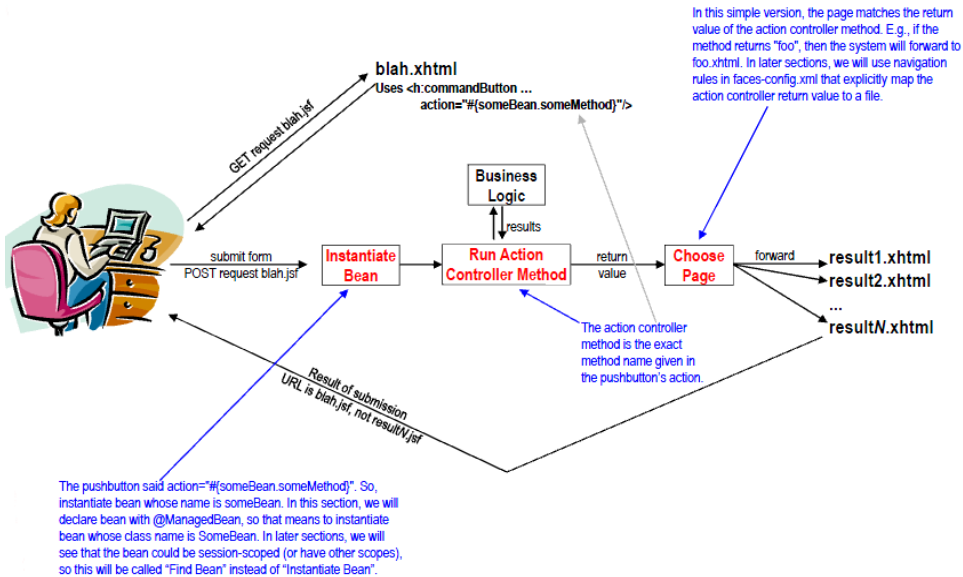
Déclaration dans Faces-config.xml

```
<managed-bean>
  <managed-bean-name>connectBean</managed-bean-name>
  <managed-bean-class>com.fst.JSF.ConnectBean</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Affichage de la valeur d'Initialisation du Bean

Clé de navigation

3. Etape d'exécution d'un Managed Bean



4. Création Page JSF 2.0 avec Facelet

- Comme *JSP*, **Facelets** est une technologie de présentation pour le développement d'applications web en *Java*.
- Une page *JSP* est transformée en une *Servlet* qui possède un cycle de vie différent de celui de *JSF*, ce qui peut être source de confusion et de problèmes. A l'inverse, **Facelets** est spécifiquement développé pour *JSF* et est plus performant et léger.
- Facelets** introduit des nouvelles fonctionnalités, comme par exemple un système de **templating** ou encore la possibilité de créer des composants personnalisés sans écrire la moindre ligne de code *Java*.
- Facelets** est basé sur **xml**, c'est pour cette raison que les vues sous **facelets** sont des pages **xhtml** (ou encore **jspx**) et qu'elles doivent impérativement respecter la structure d'un document **xml**.

4. Squelette d'une page XHTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
1.0 Transitional">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:f="http://java.sun.com/jsf/core">

<h:head>
...
</h:head>

<h:body>

<h:form>
...
</h:form>

</h:body>

</html>
```

- On peut continuer d'utiliser les balises HTML dans les pages XHTML. Cependant, il est désormais interdit d'utiliser des Scriptlets et le code Java.
- Appel aux deux bibliothèques de balises personnalisées de **JSF**:
 - Taglib « **html** » préfixé par '**h**'
 - Taglib « **core** » préfixé par '**f**'

4. Accès aux Bean managés : Facelet et Expression de Langage (EL)

- Un formulaire JSF doit être construit dans un groupe défini par la balise `<html:form> ... </html:form>`
 - **ACTION** est automatiquement à SELF (URL courante)
 - **METHOD** est obligatoirement POST
- Utilisation de composants JSF pour saisir des informations
 - `<h:inputText>` pour la balise HTML `<INPUT TYPE="Text">`
 - `<h:inputSecret>` pour la balise `<INPUT TYPE="PASSWORD">`
 - `<h:commandButton>` pour la balise `<INPUT TYPE="SUBMIT">`
- La balise `<h:commandButton>` contient un attribut **action** qui permet:
 - d'indiquer une **clé de navigation** traité par les règles de navigation définies dans `faces-config.xml`
 - Ou de faire appel à une **méthode d'action** dans un Managed Bean via une expression de langage **EL**.

4. Accès aux Managed Bean avec EL

- Les **Expressions Languages (EL)** sont utilisées pour accéder aux éléments du Bean dans les pages JSP ou XHTML
 - Un **EL** permet d'accéder simplement aux Beans des différents scopes de l'application (page, request, session et application)
 - Forme d'un Expression Language JSF **`#{expression}`**
 - Une EL est une expression dont le résultat est évalué au moment où JSF effectue le rendu de la page
- L'écriture **`#{MyBean.value}`** indique à JSF
 - de déterminer le nom de l'objet MyBean dans le fichier **faces-config.xml**
 - de chercher l'instance du Bean, qui porte le nom de **MyBean** dans son contexte puis invoque la méthode **getValue()** (chercher la propriété **value**)
- Possibilité d'accéder à un objet contenu dans le Bean
 - **`#{MyBean.myobject.value}`** : propriété value de **myobject** contenu dans **MyBean**

4. Accès aux Managed Bean avec EL

- Déclaration du Bean dans Faces-config.xml

```
...
<navigation-rule>...</navigation-rule>
<managed-bean>
  <managed-bean-name>MyBean</managed-bean-name>
  <managed-bean-class>mypackage.MyFirstBean</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
...
```

- Dans la vue, l'accès en lecture ou en écriture aux attributs du Bean et à ses méthodes s'effectue comme suit:
 - **`<h:inputText value="#{MyBean.name}"/>`**
 - ⇒ La valeur saisie sera stockée au niveau de l'attribut **name** de **MyBean**. (≈ **MyBean.setName(String s)**)
 - **`<h:inputText value="#{MyBean.name}"/>`**
 - ⇒ affiche la valeur de l'attribut **name** de **MyBean**. (≈ **MyBean.getName()**)
 - **`<h:commandButton value="Login" action="#{MyBean.authentication}" type="submit" />`**
 - ⇒ Suite au **submit** du formulaire, il aura appelé la méthode **authentication()** du Bean **MyBean**

Exemple des Expression Language

EL	Signification
<code>#{unBean.unChamp}</code>	Retourne la valeur du champ unChamp du managed bean unBean
<code>#{unBean[unChamp]}</code>	équivalente à l'écriture précédente
<code>#{unBean.unChamp.unAutreChamp.encoreUnChamp}</code>	Les EL n'imposent pas une limite quant à la profondeur
<code>#{uneListe[5]}</code>	Retourne le cinquième élément de la liste uneListe
<code>#{unMap[uneClé]}</code>	accède à la valeur enregistrée dans le map unMap sous la clé uneClé.
<code>#{unMap.unClé}</code>	équivalente à l'écriture précédente
<code>#{uneListe[unBean.unIndex]}</code>	accède à l'élément d'indice égal à la valeur du champ unIndex du bean unBean.
<code>#{unMap[unBean.unClé]}</code>	accède à la valeur enregistrée dans le map unMap sous la clé de valeur égale à la valeur du champ uneClé du bean unBean.
<code>#{unMap[unBean.unClé][2]}</code>	accède au second élément de la liste enregistrée dans le map unMap sous la clé unBean.unClé
<code>#{uneValeur>25}</code>	retourne true si une valeur est > 25. Les opérateurs suivants sont aussi supportés: <, <=, >, >=, ==, !=.
<code>#{uneValeur>25 and uneAutreValeur<10}</code>	les EL supportent les opérateurs booléens: and, or et not.
<code>#{(unChamp=="premier":"autre")}</code>	retourne la chaîne 'premier' si unChamp est égal à 1, 'autre' sinon.
le nom est <code>#{unBean.nom}</code> et l'âge est <code>#{unBean.age}</code>	Les EL peuvent être combinés avec un contenu statique.
<code>#{10+2/3*(8-1)}</code>	Les EL peuvent évaluer des expressions arithmétiques.

Suite de l'exemple d'un formulaire de connexion

- Managed Bean «**ConnectBean.java**»

```
package com.fst.JSF;
public class ConnectBean {
    private String email="user@mail" ;
    private String password="" ;
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public String connect(){
        if (email.isEmpty() || !email.contains("@") ) {
            return "Rejected";
        }
        if (password.isEmpty() || !password.equals("cpoo") ) {
            return "Rejected";
        }
        return "Accepted";
    }
}
```

Déclaration dans Faces-config.xml

```
<managed-bean>
  <managed-bean-name>connectBean</managed-bean-name>
  <managed-bean-class>com.fst.JSF.ConnectBean</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Clé de navigation

Affichage de la valeur d'Initialisation du Bean

Suite de l'exemple

- La page JSP du formulaire «connect.xhtml»

```
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:f="http://java.sun.com/jsf/core">
<h:head><title>Authentification</title></h:head>
<h:body>
  <h:form>
    <h:panelGrid border="1" columns="2">
      <h:outputText value="Adresse Mail"></h:outputText>
      <h:inputText value="#{connectBean.email}" />

      <h:outputText value="Mot de Passe"></h:outputText>
      <h:inputSecret value="#{connectBean.password}" />

      <h:commandButton value="Valider" type="submit"
action="#{connectBean.connect}" />
      <h:commandButton value="Reset" type="reset" />
    </h:panelGrid>
  </h:form>
</h:body>
</html>
```

Accès aux attributs du Bean

Appel à la méthode connect() du Bean

5. Navigation : configuration de faces-config.xml

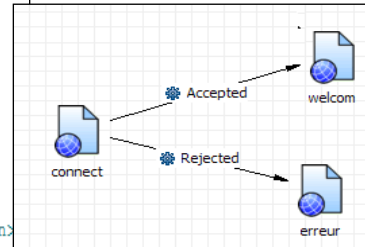
- FacesServlet décide la ressource à appeler suite à la réception d'un message grâce à la configuration décrite dans le faces-config.xml
 - Les messages sont des simples chaînes de caractères
 - Utilisation de la balise **<navigation-rule>** pour paramétrer les règles de navigation
- La balise **<from-view-id>** indique la vue source où est effectuée la demande de la redirection. La vue peut être un :
 - Formulaire (action de soumission)
 - Lien hypertext
- Pour chaque valeur de message une page de direction est indiquée dans la balise **<navigation-case>**
 - **<from-action>** : la méthode du Managed Bean qui a généré la clé
 - **<from-outcome>** : la valeur du message (clé de navigation)
 - **<to-view-id>** : la vue de direction, la page JSP/XHtml
- Pour schématiser nous distinguons deux sortes de navigation
 - **Navigation statique**: La valeur de l'**Outcome** est connue au moment de l'écriture de la JSP
 - **Navigation dynamique**: La valeur de l'**Outcome** est inconnue au moment de l'écriture de la Vue. Elle peut être calculée par un Bean Managé ou autre chose comme nous l'avons vu dans l'exemple du formulaire.

Suite de l'exemple d'un formulaire de connexion (Navigation Dynamique)

- Afin de pouvoir exécuter le formulaire, il faut définir des règles de navigation dans le Faces-config.xml qui correspond aux états retourné par le ManagedBean

```
<navigation-rule>
  <display-name>connect</display-name>
  <from-view-id>/connect.jsp</from-view-id>
  <navigation-case>
    <from-action>#{connectBean.connect}</from-action>
    <from-outcome>Accepted</from-outcome>
    <to-view-id>/welcom.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
<navigation-rule>
  <display-name>connect</display-name>
  <from-view-id>/connect.jsp</from-view-id>
  <navigation-case>
    <from-action>#{connectBean.connect}</from-action>
    <from-outcome>Rejected</from-outcome>
    <to-view-id>/erreur.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
</faces-config>
```

Définition des règles de Navigation Faces-config.xml



5. Navigation statique

- Appel direct à un message de navigation sans passer par une méthode d'action => clé de navigation statique

```
<f:view>
  <html><head><title>Connect</title></head><body>
    <h:form>
      <h:panelGrid border="1" columns="2">
        <h:outputText value="Adresse Mail"></h:outputText>
        <h:inputText value="#{connectBean.email}" />
        <h:outputText value="Mot de Passe"></h:outputText>
        <h:inputSecret value="#{connectBean.password}" />
        <h:commandButton value="Valider" type="submit" action="valider" />
        <h:commandButton value="Reset" type="reset" />
      </h:panelGrid>
    </h:form>
```

Redirection vers Welcom.jsp sans passer par le ManagedBean

Pas de balise <from-action>

```
<navigation-rule>
  <display-name>connect</display-name>
  <from-view-id>/connect.jsp</from-view-id>
  <navigation-case>
    <from-outcome>valider</from-outcome>
    <to-view-id>/welcom.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

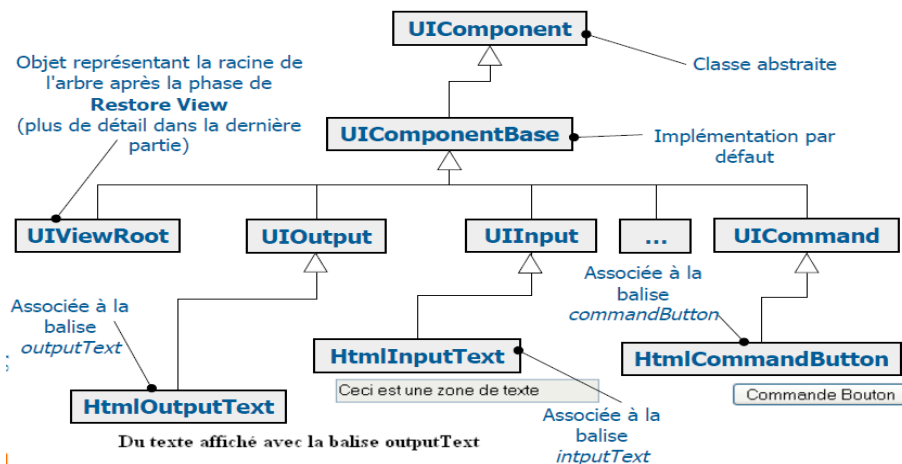
7. Principes de composants graphiques JSF

- Chaque composant graphique est défini par une classe Java qui décrit ses fonctionnalités.
- Plusieurs rendus pour un même composant
- Possibilité de définir des rendus
- Chaque composant génère **des événements (Events)**
- Le serveur possède des écouteurs (**Listeners**) qui traitent les événements de chaque composant UI (user interface)
- Les composants graphiques sont appelés par des balises spécialisées dans les pages JSP/XHtml.
- Elles peuvent aussi être manipulées à partir de ses classes Java dans les Managed Bean

42

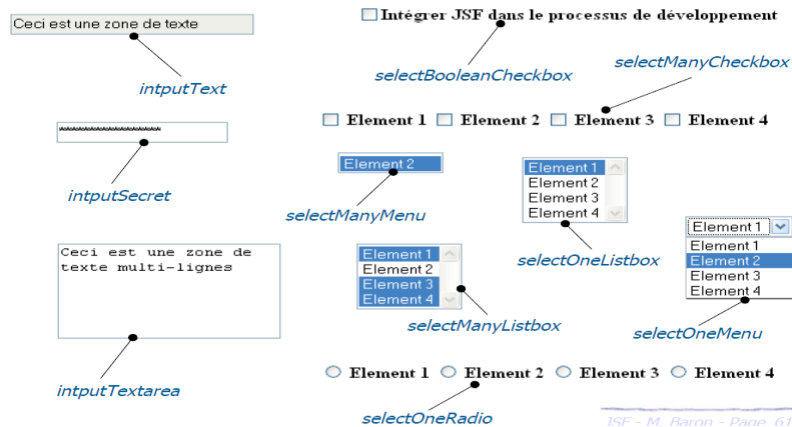
7. Composants Graphiques de JSF

- Le Framework JSF offre la possibilité d'encapsuler tout ou partie des composants UI qui composent une vue.
- Ci-dessous l'hierarchie des principaux composants JSF



7. Composants Graphiques de JSF

- JSF fournit un ensemble de composants graphiques pour la conception de l'IHM



7. Composants Graphiques de JSF



7. Composants <h:dataTable>

- Le composant `<html:dataTable>` permet de visualiser des données sur plusieurs colonnes et plusieurs lignes
- La table peut être utilisée quand le nombre d'éléments à afficher est inconnu
- Attributs de la balise :
 - `value` : une collection de données (Array, List, ResultSet,...)
 - `var` : nom donné à la variable à manipuler pour chaque ligne
 - `border`, `bgcolor`, `width` : attributs pour l'affichage
 - `rowClasses`, `headerClass` : attributs pour la gestion des styles (CSS)
- Pour chaque colonne de la table, la valeur à afficher est obtenue par la balise `<h:column>`
- Exemple :

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

7. Composants <h:dataTable>

- La modification des en-tête et pied d'une table est obtenue en utilisant la balise `<c:facet>`
- Pour le composant table trois filiations possibles
 - `header` : une filiation entre une colonne et le nom de la colonne
 - `footer` : une filiation entre la table et un nom
 - `caption` : une filiation entre le titre de la table et un nom

```
<h:dataTable var="someVar" value="#{someCollection}">
  <h:column>
    <f:facet name="header">First Heading</f:facet>
    #{someVar.property1}
  </h:column>
  ...
</h:dataTable>
```

```

public class DataTableBean {
    private List<Personne> refPersonne;
    public List getPersonne() {
        if (refPersonne == null) {
            refPersonne = new ArrayList<Personne>();
            refPersonne.add(new Personne("Baron", "Mickael", "17081976", "Développeur"));
            refPersonne.add(new Personne("Dupont", "Marcel", "21041956", "Boucher"));
            ...
        }
        return refPersonne;
    }
}

```

OutputBean.java

```

<core:view>
    <html:dataTable value="#{outputbean.personne}" var="personne" border="1"
        cellspacing="4" width="60%" >
        <html:column>
            <core:facet name="header" >
                <html:outputText value="Nom" />
            </core:facet>
            <html:outputText value="#{personne.nom}" />
        </html:column>
        <html:column>
            <core:facet name="header" >
                <html:verbatim>Prénom</verbatim>
            </core:facet>
            <html:outputText value="#{personne.firstname}" />
        </html:column>
        <html:column>
            <core:facet name="header" >
                Date de naissance
            </core:facet>
            <html:outputText value="#{personne.birthdata}" />
        </html:column>
        <html:facet name="footer">
            <html:outputText value="#{outputbean.caption}" />
        </html:facet>
        ...
    </html:dataTable>
</core:view>

```

Nom	Prénom		Emploi
Baron	Mickael	17081976	Ingénieur d'étude et de développement
Dupont	Marcel	21041956	Boucher
Martin	Alexandre	28011946	Magicien
Fox	Tracy	18021969	Sexologue
Une Simple Table			

Ne sera jamais affiché car
<core:facet> n'est associé
à aucun autre composant

datatablecomponent2.jsp
du projet
GraphicalComponents

48

8. FacesContext : principe

- **FacesContext** permet de représenter toutes les informations contextuelles associées à la requête et à la réponse
 - Notez qu'il y a autant d'instances de type FacesContext qu'il y a de vues JSF
 - FacesContext est essentiellement utilisé par les mécanismes internes de JSF. Toutefois il est possible d'en extraire des informations intéressantes
- Un objet **FacesContext** est exploitable dans un Bean ou dans une Vue via son objet implicite associé. On peut en extraire:
 - **ExternalContext** : accéder aux éléments de la requête et de la réponse
 - **Message Queue** : stocker des messages
 - **ViewRoot** : accéder à la racine de l'arbre des composants (UIViewRoot)
 - Modifier le déroulement du cycle de vie
- Accès au objet HttpServletRequest, HttpServletResponse via le code suivant:

```

ExternalContext context =
FacesContext.getCurrentInstance().getExternalContext();

HttpServletRequest request = (HttpServletRequest) context.getRequest();

HttpServletResponse
response= (HttpServletResponse) context.getResponse();

```


9. Classe Backing Bean

- Un **Backing Bean** est un Managed Bean dont certaines propriétés sont de type **UIComponent**
- Dans le Bean, il faut déclarer des accesseurs et des modificateurs sur des propriétés de type **UIComponent**

```
package beanPackage;
public class BackingBeanExemple {
    private String nom;
    private UICommand refCommand;

    public String getNom() { return nom; }
    public void setNom(String pNom) { nom = pNom; }

    public void setNomAction(UICommand ref) {
        refCommand = ref;
    }
    public UICommand getNomAction() {
        return refCommand;
    }
}
```

Le « Backing Bean »
se comporte comme
un Bean Managé

Stocker et relayer la
référence d'un
composant JSF

➤ Au niveau de la page JSP, la **liaison** entre le « Backing Bean » est la vue se fait par l'attribut **binding**

```
<html:commandButton binding="#{backingbean.nomAction}" />
```

9. Activer ou désactiver un bouton par un Bean

Page JSP

```
<html:form>
  <html:inputText value="#{backingbean.name}" binding="#{backingbean.composantNom}" />
  <html:commandButton value="Transformer" binding="#{backingbean.commandButton}"
    action="#{backingbean.doProcess}" />
</html:form>
```

```
package beanPackage;
public class BackingBean {
    private String nom = "Baron";
    private HtmlInputText composantNom;
    private HtmlCommandButton commandButton;

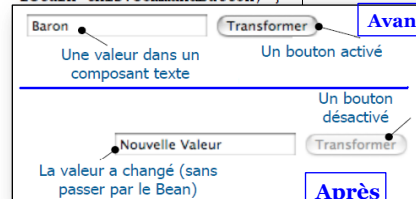
    public String getName() { return name; }
    public void setName(String pName) { this.name = pName; }

    public void setComposantNom(HtmlInputText pCommand) { composantNom = pCommand; }
    public HtmlInputText getComposantNom() { return composantNom; }

    public void setcommandButton(HtmlCommandButton pCB) { this.commandButton = pCB; }
    public HtmlCommandButton getCommandButton() { return this.commandButton; }

    public void doProcess() {
        if (commandButton != null) {
            this.commandButton.setDisabled(true);
        }
        if (composantNom != null) {
            composantNom.setValue("Nouvelle Valeur");
        }
    }
}
```

Composant UI Encapsulé



10. Message de validation: création dans les Beans

- L'API JSF fournit l'objet **FacesMessage** pour empiler des messages qui pourront être affichés dans une page XHTML
- Un objet FacesMessage est caractérisé par :
 - **Sévérité** : SEVERITY_INFO, SEVERITY_WARN, SEVERITY_ERROR et SEVERITY_FATAL
 - **Résumé** : texte rapide décrivant le message
 - **Détail** : texte détaillé: le message à affiché
- Pour envoyer des messages vers une Vue, il faut passer par le contexte courant de JSF (**FacesContext**)
 - **addMessage(String id, FacesMessage message)** : ajoute un message à un composant (identifié par id)
 - Si **id** est **null** le message n'est pas associé à un composant
- Exemple de construction et transmission d'un FacesMessage :

```
// Déclaration d'un Message
FacesMessage myFacesMessage = new FacesMessage();
myFacesMessage.setSeverity(FacesMessage.SEVERITY_INFO);
myFacesMessage.setSummary("Un petit résumé");
myFacesMessage.setDetail("Mon message qui ne sert à rien");

// Transmission d'un Message
FacesContext myFacesContext = FacesContext.getCurrentInstance();
myFacesContext.addMessage(null, myFacesMessage);
```

10. Message de validation : affichage

- La bibliothèque HTML propose deux balises personnalisées pour afficher les messages suite à la validation de champs de saisie d'un formulaire
 - **<html:messages>** : affiche tous les messages envoyés par le bean
 - **<html:message>** : affiche les messages associés à un identifiant '**id**' d'un composant du formulaire
- Les balises contiennent les attributs suivants :
 - **for** : indique l'id du composant (uniquement pour message)
 - **showDetail** : booléen qui précise si le message est détaillé
 - **showSummary** : booléen qui précise si le message est résumé
 - **tooltip** : booléen qui précise si une bulle d'aide est affichée

53

Exemple avec <h:messages />

```

public String authentication() {
    FacesContext fc = FacesContext.getCurrentInstance();
    if ("admin".equals(this.login) && "admin".equals(this.password)) {
        message = "Bienvenu " + login + ", à l'application de gestion de Bibliothèque";
        return "success";
    } else {
        if (!"admin".equals(this.login)) {
            fc.addMessage(null, new FacesMessage("Login " + this.login + " inexistant !!"));
        }
        if (!"admin".equals(this.password)) {
            fc.addMessage(null, new FacesMessage(message = "Vérifiez votre mot de passe !!"));
        }
        return null;
    }
}

```

```

<h:form>
<h:panelGrid columns="2">
    <h:outputLabel value="Login:" />
    <h:inputText value="#{loginBean.login}" size="30" />

    <h:outputLabel value="Mot de passe:" />
    <h:inputSecret id="pwdId" value="#{loginBean.password}" size="30" />

    <h:commandButton action="#{loginBean.authentication()}" value="Valider" type="submit" />
</h:panelGrid>
</h:form>
<h:messages style="COLOR: #ff0000;" />

```

8686/BibliothequeWebJSF2.0/pages/login.jsf

Authentification

Login:

Mot de passe:

- Login mjml inexistant !!
- Vérifiez votre mot de passe !!

54

Exemple avec <h:message />

Il faut ajouté un 'id' pour chaque composant du formulaire

```

<h:form id="loginForm">
    <h:panelGrid border="0" columns="3" style="width: 497px; height: 88px">
        <h:outputLabel value="Login"></h:outputLabel>
        <h:inputText value="#{loginBean.login}" id="loginId"></h:inputText>
        <h:message for="loginId" tooltip="true" showDetail="true" showSummary="true" />

        <h:outputLabel value="Mot de Passe"></h:outputLabel>
        <h:inputSecret id="pwd" value="#{loginBean.pwd}" id="pwdId"/>
        <h:message for="pwdId" tooltip="true" showDetail="true" showSummary="true" style="color: red;" />

        <h:commandButton value="Login" action="#{loginBean.authentication()}" />
        <h:commandButton id="resetID" value="Reset" />
    </h:panelGrid>
</h:form>

```

```

public String authentication() {
    FacesContext fc = FacesContext.getCurrentInstance();
    if ("admin".equals(this.login) && "admin".equals(pwd)) {
        message = this.login;
        return "success";
    } else {
        if (!"admin".equals(this.pwd)) {
            fc.addMessage("loginForm:pwdId", new FacesMessage(
                FacesMessage.SEVERITY_INFO, "Validation Pwd",
                "Vérifiez votre mot de passe !!"));
        }
        if (!"admin".equals(this.login)) {
            fc.addMessage("loginForm:loginId", new FacesMessage(
                FacesMessage.SEVERITY_INFO, "Validation Login",
                "Login " + this.login + " inexistant !!"));
        }
        return "erreur";
    }
}

```

localhost:8686/BibliothequeWebJSF2.0/pages/login.jsf

Authentification

Login: Login jjjjjjh inexistant !!

Mot de passe: Vérifiez votre mot de passe !!

11. L'internationalisation I18N

- **JSF** offre la possibilité d'internationaliser ses applications, c'est-à-dire d'adapter la langue utilisée dans l'interface (français, anglais...). Pour ce faire, il faut :
 - Ne pas écrire directement sur les JSP, les différents libellés des champs d'un formulaire et les différents messages à afficher sur la Vue. Il faut passer par les clés définies dans un fichier de ressources.
 - Un fichier de ressources comprend une liste de clés et leur valeur associée. Les clés seront utilisées dans les JSP alors que les valeurs seront affichées à l'utilisateur.
- Le fichier ressource doit :
 - Porter comme extension « **.properties** », (eg: **toto.properties**)
 - Pour chaque langage, il faut créer un fichier de ressource spécifique
 - **MessageResources.properties** : ce fichier contient les messages de la langue par défaut pour votre application.
 - **MessageResources_XX.properties** : ce fichier contient les mêmes messages dans la langue dont le code ISO est XX (fr, en, it, es)
- Les fichiers des ressources doivent être placés directement au dessus du 'src' de votre projet ou bien dans un répertoire 'resources' aussi au dessus de 'src' comme le montre la figure.

Exemple d'un fichier de ressource

ApplicationMessages.properties (Français)

```
#messages d'erreur et d'info pour formulaire HelloForm et action HelloAction
com.objis.demonstruts.dont.talk.to.atilla=Je vous ai demandé de ne pas dire Hello à Atilla!!!
com.objis.demonstruts.no.person.error=SVP entrez le nom de la <i>PERSONNE</i> à qui dire hello
#Resources spécifiques à la page hello.jsp
hello.jsp.title=Accueil - premier programme Struts
hello.jsp.page.heading=Hello World! Un premier programme Struts
hello.jsp.prompt.person=A qui souhaitez vous dire Hello ?
```

ApplicationMessages_en.properties(Anglais)

```
#messages d'erreur et d'info pour formulaire HelloForm et action HelloAction
com.objis.demonstruts.dont.talk.to.atilla=I have asked you not to rewrite Atilla!!!
com.objis.demonstruts.no.person.error=Please enter the name of the <i>Person</i> to
#Resources spécifiques à la page hello.jsp
hello.jsp.title=Hello - My first Struts Program
hello.jsp.page.heading=Hello World! My first Struts Program
hello.jsp.prompt.person=To whom you want to say Hello ?
```



11. Internationalisation

- **Faces-config.xml** permet de configurer les langues acceptées pour l'application WEB dans la balise **<application>** qui se compose des sous balises suivants:

```
<application>
  <resource-bundle>
    <base-name>messages</base-name>
    <var>msgs</var>
  </resource-bundle>
</application>
```

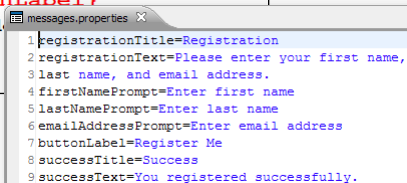
- La balise **<resource-bundle>** permet de déclarer l'emplacement de fichier de ressource à utiliser.
 - **<base-name>** : le nom de base du fichier de ressource. Dans l'exemple, ce fichier est **'messages.properties'** et il est placé directement au dessus du **'src'**
 - **<var>** : la variable qui fera référence au fichier de ressource. Elle sera utilisée dans les Vues pour accéder aux messages.
- Si les fichiers de ressources sont créés au dessous du package **'resources'** (**WEB-INF/classes/resources/messages.properties**), :

```
<application>
  <resource-bundle>
    <base-name>resources.messages</base-name>
    <var>msgs</var>
  </resource-bundle>
</application>
```

11. Internationalisation

- L'appel à un message de ressource s'effectue directement dans le vues via EL en utilisant le **'var'** déclaré dans le faces-config:
 - **#{msgs.name}**
- JSF détermine la langue par défaut en se référant à la langue configuré au niveau de votre navigateur

```
<h:form>
  #{msgs.firstNamePrompt} :
  <h:inputText value="#{person1.firstName}" />
  <br/>
  #{msgs.lastNamePrompt} :
  <h:inputText value="#{person1.lastName}" />
  <br/>
  #{msgs.emailAddressPrompt} :
  <h:inputText value="#{person1.emailAddress}" />
  <br/>
  <h:commandButton value="#{msgs.buttonLabel}"
    action="#{person1.d
  </h:form>
</h:body></html>
```



```
1 registrationTitle=Registration
2 registrationText=Please enter your first name,
3 last name, and email address.
4 firstNamePrompt=Enter first name
5 lastNamePrompt=Enter last name
6 emailAddressPrompt=Enter email address
7 buttonLabel=Register Me
8 successTitle=Success
9 successText=You registered successfully.
```

12. Templating avec JSF/Facelets

- Un **template** est une page *xhtml* ordinaire qui définit la structure du document avec des emplacements spécifiques où les pages qui utilisent ce *template* (pages clientes) inséreront leur contenu.
- Les tag responsables de la création et l'utilisation des *template* font partie du taglib '**facelets**' généralement référencé par '**ui**'

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:ui="http://java.sun.com/jsf/facelets">
```

- Les étapes de base :
 - Création d'un fichier ordinaire 'template.xhtml' :
 - Le contenu qui apparaîtra dans tous les pages clientes est entré directement dans le fichier *template*
 - Le contenu qui peut être remplacé dans le pages clientes sera entouré par le tag '**ui:insert name="section"**'
 - Création des pages clientes, qui utiliseront le template :
 - Le contenu de ces pages sera entouré par la balise '**ui:composition**'. Cette balise indique le chemin du fichier '**template**' qui sera utilisé.
 - Utilisé la balise '**ui:define**' pour réécrire le contenu de chaque section remplaçable indiqué dans le modèle (marqué dans le modèle avec **ui:insert**)

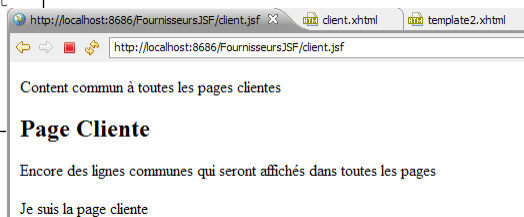
12. Templating avec JSF/Facelets

Exemple de 'template.xhtml'

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:ui="http://java.sun.com/jsf/facelets">
<h:body>
  Content commun à toutes les pages
  clientes
  <h2><ui:insert name="title">
    Un titre qui pourra être remplacé
  </ui:insert></h2>
  Encore des lignes communes qui seront
  affichés dans toutes les pages
  <ui:insert name="body">
    Un corp de page par défaut qui peut
    être remplacé
  </ui:insert>
</h:body>
</html>
```

Exemple de page cliente

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:ui="http://java.sun.com/jsf/facelets">
<ui:composition
  template="./templates/template.xhtml">
  <ui:define name="title"> Page Cliente</ui:define>
  <ui:define name="body">
    <p>Je suis la page cliente
  </ui:define>
</ui:composition>
</html>
```



12. Templating avec JSF/Facelets

- Dans le fichier template:
 - Le paramètre '**name**' du le tag `<ui:insert name="XXX">` indique le nom de la section qui peut être modifié chez la page cliente: en-tête, menu, body etc.
- Dans le fichier xhtml de la page cliente :
 - `<ui:composition template="/template.xhtml">` fait référence au fichier du template à utilisé
 - `<ui:define name="XXX">` remplace le contenu d'une section déclaré dans le fichier template.
 - `<ui:include src=" " path d'un fichier xhtml" >` permet d'insérer un fichier dans un autre.
- Généralement, un contenu commun à toutes les page est mis dans le fichier du template, un contenu spécifique est mis directement dans la page cliente et un contenu, qui peut être appelé par certaines pages et d'autres non, sera mis dans un fichier séparé et appelé par la balise 'ui:include'

12. Exemple évolué de Templating

The diagram illustrates the templating structure for a web application. It shows two browser screenshots with annotations indicating the inclusion of various templates.

Top Screenshot (welcome.jsf):

- header.xhtml:** Points to the navigation bar at the top.
- welcome.xhtml:** Points to the main content area containing the aircraft carrier image.
- Search-menu.xhtml:** Points to the search menu on the left side.
- footer.xhtml:** Points to the footer at the bottom.

Bottom Screenshot (fournisseursList.jsf):

- fournisseursList.xhtml:** Points to the main content area displaying a table of suppliers.
- Search-menu.xhtml:** Points to the search menu on the left side.
- footer.xhtml:** Points to the footer at the bottom.

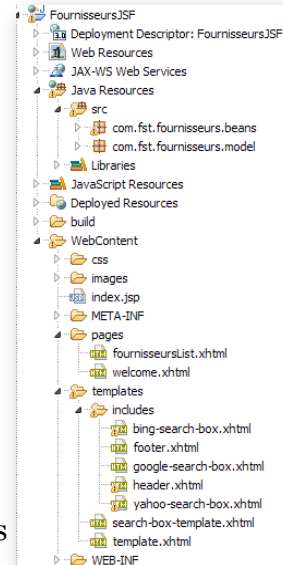
Table of Suppliers (from fournisseursList.xhtml):

Supply name	Company name	Address	Action
Salah	Mohamed	Manar 1	DeleteDelete
Fakfakh	Ahmed	Sfax Centre	DeleteDelete
Fakfakh	Ahmed	Sfax Centre	DeleteDelete
Fakfakh	Ahmed	Sfax Centre	DeleteDelete
Fakfakh	Ahmed	Sfax Centre	DeleteDelete
Fakfakh	Ahmed	Sfax Centre	DeleteDelete
Fakfakh	Ahmed	Sfax Centre	DeleteDelete
Fakfakh	Ahmed	Sfax Centre	DeleteDelete

7 (suppliers were found)

12. Exemple évolué de Templating

- Le projet est constitué de deux pages clientes 'welcome' et 'fournisseursList' qui respectent le même modèle d'IHM.
- Le fichier 'template' fait appel à des fichiers communs à toutes les pages:
 - Footer.xhtml
 - Header.xhtml
 - Google-search-box.xhtml, ce fichier respecte le template 'search-box-template.xhtml'
- Les fichiers du projet sont regroupés dans des répertoires différents:
 - Pages : les pages clientes
 - Templates: fichiers de template
 - Template/include : fichier d'inclusion
 - Images: les images utilisés dans les pages
 - Css : fichiers de style.



12. Exemple évolué de Templating

```
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets">
<h:head><title><ui:insert name="title">Title</ui:insert></title>
<link rel="stylesheet" type="text/css" href="../css/styles.css"/>
</h:head>
<h:body>
<ui:insert name="header">
  <ui:include src="../templates/includes/header.xhtml"></ui:include>
</ui:insert>
<p>
<table border="5" align="center">
  <tr><th class="title">
    <ui:insert name="title">Title</ui:insert>
  </th></tr>
</table><p>

<table width="75" align="left" cellspacing="5">
<tr><td>
<ui:insert name="menu">
  <ui:include src="../templates/includes/google-search-box.xhtml"/>
</ui:insert></td></tr>
</table><p>
<ui:insert name="body">Body</ui:insert>
<br clear="all"/> <hr/>
<ui:insert name="footer">
  <ui:include src="../includes/footer.xhtml"/>
</ui:insert>
</h:body></html>
```

C'est le fichier du
"Template"
le reste de 'src' du projet
sont joint au support du
cours

- Les projets exemples joints au support du cours :
 - Basics : exemple de projet JSF basique
 - data-tables : exemple de projet JSF avec différents cas d'utilisation des tableaux
 - FournisseursJSF : exemple basique de JSF avec Templating et tableau
 - FournisseursJSFInternational : le projet de FournisseursJSF avec Internationalisation

Exemple de Bibliothèque de Composants pour JSF

RichFaces

<http://richfaces-showcase.appspot.com/richfaces/component->



RichFaces

- **Richfaces** est né du besoin d'associer la technologie Ajax à celle de JSF.
- Il comporte un ensemble de composants permettant d'ajouter des fonctionnalités Ajax avancées aux composants standards JSF sans manipulation du code JavaScript.
- **Richfaces** permet aussi d'enrichir les composants graphiques de JSF et d'ajouter des nouveaux composants.
- L'utilisation de **RichFaces** n'apporte pas des modifications au niveau de Managed Bean, ni au cycle général de développement de JSF. C'est seulement au niveau des Vues qu'on pourra désormais bénéficier de nouvelles balises prédéfinies.

Intégration de RichFaces

- Ajout de déclaration dans le Web.xml

```
<context-param>
  <param-name>org.richfaces.SKIN</param-name>
  <param-value>classic</param-value>
</context-param>
<context-param>
  <param-name>org.ajax4jsf.VIEW_HANDLERS</param-name>
  <param-value>com.sun.facelets.FaceletViewHandler</param-value>
</context-param>
<filter>
  <display-name>RichFaces Filter</display-name>
  <filter-name>richfaces</filter-name>
  <filter-class>org.ajax4jsf.Filter</filter-class>
</filter>
<filter-mapping>
  <filter-name>richfaces</filter-name>
  <servlet-name>Faces Servlet</servlet-name>
  <dispatcher>REQUEST</dispatcher>
  <dispatcher>FORWARD</dispatcher>
  <dispatcher>INCLUDE</dispatcher>
</filter-mapping>
```

Défini le style (Skin) générale des pages, d'autres (wine, cherry)

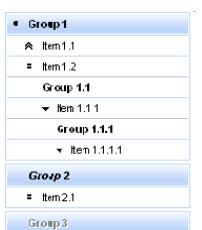
Intégration de RichFaces

- Pour ajouter les balises richfaces dans une Facelet, on ajoute à l'entête de la page les taglib A4J et Rich.

```
<!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:ui="http://java.sun.com/jsf/facelets"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:a4j="http://richfaces.org/a4j"
    xmlns:rich="http://richfaces.org/rich" >
```

- La bibliothèque `a4j` (Ajax for JSF) permet d'ajouter des traitements Ajax aux pages JSF.
- La bibliothèque `rich` contient les composants graphiques de richfaces.
- Il faut ajouter au projet les Jar de la bibliothèque RichFaces.

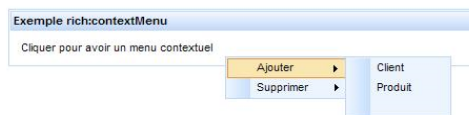
Exemple de composants RichFaces



`<rich:panelMenu>`



`<rich:tabPanel/>`



`<rich:dataTable/>`

Client(s) found(s): 10		
Name	Company	Phone
Client 0Name	company for Client 0	01-02-03-04-00
Client 1Name	company for Client 1	01-02-03-04-01
Client 2Name	company for Client 2	01-02-03-04-02
Client 3Name	company for Client 3	01-02-03-04-03
Client 4Name	company for Client 4	01-02-03-04-04

The screenshot shows a web application titled "Example richcontextMenu". It contains a table with two columns: "Nom" and "Entreprise". The table lists ten clients, each with a unique ID (Pau0 to Pau9) and the company name "company for Client X". A context menu is displayed over the table, with the "Operations" option selected. The menu has two sub-items: "Modifier Pau1Name" and "Supprimer Pau1Name".

Nom	Entreprise
Pau0Name	company for Client 0
Pau1Name	company for Client 1
Pau2Name	company for Client 2
Pau3Name	company for Client 3
Pau4Name	company for Client 4
Pau5Name	company for Client 5
Pau6Name	company for Client 6
Pau7Name	company for Client 7
Pau8Name	company for Client 8
Pau9Name	company for Client 9

Operations

- Modifier Pau1Name
- Supprimer Pau1Name

A map of North America and surrounding regions, including Greenland, Canada, the United States, Mexico, Cuba, Venezuela, Colombia, and parts of Central America. The map is overlaid with a grid. A red box highlights a specific area in the North Pacific Ocean, west of the United States. A scale bar at the bottom left indicates 1000 miles and 2000 kilometers. The map is sourced from Google, as indicated by the logo at the bottom left.

72

- **<rich:tabPanel/>**

composants pour le client

- ```
<rich:panel>
 <f:facet name="header"> Rich Panel </f:facet>
 <h:outputText value="Example panel" />
</rich:panel>
```

36

## Composants Graphiques de RichFaces

- **<rich:panelBar/>**

```
<rich:panelBar height="150" width="500">
 <rich:panelBarItem label="Panel Client"> Clients
</rich:panelBarItem>
 <rich:panelBarItem label="Panel Commandes">
 commandes
 </rich:panelBarItem>
</rich:panelBar>
```



- **<rich:simpleTogglePanel />**: Représente un panel dont le contenu peut être affiché ou caché, grâce à un clic sur son titre.



## Composants Graphiques de RichFaces

- **<rich:contextMenu />**: c'est un menu que l'on peut attacher à un autre composant JSF, il s'active par un événement JavaScript. L'exemple suivant présente un contextMenu attaché au composant rich:panel.

```
<rich:panel id="panel">
 <f:facet name="header">
 <h:outputText value="Exemple rich:contextMenu" />
 </f:facet>
 <h:outputText value="Cliquer pour avoir un menu contextuel" />
<rich:contextMenu event="onclick" attachTo="panel" submitMode="ajax">
 <rich:menuGroup value="Ajouter">
 <rich:menuItem value="Client" />
 <rich:menuItem value="Produit" />
 </rich:menuGroup>
 <rich:menuGroup value="Supprimer">
 <rich:menuItem value="Client" />
 <rich:menuItem value="Produit" />
 </rich:menuGroup>
</rich:contextMenu>
</rich:panel>
```



## Exemple avec le composant <rich:dataTable/>

- **rich:dataTable** apporte plusieurs fonctionnalités absentes dans le composant standard **h:dataTable**. En effet, **rich:dataTable** dispose des facets header et footer, il implémente les paramètres HTML **rowspan** et **colspan**. De plus, il peut contenir des sub-tables, et fournit des outils pour **filtrer**, **ordonner** les colonnes et paginer les lignes.

Exemple dataTable

| Client(s) found(s): 10 |                      |                |
|------------------------|----------------------|----------------|
| Name                   | Company              | Phone          |
| +                      | <input type="text"/> |                |
| Client 0Name           | company for Client 0 | 01-02-03-04-00 |
| Client 1Name           | company for Client 1 | 01-02-03-04-01 |
| Client 2Name           | company for Client 2 | 01-02-03-04-02 |
| Client 3Name           | company for Client 3 | 01-02-03-04-03 |
| Client 4Name           | company for Client 4 | 01-02-03-04-04 |

« « 1 2 3 » »

```

rich:dataTable
<rich:panel style="border:0;width:60%;text-align:center">
 <rich:panel>
 <f:facet name="header">
 Exemple dataTable
 </f:facet>
 <rich:dataTable cellpadding="0" cellspacing="0" border="0"
 var="list" value="#{testBean.persons}" id="table"
 style="text-align:center; rows="5" width="100%">
 <f:facet name="header">
 <rich:columnGroup>
 <rich:column colspan="3">
 <h:outputText
 value="Client(s) found(s): #{testBean.personsSize}" />
 </rich:column>
 <rich:column breakBefore="true">
 <h:outputText styleClass="headerText" value="Name" />
 </rich:column>
 <rich:column>
 <h:outputText styleClass="headerText" value="Company" />
 </rich:column>
 <rich:column>
 <h:outputText styleClass="headerText" value="Phone" />
 </rich:column>
 </rich:columnGroup>
 </f:facet>
 <rich:column sortBy="#{list.name}">
 <h:outputText value="#{list.name}" />
 </rich:column>
 <rich:column filterEvent="onkeyup" filterBy="#{list.company}">
 <h:outputText value="#{list.company}" />
 </rich:column>
 <rich:column>
 <h:outputText value="#{list.phone}" />
 </rich:column>
 </rich:dataTable>
 <rich:dataScroller for="table"></rich:dataScroller>
 </rich:panel>
</rich:panel>

```

**Pagination du tableau : (5 lignes par page)**

**Trier cette colonne**

**recherche d'un élément de cette colonne**