

JAVA Multithread

Problèmes du Multithread

- Accès simultanés
- Mise à jour

Problèmes du Multithread

Accès simultanés

- Deux thread veulent consommer une même ressource en même temps.
- Ex:

```
while(myIterator.hasNext())  
{  
    Task currentTask = myIterator.next();  
    myIterator.remove();  
}
```

Problèmes du Multithread

Accès simultanés

- Après que le thread 1 ait passé la condition du while (pour le dernier élément de la collection), mais avant qu'il l'est enlevé, un autre thread passe aussi la condition du while.
- Lorsque le dernier thread fera le .next(), l'autre thread aura déjà enlevé l'élément, la collection est vide et le programme plante.

Problèmes du Multithread

Accès simultanés

- Solution: mot-clé synchronized.
- Une fonction synchronized ne peut être appelée que par un thread à la fois, les autres devront attendre.

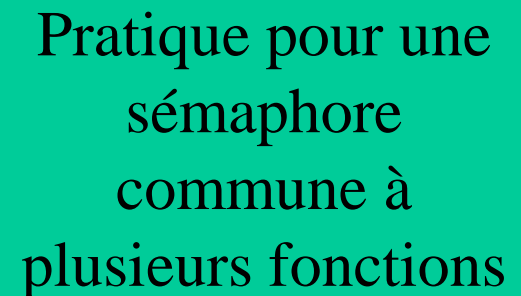
```
public synchronized void myFct()
```

Problèmes du Multithread

Accès simultanés

- Solution: mot-clé synchronized.
- Un bloc de code synchronized(Object o) indique qu'un thread doit obtenir un lock sur o avant de pouvoir exécuter le code. Chaque objet ne peut avoir qu'un seul lock.

```
public void add(E element) {  
    synchronized (this)  
    {...}}}
```



Pratique pour une
sémaphore
commune à
plusieurs fonctions

Problèmes du Multithread

Mise à jour

- Chaque thread à sa propre cache
- Une modification fait par un thread sur une variable partagée n'est pas toujours visible immédiatement par les autres thread.

Problèmes du Multithread

Mise à jour

- Solution: mot-clé volatile sur les attributs.
- Les threads ne peuvent donc pas utiliser leur cache pour cet attribut.
- Plus difficile pour le compilateur d'optimiser le code.

Problèmes du Multithread

Mise à jour

- Solution: mot-clé volatile sur les attributs.
- Volatile permet aussi d'assurer que la lecture et l'écriture sont des opérations atomiques sur l'attribut en question.

Multithread

- L'interface Runnable nous permet de définir une tâche à accomplir par un thread [méthode `.run()` à coder]
- La classe Thread nous permet de partir de nouveaux thread
 - `[new Thread(Runnable).start()]`.

Multithread

- Pour une bonne gestion des threads, les concepts de ThreadPool et Executor sont préférables à la création manuelle de threads.
- Pour plus de flexibilités, les concepts de Callable et Future peuvent être utilisé conjointement avec Executor.

Lecture

<http://www.vogella.com/articles/JavaConcurrency/article.html>