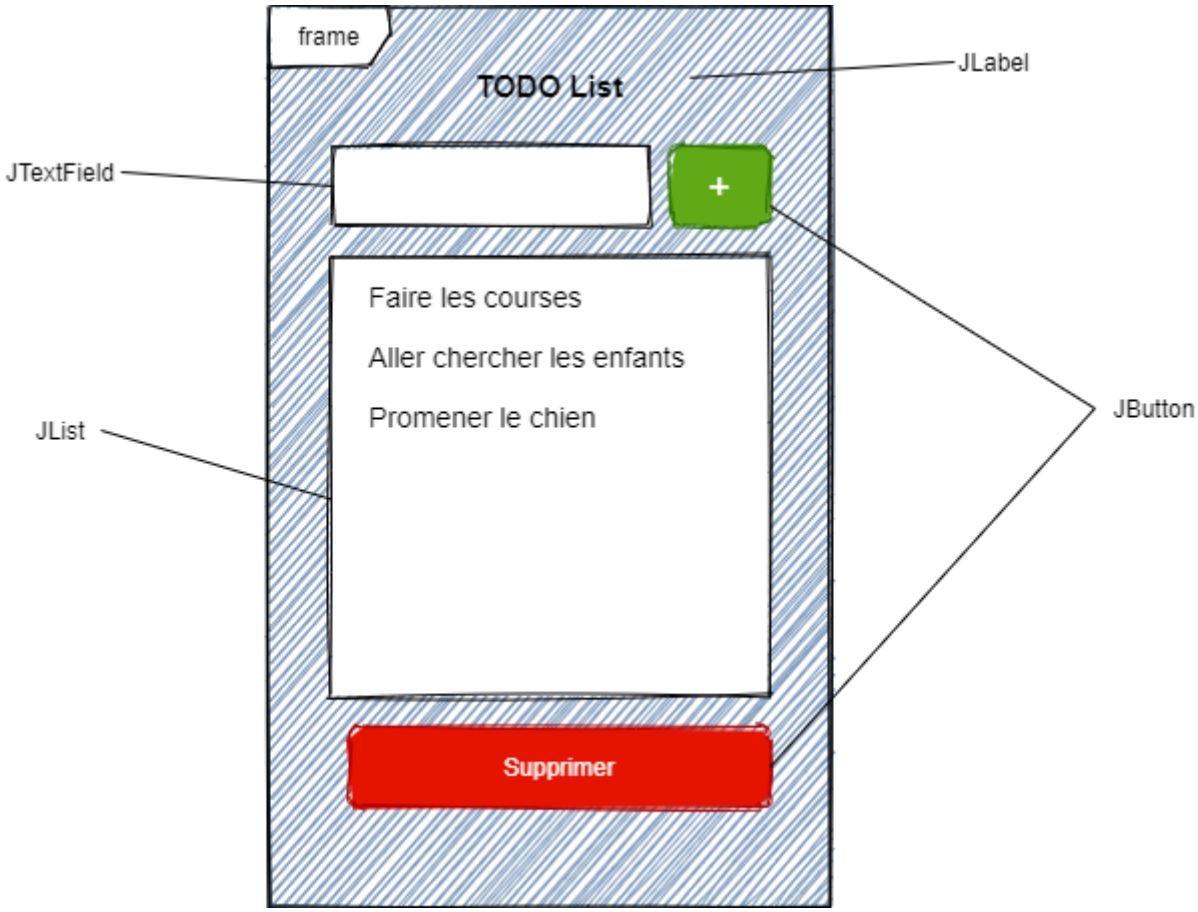
TP 10 : Composants et leurs modèles

A. Une zone de liste

Objectif : Réaliser une TODO List



Le développement de cette Todo-list est séparé en plusieurs parties.

- la **vue** ou interface est la partie graphique du champ `Jlist`, (Ce qu'on voit)
- le **modele** qui s'occupe des données de la liste.
- le **controller** est l'ensemble du code qui sera exécuté lors de *l'initialisation* du composant ou lors d'un *événement* déclenché par utilisateur.

Création d'une JList

Créer une nouvelle interface où vous y ajouterez une `JList`. Cherchez dans la palette de Windows Builder.

Ici, nous nommerons la JList "`jListTask`".

Modèle

Le model est un objet qui va s'occuper **exclusivement** des données qui sont dans la liste.

Nous allons modifier le comportement par défaut du composant `JList`, pour nous faciliter la vie.

> Important : Modifier le code source, là ou est initialiser votre composant :

```
// Construction de la JList
JList<String> jListTask = new JList<String>();
// Création d'un model de String
DefaultListModel<String> model = new DefaultListModel<String>();
// Attachement du model au composant
jListTask.setModel(model);
```

L'objet `model` de la classe `DefaultListModel` permet d'ajouter les fonctionnalités suivantes :

- `model.getSize();` => Donne le nombre d'élément dans la liste.
- `model.addElement(String element);` => Ajoute un élément à la fin de la liste.
- `model.remove(int index);` => Supprime un élément dans la liste
- `model.isEmpty();` => Retourne `true` si la liste est vide.
- `model.clear();` => Vide la liste de ses données.

Vue

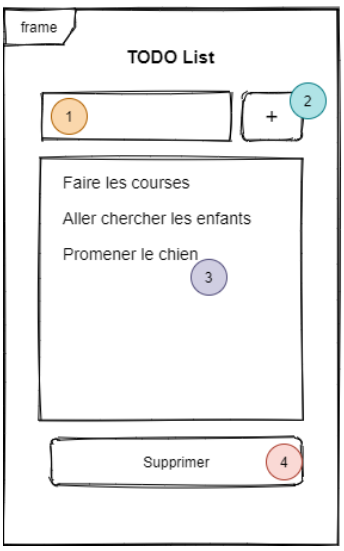
Le composant JList est le composant visuel. Il contient les propriétés comme la taille, la couleur... Mais il est aussi possible de lui ajouter des écouteurs d'evenements, car c'est lui qui recoit le "click" de la souris.

Par extension, il permet aussi de savoir quels éléments sont selectionnés.

- `jListTask.getSelectedIndex()` => retourne -1 si aucune ligne n'est sélectionnée
- `jListTask.getSelectedValue()` => retourne la valeur sélectionnée, null si aucune ligne selectionnée.

Entre autres ...

Travail à faire



Le **bouton 2** doit permettre d'insérer chaque nouvelle tache dans la liste, saisie dans le **champs 1**. Attention, on ne doit pas ajouter une tache si elle est vide.

Le **bouton 4** doit permettre de supprimer les lignes selectionnées et seulement celles là.

Un double click sur la **liste 3** permet de passer la tache à terminer.

Visuellement : ajoutez du texte "fini" à la fin de la tache.

Si l'utilisateur re-effectue un double click sur cet element, il redevient "à faire".

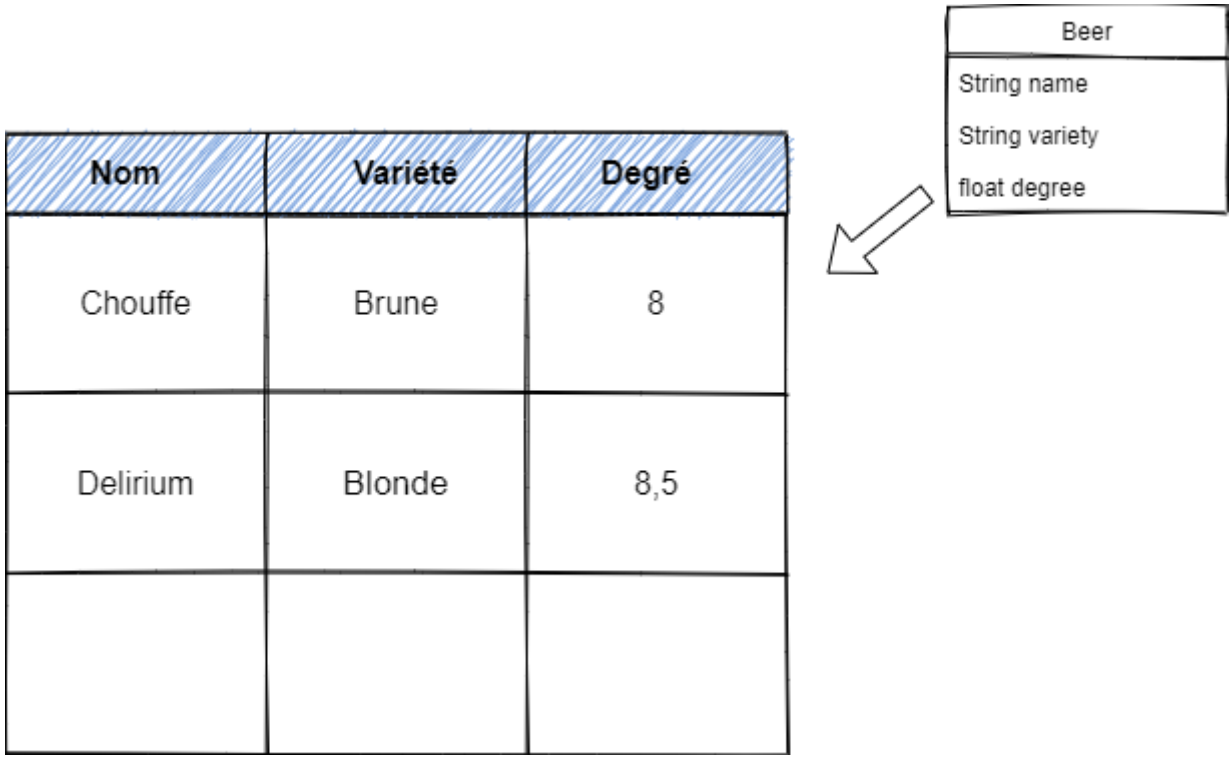
Visuellement : enlevez le texte "fini" à la fin.

Exo Bonus

Faire un tableau Kanban, avec un effet "drag and drop" !

B. les tableaux

Objectif : Afficher les informations d'une collection d'objets : des bières.



Nous allons utiliser un tableau :

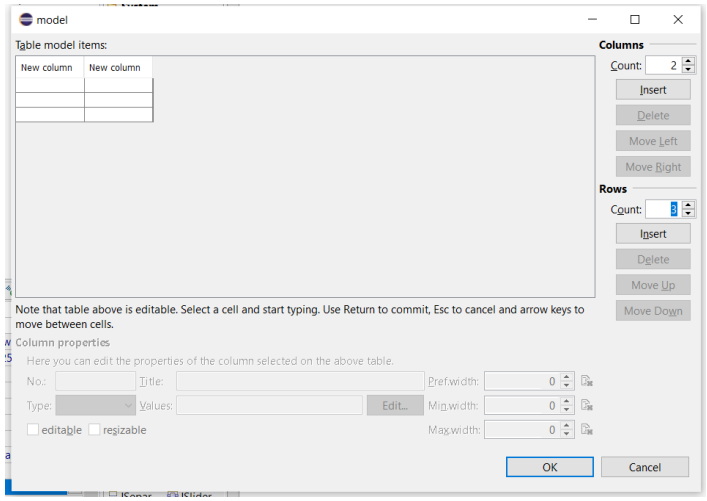
- ►la vue ou interface est la partie graphique du tableaux `JTable`,
- ► le modele gère les données du tableau. Ici, il est de type `DefaultTableModel`.
- ►le controller est l'ensemble du code qui sera exécute lors de *l'initialisation* du composant ou lors des *évenements* déclenchés par un utilisateur.

Création d'une JTable

Créer une nouvelle interface où vous y ajouterais une `JTable`.

Vous la nommerez `jTableBeer`.

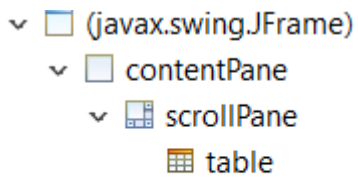
Dans properties, choisir **model** pour le modifier.



Modifiez le nom des colonnes, ainsi que le nombre de colonnes pour obtenir 2 colonnes (pour l'instant):

- Nom (Chouffe, Delirium, ...)
- Variété (Blonde, Brune, ...)

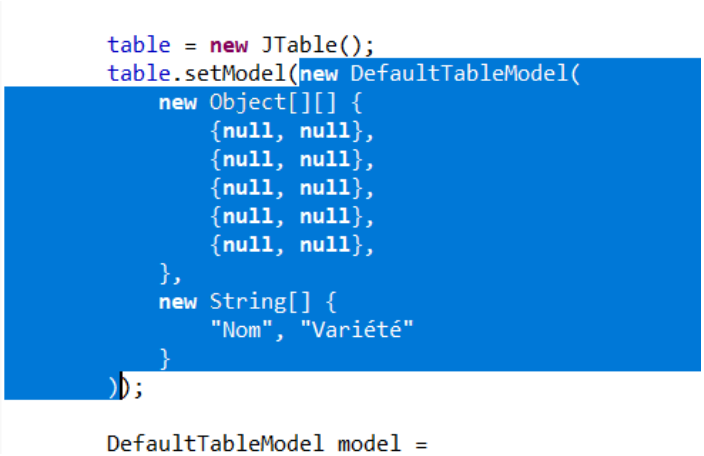
> Pour les utilisateurs d'éclipse, vous pouvez rencontrer des problèmes d'affichage de l'entete du tableau : JTable header not showing



Vous devez avoir quelques chose comme ça :

Modèle

Comme tout à l'heure on va placer notre model dans une variable pour pouvoir le manipuler.



```
table = new JTable();
table.setModel(new DefaultTableModel(
    new Object[][] {
        {null, null},
        {null, null},
        {null, null},
        {null, null},
        {null, null},
    },
    new String[] {
        "Nom", "Variété"
    }
));

DefaultTableModel model =
```

Attrapez le code qui est contenu dans `setModel()`, pour le mettre dans une variable `model`.

Finalement :

```
// Construction de la table
table = new JTable();
// Contruction du model de la table
DefaultTableModel model =  new DefaultTableModel(
    new Object[][] {
        {null, null},
        {null, null},
        {null, null},
        {null, null},
        {null, null},
    },
    new String[] {
        "Nom", "Variété"
    }
);
```

```
// Attachement du model à la table
table.setModel(model);
```

L'objet `model` de la classe `DefaultTableModel` permet d'ajouter les fonctionnalités suivantes :

- `model.addRow(rowData);` => Ajoute une ligne, rowData est un tableau d'Objet.
- `model.getRowCount();` => Donne le nombre de ligne
- `model.getValueAt(row, column);` => Donne l'élément qui se trouve sur la ligne `row` et sur la colonne `column`.
- `model.setValueAt(aValue, row, column);` => Remplace l'élément qui se trouve sur la ligne `row` et sur la colonne `column`, par la valeur `aValue`.

TODO : Modifier le code pour ajouter une troisieme colonne, nommée "degré".

Manipulation des données

Vous pouvez modifier le contenu du tableau initial, en utilisant «clic-droit» et en sélectionnant «table contents» (Netbeans) le troisième onglet `row` est un éditeur de contenu.

Autre méthode pour ajouter une rangée «a la mano»:

```
/* Une Rangée */
Object[] row1 = {"1664", "blonde", 3.4};
model.addRow(row1);
```

Une rangée peut contenir des chaines de caractères autant que des entiers, on utilisera donc un tableau de type `Object`. Cela permet de ne pas préciser la nature du tableau.

Cependant il est possible de préciser le type de chaque colonne, et ainsi empecher d'entrer un mauvais type : par exemple du String dans un float.

Les lignes s'ajoutent à la suite de votre tableau.

Pour obtenir le nombre de rangée:

```
int nb = model.getRowCount();
```

Pour effacer la rangée d'index `i`:

```
model.removeRow(i);
```

Pour effacer un tableau on peut s'imaginer faire une boucle sur la longueur du tableau pour effacer chaque colonne.

Travail à faire

Partie 1 : la classe "Beer"

Soit l'entité `Beer (name, variety, degree)`, créer la classe `Beer` qui correspond.

Créez les accesseurs GET et mutateur SET, ainsi qu'un constructeur.

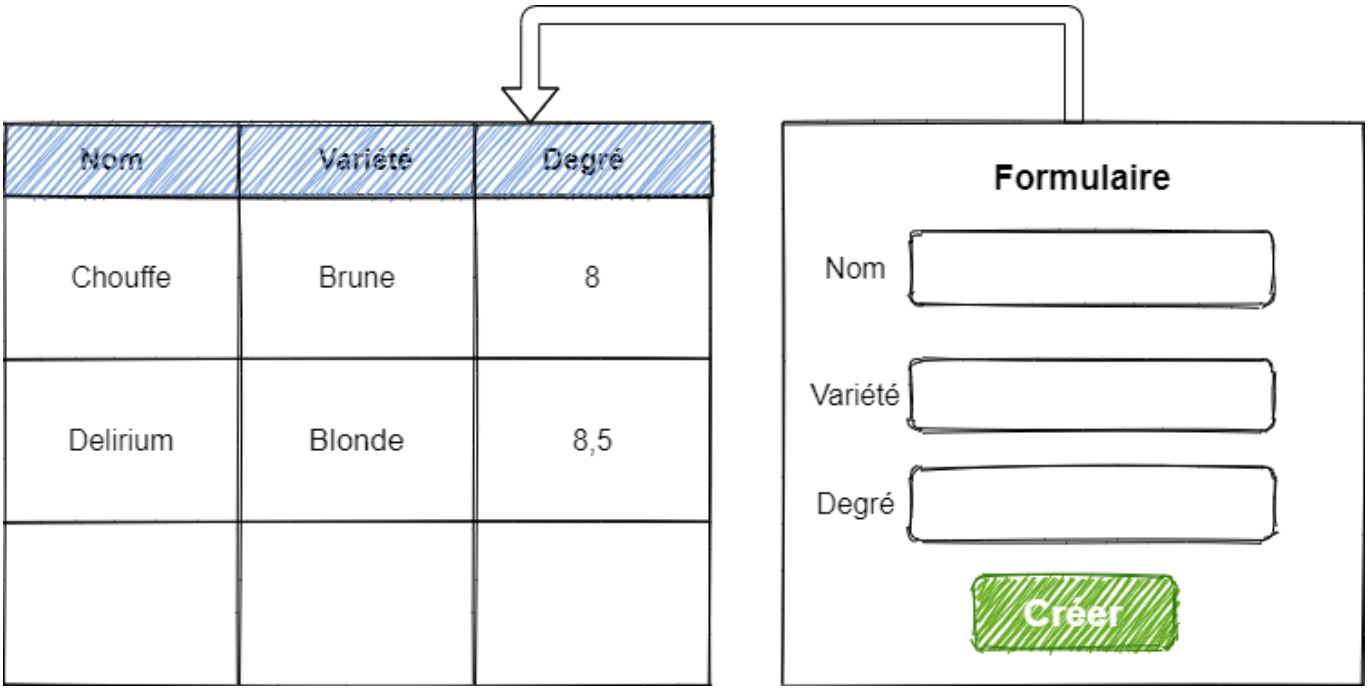
Créez une méthode `toRow()` qui transforme l'objet Beer, en tableau d'objet. Idéal pour être ajouter dans le tableau.

Le code suivant doit être possible :

```
// Construction d'un objet Beer
Beer beer = new Beer("1664", "blonde", 3.4);
// Conversion : Beer => Object[]
Object[] row1 = beer.toRow();
// Ajout d'une rangée
model.addRow(row1);
```

Partie 2 : Nouvelle bière

Créez un formulaire qui permet d'ajouter une nouvelle entrée.

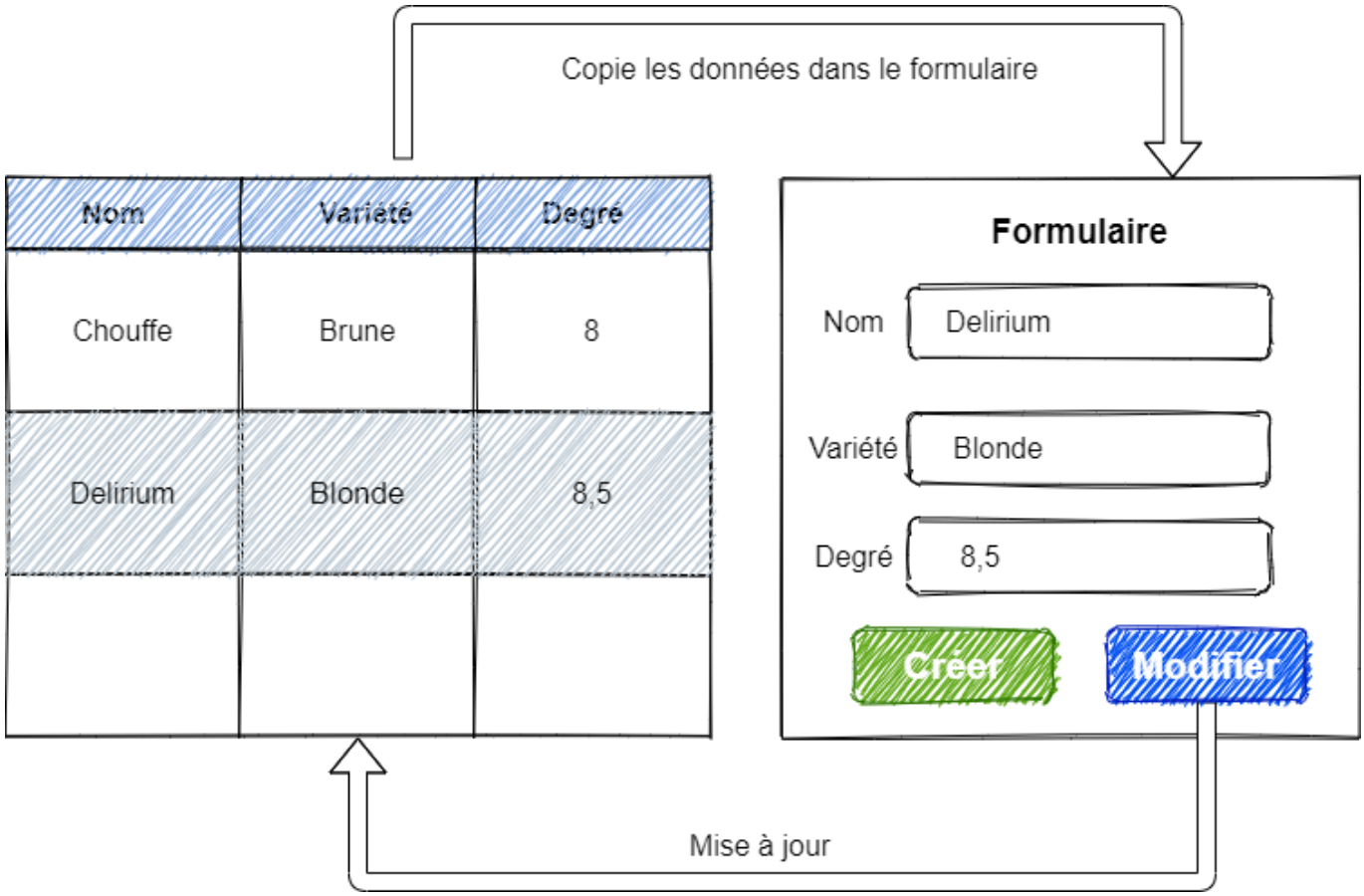


Vérifiez que les types des valeurs soient corrects et que les champs ne soient pas nuls.

Partie 3 : Modification

En utilisant l'evenement "MouseClicked", faire en sorte que lorsqu'on selectionne une ligne dans le tableau, les données de cette ligne remplissent le formulaire automatiquement.

Au click du bouton modifier, la ligne du tableau est mise à jour avec les nouvelles données.



Exo bonus : Suppression

Ajoutez la possibilité de supprimer une ligne.