Les processus légers en Java : Threads

Stephane Malandain – POO - Java

Hes·SO GENÈVE
Haute Ecole Spécialisée
de Suisse occidentale

L'avenir est à créer

h e p i a
Haute école du paysage, d'ingénierie
et d'architecture de Genève

I. Les threads

- 2 manières de définir un thread en Java :
  - En dérivant la classe `Thread`, en redéfinissant la méthode `run()`, puis en instanciant cette sous-classe de `Thread`

```
Class ThreadDerive extends Thread {
   private int nFois;
   ThreadDerive(int n) { nFois = n; }
   public void run() {
     for (int j=0;j<nFois;j++)  System.out.println("Exéc. no" + j);
   }
}
```

Et le main correspondant :

```
public class Main {
   public static void main(String[] args) {
       ThreadDerive td = new ThreadDerive(5);
       td.start();
   }
}
```

I. Les threads

- en implémentant l'interface `Runnable` (ce qui correspond à une implémentation de la méthode `run()`) puis en passant une instance de type `Runnable` à un constructeur de la classe `Thread`.

```
class Boucle implements Runnable {
   private int nFois;
   Boucle(int n) { nFois = n; }
   public void run() { for (int j=0;j<nFois;j++)
                System.out.println("Exéc. no" + j);
   }
}
```

- Création et démarrage d'un objet Runnable

```
public class Main {
   public static void main(String[] args) {
        Boucle bcl = new Boucle(4);
        Thread td = new Thread(bcl);
        td.start();
   }
}
```

I. Les threads

- La méthode `run()` constitue le corps du thread. L'envoi du message `start()` à un objet `Thread` provoque son démarrage, c'est-à-dire l'exécution de sa méthode `run()`.

- Le thread se termine lorsqu'on quitte la méthode `run()`. La méthode `sleep(delai)` permet de mettre celui-ci en sommeil pendant `delai` millisecondes.

II. Thread – exemples

```
final List liste; // liste d'objets non-triés, supposée initialisée

// classe dérivant Thread pour trier la liste en tâche de fond
class TriBackground extends Thread {
   List l;
   public TriBackground(List l) { this.l = l; }
   public void run() { Collections.sort(l);} // corps du thread
}

// création d'un thread de type TriBackground
Thread trieur = new TriBackground(liste);
// démarrage du thread avec exec de la méthode run() pendant que le thread
// original poursuit son travail.

trieur.start();

// création équivalent d'un thread anonyme pour trier en tâche de fond

new Thread(new Runnable() {
    public void run() { Collections.sort(liste); }
}).start();
```

III. Thread – Timers

- Les classes Timer et TimerTask facilitent la gestion de tâches répétitives.

```
final DateFormat f = DateFormat.getTimeInstance(DateFormat.MEDIUM);

// définition de la tâche d'affichage du temps
TimerTask displayTemps = new TimerTask() {
   public void run() { System.out.println(f.format(new Date()));}
}

// création d'un objet Timer pour gérer la tâche
Timer timer = new Timer();

// exécution de la tâche toutes les secondes dès maintenant
Timer.schedule(displayTemps,0,1000);

// arrêt de la tâche d'affichage du temps
displayTemps.cancel();
```

IV. Thread – Synchronisation (1)

- Lorsqu'on utilise plusieurs threads, il faut veiller aux accès concurrents à une ressource.
- Exemple : un thread parcourt une liste pendant que l'autre la trie.
- Un thread peut verrouiller l'objet avant de le modifier, bloquant ainsi l'accès de cette ressource aux autres threads.
- On dit que le thread détient dans ce cas le **verrou.** Tout objet java peut être verrouillé.

IV. Thread – Synchronisation (2)

- On verrouille une méthode d'une classe avec le modificateur **`synchronized`.**

- Un thread voulant envoyer un message à un objet d'une classe avec une methode `synchronized` devra d'abord obtenir le verrou sur cet objet, empêchant par-là même tout autre thread d'exécuter pendant ce temps la méthode `synchronized` de cet objet.

- Si la méthode est statique, le thread doit obtenir un verrou sur la classe, le mécanisme étant similaire.

- Il est possible de verrouiller un objet au niveau d'un bloc.

IV. Thread – Coordination

- Les méthodes `wait()` et `notify()` de la classe Objet permettent de coordonner l'action des threads, par mise en attente ou activation (réveil)

- Chaque objet Java à un verrou qui lui est associé, et gère une liste de threads en attente.

- Quand un thread appelle la méthode `wait()` d'un objet, les verrous détenus par ce thread sont temporairement restitués. Il est alors bloqué et ajouté à la liste des threads en attente de l'ôbjet.

- Lorsqu'un autre appelle la méthode `notify()` du même objet, celui-ci réveille un des threads en attente qui peut poursuivre son éxécution. La méthode `notifyAll()` les réveille tous !

IV. Thread – Coordination - exemple

```
// un thread insère un objet dans la queue via inserer(), un autre
// en retire un avec extraire(); en l'absence de données, extraire()
// met en attente tandis que inserer() avertit de la présence de données

public class Queue {
   LinkedList lst = new LinkedList ();
   public synchronized void inserer (Object obj) {
       lst.add(obj);  // ajoute l'objet en queue de liste
       this.notify(); // avertit les threads en attente
                      // que les données sont disponibles
}

public synchronized Object extraire() {
    while (lst.empty()) {
        try { lst.wait(); } // mise en attente du thread appelant
        catch( InterruptedException ie ) { }
    }
    return lst.remove(0);
 }
}
```