INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

Projet MIPP

La BIBLIOTHEQUE D'ALEXANDRIE

Doubli Hoda
Ait Taleb Assia
GM3

Encadré par : Mr Bourgais Mathieu

Juin 2024

Table des matières

Introduction

À Alexandrie en Égypte au 3e siècle avant J.-C., une bibliothèque célèbre dans le monde entier rassemble près de 500 000 rouleaux d'ouvrages philosophiques, poétiques, rhétoriques, historiques ou scientifiques. C'est la plus grande bibliothèque du monde antique. Ptolémée Ier, roi d'Égypte d'origine grecque et ancien général d'Alexandre le grand, ordonne que l'on recopie tous les livres se trouvant sur les navires faisant escale à Alexandrie. Il adresse aussi une lettre à tous les souverains et gouvernants de la terre, leur demandant de lui envoyer toutes les œuvres en leur possession, tous genres confondus. Il commande à la communauté de savants et des lettrés qui l'entourent la traduction de tous les livres non grecs. La bibliothèque devient un haut lieu de culture encyclopédique, où les informations recueillies, comparées, commentées, classées, publiées, peuvent être indéfiniment retraitées et enrichies. La bibliothèque disparaît pour des raisons mal connues, au Ier siècle avant J.-C. Elle renaît deux millénaires plus tard, reconstruite entre 1 995 et 2 002 à l'initiative de l'Unesco et de l'Égypte.

Au temps de la bibliothèque d'Alexandrie, la gestion d'une telle institution était très difficile, car chaque livre était considéré comme très précieux et devait être préservé avec soin. Cependant, à l'aube du XXIe siècle, nous avons le privilège de bénéficier des avancées technologiques qui nous permettent de réinventer et de simplifier la gestion bibliothécaire. Ainsi, dans cet esprit, nous nous lançons dans la conception d'une bibliothèque moderne, inspirée par l'héritage de l'Antiquité, mais adaptée aux exigences de notre époque.

Pour cela, nous souhaitons réaliser un logiciel pour aider à la gestion d'une bibliothèque municipale qui offre la possibilité à ses utilisateurs d'emprunter pour 1 semaine, renouvelable 2 fois, des livres, des films, des disques musicaux et même des jeux vidéos. Le logiciel permettra d'indiquer quels sont les utilisateurs de la bibliothèque et qu'ont-ils empruntés. Dans le cas de notre bibliothèque, chaque utilisateur peut emprunter 3 objets en même temps sauf les enfants qui ne peuvent en emprunter que 2 en même temps. Nous préciserons aussi quels sont les stocks de la bibliothèque et par qui ils sont empruntés.

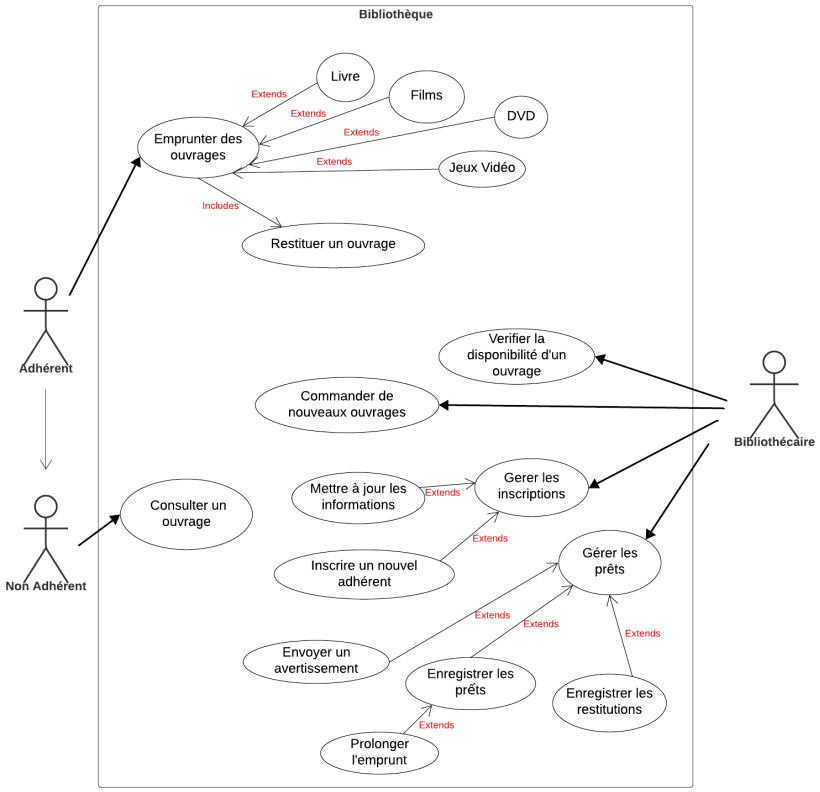
Dans un premier temps, nous analyserons et modéliserons le problème selon les principes de la programmation orientée objet, puis nous implémenterons le logiciel en Java.

1 Programmation orientée objet

Dans cette première partie, nous avons décidé d'utiliser trois diagrammes UML qui nous semblaient pertinents pour représenter notre système de gestion d'une bibliothèque. Nous avons choisi le diagramme de cas d'utilisation, celui de classe et enfin le diagramme de séquence.

1.1 Diagramme de cas d'utilisation : une première approche

Les diagrammes de cas d'utilisation identifient les utilisateurs du système appelés acteurs et leurs interactions avec le système. **Le système dans notre cas est un logiciel permettant la gestion d'une bibliothèque.** Ces diagrammes permettent de classer les acteurs, de structurer leurs besoins et les objectifs du système. Un acteur est l'idéalisation d'un rôle joué par une personne externe, un processus ou une chose qui interagit avec un système. Il se représente par un petit bonhomme avec son nom c'est à dire son rôle inscrit dessous. Le cas d'utilisation représente une fonctionnalité du système (visible de l'extérieur du système).Un cas d'utilisation se représente par une ellipse contenant le nom du cas d'utilisation. Les cas d'utilisation peuvent être connectés entre eux, ce qui permet d'ajouter des cas d'utilisation pour gérer des situations particulières ou des extensions d'un cas d'utilisation existant.



Dans la suite , nous considérerons qu'un ouvrage est soit un livre, un film, des disques musicaux ou encore un jeu vidéo.

Les différents acteurs de notre système et leur rôle

1. **Non adhérent** : Personne n'ayant pas encore adhérée à un abonnement à la bibliothèque. La seule intéraction possible alors avec le système est de pouvoir consulter un ouvrage.

2. **Adhérent** : Personne ayant adhérée à la bibliothèque. Tout comme le non adhérant, cet acteur peut également consulter des ouvrages mais son adhésion lui permet aussi de pouvoir emprunter et restituer des ouvrages.

3. **Bibliothécaire** : Personne chargée de gérer les opérations quotidiennes de la bibliothèque pour le bon fonctionnement de celle-ci. Son rôle comprend la gestion des inscriptions, des prêts, l'organisation de la bibliothèque comme commander des ouvrages et d'autres fonctionnalités que nous détaillerons dans les cas d'utilisation.

Les différents cas d'utilisations

1. **Consulter un ouvrage** : Permet aux adhérents comme au non adhérents de consulter des ouvrages. Cela peut être lire un livre, aller dans l'espace jeu et jouer à un jeu vidéo etc
2. **Emprunter des ouvrages** : Permet à un adhérent d'emprunter un ou plusieurs ouvrages disponibles dans la bibliothèque.
3. **Restituer un ouvrage** : Permet à un adhérent de retourner un ouvrage emprunté à la bibliothèque.

4. **Vérifier la disponibilité d'un ouvrage** : Permet au bibliothécaire de vérifier la disponibilité d'un ouvrage au sein de la bibliothèque.

5. **Gérer les prêts** : Ce cas d'utilisation concerne la gestion des emprunts. Il généralise trois autres cas d'utilisations ceux d'enregistrer un emprunt, d'enregistrer une restitution ou encore d'envoyer un avertissement. Ces cas sont détaillés ci-dessous. Chacun de ces cas restent optionnels mais il faut au moins en exécuter un .

6. **Enregistrer les prêts** : Permet au bibliothécaire d'enregistrer les prêts. Ce cas implique d'effectuer certaines vérifications telles que vérifier si la personne est un adhérent, si l'emprunt est possible c'est à dire s'assurer notamment que l'emprunteur n'a pas déjà dépassé le nombre maximum d'ouvrages empruntables.

7. **Enregistrer les restitutions** : Permet au bibliothécaire d'enregistrer le retour d'un ouvrage emprunté par un adhérent.

8. **Envoyer un avertissement** :Permet au bibliothécaire d'envoyer un avertissement à un adhérent en cas de retard dans le retour d'un ouvrage emprunté.

9. **Prolonger un prêt** : Permet au bibliothécaire de prolonger la durée d'un emprunt un prêt sous la demande d'un adhérant et sous réserve que la prolongation puisse être effectuée (dans notre cas un prêt peut être renouvelé deux fois seulement).

10. **Gérer les inscriptions** : Permet au bibliothécaire d'inscrire de nouveaux adhérents. Il généralise deux cas d'utilisation celui de mettre à jour les informations d'un adhérent et celui d'inscrire un nouvel adhérent. Ces cas seront détaillés ci-dessous.
11. **Inscrire un nouvel adhérent** : Permet au bibliothécaire d'inscrire des nouveaux adhérants à la bibliothèque. Cela implique d'enregistrer les informations personnelles de l'adhérent telles que son nom, prénom, son adresse mail et son âge et de lui fournir un numéro adhérent.

Parmi, les cas d'utilisation mentionnés plus haut certains, sont liés entre eux c'est ce que nous appelons **les relations entre les cas d'utilisations**. Dans notre cas, nous pouvons considérer différents types de relation :

- **La relation d'inclusion** est lorsqu'un scénario est déclenché au cours de l'exécution d'un autre. Par exemple si nous empruntons un ouvrage alors forcément, nous serons ammenés à le rendre d'où le cas d'utilisation emprunter un ouvrage engendre celui de restituer un ouvrage. Ceci est illustré par une flèche *«includes»*.

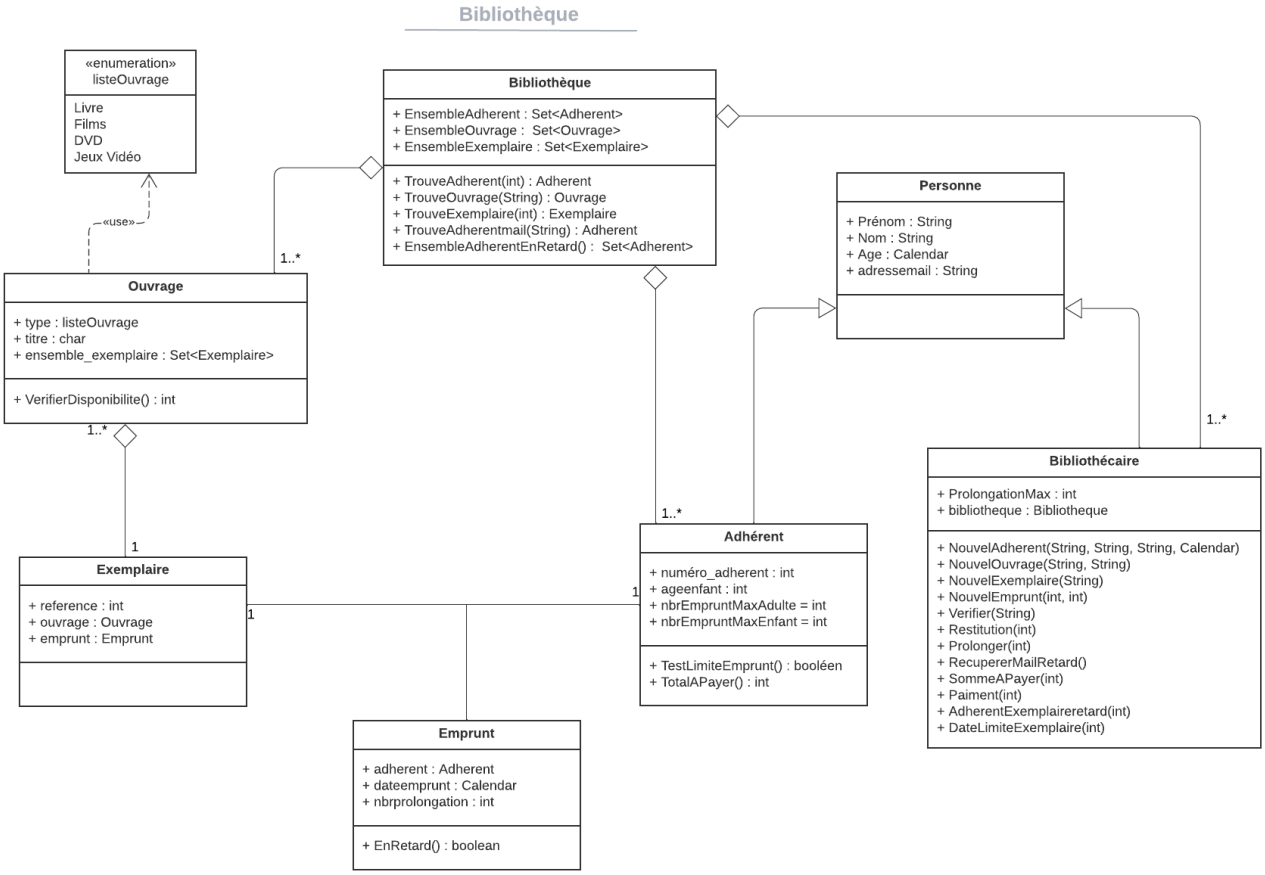
- **La relation d'extension** est lorsqu'**un scénario peut être déclenché** au cours de l'exécution d'un autre, il reste optionnel pour le scénario principal. Par exemple lorsque le bibliothécaire gère les inscriptions, il peut être amené à mettre à jour les informations d'un adhérent tout comme il peut inscrire un nouvel adhérent. Chacun de ces cas est optionnel mais il faut au moins en exécuter un.

Remarque 1 : Ce diagramme de cas d'utilisation est une première approche de notre modélisation de notre système de gestion de notre bibliothèque. Plusieurs cas d'utilisations ont été mis en place mais nous verrons qu'au fur et à mesure de des diagrammes nous affinerons notre modèle et nous allons mettre en oeuvre les cas d'utilisations qui noussemblaient les plus importants et qui répondaient au mieux au sujet !

Remarque 2 : Il est important de noter qu'ici nous avons représenté deux points de vue celui du bibliothécaire et celui de l'adhérent. Pour la suite de notre modélisation, nous nous placerons du point de vu du **du point de vue du bibliothécaire** .

1.2 Diagramme de classe vu par le bibliothécaire

Le diagramme de classe permet de décrire la structure de notre système. Le diagramme de classe n'évolue pas au cours de l'exécution du système. Ce diagramme permet de déterminer les classes dont nous avons besoin pour la conception du système , leurs fonctionnalités et leurs relations avec les autres éléments du système , à savoir les autres classes, les opérations, les attributs et les objets. Un objet est une entité concrête ou abstraite , c'est quelque dont nous avons besoin de manipuler dans le système. Les attributs sont des caractèristiques partagées par tous les objets du système. A chaque attribut est associée une valeur avec des types simples, primitifs ou énumérés. Les associations sont des liens entre les classes représentent les relations entre les différentes classes d'un système. Il existe plusieurs types d'associations que nous exploiterons dans notre système de la bibliothèque. En lien avec la notion d'association , il y a ce que nous appelons la multiplicité qui contraint le nombre **d'objets** lié par l'association.



Les différentes classes et les méthodes associées de notre système

1. **Bibliothèque** : Elle est caractérisée par un nom et représente la structure dans laquelle se trouve les ouvrages et où les intéractions entre le bibliothécaire et les adhérents sont effectuées. Au sein de notre bibliothèque de nombreuses opérations dont leur nom explicite leur fonctionnalité, peuvent être effectuées :

   - **TrouverAdherent(int)**
   - **TrouverOuvrage(String)**
   - **TrouverExemplaire(int)**
   - **NouvelEmprunt(int,int)**
   - **TrouverAdherentmail(String)**
   - **EnsembleAdherentEnRetard()**

2. **Ouvrage** : Cette classe représente les ouvrages de la bibliothèque possédant un titre , un type (livre, film, disques musicaux ou encore jeu vidéo) et un ensemble d'exemplaire. Une méthode associée à cette classe est :

   - **VérifierDisponiblité()**

3. **Personne :** Cette classe est la classe mère pour tous les types de personnes dans la bibliothèque. Une personne est caractérisée par son nom,son prénom, son âge et son adresse mail. Ses classes filles auront des particularités supplémentaires que ce soit au niveau des attributs ou des méthodes.

4. **Exemplaire :** Nous avons considéré qu'un adhérent n'emprunte pas un ouvrage directement mais plutôt un exemplaire ce qui permettrait à un autre adhérent d'emprunter ce même ouvrage. Un exemplaire est caractérisé par une référence (unique).

5. **Emprunt :** Pour représenter les emprunts d'un adhérant, nous avons considéré qu'un adhérant n'emprunte pas un ouvrage directement mais plutôt un exemplaire d'un ouvrage. Chaque exemplaire est représenté par une référence , par une date d'emprunt et le nombre de prolongations. Elle est associée aux classes adhérant et ouvrage. C'est ce que nous appelons une **classe-association.** pour laquelle nous lui avons associée la méthode **EnRetard()** permettant de savoir quels sont les ouvrages en retard d'un adhérent.

6. **Bibliothécaire** : Cette classe représente le bibliothécaire travaillant dans la bibliothèque. Dans la suite, nous avons supposé qu'un bibliothécaire n'est ni un adhérant ni un non adhérent et qu'il est seulement chargé des opérations quotidiennes de la bibliothèque (cf acteur cas d'utilisation). Ces opérations nous ont amenés à considérer les méthodes suivantes dont leur nom explicite leur fonctionnalité :

   - **NouvelAdherent(String,String,String,Clendar**
   - **NouvelOuvrage(String,String)**
   - **NouvelExemplaire(String)**
   - **NouvelEmprunt(int,int)**
   - **Vérifier(String)(vérifie l'existence d'un ouvrage au sein de la bibliothèque**
   - **Restitution (int)**
   - **Prolonger(int)**
   - **RecupererMailEnRetard ()**
   - **SommeAPayer(int)**
   - **Paiement(int)**
   - **AdherentExemplaireretard(int)**
   - **DateLimiteExemplaire(int)**

7. **Adhérant** : Cette classe représente les adhérents de la bibliothèque (enfant et adulte ce choix est expliqué plus bas). En plus des attributs hérités de la classe personne, un adhérent possède aussi un numéro adhérent. Les méthodes qui lui sont associées sont :

   - **TestLimiteEmprunt**
   - **TotalAPayer()**

Les différentes associations mises en place reliant les différentes classes

Dans ce diagramme de classe , nous avons décidé de passer par une énumération concernant les différents ouvrages car cela n'apportait pas grand chose de mettre les différents ouvrages en classe fille de la classe mère car ils possédaient tous les mêmes attributs. (Cependant si à l'avenir chaque ouvrage possédait des méthodes propres à l'ouvrage, il serait alors intéressant de mettre des classes filles.)

La notion **d'héritage** nous sera utile pour l'explication des classes.L'héritage permet de définir une nouvelle classe appelée classe fille basée sur une classe existante appelée classe mère . L'héritage permet à la classe fille d'hériter des attributs et des méthodes de la classe mère, tout en lui permettant de définir ses propres attributs et méthodes spécifiques.

1. **Bibliothèque**

   Relation avec Adherent :

   La classe Bibliothèque est associée à plusieurs Adherents. Cette association est une agrégation car les adhérents, en tant qu'humains, existent indépendamment de la bibliothèque. Si la bibliothèque venait à disparaître, les adhérents continueraient d'exister, ce qui justifie l'utilisation d'une agrégation (symbolisée par un losange vide).

   Relation avec Bibliothécaire :

   De même La classe Bibliothèque est associée à un Bibliothécaire qui existe indépendamment de la bibliothèque.

   Relation avec Ouvrage :

   La classe Bibliothèque est également associée à plusieurs Ouvrage. Cette association est une agrégation car les ouvrages existent indépendamment de la bibliothèque. Ils continueraient d'exister en tant qu'entités distinctes.

2. **Bibliothécaire**

   Héritage de Personne :

   La classe Bibliothécaire hérite également de la classe Personne, partageant les attributs communs tels que le prénom, le nom, l'âge, et l'adresse mail.

3. **Adherent**

   Héritage de Personne :

   La classe Adherent hérite de la classe Personne, ce qui signifie que chaque adhérent possède les attributs et méthodes d'une personne (comme le prénom, le nom, l'âge, et l'adresse mail).

   Relation avec Exemplaire (via Emprunt) :

   Un Adherent peut emprunter plusieurs Exemplaire. Un Exemplaire est au plus relié à un adhérent (0 s'il n'est pas emprunté), indiquant qu'un exemplaire peut être libre ou associé à un seul adhérent via un emprunt. Nous avons décidé de mettre l'emprunt en tant que classe d'association, car elle n'existe que par l'interaction entre la classe Adhérent et la classe Exemplaire, (elle possède également ses propres attributs).

4. **Ouvrage**

   Relation avec Exemplaire :

   Un Ouvrage peut être disponible en plusieurs Exemplaire. Il doit y avoir au moins un exemplaire pour qu'un ouvrage soit considéré comme disponible dans la bibliothèque. Un objet Exemplaire correspond à un ouvrage spécifique et est donc relié à un seul ouvrage. Cette relation est une composition (symbolisée par un losange rempli) car les exemplaires sont liés à un ouvrage et ne peuvent exister sans lui.

1.3 Diagramme de séquence vu par le bibliothécaire

Un diagramme de séquence est essentiellement une façon de montrer comment les différentes parties d'un système logiciel communiquent entre elles dans le **temps**. Il faut savoir que le système est représenté par les éléments du diagramme de classe et les différents scénarios sont issus des diagrammes de cas d'utilisation.

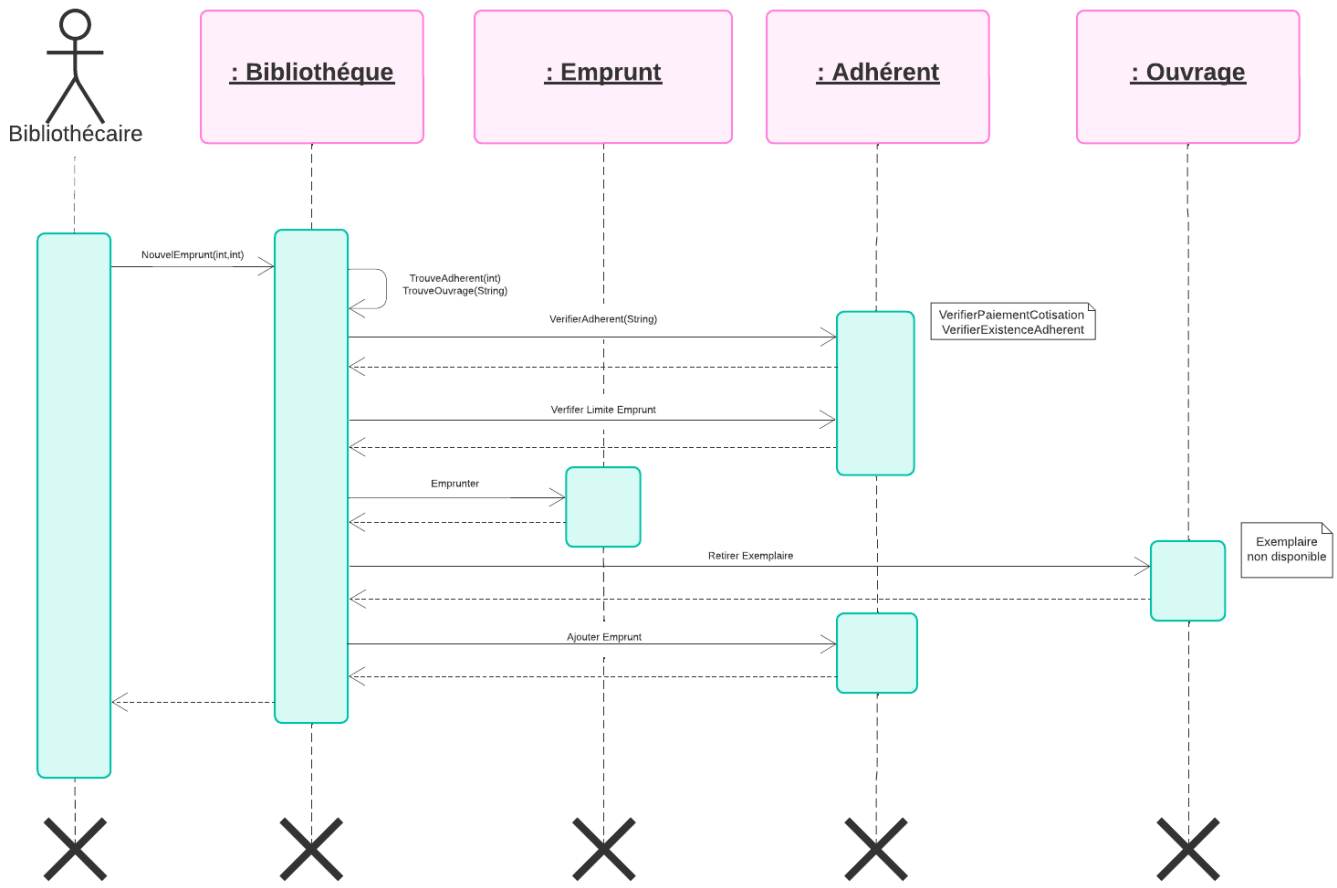
Voici ce que nous allons retrouver dans ce diagramme :

1. **Acteurs** : Ceux qui interagissent avec le système, comme les utilisateurs ou d'autres systèmes.
2. **Objets** : Les parties du système qui interagissent, comme des classes ou des instances de ces classes.
3. **Lignes de vie** : Elles représentent le temps pendant lequel les objets existent et interagissent.
4. **Messages** : Les actions ou les informations échangées entre les objets, montrées par des flèches.

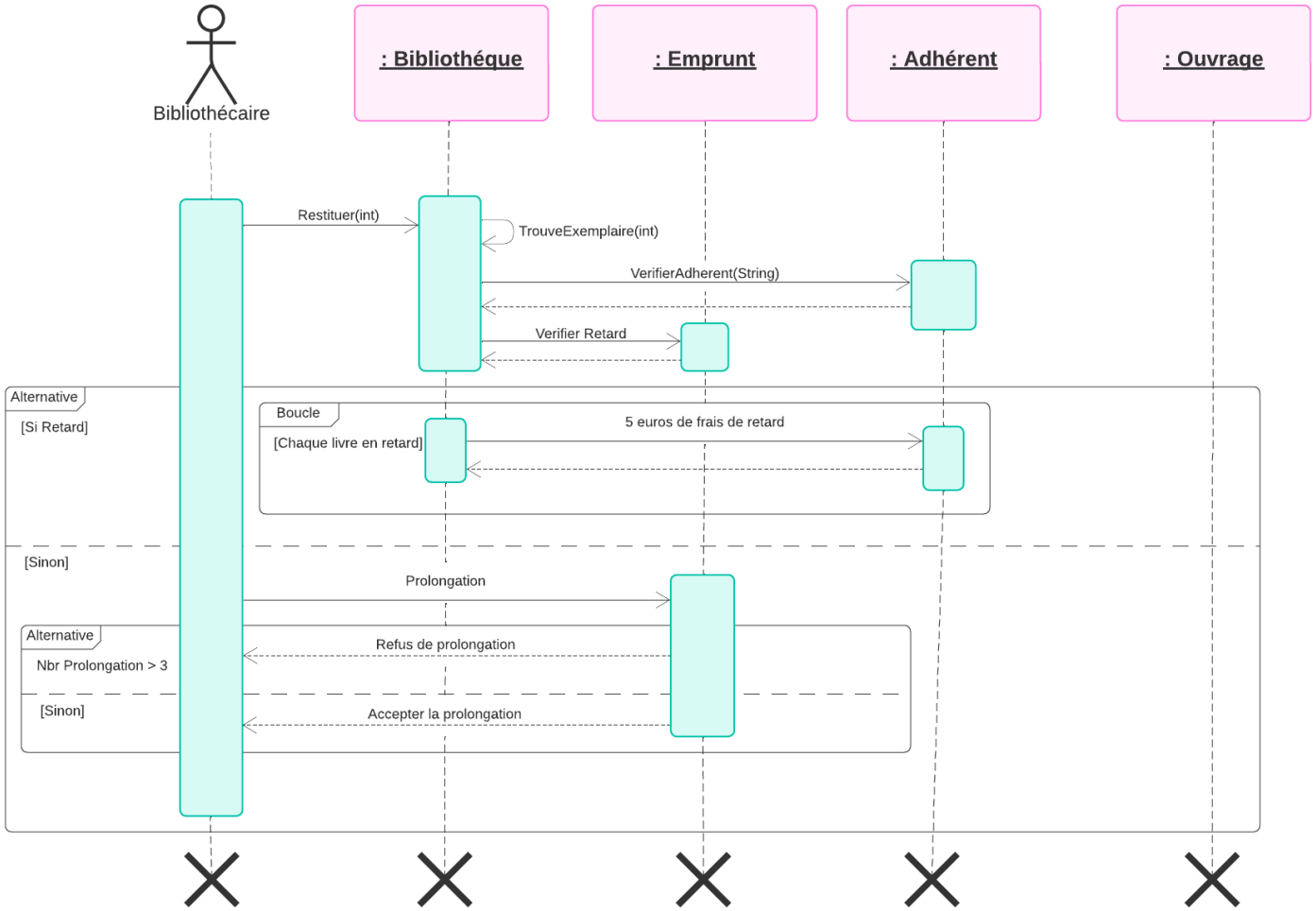
Ce diagramme nous permet donc de voir exactement comment le programme fonctionne dans des situations réelles, ce qui est très utile pour détecter les erreurs et améliorer la conception, par exemple, le fait d'avoir oublié de prendre en considération un objet.

Concernant le diagramme de séquence, nous avons décidé d'illustrer les actions qui nous semblaient les plus intéressantes à représenter à savoir l'emprunt d'un ouvrage et la restitution d'un ouvrage.

Cas où un adhèrent demande un emprunt



Cas où un adhèrent demande une restitution



A travers ces deux diagrammes de séquence, nous voyons de manière plus précise comment se déroule un emprunt et une restitution au sein de notre bibliothèque.

2 Implémentation en java

Afin d'implémenter la gestion de la bibliothèque, nous nous sommes reposées sur les diagrammes UML établis précèdemment. Nous noterons que lors de l'implémentations nous avons du adapter nos diagrammes car nous avons rencontré des problèmes qui n'étaient pas encore visibles lors de la conception des diagrammes UML.

2.1 Des notions de java utiles

2.1.1 Les collections

En Java, le terme **"collection"** désigne des structures de données permettant de manipuler des ensembles d'objets homogènes, dont la taille peut varier dynamiquement. Les collections sont basées sur des interfaces définissant leur comportement. En Java, les trois types principaux de collections sont les ensembles (Set), les listes (List) et les cartes (Map).

Dans notre projet, nous avons utilisé les ensembles (Set) pour représenter différentes relations, telles que la relation entre la bibliothèque et ses adhérents, entre la bibliothèque et ses ouvrages, ainsi qu'entre la bibliothèque et ses exemplaires. Les ensembles (Set) sont particulièrement adaptés pour garantir l'unicité des éléments, car ils ne permettent pas la présence de doublons au sein de l'ensemble .Cette approche garantit ainsi l'unicité des adhérents et des ouvrages au sein de la bibliothèque, ainsi que l'unicité des exemplaires.

2.1.2 La gestion des dates

Nous avons intégré la notion de date en Java pour gérer certains aspects de notre programme tels que la date d'emprunt d'un ouvrage ou encore la prolongation d'un prêt. Pour ce faire, nous avons consulté des tutoriels en ligne. Cela a nécessité l'importation de plusieurs bibliothèques. La bibliothèque **java.util.GregorianCalendar** permet l'implémentation du calendrier Grégorien, le calendrier le plus utilisé dans le monde. Cela garantit que notre logiciel de gestion de la bibliothèque gère correctement les subtilités telles que les années bissextiles, les changements de mois et les ajustements d'heure. La bibliothèque **java.util.Calendar** permet la manipulation des dates telle que l'addition la soustraction ou encore la comparaison de dates. Par exemple afin d'avoir la date maximale à laquelle un adhérent doit rendre un ouvrage (dans la méthode **("DatelimiteEmprunt"**), nous avons eu besoin d'additioner des dates. Pour cela, nous avons utilisé la méthode date.add ()inclue dans cette dernière. La bibliothèque **java.util.SimpleformatDate** nous a permis d'avoir l'écriture de la date sous la forme "JJ/MM/AAAA".

2.1.3 Les variables statiques

En Java, nous pouvons avoir besoin qu'un élément attribut ou méthode ne soit pas lié à une instance particulière d'une classe. Dans ce cas, on le déclare comme **static.** Dans notre cas, nous avons décidé que notre bibliothèque autorisée un nombre limite d'emprunt selon que l'adherent soit un enfant ou un adulte. Nous avons également considéré qu'un adhérent était un enfant enfant s'il est agé de 12 ans (ou moins). Nous avons alors déclaré les constantes suivantes :

- **private static final int nbrempruntmaxadulte=3,**
- **private static final int nbrempruntmaxenfant=2,**
- **private static final int ageenfant=12,**

2.1.4 Les Exceptions

Les exceptions représentent le mécanisme de gestion des erreurs intégré au langage Java. Il se compose d'objets représentant les erreurs et d'un ensemble de trois mots clés qui permettent de détecter et de traiter ces erreurs **(try, catch et finally)** mais aussi de les lever ou les propager **(throw et throws)**.Lors de la détection d'une erreur, un objet qui hérite de la classe Exception est créé (on dit qu'une exception est levée) et propagé à travers la pile d'exécution jusqu'à ce qu'il soit traité. Ces mécanismes permettent de renforcer la sécurité du code Java. Nous avons intégré plusieurs exceptions dans notre programme. Par exemple, lorsque l'utilisateur saisit une date dans un format incorrect **(version interactive)**, une exception **ParseException** est levée (une exception inclue dans la classe **Exception** de java). En plus des erreurs classiques, on peut aussi avoir envie de créer son propre jeu d'erreurs qui seront liées aux nouvelles classes et méthodes créées. Dans le cadre de notre programme de gestion de bibliothèque, nous avons mis en place l'exception personnalisée **Prolongationimpossible** qui est utilisée pour indiquer qu'une prolongation de prêt n'est pas possible et l'exception **Empruntimpossible** qui est utilisée pour indiquer qu'un emprunt de prêt n'est pas possible . Bien que nous aurions pu mettre des exceptions à de nombreux endroits dans le code pour gérer toutes sortes d'erreurs possibles, ce n'était pas notre objectif principal. Nous avons choisi de nous concentrer sur la création de ces deux exceptions qui nous semblaient particulièrement importantes pour notre application de gestion de bibliothèque.

2.1.5 Les Flux en java

En informatique, un flux (ou stream en anglais) est une abstraction qui représente une séquence d'éléments de données disponibles au fil du temps. En Java, les flux sont utilisés pour lire des données provenant de différentes sources (comme un fichier, ou encore une entrée utilisateur) et pour écrire des données vers différentes destinations. **Le Scanner** est une classe pratique en Java pour lire les entrées de l'utilisateur à partir de diverses sources, y compris les flux de caractères. Pour la version intéractive, nous avons utilisé le Scanner pour lire l'entrée de la console (System.in), qui est un flux d'entrée standard en Java.

2.2 Explications des décisions prises pour l'implémentation

Tout d'abord, en ce qui concerne la gestion des adhérents, nous avons pris la décision de ne considérer qu'une seule classe pour les adhérents, sans créer de classes filles pour adultes et enfants. Nous avons évité cette complexité supplémentaire dans le code car la principale distinction entre un adulte et un enfant réside dans le nombre maximal d'emprunts et la cotisation, qui est uniquement payée par les adultes. Pour différencier les adhérents adultes des enfants, nous avons utilisé une variable statique **ageenfant** que nous avons déclarée et qui nous a permis de gérer cette situation. Par exemple, pour gérer le nombre d'emprunts, nous avons utilisé des méthodes de comparaison de dates sur notre variable statique (au sein de la méthode) .

Ensuite, dans notre diagramme de classe, nous avions initialement une classe "non adhérent". Cependant, lors de l'implémentation, nous avons décidé de supprimer cette classe car le non adhérent n'intervenait pas dans notre programme puisque les méthodes implémentées ne concernent que des adhérents et lorsqu'une personne s'inscrit elle est directement considérée comme adhérente à la bibliothèque.

Les principaux points sur lesquels nous avons réfléchi étaient la manière de gérer l'attribution d'identifiants uniques pour les adhérents et la gestion des emprunts d'ouvrages. Dans notre conception, nous avons considéré que les adhérents n'empruntent pas directement un ouvrage, mais plutôt un exemplaire de celui-ci afin d'offrir la possibilité à plusieurs adhérents d'emprunter le même ouvrage.

Pour les identifiants des adhérents, nous avons attribué à chaque nouvel adhérent un numéro unique. Pour cela, nous avons profité du fait que nous ayons au sein de notre bibliothèque un ensemble **Set** d'adhérents et que nous pouvons connaître la taille de celui-ci grâce à la méthode **size().** Ainsi, chaque adhérent se voit attribuer un identifiant unique correspondant à la taille de l'ensemble (avant l'ajout de l'adhérent) , à laquelle nous avons ajouté 1.

Nous avons également décidé de numéroter nos exemplaires de manière unique sur l'ensemble de la bibliothèque et non pas sur l'ensemble des exemplaires de l'ouvrage correspondant. Cette approche simplifie la gestion des méthodes en se basant sur une référence unique pour chaque exemplaire, notamment lors de la restitution d'un ouvrage , il suffit juste d'avoir le numéro de l'exemplaire pour que la restitution soit faite dans le cas contraire, nous aurions également du avoir l'ouvrage.

En ce qui concerne la gestion des adhérents pour éviter les inscriptions multiples, nous avons implémenté la méthode **"TrouveAdherent()"** qui permet à partir de l'identifiant de ce dernier de savoir si l'adhérent se trouve dans la bibliothèque. Cette approche permet de se mettre directement à la place d'un bibliothécaire en face de son logiciel. En effet notre bibliothèque est composée d'un ensemble d'adhérent qui est un type que le bibliothécaire ne manipule pas rendant la méthode **contains()** "inutilisable". Donc en considérant son point de vue, il est plus simple de trouver un adhérent à partir de son identifiant d'où l'implémentation de la méthode.

2.3 Lien avec la matière Systeme et Réseau

Cette année, en cours de systèmes et réseaux, nous avons étudié la notion de fonction de hachage dans le cadre du chapitre sur la cryptographie pour la sécurité informatique. Une fonction de hachage est une fonction appliquée à une chaîne de caractères qui produit une autre chaîne de caractères binaire (plus courte), appelée empreinte (ou hash, ou encore digest). Cette notion est particulièrement utile pour la gestion des mots de passe. Dans notre implémentation **de la version interactive**, nous avons voulu mettre en place un système de mots de passe pour la connexion des bibliothécaires au logiciel. Nous nous sommes donc renseignés sur la manière d'appliquer ces concepts étudiés en systèmes et réseaux. La fonction de hachage que nous avons utilisée est SHA-256.

2.4 Résultats

2.4.1 Version non intéractive

Dans notre main, on a crée une bibliothèque et une bibliothécaire à l'aide des constructeurs prédéfinis dans chacune des classes. Le bibliothécaire instancié pourra ainsi tester toutes les méthodes de notre système. Nous avons joins ci-dessous les résultats obtenus.

```
Test que le nombre d'adherents est egal a 0, valeur de la variable: 0
Test que le nombre d'ouvrages est egal a 0, valeur de la variable: 0
Test que le nombre d'exemplaires est egal a 0, valeur de la variable: 0

Test d'un nouvel adherent
L'inscription a bien ete effectuee, son numero d'identifiant est : 1

Test d'un nouvel adherent avec la meme adresse email
L'adherent: Hoda est deja inscrit

Test d'un changement d'email
Etat de la variable adressemail :
nouveladresse@email.com

Test que la taille de notre ensemble d'adherent est egal a 1, valeur de la variable :1

Test d'un nouvel ouvrage
L'ouvrage Harry Potter a bien ete enregistre

Test d'un nouvel ouvrage avec le meme titre
L'ouvrage Harry Potter existe deja dans la bibliotheque

Test que la taille de notre ensemble d'ouvrage est egal a 1, valeur de la variable :1

Test d'un nouvel exemplaire
L'exemplaire Harry Potter a bien ete enregistre a la reference : 1

Ajout d'un second exemplaire
L'exemplaire Harry Potter a bien ete enregistre a la reference : 2

Test que la taille de notre ensemble d'exemplaire est egal a 2, valeur de la variable :2

Test de la disponiblite d'un ouvrage
L ouvrage Harry Potter est disponible en 2 exemplaire(s)

Test emprunt d'un exemplaire
L'enregistrement a ete realise avec succes

Test que la disponiblite de l'ouvrage a diminuer a 1
L ouvrage Harry Potter est disponible en 1 exemplaire(s)

Test de 3 prolongations (de quinze jours)
La prolongation a été réalisée avec succès
La prolongation a été réalisée avec succès
Erreur lors de la prolongation : Le nombre de prolongations est déjà atteint

Test de la recuperation de la date limite
La date limite est vendredi, juillet 19, 2024

Modification manuel de la date d'emprunt pour regarder les fonctions faisant intervenir un retard
Manipulation termine

Test de la recuperation des adresses emails des adherents qui ont des retards
Le mail des adherents en retard sont :
nouveladresse@email.com

Test de la recuperation des ouvrages qu'un adherent a en retard
Les ouvrages en retard sont
Harry Potter
```

```
Test de la restitution d'un exemplaire
La restitution a ete realisee avec succes

Test que la disponiblite de l'ouvrage est remonte a 2
L ouvrage Harry Potter est disponible en 2 exemplaire(s)

Modification manuel de la date du dernier paiement pour regarder les fonctions faisant intervenir un paiement
Manipulation termine

Test de la valeur du d'un adherent sachant 1 retard, et le dernier paiement un mois (egal 10)
Le montant total a payer est 10

Test du paiement
Paiment effectue

Test de la valeur du d'un adherent apres paiement (egal 0)
Le montant total a payer est 0

Test d'un nouvel adherent enfant
L'inscription a bien ete effectuee, son numero d'identifiant est : 2

Test d'un nouvel ouvrage
L'ouvrage Tchoupi a bien ete enregistre

Test d'un nouvel exemplaire
L'exemplaire Tchoupi a bien ete enregistre a la reference : 3

Ajout d'un second exemplaire
L'exemplaire Tchoupi a bien ete enregistre a la reference : 4

Ajout d'un troisième exemplaire
L'exemplaire Tchoupi a bien ete enregistre a la reference : 5

L'enregistrement a ete realise avec succes

L'enregistrement a ete realise avec succes

Limite des emprunts deja atteint, echec de l'enregistrement
```

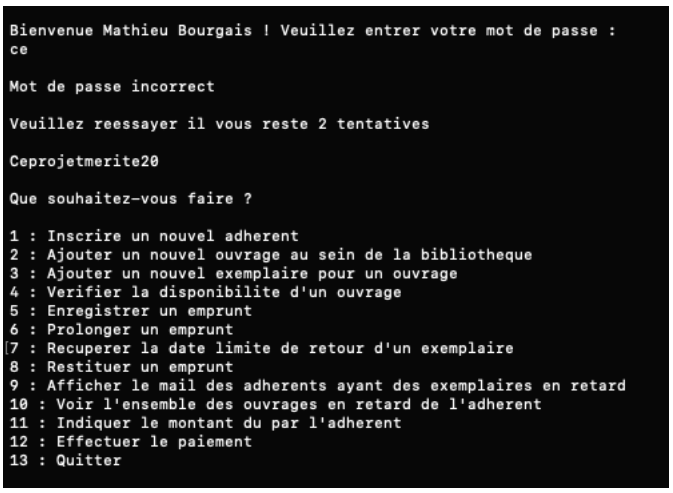
2.4.2 Version intéractive

Nous proposons une version interactive qui permet au bibliothécaire de choisir au fur et à mesure les opérations qu'il souhaite effectuer au sein de la bibliothèque. Pour cela, nous avons affiché un menu sur le terminal dans lequel le bibliothécaire aura une multitude de choix. Ce menu a été réalisé à l'aide de l'instruction **switch case** et implémente les mêmes méthodes que la version non intéractive. Ici nous testerons les quelques différences avec la version non intéractive à savoir la gestion d'un mot de passe et la gestion d'une erreur lorsque l'utilisateur entre au clavier une date.

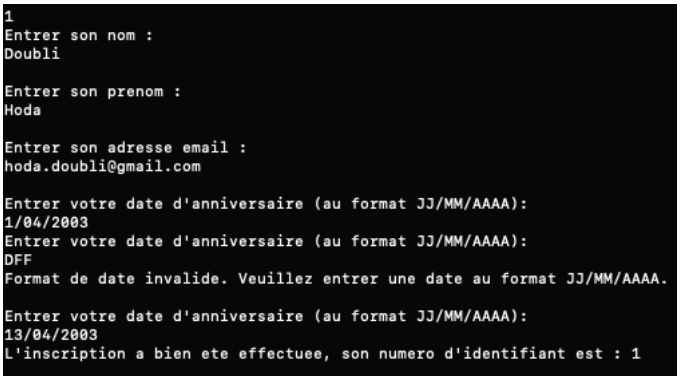
Lorsque nous exécutions le programme un message de bienvenue est affiché au bibliothécaire ainsi que la demande de son mot de passe **(Ceprojetmerite20)** pour qu'il puisse avoir accès au logiciel. Concernant le mot de passe pour des questions de sécurité, nous avons décidé que le bibliothécaire n'avait que trois tentatives dans le cas où il n'écrit pas le bon mot de passe trois fois de suite il est renvoyé vers le prompt.

```
Bienvenue Mathieu Bourgais ! Veuillez entrer votre mot de passe :
ce

Mot de passe incorrect

Veuillez reessayer il vous reste 2 tentatives

bonjour

Mot de passe incorrect

Veuillez reessayer il vous reste 1 tentatives

au revoir

Mot de passe incorrect

Veuillez reessayer il vous reste 0 tentatives

doublihoda@iMac-de-Doubli projet Mipp 3 %
```

Dans le cas où la saisie du mot de passe est correct le menu s'affiche



Voici un exemple de ce qui se passe lorsque le bibliothécaire choisi d'inscrire un nouvel adhérent :



Lorsque le bibliothécaire ente la date de naissance de l'adhérent sous un mauvais format un message d'erreur s'affiche et le bibliothécaire est ammené à resaisir la date de naissance de l'adhérent et ce jusqu'à que le format soit correct. Un message affirme l'inscription de l'adhérent et lui associe un numéro d'adherent.

De la même manière que la version non intéractives vous pouvez tester tous les choix d'opérations possibles que le bibliothécaire peux effectuer **(la version interactive implémente les mêmes méthodes que la version non interactive).**

2.4.3 Version avec interface graphique

Dans cette version avec interface graphique, nous avons utilisé la bibliothèque **JavaFX**. Pour cela nous avons installé le package correspondant via un site internet (Gluon).
Il a fallu aussi se renseigner sur comment utiliser cette bibliothèque et pour cela nous avons fait des recherches sur divers sites (dont le principale est noté dans les Sources) et vidéo YouTube.

Le contenue des classes faite avec la version interactive est identique à celle de la version avec interface graphique. Les seuls différence sont :

1. La façon dont on affiche des choses à l'ecran.
2. L'ajout d'un **fichier FXML**. En JavaFX, un fichier FXML est un fichier XML utilisé pour définir l'interface utilisateur de notre application de manière déclarative.
3. L'ajout d'un **contrôleur**. En JavaFX celui-ci joue un rôle crucial en agissant comme un intermédiaire entre la vue (définie en FXML) et la logique de l'application. Les contrôleurs gèrent les interactions utilisateur, comme les clics sur les boutons et les entrées de texte.
4. Et enfin un main définie d'une autre façon que l'on va voir par la suite.

Le **'main'** dans le contexte de JavaFX sert à lancer une application graphique. Il étend la **classe 'Application'** et utilise la méthode 'start(Stage primaryStage)' pour définir et afficher l'interface utilisateur. Cette méthode est automatiquement appelée par la méthode 'launch(args)', qui initialise l'application JavaFX. Contrairement à un 'main' standard sans interface graphique, qui exécute une série d'instructions séquentielles dans la console, ce 'main' configure **une scène ('Scene')** et **un stage ('Stage')** pour afficher une fenêtre graphique définie par **un fichier FXML ('interface.fxml')**. Il gère ainsi le cycle de vie d'une application graphique JavaFX, intégrant l'interface utilisateur et la logique de démarrage.

La commande qui permet de compiler les classes est exactement la même que précédemment, à savoir : javac Nom-Fichier.java. Cependant, les fichiers ayant importé la bibliothèque javafx doivent être compilés différemment. C'est le cas du fichier **Bibliothécaire.java , MediathequeController.java et de main.java.**
La commande est la suivante : **javac –module-path /Chemin/javafx-sdk-11.0.2/lib –add-modules javafx.fxml,javafx.controls NomFichier.java** . "Chemin" est le chemin menant au dossier javafx-sdk-11.0.2.

Et enfin on exécute le main de la même façon :

java –module-path Chemin/javafx-sdk-11.0.2/lib –add-modules javafx.fxml,javafx.controls main

Voici ce que l'on obtient à l'écran :

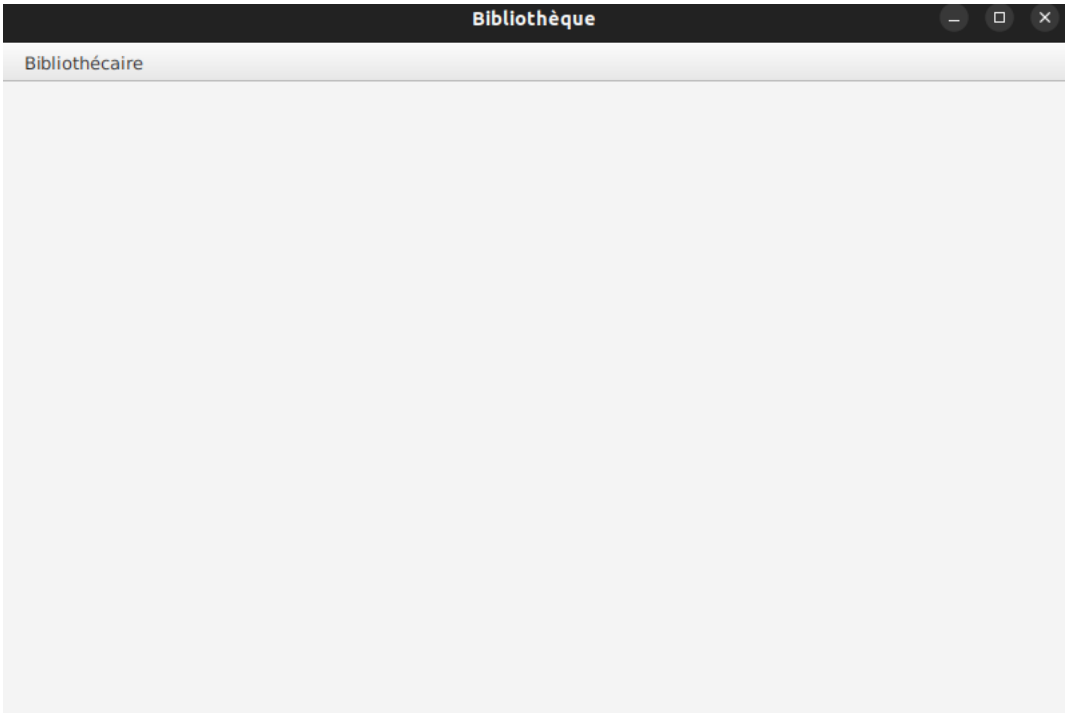
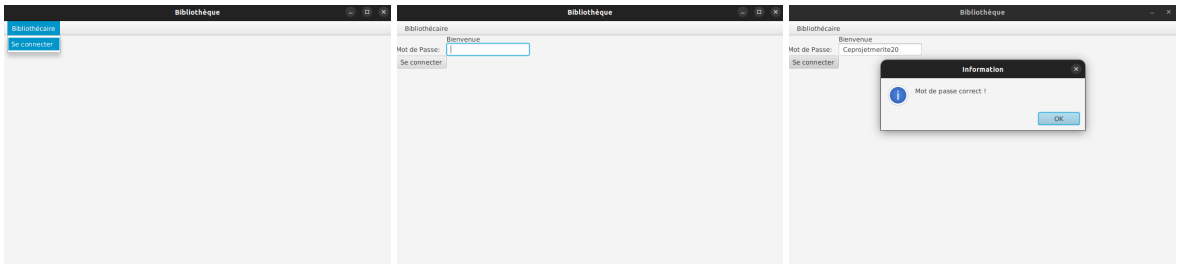


FIGURE 1 – Menu d'affichage

En déroulant le Menu, on a la possibilité de se connecter en tant que bibliothécaire. En cliquant sur ce bouton, un champ de texte pour permettre la saisie du mot de passe s'affiche ainsi qu'un bouton se connecter.

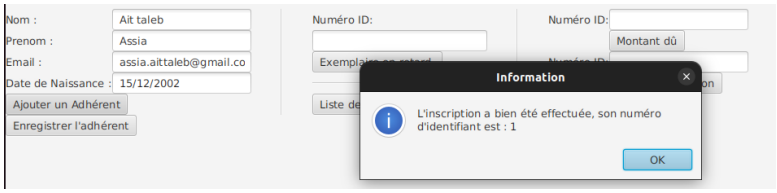


(a) Étape 1 (b) Étape 2 (c) Étape 3

FIGURE 2 – Étape d'affichage

Après s'être connecté en tant que bibliothécaire, des sous-menu s'affichent dans le menu principal à dérouler. Prenons par exemple le sous-menus Gestion Adhérents.

(a) Étape 1



(b) Étape 2

Le bouton Ajouter un Adhérent permet d'ajouter une personne dans la base de données de la bibliothèque et le bouton enregistrer l'adhérent permet quand à lui d'enregistrer toutes les informations saisies dans un fichier texte. Vous pouvez alors tester toutes les fonctionnalités de cette interface graphique.

Conclusion

En conclusion, ce projet nous a permis de mettre en pratique les connaissances acquises en GM3, en Java et en UML.Nous avons commencé par une phase de conception en utilisant des diagrammes UML, ce qui a facilité l'organisation de notre travail.Les diagrammes de cas d'utilisation et de séquence nous ont donné une vue globale du fonctionnement du système, en montrant comment les utilisateurs interagissent avec lui et comment les actions se déroulent. Le diagramme de classes nous a aidé à comprendre l'organisation et la structure du système, en identifiant les différentes parties et leurs relations. Lors de l'implémentation en Java, nous avons été confrontées à la difficulté de faire correspondre notre vision initiale à un système qui fonctionne. Cela nous a montré l'importance de bien concevoir avant de coder, ainsi que la nécessité d'adapter nos approches en cours de route.

Enfin, ce projet a été une occasion de développer notre capacité à travailler en équipe et à intégrer nos différents points de vue. Une méthode que nous avons adoptée était de travailler individuellement sur chaque diagramme, puis de les fusionner pour examiner les diverses approches abordées. Cette méthode nous a permis d'enrichir et d'améliorer nos idées pour ce projet de manière significative. De plus en programmation, nous savons qu'il n'existe pas de manière unique d'aborder un problème, et le fait d'apporter chacun notre propre perspective a été bénéfique et nous a aidé à trouver des solutions lors des différents problèmes rencontrés. Cette diversité d'approches a été importante pour aboutir à la finalité de ce projet. Cette collaboration est un premier pas de ce qui nous attend dans le monde du travail en tant que futur ingénieur.

3 Sources

Pour réaliser ce projet, nous avons bien évidemment eu besoin des cours enseignés cette année en GM3 en Java et en UML. Nous avons complété ces cours par des recherches supplémentaires dont les sources sont joins ci-dessous.

Tutoriel JavaFX

La notion de date en Java