Programmation réseaux – TP 1
Rappels : threads en Java

Vincent Balat, Emmanuel Beffara, Thierry Cachat,
Juliusz Chroboczek, Grégoire Henry, Pierre Letouzey,
Jean-Baptiste Yunès

2008

On rappelle que la documentation est accessible soit à la source sur le site Java de la compagnie Sun `http://java.sun.com/`, soit sur le mirroir `http://www.liafa.jussieu.fr/~yunes/Java/docs/api/overview-summary.html`

Exercice 1 Introduction à la concurrence interne

`java.lang.Thread`, `java.lang.Runnable`

Définir une classe `A` qui implémente l'interface `Runnable` et qui lorsqu'elle est contrôlée par un `Thread` affiche des caractères `A` à l'écran sans discontinuer. Faire de même avec une classe `B`.
Écrire un programme qui permet de lancer deux `Thread`s exécutant ces deux `Runnable`s (un `A` et un `B`) en concurrence.

Exercice 2 Introduction au contrôle de la terminaison

`java.lang.Thread.interrupt()`, `java.lang.Thread.interrupted()`, `java.lang.Thread.isInterrupted()`

Définir une classe `Affichage` qui représente un thread avec un champ booléen public `continuer` initialisé à `true` et dont la méthode `run` affiche un message toutes les secondes tant que `continuer` vaut `true`.
Écrire un programme qui lance un thread de la classe `Affichage` puis change la valeur du champ `continuer` après intervention de l'utilisateur.
Modifier le programme pour utiliser les méthodes `interrupt()` et `isInterrupted()`.

Exercice 3 Pas-en-même-temps et Attends-moi sont sur un bateau

`synchronized`, `java.lang.Thread.join()`

Écrire un programme Java qui crée 1000 threads et maintient un compteur `nb` du nombre de threads créés jusque-là. Le thread principal attend ensuite la terminaison de chacun des threads puis affiche la valeur du compteur.
Chaque thread incrémentera le compteur dès sa création en appelant la méthode suivante :

```
public static void incremente() {
  int c = nb;
  try {
    Thread.sleep(1);
  }
  catch (Exception e) {
```

```
    System.exit(0);
  }
  nb = c+1;
}
```

Ici l'appel à `java.lang.Thread.sleep()` simule un calcul d'une durée d'une milliseconde.

- Tester le comportement du programme lorsque l'on ajoute ou non le mot clé `synchronized` pour la méthode `incremente`.
- Quel verrou est utilisé ?
- Quel effet est-il induit sur le temps de calcul ?
- Que se passe-t-il lorsque l'on enlève le mot-clé `static` devant `incremente()` ?

Réimplémentez en utilisant une classe `Compteur` qui crée un compteur à chaque instanciation (et qui contiendra la méthode `incremente()`). Quel verrou est utilisé ?

Exercice 4 J'attends et Réveille-toi sont sur un bateau

`java.lang.Object.wait()`, `java.lang.Object.notify()`

1. Implémenter une classe `Evenement` fonctionnant de la manière suivante :
   - Lorsque l'événement est créé, un thread peut appeler la méthode `attend` de `Evenement`, ce qui aura pour effet de le bloquer jusqu'à ce que l'événement survienne ;
   - Un thread peut appeler la méthode `declenche` de `Evenement`, ce qui a pour effet de réveiller un (et un seul) des threads en attente de l'événement.
2. À l'aide de la classe précédente, implémenter le petit programme suivant. Trois personnes sont au bout d'un couloir et doivent franchir trois portes pour se retrouver à l'autre bout. Chaque personne est un thread qui attend un événement « ouverture de porte ». Un autre thread attend des commandes de l'utilisateur pour ouvrir une porte (par exemple taper 1 pour ouvrir la première, 2 pour la deuxième et 3 pour la troisième, q pour quitter). L'ouverture d'une porte ne laisse passer qu'une personne à la fois (la porte se referme aussitôt). Pour la visualisation, on pourra utiliser encore un autre thread qui affiche le nombre de personnes dans chaque compartiment.