1.

monitor Contador {

private int i = 0;

synchronized void incrementar() {

i++;

notifyAll();

}

synchronized void decrementar() throws InterruptedException {

i--;

notifyAll();

}

}

2.

monitor Semaforo {

private List<Thread> waitingThreads = new ArrayList<>();

synchronized void release() {

waitingThreads.add(Thread.currentThread());

notify();

}

synchronized void acquired() {

if (waitingThreads.isEmpty()) {

wait();

}

Thread threadToRelease = waitingThreads.remove(0);

notify(threadToRelease); // Despierta al primer hilo en la lista

}

}

Si, se libera en orden FIFO por lo cual garantiza que todos los hilos se ejecuten.

3.

a)

monitor Fifo {

private int capacidad;

private int i;

private List<Integer> buffer = new ArrayList<Integer>();

public Fifo(int capacidad) {

this.capacidad = capacidad;

i = 0;

}

public synchronized Integer read() {

while(buffer.isEmpty()) {

wait();

}

int readNumer = buffer.remove(0);

i--;

notifyAll();

return readNumer;

}

public synchronized void write(int valor) {

while(i == capacidad) {

wait();

}

buffer.add(valor);

i++;

notifyAll();

}

}

b)

class Productor extends Thread {

private int n;

private BufferFifo buffer;

Productor(BufferFifo buffer, int n) {

this.n = n;

this.buffer = buffer;

}

@Override

public void run() {

for(int i = 0; i < n; i++) {

buffer.add(i);

}

}

}

c)

class Consumir extends Thread {

private BufferFifo = buffer;

Consumir(BufferFifo buffer) {

this.buffer = buffer;

}

@Override

public void run() {

for(int i = 0; i < buffer.size; i++) {

System.out.print(buffer.read(i));

}

}

}

d)

class Main {

public static void main(Strings[] args) {

BufferFifo buffer = new BufferFifo(2);

Productor productor = new Productor(buffer, 5);

Consumir consumidor = new Consumirdor(buffer);

productor.start();

consumidor.start();

}

}

4)