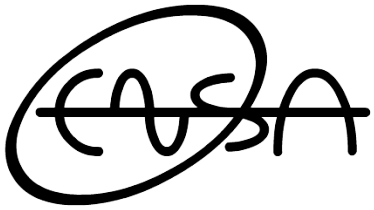
Gestion de Bibliothèque

Projet de Fin de Module en Python



LEMRANI Moad

École Nationale des Sciences Appliquées d'Oujda
Filière Génie Informatique - GI3

Le lien GitHub

Juin 2025

Table des matières

1 Diagramme de classes UML

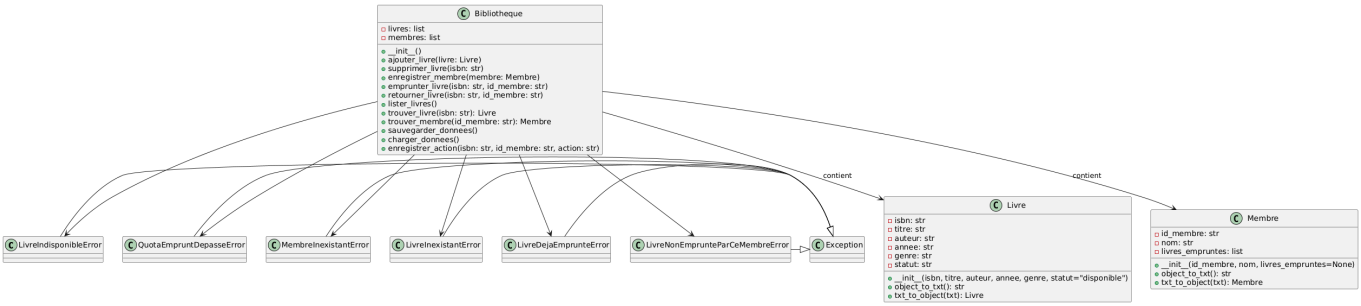


FIGURE 1.1 – Diagramme de classes UML du projet de gestion de bibliothèque

Le diagramme de classes ci-dessus représente l'architecture principale du système de gestion de bibliothèque. Les classes `Livre`, `Membre` et `Bibliothèque` modélisent respectivement les ouvrages, les utilisateurs et le gestionnaire central des opérations. L'utilisation d'exceptions personnalisées permet un traitement précis des erreurs, notamment pour les cas d'emprunts invalides, les dépassements de quota ou les retours non autorisés.

2 Explications des algorithmes clés

Ajout d'un livre

```
for l in self.livres:
    if l.isbn == livre.isbn:
        raise ValueError("Un livre avec cet ISBN existe deja")
self.livres.append(livre)
```

Cette méthode vérifie que l'ISBN est unique avant d'ajouter un nouveau livre.

Suppression d'un livre

```
livre = self.trouver_livre(isbn)
        if livre.statut == "emprunte" :
            raise LivreDejaEmprunteError()
        nouvelle_liste = []
        for l in self.livres:
            if l.isbn != isbn:
                nouvelle_liste.append(l)
        self.livres = nouvelle_liste
```

Cette méthode commence par vérifier si le livre est actuellement emprunté avant de supprimer un livre de la liste en recréant la liste sans le livre ciblé.

Emprunt d'un livre

```
if livre.statut != "disponible":
    raise LivreIndisponibleError()
if len(membre.livres_empruntes) >= 3:
    raise QuotaEmpruntDepasseError()
livre.statut = "emprunte"
membre.livres_empruntes.append(isbn)
```

Cette méthode permet d'emprunter un livre après avoir vérifié sa disponibilité et la capacité d'emprunt du membre.

Retour d'un livre

```
if livre.statut == "disponible":
    raise LivreDejaDisponibleError()
if isbn in membre.livres_empruntes:
    membre.livres_empruntes.remove(isbn)
livre.statut = "disponible"
```

Cette méthode permet de retourner un livre après avoir vérifié qu'il a bien été emprunté par le membre concerné.

Enregistrer un membre

```
for m in self.membres:
            if m.id_membre == membre.id_membre:
                raise ValueError("Un membre avec cet identifiant existe deja")
        self.membres.append(membre)
```

Cette méthode permet d'ajouter un membre tout en vérifiant que son identifiant est unique.

Charger les données

```
if os.path.exists("data/livres.txt"):
            with open("data/livres.txt", encoding="UTF-8") as f:
                livres = []
                for l in f:
                    if l.strip():
                        livre = Livre.txt_to_object(l)
                        livres.append(livre)
                self.livres = livres
```

Cette méthode charge les livres depuis le fichier ‘livres.txt‘ en s’assurant que chaque ligne non vide est convertie en objet ‘Livre‘ avant d’être ajoutée à la liste ‘self.livres‘. Le fichier est lu avec l’encodage UTF-8. (Même algorithme pour charger les membres depuis le fichier membres.txt)

Activité des emprunts sur les 30 derniers jours

```
dates = []
with open("data/historique.csv", encoding="UTF-8") as f:
    reader = csv.reader(f, delimiter=";")
    for row in reader:
        if len(row) < 4:
            continue
        date_str, _, _, action = row
        if action.strip().lower() == "emprunt":
            date = datetime.strptime(date_str.strip(), "%d-%m-%Y")
            if datetime.now() - date <= timedelta(days=30):
                dates.append(date.date())
if not dates:
    print("Aucune activite d'emprunt dans les 30 derniers jours.")
    return
```

Cette méthode collecte les dates des emprunts effectués au cours des 30 derniers jours à partir du fichier historique, afin d’analyser l’activité récente de la bibliothèque.

les 10 auteurs les plus populaires

```
isbn_auteur = {livre.isbn: livre.auteur for livre in livres}
emprunts_par_auteur = Counter({livre.auteur: 0 for livre in livres})
with open(fichier_historique, encoding='UTF-8') as f:
    reader = csv.reader(f, delimiter=';')
    for ligne in reader:
        if len(ligne) != 4:
            continue
        date, isbn, id_membre, action = ligne
        if action.strip().lower() == "emprunt":
            auteur = isbn_auteur.get(isbn)
            if auteur:
                emprunts_par_auteur[auteur] += 1
top_10 = emprunts_par_auteur.most_common(10)
noms = [auteur for auteur, _ in top_10]
freqs = [count for _, count in top_10]

if not noms:
    print("Top 10 des auteurs : Aucun auteur trouve.")
    return
```

Cette méthode identifie les auteurs les plus populaires en comptant combien de fois leurs livres ont été empruntés dans l’historique.

3 Captures d'écran des visualisations

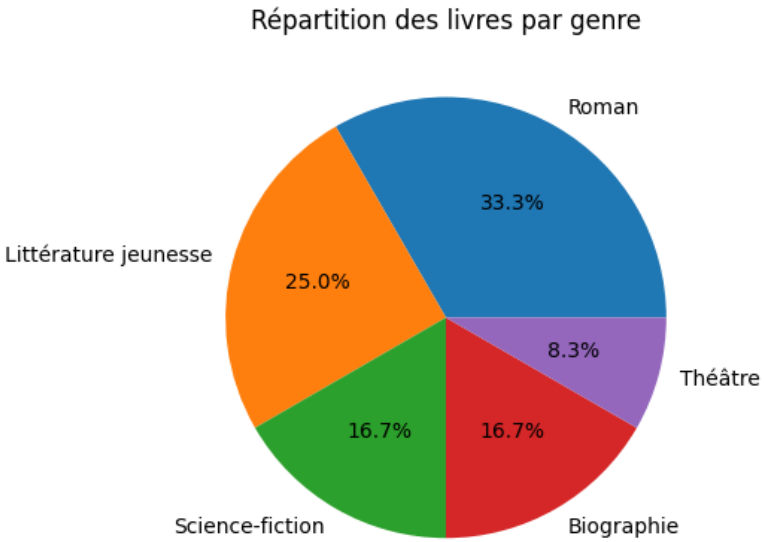


FIGURE 3.1 – Répartition des livres par genre

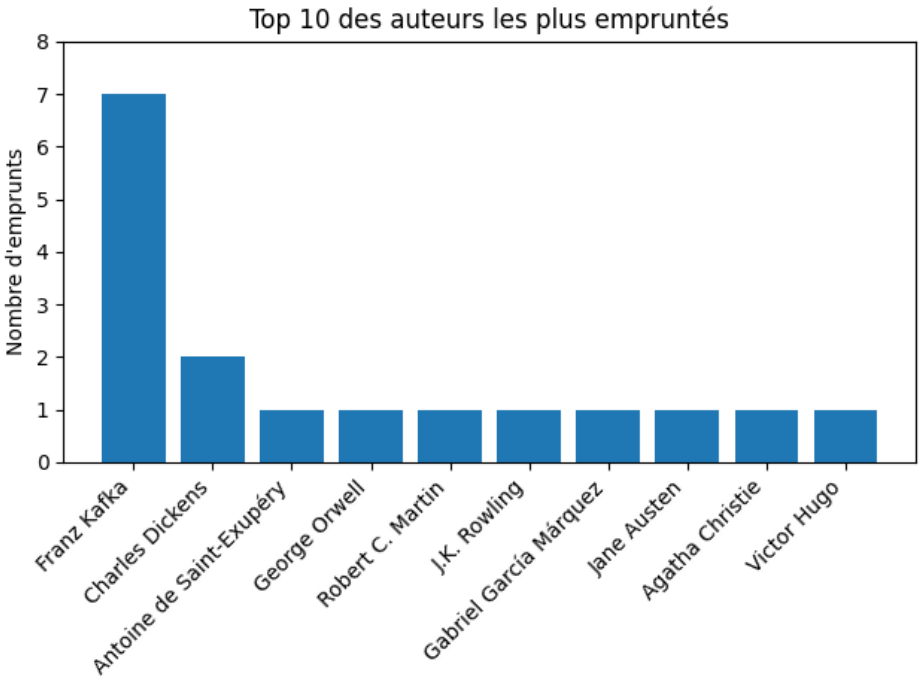


FIGURE 3.2 – Top 10 des auteurs les plus populaires

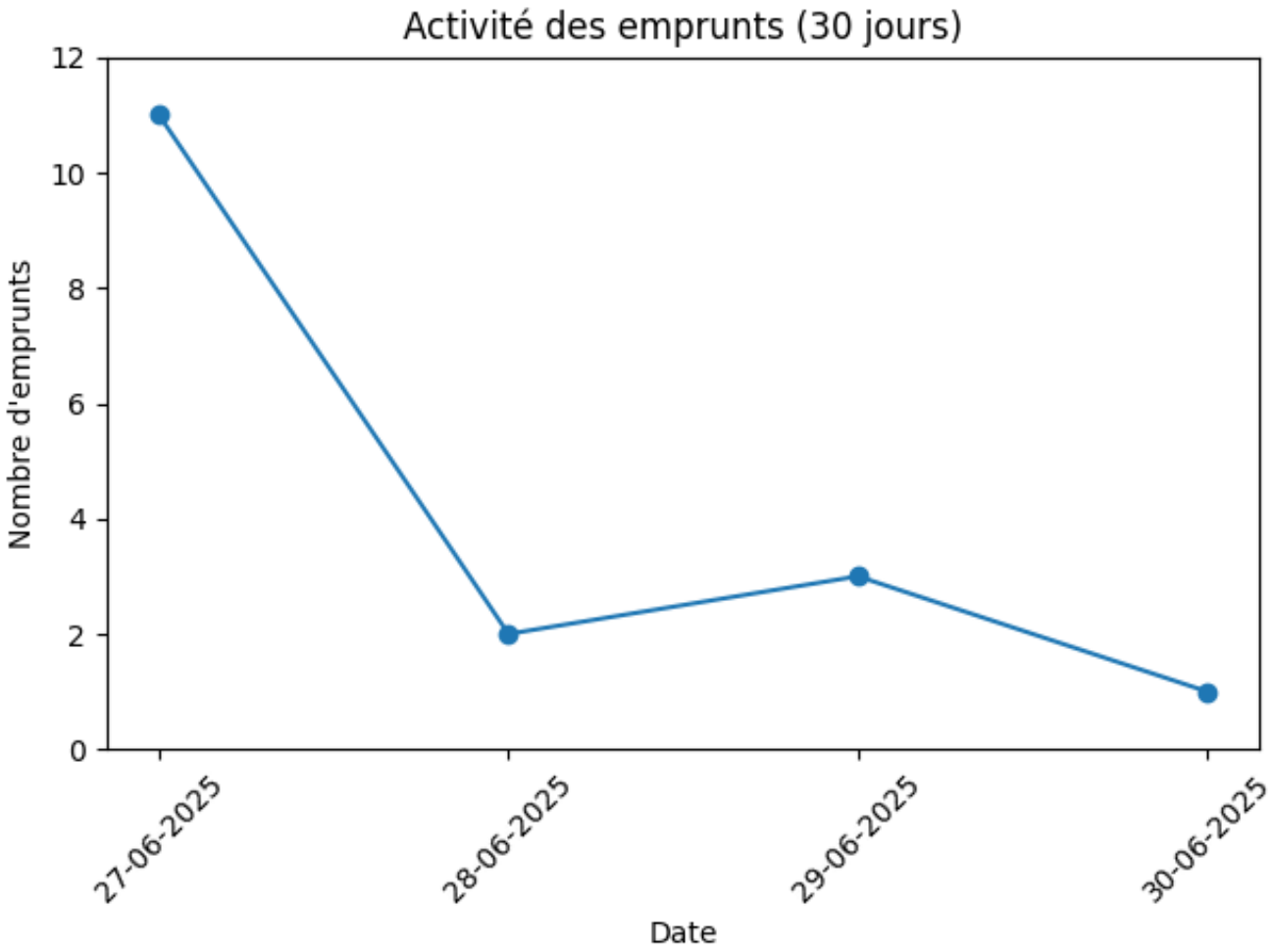


FIGURE 3.3 – Activité des emprunts sur les 30 derniers jours

4 Difficultés rencontrées et solutions

— **Gestion des erreurs spécifiques** : Implémentation d'exceptions personnalisées (`LivreIndisponibleError`, `QuotaEmpruntDepasseError`, `LivreNonEmprunteParCeMembreError`, etc.) pour mieux contrôler le flux d'exécution et fournir des messages clairs. Par exemple :

  — un membre ne peut retourner que les livres qu'il a empruntés ;
  — un livre emprunté ne peut pas être supprimé ;
  — un livre déjà emprunté ne peut pas être emprunté à nouveau ;
  — un membre ne peut pas emprunter plus de 3 livres ;
  — vérification que les livres et membres existent ;

— **Persistance des données** : Les données sont sauvegardées localement dans des fichiers .txt et .csv, permettant une reprise automatique du système après fermeture. Des méthodes spécifiques (`object-to-txt`, `txt-to-object`) assurent la transformation correcte des objets en chaînes de caractères et inversement.

— **Chargement des données sauvegardées** : Le problème d'encodage a été géré en utilisant le format UTF-8, permettant ainsi de charger correctement les données dès le lancement du programme, même avec des caractères spéciaux comme "é".

— **Validation des données** :Ajout de contrôles rigoureux dans les constructeurs des classes `Livre` et `Membre` pour éviter les entrées invalides, telles qu'un ISBN vide, un nom de membre vide, une année non valide, etc. Ces vérifications déclenchent des exceptions de type `ValueError` afin de garantir l'intégrité des données dès leur création.

— **Affichage clair des données** : Mise en forme lisible dans la console grâce à des messages explicites et l'usage d'icônes. Les statistiques sont visualisées avec `matplotlib` :

  — emploi de `MaxNLocator` pour afficher uniquement des entiers sur l'axe des ordonnées ;
  — affichage des dates au bon format dans les courbes d'activité ;
  — ouverture des trois visualisations (`courbe activité, top auteurs, top genres`) dans trois fenêtres distinctes en simultané, grâce au backend `TkAgg` ;
  — affichage des auteurs ayant zéro emprunt dans les graphiques pour ne pas les exclure de l'analyse.

— **Gestion des cas sans données à afficher** : Lorsque les fichiers `livres.txt` ou `historique.csv` sont vides, ou que les conditions (ex. : aucune activité depuis 30 jours) ne permettent pas de générer une statistique, le programme affiche un message explicite dans la console à la place d'un graphique vide. Cela évite les erreurs et rend l'interface plus conviviale.

— **Robustesse face aux fichiers mal formés** : Lors du chargement de `livres.txt`, `membres.txt`, et `historique.csv`, le système ignore les lignes incomplètes ou mal formées, assurant que le programme reste fonctionnel même si les fichiers ont été modifiés manuellement.