

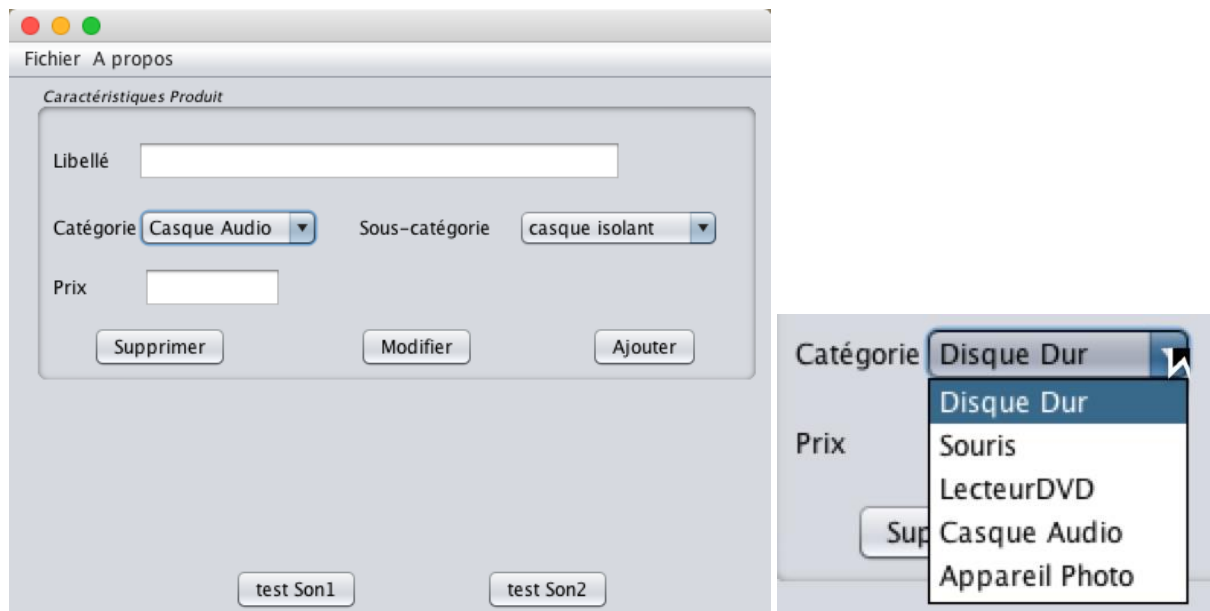
Java Avancé – TP facultatif

Boîtes de dialogue, Menus, Fenêtres modales

Durée : 4h

L'objectif du TP est de **programmer une interface Gestion de Produits, similaire à celle réalisé au TP2**, cette fois avec l'aide de l'IDE. Nous allons ensuite le compléter d'éléments annexes (contrôles, dialogues, menus).

PARTIE 1 – DEVELOPPEMENT DE L'INTERFACE SUIVANTE, DEMO GUIDEE



Créer un nouveau projet avec une Java Application, sans fichier principal, appelé par ex. gestionProduits. Sur le projet, clic droit : `new package...` que vous appellerez `vue` (par opposition aux packages `metier`, `contrôleur`, de l'architecture MVC). Puis dans ce package, créer une fenêtre : `new JFrame Form...`, donnez un nom par ex. `fenProduits`.

L'IDE 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- Panel Caractéristiques Produit

Définir un panel, ajuster sa taille à la dimension de la fenêtre, et modifier son bord dans les propriétés à droite (`Border` : cliquez sur les ...) et choisir `TitledBorder`, (saisir le titre), choisir une police de 10, en italique.

Ajoutez les 4 labels, les 2 champs texte, et les 2 comboBox.

Pour ajouter les items des comboBox, dans les propriétés, choisir **model** et saisir les items de votre choix (par ex., casque Audio, caméra, disque dur, souris, etc.).

Prévisualiser avec  pour voir ce que ça donne.

Ajouter une 2^{ème} comboBox pour la sous-catégorie. Seuls 2 items auront des sous-catégories (ici Appareil Photo et Casque audio). Cette fois les données de la 2^e CBox sont fonction de la 1^{ère} CBox (pattern Master/Detail). L'idée serait donc de pouvoir dynamiquement affecter le modèle des données de la sous-catégorie. Néanmoins on va choisir un modèle de données par défaut : choisir `Propriétés - model`. Cliquer sur ... et dans la CBox en haut, choisir `Custom Code`. Dans le champ texte, saisir un nom de variable (de type `ComboBoxModel`), qui sera affectée dans le code (par ex. `nom = modelSousCatAppareilPhoto`).

Q1- Que voyez-vous affiché maintenant dans les Propriétés pour model de cette CBox ?

Aller voir le code en cliquant sur le bouton `Source` en haut à gauche (dérouler éventuellement le code généré avec +). **Q2- quelle erreur mentionne l'IDE ?**

Corriger l'erreur en ajoutant un champ à la fenêtre. On va maintenant définir le *contenu* de la ComboBox, pour la sous-catégorie d'Appareil Photo. Pour cela, allez dans le code, dans le corps du constructeur de la fenêtre, et *avant l'appel* à `initComponents()`, définir un tableau de chaînes, l'envoyer au constructeur du modèle concerné :

```
String[] lesAppareilsPhoto = {"reflex", "analogique", "numerique"};
modelSousCatAppareilPhoto = new
    DefaultComboBoxModel(lesAppareilsPhoto);
```



Revenir au Design.

Comme tous les items n'ont pas de sous-catégorie, on va rendre **invisible** le label et la CBox concernant la sous-catégorie. Pour cela, cliquer sur le composant, dans les propriétés choisir le Bouton 'Code'. *C'est ici et seulement ici, qu'on peut insérer du code dans celui généré par l'IDE.*

Ajouter le code qui masque le composant (on l'affichera plus tard) dans la propriété : 'post-creation code' : `lbl_sousCategorie.setVisible(false)`; faire de même pour la comboBox de la sous-catégorie.

Astuce : pour retrouver rapidement le code qu'on a soi-même inséré, une solution est de mettre un commentaire reconnaissable (ex. : //tatata), puis chercher ce commentaire dans le code.

Nota : opaque n'a rien à voir avec la visibilité. Cocher la case 'opaque' revient à définir `setOpaque(true)` : Swing n'a pas à dessiner les composants qui sont situés derrière (notion d'efficacité). Mais cette méthode bug en fonction du Look&Feel choisi (de l'OS), et il est déconseillé de l'utiliser.

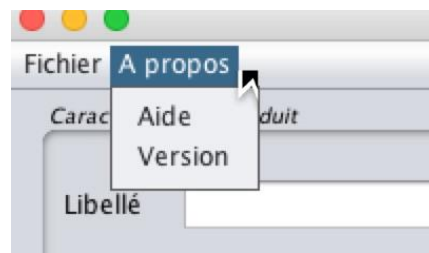
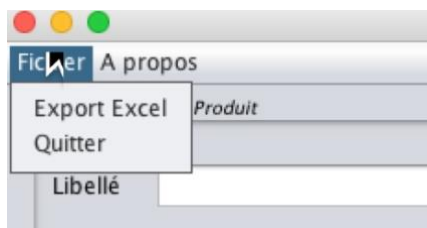
Revenir sur les Propriétés. **Ajouter les 3 boutons**, les renommer.

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'.

Q3- Prévisualisez. Qu'observez-vous ? Exécutez votre pgm. Que pouvez-vous conclure ?

Ajouter ensuite 1 item de menu dans Fichier : 'Export Excel' et 'Quitter' comme le montre l'écran suivant, et 2 items de menu dans A propos : 'aide' et 'version'.



Q4- Comment l'IDE signifie qu'une propriété d'un composant a été modifiée par rapport aux valeurs par défaut ?

3- Gestion des événements

Avec l'IDE c'est très simple : pour ajouter la gestion des événements sur un composant, soit on clique droit sur le composant et on choisit '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 vus en TP2 et que l'on doit gérer dans un premier temps sont :

- L'action sur le menu **Quitter**
- L'action sur le bouton **Modifier**
- L'action le boutons **Supprimer** (qui remet tout à blanc et masque la sous-catégorie)
- L'action sur la comboBox **Catégorie**, qui va afficher les labels et comboBox de la sous-catégorie concernée.

3.1- Action sur le menu Quitter

Se positionner sur l'item de menu, clic droit, choisir `Event - Action`. L'IDE crée 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. On n'a donc pas à choisir le mode d'implémentation de l'écouteur (être son propre écouteur, par classe anonyme, par classe interne).

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)` ;

Q5- 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 au TP2.

3.2- Action sur le bouton Modifier

Ce bouton ouvre juste une fenêtre de dialogue informant l'utilisateur qu'il peut modifier les données directement dans le formulaire de saisie. Si aucune donnée n'est déjà saisie, la boîte de dialogue ne doit pas s'afficher.

3.3- L'action le bouton Supprimer (qui remet tout à blanc et masque la sous-catégorie)

Pour l'instant, supprimer va effacer les données saisies, comme dans le TP2.

Procéder de la même façon que précédemment, et mettre les *champs texte* à blanc (par ex. `tf_libelle.setText("")`), et désélectionner l'item de la *comboBox* catégorie.

Masquer aussi les composants liés aux sous-catégories (label et `comboBox`).

Comme dans le TP2, on voudrait qu'un bouton (ici Supprimer) change d'aspect quand on passe la souris dessus (on 'entre' sur le bouton, `entered motion`). **Q6- Quel type d'évènement sera concerné ici ? Comme précédemment, donnez les lignes de codes ajoutées par l'IDE pour la gestion de cet évènement en allant voir le code d' `initComponents()`.**

Dans la méthode créée par l'IDE pour gérer l'évènement, ajouter le code suivant (dessine un bord rouge) :

```
redline = BorderFactory.createLineBorder(Color.red, 3);  
bt_supprimer.setBorder(redline);
```

Comme dans le TP2, implémentez le code réciproque, qui remet le bouton normal quand la souris 'quitte' le bouton (`exited motion`).

3.4- L'action sur la comboBox Catégorie

Pour cela, on va ajouter un évènement sur la *comboBox des Catégories* : à chaque clic sur un item, on va tester s'il y a une sous-catégorie à afficher, ou pas. Placez-vous sur la CBox,

ajouter un événement de type 'action' (clic droit). Dans le code de la méthode `actionPerformed()`, il faut tester si on est sur un *casque audio* ou un *appareil photo*, les 2 seules catégories possédant des sous-catégories. Pour cela, il faut récupérer l'item sélectionné avec `getSelectedItem()` :

```
cbb_categorie.getSelectedItem().equals("Appareil Photo") // mettre  
votre String exacte
```

Si c'est le cas : on *rend visible* les composants liés à la sous-catégorie (label et cbbox). Puis on *affecte le bon modèle* d'items, à la cbbox de la sous-catégorie avec par ex. :

```
cbb_sousCategorie.setModel( modelSousCatAppareilPhoto );
```

Attention : pour les catégories qui n'ont pas de sous-catégorie, on *garde invisible* les composants de sous-catégorie (label et cbbox).

PARTIE 2 – COMPLEMENTS DE L'INTERFACE

Ajouter les gestions des événements suivants :

- Action sur le **menu A propos – Version** : affiche une fenêtre de dialogue (Message Dialog) avec les informations sur les auteurs et la version.
- Action sur le **bouton Ajouter**, qui va ouvrir une fenêtre de type `MessageDialog` avec un résumé des informations sur le produit choisi.
- Actions de **contrôle** quand on clique sur **Ajouter** :
 - o sur le champ `Prix`, vérifier que c'est une valeur numérique (cf indications ci-dessus) ;
 - o sur les champs textes, vérifier qu'au moins un caractère est saisi ;
 - o sur le bouton `Ajouter`, que les champs sont tous remplis. Afficher un avertissement si l'utilisateur oublie de saisir un champ.
- Action du **sous-menu Aide** dans `A propos`. Ajouter une boîte de saisie (`InputDialog`) pour demander à l'utilisateur "Quelle est votre question ?" lorsqu'il clique sur l'item « Aide » et pour l'instant, écrire la question en mode texte dans la fenêtre d'exécution (on imagine qu'il y aura un traitement ultérieur, des questions posées).

Pour vérifier si le champ texte contient une valeur numérique, vous pouvez utiliser la méthode suivante :

```
Float tryParseFloat(String num) {  
    try {  
        return Float.parseFloat(num);  
    } catch (NumberFormatException e) {  
        return null;  
    }  
}
```

Ou plus court, en exploitant les expressions régulières¹ avec la méthode `matches()` de `String`.

(On fera l'export Excel dans la suite du TP).

FACULTATIF : TEST DE SONS

On aimerait proposer maintenant la possibilité de **jouer des sons** (par ex. pour tester les casques audio). Ajoutez à l'interface 2 boutons `bt_son1` et `bt_son2` comme montré sur l'interface.

Récupérez 2 fichiers sons (courts) sur internet (format `.wav` non compressés, ou `.mp3`) et les copier dans un répertoire package 'sons' dans votre projet (sur le package, clic droit : `Tools - Show in Finder`, copier / coller les fichiers sons trouvés).

Pour jouer un son par la machine virtuelle Java, il faut définir une URL qui correspond au fichier son souhaité :

```
java.net.URL url1 =
    fenTableProduit.class.getResource("/sons/roulement-tambour.wav");
```

On souhaite pouvoir tester les sons quand on clique sur les boutons ajoutés : dans la méthode associée au bouton de son :

- Créer 1 composant audio (interface `clip`) permettant de jouer des sons, ainsi qu'un objet `AudioInputStream` pour le flux audio.

```
Clip clip;
AudioInputStream ais;
```

- Instancier le clip et le flux audio avec :

```
clip = AudioSystem.getClip();
ais = AudioSystem.getAudioInputStream(url1);
```

- Puis charger le flux audio dans l'instance d'`AudioSystem` créée :

```
clip.open(ais);
```

- On joue le son avec par exemple la méthode `loop()`, regardez les autres possibilités :

```
clip.loop(1); // joue 2 fois le son (0 et 1)
```

- Veuillez à bien gérer les exceptions.

¹ Cf <https://codes-sources.commentcamarche.net/faq/1286-utilisation-des-expressions-regulieres-en-java#les-expressions-regulieres-en-java>