



Programazioa

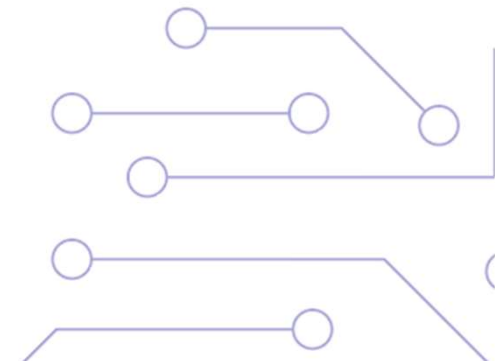
OBP Aurreratua

OBParen lau printzipioak: kapsulatzea

- Programa batean elkarreragiten duten objektuen artean ikusgarritasuna edo erabilgarritasuna mugatzea edo kontrolatzea da
 - Objektu batek beste objektu baten atributuak eta metodoak erabili ahal izango ditu bakarrik azken honetan adierazten den moduan
 - Atzipen baimenak minimora mantendu behar dira

Katua
- energia
- umorea
- loEgin()
- elikatu()
- miaukaEgin()

elikatu()
- energia++
- umorea++
- miaukaEgin()





OBParen lau printzipioak: abstrakzioa

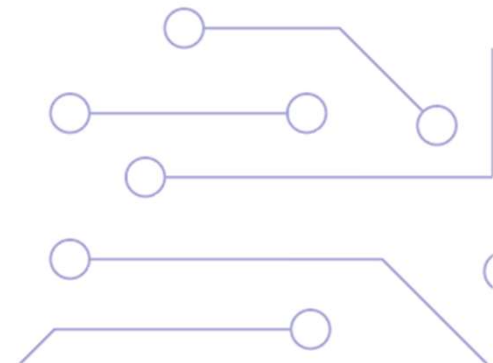
- Kapsulatzearen hedapen bat bezala uler daiteke
- Programa bat oso handia izan daiteke eta objektu askoren arteko elkarreraginak izan ditzake
 - Aldaketak egitea eta kodea mantentzea nekeza bihurtzen da
- Abstrakzioa aplikatzeak objektu bakoitzak maila altuko mekanismo bat baino ez duela azaldu behar erabiltzeko esan nahi du.
 - Mekanismo horrek barne-inplementazioaren xehetasunak ezkutatu beharko lituzke.
 - Beste objektuetarako garrantzitsuak diren eragiketak baino ez ditu erakutsi behar.

OBParen lau printzipioak: herentzia

- Batzuetan objektuak beraien artean oso antzekoak izan daitezke, bakoitzaren logika klase batean isolatuko dugu?

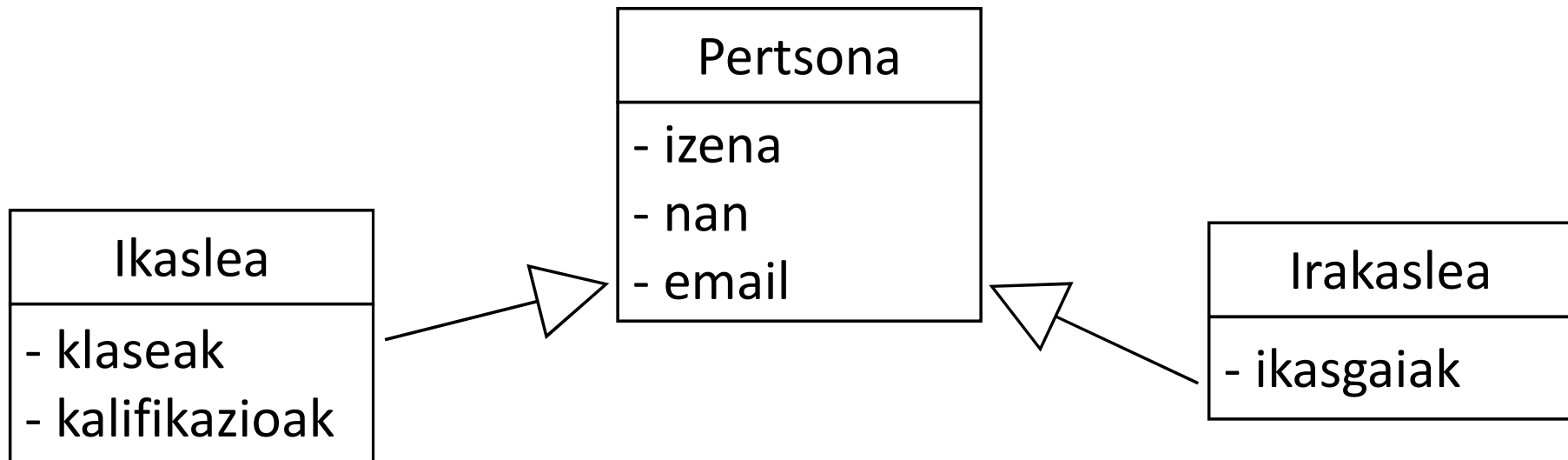
Ikaslea
- izena - nan - email - klaseak - kalifikazioak

Irakaslea
- izena - nan - email - ikasgaiak



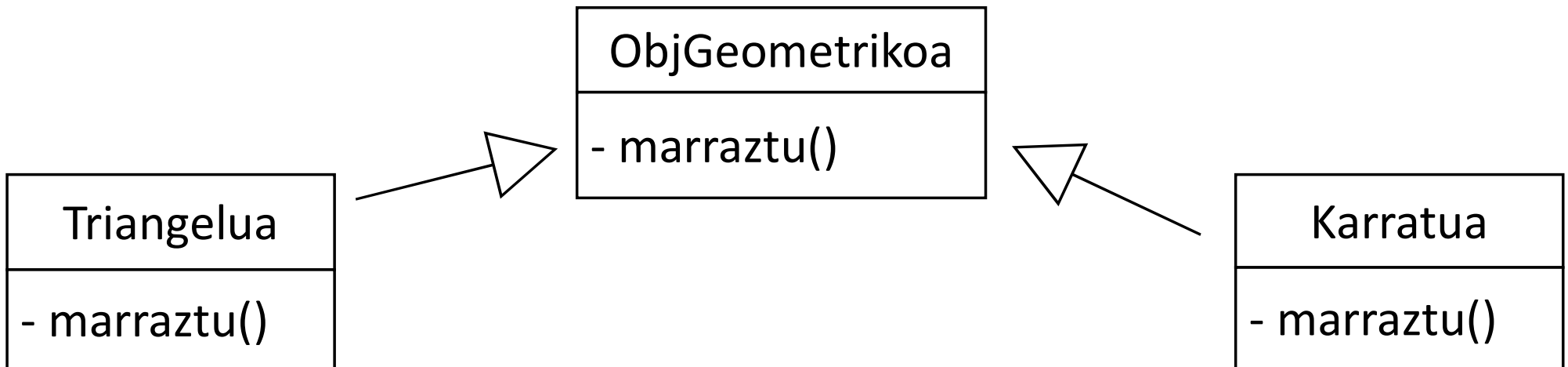
OBParen lau printzipioak: herentzia

- Amankomunak diren atributu eta metodoak klase batean definitu daitezke eta ondoren beste klaseek espezializatu daitezke
 - Atributu eta metodo berriak definitu ditzakete
 - Guraso klaseko metodoak berridatzi ditzakete



OBParen lau printzipioak: polimorfismoa

- Metodo baten deia deitu den hierarkiaren klasearen independentea da
 - Berdin zaigu guraso edo ondorengoen klase batean deitzea
 - Izen berdineko metodoek hierarkia bateko klaseetan modu desberdinetan funtzionatu dezakete



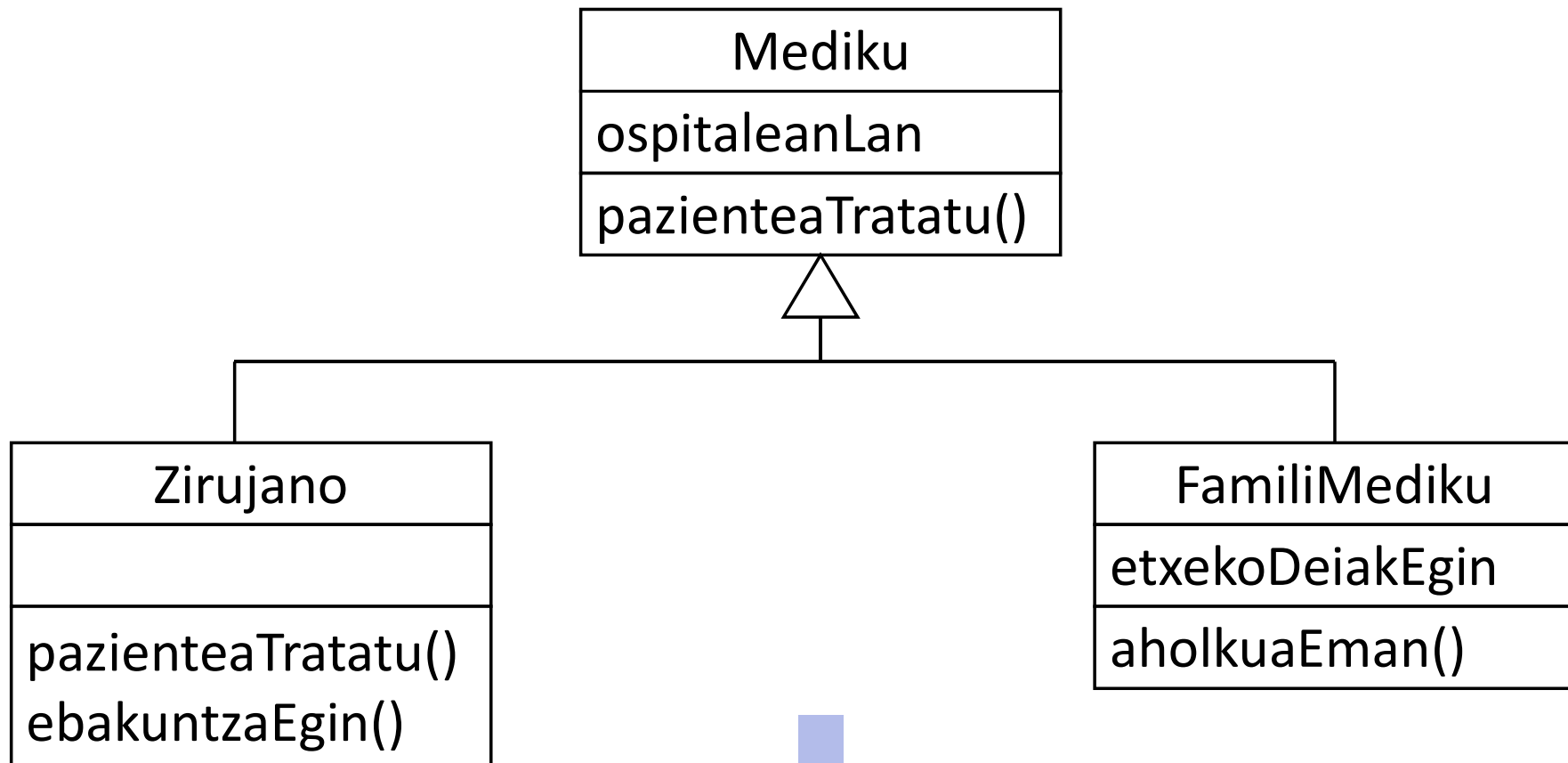


Herentzia

- Klase batean (**superklasea**) zehatzagoak diren klaseetako (**azpiklaseak**) ezaugarri amankomunak definitzeko erabiltzen da
- Azpiklase bakoitzak superklasea hedatzen (***extends***) du, honela:
 - Superklasearen atributu eta metodoak heredatzen ditu
 - Heredatutakoez gain bere atributu eta metodoak izan ditzake
 - Superklasetik heredatutako metodoak berridatzi ditzake, hurrengoa betetzen bada:
 - Metodo izen eta parametro berdinak izan behar ditu
 - Itzulera balioaren mota (baldin badu) mota berdinekoa edo mota horren azpiklasea izan behar du
 - Ezin du ikusgarritasun (edo atzipen) maila txikiagoa izan (honen inguruan gehiago aurrerago)



Herentzia





Herentzia

```
public class Mediku{
    boolean ospitaleanLan;
    public void pazienteaTratatu() {}
}
public class FamiliMediku extends Mediku{
    boolean etxekoDeiakEgin;
    public void aholkuaEman() {}
}
public class Zirujano extends Mediku{
    public void pazienteaTratatu() {}
    public void ebakuntzaEgin() {}
}
```

Atributu eta metodo
bat gehitzen ditu

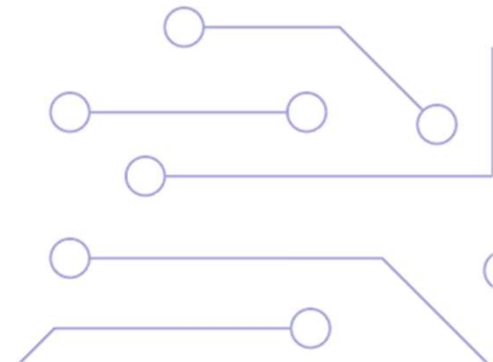
Metodo bat gehitzen du
eta bestea berridazten du





Metodoen gainkarga


- Parametro desberdineko eta izen berdineko hainbat metodo izateari deitzen zaio
 - Erabiltzaileak behar duen arabera metodo bat edo beste erabiliko da
- Ezaugarri batzuk ditu:
 - Parametro desberdinak izan behar dituzte
 - Itzulera balioaren mota aldatu daiteke, baina ezin da izan aldatzen den gauza bakarra kasu horretan
 - Ikusgarritasuna aldatu daiteke





Metodoen gainkarga


```
public class gainkarga1{  
  
    public int batu(int a, int b){  
        return a + b;  
    }  
  
    public double batu(double a, double b){  
        return a + b;  
    }  
}
```





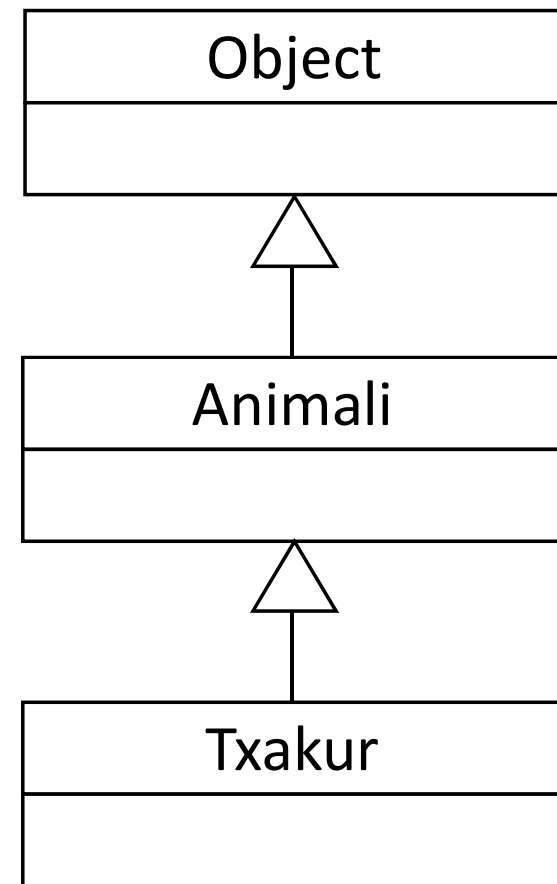
Metodoen gainkarga

```
public class gainkarga2{  
    String id;  
    public void ezarriId(String nireId) {  
        this.id = nireId;  
    }  
    public void ezarriId(int zenb) {  
        this.id = String.valueOf(zenb);  
    }  
}
```



Herentzia eta eraikitzaileak


- Klase bakoitzak eraikitzaile bat du
- Objektu baten instantzia sortzen denean (*new* hitza erabilita) hierarkia osoko klaseen eraikitzaileei deitzen zaie





Herentzia eta eraikitzaileak

```
public class Animali{
    public Animali(){
        System.out.println("Animali baten sorrera");
    }
}
public class Txakur extends Animali{
    public Txakur(){
        System.out.println("Txakur baten sorrera");
    }
}
public class ProbaTxakur{
    public static void main(String[] args){
        Txakur tx = new Txakur();
    }
}
```





Herentzia eta eraikitzaileak

- Zuzenean superklase baten eraikitzailea erabili nahi bada, **super** hitz erreserbatua erabiltzen da, ez superklasearen eraikitzailearen izena
 - *super* metodoaren deia eraikitzailearen lehen agindua izan behar da, eta jartzen ez bada konpiladoreak gehitzen du

```
public class Ahate extends Animali{  
    int tamaina;  
    public Ahate(int tam){  
        Animali();  
        tamaina = tam;  
    }  
}
```

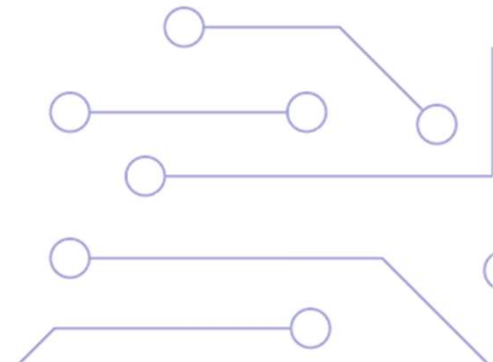
```
public class Ahate extends Animali{  
    int tamaina;  
    public Ahate(int tam){  
        super () ;  
        tamaina = tam;  
    }  
}
```



Herentzia eta eraikitzaileak

- Superklasearen eraikitzaileak parametroak baditu, hauek *super* deian pasatu behar zaizkio

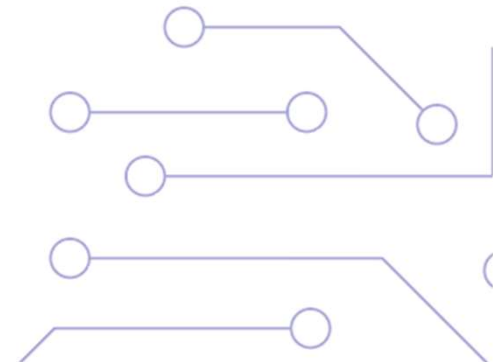
```
public class Animali{  
    String izen;  
    public Animali(String iz){  
        this.izen = iz;  
    }  
}  
public class Txakur extends Animali{  
    public Txakur(String iz){  
        super(iz);  
    }  
}
```





Herentzia eta eraikitzaileak

- Klase batean parametro desberdineko eraikitzaile bat baino gehiago badago (gainkarga) **this** metodoa erabili daiteke kodea berrerrabiltzeko
 - Dei hau eraikitzaileko lehena izan behar da, eta erabiltzen bada, ezingo da *super* deia erabili
 - Metodo honek *super* metodoaren antzera parametroak jaso ditzake





Herentzia eta eraikitzaileak

```
public class Mini extends Kotxe{  
    String kolore;  
    public Mini(){  
        this("gorria");  
    }  
    public Mini(String kol){  
        this.kolore = kol;  
    }  
}
```