Java Avancé – TP 3

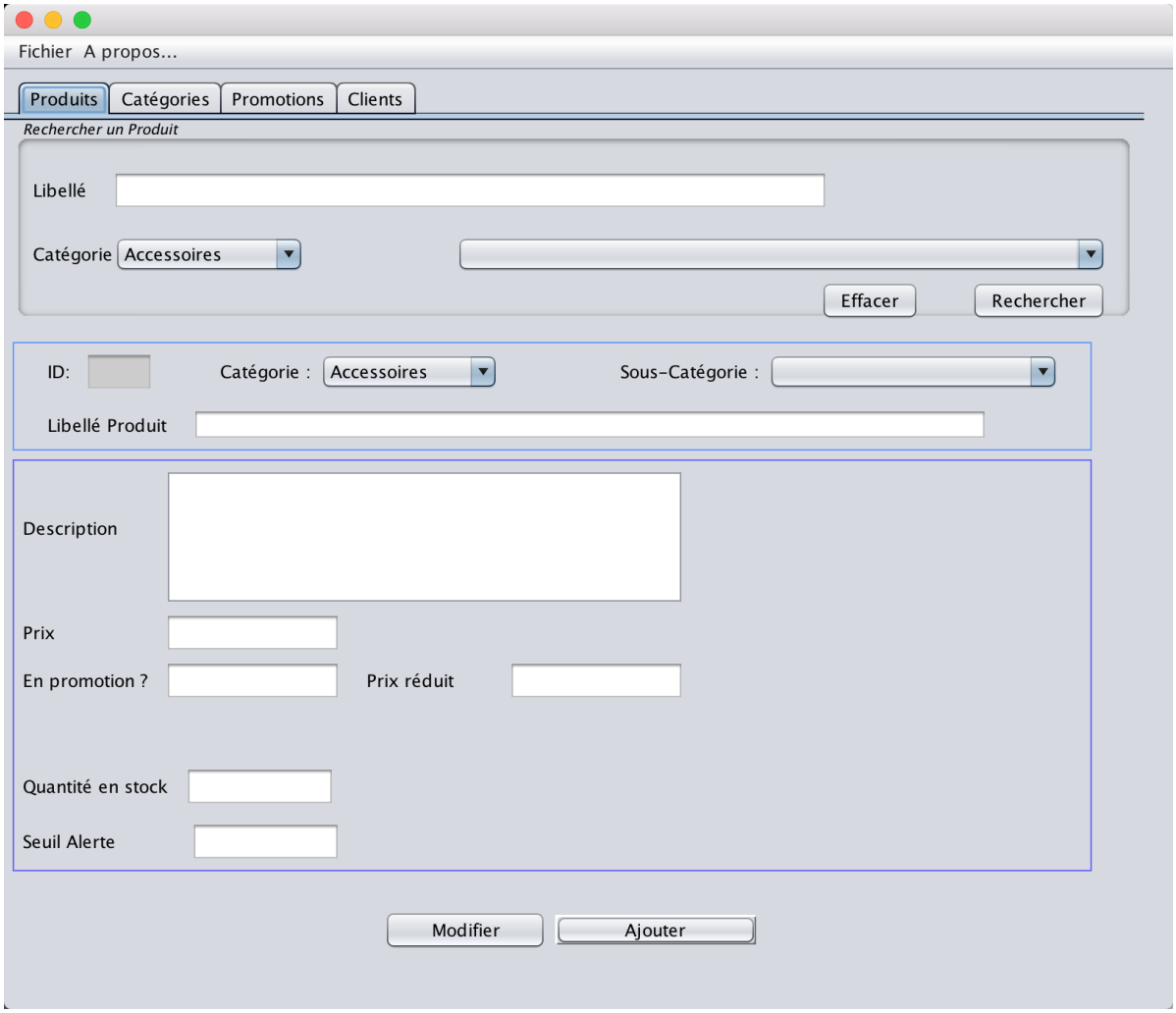
IHM Wysiwyg, Menus, *Master / Detail* de comboBox

Durée : 6h

L'objectif des prochains TP est de **programmer l'interface de Gestion de Catalogue (sujet de CPOO/PTUT)**, cette fois avec l'aide de l'IDE, avec une architecture MVC. Vous pourrez adapter l'IHM à vos propres besoins. Nous nous baserons sur l'IDE Netbeans pour ce TP. *Il est conseillé de regarder les vidéos illustrant le fonctionnement de l'utilisation de la palette avant de faire le TP.*

Dans ce premier travail, nous allons construire les bases de l'IHM et la compléter d'éléments annexes (contrôles, dialogues, menus).

Notre interface repose sur le fonctionnement suivant : le responsable va effectuer ses différentes gestions par le biais d'onglets, un par gestion :



PARTIE A – METIER

A.1 - Créer un **nouveau projet** avec une Java Application, sans fichier principal, appelé par ex. *TP3-Catalogue*. Sur le projet, clic droit : `new package…` que vous appellerez `metier` (rappel : noms de package tout en minuscules, respectez les règles d'écriture de Java, notamment *toutes les classes commencent par une MAJ*).

A.2 - Créer les **classes Métier** nécessaires. Pour ce TP, on a besoin de **Produit**.

Ajouter tous les champs nécessaires à la classe, ainsi que leurs accesseurs. Prévoir un champ String correspondant à un fichier de l'image du produit.

Gérer l'incrément automatique des ID avec un champ statique à chaque nouvelle création de produit. Créer au moins 2 constructeurs en plus de celui par défaut.

Toujours dans la classe Produit, écrire un champ statique `List<Produit> listeDesProduits,` et une **fonction statique** qui retourne la liste de produits créés.

```
public static List<Produit> getLesProduits() ;
```

Pensez à ajouter le produit créé à la **liste des produits,** dans les constructeurs.

Pour les catégories de produit, on propose ici de les gérer par une énumeration : créer une classe `enum`, par ex. pour des vêtements :

```
public enum CategorieProduit {
        ACC("Accessoires"), etc.
private final String nom;
   public String getNom() {
        return nom;
   }
```

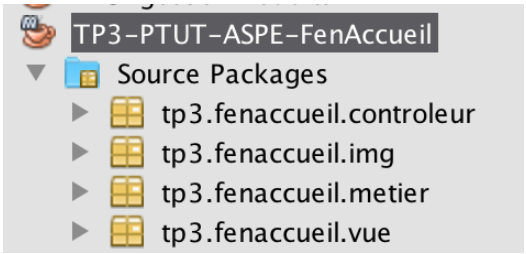
Ecrire une méthode qui retourne la liste des noms des catégories de produits, on en aura besoin pour les ComboBox : `public static List<String>` **`getLesCategories Produit()`** en exploitant la méthode **`values()`** de l'énumération. : cf la Javadoc.

Créer **autant de classes de sous-catégories** que nécessaire, aussi avec une méthode retournant la liste des noms (*String*) des sous-catégories.

PARTIE B - DEVELOPPEMENT DE L'INTERFACE

Sur le projet, clic droit : `new package…` que vous appellerez `vue`. Puis dans ce package, créer une fenêtre : **`new JFrame Form…,`** par ex. *FenAccueil*.

Créer une classe de lancement de la fenêtre (*TestFenAccueil*) avec le *main* (copier les lignes du *main* de la *JFrame* dans la classe de lancement) : on sépare bien la *vue* du *contrôle*. La placer au bon endroit dans votre architecture projet, qui doit ressembler à :



L'IDE NetBeans s'ouvre avec par défaut en mode WYSIWYG avec la palette des composants SWING à droite. C'est utile d'avoir les fenêtres Navigator, pour voir l'arborescence de vos composants (et leur nom), et à droite, la fenêtre Properties affichant les propriétés du composant sélectionné.

1- Fenêtre d'accueil

Définir le *layout* de votre choix ; par défaut, l'IDE utilise un mode 'Free Layout', c'est un *layout* propre à l'IDE-Swing. Si vous préférez un BorderLayout par ex. : clic droit sur la *JFrame* dans le navigateur, `SetLayout`, `BorderLayout`.

1.1- Les onglets

Sélectionner le `TabbedPane` de la palette et le placer sur votre fenêtre : vous pouvez le redimensionner pour qu'il occupe tout l'espace.

Vérifier dans les propriétés `Layout` – Direction, qu'il occupe la zone Centrale du `BorderLayout` si c'est ce que vous avez choisi.

Ajouter ensuite autant de panneaux (panels) que de gestions (onglets) et nommez-les correctement. Renommer vos variables pour faciliter l'écriture directe de code plus tard.

Pour le premier panneau correspondant à la *recherche* de produit : on imagine pouvoir chercher par le *libellé* d'un produit, sa *catégorie* ou sa *sous-catégorie*.

Ajouter un *JPanel* sur la fenêtre, définir un bord *TitledBorder* dans la propriété *Border* et ajouter le titre proposé. Ajouter les 3 `labels`, le `textField` et les 2 `comboBox`.

Pour ajouter les items des `comboBox`, il va falloir d'abord créer un **modèle des données**, dans le code source, car les données vont être modifiées dynamiquement selon le choix de l'utilisateur (les données entrées en dur dans les propriétés de la `ComboBox` sont immuables).

Les données des éléments des `comboBox` ici sont des *chaines* (méthodes statiques `getLesCategories()` de `CategoriesProduit` et `getLesXXX`() des sous-catégories).

Comment procéder ?

- Dans le code, ajouter ces 3 champs nécessaires, à la fenêtre *FenAccueil* :

```
private static List<Produit> lesProduits;
private ComboBoxModel<String> modeleDesCategories;
private ComboBoxModel<String> modeleDesSousCategories;
```

- Dans le constructeur avant l'appel à `initComponents()`, créer une liste de produits en dur (nous verrons la persistance plus tard): `lesProduits = new ArrayList<Produits>();`

`lesProduits.add( new Produit(--- arg. de votre constructeur ---));` etc.

Vous pouvez trier les produits selon l'« ordre naturel » avec la méthode statique `sort()` de `Collections` : `Collections.sort(lesProduits);` *Q1- Qu'appelle-ton « ordre naturel » de tri en Java ?*

- Pour définir le modèle de la `comboBox`, on doit créer une instance de `DefaultComboBoxModel(lesData)`, avec les données fournies soit sous forme de **tableau d'objets**, soit sous forme de **Vector**. Ici on va utiliser un tableau d'objets.

Pour cela, on convertit une collection de `String` en tableau : la méthode `toArrays()` de l'interface `List` réalise ce travail. Appliquez cette méthode à la liste des catégories via la méthode statique créée pour avoir la liste des catégories. Placez-le tableau obtenu dans une variable par ex. `tabCategories`. On peut ainsi créer le modèle de la comboBox avec :

```
modeleDesCategories = new DefaultComboBoxModel( tabCategories );
```

et c'est ce modèle qu'on affecte à la comboBox dans ses propriétés. Pour cela, revenir au Design : sélectionner la comboBox, choisir `model` : cliquer sur les *3 points* pour ajouter votre propre code (**Custom Code**) et mentionner le nom de votre variable dans le champ : `laComboBox.setModel( )`, ici : **`modeleDesCategories.`**

Q2- Que voyez-vous affiché maintenant dans les Propriétés pour model de cette CBox ?

Q3- Comment l'IDE signifie qu'une propriété d'un composant a été modifiée par rapport aux valeurs par défaut ?

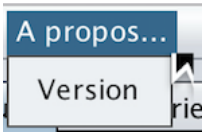
1.2- Pattern Master / Detail

Il s'agit ici d'afficher les sous-catégories correspondant à la catégorie choisie par l'utilisateur. C'est le patron d'affichage Master/Detail : la CBox Catégorie impose le modèle des sous-catégories. Nous verrons cela dans la gestion des événements.

2- Barre de menu

Sur la palette Swing, déployer les composants de Menu et ajouter une `MenuBar` à votre fenêtre. Vérifier dans le navigateur que le composant est bien placé DANS la `JFrame`. Renommez-le, changez les noms des 2 menus pré affichés, en '**Fichier** et '**A propos**. Le menu n'est jamais placé dans une zone du *layout*, il est sur une couche distincte (cf cours).

Ajouter ensuite les items de menus proposés, avec un séparateur :



Prévisualiser avec  pour voir ce que ça donne. *Q4- Que constatez-vous ? Exécutez votre programme. Que pouvez-vous donc conclure ?*

PARTIE C – GESTION DES ÉVENEMENTS

Avec l'IDE c'est très simple (cf vidéo de démonstration) : pour ajouter la gestion des événements sur un composant, on dispose de 3 méthodes :

- soit on **double-clique** sur le composant en mode Design et l'IDE nous positionne directement dans le code où on va décrire le traitement de l'action,
- soit on **clique droit** sur le composant et on choisit `'Events'` puis par ex. `'Action'`,
- soit dans la **fenêtre de Propriété**, on choisit l'affichage du bouton `'Event'`, et on modifie les propriétés selon le type d'événement.

Les événements que l'on va gérer dans un premier temps sont :

- Gérer les **sous-catégories**
- Ajouter un item de menu **Quitter** et définir une action de confirmation (demandant à l'utilisateur de vérifier que ses produits ont été sauvegardés : on verra la persistance plus tard).
- Les actions sur les **items du menu Fichier et A propos**
- L'action du **bouton « Effacer »**

C1- Action sur la comboBox des sous-catégories : master/detail

C'est le code correspondant à une action sur la comboBox des catégories : double-cliquez dessus en mode Design pour arriver à l'endroit du source où on va écrire le traitement.

Il faut d'abord récupérer la catégorie sélectionnée avec getSelectedItem().

Utiliser un **`swicthCase`** pour chaque catégorie, et pour chaque, récupérer le tableau des sous-catégories avec **`toArray(),`** comme on l'a fait pour les catégories. On peut alors définir le modèle des sous-catégories :

```
cbBoxSousCategorieModele = new DefaultComboBoxModel(tabNoms);
```

Une fois les cas traités, on peut généraliser le traitement en affectant le modèle à la bonne comboBox : **`cbb.setModel(cbBoxSousCategorieModele);`**

Q5- Ce code sera sans doute utilisé à **plusieurs** endroits de l'IHM (plusieurs affichages Catégorie / Sous-catégories) : quelle solution adopter en conséquence ?

C2- Action sur le menu Quitter

Modifier auparavant l'action par défaut de la fenêtre avec la propriété `DefaultCloseOperation` de la `JFrame`, pour ne rien faire quand l'utilisateur ferme la fenêtre.

Se positionner sur l'item de menu **Quitter**, clic droit, choisir `Event – Action`. L'IDE créé automatiquement un écouteur pour le type d'évènement `Action` (un `ActionEvent`), crée la méthode de l'écouteur associé, et place le curseur où coder le comportement à avoir quand on clique sur cet item.

Ici on va juste afficher un message de confirmation "Voulez-vous vraiment quitter ?" avec une `Confirm Dialog,` et si oui, quitter l'application : `System.exit(0);`

Q6- Observez les lignes de codes automatiquement ajoutées par l'IDE pour la gestion d'événements de l'item de menu Quitter, en allant voir le code d' `initComponents()`, et dire **quel type d'implémentation** des écouteurs est choisi par l'IDE parmi les 3 implémentations vues en cours.

*C3- L'action sur les items du menu **Fichier***

En cliquant que un item, on va ouvrir l'onglet associé avec la méthode `getSelected Component()`:

```
pannOnglets.setSelectedComponent(ongletPromotions);
```

*C4- L'action des items de menu **A propos…***

- **A propos** : affiche une fenêtre de dialogue (Message Dialog) avec les informations sur les auteurs et la version.

*C5- L'action du **bouton « Effacer »***

Faire en sorte que les champs soient mis à blanc et la comboBox des sous-catégories soit remise à vide.

TRAVAIL A RENDRE

Comme pour les autres TPs, vous déposerez votre travail + CR de TP avec les 6 questions sur Tomuss.