Université Paris Diderot
Master 2
Programmation répartie

TP1

Java
à rendre pour le 26 février

Exercice 1.— ThreadLocal et atomicité

On définit la classe MonObjet grâce à laquelle les threads partagent un objet.

```
public 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;}};
SThread    };
    public int read(){ return value;}
    public void add( ){
        last.set(new Integer(last.get()+1));
        value=value +1;
        valuebis=valuebis +1;
         }
    }
```

avec

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

```
}
public class TPC2 {
    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());
    }
}
```

- Si l'instruction `x=x+1` était atomique, qu'elles devraient être les valeurs de `o.value` et `o.valuebis` affichées lors de l'exécution de `Main2.main`?
- Après avoir exécuté le `Main2.main` qu'elles sont les valeurs affichées? L'instruction `x=x+1` est ellle atomique en java?
- Comment assurer qu'à la fin du `Main2.main` on ait `o.value=o.valuebis`= nombre totale d'écritures?
- Quelle est la difference entre les variables `value` et `last`?

Exercice 2.— ThreadLocal
On propose la classe suivante pour donner un identifiant à chaque thread que l'on crée:

```
public class ThreadID {
    private static volatile int nextID=0;
    private static class ThreadLocalID extends ThreadLocal<Integer>{
        protected synchronized Integer initialValue(){
            return nextID ++;
        }
    }
    private static ThreadLocalID threadID =new ThreadLocalID();
    public static int get(){
        return threadID.get();
    }
    public static void set (int index){
        threadID.set(index);
    }
}
```

Et deux programmes utilisant cette classe:

```
public class TPC2 {
    public static void main(String[] args) {
```

```
        MonObjet  o= new MonObjet(0);
        MyThread2 TH[]= new MyThread2[10];
        for( int i=0;i<10; i++) TH[i]=new MyThread2("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){};
    }
}

public class MyThread2 extends Thread{
/* premiere version
    public MonObjet  o;
    public int nbwrite; int x;
    public ThreadID tID;
    public MyThread2(String name){
        this.o=o;
        this.nbwrite=nbwrite;
    }
    public void run(){
      tID=new ThreadID();
      System.out.println("la thread "+tID.get());
      try{this.sleep(10);}
      catch(Exception e){e.printStackTrace();      }
      System.out.println("la thread " +tID.get() + "apres le sommeil  ");
   }
}

public class MyThread2 extends Thread{
/* deuxieme version
    public MonObjet  o;
    public int nbwrite; int x;
    public ThreadID tID;
    public MyThread2(String name){
        this.o=o;
        this.nbwrite=nbwrite;
        tID=new ThreadID();
    }
    public void run(){
       System.out.println("la thread "+tID.get());
       try{this.sleep(10);}
       catch(Exception e){e.printStackTrace();      }
       System.out.println("la thread " +tID.get() + "apres le sommeil  ");
    }
}
```

1. Exécuter TCP2 avec la première version de MyThread2 et avec la 2ème.
2. Comment expliquez vous les différences de comportement?

Exercice 3.— Volatile

Exécutez le programme suivant soit avec la déclaration:
`public static volatile int check=0;`
soit avec:
`public static int check=0;`
Comment expliquez vous ces comportements?

```
public class Main {
/* 2 versions avec ou sans volatile*/
        public  static volatile int check=0;
        public static void main(String[] args) {
                MyObject object = new MyObject();
                Stop s = new Stop();
                s.start();
                object.start();
         }
}

public class MyObject extends Thread {
          public void work()
         { int cur;
         for(cur=Main.check;cur!=10;) {
              if (Main.check>cur) {
                  System.out.println("check = "+Main.check+ " cur = "+cur);
                  cur=Main.check;
            }
                }
        }
public void run() {
                super.run(); work();System.out.println("coucou");
        }
}

public class Stop extends Thread {
        private boolean finish;
        public Stop() { finish = false; }
        public void incr()
        {
                Main.check++;
                if (Main.check == 11 )
                 {      System.out.println("received 11 stop");
                        this.finish=true;
                  }
          }
         public void run() {
                super.run();
                while(!finish){
                   try {
                       sleep(1000);
                   } catch (InterruptedException e) { e.printStackTrace();}
                        incr();
                 }
       }
```

```
}
```

Exercice 4.— Exclusion mutuelle

Dans le package `java.util.concurrent.locks` se trouve l'interface Lock.

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

On veut implementer l'interface `Lock` par l'algorithme de Peterson pour 2 threads, puis la généraliser à n threads. Ecrire le code d'une thread qui rentre régulièrement en section critique en utilisant un verrou (la section critique et la section non critique pourront être simulé par une mise en sommeil de la thread).

1. Exclusion mutuelle pour 2 threads. On rappelle le code de l'algorithme de Peterson.

```
boolean[] flag= new boolean[2];
int victim;
public Peterson(){
    for(int i=0 ; i<flag.length ; ++i)
        flag[i] = false;
}
public void lock() {
    int i = ThreadID.get() % 2;
    int j = 1 - i;
    flag [i]=true; // I'm interested
    victim = i ; // you go first
    while ( flag[j] && victim == i) {}; // wait
}

public void unlock() {
    int i = ThreadID.get() % 2;
    flag[i]=false; // I'm not interested
}
```

   Ecrire l'implémentation de Lock par cet algorithme.

2. Exclusion mutuelle pour n threads

   Pour implémenter le verrou on propose d'utiliser l'algorithme de Peterson généralisé à n threads dont on rappelle le code ci-dessous:

```
class Filter implements Lock {
     int IDLE = -1;
     int[] level;
     int[] victim;
     int size
  public Filter(int threads) {
    size   = threads;
    level  = new int[threads];
    victim = new int[threads - 1];
```

```
  }
  public void lock() {
    int me = ThreadID.get();
    for (int i = 0; i < size - 1; i++) {
      level[me]  = i;
      victim[i] = me;
      // spin while conflicts exist
      while (sameOrHigher(me, i) && victim[i] == me) {};
    }
    level[me] = size - 1;
  }
  // Is there another thread at the same or higher level?
  private boolean sameOrHigher(int me, int myLevel) {
    for (int id = 0; id < size; id++)
      if (id != me && level[id] >= myLevel) {
        return true;
      }
    return false;
  }
  public void unlock() {
    int me = ThreadID.get();
    level[me] = IDLE;
  }
}
```

Ecrire l'implémentation de Lock par cet algorithme.

En supposant que l'algorithme est utilisé par 10 Threads. Répondre aux questions suivantes (si la réponse est positive donnez un exemple d'exécution (ou indiquez quelle exécution donnerait ce résultat) si la réponse est négative expliquez pourquoi)

La thread d'identifiant id est au niveau i si $level[id] = i$

(a) Peut-il y avoir 10 Threads au niveau 0 simultanément?

(b) Peut-il y avoir 10 Threads au niveau 1 simultanément?

(c) Si il y a au moins 1 thread au niveau 0 et aucune au niveau 1 est ce qu'une des threads de niveau 0 peut passer au niveau 1?

(d) Si il y a au moins 1 thread au niveau 0 et au moins 1 thread au niveau 1 est ce toutes les threads de niveau 0 peuvent passer au niveau 1?

(e) Peut-il y avoir 9 Threads au niveau 1 simultanément?

(f) Peut-il y avoir 9 Threads au niveau 2 simultanément?

(g) Peut-il y avoir 8 Threads au niveau 2 simultanément?

(h) Si une thread accède à la section critique, à quels niveaux sont les autres threads ?

(i) Montrer la propriété suivante: pour tout j, $0 \leq j \leq size - 1$, il y a au plus $n - j$ threads au niveau j.

(j) Montrer que une thread qui demande l'accès à la section critique parviendra en section critique.