Exercice 1 : Tonneaux

Nous souhaitons réaliser un programme dans lequel on représente des tonneaux qui se vident les uns dans les autres. Tous les tonneaux ont la même capacité maximale. Un tonneau a un bouchon qui peut être enlevé, et alors le tonneau se videra à un certain débit, le débit pouvant être différent pour chaque tonneau. Les tonneaux sont disposés de telle façon que certains tonneaux déversent leur contenu dans un seul autre tonneau.

1. Créer la classe `Tonneau` pour représenter un tonneau. Un tonneau aura un nom, un volume initial, un débit de fuite et il peut connaître le tonneau dans lequel sa fuite se déverse. Tous les tonneaux ont le même volume maximum.
2. Créer deux méthodes publiques `ajoutVolume` et `retireVolume` qui respectivement ajoute et retire un volume donné au tonneau. Après chaque opération, vous afficherez le nom du tonneau, le fait que c'est un ajout ou une fuite, puis la quantité actuelle de liquide dans le tonneau. Vous ferez attention à ne pas avoir un volume négatif, et lorsque le volume dépasse la capacité maximale, vous afficherez un message comme quoi le tonneau déborde, et vous rectifierez le niveau de liquide. Les deux méthodes retourneront le volume ajouté ou retiré.
3. Faire en sorte que lorsqu'on enlève le bouchon d'un tonneau :
   a) Le tonneau se vide.
   b) S'il a un tonneau en-dessous, il remplit le tonneau de la quantité perdue.

   Ces opérations doivent se répétées toutes les secondes jusqu'à ce que le tonneau soit totalement vide.
4. En utilisant une capacité maximale de 12L, écrire un programme principal avec deux tonneaux. Le premier tonneau a 5L initialement et un débit de 1L/s. Le second tonneau a 12L initialement, un débit de 4L/s et se déverse dans le premier. Exécuter le programme en retirant le bouchon du second puis du premier.
5. Modifier le programme précédent pour prendre en compte les éventuels accès concurrents.

Exercice 2 : Tri

Nous souhaitons réaliser l'implémentation du Tri Rapide. Le but du tri rapide est le suivant :

- On choisit un pivot (arbitrairement, on choisira ici la dernière valeur du tableau)
- On positionne toutes les valeurs inférieures au pivot en début de tableau
- On positionne le pivot à sa place

L'algorithme peut donc se concevoir de cette façon :

```
partitionner(tableau T, entier premier, entier dernier) : entier
    j := premier
    pour i de premier à dernier - 1
        si T[i] <= T[dernier] alors
            échanger T[i] et T[j]
            j := j + 1
    échanger T[dernier] et T[j]
    renvoyer j

tri_rapide(tableau T, entier premier, entier dernier)
    début
        si premier < dernier alors
            pivot := partitionner(T, premier, dernier)
            tri_rapide(T, premier, pivot-1)
            tri_rapide(T, pivot+1, dernier)
        fin si
    fin
```

La fonction du tri rapide peut donc s'effectuer en parallèle pour chaque portion du tableau. Nous utiliserons donc des threads pour cela.

Implémenter la solution et la tester sur un tableau de taille raisonnable. On vérifiera le tri par l'affichage des valeurs avant et après tri.