

# Conception d'applications – L3 informatique – UBO

## Utilisation de Java FX

Si vous souhaitez réaliser une interface utilisateur graphique pour votre application de gestion d'association, vous pouvez utiliser JavaFX.

JavaFX est une librairie de composants graphiques (des widgets) qui permettent de construire une interface graphique et de lier ses éléments (listes, menus, boutons ...) au code Java de votre application.

JavaFX était officiellement intégré au JDK (Java Development Kit) jusqu'à la version 1.8. Ensuite il en a été retiré et doit être installé à part dans les versions récentes de Java. Néanmoins, cette façon de faire pose quelques problèmes alors nous allons continuer à utiliser la version de Java 1.8.

### 1. Installation de Java 1.8 (sur votre ordinateur personnel)

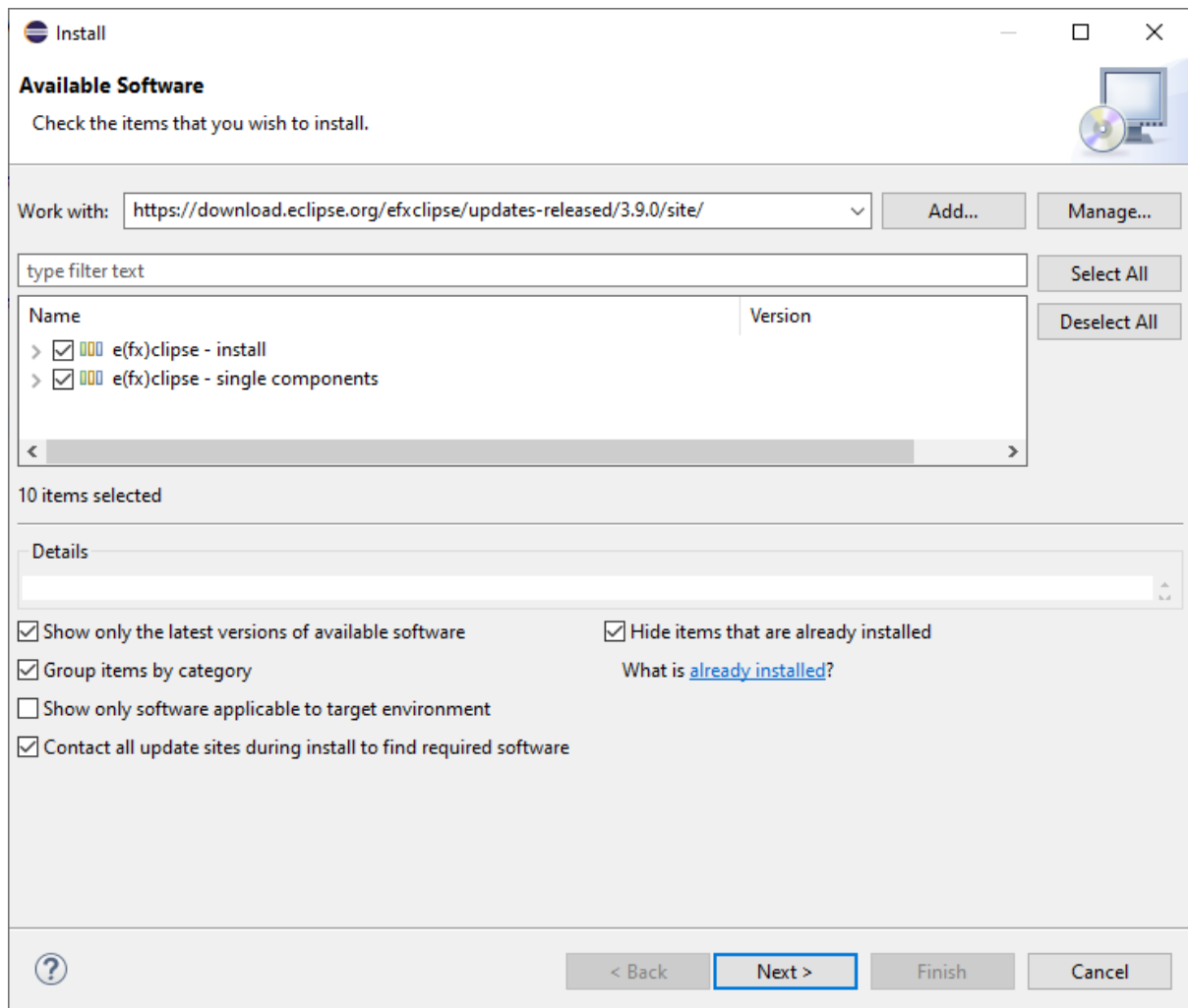
Si vous utilisez les machines des salles de TP, tout est installé. Mais si vous utilisez votre machine personnelle, il faut installer deux choses :

- Le JDK 1.8 que vous trouverez à cette adresse :  
<https://www.java.com/fr/download/manual.jsp>
- En option, rajouter des fonctionnalités Java FX en installant e(fx)clipse.

Pour installer e(fx)clipse, ouvrez Eclipse, allez dans le menu *Help -> Install New Software* et dans le champ *work with* entrez :

<https://download.eclipse.org/efxclipse/updates-released/3.9.0/site/>

Cochez les deux cases "e(fx)clipse" et cliquez sur *Next* pour lancer l'installation comme sur la fenêtre suivante :

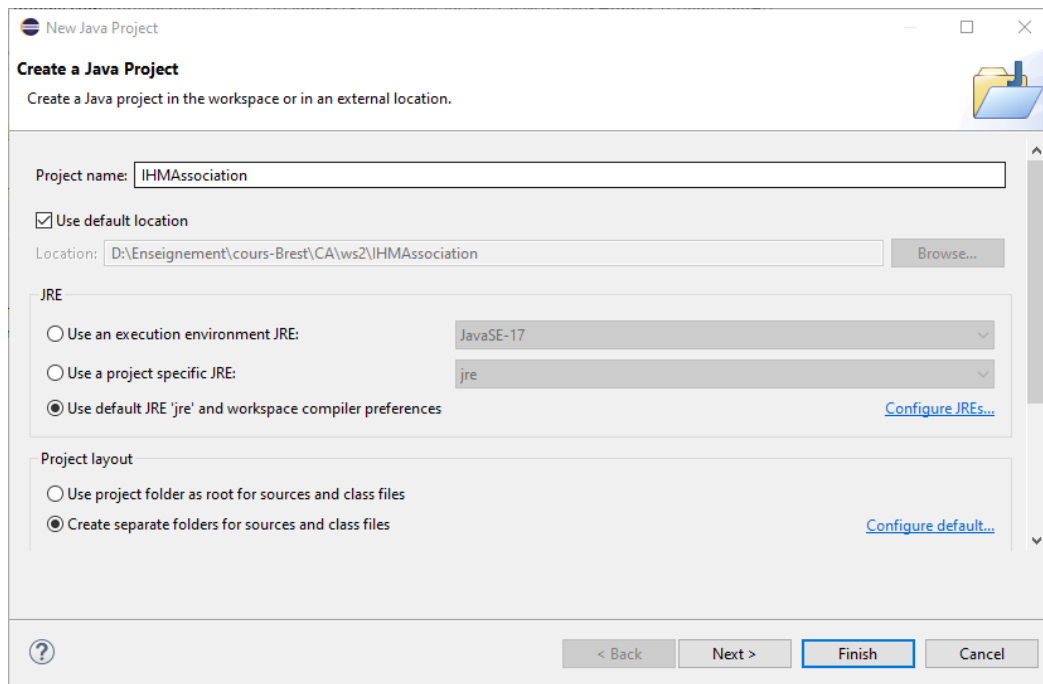


## 2. Création d'un projet Java 1.8 (pour le chef de projet)

Quand vous créez un projet Java, par défaut Eclipse utilise Java 17 qui est installé avec Eclipse. Il faut explicitement demander à Eclipse d'utiliser Java 1.8.

### 2.1 Initialisation du nouveau projet

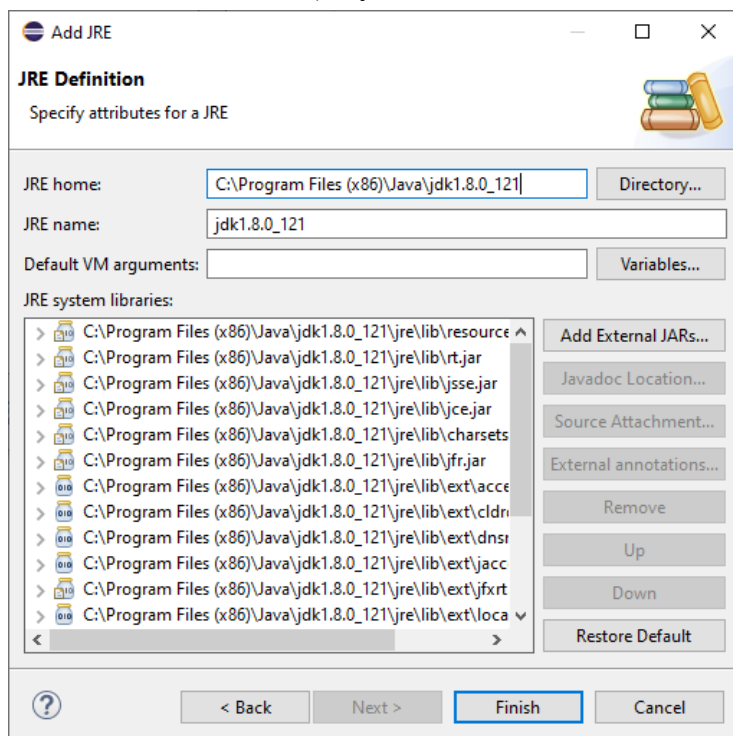
Créez un nouveau projet Java (**dans le même workspace que votre premier projet**) et cochez la case "Use default JRE ..." puis cliquez sur le lien "Configure JREs..." :



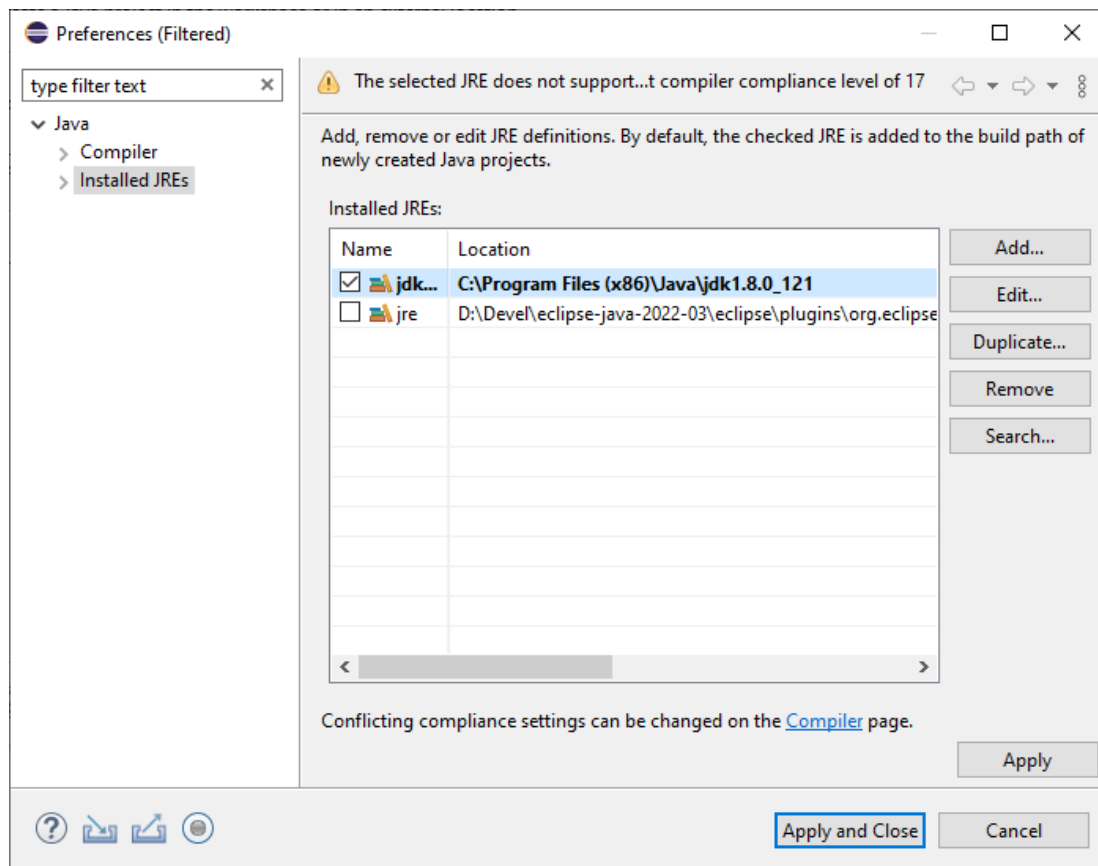
Sur la fenêtre suivante, cliquez sur le bouton “Add...”, sélectionnez “Standard VM” puis cliquez sur “Next”.

Dans la fenêtre qui s’ouvre, mettez le lien vers l’installation de Java 1.8 dans le champ “JRE Definition”. Sur les machines des salles de TP, sélectionnez le répertoire : C:\Program Files\Java\jdk1.8.0\_202

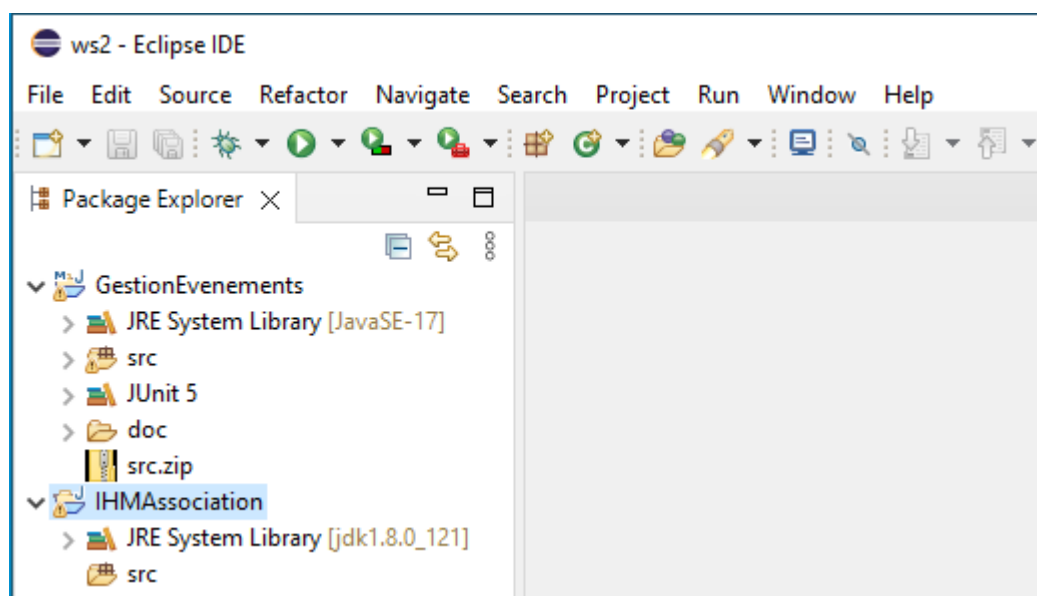
Vous devez obtenir ceci (moyennant la version exacte et le chemin sur le disque) :



Cliquez sur “Finish” puis dans la fenêtre qui se réaffiche, cochez bien la case correspondant au JDK 1.8 puis cliquez sur “Apply and Close” :



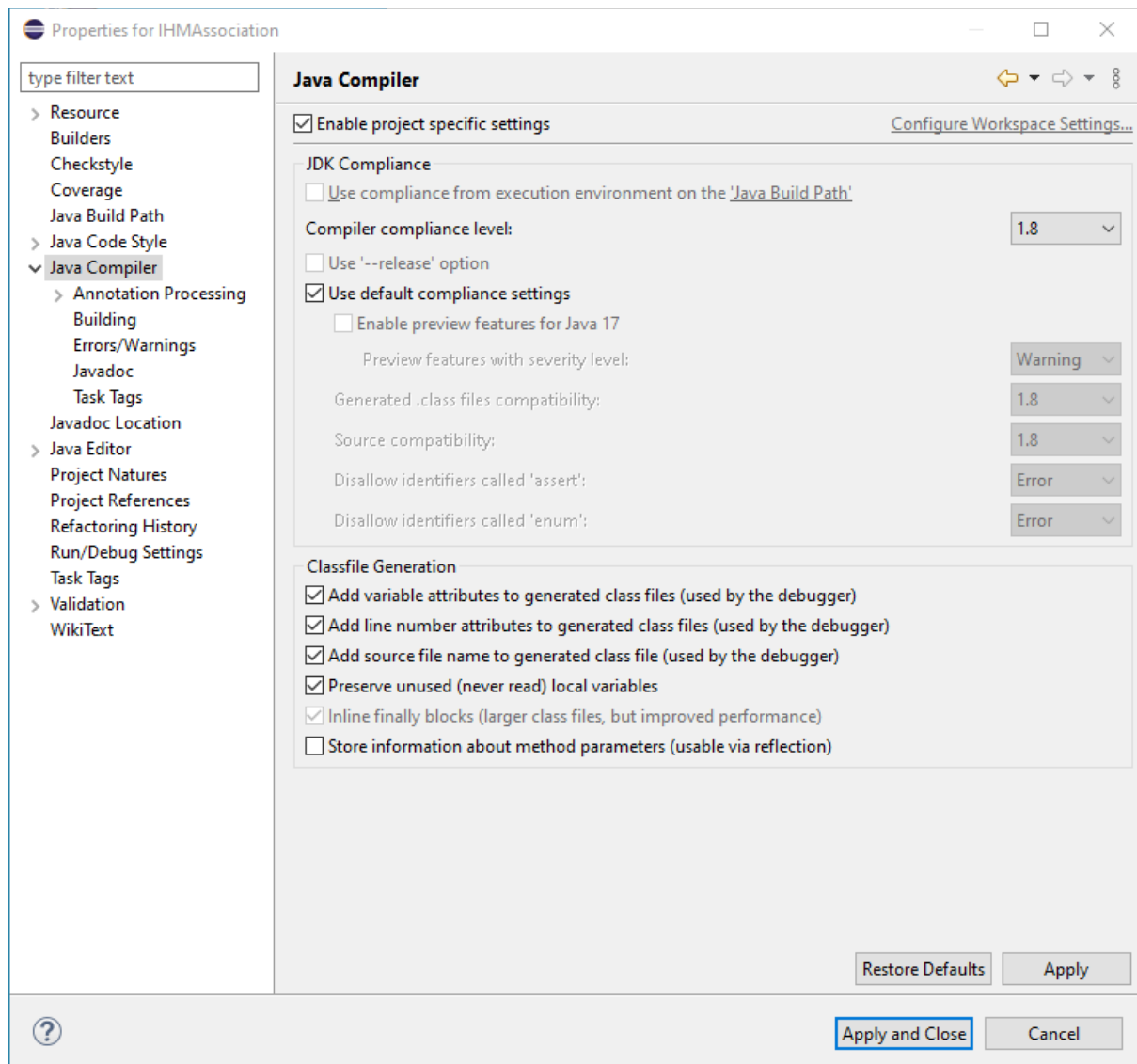
Arrivé à ce point là, vous devez avoir 2 projets Java, le premier avec le code sur l’association en Java 17 et le nouveau en Java 1.8, faites bien attention à ce que l’on voit “[jdk1.8.XXX]” dans la JRE du nouveau projet :



## 2.2 Configuration de la compilation en Java 1.8

Il reste une dernière manipulation à faire : préciser à Eclipse de compiler votre code en Java 1.8 car il compilera par défaut en Java 17.

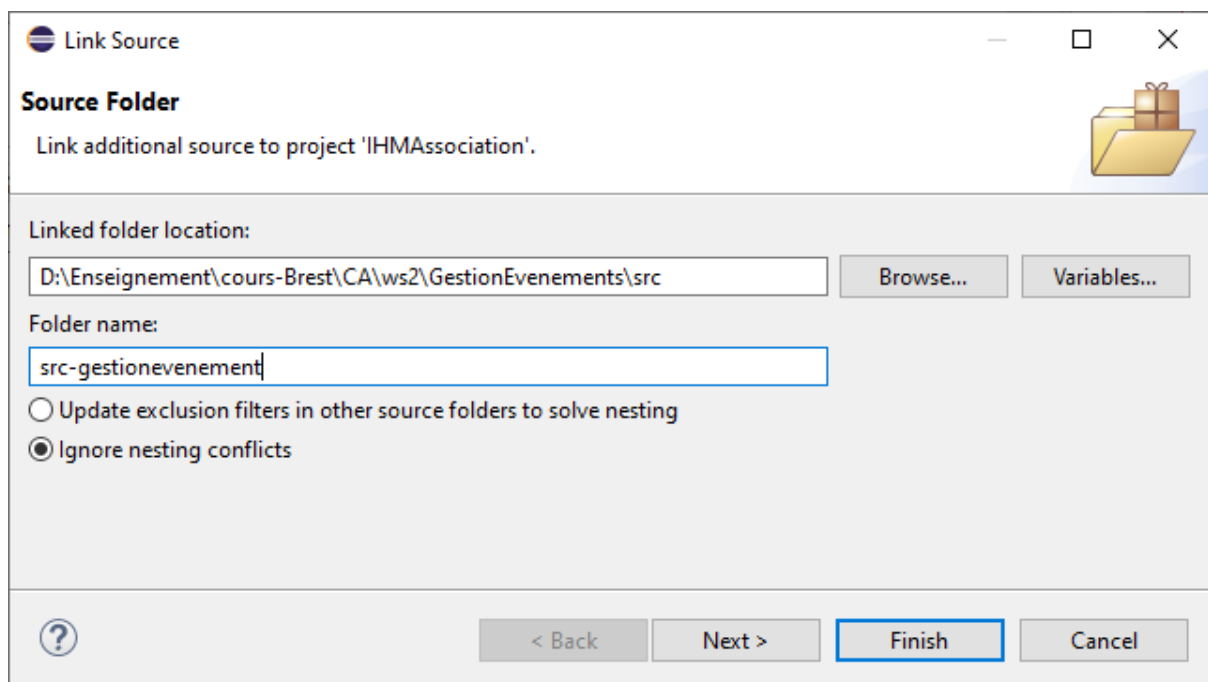
Pour cela, ouvrez les propriétés du projet, allez dans le menu “Java Compiler”, cochez la case “Enable project specific settings” et pour “Compiler compliance level:”, choisissez 1.8 et cliquez sur “Apply and Close” :



## 2.3 Liens avec les sources de l'autre projet

A ce stade, vous avez un projet en Java 1.8 fonctionnel mais vide. Il faut maintenant y intégrer les sources de votre premier projet où vous avez implémenté les classes gérant l'association.

Allez dans les propriétés de votre nouveau projet, dans "Java Build Path" et dans l'onglet "Source", cliquez sur "Link Source..." et sélectionnez le répertoire "src" de votre premier projet en lui donnant un nom différent ("src-gestionevenement" ici) sinon vous aurez un conflit de nommage sur les répertoires "src" :



Notez que les sources ne sont pas copiées, vous avez simplement créé un lien vers les classes Java de l'autre projet pour pouvoir utiliser ses classes et les instancier dans votre nouveau projet.

## **2.4 Ajout du projet dans le dépôt Git (pour le chef de projet)**

Maintenant que votre nouveau projet Java 1.8 est créé, rajoutez le dans le dépôt Git. Cela se fait directement car vous avez créé le projet dans le workspace qui est le repository local de votre projet Git. Suivez les mêmes instructions que dans le document sur Git : ajoutez dans le dépôt le répertoire du projet, son répertoire “src” et ignorez les fichiers .classpath et le répertoire .settings (gardez le fichier .project).

## **2.5 Pour les autres développeurs : récupération du projet**

Les autres développeurs récupèrent le nouveau projet Java en faisant un Pull sur le dépôt Git. En l'état, il n'est pas fonctionnel car il faut notamment lui préciser d'utiliser une JRE. Il faut lui préciser d'utiliser Java 1.8 : allez dans les propriétés du projet, ouvrir l'onglet “Libraries”, cliquez sur “Add Library”, sélectionnez “JRE System Library” et remplissez les champs comme expliqué dans la section 2.1 pour utiliser Java 1.8. De même, il faut suivre les instructions de la section 2.2 pour préciser à Eclipse de compiler en Java 1.8. Vous aurez peut-être besoin de lier les sources de l'autre projet comme expliqué dans la section 2.3 si vous ne voyez pas son répertoire source dans le nouveau projet.

### 3. Utilisation de JavaFX

La création d'une interface en JavaFX se fait de 2 façons : soit en codant directement dans le code Java la forme et le contenu des fenêtres, soit en passant par un fichier de description de l'interface en XML (format FXML précisément) puis en chargeant ce fichier. C'est cette seconde option que nous allons utiliser.

Concrètement, nous vous fournissons une interface définie par le logiciel Scenebuilder (<https://gluonhq.com/products/scene-builder/>) qui permet de concevoir la maquette d'une interface puis d'en générer le fichier FXML ainsi que le squelette du code du contrôleur : c'est une classe Java qui définit des attributs correspondant aux éléments de l'interface et des méthodes qui sont appelées quand on manipule l'interface. Par exemple, on peut référencer par un objet de type TextField un champ de texte de l'interface dont on peut récupérer le contenu ou recevoir par un appel de méthode qu'un bouton de l'interface a été cliqué.



## 3.1 Interface Java FX proposée

L'interface graphique Java FX proposée est la suivante :

The screenshot shows a JavaFX application window titled "Association". It features a menu bar with "Fichier" and "Aide". The main interface is divided into two panels: "Membres" (Members) on the left and "Evénements" (Events) on the right. The "Membres" panel includes input fields for "Nom", "Prénom", "Adresse", and "Age", followed by buttons "Nouveau", "Valider", "Supprimer", "Evénements membre", and "Evénements futurs". Below these is a label "La liste affiche : ..." and a large empty list box. At the bottom of the panel are buttons "Afficher le membre sélectionné", "Afficher tous les membres", "Inscrire membre à événement", and "Désinscrire membre à événement". A "Message :" label is at the very bottom. The "Evénements" panel includes input fields for "Nom", "Lieu", "Date", "Heure", "Durée", and "Max participants", followed by buttons "Nouveau", "Valider", "Supprimer", and "Afficher les participants". Below these is a label "La liste affiche : ..." and a large empty list box. At the bottom of the panel are buttons "Afficher l'événement sélectionné", "Afficher tous les événements de l'association", and "Afficher tous les événements futurs de l'association".

L'interface est volontairement simpliste et en conséquence pas des plus ergonomiques afin de limiter le nombre et la complexité des composants JavaFX à utiliser. Par exemple, partout sont utilisées des chaînes de caractères en entrée là où on pourrait utiliser un composant calendrier pour choisir la date ou une liste déroulante d'entiers pour l'âge d'un membre.

Si vous souhaitez implémenter une interface plus complexe, vous pouvez le faire.

Les idées générales qui guident l'interface sont les suivantes :

- La partie de gauche décrit les membres avec des champs pour éditer les informations d'un membre et gérer une liste de membres.
- La partie de droite décrit les événements avec des champs pour éditer les informations d'un événement et gérer une liste d'événements.
- En fonction des clics sur les boutons, et donc des actions choisies par l'utilisateur, les listes contiendront soit des informations globales à l'association (par exemple tous les membres) soit des informations spécifiques à un membre ou un événement (par exemple tous les membres inscrits à l'événement). C'est cette partie là qui est la

moins ergonomique mais permet de simplifier l'interface en évitant d'avoir des listes supplémentaires spécifiques aux différentes actions.

- Afin de savoir à quoi correspond une liste à un instant donné, la ligne juste au-dessus le précisera : concrètement, le “...” sera remplacé par un texte comme “tous les membres de l'association” ou “les événements futurs de Luke Skywalker”.
- Le champ “remarque” en bas à gauche permettra d'informer l'utilisateur du résultat de sa dernière action (on pourra à la place préférer ouvrir une fenêtre popup).

### 3.2 Comportement attendu de l'interface

Menu fichier :

- *Nouveau* : réinitialise l'association (efface tous les événements et membres chargés en mémoire)
- *Charger* : charge les membres et les événements de l'association à partir d'un fichier (dont on pourra optionnellement choisir l'emplacement). Une fois chargé, les deux listes affichent tous les membres et tous les événements.
- *Sauvegarder* : sauvegarde les membres et les événements de l'association dans un fichier (dont on pourra optionnellement choisir l'emplacement).
- *Quitter* : ferme l'application.

Menu aide :

- *A propos* : affiche quelques informations sur votre application.

Côté Membres, boutons :

- *Nouveau* : efface le contenu des 4 champs d'un membre afin de rajouter un nouveau membre
- *Valider* : lit les 4 champs d'un membre. Si le membre existait déjà, ses informations personnelles sont mises à jour, sinon, un nouveau membre est créé.
- *Supprimer* : efface de la liste des membres, le membre dont les informations sont affichées.
- *Événements membre* : affiche dans la liste de droite tous les événements du membre dont les informations sont dans les champs au-dessus.
- *Événements futurs* : idem mais en affichant uniquement les événements à venir pour le membre.
- *Afficher le membre sélectionné* : si un membre est sélectionné dans la liste, affiche ses informations personnelles dans les 4 champs en haut de la fenêtre.
- *Afficher tous les membres* : affiche dans la liste tous les membres de l'association.
- *Inscrire membre à événement* : si un membre est sélectionné dans la liste de gauche et un événement est sélectionné dans la liste de droite, le membre est inscrit à cet événement (dans la limite des places disponibles).
- *Désinscrire membre à événement* : si un membre est sélectionné dans la liste de gauche et un événement est sélectionné dans la liste de droite, le membre est désinscrit à cet événement.

si mbr suppr = pres  
↳ pres = null  
ou  
suppr anul

Côté Événements, boutons :

- *Nouveau* : efface le contenu des champs d'un événement afin de rajouter un nouvel événement
- *Valider* : lit les champs d'un événement. Si l'événement existait déjà, ses informations sont mises à jour, sinon, un nouvel événement est créé.
- *Supprimer* : efface de la liste des événements l'événement dont les informations sont affichées.
- *Afficher les participants* : affiche dans la liste de gauche, les participants inscrits à l'événement dont les informations sont affichées.
- *Affiche l'événement sélectionné* : si un événement est sélectionné dans la liste, affiche ses informations dans les champs en haut de la fenêtre
- *Afficher tous les événements de l'association* : afficher dans la liste tous les événements de l'association.
- *Afficher tous les événements futurs de l'association* : idem mais avec les événements à venir de l'association.

### 3.3 Fichiers fournis

Trois fichiers sont fournis et sont placés dans un package nommé "ui" à mettre dans le répertoire "src" de votre projet Java :

- *association.fxml* : c'est la description de l'interface graphique et de ses composants (ce fichier est généré par Scenebuilder quand vous sauvez un projet Scenebuilder)
- *MainIHM.java* : fichier exécutable qui affiche l'interface graphique. Vous n'avez rien à modifier dans ce fichier.
- *Controler.java* : c'est la classe Java qui fait le lien entre les éléments de l'interface et le code Java et que vous avez à compléter (Scenebuilder peut vous générer cette classe via *View -> Show Sample Controler Skeleton*).

Il y a dedans :

- Un attribut pour chacun des éléments à modifier ou lire (champ de texte, label, liste ...)
- Une méthode nommée "actionXXX" par bouton : quand l'utilisateur clique sur un bouton, on exécute la méthode associée au bouton. Les noms des méthodes correspondent directement au nom des boutons sur l'interface.
- Une méthode "initialize" qui est appelée au lancement de l'interface : c'est là que vous allez initialiser les objets de l'association.

Vous aurez donc à implémenter dans la classe "Controler.java" :

- La méthode "initialize"
- Chacune des 21 méthodes d'actions pour les boutons ou les choix de menus (les paramètres *ActionEvent* n'ont pas d'utilité pour le code à écrire).

**Attention !** Quand vous générez le squelette du code du contrôleur sous Scenebuilder, il ne garde pas le code que vous avez déjà pu modifier.

### 3.4 Quelques éléments de programmation

La Javadoc de l'API JavaFX est à cette adresse : <https://openjfx.io/javadoc/19/>

TextView est une zone de texte dont on peut lire et modifier le contenu :

```
entreeNomMembre.setText("Luke Skywalker");  
String val = entreeNomMembre.getText();
```

Label est un texte affiché (non éditable par l'utilisateur) dont on peut modifier le contenu :

```
labelListeAfficheeEvt.setText("tous les événements");
```

On peut de la même façon modifier le contenu d'un TextArea (zone de texte sur plusieurs lignes) : pour écrire sur plusieurs lignes, on insère des “\n” dans la chaîne à afficher.

ListView est une liste dont le contenu se modifie en récupérant les items qui forment une liste Java :

```
listeMembres.getItems().add("Luke Skywalker");  
listeMembres.getItems().add("Dark Vador");  
listeMembres.getItems().remove("Dark Vador");  
listeMembres.getItems().remove(0);
```

Note sur ListView : dans le code proposé, le typage des données de la liste est String. On peut mettre une classe quelconque et donc on pourrait directement avoir une `ListView<Membre>` ou `ListView<Evenement>`. Ce serait plus pratique pour gérer les éléments mais dans ce cas, ce qui est affiché dans la liste pour un objet, c'est ce que retourne la méthode `toString()` qui affiche trop de choses dans notre cas (les champs au-dessus de la liste sont là pour donner tous les détails sur le contenu des objets).

Le code suivant récupère l'élément sélectionné dans une liste puis efface tout le contenu de la liste :

```
String s = listeMembres.getSelectionModel().getSelectedItem();  
listeMembres.getItems().clear();
```

### 3.5 Lancement du programme indépendamment d'Eclipse

Générez le JAR exécutable correspondant au `MainIHM.java` comme expliqué dans l'autre document. Pour lancer l'exécutable, il faut bien faire attention d'utiliser le `java.exe` du JDK 1.8, ce qui donnera en ligne de commande :

```
> "C:\Program Files\Java\jdk1.8.0_202\bin\java.exe" -jar CAMonAssociation.jar
```

## 4. Utiliser Java FX avec Java 17

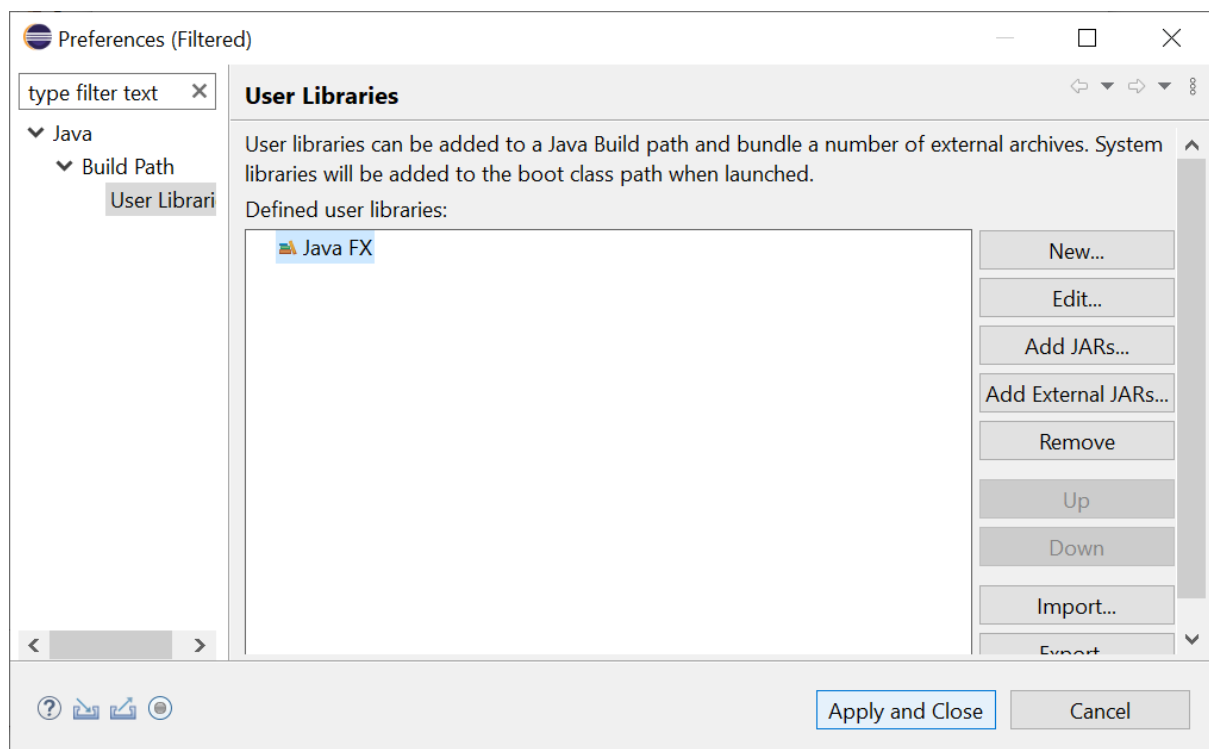
Pour utiliser Java FX avec un Java plus récent que Java 1.8, il faut télécharger Open JFX : <https://openjfx.io/>

Décompresser l'archive quelque part sur votre disque. Dans le reste des explications, on supposera que le SDK d'Open JFX est dans C:\Devel\javafx-sdk-19

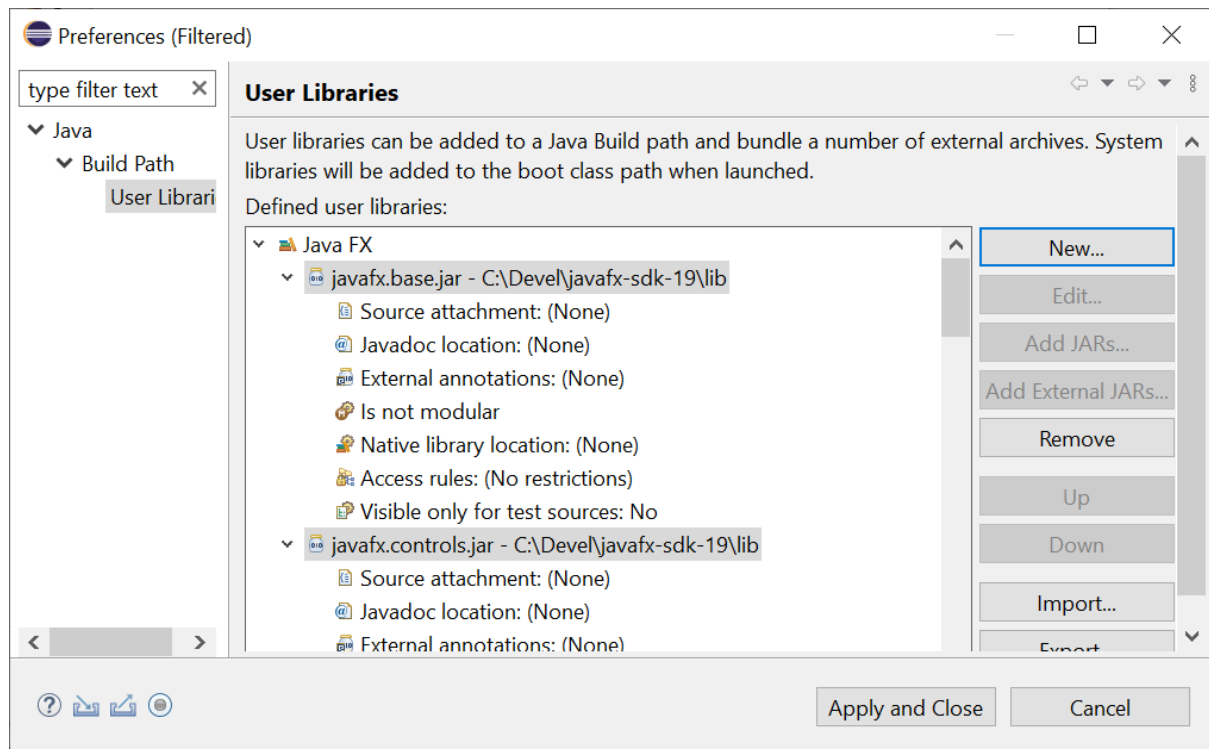
Dans votre projet Java, il faut créer une "User Library" qui va importer les JAR de la distribution d'Open JFX.

Ouvrez les propriétés du projet et "Java Build Path" puis onglet "Libraries" : sélectionnez "Classpath" puis cliquez sur "Add library...", sélectionnez "User Libraries", cliquez sur "User Libraries...". Cliquez sur "New", donnez le nom "Java FX".

Vous devez avoir ceci comme fenêtre :



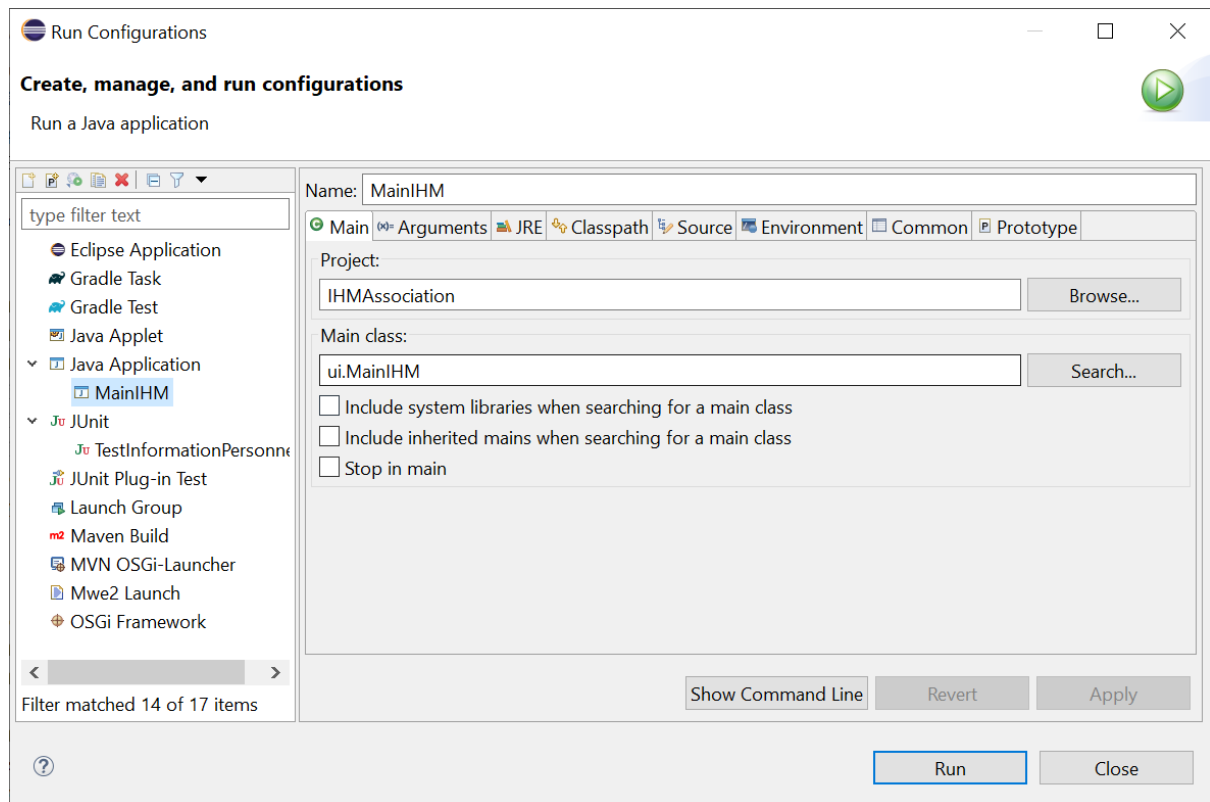
Cliquez sur "Add External JARs" pour ajouter les JAR d'Open JFX. Ouvrez le répertoire "lib" de l'installation d'Open JFX et sélectionnez tous les JAR s'y trouvant. Vous avez maintenant cette fenêtre :



Validez par “Apply and Close”. Vérifiez que vous avez bien “Java FX” dans le Classpath de votre projet Java.

A partir de là, vous pouvez utiliser les composants Java FX dans votre code, mais il reste une dernière chose à faire pour pouvoir lancer une application utilisant Java FX : rajouter le chemin vers les JAR d’Open JFX au lancement du programme.

Pour cela, sélectionnez la classe Java exécutable (MainIHM.java par exemple) : clic droit -> *run configurations*. Double-cliquez sur “Java Application” et vous devez avoir ceci comme fenêtre :

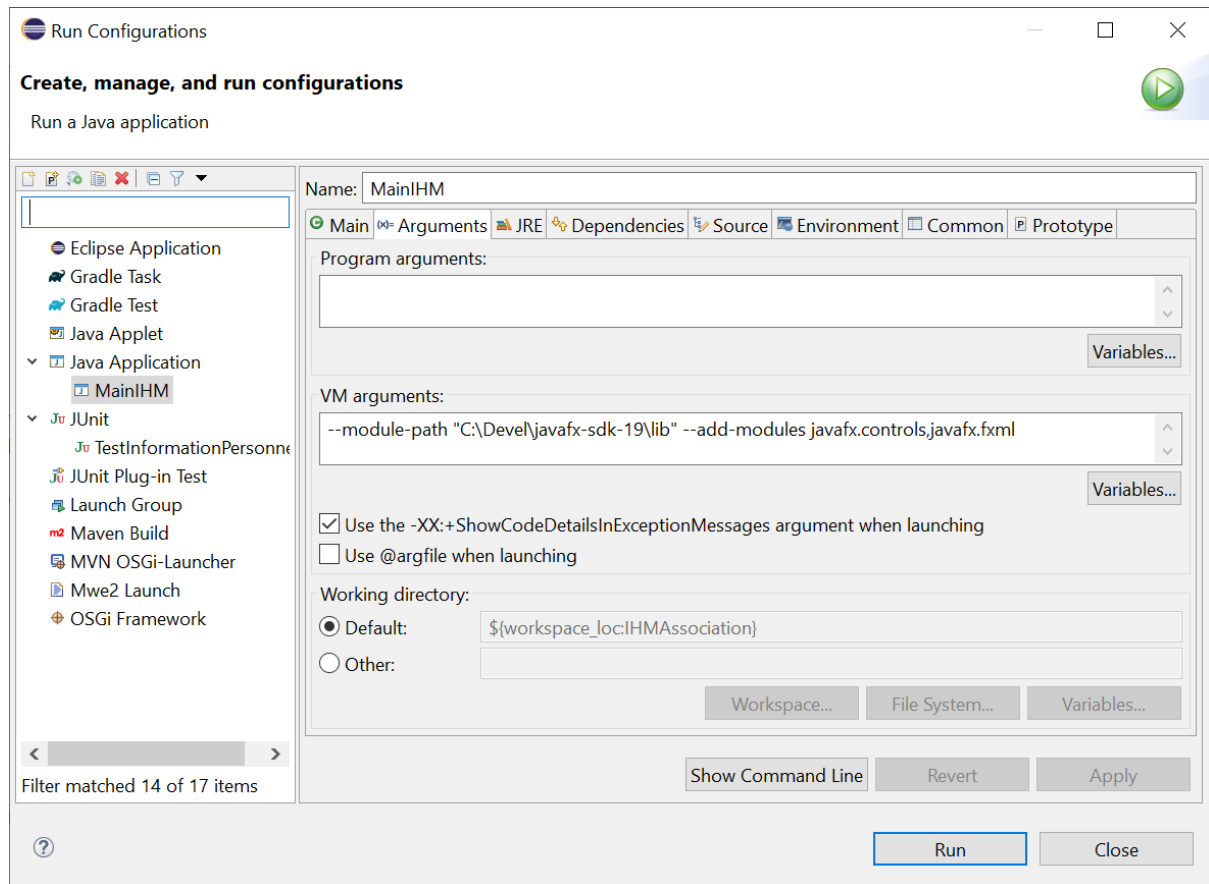


Affichez l'onglet "Arguments" et dans la zone "VM arguments", entrez la chaîne :

```
--module-path "C:\Devel\javafx-sdk-19\lib" --add-modules javafx.controls,javafx.fxml
```

Modifiez le chemin si vous avez installé Open JFX à un autre endroit.

Vous devez avoir ceci :



Et là, votre programme avec une interface Java FX se lancera sans erreur (à part vos erreurs de programmation ...).