CC de Compléments en Programmation Orientée Objet n° 2 : Concurrence

Important : Ce TP est à rendre sur Moodle avant l'heure indiquée au tableau, le dépôt sera ensuite clos. Merci de rendre une archive tar ou zip. Rédigez toutes les explications utiles à la compréhension de vos choix dans des commentaires dans le code.

Exercice 1 : (∼15 minutes)

Le code suivant contient trois propositions qui se ressemblent beaucoup.

```
1  public class A {
2
3    // proposition 1
4    // private static int a1=0, a2=0;
5    // public static synchronized void h1(){ a1++; }
6    // public static synchronized void h2(){ a2++; }
7
8    // proposition 2
9    // private int x1=0, x2=0;
10   // public synchronized void f1 () { x1++; }
11   // public synchronized void f2 () { x2++; }
12
13   // proposition 3
14   // private int y1=0, y2=0;
15   // private final Boolean b1 = false, b2 = false;
16   // public void g1 () { synchronized ( b1 ) { y1++; } }
17   // public void g2 () { synchronized ( b2 ) { y2++; } }
18 }
```

Pourriez vous expliquer leurs différences, en étant le plus clair possible sur les particularités que peuvent avoir ces approches sur des exécutions diverses et variées. (Ecrivez vos réponses dans un fichier `Exo1.txt`)

Exercice 2 : (∼1h15 minutes)

On va s'intéresser à des réactions chimiques simples entre oxygène (O) et hydrogène (H). Pour donner un peu de volume à votre code (mais pas trop) on se contentera de deux réactions :

— H + H → H2
— O + O → O2

Le contexte de cet exercice c'est que des threads seront définis à partir d'un `Runnable` issu d'une classe `Element` qu'il vous faudra écrire. Un élément portera un nom parmis : "H","O","H2","O2", et aussi éventuellement "" (la chaine vide). On procède ainsi puisqu'on va regarder une réaction comme H + H → H2, en la développant en "H" + "H" → "H2" + "", c'est à dire que si deux thread portant le nom H réagissent ensemble, l'un des deux changera son nom en "H2", et l'autre en "" (d'ailleurs, ensuite ce dernier ne sera plus utile)

Pour que les éléments se rencontrent, on met en place une unique `Chambre` partagée entre tous les éléments. Tous les threads essayeront de rentrer dans cette chambre, mais seuls deux éléments pourront y parvenir en même temps. Lorsque 2 éléments y seront, dans l'intimité de cette chambre ils verront si une réaction a lieu ou non ; ils changeront alors éventuellement de nom, puis sortiront, laissant ainsi la place à d'autres réactions. (**conseil :** si vous manquez de temps, concentrez vous sur les 4 premières questions)

1. Ecrivez une classe `Element` qui implémente `Runnable`. Ses objets possèdent un attribut `nom`, et un entier `num` qui pourra vous être utile pour débugger. Prévoyez y également une variable statique de type `Chambre` où `Chambre` est une classe interne statique pour le moment vide.

2. Dans une classe `Test` recopiez ce `main` :

```
public static void main(String[] args) {
    List<Thread> jobs=new ArrayList();
    for (int i=0;i<10;i++)
        if (i<5) jobs.add(new Thread(new Element("H",i)));
        else jobs.add(new Thread(new Element("O",i)));
    for(Thread t:jobs) t.start();}
```

3. La classe `Chambre` a plusieurs attributs qui permettront aux threads de se synchroniser à différentes étapes. On utilisera un entier pour compter le nombre d'éléments présents dans la chambre, un autre entier pour compter le nombre d'éléments qui auront compris quel sera leur nouveau nom, et un tableau d'éléments pour savoir qui est présent actuellement. Vu les réactions chimiques qui nous intéressent le nombre de présent sera limité à deux. La classe `Chambre` contiendra également les méthodes :
   (a) `int toctoc(Element x)` : un élément x qui souhaite entrer dans la chambre fera appel à cette méthode, elle lui permet de frapper à la porte de la chambre tout en s'identifiant. La méthode le mettra en attente si la chambre est déjà pleine, et sinon elle l'accueillera. La valeur retournée correspond à l'ordre d'arrivée : 0 si x arrive en premier, et 1 s'il arrive en second.
   (b) `void waitfull()` : lorsqu'un élément a pu rentrer, il doit encore attendre que la chambre se remplisse. C'est l'appel à cette méthode qui le fera patienter.

   Implémentez ces aspects de la classe `Chambre` ; puis pour vérifier que cela fonctionne, testez les dans le `run()` de la classe `Element`. On rappelle que chaque élément cherche à rentrer dans la chambre, et pour ceux qui sont rentrés, ils doivent pouvoir constater le moment où elle est pleine. Un test possible consisterait en un affichage :
   " H - no 3 - constate que la chambre est pleine"
   " O - no 8 - constate que la chambre est pleine"
   avec le reste des threads en attente (car on n'a pas encore traité la sortie de la chambre). Vérifiez également que si vous n'avez qu'un seul thread (et pas 10) alors il est bloqué et rien ne s'affiche.
4. Dans la classe `Chambre` qu'on peut supposer à présent pleine, écrivez les méthodes `boolean reaction1()` et `boolean reaction2()` qui permettent de tester si on a à faire à l'une des réactions qui nous intéressent ; puis écrivez `String newName(int no_arrivée)` qui sera utile pour proposer à un élément son nouveau nom (ce sera soit le résultat de la réaction, soit son ancien nom si ce ne sont pas des composants réactifs). Dans le `run` d'`Element` stockez le résultat de `newName` dans une variable locale `futureName`.
5. Pour sortir et ensuite pouvoir libérer la chambre, les deux éléments doivent encore se synchroniser car il faut qu'ils aient tous les deux eu connaissance de leur nouveau nom. Ecrivez une méthode `void readyToChangeName()` dans `Chambre` qui s'appuie sur le deuxième entier que nous avons prévu dans l'objet chambre. Cette méthode permettra simplement à un élément d'annoncer à la chambre qu'il a pris connaissance de son nouveau nom ; elle sera blocante tant que 2 éléments ne l'auront pas invoquée.
   Appelez la dans votre `run()` et faites la suivre du changement effectif du nom. Pour testez, affichez une trace de ce changement.
6. Ecrivez enfin une méthode `void reset()` dans `Chambre` qui la vide proprement ; et faites en sorte que ce soit l'élément ayant eu l'ordre d'arrivé 0 qui se charge de faire le `reset()`. A présent dans votre test on devrait constater 5 rencontres (et éventuellement l'apparition de nouveaux produits)