Dévelopement D’application

Alexandre Perron – 11/11/2022

Laboratoire final

Table des matières

[Contexte 2](#_Toc120785434)

[Plan de test 2](#_Toc120785435)

[Algorithme de différents tests 3](#_Toc120785436)

[Compréhension 6](#_Toc120785437)

[Distinction GetListeAttente() dans différentes classes 6](#_Toc120785438)

[Utilité et fonctionnement des méthodes NotifierEmprunt() et NotifierRetour() 6](#_Toc120785439)

[Utilité et pertinence de l’interface IDescription 6](#_Toc120785440)

[Ajout au programme 8](#_Toc120785441)

[Exceptions personnalisées 9](#_Toc120785442)

[Repertoire -> ChargerDocuments : Exception 9](#_Toc120785443)

[Bibliotheque -> TrouverMembre : Exception 10](#_Toc120785444)

[Repertoire -> TrouverDocument : Exception 10](#_Toc120785445)

[Documentation technique 11](#_Toc120785446)

[Post-mortem 12](#_Toc120785447)

[Point fort 12](#_Toc120785448)

[Point faible 12](#_Toc120785449)

[Points d’amélioration 12](#_Toc120785450)

# Contexte

# Plan de test

* AudioTest.cs
  + Constructeur ()
    - Tester si le constructeur de Audio.cs.
    - Tester la fonction GetAuteur () de Audio.cs.
* BibliotèqueTest.cs
  + LongueureArrayDifferantesBibli ()
    - Tester si les différents constructeurs créent les bibliothèques avec des tableaux de Membre de grandeurs différente.
    - Tester la fonction GetMembre ().
  + RetourEtEmprunt ()
    - Tester la fonction AjouterMembre ().
    - Tester la fonction NotifierEmprunt ().
    - Tester la fonction NotifierRetour ().
  + AjoutTropMembre ()
    - Tester la fonction AjouterMembre ().
  + RetirerMembreInnexistant ()
    - Tester la fonction RetirerMembre ().
* DocumentTest.cs
  + CompareLivre ()
    - Tester la fonction CompareTo ().
  + ListeAttente ()
    - Tester la fonction AjouterMembreListeAttente ().
    - Tester la fonction EnleverMembreListeAttente ().
* MembreTest.cs
  + NoMembre ()
    - Tester l’incrémentation du numéro de membre.
  + MaxEmprunt ()
    - Tester la limite de 3 emprunt maximum.
    - Tester la fonction ajouterDocument ().
    - Tester la fonction RetirerDocument ().
* RepertoireTest.cs
  + ChargerDoc ()
    - Tester la fonction ChargerDocuments ().
  + AjouterEtSupprimerDoc ()
    - Tester la fonction AjouterDocument ().
    - Tester la fonction TrouverDoc ().
    - Tester la fonction SuprimerDoc ().

# Algorithme de différents tests

**RetourEtEmprunt**

Créer les éléments suivant pour préparer le test :

2 Membres

1 Document

1 Bibliotheque

Essaie d’ajouter les deux membres à la bibliothèque avec la fonction AjouterMembre (Membre) et en cas d’échec, faire échouer le test.

Essayer de retourner un livre sans en avoir emprunté avec la fonction NotifierRetour (Document), en cas de réussite, faire échouer le test.

Essayer d’emprunter un livre avec la fonction NotifierEmprunt (String, Document) et en cas d’échec, faire échouer le test.

Essayer d’emprunter le livre déjà emprunter avec l’autre membre avec la fonction NotifierEmprunt (String, Document), en cas de réussite, faire échouer le test.

Vérifier Si l’autre membre est dans la liste d’attente du document avec la fonction GetListeAttente () de Document, en cas d’échec, faire échouer le test.

Essayer de retourner le livre avec la fonction NotifierRetour (Document), en cas d’échec, faire échouer le test.

Vérifier si le Membre 2 (étant normalement sur la liste d’attente du doc) est maintenant l’emprunteur avec la fonction GetEmprunteur (). En cas d’échec, faire échouer le test.

**AjouterTropMembre**

Créer les éléments suivant pour préparer le test :

1 Bibliotheque (10 places de membres)

On va boucler 10 fois sur l’algorithme suivant :

Créer un membre et l’ajouter à la bibliothèque avec la fonction AjouterMembre (Membre). Si l’ajout ne fonctionne pas, faire échouer le test.

Créer un nouveau membre.

Ajouter le membre à la bibliothèque avec la fonction AjouterMembre (Membre), si l’ajout fonctionne, faire échouer le test.

**ListeAttente**

Créer les éléments suivant pour préparer le test :

1 Document

4 Membre

Essayer de retirer le premier membre de la liste d’attente du Document(vide) avec la fonction EnleverMembreListeAttente (Membre), si ça fonctionne, faire échouer le test.

Ajouter le premier Membre à la liste d’attente avec la fonction AjouterMembreListeAttente (Membre), si ça ne fonctionne pas, faire échouer le test.

Ajouter le deuxième Membre à la liste d’attente avec la fonction AjouterMembreListeAttente (Membre), si ça ne fonctionne pas, faire échouer le test.

Ajouter le troisième Membre à la liste d’attente avec la fonction AjouterMembreListeAttente (Membre), si ça ne fonctionne pas, faire échouer le test.

Ajouter le quatrième Membre à la liste d’attente(pleine) avec la fonction AjouterMembreListeAttente (Membre), si ça fonctionne, faire échouer le test.

Essayer d’enlever le deuxième et troisième Membre de la liste d’attente avec la fonction EnleverMembreListeAttente (Membre), si ça ne fonctionne pas, faire échouer le test.

Essayer de rajouter le troisième et quatrième Membre dans la liste d’attente du Document avec la fonction AjouterMembreListeAttente (Membre), si ça ne fonctionne pas, faire échouer le test.

**MaxEmprunt**

Créer les éléments suivant pour préparer le test :

5 Document

1 Membre

Ajouter les quatre premiers Document au Membre avec la fonction AjouterDocument(Document). Si ça ne fonctionne pas, faire échouer le test.

Ajouter le cinquième Document au Membre avec la fonction AjouterDocument(Document). Si ça fonctionne, faire échouer le test.

Essayer de retirer le troisième Document au Membre avec la fonction RetirerDocument(Document). Si ça ne fonctionne pas, faire échouer le test.

Ajouter le cinquième Document au Membre avec la fonction AjouterDocument(Document). Si ça ne fonctionne pas, faire échouer le test.

**ChargerDoc**

Créer les éléments suivant pour préparer le test :

1 Document

Ajouter le Document avec la fonction AjouterDocument (Document), si ça ne fonctionne pas, faire échouer le test.

Charger les Documents avec la fonction ChargerDocuments (String)

Regarder si le Document est bien présent avec la fonction TrouverDocument (Document), si ça ne fonctionne pas, faire échouer le test.

**AjouterEtSupprimerDoc**

Créer les éléments suivant pour préparer le test :

2 Document

Essayer de Retirer le document 1 avec la fonction SupprimerDocument (Document), si ça fonctionne, faire échouer le test.

Ajouter le deuxième Document avec la fonction AjouterDocument (Document), si ça ne fonctionne pas, faire échouer le test.

Regarder si le Document est bien présent avec la fonction TrouverDocument (Document), si ça ne fonctionne pas, faire échouer le test.

Supprimer le Document avec la fonction SupprimerDocument (Document), si ça ne fonctionne pas, faire échouer le test.

Regarder si le Document est bien présent avec la fonction TrouverDocument (Document), si ça fonctionne, faire échouer le test.

# Compréhension

## Distinction GetListeAttente() dans différentes classes

**La liste d’attente dans les classes héritant de Document** a pour objectif de stocker les Membres qui désire avoir le Document quand le Document en question est indisponible. L’objectif derrière cette fonctionnalité est de pouvoir donner le livre à un utilisateur qui voulait le livre aussitôt que le livre en question est rapporter par l’emprunteur.

Tout d’abord, la classe Répertoire est une classe qui vas permettre aux utilisateurs de stocker de l’information. Toutes ses fonctionnalités vont servir à chercher, modifier, supprimer ou enregistrer de l’information en dehors du programme.

Donc, **la liste d’attente dans la classe Répertoire** a pour objectif de vérifier tout les Documents qui avait des Membres dans leurs liste d’attente pour ensuite renvoyer ses Documents dans une liste.

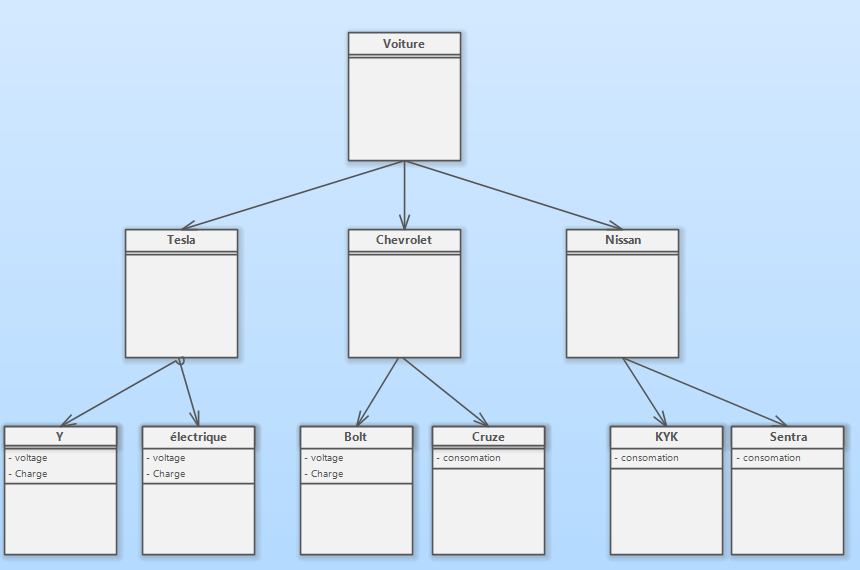
## Utilité et fonctionnement des méthodes NotifierEmprunt() et NotifierRetour()

Les Documents et les Membres on tout les deux une manière de stocker les informations relatives à l’emprunt. Autrement dit, Les membres gardent en mémoire les Documents qu’ils empruntent et les Documents Garde une Liste des Membres qui veulent louer ledit Document. Les utilisateurs de la DLL devront donc choisir une des deux manières de sauvegarder les informations précédentes ou utiliser les deux, et lors d’un retour de livre, informer le Membre que son Document n’est plus dans sa liste et le Document que son Membre n’est plus dans sa liste. Comme les informations du Membre et du Document ont tous les deux leur importance, puisque le Membre stocke jusqu’à quatre Documents empruntés et le Document stock les Membres qui n’ont pas le Document, mais qui désire l’avoir lorsqu’il sera disponible, nous devons idéalement garder les deux informations. Nous revenons donc au problème énoncé plus haut, nous devront appeler plusieurs fonctions à toutes les fois que nous voulons retourner un Document. C’est exactement la raison pour laquelle NotifierEmprunt() et NotifierRetour() existe. Elles sont les fonctions qui permettent de simplement appeler toutes les fonctions relatives à l’emprunt ou au retour pour que l’utilisateur de la DLL puisse simplement utiliser une seule fonction qui appelles les autres et éviter les problèmes d’oubli qui pourrait apporter encore plus de problèmes si le Document et le Membre ne sont pas sur la même page.

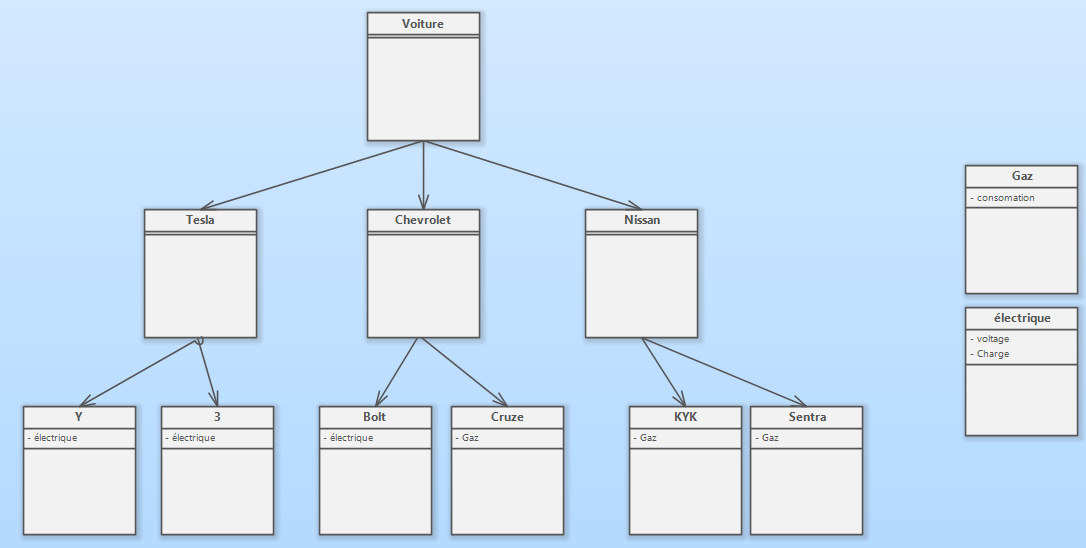
## Utilité et pertinence de l’interface IDescription

Comme toute les classes implémentant l’interface hérite de Document, il aurait été plus simple de mettre une fonction GetDescription () dans la classe Document et ainsi sauver en complexité. Ce qui est encore moins pratique est que dans la classe Document, il y a une fonction qui fait exactement le même travail et cette fonction doit être implémenté par ses filles puisqu’elle est *abstract.* Il y a donc deux fonctions dans plusieurs classes qui font exactement le même traitement.

Nonobstant l’utilisation d’une méthode *abstract*, qui,ici est une très bonne idée si on enlève l’interface IDescription, il va sans dire que l’utilisation d’interface peut être très pertinente. Les interfaces peuvent ajouter des éléments dans un héritage très compliquer comme si dessous.



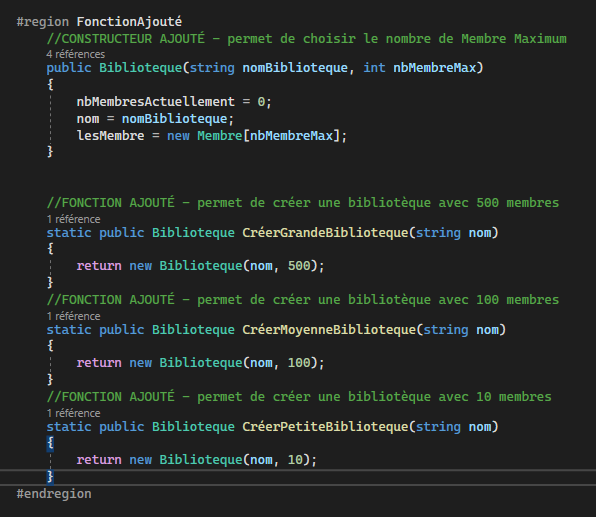
À la place, on pourrait utiliser une interface et rendre l’héritage beaucoup plus simple comme si dessous.



# Ajout au programme

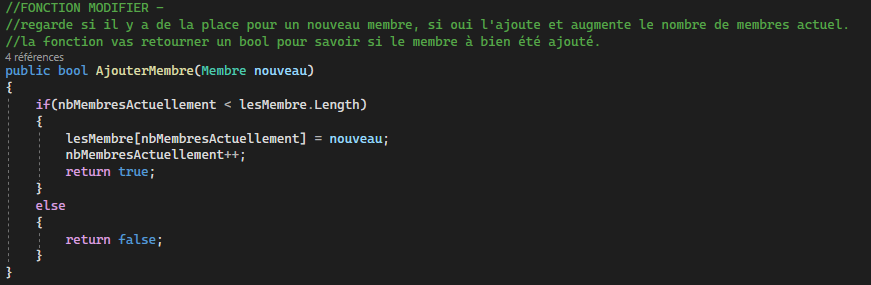
J’ai ajouté un constructeur et trois fonctions qui permettent de construire des bibliothèques de différentes grandeurs. Le constructeur ajouté permet d’ajouter un « int » en paramètre pour le nombre maximum de Membres dans la bibliothèque.

Les différentes fonctions créées permettent de créer des bibliothèques de différentes grandeurs préconçus.



Le type de retour de la fonction AjouterMembre a été changer pour un booléen dans l’objectif de confirmer à celui qui appel la fonction que le Membre a été ajouter.

Une variable a aussi été ajouter : nbMembresActuellement. Cet ajout sert pour pouvoir ajouter plus simplement un Membre dans le tableau.



# Exceptions personnalisées

Trois classes filles de la classe « Exception » ont été créer dans le document ce qui permet une gestion externe des erreurs lancés et ainsi empêcher des erreurs futures lors de l’implémentation.

### Repertoire -> ChargerDocuments : Exception

Héritage : Object -> Exception -> EnregistrementDocumentErrorException

EnregistrementDocumentErrorException : si le document n’est ni un livre, ni un audio et ni un périodique dans le document .txt, retourne une erreur.

Une image contenant texte

Description générée automatiquement

### Bibliotheque -> TrouverMembre : Exception

Héritage : Object -> Exception -> MembreNotFoundException

MembreNotFoundException : si le membre envoyé n’est pas trouvé, on renvoie une erreur de type MembreNotFoundException.

Une image contenant texte

Description générée automatiquement

### Repertoire -> TrouverDocument : Exception

Héritage : Object -> Exception -> EnregistrementDocumentErrorException

EnregistrementDocumentErrorException : si le document envoyé n’est pas trouvé, on renvoie une erreur de type EnregistrementDocumentErrorException.

Une image contenant texte

Description générée automatiquement

# Documentation technique

Nouveau diagramme de classe :

Une image contenant texte, intérieur, capture d’écran

Description générée automatiquement

Nouveau diagramme :

Une image contenant texte

Description générée automatiquement

# Structure

Je trouvais très simple comme manière d’exposé les résultats de tous les tests que de mettre en affichage console une description suivit de si le test fonctionne ou non. Chaque test teste différents aspects de chaque classes qui permet de diviser les tests de manière efficace et de présenté les résultats de manière simple et structuré.

# Post-mortem

## Point fort

La logique dans la création de fonction est probablement mon point le plus fort. J’ai beaucoup de facilité à imaginer plusieurs manières différentes de résoudre un problème et d’utiliser la meilleure méthode. C’est ce genre d’habilité qui permet de faire des codes qui sont efficace, concis et qui ne sont pas trop demandant à produire en temps et en énergie.

## Point faible

Je sais que je n’ai surement pas su voir tous les tests possibles, ma pensé critique n’est pas assez développé quant à l’évaluation des problèmes possibles à rencontrer. Je crois que ce serait un point à amélioré chez moi puisque c’est une grande lacune qui peut causer un problème de programme qui plante. Pour améliorer ce point, je crois que je peux pratiquer et ça finira par devenir plus facile, mais il est certain qu’il faudrait que je pense à mettre plus de commentaire lors du code sur les fonctions qui peuvent appeler des exceptions ou même penser à mes tests en programmant et programmer avec un optique de faire les tests sur ses fonctions.

Aussi, mon temps aurait pu être mieux géré, certains éléments que je croyais long m’ont pris peu de temps et des éléments que je croyais être court et m’ont pris beaucoup de temps. Ce n’est pas un problème qui impacte le code en soit, mais c’est un problème qui affecte les dates d’échéance et qui doit être amélioré.

## Points d’amélioration

J’avais plusieurs lacunes du côté des projets de tests et des vérifications. Je crois que le fait de travailler sur un projet où la seule manière d’exécuter le code est dans des projet autres qui utilise la dll, m’a permis de revoir ma manière de travailler et j’ai plus pris le temps de bien faire mon code du premier coup. J’ai aussi été très motivé puisque j’ai beaucoup d’intérêt à faire des projets comme une dll. Si je devais refaire une dll j’essaierais de réfléchir aux erreurs personnalisées possible en composant le code. Je crois que la manière dont j’ai réfléchi les erreurs personnalisées n’a pas été efficace étant donné qu’après avoir réaliser le code, j’ai cherché à inventer des erreurs qui aurait pu aller à quelque part, quand en fait, j’aurais dû réfléchir à ça pendant la programmation.