



Java EE

-

Cours 5

Cours de 2^e année ingénieur
Spécialisation « Génie Informatique »



JavaBeans?

- Les JavaBeans sont des classes Java (POJO) qui suivent certaines conventions
 - Doivent avoir un constructeur vide (zero argument)
 - On peut satisfaire cette contrainte soit en définissant explicitement un tel constructeur, soit en ne spécifiant aucun constructeur
 - Ne doivent pas avoir d'attributs publics
 - Une bonne pratique réutilisable par ailleurs...
 - La valeur des attributs doit être manipulée à travers des méthodes `getXxx` et `setXxx`
 - Si une classe possède une méthode `getTitle` qui retourne une `String`, on dit que le bean possède une propriété `String` nommée `title`
 - Les propriétés `Boolean` utilisent `isXxx` à la place de `getXxx`
 - Référence sur les beans, cf.
<http://java.sun.com/beans/docs/>



Pourquoi faut-il utiliser des accesseurs?

- Dans un bean, on ne peut pas avoir de champs publics
- Donc, il faut remplacer

```
public double speed;
```

par

```
private double speed;
```

```
public double getSpeed() {
```

```
    return(speed);
```

```
}
```

```
public void setSpeed (double newSpeed) {
```

```
    speed=newSpeed;
```

```
}
```

- Pourquoi faut-il faire cela pour *tout* code Java?



Pourquoi faut-il utiliser des accesseurs?

- 1) On peut imposer des contraintes sur les données

```
public void setSpeed(double newSpeed){  
    if(newSpeed < 0){  
        sendMessage(...);  
        newSpeed = Math.abs(newSpeed);  
    }  
    speed = newSpeed;  
}
```



Pourquoi faut-il utiliser des accesseurs?

2) On peut changer de représentation interne sans changer d'interface

```
// Now using miles units
```

```
public void setSpeed(double newSpeed){  
    speedInMPH = convert(newSpeed);  
}
```

```
public void setSpeedInMPH(double newSpeed){  
    speedInMPH = newSpeed;  
}
```



Pourquoi faut-il utiliser des accesseurs?

3) On peut rajouter du code annexe

```
public double setSpeed(double newSpeed){  
    speed = newSpeed;  
    updateSpeedometerDisplay();  
}
```



Utilisation basique des Beans

- `jsp:useBean`
 - Cet élément construit un nouveau bean.
Utilisation:
`<jsp:useBean id="beanName" class="package.Class" />`
- `jsp:setProperty`
 - Cet élément modifie une propriété d'un bean (i.e., appel d'une méthode `setXxx`). Utilisation:
`<jsp:setProperty name="beanName"
 property="propertyName"
 value="propertyValue" />`
- `jsp:getProperty`
 - Cet élément lit et retourne la valeur d'une propriété d'un bean.
Utilisation:
`<jsp:getProperty name="beanName"
 property="propertyName" />`



Approche générale avec des JSP autonomes et les tags `jsp:useBean`

- L'utilisateur soumet un formulaire vers une JSP
 - `<FORM ACTION="SomePage.jsp">`
- La page JSP instancie un bean
 - `<jsp:useBean id="myBean" class="..." />`
- On passe des données de la requête au bean
 - `<jsp:setProperty name="myBean,,
property="customerID,,
value="..." />`
- On affiche des valeurs issues des données de la requête
 - `<jsp:getProperty name="myBean"
property="bankAccountBalance" />`



Création de Beans : jsp:useBean

- Format
 - `<jsp:useBean id="name" class="package.Class" />`
- But
 - Permettre l'instanciation de classes Java sans programmation Java explicite (syntaxe compatible XML)
- Remarques
 - Interpretation simple:
`<jsp:useBean id="book1" class="coreservlets.Book" />`
peut être vu comme équivalent au scriptlet:
`<% coreservlets.Book book1= new coreservlets.Book(); %>`
 - Mais *jsp:useBean* a deux avantages supplémentaires:
 - Facilite l'utilisation des paramètres de requête
 - Facilite le partage des objets entre pages ou servlets



Modification de propriétés de Bean: jsp:setProperty

- Format
 - `<jsp:setProperty name="name"
property="property"
value="value" />`
- But
 - Permettre de modifier les propriétés d'un bean properties (i.e., appel de méthodes setXxx) sans programmation Java explicite
- Remarques
 - `<jsp:setProperty name="book1"
property="title"
value="Core Servlets and JavaServer Pages" />`
est équivalent au scriptlet:
`<% book1.setTitle("Core Servlets and JavaServer Pages"); %>`



Accès aux propriétés de Bean: jsp:getProperty

- Format
 - `<jsp:getProperty name="name" property="property" />`
- But
 - Permettre d'accéder propriétés d'un bean (i.e., appel de méthodes `getXxx`) sans programmation Java explicite
- Remarques
 - `<jsp:getProperty name="book1" property="title" />`
est équivalent à l'expression JSP:
`<%= book1.getTitle() %>`



Exemple: StringBean

```
package coreservlets;
public class StringBean{
    private String message="No message specified";

    public String getMessage () {
        return (message) ;
    }
    public void setMessage (String message) {
        this.message=message;
    }
}
```

- Les Beans doivent être déployés dans le même répertoire que les autres classes Java
 - WEB-INF/classes/*folderMatchingPackage*
- Les Beans doivent ***toujours*** être dans des packages!



StringBean dans une JSP

```
<jsp:useBean id="stringBean"
             class="coreservlets.StringBean" />
<OL>
<LI>Initial value (from jsp:getProperty):
    <I><jsp:getProperty name="stringBean"
                      property="message" /></I>
<LI>Initial value (from JSP expression):
    <I><%= stringBean.getMessage() %></I>
<LI><jsp:setProperty name="stringBean"
                    property="message"
                    value="Best string bean: Fortex" />
    Value after setting property with jsp:setProperty:
    <I><jsp:getProperty name="stringBean"
                      property="message" /></I>
<LI><%= stringBean.setMessage
      ("My favorite: Kentucky Wonder"); %>
    Value after setting property with scriptlet:
    <I><%= stringBean.getMessage() %></I>
</OL>
```

StringBean dans une JSP (Résultat)



The screenshot shows a Netscape browser window with the title "Using JavaBeans with JSP - Netscape". The address bar contains the URL "http://localhost/beans/StringBean.jsp". The main content area displays the following text:

Using JavaBeans with JSP

1. Initial value (from `jsp:getProperty`): *No message specified*
2. Initial value (from JSP expression): *No message specified*
3. Value after setting property with `jsp:setProperty`: *Best string bean: Fortex*
4. Value after setting property with scriptlet: *My favorite: Kentucky Wonder*

The status bar at the bottom indicates "Document: Done (0.06 secs)".



jsp:setProperty (cas1): conversion explicite & affectation

```
<!DOCTYPE ...>
```

```
...
```

```
<jsp:useBean id="entry"  
             class="coreservlets.SaleEntry" />
```

```
<%-- setItemID expects a String --%>
```

```
<jsp:setProperty  
    name="entry"  
    property="itemID"  
    value='<%= request.getParameter("itemID") %>' />
```



jsp:setProperty (cas1): conversion explicite & affectation

```
<%  
int numItemsOrdered=1;  
try{  
    numItemsOrdered=  
        Integer.parseInt(request.getParameter("numItems"));  
} catch (NumberFormatException nfe) {}  
%>
```

```
<%-- setNumItems expects an int --%>
```

```
<jsp:setProperty  
    name="entry"  
    property="numItems"  
    value="<%= numItemsOrdered %>" />
```

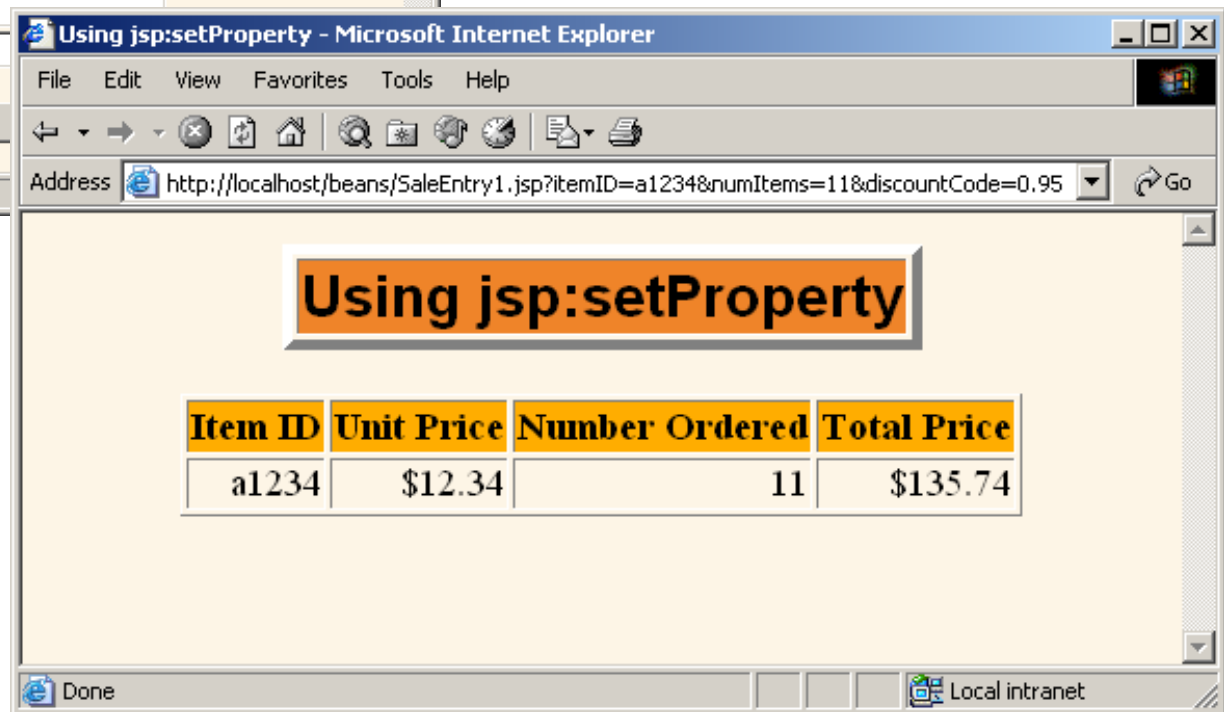
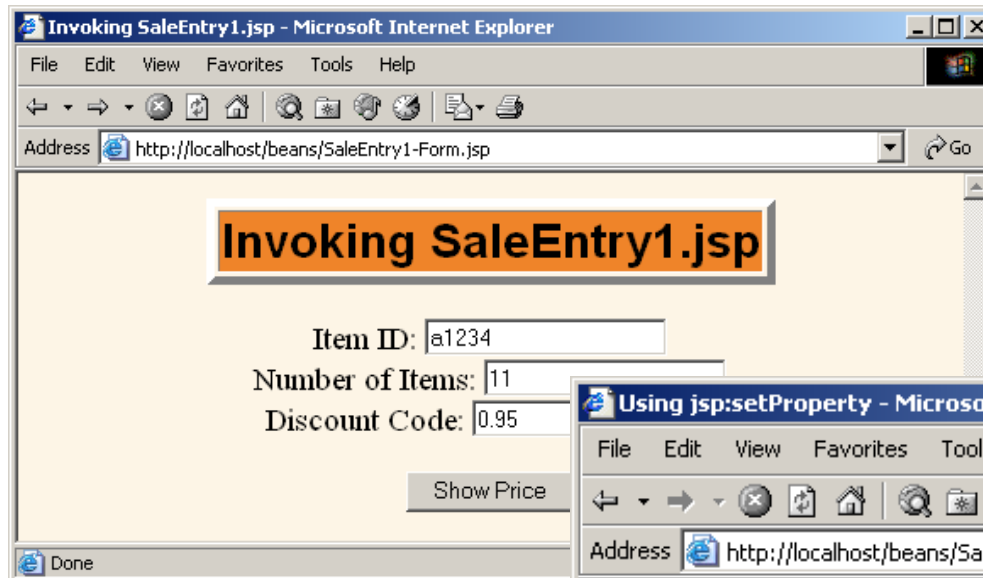



jsp:setProperty (cas1): conversionexplicite & affectation

```
<%  
double discountCode = 1.0;  
try{  
    String discountString=  
        request.getParameter("discountCode");  
    discountCode=  
        Double.parseDouble(discountString);  
} catch (NumberFormatException nfe) {}  
%>  
<%-- setDiscountCode expects a double --%>  
<jsp:setProperty  
    name="entry"  
    property="discountCode"  
    value="<%= discountCode%>" />
```



jsp:setProperty (cas1): conversion explicite & affectation





jsp:setProperty (cas2) : association propriété-paramètre

- Utilisation de l'attribut *param* de `jsp:setProperty`
 - On indique que la valeur doit provenir du paramètre de requête spécifié
 - Une conversion automatique de type est opérée pour les valeurs de types standard
 - boolean, Boolean, byte, Byte, char, Character, double, Double, int, Integer, float, Float, long, ou Long.



jsp:setProperty (cas2) : association propriété-paramètre

```
<jsp:useBean id="entry"  
            class="coreservlets.SaleEntry" />  
<jsp:setProperty  
    name="entry"  
    property="itemID"  
    param="itemID" />  
<jsp:setProperty  
    name="entry"  
    property="numItems"  
    param="numItems" />  
<jsp:setProperty  
    name="entry"  
    property="discountCode"  
    param="discountCode" />
```



jsp:setProperty (cas3) : associer toutes les propriétés aux paramètres

- Utilisation de "*" pour la valeur de l'attribut *property* de jsp:setProperty
 - On indique que la valeur doit provenir des paramètres de requête qui correspondent aux noms des propriétés
 - Une conversion automatique de type est également opérée



jsp:setProperty (cas3) : associer toutes les propriétés aux paramètres

```
<jsp:useBean id="entry"  
            class="coreservlets.SaleEntry" />  
<jsp:setProperty name="entry" property="*" />
```

- Particulièrement pratique pour réaliser des « Beans formulaires » -- objets dont les propriétés sont remplies par les données d'un formulaire
 - Ce processus peut même être décomposé entre différents formulaires, chaque soumission remplissant une partie de l'objet



Implémentation du MVC avec RequestDispatcher

1. Définir les beans pour représenter les données
2. Utiliser un servlet pour gérer les requêtes
 - Le servlet lit les paramètres de requêtes, vérifie les données manquantes ou malformées, appelle le code métier, etc.
3. Créer les beans
 - Le servlet invoque le code métier (spécifique à l'application) ou accède à une base de données pour obtenir les résultats. Les résultats sont dans les beans définis à l'étape 1
4. Placer le bean dans la requête, la session, ou le servlet context
 - Le servlet appelle `setAttribute` sur la requête, la session, ou le servlet context pour garder une référence sur les beans qui représentent le résultat de la requête



Implémentation du MVC avec RequestDispatcher

5. Transmettre la requête à la JSP (forward)
 - Le servlet détermine quelle JSP est appropriée à la situation et utilise la méthode *forward* du *RequestDispatcher* pour transférer le contrôle à la JSP
6. Extraire les données des beans
 - JSP 1.2: la JSP accède aux beans avec `jsp:useBean` et un scope correspondant au choix de l'étape 4. La JSP utilise ensuite `jsp:getProperty` pour afficher les propriétés des beans
 - JSP 2.0: la JSP utilise `#{nameFromServlet.property}` pour afficher les propriétés des beans
 - La JSP ne crée pas ou ne modifie pas les beans: c'est la vue du MVC!



Request Forwarding Example

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException{
    ... // Do business logic and get data
    String operation= request.getParameter("operation");
    if (operation == null){
        Operation = "unknown";
    }
    String address;
    if (operation.equals("order")){
        address="/WEB-INF/Order.jsp";
    } else if (operation.equals("cancel")){
        address="/WEB-INF/Cancel.jsp";
    }else{
        address="/WEB-INF/UnknownOperation.jsp";
    }
    RequestDispatcherdispatcher=
        request.getRequestDispatcher(address);
    dispatcher.forward(request, response);
}
```



Utilisation de jsp:useBean: MVC vs JSP autonomes

- La JSP ne doit pas créer les objets
 - Le servlet, pas la JSP, doit créer tous les objets-données. Donc, Pour garantir qu'une JSP ne créera pas d'objets, il faut utiliser :
`<jsp:useBean ... type="package.Class" />`
au lieu de
`<jsp:useBean ... class="package.Class" />`
- La JSP ne doit pas modifier les objets
 - Il faut donc utiliser `jsp:getProperty` mais pas `jsp:setProperty`.

Scopes:
requête, session, et
application



Différents scopes pour jsp:useBean

- request
 - `<jsp:useBean id="..." type="..." scope="request" />`
- session
 - `<jsp:useBean id="..." type="..." scope="session" />`
- application
 - `<jsp:useBean id="..." type="..." scope="application" />`
- page
 - `<jsp:useBean id="..." type="..." scope="page" />`
ou juste
`<jsp:useBean id="..." type="..." />`
 - Ce scope n'est pas utilisé dans MVC



Partage de données sur requête

- Servlet

```
ValueObject value = new ValueObject(...);  
request.setAttribute("key", value);  
RequestDispatcher dispatcher =  
    request.getRequestDispatcher("/WEB-INF/SomePage.jsp");  
dispatcher.forward(request, response);
```

- JSP 1.2

```
<jsp:useBean id="key" type="somePackage.ValueObject"  
            scope="request" />  
<jsp:getProperty name="key" property="someProperty" />
```

- JSP 2.0

```
${key.someProperty}
```



Partage de données sur requête: exemple simple

- **Servlet**

```
Customer myCustomer=  
    new Customer(request.getParameter("customerID"));  
request.setAttribute("customer", myCustomer);  
RequestDispatcher dispatcher=  
    request.getRequestDispatcher("/WEB-INF/SomePage.jsp");  
dispatcher.forward(request, response);
```

- **JSP 1.2**

```
<jsp:useBean id="customer" type="somePackage.Customer"  
            scope="request" />  
<jsp:getProperty name="customer" property="firstName"/>
```

- **JSP 2.0**

```
${customer.firstName}
```



Partage de données sur session

- Servlet

```
ValueObject value=new ValueObject(...);  
HttpSession session=request.getSession();  
session.setAttribute("key", value);  
RequestDispatcher dispatcher=  
    request.getRequestDispatcher("/WEB-INF/SomePage.jsp");  
dispatcher.forward(request, response);
```

- JSP 1.2

```
<jsp:useBean id="key" type="somePackage.ValueObject"  
            scope="session" />  
<jsp:getProperty name="key" property="someProperty" />
```

- JSP 2.0

```
${key.someProperty}
```



Partage de données sur session: variation

- Redirection vers une page au lieu d'un transfert
 - Utiliser `response.sendRedirect` à la place de `RequestDispatcher.forward`
- Différences avec `sendRedirect`:
 - L'utilisateur voit l'URL de la JSP (l'utilisateur ne voit que l'URL du servlet avec `RequestDispatcher.forward`)
 - Deux aller-retour pour le client (au lieu d'un avec `forward`)
- Avantage du `sendRedirect`
 - L'utilisateur peut accéder à la JSP séparément
 - Possibilité de mettre la JSP en marque-page
- Désavantages du `sendRedirect`
 - Deux aller-retour, c'est plus coûteux
 - Puisque l'utilisateur peut accéder à la JSP sans passer par le servlet d'abord, les beans qui contiennent les données peuvent ne pas être disponibles
 - Il faut du code en plus pour détecter cette situation



Partage de données sur application (Rare)

- Servlet

```
synchronized(this) {  
    ValueObjectvalue= new ValueObject(...);  
    getServletContext().setAttribute("key", value);  
    RequestDispatcher dispatcher=  
        request.getRequestDispatcher("/WEB-INF/SomePage.jsp");  
    dispatcher.forward(request, response);  
}
```

- JSP 1.2

```
<jsp:useBean id="key" type="somePackage.ValueObject"  
            scope="application" />  
<jsp:getProperty name="key" property="someProperty" />
```

- JSP 2.0

```
`${key}.someProperty}
```



URL relatives dans les JSP

- Problème:
 - Transférer avec un *request dispatcher* est transparent pour le client. L'URL *originale* est la seule URL que le navigateur connaît.
- Pourquoi est-ce important?
 - Que fera un navigateur avec les balises suivantes?

```
  
<link rel="stylesheet"  
      href="my-styles.css"  
      type="text/css">  
<a href="bar.jsp">...</a>
```

- Le navigateur traite les adresses relatives comme des adresses relatives à l'URL du *servlet*