

JSF (Java Server Faces)

EMSI - Université Côte d'Azur
Richard Grin
Version O 2.37 – 30/10/19

Plan de ce cours

- Présentation de JSF
- Architecture des applications JSF
- Exemple de page JSF
- Backing bean et portée CDI
- Cycle de vie
- Configuration
- Navigation
- PRG
- Messages d'erreur ou information
- Événements
- Ajax
- JSF et HTML5
- Ressources
- Flots
- Bibliographie

R. Grin

JSF

page 2

Présentation de JSF

R. Grin

JSF

page 3

JSF

- Framework Java pour le développement d'applications Web
- Utilise Facelets, langage de déclaration de page (format XHTML) pour écrire des pages JSF
- Une page JSF décrit une vue JSF (arbre de composants Java)
- Les pages et vue JSF sont enregistrées sur le serveur

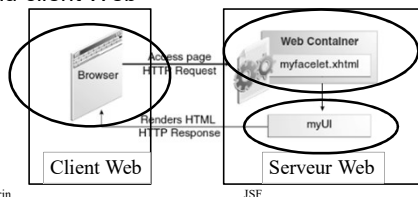
R. Grin

JSF

page 4

Page JSF : XHTML, Java, HTML

- Fichier XHTML qui contient des balises (<h:inputText>, ...)
- Ces balises décrivent des composants Java
- La page JSF est traduite en une page HTML envoyée au client Web



R. Grin

JSF

page 5

EL et backing bean

- EL (*Expression Language*) : langage utilisé pour représenter des données dynamiques ou des traitements à exécuter dans les pages JSF
- Les expressions EL sont encadrées par #{ }
- Une expression EL utilise le plus souvent un *backing bean*, classe Java liée à la page JSF :
 - une propriété du backing bean peut représenter une donnée dynamique
 - une méthode du backing bean peut représenter un traitement à exécuter

R. Grin

JSF

page 6

Exemple - page JSF `index.xhtml`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
  <h:head> <title>Présentation</title> </h:head>
  <h:body>
    <h3>Présentation</h3>
    <h:form>
      Nom : <h:inputText value="#{utilisateur.nom}" />
      <h:commandButton value="Enregistrer"
        action="#{utilisateur.direHello}" />
    </h:form>
  </h:body>
</html>
```

Espace de noms par défaut

Espace de noms JSF

Vous devinez comment cette page sera utilisée ?

R. Grin

JSF

page 7

Exemple - page JSF `index.xhtml`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
  <h:head> <title>Présentation</title> </h:head>
  <h:body>
    <h3>Présentation</h3>
    <h:form>
      Nom : <h:inputText value="#{utilisateur.nom}" />
      <h:commandButton value="Enregistrer"
        action="#{utilisateur.direHello}" />
    </h:form>
  </h:body>
</html>
```

Formulaire

utilisateur : nom backing bean

Propriété nom du backing bean

Bouton soumet le formulaire par une requête POST envoyée au serveur

Méthode `direHello` du backing bean

R. Grin

JSF

page 8

Exemple - Backing bean

```
@Named
@RequestScoped
public class Utilisateur {
  private String nom;
  public String getNom() { return nom; }
  public void setNom(String nom) {
    this.nom = nom;
  }
  public String direHello() {
    return "hello";
  }
  ...
}
```

Nom par défaut : « utilisateur »

@Named permet au backing bean d'être utilisé dans une expression EL

#{utilisateur.xxx} provoque la création par le container CDI d'une instance du bean, s'il n'y en a pas déjà une

R. Grin

JSF

page 9

Exemple - Backing bean

```
@Named
@RequestScoped
public class Utilisateur {
  private String nom;
  public String getNom() { return nom; }
  public void setNom(String nom) {
    this.nom = nom;
  }
  public String direHello() {
    return "hello";
  }
  ...
}
```

Portée du bean ; il sera supprimé à la fin de la requête

Propriété nom

Nom prochaine page à afficher

R. Grin

JSF

page 10

Exemple - page JSF `hello.xhtml`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
  <h:head> <title>Hello</title> </h:head>
  <h:body>
    <h3>Hello</h3>
    Hello #{utilisateur.nom}
  </h:body>
</html>
```

Que fait cette page ?

Lien avec la propriété nom du backing bean

Est-ce que nom contiendra le nom tapé par l'utilisateur ?

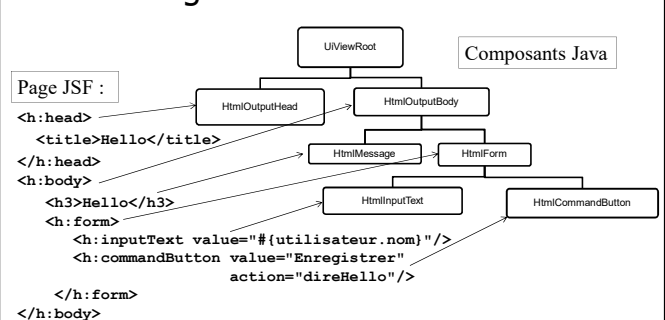
Quand la page HTML est créée sur le serveur, pour être retournée à l'utilisateur, la requête POST est toujours en cours de traitement. C'est donc la même instance du backing bean que celle utilisée par `index.xhtml`. Donc `nom` contient le nom saisi par l'utilisateur.

R. Grin

JSF

page 12

Page JSF - « Vue » Java



R. Grin

JSF

page 12

Principaux services rendus par JSF

- ❑ Conversion entre les *textes* saisis par l'utilisateur et les *valeurs Java* du serveur
- ❑ Validation des données saisies par l'utilisateur
- ❑ Automatisation de l'affichage des messages d'erreur si problèmes de conversion ou validation
- ❑ Internationalisation
- ❑ Support d'Ajx sans programmation
- ❑ Templates graphiques
- ❑ Librairies de composants

R. Grin

JSF

page 13

Les traitements

- ❑ Séparation très nette entre
 - les pages JSF
 - les traitements à exécuter
- ❑ Même les traitements directement liés à l'interface utilisateur sont effectués *en dehors de la page JSF*, par du code Java (*backing bean*)

R. Grin

JSF

page 14

Architecture des applications JSF

R. Grin

JSF

page 15

Répartition des tâches

- ❑ Pages JSF pour l'interface avec l'utilisateur ; ne contiennent aucun traitement
- ❑ Backing beans pour les traitements liés directement à l'interface utilisateur
- ❑ Backing beans font appel à des EJB ou d'autres classes Java pour les autres traitements (traitements métier, accès aux sources de données,...)
- ❑ Accès aux bases de données utilisent souvent JPA et donc des classes entités

R. Grin

JSF

page 16

Backing bean

- ❑ Souvent, mais pas obligatoirement, un backing bean par page JSF
- ❑ Géré par le container CDI

Qu'est-ce que ça signifie ?

Le container crée le backing bean et le supprime

A quel moment le backing bean est créé par le container ?

La 1^{ère} fois qu'il y a une référence au backing bean

A quel moment le container supprime le bean ?

A la fin de sa portée

R. Grin

JSF

page 17

Code d'un backing bean

- ❑ Propriétés pour valeurs à afficher ou saisies par l'utilisateur :

```
<h:inputText value="#{utilisateur.nom}"/>
```

setter et getter utilisés en interne

- ❑ Traitements lancés par l'utilisateur, par un clic bouton par exemple :

```
<h:commandButton value="Enregistrer"
  action="#{utilisateur.direHello()}" />
```

- ❑ Expression qui indique si un composant doit être affiché :

```
<h:inputText
  value="#{bean.client.pourcentageRemise}"
  rendered="#{bean.client.bonClient}" />
```

R. Grin

JSF

page 18

Servlet « Faces »

- Le servlet JSF gère le cycle de vie JSF, fondamental pour JSF
- Ce servlet est fourni par JSF et caché au développeur
- Un mapping indique les requêtes qui seront traitées par ce servlet ; par défaut celles avec les URLs /faces/*, *.jsf, *.faces, *.xhtml ; correspond toujours à un fichier *.xhtml (par exemple, page.jsf correspond à page.xhtml) ; mapping modifiable dans web.xml

R. Grin

JSF

page 19

Template

- Les pages d'un site Web respectent le plus souvent une charte graphique
- L'aspect commun aux pages est défini dans un fichier XHTML à part, appelé *template*
- Un template est une page JSF avec des parties « variables » :

```
<ui:insert name="header"/>
```

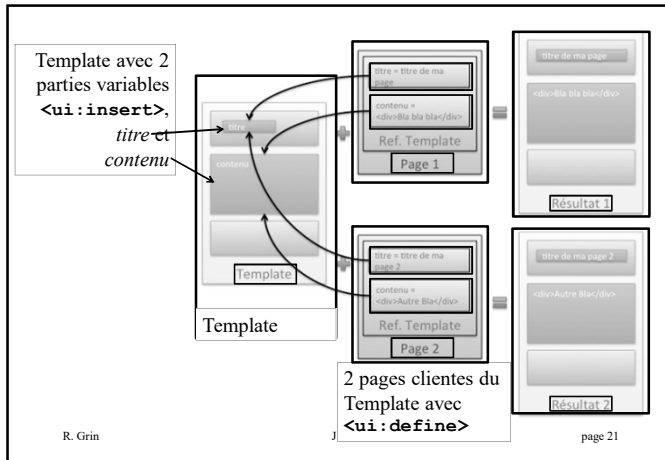
- Une page JSF qui utilise ce template (page cliente du template) a l'aspect défini par le template ; elle définit les parties variables :

```
<ui:define name="header">...</ui:define>
```

Contenu de *header*

R. Grin

page 20



R. Grin

JSF

page 21

Conversion et validation

- Les données saisies dans les formulaires HTML arrivent sur le serveur sous la forme de chaînes de caractères (valeurs des requêtes HTTP)
- Il faut les convertir dans le bon type Java (celui de la propriété du backing bean) :

```
<h:inputText id="age" value="#{bean.age}" >
```

- Elles doivent être validées (18 < age < 70, par exemple) **avant** d'exécuter les traitements qui les utilisent

R. Grin

JSF

page 22

Convertisseurs et validateurs

- La conversion peut être implicite ; par exemple « 12 » converti en le nombre entier 12
- La conversion ou la validation peut être effectuée par un convertisseur ou validateur standard fourni par JSF
- Le développeur peut aussi écrire son propre code pour convertir ou valider

R. Grin

JSF

page 23

Exemple de convertisseur standard

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

Si erreur de conversion, le message est affiché ici

Utilisable aussi avec `<h:outputText>`

Selon le type Java de *dateNaissance*

R. Grin

JSF

page 24

Exemple de validateur standard

```
<h:inputText id="email" value="#{bean.email}"
  required="true" label="Email"
  validatorMessage="Email n'est pas valide">
  <f:validateRegex
    pattern="^[_A-Za-z0-9-\\+](\\.[_A-Za-z0-9-\\+])*@[A-Za-
z0-9-](\\.[A-Za-z0-9-\\+]*(\\.[A-Za-z]{2,})$" />
  </h:inputText>
<h:message for="email" />
```

R. Grin

JSF

page 25

Convertisseur non standard

- Il faut écrire une classe annotée avec `@FacesConverter` ; on passe en paramètre l'id du convertisseur qui servira à l'identifier dans une page JSF
- La classe doit implémenter l'interface `javax.faces.convert.Converter<T>` qui contient les 2 méthodes
 - `T getAsObject(FacesContext ctx, UIComponent composant, String valeur)`
 - `String getAsString(FacesContext ctx, UIComponent composant, T valeur)`

R. Grin

JSF

page 26

Validateur non standard

- Il faut écrire une classe annotée avec `@FacesValidator` ; on passe en paramètre l'id du validateur qui servira à l'identifier dans une page JSF
- La classe doit implémenter l'interface `javax.faces.convert.Validator<T>` qui contient la méthode
 - `void validate(FacesContext context, UIComponent composant, Object valeur)`
La méthode lance une `ValidatorException` si la valeur n'est pas validée

R. Grin

JSF

page 27

Convertisseur avec injection CDI

- Avant JSF 2.3 il n'était pas possible d'injecter des beans CDI ou des EJB dans les classes des convertisseurs ou des validateurs
- Maintenant possible avec l'ajout de l'attribut `managed` de `@FacesConverter`, si on lui donne la valeur `true` (idem pour `@FacesValidator` des validateurs)

R. Grin

JSF

page 28

Exemple convertisseur (classe)

```
@ApplicationScoped // Pour en faire un bean CDI
@FacesConverter(value = "converterDiscount", managed = true)
public class ConverterDiscount implements Converter<Discount> {
    @EJB
    private DiscountManager discountManager;
    @Override
    public DiscountCode getAsObject(FacesContext context,
        UIComponent component, String value) {
        if (value == null) {
            return null;
        }
        return discountManager.findById(value);
    }
}
// Ne marche que si on a ajouté un bean CDI annoté par
// @FacesConfig et @ApplicationScoped ou si le convertisseur est
// annoté avec @FacesConfig ; voir section « Configuration »
```

R. Grin

Désigner convertisseur et validateur

- Plusieurs façons d'indiquer dans une page JSF qu'un des composants utilise un convertisseur ou un validateur ; par exemple pour `<h:selectOneMenu>`
- Par une sous-balise spéciale pour convertisseurs ou validateurs standards JSF ; par exemple `<f:convertDateTime>` ou `<f:validateLength>`
- Par une balise `<f:convert>` ou `<f:validate>` ; on donne l'id du convertisseur ou validateur
- Par un attribut `converter` ou `validator` qui désigne une propriété ou une méthode qui retourne un convertisseur ou un validateur

R. Grin

JSF

page 30

Exemple convertisseur

```
<h:selectOneMenu id="discount"
value="#{customerBean.discount}"
title="Discount" required="true"
requiredMessage="Le taux de réduction est requis.">
<f:converter converterId="converterDiscount"/>
<f:selectItems value="#{customerBean.discounts}"
var="code"
itemLabel="#{code.id += ' (' += code.rate += '%)'}"/>
</h:selectOneMenu>
```

« += » est la concaténation
dans le langage EL

R. Grin

JSF 2.3

page 31

Exemple du TP 1

```
@EJB private DiscountFacadeEjb ejb;
...
public Converter<Discount> getDiscountConverter() {
return new Converter<Discount>() {
@Override public Discount getAsObject(
FacesContext ctx, UIComponent cmp, String val) {
if (val == null) return null;
return ejb.findById(val);
}
@Override
public String getAsString(
FacesContext ctx, UIComponent cmp, Discount d) {
if (d == null) return "";
return d.getDiscountCode();
};
}; }
```

Qui peut expliquer ?

Utilisé dans TP 1.
Pour faire quoi ?

R. Grin

JSF

page 32

Convertisseur par défaut pour un type

- Au lieu de désigner à plusieurs endroits un convertisseur pour un même type, on peut indiquer qu'un certain type Java devra toujours utiliser un certain convertisseur
- Pour cela, il suffit d'indiquer le type dans l'annotation `@FacesConverter` de la classe convertisseur :
`@FacesConverter(forClass=Compte.class)`

R. Grin

JSF

page 33

Validation des beans

- La validation peut aussi se faire en annotant le champ (d'un backing bean ou d'une classe entité JPA) qui va recevoir la valeur

```
@Min(18) @Max(70)
private int age;
```

R. Grin

JSF

page 34

Composants des pages JSF

- Nombreux composants standards qui correspondent aux balises HTML :
 - saisie des données `<h:inputText>`
 - listes déroulantes `<h:selectOneMenu>`
 - boutons `<h:button>` ou `<h:commandButton>`
 - tables `<h:dataTable>`
 - ...
- Pour améliorer ces composants ou pour en avoir de nouveaux le développeur peut
 - utiliser des bibliothèques de composants
 - créer ses propres composants

R. Grin

JSF

page 35

Exemple plus complet de page JSF

R. Grin

JSF

page 36

En-tête

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
<h:head>
  <title>Inscription</title>
  <h:outputStylesheet name="css/styles.css"/>
</h:head>
```

Fichier CSS ; chemin par rapport au répertoire /resources

R. Grin

JSF

page 37

Panel et liste déroulante

```
<h:body>
<h3>Inscription</h3>
<h:form>
  <h:panelGrid columns="3"
    columnClasses="rightalign, leftalign, leftalign">
    C1 <h:outputLabel value="Titre : " for="titre"/>
    C2 <h:selectOneMenu id="titre" label="Titre"
      value="#{inscriptionBean.titre}" >
      <f:selectItem itemLabel=" " itemValue="#{null}"/>
      <f:selectItem itemLabel="M." itemValue="M"/>
      <f:selectItem itemLabel="Mme" itemValue="Mme"/>
    </h:selectOneMenu>
    C3 <h:message for="titre"/>
  </h:panelGrid>
  Inscription
  Titre : 
```

Présentation avec 3 colonnes : intitulé, zone de saisie, message d'erreur éventuel

Classes définies dans fichier CSS

Pour choix d'un item

R. Grin

JSF

page 38

Liste déroulante

```
<h:body>
<h3>Inscription</h3>
<h:form>
  <h:panelGrid columns="3"
    columnClasses="rightalign, leftalign, leftalign">
    <h:outputLabel value="Titre : " for="titre"/>
    <h:selectOneMenu id="titre" label="Titre"
      value="#{inscriptionBean.titre}" >
      <f:selectItem itemLabel=" " itemValue="#{null}"/>
      <f:selectItem itemLabel="M." itemValue="M"/>
      <f:selectItem itemLabel="Mme" itemValue="Mme"/>
    </h:selectOneMenu>
    Inscription
  <h:message for="titre"/>
  Titre : 
```

Le choix de l'utilisateur sera rangé dans cette propriété du backing bean

Les items

Inscription

Inscription

R. Grin

JSF

9

Nom et email (avec validateur)

```
<h:outputLabel value="Nom : " for="nom"/>
Saisie du nom <h:inputText id="nom" label="Nom"
  required="true"
  value="#{inscriptionBean.nom}" />
  <h:message for="nom" />
  Message si nom pas saisi
  <h:outputLabel value="Email : " for="email"/>
  Saisie email <h:inputText id="email" label="Email"
    required="true"
    value="#{inscriptionBean.email}"
    <f:validator validatorId="validateurEmail"/>
  </h:inputText>
  <h:message for="email" />
  Valideur spécial pour email, écrit dans l'application
```

R. Grin

JSF

page 40

Bouton de commande

```
<h:panelGroup>
  <h:commandButton id="enregistrement"
    value="Enregistrer"
    action="#{inscriptionBean.confirmer()}" />
</h:panelGroup>
</h:form>
</h:body>
</html>
```

Bouton pour soumettre formulaire

Méthode du bean lancée au clic du bouton

Valeur retour indiquera la prochaine page à afficher

R. Grin

JSF

page 41

CSS et JSF – fichier des styles

- Utiliser `<h:outputStylesheet>` à la place de `<link rel="stylesheet">` :
`<h:outputStylesheet name="css/style.css" />`
- JSF va chercher le fichier dans le répertoire **resources** placé sous la racine des pages Web ; dans **resources/css/style.css** pour l'exemple

R. Grin

JSF

page 42

CSS et JSF – style d'un composant

- Pour donner du style à un composant JSF, utiliser l'attribut `styleClass` (ou `style` pour du CSS intégré) ou un des attributs particuliers à un composant (comme `columnClasses` de `<h:panelGrid>`)
- Attention si vous utilisez `#id` dans le fichier CSS : l'id d'un composant JSF ne correspond pas à l'id du composant HTML envoyé au client

R. Grin

JSF

page 43

Commentaires

- Par défaut les commentaires HTML (`<!-- -->`) ne fonctionnent pas bien pour JSF, mais on peut configurer `web.xml` pour qu'ils soient pris en compte
- On peut aussi entourer les composants que l'on veut enlever temporairement par la balise `<ui:remove>`

R. Grin

JSF

page 44

Backing bean et portée CDI (*scope*)

R. Grin

JSF

page 45

Backing bean

- Backing bean géré par CDI, annoté par `@Named`
- (Ne pas utiliser les beans gérés par JSF, annotés par `@ManagedBean` !)
- Initialiser les ressources injectées dans une méthode annotée par `@PostConstruct`, pas dans un constructeur (il existe aussi `@PreDestroy` pour « faire le ménage » avant la suppression du bean)

Déjà vu pour quel autre composant ?

R. Grin

JSF

page 46

Exemples

- Ce bean sera désigné par `employeBean` :

```
@Named
@RequestScoped
public class EmployeBean { ... }
```

- Celui-ci sera désigné par `employe` :

```
@Named("employe")
@RequestScoped
public class EmployeBean { ... }
```

R. Grin

JSF

page 47

Bean géré

- Quand une page JSF désigne un bean, par exemple `#{inscriptionBean.nom}` le container CDI regarde s'il existe un bean du même type dans la portée
- Si c'est le cas, le bean existant est utilisé par la page JSF
- Sinon un nouveau bean est créé par le container et utilisé par la page JSF

R. Grin

JSF

page 48

getter

- ❑ Quand une page JSF utilise une propriété d'un backing bean, le getter de la propriété peut être appelé plusieurs fois
- ❑ Un getter ne doit donc pas exécuter à chaque appel un code coûteux, tel qu'un accès à une base de données

Exemple

```
@Named
@ViewScoped
public class CustomerMBean implements Serializable {
    private List<Customer> customerList;
    @EJB
    private CustomerManager customerManager;

    public List<Customer> getCustomers() {
        if (customerList == null) {
            customerList = customerManager.findAll();
        }
        return customerList;
    }
}
```

Qui peut expliquer ?

Si on veut recharger la liste, il faut tenir compte du cache JPA de 2^{ème} niveau

Utilité de la portée

- ❑ Détermine la durée de vie du backing bean
- ❑ Si la portée est la session, il va durer jusqu'à la fin de la session de travail de l'utilisateur ; si une autre expression EL y fait référence pendant la même session, c'est le même bean qui sera utilisé
- ❑ On peut ainsi conserver des informations entre 2 requêtes, 2 pages ou 2 traitements
- ❑ Il faut choisir la bonne portée, pour ne pas encombrer la mémoire du serveur tout en permettant le partage d'information

Rappel des portées CDI

- ❑ **Dependent** : (portée par défaut) nouvel objet créé à chaque référence ; l'objet est supprimé quand l'objet dans lequel il est injecté est supprimé
- ❑ **RequestScoped** : requête HTTP
- ❑ **ViewScoped** : associé à une vue (page JSF)
- ❑ **FlowScoped** : associé à un ensemble de vues
- ❑ **ConversationScoped** : début et fin par code Java
- ❑ **SessionScoped** : session de travail de l'utilisateur
- ❑ **ApplicationScoped** : durée de vie de l'application ; partagé entre tous les utilisateurs

Serializable

- ❑ Un backing bean qui a une autre portée que **RequestScoped** et **Dependent** doit implémenter **Serializable** car il peut être retiré temporairement de la mémoire centrale par le container

2 types de portée

- ❑ JSF a aussi ses portées qui étaient associées à l'ancienne annotation **@ManagedBean** (*deprecated* et remplacée par **@Named**) mais il faut utiliser les portées CDI
- ❑ Attention à importer le bon paquetage
- ❑ Ne jamais utiliser les portées du paquetage **javax.faces.bean** !

Le cycle de vie

R. Grin

JSF

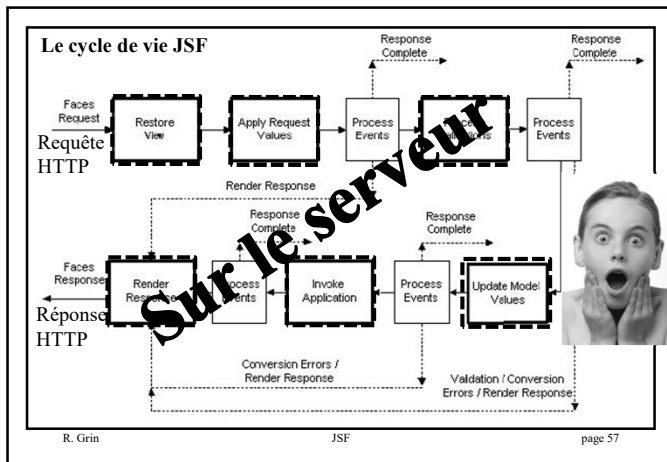
page 55

- Indispensable de bien comprendre tout le processus qui se déroule entre
 - la soumission d'un formulaire par l'utilisateur
 - la réponse du serveur sous la forme d'une page HTML

R. Grin

JSF

page 56



Simple demande d'une page

- Cas simple d'une requête HTTP GET d'un client qui demande une page JSF, *sans passer de paramètres dans la requête*

R. Grin

JSF

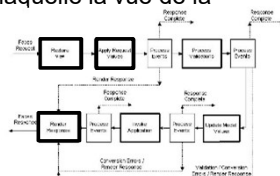
page 58

Cycle de vie simplifié

- La phase « Restore View » construit une vue vide (ne contient aucun composant Java)
- La phase « Apply Request Values » s'aperçoit qu'il n'y a pas de paramètres dans la requête
- Le cycle de vie avance donc à la dernière phase « Render Response » dans laquelle la vue de la page JSF demandée est créée, et codée en une page HTML envoyée au client

R. Grin

JSF



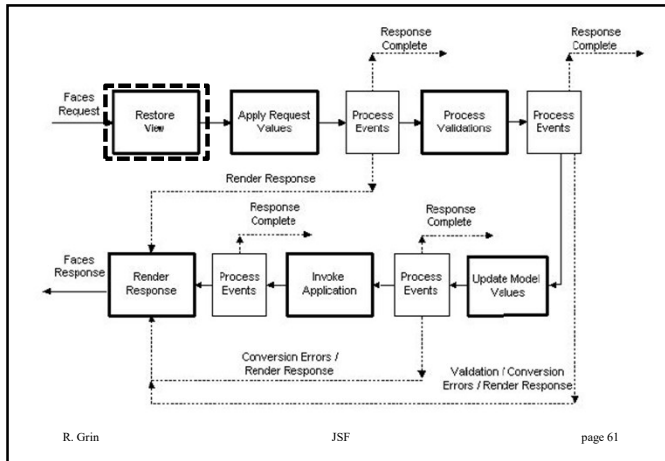
Traitement d'un formulaire

- Cas d'une requête HTTP POST générée par la soumission d'un formulaire d'une page JSF
- Sur le poste client, l'utilisateur a saisi des valeurs dans ce formulaire
- Ces valeurs sont mises en paramètres de la requête POST envoyée au serveur Web

R. Grin

JSF

page 60



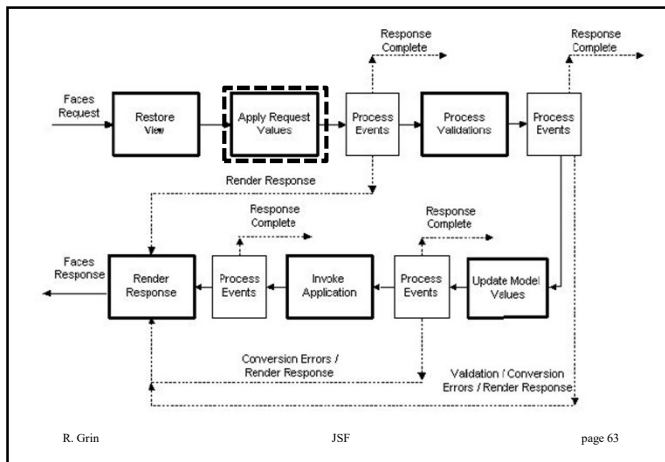
Phase de restauration de la vue

- La vue qui correspond à **la page qui contient le formulaire** est restaurée (phase « Restore View »)

R. Grin

JSF

page 62



Phase d'application des paramètres

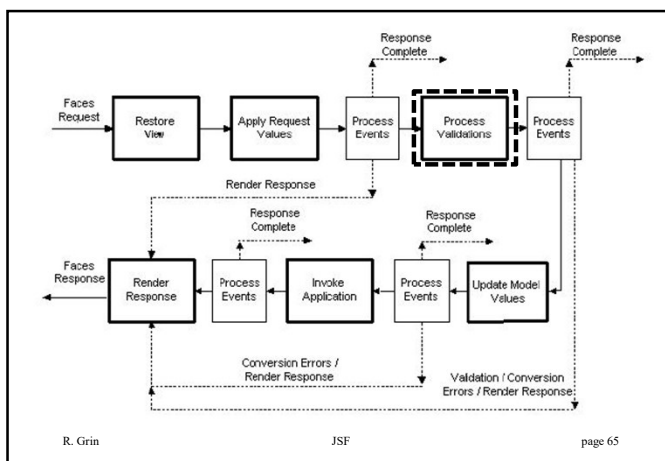
- Les valeurs des paramètres de la requête HTTP (saisies par l'utilisateur dans le formulaire) sont attribuées aux composants Java de la vue : phase « Apply Request Values »
- Par exemple, si l'utilisateur a saisi un âge dans un composant `<h:inputText>` du formulaire HTML, le composant Java associé met ce nom dans une variable interne

De quel type ?

R. Grin

JSF

page 64



Phase de validation

- Les expressions EL sont utilisées pour savoir dans quelles propriétés Java mettre les valeurs récupérées à l'étape précédente :
`<h:inputText value="#{bean.age}" ... />`
- Les données sont converties dans le type Java et ensuite validées pour voir si elles respectent les contraintes indiquées dans la page JSF et le bean
- Si une conversion ou validation échoue, la phase suivante sera la phase « Rendu de la réponse » qui retourne le formulaire avec tous les messages d'erreur

R. Grin

JSF

page 66

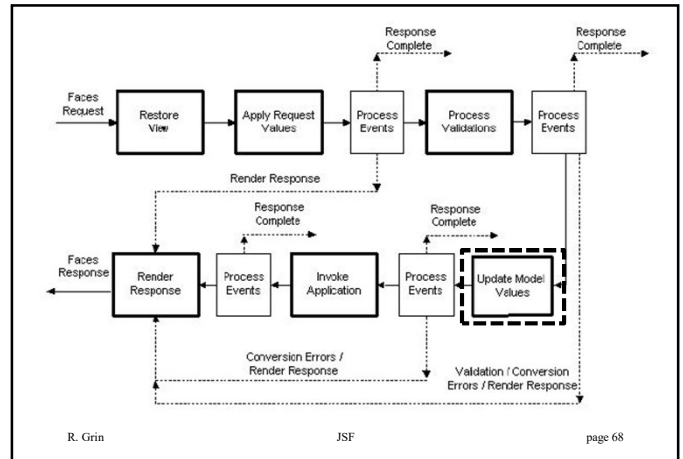
Validation sur le client HTTP

- Possible de valider avec une fonction JavaScript par une balise JSF personnalisée Intérêt ?
- Attention, ne remplace pas une validation sur le serveur Pourquoi?

R. Grin

JSF

page 67



R. Grin

JSF

page 68

Phase de mise à jour du modèle

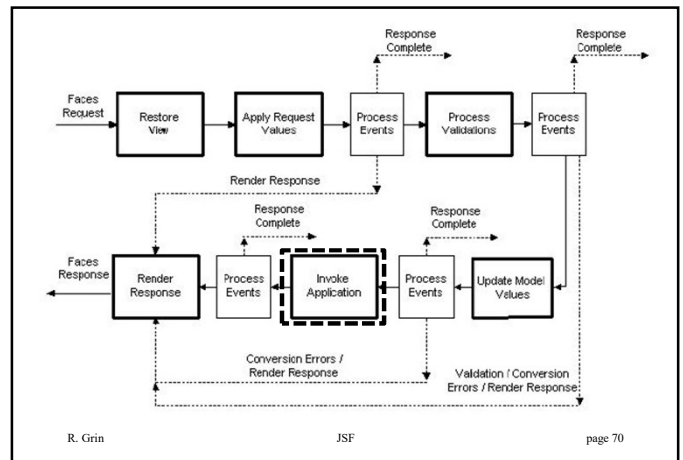
- Toutes les données ont été validées ; elles peuvent être mises dans les propriétés des *backing beans* associées (expressions EL) :

```
<h:inputText id="nom"
  value="#{bean.utilisateur.nom}" />
```

R. Grin

JSF

page 69



R. Grin

JSF

page 70

Phase d'invocation de l'application

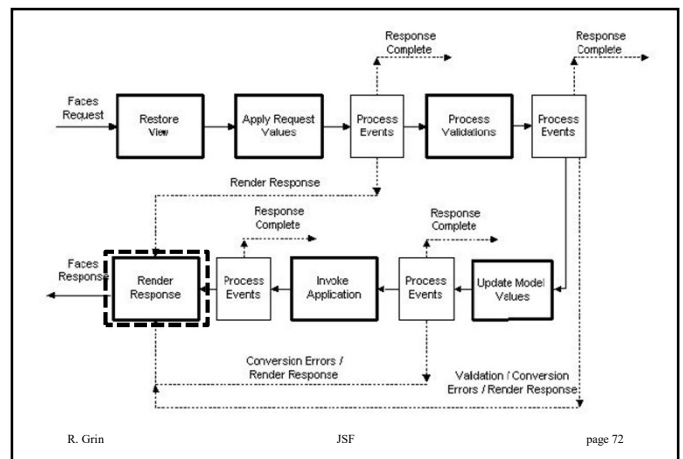
- Maintenant que les données sont enregistrées dans les backing beans, l'action associée au bouton ou au lien cliqué par l'utilisateur peut être lancée ; une méthode des backing beans est exécutée :

```
<h:commandButton ...
  action="#{inscriptionBean.confirmer}" />
```
- La valeur de retour de cette méthode détermine la prochaine page à afficher (navigation)

R. Grin

JSF

page 71



R. Grin

JSF

page 72

Phase de rendu de la réponse

- La page déterminée par la valeur retour de la méthode « action » est codée en HTML et envoyée vers le client HTTP

R. Grin

JSF

page 73

Cycle de vie avec une requête GET

- Si l'URL de la requête contient des paramètres `http://serveur/appli/adresse?p1=v1&p2=v2` et si la page JSF demandée contient des paramètres de vue (étudiés plus loin) pour ranger ces valeurs dans un backing bean, le cycle de vie complet est exécuté
- Comme si les valeurs des paramètres de la requête avaient été saisies par un utilisateur dans un formulaire soumis avec une méthode POST

R. Grin

JSF

page 74

Sauter des phases

- Il est quelquefois indispensable de sauter des phases du cycle de vie
- Il est possible de sauter toutes les phases restantes avec la méthode `FacesContext.responseComplete()`, par exemple si la réponse a déjà été générée sous la forme d'un fichier PDF
- L'attribut « `immediate="true"` » permet aussi de sauter des phases pour certains composants, par exemple pour un bouton d'annulation de formulaire

R. Grin

JSF

page 75

Configuration

R. Grin

JSF

page 76

Exemple de web.xml (1/2)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="4.0"
  xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd">
  <context-param>
    <param-name>javax.faces.PROJECT_STAGE</param-name>
    <param-value>Development</param-value>
  </context-param>
```

Production
pour livrer au client

R. Grin

JSF

page 77

Exemple de web.xml (2/2)

```
<session-config>
  <session-timeout>30</session-timeout>
</session-config>
<welcome-file-list>
  <welcome-file>index.xhtml</welcome-file>
</welcome-file-list>
</web-app>
```

R. Grin

JSF

page 78

Phase de projet

- ❑ Attention, la phase de projet **Development** ajoute automatiquement les messages d'erreurs, même s'il n'y a aucun composant pour les messages dans la page (<h:message> ou <h:messages>)
- ❑ Ne pas oublier de mettre ces composants dans la page pour que l'utilisateur voie les messages quand l'application sera **en production**

R. Grin

JSF

page 79

beans.xml

- ❑ Configuration pour CDI
- ❑ Sous WEB-INF si fichier WAR ou sous META-INF sinon

R. Grin

JSF

page 80

WEB-INF/faces-config.xml

- ❑ Configuration pour JSF
- ❑ Par exemple pour indiquer des fichiers de textes pour l'internationalisation (<resource-bundle>) ou pour donner des règles de navigation

R. Grin

JSF

JSF - page 81

Depuis JSF 2.3 (Java EE 8)

- ❑ Pour utiliser les ajouts de JSF 2.3 liés aux injections de beans CDI, il faut annoter un bean CDI avec @FacesConfig :

```
@ApplicationScoped
@FacesConfig
public class Config {
}
```

R. Grin

JSF

page 82

Navigation

R. Grin

JSF

page 83

Navigation dans JSF

- ❑ Navigation = affichage d'une nouvelle page
- ❑ Composants JSF standards pour la navigation :
<h:commandButton>, <h:button>, <h:commandLink>, <h:link>, <h:outputLink>
- ❑ <h:commandButton> et <h:commandLink> doivent être dans une balise <h:form> ; ils soumettent le formulaire
- ❑ Les autres composants peuvent ne pas être dans un formulaire et, s'ils le sont, ils ne le soumettent pas

R. Grin

JSF

page 84

POST ou GET

- `<h:commandButton>` et `<h:commandLink>` soumettent le formulaire par une requête POST et affichent la page indiquée par l'attribut **action** :

```
<h:commandButton value="Voir liste"
                 action="liste.xhtml" />
```

- `<h:button>` et `<h:link>` lancent une requête GET et affichent la page indiquée par l'attribut **outcome** :

```
<h:link value="Voir les notes"
        outcome="/notes/voirNotes" />
```

- `<h:outputLink>` lance une requête GET et affiche la page indiquée par l'attribut **value** (page peut être en dehors de l'application) ; peut le plus souvent être remplacé par un simple « `<a href=` »)

R. Grin

JSF

page 85

Recommandation

- Requête GET si on veut juste afficher une page, par exemple afficher la liste de tous les clients
- Requête POST si l'utilisateur a saisi des données qu'on veut enregistrer sur le serveur, par exemple si l'utilisateur a modifié le nom d'un client

R. Grin

JSF

page 86

Adresse affichée par le navigateur

- Après une requête POST (`<h:commandButton>` et `<h:commandLink>`), le navigateur affiche l'URL du formulaire qui vient d'être soumis, alors que la nouvelle page est affichée
- Si on veut voir le bon URL, il faut ajouter une redirection
- Après une requête GET (`<h:button>` et `<h:link>`), le navigateur affiche le bon URL
- Dans les 2 cas, JSF complète les URL s'ils ne se terminent pas par « `.xhtml` »

R. Grin

JSF

page 87

Exemple

`<h:form>` Dans formulaire.xhtml

...

```
<h:commandButton
value="Aller à page2"
action="page2" />
```

`</h:form>`

page2.xhtml sera affichée après la soumission du formulaire, mais le navigateur affichera toujours l'URL de formulaire.xhtml

R. Grin

JSF

page 88

Navigation statique et dynamique

- La valeur de l'attribut **action** qui détermine la nouvelle page peut être
 - définie « en dur » au moment de l'écriture de l'application ; la navigation est statique
 - définie par la valeur retournée par une méthode Java ; la navigation est dynamique

R. Grin

JSF

page 89

Exemple de navigation statique

- Dans page1.xhtml :

```
<h:commandButton
action="page2"
value="Aller à page2" />
```

Page suivante si clic sur bouton ?

R. Grin

JSF

page 90

Exemple de navigation dynamique

- Dans page1.xhtml :

```
<h:commandButton  
  action="#{bean.faire()}"  
  value="Faire ..." />
```

- Dans le backing bean :

```
@Named  
@RequestScoped  
public class Bean {  
  public String faire(){  
    ...  
    if (...) return "page2";  
    else return "autrePage";  
  }  
}
```

Page suivante
si clic sur bouton ?

R. Grin

JSF

page 91

Navigation déclarative ou implicite

- Si la valeur qui détermine la nouvelle page correspond à une règle de navigation, la règle détermine la prochaine page à afficher ; c'est la navigation déclarative
- Sinon c'est la navigation implicite (ce qu'on a vu jusqu'à maintenant)

R. Grin

JSF

page 92

Exemple de règle de navigation

- Règle de navigation dans faces-config.xml :

```
<navigation-rule>  
  <from-view-id>page1.xhtml</from-view-id>  
  <navigation-case>  
    <from-outcome>ok</from-outcome>  
    <to-view-id>/succes.xhtml</to-view-id>  
  </navigation-case>  
  <navigation-case>  
    <from-outcome>ko</from-outcome>  
    <to-view-id>/echec.xhtml</to-view-id>  
  </navigation-case>  
</navigation-rule>
```

- Dans page1.xhtml :

```
<h:commandButton action="ok"  
  value="Valider" />
```

Page suivante si clic sur bouton ?

R. Grin

JSF

page 93

Exemple sans règle de navigation

Pas de règle de navigation

- Dans page1.xhtml :

```
<h:commandButton action="ok"  
  value="Valider" />
```

Page suivante si clic sur bouton ?

R. Grin

JSF

page 94

Redirection

- Dans le cas d'une navigation statique, ajouter ?faces-redirect=true à l'URL de la page à afficher :
action="page2?faces-redirect=true"
- Dans le cas d'une navigation dynamique, ajouter à la valeur retournée par la méthode :
return "page2?faces-redirect=true";

R. Grin

JSF

page 95

- Exercices 1 et 2 du TP 2 (modèle PRG)

R. Grin

JSF

page 96

PRG

Problèmes avec POST

- Un formulaire est dans une page **P1** ; la soumission du formulaire par une requête POST affiche une page **P2** en réponse
- 2 problèmes :
 - quand **P2** est affichée en réponse à la requête POST, le navigateur affiche l'adresse de **P1**
 - un refresh de **P2** après le POST soumet à nouveau le formulaire

La raison du problème d'URL

- C'est JSF qui dirige vers la nouvelle page (à la fin de la phase « Invoke Application »)
- Le navigateur n'en a pas connaissance (navigation dynamique) ; il continue à afficher l'adresse du formulaire

Refresh ou *reload* d'une page

- Si l'utilisateur demande un *refresh* de la page **P2**, c'est qu'il veut la dernière version de l'information affichée dans la page
- Le navigateur envoie donc à nouveau la requête qui a permis d'obtenir la page
- Si la page a été affichée en réponse à une requête POST, le navigateur relance donc la requête POST avec les mêmes paramètres, ce qui correspond à une nouvelle soumission du formulaire

Les conséquences du problème

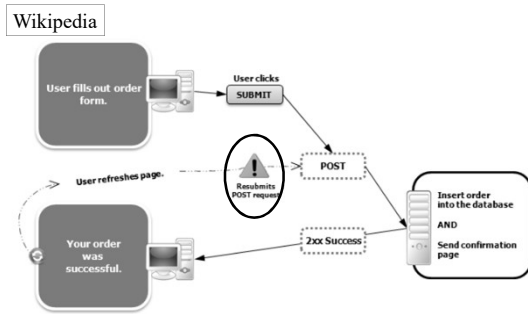
- Impossible de garder un marque-page pour la page affichée après un POST
- Commande ou paiement involontaire dû à une soumission multiple d'un formulaire

Qu'est-ce qui est le plus grave ?

La solution : Post – Redirect - Get (PRG)

- Le modèle Post – Redirect - Get :
 - Ne jamais montrer une page en réponse à un POST
 - Charger les pages uniquement avec des GET
 - Utiliser la redirection pour passer de POST à GET

Sans PRG

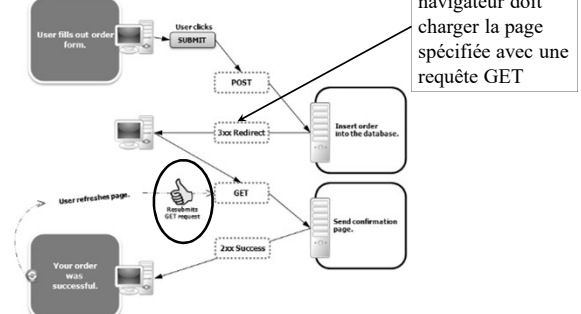


R. Grin

JSF

page 103

Avec PRG



R. Grin

JSF

page 104

Problèmes de PRG

- La redirection peut poser un problème :
 - Informations conservées dans la requête du POST ne sont plus disponibles après la redirection
 - Messages de succès ou d'information, générés par programmation, sont conservés le temps d'une requête et donc n'apparaissent plus après une redirection
- Solutions à ces problèmes dans les transparents suivants

R. Grin

JSF

page 105

Portée session ?

- Une des solutions serait de ranger les informations dans la session plutôt que dans la requête
- Mais cette solution est **mauvaise** car elle peut conduire à une session trop encombrée ou à d'autres problèmes, en particulier si l'utilisateur travaille en parallèle avec l'application en ouvrant plusieurs onglets, car la session ne fait pas de différence entre des onglets du navigateur ouverts sur un même domaine d'URL

R. Grin

JSF

page 106

Solutions

- JSF offre des possibilités pour conserver les informations plus longtemps que la requête, sans passer en portée session
- Le plus intéressant est d'utiliser les paramètres de vue (permet de rester en portée « requête »)

R. Grin

JSF

page 107

Paramètres de vue

- Une requête GET avec 2 paramètres nom et age :
http://serveur.fr/appli/page2.xhtml?nom=Bob&age=23
- Avec des paramètres de vue les valeurs « Bob » et « 23 » peuvent être récupérées dans des propriétés d'un backing bean
- Un paramètre de vue est créé par une balise <f:viewParam> placée à l'intérieur d'une balise <f:metadata> dans la page demandée par GET (page2.xhtml pour l'exemple)

R. Grin

JSF

page 108

Exemple

Requête GET
http://serveur.fr/appli/page2.xhtml?nom=Bob

- Dans page2.xhtml (juste avant <h:head>) :


```
<f:metadata>
  <f:viewParam name="nom"
               value="#{bean.p1}" />
</f:metadata>
```
- « Bob » sera mis dans la propriété p1 de bean, avant l'affichage de la page
- Dans <f:viewParam>, comme dans <h:inputText>, on peut ajouter un convertisseur, un validateur ou un attribut required

R. Grin

JSF

page 109

Placement de <f:metadata>

- Doit être au « top level », pas dans une autre balise
- Si un template est utilisé, il ne faut pas mettre <f:metadata> dans le template ; il faut le mettre dans le client du template

R. Grin

JSF

page 110

<f:viewAction>

- Balise d'une page demandée par GET ; doit être placée à l'intérieur de <f:metadata>
- Exécute une méthode d'un backing bean
- Exemple :


```
<f:viewAction action="#{bean.m()}" />
```
- Cette méthode est exécutée
 - après l'enregistrement par <f:viewParam> des valeurs des paramètres de la requête GET dans un backing bean
 - avant l'affichage de la page

R. Grin

JSF

page 111

Exemple d'utilisation

- Soit une requête GET http://.../employe.xhtml?id=12
- Un paramètre de vue de la page employe.xhtml récupère la valeur de l'id de la requête (12) dans la propriété id d'un backing bean
- L'action de <f:viewAction> récupère dans la base de données l'employé qui a l'id 12 et le met dans une propriété employe (type Employe) du backing bean
- Les informations sur l'employé sont ensuite affichées dans la page employe.xhtml, par exemple par `#{bean.employe.nom}`

R. Grin

JSF

page 112

GET http://.../employe.xhtml?id=12

Code de l'exemple

- Dans la page JSF :


```
<f:metadata>
  <f:viewParam name="id" value="#{bean.id}" />
  <f:viewAction action="#{bean.findEmp()}" />
</f:metadata>
...
#{bean.employe.nom}
```
- Dans le backing bean (id et employe sont des propriétés du bean) :


```
public void findEmp() {
  this.employe = ejb.findEmp(this.id);
}
```

Qui veut expliquer ?

R. Grin

JSF

page 113

Dans les TPs

- Vous vous souvenez quand on a utilisé <f:viewAction> dans le TP 1 ?
- Ce procédé sera encore utilisé dans d'autres TPs
- Une autre façon de faire :
 - utiliser un paramètre de vue
 - un convertisseur dans <f:viewParam> transforme l'id en Employe (remplace l'utilisation de <f:viewAction>)
- Vous vous souvenez quand on a utilisé un convertisseur dans le TP 1 ?

R. Grin

JSF

page 114

includeViewParams (1/2)

- Cet attribut peut être ajouté à tout lien vers une page qui contient des paramètres de vue :

```
<h:commandButton value=...  
  action="page2?faces-redirect=true&  
  includeViewParams=true" />
```

- `includeViewParams=true` signifie
« inclure dans le lien les paramètres qui correspondent aux paramètres de vue *de la page de destination*, avec les valeurs définies par l'expression EL du paramètre de vue »

R. Grin

JSF

page 115

includeViewParams (2/2)

- Si l'action désigne une méthode, il faut ajouter `includeViewParams=true` dans la valeur retournée par la méthode :

```
return "page2?faces-redirect=true&  
  includeViewParams=true";
```

R. Grin

JSF

page 116

Exemple

- `page2.xhtml` (backing bean **bean**) :

```
<f:viewParam name="n1"  
  value="#{bean.p1}" />
```
- Lien de `page1.xhtml` (backing bean **bean**) :

```
<h:commandButton value=...  
  action="page2?faces-redirect=true  
  &includeViewParams=true" />
```
- `includeViewParams=true` ajoutera le paramètre `n1` à la requête GET de redirection lancée après la requête POST, avec la valeur donnée par `#{bean.p1}` (au moment de l'envoi de la requête)

R. Grin

JSF

page 117

Utilité de includeViewParams

- Si `page1` et `page2` utilisent des backing beans de portée requête *de la même classe*, ça permet de passer des valeurs du backing bean de `page1` dans le backing bean de `page2`, après une redirection
- Le 1^{er} bean ne dure que le temps de la requête POST ; les valeurs des paramètres de vue de `page2` (les valeurs du 1^{er} bean) sont passées dans la requête GET de redirection
- Le 2^{ème} bean utilisé par la page d'arrivée met ces valeurs dans ses propriétés (grâce aux paramètres de vue)

R. Grin

JSF

page 118

Exemple

- `page2.xhtml` (backing bean **bean**) :

```
<f:viewParam name="n1"  
  value="#{bean.p1}" />
```
- Lien de `page1.xhtml` (backing bean **bean**) :

```
<h:commandButton value=...  
  action="page2?faces-redirect=true  
  &includeViewParams=true" />
```
- Supposons que `bean.p1 = 5` au départ de la requête ; la requête GET aura le paramètre « `n1="#{bean.p1}"` » donc « `n1=5` »
- En arrivant dans `page2.xhtml`, la valeur 5 sera donc mise dans la propriété `bean.p1`

La valeur 5 a été passée de la 1^{ère} à la 2^{ème} instance de **Bean**

R. Grin

JSF

page 119

includeViewParams avec GET

- Avec un composant « GET » comme `<h:link>`, utiliser l'attribut `includeViewParams` :

```
<h:link outcome="page"  
  includeViewParams="true" ... >
```

R. Grin

JSF

page 120

Donner une valeur à un paramètre (1/2)

- Plusieurs autres façons de donner une valeur à un paramètre de requête GET :

- `<h:link outcome="page?p1=4&p2='bibi'" ... >`
- `<h:link outcome="page?p1=#{bean.prop + 2}" ... >`
- `<h:link outcome="page" ...>`
 `<f:param name="p1" value="4"/>`
`</h:link>`

R. Grin

JSF

page 121

Donner une valeur à un paramètre (2/2)

- Dans le cas d'une requête POST avec redirection, écrire une méthode action dont la valeur de retour contient la valeur du paramètre :

```
public String faire() {  
    return "page2?nb=" + nombre  
        + "&faces-redirect=true";  
}
```

R. Grin

JSF

page 122

Messages après redirection

- Un message d'information ou de succès généré par programmation peut être affiché après une redirection en utilisant la classe `Flash` (voir TP ou la section suivante)

R. Grin

JSF

page 123

- Finir TP 2
- TP 3 (templates)
- Étudier la suite de ce support seul

R. Grin

JSF

page 124

Messages d'erreur ou d'information / de succès

R. Grin

JSF

page 125

Cas d'utilisation

- Les conversions et validations peuvent conduire à l'affichage de messages d'erreur
- Lorsqu'une action s'est bien déroulée il faut le signaler à l'utilisateur par un message d'information / de succès

R. Grin

JSF

page 126

Affichage des messages

- ❑ `<h:messages>` indique un emplacement dans la page où sont affichés tous les messages
- ❑ `<h:message>` indique un emplacement pour afficher un message pour un seul composant identifié par l'attribut `for`
- ❑ Ces 2 balises ont de nombreux attributs

R. Grin

JSF

page 127

Format d'affichage des messages

- ❑ Les attributs `styleClass` et `style` indiquent un style CSS pour affichage des messages
- ❑ D'autres attributs définissent un style lié à la sévérité du message (`errorStyle`, `errorClass`, `infoStyle`, `infoClass`,...)

R. Grin

JSF

page 128

Exemples

- ❑ Messages de sévérité « ERROR » affichés avec la classe CSS « erreur » et messages de sévérité « INFO » affichés en vert et caractères gras
`<h:messages errorClass="erreur" infoStyle="color:green; font-weight:bold;"/>`
- ❑ `<h:messages globalOnly="true"/>` affiche la liste des messages qui ne sont pas liés à un composant particulier
- ❑ `<h:message for="nom" />`
- ❑ `<h:message for="nom" showSummary="true" showDetail="false" />`

R. Grin

JSF

page 129

Messages standards

- ❑ Les convertisseurs et validateurs standards produisent des messages d'erreur standards
- ❑ Les attributs des composants standards `requiredMessage`, `converterMessage` ou `validatorMessage` permettent de modifier ces messages
- ❑ Exemple :
`<h:inputText id="nom" required="true" requiredMessage="Le nom est requis"/>`

R. Grin

JSF

page 130

Améliorer les messages standards

- ❑ Un message d'erreur peut comporter l'identifiant du composant concerné par l'erreur
- ❑ Pour rendre les messages plus clairs, il faut donner des identifiants significatifs (attribut `id`) aux formulaires et aux composants pour éviter les « `j_idt6:j_idt8` » dans les messages
- ❑ On peut aussi utiliser l'attribut `label` des composants ; ce label sera utilisé pour désigner le composant dans les messages

R. Grin

JSF

page 131

Messages personnalisés

- ❑ Les convertisseurs et validateurs personnalisés permettent aussi d'afficher facilement des messages d'erreur
- ❑ S'ils ne peuvent pas être utilisés (par exemple validation qui englobe plusieurs composants) ou pour afficher un message de succès, le développeur peut ajouter ses messages dans les pages JSF par programmation Java
- ❑ Les transparents suivants étudient les classes et méthodes Java utilisées pour afficher ces messages

R. Grin

JSF

page 132

Classe **FacesMessage**

- La classe `FacesMessage` du paquetage `javax.faces.application` représente un message affiché par `<h:messages>` ou `<h:message>`
- Propriétés d'un message :
 - sévérité (`SEVERITY_FATAL`, `SEVERITY_ERROR`, `SEVERITY_WARN`, `SEVERITY_INFO` de la classe interne `Severity`)
 - message détaillé
 - message résumé

R. Grin

JSF

page 133

Faire afficher son propre message

- Il suffit de passer une instance de `FacesMessage` à la méthode `addMessage` de `FacesContext` pour que le message soit affiché par les `<h:messages>` ou `<h:message>`
- `addMessage` a 2 paramètres :
 - id client du composant avec lequel le message est associé (de type `String` ; `null` si le message n'est pas attaché à un client particulier)
 - Le message (classe `FacesMessage`)

R. Grin

JSF

page 134

Exemple

```
String msgResume = "...";
String msgDetail = "...";
FacesMessage.Severity niveau =
    FacesMessage.SEVERITY_ERROR;
FacesMessage fm =
    new FacesMessage(niveau,
        msgResume, msgDetail);
FacesContext.getCurrentInstance().addMessage(null, fm);
```

Message pas lié à un composant

R. Grin

JSF

page 135

Exemple (lié à un composant)

```
FacesContext context = FacesContext.getCurrentInstance();
context.addMessage(
    "Insertion:Login",
    new FacesMessage(FacesMessage.SEVERITY_ERROR,
        "Le message résumé",
        "Le message détaillé"));
```

id client du composant

R. Grin

JSF

page 136

Message de validation ou conversion

- Quand il y a une erreur dans un convertisseur ou un validateur, le code doit lancer une exception (`ConverterException` ou `ValidatorException`) et c'est le message passé au constructeur de l'exception qui est affiché
- Pas besoin de préciser le composant associé au message puisque c'est celui qui est validé

R. Grin

JSF

page 137

Exemple de message dans validateur

```
public void validateId(FacesContext context,
    UIComponent composant, Object id) {
    Compte compte = ejb.getCompte((Long) id);
    if (compte == null) {
        FacesMessage message =
            new FacesMessage("Pas de compte avec id " + id);
        throw new ValidatorException(message);
    }
}
```

R. Grin

JSF

page 138

Id client d'un composant

- ❑ Si on veut associer un message à un composant et qu'on n'est pas dans un convertisseur ou un validateur il faut trouver l'id client du composant
- ❑ Exemple de désignation d'un composant : si le composant a l'id « montant » dans le formulaire d'id « transfert », l'id du composant à passer en 1^{er} paramètre de addMessage est « transfert:montant »

R. Grin

JSF

page 139

Identificateur « client »

- ❑ Pas toujours facile de trouver les id « client »
- ❑ Regarder le code HTML généré aide
- ❑ Identificateur client composé de plusieurs niveaux, avec le séparateur « : » ; par exemple, « formulaire:j_idt41 »
- ❑ Pour faciliter, donner des ids aux composants (y compris composants englobants) et/ou utiliser l'attribut prependId="false" pour les formulaires qui les contiennent
- ❑ Identificateur doit exister dans la page HTML générée (attention aux rendered="false")

R. Grin

JSF

page 140

Message après redirection

- ❑ Puisqu'un message ne dure qu'une seule requête, il ne s'affichera pas après une redirection
- ❑ Si on veut qu'il s'affiche, il faut un code un peu plus complexe qui utilise la classe Flash

R. Grin

JSF

page 141

Ajouter un message « flash »

```
public void addFlashMessage(FacesMessage message) {  
    FacesContext facesContext = FacesContext.getCurrentInstance();  
    Flash flash = facesContext.getExternalContext().getFlash();  
    flash.setKeepMessages(true);  
    facesContext.addMessage(null, message);  
}
```

R. Grin

JSF

page 142

Messages internationalisés

- ❑ L'internationalisation d'une application n'est pas difficile mais pas étudiée dans ce cours d'introduction
- ❑ Juste une petite difficulté pour faire afficher des messages créés par programmation, qui dépendent de la langue il faut récupérer le ResourceBundle qui contient les textes en différentes langues :

```
ResourceBundle textes =  
    ResourceBundle.getBundle(CHEMIN_BUNDLE);  
...  
texte = textes.getString(nomPropTexte);
```

R. Grin

JSF

page 143

Événements

R. Grin

JSF

page 144

Types d'événements

- 2 grands types d'événements :
 - Générés par une action de l'utilisateur (de type « *application* »)
 - Générés à un moment du cycle de vie (de type « *lifecycle* »)

R. Grin

JSF

page 145

Événements « application »

- ValueChangeEvent : valeur d'un composant modifiée par l'utilisateur (champ de saisie de texte, menu, liste déroulante,...) ; liés aux composants tels que `<h:selectOneMenu>` ou `<h:inputText>`
- ActionEvent : clic sur un bouton ou un lien ; liés aux composants tels que `<h:commandButton>` ou `<h:commandLink>`

R. Grin

JSF

page 146

Écouteur

- Les événements sont traités par des écouteurs (*listeners*) durant le cycle de vie JSF par,
 - des méthodes (attribut xxxListener)
 - ou des classes (sous-balise `<f:xxxListener>`)
- Remarque : les actionListeners sont exécutés avant les méthodes « action », durant la phase « Invoke Application » du cycle de vie

R. Grin

JSF

page 147

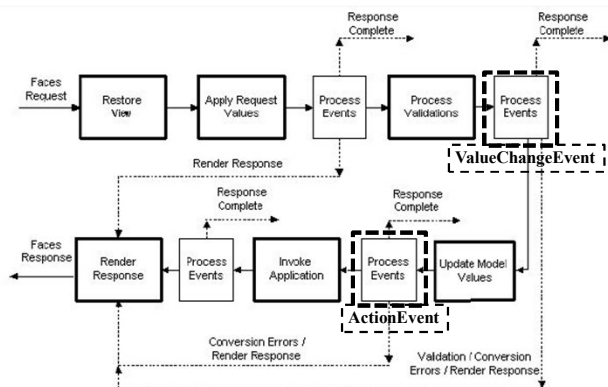
Action ou ActionListener ?

- Action pour les traitements métier
- ActionListener pour les événements liés à l'interface utilisateur
- Action n'a pas accès aux composants de l'interface utilisateur alors que ActionListener y a accès (par le paramètre ActionEvent de la méthode qui est lancée)

R. Grin

JSF

page 148



R. Grin

JSF

page 149

Exemples

- Classe écouteur :

```
<h:selectOneMenu value="#{bean.codePays}"
  onchange="submit()" />
<f:valueChangeListener
  type="fr.unice.jsf.PaysListener" />
<f:selectItems value="#{bean.listePays}" />
</h:selectOneMenu>
```

- Méthode écouteur :

```
<h:selectOneMenu value="#{bean.codePays}"
  onchange="submit()"
  valueChangeListener="#{bean.modifPays}" />
<f:selectItems value="#{bean.listePays}" />
</h:selectOneMenu>
```

R. Grin

JSF

page 150

Méthode valueChangeListener

- **ValueChangeEvent** en paramètre et **void** en retour

- Exemple :

```
public void modifPays(ValueChangeEvent evt) {  
    for (Locale locale : pays) {  
        if (locale.getCountry().equals(evt.getNewValue())) {  
            FacesContext.getCurrentInstance()  
                .getViewRoot().setLocale(locale);  
        }  
    }  
}
```

Que fait cette méthode ?

R. Grin

JSF

page 151

Événements « cycle de vie »

- **PhaseEvent** : provoqué par un changement de phase du cycle de vie. Par exemple, **APPLY_REQUEST_VALUES**
- Système (il y a beaucoup d'événements de ce type) : provoqué par divers « sous-phases » du cycle de vie ; par exemple, **PreRenderComponentEvent** ou **PostValidateEvent**

R. Grin

JSF

page 152

Ajax

R. Grin

JSF

page 153

Ajax

- *Asynchronous Javascript and XML*
- Une requête Ajax envoyée par le client est lancée « en arrière-plan » ; l'utilisateur peut continuer à travailler avec la page en cours
- La réponse du serveur à la requête ne modifie qu'une partie de la page en cours

R. Grin

JSF

page 154

<f:ajax>

- Ajoute des fonctionnalités « Ajax » au composant JSF dans lequel il est inclus
- Un événement déclencheur sur ce composant provoque l'envoi d'une requête Ajax au serveur sans attendre la soumission du formulaire
- Par exemple, une liste déroulante peut envoyer une requête Ajax quand l'utilisateur fait un nouveau choix ; la réponse du serveur ne mettra à jour qu'une partie de la page, par exemple une 2^{ème} liste déroulante dont les valeurs dépendent du choix fait par l'utilisateur dans la 1^{ère} liste

R. Grin

JSF

page 155

<f:ajax> - les attributs

- **event** : événement déclencheur de la requête (doit être un événement compatible avec le composant) ; valeur par défaut dépend du composant, par exemple, **valueChange** pour les **<h:inputText>** ou **action** pour les **<h:commandButton>**
- **execute** : composants pris en compte pour l'envoi de la requête (espace séparateur) ; valeur par défaut **@this**, le composant qui contient **<f:ajax>**
- **render** : composants mis à jour au moment de la réponse du serveur (espace séparateur) ; valeur par défaut **@none**

R. Grin

JSF

page 156

Exemple 1

```
<h:form prependId="false">
  <h:inputText value=... />
  <h:commandButton value="Enregistrer" ...>
    <f:ajax execute="@form"
      render="out1 out2" />
  </h:commandButton>
  <h:outputText id="out1" .../>
  <h:outputText id="out2" .../>
</h:form>
```

Formulaire englobant

Exemple 2

```
<h:form>
  <h:input value="#{bean.prop}" />
  <h:selectOneMenu value="#{bean.selected}">
    <f:selectItems value="#{bean.items}" />
    <f:ajax execute="@form"
      render="@form" />
  </h:selectOneMenu>
  ...
</h:form>
```

Identificateurs « client »

- Utilisés dans les attributs **execute** et surtout **render**
- Revoir section « Messages d'erreur ou d'information »

Mélange d'Ajax et de requête normale

- Quand un champ de saisie lance une requête Ajax, et que l'utilisateur appuie sur la touche [Entrée] après avoir saisi la valeur, un message d'avertissement indique qu'une requête Ajax a été mélangée avec une requête « normale » (l'appui sur la touche [Entrée] déclenche la soumission du formulaire en même temps que l'action « Ajax »)
- Pour ne plus avoir cet avertissement, il suffit de passer le bouton de soumission en Ajax :

```
<h:commandButton ...><f:ajax execute="@form"
  render="@form"/></h:commandButton>
```

Listener

- L'attribut **listener** de **<f:ajax>** désigne une méthode qui est appelée quand l'événement déclencheur de l'appel Ajax survient :

```
<f:ajax event="keyup" render="table"
  listener="#{bb.m}" />
```
- Le plus souvent la méthode modifie le backing bean utilisé par la page, ce qui modifie la page

Exemple – valeur composant

```
public void m(AjaxBehaviorEvent event) {
  UIInput input = (UIInput) event.getComponent();
  UnType valeur = (UnType) input.getValue();
  ...
}
```

La méthode « listener » doit avoir un paramètre de type **AjaxBehaviorEvent**

Exemple d'utilisation

- Un écouteur peut récupérer
 - un client à partir d'un id saisi par l'utilisateur
 - tous les noms qui commencent par les caractères tapés par l'utilisateur sont affichés comme suggestion (complétion de nom)

JSF et HTML5

Pages HTML5

- Les pages HTML générées par JSF peuvent être déclarées comme des documents HTML5 avec `<!DOCTYPE html>`

Composants HTML5

- Avant JSF 2.2, pour profiter d'un nouveau composant HTML dans JSF il fallait écrire un composant JSF
- JSF 2.2 permet de profiter des services offerts par JSF (validation de données, affichage automatique des messages d'erreur, utilisation simple d'Ajax, ...) tout en utilisant un attribut ou un composant HTML5

2 possibilités

- On peut utiliser un attribut HTML5 (attribut « *pass-through* ») dans un composant JSF
- Exemple : attribut `placeholder` de `<input>` alors qu'il n'est pas un attribut de `<h:inputText>`
- L'attribut sera passé tel quel à la balise HTML générée à la fin du cycle de vie JSF
- On peut aussi utiliser une balise HTML5, tout en profitant des facilités de JSF avec des attributs traités par JSF

Attributs HTML5 en JSF

- Etudions d'abord le cas d'un attribut HTML5 dans un composant JSF

Passer un attribut HTML5

Remarque : donner un autre alias que « p » si PrimeFaces est utilisé

1. Avec l'espace de noms « *passthrough* » :

```
<html ...
xmlns:p="http://xmlns.jcp.org/jsf/passthrough">
  <h:form>
    <h:inputText value="#{bean.nom}"
      p:placeholder="Votre nom"/>
  </h:form>
</html>
```

2. Avec une balise `<f:passThroughAttribute>` :

```
<h:inputText value="#{bean.nom}" >
  <f:passThroughAttribute
    name="placeholder" value="Votre nom" />
</h:inputText>
```

R. Grin

JSF

page 169

Balise HTML5 en JSF

- Etudions maintenant comment insérer une balise HTML5 dans une page JSF, tout en profitant des avantages de JSF (expressions EL comme valeurs, conversions, validations, messages d'erreur,...)

R. Grin

JSF

page 170

Composants HTML5 *pass-through*

- Une page JSF peut contenir des balises HTML5
- Pour qu'un élément HTML5 soit traité par JSF il suffit qu'un de ses attributs ait l'espace de noms `http://xmlns.jcp.org/jsf` :

```
<html xmlns:jsf="http://xmlns.jcp.org/jsf">
...
<input type="text" jsf:id="nom"
  placeholder="Votre nom"
  value="#{bean.name}"/>
```

R. Grin

JSF

page 171

Exemple

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:jsf="http://xmlns.jcp.org/jsf">
  <form jsf:id="form">
    <input name="nom" type="text"
      jsf:id="nom" value="#{bean.nom}"/>
    <input type="submit" value="Soumettre" />
    <p>Texte saisi : #{bean.nom}</p>
  </form>
</html>
```

la valeur de **value** sera associée à la propriété **nom** du backing bean

R. Grin

JSF

page 172

Composant HTML5 traité par JSF

- Le composant HTML5 est transformé en un composant JSF suivant un mapping qui tient compte du nom de la balise et éventuellement de certains attributs de la balise
- Exemple : `<input type="file">` est transformé en `<h:inputFile>`

R. Grin

JSF

page 173

Composant HTML5 sans équivalent JSF

- Si le composant HTML5 n'a pas de correspondance dans JSF, JSF crée un composant spécial de balise `<jsf:element>`
- On peut ajouter des capacités Ajax à un tel composant (voir exemple à suivre), mais seulement quelques attributs JSF liés aux événements JavaScript
- Les valeurs des attributs HTML5 peuvent contenir des expressions EL comme pour les composants JSF

R. Grin

JSF

page 174

Exemple

- Ajouter un comportement Ajax à la balise `div` :

```
<div jsf:id="compteurClics">
  Clics: #{bean.compteurClicsSouris}
  <f:ajax event="click" render="@this"
    listener=#{bean.incrementer} />
</div>
```

R. Grin

JSF

page 175

Exemple avec `<input type="date">`

```
<h:form>
  <h:outputLabel value="Date : " for="date" />
  <input type="date" jsf:id="date" name="date"
    value=#{reservationBean.date}>
    <f:convertDateTime pattern="yyyy-MM-dd" />
  </input>
  <br/>
  <h:commandButton value="Enregistrer la date"
    action="resultat" />
</h:form>
```

Le type de `date` est `java.util.Date`

R. Grin

JSF

page 176

Ressources

R. Grin

JSF

page 177

Ressources des pages Web

- Les pages HTML font le plus souvent appel à des ressources externes : images, fichier CSS, code Javascript,...
- Il faut utiliser des balises JSF dédiées aux ressources à la place des balises HTML ; en ce cas JSF va chercher les ressources dans le répertoire **resources** placé sous la racine des pages Web

R. Grin

JSF

page 178

CSS

- Le plus souvent on met les fichiers CSS dans un sous-répertoire **css** du répertoire **resources**

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

R. Grin

JSF

page 179

JavaScript

- Le plus souvent on met les fichiers JavaScript dans un sous-répertoire **js** du répertoire **resources**
- ```
<h:outputScript name="js/jsf.js" />
```
- Par défaut, le script sera placé dans la page HTML à l'endroit où on a mis la balise ; on peut ajouter un attribut **target** pour indiquer où le mettre (par exemple **target="head"** pour le mettre à la fin de la section head de la page)

R. Grin

JSF

page 180

## Image

```
<h:graphicImage name="image/photo.jpg" />
```

R. Grin

JSF

page 181

## Thèmes

- Depuis JSF 2.2 on peut avoir des thèmes qui réunissent les ressources en utilisant l'attribut **library** des balises que l'on vient de voir
- En ce cas, chaque librairie correspondra alors à un thème différent, par exemple on aura un thème « default » et un thème « dark »
- Sous chacun des répertoires default et dark (placés sous le répertoire resources), on retrouvera les répertoires css, js et image qu'on vient de voir
- Il suffit alors de changer de librairie pour changer à la fois le CSS, le JavaScript et les images

R. Grin

JSF

page 182

## Flots (*faces flow*)

R. Grin

JSF

page 183

## Présentation

- A la manière des « *wizards* », les flots de pages (en fait, flots de nœuds, comme on le verra plus loin) sont des ensembles de pages qui partagent des informations communes (entreposées dans un backing bean de portée Flow)

R. Grin

JSF

page 184

## Exemple

- Les pages qui permettent à un utilisateur de créer un compte avec ses informations personnelles, l'adresse de livraison, l'adresse de facturation et les informations sur sa carte de crédit
- Chaque type d'information est donné sur une page différente
- Souvent la dernière page demande confirmation de toutes les données saisies et on peut lancer leur enregistrement ou un traitement avec les informations recueillies sur toutes les pages

R. Grin

JSF

page 185

## Navigation pour entrer et sortir du flot

- On ne peut entrer dans le flot que par la page de démarrage (utiliser une requête GET pour aller vers cette page) ; les autres pages du flot ne sont pas atteignables de l'extérieur du flot
- Quand on est dans le flot, on ne peut aller que vers des pages du flot ou vers la page de sortie
- Pour sortir du flot il suffit de naviguer depuis une page du flot vers la page de sortie ou vers une page qui n'est pas dans le flot (il faut y aller avec une requête POST, éventuellement avec une redirection ; une requête GET ne marche pas)

R. Grin

JSF

page 186

## Suppression des backing beans

- Dès que la navigation amène à la page de sortie, les backing beans de portée flow sont supprimés automatiquement par CDI

R. Grin

JSF

page 187

## Navigation par requête POST

- La navigation ne marche pas toujours bien entre pages du flot si on utilise des requêtes GET
- Il est donc conseillé de n'utiliser que des requêtes POST, avec éventuellement des redirections pour avoir des URL affichés qui correspondent aux pages affichées

R. Grin

JSF

page 188

## Plusieurs flots

- Les flots peuvent être enchaînés séquentiellement ou bien emboîtés les uns dans les autres

R. Grin

JSF

page 189

## Portée

- Comme la portée « conversation » la portée « flot » (annotation des backing beans par `@FlowScoped`) est entre la portée requête et la portée session
- La portée flot est une portée CDI qui commence quand l'utilisateur navigue dans la page d'entrée du flot et se termine quand l'utilisateur quitte le flot en navigant vers une page de sortie du flot
- Au contraire de la portée session, la portée flot n'a pas de problème si l'utilisateur ouvre plusieurs onglets du navigateur sur la même application

R. Grin

JSF

page 190

## Utilité

- L'utilisation d'un enchaînement de pages pour obtenir de l'utilisateur un ensemble d'informations est fréquent
- Avant JSF 2.2, un backing bean de portée conversation était le plus souvent utilisé, avec les difficultés occasionnées par cette portée
- Un flot de pages
  - simplifie la programmation d'une telle situation
  - facilite la modularité
  - peut être réutilisé dans des applications différentes

R. Grin

JSF

page 191

## Similaire à une méthode Java

- Un seul point d'entrée, plusieurs points de sortie possibles
- Paramètres d'entrée et valeur de retour
- Peut être appelé d'un autre point de l'application
- Interface bien définie mais les détails de l'implémentation sont cachés
- Un flot peut en appeler un autre et les portées s'empilent et se déplient comme dans une pile d'exécution de méthodes

R. Grin

JSF

page 192



## Définition d'un flot

- Suit le principe « Convention plutôt que configuration » : on peut définir un flot en créant un répertoire dans la racine des pages JSF de l'application
- Si on n'utilise pas la « convention », le flot peut être configuré dans le fichier `<nom-flot>-flow.xml` placé dans le répertoire qui contient les pages du flot, dans le fichier `WEB-INF/faces-config.xml` ou bien par programmation avec un constructeur de Flow CDI (pas étudié dans ce support)

R. Grin

JSF

page 193

## Définition d'un flot par convention

- Répertoire dans la racine des pages, du même nom (identificateur du flot) que le flot
- Dans le répertoire,
  - Une page du même nom que le répertoire (et suffixe .xhtml) comme nœud de démarrage du flot
  - Les autres pages qui sont dans le flot
  - Un fichier de configuration obligatoire nommé `<nom-flot>-flow.xhtml` (vide si on n'a rien à configurer)
- En dehors du répertoire une seule page de sortie de nom `<nom-flot>-return.xhtml`

R. Grin

JSF

page 194

## Code pour entrer et sortir

- Les exemples suivants supposent que la convention a été suivie (pas de configuration)
- Exemple de code pour entrer dans le flot nommé `flot1` (page `flot1.xhtml` mise dans le répertoire `flot1`) :  

```
<h:button id="start1"
 value="Entrer dans Flot1"
 outcome="flot1"/>
```
- Exemple de code pour sortir du flot nommé :  

```
<h:commandButton id="home" value="home"
 action="/flot1-return" />
```

Fichier `flot1-return.xhtml` situé en dehors du répertoire `flot1`

R. Grin

JSF

page 195

## Code du backing bean

```
@Named
@FlowScoped("flot1")
public class Flot1Bean implements Serializable {
 private String val;

 public String getVal() {
 return val;
 }

 public void setVal(String val) {
 this.val = val;
 }
}
```

Propriété utilisée  
par le flot

R. Grin

JSF

page 196

## Configuration avec fichier XML (1/2)

- Si la convention ne convient pas, on peut définir la configuration dans un fichier de configuration non vide (fichier placé dans le répertoire du flot ou fichier global `WEB-INF/faces-config.xml`)
- Il doit alors contenir au moins la définition de l'id du flot et une déclaration de page de sortie du flot (il peut y avoir plusieurs balises `<flow-return>` pour plusieurs pages de sorties) :

```
<flow-return id="first-return-id">
 <from-outcome>retour</from-outcome>
</flow-return>
```

R. Grin

JSF

page 197

## Configuration avec fichier XML (2/2)

- Le fichier peut aussi configurer
  - le nœud de démarrage
  - d'autres nœuds du flot
  - des paramètres à passer au flot (`<inbound-parameter>`)
  - un initialiseur, méthode exécutée avant l'entrée dans le flot ; pourra avoir accès aux paramètres dans la version JSF 2.3, buggé dans les versions précédentes
  - un « finalizer », méthode appelée juste avant la sortie du flot

R. Grin

JSF

page 198

## Exemple de fichier XML minimal

```
<faces-config version="2.2"
 xmlns="http://xmlns.jcp.org/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
 http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd">
 <flow-definition id="flot1">
 <flow-return id="return1">
 <from-outcome>#{bean.fin()}</from-outcome>
 </flow-return>
 </flow-definition>
</faces-config>
```

En valeur retour, l'id de la vue  
ou bien, si la vue n'est pas définie,  
le chemin par rapport à la racine ;  
par exemple /index (sans le suffixe)

R. Grin

JSF

page 199

## Types de nœuds

□ Les types de nœud sont décrits dans la javadoc de la classe `javax.faces.flow.FlowHandler` :

- View, page JSF « normale »
- Switch, liste d'expressions EL de type boolean ; la 1<sup>ère</sup> expression qui est évaluée à true détermine le prochain nœud
- Return, arrête le flot en cours et définit la page affichée à la sortie du flot (page qui n'est pas dans le flot)
- Method Call, appel d'une méthode (avec paramètres) dont la valeur de retour détermine le prochain nœud
- Faces Flow Call, appel d'un autre flot ; au retour de cet autre flot, le flot reprend la main (pile de flots)

R. Grin

JSF

page 200

## En savoir plus sur les flots...

- Les informations détaillées sur ces nœuds et la configuration d'un flot par programmation ne seront pas étudiées dans cette section d'introduction aux flots
- Références pour plus de détails :
  - Tutoriel Java EE 7, partie III :  
<https://docs.oracle.com/javaee/7/tutorial/jsf-configure003.htm#CHDGFCJF>
  - <http://www.omnifaces-fans.org/2015/11/jsf-scopes-tutorial-cdi-flow-scope.html> (extrait livre MASTERING JAVASERVER FACES 2.2 de Anghel Leonard).

R. Grin

JSF

page 201

## Bibliographie

- The Definitive Guide to JSF in Java EE 8. Bauke Scholtz et Arjan Tijns, Apress. Le plus complet, et à jour sur JSF 2.3
- Core JSF, 3<sup>ème</sup> édition mise à jour pour JSF 2.0. David Geary et Cay Horstmann. Prentice Hall. Le meilleur livre pour débiter.

R. Grin

JSF

page 202