Gestion d'un modèle de contact en Java avec JavaFX ,sérialisation et utilisation de sceneBuilder pour l'interface visuelle.

1. Objectif du projet :

Ce projet vise à créer une classe Contact modélisant une liste de contacts, intégrant des fonctionnalités adaptées à une interface JavaFX, tout en permettant la sérialisation/désérialisation des objets pour sauvegarde ou transfert.

2. Structure et technologies utilisées :

fx:controller="fr.afpa.ContactController" ?:

Lien avec le contrôleur JavaFX qui gère la logique métier.

Permet d’avoir un code MVC (Modèle – Vue – Contrôleur)

Langage : Java

Technologies :

JavaFX : pour la gestion des interfaces graphiques et les Property (pour liaison de données): StringProperty encapsule les attributs pour qu'ils deviennent "observable" dans la vue.

Jackson : exporter en Json

Sérialisation binaire : pour sauvegarder/restaurer des objets Contact

3. Contenu de la classe Contact (model) :

Méthodes :

Getters/setters

toString() : pour affichage simplifié

Eléments principaux :

- Affichage de la liste de contact dans la tableview à gauche
- Formulaire de modification et saisie à droite
- Boutons d'actions (CRUD)
- Option d'exportation

Actions reliées avec onAction="#..." dans le sceneBuilder
Liaison avec les composants JavaFX (TextField, etc.)

Permet de lier le sceneBuilder au contrôleur Java (ContactController)

Utilisation d’un fichier CSS externe pour gérer l’apparence=>Bon pour la séparation de la logique et du style.

Structure principale :

- AnchorPane :

=> Permet d'ancrer les composants aux bord de la fenêtre, flexible

=> Vbox : Organisation verticale des éléments

=> Titre(label)

=> Un contenu central Hbox à gauche(alignement horizontal)
==========
==========

A gauche, tableview :

=>Affichage de la liste des contacts :

Exportation avec MenuButton :

=>Pourquoi une MenuButton pour exporter ?
    Affiche les différents formats disponibles (CSV, JSON, VCF).

    Intuitif pour l’utilisateur, regroupe plusieurs options dans un seul bouton.

==========
==========

A droite :

=> GridPane :

Permet une mise en grille pour les labels et les champs, alignement facile.

=> TexField : pour saisir des données texte(nom, email...)

=> Label : pour l'intitulé des champs

=>MenuButton : pour le choix du genre et plusieurs choix avec le MenuItem

=>Hbox : Organisation horizontale des boutons

Création d’un Contact, modification, suppression choix du genre. Utilisation des Regex pour le format de l'email (si pas conforme : visualisation différente des champs quand la saisie est erronnée.
Et pour les champs obligatoires, même chose : alerte pour signaler à l'utilisateur que les champs ne sont pas remplis.
Sélection obligatoire dans la tableview d'un contact pour supprimer et modifier un contact.
Bouton filtre qui filtre avec les champs obligatoires à droite. Possibilité de désactiver le filtre pour afficher tous les contacts.

Attributs : nom, prénom, ville, date de naissance, genre, pseudo, adresse, téléphone perso/pro, email, département, code postal, lien GitHub

Utilisation de StringProperty et ObjectProperty :

Permet une liaison directe avec les composants JavaFX (text fields, combo box…)

Enumération Genre :

Représente le genre de la personne

Méthodes personnalisées writeObject() et readObject() : permettent de sérialiser les Property JavaFX, qui ne sont pas sérialisables par défaut

4. Points techniques importants

Insiste sur la logique métier :

Pourquoi transient ?

Les Property JavaFX ne sont pas sérialisables, il faut donc les exclure de la sérialisation Java standard

Les champs sont convertis en types simples (String, Enum) pour être enregistrés

Sauvegarde/restauration dans un fichier .ser via ObjectOutputStream

Export possible en JSON avec Jackson si besoin