PHILIPPE TOM
TABARE RUBEN
WANG ANTOINE

APPLIS SERVEUR JAVA

CCII

Bibliothèque

Projet

SOMMAIRE

CHAPITRE I

Classe « Alert »

Cette classe implémente Runnable, le lancement d'une alerte ne devant pas bloquer l'exécution du programme (d'où le choix d'une exécution en parallèle). Ce thread est placé en attente passive sur la ressource partagée (le livre) via la méthode wait(). Lorsque le livre devient disponible, à la fin de la méthode retour(), la méthode notifyAll() est invoquée pour notifier le thread « Alert » placé en attente.

CHAPITRE II

Classe « Abonne »

Acte I – Gestion des alertes

La classe abonnée possède une liste de documents. Ce sont les documents sur lesquels une alerte a été placée. Nous pouvons ainsi gérer les alertes et empêcher la superposition d'alertes sur un même livre pour un même abonné.

Acte II – Méthodes synchronized

public synchronized void alerter(Document doc) :

On suppose que l'abonné puisse effectuer plusieurs réservations en même temps (sur plusieurs postes par exemple). Il serait donc possible que cette fonction soit appelée simultanément plusieurs fois sur le même abonné.

public synchronized void interdire() et public synchronized void finInterdiction() :

L'abonné doit être interdit d'emprunt lorsque la fin de la méthode interdire() est exécutée. Si finInterdiction() était appelée lors de l'exécution de interdire(), cela ne serait pas le cas. On verrouille donc la ressource partagée (ici l'abonné), pour ces deux méthodes. De même il ne faudrait pas que interdire() soit appelée lors de d'exécution de finInterdiction(). Le timer d'interdiction pourrait être cancel dans la méthode finInterdiction() mais relancé dans interdire(), provoquant une exception.

CHAPITRE III

Classe « Livre »

Acte I – Méthodes synchronized

public void reserver(Abonne ab) :

On a ici un premier synchronized sur l'abonné, puisque l'on teste l'autorisation d'emprunt de l'abonné avant d'effectuer la réservation. Etant donné que nous testons ensuite l'état du livre avant de le réserver, un deuxième bloc synchronized, cette fois ci sur le livre, est imbriqué dans le premier. Il en va de même pour la méthode public void emprunter(Abonne ab).

CHAPITRE IV

Timer d'inactivité

La principale difficulté pour implémenter une solution permettant la déconnexion automatique d'un utilisateur pour cause d'inactivité était la nécessité de relancer le compte à rebours à chaque nouvelle saisie. Pour cela on utilise une instance de la classe TimerInactif (héritant de Timer) qui implémente deux méthodes :

- public void lancer(Socket s, long delai), qui prend en paramètre la socket à fermer une fois le délai d'inactivité, lui aussi passé en argument, dépassé. Cette méthode crée une instance de la classe TaskInactif (héritant de TimerTask) et plannifie cette tâche via l'utilisation de la méthode schedule() à laquelle on passe l'instance de TaskInactif.
- public void relancer(long delai), qui servira à relancer le Timer après chaque saisie de l'utilisateur. Supposant que le Timer pourrait être relancé pour une durée différente, on passe ici aussi le délai en paramètre. Cette méthode annule la tâche déjà planifiée et prévoit le déclenchement d'une nouvelle après le délai passé en paramètre.