Université Paris 7
M2 II
Programmation Répartie

TP3

Concurrence

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);

        }
}
```

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

```
public class MonObjet {
    ThreadLocal<Integer> last;
    int value;
    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(new Integer(last.get()+1));
        value=value +1;
        System.out.println("la thread "+ThreadID.get()+
```

```
                " valeur  "+ value+
                " pour sa "+last.get()+" eme ecriture ");
          }
    }
```

avec

```
public class MyThread extends Thread{
    public MonObjet  o;
    public int nbwrite;
    public MyThread( 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();
          }
}
}
```

```
 public class Main {
  public static void main(String[] args) {
        MonObjet  o= new MonObjet(0);
        MyThread W,R;
        W= new MyThread(o,10);
        R= new MyThread(o,9);
        W.start();R.start();
        try{R.join();W.join();} catch(InterruptedException e){};
    }
}
```

Que se passe-t-il lors de l'exécution de la classe Main? Que se passe-t-il quand on déclare `last` comme `int` (et qu'on l'incrémente à chaque addition)?

2 Atomicité

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(){
    if (ThreadID.get()==2) ThreadID.set(0);
    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 seul= new FileConcur();
        FileConcur deux= new FileConcur();
        MyThread W,R;
        W= new MyThread(10000,seul);
        W.start();
        try{W.join();} catch(InterruptedException e){};
```

```
        System.out.println( "file utilisée par une thread "+
                                            seul.head +" tete/fin "+seul.tail);
        System.out.println();

        W= new MyThread(10000,deux);
        R= new MyThread(10000,deux);
        W.start();R.start();
        try{R.join();W.join();} catch(InterruptedException e){};
        System.out.println( "file utilisée par deux threads "+
                                          deux.head +" tete/fin "+deux.tail);

    }
}
```

Un exemple d'exécution :

```
Je suis la thread 0
Thread 0 somme 10000
file utilisée par une thread 10000 tete/fin 10000

Je suis la thread 1
Je suis la thread 0
Thread 1 somme 15061
Thread 0 somme 14905
file utilisée par deux threads 19993 tete/fin 19977
```

- La classe FileConcur implémente-t-elle correctement une file si elle est utilisée par une thread? par deux threads?
- L'exécution précédente est elle encore possible si on ajoute "synchronized" lors des définitions de `enq` et `deq` dans FileConcur?

3 Consistence

On propose l'implémentation suivante d'un registre SWMR:

```
public class MRSW {
    ThreadLocal<Integer> ind;
    final static int SIZE=100;
    int tab[]=new int[SIZE];
    public MRSW(int init){
        tab[0]=init;
        //init val initiale non nulle; les autres à 0
        ind=new ThreadLocal<Integer>(){
```

```
            protected Integer initialValue() {return 1;}};
    };
    public int read(){
        ind.set(new Integer(SIZE-1));
        while (tab[ind.get()]==0){
            ind.set(new Integer(ind.get()-1));
        }
        return tab[ind.get()];}

    public void write(int x ){
        tab[ind.get()]=x;
        ind.set(new Integer(ind.get()+1));
         }
}
```

On suppose que la lecture et l'écriture d'un élément du tableau est atomique.

- Si on se limite à SIZE-1 écritures, quelle condition de consistence est assurée par cette implementation (quiescente? sequentielle? linearisable?)
- Ecrire une implémentation assurant la même condition de conistence lorsqu'on ne se limite pas à SIZE-1 écritures.
- A quels problèmes est on confronté lorsqu'on veut réaliser un MWMR atomique?