Programación Concurrente 2018
Trabajo Práctico Nº 4

1. Considere la situación siguiente: En una pequeña empresa que tiene 1 jefe y 5 empleados se acostumbra que el jefe sea el primero en saludar y lo haga solo cuando han llegado todos sus empleados. Además cada empleado debe responder el saludo al jefe

a) el código siguiente no logra lo indicado, indique claramente cual es el problema y corríjalo

b) modifique el código de manera de *mejorarlo,* para que "el jefe" no este en espera activa.

c) Reorganícelo como lo crea mas conveniente para mejorarlo desde el punto de vista de la Programación orientada a objetos.

```
public class Personal extends Thread{

        private String nombre;
        private Saludo saludo;
        private boolean esJefe;
        static int llegaron = 0;
        private int numEmp;

        Personal (Saludo s, String n) {
                esJefe = false;
                nombre = n;
                saludo = s;
        }

        Personal (Saludo s, String n, int x) {
                esJefe = true;
                nombre = n;
                saludo = s;
                numEmp = x;
        }
```

```
		public void run() {
			System.out.println("("+nombre+" llega)");
			if (esJefe){
				while (llegaron < numEmp) {
					System.out.println("(Esperando...)"); }
				saludo.saludoJefe();
			} else {
				synchronized(this) {
					llegaron++;
				}

				saludo.esperarJefe(nombre);
			}
		}
	}

public class Saludo {
	synchronized void esperarJefe(String empleado) {
		try {
			wait();
			System.out.println(empleado + "> Buenos dias
				jefe!");
		} catch (InterruptedException e) {
			System.out.println(e.toString());
		}
	}

	synchronized void saludoJefe() {
		System.out.println("JEFE> Buenos dias!");
		notify();
	}
}

public class Ejercicio1 {
```

```
        public static void main(String argv[]) {
                String[] nombresEmpleados = {"Pablo", "Luis", "Andrea",
                                        "Pedro", "Paula"};
                Saludo hola = new Saludo();
                Personal[] elPersonal= new Personal[6];
                elPersonal[0] = new Personal(hola, "JEFE", 5);
                for (int i=1; i<6; i++)
                        elPersonal[i] = new Personal(hola,
                                        nombresEmpleados[i-1]);
                for (int i=0; i<6; i++)
                        elPersonal[i].start();
                try {
                        for (int i=0; i<6; i++)
                                elPersonal[i].join();
                } catch (Exception e) {
                System.out.println(e);
                }
        }
}
```

2

Se quiere desarrollar un sistema para controlar la temperatura y el número de personas que se encuentran en una sala de un museo. En condiciones normales, se permiten 50 personas en la sala. Si la temperatura sube por encima de un umbral (tUmbral = 30), se limita el número de personas a 35. Si cuando se detecta este suceso el número de personas en la sala es mayor que 35, no es necesario desalojarlas.

Si una persona jubilada intenta entrar, tendrá prioridad frente al resto de personas que estén esperando.

Cada persona se representa mediante una hebra. Además, hay una hebra que mide periódicamente la temperatura de la sala y notifica su valor al sistema. Se pide desarrollar un monitor (GestorSala) que sincronice a las hebras que representan personas y a la hebra que mide la temperatura, de acuerdo con las especificaciones anteriores.

Utilizar monitor o locks, proporcionando los siguientes métodos:

```
... void entrarSala()
// se invoca cuando una persona quiere entrar en la sala.

... void entrarSalaJubilado()
// se invoca cuando una persona jubilada quiere entrar en la sala.

... void salirSala()
// se invoca cuando una persona, jubilada o no, quiere salir de la sala.

... void notificarTemperatura(int temperatura)
// lo invoca la hebra que mide la temperatura de la sala para indicar el
último valor medido.
```

No es necesario garantizar que el orden de acceso a la sala coincide con el orden de llegada a la puerta de entrada.

3. En un **centro de impresión** se cuenta con dos tipos distintos de impresoras para dar servicio a los usuarios: impresoras de tipo A e impresoras de tipo B. Obviamente, el número de impresoras de cada tipo es limitado: NumA impresoras de tipo A y NumB impresoras de tipo B. Para imprimir un trabajo en una impresora de un tipo determinado (A o B) es necesario hacer la solicitud indicando el tipo de impresión requerida. A la hora de imprimir los trabajos se pueden considerar tres grupos distintos de procesos usuarios:

1. Los que requieren una impresora de tipo A.
2. Los que requieren una impresora de tipo B.
3. Los que pueden utilizar una impresora de uno cualquiera de los dos tipos.

Los hilos usuarios generan trabajos y los imprimen. Como restricción: dos hilos no pueden ejecutar simultáneamente las operaciones de impresión (Imprimir A o Imprimir B) sobre una misma impresora.
Cuando un usuario quiera imprimir un trabajo deberá hacerlo sobre una impresora compatible con él y que no esté siendo utilizada por otro usuario. En otro caso el proceso deberá esperar.

4. Considere el problema del Productor/Consumidor, realice el diagrama de estado correspondiente y diseñe una solución general que modele la situación, identificando los

objetos activos y pasivos.

Considerando los casos siguientes:

considerando un buffer limitado de tamaño n

considerando un buffer ilimitado

Debe garantizarse la gestión consistente de los elementos. Debe considerar el caso en que los elementos sean consumidos en el orden en que son agregados a la estructura.

Considere resolverlo con

a) monitores

b) semáforos binarios