Bordeaux INP – ENSEIRB-MATMECA
Filière Systèmes Electroniques Embarqués

Rapport Technique IF225
Conception logicielle

Gestion d'une médiathèque

DIRIS Jean-Baptiste & SANCHEZ Mathieu

Délivré le : 20/04/2022

Table des matières

1 Présentation

a) Objectif du projet

L'objectif du projet est de mettre en œuvre les notions appréhendées durant l'enseignement d'IF225. Durant cet enseignement, nous avons pu acquérir les notions de base de la programmation orientée objets appliquées au langage C++.
Afin de mener à bien ce projet, il nous sera nécessaire de :

– lire le cahier des charges,
– identifier les besoins exprimés par le client,
– comprendre ce qu'y est à développer,
– élaborer un modèle de solution,
– développer et mettre au point une application.

Dans un premier temps, il est nécessaire de mettre en pratique les notions d'UML vue en cours de Java. Ceci permettra de structurer l'architecture de notre projet facilitant ainsi la compréhension, sa prise en main et la répartition des tâches.
Dans un second temps, il est demandé de mettre en application les notions vues en cours telles que l'héritage, le polymorphisme, la surcharge, etc.).

b) Rappel du cahier des charges

Le projet s'intéresse au développement d'une application permettant la gestion d'une médiathèque. Au vu du temps imparti, l'application sera développée en mode mono-utilisateur, cela signifie qu'un seul et unique utilisateur pourra utiliser l'application à un instant donné.

L'application doit permettre de gérer la disponibilité de l'ensemble des ressources disponibles dans la médiathèque :

- Livres,
- CD,
- DVD,
- Revues,
- VHS,
- Ressources numériques

Les informations mémorisées pour chacune des ressources contenues dans la médiathèque sont dépendantes du type de la ressource. Les informations liées aux ressources seront par exemple : le titre, l'auteur, la durée, le nombre de pages, etc.

L'application devra au minimum fournir à l'utilisateur les fonctionnalités suivantes :

– Ajout et suppression de médias via ***ADD type***, ***DELETE id*** et ***RESET***,
– Consultation des ressources via ***LIST*** et ***SHOW id***,

- Sauvegarde et de chargement de contenu à partir d'un fichier via ***SAVE filename*** et ***LOAD filename***,
- Recherche d'un média à partir d'une information via ***SEARCH chaine*** et ***CLEAR***,
- Réservation, emprunt ou remise d'une ressource.

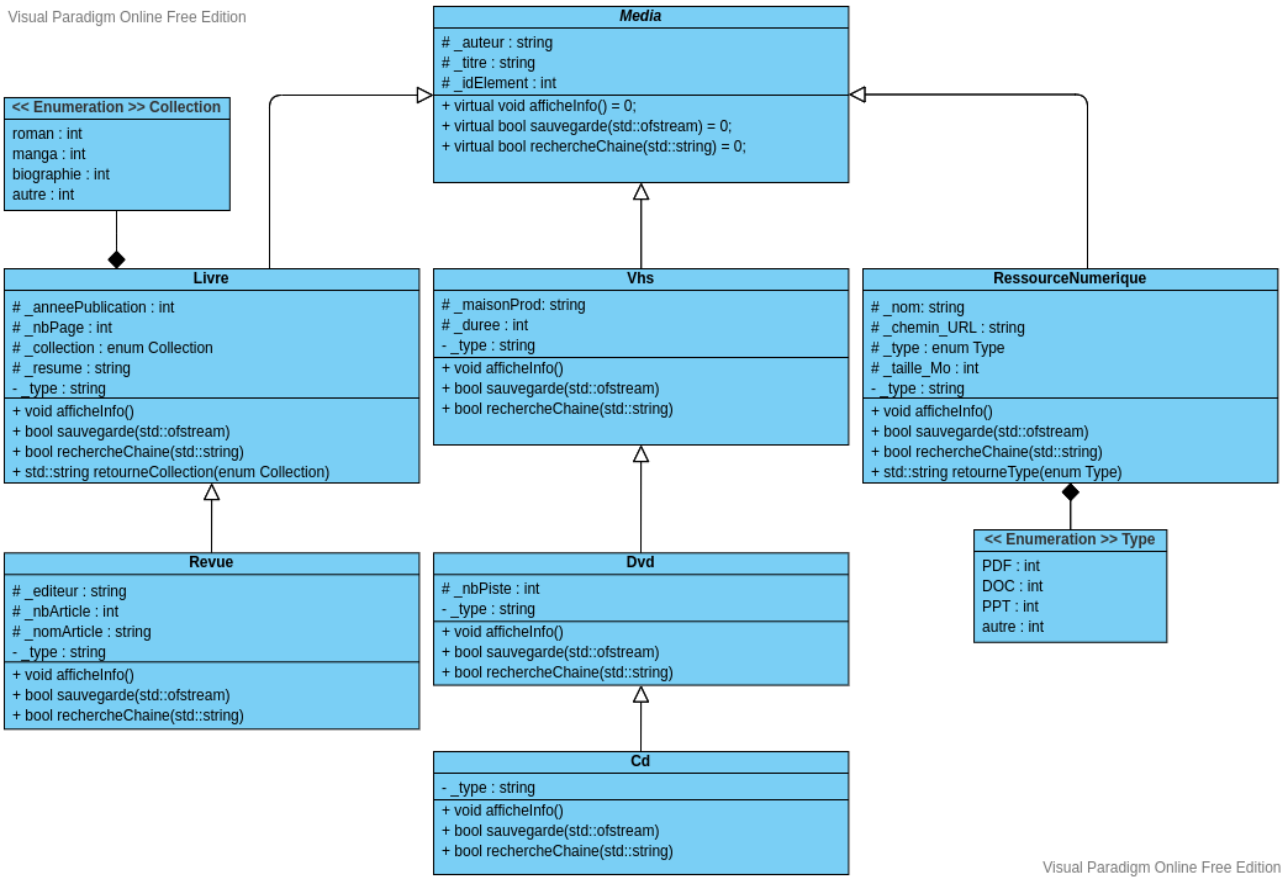
2 Présentation de l'architecture

a) Utilisation de l'héritage et du polymorphisme

Afin d'aborder le projet correctement, nous avons réalisé dans un premier temps un diagramme de classe dans le langage UML. Ce type d'outil nous a permis de décrire correctement la structure de notre médiathèque en modélisant ses classes, ses attributs, ses méthodes et la relation entre chaque classe.

La représentation par diagramme permet à chaque membre de l'équipe du projet, d'avoir le même niveau d'information et de d'entamer le projet de la meilleure des manières.

Ci-dessous le diagramme de classe mettant en évidence la notion d'héritage que nous avons su exploiter dans le cadre de ce projet.



Chaque classe doit être constituée de deux fichiers :

- Le fichier header (.hpp) dans lequel sera cité le nom des attributs ainsi que les prototypes du constructeur et des méthodes.
- Le fichier source (.cpp) dans lequel nous définirons la liste d'initialisation du constructeur, les méthodes et l'implémentation.

La vision d'héritage entre les différentes classes nous a permis de réfléchir à une notion abordée en cours : le polymorphisme. Etant donné que plusieurs classes héritent toutes d'une classe mère, il est donc judicieux de chercher à optimiser notre code afin d'éviter une certaine redondance dans ses sous-classes.

L'objectif est d'obliger le développeur à définir (ou redéfinir) du code au sein de méthodes déjà déclarées dans une classe supérieure.

Pour cela, il est nécessaire d'utiliser la syntaxe suivante :

virtual typeMéthode nomMéthode(typeParamètre nomParamètre) **= 0**;

Une fois ce type de prototype déclaré dans le fichier header de notre classe mère, cette dernière devient alors une classe abstraite ce qui implique qu'elle ne pourra plus être instanciée.

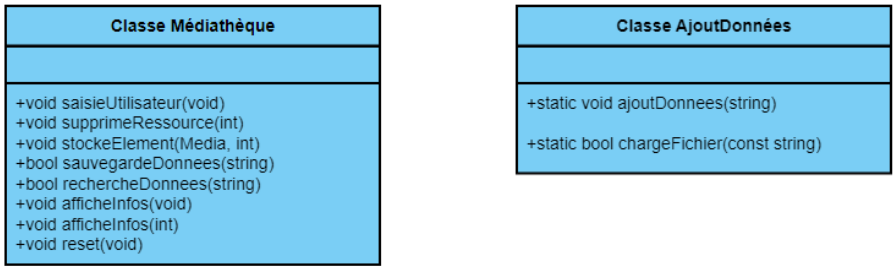
Après avoir déclaré l'ensemble des prototypes des méthodes, nous avons entamé leurs descriptions dans les fichiers source des différentes classes. Voici une description succincte des méthodes en question :

- void NomClasse::afficheInfo(void) : Affiche sur la sortie standard les informations de l'objet pointé.
- bool NomClasse::sauvegarde(ofstream &outfile) : Ecrit les données de l'objet visé dans un fichier.
- bool NomClasse::rechercheChaine(string data) : Retourne 1 si la donnée passée en paramètre est présente dans l'un des attributs de la classe visée sinon 0.
- string Livre::retourneCollection(enum Collection _collection) : Retourne une chaîne de caractère qui correspond à l'énumération passée en paramètre.
- string RessourceNumerique::retourneType(enum Type _type) : Retourne une chaîne de caractère qui correspond au type passé en paramètre.

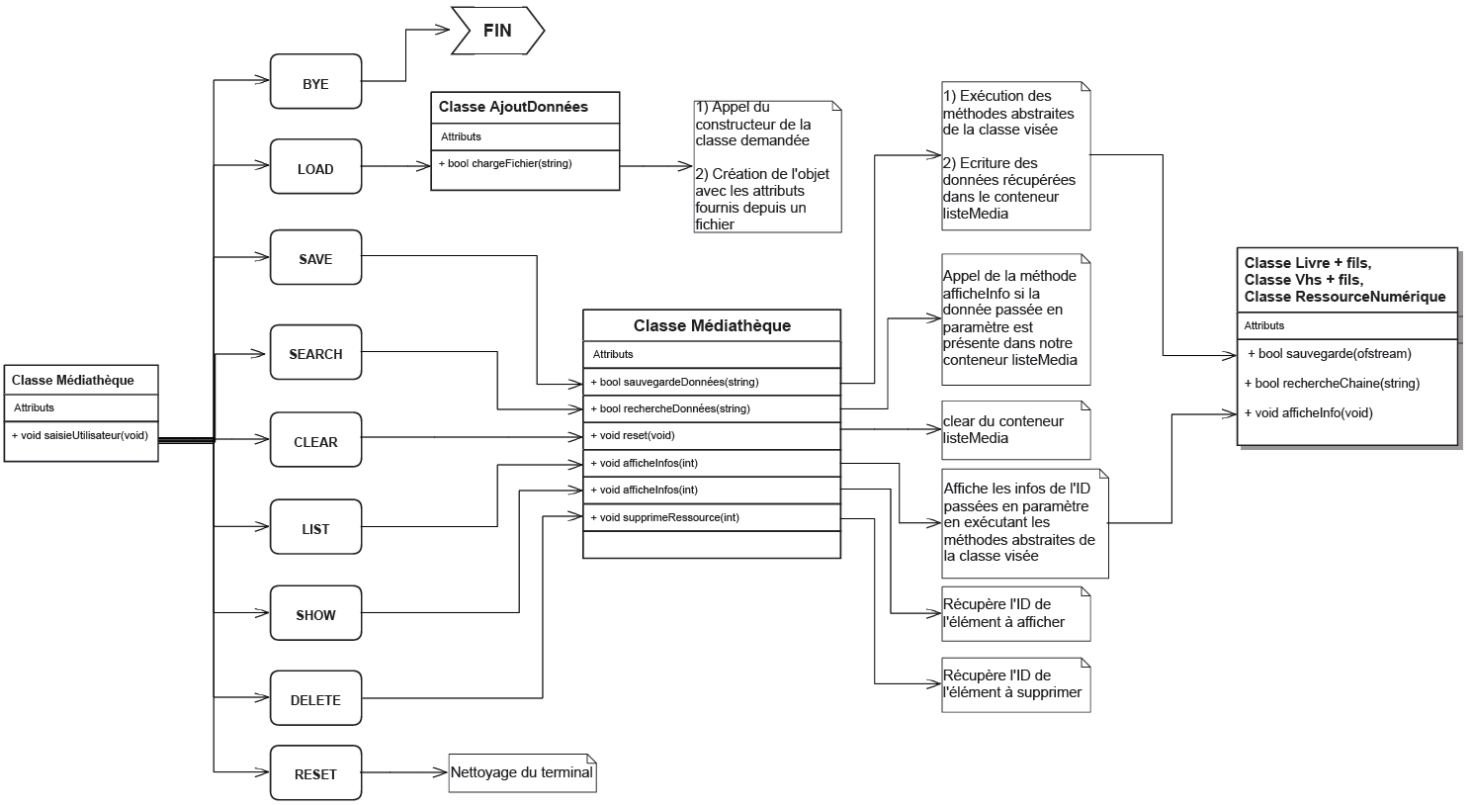
b) Classes permettant l'exécution des commandes

Afin de réaliser l'ensemble des fonctionnalités imposées dans le cahier des charges, nous avons dû enrichir notre architecture de deux classes supplémentaires. Il s'agit là des classes *Médiathèque* et *AjoutDonnée*. La description à travers le diagramme de langage UML ci-dessous permet d'illustrer la constitution de ces deux classes.

La classe AjoutDonnées a été créé afin de diminuer les lignes de codes contenues dans la classe Médiathèque.



L'ensemble des méthodes définies dans ces classes nous ont permis de définir les fonctionnalités de notre application. Afin de mieux comprendre comment nous avons décrit ces méthodes ainsi que leurs interactions avec les différentes classes, il nous a semblé plus compréhensible de représenter cela au travers d'un schéma.



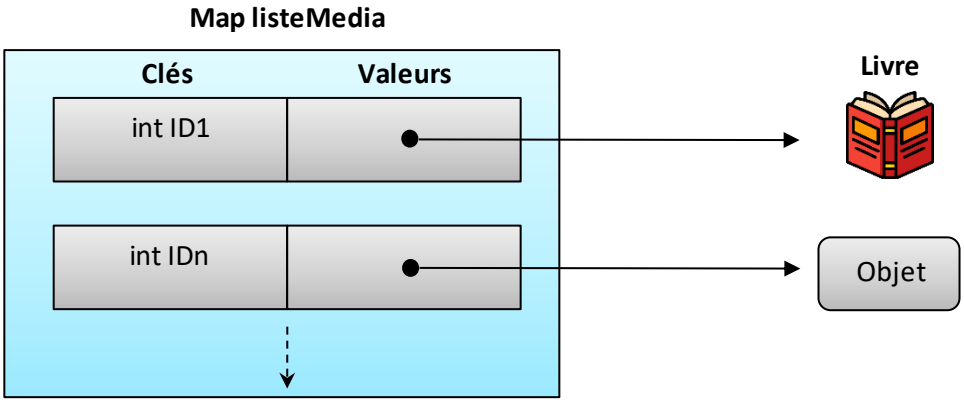
c) Stockage des données

Nous avons vu lors du cours de conception logicielle qu'il est possible d'utiliser des tableaux dynamiques appelés "vector ». Au début de notre projet, nous étions partis sur ce type de tableau à allocation dynamique pour stocker l'ensemble des données de chaque média.

Cependant, lors de la description des fonctionnalités DELETE et SHOW, nous avons rencontrés des difficultés. Afin de pouvoir retrouver nos données facilement, il nous a été nécessaire d'associer un lien entre l'identifiant du média concerné et les données le concernant. C'est pourquoi nous avons décidé de stocker nos données dans un conteneur de type "map".

Ce dernier est comparable à un dictionnaire de données composé d'une clé unique par paire de valeurs. Et l'ensemble des clés ("key") le constituant sont classées. Dans notre cas, on aura un identifiant et un pointeur vers un objet.

Ci-dessous, un schéma représentant la structure de ce conteneur différent des tableaux traditionnels vue lors des cours de C et JAVA.



3 Guide d'utilisateur

Le fichier « readme.md » joint dans le même dossier que ce rapport, permet à l'utilisateur d'avoir connaissance de la marche à suivre pour faire fonctionner notre application.

4 Conclusion

Ce projet fut dans son ensemble très utile, car il nous a permis de monter en compétences sur l'utilisation du langage POO. En effet, les notions vues en Java lors du 1er semestre de deuxième année ont servi à mieux appréhender ce cours de conception logicielle.

Néanmoins, ce projet nous a demandé beaucoup de travail personnel, car les 8h allouées n'étaient clairement pas suffisantes. Notre application est tout de même fonctionnelle et respecte le cahier des charges imposé par l'enseignant.

Cependant, nous nous sommes rendu compte le jour du livrable, de l'oubli de la création des destructeurs. Nous avons conscience de l'intérêt qu'ils portent dans la libération de l'espace mémoire. Ceci n'a pas d'incidence directe sur l'utilisation de notre application, mais pourrait engendrer des difficultés si elle devait être réellement utilisée par un utilisateur.

Afin de ne prendre aucun risque sur le fonctionnement de notre application, nous avons pris la décision de ne pas modifier notre code.