

T.P.

Programmation par composants avec les frameworks JSF et Seam

Thierry CHATEL, société **IOCEAN**

Objectif : appréhender la programmation par composants proposée par les frameworks JSF et Seam.

Contexte du TP

L'exercice proposé est la réalisation d'une boutique de vente en ligne.

Il s'agit d'un exercice volontairement très simplifié qui ne constitue évidemment pas un exemple réaliste de développement d'un site commercial, mais qui permet d'utiliser quelques uns des aspects les plus marquants des frameworks JSF et Seam dans un contexte qui vous est certainement familier.

En particulier, tout l'affichage se fera dans une même page web, dans un but de simplification.

Lexique

Composant JSF. Composant d'affichage, placé dans la page web (fichier Facelets), pour afficher un élément de la page : texte, bouton, table, lien, etc. Il pourra être lié à un objet Java, pour afficher ou permettre de saisir la valeur d'une propriété, dans le cas d'un texte ou d'un champ de saisie, ou pour appeler une méthode de l'objet, dans le cas d'un bouton ou d'un lien.

Un composant JSF s'écrit dans une page Facelets sous la forme `<h:inputText>`, où le préfixe avant le signe ':' est le *namespace* de la bibliothèque de composants à laquelle il appartient. Deux bibliothèques de composants, préfixées par h: et f: sont fournies en standard avec JSF, d'autres composants sont apportés par le framework Seam (ceux préfixés par s:) ou la technologie Facelets.

Langage JSF EL, ou JSF Expression Langage. C'est le langage d'expression utilisé dans les vues JSF (fichiers Facelets). Il est similaire au langage d'expression des JSP. Les expressions sont délimitées par `#{...}`

Vue Facelets. La technologie Facelets est une alternative aux JSP pour écrire les pages web d'une application JSF. Les pages Facelets s'écrivent en XHTML, dans des fichiers avec une extension .xhtml, contenant une hiérarchie de balises XHTML et de composants JSF.

Composant Seam. Classe Java dont l'instance est repérée par un nom de composant, et publiée par le framework Seam dans un certain contexte, lié à l'application, à la session utilisateur, à la page courante, etc. Le framework Seam gère l'injection de dépendances entre les composants, en alimentant lors de chaque appel d'une méthode d'un composant toutes ses variables d'instance faisant référence à d'autres composants.

Mise en place des outils et du projet

A) Copier les répertoires de Tomcat 6 et du projet dans votre répertoire d'accueil :

```
# cd
# cp -r ~cdony/tomcat6 .      (attention, le '.' est important !)
# cp -r ~cdony/CoursJSF .    (idem)
```

B) Lancer Eclipse, en utilisant la commande eclipse-jee si elle est définie sur votre compte, ou en le lançant depuis le menu :

```
# eclipse-jee
```

Fermer la vue « **Welcome** ».

C) Importer le projet dans Eclipse : menu **File / Import**, choisir **General / Existing Projects into Workspace**, cliquer sur le bouton « **Next** », et choisir le sous-répertoire **CoursJSF** de votre répertoire d'accueil, puis cliquer sur le bouton « **Finish** ».

D) Toujours dans Eclipse, dans la vue « **Servers** » (en bas de l'écran) :

- cliquer avec le bouton droit et choisir **New / Server**
- choisir le type **Apache / Tomcat v6.0 Server**
- cliquer sur le bouton « **Next** »
- indiquer comme répertoire d'installation le répertoire **tomcat6** de votre répertoire d'accueil, et choisir comme **JRE** dans la liste déroulante un **JDK 1.5** ou **1.6** (ou à défaut un **JRE 1.5** ou **1.6**)
- cliquer encore sur le bouton « **Next** »
- ajouter le projet **CoursJSF** en le sélectionnant et en cliquant sur le bouton « **Add** »
- cliquer sur le bouton « **Finish** ».

E) Régler le timeout de démarrage de Tomcat : dans la vue « **Servers** », clic droit sur la ligne du serveur « **Tomcat 6** », et « **Open** ». Puis dans la partie droite de la vue des propriétés du serveur Tomcat qui vient de s'ouvrir, déplier la rubrique « **Timeouts** » et mettre le timeout de démarrage (« **Start** ») à **200**. Puis enregistrer (menu **File / Save** ou **Control+S**).

F) Lancer le projet sur Tomcat depuis Eclipse : dans la vue « **Servers** », clic droit sur la ligne du serveur « **Tomcat 6** », et « **Start** ». Ou utiliser l'icône de la vue, avec un triangle blanc sur un rond vert.

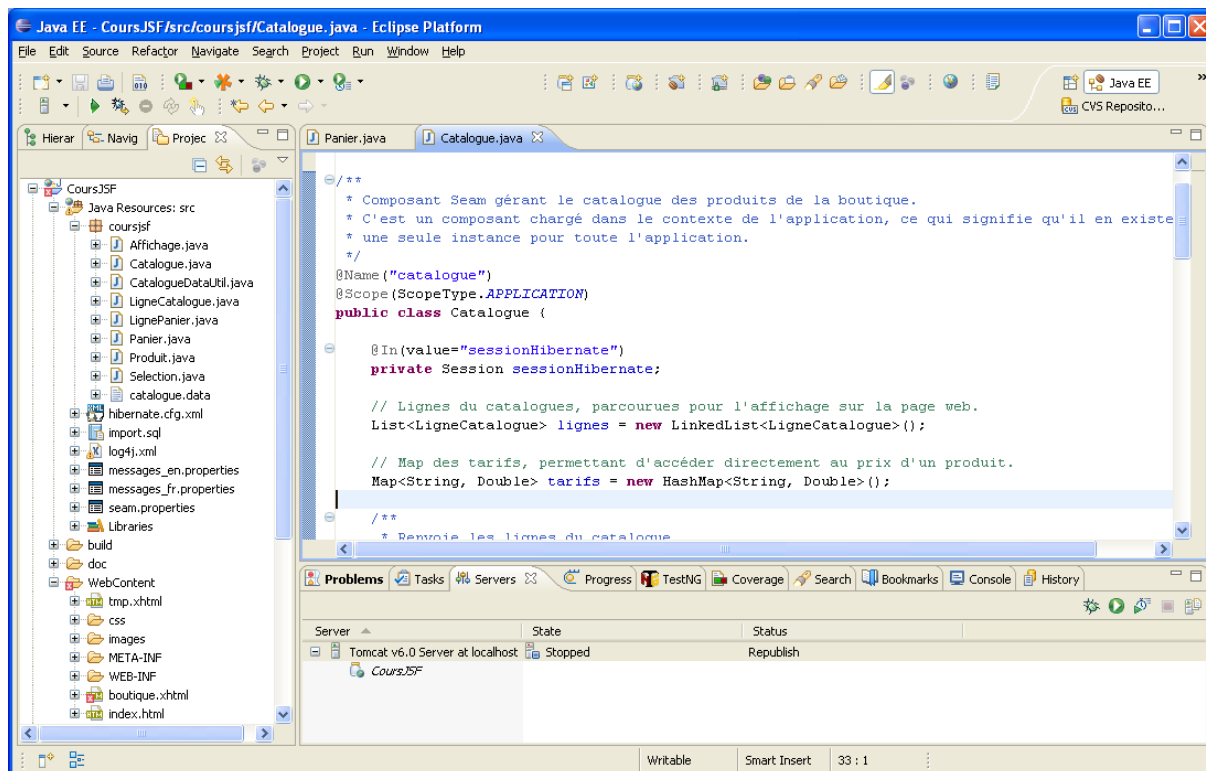
G) Ouvrir l'adresse suivant dans un navigateur Web : <http://localhost:8080/coursjsf/>
Ca doit rediriger vers l'adresse <http://localhost:8080/coursjsf/boutique.seam> et afficher une page blanche avec juste le titre « *Boutique* ».

Les différents répertoires du projet sont décrits en annexe, à la fin de ce document.

Utilisation d'Eclipse et lancement du serveur

Lancer **Eclipse** (version **Eclipse-JEE**).

Les fichiers du projet sont visibles dans la vue « Project Explorer » d'Eclipse, qui constitue la partie gauche de la copie d'écran ci-dessous.



Sous la zone de l'éditeur affichant les fichiers Java et XHTML ouverts, la vue « Servers » permet de démarrer ou d'arrêter le serveur, en utilisant le menu contextuel ou les icônes en haut à droite de la vue.

Lorsque le serveur est démarré, vous pouvez accéder à la page web du TP en ouvrant un navigateur web et en tapant l'adresse suivante (qui redirigera automatiquement vers l'adresse réelle) :

<http://localhost:8080/coursjsf/>

Redémarrage du serveur :

Les modifications des pages web (fichiers .xhtml) ne nécessitent pas un redémarrage du serveur, elles seront prises en compte automatiquement. Ça peut parfois prendre 2 ou 3 secondes le temps qu'Eclipse republie le nouveau fichier.

Par contre **les modifications dans les classes Java imposent un redémarrage**. Par défaut, en cas de modification d'une classe Java, le serveur essaie de recharger automatiquement tout le contexte de l'application web. Ça marche plus ou moins, il est parfois nécessaire ou préférable de redémarrer le serveur manuellement.

TP 1 : Affichage du catalogue des produits

Liste des produits

Le catalogue des produits est géré par la classe Java *Catalogue*, qui est un composant Seam accessible sous le nom « *catalogue* » (sans majuscule). Il s'agit d'un JavaBean avec une propriété *lignes*, renvoyée par la méthode *getLignes()*, qui contient une liste d'objets de la classe *LigneCatalogue*, regroupant un produit et son tarif.

Affichage de la liste

C'est le fichier *WebContent/boutique.xhtml* qui constitue la page web de la boutique en ligne. Il s'agit d'une page Facelets, en XHTML, incluant des composants JSF tels que `<h:form>`. Pour l'instant la page n'affiche qu'un titre. Pour la voir, il faut lancer le serveur depuis la vue *Servers* dans Eclipse, et afficher dans le navigateur web l'adresse suivante :

```
http://localhost:8080/coursjsf/boutique.seam
```

A vous d'afficher le catalogue des produits dans cette page, sous la forme d'une table présentant les lignes du composant Seam *catalogue*. Affichez le libellé de chaque produit et son tarif.

Vous utiliserez pour cela le composant JSF `<h:dataTable>` dont la syntaxe est décrite en annexe. Attention, le code dans le fichier *boutique.xhtml*, qu'il s'agisse de balises HTML ou de balises indiquant des composants JSF, doit respecter une syntaxe XML stricte, et notamment toutes les balises doivent être fermées (ne pas écrire `
` mais `
` ou `
</br>`) et correctement imbriquées.

L'accès au composant Seam *catalogue* se fait dans une expression JSF EL (JSF Expression Language) simplement en indiquant son nom, le framework Seam se chargeant automatiquement de retrouver le composant, ou de l'instancier s'il n'existe pas encore :

```
{catalogue}
```

L'accès à n'importe quelle propriété d'un objet Java, au sens de la norme JavaBean c'est-à-dire une valeur renvoyée par un *getter** normalisé, se fait en JSF EL en spécifiant l'objet suivi d'un point et du nom de la propriété :

```
{catalogue.lignes}
```

***Getter normalisé** (norme JavaBean) :

Pour une propriété « **prop** » (sans majuscule), le *getter* doit être une méthode **getProp()**, ou **isProp()** si la valeur renvoyée est un booléen.

Formatage des prix

Une fois la table affichée, vous pouvez constater que par défaut les prix, insérés par un simple composant :

```
<h:outputText value="{ligneCatalogue.prix}"/>
```

ne sont pas correctement formatés avec deux décimales. Pour les afficher de façon adéquate, il est possible de spécifier un format au composant `<h:outputText>`, en lui ajoutant un composant enfant de

type « *convert* ». JSF contient en standard un *convert* destiné au formatage des nombres (et un autre pour les dates) :

```
<h:outputText value="#{ligneCatalogue.prix}">
  <f:convertNumber pattern="###0.00 €"/>
</h:outputText>
```

Détail d'un produit

Vous allez maintenant créer votre premier composant Seam, destiné à gérer le produit sélectionné par le client.

Création d'un composant Seam gérant le produit sélectionné

Créez pour cela une classe *Selection*. Vous pouvez la mettre dans le même package que les autres classes Java du TP.

N'importe quelle classe Java peut devenir un composant Seam, à condition qu'elle ait un constructeur public **sans argument**, qui peut être le constructeur implicite créé par Java quand il n'y a aucun constructeur explicite dans la classe.

Pour faire d'une classe Java un composant Seam, il suffit d'utiliser l'annotation suivante, où *selection* est le nom du composant (sans majuscule par convention) :

```
@Name("selection")
```

L'annotation s'applique à la classe Java elle-même, donc il faut la placer juste au-dessus de la définition de la classe (vous en avez un exemple dans la classe *Catalogue*).

Il faut aussi spécifier le contexte (« scope » en anglais) du composant, pour savoir s'il s'agit d'un composant devant être partagé pour toute l'application, ou pour la session d'un utilisateur, ou alors si le composant est propre à la page courante. Dans votre classe *Selection*, indiquez qu'il s'agit d'un composant de contexte « Page », en utilisant l'annotation suivante, à placer juste en dessous de l'annotation `@Name`, toujours avant la définition de la classe :

```
@Scope(ScopeType.PAGE)
```

Votre classe Java définit maintenant un composant *selection* contextuel à la page web courante.

Ajoutez à la classe une variable d'instance *produit* de type *Produit*, qui contiendra le produit sélectionné par le client, et une méthode *getProduit()* renvoyant la valeur de cette variable.

Afin de pouvoir tester l'affichage avant d'avoir mis en place la sélection d'un produit, vous allez initialiser cette variable avec un produit au hasard provenant du catalogue. La classe *Catalogue* contient une méthode *getProduitAuHasard()* qui comme son nom l'indique renvoie justement un produit au hasard. Pour accéder au composant *catalogue* depuis le composant *selection*, vous allez l'injecter comme une variable d'instance.

Définissez une variable *catalogue* de type *Catalogue*, et annotez-la en faisant précéder sa définition d'une annotation `@In` :

```
@In(create=true)
private Catalogue catalogue;
```

L'annotation **@In** va indiquer au framework Seam qu'il doit injecter le composant correspondant dans la variable, avant tout appel de méthode de la classe **Selection**. Par défaut c'est le nom de la variable qui sert à trouver le composant à injecter, s'il porte un nom différent il faut le spécifier comme paramètre de l'annotation **@In**. L'option `create=true` indique au framework qu'il doit créer le composant si celui-ci n'existe pas encore ; sans cette option, si le composant à injecter n'existe pas, le framework signale une erreur.

Reste à utiliser le composant **catalogue** pour initialiser la variable **produit**. En programmation objet, ce genre d'initialisation se fait généralement dans le constructeur de l'objet. Seulement l'injection des dépendances par le framework Seam est effective pour toutes les méthodes de la classe sauf pour les constructeurs. La solution est de définir une méthode d'initialisation du composant **selection**, qui sera appelée systématiquement par le framework Seam lors de l'instanciation du composant. Il faut pour cela définir une méthode sans aucun paramètre (son nom n'a aucune importance), et l'annoter avec l'annotation **@Create** :

```
@Create
public void init() {
    ...
}
```

Utilisez dans cette méthode le composant injecté **catalogue** pour initialiser la variable **produit** avec un produit au hasard.

Affichage du produit sélectionné

Vous allez maintenant afficher sur la page web les détails du produit sélectionné, fournis par les méthodes de la classe **Produit**.

Attention, la description du produit est au format HTML. Par défaut le composant `<h:outputText>` convertit les caractères du contenu à afficher qui auraient une signification spéciale en HTML, pour qu'ils n'entrent pas en conflit avec la structure de la page. Dans le cas où le contenu est déjà en HTML, il faut désactiver cette conversion automatique, en ajoutant l'attribut **escape="false"** :

```
<h:outputText value="#{...}" escape="false" />
```

Démarrez ensuite le serveur, et vérifiez qu'en arrivant sur la page un produit au hasard est affiché. Le contexte « PAGE » dans lequel a été placé le composant Seam **selection** n'est conservé que lors de la soumission d'un formulaire sans redirection vers une autre page, donc si vous rechargez la page dans le navigateur web le contexte n'est pas conservé, un nouveau composant est instancié, et vous devez voir un autre produit choisi aléatoirement.

Sélection d'un produit dans la liste

Pour l'instant le composant **selection** ne sélectionne rien mais affiche simplement un produit au hasard. Ajoutez-lui une méthode de sélection d'un produit, qui prend un objet de la classe **Produit** en paramètre.

Affichez dans la page web, sur chaque ligne du catalogue, un bouton « Détails » pour sélectionner le produit pour lequel le client veut voir les informations détaillées. Utilisez pour cela le composant JSF `<h:commandButton>`, avec un attribut **action** appelant votre méthode de sélection d'un produit. L'expression JSF EL à mettre dans l'attribut **action** est l'appel de la méthode du composant **selection** avec en paramètre l'objet produit de la ligne de catalogue concernée. La syntaxe est la même que pour un appel de méthode en Java :

```
#{composant.methode(parametre)}
```

Cette syntaxe d'appel de méthode avec un ou plusieurs paramètres est en fait une extension du langage d'expression JSF EL apportée par le framework Seam. Le langage JSF EL standard permet de spécifier la méthode à appeler, sans parenthèses, mais ne permet pas de passer des paramètres.

Vérifiez maintenant qu'en cliquant sur le bouton « Détails » le client peut choisir le produit sur lequel il souhaite voir plus d'informations.

Photo du produit

Des images sont fournies, dans le répertoire *WebContent/images*, pour illustrer les produits du catalogue.

A chaque produit est associé une petite image *logo_CODE.jpg* (où CODE est le code du produit) pour afficher dans la liste des produits du catalogue, et une grande image *photo_CODE.jpg* pour afficher dans le descriptif du produit.

Pour afficher une image vous pouvez utiliser simplement la balise HTML :

```

```

Naturellement il est aussi possible d'utiliser le composant JSF `<h:graphicImage>` qui va générer la même chose :

```
<h:graphicImage value="chemin_du_fichier_image"/>
```

Il ne faut pas indiquer le répertoire *WebContent* dans le chemin du fichier, juste le répertoire *images*, par exemple : *images/logo_PR.jpg*.

Pour construire dynamiquement le chemin de l'image d'un produit donné, remplacez simplement la partie correspondant au code du produit par une expression JSF EL renvoyant le code du produit.

Affichez ainsi pour chaque produit la petite image dans le catalogue, et la grande dans le descriptif. Pour la grande image, vous pouvez ajouter à la balise `` un attribut *align="right"*, qui fait que l'image va être positionnée sur la droite, insérée dans le texte.

Pour que le client puisse sélectionner un produit en cliquant sur l'image dans le catalogue, faites-en un lien ayant le même effet que le bouton « Détails ». Utilisez pour cela le composant JSF

```
<h:commandLink> avec un attribut action identique à celui du bouton « Détails » (composant <h:commandButton>).
```

TP 2 : Gestion du panier

Vous allez maintenant implémenter la gestion du panier des produits qui seront commandés. Le client pourra ainsi ajouter au panier les produits consultés, en modifier la quantité, voir le montant total, enlever du panier certains produits ou le vider complètement. La phase de commande proprement dite ne sera pas abordée dans ce TP.

Affichage du panier

Composant Seam de gestion du panier

Créez maintenant une classe *Panier*, et faites-en un composant Seam nommé *panier*. Le panier étant conservé tout au long de la session du client sur la boutique en ligne, le composant devra être placé dans le contexte de session (`ScopeType.SESSION`). Ajoutez aussi à la classe *Panier* l'annotation `@Autocreate`, pour indiquer qu'il faut créer automatiquement le composant s'il n'existe pas :

```
@Autocreate
```

Cette annotation `@Autocreate` est généralement superflue, car le framework Seam crée automatiquement un composant quand il est accédé dans une expression JSF EL (JSF Expression Language) renvoyant une valeur. Mais ce n'est pas le cas lors de l'appel d'une méthode d'un composant Seam dans une expression JSF EL désignant une méthode Java. Dans le cas particulier de ce composant panier, si vous appelez une méthode pour ajouter un produit au panier alors qu'aucune expression de la page n'accède auparavant en lecture à ce composant panier, il n'aura pas été créé automatiquement, et dans ce cas l'annotation `@Autocreate` est indispensable.

Le panier contiendra une liste d'objets de la classe fournie *LignePanier*, regroupant un produit et une quantité.

Le composant aura une propriété JavaBean *lignes* correspondant aux lignes contenues dans le panier. Créez pour cela un *getter* normalisé renvoyant la valeur de cette propriété. Créez aussi un *getter* pour une propriété correspondant au montant total des lignes du panier.

Affichage du panier dans la page web

Dans la page web, affichez maintenant le panier sous le descriptif du produit courant.

Comme pour la liste des produits du catalogue, utilisez le composant JSF `<h:dataTable>` pour parcourir les lignes du panier, et afficher pour chacune le nom du produit, la quantité, le prix unitaire et le prix pour le nombre de produits demandés.

Affichez également le montant total des produits dans le panier, calculé en tenant compte des quantités demandées.

Message si le panier est vide

Pour l'instant le panier est vide. Dans ce cas il vaudrait mieux afficher un message explicite « *Votre panier est vide* », plutôt qu'une table avec juste les entêtes et un montant total à zéro.

Il est possible de conditionner l'affichage d'un composant JSF en plaçant une condition, écrite avec le langage d'expression JSF EL, dans l'attribut *rendered*. Le composant sera affiché dans la page si l'expression booléenne est évaluée à vrai, et sera invisible dans le cas contraire. La condition peut être

écrite très facilement avec l'opérateur *empty* de JSF EL, évalué à vrai si la liste qui suit est vide. Ainsi l'expression suivante sera vraie si la propriété *lignes* du composant *panier* est une liste vide :

```
{empty panier.lignes}
```

L'opérateur *not* permet d'obtenir la condition inverse :

```
{not empty panier.lignes}
```

Vous pouvez mettre l'une ou l'autre de ces conditions dans l'attribut *rendered* du composant `<s:fragment>` qui n'a aucun effet visible mais sert juste à regrouper tout son contenu (placé entre `<s:fragment>` et `</s:fragment>`) justement dans le but d'en conditionner l'affichage.

Utilisez donc un fragment qui sera visible s'il y a au moins un produit dans le panier, et un fragment qui sera visible si le panier est vide, en les conditionnant de façon adéquate :

```
<s:fragment rendered="{...}">
    ...contenu dont l'affichage est conditionné...
</s:fragment>
```

Puisque le panier est toujours vide, vous devez donc maintenant voir le message spécifique prévu dans ce cas, et non plus la table vide avec le montant total à zéro.

Ajout de produits dans le panier

Il est temps de remplir le panier.

Méthode d'ajout au panier

Créez pour cela une méthode d'ajout d'un produit au panier, qui prend un objet de la classe *Produit* en paramètre. Vous pouvez commencer par ajouter systématiquement une nouvelle ligne dans le panier, sans tenir compte du fait que le produit peut y être déjà présent.

Vous avez pu remarquer que le constructeur de la classe *LignePanier* prend en paramètre, en plus du produit, le composant *catalogue*, qui est utilisé pour récupérer le prix unitaire du produit.

Commencez par injecter le composant *catalogue* dans le composant *panier*, en ajoutant à la classe *Panier* une variable d'instance de type *Catalogue*, avec l'annotation *@In*. Attention à bien nommer la variable *catalogue*, c'est-à-dire à lui donner le nom du composant à injecter, sinon il vous faut préciser le nom du composant en paramètre de l'annotation *@In*. Vous pouvez alors passer la valeur de cette variable, qui aura été injectée par le framework Seam, au constructeur de la classe *LignePanier*.

Boutons d'ajout dans la page web.

Après avoir écrit la méthode d'ajout d'un produit au panier, modifiez la page web pour afficher un bouton « Ajouter au panier » avec le descriptif d'un produit. Affichez aussi un bouton similaire sur chaque ligne du catalogue.

Puis vérifiez, en exécutant la page, que tous les boutons fonctionnent.

Incrémenter la quantité

Modifiez la méthode du composant *panier* permettant l'ajout d'un produit, pour qu'elle incrémente sa quantité au lieu d'ajouter une nouvelle ligne si le produit est déjà présent dans le panier.

Et vérifiez dans la page web que le nouveau comportement est bien celui attendu.

Vider le panier

Ajouter au composant *panier* une méthode qui en vide le contenu.

Puis affichez dans la page web, sous le panier, un bouton « Vider le panier » appelant cette méthode.

Basculer entre l’affichage détail ou panier

On veut maintenant afficher soit les informations détaillées d’un produit, soit le contenu du panier, et non plus les deux en même temps. C’est-à-dire que lorsque l’utilisateur sélectionne un produit, la colonne de droite de la page va afficher le détail de ce produit, et si ensuite il clique sur le bouton d’ajout au panier, l’affichage du détail du produit va être remplacé par celui du contenu du panier.

Vous allez créer pour cela un composant Seam nommé *affichage*, placé dans le contexte de la page (comme le composant *selection*), et qui va gérer une propriété booléenne *panier* indiquant s’il faut afficher le panier. Rappel : la convention de la norme JavaBean pour les propriétés de type booléen est de préfixer le *getter* par *is* au lieu de *get*, c’est-à-dire que la méthode s’appelle *isProp()* au lieu de *getProp()*.

Ajoutez à ce composant deux méthodes sans paramètre pour spécifier d’afficher le panier ou d’afficher le produit sélectionné, et appelez ces méthodes (après avoir injecté le composant *affichage*) aux bons endroits dans les classes *Selection* et *Panier*, pour que l’affichage bascule sur le mode opportun lors de la sélection d’un article et lors de l’ajout dans le panier.

Il vous reste à conditionner l’affichage de l’une ou l’autre des parties de la page web, en utilisant comme précédemment un composant JSF `<s:fragment>` avec une condition dans un attribut *rendered*.

Vous pouvez ensuite afficher systématiquement sur la page web un bouton « Voir le panier ». Si vous voulez afficher un lien plutôt qu’un bouton, utilisez le composant JSF `<h:commandLink>` à la place du `<h:commandButton>`.

Saisie de la quantité dans le panier (POUR EN FAIRE PLUS)

Remplacez sur chaque ligne du panier l’affichage de la quantité par une zone de saisie. Utilisez pour cela le composant JSF `<h:inputText>` qui génère une zone de saisie. La propriété JavaBean correspondante est indiquée de la même façon dans l’attribut *value*, par une expression JSF EL identique à celle utilisée pour l’affichage. Il est judicieux par contre d’ajouter un attribut *size* pour limiter la taille de la zone de saisie à un petit nombre de caractères.

Ajoutez aussi un bouton « Recalculer », appelant une méthode *recalcule()* sans contenu puisqu’il n’y a rien à faire, le montant total étant calculé lors de l’affichage.

Suppression de produits du panier (POUR EN FAIRE PLUS)

Puisque la quantité est modifiable, on veut maintenant supprimer les lignes pour lesquelles le client met la quantité à zéro. Ecrivez donc le traitement nécessaire dans la méthode ***recalcule()***. Attention si vous parcourez la liste des lignes du panier avec une boucle de type *foreach*, il ne faut pas supprimer des éléments de la liste que vous êtes en train de parcourir. Vous pouvez par exemple créer une liste temporaire qui soit une copie de la vraie liste (créez une ArrayList en passant la vraie liste au constructeur), parcourir cette liste temporaire, et supprimer les lignes de la liste réelle.

Ajoutez aussi sur chaque ligne du panier un bouton « Supprimer », et la méthode Java qui va avec.

Annexe A : composants JSF

<h:outputText>

Affiche le texte défini par l'attribut *value*, qui peut contenir un texte en dur ou une expression JSF EL.

```
<h:outputText value="Title : " />
<h:outputText value="#{book.title}" />
```

<f:convertNumber>

Sert à formater une valeur numérique. En le mettant à l'intérieur d'un composant <h:outputText>, la valeur à afficher est convertie selon le format spécifié dans l'attribut *pattern*.

```
<h:outputText value="#{book.price}">
  <f:convertNumber format="###0.00 $" />
</h:outputText>
```

<h:inputText>

Champ de saisie d'un texte. Il est associé à une propriété d'un objet JavaBean, définie dans son attribut *value*. Le composant affiche la valeur actuelle de la propriété, et la met à jour lorsque le formulaire est soumis avec la valeur saisie par l'utilisateur.

```
<h:inputText value="#{book.title}" />
```

<h:commandButton>

Affiche un bouton qui soumet le formulaire et déclenche une action. L'attribut *value* définit le libellé du bouton. L'attribut *action* désigne via une expression JSF EL la méthode Java à exécuter. Si la méthode prend des paramètres, ils doivent être indiqués entre parenthèses, sinon les parenthèses sont facultatives.

```
<h:commandButton value="Save" action="#{catalog.save(book)}" />
```

<h:commandLink>

Affiche un lien dans la page web, qui soumet le formulaire et déclenche une action. Le contenu du lien (texte, image) pourra être défini soit par le contenu du composant, soit s'il s'agit d'un texte par l'attribut *value*. L'attribut *action* désigne via une expression JSF EL la méthode Java à exécuter. Si la méthode prend des paramètres, ils doivent être indiqués entre parenthèses, sinon les parenthèses sont facultatives.

```
<h:commandLink action="#{catalog.save(book)}">Save</h:commandLink>
```

<h:dataTable>

Composant affichant une table. Les colonnes sont définies dans son contenu par des composants <h:column>. Le titre de chaque colonne, qui va s'afficher sur l'entête de la table, est défini dans le contenu de la colonne par un composant <f:facet name="header">.

La liste d'éléments à afficher dans la table est fournie sous la forme d'une expression JSF EL dans l'attribut *value*. L'attribut *var* définit une variable qui prendra successivement la valeur de chacun des éléments de la liste, et sera utilisée dans les différentes colonnes pour désigner les propriétés de l'élément courant.

```
<h:dataTable id="books"
    value="#{bookListBean.books}"
    var="book">
  <h:column>
    <f:facet name="header">Author</f:facet>
    <h:outputText value="#{book.author}" />
  </h:column>
  <h:column>
    <f:facet name="header">Title</f:facet>
    <h:outputText value="#{book.title}" />
  </h:column>
</h:dataTable>
```

<s:fragment>

Regroupe une partie de la page pouvant être composée d'éléments texte et de composants JSF à l'intérieur dans un unique fragment. N'a aucun effet visuel, mais c'est très utile pour conditionner l'affichage de toute une zone de la page via son attribut *rendered*.

```
<s:fragment rendered="#{book.available}">
  ...contenu...
  ...contenu...
</s:fragment>
```

<h:form>

Définit un formulaire dans la page web.

Annexe B : répertoire et fichiers du projet

src : répertoire des fichiers sources des classes Java.

build : répertoire dans lequel Eclipse compile automatiquement les classes Java. Son contenu est déployé sur le serveur d'application dans le répertoire **WEB-INF/classes**.

WebContent : répertoire dont le contenu est déployé sur le serveur web. Il contient tout le contenu web (pages Facelets, images, feuilles de styles CSS, etc.) à l'exception des classes Java.

WebContent/WEB-INF : répertoire WEB-INF de configuration de l'application web. Il contient principalement les fichiers suivants :

- **web.xml**, fichier standard de configuration de l'application web (servlets, etc.)
- **components.xml**, fichier de configuration externe des composants Seam
- **pages.xml**, fichier de configuration des pages de l'application, permettant de spécifier des actions à exécuter sur certaines pages, d'imposer l'authentification de l'utilisateur, ou de gérer la navigation entre les pages web de façon externe