Exercicis Monitors.
Aplicacions i Serveis Telemàtics

GrupEx. En un sistema hi ha M threads. Es vol accedir a un recurs compartit per $n << M$ threads alhora. Per això es dissenya una classe, que s'ha completar mitjançant monitors:

```
public class GroupEx {
  ...
  // constructor, n és el nombre de threads que accedeix al recurs
  public GroupEx(int n) { ... }

  // si el recurs no esta sent usat i és l'últim thread dels n que
    demanen accedir,
  // el grup d'n threads passarà a usar el recurs, sinó s'espera
  public void enter() { ... }

  // deixa d'usar el recurs, si és l'ultim permet que altres n threads
  // puguin accedir al recurs
  public void exit() { ... }
}
```

Broadcast

Es vol realitzar un mecanisme de pas de valors des d'un procés productor a N processos consumidors. El procés productor (podeu suposar que només n'hi ha un) invoca `putValue(v)` per indicar el valor a transmetre als consumidors, i els consumidors invoquen `getValue(id)` per obtenir el valor del productor (el paràmetre `id` és l'identificador del consumidor de 0 a $N-1$). Si el valor del productor encara no ha sigut consumit per tots els consumidors aleshores la següent invocació a `putValue` haurà d'aturar el productor (podeu considerar el mecanisme com un *buffer* de capacitat 1).

Suggeriment - Fer servir el següent esquema:

```
public class BufferBroadcastUn {
    protected Object espai;

    protected boolean[] disponible;
    // disponible[i] indica que el i-essim consumidor
    // no ha consumit l'objecte actual en espai
    ...
    altres atributs, constructor i mètodes
}
```

Pont Llevadís

Sobre un riu hi ha un pont llevadís que s'obre per deixar passar vaixells. Múltiples vaixells poden passar alhora quan el pont està aixecat i múltiples cotxes poden passar alhora quan el pont està abaixat.

Se suposa que hi ha un procés Control que activa una senyal cada vegada que el pont ha de canviar d'estat. Aquesta senyal es visible tant pels cotxes com pels vaixells.

Es demana implementar un monitor que modeli el comportament del pont.

```
public class Pont . . .{

. . .

public void entrar(char tipus){
        . . .
    }

    public void sortir(char tipus){
        . . .
    }

    public void canviar(){
        . . .
    }

}
```

Dobles mixtes de tennis.

Un seguit de noies i nois estàn intentant agrupar-se en grups de 2 nois i 2 noies per jugar un partit de dobles mixtes tennis. Cada noia crida a un mètode `noiaPreparada()` quan vol començar a jugar. De la mateixa manera cada noi crida `noiPreparat()`.

Per a la simulació es defineix una classe `DoblesMixtes` que implementa `noiaPreparada()` i `noiPreparat()`. Una noia ha d'esperar en `noiaPreparada()` fins que una altra noia també hagi fet la crida a `noiaPreparada()` i dos nois hagin fet la crida a `noiPreparat()`. Pels nois la condició d'espera és la mateixa amb els canvis evidents.

Implementar la classe `DoblesMixtes` fent servir **Monitors**.

```
public class DoblesMixtes {
    . . .

    public DoblesMixtes() { . . . }

    public void noiaPreparada() { . . . }
    public void noiPreparat() { . . . }

}
```

Las classes `Noia` i `Noi` consisteixen en:

```
public class Noia implements Runnable {

    protected DoblesMixtes dobles;

    public H(DoblesMixtes db){ dobles = db; }

    public void run(){
        dobles.noiaPreparada();
    }
}

public class Noi implements Runnable {

    protected DoblesMixtes dobles;

    public H(DoblesMixtes db){ dobles = db; }

    public void run(){
        dobles.noiPreparat();
    }
}
```

One Lane Bridge.

Per un pont poden circular cotxes, que es modelen com a threads, en un sentit o en l'altre però no en ambdós sentits a la vegada. Per això es dissenya un monitor `Bridge`. El monitor `Bridge` té definits dos mètodes: `entrar(boolean sentitMeu)` i `sortir(boolean sentitMeu)` . En `entrar(boolean sentitMeu)`, si el pont està sent accedit per fils en sentit contrari al seu, aquest thread s'haurà d'aturar.

Resoldre:

1. sense tenir en compte cap criteri de justicia a l'hora d'accedir al pont.
2. tenint compte el següent criteri de justicia. Si hi ha cotxes passant en un sentit i hi ha cotxes esperant en l'altre sentit, els cotxes que volen accedir al pont en el sentit actual hauran d'esperar. Un cop hagin sortit tots els que estan passant, passaran els del sentit contrari.

La Barberia.

Suposem una barberia amb unes quantes cadires de tallar cabells i dues portes, una d'entrada i una de sortida. La sincronització és la següent:

- Quan no hi ha ningú a la barberia, el barber dorm en la cadira de tallar cabells.
- Quan un client arriba i troba el barber dormint, el desperta, seu ell en una cadira de tallar cabells i dorm mentre el barber li talla els cabells.
- Si quan arriba un client, el barber està ocupat, el client dorm en una sala d'espera.

- Després de tallar el cabell, el barber obre la porta de sortida perque surti el client i **un cop ha sortit** la tanca.
- Si hi ha clients esperant, el barber en desperta un i espera que el client segui a la cadira de tallar cabells.

Es demana implementar mitjançant monitors la classe `Barberia` que disposa de 3 mètodes: `demanarTall`, `demanarClient` i `tallAcabat`. Cada client crida `demanarTall`, el mètode acaba un cop li han tallat els cabells **i ha sortit per la porta**. El barber va iterant les següents crides: crida `demanarClient` per esperar que un client s'assegui a la cadira de tallar cabells, llavors talla els cabells, i finalment crida `tallAcabat` per permetre al client sortir per la porta.

Productor-Consumidor amb ordre.

Implementar la classe `BufferOrdre` on les accions de `put` i `get` es fan en ordre d'arribada. L'ordre es entre operacions `put` per un cantó i entre operacions `get` per un altre.

```
public class BufferOrdre{

    ...

    public void put(Object elem){ ... }

    public Object get(){ ... }
}
```