Spécification de la concurrence
Master 1 Informatique UPEC 2015/2016
TP 5 : Variation sur le dîner des philosophes

Le problème du diner des philosophes est un problème classique de partage de ressources en programmation concurrente. Il y a n philosophes qui se trouvent autour d'une table ronde. Chaque philosophe a devant lui une assiette de riz. Entre deux assiettes se trouve une baguette pour manger (il y a donc n baguettes). Un philosophe fait deux choses : penser et manger. Pour manger, il a besoin des deux baguettes qui sont à coté de son assiette. Chaque philosophe agit de la façon suivante : quand il a envie de manger il prend d'abord la baguette à sa gauche, *ensuite* la baguette à sa droite. Quand il termine de manger, il rend les deux baguettes et il recommence à penser.

Le but de ce TP est de modéliser en Java le problème du dîner des philosophes. Chaque philosophe sera implémenté par un thread.

Exercice 1: Avec Lock

Dans cette première implémentation on utilisera l'interface `Lock` du package `java.util.concurrent.locks`. Un objet d'une classe qui implémente l'interface est un verrou qu'on acquiert de façon atomique. Quand on possède un verrou, aucun autre thread peut l'acquérir.

Parmi les méthodes de l'interface on a

```
void lock();
//Acquires the lock.
boolean tryLock();
//Acquires the lock only if it is free at the time of invocation.
void unlock();
//Releases the lock
```

On modélisera les baguettes par des verrous.

1. Implémentez et testez le dîner de 5 philosophes.
2. Essayez d'obtenir un inter-blocage.
3. Proposez une solution pour résoudre ce problème d'inter-blocage.

Exercice 2: Avec `synchronized`

Ici on modélise les baguettes par une classe qui contient deux méthodes `synchronized` pour prendre et relâcher la baguette et un booléen spécifiant si la baguette est libre ou non. Pour prendre une baguette, un philosophe vérifie si elle est libre, sinon il se met en attente sur la file d'attente pour la baguette, grâce à la méthode `wait`. Quand un philosophe pose la baguette, il appelle `notify` pour réveiller le philosophe en attente.

1. Écrivez les classes décrites ci-dessus.
2. Testez votre programme avec 5 philosophes.
3. Essayez d'obtenir un deadlock en faisant "dormir" le processus caractérisant un philosophe à un certain moment.

Exercice 3: Avec les sémaphores

Ici on utilisera une autre notion, les sémaphores, implémentés dands la classe `Semaphore` du package Java `java.util.concurrent`.

Un sémaphore ressemble à un verrou, mais il peut donner donner accès à plusieurs threads à la fois. Un sémaphore est créé avec un nombre de permissions. De façon atomique on peut acquérir une permission, s'il y en a encore disponibles.

```
Semaphore(int permits);
// Creates a Semaphore with the given number of permits
void acquire();
// Acquires a permit from this semaphore, blocking until one is available.
void release();
// Releases a permit, returning it to the semaphore.
```

Dans cet exercice les philosophes mangent une raclette (avec les baguettes !). Pour manger un philosophe doit avoir les deux baguettes et avoir accès au grill électrique au milieu de la table. Au plus deux philosophes peuvent utiliser le grill en même temps.

1. Écrivez les classes nécessaires
2. Comme pour les questions précédentes, testez votre programme avec 5 philosophes et essayer d'obtenir un deadlock.

Exercice 4: *Encore producteur-consommateur*

Comme dans le précédant, on programmera un producteur et plusieurs consommateurs, de façon cette fois de garantir l'exclusion mutuelle, à l'aide des Lock. A un moment choisi au hasard, un producteur ajoute un entier choisi au hasard à une file. Chaque consommateur attend que la file ne soit pas vide, et ensuite il enlève l'entier de la tête de la file et il l'affiche, et il recommence du début.

Observez le comportement de votre programme. Est-ce que la famine peut se produire ?

Essayez de réaliser la même fonctionnalité sans Lock, mais avec `sychronized, wait, notify`.