ESIL - Informatique - Programmation distribuée
Nicolas Baudru - Traian Muntean

TD1 - Introduction aux threads JAVA

Exercice 1 (Création de threads)

En Java, il existe deux méthodes pour créer des threads. La première étend la classe Thread :

```
class MonThread extends Thread {
      public void run() {
            // ici se trouve la description du comportement du thread
      }
      public static void main (String[] args) {
            MonThread t1 = new MonThread(); //on crée une instance de l'objet MonThread
            t1.start(); // puis on lance le thread. t1 exécute alors la méthode run().
      }
}
```

La deuxième méthode implémente l'interface Runnable :

```
class Tache implements Runnable {
      public void run() {
            // ici se trouve la description du comportement du thread
      }
      public static void main (String[] args) {
            Tache job = new Tache(); //on crée une tache pour un thread
            Thread t1 = new Thread(job); //on crée une instance de Thread avec la tache job
            t1.start(); //puis on lance le thread. t1 exécute alors la méthode run() de
                        // la tache job.
      }
}
```

Pourquoi existe-t-il deux méthodes ? Quels sont les différences entre ces deux méthodes ?

Exercice 2 (Threads et JVM)

Ecrivez en java un programme qui utilise deux threads en parallèle :

- le premier affichera les 26 lettres de l'alphabet ;
- le second affichera les nombres de 1 à 26.

Déterminez le résultat de ce programme lors de son exécution ?
A votre avis, par qui et comment les threads sont gérés dans la machine ?

Exercice 3 (Méthodes de la classe Thread)

Information sur les threads - A quoi sert les méthodes suivantes :

- static int activeCount()
- static int enumerate(Thread[] tarray)
- static Thread currentThread()

Ordonnancement sur les threads - A quoi sert les méthodes suivantes :

- void setPriority(int)
- int getPriority()
- static void yield()

Manipulation sur les threads - Quelles méthodes provoquent :

- la mise en attente
- l'attente de l'arrêt d'un thread donné
- l'interruption d'un thread

Ecrivez un programme dont le thread principal lance et nomme trois nouveaux threads. Chaque thread ainsi créé doit effectuer 10 fois les actions suivantes :
- attendre un temps aléatoire compris entre 0 et 200 ms,
- puis afficher son nom.

Le thread principal devra attendre la fin de l'exécution des trois threads qu'il a créés avant de terminer son exécution.

Exercice 4 (Threads et concurrence)

L'exercice suivant montre ce qui peut se passer quand deux threads (ici Sylvie et Bruno) partagent un même objet (ici un compte en banque commun). Pour cela vous allez écrire un programme qui comprendra deux classes, `Compte` et `JobSylvieEtBruno`.

La classe `Compte` est très simple. Elle comprend :
- un attribut privé `solde` initialisé à 100 représentant le solde courant du compte
- une méthode `retirer(int montant)` permettant de retirer un certain montant du compte.

La voici en détaille:

```
class Compte {
      private int solde = 100;

      public int getSolde() {
            return solde;
      }
      public void retirer(int montant){
            solde = solde - montant;
      }
}// fin class Compte
```

La classe `JobSylvieEtBruno` implémente `Runnable` et représente le comportement que Sylvie et Bruno ont tous les deux. Leur comportement est assez particulier puisque Sylvie et Bruno s'endorment très souvent, et en particulier pendant qu'ils effectuent un retrait . Cette classe contient :
- un attribut `compte` de type `Compte` représentant le compte en banque de Sylvie et Bruno.
- une méthode `effectuerRetrait(int montant)` permettant à Sylvie ou Bruno d'effectuer un retrait sur leur compte en banque. Le comportement de cette méthode est le suivant : la personne voulant effectuer le retrait vérifie le solde, puis s'endort 500 ms, puis à son réveil (au bout des 500 ms) effectue le retrait. Le nom de la personne effectuant le retrait doit être signalé.
- une méthode `run()` décrivant le comportement de Sylvie et Bruno. Il s'agit d'effectuer en boucle (par exemple 10) des retraits de 10 euros.
- enfin la méthode `main(String[] args)` crée deux threads (Sylvie et Bruno) avec le même Runnable (ici de type JobSylvieEtBruno), les nomme Sylvie et Bruno, puis les lance.

Après avoir écrit la classe `JobSylvieEtBruno`, examinez le comportement de ce programme.

Exercice 5 (Threads et attributs partagés)

Ecrivez un (ou des) petit(s) programme(s) mettant en évidence si deux threads d'une classe A implémentant Runnable partagent les attributs `static, public, private`,... de cette classe A.