UTILISER DES OBJETS JAVA

A. LES JAVABEANS

Définition

Certaines classes de l'application - *principalement les classes Entités* - doivent suivre les règles suivantes :

- La classe doit être publique.
- Doit avoir au moins un constructeur sans paramètres.
- Doit implémenter l'interface *Serializable* pour que l'état de ses objets puisse être enregistré à tout moment.
- Tous les attributs doivent être privés ou protégés, et accessibles uniquement via des getters et des setters.
- Ne doit jamais contenir aucun traitement supplémentaire sur les données (logique métier, accès au stockage de données, …)

Toute classe qui est définie de cette manière est un JavaBean ou Bean.

Pourquoi utiliser des JavaBeans

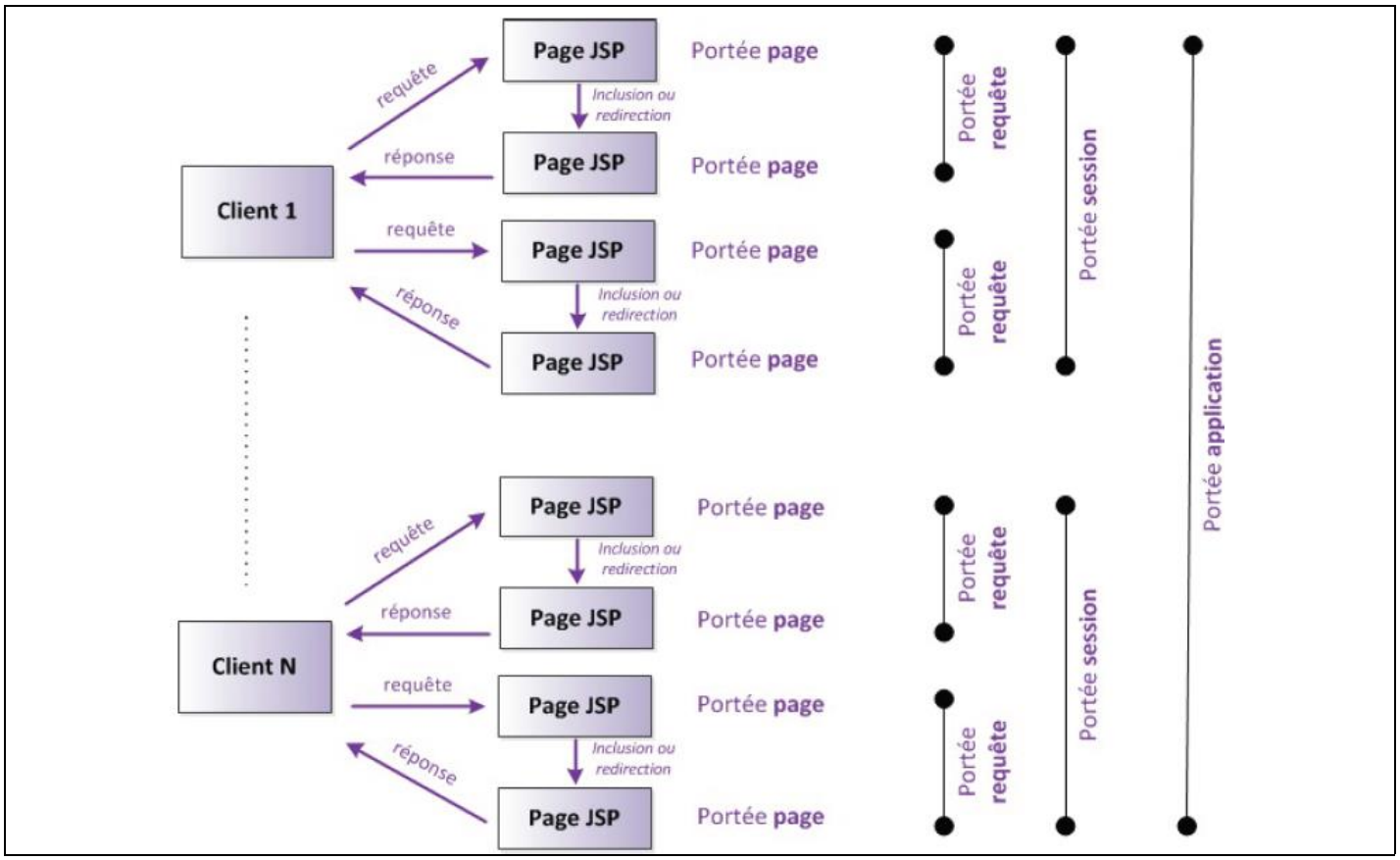
- D'après les recommandations du modèle MVC, le JavaBean est le seul composant qui doit contenir les données affichées dans la vue.
- La sérialisation.
- L'introspection : un mécanisme présent dans beaucoup de langages orientés objets (y compris Java), qui permet de connaitre les propriétés et comportements d'un objet, sans connaitre forcément son code source. Ce mécanisme est très souvent utilisé par les compilateurs orientés objets, notamment lors d'échanges d'objets entre applications (appels d'objets distants, services web, …).
- La réutilisation : le JavaBean ne participe pas directement dans l'implémentation de la couche de présentation (pas comme le contrôleur par exemple), ni dans la couche d'accès aux données, ni dans la couche métier.
  Vu son caractère indépendant, il peut donc être complètement réutilisable dans d'autres applications, quel que soit leur type (web, mobile, …).

B. LA PORTEE D'UN BEAN

Définition

Chaque bean créé dans une application JavaEE, possède une « **portée** », c'est un degré d'accessibilité, semblable à la portée d'une variable dans un programme :

- **page** : l'objet est utilisable uniquement dans la page JSP. En dehors de la page, d'autres pages ou servlets ne pourront pas le reconnaitre.
- **request** : l'objet est utilisable dans toute page ou servlet qui manipule la requête courante, tant que cette dernière existe encore. (Tant que la réponse n'a pas encore été envoyée au client)
- **session** : l'objet est utilisable dans toutes les requêtes de l'utilisateur courant, mais invisible pour les autres utilisateurs.
- **application** : l'objet est utilisable durant l'existence de toute l'application, quel que soit l'utilisateur.


Client 1
Client N
requête
réponse
Page JSP
Inclusion ou redirection
Portée page
Portée requête
Portée session
Portée application

C. LES BALISES JSP BEAN

La balise jsp:usebean

```
<jsp:usebean id="…" class="…" scope="…">
```

Permet d'utiliser un bean. Ou de le créer s'il n'existe pas dans le scope spécifié.

- **id** : identifiant de l'objet, il sera reconnu par cet identifiant dans toutes ressources (jsp, servlets) faisant partie de sa portée.
- **class** : nom complet de la classe.
- **scope** : portée du bean (page, request, session ou application). Le scope par défaut est **page**.

Exemple

```
<jsp:useBean id="prod1" class="model.Produit" scope="request"></jsp:useBean>
```

Équivalent en utilisant un scriptlet Java :

```
<% Produit prod1 = (Produit) request.getAttribute("prod1");
if(prod1 == null) {
   prod1 = new Produit();
   request.setAttribute("prod1", prod1);
} %>
```

La balise jsp:setProperty

Permet d'initialiser les champs du bean.

```
<jsp:setPorperty name="nom_du_bean" property="nom_attribut" value="une_valeur" />
```

Exemple

```
<jsp:setPorperty name="prod1" property="designation" value="une_valeur" />
```

Équivalent en utilisant un scriptlet Java :

```
<% prod1.setDesignation("une_valeur"); %>
```

La balise jsp:getProperty

Permet de récupérer la valeur d'un champ du bean.

```
<jsp:getPorperty name="nom_du_bean" property="nom_attribut" />
```

Exemple

```
<jsp:getPorperty name="prod1" property="designation" />
```

Équivalent en utilisant une balise d'expression :

```
<%= prod1.getDesignation(); %>
```

REMARQUE

Les balises jsp:bean présentent plusieurs défauts et **ne sont plus utilisées actuellement**, elles ont été remplacées par une technologie beaucoup plus efficace : *Expression Language* ou *EL*.

8. « EXPRESSION LANGUAGE »

Le « langage d'expressions » ou « Expression Language » ou « EL »

Les expressions EL représentent une syntaxe simple et efficace, qui permettent de remplacer les balises d'expression, les scriptlets et les balises jsp:bean. Càd d'enlever le code Java de la vue.

Les expressions EL peuvent **uniquement** faire :
- Afficher des valeurs,
- Evaluer des valeurs via des tests,
- Accéder aux attributs des beans ou de collections de beans.

Les structures de contrôle comme *if, switch, for, …* sont disponibles lors qu'on utilise EL conjointement avec **JSTL** (La bibliothèque de balises JSTL fera le sujet du prochain chapitre).

Syntaxe générale des expressions EL :

```
${ expression }
```

Afficher une valeur

Valeur (nombre, chaine, booléen, ou résultat d'une opération de calcul) définie dans le code de la page :

```
${ valeur } // les chaines de caractères sont mises entre simples cotes.
```

Valeur définie dans une variable :

```
${ nom_variable }
```

Evaluer une valeur

- Opérateurs arithmétiques :
  + - / * % (ou **mod**)
- Opérations logiques :
  - **! && ||**
  - **==** ou **eq**, **!=** ou **ne**, **<** ou **lt**, **>** ou **gt**, **<=** ou **le**, **>=** ou **ge**
  - **equals( )** et **compareTo( )** (pour les objets qui ont une implémentation pour ces méthodes)
- L'opérateur ternaire :
  test_logique ? valeur1 : valeur2
- Test de vacuité d'une variable :
  Opérateur **empty**

Exemple

```
<div>test 1 : ${22 mod 5}</div>

<div>test 2 : ${(4+5) eq 10}</div>

<div>test 3 : ${!empty ''}</div>

<div>test 4 : ${"ABC" == 'ABC'}</div>

<div>test 5 : ${'A' gt 'B'}</div>

<div>test 6 : ${22 / 0}</div>
```

Possibilité d'utiliser des parenthèses

Il est préférable d'utiliser des simples-cotes plutôt que des doubles-cotes pour les chaines de caractères, car les balises HTML possèdent déjà des doubles-cotes pour leurs attributs.

La valeur sera false car le code ASCII de A n'est pas plus grand que celui de B.

```
test 1 : 2
test 2 : false
test 3 : false
test 4 : true
test 5 : false
test 6 : Infinity
```

Utiliser des objets

- Reconnaitre un bean ou un objet dans la page :
  Si un objet est utilisé à l'intérieur des accolades { }, l'expression EL le reconnaitra automatiquement, et ce quel que soit son scope (page, request, session ou application). Donc si une page jsp souhaite récupérer un attribut contenu dans le request, elle n'a qu'à utiliser le nom de l'attribut dans ses expressions EL, sans aucune déclaration.
- Accéder à un objet :

```
// tester si le bean est vide :
${ empty nom_bean }
// accéder aux attributs :
${ nom_bean.nom_attribut } ou ${ nom_bean.getter_attribut() }
Les deux manières sont correctes même si l'attribut est protégé ou privé.
```

REMARQUE

Le principe orienté objet de « l'introspection » permet à l'expression EL de trouver le getter associé à l'attribut demandé. Il faut donc que l'objet respecte les spécifications des Javabeans citées dans le paragraphe précèdent, car les EL se basent sur celles-ci.

- Accéder à un élément d'une liste d'objets :

```
/* Supposons l'existence d'une liste de beans (Ex. ArrayList) dans le request, la session, …
   Toutes ces syntaxes sont correctes pour accéder par ex. au 1ier élément : */
${ liste.get(0) }
${ liste[0] }
${ liste['0'] }
${ liste["0"] }
S'il s'agit d'une Map (clef/valeur)
${ nom_map['nom_clef'] }
```

Exemple

```
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/r
    <servlet>
        <servlet-name>Control</servlet-name>
        <servlet-class>controls.Control</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Control</servlet-name>
        <url-pattern>/control</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>control</welcome-file>
    </welcome-file-list>
</web-app>
```

```
// Servlet Control.java
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    ArrayList<Produit> listeP = new ArrayList<>();
    Produit p = new Produit(1, "Doliprane 1000mg comprimés", "MEDICAL", "", 14.50);
    request.setAttribute("produit", p);
    listeP.add(p);
    p = new Produit(2, "Doliprane 500mg comprimés effervescents", "MEDICAL", "", 14.50);
    listeP.add(p);
    p = new Produit(3, null, "MEDICAL", "", 14.50);
    listeP.add(p);
    request.getSession().setAttribute("produits", listeP);
    getServletContext().getRequestDispatcher("/WEB-INF/jsp1.jsp").forward(request, response);
}
// Page jsp1.jsp
<div>test 1 : ${produit.designation}</div>

<div>test 2 : ${produits[1].designati

<div>test 3 : ${produits[2].designation}</div>

<div>test 4 : ${empty produits[2].designation ? 'Aucune désignation' :
produits[2].designation}</div>
```

produit : récupéré depuis le **request**
produits : récupéré depuis la **session**

La désignation du 3ieme élément est un objet nul. L'expression n'affiche donc rien.

test 1 : Doliprane 1000mg comprimés
test 2 : Doliprane 500mg comprimés effervescents
test 3 :
test 4 : Aucune designation

COUCHE PRESENTATION - AMELIORER LA VUE AVEC JSTL

1. DEFINITION ET POINTS FORTS DE JSTL

Définition : JSTL

JSTL (JSP STANDARD TAG LIBRARY) est une bibliothèque de balises qui facilitent au développeur la réalisation de pages jsp, en répondant aux besoins communs entre toutes les applications web (tests, boucles, traitement de données, blocs try/catch, fonctionnalités XML, …), sans qu'il soit obligé d'introduire du code Java dans la page jsp.
D'ailleurs l'une des plus importantes recommandations du modèle MVC c'est cette séparation entre Java et la vue, qui ne doit être consacrée qu'à l'affichage.

Les avantages d'utiliser les balises JSTL :
- Ecrire moins, pour réaliser plus
- Réutilisabilité du code
- JSTL se base sur le langage EL
- La maintenance, l'évolution et les tests de la couche vue, sont considérablement plus faciles si elle ne contient que des balises JSTL, non mélangées avec du code Java.

La 1iere version de JSTL est sortie en 2002, la dernière mise à jour date de 2006.

2. LES BIBLIOTHEQUES DE JSTL

Les sous-familles de JSTL

Les balises de JSTL se divisent en plusieurs familles, qui sont définies dans des sous-bibliothèques :
- **Core** : Bibliothèque principale de balises. Contient les balises de tests, boucles, …
- **Format** : Bibliothèque de balises de formatage de données.
- **XML** : Bibliothèque de balises offrant des fonctionnalités autour de XML.
- **SQL** : Bibliothèque de balises d'accès aux données d'un SGBD.
- **Function** : Bibliothèque de fonctions utilisables dans les expressions EL.

Pour utiliser une bibliothèque de JSTL dans une page jsp on doit d'abord l'inclure au début de la page grâce à la directive jsp **<%@ taglib … %>**.
Syntaxe :

```
<%@ taglib uri="……" prefix="…" %>
```

Core	`<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>`
Format	`<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>`
XML	`<%@ taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x" %>`
SQL	`<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>`
Function	`<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>`

3. LA BIBLIOTHEQUE « CORE »

Inclure une page jsp dans une autre

```
<c:import url="……"></c:import>
// équivalent en syntaxe jsp
<%@include file="……" %>
```

- Permet d'importer une ressource, par ex. : une autre page jsp, comme **<jsp:include>**.
- L'avantage par rapport à **<jsp:include>** c'est que la page peut être dans une autre application ou même dans un autre serveur.

Utiliser une variable / un objet

- **Déclaration** : création d'une variable simple / objet dans un scope.

```
<c:set var="nom_variable" value="une_valeur" scope="……"/>
```

  - scope : page, request, …. Le scope par défaut est « page »
  - value : valeur d'initialisation de la variable.
- **Modification** :
  - Variable simple : même syntaxe que pour la création.
  - Propriété (attribut) d'un objet :

```
<c:set target="${nom_objet}" property="nom_attribut" value="une_valeur" />
// l'objet est recherché automatiquement dans tous les scopes, en commençant par le scope « page ». En cas d'ambiguïté on peut spécifier le scope ciblé :
<c:set target="${nom_scope['nom_objet']}" property="nom_attribut" value="une_valeur"/>
Ex.: <c:set target="${session['client']**}" property="prenom" value="……"/>

** Les objets implicites dans les expressions EL :
- pageScope : Map qui contient toutes les variables définies dans le scope page.
- requestScope : Map qui contient toutes les variables définies dans le scope request.
- sessionScope : Map qui contient toutes les variables définies dans le scope session.
- applicationScope : Map qui contient toutes les variables définies dans le scope application.
- param : Map qui contient tous les paramètres de la requête courante.
- cookie : Map qui contient tous les objets cookies créés par l'application.
```

- **Suppression** :

```
<c:remove var="nom_variable" scope="……"/> // comme toujours, le scope par défaut est page.
```

```
<c:out value="valeur ou expression EL" default="……" escapeXml="true ou false"/>
```

- **default** : afficher une valeur par défaut au cas où l'expression est nulle ou vide.
- **escapeXml** : permet l'échappement des caractères HTML spéciaux (< " ' > &), cet attribut est activé par défaut.

- **Structure optionnelle (IF)** :

```
<c:if test="expression EL" var="une_variable" scope="……">
  // valeurs à afficher ou instructions à exécuter
</c:if>
```

La variable déclarée dans l'attribut **var** servira à stocker le résultat du test, dans le scope spécifié. Ce n'est pas obligatoire, mais cela servira à ne pas effectuer le test plusieurs fois, surtout quand le test nécessite beaucoup de ressources mémoire.

- **Structure alternative (IF/ELSE ou SWITCH) :**

```
<c:choose>
  <c:when test="……">
    // valeurs à afficher ou instructions à exécuter
  </c:when>
  <c:when test="……">
    // valeurs à afficher ou instructions à exécuter
  </c:when>
  ……
  <c:otherwise>
    // valeurs à afficher ou instructions à exécuter
  </c:otherwise> // équivalent à ELSE ou DEFAULT pour la structure switch
</c:choose>
```

Les boucles

- **Boucle avec compteur** :

```
<c:forEach var="nom_compteur" begin="…" end="…" step="…">
  // instructions ou code html à répéter
</c:forEach>
```

- **Boucle de parcours d'une liste :**

  Les attributs **begin** et **end** ne servent qu'à limiter l'affichage des éléments de la liste, au lieu d'afficher tous les éléments. Ils ne sont pas obligatoires.

```
<c:forEach var="nom_objet" items="nom_liste" begin="…" end="…">
  // instructions ou code html à répéter
</c:forEach>
```