TP 3: Java Threads Concurents

« Une conversation enseignant-étudiants »

Master 2 STIC

Module : Les applications réparties

Département Informatique

Université de Guelma

2020/2021

Exercice 1 : Des threads indépendants

Le but est de créer un objet Enseignant contenant une méthode question() et un objet Etudiant contenant une méthode reponse(), tous deux héritant de la classe Thread.

Le but est simple : au lancement du code, un enseignant pose une question et l'étudiant attend la fin de la question. Une fois celle-ci posée, c'est l'enseignant qui attend la réponse et ainsi de suite…

Java propose deux mécanismes pour poser des verrous :

- les moniteurs qui s'utilisent avec le mot clé synchronised
- les verrous qui sont définis dans l'interface

java.util.concurrent.locks….

Les classes Lock et Condition

- Java 5 propose de nouvelles fonctionnalités, regroupées dans le package java .util.concurrent.locks, pour gérer les accès concurrents grâce à des verrous.
- Ces verrous reposent sur l'utilisation d'objets : il est donc nécessaire d'en créer une instance et d'invoquer des méthodes, ce qui rend le code à produire plus verbeux par rapport au mot clé synchronized qui est intégré dans le langage.
- L'interface java.util.concurrent.locks.Lock définit les fonctionnalités d'un mécanisme de verrous permettant de contrôler l'accès par plusieurs threads à une portion de code.

La classe « lock »

Méthode	**Rôle**
void lock()	Obtenir le verrou : attente indéfinie si celui-ci est déjà pris
void lockInterruptibly()	Obtenir le verrou : attente jusqu'à son obtention ou si le thread courant est interrompu
Condition newCondition()	Obtenir une instance de type Condition associée à l'instance
boolean tryLock()	Obtenir le verrou immédiatement : pas d'attente. Elle renvoie un booléen qui indique si le verrou est obtenu
boolean tryLock(long time, TimeUnit unit)	Obtenir le verrou : attente maximale pour la durée précisée en paramètre ou si le thread courant est interrompu
void unlock()	Libérer le verrou

La classe « Lock »

Exemple (code Java 5.0) :

```
Lock verrou = new ReentrantLock();
verrou.lock();
try {
// section critique protegee par le verrou
} finally {
verrou.unlock(); }
```

La classe « Condition »

- Une instance de type **Condition** permet de mettre en attente un thread jusqu'à ce qu'il reçoive une notification lorsque la condition est remplie.

Méthode	**Rôle**
void await()	Le thread courant suspend son exécution et attend de recevoir un signal ou d'être interrompu
boolean await(long time, TimeUnit unit)	Le thread courant suspend son exécution et attend de recevoir un signal ou d'être interrompu ou que le timeout précisé en paramètre soit atteint
long awaitNanos(long nanosTimeout)	Le thread courant suspend son exécution et attend de recevoir un signal ou d'être interrompu ou que le timeout précisé en paramètre soit atteint
void awaitUninterruptibly()	Le thread courant suspend son exécution et attend de recevoir un signal : il ne peut pas être interrompu
boolean awaitUntil(Date deadline)	Le thread courant suspend son exécution et attend de recevoir un signal ou d'être interrompu ou que la date/heure limite précisée en paramètre soit atteinte
void signal()	Envoyer un signal à un thread qui est en attente d'un signal de cette condition
void signalAll()	Envoyer un signal à tous les threads qui sont en attentes d'un signal de cette condition

Référence

- Jean-Michel DOUDOUX

Chapitre 33 (**La gestion des accès concurrents)**

https://jmdoudoux.developpez.com/cours/developpons/java/chap-acces_concurrents.php#acces_concurrents-3