EPHEC (LLN) 2016 - 2017

Rapport de projet Java

Mathieu Rousseau – Nathan Dolinski – Sebastien Hals

**Développement informatique avancé – orienté application**

Rapport de projet Java – Konopolis

Table des matières

[1. Introduction et cahier des charges : 2](#_Toc470193477)

[Composition du groupe (Groupe 3) : 2](#_Toc470193478)

[Concrètement : 2](#_Toc470193479)

[2. Diagramme UML 2](#_Toc470193480)

[2.1 Brèves explications 3](#_Toc470193481)

[3. Choix d’implémentation effectués : 3](#_Toc470193482)

[3.1 Architecture : 3](#_Toc470193483)

[3.2 Structures de données : 4](#_Toc470193484)

[3.3 Partie réseau : 5](#_Toc470193485)

[3.4 Threads : 5](#_Toc470193486)

[3.5 Java 8 : 5](#_Toc470193487)

[4. Difficultés rencontrées 5](#_Toc470193488)

[5. Pistes d’améliorations éventuelles 6](#_Toc470193489)

[6. Commentaires 6](#_Toc470193490)

[6.1 Dolinski Nathan : 6](#_Toc470193491)

[6.2 : Sébastien Hals 7](#_Toc470193492)

[6.3 Mathieu Rousseau 7](#_Toc470193493)

# Introduction et cahier des charges :

Le but de notre projet était de créer une application de type « utilitaire » qui permet de gérer un cinéma.

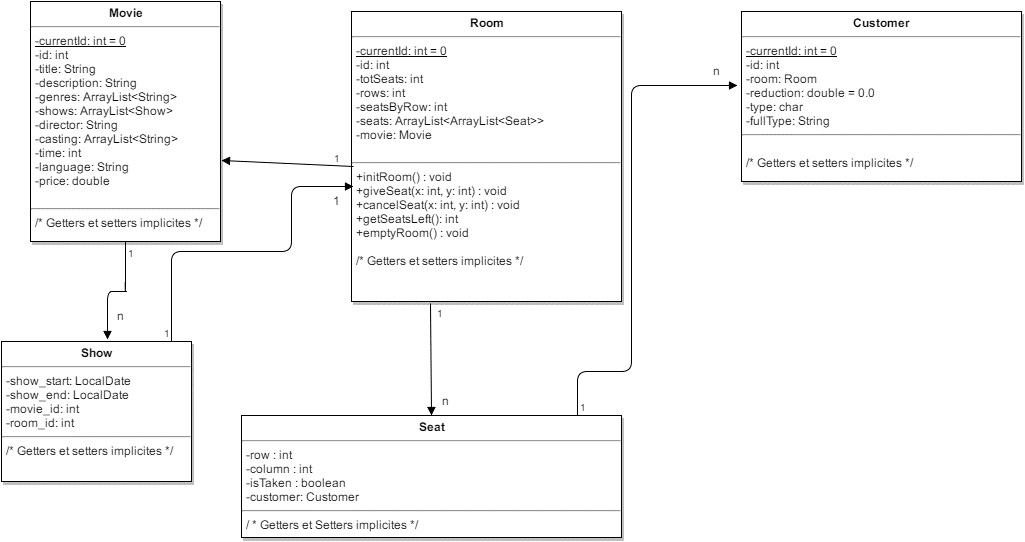
## Composition du groupe (Groupe 3) :

* Mathieu Rousseau
* Nathan Dolinski
* Sébastien Hals

## Concrètement :

* 2 interfaces (CLI / GUI).
* Une fenêtre affiche les sièges disponibles/occupés en salle.
* Réduction en fonction du statut de la personne (Etudiant, VIP,...).
* Possibilité de "switcher" entre les salles selon le choix du film.
* Possibilité pour un administrateur d’ajouter un film à la base de données.
* Une base de données contient les films, le nombre de sièges libres restant, le numéro des sièges pris, ...

# Diagramme UML



## Brèves explications

* La classe Movie contient toutes les infos nécessaires pour un film.
* La classe Show contient les dates de séances associées à un film et une salle.
* La classe Room contient toutes les infos à propos d’une salle de cinéma (nombre de rangées, nombre de sièges par rangée) ainsi qu’un « ArrayList » « d’ArrayList de siège » qui permet de créer la représentation graphique de la salle où le « siège » indique si celui-ci est pris ou non.

Elle contient des méthodes permettant de créer la salle, de réserver un siège, de savoir combien de sièges libres il reste dans la salle.

* La classe Seat représente un siège et indique ses coordonnées dans la salle ainsi que si celui-ci est pris ou non.
* La classe Customer représente un client et indique dans quelle salle il a réservé sa place ainsi que la réduction qui lui est attribuée.

# Choix d’implémentation effectués :

## Architecture :

L’application utilise l’architecture MVC (Modèle – Vue – Contrôleur).



* La vue représente ce que l’utilisateur voit, que ce soit une interface console (CLI) ou graphique (GUI). Les traitements effectués sur celles sont réalisés par le contrôleur et la vue « envoie » au contrôleur les actions utilisateurs (click,...).
* Le contrôleur envoie les changements au modèle. C’est lui qui contrôle les données et fait le lien entre la vue et le modèle.
* Le modèle reçoit les requêtes de la vue et notifie les changements d’état directement à la vue (au moyen de NotifyObservers() et SetChanged()).

C’est également là que se trouvent les données récupérées de la base de données.

## Structures de données :

L’application utilise plusieurs types de données, chacune étant minutieusement choisie.

* ArrayList : Le problème que posent les tableaux (Array) est que leur taille doit être prédéfinie. Dans la quasi-totalité des cas, nous ne pouvions pas prévoir à l’avance leur taille (Salle de cinéma pouvant avoir plusieurs tailles,…). Les ArrayList sont donc un choix tout désigné puisqu’elles ne demandent pas qu’on leur fournisse une taille fixe au préalable.
* LinkedHashMap : Ici, c’est un petit peu plus compliqué. La raison pour laquelle nous avons utilisé une LinkedHashMap et non pas une « simple » HashMap est la suivante. Cette première permet de stocker la paire « id de film » - « titre du film » afin de remplir une liste de films disponible au cinéma. Seulement, par exemple, le 1er film de la liste n’est pas spécialement le film ayant l’id (clé primaire de la table tbrooms) n°1 dans la base de données. Donc, quand un film est sélectionné par l’utilisateur, il faut aller rechercher son id (ou son titre) dans la liste.

C’est là que l’HashMap pose problème, celle-ci ne possédant aucun index car ce n’est pas une liste ordonnée contrairement aux LinkedHashMap.

* HashMap : Pour toute structure de données ne demandant pas d’être ordonnées, les HashMap permettent de facilement stocker des données sous forme de paires

« clé » - « valeurs ».

## Partie réseau :

* L’application utilise une base de données MYSQL et JDBC pour la connexion et la requête à celle-ci.

## Threads :

* L’interface console et graphique fonctionnant en même temps, il fût nécessaire d’implémenter des Threads et de gérer la concurrence afin que les 2 instances soient « synchronisées ».

## Java 8 :

* L’application profite de quelques fonctionnalités sympathiques de Java 8 tels que les « lambda » ( () -> ) afin de simplifier l’écriture de certaines méthodes (boucles forEach, actionListeners,…).

# Difficultés rencontrées

* La base de données a été difficile à implémenter car on a souvent dû la remodifier du fait qu’on avait du mal à avoir une vision globale et claire du projet ainsi que des nécessités qu’on allait avoir.
* Beaucoup de problèmes avec Swing au niveau de la position des éléments et du traitement des entrées de l’utilisateur qui provoquaient quelques bugs assez contraignants.
* L’implémentation du pattern MVC a été assez compliquée car le concept était assez difficile à comprendre et à s’approprier surtout en ce qui concerne les méthodes telles que setChanged(), *notifyObservers()* et *Update()* qui soit entrainaient des bugs ou soit des problèmes de la méthode d’implémentation.
* Parfois certains bugs, surtout au niveau de la console, entrainaient à leur correction d’autres bugs qui pouvaient s’avérer difficile à corriger.

# Pistes d’améliorations éventuelles

* Le programme pourrait être bien mieux optimisé car les requêtes SQL entrainent un gros ralentissement des performances. Au lieu de faire plusieurs petites requêtes, il serait peut-être mieux de faire de grandes requêtes au début qui rempliraient des collections et des variables afin de ne plus avoir à en faire (ou alors très peu). Cela demanderait donc une grosse utilisation des ressources à l’initialisation mais ensuite le programme serait plus fluide, plus allégé.
* L’interface graphique n’est pas très jolie et mériterait un petit lifting notamment en utilisant JavaFX.
* Une petite amélioration de l’implémentation du pattern MVC pourrait bien améliorer les performances et la cohérence du programme.

# Commentaires

## 6.1 Nathan Dolinski :

* Le choix du programme à réaliser a été difficile ce qui m’a laissé indécis pendant un bon moment. Finalement, nous avons choisi de faire ce programme de gestion de cinéma qui, bien qu’il m’ait permis de bien saisir les concepts du langage JAVA, n’a pas été une grande source de plaisir et peut-être qu’un autre sujet aurait dû être choisi. Peut-être un logiciel un peu plus ludique aurait pu être plus intéressant.

## 6.2 : Sébastien Hals

* L’idée de faire un programme de gestion pour le cinéma m’a fort intéressé au début, cela m’a permis d’un peu mieux comprendre le langage JAVA. Mais j’ai éprouvé beaucoup de difficultés à comprendre le modèle MVC (De la réaliser, la mettre en place).

## 6.3 Mathieu Rousseau

* Ayant déjà réalisé plusieurs projets auparavant, que ce soit des projets personnels ou de groupes, j’avais déjà une petite expérience en la matière et j’avais l’avantage de déjà connaître l’outil de versionning « git ». Lors de l’annonce du projet Java, je venais d’avoir eu un projet avec *Julien Castiaux (2TL2)* qui consistait en la réalisation d’un Bomberman (jeu) dans le navigateur (html – css – js – websockets et python). Le problème était que j’ai eu beaucoup de mal à gérer les sprites ; ce qui a conduit à « l’abandon du projet ». J’étais donc moins motivé à l’idée de réaliser un jeu en Java connaissant toutes les problématiques que cela apporte. En discutant avec mon groupe, on a décidé de quand même chercher de ce côté mais le projet imaginé était soit trop ambitieux, soit déjà pris par un autre groupe. Nous avons finalement décidé de faire une application de gestion de salle de cinéma. Au début, cela avait l’air sympa mais au fur et à mesure que le temps passait, j’ai perdu le plaisir à le réaliser et à la fin de l’échéance, l’énorme manque de motivation a fait que j’ai eu du mal à terminer le projet. J’ai finalement passé la plupart du temps sur ce projet à faire de l’administration, des requêtes SQL,… Mais pas tellement de logique et algorithmique pur que j’aurais surement préféré faire. Malgré tout, ce projet m’a permis d’entrer plus en profondeur dans ce langage que je ne connaissais quasi pas et de découvrir un langage certes lourd (« Java plus c’est long, plus c’est bon »1) mais propre et structuré en même temps.

Et même si je compte faire un tour du côté du C#, je vais sûrement utiliser Java pour certains projets (perso ou non) par exemple avec JavaFX. Pour ce qui est de ma contribution au projet, j’ai réalisé le modèle, les classes « Room » et « Seat » ainsi que les 2 vues (CLI et GUI) avec Nathan. Pour conclure, je pense qu’il est mieux de se lancer dans un projet qui certes peut paraître ambitieux et fort compliqué mais que l’on sait que l’on va vraiment aimer plutôt qu’un projet qui parait facile mais qui au final ne l’est éventuellement pas tellement mais surtout, au-delà de cela, qui ne nous amuse pas du tout. La motivation permettra de faire face et de surmonter les difficultés rencontrées lors du développement du projet.

Voilà une leçon que j’ai apprise durant ce projet.