Laborator 8
Calcul parallel in java

Reamintim cateva elemente din curs :

1. Un exemplu de clasa imutabila

```
public final class Planet
{
private final double fMass;
private final String fName;
private final Date fDateOfDiscovery;

public Planet (double aMass, String aName, Date aDateOfDiscovery)
        {   fMass = aMass;
        fName = aName;
        fDateOfDiscovery = new Date(aDateOfDiscovery.getTime());
        }
 public double getMass()
        { return fMass;  }
 public String getName()
        {  return fName; }
 public Date getDateOfDiscovery()
        { return new Date(fDateOfDiscovery.getTime()); }
}
```

2. Sincronizarea accesului la campuri “mutable”

```
public final class MutablePlanet
{  public MutablePlanet(int aId, String aName, Date aDateOfDiscovery) {
    fId = aId;
    fName = aName;
    fDateOfDiscovery = new Date(aDateOfDiscovery.getTime());
  }
  public synchronized int getId()
        {  return fId;  }
  public synchronized void setId(int aNewId)
        { fId = aNewId; }
  public synchronized String getName()
        {  return fName; }
  public synchronized void setName(String aNewName)
        {  fName = aNewName;  }
  public synchronized Date getDateOfDiscovery()
        { return new Date(fDateOfDiscovery.getTime());  }
  public synchronized void setDateOfDiscovery(Date aNewDiscoveryDate)
        { fDateOfDiscovery.setTime(aNewDiscoveryDate.getTime());  }
  private int fId;
  private String fName;
  private Date fDateOfDiscovery;
}
```

3. Corectati codul de mai jos pentru a-l putea folosi mai departe in teme

```
public class Buffer
{ private LinkedList objects = new LinkedList();
   public synchronized add( Object x )
        {
        objects.add(x);
        this.notifyAll();
        }
   public synchronized Object remove()
        {
        while (objects.isEmpty())
                {this.wait();   }
        return objects.removeFirst();
        }
}
```

4. Exemplu incomplet de pool

```
public class TaskManager
{  LinkedList taskQueue = new LinkedList();
   List threads = new LinkedList();
   public TaskManager(int numThreads)
        {
        for(int i=0; i<numThreads; ++i)
                {
        Thread worker = new Worker(taskQueue);
        threads.add(worker);
        worker.start();
                }
        }

   public void execute(Runnable task)
        { synchronized(taskQueue)
                { taskQueue.addLast(task);
                 taskQueue.notify();
                }
         }
}
```

Tema 1. Sa se implementeze un program Java care va implementa un scheduler pentru thread pool (ferma de cai pardon de fire de executie) in conformitate cu algoritmul FCFS prezentat la curs)

Tema 2. Pornind de la exemplul de buffer inplementat cu producator consummator din curs (cu eliminarea blocajului) sa se implementeze o coada folosita In transmiterea si receptia de siruri de caractere intre mai multhe thread-uri (un fel de chat unde fiecare client este pe cate un thread)

Tema 3. sa se construiasca un pool de thread-uri care sa faca niste calculi simple (gen sume, inductie etc) sa puna in evidenta hazardul de curse (vez I in curs)

Tema 4. Sa se implementeze un calcul $\sum_0^n i$ cu 4 thread-uri simultane care fiecare calculeaza pe un subinterval folosind modelul master/slave

Tema 5 sa se proceseze calcul $\sum_0^n i$ simultan pentru 4 valori diferite a lui n luate dintr-o coada de catre 4 task-uri diferite (model peer)

Tema 6. Sa se realizeze o procesare dupa model pipeline a unui tablou de intregi. Primul thread din pipe inmulteste toate elemental vector V cu o constanta alpha, urmatorul thread din pipe va ordona vectorul iar final ultimul thread il va afisa in coordinate x si y

Tema 7. Sa se analizeze exemplul de mai jos si sa se puna in evidenta situatiile de lock always, never, sometimes si hostile daca este cazul. Sistemul are evitat blocajul? De ce?

```
import java.util.Random;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;

public class ProducerConsumer {

	private final BlockingQueue<Integer> queue;
	private static final Random rnd = new Random();

	public static void main(String[] args) {
		ProducerConsumer prodconsumer = new ProducerConsumer();
		prodconsumer.init();
	}

	public ProducerConsumer() {
		queue = new LinkedBlockingQueue<>(3);
	}

	public void init() {
		for (int i = 0; i < 100; i++) {
			new Thread(new Producer(), "Producer-1 of iteration "+i).start();
			new Thread(new Producer(), "Producer-2 of iteration "+i).start();
			new Thread(new Producer(), "Producer-3 of iteration "+i).start();

			new Thread(new Consumer(), "Cosumer-1 of iteration "+i).start();
			new Thread(new Consumer(), "Cosumer-2 of iteration "+i).start();
			new Thread(new Consumer(), "Cosumer-3 of iteration "+i).start();
			try {
				Thread.sleep(1000);
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
		}
	}

	private class Producer implements Runnable {
		@Override
		public void run() {
			Integer e = rnd.nextInt(100);
			System.out.println("Inserting Element " + e);
			try {
				queue.put(e);
				Thread.sleep(1000);
```

```
                } catch (InterruptedException e2) {
                        // TODO Auto-generated catch block
                        e2.printStackTrace();
                }

        }
    }

    private class Consumer implements Runnable {

        @Override
        public void run() {
            try {
                System.out.println("Removing Element " + queue.take());
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

        }

    }

}
```

Tema 8. Sa se implementeze mai multe cozi de thread-uri cu prioritati diferite si sa se implemnteze mecanismul de prevenire a infomatarii (cel de imbatranire)

Tema 9. Pentru sincronizarea unor thread-uri sa se implemnteze protocolul cu simularea limitarii prioritatii (Highest locker)

Tema 10. Sa se scrie un program simplu cu thread-uri (pornind de la exemplul din curs) care sa foloseasca atat lock-ul pe instanta cat si cel static

Tema 11. Sa se modifice/corecteze programul de mai jos pentru a introduce inca o clasa C pastrand idea generala

```
class A
{
synchronized void foo(B b)
  {
   String name = Thread.currentThread().getName();
    System.out.println(name + " entered A.foo");
    try
    { Thread.sleep(1000);}
    catch(Exception e)
    {  }
    System.out.println(name + " trying to call B.last()");
    b.last();
  }
  synchronized void last()
  { System.out.println("Inside A.last");}
}
```

```
class B
{
synchronized void bar(A a)
  {
  String name = Thread.currentThread().getName();
   System.out.println(name + " entered B.bar");
    try
    { Thread.sleep(1000); }
    catch(Exception e)
    {   }
 System.out.println(name + " trying to call A.last()");
    a.last();
  }
 synchronized void last()
  { System.out.println("Inside B.last");   }
}

class DeadLock implements Runnable
{
A a = new A();
 B b = new B();
 DeadLock()
            {Thread.currentThread().setName("Main Thread");
   new Thread(this).start();
   a.foo(b);
  System.out.println("Back in the main thread.");
  }
 public void run()
           {Thread.currentThread().setName("Racing Thread");
   b.bar(a);
  System.out.println("Back in the other thread");
  }
public static void main(String args[])
           { new DeadLock(); }
}
```

Tema 12. Sa se modifice exemplul de mai jos ca sa am pool de thread-uri producer cu pool de thread-uri consumer dar care folosesc o singura coada

```
class Queue
{
  int n;
  boolean ValueSet = false;
  synchronized int get()
  {
    try
    {
      if(!ValueSet)
        wait();
    }
```

```
catch(InterruptedException e)
    {
    }
System.out.println("Got:"+n);
    ValueSet = false;
    notify();
    return n;
  }
synchronized void put(int n)
  {
    try  {
      if(ValueSet)
        wait();
    }
    catch(InterruptedException e)
    {  }
    this.n = n;
    System.out.println("Put : "+n);
    ValueSet = true;
    notify();
  }
}
class Producer implements Runnable
{
  Queue Q;
  Producer(Queue q)
  {
    Q = q;
    new Thread( this, "Producer").start();
  }
public void run()
  {
    int i = 0;
    while(true)
      Q.put(i++);
  }
}
class Consumer implements Runnable
{
  Queue Q;
  Consumer(Queue q)
  {
    Q = q;
    new Thread( this, "Consumer").start();
  }
  public void run()
  {
    while(true)
      Q.get();
```

```
  }
}
class PCnew
{
  public static void main(String[] args)
  {
    Queue Q = new Queue();
    new Producer(Q);
    new Consumer(Q);
  }
}
```

Tema pe acasa: Fiecare nava din joc este un thread dintr-un pool. Thread-urile de nave au prioritati diferite si de aceea cand comunica (vezo observer etc) trebuie utilizat algoritmul highest locker