

# Java Server Faces

Reda Bendraou

*Ce support s'inspire des lectures citées à la fin de la présentation*

# Plan

- Introduction à JSF
  - JSF: les Avantages / Inconvénients
  - Les nouveautés dans JSF
- Les Managed Bean et les Facelets
- La Navigation
- EL:Expression Language
- Les Messages et l'i18n
- La Validation des formulaires
- L'affichage des collections à longueur variables

# Introduction

# JSF: Présentation

- **Java Server Faces** est un Framework de développement d'applications Web en Java permettant de respecter le modèle d'architecture **MVC** et basé sur des composants côté présentation
  - Version actuelle 2.2 (sortie en Avril 2013)
- **JSF permet**
  - une séparation de la couche présentation des autres couches (MVC)
  - un ensemble de composants riches et réutilisables
  - une liaison simple entre les actions côté client de l'utilisateur et le code Java côté serveur
- **Autres alternatives**
  - Struts, JSP/Servlets, Spring MVC, Apache Wicket et Tapestry

# JSF: Comparaison avec l'existant (JSP/Servlet)

- Une palette de composants graphiques plus riche
  - Possibilité de créer ces propres composants
- Plus de flexibilité dans l'invocation de code coté serveur lors des clicks sur les boutons coté client
- La notion de « Managed Bean »: permet de peupler des objets métiers à partir de la soumission d'un formulaire
- Intégration avec Ajax
  - Utilisation de tags simples, pas besoin de coder explicitement du JavaScript
- Plus de fonctionnalités pour la validation de formulaire
- **Les moins:** moins transparent, peu documenté comparé aux JSP/Servlet

# JSF: Comparaison avec l'existant (Struts)

- Struts limité à HTML
  - D'autres formats d'affichage sont fournis par JSF (XUL, WML, XML, etc.)
- Flexibilité des Bean JSF comparé à Struts
  - Possibilité d'y faire référence dans le formulaire alors que Struts, le processus est plus compliqué
- EL (Expression Language) de JFS plus puissant que le bean:write de Struts 1.x
- JSF n'exige pas que vos classes héritent d'autres classes comme dans Struts pour les Actions et les Forms
- Fichier de config xml plus simple dans JSF
- JSF fait partie du standard JEE, pas Struts
- **Les moins:** Maturité de Struts, largement utilisé dans l'industrie comparé à la jeunesse de JSF
- Fonctionnalités de validation plus importante dans Struts

# JSF: Implémentations

- Sun/Oracle Mojarra
  - Main page: <http://javaserverfaces.java.net/>
- Apache MyFaces
  - Main page: <http://myfaces.apache.org/core20/>
- Any Java EE 6 server
  - JSF 2.0 fait partie officielle de Java EE 6
  - JBoss 6, Glassfish 3, WebLogic 11, WebSphere 8,

# JSF2.0: Nouveautés

- Les Annotations ce qui évite d'écrire des lignes dans faces-config.xml
- Le support d'Ajax
- Utilisation des facelets
- Plus de composants et de validateurs
- Le support de Groovy
- Plusieurs paramètres considérés comme étant par défaut



# JSF2.0:Environnement de Développement

- Java
  - Tomcat 6=> Java 5 et plus
  - Tomcat 7=> Java 6 et plus
- IDE
  - Eclipse, 3.6 et plus
- Serveur
  - Tomcat 6 et plus + deux jars: jsf-api.jar & jsf-impl.jar
    - À mettre dans le répertoire WEB-INF/lib de votre projet web
  - Jboss 6 et plus, Galssfish 3
- JSF implémentations
  - Oracle [Mojarra](#) ou Apache MyFaces

# JSF2.0: Modification du Web.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" version="2.5">
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.jsf</url-pattern>
  </servlet-mapping>
  <context-param> <!-- pour plus de messages d'erreurs, détails-->
    <param-name>javax.faces.PROJECT_STAGE</param-name>
    <param-value>Development</param-value>
  </context-param>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
</web-app>
```

# JSF2.0: le fichier faces-config.xml

- Si vous utilisez les annotations, le fichier contiendra peu d'entrées

```
<?xml version="1.0"?>
<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">
  <!-- Empty for now. Your entries will go here. But even
    if you have no entries in faces-config.xml, you are required
    to have a valid faces-config.xml file with legal
    start and end tags. -->
</faces-config>
```

# Les Managed Beans & Les Facelets

# Managed Bean

- Bean: classe Java avec des conventions d'écriture
  - Un constructeur sans arguments
  - Pas de propriétés d'instance en public
  - Les valeurs persistantes doivent avoir des getters/setters
  - Etc.,
- Managed Bean: des Beans gérés automatiquement par JSF
  - Instanciation et cycle de vie
  - Scoping: Session ou Request (et + encore en JSF2.0.cf plus loin)
- Comment les déclarer?
  - **@ManagedBean** avant le nom de la classe **ou**
  - `<managed-bean>` dans le fichier faces-config.xml

# Managed Bean: Structure

- Les propriétés ainsi que leur getters/setters
  - Les setters sont automatiquement appelés par JSF quand le formulaire est soumis afin de peupler le Bean
  - Appelées avant l'appel des méthode d'Actions
  - Les getters sont automatiquement appelés par JSF quand le formulaire est affiché pour la 1<sup>ère</sup> fois afin de mettre les valeurs initiales des propriétés du bean dans les champs du formulaire (tant que différent de null ou "")
- Les méthodes d'Actions
  - Une ou plusieurs selon que le formulaire possède un ou plusieurs boutons
  - Comportement à exécuter en réponse à un click/soumission du formulaire
- Des variables « conteneurs » de résultat (en option)
  - Souvent des variables d'instances utilisées pour stocker le résultat de l'exécution de la méthode d'action
  - À utiliser par les pages résultats/redirections

# Managed Bean: Exemple

/\*annotation qui déclare le Bean à JSF. Cela évite de le faire dans le fichier faces-config.xml  
Possible de rajouter l'attribut "name" =>un nom d'alias. Autrement, il sera référencé par controlAccess (le **c en minuscule**)\*/\*

**@ManagedBean(name="control")**

//vous devez implementer Serializable si vous souhaitez scoper vos Bean en Session par exemple

public class ControlAccess implements Serializable {

private String login, password;

public String getLogin() { return login;}

public void setLogin(String login) { this.login = login;}

public String getPassword() {return password;}

public void setPassword(String password) {this.password = password;}

//Methode Action Controller qui sera référencée dans le formulaire de la page login.xhtml et appelée par JSF lors du submit

public String checkAccess(){

//Contrôle que les champs sont non-vides

if (isMissing(login) || isMissing(password)) {

//si les champs ne son pas renseignés, JSF forwardera l'execution vers la page missing-login-pass.xhtml

return("missing-login-pass");

} else if (login.equals(password))

{

//si le pass est correct, JSF forwardera l'execution vers la page welcome-page.xhtml

return("welcome-page");

} else {

//Sinon, vers la page bad-login.xhtml

return("bad-login");

}}

}

Propriétés +  
getters/setters

Action  
Controller

Les pages  
suivantes  
possibles selon le  
résultat du  
traitement

# Managed Bean: l'attribut « name »

**@ManagedBean(name="control")**

```
public class ControlAccess {..}
```

- L'annotation **@ManagedBean** vous évite de rajouter une ligne (une entrée) dans le fichier **faces-config.xml**
- L'attribut '**name**' vous permet de faire référence à votre Mbean à partir des facelets par un nom d'allias
  - Très pratique si vous devez changer le nom de la classe du Bean par exemple, l'alias lui restera toujours valide coté facelets
  - **Autrement**, c'est le nom de la classe avec la première lettre en **miniscule** i.e. **controlAccess.maMethode**



# Partie Formulaire : Page Facelet (login.xhtml)

```
<!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:f="http://java.sun.com/jsf/core">
```

```
<h:head><title>Login Page</title></h:head>
```

```
<h:body>
```

```
<div align="center">
```

```
<fieldset>
```

```
<legend style="font-size: large;font:bold;">Enter your Login & Password!</legend>
```

```
<h:form>
```

```
<br/>
```

```
<!-- si vous ne définissez pas un nom d'alias=>nom de classe en minuscule à la place:
#{controlAccess.login} -->
```

```
Login <h:inputText value="#{control.login}"/> <br/> <br/>
```

```
Password <h:inputSecret value="#{control.password}"/> <br/> <br/>
```

<!-- lors du click, JSF appellera la méthode checkAccess au niveau du bean controlAccess. La méthode doit rendre un String et ne prend rien en param. Un retour nul fera réafficher le formulaire -->

```
<h:commandButton value="Login" action="#{control.checkAccess}"/> <br/>
```

```
</h:form>
```

```
</fieldset>
```

```
</div>
```

```
</h:body>
```

```
</html>
```

Référence au managed bean par son nom d'alias

1<sup>ère</sup> réf sur getPassword (1<sup>er</sup> affichage du form) et setPassword (lors de la soumission du form)

2<sup>ème</sup> réf sur la méthode Action Controller checkAcces

# Page résultat en cas de login correct: welcome-page.xhtml

```
<!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">
```

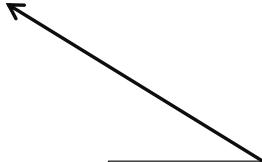
```
<h:head><title>Welcome</title></h:head>
<h:body>
```

```
<table class="title">
  <tr><th>Welcome Page!!</th></tr>
</table>
```

<!--On a toujours accès à l'instance du MBean car toujours dans le scope (Request) -->

```
<h2>Welcome Mr. <h:outputText value="#{control.login}"/></h2>
```

```
</h:body>
</html>
```



Référence au managed bean par son nom d'alias. Le bean est toujours dans le scope de la requête (Request)

# Facelet: exemples de composants

- Quelques exemples de composants coté formulaire et leur initialisation avec les propriétés du Managed Bean
  - `<h:inputText value="#{beanName.propertyName}"/>`
    - `getPropertyName()` est appelée par JSF quand le formulaire est affiché pour la 1<sup>ère</sup> fois afin d'afficher la valeur initiale de la propriété (tant que != de null ou "")
    - Utilisez les Type Wrapper (ex INTEGER ou lieu d'int) si vous voulez avoir les champs de texte vide (au lieu de « 0 » pour un int par exemple)
    - `setPropertyNames()` est appelée par JSF quand le formulaire est soumis afin de peupler le Bean
    - Possibilité de faire une chaîne `beanName.a.b.c.propertyName` ou `a`, `b` et `c` représentent des références d'objets liés par des associations

# Facelet: exemples de composants

- `<h:selectBooleanCheckbox value="#{beanName.propertyName}"/>`
  - Dans le cas d'un boolean, lors de du 1<sup>er</sup> affichage du form, `isPropertyName()` est appelée. Si elle retourne true, la checkBox est initialement cochée
- `<h:selectOneMenu value="#{beanName.propertyName}">`  
    `<f:selectItems value="#{beanName.choices}"/>`  
    `</h:selectOneMenu>`
  - lors de du 1<sup>er</sup> affichage du form:
    - Appel de `getChoices()` qui retourne une `List<SelectItem>` qui servira à remplir les entrées du menu
    - Appel de `getPropertyName()` afin de retourner l'entrée qui sera initialement positionnée dans le menu. Si aucun résultat ne correspond, choisir la première entrée de la liste

# Managed Bean: Portée (Scope)

- Il s'agit de déterminer la durée de vie des instances Mbean
- Dans JSF2.0, Six scopes possibles (contre 3 dans JSF1.x)
  - Request (**par défaut**), session, application, view, none, custom
  - À spécifier soit dans le fichier XML soit en utilisant l'annotation **@ScopeKindScoped** (ex. @SessionScoped)
  - À mettre après l'annotation @ManagedBean
- @RequestScoped
  - Une nouvelle instance du Mbean est créée pour chaque requête http

# Managed Bean: Portée (Scope)

- @SessionScoped

- L'instance est dans la session. La même instance est utilisée tant que l'utilisateur est reconnu (à travers un cookie ou réécriture d'URL) et avant le timeout
- Le bean doit être Serializable

- @ApplicationScoped

- L'instance est partagée par tous les utilisateurs
- Possibilité de créer l'instance au lancement de l'application par le serveur en utilisant **@ManagedBean(eager=true)**
- Attention: pensez à synchroniser l'accès

- @ViewScoped

- La même instance est utilisée tant que l'utilisateur est sur la même page
- Utile dans le cas d'utilisation des gestionnaires d'événements ou bien Ajax
- Le bean doit être Serializable

# Managed Bean: Portée (Scope)

- `@CustomScoped(value="#{someMap}")`
  - L'instance est stockée dans le Map
- `@NoneScoped`
  - L'instance n'est placée dans aucun scope
  - Utile pour des Mbean qui sont référencés par d'autres Mbean scopés

# Managed Bean: Portée (Scope)

## Exemple de ViewScoped

Delete	John
Delete	Charley
Delete	Priscila
Delete	Kate
Delete	Emily
Delete	Barack
Delete	Mia
Delete	Arthur



```
<h:form>
  <h:dataTable value="#{userList.users}" var="user">
    <h:column>
      <h:commandButton action="#{userList.delete}" value="Delete" >
        <f:setPropertyActionListener value="#{user}"
          target="#{userList.selectedUser}" />
      </h:commandButton>
    </h:column>
    <h:column>
      <h:outputText value="#{user}" />
    </h:column>
  </h:dataTable>
</h:form>
```

```
@ManagedBean(name="userList")
@ViewScoped
public class UserList {

    private ArrayList<String> users;
    private String selectedUser;

    @PostConstruct
    public void create () {
        users = new ArrayList<String> ();
        users.add("John");
        users.add("Charley");
        users.add("Priscila");
        users.add("Kate");
        users.add("Emily");
        users.add("Barack");
        users.add("Mia");
        users.add("Arthur");
    }

    public void delete () {
        users.remove(selectedUser);
    }

    public String getSelectedUser () {
        return selectedUser;
    }

    public void setSelectedUser(String selectedUser) {
        this.selectedUser = selectedUser;
    }

    public ArrayList<String> getUsers () {
        return users;
    }
}
```



# Les Objets Request et Response

- Pas d'accès direct à ces objets à partir des méthodes action controller
- Pas une vraie contrainte mais utile parfois pour
  - Jouer avec la session, les cookies, données machine client, etc.
- Possibilité d'y accéder mais un code verbeux

```
ExternalContext context =FacesContext.getCurrentInstance().getExternalContext();
HttpServletRequest request =(HttpServletRequest)context.getRequest();
HttpServletResponse response =(HttpServletResponse)context.getResponse();
```

# Injection de dépendances (DI)

- JSF offre la possibilité de faire de la DI
- Syntaxe:  
**@ManagedProperty(value="#{yourBean}")**  
private yourBeanType someField;
  - Vous devez avoir un setter pour la propriété (utiliser l'attribut *name* si le nom du setter est différent de celui de la propriété)
- Possibilité de le faire également dans le fichier faces-config.xml pour plus de flexibilité (à la Spring)

# La Navigation

# JSF2.0: La navigation (Option 1)

- **Objectif:** décrire la cinématique, l'enchaînement entre les pages selon le résultat d'exécution des méthodes Action Controller
- Mapping défini : par rapport aux valeurs de retour dans les méthodes Action Controller

```
if (isMissing(login) || isMissing(password)) {  
    //si les champs ne son pas renseignés, JSF forwadera  
    l'execution vers la page missing-login-pass.xhtml  
    return("missing-login-pass");  
}
```

- **Avantage:** facile pour débiter
  - (inconvenient cf l'avantage de la deuxième option)

# JSF2.0: La navigation (Option 2)

- Exemple de déclaration d'une règle de navigation dans le fichier xml **faces-config.xml**

```
<faces-config>
<navigation-rule>
  <from-view-id>/some-start-page.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>return-condition-1</from-outcome>
    <to-view-id>/result-page-1.xhtml</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>return-condition-2</from-outcome>
    <to-view-id>/result-page-2.xhtml</to-view-id>
  </navigation-case>

  <!--More navigation-case entries for other conditions-->
</navigation-rule>
.....
</faces-config>
```

- Avantage:** toute la cinématique/mappings sont regroupés dans un seul fichier. Permet d'avoir une vision globale

# JSF2.0: La navigation

- Possibilité d'utiliser " \* " à la place de `some-start-page.xhtml` dans `<from-view-id>/some-start-page.xhtml</from-view-id>` afin de matcher plusieurs pages de départ
  - Permet de regrouper le même traitement pour un ensemble de pages ayant la même valeur de retour
  - Évite de multiplier les `<navigation-case>`
  - Pratique pour spécifier les pages d'erreur pour un site par exp.
- Si `<from-outcome>return-condition-1</from-outcome>` est omis alors n'importe quelle condition de retour match
  - Sauf le null qui lui réaffichera le même formulaire

# JSF2.0: La navigation

- Possibilité de conditionner la navigation par une expression EL

## Exemple

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

# JSF2.0: La navigation

- Possibilité de calculer la page de destination directement dans le fichier faces-config.xml sans passer par les valeurs de retour des méthodes d'Action

```
<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>
```



# JSF2.0: La navigation

- Les exemples donnés jusqu'à présent sont liés à la navigation dynamique i.e. calculer par des valeurs de retour, des expressions EL, etc.
- Possibilité d'utiliser de la navigation statique sans forcément lancer un traitement. Dans ce cas:

`<h:commandButton ... action="fixed-page"/>`

`<h:link value="Back to Login page" outcome="login"/>`

# EL: Expression Language

# EL: Expression Language

Langage utilisé dans les Facelets et qui fait référence à des propriétés et méthodes de Mbeans, ou bien à des variables scoppées

## Exemples d'expressions

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

- premier affichage du form: appel de `getFirstName()` pour initialiser la valeur par défaut du champ
- Appel de `setFirstName()` lors de la soumission du formulaire

`<h:outputText value="#{beanName.propertyName}" />`

- Appel de `getPropertyNames()` pour l'afficher en sortie cote facelet

`#{employee.addresses[0].zip}`

- Rend la valeur de la propriété zip de la premier adresse de la collection addresses d'employee

`#{beanName.stateCapitals["maryland"]}`

- Pour les Map, rend la valeur de la clé « maryland » de la map stateCapitals

`#{company.president.firstName}`

- Rend le firstName de l'objet president lié au mbean company

`#{test ? option1 : option2}`

- Permet d'exprimer une condition avec des valeurs en sortie. Ne peut pas outputer de l'HTML, que tu texte

# EL: Objets Implicites

- **facesContext**. L'objet FacesContext
  - E.g. `#{facesContext.externalContext.remoteUser}`
- **param**. Les paramètres de Request
  - E.g. `#{param.custID}`
- **header**. L'entête Request
  - E.g. `#{header.Accept}` or `#{header["Accept"]}`
  - `#{header["Accept-Encoding"]}`

# EL: Objets Implicites

- **cookie**. L'objet Cookie(et non pas la valeur).
  - E.g. `#{cookie.userCookie.value}` ou `#{cookie["userCookie"].value}`
- **request, session**
  - `#{request.queryString}`, `#{session.id}`

## **ATTENTION:**

- L'utilisation des objets implicites est déconseillée afin de respecter au maximum le modèle MVC
- La manipulation de ces objets doit se faire coté Java et non pas coté Facelet

# L'utilisation de l'attribut rendered

- Affichage des composants conditionné par un boolean

```
<h:outputText value="#{someValue}" rendered="#{someCondition}"/>
```

# Les Messages et l'i18n

# Messages avec properties :étapes

1) Créer un fichier dans votre répertoire src de projet **.properties** contenant des paires Clé=Valeur

2) Déclarer le resource-bundle dans **faces-config.xml**

- **<base-name>** donne la base du nom de fichier
- **'var'** donne le nom de la variable scopée de type Map qui contiendra le contenu du fichier de propriétés (Clé=>Valeur)

Exp.

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

3) Afficher les messages en utilisant des expression EL

- Ex. **#{msgs.keyName}** msgs ici est le nom de la Map et KeyName, une clé



# Messages Paramétrés

## 1) Créer un fichier dans votre répertoire src de projet **.properties**

- Les valeurs cette fois-ci contiennent des jokers {0}, {1}, {2}, etc.
- Exp., someName= texte valeur {0} texte suite {1}

## 2) Déclarer le resource-bundle dans **faces-config.xml**

## 3) Afficher les messages en utilisant h:outputFormat

- Exp:

```
<h:outputFormat value="#{msgs.someName}">  
    <f:param value="Literal value for 0th entry"/>  
    <f:param value="#{someBean.calculatedValForEnty1}"/>  
</h:outputFormat>
```

# Exemple: le fichier .properties

```
registrationTitle=Registration  
firstName=First Name  
lastName=Last Name  
emailAddress=Email Address  
registrationText=Please Enter Your {0}, {1}, and {2}.  
prompt=Enter {0}  
buttonLabel=Register Me  
successTitle=Success  
successText=You Registered Successfully.
```

# Exemple: le fichier faces-config.xml

```
<?xml version="1.0"?>
<faces-config ...>
  <application>
    <resource-bundle>
      <base-name>messages1</base-name>
      <var>msgs1</var>
    </resource-bundle>
    <resource-bundle>
      <base-name>messages2</base-name>
      <var>msgs2</var>
    </resource-bundle>
  </application>
</faces-config>
```

# Exemple: La Facelet

```
<!DOCTYPE ...>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:f="http://java.sun.com/jsf/core"
xmlns:h="http://java.sun.com/jsf/html">
<h:head><title>#{msgs2.registrationTitle}</title>
...
<h3>
    <h:outputFormat value="#{msgs2.registrationText}">
        <f:param value="#{msgs2.firstName}"/>
        <f:param value="#{msgs2.lastName}"/>
        Replaces {0} in registrationText
        Replaces {1} in registrationText
        <f:param value="#{msgs2.emailAddress}"/>
    </h:outputFormat>
</h3>
```

# L'Internationalisation (i18n)

- **Objectif:** créer des messages en plusieurs langues pour votre application
- Faudra créer plusieurs fichiers de messages, un par langue supportée
- En cas d'une nouvelle langue, le code ne change pas

# L'Internationalisation (i18n) : étapes

## 1. Créer plusieurs fichiers .properties

- messages.properties, messages\_es.properties, messages\_es\_mx.properties, etc.

## 2. Utiliser f:view et l'attribut locale

- `<f:view locale="#{facesContext.externalContext.requestLocale}">`
- Détermine la langue à partir des paramètres de votre browser

## 3. Déclarer le resource-bundle dans **faces-config.xml**

- Le fichier de messages approprié sera choisi selon les infos de l'objet Locale

## 4. Afficher les messages en utilisant h:outputFormat

# L'Internationalisation (i18n): Exemple

- **messages.properties**
  - company=JsfResort.com
  - feature=Our {0}:
  - pool=swimming pool
- **messages\_es.properties**
  - company=JsfResort.com
  - feature=Nuestra {0}:
  - pool=piscina
- **messages\_es\_mx.properties**
  - company=JsfResort.com
  - feature=Nuestra {0}:
  - pool=alberca

# L'Internationalisation (i18n): Exemple

```
<?xml version="1.0"?>  
<faces-config ...>  
  <application>  
    <resource-bundle>  
      <base-name>messages</base-name>  
      <var>msgs</var>  
    </resource-bundle>  
  </application>  
</faces-config>
```



# L'Internationalisation (i18n): Exemple

```
<!DOCTYPE ...>
<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:view
locale="#{facesContext.externalContext.requestLocale}">
...
<h1>#{msgs.company}</h1>
<h2>
<h:outputFormat value="#{msgs.feature}">
    <f:param value="#{msgs.pool}"/>
</h:outputFormat>
</h2>

...
</f:view></html>
```

# La validation des formulaires

# Validation : Option

- **Validation Manuelle**
  - Validation dans la méthode Action Controller
- **Validation automatique implicite**
  - Conversion de type et utilisation de l'attribut “required”
- **Validation automatique explicite**
  - Utilisation de `f:validateLength`, `f:validateRegex`, etc.
- **Définir vos propres méthodes de validation**
  - Utilisation de l'attribut “validator”

# Validation Manuelle

- Vérifier les valeurs au niveau des setters des Mbean ou
- Vérifier au niveau des Méthodes Action Controller
  - Si une donnée du formulaire est mal renseignée écrire un message d'erreur dans le **FacesContext** . Exemple de code à l'intérieur d'une méthode d'Action:  
....  
**FacesContext** context = **FacesContext.getCurrentInstance()**;  
if (getUserID().equals("")) {  
    **context.addMessage**(null, **new FacesMessage**("UserID required"));  
} .....
  - Le « null » ici veut dire que le message s'adresse à tout le formulaire et non pas à un composant donné (sera affiché en bas ou en haut du formulaire par exemple)
  - Si le message est destiné à un composant précis, utilisez:  
**context.addMessage("someId", someFacesMessage);**
- Utiliser h:messages pour afficher les messages d'erreurs
  - Utilisez **<h:messages styleClass="error"/>** pour des messages destinés à tout le formulaire
  - Utilisez user **<h:message for="userId"/>** quand c'est destiné à un composant donné

# Validation Manuelle: Exemple

- La méthode Action Controller d'un Mbean nommé **BidBean1**

```
public String doBid() {  
  
    FacesContext context = FacesContext.getCurrentInstance();  
    if (getUserID().equals("")) {  
        context.addMessage(null, new FacesMessage("UserID required"));  
    }  
    if (getKeyword().equals("")) {  
        context.addMessage(null, new FacesMessage("Keyword required"));  
    }  
    if (getNumericBidAmount() <= 0.10) {  
        context.addMessage(null, new FacesMessage("Bid amount must be at least $0.10."));  
    }  
  
    } //vérifier si la liste des messages et non vide alors réafficher le from avec les messages  
    d'erreurs  
    if (context.getMessageList().size() > 0) {  
        return(null);  
    } else {  
        doBusinessLogicForValidData();  
        return("show-bid1");  
    }  
}
```

# Validation Manuelle: Exemple

```
<h:form>
  <h:messages/>
  <table>
    <tr>
      <td>User ID:
      <h:inputText value="#{bidBean1.userID}"/></td></tr>
    <tr>
      <td>Keyword:
      <h:inputText value="#{bidBean1.keyword}"/></td></tr>
    <tr>
      <td>Bid Amount:
      <h:inputText value="#{bidBean1.bidAmount}"/></td></tr>
    <tr>
      <td>Duration:
      <h:inputText value="#{bidBean1.bidDuration}"/></td></tr>
    <tr><th>
      <h:commandButton value="Send Bid!"
      action="#{bidBean1.doBid}"/></th></tr>
  </table>
</h:form>
```

Enter your Login and Password!

Login required  
Password required

Login

Password

Login

# Validation par Conversion de Type

- Validation automatique implicite
  - Définissez les propriétés de vos beans avec des types simples
    - Rappel: Utilisez les Type Wrapper (ex INTEGER ou lieu d'int) si vous voulez avoir les champs de texte vide (au lieu de « 0 » pour un int par exemple) lors de l'affichage initial du formulaire
  - JSF tentera une conversion automatiquement
    - Formulaire réaffiché en cas d'erreurs
    - Les messages d'erreurs sont générés automatiquement
    - Utilisez l'attribut **converterMessage** pour les customiser
  - Champs obligatoires
    - Possibilité de rajouter l'attribut « required » pour tout champs input
    - Utilisez l'attribut **requiredMessage** pour les customiser le message d'erreur
  - Utilisez **h:message** pour afficher les messages d'erreurs au niveau du champs/composant
- Le Mbean est déchargé de cette validation
  - Pourra se concentrer sur la validation des données par rapport au métier (voir plus loin)

# Validation par Conversion de Type: Exemple

```
<tr>
  <td>Bid Amount:
  <h:inputText value="#{bidBean2.bidAmount}"
    required="true"
    requiredMessage="You must enter an amount"
    converterMessage="Amount must be a number"
    id="amount"/></td>
  <td><h:message for="amount" styleClass="error"/></td>
</tr>
<tr>
  <td>Duration:
  <h:inputText value="#{bidBean2.bidDuration}"
    required="true"
    requiredMessage="You must enter a duration"
    converterMessage="Duration must be a whole number"
    id="duration"/></td>
  <td><h:message for="duration" styleClass="error"/></td>
</tr>
```



# Validation en utilisant les Tags Validate

- Définissez les propriétés de vos beans avec des types simples
- Utilisez les tags **f:validateXXX**
  - <f:validateLength .../>
  - <f:validateLongRange .../>
  - <f:validateDoubleRange .../>
  - <f:validateRegex .../>
- Si la validation des champs ne passe pas
  - Le formulaire est réaffiché avec les messages d'erreur
  - Utiliser l'attribut **validatorMessage** pour customiser le message d'erreur

## Validation en utilisant les Tags Validate: Exemple

```
<tr>
  <td>Bid Amount:
  <h:inputText value="#{bidBean2.bidAmount}"
  required="true"
  requiredMessage="You must enter an amount"
  converterMessage="Amount must be a number"
  validatorMessage="Amount must be 0.10 or greater"
  id="amount">
  <f:validateDoubleRange minimum="0.10"/>
  </h:inputText></td>
  <td><h:message for="amount" styleClass="error"/></td>
</tr>
```

# Ordre des tests de validation

- **Important:** il y'a un ordre de précedence pour la validation
- Required > Type (conversion) > Validators

# Validation en utilisant le tag Validator

- Une fois les tests de types, de format et de présence « required » réalisés, certaines données ont besoin d'être validées avec le métier coté Mbean
  - Exemple: besoin d'un accès base ou un appel à une opération métier avant de pouvoir valider la donnée
- Pour cela on utilise soit la méthode Action Controller (vue avant dans la validation manuelle)
- Soit on utilise le tag validator

# Validation en utilisant le tag Validator

- Côté Facelets

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

- Côté Java

- Une méthode (someMethod) dans le Mbean(someBean) qui throws **ValidatorException** avec un FacesMessage si jamais la validation échoue
- La méthode prend en arguments
  - FacesContext (le contexte comme pour les servlets)
  - UIComponent (l'élément/champ dans la facelet à valider)
  - Object (la valeur soumise du champ à valider)

# Validation en utilisant le tag Validator: Exemple

- Code de la Facelet

```
<tr>
  <td>Bid Amount:
  $<h:inputText value="#{bidBean2.bidAmount}"
    required="true"
    requiredMessage="You must enter an amount"
    converterMessage="Amount must be a number"
    validator="#{bidBean2.validateBidAmount}"
    id="amount"/>
  </td>
  <td><h:message for="amount" styleClass="error"/></td>
</tr>
```

# Validation en utilisant le tag Validator: Exemple

- Coté Java Mbean

.....

.....

```
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);  
    }  
}
```

.....

# Résumé sur les options de validation

- Validation manuelle / par l'action controller
  - Validation liée au métier
  - Besoin de comparer les différentes valeurs des/entre champs
  - La même méthode d'action pour valider tous les champs
- Validation par Conversion de type et Required
  - Forcer le typage
  - S'assurer d'avoir des champs non vides
- Validation par f:validate
  - Plus pour le format, des valeurs entre un intervalle min/max, etc.
- Créer vos propres méthodes de validation et utiliser l'attribut validator du champ
  - Une méthode de validation par champs, exécutée coté Mbean
  - Pour les tests de validation complexes liés au métier



# Manipuler les collections à longueur variable

- Options possibles
  - Dans le code du Mbean utiliser les iterator Java pour itérer sur une collection et générer l'HTML qui va avec
  - Utiliser `h:dataTable`
  - Créer votre propre composant
  - Utiliser `ui:repeat`

# h:dataTable

Le nom du bean et le nom de la collection sur laquelle itérer

```
<h:dataTable value="#{listContact.listContacts}" var="contact"
border="1" cellspacing="4" width="60%">
  <h:column>
    <f:facet name="header">
      <h:outputText value="Id" />
    </f:facet>
    <h:outputText value="#{contact.id}" />
  </h:column>
  <h:column>
    <f:facet name="header">
      <h:outputText value="First Name" />
    </f:facet>
    <h:outputText value="#{contact.firstName}" />
  </h:column>
  <h:column>
    <f:facet name="header">
      <h:outputText value="Last Name" />
    </f:facet>
    <h:outputText value="#{contact.lastName}" />
  </h:column>
  <h:column>
    <f:facet name="header">
      <h:outputText value="Email" />
    </f:facet>
    <h:outputText value="#{contact.email}" />
  </h:column>
</h:dataTable>
```

La variable qui contient l'instance à chaque itération

Les entêtes

Les données en sortie

# Les Templates

- Servent à éviter les répétitions dans les facelets

## Exemple

/templates/template-1.xhtml

...

<h:body>Content shared by all client files


    <h2><ui:insert name="title">Default Title</ui:insert></h2>

        More content shared by all clients

    <ui:insert name="body">Default Body</ui:insert>

</h:body> ...

Sert à définir des valeurs joker à renseigner par la page client



client-file-1.xhtml

<ui:composition template="/templates/template-1.xhtml">

    <ui:define name="title">Title text</ui:define>

    <ui:define name="body">

        Content to go in "body" section of template

    </ui:define>

</ui:composition>

Renseigner les valeurs



# Conclusion

- Framework puissant offrant de nombreuses fonctionnalités /composants
- S'intègre facilement avec l'existant (EJB, Spring, hibernate, etc.)
- De plus en plus d'outils mais pas encore très au point
- Maturité: Jeunesse par rapport à Struts et les traditionnelles JSP/Servlet/Taglibs

# Lectures

- JSF 2.0 Cookbook, de Anghel Leonard, Packt Publishing Limited (30 mai 2010), **ISBN-10: 1847199526**
- Beginning JSF 2 APIs and JBoss Seam de K.Tong, APress; **Édition : 1** (1 mai 2009), **ISBN-10: 143021922X**
- Java Server Faces (JSF) avec Eclipse - Mise en œuvre pour la conception d'applications web, Editions ENI (6 avril 2009), François-Xavier Sennesal, **ISBN-10: 2746048124**