Thème 10 - TD

Objectifs

— Classe `Semaphore`

Exercices

Exercice 15 – Le problème de la piscine avec des sémaphores

Nous reprenons le problème de la piscine dans un contexte où l'on ne souhaite pas identifier les ressources utilisées (les paniers et les cabines ne sont plus numérotés) mais simplement examiner comment la disponibilité de ces ressources permet de contrôler l'accès au bassin.

Question 1

Proposez une nouvelle implémentation de la classe `Piscine` à base de sémaphores, dans laquelle les accès au bassin restent contraints par la disponibilité d'un panier et d'une cabine.

Question 2

Proposez une implémentation de la classe `Nageur`.

Exercice 16 – Les babouins du parc Kruger

Ce problème est un problème classique de synchronisations entre classes d'utilisateurs. On en trouve différents habillages dans beaucoup de livres de système. Celui-ci est extrait de *The Little Book of Semaphores, A.B. Downey*.

Il y a dans le parc national Kruger en Afrique du Sud un profond canyon dont la rive nord et la rive sud sont reliées par une corde. Les babouins peuvent traverser en avançant une main après l'autre sur la corde, mais si deux babouins se retrouvent face à face, ils se battent, tombent et meurent.

En supposant que l'on puisse apprendre aux babouins à utiliser des primitives de synchronisation, nous voulons construire sous différentes hypothèses des programmes qui doivent garantir que tous les babouins survivront.

Nous utiliserons pour représenter la position des babouins le type énuméré suivant :

```
// An enum constant may be followed by arguments, which are passed to the constructor of the enum when
// the constant is created during class initialization (Java Language Specification, chap. 8.9).

public enum Position {
   NORD(0), SUD(1);

   private Position(int index) {
      this.index = index;
   }

   private final int index;
   public int index() {return index;};
}
```

On note qu'on peut ajouter aux constantes du type `enum` des arguments constants, ici un champ `index` qui vaut `0` pour la constante `NORD` et `1` pour la constante `SUD`. On pourra ainsi lier facilement une position à un indice de tableau.

Quelle que soit la stratégie adoptée pour la traversée du canyon, les babouins exécutent la suite d'opérations ci-dessous :

```
public void run() {
   try {
      laCorde.acceder(position);
      System.out.println(this.toString() + " a pris la corde.");
      traverser();
      System.out.println(this.toString() + " est arrive.");
      laCorde.lacher(position);
   }
   catch (InterruptedException e) {
      System.out.println("Pb babouin !");
   }
}
```

où `laCorde` est une référence sur un objet d'une classe `Corde` et `traverser()` une méthode interne à la classe `Babouin` simulant par un délai aléatoire la traversée du canyon. La position passée en paramètre des méthodes correspond à la rive d'où part (ou est parti) le babouin.

Question 1

Si on suppose que la corde n'est pas très solide ou les babouins pas très intelligents (ils ne savent pas compter), on choisit de mettre en place une stratégie où un seul babouin peut à un instant donner se trouver sur la corde.

Proposez une implémentation de la classe `Corde` d'abord en réalisant les synchronisations avec un (des) objet(s) de la classe `Lock`, puis avec un (des) objet(s) de la classe `Semaphore`. Pourrait-on utiliser des méthodes synchronisées pour réaliser cette stratégie ?

Il semble que la corde soit en fait suffisamment solide pour qu'on n'ait pas à limiter le nombre de babouins qui y sont accrochés. Nous souhaitons donc modifier la stratégie précédente pour permettre une meilleure utilisation de la corde, tout en préservant la sécurité des babouins.

Question 2

A quelle condition un babouin peut-il s'engager sur la corde ? Comment peut-on évaluer cette condition ? Proposez une implémentation des méthodes `acceder` et `lacher` en utilisant des `Lock`.

La construction directe d'une solution à base de sémaphores comporte de multiples pièges (nous allons y revenir...). Mais disposer d'une telle solution est important dans la mesure où ce type de mécanisme de synchronisation est très répandu (la norme Posix en propose par exemple une implémentation). Une possibilité pour aboutir à la solution est de réécrire avec des sémaphores les opérations exécutées sur un moniteur.

Question 3

Proposez une implémentation des méthodes `lock()` et `unlock()` avec des sémaphores.

Nous nous intéressons maintenant au blocage / déblocage sur une condition. Nous rappelons quelques éléments sur la sémantique des moniteurs :

- lorsqu'aucun processus n'est en attente sur la condition, le `signal()` est perdu ;
- les moniteurs Java utilisent une sémantique de type *signal and continue* : le processus qui exécute un signal sur une condition ne libère pas (forcément) le processeur au profit du processus réveillé.

Question 4

Proposez une implémentation des méthodes `signal()` et `await()` sur une instance `cond` de `Condition`.

Nous nous intéressons maintenant à une tentative d'implémentation directe avec des sémaphores. La stratégie est la suivante : un sémaphore binaire à chaque extrémité de la corde constitue la "porte d'entrée" ; un babouin qui trouve la corde vide verrouille la porte en sens inverse. Il passe ensuite sa porte en prenant soin de la laisser ouverte pour ceux qui arrivent derrière lui. On a alors le code suivant :

```
public void acceder(Position p) throws InterruptedException {
   mutex.acquire();
   if (cptIn == 0) {
      semAcces[(p.index()+1)%2].acquire();
   }
   semAcces[p.index()].acquire();
   cptIn++;
   mutex.release();
   semAcces[p.index()].release();
}
```

Question 5

Expliquez pourquoi cette solution ne fonctionne pas.

Question 6

On tente maintenant une solution avec un compteur et un sémaphore à chaque extrémité :

```
public void acceder(Position p) throws InterruptedException {
   mutex.acquire();
   if (cpt[0] == 0 && cpt[1] == 0) {
      semAcces[(p.index()+1)%2].acquire();
   }
   cpt[p.index()]++;
   mutex.release();
   semAcces[p.index()].acquire();
   semAcces[p.index()].release();
}
```

Expliquez pourquoi ça ne fonctionne toujours pas...