Travaux Dirigés n°8

Java Avancé

—M1—

Threads - Attentes

Wait, notify, etc

`https://classroom.github.com/a/UHZ_RtwL`

Dans un programme monothread, une pré-condition non vérifiée (file vide pour un retrait par exemple) ne deviendra jamais vraie et on peut faire échouer l'appel. En revanche, dans un programme multi-threadé, la pré-condition peut évoluer grâce à l'action des autres threads (par exemple, une file vide peut avoir été remplie par un autre thread). Plutôt que de faire échouer la méthode, on préfère faire *attendre* le thread, et de manière passive afin d'éviter une charge d'activité inutile.

▸ Exercice 1. Producteurs et consommateurs

On va définir des threads produisant des gâteaux et des threads consommateurs de gâteaux. Une file d'attente sera utilisée pour éviter le flux tendu entre producteurs et consommateurs.

1. Créer une classe `Cake` prenant un type de gateau en paramètre (un `String`).
2. Créer une classe `Restaurant` utilisant une `ArrayDeque`.
3. Écrire une méthode retournant un `Runnable` et prenant un type de gâteau et un temps de préparation (entier) en argument. Le run de ce runnable effectue infiniment la création d'un gâteau, l'ajoute à la file (via `add`) et puis s'endort pendant le temps de préparation (`sleep`).
4. Écrire une autre méthode retournant un `Runnable` et prenant en paramètre le nom d'un mangeur, et un temps pris pour manger un gateau. Le run de ce runnable effectue infiniment la prise d'un gâteau dans la file (via `poll`) puis s'endort pendant le temps pris pour manger (`sleep`).
5. Tester votre code avec 1 producteur et 5 consommateurs, avec le même temps de préparation que pour manger. Pour cela, faites des nouveau `Thread`s prenant en argument du constructeur vos `Runnable`s, puis lancez-les. Que se passe-t-il ?
6. Et s'il y a plus de producteurs que de consommateurs ?
7. Avez-vous pensé à gérer la concurrence ? (`ArrayDeque` n'est pas Thread safe par exemple...)

▸ **Exercice 2. FIFO synchro**

On souhaite écrire une classe d'une FIFO bloquante.

1. Écrire un constructeur prenant en paramètre la taille maximale de la FIFO et initialisant une `ArrayDeque` (qui sera le support de votre file).
2. Écrire une méthode `add(E)` qui ajoute un objet à la fin de la file et une méthode `remove()` qui en retire un. Si la file est pleine ou vide, la méthode doit bloquer. On utilisera des blocs `synchronized` et les méthodes `wait` et `notify` (ou `notifyAll ?`) pour l'attente. Lire la doc...
3. Est-ce que cela poserai un problème de faire le test de file vide/pleine dans une méthode à part ?
4. Tester votre file en l'utilisant avec les producteurs/consommateurs de l'exercice précédent. Attention, votre file est thread-safe : il n'y a plus besoin de gérer la concurrence au sein des producteurs et consommateurs !

▸ **Exercice 3. FIFO lockée**

On veut utiliser des `Lock` plutôt que des blocs synchronized.

1. Chercher comment on crée une `Condition` à partir d'un `Lock`.
2. Quel est l'avantage d'utiliser des `Conditions` (avec les méthodes `await` et `signal`) que des `wait/notify` ?
3. Modifier votre code afin d'utiliser des `Locks` et des `Conditions`.
4. Tester avec les producteurs / consommateurs.