Cours 420-236LI Développement de programme Automne 2023 Cégep Limoilou Département d'informatique Professeur : Martin Simoneau	**Formatif 5** *TableView* *Cellule maison de ListView*

Objectifs

1. Utiliser des *TableView*
2. Créer des cellules personnalisées

À remettre :

3. Le travail sera remis sur Léa à la date indiquée.

Contexte :

- Remettre vos projets sur Léa
- Une remise par étudiant
- Pour en apprendre plus sur le *TableView* :
  - http://tutorials.jenkov.com/javafx/tableview.html
  - https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/TableView.html

À faire

- **Ouvrez le projet *formatif5-tableView* et observez le code :**
  - La classe ***AppFormatif5*** est complète et ne fait que lancer l'application
  - Vous allez travailler principalement dans la classe **Controller**.
  - Le package *model* comporte également les classes *Personne* qu'on veut afficher dans la table.

1. On va programmer la méthode *Controller.**createTableView*** qui configure correctement la *TableView*
   - La *TableView* va contenir des objets de type *Personne*.
   - On ne peut pas utiliser une *TableView* sans un minimum de configuration. Comment une personne s'affiche sur plusieurs colonnes ?
   - Enregistrez sur chaque colonne une classe (appelée fabrique) qui permet de convertir l'objet *Personne* avec la méthode ***setValueFactory***. **Ne pas** confondre avec la méthode *setCellFactory* que nous utiliserons plus loin.
     - *JavaFX* possède un objet générique par défaut qui peut faire le travail simplement. *PropertyValueFactory* est un objet qui retourne la valeur associée à l'attribut dont le nom a été passé en paramètre. Il suffit de passer une instance de cette classe à la méthode *setValueFactory*().

```
new PropertyValueFactory<Personne, String>("nom")
```

     - Le constructeur de la fabrique prend en paramètre le nom de l'attribut de l'objet contenu dans la table (ici *Personne*) qui sera affiché dans la colonne. La classe est générique et elle doit avoir le paramétrage suivant :
       - Le premier paramètre doit être le même que celui de la colonne (ici *Personne*).
       - Le second paramètre est le type de donnée affichée dans la colonne (ici une chaîne de caractères).
     - Attention, avec l'âge il faut gérer un entier plutôt qu'une chaîne de caractères.

- Ajoutez quelques personnes dans la table de façon statique avec la méthode *getItems*(). La table fonctionne comme la *ListView*. Elle emmagasine ses données dans une *List* et elle utilise un selectionModel pour gérer la sélection de l'utilisateur.
- Notez que la *TableView* vous permet de trier les personnes simplement en cliquant dans les en-têtes.

2. Programmez les méthodes *ajoute*, *efface* et *modifie* :
   - *ajoute*
     - ajoute un Personne générique avec des valeurs par défaut. Faites comme pour une *ListView*.
     - Pour pratique les fichiers FXML on vous demande de demander les informations de la nouvelle personne à l'aide d'un Dialog personnalisé dont la partie « custom » est le *GridPane* contenu dans le fichier ***personne.fxml***. On peut utiliser un fichier FXML sans qu'il soit associé à un controller *javaFX*. Pour faire le lien entre les éléments FXML et le code qu'on utilise les FXID. Par exemple pour récupérer un TextField avec un FXID dans le fichier FXML on peut faire (après avoir chargé le fichier fxml avec la méthode *load*() évidemment) :

```
TextField nomTextField = (TextField) loader.getNamespace().get("fxid");
```

       - Ici loader est le *FXMLLoader* et le *fxid* est le *fxid* utilisé dans le fichier *fxml* pour le *TextField* concerné.
     - On doit donc :
       - Charger le fichier persone.fxml.
       - Prendre une référence sur chacun des TextField.
       - Mettre le *Gridpane* reçu en chargeant le fichier *fxml* dans un *Alert* (en mode confirmation) avec :

```
dialog.getDialogPane().setContent(gridpane);
```

       - On affiche le Dialog et on récupère les informations qui se retrouveront dans les 3 *TextField* dont on a récupéré les références un peu plus tôt.
       - On instancie une nouvelle Personne en lui donnant les informations reçues dans les 3 *TextField.*
       - On ajoute la nouvelle *Personne* dans la table.
   - *efface*
     - Retire la personne qui est sélectionnée dans la *tableView*. Faites comme pour une *ListView*.
   - *modifie*
     - On fait exactement comme pour *ajoute*, mais cette fois on récupère d'abord les informations de la personne sélectionnée pour les afficher d'emblée dans le dialogue présenté à l'utilisateur.
     - Au lieu d'ajouter une nouvelle personne, on va modifier les informations de la personne sélectionnée avec les informations reçues des 3 *TextField*.
     - Les modifications ne s'afficheront pas automatiquement. Il faut demander à la table de se remettre à jour lorsqu'on modifie un élément qui s'y trouve déjà avec la méthode ***refresh***().

```
tableView.refresh();
```

3. Programmez la liste avec cellules maison
   - Il faut d'abord créer une classe pour gérer la nouvelle sorte de Cellule. Allez dans la classe ***cell.PersonneListCell*** :
     - Cette classe doit être enfant de la classe *ListCell<Personne>*.

- Pour simplifier les choses, on va utiliser le fichier *personne.fxml* comme base pour notre cellule maison. La cellule sera donc exactement comme le dialogue qui fait la saisie des personnes. Dans le constructeur de *PersonneListCell*:
    - Ici on ne veut pas que le *FXMLLoader* instancie automatiquement un nouveau *contrôleur JavaFX*. Il faut donc lui en fournir un avant d'appeler sa méthode *load*(). La cellule qu'on est en train de créer doit justement servir de *contrôleur JavaFX*. Utilisez la méthode *setController* du *FXMLLoader* pour lui passer la cellule en cours de création comme *contrôleur JavaFX*.
    - Dans le constructeur de *PersonneListCell*, charger le fichier FXML avec *load.*
    - Placez le nœud obtenu dans l'attribut *cellRoot*. Ce dernier nous permettra de donner l'apparence voulue à chaque cellule.
    - Rendez tous les champs non *éditables* avec la méthode *setEditable*().
- Chaque fois que la *ListView* doit afficher une personne, elle demandera à une fabrique de créer une nouvelle cellule. On va faire la fabrique un peu plus tard, mais pour l'instant il faut gérer la méthode *updateItem*() qui gère le contenu de la cellule. Dans la méthode *updateItem*(*Personne personne*, boolean *empty*)
    - *Si la personne reçue en paramètre n'est pas null ou empty est vrai:*
        - *Placez les informations de la personne reçue dans les différents TextField correspondants. Vous devez également placer la racine fabriquée par le FXMLLoader comme nœud de cellule avec la méthode* **setGraphic***()*
    - *Sinon:*
        - Enlevez *l'item* de la cellule ainsi que le graphique en envoyant un *null* aux méthodes *setItem* et *setGraphic*.
- Pour indiquer au ListView qu'il doit utiliser la nouvelle cellule au lieu de la cellule par défaut, on doit lui fournir une méthode qui fabrique la nouvelle cellule. On appelle cette méthode une fabrique. Dans la méthode createListView() appelez la méthode *setCellFactory* de l'objet personnneList et passez-lui un callback qui retourne un objet de type *PersonneListCell.* Le callback recevra en paramètre la liste *personneListView*.

FIN