Rapport de TP

SE - M1 Info GL2

Mullier Antoine
Sadok Romain

Sommaire

1 - Les entités du projet

1.1 - Le bowling

1.1.1 - Les pistes de bowling

Les pistes de bowling sont **des ressources** que possède le bowling. Une piste peut se voir être affecté à un groupe par le bowling elle apparaît donc comme occupée pour les autres clients du bowling ne faisant pas partie du groupe.

1.1.2 - Le moniteur Bowling

La classe *Bowling* prend le rôle du **moniteur** gérant l'accès aux pistes pour les différents groupes de clients présents dans le bowling. Le bowling peut donc **réserver une piste** pour un groupe et **libérer une piste**.

1.2 - Le guichet

1.2.1 - Partie 1 : un seul guichet

La classe *Guichet* permet d'assurer deux actions dans le processus d'action du client. D'une part, il permet de gérer l'**affectation d'un groupe à un client**. De plus, **le règlement des clients** doit se faire client par client, le guichet assure donc aussi la synchronisation des clients lors du règlement.

1.2.2 : Partie 2 : plusieurs guichets

L'énoncé de la partie 2 nous impose 3 guichets mais le problème de 3 guichets ou n guichets est globalement le même, on introduit donc dans le programme principal une variable ***NB_GUICHETS***. Tout d'abord il faut introduire un nouveau moniteur *GerantGuichet* qui va gérer l'accès à un ensemble de guichets par les clients.

Le gérant de guichet possède donc les opérations **attribuerGuichet** et **libererGuichet**. Nous évoquerons le problème de synchronisation liée à la gestion de plusieurs guichets dans le *paragraphe 2.2.2*.

1.3 - La salle de chaussures

1.3.1 - Les paires de chaussures

La classe paire de chaussures désigne simplement la ressource gérée par la classe *SalleChaussures*.

1.3.2 - Le moniteur SalleChaussures

La classe *SalleChaussures* prend le rôle de **moniteur** gérant l'accès aux paires de chaussures des différents groupes de clients. Ce moniteur gère donc **l'accès aux chaussures** mais aussi **l'attente du groupe** avant de sortir de la salle.

1.4 - Le Groupe de client

La classe *Groupe* interagit avec les clients avec le rôle de **moniteur** pour gérer les différents problèmes de synchronisation notamment pour les actions d'attente de groupe (quand un client doit attendre son groupe avant de faire des actions) le groupe permet de savoir dans quel état se trouvent les clients du groupe.

Le groupe intervient donc dans les problèmes de synchronisations suivants :

- L'ajout de client au groupe
- L'attente des clients du groupe en dehors de la salle des chaussures quand le groupe n'est pas complet
- L'attente des clients du groupe dans la salle des chaussures tant que tous les clients n'ont pas de chaussures.

1.5 - Le client

Il s'agit de la classe qui va simuler le comportement d'un client au bowling. La classe *Client* est donc un **Thread**. C'est le client qui effectue la plupart des traces d'exécutions. Il va donc exécuter les actions ordonnées suivantes :

- ***[Partie 2] :*** Chercher un guichet
- Demander à un guichet de lui attribuer un groupe
- ***[Partie 2] :*** Libérer le guichet pour les autres clients
- Aller dans la salle des chaussures
- Une fois qu'il a des chaussures lui et son groupe il va réserver une piste pour son groupe
- Il attend son groupe avant de commencer à jouer
- ***[Partie 2] :*** Il enlève son groupe de la file de priorité du bowling (ce n'est pas vraiment une action réelle de l'utilisateur mais elle est nécessaire pour le fonctionnement général du bowling)
- Une fois tout le monde sur la piste la partie commence
- ***[Partie 2] :*** Chercher un guichet pour payer
- Le client paye sa partie
- ***[Partie 2] :*** Libérer le guichet pour les autres clients
- Le client remet ses chaussures et sort du bowling.

2 - Les problèmes de synchronisations

Pour chaque moniteur on doit gérer des problèmes de synchronisation.
Les classes Client et Groupe sont suffisamment décrites dans la Partie 1 de ce rapport.

2.1 - Le bowling

2.1.1 - Partie 1 : la gestion des pistes

Le bowling gère l'accès aux pistes des groupes de client. La classe *Bowling* possède donc deux opérations **synchronized :** `reservePiste` et `liberePiste`.

Pour la méthode `reservePiste` le but est de chercher une piste disponible pour le groupe et si jamais aucune piste n'est disponible on fait attendre le processus avec l'opération `wait()`.

Avec la problématique de priorité de la partie 2 on regarde si le groupe du client est prioritaire. Dans le cas contraire on le fait attendre avec un `wait()`.

Lorsqu'un client est en attente cela correspond donc à deux attentes possibles : soit le client ne fait pas partie du groupe prioritaire et attend donc le tour de son groupe, soit le client attend qu'une piste se libère.

Pour la méthode `liberePiste(Piste p)` le but est de simplement libérer la piste p. En libérant une piste on doit aussi réveiller au moins un personne attendant sur le `wait()` de l'attente de piste. Comme on ne sait pas où le client qu'on va réveiller sera bloqué il faut tous les réveiller et re-tester les conditions de la méthode `reservePiste()`.

2.2.2 - Partie 2 : la gestion de l'ordre d'arrivée des groupes

La partie 2 ajoute une contrainte supplémentaire qui doit être gérée par le bowling. Il s'agit de **l'ordre d'arrivée des groupes** dans la salle des chaussures qui attendent pour réserver une piste pour leur groupe.

Nous gérons la priorité avec une **liste des groupes** dans laquelle nous introduisons les groupes selon leur ordre d'arrivée (en gérant les doublons). Une fois la liste établie nous savons que seul **le groupe en tête de liste** peut **réserver une piste**. Une fois la piste réservée pour le groupe il faut enlever le groupe en tête de liste pour laisser les autres groupes réserver une piste à leur tour.

2.2 - Le guichet

2.2.1 - Partie 1 : un seul guichet

Le guichet est en charge de deux actions principales : **l'attribution d'un groupe** pour le client et **le règlement des clients**.

On a donc deux actions qui sont gérées par deux méthodes de la classe *Guichet* : `attribuerGroupe(Client)` et `payerPartie()`.

La méthode `attribuerGroupe(Client)` va chercher un groupe pour le client en s'assurant bien que celui-ci ne soit pas complet. Conformément au sujet on prend un certain temps pour chercher un groupe disponible.

La méthode `payerPartie()` prend un certain temps d'exécution

Remarque : Le guichet ne peut traiter qu'un seul client à la fois toutes les méthodes de la classe *Guichet* sont donc **synchronized**.

2.2.2 - Partie 2 : plusieurs guichets disponibles

Dans la partie 2, le guichet doit également avoir un statut `estLibre` pour savoir si le guichet peut être pris par un client. L'accès à cet attribut est protégé par des **getter et setter** tout deux **synchronized**.

La gestion de plusieurs guichets en même temps requiert une attention particulière à la gestion de l'attribution des groupes et notamment au fait de rajouter des clients dans un groupe. En effet, il faut pouvoir rajouter un client en testant dans **une seule méthode synchronized** du groupe si le groupe est complet. Dans le cas contraire, on peut avoir un problèmes d'exclusion mutuelle des groupes pour les guichets.

Prenons un exemple :

> Si un groupe a une taille maximale de 3 membres et qu'il comporte déjà deux clients. Imaginons deux guichets souhaitant ajouter un membre dans ce groupe.
> Il faut donc que ce soit **le groupe et lui seul** qui gère le fait d'ajouter des clients et notifier le guichet s'il l'a effectivement ajouter ou pas.

Cela est donc gérer avec la fonction `ajoutClientDansGroupe()` de la classe Groupe qui retourne un booléen pour savoir si le client à été ajouté.

De plus, le guichet permet d'ajouter les groupes à la file de priorité du *Bowling* ce qui permet la bonne gestion de la synchronisation des clients pendant leur attente.

2.2.3 - Partie 2 : le gérant de guichet

En introduisant plusieurs guichets disponibles pour les clients il faut donc gérer l'accès à ces guichets par les clients. C'est l'objectif de la classe *GerantGuichet*. Il gère aussi la priorité des clients selon leur ordre d'arrivée à la manière de la gestion de l'ordre des groupes pour le bowling (paragraphe 2.1.2). Le gérant de guichet possède donc **deux méthodes** : `attribuerGuichet()` et `libererGuichet()`.

En ce qui concerne la gestion de la synchronisation la méthode `attribuerGuichet()` va faire attendre tous les clients qui, soit ne sont pas prioritaires, soit n'ont pas de guichet disponible.

La méthode `libererGuichet()` doit donc faire un `notifyAll()` après avoir libérer le guichet pour pouvoir espérer réveiller un client qui est prioritaire et cherche un guichet disponible.

2.3 - La salle des chaussures

2.3.1 - Partie 1 : Pas de gestion de priorité

La salle des chaussures doit gérer l'accès par les groupes de clients aux paires de chaussures. La classe *SalleChaussures* possède donc deux méthodes `donnerChaussures()` et `remettreUnePaire()`.

La méthode `donnerChaussures()` doit faire attendre les clients qui souhaitent des chaussures si leur groupe n'est pas encore complet. De plus, avant de sortir de salle des chaussures on s'assure que tous les membres du groupe ont des chaussures en faisant appel à la méthode `toutLeMondeADesChaussures()` de la classe *Groupe*.

2.3.2 - Partie 2 : Gestion de priorité

Avec les spécifications de la partie 2, lorsqu'un groupe n'est pas prioritaire il ne peut pas rentrer dans la salle des chaussures on fait donc **attendre le client non prioritaire** pour qu'un client du groupe prioritaire passe avant lui.

Nous prenons donc en compte cette modification dans la méthode `remettreUnePaire()` et `attenteAvantDeSortir()`[1] dans lesquelles on met un `notifyAll()` pour que les utilisateurs bloqués sur l'attente de chaussures aient une chance de se réveiller pour prendre une paire à leur tour.

[1] Méthode privée de la classe *SalleChaussures* qui est appelée dans `donnerChaussures()`