Rapport du Projet : Système de Gestion de Bibliothèque en Python

Nom & PrenomFilière : BEN DAMOU Mohammed
Filière : GI3 - ENSAO 2024/2025
Matière : Programmaton Avancée en Python

Introduction

Ce rapport détaille la conception et l'implémentation d'un système de gestion de bibliothèque développé en Python. Le projet vise à fournir une solution complète pour la gestion des livres, des membres, des emprunts et retours, ainsi que des outils de visualisation pour les données de la bibliothèque. L'application est dotée d'une interface utilisateur graphique (GUI) construite avec `Tkinter`, facilitant l'interaction pour les administrateurs de la bibliothèque.

Architecture du Projet

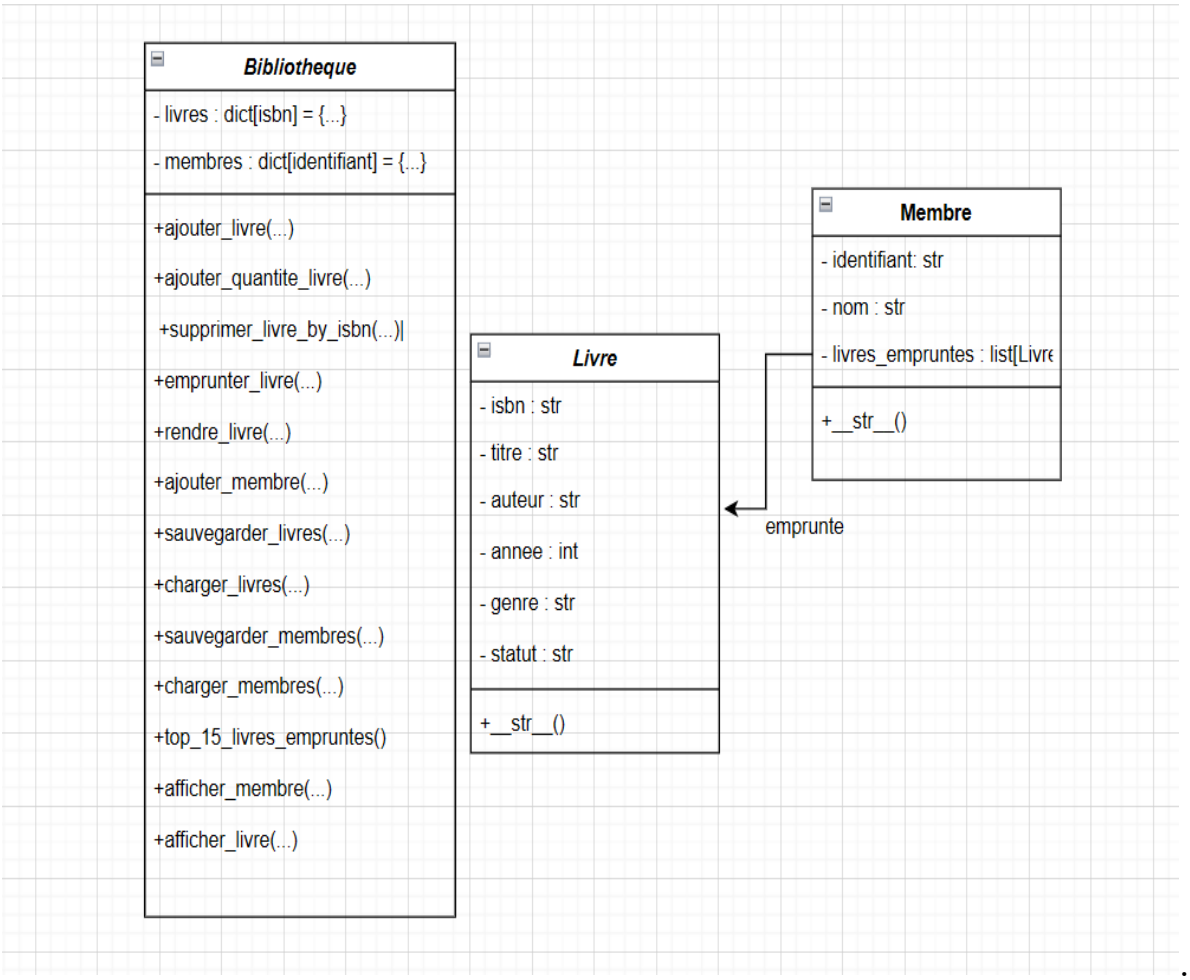
Le projet est structuré en plusieurs modules Python, chacun ayant une responsabilité spécifique, favorisant ainsi la modularité et la maintenabilité du code.

Modules Principaux :

- **`livre.py`**: Définit la classe `Livre`.
- **`membre.py`**: Définit la classe `Membre`.
- **`bibliotheque.py`**: Contient la logique principale de la bibliothèque, gérant les interactions entre les livres et les membres.
- **`exceptions.py`**: Centralise la définition des exceptions personnalisées pour une gestion d'erreurs robuste.
- **`visualisation.py`**: Gère la création de graphiques pour visualiser les données de la bibliothèque.
- **`main.py`**: Implémente l'interface utilisateur graphique (GUI) en utilisant `Tkinter` et intègre toutes les fonctionnalités du backend.

Diagramme de Classes UML

Voici le diagramme de classes UML représentant la structure des principales entités et leurs relations au sein du système. Ce diagramme illustre les classes, leurs attributs, leurs méthodes et les liens qui les unissent.



Détails de l'Implémentation

1. Classe `Livre`

Le fichier `livre.py` définit la classe `Livre`, qui représente un livre dans le système de la bibliothèque.

- **Attributs**:
  - `isbn`: Numéro ISBN unique du livre.
  - `titre`: Titre du livre.
  - `auteur`: Auteur(s) du livre.

    - `annee`: Année de publication.
    - `genre`: Genre littéraire du livre.
    - `statut`: Statut de disponibilité du livre ("disponible" ou "indisponible").
- **Méthodes spéciales**:
    - `__str__` et `__repr__`: Permettent une représentation textuelle conviviale des objets `Livre`.

2. Classe `Membre`

Le fichier `membre.py` définit la classe `Membre`, représentant un utilisateur ou un membre de la bibliothèque.

- **Attributs**:
    - `identifiant`: Identifiant unique du membre.
    - `nom`: Nom complet du membre.
    - `livres_empruntes`: Une liste des objets `Livre` actuellement empruntés par le membre.
- **Méthodes**:
    - `emprunter(livre)`: Ajoute un livre à la liste `livres_empruntes` du membre.
    - `rendre(livre)`: Retire un livre de la liste `livres_empruntes` du membre.
    - `__str__`: Permet une représentation textuelle conviviale des objets `Membre`.

3. Classe `Bibliotheque`

Le cœur de la logique métier est encapsulé dans la classe `Bibliotheque`, définie dans `bibliotheque.py`. Elle gère les collections de livres et de membres, ainsi que toutes les opérations liées à la gestion de la bibliothèque.

- **Attributs**:
    - `livres`: Un dictionnaire pour stocker les livres, où la clé est l'ISBN et la valeur est un dictionnaire contenant l'objet `Livre`, la `quantite` disponible et un indicateur `est_supprimer`.
    - `membres`: Un dictionnaire pour stocker les membres, où la clé est l'identifiant du membre et la valeur est un dictionnaire contenant l'objet `Membre` et leur `historique_livre_emprunte`.
- **Fonctionnalités et Méthodes Clés**:
    - **Gestion des emprunts et retours**:

        - `emprunter_livre(membre, list_livre)`: Permet à un membre d'emprunter un ou plusieurs livres, gérant les vérifications de disponibilité et mettant à jour les quantités.
        - `rendre_livre(membre, list_livre)`: Permet à un membre de rendre un ou plusieurs livres, mettant à jour les quantités et les statuts.
    - **Gestion des livres**:
        - `ajouter_livre(livre, quantite)`: Ajoute un nouveau livre à la bibliothèque avec une quantité spécifiée.
        - `ajouter_quantite_livre(livre, quantite)`: Augmente la quantité d'un livre existant.
        - `supprimer_livre_by_isbn(isbn)`: Marque un livre comme "supprimé" (non physiquement supprimé pour l'historique) et ajuste sa quantité à zéro.
        - `rechercher_livre(critere, valeur)`: Recherche des livres basés sur un critère (ISBN, titre, auteur, genre, année).
        - `afficher_livres()`: Affiche tous les livres disponibles.
    - **Gestion des membres**:
        - `ajouter_membre(membre)`: Ajoute un nouveau membre à la bibliothèque.
        - `supprimer_membre(identifiant)`: Supprime un membre de la bibliothèque.
        - `rechercher_membre(critere, valeur)`: Recherche des membres par identifiant ou nom.
        - `afficher_membre()`: Affiche tous les membres.
    - **Statistiques et rapports**:
        - `top_15_livres_empruntes(membres, livres)`: Retourne les 15 livres les plus empruntés.
        - `top_15_membres_actifs(membres)`: Retourne les 15 membres les plus actifs (ceux qui ont emprunté le plus de livres).
    - **Persistance des données**:
        - `sauvegarder_livres(chemin)`: Sauvegarde les données des livres dans un fichier JSON.
        - `charger_livres(chemin)`: Charge les données des livres depuis un fichier JSON.
        - `sauvegarder_membres(chemin)`: Sauvegarde les données des membres dans un fichier JSON.
        - `charger_membres(chemin)`: Charge les données des membres depuis un fichier JSON.

4. Exceptions Personnalisées

Le fichier `exceptions.py` définit une série d'exceptions personnalisées pour gérer des scénarios d'erreur spécifiques au domaine de la bibliothèque. Cela permet une gestion d'erreurs plus claire et plus spécifique.

Exemples d'exceptions:

- `LivreEmprunteError`
- `LivreRendreError`
- `LivreAjoutError`
- `LivreQuantityError`
- `LivreSupprimerError`
- `MembreAjoutError`
- `MembreEmpruntError`
- `MembreRendreError`

5. Visualisation des Données

Le module `visualisation.py` utilise la bibliothèque `matplotlib` pour créer des représentations graphiques des données de la bibliothèque.

- **Fonctionnalités**:
  - `visualiser_par_genre(livres)`: Crée un graphique en secteurs montrant la répartition des livres par genre.
  - `visualiser_par_quantite_max(livres)`: Affiche un graphique à barres des livres les plus disponibles.
  - `visualiser_par_quantite_min(livres)`: Affiche un graphique à barres des livres les moins disponibles.
  - `visualiser_membres_plus_actifs(membres, top_n)`: Présente un graphique à barres des membres ayant effectué le plus d'emprunts.
  - `visualiser_auteurs_plus_populaires(Top_15_livres, livres)`: Génère un graphique des auteurs dont les livres sont les plus empruntés.

6. Interface Utilisateur Graphique (GUI) avec `Tkinter`

Le fichier `main.py` est le point d'entrée de l'application graphique. Il utilise `Tkinter` pour construire une interface conviviale.

- **Structure de l'application**:

    - `App(tk.Tk)`: La classe principale de l'application, initialisant la fenêtre principale, la barre de navigation et les différentes vues (frames).
    - `LivreFrame`: Gère l'affichage, l'ajout, la suppression et la modification des quantités de livres. Inclut une barre de recherche et un tableau (Treeview) pour afficher les livres.
    - `MembreFrame`: Gère l'affichage, l'ajout, la suppression des membres et l'historique de leurs emprunts. Elle permet également l'emprunt et le retour de livres via l'interface.
    - `VisualisationFrameLivre1`, `VisualisationFrameLivre2`, `VisualisationFrameMembre`: Intègrent les fonctionnalités de visualisation des données en affichant les graphiques générés par `matplotlib` directement dans l'interface Tkinter.
- **Interactions**: L'interface permet aux utilisateurs de:
    - Ajouter de nouveaux livres avec leurs détails et quantités.
    - Supprimer des livres.
    - Ajouter des quantités à des livres existants.
    - Emprunter et rendre des livres pour les membres.
    - Ajouter et supprimer des membres.
    - Rechercher des livres et des membres.
    - Consulter diverses statistiques via des graphiques (répartition par genre, livres les plus/moins disponibles, membres les plus actifs, auteurs les plus populaires).
    - Les données sont automatiquement sauvegardées et chargées depuis des fichiers JSON.

Gestion des Données

Le projet utilise des fichiers JSON (`data/livres.json` et `data/membres.json`) pour la persistance des données. Cela permet de conserver l'état de la bibliothèque entre les différentes exécutions de l'application.

Conclusion

Ce système de gestion de bibliothèque en Python est une application robuste qui combine une logique backend solide avec une interface utilisateur intuitive. L'utilisation de classes dédiées, d'exceptions personnalisées et de la persistance des données garantit un système fiable et facile à utiliser. Les fonctionnalités de visualisation ajoutent une valeur significative en offrant des aperçus rapides sur l'état de la bibliothèque et les habitudes d'emprunt.