Master 2
Programmation répartie

TP

Objets concurrents (Partie 2)

Exercice 1.— On considère une pile concurrente (i.e qui peut être utilisée par plusieurs threads) d'entier.

1. En vous inspirant de la définition classique d'une pile donné la définition et la spécification séquentielle de cet objet
2. On considère l'implémentation suivante

```
public class PileSync {
    int t[];
    int sommet=-1;
    PileSync(int n){
     t= new int[n];
    }

    synchronized void empiler(int j)
    {sommet=sommet+1;
    t[sommet]=j;
    }

    synchronized int depiler(){
        if (sommet==-1)return -1;
        sommet=sommet-1;
        return t[sommet+1];
    }
}
```

Que l'on peut utiliser avec :

```
public class AjoutThread2 extends Thread{
    public PileSync p;
    int id;
    public AjoutThread2 ( PileSync p, int id){
        this.p=p;
        this.id=id;
    }

    public void run(){
        for(int i=1;i<11;i++)
        {
          p.empiler(i+100*id);
          try{Thread.sleep(10);}
          catch(Exception e) { System.out.println( "probleme" );}
          this.yield();
          }
```

```
        for(int i=1;i<11;i++)
        {
          System.out.println( "Thread  "+id+"  retire " +p.depiler());
          try{Thread.sleep(10);}
          catch(Exception e) { System.out.println( "probleme" );}
          this.yield();
          }
    }
}
=====
public class Main2 {
     public static void main( String [] v)
    {
    Thread[] Th = new Thread[3];
    PileSync f=new PileSync(100);

    for (int i=0;i<3;i++)
    {
         Th[i]= new AjoutThread2(f,i);
         Th[i].start();
    }
   for (int i=0;i<3;i++)
   {    try{Th[i].join();}
         catch(Exception e) {System.out.println( "probleme" );}
   }
}
}
```

Cette implémentation réalise-t-elle une implémentation linéarisable d'une pile ?

3. On supprime les synchronized, l'implémentation réalise-t-elle une implémentation linéarisable d'une pile ?

Exercice 2.— On considère un objet TS. Il est définit par un état interne c initialisé à 1. Il n'a qu'une seule méthode ts() qui retourne un entier. Sa spécification séquentielle est :
{c=1} TS.ts(){c=0; return 1;}
{c=0} TS.ts(){return 0;}

1. Donner un exemple d'exécution de 4 threads partageant un objet TS et faisant chacune 2 appels a ts() de cet objet.
2. A l'aide de synchronized donner une implémentation linéarisable de cet objet
3. En utilisant un AtomicBoolean du package java.util.concurrent.atomic., implémenter un objet TS
4. En utilisant un AtomicInteger du package java.util.concurrent.atomic, implémenter un objet TS

Exercice 3.— On considère un objet Compteur. Il est définit par un état interne c initialisé à 0. Il n'a qu'une seule méthode add() qui retourne un entier. Sa spécification séquentielle est :
{c=x} compteur.add(){c=x+1; return x;}

1. Donner un exemple d'exécution de 4 threads partageant un objet compteur et faisant chacune 2 appels add() de cet objet.
2. On considère l'implémentation suivante

```
public class Compteur {
     int c=0;
     int add(){ c=c+1; return c-1;}

   }
```

Cette implémentation est-ellle linéarisable ?

programme de test :

```
public class AjoutThread3 extends Thread{
    public Compteur p;
    int id;
    public AjoutThread3 ( Compteur p, int id){
        this.p=p;
        this.id=id;

    }
    public void run(){
        for(int i=1;i<11;i++)
        {
         System.out.println( p.add());
          try{Thread.sleep(10);}
          catch(Exception e) { System.out.println( "probleme" );}
          this.yield();
          }

    }
}
==============
public class Main3  {
     public static void main( String [] v)
    {

    Thread[] Th = new Thread[3];
    compteur f=new Compteur();

    for (int i=0;i<3;i++)
    {
         Th[i]= new AjoutThread3(f,i);
         Th[i].start();
    }
   for (int i=0;i<3;i++)
   {    try{Th[i].join();}
         catch(Exception e) {System.out.println( "probleme" );}
   }
 /*    for ( int i=0; i<34;i++) {
            System.out.println(cur.get().value+ " ");
```

```
            cur = cur.get().next;
    }  */
}

}
```

3. On considère l'implémentation suivante

```
public class Compteur {
     TS t[]= new TS[100];
     compteur(){
         for(int i=0;i<100; i++) t[i]=new TS();}
     int add(){
         int i=0;
         while( t[i].ts()==0)i++;
         return i;}

}
```

Cette implémentation est-elle linéarisable (on suppose qu'il y a moins de 100 appels à un objet compteur) ?

Exercice 4.— On considère un objet concurrent $file$ d'entier positif avec les opérations *mettre* et *enlever* Si la file est vide l'opération mettre retourne -1. Il n'y a pas de problème de file pleine (l'espace de stokage est suffisant).

1. Ecrire la spécification séquentielle de cet objet.
2. A l'aide de la classe LinkedBlockingDequeue<E> du package jav.util.concurrent réaliser une implémentation linéarizable de cet objet . Pour tester votre implémentation vous écrirez: le programme d'une classe MyThreadMettre extends Thread qui met dans la file des 10 entiers; le programme d'une thread MyThreadEnleve extends Thread qui enlève de la file des 10 entiers; et un programme principal qui lance 4 threads deux d'entre elles sont des instances de MyThreadMettre et deux sont des instances de MyThreadEnleve (vous pourrez particulariser les éléments mis en utilisant les identités des threads)
3. On propose l'implémentation suivante de $file$

```
public class MaFile {
     AtomicInteger head= new AtomicInteger(0);
     AtomicInteger tail= new AtomicInteger(0);
     int[] items= new int[100];

     public MaFile ( )
     {for ( int i=0; i<99; i++) items[i]=-1;}

     public void mettre ( int z)
     {  // pas de verification mais on suppose z>=0
         int slot;
```

```
          do {slot= tail.get();
             } while (!tail.compareAndSet(slot,slot+1));

         items[slot]=z;


     }
     public int enleve(){
     int value,slot;
     do {   slot=head.get();
            value=items[slot];
             if ( value==-1) return -1;
         } while( !head.compareAndSet(slot,slot+1));
     return value;
     }
}
```

(a) Montrer que quand la file est utilisée par 2 threads qui mettent et enlèvent des éléments dans la file, cette implémentation n'est pas linéarisable.

(b) Montrer que quand la file est utilisée par 2 threads l'une qui met des elements et l'autre qui enlève des elements cette implémentation est linéarisable.

(c) Tester l'implémentation dans ce cas.