TD 2 - Programmation concurrentielle

Vendredi 15 février 2013

Exercice 1 (Au bar...)

Trois clients se trouvent au bar. Il est tard, ils sont fatigués, et ils s'endorment sur le bar. Chacun se réveille après s'être brièvement assoupi pendant une durée aléatoire comprise entre 0 et 1000 millisecondes (que vous pouvez obtenir avec `Thread.sleep((int)(Math.random()*1000))`).

Au réveil, chaque client demande ce qu'il veut : un "Java" (beurk), un "Expresso" (miam) ou un "Cappucino" (miam miam). Et parce qu'il est si fatigué, il s'assoupit à nouveau pour une durée aléatoire entre 0 et 1000 millisecondes. Malheureusement, le serveur s'est également assoupi, et chaque client va abandonner le bar, profondément frustré, à la cinquième demande.

Le compte-rendu d'une soirée se présente de la manière suivante :

```
Java!
Cappuccino!
Java!!
Cappuccino!!
Espresso!
Cappuccino!!!
Java!!!
Cappuccino!!!!
Espresso!!
Cappuccino!!!!!
Java!!!!
Espresso!!!
Espresso!!!!
Java!!!!!
Espresso!!!!!
```

Ecrivez un programme qui modélise ces clients (fatigués et assoiffés) et le serveur (peu coopératif).

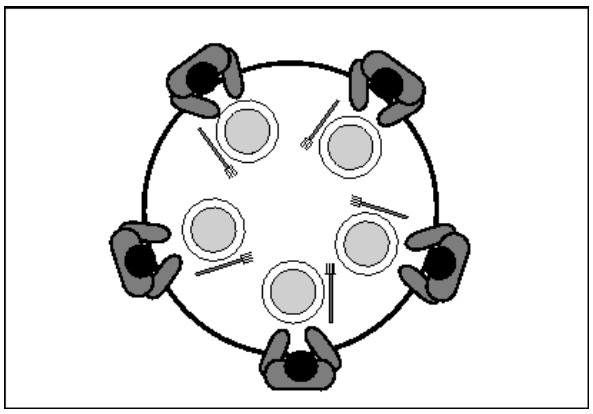
Exercice 2 (Une banque pas très fiable...)
Une banque très (trop) simple à été implémentée avec la classe suivante :

```
class SimpleBank {
  static int[] account = {30, 50, 100};

  public void transfer(int from, int to, int amount) {
    int namount;

    namount = account[from];
    namount -= amount;

    // ... difficult task of non-deterministic length

    account[from] = namount;

    namount = account[to];
    namount += amount;
    account[to] = namount;
  }

  public void balance() {
    for(int i = 0; i < account.length; i++)
      System.out.println("Account "+ i +": " + account[i]);
  }
}
```

Des employés qui travaillent dans la banque (une autre classe `Employee`). Le travail d'un employé consiste à transférer une somme d'argent `amount` du compte `from` vers le compte `to`.

1. Définissez ces deux classes, ainsi qu'une classe `SimpleBankDemo` (contenant la méthode `main`) qui crée la banque avec trois employés et qui organise un transfert cyclique de 20 EUR entre les différents comptes de la banque (chaque employé est responsable pour le transfert entre deux comptes particuliers).
2. Réalisez trois différentes exécutions de `SimpleBankDemo.main`, et observez les différentes balances résultant de cette implémentation.
3. Que constatez-vous ? Comment corriger ce problème ?

Exercice 3 (Les 5 philosophes...)
Dans la programmation concurrentielle, une fois le problème de l'accès synchronisé aux ressources (attributs, listes, etc.) résolu, d'autres problèmes surgissent. En voici un exemple classique :



Cinq philosophes alternativement pensent et mangent. Pour manger, un philosophe doit acquérir deux fourchettes : celui de sa gauche et de sa droite. Pour des raisons de précarité, il n'y a que 5 fourchettes, une entre chaque philosophe...

Proposez une solution qui synchronise l'accès aux fourchettes en implémentant une classe `Fork` (*fourchette* en français) et une classe `Philosopher` (*philosophe* en français). La classe `Fork` devra avoir une méthode permettant à une instance de `Philosopher` de *saisir* la fourchette. Une classe `PhilosopherDemo` mettra en œuvre ces classes dans sa méthode `main` en créant 5 instances de `Fork` et 5 instances de `Philosopher`.

Voici un exemple du résultat d'exécution :

```
There they go ...
Philosopher 0 starts thinking.
Philosopher 1 starts thinking.
Philosopher 2 starts thinking.
Philosopher 3 starts thinking.
Philosopher 4 starts thinking.
Philosopher 0 starts eating.
Fork 0 grabbed by philo 0.
Fork 1 grabbed by philo 0.
Philosopher 1 starts eating.
Philosopher 2 starts eating.
Fork 2 grabbed by philo 2.
Fork 3 grabbed by philo 2.
Philosopher 3 starts eating.
Fork 0 released by philo 0.
Fork 1 released by philo 0.
Philosopher 0 starts thinking.
```

Que constatez-vous ? Comment s'appelle ce phénomène et comment l'expliquez-vous ?