

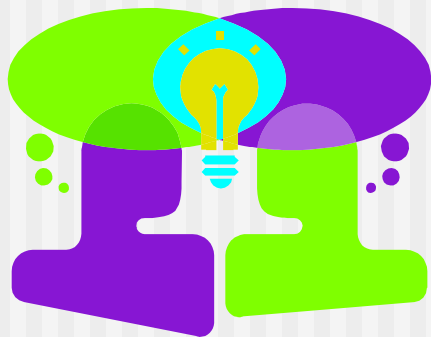
Aurkibidea

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

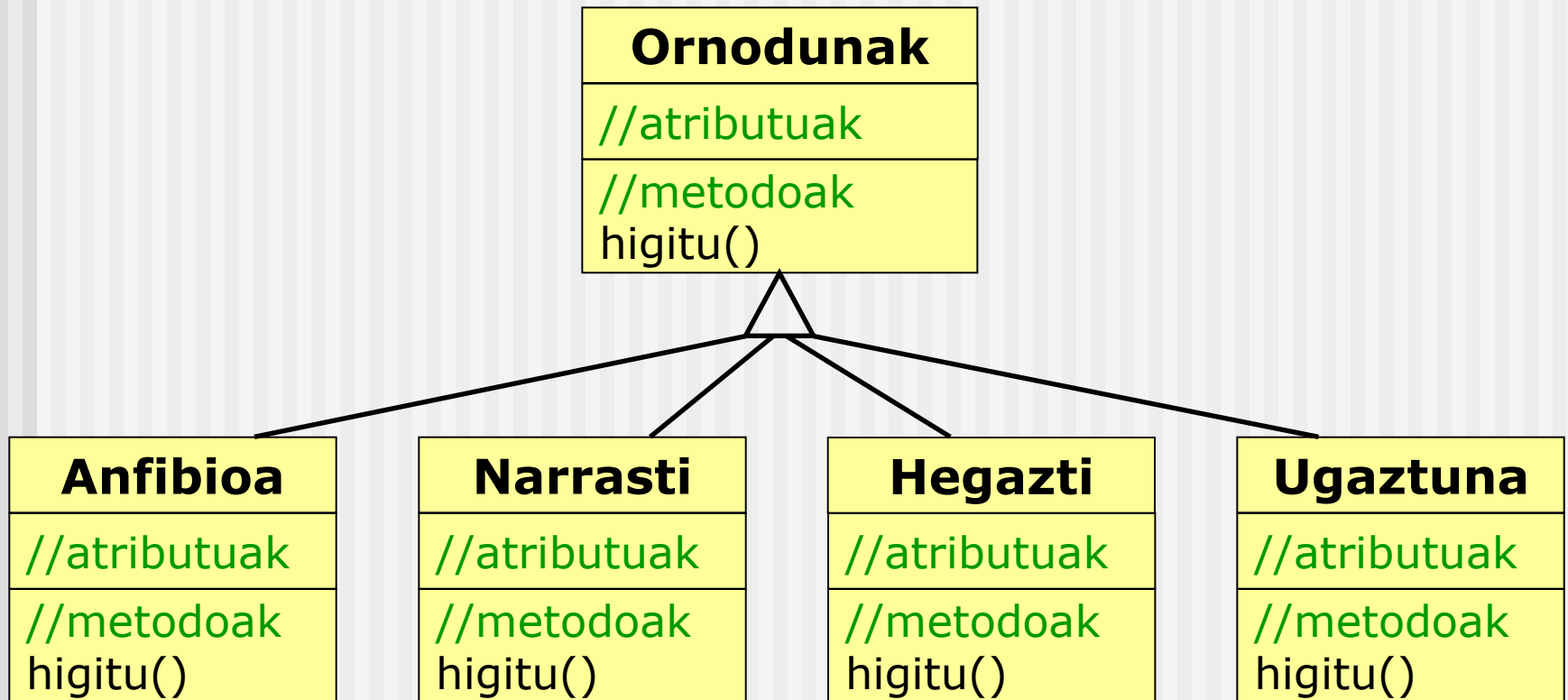


Polimorfismoa

- Metodo berdinak → jokaera desberdinak
 - ❖ Lotura dinamikoa eta gainidazketa konbinatuz lortzen da
 - Klase desberdinetako objektuak (herentziaren bidez erlazionatuta) metodo bera egikaritzeko ahalmena dute, non metodoak klase desberdinetan klaseari lotutako jokaera berezia izango duen



Adibideak:Ornodunak



Adibidea:

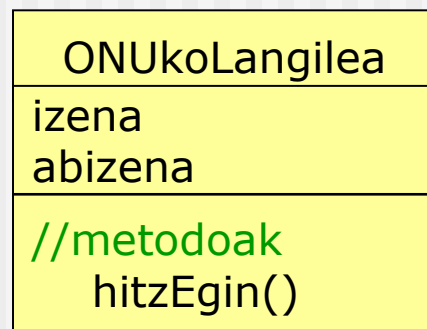
ONU/NBE-ko langileak

- NBE → herrialde desberdinetako langileak
 - ❖ Onuko langile superklaseak *hitzEgin()* metodo bat izango du → soinuak atera hizkuntza konkretu batean

```
public class ONUkoLangilea
{
    private String izena;
    private String abizena;
    public ONUkoLangilea(String pIzen, String pAbizena)
    {
        this.izena = pIzena;
        this.abizena = pAbizena;
    }
    public void hitzEgin() { ..... }
    // momentuz ez dugu implementatuko
    ...
}
```

Azpiklaseak

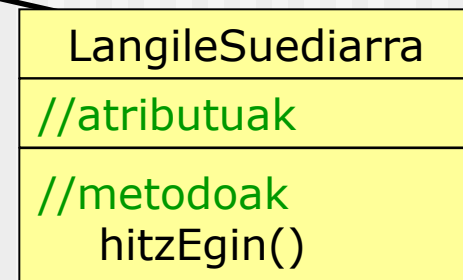
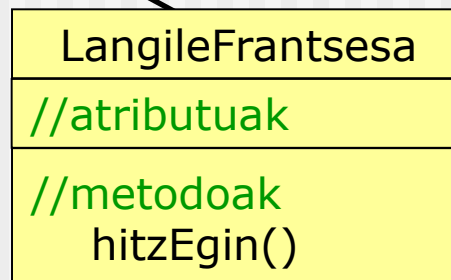
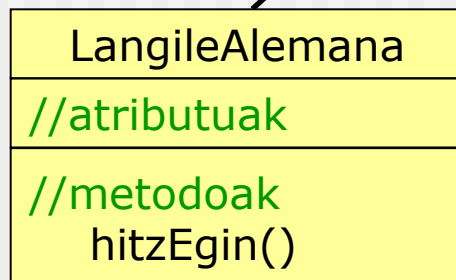
- *hitzeEgin()* metodoa gainidazten dute



Je parle
Français!!!

Jag talar
Svenska!!!

Ich spreche
Deutsch!!!



Azpiklaseak

```
public class LangileAlemana extends ONUkoLangilea
{
    public LangileAlemana (String pIzen, String pAbizena)
    {
        super(pIzen, pAbizen);
    }
    public void hitzEgin() {System.out.println("Ich spreche Deutsch!!!");}
    ....
}
```

```
public class LangileSuediarra extends ONUkoLangilea
{
    public LangileSuediarra (String pIzen, String pAbizena)
    {
        super(pIzen, pAbizen);
    }
    public void hitzEgin() {System.out.println("Jag tallar Svenska!!!");}
    ....
}
```

```
public class LangileFrantsesa extends ONUkoLangilea
{
    public LangileFrantsesa (String pIzen, String pAbizena)
    {
        super(pIzen, pAbizen);
    }
    public void hitzEgin() {System.out.println("Je parle Français!!!");}
    ....
}
```

ListaONUkoLangileak

- ONUko langileen zerrenda gordeko du

```
public class ListaONUkoLangileak
{
    // atributua
    private ArrayList<ONUkoLangilea> lista;
    // eraikitzailea
    public ListaONUkoLangileak()
    { this.lista = new ArrayList<ONUkoLangilea>(); }
    // metodoak
    private Iterator<ONUkoLangilea> getIteradorea()
    { return this.lista.iterator(); }











    public void gehituLangilea(ONUkoLangilea pLang)
    { this.lista().add(pLang); }
    ...
}
```



ListaONUkoLangileak

➤ ArrayList<ONUkoLangilea>

❖ Nahiz eta 3 langile Aleman, langile frantses nahiz suediar gorde, (gogoratu hauek hedaduraz ONUkoLangilea direla)

									
Friedrik Schultz 	Julien Boullie 	Angela Merkel 	Simon Häes 	Jan Holmqvist 					
//atributuak	//atributuak	//atributuak	//atributuak	//atributuak					
//metodoak	// metodoak	// metodoak	// metodoak	// metodoak					

ListaONUkoLangileak

```
LangileAlemana tAle= new LangileAlemana("Friedrik", "Schultz");  
LangileAlemana tAle2 = new LangileAlemana("Angela", "Merkel");  
LangileAlemana tAle3 = new LangileAlemana("Simon", "Haës");  
LangileFrantsesa tFran= new LangileFrantsesa("Julien", "Boullié");  
LangileSuediarra tSue= new LangileSuediarra("Jan", "Holmqvist");
```

```
ListaONUkoLangileak OnuLangLista= new ListaONUkoLangileak();  
  
OnuLangLista.gehituLangilea(tAle);  
OnuLangLista.gehituLangilea(tFran);  
OnuLangLista.gehituLangilea(tAle2);  
OnuLangLista.gehituLangilea(tAle3);  
OnuLangLista.gehituLangilea(tSue);
```

Hitz egin dezatela (banan banan)

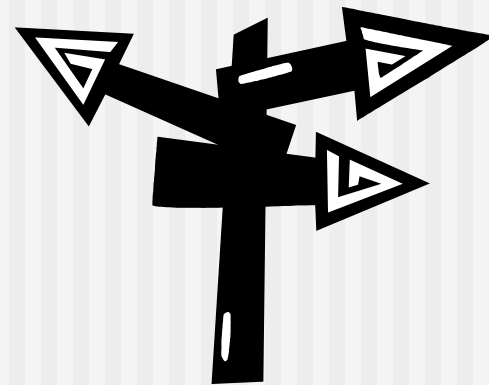
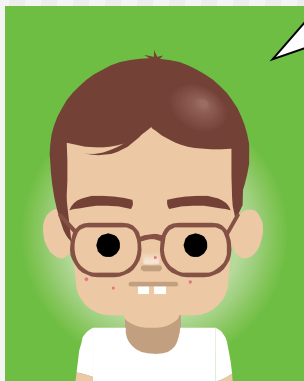
```
public class ListaONUkoLangileak
{ ...
    public void hitzEginBananBanan()
    {
        Iterator<ONUkoLangilea> itr = this.getIteradorea()
        ONUkoLangilea langileBat;
        while (itr.hasNext())
        {
            langileBat= itr.next();
            langileBat.hitzEgin(); // lotura dinamikoa
        }
    }
}
```



Lotura dinamikoa

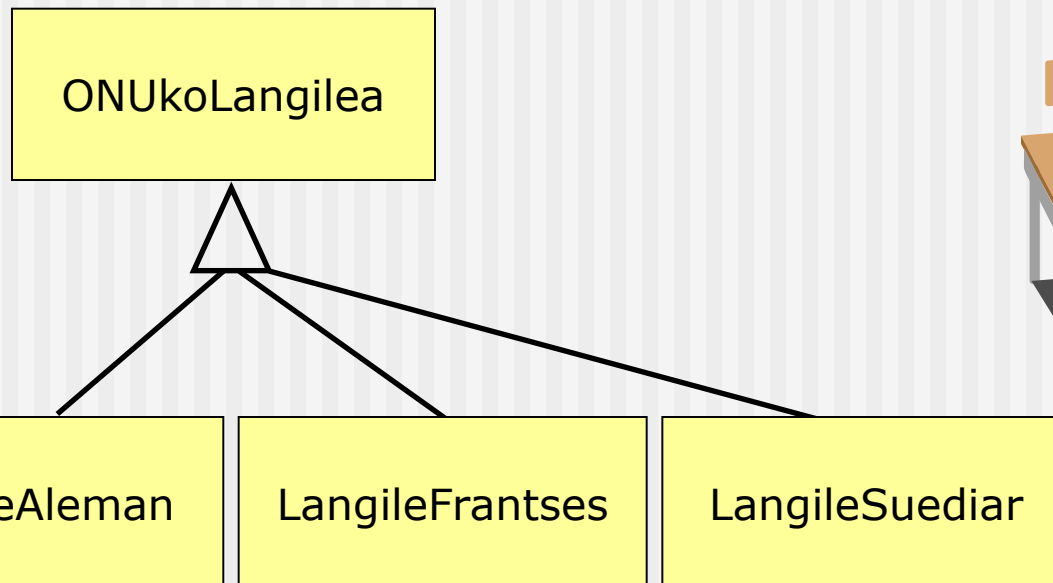
- Egikaritze momentuan, metodo egokia hautatzea ahalbidetzen du, objektuaren klasearen arabera

Berari esker, momentu bakoitzean konpilatzaileak badaki hitzEgin() metodo guztien artetik zein egikaritu behar duen.



1. Suposizioa

- Demagun, langile bakoitzak bere hizkuntzaz gain, ingelesez ere hitz egiten duela. Zer gertatuko litzateke ONUkoLangile klaseko *hitzeEgin()* metodoarekin?



Superklasearen metodoa

- ONUkoLangilea klaseko azpiklase guztiek *hitzEgin()* metodoa heredatuko dute. Metodo honen bitartez, “defektuz” langile guztiek ingelesez hitzegiten dute.

```
public class ONUkoLangilea
{
    ...

    public void hitzEgin()
    {
        System.out.println("I speak English!!!");
    }
}
```

Azpiklaseen metodoak

- Heredatutako *hitzEgin()* metodoa gainidazten dute, baina aldi berean erabili egiten dute

```
public class LangileAlemana extends ONUkoLangilea  
{.... public void hitzEgin()  
    { super.hitzEgin();  
      System.out.println("Ich spreche Deutsch!!!");}  
.... }
```

```
public class LangileSuediarrra extends ONUkoLangilea  
{.... public void hitzEgin()  
    { super.hitzEgin();  
      System.out.println("Jag tallar Svenska!!!");}  
.... }
```

```
public class LangileFrantsesa extends ONUkoLangilea  
{.... public void hitzEgin()  
    { super.hitzEgin();  
      System.out.println("Je parle Français!!!");}  
.... }
```

Hitz egin dezatela!

**I speak English!!!
Ich spreche Deutsch!!!**



ONUkoLangilea
//atributuak
//metodoak hitzEgin()

LangileAlemana
//atributuak
//metodoak hitzEgin()

LangileFrantsesa
//atributuak
//metodoak hitzEgin()

LangileSuediarrra
//atributuak
//metodoak hitzEgin()



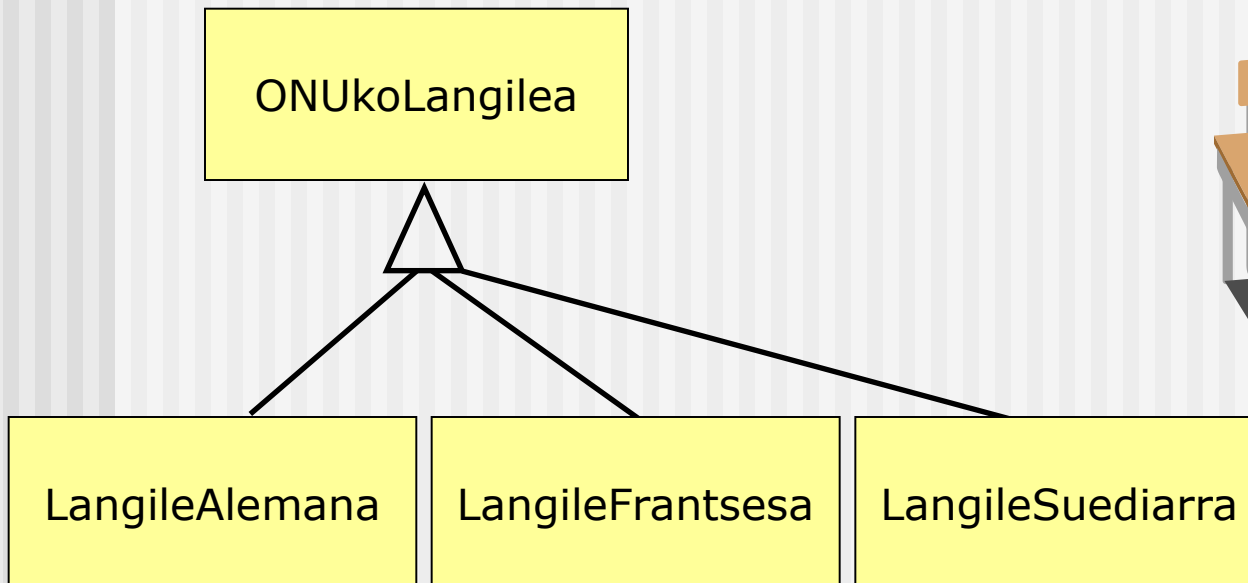
**I speak English!!!
Je parle Français!!!**

**I speak English!!!
Jag talar Svenska!!!**



2. Suposizioa

- ONUko langile batek bakarrik bere ama hizkuntza hitz egingo balu, zer gertatzen da *hitzeEgin()* metodoarekin?



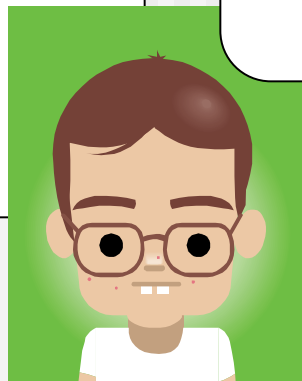
Metodo abstraktoak

- Klase batetan definitzen dira, baldin eta klase horren azpiklaseak metodo hori izatera behartu nahi badugu.
 - ❖ Baina, metodoa definitzen duen klaseak, ez du metodoa inplementatzen

```
public abstract class ONUkoLangilea
{
    ...

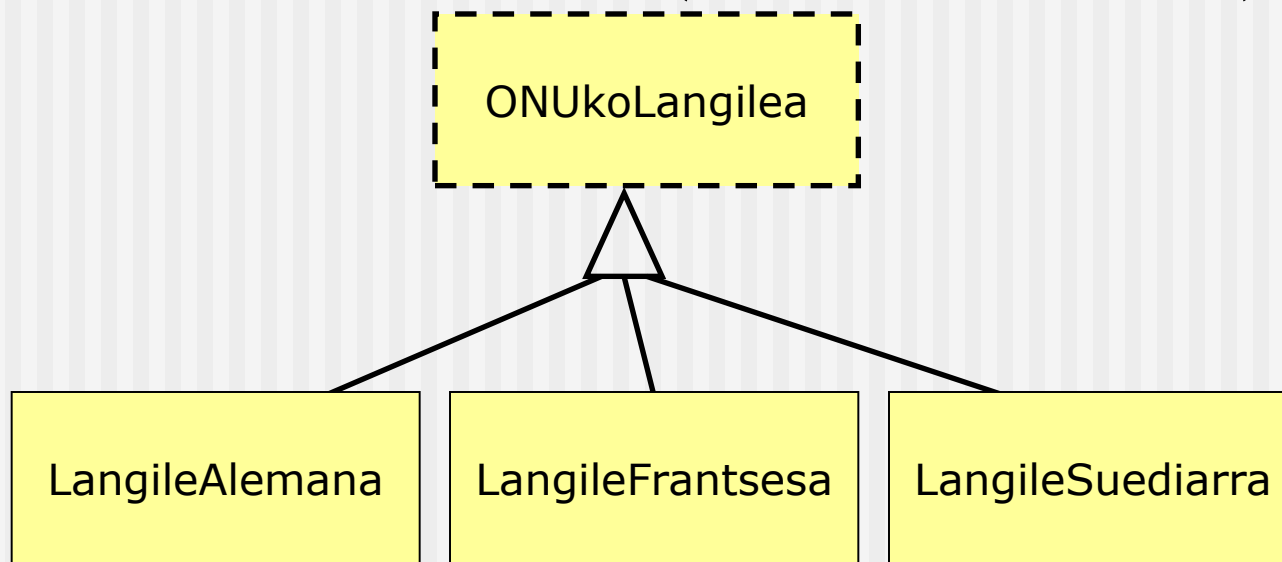
    public abstract void hitzEgin();
}
```

Klase batek metodo abstraktu bat duenean, klasea abstraktua izan behar da, nahi ta nahi ez.



Klase abstraktuak

- Ezin dira zuzenean instantziatu (*new* bitartez)
 - ❖ Ala ere, eraikitzailea inplementatu beharra dago
- Baina deklaratu eta erabil daitezke mota horretako erakusleak (lotura dinamikoa)

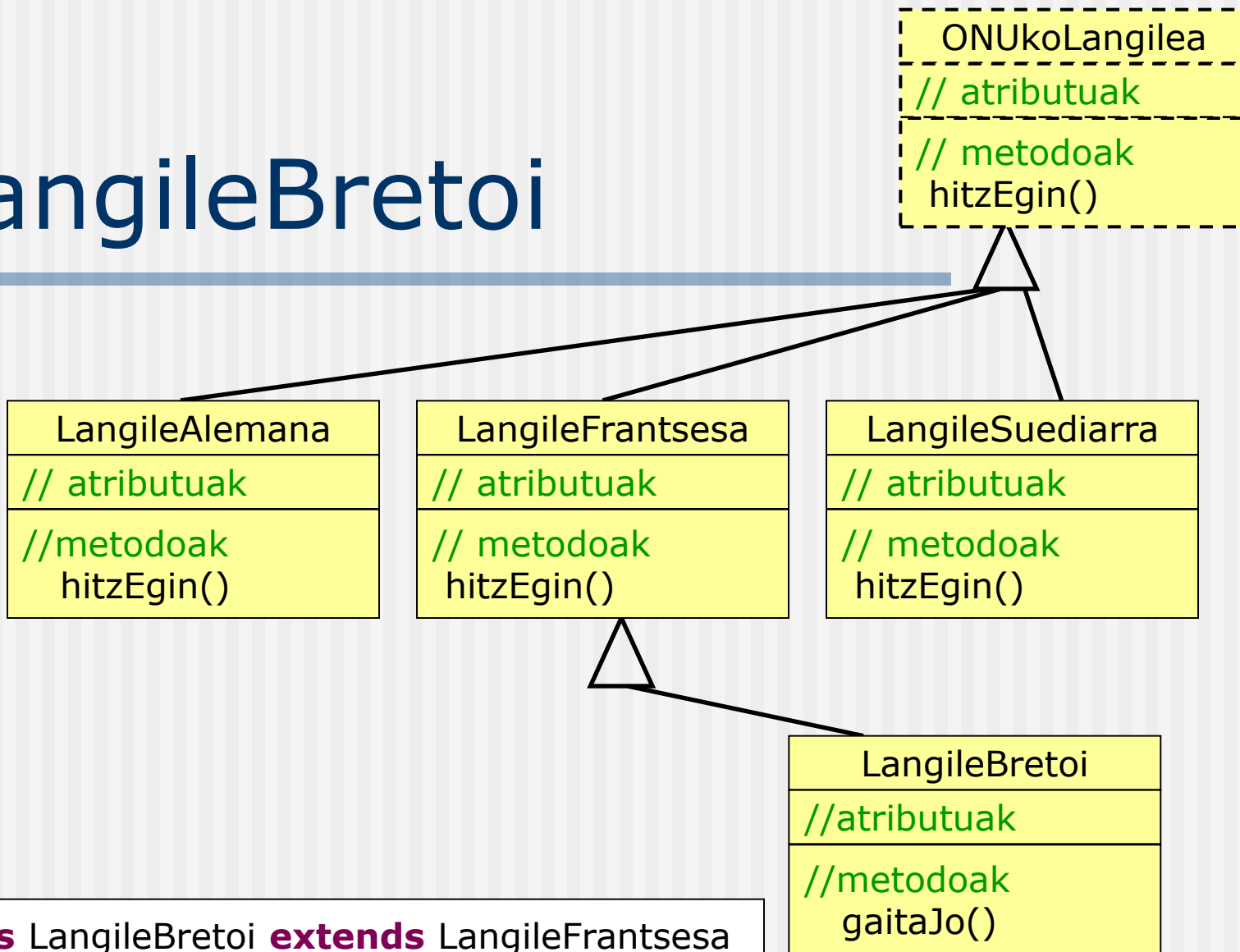


Adi!! galdera

- Egon al daiteke klase abstraktu bat, metodo abstrakturik gabe? Ze helbururekin?
 - ❖ Zergatik?



LangileBretoi



```
public class LangileBretoi extends LangileFrantsesa
{
    ...
    public void gaitaJo();
}
```

eta *hitzEgin()*
metodoa?

Frantsesa vs. Bretoi

```
LangileFrantsesa tFran;  
tFran = new LangileFrantsesa();
```

```
LangileBretoi tBret;  
tBret = new LangileBretoi();
```

tFran

LangileFrantsesa

//atributuak

//metodoak
hitzEgin()

tBret

LangileBretoi

//atributuak

//metodoak
gaitaJo()



tBret = tFran;

Konpilazio errorea: *cannot convert from
LangileFrantsesa to LangileBretoi*



tFran = tBret;

Egokia, baina jarraian egiten dugunarekin
Kontuz ibili beharko gara

Mota estatikoa vs dinamikoa

- Mota estatikoa (deklarazioan): *LangileFrantsesa*
- Mota dinamikoa (egikariketan): *LangileBretoi*

```
tFran = tBret;
```

tFran

LangileBretoi
//atributuak
//metodoak gaitaJo()

```
tFran.hitzEgin();
```



**I speak English!!!
Je parle Français!!!**

(suposizio 1)

Je parle Français!!!

(suposizio 2)

Mota estatikoa vs dinamikoa

- Mota estatikoa (deklarazioan): *LangileFrantsesa*
- Mota dinamikoa (egikariketean): *LangileBretoi*

```
tFran.gaitaJo();
```

Konpilazio errorea: *The method gaitaJo() is undefined for the type LangileFrantzesez*



Nahiz eta logikoa iruditu, konpiladoreak ez du onartzen ezin duelako jakin erakusle bat dinamikoki ze nolako objektura apuntatuko duen. Bakarrik jakin dezake LangileFrantses-ak ez duela *gaitaJo()* metodoa espezifikatuta.

Orduan...?

- *downcasting* egin beharra dago
 - ❖ Baina egin baino lehenago, programatzaileak, erakuslea dinamikoki LangileBretoi objektu bati apuntatzen dagoela konprobatu behar du.

```
if (tFran instanceof LangileBretoi)
{
    ((LangileBretoi)tFran).gaitaJo();
}
```



***downcasting*-ak ez du objektua aldatzen. Objektua, beretik eratorritako klase baten mota berekoa izango balitz bezala interpretatzen du. Horregatik garrantzitsua da egin aurretik konprobatzea.**

Ariketa: Autopistak

Europistas S.A. enpresak ordain lekuetan erabiltzaileen ordainketak kudeatu nahi dituzte. Ibilgailu bakoitzak egin behar duen ordainketa kalkulatzeko hurrengo informazioa izango da kontutan: ibilitako kilometroak eta salneurria kilometroko. Ibitako kilometroak ibilgailuaren autopistako sarreraren eta autopistako irteeraren arabera kalkulatzen dira (sarrera kilometro kopuru batez identifikatzen da, baita irteera ere). Salneurria kilometroko 0,05 eurokoa da.

Oinarrizko ordainketa *Km kopurua x Salneurria kilometroko kalkulatzen da, baina hurrengo irizpideen arabera aldatu daiteke.*

Hasteko, bidaiari eta karga ibilgailuen arteko desberdintasuna egiten da.

Karga eramateko ibilgailuei, oinarrizko ordainketari kopuru finko bat gehituko zaio:

- 10 euro kasu honetan

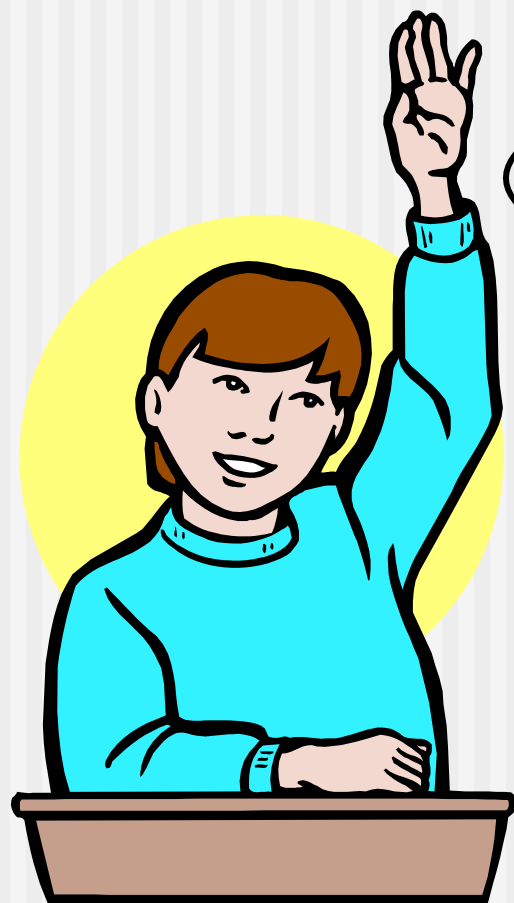
Bidaiariak eramateko ibilgailuek aldiz:

- Motorrei oinarrizko ordainketa mantentzen zaie.
- Kotxeei, oinarrizko ordainketari * 1,5 aplikatuko zaie.
- Autobusei oinarrizko ordainketari * 4 aplikatuko zaie. Horrez gain, eserleku bakoitzeko deskontu bat aplikatuko zaie; 0.05 eserleku bakoitzeko (Adibidez, 10 eserleku baditu, 50 zentimoko deskontua)

Eskatzen dena

- Irudikatu klase diagrama eta atributuak nahiz metodoak espezifikatu (informazioa modu optimo batean banatzeko)
- Inplementatu eraikitzaileak.
- Inplementatu *lortuOrdainketa()* metodoa, behar den klaseetan.





¿Galderarik?