

5. gaia

Monitoreak eta baldintzen sinkronizazioa

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:

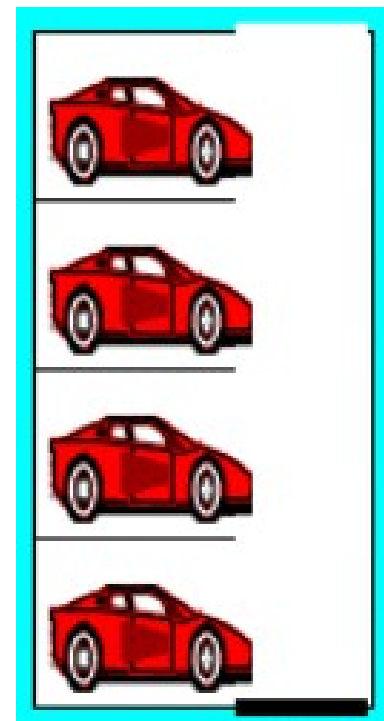
- Datu pribatuak eta metodo sinkronizatuak (elkar-bazterketa).
- `wait()`, `notify()` eta `notifyAll()` baldintzen sinkronizaziorako.
- Monitorean hari aktibo bakarra une bakoitzean.

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).



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)

AparkalekuaApp aplikazioaren main metodoak haien instantziak sortzen ditu :

```
class AparkalekuaApp{
    final static int Plazak = 4;
    public static void main (String args[]) {
        Kontrolatzailea k = new Kontrolatzailea (Plazak);
        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) {  
            aparkalekua.sartu();  
            System.out.println("Sartu da kotxe bat");  
        }  
    } catch (InterruptedException e) {}  
}
```

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

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

Kontrolatzailea monitoreak

```
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.

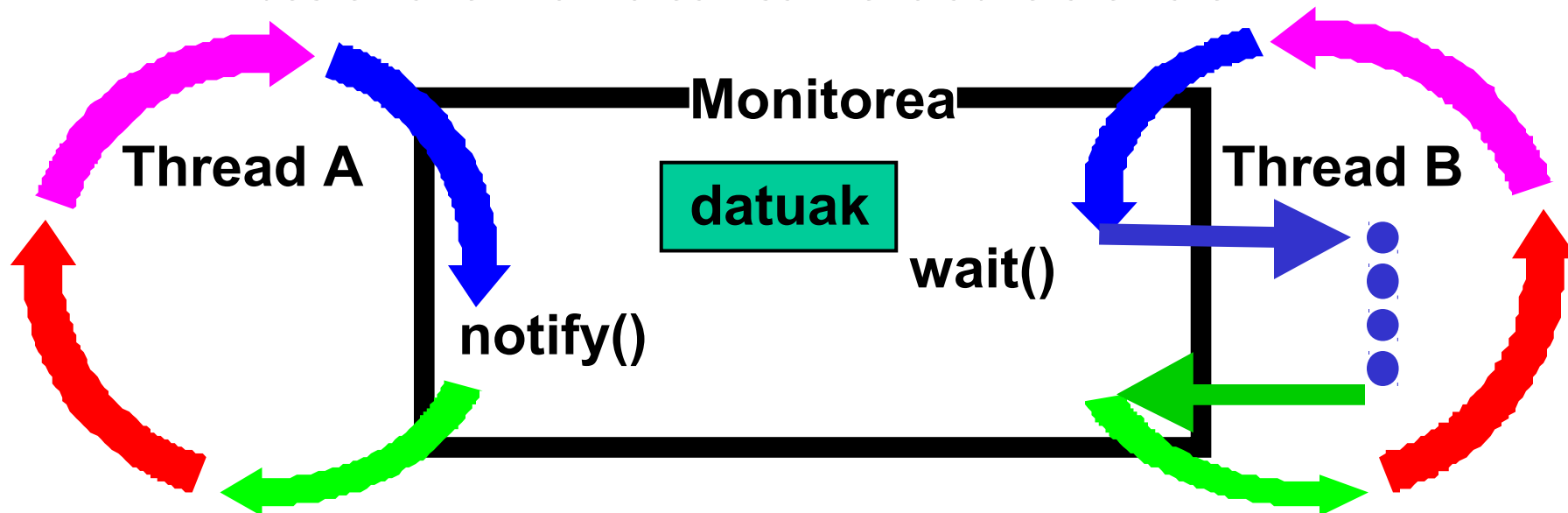
Notify egiten denean, hariak monitorearen berriro hartu arte itxaron.

Baldintzen sinkronizazioa Java-n

Esaten dugu

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

wait() – haria monotoretik 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;

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

    synchronized void sartu() throws InterruptedException {
        while !(kop<plazak) wait();
        ++kop;
        notify();
    }

    synchronized void irten() throws InterruptedException {
        while !(kop>0) wait();
        --kop;
        notify();
    }
}
```

Laburpena: Prozesuaren eredutik -> Java-ko monitorera

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

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

Ereduko ekintza babestu bakoitza
synchronized metodo batekin implementatzen da.

Metodo honek babesa implementatzeko
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.

Basatien festa ariketaren traza

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	[**]	

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.

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 K = 3
range KR = 1..K
const BK = 2
range BR = 1..BK
```

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

```
SUKALDARIA = ( random[r:1..N] -> bota[r] -> SUKALDARIA ).
```

```
LAPIKOA = LAPIKOA[0],
LAPIKOA[i:0..N] =
    ( when (i<N)  s.bota[b:1..N-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 geldituko 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:0..N] ->
 if (k<N) then (random[r:1..N-k] -> bota[r] -> SUKALDARIA)
 else
 SUKALDARIA
).

LAPIKOA = LAPIKOA[0],
 LAPIKOA [i:0..N] = (when (i<N) s.bota[b:1..N-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.

5.2 Semaforoak

s semaforoa:

soilik balio ez-negatiboak har ditzakeen osoko aldagaia da.

s-k onartzen dituen eragiketa bakarrak: *gora()* eta *behera()*.

```
behera(s):  if s > 0
              then gutxitu s
              else deia egin duen prozesuaren exekuzioa blokeatu
```

```
gora(s):    if prozesu blokeatuak daude s-n
              then haietako bat esnatu
              else gehitu s
```

Blokeatutako prozesuak FIFO ilara batean gelditzen dira.

Semaforoak modelatzen

Analizatu ahal izateko, balio multzo finitua har dezaketen semaforoak modelatuko ditugu.

Balio horietatik ateratzen bada **ERROR** itzuliko du.

```

const Max = 3
range Int = 0..Max

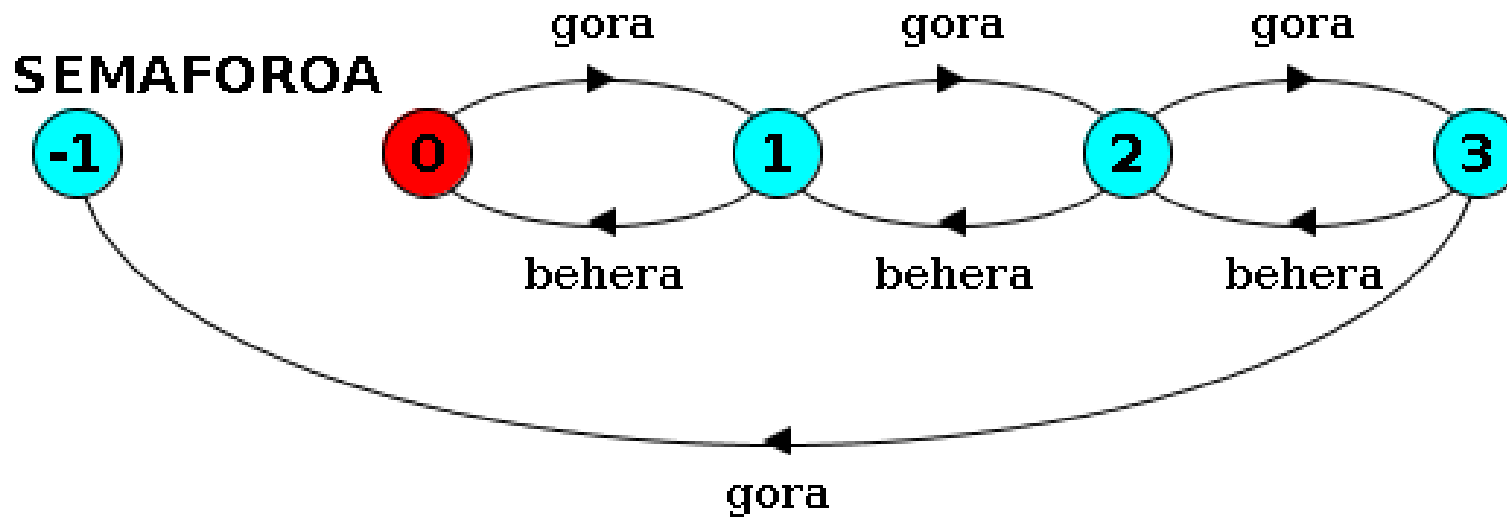
SEMAFOROA (N=0) = SEMA[N] ,
SEMA[b:Int]     = (
                    | when (b>0) gora ->SEMA[b+1]
                    | when (b>0) behera->SEMA[b-1]
                    ) ,
SEMA[Max+1]     = ERROR.

```

N hasierako balioa da.

lerro hau ez da
beharrezkoa

Semaforoak modelatzen



behera ekintza onartzen da soilik
semaforoaren v balioa 0 baino handiagoa denean ($v > 0$).

gora ekintza ez da babesten.

Errorera eramaten duen traza: **gora** → **gora** → **gora** → **gora**

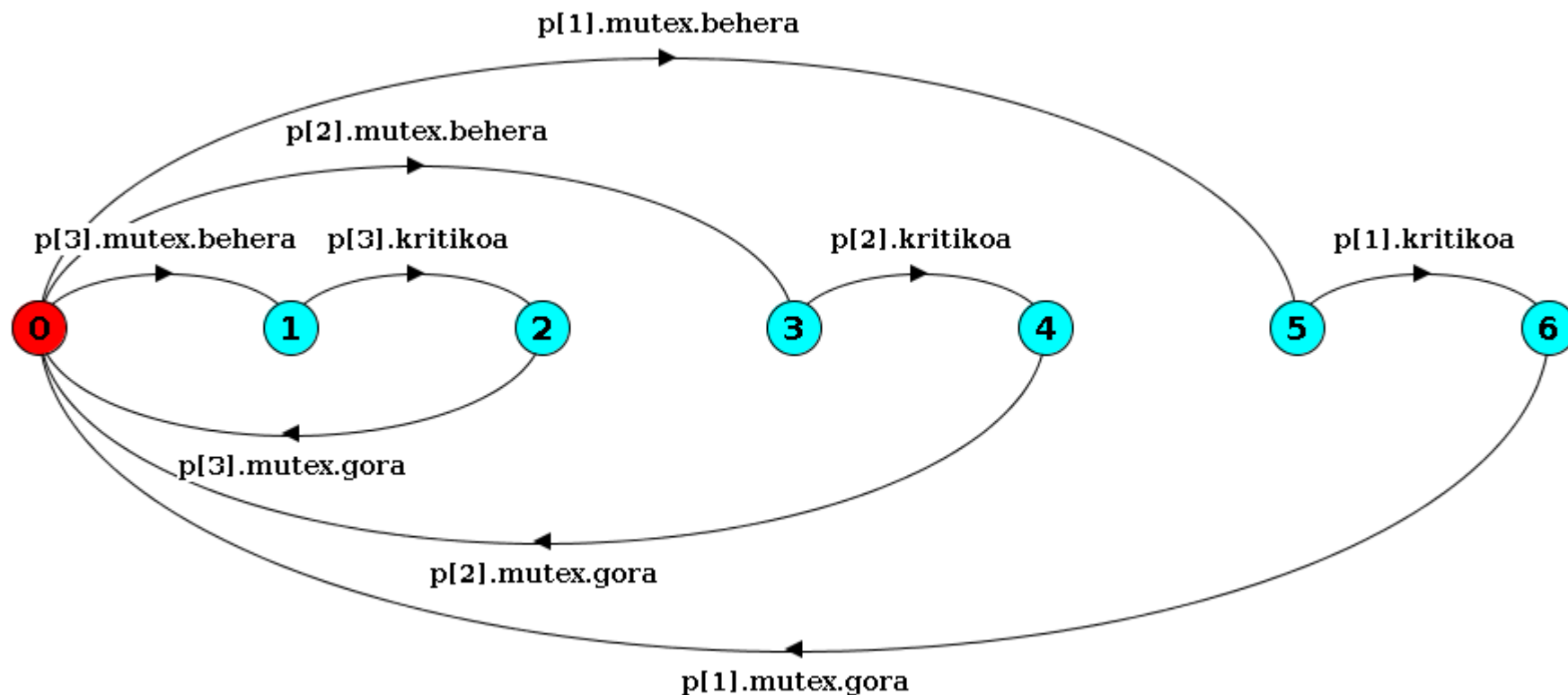
Semaforoekin adibidea - eredua

Adibide bezala semaforo baten erabilera modelatuko dugu elkar-bazterketa ziurtatzeko.

Hiru prozesu **p[1..3]** erabiltzen dute **mutex** semaforo bat baliabide baten atzipenean (**kritikoa** ekintza) elkar-bazterketa ziurtatzeko

```
LOOP = (mutex.behera->kritikoa->mutex.gora->LOOP) .  
|| SEMA DEMO = (    p[1..3]:LOOP  
                  || {p[1..3]} :: mutex:SEMAFOROA(1) ) .
```

Semaforoekin adibidea - eredu



- Elkar-bazterketa ziurtatzeko, semaforoaren hasierako balioa 1 da. **Zergatik?**
- **SEMADEMO** iritsi daiteke **ERROR** egoerara?
- Semaforo **bitarra** nahikoa al da (i.e. **Max=1**) ?

Semaforoak Java-n

Semaforoak objektu pasiboak dira, **monitore** bezala inplementatuak.

(Semaforoa behe-mailako mekanismo bat da, askotan goi-mailako monitoreak eraikitzeke erabilita)

```
public class Semaforo {  
    private int balioa;  
  
    public Semaforo (int hasierakoa)  
        {balioa = hasierakoa;}  
  
    synchronized public void gora() {  
        ++balioa;  
        notify();  
    }  
  
    synchronized public void behera()  
        throws InterruptedException {  
        while !(balioa>0) wait();  
        --balioa;  
    }  
}
```

SEMADEMO programa - MutexLoop

```
class MutexLoop extends Thread {
    Semaforo mutex;
    String tarteia;
    int luz;
    MutexLoop (Semaforo sema, int zenbat, String tabul) {
        mutex=sema; luze=zenbat; tarteia=tabul;
    }
    public void run(){
        try {while(true) {
            for (int i=1;i<=6;i++)    bisualizatu("|");
            mutex.behera(); // eskuratu elkar-bazterketa
            for (int i=1;i<=luze;i++) bisualizatu("*");// Ekintza kritikoa
            mutex.gora();    // askatu elkar-bazterketa
        }
        } catch (InterruptedException e){}
    }
    void bisualizatu(String ikurra) {
        try {System.out.println(tarteia+ikurra);
            sleep((int)(Math.random()*1000));
        }catch (InterruptedException e) {}
    }
}
```

Hariak eta semaforoa **main** metodoan sortzen dira.

Probatu sekzio kritikoan ematen den tarteia aldatzen, eta aztertu gatazka gehiago edo gutxiago ematen diren.

SEMADEMO pantailan

| : ekintza ez-kritikoa

* : ekintza kritikoa

2. haria zain dago

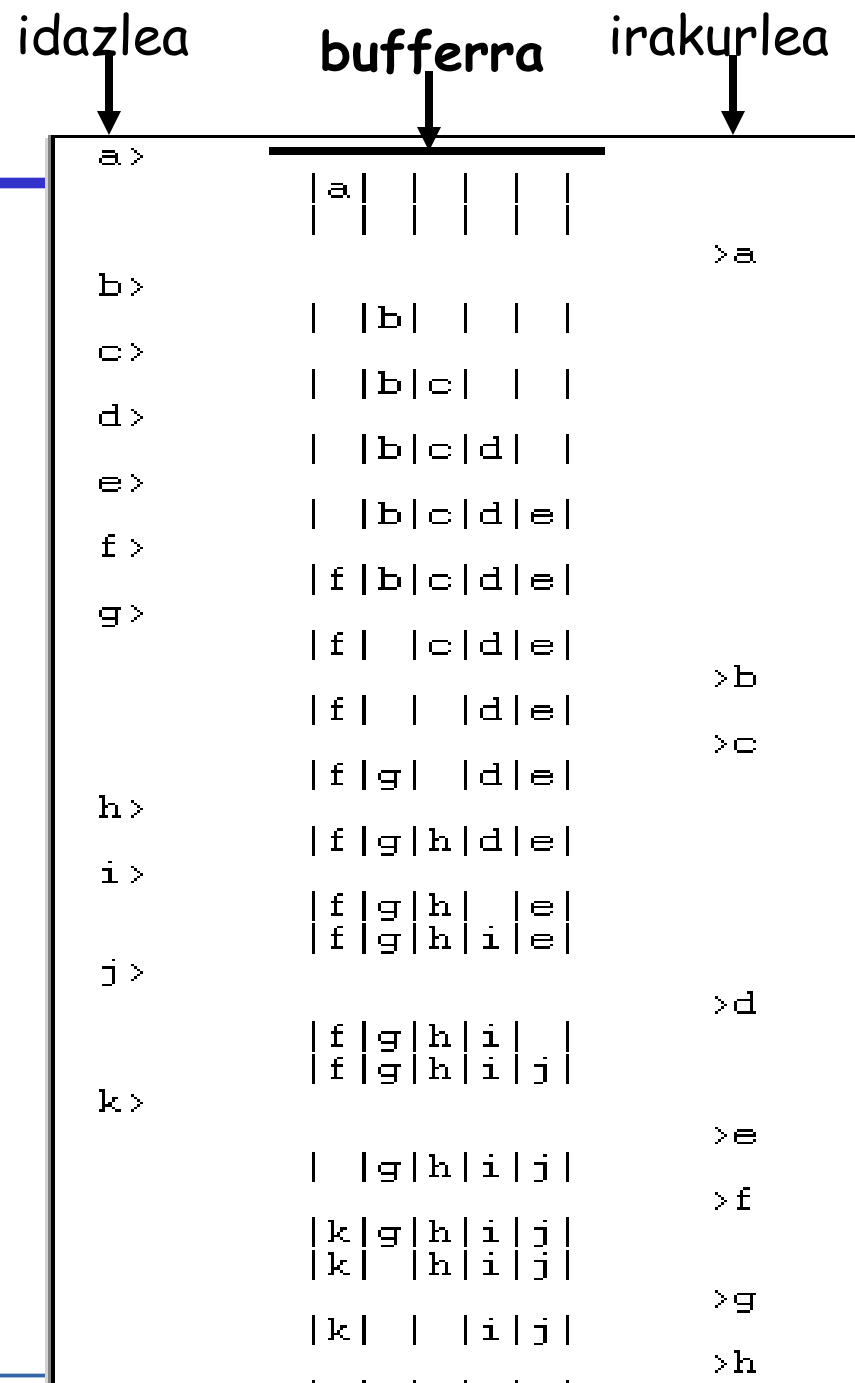
[illegible]

5.3 Buffer mugatuak

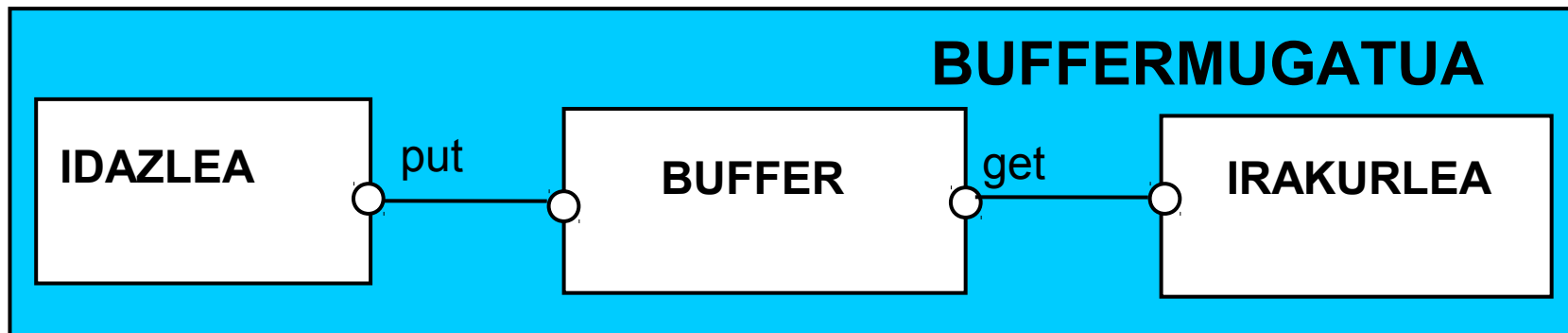
Buffer mugatu bat slot kopuru finko batez osatua dago.

Bufferrean:

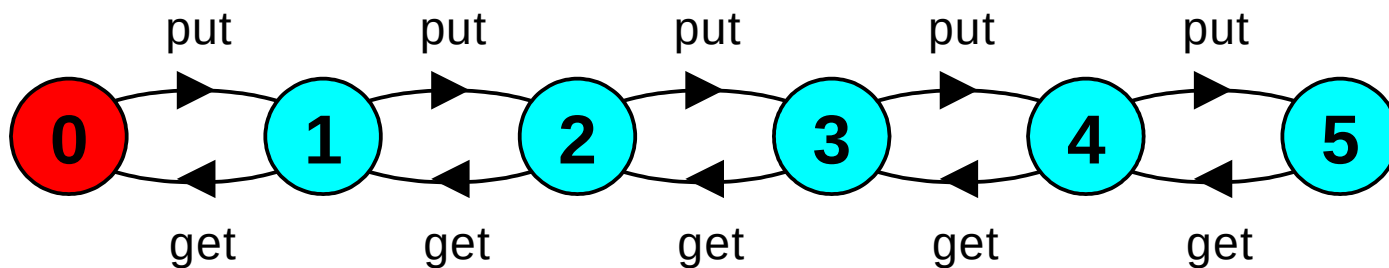
- > **idazle** prozesu batek sartzen ditu itemak, eta
- > **irakurle** prozesu batek ezabatzen ditu.



Buffer mugatua



LTS:



Buffer mugatua

```
const N = 5
BUFFER = COUNT[0],
COUNT[i:0..N] = ( when (i<N) put->COUNT[i+1]
                    | when (i>0) get->COUNT[i-1]
                    ) .
```

```
IDAZLEA    = (put->IDAZLEA) .
IRAKURLEA  = (get->IRAKURLEA) .
```

```
|| BUFFERMUGATUA = (IDAZLEA || BUFFER
                    || IRAKURLEA) .
```

Ariketa:

6. Egokitu buffer mugatuaren FSP eredua

put eta get egitean, jarri eta hartu behar den posizioa adierazteko

Buffer mugatua implementatzen: Buffer monitoreo

```
class Buffer{
    //aldagai lokalak eta eraikitzailea
    public synchronized void put(char c)
        throws InterruptedException {
        while !(kont<tam) wait();
        buf[in] = c; ++kont; in=(in+1)%tam;
        erakutsi();
        notify();
    }

    public synchronized char get()
        throws InterruptedException {
        while !(kont>0) wait();
        char c = buf[out];
        buf[out]=' '; --kont; out=(out+1)%tam;
        erakutsi();
        notify();
        return (c);
    }
}
```

erakutsi()
metodoak
bufferaren
edukiera
erakusten duen
metodoa da.

Buffer mugatua implementatzen: Idazlea haria

```
class Idazlea extends Thread {  
    Buffer buf;  
    String alphabet= "abcdefghijklmnopqrstuvwxyz";  
  
    Idazlea(Buffer b) {buf = b;}  
  
    public void run() {  
        try {  
            int ai = 0;  
            while(true) {  
                if (Math.random()<0.3) sleep(1000);  
                System.out.println(alphabet.charAt(ai)+">");  
                buf.put(alphabet.charAt(ai));  
                ai=(ai+1)%alphabet.length();  
            }  
        } catch (InterruptedException e){}  
    }  
}
```

Irakurlea antzekoa
izango da `buf.get()`
deituz.

5.4 Monitore habiratuak

Suposatu *kont* aldagaia eta baldintzen sinkronizazioa zuzenean erabili beharrean, bi semaforo (*okupatuak* eta *libreak*) erabiltzen ditugula bufferraren egoera kontrolatzeko.

```
class SemaBuffer{
...
    Semaforo okupatuak; // item kopurua zenbatzen du
    Semaforo libreak; // toki kopurua zenbatzen du
    SemaBuffer(int tam) {
        this.tam = tam;
        buf = new char[tam];
        for (int i=0; i<tam ; i++) buf[i]= ' ';
        okupatuak = new Semaforo(0);
        libreak    = new Semaforo(tam);
    }
...
}
```

Monitore habiratuak - buffer mugatua implementatzen

```
public synchronized void put(char c)
    throws InterruptedException {
    libreak.behera();
    buf[in]=c; ++kont; in=(in+1)%tam;
    erakutsi();
    okupatuak.gora();
}
public synchronized char get()
    throws InterruptedException {
    okupatuak.behera();
    char c=buf[out];
    buf[out]=' '; --kont; out=(out+1)%tam;
    erakutsi();
    libreak.gora();
    return (c);
}
```

Ondo ibiltzen al da hau?

libreak dekrementatzen da **put** eragiketan, **libreak** zero bada blokeatuz.

okupatuak dekrementatzen da **get** eragiketan, **okupatuak** zero bada blokeatuz.

Monitore habiratuak - buffer mugatua implementatzen

```
public class Semaforo {  
    private int balioa;  
  
    public Semaforo (int hasierakoa)  
    {balioa = hasierakoa;}  
  
    synchronized public void gora() {  
        ++balioa;  
        notify();  
    }  
  
    synchronized public void behera()  
        throws InterruptedException {  
        while !(balioa>0) wait();  
        --balioa;  
    }  
}
```

Monitore habiratuak - buffer mugatuaren eredua

```
const Max = 5
range Int = 0..Max

//SEMAFOROA ...lehen bezala...

BUFFER = (put -> libreak.behera    ->okupatuak.gora ->BUFFER
          |get -> okupatuak.behera ->libreak.gora   ->BUFFER
          ) .

IDAZLEA  = (put -> IDAZLEA) .
IRAKURLEA = (get -> IRAKURLEA) .

||BUFFERMUGATUA = (IDAZLEA || BUFFER || IRAKURLEA
                  ||libreak:SEMAFOROA(5)
                  ||okupatuak:SEMAFOROA(0)
                  )@{put,get} .
```

Ondo ibiltzen al da hau?

Monitore habiratuaren arazoa

LTSA –ak aurreikusten du

ELKAR-BLOKEAKETA (DEADLOCK) posiblea:

Composing

potential DEADLOCK

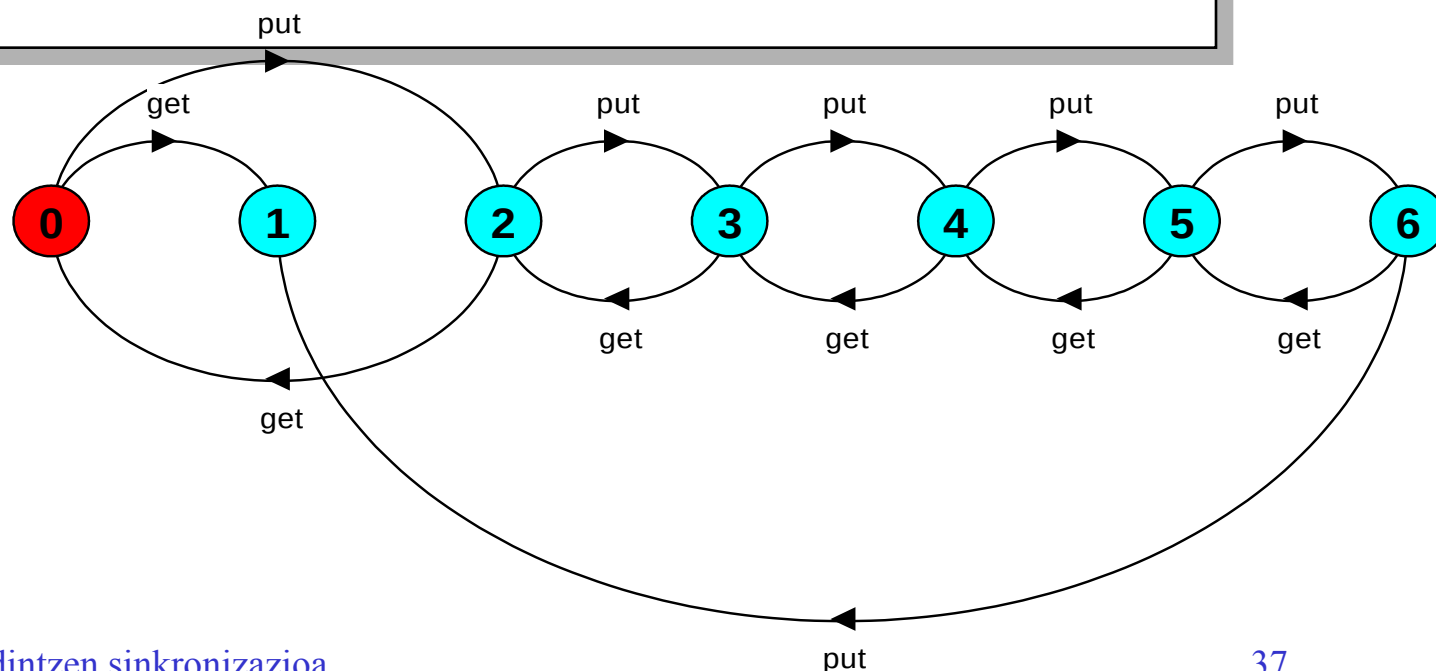
States Composed: 28 Transitions: 32 in 60ms

Trace to DEADLOCK:

get

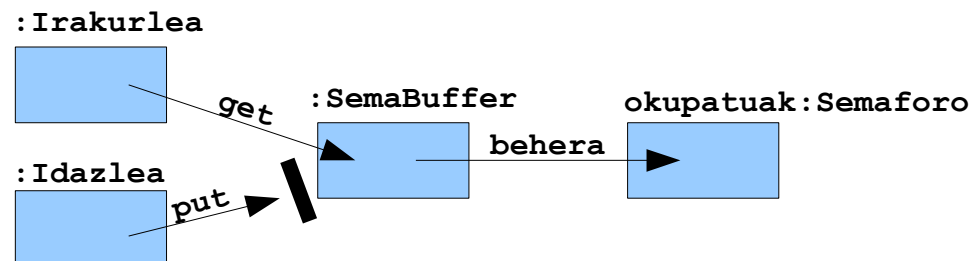
put

Egoera honi
monitore
habiratuaren
arazoa
deitzen zaio.



Monitore habiratuaren arazoa

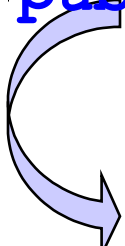
- **Irakurlea** karaktere bat hartzen (**get**) saiatzen da, **Buffer** monitorearen blokeoa eskuratzen du eta **okupatuak.behera()** deitzean **okupatuak** semaforoaren blokeoa eskuratzen du, bufferrean zerbait dagoen ikusteko.
- Hasieran **Buffer** hutsik dagoenez, **okupatuak.behera()** deiak **while (balioa == 0) wait();** eginez **Irakurlea** blokeatzen du eta **okupatuak** semaforoaren blokeoa askatzen du.
- Hala ere ez du **Buffer** monitorearen blokeoa askatzen.
- Beraz **Idazlea** ezin da sartu **Buffer** monitorean karaktere bat jartzeko, eta blokeatzen da.
- Ez **Idazlea** ez **Irakurlea** prozesuek ezin dute aurrerapenik egin.
- Elkar-blokeaketa ematen da...



Monitore habiratuak - buffer mugatua implementatzen II

Arazo hau Java-n saihesteko modu bakarra arretaz diseinatzea da. Adibide honetan elkar-blokeaketa ezabatu daiteke ziurtatzen badugu buffer monitorearen blokeoa ez dela eskuratzen semaforoak dekrementatuak izan arte.

```
public void put(char c)
    throws InterruptedException {
    libreak.behera();
    synchronized(this) {
        buf[in] = c; ++kont; in=(in+1)%tam;
    }
    okupatuak.gora();
}
```



Monitore habiratuak - buffer mugatuaren eredua II

```
BUFFER = (put -> BUFFER  
          |get -> BUFFER  
          ) .
```

```
IDAZLEA =  
    (libreak.behera ->put->okupatuak.gora->IDAZLEA) .  
IRAKURLEA =  
    (okupatuak.behera->get->libreak.gora ->IRAKURLEA) .
```

Semaforoaren ekintzak jarri dira idazlean eta irakurlean (semaforoaren ekintzak monitorearen kanpoan dauden implementazioan bezala, hau da monitorearen blokeoa hartu baino lehen).

Ondo ibiltzen al da hau?

LTS minimizatua?

5.5 Monitoreen inbarianteak

Monitore baten **inbariantea** monitorearen aldagaiei buruzko baizeztapen bat da. Baieztapen hau bete behar da beti, hari bat monitore barruan egikaritzen ari denean ezik. Hau da, bete behar da hari bat monitorean sartu aurretik eta ateratzean.

Kontrolatzailea-ren inbariantea:	$0 \leq kop \leq N$
Semaforo-aren inbariantea:	$0 \leq balioa$
Buffer-aren inbariantea:	$0 \leq kont \leq tam$
	and $0 \leq in < tam$
	and $0 \leq out < tam$
	and $in = (out + kont) \% tam$

Inbarianteak lagungarri izan daitezke monitoreen zuzentasunari buruz hausnartzeko, frogapenean oinarritutako hurbilpen logikoa erabiliz. Guk erduetan oinarritutako hurbilpena erabiliko dugu frogatze mekanikoa egin ahal izateko.

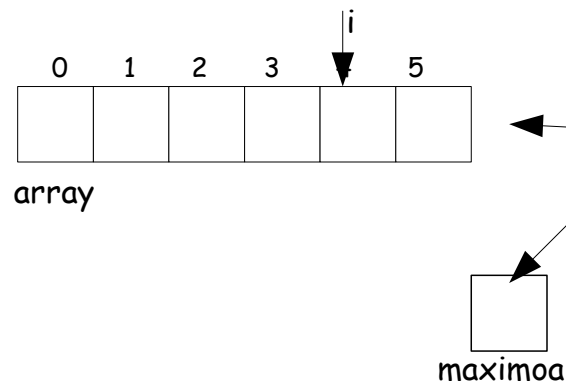
Ariketa: Maximoa ukeratu

7. Array bateko zenbakien artean maximoa aukeratu.

FSP eredu eman, horrela soluzionatuz:

Zenbakiak dituen array-az gain aldagai laguntzaile bat erabiliko dugu maximoa gordetzeko.

Prozesu konkurrenteek ondokoa egiten dute: array-tik (hartu gabeko) zenbaki bat hartu eta uneko maximoa baino handiagoa bada, maximoan sartu zenbaki hori. Jakiteko zein den array-tik hartu beharreko zenbakia indize bat erabiliko dugu eguneratzen joan beharko duguna.



- Hartu i-n dagoena eta maximoa-n dagoena
- $i=i-1$
- Konparatu bi elementuak eta zenbaki berria maximoa baino handiagoa bada orduan zenbaki berria gorde maximoa-n.

Hausnartu zergatik soluzioa hau ez den batere eraginkorra.

5.6 Agendaren eredua

Agendaren ereduan,
array bateko elementuak prozesatu behar ditugunean ondokoa egiten dugu:

- Elementuak array-tik hartu
- Prozesatu hartutako elementuak
- Prozesatutako emaitza array-an sartu

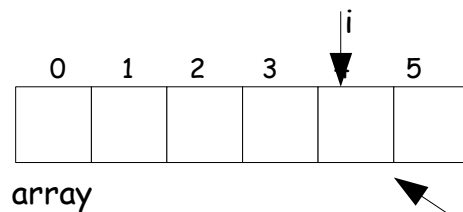
Elementu guztiak prozesatzen bukatu arte.

Jakiteko nondik hartu behar diren prozesatu beharreko elementuak eta non gorde behar den soluzioa, indize bat erabiliko dugu eguneratzen jona beharko dena.

Ariketa: Maximoa aurkitu agendaren eredua erabiliz

8. Array bateko zenbakien artean maximoa aukeratu.

FSP eredua eman eta Javaz implementatu, agendaren eredua erabiliz.



- Hartu i eta $i-1$ posizioetan dauden elementuak
- $i=i-2$
- Konparatu bi elementuak
- Handiena gorde oraingo i posizioan
- $i=i+1$



Ondo pentsatu noiz bukatzen den prozesaketa.

Ariketak

Ondoko problemak FSPz modelatu eta Java-z inplementatu:

9. FIFO ilara batean prozesuak sartu eta ateratzen dira.

10. LIFO ilara batean prozesuak sartu eta ateratzen dira.

11. Basatien festa eroa:

Misiolariak iristean, sukaldariak akatzen ditu, zatitu, puskak hozkailuan sartu, eta hortik lapikora..., begiratuz beti ea tokia dagoen lapikoan, hozkailuan...