IUT de LYON, ASPE semestre 2
2018/2019

# Java Avancé – TP4

## JTable, JList, Barre d'outils

*Durée : 6h*

L'objectif du TP est de **poursuivre notre interface Gestion de Produits.** On imagine qu'il s'agit d'une application du service Approvisionnement, pour des personnels d'une entreprise, qui permet de faire l'inventaire. On veut pouvoir saisir et stocker des produits, présents dans l'entreprise.
Dans ce TP on va aussi apprendre à *débugguer* son programme. Une démo sera effectuée en séance mais vous pouvez aussi lire : https://fr.netbeans.org/edi/36/utilisation/debug.html

---

1- On va modifier l'interface pour **gérer les sous-catégories de façon dynamique**, avec une **JList** modifiable (on supprime la `comboBox` sous-catégorie du précédent TP). Pour cela :

a- Créer autant de **modèles de liste** qu'il y a de produits, comme nouveaux attributs de la fenêtre. Comme on veut que ces listes soient modifiables, on va utiliser `DefaultListModel`.

b- Dans le constructeur de la fenêtre, pré-initialiser les sous-catégories pour 2 listes (TP2 : audio et appareils photos). Pour cela une boucle est nécessaire, il n'y a pas de constructeur de `DefaultListModel`() avec un conteneur d'items.

c- Ajouter un champ texte et deux boutons (« Ajouter sous-catégorie » et « Supprimer sous-catégorie ») sous le label Catégorie, ainsi qu'une liste, rendue invisible.

d- Gérer les événements sur les 2 boutons précédents pour que l'utilisateur puisse mettre à jour la liste (ajouter et supprimer des sous-catégories). Pour ajouter une sous-catégorie, l'utilisateur va saisir la valeur dans le champ texte avant de cliquer sur le bouton « Ajouter sous-catégorie ». Pour supprimer, il faudra qu'un élément soit sélectionné dans la liste.

e- Faire en sorte que 3 éléments seulement soient **affichés à la fois** dans la liste.

3- Créer une **JTable** tabProduits pour afficher tous les produits.

a. La table contient 5 colonnes pour afficher les informations : Libellé, Catégorie, Sous-Catégorie, Prix, Quantité. Ajouter donc un label et champ texte au formulaire, pour la **quantité**.

b. Ajouter la table sous le panneau Caractéristiques Produit. Agrandir la fenêtre et ajouter le composant `Table` de la palette. On va définir notre modèle de table directement avec l'IDE : dans la propriété `Model`, choisir '`Table model customizer`' et saisir le nombre de colonnes (5), leur nom et leur type.

c. Faire en sorte que l'utilisateur puisse sélectionner plusieurs lignes à la fois, par ex. pour les supprimer.

4- Modifier l'événement associé au **bouton `Ajouter`**, qui va ajouter une ligne dans la `JTable` (supprimer le code qui affichait les données).
C'est donc un transfert de la vue (champs `textField` et `comboBox`) vers les données de la table, donc dans le modèle.

Utiliser pour cela la méthode `addRow()` du modèle, qu'il faut donc récupérer. **Que prend `addRow()` en paramètre ?** On enverra donc une instance de la classe attendue avec `new àVousDeTrouver[] {valeur_premiere_colonne, valeur_deuxieme_colonne, … }`.
Pour les valeurs, on va récupérer les données saisies par l'utilisateur dans les champs de l'interface avec les méthodes `getText()` pour les `textField`, la méthode `getSelectedItem()` pour la `comboBox`.

Attention, il faut envoyer les données *du type attendu* par les colonnes de la table : on transformera un élément de la `comboBox` en `String` avec `toString()`, et on utilisera les méthodes `parseXXX()` des wrappers (`Double`, `Integer`) pour extraire un réel/entier d'une chaîne.

5- Il s'agit maintenant de **synchroniser les champs texte avec la table** quand on clique sur un élément de la table.

- Définir une méthode `synchroniser`() qui permet d'afficher les valeurs de la ligne sélectionnée dans le formulaire, pour que l'utilisateur puisse par exemple le modifier. On utilisera la méthode **`mouseClicked()`** de l'adaptateur `MouseAdapter` (sélectionner la table, dans les `Events`, choisir **`MouseClicked`**, et ajouter un 'handler' appelé *synchroniser*).
  Récupérer le modèle, lire les données avec les méthodes `getValueAt(lig, col)`. L'argument 'ligne' sera définie par `getSelectedRow()` de la table. La 'colonne' sera successivement 0, 1 … 4 pour les valeurs des colonnes. Affecter chaque `textField` de ces valeurs avec l'utilisation de `toString`() quand nécessaire.

- Ajouter une bulle d'aide au bouton **`Modifier`** qui explique à l'utilisateur qu'il doit modifier les valeurs dans le formulaire, puis cliquer sur modifier pour changer les valeurs de la table (utiliser les propriétés : `toolTipText`).

6- Définir la méthode qui permet de **supprimer la ligne sélectionnée** par l'utilisateur quand ce dernier clique sur le bouton « `Supprimer` ». Afficher un *warning* si l'utilisateur clique sur le bouton supprimer sans sélectionner au moins une ligne de la table. Faire qu'on puisse supprimer plusieurs lignes à la fois.

7- Placer le curseur dans le champ libellé après avoir supprimer la saisie courante (bouton supprimer). Pour cela, utilisez la **gestion du focus**, qui marche en 2 temps : avant d'affecter le focus à un `textField` avec `requestFocus`(), il faut vérifier que le champ est 'focusable' avec `isFocusable`(). S'il ne l'est pas, on le rend : `setFocusable(true)`.

8- Corrigez aussi l'action du bouton ***Modifier*** du TP2, pour qu'on puisse modifier les valeurs de la ligne sélectionnée avec les valeurs du formulaire, quand l'utilisateur clique sur le bouton *Modifier*. C'est un code très similaire à celui du bouton *Ajouter*.

**Testez votre interface avec Modifier**. Quel type d'exception est levée quand on n'a sélectionné aucune ligne de la table ? Ajoutez un message d'erreur si l'utilisateur clique sur le bouton modifier sans sélectionner une ligne.

---

**En final, testez votre interface et listez tous les éléments d'ergonomie qui seraient nécessaires pour améliorer votre pgm. Implémentez en le plus possible.**

---

## FACULTATIF… Export Excel et Barre d'outils

9- Ajouter les boutons 'Aide' et 'à Propos' dans une **barre d'outils** plutôt que par menu, ainsi qu'un bouton 'Afficher' pour afficher la liste des sous-catégories du produit dans une fenêtre indépendante.

a- Créer la barre d'outils avec un bouton (sans bord), et ajoutez-la à votre fenêtre.

b- Modifier votre code pour afficher la liste des sous-catégories du produit dans une fenêtre NON modale, quand l'utilisateur clique sur le bouton 'Afficher' de la barre d'outils, ou un message d'erreur « Affichage des sous-catégories : pas de produit sélectionné » (warning message). Gérer le texte du bouton pour qu'il passe à 'Masquer', et implémenter la gestion qui convient.

10- Ajouter **l'export Excel** du contenu de la table.

Quelques indices :

- (Nul besoin de passer par l'API `jexcelapi` !)
- Une solution possible est, dans l'écouteur approprié, de choisir le fichier pour l'export, puis d'exporter le contenu de la table dans le fichier comme illustré ici :

```
void xxxxActionPerformed(ActionEvent e) {
File fichier = this.choisirFichier() ;      // méthode à écrire
exporterExcel( maTableProduits, fichier) ; // méthode à écrire
}
```

- Pour `choisirFichier()`, on utilisera une `FileDialog` : étudier la javadoc.
- Dans la méthode `exporterExcel()`, il suffit décrire les entêtes de colonnes d'abord, séparées par des tabulations dans par ex. un fichier texte type FileWriter :

```
FileWriter fo = new FileWriter(fichier); // fo pour 'fichier out'

for (int i = 0; i < monModeleTable.getColumnCount(); i++) {
      fo.write(monModeleTable.getColumnName(i) + "\t");
}
```

- Puis d'y placer un caractère de *fin de ligne* :

```
fo.write("\n");
```

- Ensuite on écrit chaque ligne de la table, avec une tabulation entre les champs :
```
fo.write(monModeleTable.getValueAt(i, j).toString() + "\t");
```
- Puis de placer une *fin de ligne* entre chaque ligne de la table :
```
fo.write("\n");
```
- On lance le tableur avec l'instruction :
```
Desktop.getDesktop().open(fichier); // le File fichier précédent
```

**NOTA : l'OS reconnaît qu'il s'agit du tableur par défaut, par l'extension du fichier (.xls).**

Q – Que se passe-t-il si on ne mentionne pas d'extension ?