TP3
Votre propre sémaphore

Ce TP est à résoudre en utilisant exclusivement les fonctions de moniteur des objets Java (`wait`, `notify`, `notifyAll`).

On considère l'interface générique suivante que doit fournir un sémaphore.

```
public interface Semaphore {
     public void acquire(int nbJetons)
          throws InterruptedException;
     public void release(int nbJetons);
}
```

Implémenter en Java quatre classes correspondant aux quatre versions de sémaphore suivantes:

1. Dans le cas le plus simple, le *Sémaphore* ne doit pas gérer lui-même la liste des tâches en attente. La fonction `release` réveille une tâche au hasard, en utilisant `notify`.
   Cette version peut-elle fonctionner correctement dans le cas de `acquire` faits avec `nbJetons > 1` ? Pourquoi ?
2. Modifier la première version pour utiliser `notifyAll` au lieu de `notify`, tout en préservant le fonctionnement des opérations `acquire` et `release`. Quelle est la différence ?
3. Donner une implémentation du *Sémaphore* qui utilise une liste FIFO (circulaire) pour les tâches suspendues, en utilisant `notifyAll` pour le réveil. Expliquer pourquoi c'est inefficace.
4. Enfin, donner une implémentation efficace pour le *Sémaphore* FIFO, qui ne réveille que la tâche en tête de liste, en utilisant la méthode `Thread.interrupt`.

Toutes les versions doivent aussi fournir un constructeur parametrés permettant d'initialiser le nombre de jetons du sémaphore.

Agrémenter le code des méthodes d'affichages permettant de tester leur fonctionnement et leurs propriétés (e.g. FIFO, affichage des réveils et des entrées en attente, etc.).

***Validation** : Concevoir un petit programme de test pour valider chacune des versions. Une fois que l'implantation est considérée "stable" pour chacune de ces versions, tester le fonctionnement en l'utilisant pour résoudre **l'exercice 1 du TP2**.*