Programmation Temps Réel
TP Thread/Executor/Lock

February 14, 2018

Thread/Runnable en java

Les threads permettent de concevoir des programmes "multi-tâches", c'est à dire des programmes qui réalisent plusieurs actions simultanément. En Java, le programme principal est lui-même un thread mais il est possible de faire démarrer d'autres threads sans l'interrompre. Dans la pratique, le codage de thread peut se faire de deux façon :

- une classe peut hériter de la classe `Thread` Déclaration du Thread :

  ```
  public class MonThread extends Thread{

    public void run(){
      // code de ce que fait le thread
    }
  }
  ```

  Lancement du Thread :

  ```
  MonThread t = new MonThread();
  t.start();
  ```

- une classe peut implémenter l'interface `Runnable`. Elle ne comporte qu'une seule méthode : la méthode `run()`. Lorsqu'une classe implémente

cette interface, il faut donc surcharger cette méthode. On y écrit le code exécuté par le thread.

Ensuite, on peut instancier un Thread avec une instance de cette classe. La méthode `run()` sera exécutée lorsque la méthode `start()` du thread sera appelée.

Déclaration du Thread :

```
public class MonThread implements Runnable{

  public void run(){
    // code de ce que fait le thread
  }
}
```

Lancement du Thread :

```
Thread t = new Thread(new MonThread());
t.start();
```

Travail à réaliser

Exercice 1

Question 1

Ecrivez une classe `MonThread` qui hérite de `Thread`. Elle comprend un attribut `compteur` de type `int`.

Lorsqu'il sera exécuté, le thread exécutera 10 fois la séquence suivante :

- affichage du nom du thread courant et la valeur du compteur ;
- effectuer une pause d'une durée aléatoire comprise entre 0 et 1 seconde.

A la fin de son exécution, le thread indique qu'il s'arrête en affichant une chaîne de caractères (qui comprend son nom).

Question 2

Testez votre classe dans une nouvelle classe `TestMonThread` qui est exécutable. Cette classe crée autant de threads que demandé sur la ligne de commande et les lance simultanément.

Question 3

On veut maintenant afficher le message "Fin de tous les threads" lorsque tous les threads sont terminés. Utilisez la méthode `join` pour attendre l'arrêt des threads.

Exercice 2

L'exercice représente plusieurs files d'impression d'une imprimante.

Écrivez une classe `Impression` qui implémente l'interface `Runnable`. La classe possède un attribut nom qui contient une chaîne de caractères (le nom du document) et un autre attribut qui indique le nombre de pages. Écrivez le constructeur de cette classe, le construteur prend les deux paramètres (nom et nombre de pages).

La classe redéfinit la méthode `run` et affiche n pages (une page sera simplement représentée par son numéro et le nom du document).

Dans une classe exécutable `TestImpression`, vous devez instancier deux Threads avec des objets `Impression`. Lancez les threads et observez le résultat.

Explication : puisque plusieurs threads peuvent imprimer en même temps, on n'a pas de garantie de l'ordre d'impression des pages des différents documents. Ce qui peut poser problème ...

Solution : imposer que certaines parties du codes ne soient exécutables que par un seul thread à la fois. Pour cela, on utilise le mot clé `synchronized`. Le mot clé `synchronized` prend un paramètre : le nom d'un objet. Un objet java possède un mécanisme de verrouillage. Lorsque la méthode est synchonisée sur un objet aucune autre partie de code synchronisée sur le même objet ne peut être exécutée. Dans le cas présent, vous pouvez par exemple verrouiller l'utilisation de la sortie standard :

```
synchronized(System.out){
  ... // code protégé
}
```

Exercice 4 : arrêter des Threads

- règle de base : le thread est terminé lorsqu'il sort de sa méthode run...

- Il ne faut pas utiliser la méthode stop() qui est deprecated (risque de deadlock)
- Utiliser la méthode interrupt() qui permet de lever l'exception InterruptedException (lorsque le thread est placé en état d'attente).
- Le thread n'est pourtant pas interrompu, il faut prévoir dans la clause catch de postionner le flag d'interruption avec Thread.currentThread().interrupt();
- ensuite prévoir une boucle (dans la méthode run) sur ! isInterrupted().

Exercice à réaliser

Ecrire une classe qui hérite de Thread. La méthode run sera un simple compteur qui se met en sommeil pendant 1 seconde (en boucle).

Ecrire ensuite un programme qui lance ce thread puis attend une saisie de l'utilisateur. Suite à cette saisie, le thread est interrompu. On affichera le temps passé dans chaque boucle d'attente.

Executor

Il est possible en java d'utiliser un service d'ordonnancement.

Les interfaces java.util.concurrent.Executor et java.util.concurrent.ScheduledThreadPoolExecut décrivent des fonctionnalités permettant l'exécution différée de tâches implémentées sous la forme de Runnable.

(détail : https://www.jmdoudoux.fr/java/dej/chap-executor.htm)

Exercice à réaliser

Reprendre les exercices précédents en implémentant ces interfaces.