Université Paris Diderot – Paris 7 **Programmation réseau**
L3 Informatique **Anne 2010-2011, 2ème semestre**

TD n°1

Programmation concurrente en Java

Exercice 1 [Threads, attente] Écrivez une classe qui démarre deux threads. Le premier thread doit afficher dix fois `"Bonjour !"` en laissant passer une seconde entre chaque affichage. Le deuxième thread doit afficher cinq fois `"Salut !"`, en lassant passer deux secondes entre chaque affichage.

Indication : La méthode statique `sleep` *de la classe* `Thread` *permet de mettre un thread en attente.*

Exercice 2 [Variable partagée entre threads] On souhaite trouver toutes les occurrences d'un élément dans un tableau donné. Pour cela, si le tableau est long (plus qu'une constante TAILLE_MIN), on peut le couper en deux et effectuer la recherche indépendamment dans les deux moitiés, en confiant la seconde partie à un nouveau thread travaillant en parallèle. Écrire un programme effectuant la recherche de cette manière ; le programme devra afficher un message pour chaque occurrence trouvée, puis afficher le nombre total d'occurences l'élément dans le tableau. Pour cet exercice, vous pouvez utilisez la classe `CompteurConcurrent` que vous trouverez ci-dessous.

```
class CompteurConcurrent {
    private int cpt = 0;

    public int getValue() { return cpt; }

    public String toString() { return ""+cpt; }

    synchronized public void incremente() {
        cpt++;
    }

    synchronized public void incremente(int i) {
        cpt = cpt+i;
    }
}
```

Exercice 3 [Synchronisation entre threads] Il s'agit ici d'implémenter un système de producteurs/consommateurs. Ces producteurs/consommateurs communiquent

par un buffer partagé pouvant contenir une chaîne de caractères. Si le buffer est vide, un producteur a le droit de le remplir et il signale ensuite à tout le monde qu'il l'a rempli, au contraire si le buffer contient une chaîne de caractères (c'est-à-dire qu'il est plein), le producteur doit attendre qu'un consommateur consomme ce qu'il y a dans le buffer. Quant au consommateur, il essaie de lire ce qu'il y a dans le buffer, si il y arrive, il vide le buffer et il signale cela à tout le monde, sinon il attend qu'un producteur produise une donnée dans le buffer.

1. Dans un premier temps, les classes `Consommateur` et `Producteur` en utilisant la classe `ArrayBlockingQueue` pour le buffer et testez les avec par exemple deux producteurs et deux consommateurs.
2. Dans un deuxième temps, réécrivez les classes `Consommateur` et `Producteur` avec cette fois-ce une classe `Buffer` implémentée par vos soins.

   Indication : Pour cet exercice, on utilisera les méthodes `wait` *et* `notifyAll` *de la classe* `Thread`*. Lorsque la méthode* `wait` *est invoquée à partir d'une méthode* `synchronized`*, en même temps que l'exécution est suspendue, le verrou posé sur l'objet par lequel la méthode a été invoquée est relâché. Dès que la condition de réveil survient, le thread attend de pouvoir reprendre le verrou et continuer l'exécution. La méthode* `notify` *réveille un seul thread. Si plusieurs threads sont en attente, c'est celui qui a été suspendu le plus longtemps qui est réveillé. Lorque plusieurs threads sont en attente et qu'on veut tous les réveiller, il faut utiliser la méthode* `notifyAll`*.*