3. Gaia

OBP Gai Aurreratuak

Aurkibidea

- **Gaiaren Helburuak**
- Herentzia
- Polimorfismo eta lotura dinamikoa

Helburuak

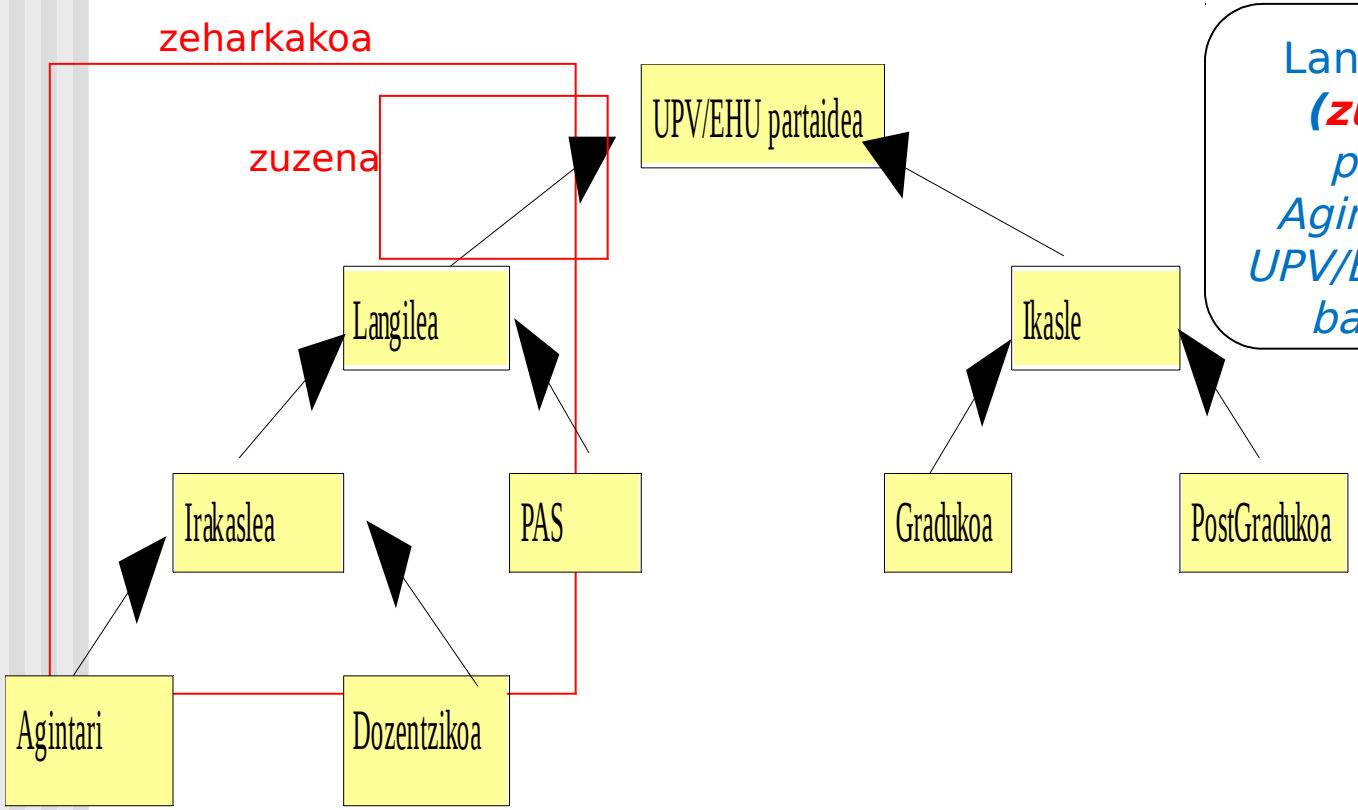
- Herentzia ber-erabilen modu bezala
- Superklase eta azpiklase
- *extends* hitz berezia eta *protected* ikusgarritasuna
- Eraikitzaileak eta herentzia
- Object superklasea eta bere metodoak
- Polimorfismo, lotura dinamikoa, eta klase eta metodo abstraktoak

Aurkibidea

- Gaiaren helburuak
- **Herentzia**
- Polimorfismo eta lotura dinamikoa

Herentzia erlazioak

- Erlazioa hierarkia “bat-da”/“is-a”



Langile bat ***hedaduraz (zuzena)*** *UPV/EHUko partaide bat da eta Agintari bat* ***hedaduraz*** *UPV/EHUko partaide bat da baita* ***(zeharkakoa)***

Herentziaren ondorioak

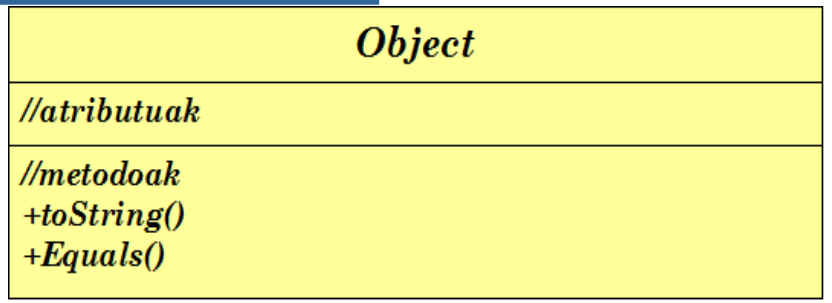
- OBP hedadurak azpiklase batek superklase batetik **atributu/metodoak heredatzea** ahalbidetzen du
  - Errealitatean: Guraso-umeen arteko herentzia genetikoa existitzen da
    - atributuak: begi kolorea, altuera
    - metodoak: ibiltzeko modua/keinuak egiteko modua

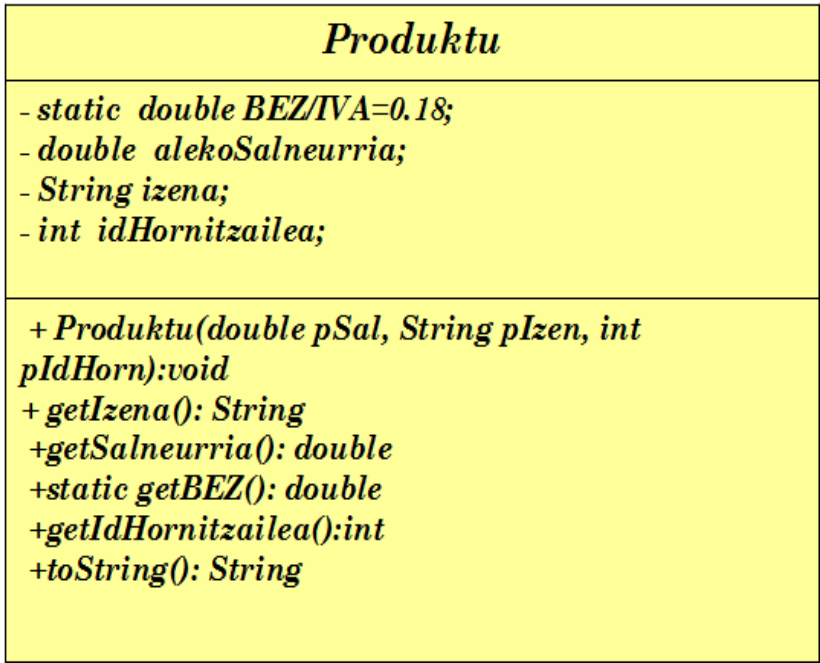
Object superklasea

- Javaz, klase zuhaitzen erroa Object klasea da
  - Beraz, edozein klase berri (“bat-da”/”is-a”) Object klasearen azpiklase (zuzen edo zeharka) bat da

Adibidea

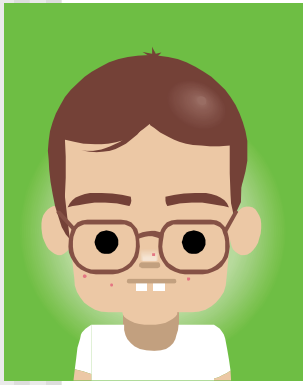
- Metodo heredatuak
  - equals()
  - toString()





Adi, galdera

- Zergaitik Produktu Klasean *toString()* metodo bat agertzen da eta ez equals() metodo bat? Ez al dira biak heredatzen?



Ez egin tranparik!! Ez begiratu hurrengo gardenkia... Behintzat erantzuna pentsatu arte

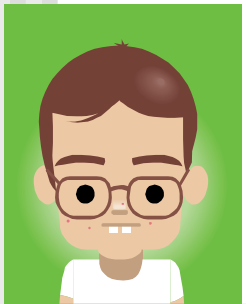
Gainidazketa

- *toString()* metodo heredatuak objektuaren memoria helbidea String batetan bihurtzen du
- Hau ez denez oso erabilgarria, gainidaztea pentsatu da
  - Superklasean duen buru bera izan behar du
  - Inplementazio berria (programatzailearen eskuetan)

```
//toString metodoaren gainidazketa )
public  String toString(){
  return(String.format("izena: %s, hornitzailearen id :
%d, salneurria %.2f", this.getIzena(),  this.getHornitzailea(),this.getSalneurria()));
}
```

Gainidazketa

- *equals()* metodo heredatuak bi objektu memorian helbide bera duten konprobatzen du (hau da, objektu bera den)
  - Kontuz!! Bi objektuen edukia berdina denean, baina ez denean objektu bera, *equals()*-k false buelatzen duno
- Produktu klasea, *equals()* ez da gainidatzi



Baina adibidez, String klaseak gainidatzita du, honela edukia konparatzeko gainidatzi da, eta ez helbideak

Gainidazketa

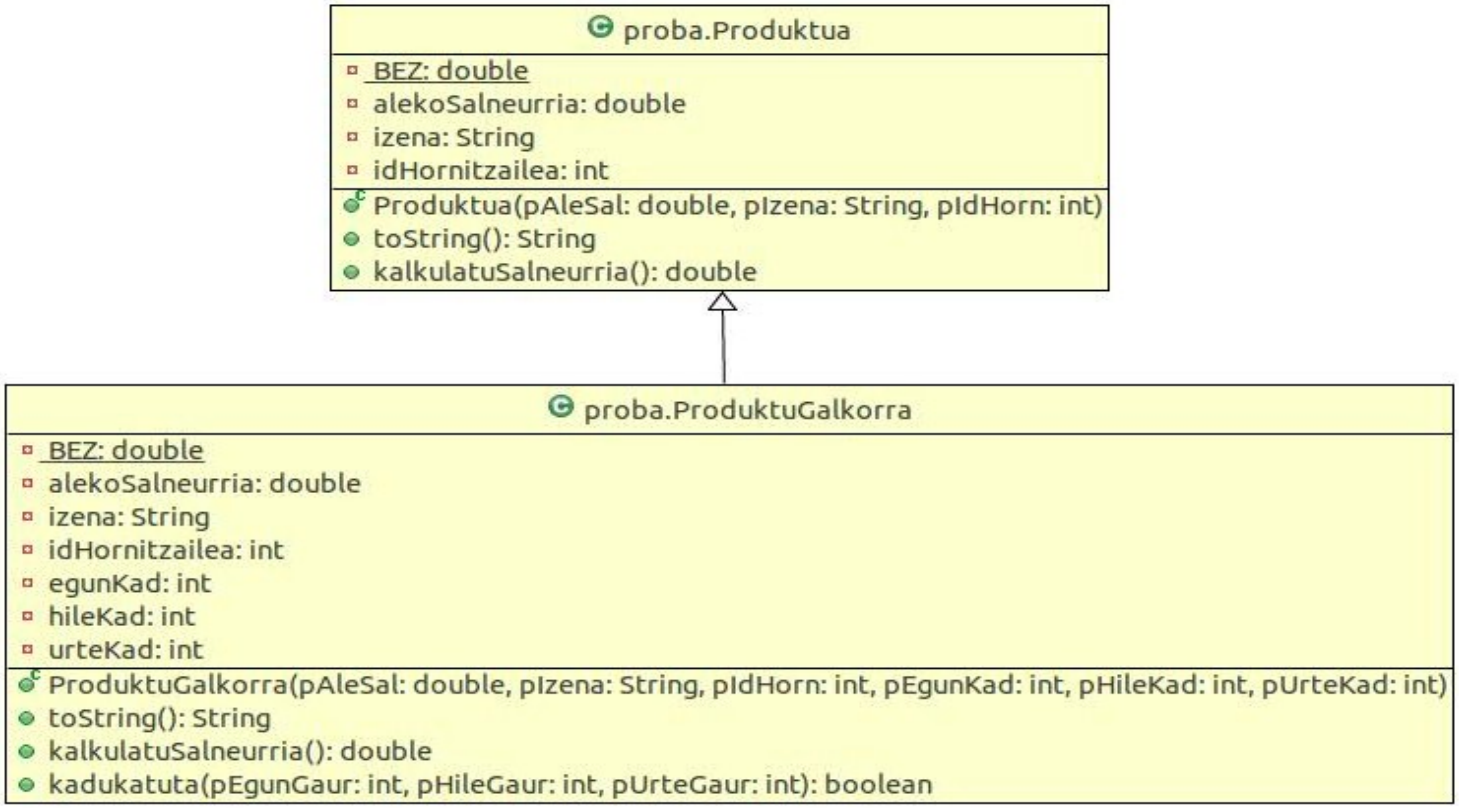
- Klase batean heredatutako metodo bat egikaritzen denean, konpiladoreak inplementazio lokaleena egikaritzen du (hau da, klasean ez badu topatzen, gorago doa, ez badu topatzen gorago, … Objekt arte)
  - Beraz, ez du zertan klasean, ez bere ama klasean egon behar… zuhaitzean nunbait egon behar da.

Adi, galdera

- Ikusi dugun bezala, heredatzen diren metodoak gainidaz daitezke; baina, zentzua al du heredatzen diren atributuak gainidaztea?

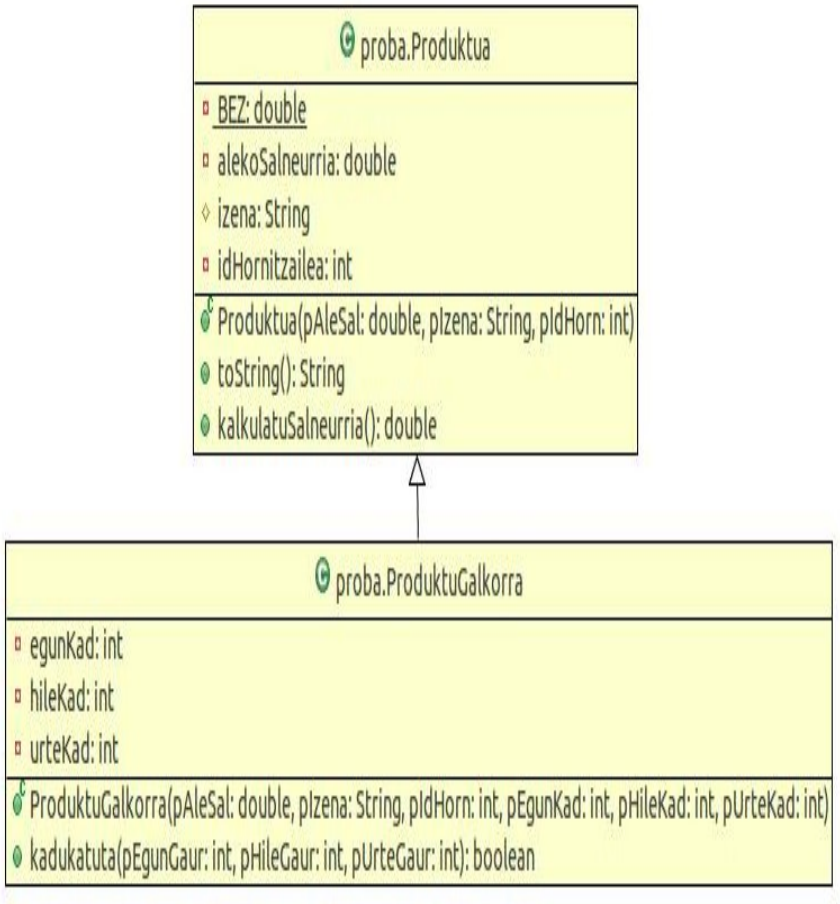
Adibidea

- ProduktuGalkor bat Produktu BAT-DA
  - Biek, atributu eta metodoak konpartitzen dituzte, baina produktu galkor batek atributu berezi batzuk ditu



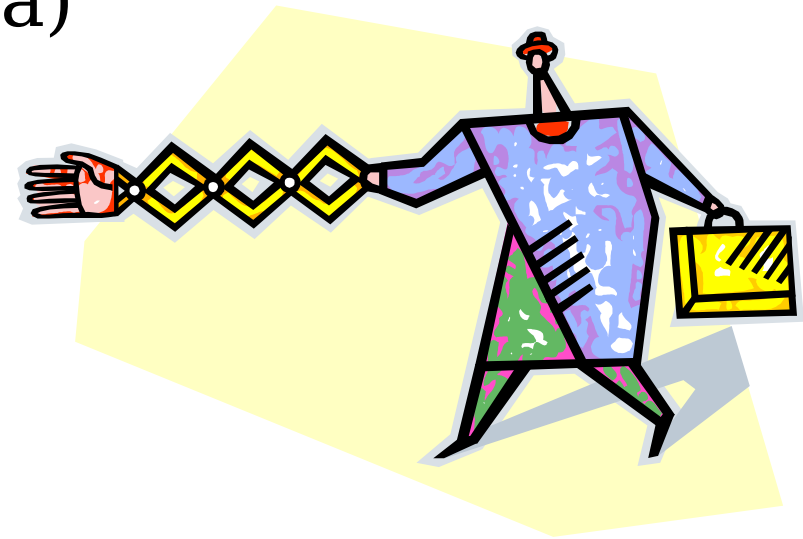
Adibidea (jarraipena)

- Horrela ProduktuGalkorra klasean, bakarrik klase honentzat bereziak diren atributuak eta metodoak gehitu beharko genituzke, besteak herentzia dela medio Produktutik jasoko lituzke



extends hitz berezia

- herentzia inplementatzeko erabiltzen da
  - Defektuz, sortzen dugun klase bakoitza *extends Object* da (gogoratu Objekt dela superklaserik gorena)



ProduktuGalkorra

```
public  class ProduktuGalkorra extends Produktu
{
//atributuak
 private int egunKad;
 private int hileKad;
 private int urteKad;
  // eraikitzailea
  public ProduktuGalkorra(double pSal, String pIzen, int pIdHorn, int pEgunKad, int pHileKad,
                                                                                  int pUrteKad)
  {
      super (pSal, pIzen, pIdHorn);

      this.egunKad =pEgunKad ;
      this.hileKad =pHileKad ;
      this.urteKad =pUrteKad ;
  }
```

Eraikitzailea

- Azpiklasearen eraikitzaileak, *nahita nahiez*, superklasearen eraikitzaileari deitu behar dio (*super()*)

- Super() eraikitzailea deitu ostean, eta beharrezkoa balitz, bereziak diren atributuak hasieratu beharko lituzke

Inplementazioa

- Bakarrik inplementatu behar dira, azpiklasean *bereziak* diren metodoak eta gainidatzi nahi direnak, heredatzen diren bezala erabilgarriak ez direlako

```
public boolean kadukatua(int pEgungoEguna, int pEgungoHilea, int pEgungoUrtea)
 {    boolean emaitza= false;
       if (pEgungoUrte> this.kadUrte) { emaitza = true; }
       else{if (pEgungoUrte==this.kadUrte && pEgungoHile> this.kadHile) { emaitza = true; }}
            else{if (pEgungoUrte== this.kadUrte && pEgungoHile== this.kadHile &&
                      pEgungoEgun> this.kadEgun ) { emaitza = true; } }
      return emaitza;}
```

Adi, galdera

- ProduktuGalkorra-ren metodoen artean, zein komeni da gainidaztea? (*toString() adibidez?*)

- Zein litzateke bere inplementazioa?

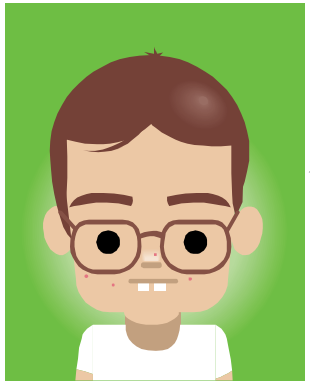
```
this.elementu
      vs.
super.elementu
```

Herentzia erabiltzearen abantailak

- Herentziari esker, kodearen bererabilpena ahalbidetzen (heredatzen delako) **bikoizketa saiheztuz.**
  - Heredatzen diren metodoak erabili nahi badira dauden bezala ez dira berriro inplementatu behar

Ikusgarritasuna eta herentzia

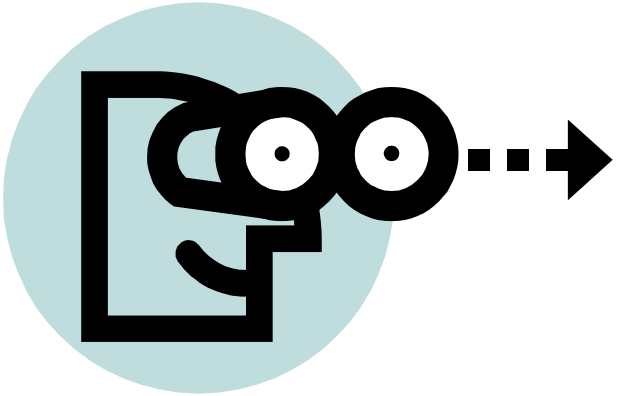
- Zelan atzitu heredatutako atributuak?
  - KONTUZ!!! private ez dira zuzenean atzigarriak



Superklasearen metodo pribatuekin gauza bera gertatzen da. Nahiz eta bere azpiklaseak heredatu, ezin dira erabili ez baitira ikusgarriak azpiklasetik

Orduan, zer?

- Metodoentzat (atributuetan ere aplikagarria)
  - *protected* ikusgarritasuna erabili, *private* erabili beharrean

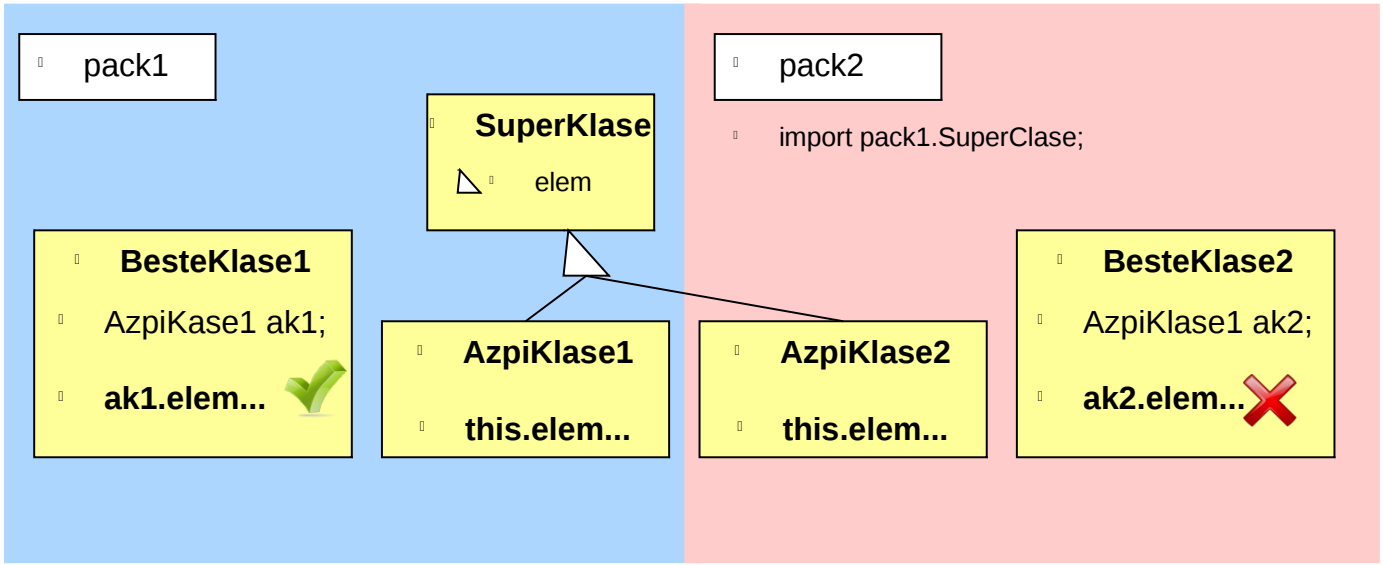


Protected ikusgarritasuna

- Elementua bera pribatua izaten jarraitzen du, baina klasearen ondorengoak (azpiklaseak) ikus dezaketa eta atzitu ere. Ez horrela, gainontzeko klaseak

KONTUZ!! *protected* ikusgarritasuna pakete desberdinen artean propagatu daiteke, eta horrek, batzuetan alde-ondorioak ekar ditzazke

Protected ikusgarritasuna




Galderarik?