ensiie
école nationale supérieure d'informatique pour l'industrie et l'entreprise

Langages Objets:

JavaFX Unit Converter

Introduction

Le but de ce TP est de vous faire construire l'interface graphique ainsi que le contrôleur d'un Convertisseur d'unités en JavaFX.

Le convertisseur d'unité fonctionne comme suit :

- On choisit le type de grandeur : Par exemple *Longueur*, *vitesse*, *température*, etc.
- On choisit l'unité source : Par exemple *mètres*, *yard*, *mille*, *année lumière*, etc.
- On choisit l'unité destination.
- Puis l'on tape la valeur dans l'unité source et on obtient la valeur dans l'unité destination.

JavaFX implémente l'architecture MVC (Model / View / Controller). Nous allons nous concentrer pour ce TP sur :

- La partie *View* qui sera constituée d'un fichier FXML décrivant l'interface graphique et que vous pourrez éditer avec le logiciel SceneBuilder.
- La partie *Controller* qui sera constitué d'une classe `Controller` contenant les différents "callbacks"
- Nous aborderons la partie *Model* dans les TPs restants.

Une archive contenant un projet Eclipse à compléter se trouve dans `/pub/FISE_LAOB12/TP4/TP-Converter.zip`. Vous pourrez importer le projet, mais il faudra passer par les configurations indiquées en Annexe pour que le projet fonctionne.

User Interface (UI)

L'interface graphique à construire dans le fichier `ConverterFrame.fxml` ressemble à ceci :

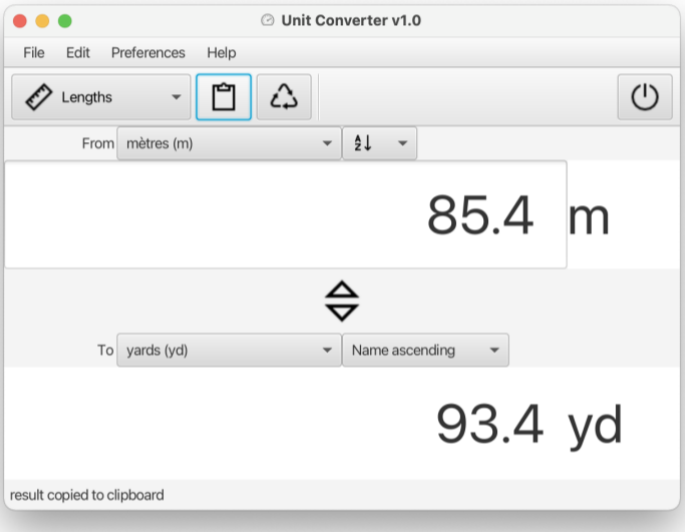

Figure 1 : Unit Converter UI

Structure du conteneur

Le conteneur principal de l'application est un `BorderPane` : Le contenu est donc composé de :

- Une VBox en haut contenant :
  - Une barre de menus (contenant des Menus
  - Une ToolBar contenant différents outils
- Une VBox au centre contenant les différents éléments nécessaires à la conversion :
  - Unités source et destination
  - Valeur de l'unité source à taper
  - Valeur dans l'unité destination
- Une HBox en bas contenant seulement de quoi afficher des messages d'information.

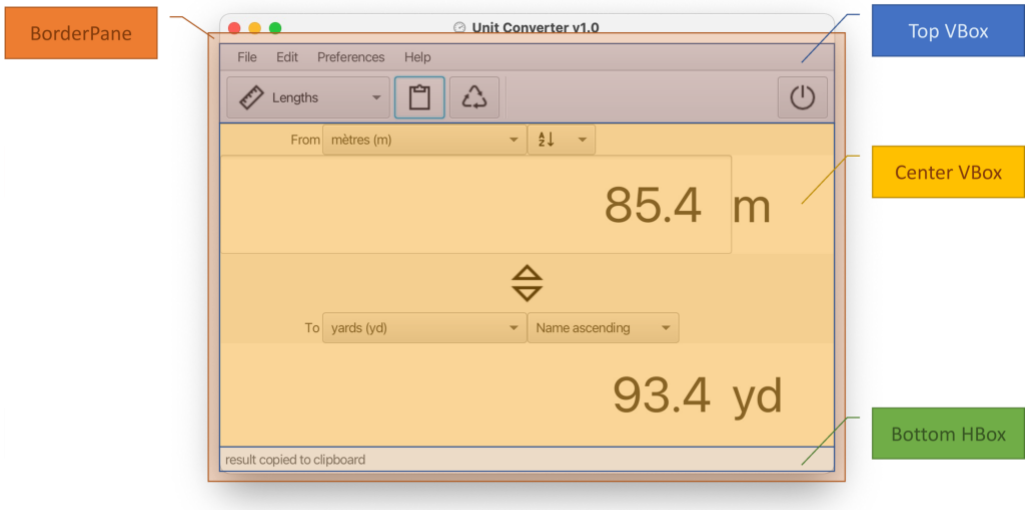

Figure 2 : Structure du contenu

Structure de la partie haute

La partie haute se compose de :

- Une Barre de Menu (MenuBar) contenant :
  - Des Menus contenant :
    - Des MenuItems, des RadioMenuItems ou bien d'autres Menus
- Une barre d'outils (ToolBar) contenant
  - Un ComboBox permettant de choisir le type de grandeurs des unités : `#measuresComboBox`.
    - Vous pourrez noter que pour obtenir des éléments de ComboBox décorés par des images, nous avons utilisé trois éléments dans le package `application.cells` :
      - Un fichier `MeasuresCell.fxml` contenant la description du contenu des cellules customisée constitué ici d'un Label augmenté d'un ImageView.
      - Une classe `MeasuresCell` héritant de la classe `CustomCell<MeasureType>` qui est une spécialisation de la classe `ComboBoxListCell<T>` qui représente les cellules customisées pour les ComboBoxes.
      - Une classe `MeasuresCellController` héritant de la classe abstraite `AbstractCustomCellController<MeasureType>` qui représente le contrôleur de ces cellules customisées de manière à :
        - Mettre en place le texte de la cellule.
        - Mettre en place l'icône de la cellule.

    - Vous aurez l'occasion de répéter ce procédé pour customiser d'autres cellules.
  - Un Button permettant de copier le résultat (formatté) dans le presse-papier (Clipboard) : `#copyButton`
  - A l'intérieur de ce bouton se trouve un ImageView de taille 32x32 contenant l'icône que l'on veut afficher dans le bouton. Si l'on souhaite n'afficher que cette image dans le bouton, il faudra sélectionner `GRAPHICS_ONLY` dans le champ "Content Display" du Button.
  - Les méthodes `onDisplayButtonsWith...` dans le la classe `Controller` permettent de modifier la présentation de divers Buttons dynamiquement.
  - Un Button augmenté d'un ImageView permettant d'effacer les valeurs des unités source et destination : `#clearButton`
  - Un espace constitué d'un HBox (contenant l'attribut `HBox.hgrow="ALWAYS"` que vous pourrez éditer manuellement) qui permet de "pousser" le prochain widget en bout de ligne.
  - Un Button augmenté d'un ImageView permettant de quitter l'application : `#quitButton`.

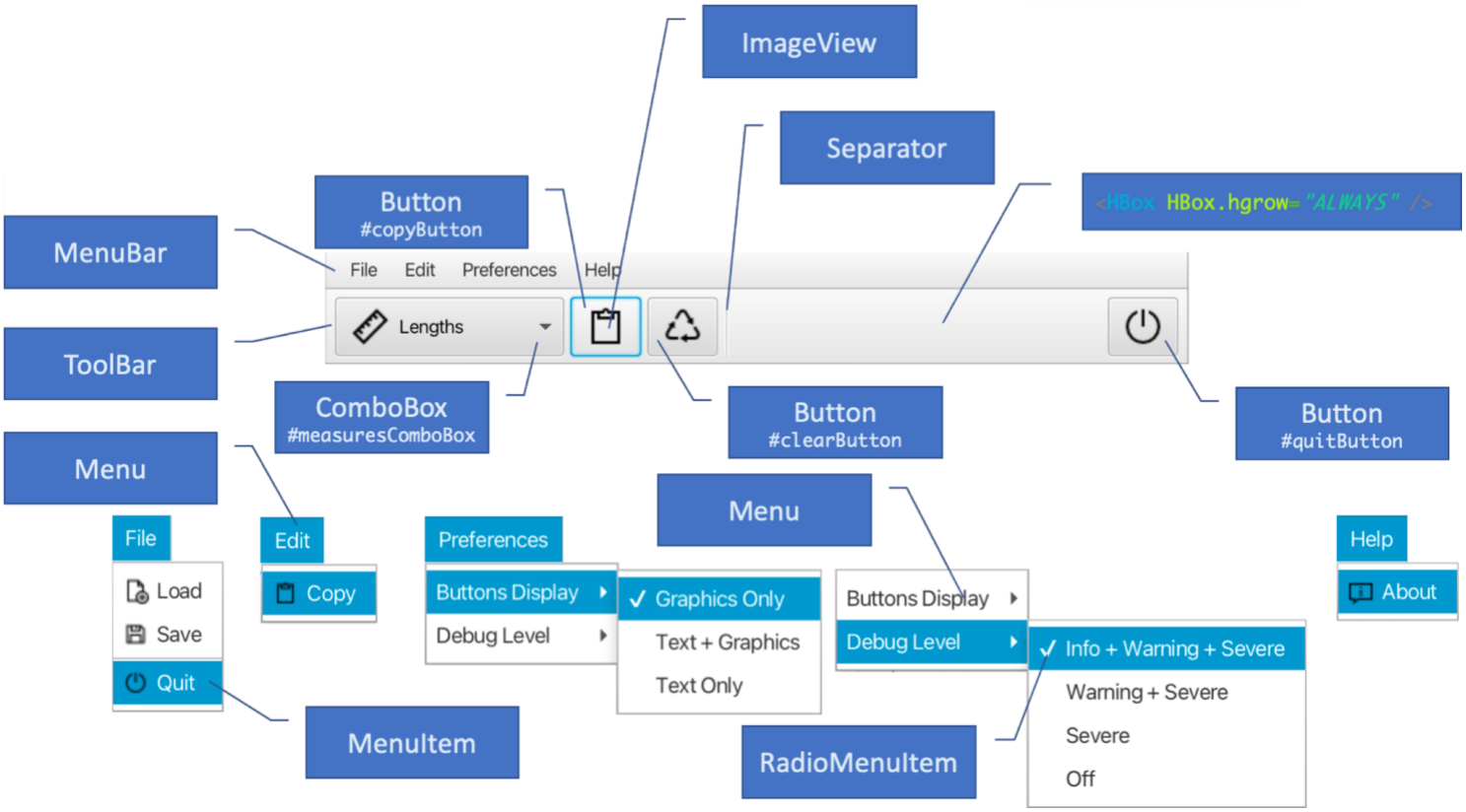

Figure 3 : Structure de la partie haute

Structure de la partie centrale

La partie centrale de l'application est composée d'une VBox contenant soit des widgets seuls, soit des groupes de widgets arrangés dans des HBox.

- La première rangée en haut contient un Label, un premier ComboBox permettant de choisir l'unité source ainsi qu'un second ComboBox permettant de choisir l'ordre des unités (voir la customisation de ces cellules).
- La seconde ligne contient un TextField permettant de taper la valeur dans l'unité source ainsi qu'un Label affichant l'abréviation de l'unité source.
- La troisième ligne ne contient qu'un Button augmenté d'un ImageView permettant (lorsque cela est possible) d'intervertir l'unité source et l'unité destination ainsi que leurs valeurs respectives.
- La quatrième ligne contient la même chose que la première mais pour l'unité de destination.

- La cinquième ligne contient un Label permettant d'afficher la valeur dans l'unité destination. Ce Label possède des réglages qui lui donnent une apparence similaire au TextField de la seconde ligne.

La partie basse de l'application ne contient qu'un HBox contenant lui-même un Label pour afficher des messages d'information sur la dernière action effectuée.

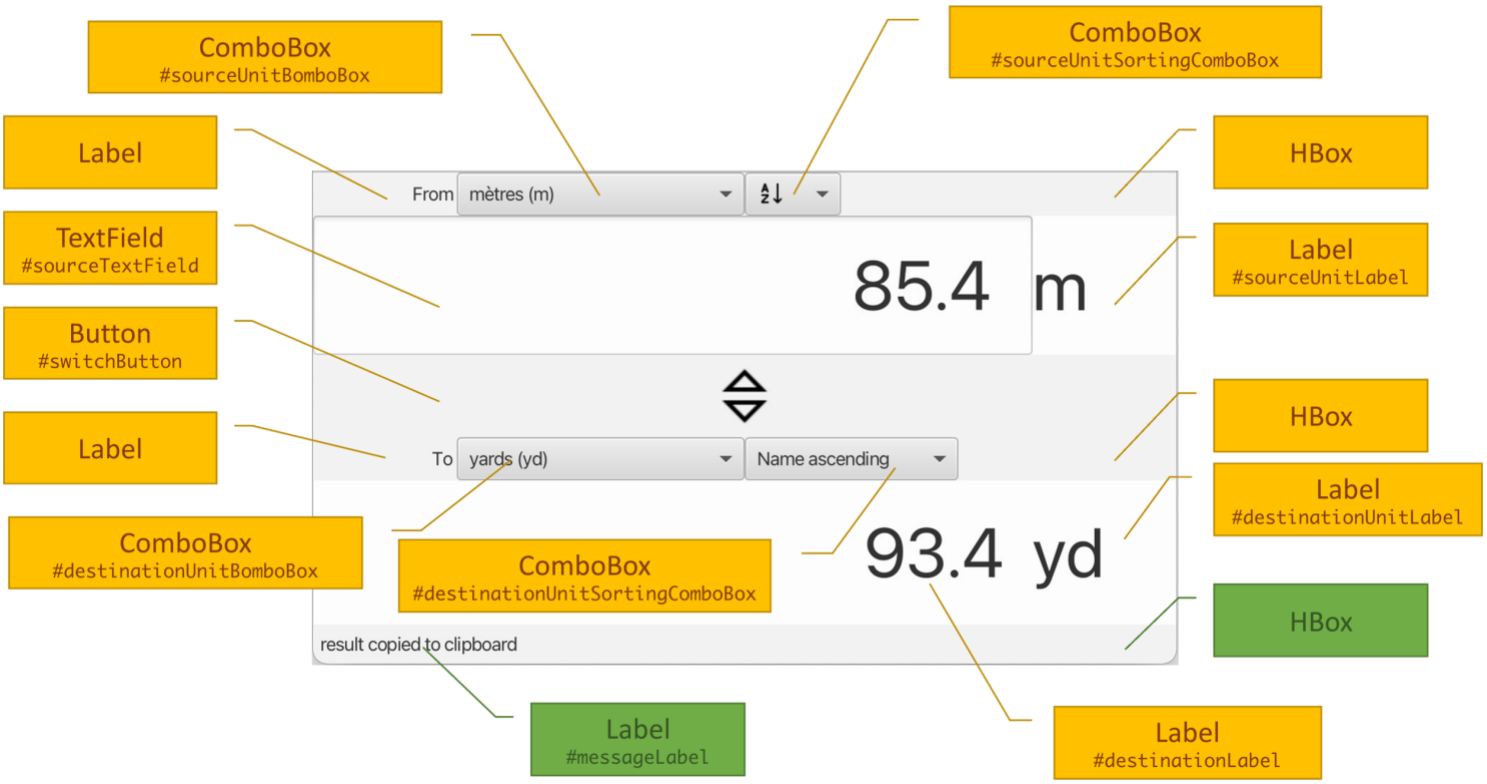

Figure 4 : Structure de la partie centrale

Actions (callbacks) associés aux éléments (widgets) de l'UI.

La classe `Controller` contient les callbacks associés à l'interface graphique décrite dans le fichier `ConverterFrame.fxml`. Ceux-ci sont annotés avec l'annotation `@FXML`. Exemple :

```
/**
 * Action to quit the application
 * @param event event associated with this action
 */
@FXML
public void onQuitAction(ActionEvent event)
{
    ...
}
```

Pour associer un callback à un élément d'UI, il suffit de préciser dans l'onglet **Code** (à droite) de SceneBuilder le callback à appeler dans la case "On Action" :

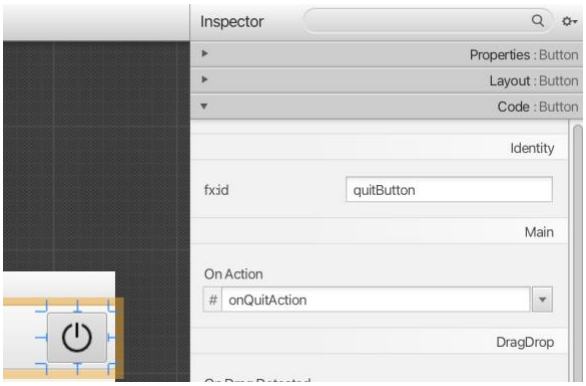

Figure 5 : fx:id et "OnACtion" dans SceneBuilder

Les actions associées aux éléments de menu sont décrites dans la figure suivante :

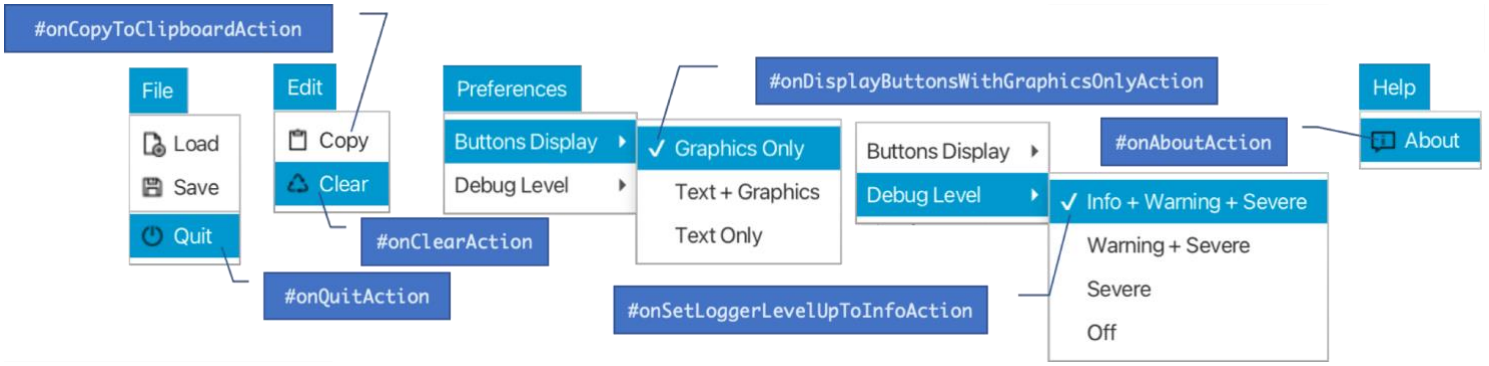

Figure 6 : Actions associées aux éléments de menus

Les actions associées aux éléments centraux de l'UI sont décrites dans la figure suivante :

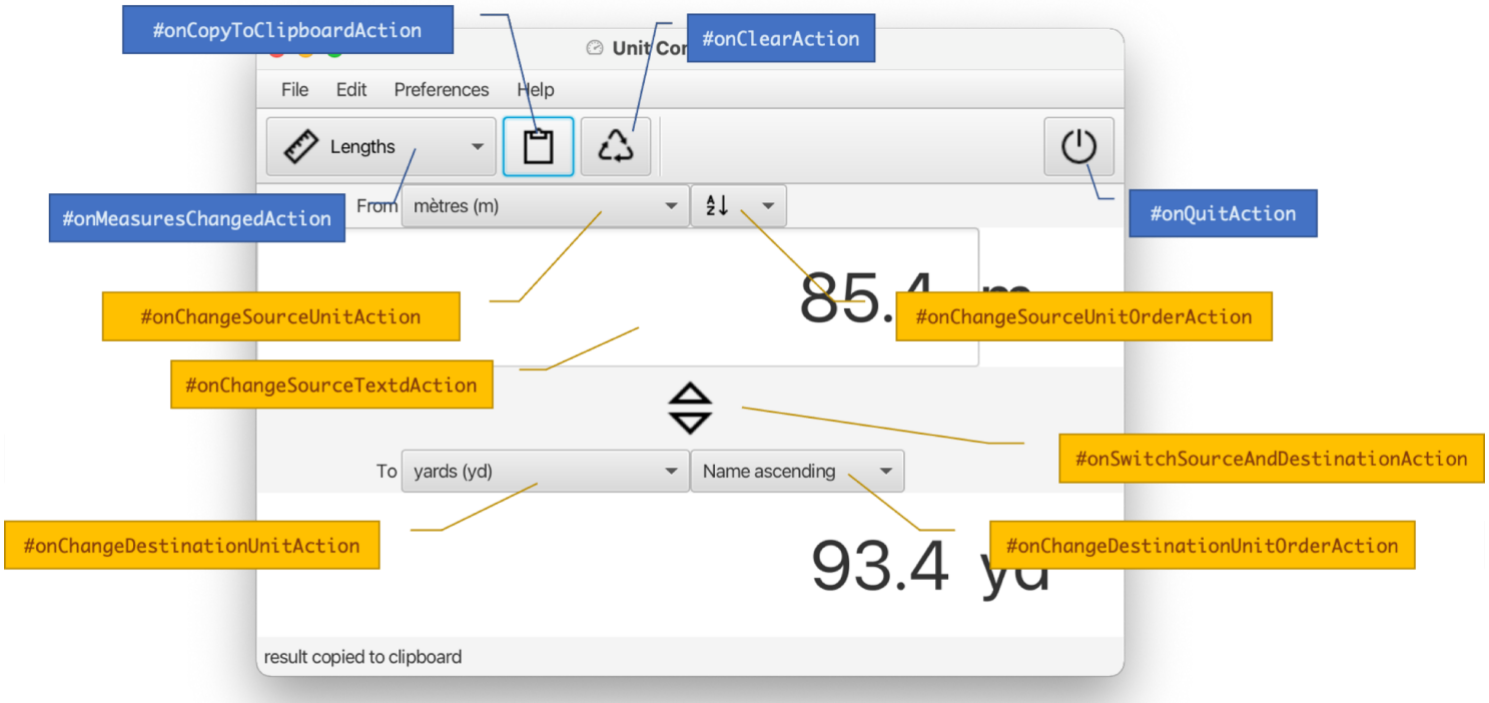

Figure 7 : Actions associées aux éléments de la partie centrale

Contenu de la classe `Controller`

La classe `Controller` contient :

- Les attributs FXML (annotés `@FXML`) qui reflètent le contenu de l'interface graphique décrite dans le fichier `ConverterFrame.fxml`. Par exemple:

```
/**
 * Toolbar ComboBox to choose the measures
 */
@FXML
private ComboBox<MeasureType> measuresComboBox;
```

- Les attributs non FXML comme le `Logger`, ou bien à terme notre modèle de données `Converter`.
- Un constructeur dans lequel on pourra initialiser les attributs non-FXML.
- Une méthode d'initialisation du contrôleur dans laquelle on pourra initialiser les attributs FXML :
  - On pourra par exemple remplir le contenu des ComboBoxs
  - On pourra lier des propriétés de l'UI avec des propriétés de notre futur modèle de données.

```
/**
 * Controller initialization to initialize FXML related attributes.
 * @param location The location used to resolve relative paths for the root
 * object, or null if the location is not known.
 * @param resources Resource Bundle containing translations resources for
 * the UI (or null)
 */
@Override
public void initialize(URL location, ResourceBundle resources)
{
    ...
}
```

- Les différentes méthodes associées aux actions déclenchées par les éléments de l'UI et référencées dans le fichier `ConverterFrame.fxml`. Par exemple :

```
/**
 * Action when Measures are changed
 * @param event event associated with this action
 */
@FXML
public void onMeasuresChangedAction(ActionEvent event)
{
    String content = measuresComboBox.getValue().toString();
    logger.info("Measures changed action triggered: Measures =  "
        + content);
    messageLabel.setText("Measures set to " + content);
    ...
}
```

  Dans laquelle on peut faire référence à `measuresComboBox` ou bien `messageLabel` puisqu'ils font partie des attributs FXML.

Travail à réaliser

ConverterFrame

- Éditez le fichier `ConverterFrame.fxml` avec SceneBuilder :
  - Ajoutez les différents widgets en n'oubliant pas :
    - De référencer les attributs FXML déjà présents dans la classe `Controller` dans l'onglet "Code" : champ *fx:id* lorsque cela est nécessaire.
    - De référencer la méthode déclenchée par une action sur un élément de l'UI toujours dans l'onglet "Code" : champ *On Action*. (Voir Figure 5 : fx:id et "OnACtion" dans SceneBuilderFigure 5 page 5)
- Configurez votre projet dans votre IDE pour utiliser JavaFX (voir Installation et utilisation de JavaFX).
- Lancez le programme principal : `application.Main`. Attention, s'il y a un problème dans la classe `Controller` le seul message que vous obtiendrez sera `"Can't load FXML file ..."` vous devrez alors débugger l'exécution pour trouver où se situe le problème.

Classe Contrôleur

- Lorsque l'UI dans `ConverterFrame.fxml` sera terminée, vous pourrez décommenter les `TODO` dans la méthode `initialize` de la classe `Controller`.
- Vous pourrez aussi commencer à personnaliser la classe `Controller` puisque c'est ce que nous ferons durant les séances suivantes.

Custom Cells

- Si vous souhaitez customiser le contenu des cellules des ComboBox `#sourceUnitSortingComboBox` et `#destinationUnitSortingComboBox` il vous faudra créer les éléments suivants dans le package application.cells :
  - Un fichier `SortOrderCell.fxml` contenant l'UI des cellules (typiquement un Label augmenté d'une ImageView). Vous pourrez vous inspirer de `MeasuresCell.fxml`.
  - Une classe `SortOrderCell extends CustomCell<SortOrder>` qui chargera l'UI définie dans `SortOrderCell.fxml`. Vous pourrez vous inspirer de la classe `MeasuresCell`.
  - Une classe `SortOrderCellController extends AbstractCustomCellController<SortOrder>` servant de contrôleur pour ces cellules customisées et contenant :
    - Les différentes Images à mettre en place dans l'ImageView `#iconView`.
    - Une implémentation de la méthode `void setIcon(SortOrder value)` pour mettre en place ces Images
    - L'UI décrite dans `SortOrderCell.fxml` doit référencer la classe `SortOrderCellController` dans l'onglet "Controller".

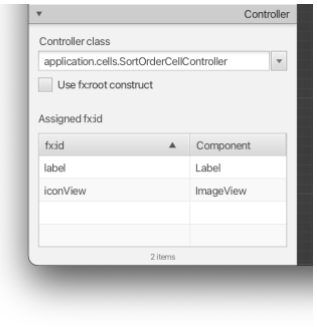

Figure 8 : référence au contrôleur dans un fichier FXML

  - Dans la méthode `initialize` de la classe `Controller`, il faudra ensuite mettre en place sur `#sourceUnitSortingComboBox` et `#destinationUnitSortingComboBox`:
    - `....setButtonCell(new SortOrderCell());`

- `....setCellFactory(combobox -> new SortOrderCell());`

Annexes

Installation et utilisation de JavaFX

D'après le tutoriel Getting Started with JavaFX

Depuis Java 11, JavaFX ne fait plus partie de la distribution standard de Java afin de réduire la taille du SDK. Il faut donc installer JavaFX séparément.

1. A l'école, la version 20 de JavaFX pour Linux est installée dans `/pub/FISE_LAOB12/javafx-sdk-20/lib`. Néanmoins, si vous souhaitez installer JavaFX sur votre machine, vous pouvez le télécharger à l'adresse https://gluonhq.com/products/javafx/. Veillez à bien choisir l'archive pour votre système d'exploitation (*Linux*, *MacOS* ou *Windows*), votre architecture (a priori *x64* si vous avez un processeur Intel et *aarch64* autrement) ainsi que le type d'archive, ici *SDK*.
2. Une fois JavaFX installé, il faut préciser à Eclipse où se trouve JavaFX en
   - créant une nouvelle bibliothèque utilisateur ("User Library") : Preferences -> Java -> Build Path -> User Libraries -> New
   - que vous pouvez appeler "JavaFX" (vous pouvez la considérer comme une System Library)
   - Vous allez ensuite ajouter à cette bibliothèque les "external JARs" qui se trouvent à l'endroit où JavaFX est installé. En l'occurrence ici : `/pub/FISE_LAOB12/javafx-sdk-20/lib`
   - Vous pourrez alors ajouter cette bibliothèque à votre projet JavaFX : Clic droit à la racine de votre projet : Properties -> Java Build Path -> Onglet Libraries -> Add Library (à Classpath) -> User Library -> Next -> choisissez la bibliothèque JavaFX que vous avez créée à l'étape précédente.
3. Pour pouvoir lancer un programme principal Java utilisant JavaFX (voir la classe `Main` dans le package `application`), Il ne suffit **PLUS** d'un clic droit sur la classe `Main` -> Run As … -> Java Application : vous obtiendrez une erreur car le run-time java ne sais pas où se trouve JavaFX. Il faut donc créer une **Run Configuration** : clic droit sur la classe `Main` -> Run As … -> Run Configurations …
   - Double cliquez sur "Java Application" dans la colonne de gauche pour créer une nouvelle configuration que vous pourrez appeler "Converter" puis dans l'onglet "Arguments" de la partie droite :
     - Ajoutez les "VM arguments" : `--module-path ${PATH_TO_FX} --add-modules javafx.controls,javafx.fxml`
     - Avec `${PATH_TO_FX}` une variable pointant vers `/pub/FISE_LAOB12/javafx-sdk-20/lib`
     - **Désélectionnez** la case "Use the -XStartOnFirstThread argument when launching with SWT" si elle est présente.

Les instructions mentionnées ici sont tirées des tutoriels originaux de JavaFX pour tous les IDEs :

- IntelliJ
- NetBeans
- Eclipse
- Visual Studio Code
- Suivez les instructions pour les *Non-modular projects*.

Installation de SceneBuilder

A l'école SceneBuilder est installé dans `/opt/scenebuilder`. Vous pourrez donc le lancer avec la commande : `/opt/scenebuilder/bin/SceneBuilder`

Si vous souhaitez installer SceneBuilder sur votre machine, celui-ci est téléchargeable à l'adresse https://gluonhq.com/products/scene-builder/. Il n'a (a priori) pas besoin de JavaFX car il contient son propre runtime Java + JavaFX.

Plugins JavaFX pour votre IDE

Si vous utilisez Eclipse vous pourrez installer le plugin e(fx)clipse pour utiliser SceneBuilder directement depuis eclipse. Suivez cette procédure.

Vous pouvez tout à fait utiliser Scenebuilder sans connexion avec Eclipse, néanmoins, chaque fois que vous éditerez un fichier fxml avec Scenebuilder il faudra **impérativement** rafraîchir votre projet Eclipse (F5) avant de lancer l'application, sans quoi le fichier fxml mis à jour ne sera pas pris en compte.