GRIFFITH COLLEGE DUBLIN

Course: Parallel and Distributed Programming
Module: MSCBD-PDP
Semester: Semester I
Assignment Number: 1.
Date of Title Issue: 19 Oct 2022
Assignment Deadline: 20 Nov 2022
Assignment Weighting: 20/50
Please state the assignment title / brief. Please specify details such as:
Answer the questions on the accompanying sheet.
Learning Outcomes
Please state the programme and related module learning outcomes that this assignment is assessing. 1,2,4,5,6
Assessment Criteria

Please state the assessment criteria applied to this assignment, such as: Correctness of the work. Presentation, including compliance with the specified file format. Evidence of critical thinking and analysis. Originality, quality and thoroughness of the work. Research correct academic approach. Proper treatment of sources.

Academic Dishonesty: All of your assignments need to represent your own effort. Assignments should be done without consultation with other students and you should not share your source code with others. Any assignment submitted that is essentially the same, as someone else's will not be accepted. **ALL matching assignments will receive 0 credits.**

Question 1 (30%)

Given below is a generic class CircularQueue<T> that implements a queue modelled by a circular array. There are two constructors a default one that sets the size to an arbitrary value of 20 and one that takes the maximum size of the queue as an argument n. In both cases head and tail are set to zero. This class is not thread safe. Please note that a queue simply manages the order of insertion and removal, it does not interact in any way with the data under its control. Your task is to re-write it so that it is thread safe and solves issues that may arise around iteration.

```
class CircularQueue<T> implements Iterable<T>{
   private T queue[];
   private int head, tail, size;
   public CircularQueue(){
     queue = (T[])new Object[20];
     head = 0; tail = 0; size = 0;
   }
   public CircularQueue(int n){ //assume n >=0
     queue = (T[])new Object[n];
     size = 0; head = 0; tail = 0;
   }
   public boolean join(T x){
      if(size < queue.length){
       queue[tail] = x;
       tail = (tail+1)%queue.length;
       size++;
       return true;
      }
      else return false;
   }
   public T top(){
      if(size > 0)
       return queue[head];
      else
       return null;
   }
   public boolean leave(){
      if(size == 0) return false;
      else{
       head = (head+1)%queue.length;
       size--;
       return true;
      }
   }
   public boolean full(){return (size == queue.length);}
   public boolean empty(){return (size == 0);}
  public Iterator<T> iterator(){
      return new QIterator<T>(queue, head, size);
    }
   private static class QIterator<T> implements Iterator<T>{
      private T[] d; private int index;
     private int size; private int returned = 0;
     QIterator(T[] dd, int head, int s){
       d = dd; index = head; size = s;
      }
      public boolean hasNext(){ return returned < size;}
      public T next(){
         if(returned == size) throw new NoSuchElementException();
         T item = (T)d[index];
         index = (index+1) % d.length;
         returned++;
        return item;
       }
       public void remove(){}
   }
}
```

Question 2 (30%)

A thread safe class that implements a Hash-table is given. This class uses coarse-grained locking to handle concurrent access for threads sharing an instance of the class. Your task is to implement a fine-grained solution for this class that optimizes concurrent access for threads. The class ConcurrentHashList is given as an appendix at the end of the document. **Keep using ReentrantLock in your fine-grained solution.**

```
import java.util.*;
import java.util.concurrent.locks.*;
public class ConcurrentHashList<E> implements Iterable<E>{
    private LinkedList<E> data[];
    private Lock lock = new ReentrantLock();
    @SuppressWarnings("unchecked")
    public ConcurrentHashList(int n){
        if(n > 1000)
          data = (LinkedList<E>[])(new LinkedList[n/10]);
        else
        data = (LinkedList<E>[])(new LinkedList[100]);
     for(int j = 0; j < data.length;j++)
             data[j] = new LinkedList<E>();
}
@SuppressWarnings("unchecked")
public ConcurrentHashList(Collection<? extends E> cl){
     if(cl.size() > 1000)
       data = (LinkedList<E>[])(new LinkedList[cl.size()/10]);
     else
             data = (LinkedList<E>[])(new LinkedList[100]);
     for(int j = 0; j < data.length;j++)
             data[j] = new LinkedList<E>();
     for(E x : cl) this.add(x);
}
private int hashC(E x){
     int k = x.hashCode();
     int h = Math.abs(k % data.length);
     return(h);
}

public void add(E x){
  if(x != null){
    lock.lock();
    try{
               int index = hashC(x);
               if(!data[index].contains(x))
               data[index].add(x);
       }finally{lock.unlock();}
     }
}
public boolean contains(E x){
     if(x == null) return false;
     lock.lock();
     try{
       int index = hashC(x);
       return(data[index].contains(x));
     }finally{lock.unlock();}
}
public boolean remove(E x){
 if(x == null) return false;
 lock.lock();
 try{
   int index = hashC(x);
   return data[index].remove(x);
 }finally{lock.unlock();}
}

public String toString(){
  lock.lock();
  try{
       StringBuffer s = new StringBuffer(this.size());
       s.append('<');
       int ind = 0;
       while(ind < data.length){
                 Iterator<E> it = data[ind].iterator();
                 while(it.hasNext())
                           s.append(it.next()+", ");
                 ind++;
       }
       s.deleteCharAt(s.length()-1);
       s.setCharAt(s.length()-1,'>');
       return s.toString();
     }finally{lock.unlock();}
}
public int size(){
  lock.lock();
```

```
      try{
           int j = 0;
           for(LinkedList<E> lst : data) j += lst.size();
           return j;
         }finally{lock.unlock();}
    }

    public Iterator<E> iterator(){
      lock.lock();
      try{
        ArrayList<E> items = new ArrayList<E>();
        int ind = 0;
        while(ind < data.length){
                    Iterator<E> it = data[ind].iterator();
                    while(it.hasNext())
                            items.add(it.next());
                    ind++;
        }
        return items.iterator();
      }finally{lock.unlock();}
    }
}
```

Question 3 (40%)

A platform has space for at most 100 people at any one time. People are only admitted when the platform is open and the number of persons does not exceed the prescribed limit. Using condition variables write a class that could be used to control access to the platform. By creating multiple threads to represent people accessing the platform write a simulator for your control.