Université Paris 7
M2 II
Programmation Répartie

TP1

Rappel Java et Atomicité

1 Class ThreadLocal<T>

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);
        }
}
```

2 Atomicité et méthodes *synchronized*

On considère l'implémentation Java suivante d'une file

```
public class FileConcur {
    final static int QSIZE=20000;
    int head=0;
    int tail=0;
    int[] cell= new int [QSIZE];
    public void enq(int o){
        cell[(tail++)%QSIZE]=o;
    }
    public int deq(){
       return  cell[(head++)%QSIZE];
    }
}
```

Avec

```
public class MyThread extends Thread{
    public FileConcur f;
    public int nb;
    public MyThread( int nb, FileConcur f){
        this.nb=nb;
        this.f=f;
    }
    public void run(){
        System.out.println("Je suis la thread " +ThreadID.get());
        int k=ThreadID.get()+1;
        for (int i=0;i<nb;i++)
            { f.enq(k);
          this.yield();
        }
        int x=0;
        for (int i=0;i<nb;i++)
            {x=x + f.deq();
            this.yield();
            }
        System.out.println("Thread "+ThreadID.get()+ " somme " +x);

}
}

public class Main {
    public static void main(String[] args) {
        FileConcur f= new FileConcur();
        MyThread W= new MyThread(10000,f);
        MyThread R= new MyThread(10000,f);
        W.start();R.start();
        try{R.join();W.join();} catch(InterruptedException e){};
        System.out.println( "file utilisée par deux threads "+
                                          f.head +" tete/fin "+f.tail);

    }
}
```

- Si les instructions cell[(tail++)%QSIZE]=o; et cell[(head++)%QSIZE]; étaient atomiques la classe FileConcur implémenterait-elle correctement une file utilisée par plusieurs threads (on suppose qu'on ne met pas plus de QSIZE éléments dans la file et qu'on enlève uniquement des éléments quand la file n'est pas vide)
- Si les instructions cell[(tail++)%QSIZE]=o; et cell[(head++)%QSIZE];

étaient atomiques quelles seraient les valeurs de *head* et de *tail* affichées lors de l'exécution de Main.main.

- Après exécution de Main.main quelles sont les valeurs de *head* et de *tail* effectivement affichées. Qu'en déduisez vous sur l'atomicité en java.
- L'exécution précédente est elle encore possible si on ajoute "synchronized" lors des définitions de `enq` et `deq` dans FileConcur?

3 Atomicité et blocs *synchronized*

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 value2;//valeur commune
    public MonObjet(int init){
        value=init;value2=init;
        last=new ThreadLocal<Integer>(){
            protected Integer initialValue() {return 0;}};
    };
    public int read(){ return value;}
    public void add( ){
        last.set(new Integer(last.get()+1));
        value=value +1;
        value2=value2 +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++)
        { int x= o.read();
          o.add();
          this.yield();
          }
```

```
        System.out.println("la thread "+ThreadID.get()+
             " valeurs  "+ o.value+" et " +o.value2 +
              " pour sa "+ o.last.get()+" eme ecriture (derniere) ");
}
}
 public class Main2 {
  public static void main(String[] args) {
        MonObjet  o= new MonObjet(0);
        MyThread2 W,R;
        W= new MyThread2(o,1000);
        R= new MyThread2(o,1000);
        W.start();R.start();
        try{R.join();W.join();} catch(InterruptedException e){};

    }
}
```

- Si l'instruction x=x+1 était atomique, qu'elles devraient être les valeurs de o.value et o.value2 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 était-elle atomique en java?
- Comment assurer qu'à la fin du Main2.main on ait o.value=o.value2= nombre totale d'écritures?