```java
//Lucas Miranda Lin – lml
//1
import java.util.Vector;

public class VetorSeguro {
    private Vector<Integer> v;

    public VetorSeguro() {
        v = new Vector<Integer>();
    }

    public synchronized int get (int index) {
        return v.get(index);
    }

    public synchronized int set (int index, int element) {
        return v.set(index, element);
    }

    public synchronized void swap (int index1, int index2) {
        int temp = this.get(index1);
        this.set(index1, this.get(index2));
        this.set(index2, temp);
    }
}

//2
public class FilaBloqueante {
    private class Node {
        Node next;
        int val;

        Node(int i) {
            next = null;
            val = i;
        }
    }

    private Node head;
    private Node tail;

    public FilaBloqueante() {
        head = null;
        tail = null;
    }

    public synchronized void put(int elem) {
        if(head == null) {
            head = new Node(elem);
            tail = head;
        } else {
            tail.next = new Node(elem);
            tail = tail.next;
        }
    }

    public synchronized int take() {
        int i = head.val;
        head = head.next;
        return i;
    }
}
```

```
//3
import java.util.LinkedList;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;

public class Padaria {
    private static final int tempoAssar = 2000;
    private static final int tempoAbastecer = 200;
    private static int paesAssados = 0;

    private class Pao {
        long tempoDeRetirar;
        public Pao(long tempoPosto) {
            tempoDeRetirar = tempoPosto + tempoAssar;
        }
    }

    private BlockingQueue<Pao> Forno;

    public Padaria() {
        Forno = new ArrayBlockingQueue<Pao>(50);
    }

    private class Assar extends Thread {
        public void run() {
            while(true) {
                //aguarda até o forno não estar vazio
                synchronized(Forno) {
                    while(Forno.isEmpty()) {
                        try {
                            Forno.wait();
                        } catch (InterruptedException e) {
                            break;
                        }
                    }
                }

                //aguarda até o próximo pão assar
                Pao p = Forno.peek();
                long tempoAtual = System.currentTimeMillis();
                if (tempoAtual < p.tempoDeRetirar) {
                    try {
                        Thread.sleep(p.tempoDeRetirar - tempoAtual);
                    } catch (InterruptedException e) {}
                }

                //retira um pão assado
                try {
                    Forno.take();
                } catch (InterruptedException e) {}
                ++paesAssados;
                if(paesAssados%10 == 0) {
                    System.out.println(paesAssados + " paes assados");
                }

                //notifica o abastecedor quando o forno esvazia
                if(Forno.isEmpty()) {
                    synchronized(Forno) {
                        Forno.notify();
                    }
                }
            }
```

```
        }
    }

    private class Abastecer extends Thread {
        public void run() {
            while(true) {
                //aguarda até o forno estar vazio
                synchronized(Forno) {
                    while(!Forno.isEmpty()) {
                        try {
                            Forno.wait();
                        } catch (InterruptedException e) {
                            break;
                        }
                    }
                }

                //abastece o forno com 50 pães
                for(int i = 0; i < 5; ++i) {
                    //abastece o forno com um lote de 10 pães
                    LinkedList<Pao> paes = new LinkedList<Pao>();
                    Long tempoPosto = System.currentTimeMillis();
                    for(int j = 0; j < 10; ++j) {
                        paes.add(new Pao(tempoPosto));
                    }
                    boolean estavaVazio = Forno.isEmpty();
                    Forno.addAll(paes);
                    System.out.println("forno abastecido com 10 pães");

                    //notifica o assador quando o forno não está mais vazio
                    if(estavaVazio) {
                        synchronized(Forno) {
                            Forno.notify();
                        }
                    }

                    //aguarda o tempo de abastecimento
                    try {
                        Thread.sleep(tempoAbastecer);
                    } catch (InterruptedException e) {}
                }
            }
        }
    }

    public static void main (String[] args) {
        Padaria massaNobre = new Padaria();
        new Thread(massaNobre.new Assar()).start();
        new Thread(massaNobre.new Abastecer()).start();
    }
}
```

```
--4
import Control.Concurrent.STM
import Control.Concurrent.STM.TVar

type Buffer a = TVar [a]

newBuffer :: a -> IO (Buffer a)
newBuffer x = newTVarIO [x]

put :: Buffer a -> a -> STM()
```

```
put buffer elem = do
    buf <- readTVar buffer
    writeTVar buffer (buf ++ [elem])

isNotEmpty :: [a] -> Bool
isNotEmpty [] = False
isNotEmpty _ = True

get :: Buffer a -> STM a
get buffer = do
    tempbuf <- readTVar buffer
    check(isNotEmpty tempbuf)
    (x:buf) <- readTVar buffer
    writeTVar buffer buf;
    return x
```