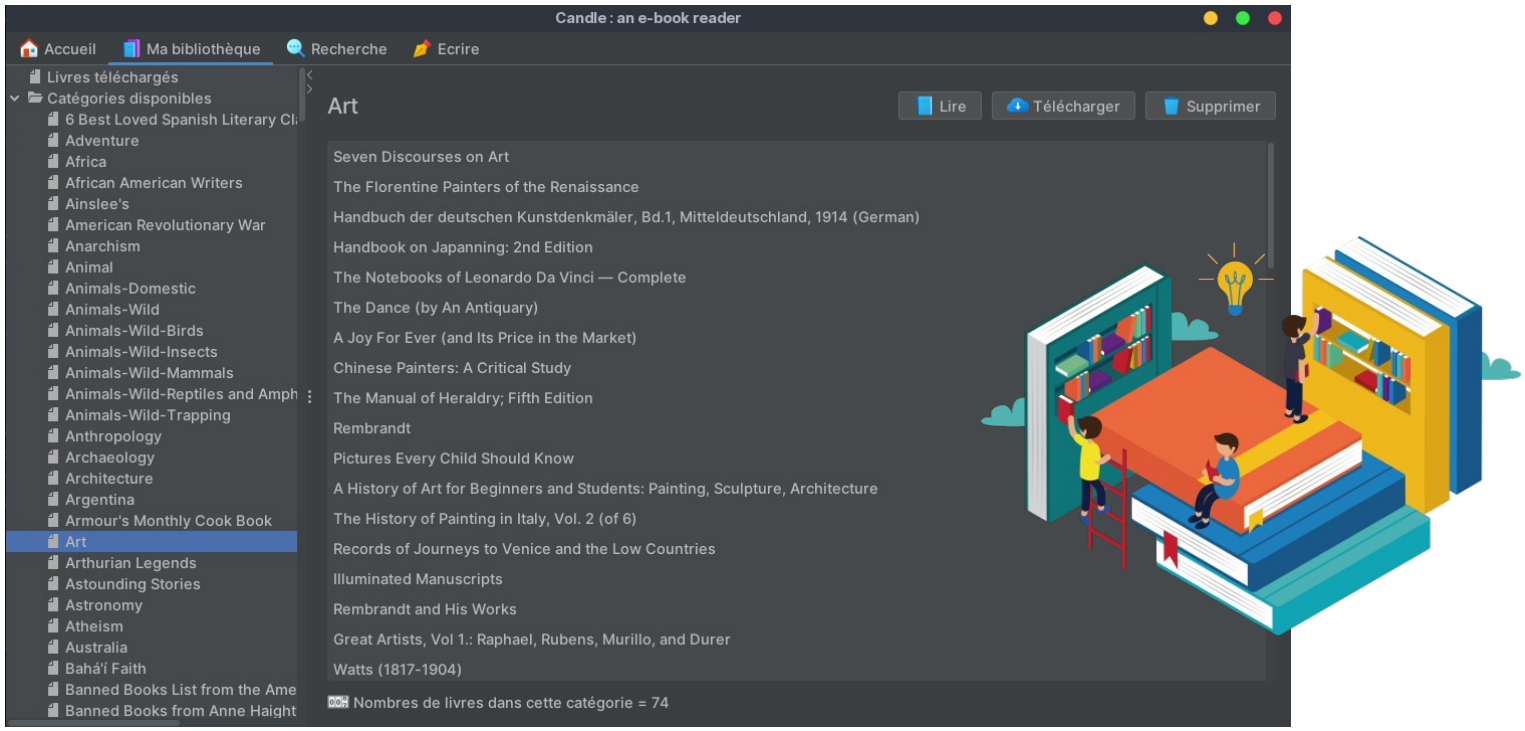
PROJET LISEUSE – POO2 L2S4A

Saday Arhun - 21803102



Introduction

L'objectif de ce projet était de créer une liseuse de livres en utilisant un langage orienté objet et une interface graphique. Dans ce rapport je présenterais le travail effectué, les choix de modélisation et d'implémentations. Je montrereais également le diagramme des classes de mon application et je parlerais es quelques problèmes que j'ai pu rencontrer et la manière dont je les ai résout.

Choix de modélisation et organisation des classes

Tout d'abord, j'ai réalisé ce projet en utilisant le langage Java et Swing comme librairie pour l'interface graphique et quelques-uns autres externes.

Pour réaliser cette application quelque peu complexe, il était important de choisir des design patterns notamment pour avoir un code propre, facile à débuguer, tester et à améliorer.

Pour ce qui est du patron d'architecture, j'ai décidé d'utiliser celui appellé "modèle-vue-présentateur" qui dérivé du fameux "modèle-vue-controlleur". J'ai pris cette décision pour comparer les deux mais également parce que celui-ci me paraissait plus simple, plus intuitive puisqu'il élimine les interactions entre le modèle et la vue. L'idée est simple, le modèle s'occupe de tout ce qui est liées aux données et à la logique de celle-ci. La vue affiche simplement l'interface et redirige les interactions utilisateur vers le présentateur. Le présentateur redirige vers le modèle qui effectue des tâches. Ensuite le modèle notifie le présentateur avec le résultat et le présentateur redirige le résultat vers la vue. Le présentateur connait le modèle et la vue mais les deux ne connaissent que le présentateur.

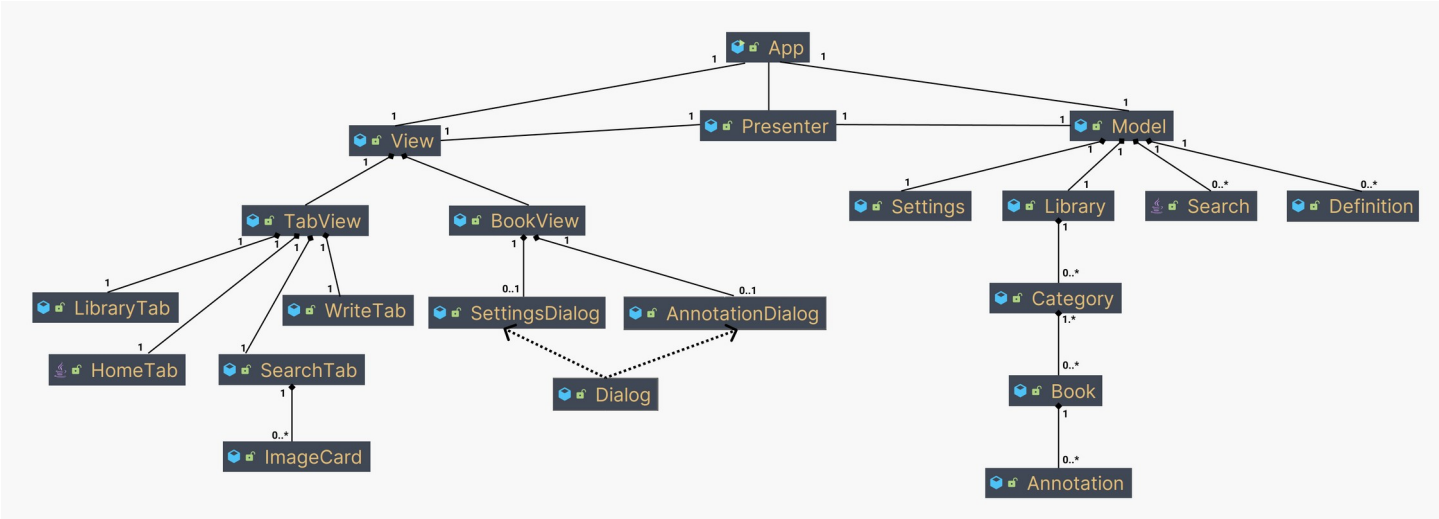
Je vais donner un example dans le cas de ce projet pour illustrer cette organisation. Lorsque l'application est lancé, le présentateur récupère les données du modèle et les donne à la vue pour que la vue s'initialise puis ensuite affiche la vue. La vue est constitué d'un seule classe mais les composants de la vue sont séparés dans des classes distincts. La vue instancie tout ces classes qui le composent en leur passant une référence à soi.

Par exemple lorsque l'utilisateur clique pour télécharger un livre, l'écouteur d'évènement dans la classe "LibraryTab" appelle un fonction dans la vue appellée "notify_download_performed" qui notifie le présentateur qu'on veut télécharger un certain livre. Le présentateur relaie l'information au modèle et attend une réponse. Le modèle appelle la fonction "download_book" de la classe livre puis renvoie le résultat au présentateur. Le présentateur peut maintenant notifier la vue qui peut afficher le résultat de l'opération et ainsi de suite.

Modélisation UML

Voici la modélisation UML des classes. Il permet d'observer l'architecture globale de l'application, les relations hiérarchique entre les différentes classes etc. J'ai choisi de ne pas mettre les variables et les méthodes pour ne pas encombrer et ne voir que la structure essentielle. Malgrès cela si c'est difficilement lisible, voici un lien vers l'image dans une plus grande dimension.

https://i.imgur.com/1Z9YDnm.png



Fonctionnalités et choix d’implémentations

Voici un briève résumé des fonctionnalités et des choix faits durant l’implémentation de celles-ci.

Tout d’abord l’affichage des étagères virtuelles correspondent aux catégories disponibles sur le site. Je récupère cette liste en faisant une requête http au page web "https://www.gutenberg.org/ebooks/bookshelf/", je stocke le résultat et je lis le contenu ligne par ligne pour détecter ce qui ressemble à un lien vers une catégorie. C’est assez fastidieux donc pour les autres problèmes de ce genre, j’ai utilisé une librairie qui s’appelle Jsoup et qui permet par exemple d’utiliser la syntaxe de sélection du langage CSS pour récupèrer directement la liste des élements correspondants. Ensuite lorsque l’utilisateur clique sur une catégorie qui n’est pas “Livres téléchargés” je fais une rêquete à une autre page pour récupèrer la liste des livres. Je fais cela à chaque fois, j’aurais pu choisir de télécharger la liste au lancement de l’application et utiliser par la suite ce fichier et cela rendrait l’application beaucoup plus rapide mais au détriment d’une liste pas forcément à jour. Une solution hybride serait surement la plus adaptée.

Un autre problème s’est présentée lors de la lecture ou téléchargement d’un livre. La plupart des livres sur le site Gutenberg ont un fichier au format texte qui se trouve dans un lien finissant par “cache/epub/67042/pg67042.txt” où il suffit de remplacer le numéro par l’identifiant du livre pour obtenir le contenu. Malheureusement ce n’était pas le cas pour tous les livres, j’ai trouvé en tout 4 types de nom/lien différents. J’ai d’abord essayé de gérer cela en faisant une rêquete à une page où on pouvait trouver les différents formats pour un livre donné mais cela introduisait trop de délai alors je suis revenu en arrière et pour la plupart des livres qui ont un lien différent j’affiche un message qui dit que le livre n’a pas été trouvé. Ce problème est malheuresement liée au faite que le site ne dispose pas d’une vraie API/interface pour les développeurs et donc il faut trouver des solutions malins et pas toujours parfaits pour extracter des informations.

Le téléchargement des livres se fait assez facilement, j’ai choisi de créer un sous dossier caché dans le répertoire de base de l’utilisateur ($HOME). Lors que je télécharge le livre, je stocke le fichier texte correspondant au texte et un autre fichier au format .json pour stocker entre autres les annotations et la dernière position dans la lecture pour pouvoir les recharger lorsque l’utilisateur

quitte et rouvre l’application. Pour cela j’ai utilisé la librairie json-simple qui facilite le travail avec les fichiers .json.

Dans la vue pour lire un livre, j’affiche un pourcentage de progression comme sur un Kindle au lieu d’avoir une pagination. L’avantage est qu’on peut parcourir et visualiser le contenu du livre. Le menu de clique droit dans un livre fait apparaitre un menu où on peut copier la sélection, ajouter, visualiser, supprimer une annotation. Les annotations ont un champs qui indique le début, un qui indique la fin et un autre qui indique le texte. Je vérifie par exemple lors de l’ajout s’il n’y a pas d’annotations déjà placés à cette endroit, s’il y a bien un texte sélectionné etc. Pour afficher visuellement ces annotations, j’utililise l’interface Highlighter de Java qui permet de marquer un bout de texte avec un fond coloré.

J’ai également une fonctionnalité supplémentaire dans la vue livre, un bouton dans le menu de clique droit pour obtenir la définition d’un mot. Cela fait simplement une rêquete à une API pour obtenir le résultat et l’afficher. Deux autres fonctionnalités intéressantes sont l’onglet pour faire une recherche et afficher les livres correspondants avec les images et sur lesquelles on peut cliquer pour les lire. Le dernier onglet est petit éditeur de texte pour ceux qui seraient inspirés d’écrire après avoir lu.

La plupart des erreurs sont gérés en renvoyant une valeur anormale comme null ou comme dans le cas du téléchargement des livres puisse afficher un message d’erreur ou comme dans le cas du téléchargement, lorsque le modèle fini l’opération, envoie une chaine de caractères au présentateur qui à son tour notifie la vue d’afficher le message. Ce message peut être “le livre X a été téléchargé” ou par exemple “le livre X est déjà téléchargé”. Une meilleure solution aurait été d’avoir une seule fonction dans le modèle, présentateur et vue seulement pour l’affichage de messages d’informations et d’erreurs pour éviter les redondances et rendre le code encore plus simple.

Enfin, pour les opérations vraiment lourdes et longs, je profite évidemment du multi-threading en créant à chaque fois un SwingWorker qui permet de faire des opérations dans un nouveau thread pour ne pas bloquer les interactions utilisateur.

Conclusion

Dans l’ensemble, ce projet était très intéressant pour son aspect appliqué. Les difficultés n’étaient pas très frustrants car le langage est très facile à débuguer avec un IDE et ils sont plutôt liées aux choix de modélisation et de structuration du code. La création de cette application m’a permis de me rendre compte un peu plus de l’importance des choix qu’on fais au tout début d’un projet car ces choix nous suivent tout au long et s’ils sont mauvais, ils peuvent être handicapantes.