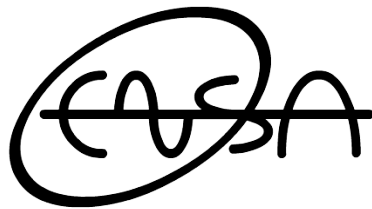


Gestion de Bibliothèque

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	2
2	Explications des algorithmes clés	3
3	Captures d'écran des visualisations	5
4	Difficultés rencontrées et solutions	7

1 Diagramme de classes UML

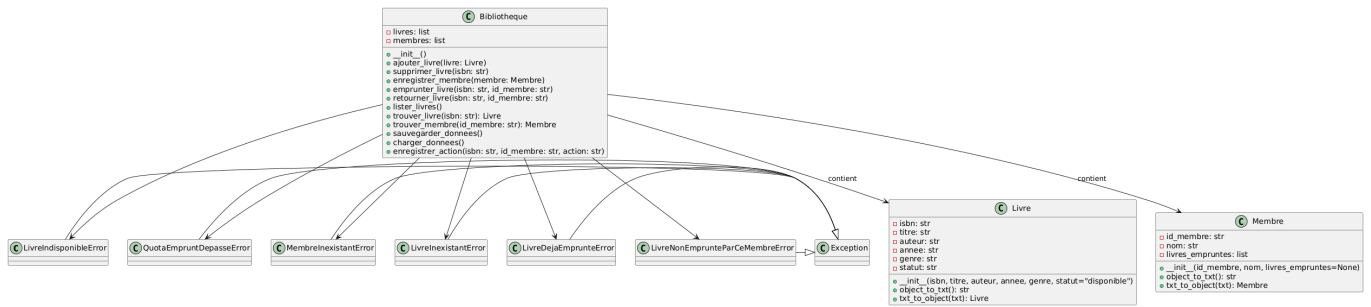


FIGURE 1.1 – Diagramme de classes UML du projet de gestion de bibliothèque

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

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

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

Suppression d'un livre

```
1 livre = self.trouver_livre(isbn)
2     if livre.statut == "emprunte" :
3         raise LivreDejaEmprunteError()
4     nouvelle_liste = []
5     for l in self.livres:
6         if l.isbn != isbn:
7             nouvelle_liste.append(l)
8     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

```
1 if livre.statut != "disponible":
2     raise LivreIndisponibleError()
3 if len(membre.livres_empruntes) >= 3:
4     raise QuotaEmpruntDepasseError()
5 livre.statut = "emprunte"
6 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

```
1 if livre.statut == "disponible":
2     raise LivreDejaDisponibleError()
3 if isbn in membre.livres_empruntes:
4     membre.livres_empruntes.remove(isbn)
5 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

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

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

Charger les données

```

1 if os.path.exists("data/livres.txt"):
2     with open("data/livres.txt", encoding="UTF-8") as f:
3         livres = []
4         for l in f:
5             if l.strip():
6                 livre = Livre.txt_to_object(l)
7                 livres.append(livre)
8         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

```

1 dates = []
2 with open("data/historique.csv", encoding="UTF-8") as f:
3     reader = csv.reader(f, delimiter=";")
4     for row in reader:
5         if len(row) < 4:
6             continue
7         date_str, _, _, action = row
8         if action.strip().lower() == "emprunt":
9             date = datetime.strptime(date_str.strip(), "%d-%m-%Y")
10            if datetime.now() - date <= timedelta(days=30):
11                dates.append(date.date())
12 if not dates:
13     print("Aucune activite d'emprunt dans les 30 derniers jours.")
14     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

```

1 isbn_auteur = {livre.isbn: livre.auteur for livre in livres}
2 emprunts_par_auteur = Counter({livre.auteur: 0 for livre in livres})
3 with open(fichier_historique, encoding='UTF-8') as f:
4     reader = csv.reader(f, delimiter=',')
5     for ligne in reader:
6         if len(ligne) != 4:
7             continue
8         date, isbn, id_membre, action = ligne
9         if action.strip().lower() == "emprunt":
10            auteur = isbn_auteur.get(isbn)
11            if auteur:
12                emprunts_par_auteur[auteur] += 1
13 top_10 = emprunts_par_auteur.most_common(10)
14 noms = [auteur for auteur, _ in top_10]
15 freqs = [count for _, count in top_10]
16
17 if not noms:
18     print("Top 10 des auteurs : Aucun auteur trouv .")
19     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

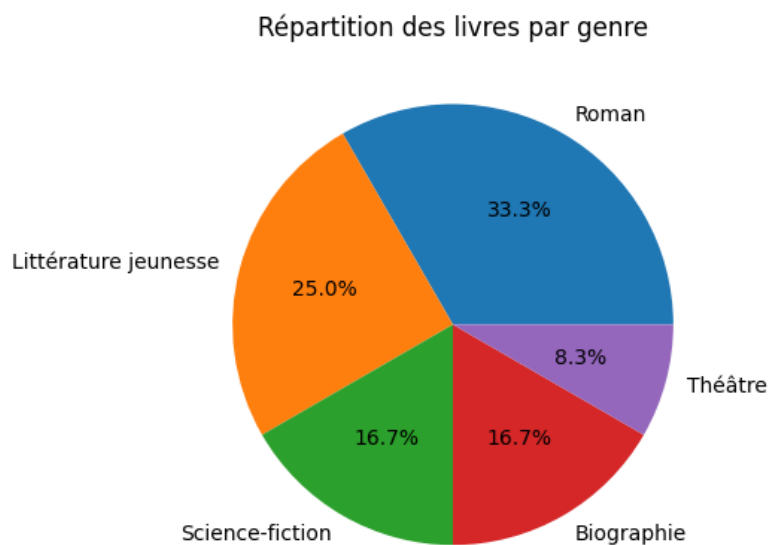


FIGURE 3.1 – Répartition des livres par genre

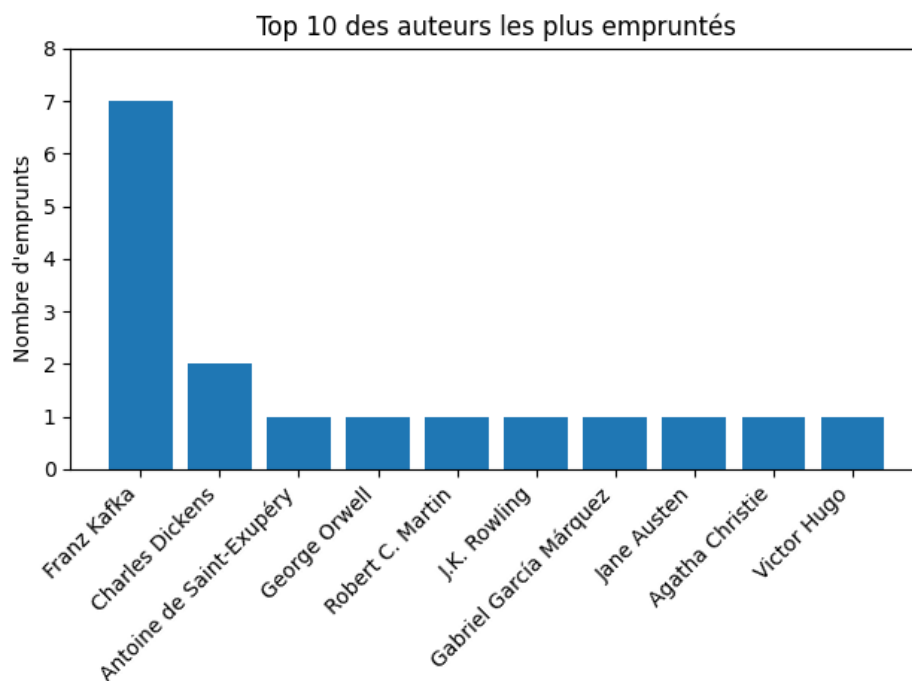


FIGURE 3.2 – Top 10 des auteurs les plus populaires

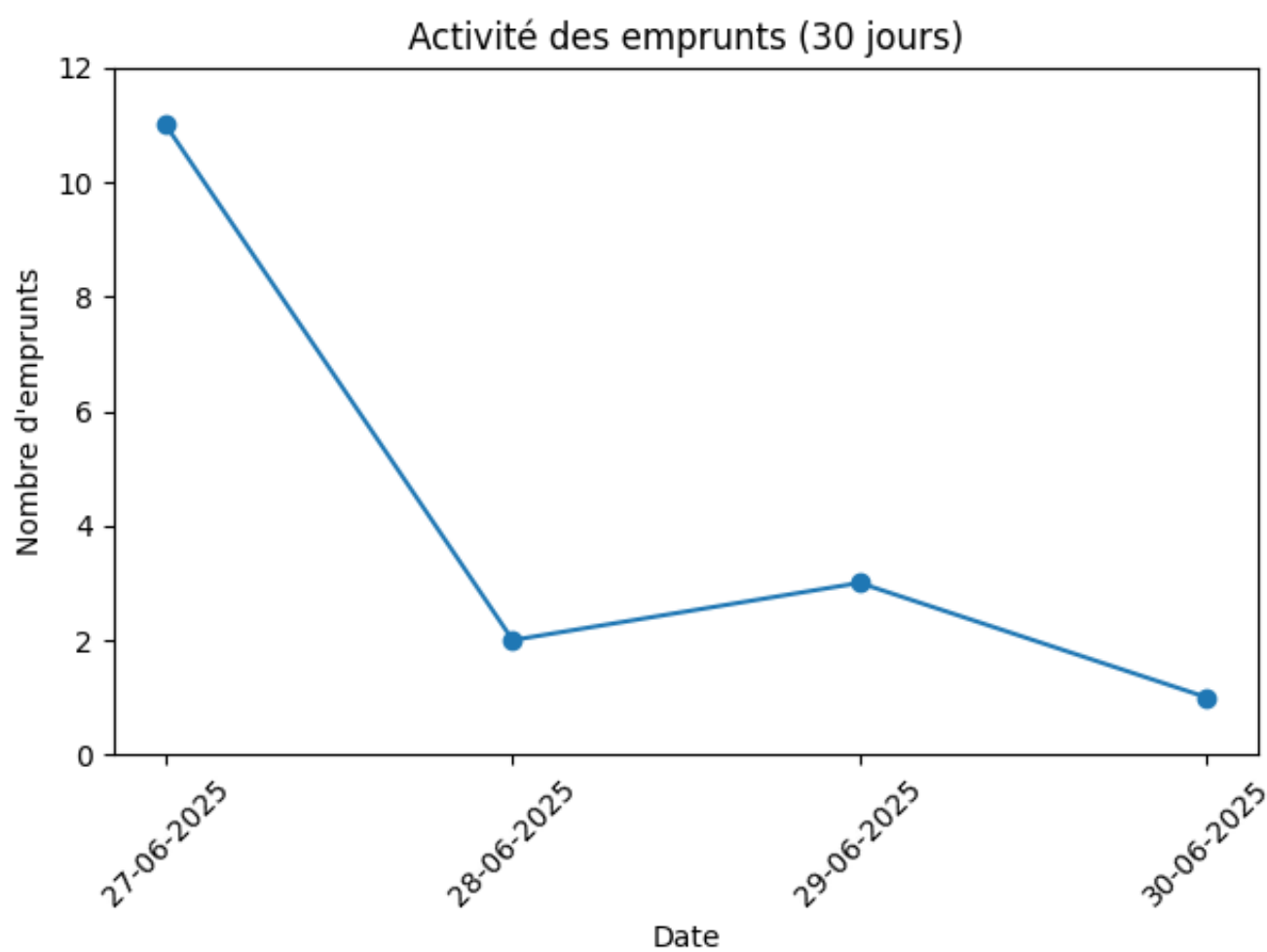


FIGURE 3.3 – Activité des emprunts sur les 30 derniers jours

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** : Choix de fichiers texte (.txt, .csv) pour enregistrer les données, ce qui a nécessité la création de méthodes de conversion comme "object-to-txt" et "txt-to-object" pour transformer les objets en texte et les reconstituer.
- **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 soignée dans la console, utilisation de `matplotlib` pour les statistiques visuelles, emploi de `MaxNLocator` pour afficher uniquement des nombres entiers sur l'axe des ordonnées et affichage des dates des activités au bon format pour générer des graphiques de bonne qualité, et ouverture des trois graphiques dans trois fenêtres distinctes en même temps grâce au backend `TkAgg`.