

Ressource R2.02 : Développement d'application avec IHM

Séance TP 5 : une mini-application

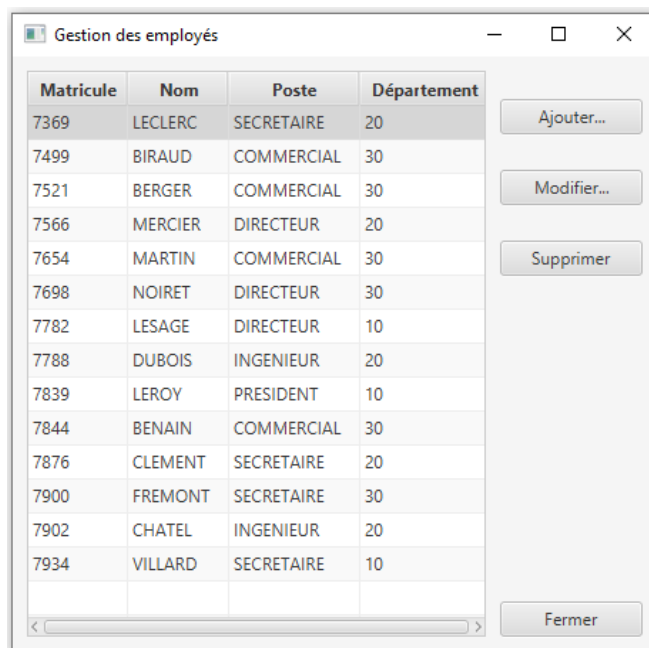
À travers un programme très simple de gestion de données sur des employés, cette séance présente l'architecture d'une mini-application JavaFX composée de plusieurs fenêtres. La séance aborde aussi des nouvelles notions comme le menu contextuel et le double-clic.

1 – Présentation de la mini-application

L'exemple qui sert de support est composé de trois fenêtres et d'une boîte à message de confirmation pour la suppression.

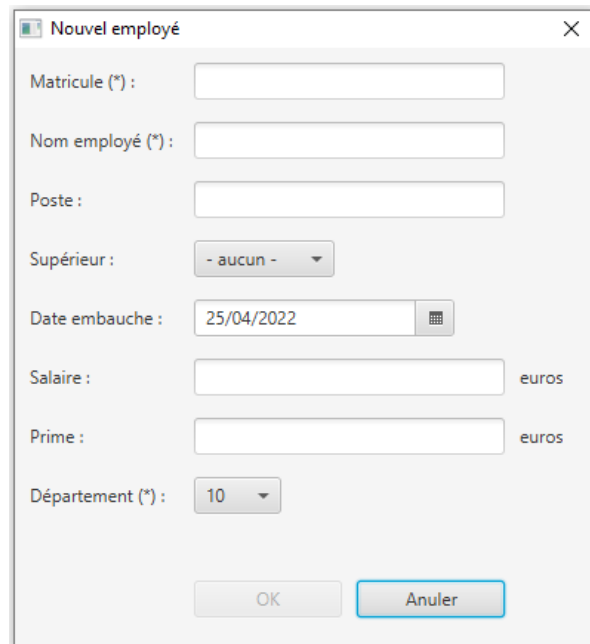
1.1 Fenêtre "Liste des employés"

Fenêtre redimensionnable, elle est composée d'un *TableView*, de trois boutons pour les actions de base sur les employés et d'un bouton *Fermer* qui met fin à l'exécution.



1.2 Fenêtre "Nouvel employé"

Fenêtre modale et non redimensionnable, elle contient des composants adaptés à la saisie des informations d'un employé. Les champs obligatoires y sont signalés par (*). Certains champs ont des valeurs par défaut. Remarquez enfin que le bouton OK est grisé par défaut.

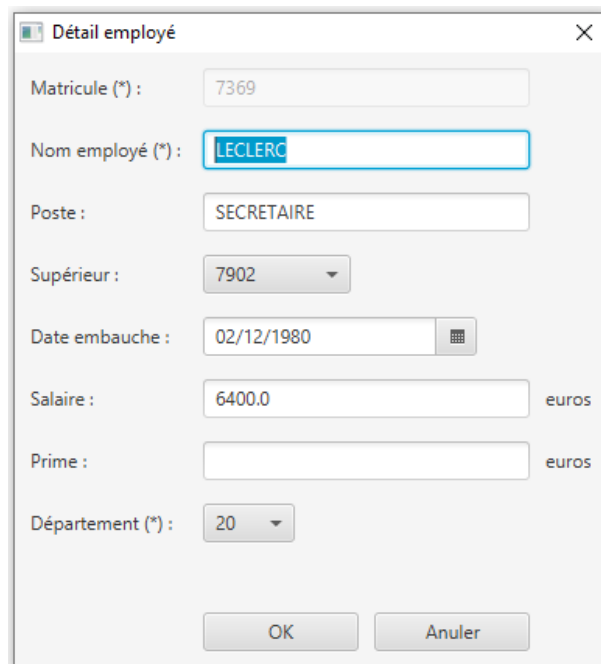


The screenshot shows a modal window titled "Nouvel employé" with a close button (X) in the top right corner. The window contains the following fields and controls:

- Matricule (*): A text input field.
- Nom employé (*): A text input field.
- Poste: A text input field.
- Supérieur: A dropdown menu with the value "- aucun -".
- Date embauche: A date input field showing "25/04/2022" with a calendar icon.
- Salaire: A text input field followed by the unit "euros".
- Prime: A text input field followed by the unit "euros".
- Département (*): A dropdown menu with the value "10".
- At the bottom, there are two buttons: "OK" (disabled, greyed out) and "Annuler" (active, blue border).

1.3 Fenêtre "Détail employé"

Fenêtre modale et non redimensionnable, elle est identique à la précédente dans sa forme mais son rôle est différent : les champs sont pré-remplis avec les données de l'employé qui a été sélectionné. On remarquera que le matricule (identifiant) n'est pas modifiable.

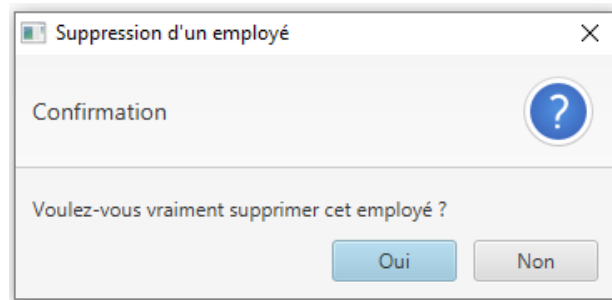


The screenshot shows a modal window titled "Détail employé" with a close button (X) in the top right corner. The window contains the following fields and controls, pre-filled with data:

- Matricule (*): A text input field containing "7369".
- Nom employé (*): A text input field containing "LECLERC".
- Poste: A text input field containing "SECRETAIRE".
- Supérieur: A dropdown menu with the value "7902".
- Date embauche: A date input field showing "02/12/1980" with a calendar icon.
- Salaire: A text input field containing "6400.0" followed by the unit "euros".
- Prime: A text input field followed by the unit "euros".
- Département (*): A dropdown menu with the value "20".
- At the bottom, there are two buttons: "OK" (active, greyed out) and "Annuler" (active, greyed out).

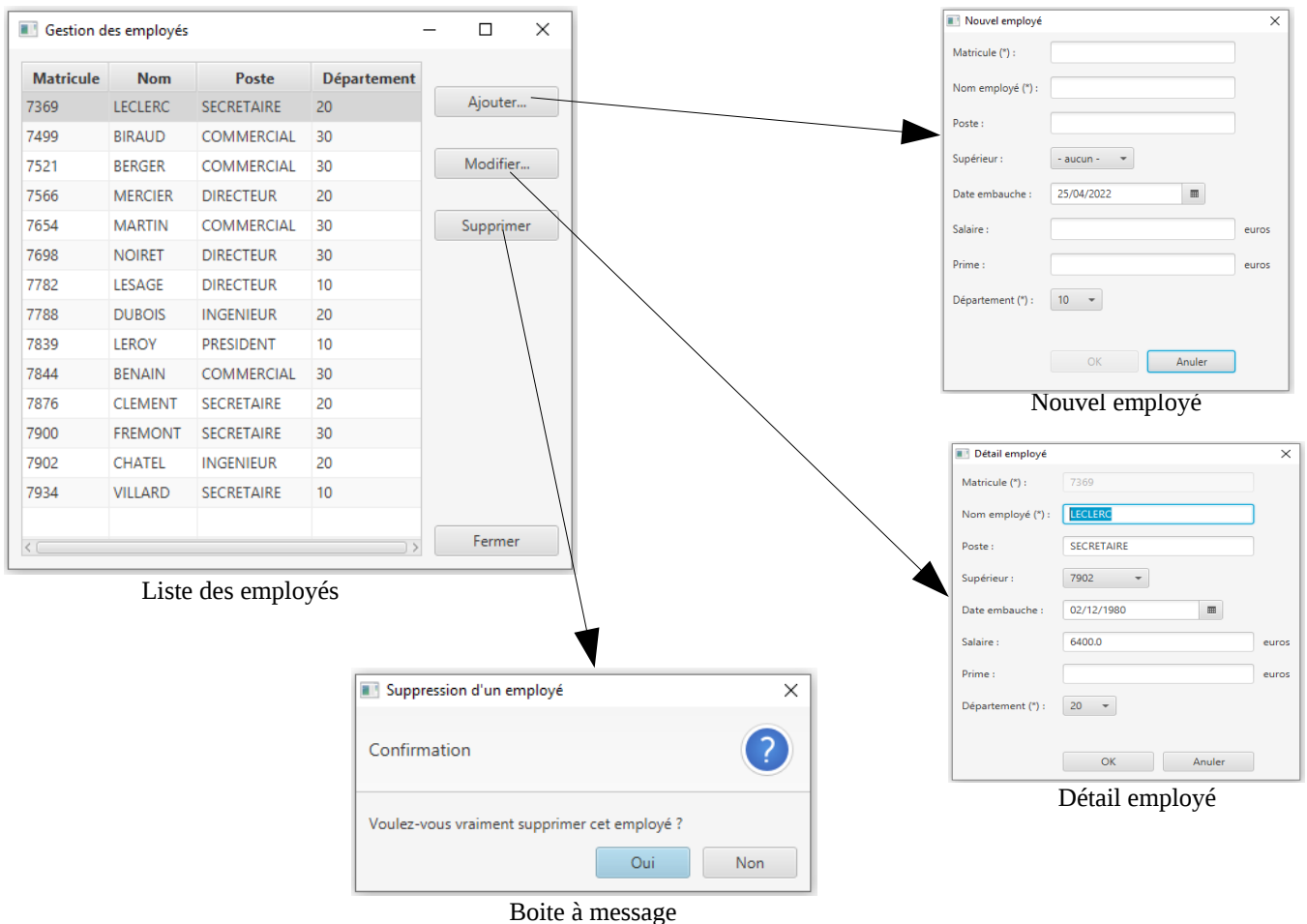
1.4 Boîte à message de confirmation

Ce n'est pas une fenêtre à proprement parler (ce n'est pas une "Stage"), c'est une popup créée grâce à la classe `Alert`.



1.5 Fonctionnement de l'application

Le comportement attendu de l'application peut être schématisé comme ceci :



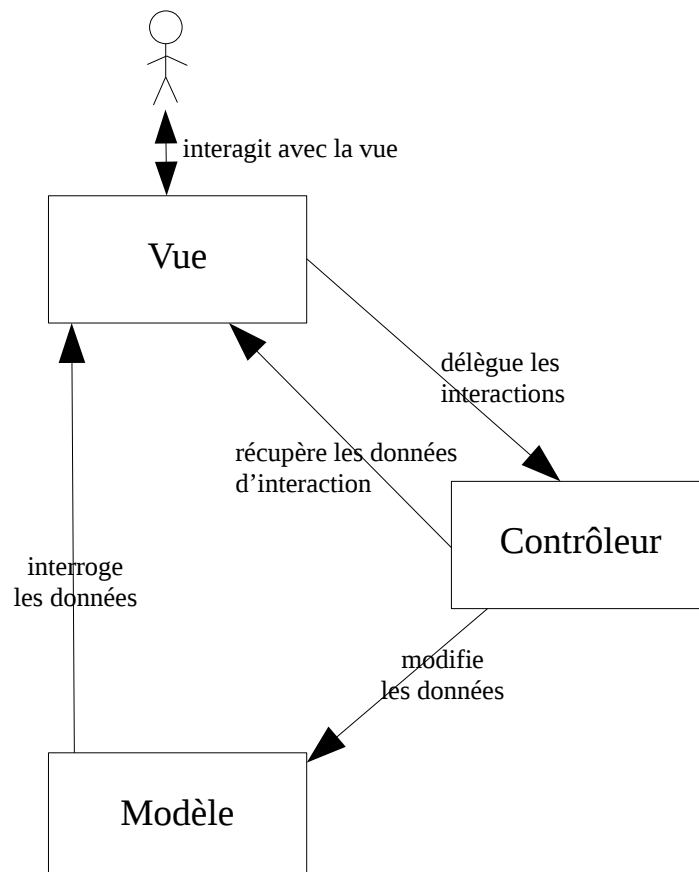
2 – Architecture de l'application

2.1 Architecture MVC

Une architecture communément admise en conception logicielle est l'architecture MVC pour Modèle–Vue–Contrôleur. L'intérêt d'une telle architecture réside dans la séparation des responsabilités (logique métier / logique de présentation).

- Le **modèle** contient la logique métier et l'état courant de l'interface durant le cycle dialogue avec l'utilisateur. Il peut par exemple contenir des requêtes à une base de données.
- La **vue** porte toute la logique de présentation. Son travail consiste à afficher les données du domaine et à recevoir des interactions de l'utilisateur. La vue ne gère pas les interactions avec les utilisateurs mais les délègue à un autre composant : le contrôleur.
- Le **contrôleur** reçoit les interactions de l'utilisateur de la vue et les traite en modifiant le modèle.

Schéma de l'architecture MVC :

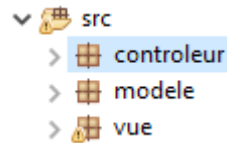


2.2 Nouvelle structuration du projet

Nous allons essayer de nous approcher de cette architecture, d'abord en modifiant la structure du projet JavaFX.

2.2.1 Les paquetages

Les classes du *modèle*, celles de la *vue* et celles de la partie *contrôleur* seront réparties dans trois paquetages distincts.



2.2.2 Les "classes fenêtres"

Les classes "fenêtre" (et leur classe "contrôleur" si l'IHM est en fxml) seront créées dans le paquetage "vue".

Pour chaque fenêtre il y aura désormais :

- une classe "fenêtre" (qui hérite de Stage),
- sa classe "contrôleur" associée,
- un fichier *fxml* décrivant l'IHM,
- un éventuel fichier *css*.

2.2.3 La classe principale

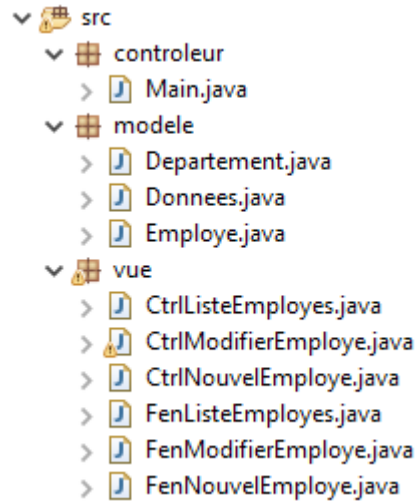
Dans le paquetage contrôleur, la classe principale va déclarer et initialiser autant de variables de classe qu'il y a de fenêtres dans l'application.

```
public class Main extends Application{
    static private FenNouvelEmploye fNouvEmp;
    static private FenModifieurEmploye fModifieurEmp;
    static private FenListeEmployes fListeEmp;

    public void start(Stage f) throws Exception {
        fNouvEmp = new FenNouvelEmploye();
        fModifieurEmp = new FenModifieurEmploye();
        fListeEmp = new FenListeEmployes();
        ...
    }
    ...
}
```

À FAIRE

Créez un nouveau projet sur Eclipse et créez les trois paquetages mentionnés. Puis ajoutez les classes comme sur la figure ci-dessous. Les classes (à compléter) sont disponibles sur Moodle.



3 – Quelques précisions

3.1 Modalité

On rappelle qu'une fenêtre modale est une fenêtre qui "bloque" l'utilisateur en l'empêchant d'accéder aux autres fenêtres de l'application. C'est souvent le cas des boîtes de dialogue qui forcent l'utilisateur à valider (ou annuler) l'action commencée avant de pouvoir faire autre chose. Dans notre application, c'est le cas des formulaires de création et de modification d'un employé : pas question que l'utilisateur laisse une telle fenêtre ouverte et passe à autre chose.

Rappel : `fNouvel.initModality(Modality.APPLICATION_MODAL);`

À FAIRE

Dans la méthode start() de la classe Main, faites en sorte que les fenêtres fNouvel et fModifier soient modales.

3.2 Les données

Dans cette version de l'application, les données ne sont pas persistantes (pas d'enregistrement dans des fichiers ni dans une base de données) : la classe *Donnees* gère des listes initialisées avec quelques valeurs.

Note : les valeurs non définies sont codées par "" pour les chaînes de caractères et par -1 pour les nombres.

4 – La fenêtre "Liste des employés"

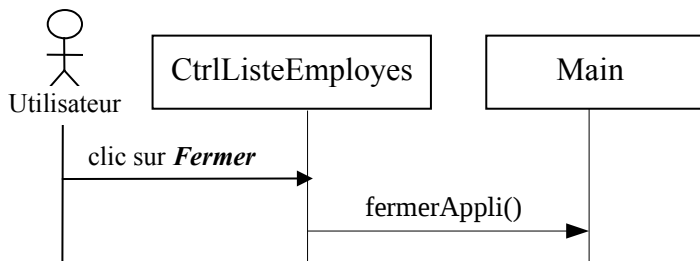
4.1 Gestion du redimensionnement

À FAIRE

Comme toute fenêtre redimensionnable, la fenêtre "Liste des employés" doit avoir une taille minimale cohérente, qui l'empêche d'afficher un contenu dégradé (titre masqué, composants qui se chevauchent ou disparaissent, etc.). Définissez une taille minimum pour cette fenêtre.

4.2 Le bouton Fermer

Ce n'est pas la fenêtre elle-même qui décide de sa fermeture. Quand elle détecte un clic sur le bouton *Fermer*, elle en informe la classe principale qui va procéder à la fermeture. Ce fonctionnement peut être schématisé par un diagramme de séquence :



À FAIRE

Programmez le clic sur le bouton *Fermer* : dans la classe *CtrlListeEmployes*, complétez la méthode *clicFermer()* puis dans la classe *Main* complétez la méthode *fermerAppli()*.

Note : pour mettre fin à l'exécution on utilise `System.exit(0)`.

4.3 La suppression

4.3.1 Gérer l'état du bouton Supprimer

Le bouton *Supprimer* doit être actif lorsqu'un élément est sélectionné dans la liste. À l'inverse, il doit être "grisé" (inactif) si aucun élément n'est sélectionné dans le `TableView`.

À FAIRE

Dans la méthode `initialize()`, en vous servant de la propriété **rien**, faites en sorte que l'état du bouton *Supprimer* soit cohérent. Au passage, faites de même avec le bouton *Modifier*.

```
BooleanBinding rien =
    Bindings.equal(tvListeEmployes.getSelectionModel().selectedIndexProperty(),
        -1);
```


4.3.2 Message de confirmation

Il est impératif de toujours demander une confirmation à l'utilisateur lors d'actions irréversibles comme la suppression. La classe `Alert` permet de construire facilement une telle boîte à message. Pour rappel, il faut créer et configurer une instance de la classe `Alert`, en indiquant le type de popup (`CONFIRMATION` ici), le message et les boutons qui doivent être présents sur la popup (`OUI` et `NON` ici).

```
Alert alert = new Alert(  
    AlertType.CONFIRMATION,  
    "Voulez-vous vraiment supprimer cet employé ?",  
    ButtonType.YES,  
    ButtonType.NO  
);
```

La nature du message peut figurer dans le titre de la popup :
`alert.setTitle("Confirmation de suppression");`

Enfin, il faut afficher la popup :
`alert.showAndWait();`

À FAIRE

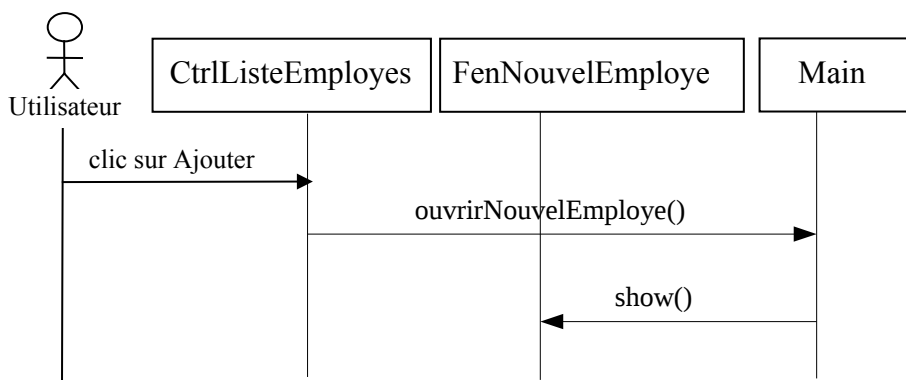
Faites en sorte qu'un clic sur le bouton Supprimer fasse apparaître une boîte à message de confirmation.

Note : en cas de confirmation, contentez-vous d'afficher un message dans la console ; la suppression effective de l'employé sera vue plus loin.

4.4 Déclencher l'ouverture de la fenêtre "Nouvel employé"

Un clic sur le bouton *Ajouter* doit déclencher l'ouverture de la fenêtre "Nouvel employé". Mais comme vu précédemment, c'est la classe `Main` qui gère l'ensemble des fenêtres de l'application et en contrôle l'ouverture.

La classe `CtrlListeEmployes` va donc informer la classe `Main` qui elle, va déclencher l'ouverture de `FenNouvelEmployé`. Un tel fonctionnement est résumé par ce diagramme de séquence :



À FAIRE

Dans la classe *CtrlListesEmployes* programmez le clic sur le bouton *Ajouter*. en suivant le diagramme de séquence. Testez successivement la saisie de deux nouveaux employés. Que se passe-t-il la deuxième fois ?

Avant l'affichage de la fenêtre "Nouvel employé", il faut réinitialiser les champs du formulaire. Une nouvelle méthode de la classe *CtrlNouvelEmploye* va se charger :

- d'effacer les champs texte (exemple : `txtMatricule.clear()`)
- de sélectionner une valeur dans les listes déroulantes (par exemple : `cbSuperieur.setValue(cbSuperieur.getItems().get(0));`)
- et de remettre la date du jour dans le champ date d'embauche (par exemple : `dpDateEmbauche.setValue(LocalDate.now());`).

Dans la classe *FenNouvelEmploye*, on crée une variable d'instance qui va représenter le contrôleur de la fenêtre :

```
private CtrlNouvelEmploye ctrl;
```

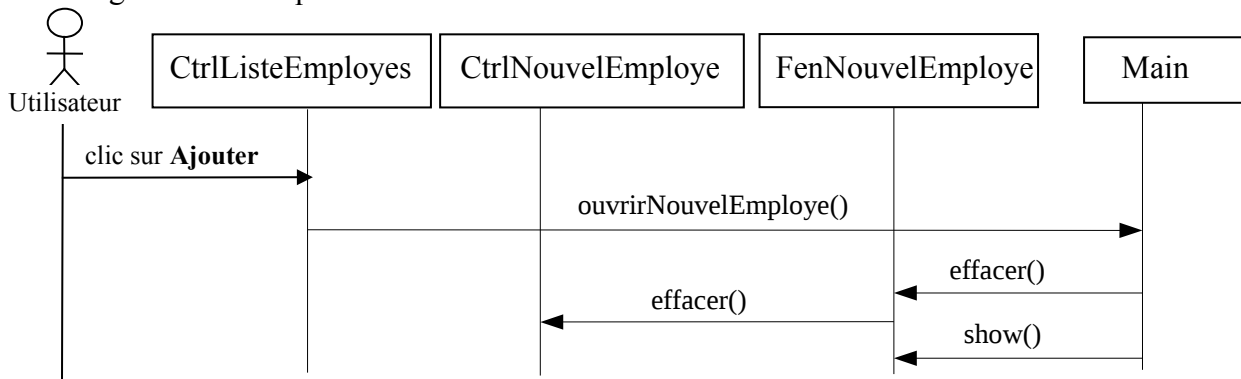
Cette variable est initialisée juste après le chargement du fichier fxml :

```
ctrl = loader.getController();
```

Dès lors, la "classe fenêtre" *FenNouvelEmploye* peut communiquer avec sa "classe contrôleur" pour lui demander de réinitialiser les champs du formulaire :

```
public void effacer() {  
    ctrl.effacer();  
}
```

Le diagramme de séquence devient :



Note : les méthodes `effacer()` ont le même nom mais il s'agit bien de deux méthodes différentes puisque situées dans des classes différentes.

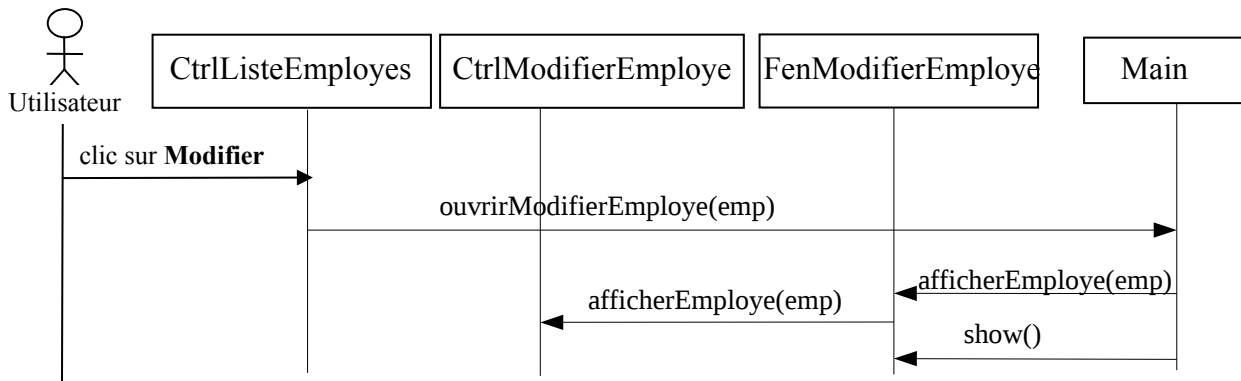
À FAIRE

En vous basant sur le nouveau diagramme de séquence, ajoutez les variables et méthodes décrites ci-dessus pour que le formulaire soit réinitialisé avant chaque ouverture.

4.5 Déclencher l'ouverture de la fenêtre "Modifier employé"

À FAIRE

Programmez de la même manière le clic sur le bouton Modifier de la fenêtre "Liste des employés". Mais cette fois, le formulaire doit afficher les informations de l'employé sélectionné. Basez-vous sur le diagramme ci-dessous.

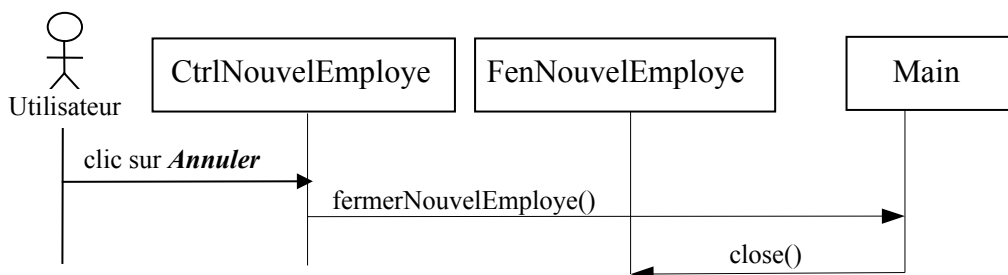


Note : les méthodes `afficherEmploye(e)` sont fournies. Elles ont le même nom mais il s'agit bien de deux méthodes différentes.

4.6 Le bouton Annuler

Dans chaque formulaire, il reste à programmer le clic sur le bouton Annuler. Le principe est identique à celui du bouton Fermer de la fenêtre "Liste des employés".

Par exemple, pour la fenêtre "Nouvel employé" cela donne :



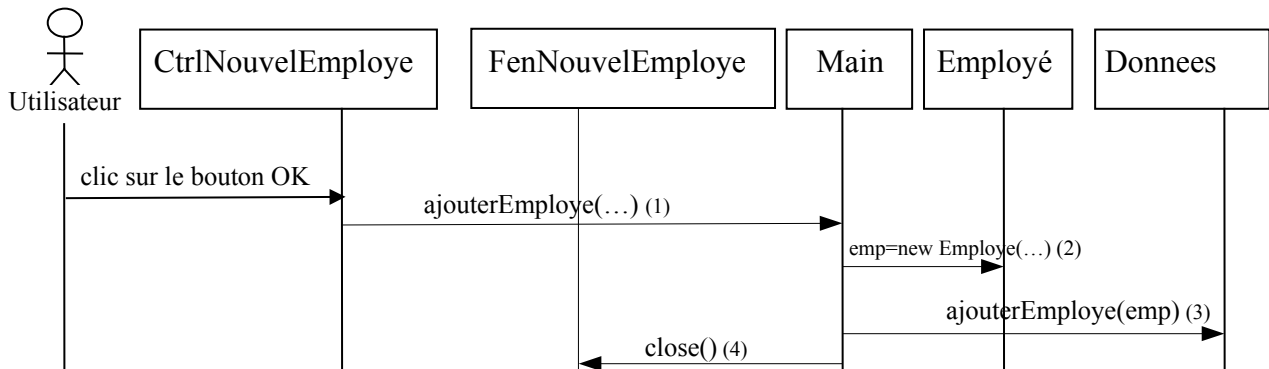
À FAIRE

Programmez ainsi les deux boutons Annuler.

5 – Gestion des données

5.1 Ajout d'un nouvel employé

Que doit-il se passer lorsque l'utilisateur valide la création d'un nouvel employé en cliquant sur OK dans la fenêtre "Nouvel employé" ? Là encore c'est la classe Main qui va contrôler la mise à jour des informations. Elle doit procéder à l'enregistrement du nouvel employé avant de fermer la fenêtre "Nouvel employé", comme montré sur ce diagramme :



- (1) la classe CtrlNouvelEmploye passe en paramètre chaque valeur saisie dans le formulaire de création, avec les éventuelles conversions de type,
- (2) la classe Main crée une nouvelle instance d'Employé à partir des valeurs reçues,
- (3) puis ajoute cette nouvelle instance à la liste des employés gérée par la classe Donnees,
- (4) et enfin ferme la fenêtre.

À FAIRE

En suivant le diagramme de séquence, programmez le clic sur le bouton OK de la fenêtre "Nouvel employé".

Note : dans la classe CtrlNouvelEmploye, la méthode `valider()` est déjà complétée. Vous devez seulement écrire la méthode `ajouterEmploye(...)` de la classe Main.

Pour aller plus loin

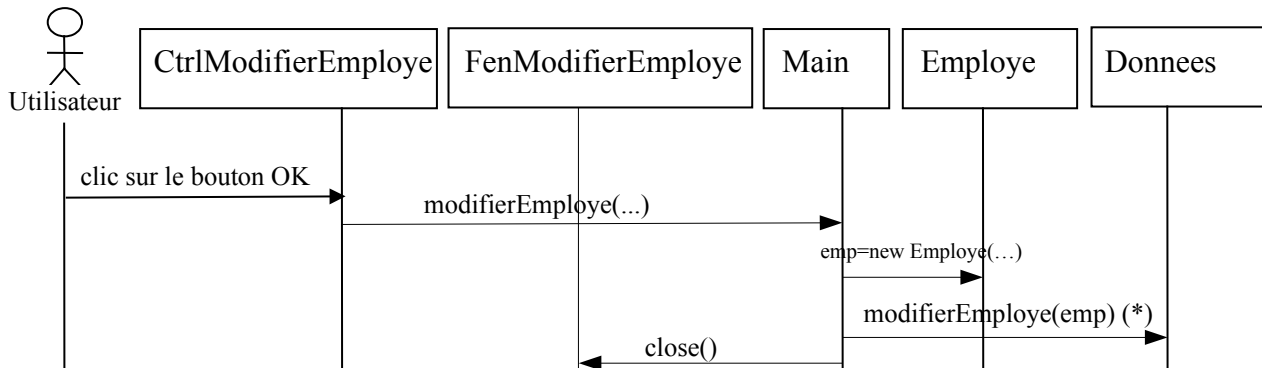
Au moment de la validation, votre programme pourra vérifier que le matricule (entier), le salaire et la prime éventuels (décimaux) sont bien des **données numériques**. Servez-vous de ces deux méthodes :

```
private boolean estDecimal(String str) {
    return str.matches("\\d*(\\.\\d+)?");
}

private boolean estEntier(String str) {
    return str.matches("\\d*");
}
```

5.2 Modification d'un employé

On retrouve le même principe lors de la validation de la modification d'un employé, lorsque l'utilisateur clique sur OK dans la fenêtre "Modifier un employé". Le diagramme de séquence est le suivant :



(*) Le nouvel employé remplace l'ancien.

À FAIRE

Programmez le clic sur le bouton OK de la fenêtre "Modifier employé".

Note : dans la classe CtrlModifierEmploye, la méthode clicOK() est déjà complétée. Vous devez seulement écrire la méthode modifierEmploye(...) de la classe Main.

Là encore, vous pourrez vérifier que le salaire et la prime éventuelle sont bien des **données numériques**.

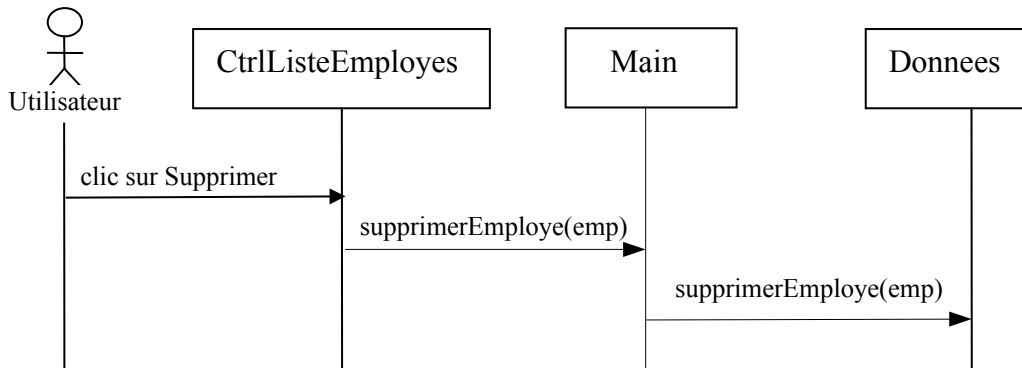
Pour aller plus loin

Pour le supérieur hiérarchique, vous pourrez ajouter cette vérification préalable à l'enregistrement de la modification : **l'employé ne peut pas être chef de lui-même**.

Comment pourrait-on empêcher cette erreur ?

5.3 Suppression d'un employé

Comme vu précédemment, cliquer sur le bouton *Supprimer* de la fenêtre "Liste des employés" déclenche l'apparition d'un message de confirmation. Si l'utilisateur confirme, la suppression doit s'effectuer conformément à ce diagramme :



À FAIRE

Dans la fenêtre "Liste des employés", programmez le clic sur le bouton Supprimer. Il faut bien sûr passer l'employé sélectionné en paramètre.

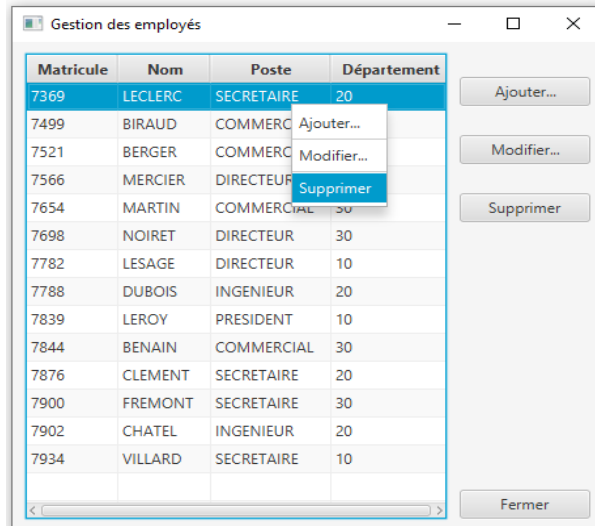
Pour aller plus loin

*Programmez cette vérification préalable à l'enregistrement de la suppression : **l'employé à supprimer ne doit pas être le supérieur hiérarchique de quelqu'un.***

6 – Compléments

6.1 Menu contextuel

Un menu contextuel va être ajouté au TableView (voir figure ci-dessous). Remarquez que les options du menu contextuel sont les mêmes (et dans le même ordre !) que les actions proposées par les boutons. Ils doivent fonctionner de la même manière : ils déclenchent les mêmes actions que les boutons et les options "Modifier" et "Supprimer" sont grisées quand aucun employé n'est sélectionné.



Pour créer un menu contextuel, il faut d'abord créer ses options (ses "items") :

```
private MenuItem optionAjouter = new MenuItem("Ajouter...");  
private MenuItem optionModifier = new MenuItem("Modifier...");  
private MenuItem optionSupprimer = new MenuItem("Supprimer");
```

Puis il faut créer la variable "menu contextuel" en lui associant les options. Remarquez que les options sont séparées par un trait (Separator) :

```
private ContextMenu menu = new ContextMenu( optionAjouter,  
                                             new SeparatorMenuItem(),  
                                             optionModifier,  
                                             new SeparatorMenuItem(),  
                                             optionSupprimer  
                                             );
```

C'est par un clic droit sur le TableView que s'affichera le menu ; il faut donc lui associer ce menu contextuel et c'est la méthode `setContextMenu(...)` qui le permet.

Enfin, il faut gérer l'événement "clic" sur chacune des trois options du menu contextuel, grâce à la méthode `setOnAction(...)`. Les actions à déclencher sont les mêmes que pour les boutons.

À FAIRE

Dans la classe contrôleur de la "Liste des employés", définissez le menu contextuel demandé et programmez le clic sur chacune des options.

***Note** : bien sûr, on peut aussi créer le menu contextuel avec SceneBuilder.*

6.2 Penser au double-clic !

Dans la fenêtre *Liste des employés*, double-cliquer sur un employé constitue un raccourci pour l'action "modifier employé".

À FAIRE

Dans la classe contrôleur de la "Liste des employés", déposez un écouteur de type `MouseClicked` sur le `TableView`. Dans le traitement de l'événement, il faut tester si le nombre de clics sur le bouton principal (celui de gauche en général) est égal à 2 pour reconnaître un double-clic :

```
if (e.getClickCount()==2 && e.getButton()==MouseButton.PRIMARY) ...
```

Il reste à tester que l'utilisateur n'a pas double-cliqué dans le vide, mais bien sur du texte :

```
if ( e.getClickCount()==2
    && e.getButton()==MouseButton.PRIMARY
    && e.getTarget() instanceof Text) ...
```

Note : au final, l'utilisateur disposera de trois moyens pour déclencher la modification d'un employé : bouton, menu contextuel et double-clic.