# Théorie et pratique de la concurrence – Master 1 Informatique
# TP 6 : Concurrence et Distribution

Dans ce Tp on fera quelques exercices de programmation distribuée sans l'utilisation de variables partagées. La communication entre threads se fera à l'aide de passage de message en utilisant le protocole TCP. Pour cela on utilisera les classe `Socket` et `ServerSocket` du package `java.net` Les messages seront de type `OutputStream()` et `InputStream()`.

**Exercice** 1:

Programmez une classe `ConcurrentBuffer` avec un champ de type `LinkedList<String>` et deux méthodes `put` et `get` qui agissent en exclusion mutuelle :

- La méthode `put` ajoute une chaîne de caractères à la liste et ensuite réveille les threads en attente sur cette liste.
- La méthode `get`, si la liste est vide, met en attente le thread appelant, sinon elle renvoie la première chaîne de caractères de la liste, et la retire de la liste.

**Exercice** 2:

Implémentez en Java une version distribuée de l'algorithme producteur/consommateur . Pour ce faire, un serveur écoute sur deux ports. Sur un port, le serveur attend les connexions d'un client producteur et sur l'autre port les connexions d'un client consommateur. Un client producteur envoie au serveur une chaîne de caractères qu'il aura reçu par l'entrée standard. Le serveur stocke alors cette chaîne de caractères dans un objet de type `ConcurrentBuffer` Un client consommateur attend que le serveur lui envoie des chaînes de caractères qui ont été placés dans le buffer.

Attention : le serveur attende des connexion sur deux ports différents.

**Exercice** 3:

Modifiez le programme de l'exercice précédent pour avoir plusieurs producteurs et plusieurs consommateurs

**Exercice** 4:

Implémentez une version distribuée du dîner des philosophes. Les baguettes ne sont pas représentées explicitement. Quand un philosophe veut prendre une baguette, demande la permission à son voisin. Si celui ci est en train de manger, le premier philosophe attend que son voisin lui envoie le message en disant qu'il a terminé.