NFP – 121

Java et les Threads

Présentation : Thierry Escalarasse

Mai 2007

Plan du cour

- Présentation de la notion de Threads
- La classe Thread
- L’interface Runnable
- Les états d’un thread
- La Synchronisation

Introduction à la notion de Thread

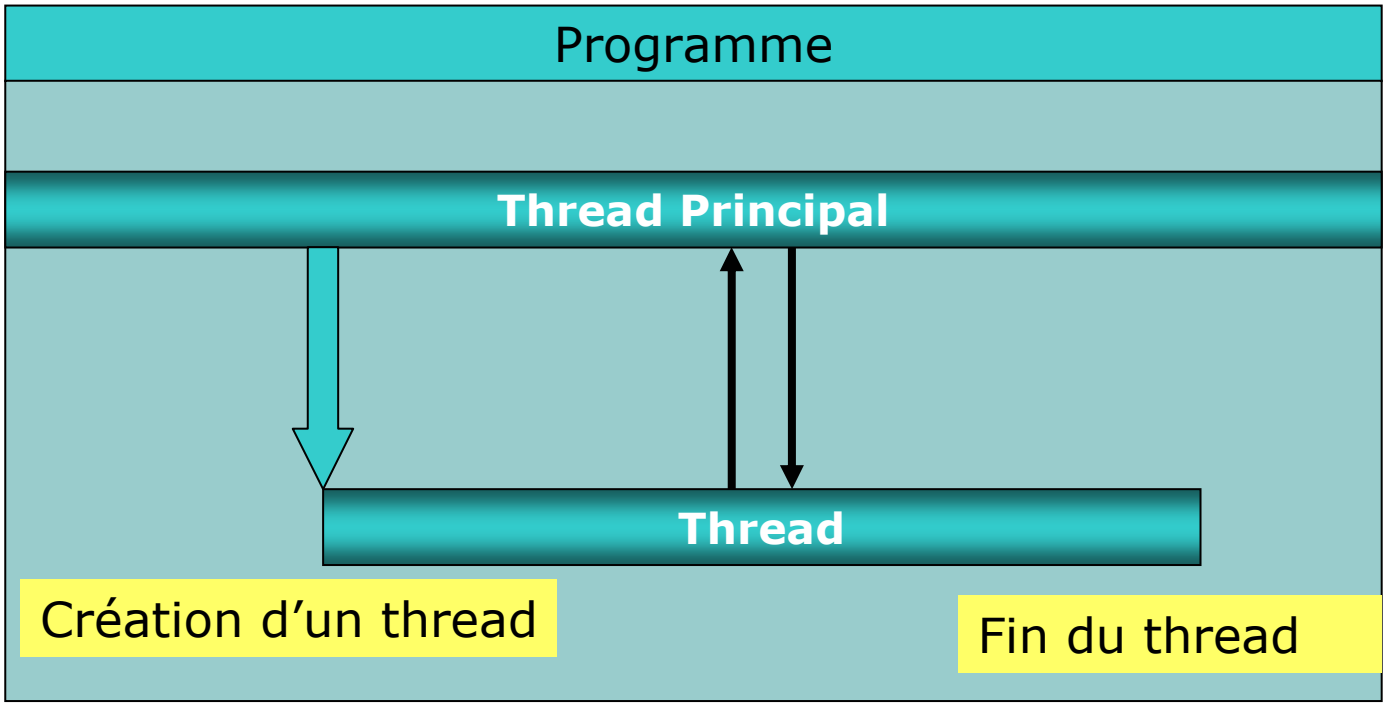
Introduction aux Threads

- Dans la programmation que nous avons vu jusqu'à aujourd'hui, les programmes suivaient de manière séquentielle les instructions du programme.

- Limites : Si un programme est en train de traiter des donner, il ne peut pas faire la mise à jour d'une barre de défilement. L'interface utilisateur peut être gelée le temps du traitement, ou par exemple l'écoute des boutons mise en attente.

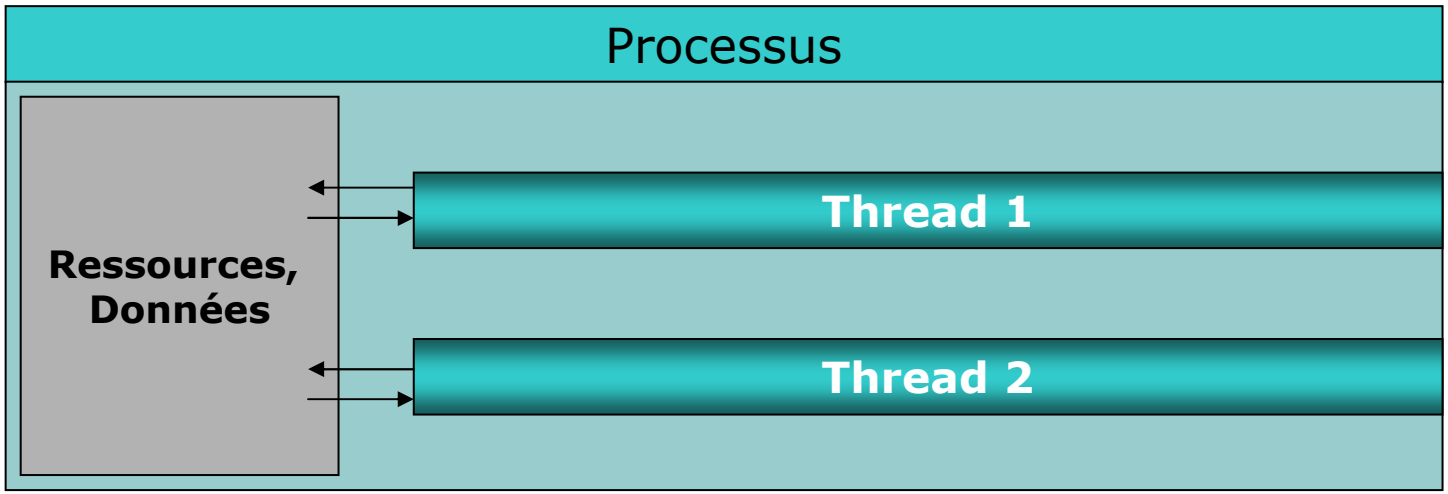
- Pour palier à ces problèmes, l'idée est d'avoir plusieurs fils d'exécution. Ces fils d' exécution s'appellent les **Threads**.

Introduction aux Threads



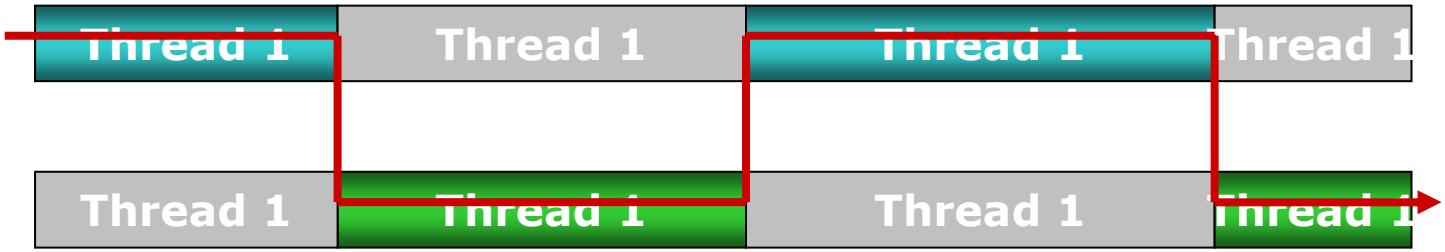
Introduction aux Threads

- Chaque thread possède :
  - Un état d'exécution
  - Sa propre pile
  - Un espace dédié à des variables propres au thread
  - L'accès aux variables du processus
  - L'accès aux ressources du processus (exemple : accès à un fichier ouvert par un autre thread du même processus).



Introduction aux Threads

- **Simultanéité physique :** ceci est possible sur des machines multi-processeurs. L'exécution de deux threads est répartie sur 2 processeurs.
- **Simultanéité logique :** chaque thread s'exécute à un moment donné, le processeur permute rapidement l' exécution de chacun donnant une illusion de simultanéité



Introduction aux Threads

- Les threads sont parfois appelés « **processus légers** »
- **Multithreading** : permet une simultanéité dans un contexte de processus unique
- Avantages du multithreading par rapport aux processus :
  - partage plus efficace des ressources communes
  - moins de surcharge au niveau du processeur
- Les threads sont au cœur même de Java et de la machine virtuelle : tout programme Java est exécuté en tant que Thread de la machine virtuelle. (Garbage collector, thread principal, pile des évènements graphique)

La classe Thread

La classe Thread

Package : java.lang

La classe java.lang.Thread implémente le mécanisme permettant la création d'un nouveau fil d'exécution.

Mise en œuvre :

- Faire dériver la classe de java.lang.Thread
- Puis fournir un point d'entrée au nouveau thread en surchargeant la méthode run()
- Appel de la méthode start(), sur une instance dérivant de Thread, afin de créer le nouveau fil d'exécution.

```
Class C extends Thread {
        public C {
        // … }
        public void run() {
        //… }
}
```

La classe Thread

Exercice :

Créer une classe ClasseThreadee, dérivant de Thread, ayant pour fonction d'écrire sur la sortie standard son nom, ainsi qu'un compteur.(de 1 à 100 000 par exemple)

Depuis une classe à part, créer 4 objets de ClasseThreadee s'exécutant simultanément.

La classe Thread

- **Avantages**

Simplicité de mise en œuvre

- **Inconvénients**

Java n'autorisant pas l'héritage multiple, le fait de dériver de Thread apporte certaines contraintes de conception objet.

- Autre solution pour le multi-threading : utilisation de l'interface j**ava.lang.Runnable**.

L’interface Runnable

L'interface Runnable

- La deuxième approche pour la création de thread est l'utilisation de l'interface **Runnable.**
- Vous pouvez transformer n'importe quelle classe en Thread, en la faisant implémenter l'interface Runnable.
- Concrètement, cela implique d'implémenter la méthode d'entrée dans le thread : **run().**

```
Class C implements Runnable {
        public C {
        // … }
        public void run() {
        //… }
}
```

L'interface Runnable

- Comment utiliser l'objet implémentant Runnable pour le faire s'exécuter dans un nouveau Thread ?
- Réponse : on crée un nouvel objet Thread, en passant en argument au constructeur une référence à l'objet implémentant l'interface Runnable

```
C obj = new C();
Thread nouvThread = new Thread(obj);
nouvThread.start();
```

L’interface Runnable

Exercice :

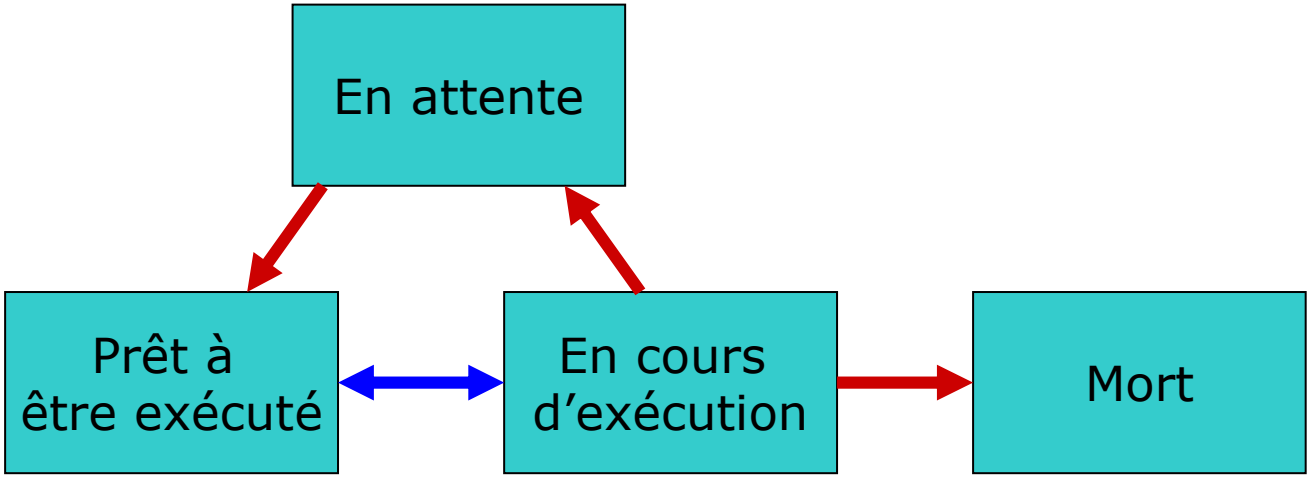
Modifiez le code de l’exercice précédent de sorte à utiliser l’interface Runnable.

Les états d’un Thread

Les états d'un Thread

- Comme nous l'avons vu en introduction, la machine virtuelle et le système allouent des temps d'exécution aux threads.
- On peut déjà imaginer que le thread peut être « en cours d'exécution », « en attente », « fini ».

- En fait il existe 4 états :



Les états d'un Thread

- Nous allons à présent nous intéresser aux méthodes qui permettent de changer l'état d'un thread par programmation.
- Méthode sleep(long timeInMillisecond) : cette méthode permet de stopper le thread en train de s'exécuter pour une période donnée. Si d'autres threads sont en état «Prêt à exécuter », le système autorise l'exécution de l'un d'eux.
- Méthode yield() : laisse l'opportunité à d'autres Threads de s'exécuter.
- Remarque : d'une manière générale, dès qu'un thread est en état d'attente demandé par programmation, il faut gérer les exceptions InterruptedException.
- La méthode isInterrupted(), permet de savoir si le flag de demande d'interruption a été activé.
- Nota : à ne pas confondre avec la méthode isAlive(), qui elle permet se savoir si le thread en question est encore dans un état autre que « mort ».

Les états d'un Thread

- Un thread peut se mettre en attente de la fin d'un autre thread.
- Pour cela on utilise la méthode join().
- Nota : possibilité de passer un timeout en paramètre.
- Nota 2 : ici encore, on est en attente demandée par programmation, il faudra donc traiter les exceptions de type InterruptedException.

```
thread1.join(); // Attends la mort de thread1
```

Les états d'un thread

- Complément : les threads possèdent des noms.
- On y accède par setName(), getName().

- Lorsqu'aucun nom n'a été spécifié, le système en attribue un par défaut en utilisant la règle suivante :

`Thread[Compteur]`