المدرسة العليا للتربية والتكوين - وجدة
ⵜⵜⴰⵙⵙⴰⵡⵜ ⵜⴰⵍⵉⵢⴰⵜ ⵏ ⵜⵓⵎⵎⴰⵔⵜ ⴰ ⵜⵓⵙⵙⵉⵔⵜ – ⵡⵊⴷⴰ
ÉCOLE SUPÉRIEURE DE L'ÉDUCATION ET DE LA FORMATION - OUJDA

Rapport de mini projet : Système de Gestion de Bibliothèque en Java et Java Fx

		HAMMAMI Fatima-Zahra
Réalisé par	:	Oufkir Fayza
		JBOUB Wissal
Encadré par	:	Mme.ROUBI Sarra

Filière	:	Licence d'Education enseignement secondaire **Option** Informatique
Module	:	Programmation Orientée Objets

Année universitaire : 2023-2024

Table des matières

Table des figures

Chapitre 1 : Cahier de charges

I. Objectif

L'objectif principal du système de gestion de bibliothèque est de permettre à l'administrateur d'ajouter, de lister, de supprimer et de modifier des livres dans la bibliothèque.

II. Fonctionnalités

➢ Ajout de livre

- L'administrateur peut ajouter de nouveaux livres au système.
- Chaque livre doit avoir les informations suivantes :
  - ID ;
  - Titre ;
  - Auteur ;
  - Genre ;
  - Année de publication ;
  - Quantité disponible.

➢ Listage de livre

- Le système doit permettre à l'administrateur de lister tous les livres présents dans la bibliothèque.
- La liste doit inclure toutes les informations sur chaque livre.

➢ Modification de livre

- L'administrateur peut modifier les informations d'un livre existant.
- Les champs modifiables sont : ID, Titre, Auteur, Genre, Année de publication, Quantité disponible.

➢ Suppression de livre

- L'administrateur peut supprimer un livre du système.
- La suppression doit être confirmée pour éviter toute suppression accidentelle.

III. Outils et technologie utilisés

➢ Java

Figure 1 : Java

Java a été l'un des langages de programmation les plus populaires et est enseigné largement dans les programmes éducatifs en informatique.
C'est un langage de programmation polyvalent et orienté objet, créé par Sun Microsystems (maintenant détenu par Oracle). Il a été conçu pour être indépendant de la plate-forme, ce qui signifie que les programmes Java peuvent être exécutés sur n'importe quel appareil équipé d'une machine virtuelle Java (JVM).

Les caractéristiques clés de Java incluent sa portabilité, sa simplicité, sa sécurité et sa capacité à prendre en charge des applications distribuées. Il est largement utilisé pour le développement d'applications web, d'applications mobiles (Android est basé sur Java), de logiciels d'entreprise et de nombreux autres types de programmes.

Un autre aspect important de Java est son écosystème riche, comprenant une vaste bibliothèque standard ainsi que de nombreux frameworks et outils qui simplifient le développement d'applications.

➢ JavaFX

Figure 2 : JavaFX

JavaFX est un framework de développement d'interfaces utilisateur (UI) graphiques pour les applications Java. Il a été introduit par Sun Microsystems et est maintenant maintenu par Oracle. JavaFX fournit un ensemble d'outils, de bibliothèques et de composants pour créer des applications graphiques riches et interactives pouvant s'exécuter sur diverses plates-formes, notamment Windows, macOS et Linux.

➢ Eclipse

Figure 3 : Eclipse

Eclipse est un environnement de développement intégré (IDE) largement utilisé, principalement pour le développement de logiciels en Java, bien qu'il supporte également d'autres langages de programmation tels que C++, PHP, et plus encore.

Eclipse fournit des fonctionnalités telles que l'édition de code, le débogage, la gestion de projet, la navigation dans le code, et l'intégration avec des systèmes de contrôle de version.

➢ Scene Builder

Figure 4 : Scene Builder

Scene Builder est un outil de conception graphique qui simplifie la création d'interfaces utilisateur pour les applications JavaFX. Il permet aux développeurs et aux concepteurs de créer des interfaces utilisateur graphiques de manière visuelle sans avoir à écrire manuellement le code correspondant.

IV. Conception et modélisation

La conception est une phase importante avant la réalisation de tout projet, Cette phase nécessite des méthodes permettant de mettre en place un modèle sur lequel on va s'appuyer. C'est-à-dire créer une représentation similaire à la réalité de telle façon à faire ressortir les points auxquels on s'intéresse. Pour ce travail nous avons opté pour le langage de modélisation UML.

➤ UML

Figure 5 : Unified Modeling Language

UML est un standard ouvert contrôlé par l'OMG, un consortium d'entreprises qui a été fondé pour construire des standards qui facilitent l'interopérabilité et plus spécifiquement, l'interopérabilité des systèmes orientés objet. UML est issu de l'unification de nombreux langages de modélisation graphique orientée objet. Il unifie à la fois les notations et les concepts orientés objets.

a. Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation est l'un des diagrammes UML (Unified Modeling Language) qui est largement utilisé dans l'ingénierie logicielle pour représenter visuellement les interactions entre les utilisateurs (acteurs) et un système logiciel. Il met l'accent sur les fonctionnalités offertes par le système du point de vue des utilisateurs.

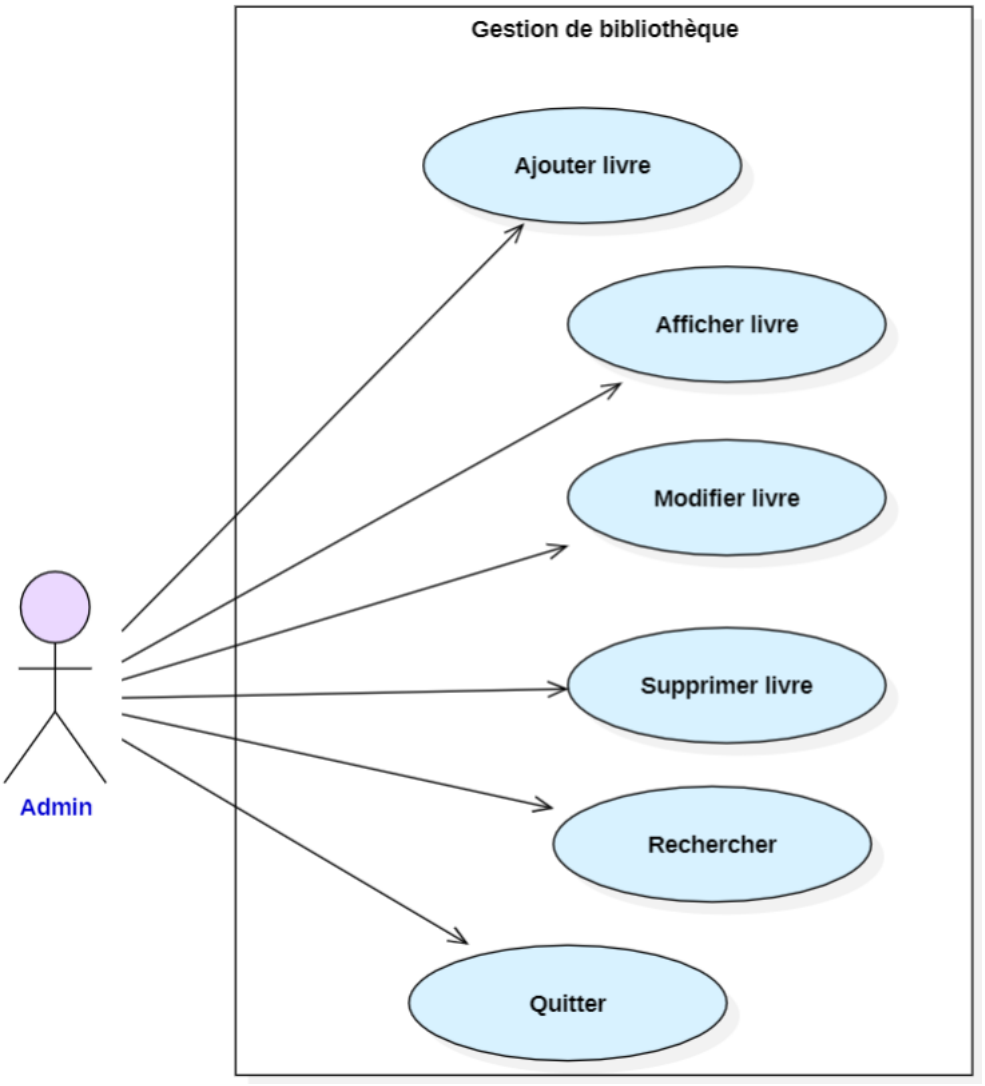

Figure 6 : Diagramme de cas d'utilisation

b. Diagramme de classe

Un diagramme de classe en UML (Unified Modeling Language) est un type de diagramme structurel qui représente la structure statique d'un système en illustrant les classes du système, les attributs de ces classes, les méthodes qu'elles implémentent, et les relations entre les classes. Les diagrammes de classe sont largement utilisés dans le domaine de l'ingénierie logicielle pour visualiser et documenter la conception des systèmes orientés objet.

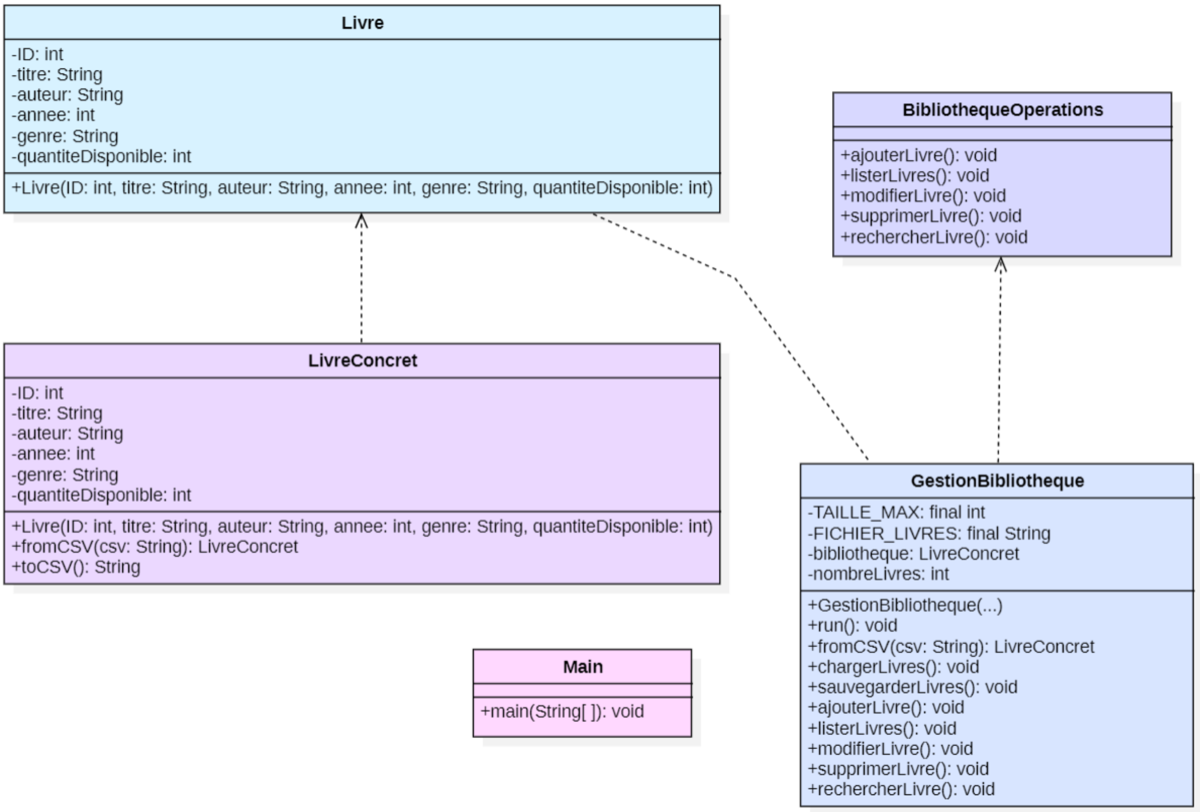

Figure 7 : Diagramme de classe

Explications :

La classe **Livre** est la classe abstraite de base contenant les attributs communs aux livres.

La classe **LivreConcret** étend Livre et ajoute des méthodes spécifiques pour la gestion des livres concrets.

L'interface **BibliothequeOperations** définit les opérations de base à effectuer sur la bibliothèque.

La classe GestionBibliotheque implémente l'interface BibliothequeOperations et contient la logique de gestion de la bibliothèque.

La classe **Main** contient la méthode principale main servant de point d'entrée pour l'exécution du programme.

c. Diagramme de paquetage

En UML (Unified Modeling Language), le diagramme de paquetage est un diagramme structurel qui représente l'organisation logique et physique d'un système en termes de paquets. Les paquets sont des conteneurs utilisés pour regrouper des éléments UML apparentés, tels que des classes, des diagrammes, des sous-systèmes.

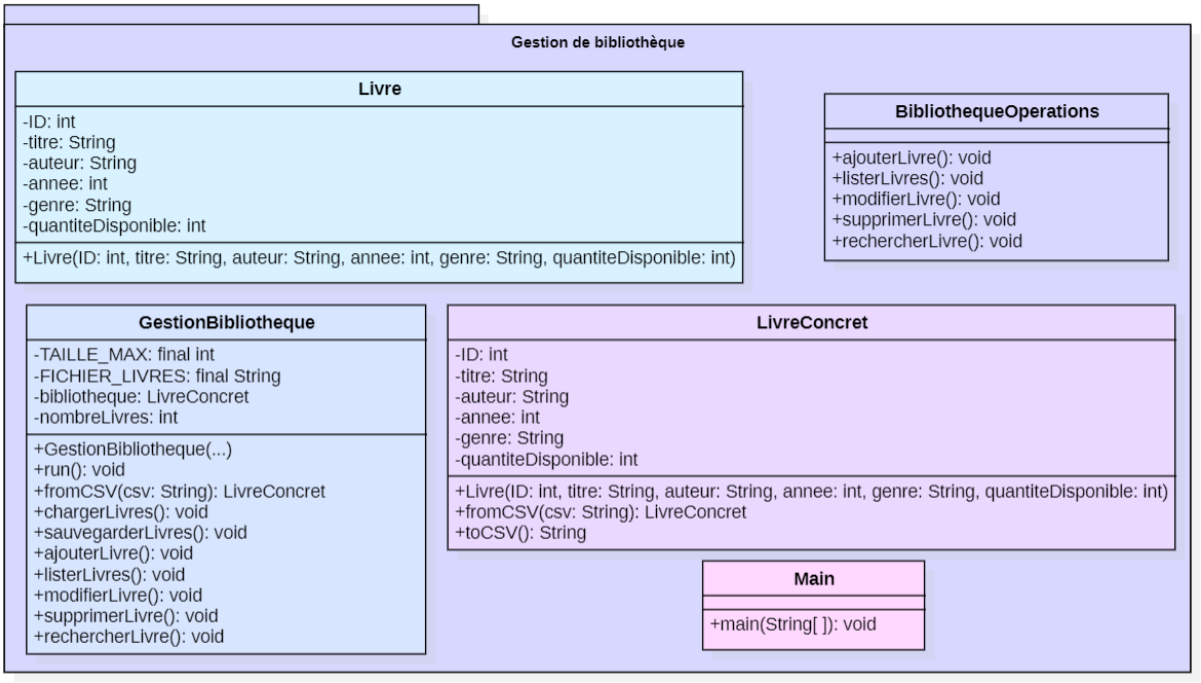

Figure 8 : Diagramme de paquetage

V. Interface utilisateur

➢ Menu principal

- Affiche les options suivantes :
  - Ajouter un livre ;
  - Lister les livres ;
  - Modifier un livre ;
  - Supprimer un livre ;
  - Quitter.

➢ Ajout de livre

- L'utilisateur doit fournir toutes les informations requises.
- Un message de succès doit être affiché après l'ajout.

➢ Listage de livre

- Affiche toutes les informations de chaque livre.

➢ Modification de livre

- L'utilisateur doit sélectionner le livre à modifier en entrant l'ID.
- Affiche les informations actuelles du livre et permet à l'utilisateur de les modifier.

➢ Suppression de livre

- L'utilisateur doit confirmer la suppression en entrant l'ID du livre à supprimer.

Chapitre 2 : Réalisation du mini projet dans la console en utilisant Java

VI. Classes et explications

Pour la réalisation du projet "Système de Gestion de Bibliothèque en Java", nous nous sommes basés sur la charte que nous avons établie. Comme vous pouvez le constater, nous avons décidé d'intégrer un menu de choix à travers lequel l'utilisateur peut sélectionner l'action qu'il souhaite effectuer, que ce soit ajouter, supprimer, modifier ou rechercher.

Nous avons créé plusieurs classes pour assurer le bon fonctionnement de notre projet, les voici :

- ✓ Class abstraite Livre ;
- ✓ Class LivreConcrete ;
- ✓ Interface BibliothequeOperation ;
- ✓ Class BibliothequeGestion ;
- ✓ Class Main ;

➢ Classe Livre

Cette classe est une classe abstraite sert à donner le modèle de base pour tous les livres. Elle définit les caractéristiques communes à tous les livres, telles que l'ID, le titre, l'auteur, l'année, le genre et la quantité disponible.

- **Les Attributs** :
  - ❖ ID : Un identifiant unique pour chaque livre.
  - ❖ titre : Le titre du livre.
  - ❖ auteur : Le nom de l'auteur du livre.
  - ❖ annee : L'année de publication du livre.
  - ❖ genre : Le genre du livre.
  - ❖ quantiteDisponible : La quantité disponible de ce livre dans la bibliothèque.
- **Constructeur** :

Le constructeur de la classe Livre est utilisé pour initialiser les attributs de l'objet avec les valeurs fournies en paramètres.

➢ Classe LivreConcret

Cette classe étend de la classe abstraite 'Livre', il est utilisé pour créer,

convertir et représenter des livres dans le système de gestion de bibliothèque. Et facilitant ainsi la persistance des données et l'échange d'informations avec d'autres parties du système.

- **L'héritage :**

Elle hérite de tous les attributs et méthodes de la classe de base.

- Constructeur :

Le constructeur de LivreConcret appelle le constructeur de la classe Livre à l'aide du mot-clé super, ce qui permet d'initialiser les attributs de la classe de base.

- **Méthode fromCSV** :

Cette méthode statique est utilisée pour créer une instance de LivreConcret à partir d'une chaîne CSV (Comma-Separated Values). Elle prend une chaîne CSV en entrée, la divise en parties, puis crée et retourne un nouvel objet LivreConcret avec les valeurs extraites.

```
public static LivreConcret fromCSV(String csv) {
   String[] parts = csv.split(",");
   int ID = Integer.parseInt(parts[0]);
   String titre = parts[1];
   String auteur = parts[2];
   int annee = Integer.parseInt(parts[3]);
   String genre = parts[4];
   int quantiteDisponible = Integer.parseInt(parts[5]);
   return new LivreConcret(ID, titre, auteur, annee, genre, quantiteDisponible);
                                                    }
```

La méthode prend une chaîne CSV en paramètre, représentant les attributs d'un livre.

La chaîne CSV est divisée en parties distinctes à l'aide de la méthode split(","). Cela crée un tableau (parts) où chaque élément correspond à une partie de la chaîne séparée par une virgule.

Les valeurs nécessaires pour créer un objet LivreConcret (ID, titre, auteur, année, genre, quantité disponible) sont extraites du tableau parts.

Les valeurs qui sont censées être des entiers (ID et annee) sont converties de String à int à l'aide de Integer.parseInt.

Une nouvelle instance de LivreConcret est créée avec les valeurs extraites

et retournée.

- **Méthode toCSV** :

Cette méthode retourne une représentation CSV de l'objet LivreConcret. Elle concatène les attributs de l'objet avec des virgules pour créer une chaîne CSV.

```
public String toCSV() {
        return ID + "," + titre + "," + auteur + "," + annee + ","
    + genre + "," +quantiteDisponible;
    }
```

Cette méthode retourne une chaîne de caractères qui représente l'objet LivreConcret au format CSV. Les attributs de l'objet (ID, titre, auteur, annee, genre, quantiteDisponible) sont concaténés en une seule chaîne de caractères. Chaque attribut est séparé par une virgule. La chaîne CSV ainsi formée est retournée.

Les méthodes **fromCSV** et **toCSV** dans la classe LivreConcret sont utilisées pour convertir des objets LivreConcret entre leur représentation interne dans le programme et une représentation en chaîne de caractères au format CSV, qui est un format couramment utilisé pour stocker des données tabulaires.

- **Visibilité :**

La classe LivreConcret est déclarée public, ce qui signifie qu'elle peut être utilisée à l'extérieur du package Gestion.

➢ Interface BibliothequeOperation

Cette interface définit un ensemble d'opérations standard que chaque classe de gestion de bibliothèque doit mettre en œuvre. Cela permet de garantir une certaine cohérence dans la manière dont ces opérations sont définies et appelées, indépendamment de la classe concrète qui les implémente.

Les méthodes déclarées dans l'interface représentent des actions courantes que l'on pourrait effectuer dans une bibliothèque :

- ajouterLivre() : Ajoute un nouveau livre à la bibliothèque.
- listerLivres() : Affiche la liste de tous les livres disponibles dans la bibliothèque.
- modifierLivre() : Modifie les détails d'un livre existant dans la bibliothèque.
- supprimerLivre() : Supprime un livre de la bibliothèque.
- rechercherLivre() : Recherche un livre dans la bibliothèque.

La classe qui implémente cette interface fournit une implémentation pour

chacune de ces méthodes. Ainsi, l'interface définit un contrat que la classe concrète doit respecter.

➢ Classe BibliothequeGestion

Cette classe était faite pour gérer les opérations courantes d'une bibliothèque en utilisant les méthodes définies dans l'interface ‘BibliothequeOperations’. Telles que ajouterLivre, listerLivres, modifierLivre, supprimerLivre, et rechercherLivre des opérations liées à la gestion des livres dans la bibliothèque.

A la place d’utiliser une base de données spécifique comme MYSQL on a utilisé un fichier pour sauvgarder les livres ajouter ou bien modifier c’est pour cela on a utilisé les opérations de lecture (FileReader, BufferedReader) et d'écriture (FileWriter) de fichiers.

```
package Gestion;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;
import java.util.Scanner;
```

Cette partie du code définit le package en tant que Gestion et inclut les bibliothèques Java nécessaires pour les opérations sur les fichiers (java.io), la gestion des exceptions et l'entrée utilisateur (java.util).

```
« String genre, int quantiteDisponible) {
   super(ID, titre, auteur, annee, genre, quantiteDisponible);
}
```

Il s'agit d'un constructeur pour la classe GestionBibliotheque, appelant le

constructeur de la superclasse (LivreConcret). On utilise le mot-clé "super" pour appeler le constructeur de la classe parente LivreConcret. Plus précisément, cela signifie que le constructeur de la classe GestionBibliotheque utilise les paramètres fournis pour initialiser les membres de la classe parente (LivreConcret). Cette approche permet de réutiliser le code du constructeur de la classe parente et garantit que les attributs de la classe parente sont correctement initialisés avec les valeurs passées en paramètre lors de la création d'un objet GestionBibliotheque. En d'autres termes, cela indique que la classe GestionBibliotheque hérite du comportement du constructeur de sa classe parente (LivreConcret).

```
public void run() {
        //nos instructions ;
}
```

La méthode run dans notre code est le point d'entrée principal du programme. Dans notre cas, elle est utilisée comme point central d'exécution pour les opérations de gestion de bibliothèque.
Cette méthode semble effectuer les actions suivantes :

- **Charger les livres depuis le fichier** : Appelle la méthode chargerLivres pour initialiser la bibliothèque avec les données stockées dans un fichier.
- **Afficher un menu interactif** : Utilise un objet Scanner pour lire l'entrée de l'utilisateur et affiche un menu interactif proposant plusieurs options telles que l'ajout de livre, la liste des livres, la modification, la suppression, la recherche, et la sortie.
- **Exécuter l'action en fonction du choix de l'utilisateur** : Utilise une structure switch pour appeler les méthodes correspondant aux actions sélectionnées par l'utilisateur (par exemple, ajouterLivre, listerLivres, etc.).
- **Répéter jusqu'à ce que l'utilisateur choisisse de quitter** : Utilise une boucle do-while pour continuer l'exécution du menu tant que l'utilisateur ne choisit pas de quitter (choix 0).
- **Sauvegarder les livres dans le fichier** : Appelle la méthode sauvegarderLivres pour enregistrer les modifications de la bibliothèque dans le fichier.

- **Fermer le scanner :** Ferme l'objet Scanner pour éviter les fuites de

```
public static LivreConcret fromCSV(String csv){
        //nos instructions ;
}
```

ressources.

Cette méthode convertit une chaîne CSV (valeurs séparées par des virgules) en un objet LivreConcret.

```
private void chargerLivres() {
try (BufferedReader br = new BufferedReader(new
FileReader(FICHIER_LIVRES))) {
String line;
while ((line = br.readLine()) != null) {
LivreConcret livreConcret = LivreConcret.fromCSV(line);
bibliotheque[nombreLivres++] = livreConcret;
}
} catch (FileNotFoundException e) {
System.err.println("Le fichier " + FICHIER_LIVRES + " n'existe pas.
La bibliothèque sera initialisée vide.");
} catch (IOException e) {
System.err.println("Erreur lors de la lecture du fichier : " +e.getMessage());
}
}
```

Crée un objet **BufferedReader** pour lire le fichier "livres.txt". Le FileReader est utilisé pour ouvrir le fichier en mode lecture. **BufferedReader** est une classe de la bibliothèque standard Java faisant partie du package java.io. Elle est employée pour lire des données depuis un flux de caractères, tel qu'un fichier texte, tout en offrant des fonctionnalités de mise en mémoire tampon. La mise en mémoire tampon est une technique consistant à stocker temporairement des données dans la mémoire avant de les lire, améliorant ainsi l'efficacité de l'opération de lecture.

Déclare une variable "line" pour stocker chaque ligne lue depuis le fichier.

La boucle while dans cette partie sert à lire chaque ligne du fichier jusqu'à ce qu'il n'y ait plus de lignes à lire.

bibliotheque[nombreLivres++] = livreConcret;: Ajoute l'objet LivreConcret à la position actuelle du tableau bibliotheque et incrémente le compteur nombreLivres.

On a également utilisé des exceptions :

FileNotFoundException: Capture une exception si le fichier "livres.txt" n'est pas trouvé. Dans ce cas, un message d'erreur est affiché.

IOException: Capture une exception liée à une erreur lors de la lecture du fichier. Un message d'erreur est affiché, indiquant la nature de l'erreur.

Cette méthode gère la lecture des informations sur les livres à partir d'un fichier, convertissant chaque ligne en un objet **LivreConcret** et les ajoutant au tableau **bibliotheque**. Les erreurs éventuelles, telles que l'absence du fichier, sont gérées de manière appropriée.

```
private void sauvegarderLivres() {
            try (FileWriter writer = new FileWriter(FICHIER_LIVRES)) {
               for (int i = 0; i < nombreLivres; i++) {
                  writer.write(((LivreConcret) bibliotheque[i]).toCSV());
                  writer.write(System.lineSeparator());
               }
            } catch (IOException e) {
               System.err.println("Erreur lors de l'écriture dans le fichier : " +
   e.getMessage());
            }
         }
```

Crée un objet **FileWriter** qui sera utilisé pour écrire dans le fichier "**livres.txt**". La syntaxe **try-with-resources** est utilisée ici, ce qui garantit que le **FileWriter** sera correctement fermé après utilisation, même en cas d'exception.

Itère sur le tableau **bibliotheque** pour chaque livre présent. On utilise la méthode **toCSV()** de la classe **LivreConcret** (qui devrait retourner une représentation CSV du livre) pour écrire les informations du livre dans le fichier. **writer.write(System.lineSeparator());** sert a ajouté une

nouvelle ligne après chaque livre pour séparer les enregistrements.

IOException: Attrape une exception en cas d'erreur lors de l'écriture dans le fichier.

System.err.println("Erreur lors de l'écriture dans le fichier : " + e.getMessage()); : sert a affiché un message d'erreur indiquant qu'il y a eu une erreur lors de l'écriture dans le fichier, suivi du message d'erreur spécifique obtenu avec e.getMessage().

Le fonctionnement de chaque méthode déclarée dans l'interface 'BibliothequeOperations' est décrit comme suit :

- **ajouterLivre:**

Dans cette méthode, l'utilisateur est invité à fournir les informations du livre telles que l'ID, le titre, l'auteur, l'année, le genre et la quantité disponible. Un nouvel objet LivreConcret est créé avec ces informations. Ce nouvel objet est ajouté à la bibliothèque si la limite de taille n'est pas atteinte. Un message indique le succès de l'opération qui est ensuite affiché.

- **listerLivres:**

Si la bibliothèque contient des livres, cette méthode affiche la liste de tous les livres avec leur titre, auteur et année de publication. Sinon, elle indique que la bibliothèque est vide.

- **modifierLivre:**

L'utilisateur est invité à fournir le titre du livre qu'il souhaite modifier. Si le livre est trouvé dans la bibliothèque, l'utilisateur peut saisir de nouvelles informations pour le livre (titre, auteur, année, genre, quantité disponible). Les informations du livre sont mises à jour, et un message indique le succès de l'opération. Si le livre n'est pas trouvé, un message informe l'utilisateur que le livre n'est pas dans la bibliothèque.

- **supprimerLivre:**

L'utilisateur est invité à fournir le titre du livre qu'il souhaite supprimer. Si le livre est trouvé dans la bibliothèque et a une quantité disponible supérieure à zéro, il est supprimé. Les livres suivants dans la liste sont déplacés d'une position vers la gauche pour remplir l'emplacement supprimé. La quantité totale de livres dans la bibliothèque est mise à jour, et un message indique le succès de

l'opération. Si le livre n'est pas trouvé, un message informe l'utilisateur que le livre n'est pas dans la bibliothèque. Si le livre a une quantité disponible de zéro, il ne peut pas être supprimé, et un message en informe l'utilisateur.

- **rechercherLivre:**

L'utilisateur est invité à fournir le titre du livre qu'il souhaite rechercher. Si le livre est trouvé dans la bibliothèque, ses détails (titre, auteur, année) sont affichés. Si le livre n'est pas trouvé, un message informe l'utilisateur que le livre n'est pas dans la bibliothèque.

➢ Class Main

Est la méthode d'entrée principale du programme Java. Elle est exécutée lorsque le programme démarre. Cette méthode crée une instance de la classe GestionBibliotheque appelée gestionBibliotheque. Cette instance est créée avec des valeurs spécifiques passées en arguments au constructeur :

- ID: 1
- Titre: "Java Book"
- Auteur: "John Doe"
- Année: 2023
- Genre: "Programming"
- Quantité Disponible: 10

L'utilisation d'une instance préexistante permet de démarrer l'application avec des données déjà présentes, ce qui peut être utile pour une démo ou des tests. Cela offre la possibilité de montrer comment l'application fonctionne avec un exemple concret. Les utilisateurs pourraient commencer avec au moins un livre dans la bibliothèque plutôt que de commencer à partir de zéro. L'instance assure qu'il y a au moins un point de données avec lequel l'application peut interagir. Exécution de la méthode 'run' de l'instance de GestionBibliotheque est appelée. Cette méthode semble être la méthode principale qui gère le flux d'exécution du programme. Gère l'interaction avec l'utilisateur, en lui permettant de choisir différentes options telles que l'ajout de livres, la modification, la suppression, la recherche, etc. Le programme continue d'exécuter ces opérations jusqu'à ce que l'utilisateur choisisse de quitter (option 0). L'interaction avec l'utilisateur semble d'utiliser la classe Scanner pour lire les entrées depuis la console.

VII. Démonstration de l'exécution du code dans la console

Une fois le programme lancé, vous serez invité par un menu interactif offrant différentes options.

```
Main (1) [Java Application] C:\Program Files\Java\jdk-20\bin\javaw.exe  (22 déc. 2023, 19:53:09) [pid: 13792]
1. Ajouter un livre
2. Liste des livres
3. Modifier un livre
4. Supprimer un livre
5. Rechercher un livre
0. Quitter
Choisissez une option :
```

Figure 9 : Menu du choix

Si on a choisi l'option 1 pour ajouter un livre. On suive les instructions pour entrer les détails du livre.

```
Choisissez une option : 1
ID du livre : 5
Titre du livre : The C Programming Language
Auteur du livre : Brian W. Kernighan
Année de publication : 1978
Genre du livre : Programmation
Quantité disponible : 1
```

Figure 10 : Ajouter un livre

Si on a choisi l'option 2 pour afficher la liste des livres ajoutés.

```
Choisissez une option : 2
Liste des livres dans la bibliothèque:
1. The Soul of A New Machine  par Tracy Kidder (Année : 1981)
2. Antigone      par Pavelove (Année : 1950)
3. Boite à merveille par Mohammed SEFRIOUI (Année : 1900)
4. The C Programming Language par Brian W. Kernighan (Année : 19
```

Figure 11 : Afficher un livre

Si on a choisi l'option 3 pour modifier un livre.

```
Choisissez une option : 3
Entrez le titre du livre à modifier : Boite à merveille
Nouveau titre : Boite à merveille
Nouvel auteur : MOHAMMED SEFRIOUI
Nouvelle année de publication : 1900
Nouveau genre : xxxxxx
Nouvelle quantité disponible : 2
Livre modifié avec succès.
```

Figure 12 : Modifier un livre

On vérifie est ce que la modification était faite par le listage des livres donc on choisit l'option 2 pour lister et voir est ce que la modification était bien faite :

```
Choisissez une option : 2
Liste des livres dans la bibliothèque:
1. The Soul of A New Machine  par Tracy Kidder (Année : 1981)
2. Antigone      par Pavelove (Année : 1950)
3. Boite à merveille par MOHAMMED SEFRIOUI (Année : 1900)
4. The C Programming Language par Brian W. Kernighan (Année : 19
```

Figure 13 : Vérification de la modification d'un livre

Si on a choisi l'option 4 pour supprimer un livre.

```
Choisissez une option : 4
Entrez le titre du livre à supprimer : Boite à merveille
Livre supprimé avec succès.
```

Figure 14 : Supprimer un livre

On vérifier :

```
Liste des livres dans la bibliothèque:
1. The Soul of A New Machine  par Tracy Kidder (Année : 1981)
2. Antigone      par Pavelove (Année : 1950)
3. The C Programming Language par Brian W. Kernighan (Année : 19
```

Figure 15 : Vérification de la suppression d'un livre

On constate que le livre était supprimer.

Si on a choisi l'option 5 pour chercher est ce que le livre est disponible ou nom :

```
Choisissez une option : 5
Entrez le titre du livre à rechercher : Boite à merveille
Livre non trouvé dans la bibliothèque.
```

```
Choisissez une option : 5
Entrez le titre du livre à rechercher : The C Programming Language
Livre trouvé dans la bibliothèque:
Titre : The C Programming Language
Auteur : Brian W. Kernighan
Année de publication : 1978
```

Figure 16 : Rechercher un livre

Si on a choisi l'option 0 on va quitter le programme :

```
1. Ajouter un livre
2. Liste des livres
3. Modifier un livre
4. Supprimer un livre
5. Rechercher un livre
0. Quitter
Choisissez une option : 0
Programme terminé.
```

Figure 17 : Quitter le programme

Si un numéro est entré qui ne figure pas parmi les options proposées par le programme, un message d'erreur s'affichera, invitant l'utilisateur à saisir à nouveau son choix.

```
Main (1) [Java Application] C:\Program Files\Java\jdk-20\bin\javaw.exe (22 déc. 2023, 22:56:17) [pid: 14256]
1. Ajouter un livre
2. Liste des livres
3. Modifier un livre
4. Supprimer un livre
5. Rechercher un livre
0. Quitter
Choisissez une option : 6
Option invalide. Veuillez réessayer.
1. Ajouter un livre
2. Liste des livres
3. Modifier un livre
4. Supprimer un livre
5. Rechercher un livre
0. Quitter
Choisissez une option :
```

Figure 18 : Autre choix (choix invalide)

Chapitre 3 : Réalisation du mini projet dans l'interface graphique en utilisant JavaFX

VIII. Explication du code

Pour la réalisation du projet Système de Gestion de Bibliothèque en java avec une interface graphique on a utilisé JavaFX (Pour simplifier la conception de l'interface utilisateur) et SceneBuilder (un outil graphique, peut être intégré au processus de développement).

On a créé plusieurs class pour le bon fonctionnement de notre projet les suivant :

- Class Livre.java ;
- Fichier Main.fxml ;
- Class MainController.java;
- Class Main.java;
- Class Application.css.

➢ Classe Livre.java

```
package application; import java.io.Serializable;
public class Livre implements Serializable {
private int id; private String titre;
private String auteur; private String genre; private int quantite;
public Livre(int id, String titre, String auteur, String genre, int quantite) {
this.id = id;
this.titre = titre;
this.auteur = auteur;
this.genre = genre;
this.quantite = quantite;
}
public int getId() {
return id;
}
public String getTitre() {
return titre;
}
public String getAuteur() {
 return auteur;
}
public String getGenre() {
return genre;
}
public int getQuantite() {
return quantite;
}
```

```
public void setId(int id) {
    this.id = id;
}

public void setTitre(String
    titre) {this.titre =
    titre;
}

public void setAuteur(String
    auteur) { this.auteur =
    auteur;
}

public void setGenre(String
    genre) {this.genre =
    genre;
}

public void setQuantite(int
```

Cette classe Livre représente un objet de type livre qui implémente l'interface Serializable (qui signifie que ses instances peuvent être sérialisées (converties en une séquence d'octets) pour être stockées dans des fichiers ou transmises sur le réseau.). Avec des attributs tels que l'identifiant (type entier), le titre (type String), l’auteur (type String), le genre (type String) et la quantité (type entier) disponible.

Le constructeur de la classe prend en paramètres les valeurs initiales pour les attributs id, titre, auteur, genre, et quantite et initialise les propriétés de l'objet Livre.

Contient aussi les méthodes publiques **(getId, getTitre, getAuteur, getGenre, getQuantite)** permettent d'accéder aux valeurs des attributs. Et les méthodes publiques **(setId, setTitre, setAuteur, setGenre, setQuantite)** permettent de modifier les valeurs des attributs.

- **Ficher Main.fxml**

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TableColumn?>
<?import javafx.scene.control.TableViw?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.text.Font?>

<AnchorPane id="anchorPane" fx:id="anchorPane" maxHeight="-Infinity" maxWidth="-Infinity" prefHeight="600.0" prefWidth="800.0" stylesheets="application.application.css" xmlns="http://javafx.com/javafx/21" xmlns:fx="http://javafx.com/fxml/1" fx:controller="application.MainController">
<children>
<VBox layoutX="22.0" layoutY="64.0" prefHeight="271.0" prefWidth="73.0" spacing="15.0">
<children>
<Label text="id">
<font>
<Font name="System Bold Italic" size="14.0" />
</font>
</Label>
<Label text="titre">
<font>
<Font name="System Bold Italic" size="14.0" />
</font>
</Label>
<Label text="auteur">
<font>
<Font name="System Bold Italic" size="14.0" />
</font>
</Label>
<Label text="genre"> /*Voir la suite de code dans le projet Fichier Main.fxml*/
```

Ce fichier FXML définit la structure de l'interface utilisateur de votre application JavaFX, avec des étiquettes, des champs de texte, une table, et des boutons pour interagir avec des instances de la classe Livre. La logique associée à cette interface utilisateur est probablement gérée par la classe MainController.

Les éléments clés dans ce code on a **<AnchorPane>**(Un conteneur qui peut être utilisé pour organiser d'autres éléments graphiques. Il s'ajuste automatiquement à la taille de ses enfants.), **<VBox>**(Un conteneur vertical qui organise ses enfants les uns en dessous des autres. Il est utilisé ici pour organiser des étiquettes (Label) affichant des noms de champs.), **<Label>** (Étiquettes de texte utilisées pour afficher les noms de champs tels que "id", "titre", "auteur", "genre", "quantite".),**<TextField>** (Champs de texte qui permettent à l'utilisateur d'entrer des données. Chaque champ de texte est associé à un attribut de la classe (par exemple, fx:id="tfid" est associé à l'attribut tfid de la classe).),

<TableView> (Une table utilisée pour afficher des données sous forme de lignes et de colonnes. Elle a plusieurs colonnes (<TableColumn>) correspondant aux attributs de la classe Livre tels que "id", "titre", "auteur", "genre", "quantite".)

<TableColumn> (Colonnes de la table avec des en-têtes comme "id", "titre", "auteur", "genre", "quantite".), **<Button>** (Boutons permettant à l'utilisateur d'effectuer des actions telles que l'ajout, la suppression, la modification, la recherche et la liste des livres. Chaque bouton est associé à une méthode (onAction="#handButtonAction") qui sera appelée lorsqu'il est pressé.), **fx:controller="application.MainController"** (Spécifie la classe de contrôleur associée à cet élément FXML. Le contrôleur est responsable de la logique de l'interface utilisateur.), **stylesheets="application.application.css"** (Référence à une feuille de style CSS qui peut être utilisée pour styliser l'interface utilisateur).

➢ Main Controller.java

```
package application;
import javafx.scene.layout.BackgroundSize;
.....
import java.io.*;
import java.util.ArrayList;
import java.util.List;
public class MainController { @FXML
private AnchorPane anchorPane; @FXML
private TextField tfid; @FXML
private TextField tftitre; @FXML
private TextField tfauteur; @FXML
private TextField tfgenre;
/*Suite de code est dans la classe
```

Le code Java présent constitue le contrôleur d'une application JavaFX permettant la gestion de livres. L'interface utilisateur, définie dans un fichier FXML, inclut des champs de texte, une table pour afficher les livres, et des boutons pour effectuer des actions telles que l'ajout, la suppression, la modification et la liste des livres. Les éléments de l'interface sont injectés dans le contrôleur via des annotations @FXML. La méthode initialize configure la table, charge une image en arrière-plan, et initialise les colonnes de la table. Les boutons sont associés à des méthodes telles que ajouterLivre(), modifierLivre(), supprimerLivre() et listerLivres(). Ces méthodes manipulent une liste observable de livres, mettent à jour l'interface utilisateur et utilisent la sérialisation pour sauvegarder et charger les données depuis un fichier. L'observabilité de la liste assure la synchronisation automatique avec la table. L'application gère également la sélection d'une ligne dans la table pour afficher les détails du livre correspondant dans les champs de texte. En résumé, ce contrôleur joue le rôle d'intermédiaire entre l'interface utilisateur et la logique de gestion des livres, assurant une interaction fluide et une persistance des données.

➢ Classe Main

```
package application;
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.BorderPane;
public class Main extends Application {
	@Override
	public void start(Stage primaryStage) {
		try {
			BorderPane root = new BorderPane();
			 Scene scene = new Scene(root,400,400);
		scene.getStylesheets().add(getClass().getResource("application.
	css").toExternalForm());
			primaryStage.setScene(scene);
			priamaryStage.show();
		} catch(Exception e) {
			 e.printStackTrace();}}
	public static void main(String[] args) {
		launch(args);}}
```

Ce code Java représente la classe principale d'une application JavaFX. En héritant de la classe Application, il définit la méthode start qui est appelée lors du lancement de l'application. Un objet BorderPane est créé pour servir de conteneur principal, mais le contenu spécifique n'est pas défini dans ce code. Une scène est ensuite créée, associant le BorderPane et définissant une taille initiale de 400x400 pixels. Une feuille de style CSS externe, "application.css", est chargée pour la stylisation visuelle. La scène est ensuite associée à la fenêtre principale (primaryStage), qui est finalement affichée à l'écran. Bien que la structure de base de l'application soit mise en place, le code actuel ne spécifie pas le contenu réel de l'interface utilisateur. Pour une fonctionnalité complète, des éléments tels que des panneaux, des boutons, ou d'autres composants graphiques devraient être ajoutés au BorderPane.

IX. Démonstration de l'exécution du code dans l'interface graphique

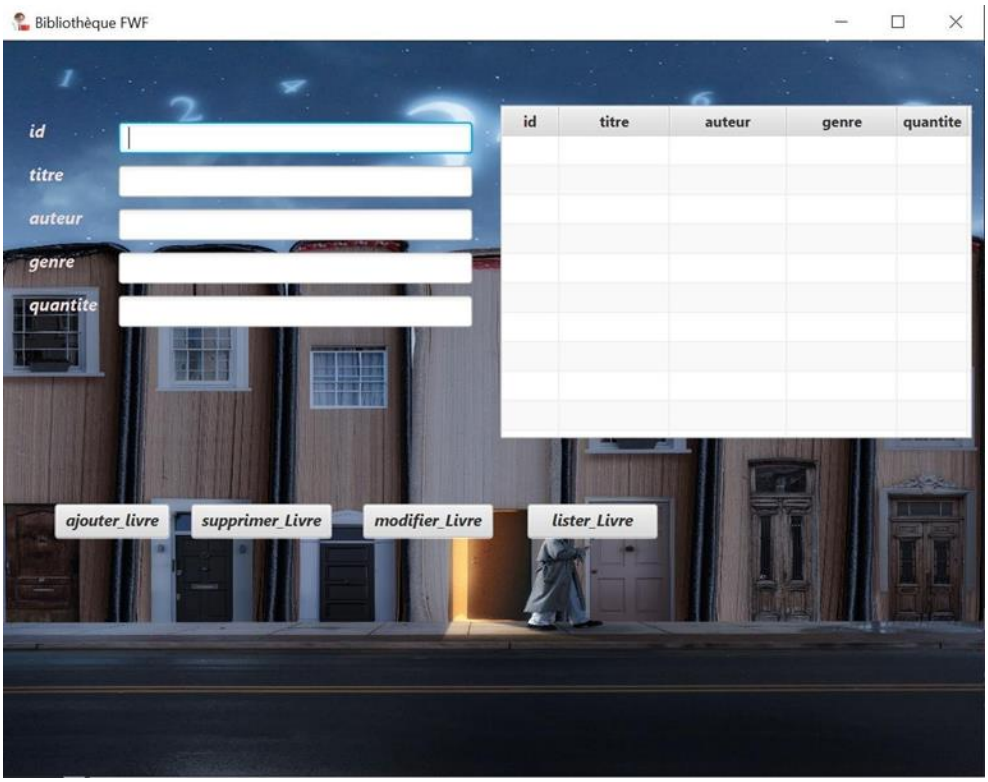

Figure 19 : Page principale de l'interface graphique

Lorsqu'on veut ajouter un nouveau livre ont rempli tous les champs (id, titre, auteur, genre et quantité).

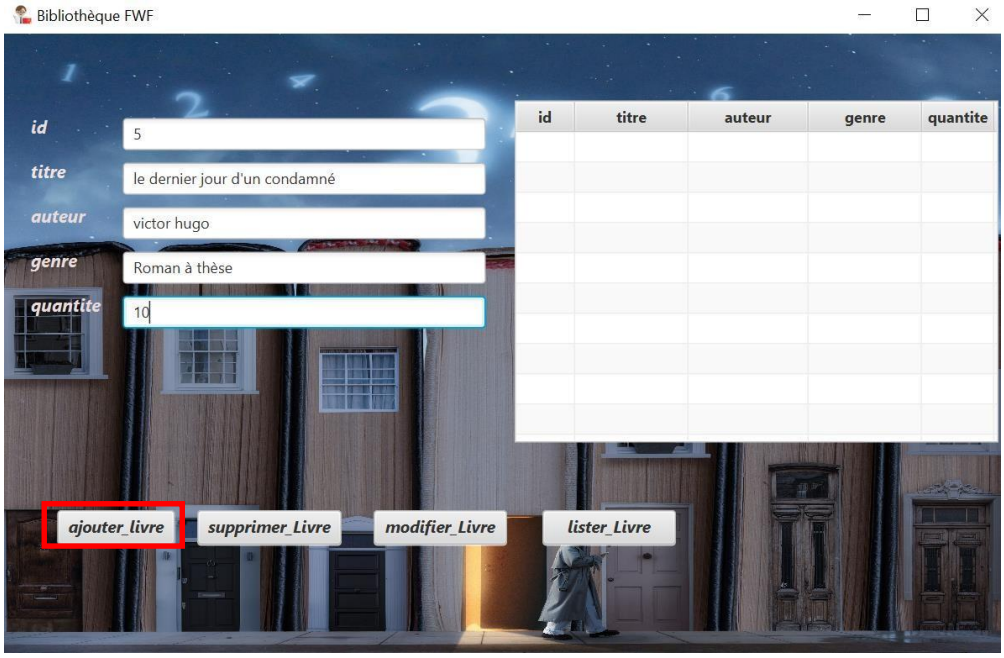

Figure 20 : Ajouter_Livre

Après le remplissage de tous les champs, on clique sur ajouter_livre, on constate que le livre est ajouté au tableau.

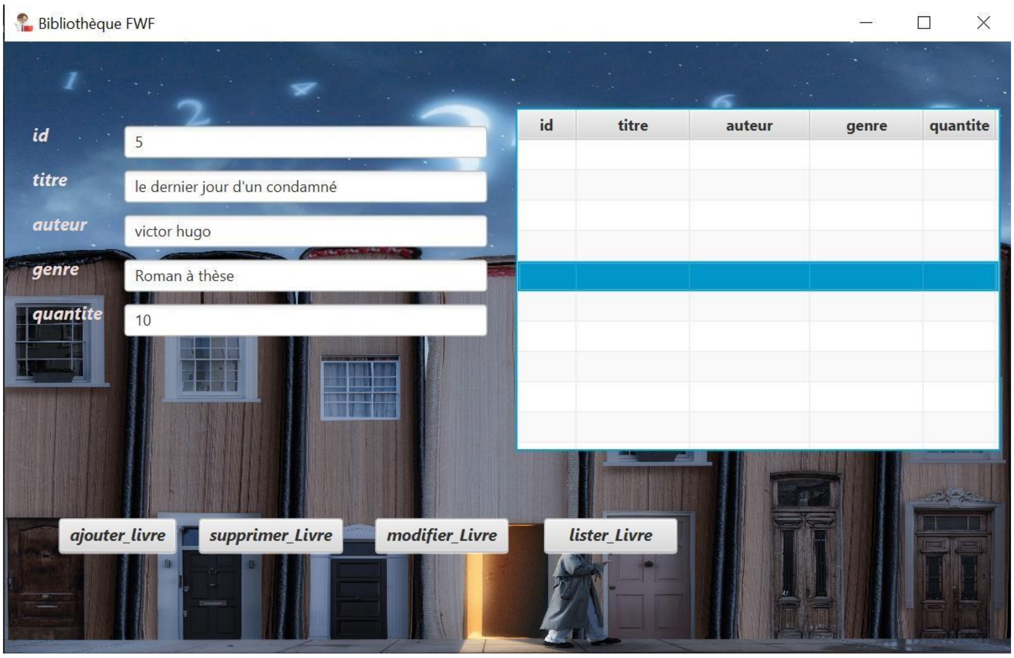

Figure 21 : Le livre est ajouté

Antigone est déjà un livre dans la bibliothèque (tableau), pour le supprimé, on doit le sélectionner puis ont choisi supprimer livre.

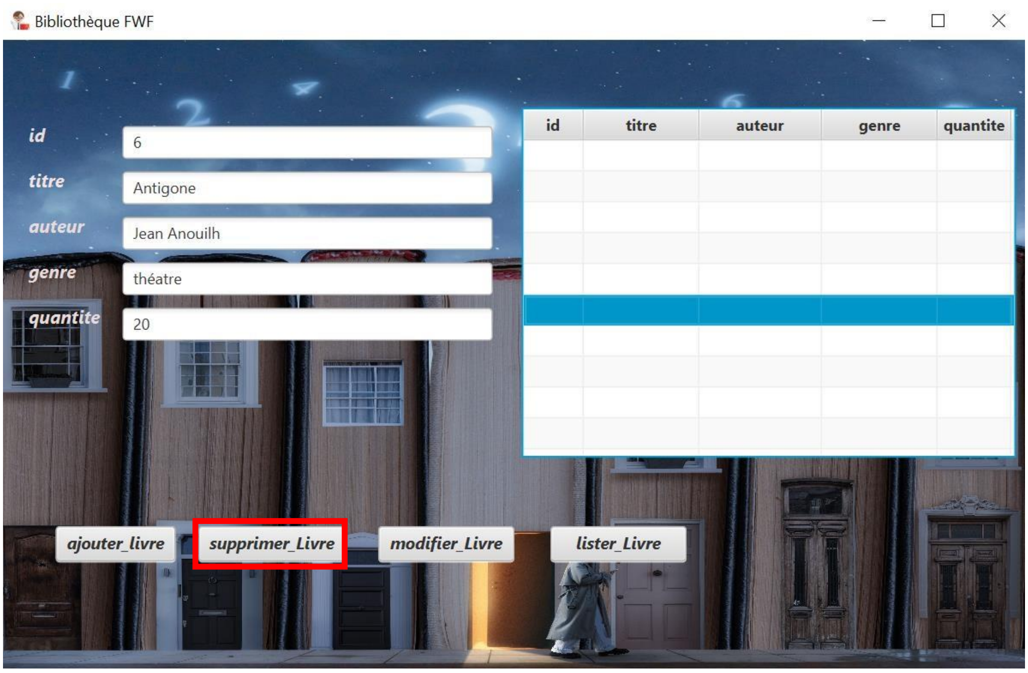

Figure 22 : Supprimer_Livre

Pour la modification d'un livre, on le sélectionne pour le modifié comme vous voyez on a modifié le titre de livre, le nom de l'auteur on majuscule et la quantité.

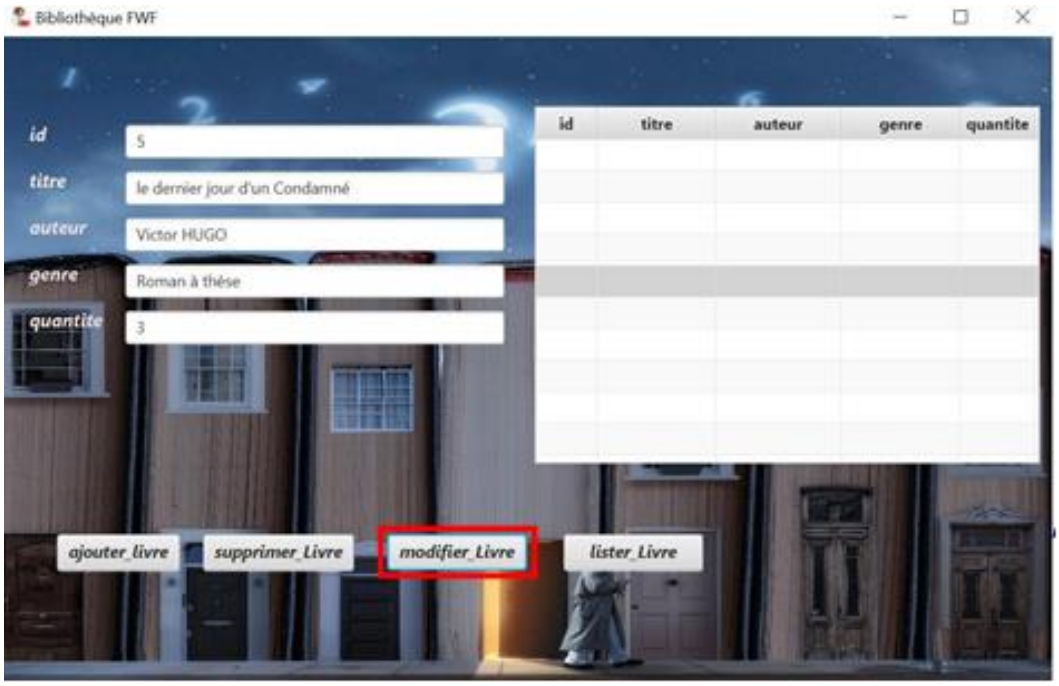

Figure 23 : Modifier_Livre

Conclusion

En conclusion, ce mini-projet de Système de Gestion de Bibliothèque en Java a été une expérience captivante, combinant la puissance de la programmation orientée objet, l'interaction utilisateur avec JavaFX, et la modélisation efficace avec UML.

L'objectif de créer un système de gestion bibliothécaire a été atteint avec succès, offrant des fonctionnalités essentielles telles que l'ajout, la modification, la suppression et le listing de livres. Les outils choisis, notamment Java, JavaFX, Eclipse et Scene Builder, ont démontré leur utilité dans la réalisation d'une application fonctionnelle et conviviale.