**Développement informatique avancé – orienté application**

Rapport de projet Java – Konopolis

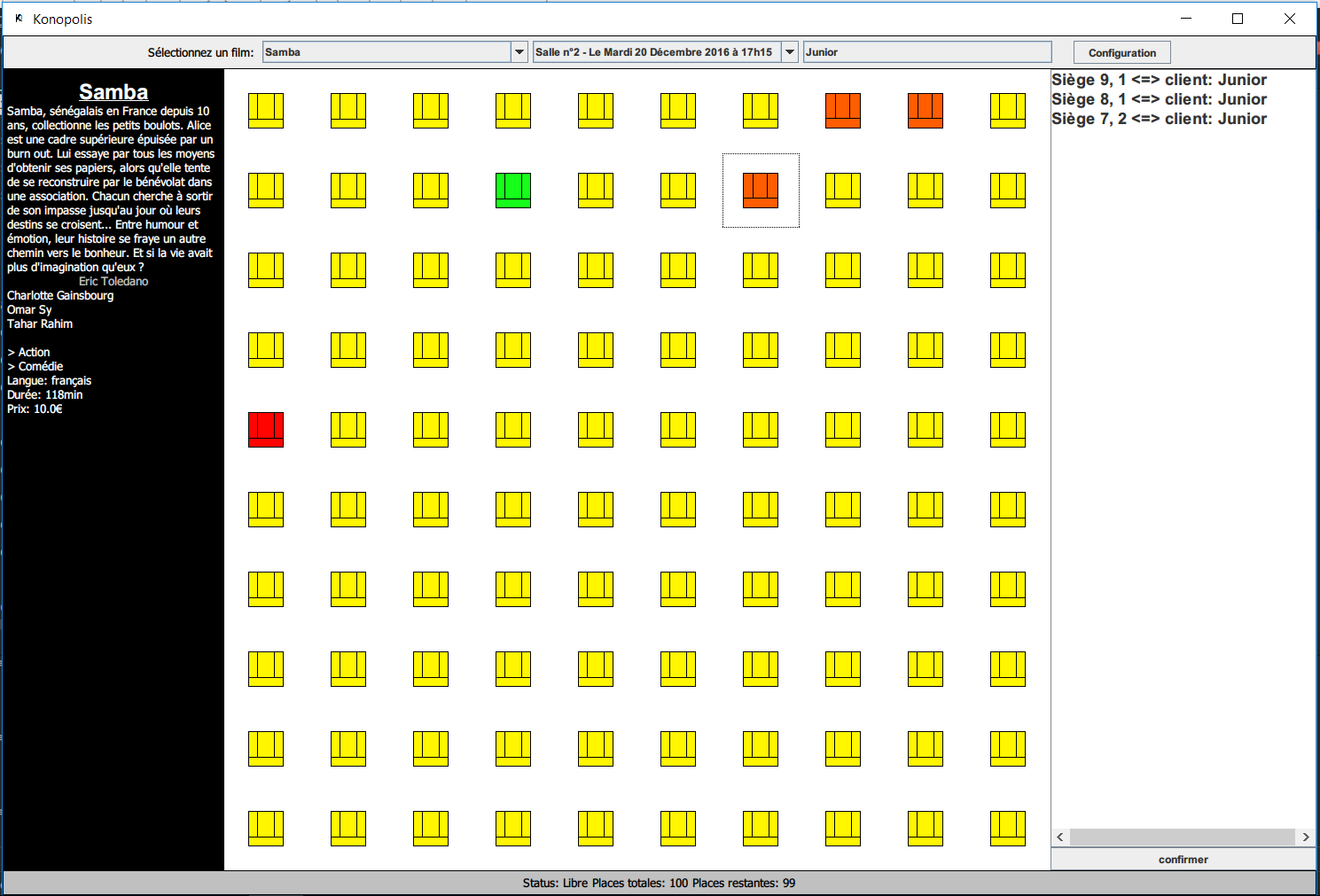


Table des matières

[1. Introduction et cahier des charges : 3](#_Toc470114180)

[Composition du groupe (Groupe 3) : 3](#_Toc470114181)

[Concrètement : 3](#_Toc470114182)

[2. Diagramme UML 4](#_Toc470114183)

[2.1 Brèves explications 4](#_Toc470114184)

[3. Choix d’implémentation effectués : 5](#_Toc470114185)

[3.1 Architecture : 5](#_Toc470114186)

[3.2 Structures de données : 6](#_Toc470114187)

[3.3 Partie réseau : 6](#_Toc470114188)

[3.4 Threads : 6](#_Toc470114189)

[3.5 Java 8 : 6](#_Toc470114190)

# 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
* Sebastien 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 admin 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 siège libre 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 attribué.

# 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 « envoi » 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 trouve les données récupérées de la base de données.

## Structures de données :

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

* ArrayList : Le problème que poses 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 suivant. Cette première permet de stocker la paire « id de film » - « titre du film » afin de remplir une liste de film 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 fonctionnalitées sympathiques de Java 8 tels que les « lambda » ( () -> ) afin de simplifier l’écriture de certaines méthodes (boucles forEach, actionListeners,…).