



# Introduction à JavaFX

## *A travers la création d'une application « TodoList »*

### Sommaire

Sommaire.....	1
Mise en situation.....	2
Introduction à JavaFX .....	2
Qu'est-ce que JavaFX ? .....	2
SceneBuilder .....	2
Hiérarchie des classes en JavaFX.....	3
Composant simple de JavaFX.....	4
Références du cours .....	5
Mise en place de l'environnement.....	5
Création du projet JavaFX avec IntelliJ .....	5
Structure de base d'un projet JavaFX.....	8
Compréhension des fichiers initiaux.....	10
Lancement de notre projet TodoList !.....	13
Étape 1 : Respecter les conventions de nommage .....	13
Étape 2 : Organiser la structure du projet.....	13
Étape 3 : Retoucher l'interface LoginView.fxml .....	14
Étape 4 : Implémenter LoginController.java.....	15
Étape 5 : Création de InscriptionView.fxml et de son contrôleur .....	17
Étape 6 et 7 : Tester et valider le bon fonctionnement.....	18
Étape 8 : Mettre le projet sur GitHub.....	19
Bilan de ce cours.....	20
Annexes.....	21
Annexe 1 : Guide d'utilisation de Scene Builder .....	21



# Mise en situation

Vous êtes toujours au sein de l'entreprise NéoTech et après plusieurs jours de travail acharné, vous avez finalement réussi à développer une première version fonctionnelle de votre application de gestion de tâches en Java. Cependant, vous êtes conscients qu'il est possible d'améliorer l'expérience utilisateur en ajoutant une interface graphique intuitive. C'est pourquoi vous décidez de vous lancer dans l'apprentissage de JavaFX pour créer une version de votre application avec une interface graphique conviviale. Vous êtes confiants que cela permettra à vos collègues de gérer leurs tâches plus efficacement et de manière plus agréable.

## Introduction à JavaFX

### Qu'est-ce que JavaFX ?

**JavaFX** est une bibliothèque graphique développée par [Oracle en 2008](#) pour remplacer Swing. Elle permet de créer des [interfaces riches et interactives](#) pour les applications de bureau, mobiles et web.

Principales caractéristiques :

- [Intégration Java](#) : Inclus dans Java SE depuis la version 8, disponible en module depuis la version 9.
- [Interfaces modernes](#) : Effets visuels, animations, transitions, formulaires, graphes et tableaux.
- [Personnalisation](#) : Support du CSS pour le stylage et les thèmes.
- [Interopérabilité](#) : Gestion des événements, threads, bases de données et services Web.
- [Multiplateforme](#) : Compatible Windows, Mac OS, Linux et mobiles.

En résumé : JavaFX est une solution puissante et moderne pour créer des interfaces utilisateur en Java, avec une forte intégration à l'écosystème Java et une grande portabilité.

## SceneBuilder

Scene Builder est un [outil graphique](#) qui permet de concevoir des interfaces JavaFX [sans coder manuellement](#). Grâce à une interface glisser-déposer, il simplifie l'ajout de composants comme des boutons, champs de saisie ou menus.

Fonctionnement :

- Génère / édite des fichiers FXML, un format XML décrivant l'interface.
- Facilement intégrable dans une application JavaFX pour y ajouter la logique métier.
- Idéal pour créer rapidement des interfaces complexes sans toucher directement au code.

En résumé : Scene Builder facilite la création d'interfaces JavaFX en générant du FXML, tout en restant totalement compatible avec le code Java.



# Hiérarchie des classes en JavaFX

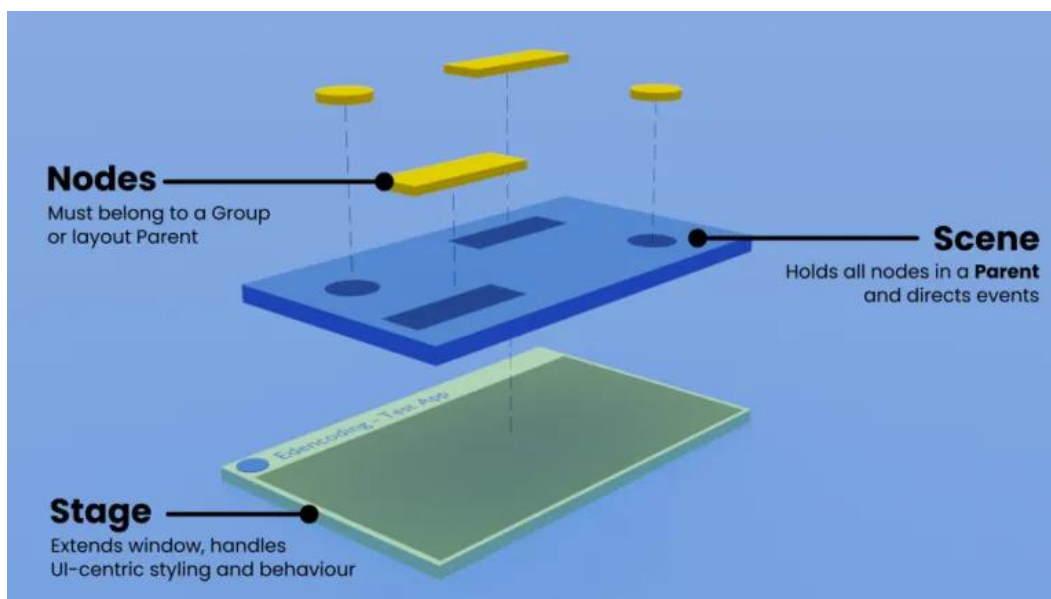
JavaFX organise son interface utilisateur sous forme d'une hiérarchie de classes, où chaque élément est un nœud pouvant contenir d'autres éléments.

Principales classes :

- **Stage** : Fenêtre principale ou boîte de dialogue contenant une ou plusieurs scènes.
- **Scene** : Contient tous les éléments graphiques d'une fenêtre. (les Nodes)
- **Node** : Élément de base de l'interface (bouton, image, champ de texte, etc.).
- **Parent** : Conteneur pouvant regrouper plusieurs Node (ex. : panneaux, boîtes de dialogue).
- **Control** : Élément interactif permettant la saisie ou l'affichage d'informations (boutons, champs de texte, cases à cocher...).

*Pourquoi cette hiérarchie ?*

Elle permet une organisation logique et facilite la conception d'interfaces complexes en imbriquant les éléments les uns dans les autres.





# Composant simple de JavaFX

Voici une liste de composants basiques de JavaFX :

- **TextField** : un champ de texte permettant de saisir des données. Il peut être utilisé pour saisir du texte, des nombres ou des dates.
- **Label** : une étiquette pour afficher des informations ou des messages d'erreur. Il peut être utilisé pour afficher du texte, des images ou des icônes.
- **Button** : un bouton qui peut être cliqué pour effectuer une action. Il peut être utilisé pour effectuer une opération, comme sauvegarder ou supprimer des données.
- **TableView** : une table qui peut être utilisée pour afficher et éditer des données. Il peut être utilisé pour afficher des données sous forme de tableau, où chaque ligne représente un enregistrement et chaque colonne représente un champ de données.
- **TableColumn** : une colonne dans une table, qui peut être utilisée pour afficher et éditer une colonne de données. Il peut être utilisé pour définir la structure d'une table, y compris le nom de la colonne, le type de données et les options d'édition.
- **CheckBox** : une case à cocher permettant de sélectionner ou de désélectionner une option. Il peut être utilisé pour permettre à l'utilisateur de sélectionner une option ou de confirmer une action.
- **ComboBox** : une liste déroulante permettant de sélectionner une option parmi plusieurs. Il peut être utilisé pour afficher une liste de choix à l'utilisateur, où il peut sélectionner une seule option.
- **DatePicker** : un calendrier permettant de sélectionner une date. Il peut être utilisé pour permettre à l'utilisateur de sélectionner une date ou une plage de dates.
- **Dialog** : une boîte de dialogue qui peut être utilisée pour afficher des messages, des alertes ou des demandes de confirmation. Il peut être utilisé pour afficher des messages à l'utilisateur, ou pour demander une confirmation avant d'effectuer une action.



## Références du cours

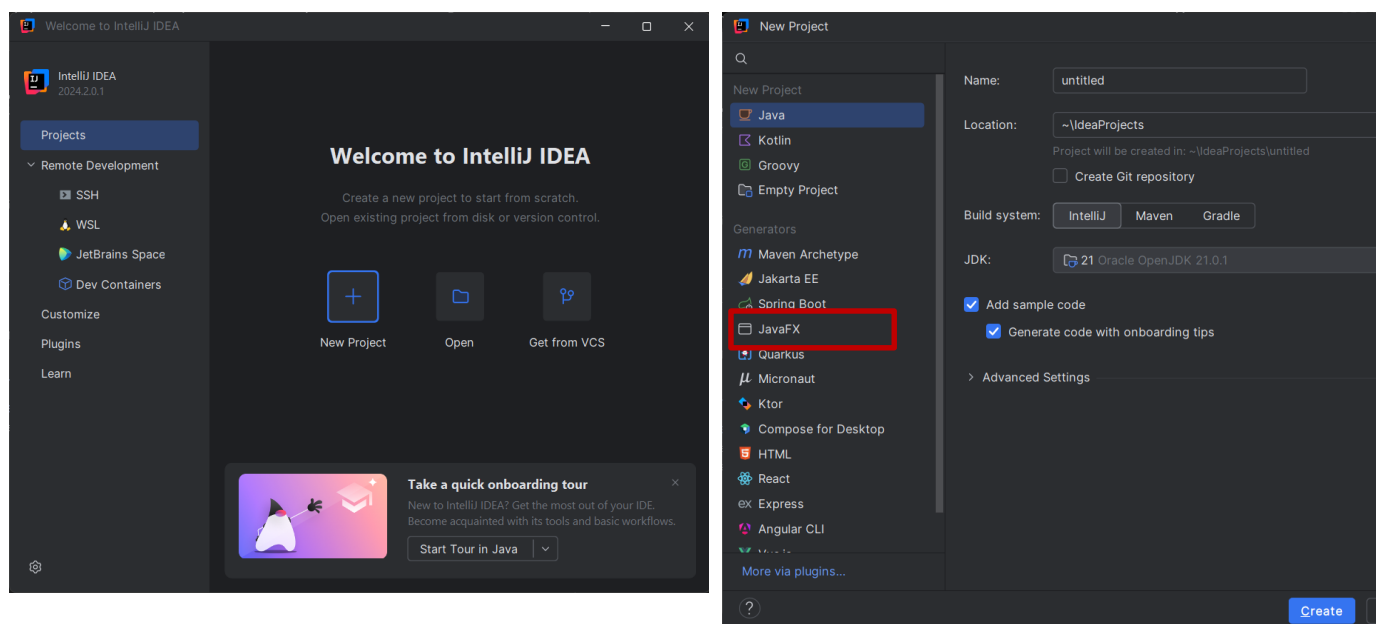
- <https://fr.wikipedia.org/wiki/JavaFX>
- <https://openjfx.io/>
- <https://gluonhq.com/products/scene-builder/>
- <https://mikarber.developpez.com/tutoriels/java/introduction-javafx/>

## Mise en place de l'environnement

### Création du projet JavaFX avec IntelliJ

#### *Etape 1: Démarrer un nouveau projet*


1. Ouvrez IntelliJ IDEA.
2. Cliquez sur "New Project".
3. Dans la liste des options, sélectionnez "JavaFX".



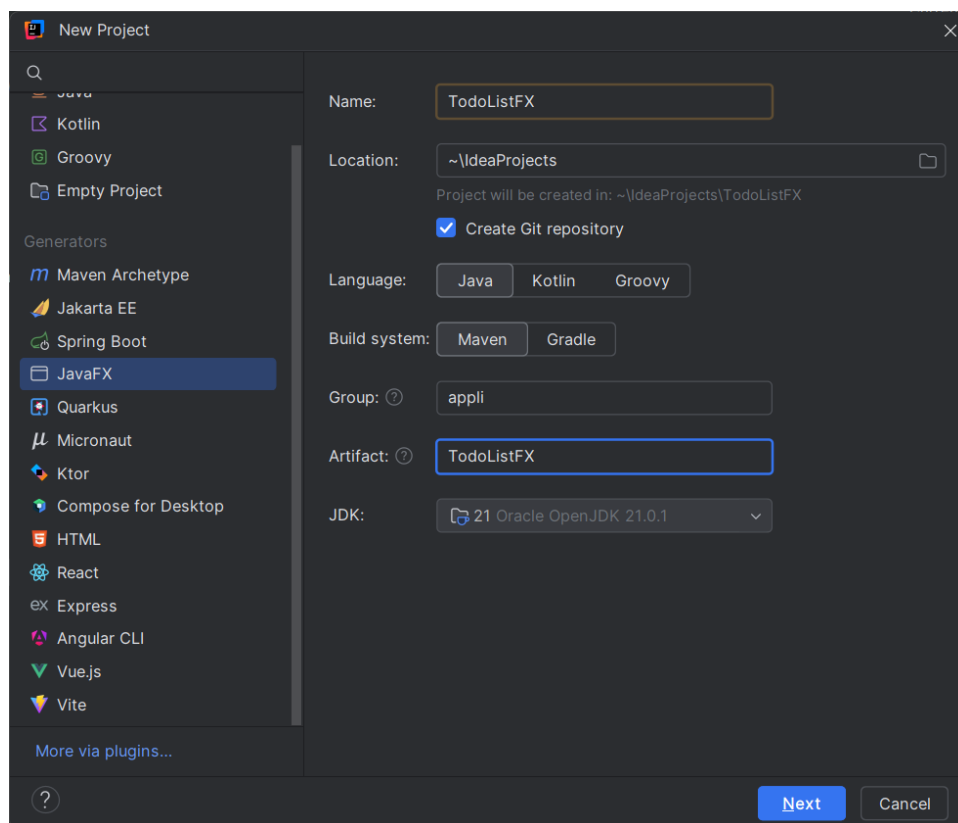


## Etape 2 : Configurer le projet

Dans la fenêtre de configuration, renseignez les paramètres suivants :

Option	Valeur à mettre	Explication
Nom du projet	ToDoListFX	C'est le nom de votre projet.
Localisation	A vous de choisir	Endroit où sera enregistré le projet.
Create a Git repository	Oui (conseillé)	Permet de versionner votre code.
Langage	Java	On garde Java comme langage principal.
Build System	Maven	Gère les dépendances du projet.
Group	appli	Définit l'organisation du projet.
Artifact	ToDoListFX (prérempli)	Identifiant du projet, souvent identique au nom du projet.
JDK	 Le plus récent disponible	Toujours utiliser une version récente pour éviter les incompatibilités.

**Astuce :** Le **Build System** permet de gérer les bibliothèques et dépendances de votre projet. Ici, nous utilisons **Maven**, un outil puissant pour automatiser cela.





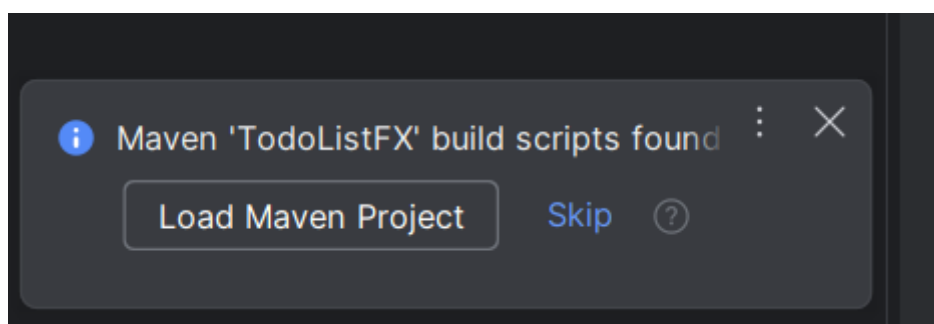
## *Étape 2 bis : Librairies additionnelles*

La liste des librairies additionnelles proposées par IntelliJ sont des librairies vous permettant d'ajouter des nouveaux éléments graphiques à vos interfaces autre que ceux nativement proposés par JavaFX. En exemple avec la première proposée nommée « BootstrapFX » qui vous permet d'ajouter un ensemble de feuilles de styles similaires à Bootstrap pour personnaliser vos rendus.

Ces éléments graphiques ne seront pas vu dans ce cours. Si vous souhaitez les utilisés dans vos projets, ce sera grâce à vos recherches. Je reste disponible si besoin.

## *Étape 3 : Vérifier et finaliser la création du projet*

1. Une fois les paramètres renseignés, cliquez sur "Create".
2. IntelliJ génère automatiquement la structure du projet.
3. Attendez quelques secondes que les dépendances **Maven** se chargent (icône en bas à droite : "Loading Maven Projects").



Si le chargement ne démarre pas :

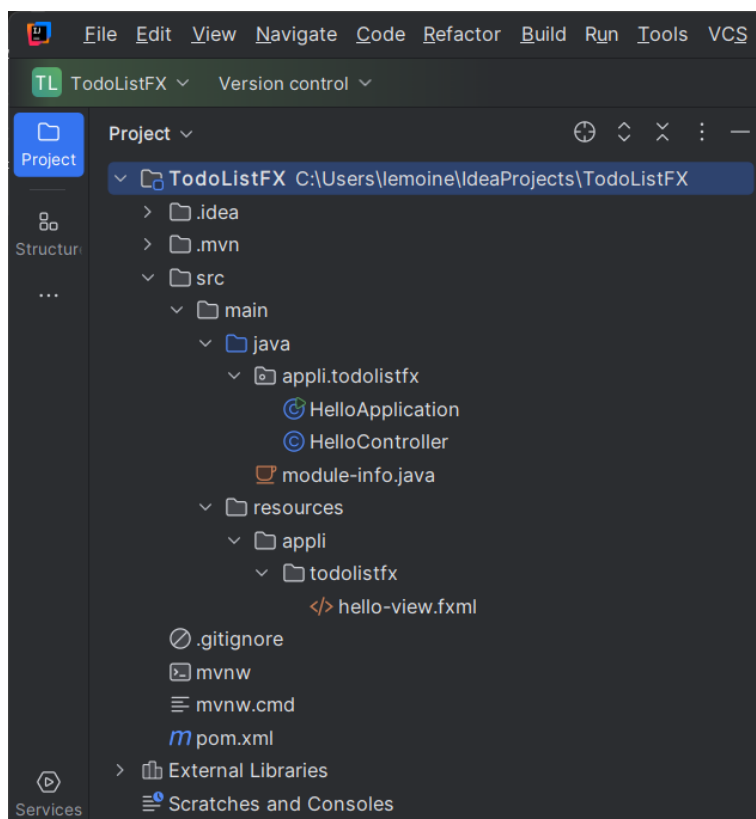
- Cliquez sur **pom.xml** > icône en haut à droite "Load Maven Projects".
- Assurez-vous que **votre JDK est bien configuré** dans IntelliJ.



# Structure de base d'un projet JavaFX

Une fois le projet créé, IntelliJ génère plusieurs dossiers et fichiers. Voici leur [rôle et organisation](#).

## *Arborescence du projet*



Explication des dossiers et fichiers

Dossier/Fichier	Rôle
src/main/java	Contient tout le code source Java de l'application.
appli/	Package principal de l'application.
HelloApplication.java	Point d'entrée du projet JavaFX (équivalent à main en Java).
HelloController.java	Gère les événements et la logique de la page hello-view.fxml
resources/	Contient les fichiers non Java : FXML, CSS, images, etc.
pom.xml	Fichier Maven : gère les dépendances et la configuration du projet.
module-info.java	Gère les modules Java (obligatoire avec JavaFX).

Le « . » dans « appli.todolistfx » est l'équivalent de « appli/todolistfx », ce sont bien deux packages distincts.





Dans notre projet actuel, nous avons deux packages (appli > todolist) qui ont été créés automatiquement qui a engendré également la création de deux dossiers dans ressources portant le même nom.

Chaque fichier fxml est liée à un fichier java.

## *Bonnes pratiques pour une structure claire*

- **Respecter l'organisation** : Séparer les contrôleurs, vues et fichiers CSS.
- **Nommer les fichiers de manière cohérente** (loginView.fxml ↔ LoginController.java).
- **Utiliser des packages clairs** (controllers, views, models si besoin).
- **Ne pas modifier pom.xml au hasard** : Il gère les dépendances du projet.

Vous serez amené à traiter l'ensemble de ces points durant les différents TP qui vous amèneront à une organisation type ainsi qu'à des conventions d'écritures qui devront être respectés dans les projets suivants.

La première règle à mettre en place concerne l'organisations et le nommage des fichiers.

- **Le contrôleur** d'une page terminera toujours par le terme « Controller.java »
  - o Ex : LoginController.java
- **La page** devra toujours se terminer par « View.fxml »
  - o Ex : LoginView.fxml
- **L'organisation** dans le dossier « java » et « ressources » doivent être **identique**.
  - o Ex : Une page qui se trouve dans le dossier « ressources/appli/admin/Accueil.fxml » aura son contrôleur qui se trouvera automatiquement dans « java/appli/admin/AccueilController.java »



# Compréhension des fichiers initiaux

## *Le fichier HelloApplication.java (Point d'entrée de l'application)*

Ce fichier contient la classe principale de l'application JavaFX. Il hérite d'Application et définit la méthode start, qui est appelée lors du lancement du programme.

```
package appli.todolistfx;
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.stage.Stage;

import java.io.IOException;

public class HelloApplication extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(HelloApplication.class.getResource("hello-
view.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 320, 240);
        stage.setTitle("Hello!");
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch();
    }
}
```

Ligne	Explication
extends Application	La classe hérite d'Application, ce qui en fait une application JavaFX.
public void start(Stage stage)	Méthode obligatoire qui définit la fenêtre principale de l'application.
FXMLLoader fxmlLoader = new FXMLLoader(...)	Charge le fichier FXML contenant l'interface graphique.
Scene scene = new Scene(fxmlLoader.load(), 320, 240);	Crée une scène avec la vue FXML.
stage.setTitle("Hello!")	Définit le titre de la fenêtre.
stage.setScene(scene)	Associe la scène à la fenêtre.
stage.show()	Affiche la fenêtre.
public static void main(String[] args)	Méthode principale qui lance l'application.
launch();	Appelle start() pour démarrer JavaFX.



## *Le fichier hello-view.fxml (L'interface utilisateur)*

Le fichier FXML décrit l'interface graphique sous forme de balises XML. Voici son contenu simplifié :

```
<VBox xmlns="http://javafx.com/javafx" xmlns:fx="http://javafx.com/fxml"
fx:controller="appli.todolistfx.HelloController">

    <Label fx:id="welcomeText" text="Welcome!" />

    <Button text="Click me!" onAction="#onHelloButtonClick" />

</VBox>
```

Elément	Description
<VBox>	Conteneur vertical pour organiser les éléments.
<Label fx:id="welcomeText" ...>	Un texte affiché à l'écran.
<Button text="Click me!" onAction="#onHelloButtonClick" />	Un bouton qui appelle onHelloButtonClick lorsqu'on clique dessus.

**Astuce** : On peut modifier ce fichier [visuellement](#) grâce à Scene Builder. (Clique droit sur le fichier > Open with SceneBuilder).

Si cela ne fonctionne pas, fiez-vous à [l'annexe 1](#) pour installer et lier SceneBuilder à IntelliJ. DE plus, vous avez une aide pour comprendre l'interface de SCeneBuilder



## *Le fichier HelloController.java (Gestion des interactions)*

Ce fichier contrôle les interactions entre l'utilisateur et l'interface FXML.

```
package appli.todolistfx;

import javafx.fxml.FXML;
import javafx.scene.control.Label;

public class HelloController {
    @FXML
    private Label welcomeText;

    @FXML
    protected void onHelloButtonClick() {
        welcomeText.setText("Welcome to JavaFX Application!");
    }
}
```

Ligne	Explication
@FXML private Label welcomeText;	Récupère le Label défini dans le FXML.
@FXML protected void onHelloButtonClick()	Méthode appelée lors du clic sur le bouton.
welcomeText.setText(...)	Modifie le texte affiché dans le Label.

### *Objectif : Comprendre et expérimenter*

#### À vous de jouer !

1. Lancez l'application et observez le résultat.
2. Modifiez le texte du bouton dans hello-view.fxml, puis relancez le projet.
3. Renommez le fichier « hello-view » en « helloView.fxml » pour respecter les conventions de nommage
4. Ajoutez un deuxième bouton et créez une nouvelle méthode pour afficher un autre message.

[Une fois fait, appelez-moi pour vérifier ce que vous avez fait](#)

**Pourquoi fait-on ça ?** Parce que JavaFX sépare la logique (Java) de l'interface (FXML), ce qui facilite la maintenance et l'évolution du projet.

**Prochaine étape :** Approfondir Scene Builder et organiser notre projet !



# Lancement de notre projet TodoList !

## Étape 1 : Respecter les conventions de nommage

L'objectif de cette première étape est de renommer les fichiers du projet pour [respecter des conventions claires et logiques](#).

### Modifications à effectuer :

1. Renommer HelloApplication.java → StartApplication.java
  - Ce fichier est le [point d'entrée de l'application](#), il doit avoir un nom explicite.
  - Vérifiez que la classe interne porte bien le même nom.
2. Renommer HelloView.fxml → LoginView.fxml et HelloController.java → LoginController.java
  - La page affichée par défaut sera la [page de connexion](#)

## Étape 2 : Organiser la structure du projet

Nous allons structurer les fichiers pour que le projet soit clair et facile à maintenir.

### Nouvelle organisation des fichiers

```
src/main/java
├── appli
│   ├── StartApplication.java
│   ├── accueil
│   │   ├── LoginController.java
│   │   └── InscriptionController.java
└── src/main/resources
    ├── appli
    │   ├── accueil
    │   │   ├── LoginView.fxml
    │   │   └── InscriptionView.fxml
```

### Actions à réaliser :

1. [Déplacer StartApplication.java](#) dans le package appli.
2. [Créer un package accueil](#) dans appli.
3. [Déplacer LoginController.java](#) dans accueil.
4. [Créer un fichier InscriptionController.java](#) dans accueil.
5. [Déplacer LoginView.fxml](#) dans resources/appli/accueil.
6. [Créer un fichier InscriptionView.fxml](#) dans resources/appli/accueil.



### Pourquoi faire cela ?

- Le fichier principal (StartApplication.java) reste accessible à la racine du package appli.
- Les vues et contrôleurs sont regroupés logiquement dans accueil.
- Les fichiers FXML se trouveront au même endroit que leur contrôleur java.

## Étape 3 : Retoucher l'interface LoginView.fxml

Nous allons maintenant modifier LoginView.fxml pour créer une véritable page de connexion.

### Objectif : Créer une interface fonctionnelle pour la connexion

Modifications à effectuer :

- Ajouter un titre à votre page
- Ajouter des champs de saisie pour l'email et le mot de passe. (avec un libellé)
  - o TextField pour l'email
  - o PasswordField pour le mot de passe
  - o Label pour les libellés de champs
- Associer chaque élément à un fx:id pour pouvoir les manipuler dans LoginController.java.
- Ajouter trois boutons : Connexion, Inscription et Mot de passe oublié.
- Lier les boutons à des méthodes d'action (onAction) pour gérer les interactions.
- Organiser l'interface avec un conteneur clair de votre choix
- Prévoyez un label pour les erreurs

Exemple d'une interface regroupant les éléments demandés :

Bienvenu !

Votre Email :

Votre mot de passe :

Label pour erreur

Connexion

Mot de passe oublié

Inscription



## Étape 4 : Implémenter LoginController.java

Maintenant que LoginView.fxml est prêt, nous allons écrire le contrôleur LoginController.java pour gérer les interactions.

*N'oubliez pas que SceneBuilder Peut vous donner le squelette de votre contrôleur*

### Objectif : Gérer la connexion et la navigation

Fonctionnalités à implémenter :

- Récupérer les valeurs saisies dans les champs emailField et passwordField.
- Afficher les informations en console (pas encore de base de données !).
- Simuler une validation simple et afficher un message d'erreur si les identifiants sont incorrects.
  - o En exemple, vous pouvez tester si l'email vaut à votre email et au mot de passe « Azerty1234 »
- Ajouter la navigation vers la page d'inscription (InscriptionView.fxml).

### Spécificités : La navigation

Pour la navigation, nous allons ajouter une méthode dans StartApplication.java vous permettant le changement de pages.

```
public static void changeScene(String nomDuFichierFXML) throws IOException {  
    FXMLLoader fxmlLoader = new  
FXMLLoader(StartApplication.class.getResource(nomDuFichierFXML + "View.fxml"));  
    Scene scene = new Scene(fxmlLoader.load());  
    mainStage.setScene(scene);  
}
```

Ligne	Explication
FXMLLoader fxmlLoader = new FXMLLoader(...)	Charge dynamiquement le fichier FXML correspondant à la vue demandée.
new Scene(fxmlLoader.load())	Crée une nouvelle scène à partir du fichier chargé.
mainStage.setScene(scene);	Remplace l'affichage actuel par la nouvelle scène.

Pourquoi cette méthode est utile ?

- Elle permet de changer de page facilement sans dupliquer du code.
- On peut l'appeler depuis n'importe quel contrôleur.



Or, la variable « mainStage » n'a pas été créée. mainStage va représenter la fenêtre principale fournie lors du lancement de notre projet (notre méthode start()).

Modifier la classe pour remplacer le code de la méthode start() par :

```
private static Stage mainStage;

@Override
public void start(Stage stage) throws IOException {
    mainStage = stage;
    FXMLLoader fxmlLoader = new
FXMLLoader(StartApplication.class.getResource("accueil/loginView.fxml"));
    Scene scene = new Scene(fxmlLoader.load());
    mainStage.setTitle("Hello!");
    mainStage.setScene(scene);
    mainStage.show();
}
```

Explication des changements apportés :

- Nous avons ajouté un attribut de type Stage nommé mainStage
  - o Elle représente notre fenêtre principale (au lancement du projet)
- Dans la méthode
  - o L1 : « mainStage = stage; » on sauvegarde la fenêtre donnée en paramètre pour la manipuler à notre guise ! (Et devient notre fenêtre principale)
  - o Pour le reste du code, on a remplacé la variable « stage » par « mainStage » car on souhaite être sûr que la personnalisation soit sur notre fenêtre sauvegardées !

Objectif : Placer le code dans StartApplication et utilisez-le dans le loginController





## Etape 5 : Création de InscriptionView.fxml et de son contrôleur

Nous allons maintenant créer la page d'inscription et son contrôleur pour permettre aux utilisateurs de s'enregistrer.

### Objectif 1 : Créer une interface fonctionnelle pour l'inscription

Ajout à effectuer :

- Ajouter un titre à votre page
- Ajouter des champs de saisie pour le nom, prénom, l'email, le mot de passe et la confirmation du mot de passe. (Avec un libellé)
  - o Utiliser les mêmes types de champs que pour la connexion
- Associer chaque élément à un fx:id pour pouvoir les manipuler dans inscriptionController.java.
- Ajouter deux boutons : S'inscrire, Connexion.
- Lier les boutons à des méthodes d'action (onAction) pour gérer les interactions.
- Organiser l'interface avec un conteneur clair de votre choix
- Prévoyez un label pour les erreurs

Exemple d'une interface regroupant les éléments demandés :

The mockup shows a registration form titled "Inscription". It contains five input fields, each with a label above it: "Nom", "Prenom", "Email", "Mot de passe", and "Confirmation". Below the input fields are two buttons: "Inscription" and "Retour".



## Objectif 2 : Gérer l'inscription et la navigation

Fonctionnalités à implémenter :

- Récupérer les valeurs saisies dans les champs.
- Afficher les informations en console (pas encore de base de données !).
- Simuler une validation simple et afficher un message d'erreur
  - o Si les mots de passes sont différents
  - o Si l'email est déjà utilisés !
    - Avec l'email qui est renseigné dans la connexion
  - o Si l'un des champs est vide
- Ajouter la navigation vers la page de connexion (loginView.fxml).

## Étape 6 et 7 : Tester et valider le bon fonctionnement

Maintenant que l'inscription et la connexion sont en place, nous allons tester si tout fonctionne correctement et s'assurer que la navigation entre les pages est fluide.

### Tâches de test

Vérifier le bon affichage des interfaces

- ✓ Lancez l'application et assurez-vous que la page de connexion s'affiche correctement.
- ✓ Naviguez vers la page d'inscription et vérifiez son affichage.
- ✓ Retournez à la page de connexion et assurez-vous qu'aucune erreur visuelle ne se produit.

Tester les interactions des formulaires

### *Connexion*

- ✓ Saisissez un email et un mot de passe et cliquez sur "Connexion".
- ✓ Vérifiez que les informations sont bien affichées en console.
- ✓ Essayez avec des identifiants incorrects et assurez-vous que le message d'erreur s'affiche.

### *Inscription*

- ✓ Remplissez tous les champs et cliquez sur "S'inscrire".
- ✓ Testez les différents scénarios :
  - o Laissez des champs vides → le message d'erreur doit s'afficher.
  - o Saisissez deux mots de passe différents → le message d'erreur doit s'afficher.
  - o Remplissez correctement tous les champs → l'inscription doit être validée en console.



Tester la navigation entre les pages

- ✓ Vérifiez que le bouton "Inscription" dans LoginView.fxml dirige bien vers la page d'inscription.
- ✓ Vérifiez que le bouton "Retour" dans InscriptionView.fxml renvoie bien à la connexion.

Si un bug est détecté, utilisez les messages en console pour comprendre l'erreur et la corriger !

Je reste à votre disposition pour vous aider si besoin !

## Étape 8 : Mettre le projet sur GitHub

Avant de créer votre dépôt et d'envoyer le code sur GitHub. N'hésitez pas à m'appeler pour valider votre travail !

Félicitations ! Votre projet est maintenant bien structuré, testé et sauvegardé sur GitHub !



# Bilan de ce cours

Félicitations ! Nous avons construit ensemble une **première application JavaFX structurée et fonctionnelle**. Voici un résumé des différentes étapes et compétences acquises :

- ✓ **Création et structuration du projet JavaFX :**
  - Mise en place des fichiers et respect des conventions de nommage.
  - Organisation des packages pour une meilleure maintenabilité.
- ✓ **Développement des interfaces avec FXML et Scene Builder :**
  - Création de LoginView.fxml et InscriptionView.fxml.
  - Utilisation de Scene Builder pour faciliter la mise en page.
- ✓ **Gestion des interactions avec les contrôleurs :**
  - Implémentation de LoginController.java et InscriptionController.java.
  - Récupération des saisies utilisateur et affichage des messages d'erreur.
- ✓ **Navigation entre les vues :**
  - Implémentation d'une méthode globale changeScene() pour changer d'interface facilement.
- ✓ **Débogage et affichage en console :**
  - Vérification des valeurs entrées par l'utilisateur.
  - Affichage des erreurs et messages de confirmation.
- ✓ **Test et validation de l'application :**
  - Vérification du bon fonctionnement des formulaires et de la navigation.
  - Correction des erreurs potentielles en utilisant la console.



# Annexes

## Annexe 1 : Guide d'utilisation de Scene Builder

Scene Builder est un outil permettant de créer et modifier visuellement des interfaces JavaFX sans écrire directement du code XML.

### *Installation et configuration dans IntelliJ*

#### Télécharger Scene Builder

Scene Builder est disponible sur le site officiel :

- <https://gluonhq.com/products/scene-builder/>.
- Téléchargez la version correspondant à votre système d'exploitation (Windows, macOS, Linux).
- Installez-le en suivant les instructions.

#### Configurer Scene Builder dans IntelliJ

- Ouvrez IntelliJ IDEA.
- Allez dans File > Settings (ou Preferences sur macOS).
- Dans la barre de recherche, tapez JavaFX.
- Dans la section Path to Scene Builder, cliquez sur l'icône du dossier.
- Sélectionnez l'exécutable SceneBuilder.exe (Windows) ou le fichier correspondant sur macOS/Linux.
- Cliquez sur OK.

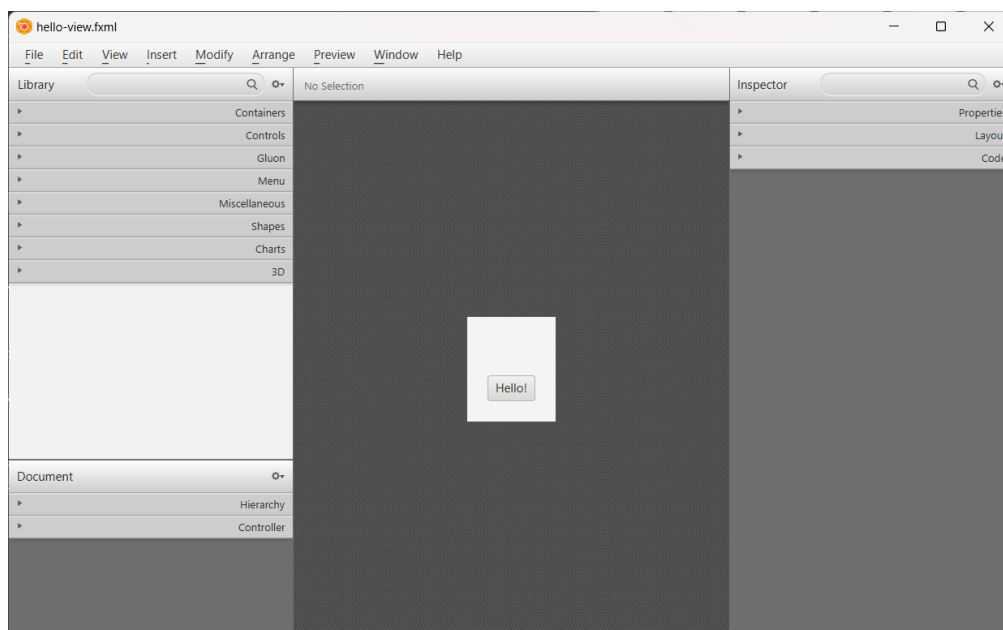
#### Testez l'installation :

- Faites un clic droit sur un fichier .fxml et sélectionnez « Open with Scene Builder ».



## *Interface de Scene Builder*

Lorsque vous ouvrez un fichier .fxml avec Scene Builder, voici les zones principales de l'interface :



### Zone centrale : Aperçu de l'interface

- Permet de voir et modifier l'interface graphique en temps réel.
- Les éléments peuvent être déplacés par glisser-déposer.

### Panneau gauche : Composants (Library)

- Contient tous les éléments graphiques disponibles (boutons, labels, champs de texte, etc.).
- Organisé par catégories (Containers, Controls, Shapes...).

### Panneau droit : Propriétés et événements

- Properties : Modifier le texte, la couleur, la taille des éléments.
- Layout : Gérer l'alignement et la position.
- Code : Associer un élément à un identifiant (fx:id) et définir ses actions (onAction).

### Panneau en bas : Hiérarchie des éléments (Document)

- Affiche la structure de l'interface sous forme d'arborescence.
- Permet de sélectionner et organiser les composants.



## *Modifier une interface avec Scene Builder*

### Ajouter un composant

- Faites glisser-déposer un élément (ex : Button) depuis la Library vers la zone centrale.
- Sélectionnez l'élément et modifiez ses propriétés dans le panneau Properties.

### Associer un identifiant à un élément

- Sélectionnez l'élément (ex : Label).
- Dans l'onglet Code, renseignez un fx:id (ex : welcomeLabel).

### Ajouter un événement à un bouton

- Sélectionnez un Button.
- Dans l'onglet Code, renseignez onAction avec le nom d'une méthode (ex : onButtonClick).

### Générer un squelette de contrôleur Java

- Allez dans View > Show Sample Controller Skeleton.
- Copiez le code et collez-le dans votre fichier contrôleur.