Monitoreak eta baldintzen sinkronizazioa (I)

Monitoreak eta baldintzen sinkronizazioa

Kontzeptuak:

Monitoreak: Kapsulatutako datuak + atzipen prozedurak
Elkar-bazterketa + baldintzen sinkronizazioa
Monitorean atzipen prozedura aktibo bakarra

Monitore habiaratuak

Ereduak:

Ekintza babestuak

Implementazioa:

- Datu pribatuak eta metodo sinkronizatuak (elkar-bazterketa).
- wait(), notify() eta notifyAll() baldintzen sinkronizaziorako.
- Monitorean hari aktibo bakarra une bakoitzean.

5.1 Baldintzen sinkronizazioa: aparkalekuaren adibidea

Aparkaleku baterako kontrolatzaile bat behar dugu.

Honek utziko du kotxeak sartzen beteta ez badago, eta ez du utziko kotxerik ateratzen aparkalekuan ez badago kotxerik.

Hari batek simulatuko du kotxeen sartzea (sarrerako atea)

eta beste batek kotxeen irtetzea (irteerako atea).

```
sartu           [*    ]
sartu           [**   ]
        irten   [*    ]
sartu           [**   ]
sartu           [***  ]
        irten   [**   ]
sartu           [***  ]
        irten   [**   ]
        irten   [*    ]
sartu           [**   ]
sartu           [***  ]
        irten   [**   ]
sartu           [***  ]
        irten   [**   ]
sartu           [***  ]
        irten   [**   ]
sartu           [***  ]
        irten   [**   ]
sartu           [***  ]
sartu           [**** ]
        irten   [***  ]
        irten   [**   ]
        irten   [*    ]
        irten   [     ]
```

Aparkalekuaren eredua

- Interesa duten ekintzak (gertaerak)?

  sartu eta irten

- prozesuak identifikatu.

  sarrerak, irteerak eta kontrolatzailea

- Egitura definitu: prozesu bakoitza eta elkarrekintzak.

Aparkalekuaren eredua

```
const Plazak = 4
range R = 0..Plazak

SARRERAK = (sartu->SARRERAK).
IRTEERAK = (irten->IRTEERAK).

KONTROLATZAILEA = KONTROL[0],
KONTROL[kop:R] =( when(kop<Plazak) sartu->KONTROL[kop+1]
                 |when(kop>0)      irten->KONTROL[kop-1]
                ).
||APARKALEKUA =
 (SARRERAK||KONTROLATZAILEA||IRTEERAK).
```

Ekintza babestuak erabiltzen dira kontrolatzeko: **sartu** eta **irten**.

LTS?

Hariak eta monitoreak

- Eredua

  entitate guztiak ekintzen bidez elkarregiten duten prozesuak dira

- Programa

  identifikatu behar ditugu hariak eta monitoreak:

  - haria: (output) ekintzak abiarazten dituen entitate **aktiboa**
  - monitorea: (input) ekintzei erantzuten dion entitate **pasiboa**

Implementatzen aparkalekua

Sarrerak eta **Irteerak** implementatzen dute **Thread**

Kontrolatzailea–k kontrola ematen du (baldintzen sinkronizazioa)

Pantaila-k trazaren idazketa kontrolatzen du

AparkalekuaApp aplikazioaren **main** metodoak haien instantziak sortzen ditu :

```
class AparkalekuaApp{
    final static int Plazak = 4;
    public static void main (String args[]) {
     Pantaila pant = new Pantaila(Plazak);
     Kontrolatzailea k = new Kontrolatzailea (Plazak,pant);
     Sarrerak sar = new Sarrerak(k);
     Irteerak irt = new Irteerak(k);
     sar.start();
     irt.start();
     }
}
```

Sarrerak eta Irteerak hariak

```
class Sarrerak extends Thread {
  Kontrolatzailea aparkalekua;
  Sarrerak(Kontrolatzailea k){
     aparkalekua = k;
  }
  public void run() {
    try { while(true) {
            sleep((long)(Math.random()*1000));
            aparkalekua.sartu();
          }
    } catch (InterruptedException e){}
  }
}
```

Irteerak antzekoa egingo du baina **aparkalekua.irten()** deitzen.

Nola implementatzen dugu **Kontrolatzailea**-k egin beharreko kontrola?

Kontrolatzailea monitorea

```
class Kontrolatzailea {
  private int kop;
  private int plazak;

  Kontrolatzailea(int p)
    {plazak=p; kop=0;}

  synchronized void sartu() {
        …
        ++kop;
        …
        }

  synchronized void irten() {
        …
        --kop;
        …
        }
}
```

elkar-bazterketa lortzeko: **`synchronized`** *metodoak*

Baldintzen sinkronizazioa?

- *beteta baldin badago blokeatu:* *!(kop<plazak)*
- *hutsik baldin badago blokeatu:* *!(kop>0)*

Baldintzen sinkronizazioa Java-n

Java-k ematen du **ixaron-ilara (wait queue)** hari bat monitore bakoitzarentzat (objektu bakoitzarentzat), ondoko metodoekin:

`public final void notify()`

Objektu honen itxaron-ilaran zain dagoen hari bakar bat esnatzen du.

`public final void notifyAll()`

Objektu honen itxaron-ilaran zain dauden hari guztiak esnatzen ditu.

`public final void wait()`
`throws InterruptedException`

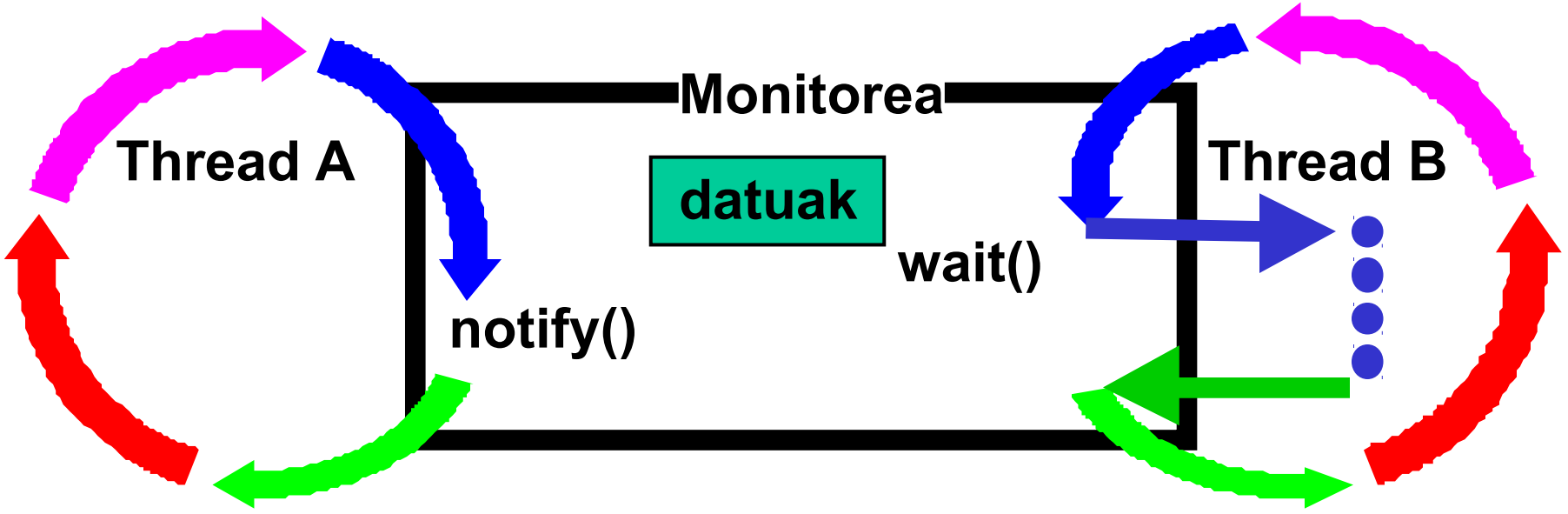
Itxaron beste hari batek jakinarazi (**`notify`**) arte.
Zain gelditzen den hariak askatzen du monitoreari dagokion blokeoa.
`Notify` egiten denean, hariak monitorea berriro hartu arte itxaron.

Baldintzen sinkronizazioa Java-n

Esaten dugu

- hari bat ***sartzen*** dela monitore batean,
  monitoreari dagokion elkar-bazterketa blokeoa eskuratzen duenean.
- eta hari bat ***ateratzen*** dela monitoretik
  blokeoa askatzen duenean.

wait() – haria monitoretik ateratzea eragiten du,
beste hariei monitorean sartzeko aukera ematen.



Baldintzen sinkronizazioa Java-n

```
FSP:   when bald ekintza -> EGOERABERRIA
```

```
Java:  public synchronized void ekintza()
                    throws InterruptedException
       {
         while (!bald ) wait();
         // aldatu monitorearen datuak
         notifyAll()
       }
```

`while` bigiztan: *bald* baldintza ebaluatzen da, ziurtatu dadin *bald* betetzen dela monitorean berriz sartzean.

`notifyAll()`: Zain egon daitezkeen beste haria(k) esnatzen d(it)u, monitorean sartzeko orain, monitorearen datuak aldatu egin direlako.

Kontrolatzailea - Baldintzen sinkronizazioa

```
class Kontrolatzailea {
  private int kop;
  private int plazak;
  private Pantaila pantaila;

  Kontrolatzailea(int pl, Pantaila pant)
    {plazak=pl; pantaila=pant; kop=0;}

  synchronized void sartu() throws InterruptedException {
    while (!(kop<plazak)) wait();
    ++kop;
    pantaila.sartu(kop);
    notify();
  }
  synchronized void irten() throws InterruptedException {
    while (!(kop>0)) wait();
    --kop;
    pantaila.irten(kop);
    notify();
  }
}
```

Pantaila – Trazaren idazketa

```
class Pantaila {
     private int plazak;
     public Pantaila(int p) {
         plazak=p;
     }
     synchronized public void sartu(int k) {
        int kop=k;
         System.out.print("sartu\t\t");
         margotuKotxeak(kop);
     }
     synchronized public void irten(int k) {
         int kop=k;
         System.out.print("\tirten\t");
         margotuKotxeak(kop);
     }
     synchronized public void margotuKotxeak(int k) {
         System.out.print("[");
         for (int i=0; i<k;       ++i) { System.out.print("*"); }
         for (int i=k; i<plazak; ++i) { System.out.print(" "); }
         System.out.println("]");
     }
}
```

Laburpena: Prozesuaren eredutik -> Java-ko monitorera

Entitate **aktiboak** (ekintzak hasten dituztenak) **hariekin (threads)** inplementatzen dira.

Entitate **pasiboak** (ekintzei erantzuten dietenak) **monitorekin** inplementatzen dira.

Ereduko ekintza babestu bakoitza
`synchronized` metodo batekin inplementatzen da.

Metodo honek babesa inplementatzeko
while begizta bat eta `wait()` erabiltzen ditu.

Begiztako baldintza
ereduko babesaren baldintzaren ezeztapena da.

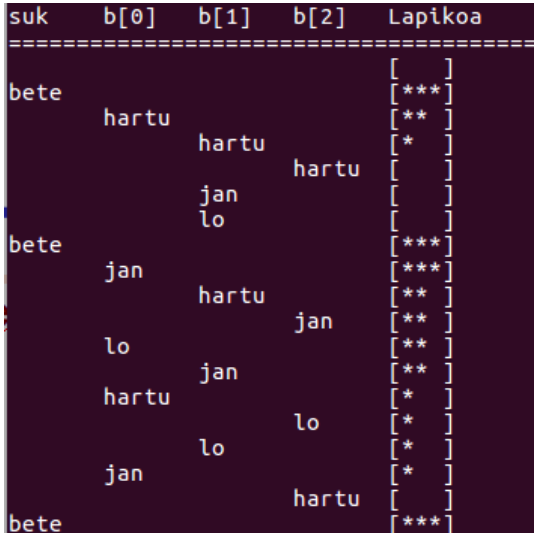
Monitorearen egoeran egindako aldaketak, zain daduen hariei `notify()` edo `notifyAll()` erabiltzen adierazten zaizkie.

Ariketak: Basatien festa

Ondoko problemak FSPz modelatu eta Java-z inplementatu:

1. Basatien festa:

   - Basati bakoitzak lapiko batetik misiolari-puska bat hartzen du; puska hori jaten bukatzean, tripazgora jarri eta ondoren beste bat hartzen du...
   - Basati sukaldariak lapikoa hutsik dagoenean lapikoa betetzen du misiolari-puskekin.



2. Basatien festa,
   baina orain sukaldariak
   aldi bakoitzean 3 puska botatzen ditu
   (lapikoan 3 puska baino gehiago sartzen dira).

Basatien festa ariketaren traza

Ariketa guztietako traza pantailan idazten denez, pantaila konpartituriko baliabide bat da, eta beraz elkar-bazterketa bermatu behar dugu.

Hau lortzeko, pantailan idazketa guztiak (**`println`** aginduak) **`Pantaila`** klase bateko **`synchronized`** metodoetan egingo dira.

1. hurbilpena

```
Sukaldariak bota
Lapikoan 3
Basati[1]-k hartu
Lapikoan 2
Basati[2]-k hartu
Lapikoan 1
Basati[1]-k hartu
Lapikoan 0
Sukaldariak bota
Lapikoan 3
```

2. hurbilpena

```
suk    b[1]     b[2]     b[3]     Lapikoa
=======================================
                                     0
bota                                 3
                hartu                2
       hartu                         1
                hartu                0
bota                                 3
                         hartu       2
```

3. hurbilpena

```
suk    b[1]     b[2]     b[3]     Lapikoa
=======================================
                                     [   ]
Bota                                 [***]
                hartu                [** ]
       hartu                         [*  ]
                hartu                [   ]
bota                                 [***]
                         hartu       [** ]
```

Adibidea: Basatien festa hainbat puskekin (I)

Basatien festaren beste aldaera bat:

- Basati bakoitzak puska bat hartzen du.
- Basati sukaldariak **hainbat** misiolari-puska botatzea erabakitzen du eta botatzen ditu.

```
const PK = 3        // Puska kopuru maximoa
range PR = 0..PK    // Lapikoan egon daitekeen puska kopuruaren rangoa
range SPR = 1..PK   // Sukaldariak bota dezakeen puska kopuruaren rangoa
const BK = 2        // Basati kopurua
range BR = 1..BK    // Basatien rangoa

BASATIA    = ( hartu -> BASATIA ).
SUKALDARIA = ( random[r:SPR]  -> bota[r]  -> SUKALDARIA ).

// i: lapikoan dagoen puska kopurua
LAPIKOA = LAPIKOA[0],
LAPIKOA[i:PR] =     ( when (i<PK)   s.bota[b:1..PK-i] -> LAPIKOA[i+b]
                    | when (i>0)    b[BR].hartu       -> LAPIKOA[i-1]
                    ).

||JANARIA = ( b[BR]:BASATIA || s:SUKALDARIA || LAPIKOA ).
```

Adibidea: Basatien festa hainbat puskekin (II)

Aurreko soluzioan, sukaldariak

- erabakitzen du zenbat bota jakin gabe ea sartuko den lapikoan, eta ez bazaizkio sartzen hor gelditukoko da zai lekua egon arte.

Egin dezagun orain sukaldariak

- begira dezala lapikoan zenbat puska sartzen diren, erabaki aurretik zenbat botako dituen.

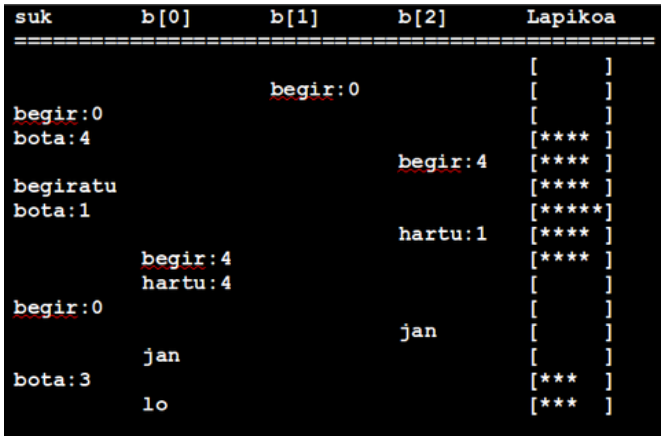
```
BASATIA    = ( hartu -> BASATIA ).

SUKALDARIA = ( begiratu[k:PR] ->
                    if (k<PK) then ( random[r:1..PK-k] -> bota[r] -> SUKALDARIA )
                              else                                   SUKALDARIA
             ).

LAPIKOA = LAPIKOA[0],
LAPIKOA [i:PR] =  (    when (i<PK)      s.bota[b:1..PK-i]   -> LAPIKOA[i+b]
                     | when (i>0)       b[BR].hartu         -> LAPIKOA[i-1]
                     |                  s.begiratu[i]       -> LAPIKOA[i]
                  ).

||JANARIA = ( b[BR]:BASATIA || s:SUKALDARIA || LAPIKOA ).
```

Ariketak

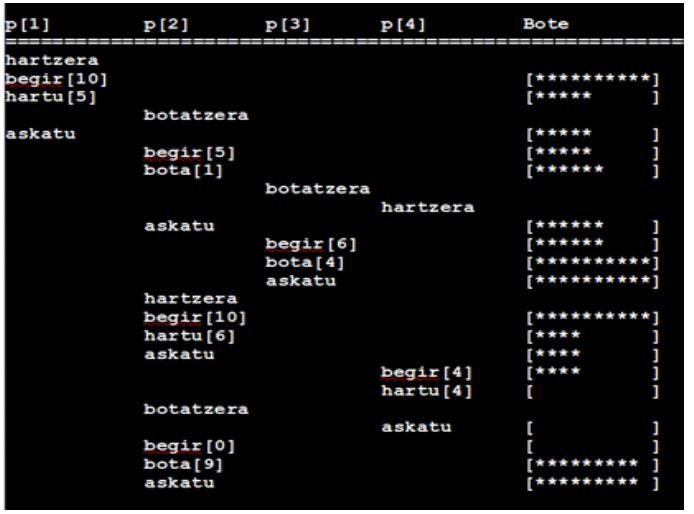
Ondoko problema FSPz modelatu eta Java-z inplementatu:

3. Basatien festa, baina orain

   - sukaldariak hainbat puska bota, eta
   - basatiek hainbat puska hartu, eta
   - bota edo hartu aurretik,
     lapikoan zenbat dagoen begiratzen dute



4. Ikasle jator batzuen pisuan gastuetarako bote bat dute. Norberak ahal duen heinean botean dirua sartzen du, eta behar duen neurrian hartu.



5*. Hainbat prozesu sinkronizatzen dira denek batera ekintza jakin bat egiteko.