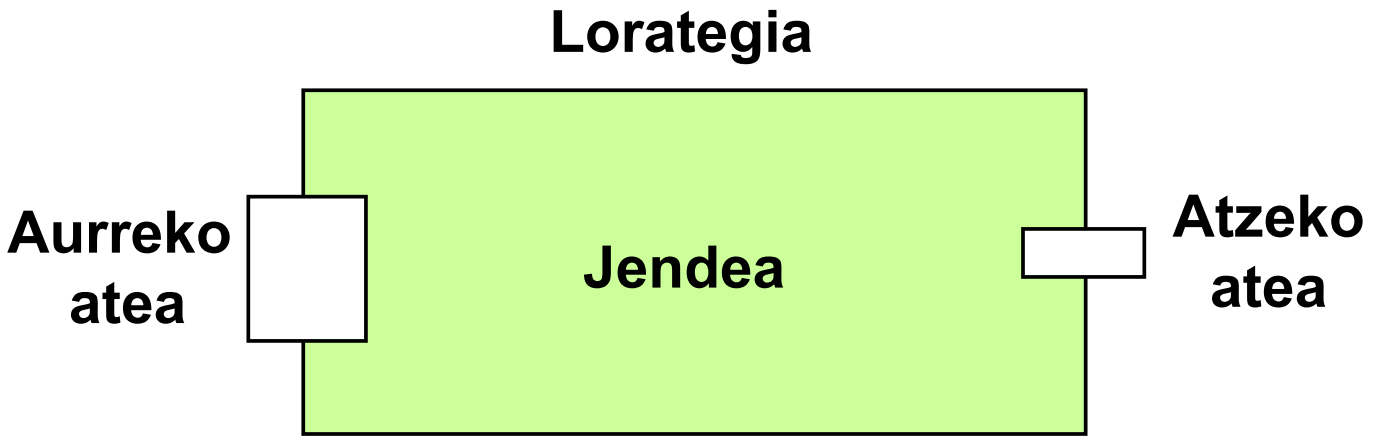
Objektu konpartituak eta elkar-bazterketa

4.1 Interferentzia

Lorategiaren problema:

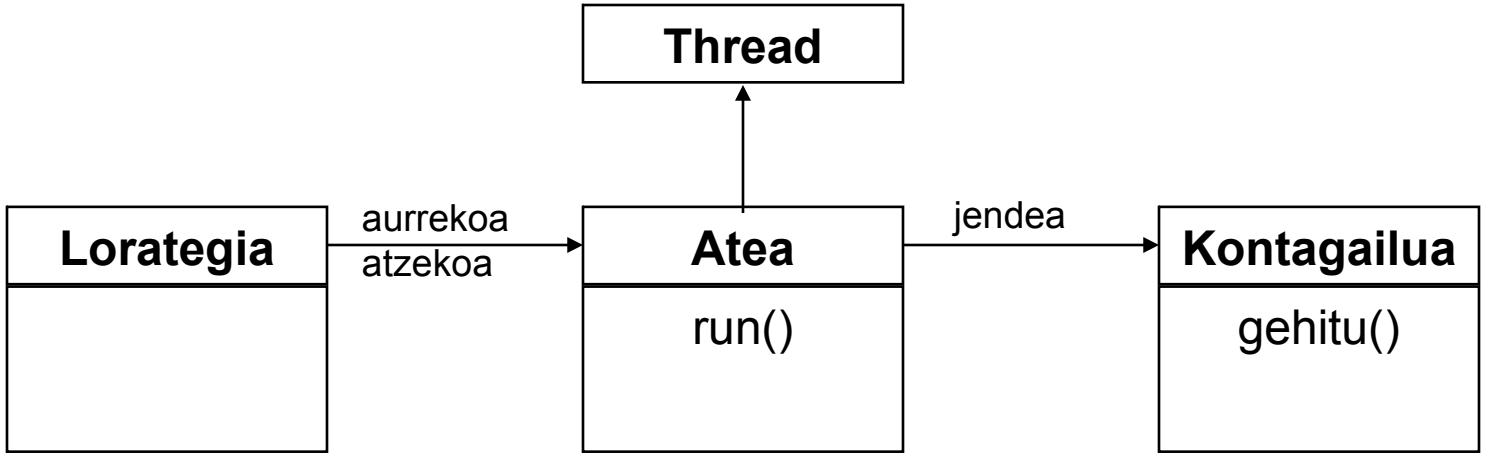
Lorategia bisitatzeko bi sarrera daude.
Lorazainak jakin nahiko luke zenbat jende dagoen lorategian momentu bakoitzean.



Suposatuko dugu:

- jendea sartu bai, baina ez dela ateratzen.
- jendea segundo bateko tartearekin iristen dela.

Lorategiaren problema - klase diagrama



Programa konkurrenteak bi hari eta kontagailu objektu konpartitu bat izango ditu.

Atea hariak simulatuko du jendearen aldizkako ailegatzea:

- segundo batez **sleep()** egingo du, eta ondoren
- kontagailua objektuaren **gehitu()** metodoari deituko dio.

LorategiaApp inplementazioa

LorategiaApp aplikazioaren **main** metodoak **Kontagailua** objektua eta **Atea** hariak sortzen ditu :

```
class LorategiaApp{
   public final static int MAX = 6;

   public static void main (String args[]) {
         System.out.println("LORATEGIA: return sakatu hasteko");
         try{int c = System.in.read();}catch(Exception ex){}
         System.out.println("Aurre \tAtze \tGuztira");

         Kontagailua k = new Kontagailua();
         Atea aurrekoa = new Atea("",k);
         Atea atzekoa  = new Atea("\t",k);
         aurrekoa.start();
         atzekoa.start();
   }
}
```

Atea klasea

```
class Atea extends Thread {
  Kontagailua kont;
  String atea;

  public Atea (String zeinAte, Kontagailua k){
      jendea=k;  atea=zeinAte;
  }

  public void run() {
    try{
       for (int i=1;i<=LorategiaApp.MAX;i++){
            sleep((long)(Math.random()*1000));
                  //ausazko denbora itxaron (0 eta 1 segunduren tartean)
            System.out.println(atea+i);
            kont.gehitu();
       }
    } catch (InterruptedException e) {}
  }
}
```

LorategiaApp.MAX atearen
bisitari kopuru maximora iristean **run()**
metodotik ateratzen da eta haria hiltzen da.

Kontagailua klasea

```
class Kontagailua {
   int balioa=0;

   Kontagailua() {
      System.out.println("\t\t"+balioa);
   }

   void gehitu() {
      int lag;
      lag=balioa;      //balioa irakurri
      Simulatu.HWinterrupt();
      balioa=lag+1;    //balioa idatzi
      System.out.println("\t\t"+balioa);
   }
}
```

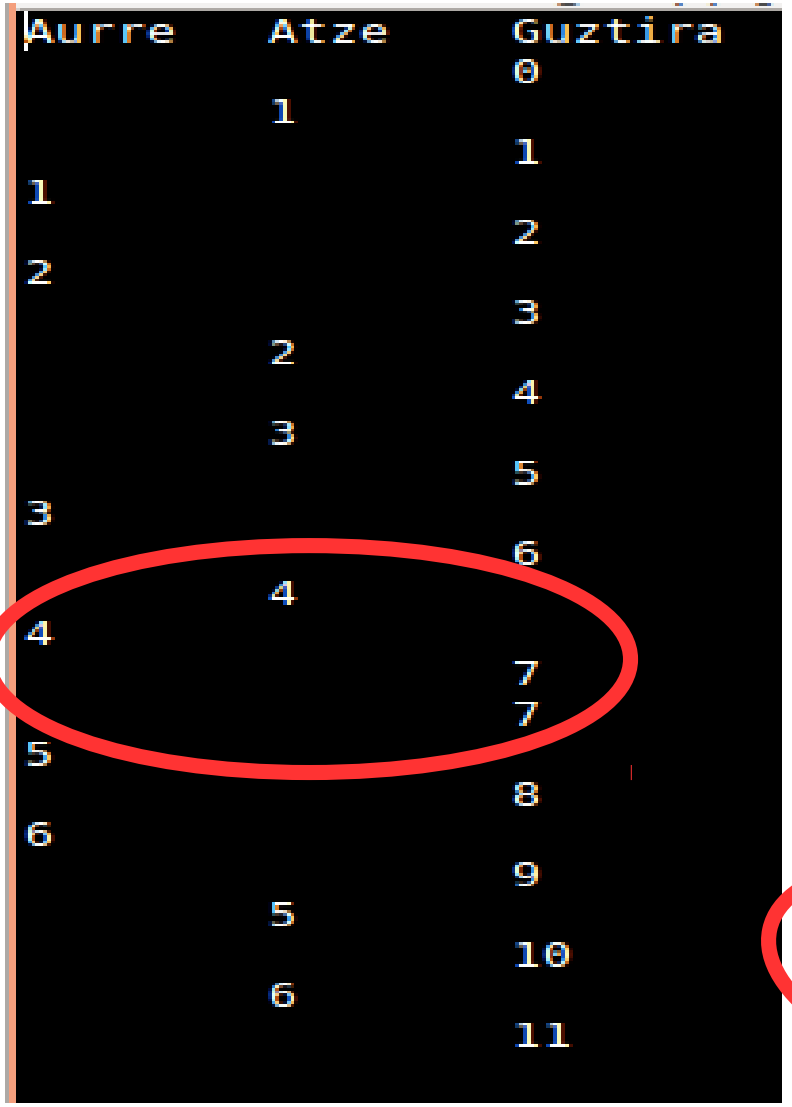
Hardware-etena edozein momentuan gerta daiteke. **Kontagailua**-k hardware-eten bat simulazen du **gehitu()** egiten ari denean, **balioa** kontagailu konpartitua irakurri eta idatzi bitartean.

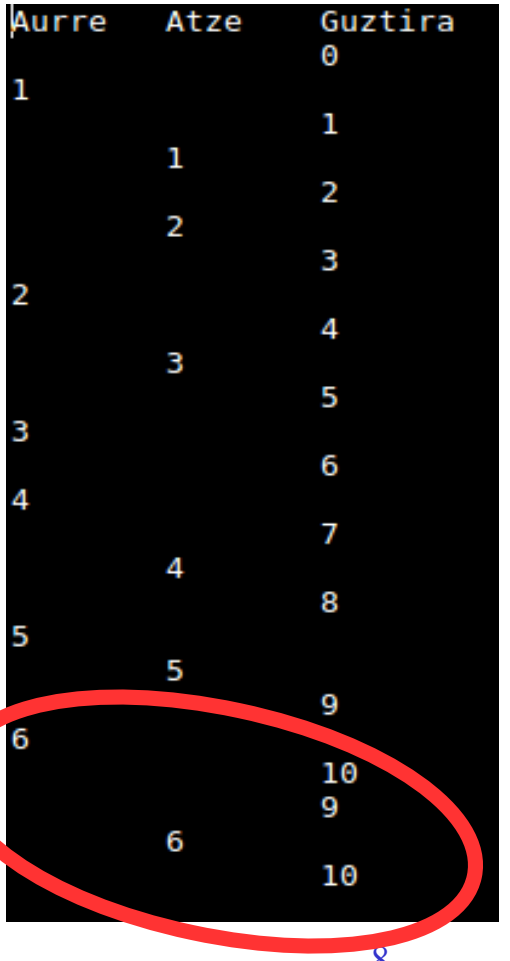
Kontagailua klasea: Hardware-etenaren simulazioa

```
class Simulatu {
    public static void HWinterrupt() {
        if (Math.random()<0.2)
           try{Thread.sleep(200);}
           catch(InterruptedException e){};
    }
}
```

LorategiaApp martxan

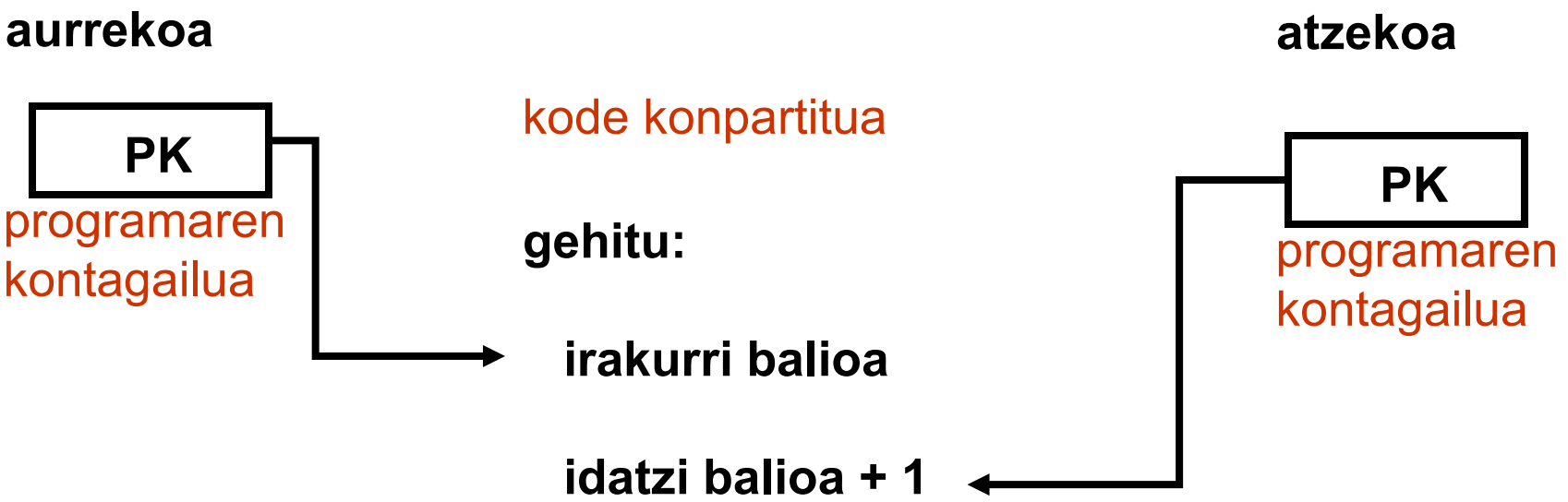
?!





Metodoen aktibazio konkurrentea

- Java metodoen aktibazioak ez dira atomikoak.
- **aurrekoa** eta **atzekoa** hariek **gehitu()** metodoaren kodea aldi berean egikaritu dezakete.



Interferentzia eta elkar-bazterketa

Interferentzia:
irakurri eta idatzi ekintzen tartekatze arbitrarioagatik eragindako eguneratze suntsitzailea.

- Interferentzia-erroreak aurkitzea oso zaila da.
- Soluzio orokorra: metodoen *elkar-bazterketa* objektu konpartituen atzipenean
- Elkar-bazterketa ekintza atomikoen modura modelatu daiteke.

4.2 Elkar-bazterketa Java-n

Nola egin Java-n metodo baten aktibazio konkurrentean elkar-bazterketa lortzeko?

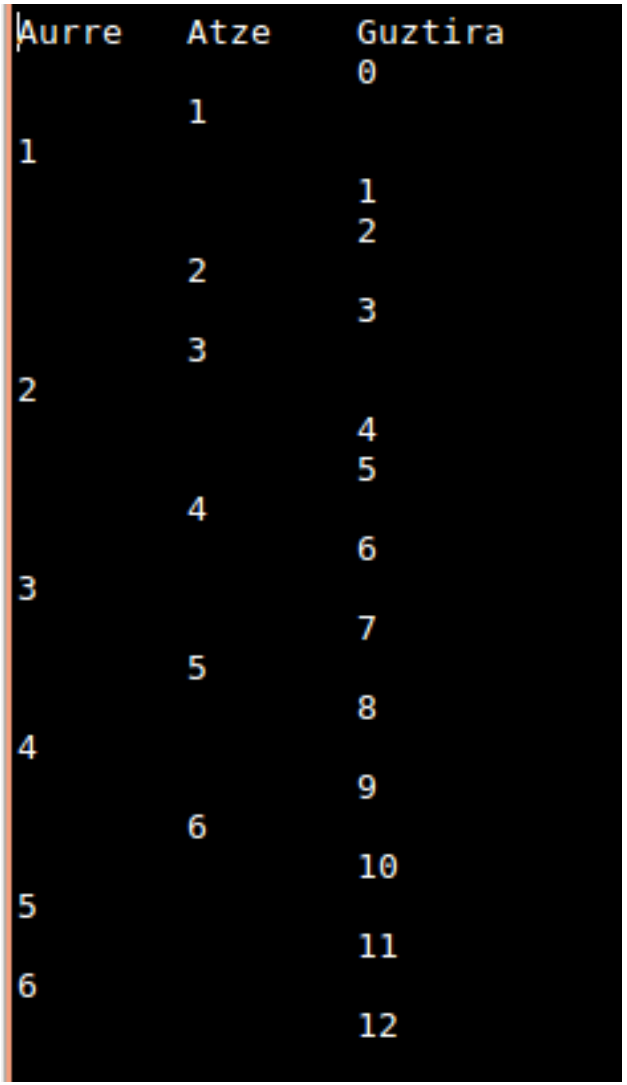
Metodoan **synchronized** hitz erreserbatua jarriz

Kontagailua klasearen gehitu metodoa synchronized egingo dugu:

```
class Kontagailua {
      ...

      synchronized void gehitu() {
          ...
      }
}
```

Elkar-bazterketa - Lorategia



Java-k objektu guztiei sarraila/blokeo (***lock***) bat ezartzen die.

Java konpilatzaileak kodea gehitzen du:

- blokeoa eskuratzeko synchronized metodoaren gorputza egikaritu baino lehen, eta
- blokeoa askatzeko metodoa itzuli aurretik.

Hari konkurrenteak blokeatzen dira blokeoa askatu arte.

Java-ko synchronized sententzia

Objektu baten atzipena ere elkar-bazterketa bete dezan egin daiteke **synchronized** sententzia erabiliz:

synchronized (*objektua*) { *aginduak* }

Aurreko adibidea zuzentzeko beste modu bat (ez hain elegantea) litzateke **ATEA.run()** metodoa modifikatzea:

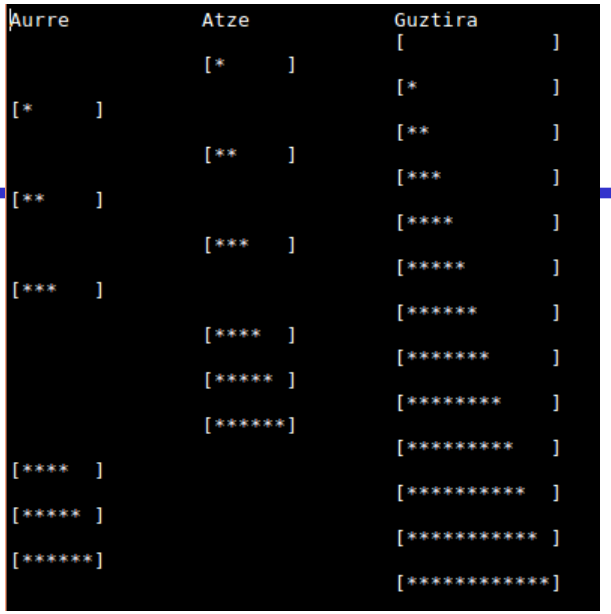
```
synchronized(jendea) {jendea.gehitu();}
```

Zergatik ez da hain elegantea?
Objektuaren erabiltzaileak daukalako blokeoa ezartzearen ardura, eta erabiltzaileak ez badu arduratsu jokatzen interferentzia eman daiteke.

Objektu baten atzipenean elkar-bazterketa ziurtatzeko,
objektuaren metodo guztiek behar dute izan **synchronized**.

Ariketak

1. Lorategiaren programa egokitu,
   zenbakiak idatzi beharrean,
   kopuruak izartxoekin adierazteko.



2. Aurreko programa egokitu,
   izartxo bat idazten den bakoitzean,
   HWInterrupt dei bat egiten.
   Ikusi gertatzen den interferentzia.

3. Aurreko programa egokitu,
   pantailako idazketa guztiak **`Pantaila`** klase batean egiteko,
   eta ziurtatu pantailako idazketetan ez dela interferentziarik ematen.