Métodos sincronizados y lock

1.- Supón que hay un río cerca de la Escuela de Informática que la separa de un centro de ocio para los estudiantes. Hay estudiantes de dos tipos, los que utilizan móviles **Android**, y los que utilizan **IPhones**. Para cruzar el río, existe una barca que tiene 4 asientos, y que no se mueve hasta que no está completa (hay exactamente 4 estudiantes subidos en ella). Para garantizar la seguridad de los pasajeros, no se permite que haya un estudiante **Android** con tres estudiantes **IPhones**, ni un estudiante **IPhone** con tres **Android**. Cualquier otra configuración de estudiantes en la barca es segura. Implementa una solución a este problema **utilizando métodos sincronizados** para gestionar la sincronización, teniendo en cuenta que deben satisfacerse las dos siguientes condiciones de sincronización:

-CS1: Un estudiante no puede subirse en la barca hasta que no hay algún asiento libre en la barca y la configuración para él es segura.

-CS2: Un estudiante no puede bajarse de la barca hasta que no se ha terminado el viaje.

El esqueleto de la solución está en el campus virtual. Los estudiantes Android/IPhone llaman al método **public void android(int id)/public void iphone(int id)** del objeto barca cuando quiere cruzar el río. Para simplificar el problema, se supone que el último estudiante que sube a la barca (el que ocupa el cuarto asiendo) es el que maneja la barca y la lleva al otro extremo del río, es decir, este estudiante es el que decide cuando se ha terminado el viaje, y avisa al resto de los ocupantes de la barca para que se bajen. NOTA: No hay que preocuparse por la orilla desde la que los estudiantes se suben/bajan de la barca.

Implementa una segunda **versión con Lock y condiciones** para gestionar de forma eficiente el despertar de las hebras que están en la barca y las que están esperando para entrar.

2.- Se dispone de una sala donde distintas parejas tienen citas, para ello un hombre y una mujer tienen que encontrarse (sincronizarse) en la sala. Toda la sincronización del sistema se realiza en un objeto Parejas. Cuando el hombre id quiere tener una cita llama al método *llegaHombre(int id),* y lo mismo para la mujer id que llama a *llegaMujer(int id)* cuando quiere tener una cita.

Las condiciones de sincronización del sistema son:

- Un hombre no puede entrar en la sala, si ya hay otro dentro
- Un hombre tiene que esperara en la sala, si no hay ya dentro una mujer
- Las condiciones de sincronización para las mujeres son las mismas.

Implementa una segunda **versión con Lock y condiciones** para gestionar de forma eficiente el despertar de las hebras que están esperando en la puerta y la que está esperando en la sala de citas.

3.- Varios procesos (N) compiten por utilizar unos cuantos recursos Rec en exclusión mutua. El uso correcto de los recursos es gestionado por una clase **Control**. Cada proceso pide un numero de recursos llamando al método **qrecursos(id,num),** donde **id** es el identificador del proceso que hace la petición y **num** es un número entero que especifica el número de recursos que pide el proceso. Después de usar los recursos, el proceso los libera ejecutando el método **librecursos(id,num).** Para asignar los recursos **Control** utiliza la técnica FCFS (First Come, First Serve) que significa que los procesos son servidos siempre en el orden en el que han realizado sus peticiones. Así un proceso que pide **num** recursos debe esperarse si hay otros procesos esperando aún cuando existan **num** recursos disponibles en el sistema. Implementa la clase **Control**, y varias hebras usuarios que realicen las peticiones a **Control.**

Implementa una segunda **versión con Lock y condiciones** para gestionar de forma eficiente la gestión de turnos y otro para la espera de recursos.