Université Paris Diderot
Master 2
Programmation répartie

TP1

Java
à rendre pour le 4 mars 2019

Le tp peut être rendu sur moodle par groupe de 4 étudiants maximum. Il est demandé de rédiger un texte de réponse aux questions ; pour les exercices 3 et 4 on demande en plus de rendre un fichier tar exécutable (par la commande java -tar). contenant les programmes.

Exercice 1.— ThreadLocal et atomicité On considère :

```
class MonObjet {

    ThreadLocal<Integer> last;//nb ecriture de chaque thread
    int value;//valeur commune
    int valuebis;//valeur commune

    public MonObjet(int init) {
        value = init;
        last = new ThreadLocal<Integer>() {
            protected Integer initialValue() {
                return 0;
            }
        };
    }
    public int read() {
        return value;
    }

    public void add() {
        last.set(last.get() + 1);
        value = value + 1;
        valuebis = valuebis + 1;
    }
}

class MyThread2 extends Thread {

    public MonObjet o;
    public int nbwrite;

    public MyThread2(MonObjet o, int nbwrite) {
        this.o = o;
        this.nbwrite = nbwrite;
    }

    public void run() {
```

```
        for (int i = 0; i < nbwrite; i++) {
            o.add();
            this.yield();
        }
        System.out.println("la thread "+this.getName()+
        " value=" + o.value + ", " + "valuebis=" + o.valuebis + " et "
                + " last=" + o.last.get());
    }
}

class TP {

    public static void main(String[] args) {
        MonObjet o = new MonObjet(0);
        MyThread2 W;
        MyThread2 R;
        W = new MyThread2(o, 1000);
        R = new MyThread2(o, 5000);
        W.start();
        R.start();
        try {
            R.join();
            W.join();
        } catch (InterruptedException e) {
        }
        System.out.println("value=" + o.value + ", " + "valuebis=" + o.valuebis + " et "
                + " last=" + o.last.get());
    }
}
```

1. Executer ce programme et expliquer les résultats obtenus.
2. Si l'instruction `value=value +1` et `valuebis=valuebis +1` étaient atomiques, qu'elles devraient être les valeurs de `o.value` et `o.valuebis` affichées lors de l'exécution de `TP.main` ?
3. Si les variables `value` et `valuebis` sont déclarées comme `volatile` le résultat est-il différent ?
4. Si on au lieu de `value=value +1` et `valuebis=valuebis +1` on utilisait `value++` et `valuebis++` le résultat est-il différent
5. Expliquer l'effet de l'usage de `ThreadLocal` pour la variable `last`.

Exercice 2.— ThreadLocal On considère la classe suivante qui permettra de nommer les threads par des entiers de façon à ce qu'un code quelconque puisse connaître le nom du thread qui l'exécute :

```
public class ThreadID {
    static volatile int nextID=0;
    private static class ThreadLocalID extends ThreadLocal<Integer>{
        @Override
        protected synchronized Integer initialValue(){
            return nextID ++;
        }
    }
    private static final ThreadLocal<Integer> threadID =new ThreadLocal<Integer>(){
                @Override
```

```
                protected synchronized Integer initialValue(){
            return nextID ++;
        }
    };
    public static int get(){
        return threadID.get();
    }
    public static void set (int index){
        threadID.set(index);
    }
}
```

et la classe :

```
public class TPb {

    public static void main(String[] args) {
        MonObjet o = new MonObjet(0);
        Thread TH[] = new Thread[10];
        for (int i = 0; i < 10; i++) {
            TH[i] = new MyThreadA("nom=" + i);
        }
        try {
            for (int i = 0; i < 10; i++) {
                TH[i].start();
            }
            for (int i = 0; i < 10; i++) {
                TH[i].join();
            }
        } catch (InterruptedException e) {
        }
    }

}

class Travail {

    public void faire() {
        ThreadID tID = new ThreadID();
        System.out.println("la thread " + tID.get() + " travaille : Thread id "
                        + (Thread.currentThread()).getId());
    }
}

class MyThreadA extends Thread {

    public MyThreadA(String name) {
        super(name);
    }

    @Override
    public void run() {
        ThreadID tID = new ThreadID();
```

```
        System.out.println("la thread " + tID.get() + " débute: Thread id"
                    + (Thread.currentThread()).getId());

        (new Travail()).faire();
        System.out.println("la thread " + tID.get() + " termine : Thread id "
                    + (Thread.currentThread()).getId());
    }
}
```

1. Exécuter ces programmes
2. Expliquer les résultats
3. Que fait la class `ThreadLocal` ?

Exercice 3.— Algorithmes de Peterson On considère l'interface :

```
public interface Lock{
   public void lock();
   public void unlock();
}
```

Les algorithmes d'exclusion des questions suivantes implémentent cette interface.

1. Proposer une implémentation de l'algorithme de Peterson à deux threads proposé en cours. Fournir des exemples d'exécution de l'algorithme.
2. Proposer une implémentation de l'algorithme de la boulangerie de Lamport décrit en cour. Fournir des exemples d'exécution de l'algorithme.

Exercice 4.— Registres
On considère l'interface :

```
public interface Register<T> {
  public T read();
  public void write(T v);
}
```

On suppose que l'on dispose d'une implémentation de cette interface :

```
public class SafeSMRSWBool implements Register<Boolean> {
  public boolean read() { }
  public void write(boolean x) {}
}
```

qui réalise des registres booléens sûrs avec 1 lecteur et 1 écrivain

1. Définir une classe `SafeMRSWBool` qui implémente des registres MRSW sûrs à valeurs booléennes en utilisant de la classe `SafeSMRSWBool`.
2. Définir une classe `RegMRSWBool` qui implémente des registres MRSW réguliers à valeurs booléennes en utilisant les classes précédentes.
3. Définir une classe `RegMRSWInt` qui implémente des registres MRSW réguliers à valeurs entières en utilisant les classes précédentes.
4. Définir une classe `AtomicMRSWInt` qui implémente des registres atomiques à valeurs entières en utilisant les classes précédentes.