Thème 8 - TD

Objectifs

— Vivacité, famine, interblocage.
— Interface `Lock : tryLock()`

Exercices

Exercice 11 – Rappels de cours

Question 1
Rappelez les quatre conditions qui caractérisent un interblocage.

Question 2
Qu'est-ce qu'une famine ?

Exercice 12 – Dîner des philosophes

Nous souhaitons programmer un problème classique de programmation concurrente, "le dîner des philosophes", dans lequel `N` philosophes sont assis autour d'une table ronde. Il y a `N` baguettes sur la table, chaque baguette est posée entre deux philosophes. Pour un dîner de 5 philosophes, la configuration est la suivante :

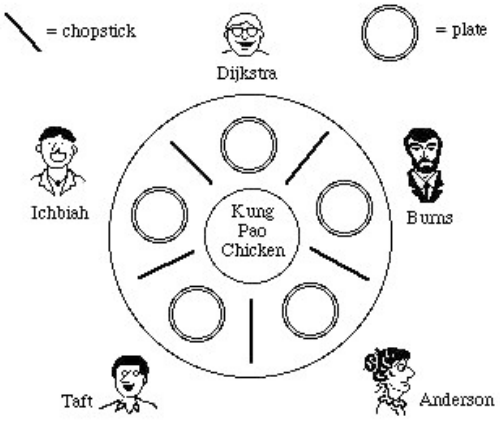

FIGURE 1 – Le dîner des 5 philosophes

Lorsqu'il est fatigué de penser, un philosophe doit manger pour reprendre des forces. Pour cela, il faut qu'il s'empare de la baguette qui se trouve à sa gauche et de celle qui se trouve à sa droite. Il peut manger dès qu'il y parvient. Une fois rassasié, le philosophe repose ses baguettes et repart dans une phase de réflexion.

Nous décidons dans un premier temps de programmer le partage des baguettes au moyen d'un objet synchronisé. Si une baguette n'est pas disponible, la méthode qui permet de prendre la baguette doit mettre le `thread` appelant en attente.

Question 1
Programmez la classe `ChopStick` qui décrit le comportement d'une baguette.

Question 2
Programmez la classe `Philo` qui décrit le comportement d'un philosophe. Vous imposerez un délai d'attente aléatoire entre la prise de baguette à gauche et la prise de baguette à droite.

Question 3
Programmez une classe `TestPhilo` qui permet de tester les deux classes que vous venez de programmer. Quel problème peut-on rencontrer à l'exécution de ce programme ?

Nous souhaitons modifier la manière dont un philosophe s'empare de ses baguettes : s'il n'arrive pas à entrer en possession des deux baguettes, il repose celle qu'il a éventuellement réussi à prendre, se donne un délai d'attente et refait une tentative.

Question 4
Quel est le mécanisme de synchronisation adapté pour implémenter ce comportement ?

Question 5
Donnez une nouvelle version des classes `ChopStick` et `Philo`.

Exercice 13 – Prise de ressources multiples : le problème de la piscine

Le problème de la piscine est un schéma classique d'accès à des ressources multiples.

La piscine dispose d'un certain nombre de cabines et de paniers. Avant d'accéder au bassin, un baigneur :
- entre dans une cabine ;
- demande au personnel du vestiaire un panier dans lequel il dépose ses affaires ;
- confie ce panier, rempli, au personnel et quitte la cabine pour aller se baigner.

Lorsqu'il est fatigué de nager, le baigneur doit à nouveau trouver une cabine. Il demande alors son panier, qu'il vide avant de le rendre. Il peut ensuite sortir de la cabine et quitter la piscine.

Question 1
Nous considérons une piscine disposant de 3 cabines et 5 paniers. Que peut-il se passer si 10 nageurs demandent à accéder au bassin ?

Question 2
Proposez une solution pour éviter cet interblocage sans modifier le nombre de nageurs ni le nombre de ressources (paniers et cabines).

Question 3
Quel problème peut se poser si le nombre de nageurs n'est pas limité *a priori* ?

Nous nous intéressons maintenant à l'implémentation de ce système en Java.

Question 4
Proposez le code d'une classe `Ressource` à partir de laquelle vous construirez deux sous-classes `Panier` et `Cabine`. Pour pouvoir suivre l'évolution du système, vous affecterez un numéro aux cabines et aux paniers.

Question 5
De quel schéma de synchronisation classique se rapproche la gestion des ressources ? Proposez une structure de données adaptée pour la gestion d'un ensemble de ressources. Comment garantir la cohérence de cette structure ?

Question 6
Proposez une implémentation de la classe `Piscine` permettant la gestion des ressources qui lui sont associées.

Question 7
On revient sur le cas où le nombre de nageurs n'est pas limité *a priori*. Est-il judicieux de créer un thread par nageur ? De quelle solution alternative dispose-t-on ?