PR6 – Programmation réseaux
TP n° 5 : Producteurs-Consommateurs en réseaux

Exercice 1 : programmation d'un buffer concurrent

Le but de cet exercice est de programmer une liste chaînée acceptant des accès concurrents en exclusion mutuelle. La fonction qui permet de retirer un élément utilisera une attente passive.
Écrire une classe `BufferConcurrent` avec un champ de type `LinkedList<String>` représentant une liste de chaînes de caractères et deux méthodes `ajouter` et `retirer`.

- La méthode `ajouter` prend le verrou sur la liste, ajoute une chaîne de catactères à la liste, puis réveille les threads en attente d'accès à cette liste. Lorqu'elle termine elle libère le verrou.
- La méthode `retirer` prend le verrou sur la liste et si la liste est vide, elle met en attente le thread appelant, sinon elle renvoie la première chaîne de caractères présente dans la liste qu'elle retire également. Lorsqu'elle termine elle libère le verrou et réveille les threads en attente d'accès à cette liste.

Indication : Pensez à utiliser le mot clef `synchronized` ainsi que les instructions `wait`, `notify` et `notifyAll`.

Exercice 2 : un producteur et un consommateur

Le but de cet exercice est de programmer une application réseau avec un client producteur et un client consommateur. Pour ce faire, le 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. L'idée est la suivante : un client producteur envoie au serveur une chaîne de caractères qu'il aura reçu par l'entrée standard (le clavier), le serveur stocke alors cette chaîne de caractères dans un buffer (par exemple un objet de type `BufferConcurrent`). Quant au client consommateur il attend que le serveur lui envoie des chaînes de caractères qui ont été placés dans le buffer. Il faudra donc programmer deux types de client `ClientCons` et `ClientProd` et un serveur `Serv`.
Indication : Avant de commencer, réfléchissez comment faire pour que le serveur attende des connexions sur deux ports différents, pourquoi pas en faisant un serveur multi-threadé ? Que doivent alors partager les threads du serveur ?

Exercice 3 : plusieurs producteurs et plusieurs consommateurs

Modifier le programme de l'exercice précédent afin de gérer plusieurs producteurs et plusieurs consommateurs (réfléchir à comment faire en multi-threadé et à quel moment créer les différents threads et quels sont les objets partagés par ces threads).