Java Avancé

Thread

Emerite NEOU

Thread

Thread

Thread

Thread

Probleme 1

On veut executer plusieurs "code" en même temps

- Du code identique (serveur web)
- Du code différent (jeu : gérer l'affichage et l'IA)

Probleme 2

On veut des programme qui tourne plus vite

Probleme 2

On veut des programme qui tourne plus vite
mais les processeur sont limité physiquement (Free lunch is over).

Probleme 2

On veut des programme qui tourne plus vite
mais les processeur sont limité physiquement (Free lunch is over).
Heureusement, plus de coeurs.

Solution

THREAD

Solution

THREAD

```
Runnable toDo = ... ;


Thread thread = new Thread(toDo) ;
thread.start() ;

// OUs

Executor executor = anExecutor ;
executor.execute(toDo) ;
```

Comment ca marche

- Plusieurs threads peuvent tourner en même temps
- Chaque thread à sa propre pile (stack) avec stockage de valeurs.
- Une mémoire partagée entre les threads (tas ou heap) pour échange d'information entre threads.
- Un ordonnanceur pour partager le temps de calcul sur le processeur.

C'est bien mais

- Changer les algo.
- Le programmeur n'a pas la main sur l'archi du CPU.
- Gérer la mémoire partager entre les threads
- Le scheduler peut arrêter une thread quand et où il veut (au milieu d'un test, au milieu d'un changement de variable).

Scheduler

- Le scheduler fait comme il veut...
  - On ne peut **pas contrôler l'ordre** d'exécution entre les threads.
  - Cet **ordre change** d'une exécution à une autre... AIE AIE AIE.
  - Est équitable : quand le temps va vers l'infini, chaque thread aura eu autant de temps de calcul.

Terminer

- Un thread est terminé lorsque le code de son run() est terminé.
- Apres un thread.join().
- Pas possible de tuer un Thread. On peut juste lui demander.

ExecutorService

- Sépare la création de la thread de l'éxecution.
- Un ExecutorService utilise des threads.

ExecutorService

- Sépare la création de la thread de l'éxecution.
- Un ExecutorService utilise des threads. Mais il sait les réutiliser.

ExecutorService

- Sépare la création de la thread de l'éxecution.
- Un ExecutorService utilise des threads. Mais il sait les réutiliser.
- Permet de gérer le nombre max de threads, attendre la fin de tous les threads, valeurs de renvoie etc...

ExecutorService

```
ExecutorService executionContext = Executors.newFixedThreadPool(4)  ;

Callable toDoWithReturn = ... ;

Future< ?> res = executionContext.submit(executionContext) ;
```