Programmation Réseaux et Concurrente
Master 1 Informatique UPEC 2015/2016
TP 3 : Concurrence sans synchronisation

Rappel création de *thread* en Java

Pour programmer avec des *threads*, il faut suivre les étapes suivantes :

1. écrire une nouvelle classe qui étend la classe `Thread` ou qui implémente l'interface `Runnable` ;
2. dans cette nouvelle classe, définir une méthode `run` qui contient le code à exécuter par le *thread* ;
3. créer un objet de cette classe avec `new` ;
4. lancer le *thread* en appelant la méthode `start`.

Synchronisation de *thread*

L'exclusion mutuelle en Java est fournie par le mot clé `synchronized`, qui peut être utilisé comme attribut d'une méthode où d'une séquence d'instructions. Par exemple, si on a une donnée partagée `data` dans une classe `Interfere`, sa mise à jour en exclusion mutuelle peut être faite de deux façons :

1. exclusion mutuelle sur la méthode de mise à jour :

```
class Interfere {
  private int data = 0;
  public synchronized void update () {
    data++;
  }
}
```

2. exclusion mutuelle sur une séquence d'instructions :

```
class Interfere {
  private int data = 0;
  public void update () {
    synchronized (this) { data++; }
  }
}
```

   Dans le deux cas l'exlusion mutuelle est faite sur l'objet qui contient la méthode.

Un processus qui exécute un code en exclusion mutuelle peut appeler une autre méthode `synchronized`. Si cette méthode appartient à un autre objet, l'exclusion mutuelle est maintenue pour le premier objet durant l'appel. Ceci peut provoquer des deadlocks si une méthode `synchronized` d'un objet `o1` appelle une méthode `synchronized` d'un autre objet `o2` et vice-versa.

Exercices

Pour chacun des exercices ci-dessous, essayer d'observer le comportement du programme en obtenant des ensemble d'exécution différentes (on peut endormir des threads pendant un moment grâce à la méthode statique `sleep` de la classe `Thread`).

Exercice 1:

Dans une première version, on s'intéresse qu'à la concurrence, pour observer l'entrelacement. Écrire deux threads qui affichent de façon complètement indépendante les nombres de 1 à 100.

Observer le comportement de votre programme.

Exercice 2:

Dans une deuxième version, on s'intéresse également à observer une possible interférence. écrire deux threads qui incrémentent chacun à son tour une variable partagée, et qui ensuite affichent son contenu.

Observer le comportement de votre programme.

Exercice 3:

Dans cet exercice on programmera un thread producteur et plusieurs thread consommateurs, pour observer l'absence d'exclusion mutuelle et la famine. Un producteur vérifie qu'un drapeau soit `false`,incrémente une variable partagée, met à `true` le drapeau et recommence du début. Chaque consommateur attend que le drapeau soit à `true`, ensuite il affiche son nom (passé en paramètre à la création) et le contenu de la variable, met le drapeau à `false` et recommence du début.

Observer le comportement de votre programme. Il faudra trouver un moyen d'observer l'absence d'exclusion mutuelle, c'est à dire le fait que deux consommateurs consomment la même valeur. On pourra aussi noter si tous les consommatuers ont une chance de consommer une valeur.

Exercice 4:

Dans cet exercice on programmera des threads pour observer l'interblocage. Les deux threads ont accès à deux drapeaux, `x1` et `x2`. Le premier thread attend que `x1` est `true`, et le mets à `false`, ensuite attend que `x2` soit `true` et le mets à `false`, et ensuite affiche un message, remet les drapeaux à `true` et il recommence. Le deuxième fait la même chose avec les deux drapeaux inversés.

Trouvez le moyen d'observer l'interblocage (livelock).