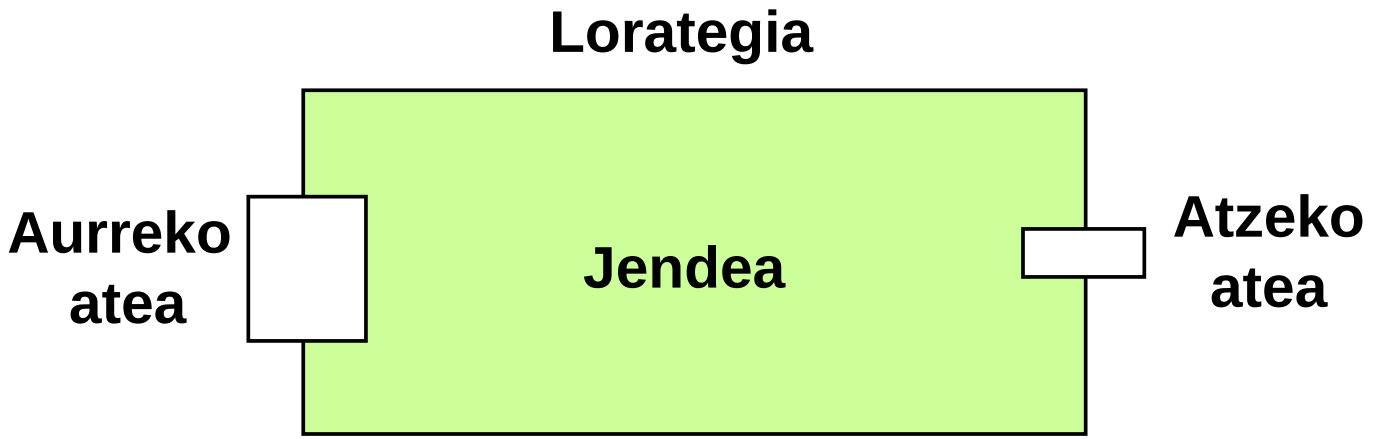
Objektu konpartituak eta elkar-bazterketa

4.1 Interferentzia

Lorategiaren problema:

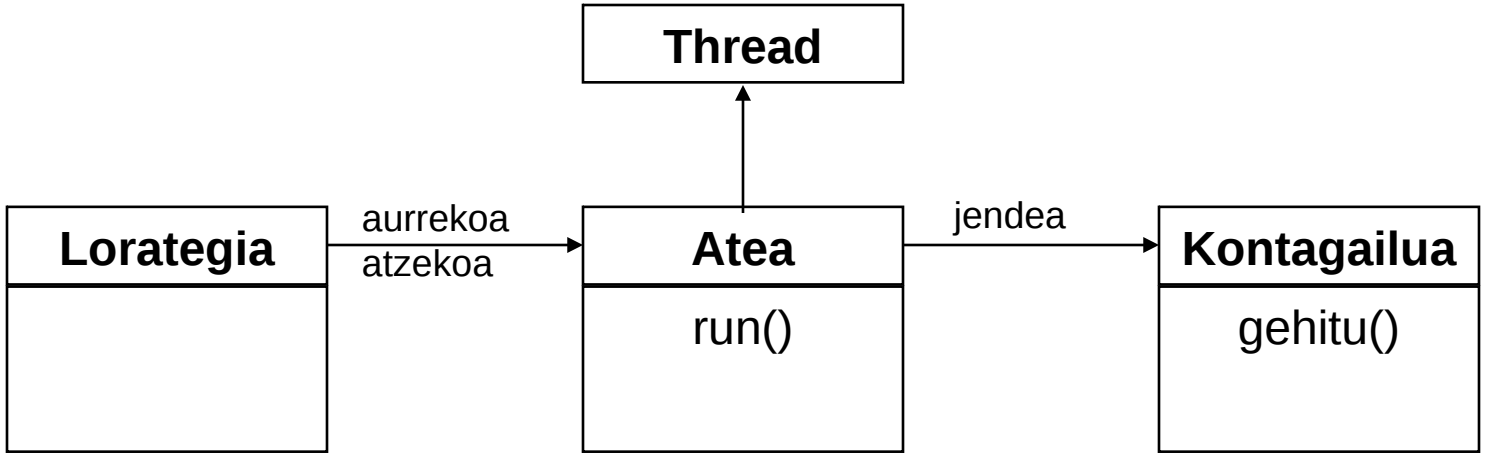
Lorategia bisitatzeko bi sarrera daude.
Lorazainak jakin nahiko luke zenbat jende dagoen lorategian momentu bakoitzean.



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){}

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

Atea klasea

```
class Atea extends Thread {
  Kontagailua jendea;
  String atearenIzena;

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

  public void run() {
    try{
       System.out.println(atearenIzena+">"+0);
       for (int i=1;i<=LorategiaApp.MAX;i++){
            Thread.sleep(1000); //segundu bat itxaron
            System.out.println(atearenIzena+">"+i);
            jendea.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\tguztira:"+balioa);
   }

   void gehitu() {
      int lag;
      lag=balioa;      //balioa irakurri
      Simulatu.HWinterrupt();
      balioa=lag+1;    //balioa idatzi
      System.out.println("\t\tguztira:"+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()<0.5)
           try{Thread.sleep(200);}
           catch(InterruptedException e){};
    }
}
```

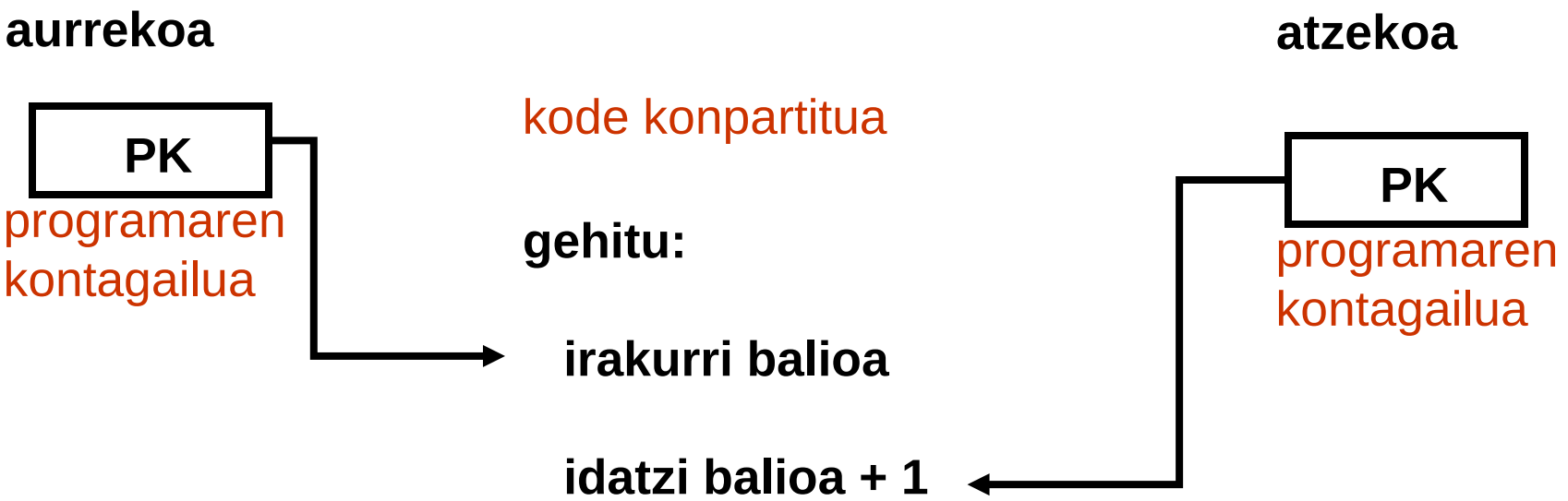
LorategiaApp martxan

?!

```
                    guztira:0
aur>0
        atz>0
aur>1
                    guztira:1
        atz>1
                    guztira:2
aur>2
                    guztira:3
        atz>2
                    guztira:4
aur>3
        atz>3
                    guztira:5
                    guztira:6
aur>4
                    guztira:7
        atz>4
                    guztira:8
aur>5
                    guztira:9
        atz>5
                    guztira:10
aur>6
        atz>6
                    guztira:11
                    guztira:11
aur>7
```

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 azpiklase bat instanziatuko dugu, eta bertan gehitu metodoa synchronized egingo dugu:.

```
class KontagailuSinkr extends Kontagailua {
     KontagailuSinkr() {
        super();
     }
     synchronized void gehitu() {
        super.gehitu();
     }
}
```

Elkar-bazterketa - Lorategia

```
C:\WINNT\System32\cmd.exe
LORATEGIA: return sakatu hasteko

                guztira:0
aur>0
        atz>0
aur>1
                guztira:1
        atz>1
                guztira:2
aur>2
        atz>2
                guztira:3
                guztira:4
        atz>3
aur>3
                guztira:5
                guztira:6
        atz>4
aur>4
                guztira:7
                guztira:8
        atz>5
aur>5
                guztira:9
                guztira:10
        atz>6
aur>6
                guztira:11
                guztira:12
Presione una tecla para continuar . . .
```

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**.