POO en Java – Projet

Contexte :

Ce projet Java a été réalisé dans le cadre de la pratique des concepts de la Programmation Orientée Objet (POO), incluant :

- ❖ Héritage, encapsulation, abstraction
- ❖ Gestion des exceptions
- ❖ Manipulation de collections
- ❖ Utilisation d'une interface graphique simple (Swing)

Objectif :

L'objectif principal du projet est de mettre en œuvre les principaux concepts de la POO ainsi que la gestion des exceptions.

Thème :

Gestion d'une médiathèque

L'application simule une médiathèque qui propose différents supports culturels (livres, CD, DVD) que les utilisateurs peuvent consulter ou emprunter. L'interface permet la gestion de ces supports via un menu interactif avec des boutons.

Fonctionnalités principales :

- Ajout initial de documents (dans le main).
- Affichage de tous les documents : empruntés, disponibles.
- Recherche de documents par titre.
- Filtrage par type de support (Livre, CD, DVD).
- Gestion des emprunts par utilisateur (saisie nom & prénom).
- Interface utilisateur simple avec Java Swing.

Outils utilisés :

- Java 21.0.5
- Java Swing
- Collections (List, ArrayList)
- POO (classes abstraites, héritage, interfaces)
- Architectures-en packages

Arborescence du projet & diagramme UML :

```
projet_poo_java_POLUTELE_DYLAN
├── bin
│   └── mediatheque
│       ├── model
│       └── ui
└── src
    └── mediatheque
        ├── model
        │   ├── CD.java
        │   ├── Document.java
        │   ├── DocumentDejaEmprunteException.java
        │   ├── DocumentNonEmprunteException.java
        │   ├── DVD.java
        │   ├── Livre.java
        │   └── Utilisateur.java
        └── ui
            └── Main.java
```

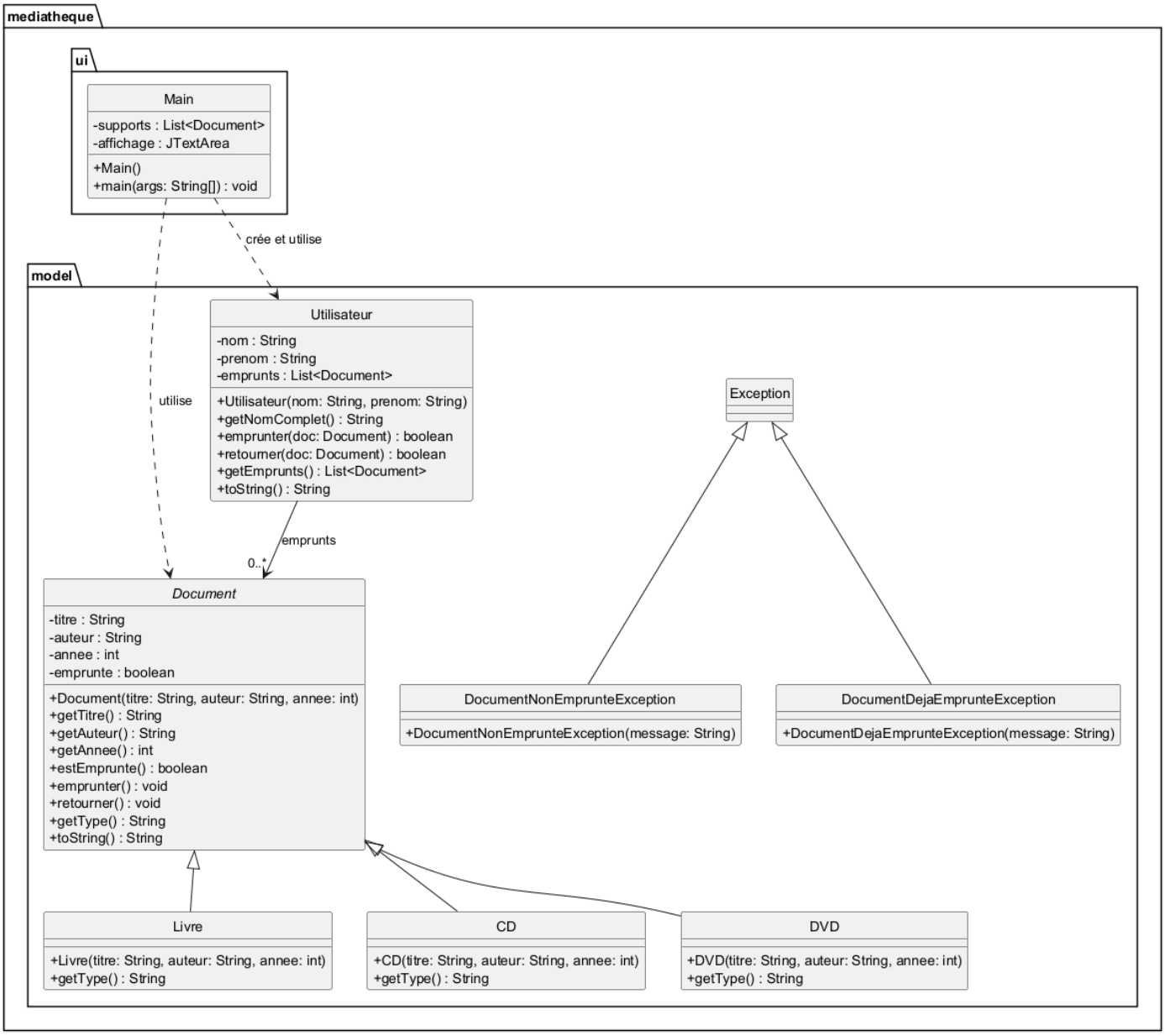

Illustration des concepts développés :

- **Héritage :** La classe abstraite Document sert de base pour les différentes catégories de documents. Les classes Livre, CD, et DVD héritent de Document, partageant ainsi des attributs communs tout en ajoutant leurs spécificités.

- **Encapsulation :** Les attributs des classes sont déclarés en private ou protected, et l'accès à ces attributs se fait via des méthodes getters et setters, assurant ainsi le contrôle sur la modification des données interne (exemple : dans la classe Document, l'encapsulation est utilisée pour protéger les données comme le titre, l'auteur ou l'état d'emprunt du document. Ces informations sont privées et accessibles uniquement via des méthodes (getters).

- **Abstraction :** La classe Document est une classe abstraite qui représente un document de manière générale, sans en définir les détails concrets. Elle contient une méthode abstraite getType() que chaque sous-classe (comme Livre, CD, etc.) doit implémenter. Cela permet de manipuler différents types de documents de façon uniforme, tout en laissant chaque sous-classe définir son propre comportement spécifique.

- **Gestion des exceptions :** La gestion des erreurs est faite proprement avec des exceptions personnalisées DocumentDejaEmprunteException qui est levée quand on essaie d'emprunter un document déjà emprunté et DocumentNonEmprunteException qui est levée quand on tente de retourner un document qui n'est pas emprunté, ces exceptions sont capturées et gérées dans la classe Utilisateur, avec un message affiché en cas d'erreur.

- **Manipulation des collections :** La classe Utilisateur utilise une collection Java (ArrayList) qui stocke tous les documents empruntés dans une liste emprunts. Les méthodes emprunter() et retourner() ajoutent ou retirent des documents de cette liste, cela permet de gérer dynamiquement les emprunts, de les consulter et de les modifier facilement.

- **Interface graphique simple :** Dans ce projet, l'interface utilisateur est gérée dans le package ui, plus précisément dans la classe Main.java. Elle repose sur Swing, et utilise principalement JOptionPane pour créer des boîtes de dialogue simples (messages, saisies, confirmations). J'ai choisi cette approche car j'ai l'habitude de l'utiliser dans les projets de génie logiciel, et elle me permet de créer rapidement une interface utilisateur claire sans complexité inutile.

Conclusion :

Ce projet m'a permis de mettre en pratique les bases de la Programmation Orientée Objet (POO) dans un cas simple et concret. J'ai appliqué des notions clés comme l'héritage, l'encapsulation, l'abstraction, la gestion des exceptions, et la manipulation de collections.

La réalisation de ce projet m'a aidé à mieux comprendre comment structurer un code orienté objet dans un contexte précis.

Le fait de travailler en parallèle sur un autre projet orienté POO m'a permis de renforcer ma compréhension, ce qui m'a aidé à avancer plus efficacement sur cette médiathèque.