Thème 11

Objectifs

– Barrière de synchronisation

Exercices TD

Exercice 32 – Le traineau du Père Noël

Noël approche, il est temps pour les 9 rennes du Père Noël de rentrer de leurs vacances dans le Pacifique Sud... Comme toujours, ils ont attendu la dernière minute et le Père Noël est impatient. Dès qu'il en voit arriver un, il s'empresse de l'atteler au traineau. Les rennes sont polis : ils vont saluer le Père Noël pour lui signaler leur retour. Ils sont aussi dociles : ils attendent pour commencer à tirer le traineau que tous soient attelés.

Question 1

Comment garantir qu'un renne ne commence pas à tirer le traineau avant que tous soient attelés ?

Question 2

Proposez une implémentation de la classe `Renne` (un renne appelle la méthode `sayHello` de la classe `PereNoel` pour signaler son retour) et écrivez une classe permettant de tester l'exécution de ce système.

Le froid rend le Père Noël maladroit, il a du mal à atteler les rennes et cela prend du temps pour chacun. Les rennes peuvent donc se retrouver à faire la queue. Un des lutins propose au Père Noël d'exécuter les opérations suivantes :

```
import java.util.concurrent.locks.*;

public class PereNoel implements Runnable {
    private final int NB_TOTAL_RENNES;
    private int nbRennesAtteles = 0;
    private boolean busy = false;

    public PereNoel(int nb) {
        NB_TOTAL_RENNES = nb;
    }

    public synchronized void sayHello() throws InterruptedException {
        while (busy) {
            wait();
        }
        busy = true;
        notify();
    }

    private synchronized void attelerRenne() throws InterruptedException {
        while (! busy) {
            wait();
        }
        Thread.sleep(300); // Ca prend du temps d'attacher le bestiau !
        nbRennesAtteles++;
        System.out.printf("Encore un renne attele. ");
        System.out.printf("Il y en a maintenant %d\n", nbRennesAtteles);
        busy = false;
        notify();
    }

    public void run() {
        try {
```

```
            while (nbRennesAtteles != NB_TOTAL_RENNES) {
                attelerRenne();
            }
            Thread.sleep(100);      // Le Pere Noel a du mal a monter !
            System.out.println("Je monte dans le traineau !");
        }
        catch (InterruptedException e) {
            System.out.println("Pere Noel interrompu en plein travail !");
        }
    }

}
```

Question 3

Le Père Noël est fâché parce qu'il se trouve coincé (il y a 9 rennes à atteler et l'exécution produit l'affichage ci-dessous) :

```
Encore un renne attele. Il y en a maintenant 1
Encore un renne attele. Il y en a maintenant 2
```

Aidez le lutin et expliquez-lui pourquoi.

Question 4

Proposez une implémentation alternative des méthodes `sayHello` et `attelerRenne`, en utilisant d'autres mécanismes de synchronisation et en garantissant que le Père Noël attèle tous les rennes.

Question 5

Tous les rennes sont-ils bien attelés au moment du démarrage du traineau ? Modifiez les méthodes `sayHello` et `attelerRenne` pour régler le problème.

Le Père Noël a beaucoup mangé pendant sa période d'inactivité, il est donc un peu lourd et a du mal à monter dans le traineau.

Question 6

Modifiez le programme pour garantir que les rennes ne partent pas avant que le Père Noël se soit hissé dans le traineau.

Exercices TME

Lors de chaque TME, vous devrez créer un répertoire pour chaque exercice, y mettre les fichiers s'y rapportant et soumettre ce répertoire à la fin du TME.

Exercice 33 – Le réveillon au restaurant

Nous nous intéressons à l'organisation du réveillon dans un restaurant. Le restaurant dispose de `10` tables pouvant chacune accueillir 2 personnes. Les tables peuvent être déplacées de manière à accueillir des groupes. Par exemple, un groupe de 5 personnes occupera 3 tables.

Les clients sont organisés en groupes mais se comportent de manière indépendante : le thread principal crée des instances de la classe `GroupeClients` et la création d'un groupe induit la création de l'ensemble des clients qui le composent, qui sont tous exécutés par des threads indépendants.

Question 1

Proposez une implémentation de la classe `TestRestaurant` exécutée par le thread principal.

Question 2

Proposez une implémentation de la classe `Restaurant`. Le restaurant est caractérisé par son nombre de tables, et la

classe propose une méthode `reserver` qui renvoie un numéro de réservation ou **`null`** lorsqu'il n'y a pas suffisamment de tables disponibles pour accueillir le groupe.

Nous nous intéressons maintenant à la classe `GroupeClients`. Nous souhaitons associer un identifiant à chaque groupe de manière à pouvoir tracer l'exécution. De plus, la construction d'un groupe de clients doit créer et lancer les threads correspondant à chacun des clients. Nous souhaitons conserver une référence sur chacun des threads créés.

Question 3
Donnez les variables d'instance et le constructeur de la classe `GroupeClients`.

Les clients d'un groupe sont tous identiques, en particulier il n'y a pas un leader chargé de la réservation. Tous vont donc appeler une méthode `reserver` de la classe `GroupeClients` dont on définit le comportement comme suit :

si le numéro de table du groupe n'est pas affecté, la méthode appelle la méthode `reserver` de la classe `Restaurant`. Si celle-ci renvoie **`null`**, il n'y a pas suffisamment de place pour accueillir le groupe, la tentative de réservation est un échec. Le client modifie alors l'*interrupted status* de l'ensemble des clients de son groupe pour leur signaler cet échec. Un client dont l'*interrupted status* est positionné doit donc immédiatement interrompre sa tentative de réservation.

Si tout se passe bien, la méthode affecte un numéro de table au groupe.

Question 4
Proposez une implémentation de la méthode `reserver` de la classe `GroupeClients`.

Nous nous intéressons maintenant au comportement d'un client. Celui-ci essaie de faire une réservation pour son groupe. Si la réservation échoue, il se termine. Par contre, si le groupe obtient une table, le client va prendre un temps aléatoire pour se rendre au restaurant. Lorsque le dernier membre du groupe arrive à table, les clients peuvent alors appeler le maître d'hôtel pour passer la commande.

Question 5
Modifiez l'implémentation de la classe `GroupeClients` en conséquence et proposez une implémentation de la classe `Client`.