GRAU D'ENGINYERIA INFORMÀTICA

PROGRAMACIÓ II

Bloc 2:

Programació Orientada a Objectes (3)

Laura Igual
Departament de Matemàtiques i Informàtica
Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

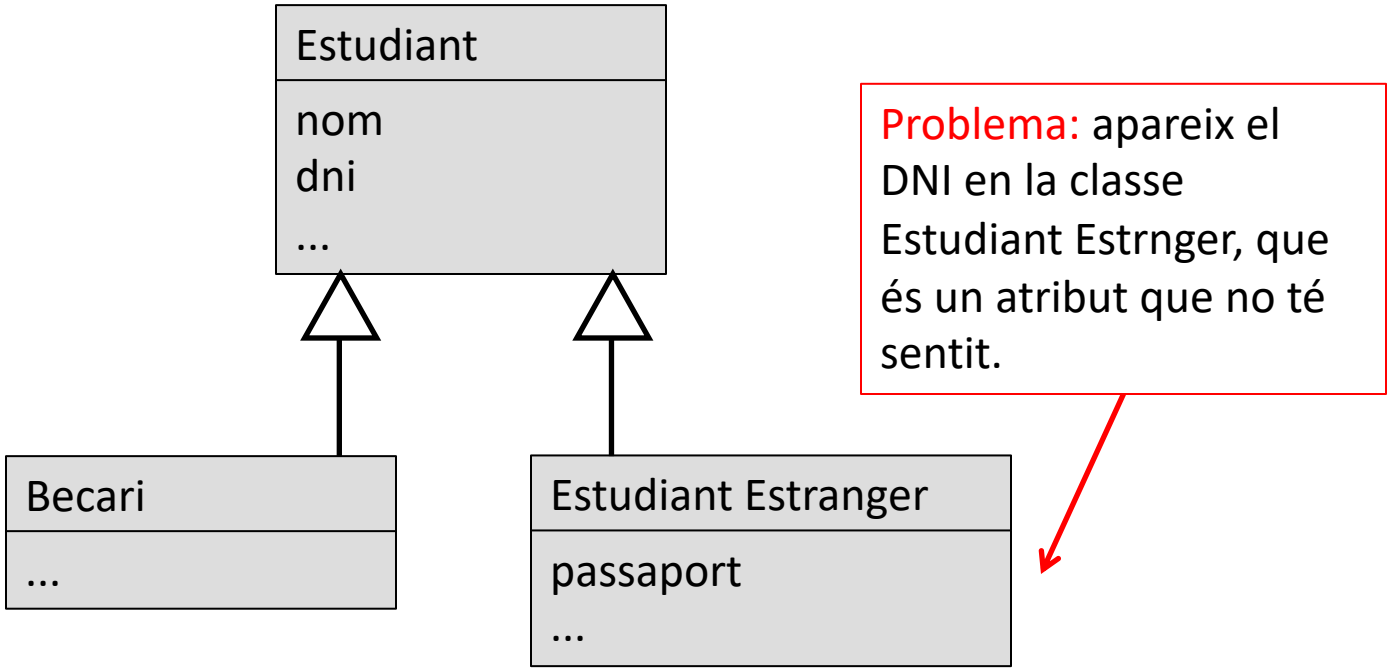
UNIVERSITAT DE BARCELONA

MÉS SOBRE HERÈNCIA I JERARQUIA DE CLASSES

Errors típics de l'herència

1. Creació de superclasses poc generals
2. Ús de subclasses en comptes d'una superclasse

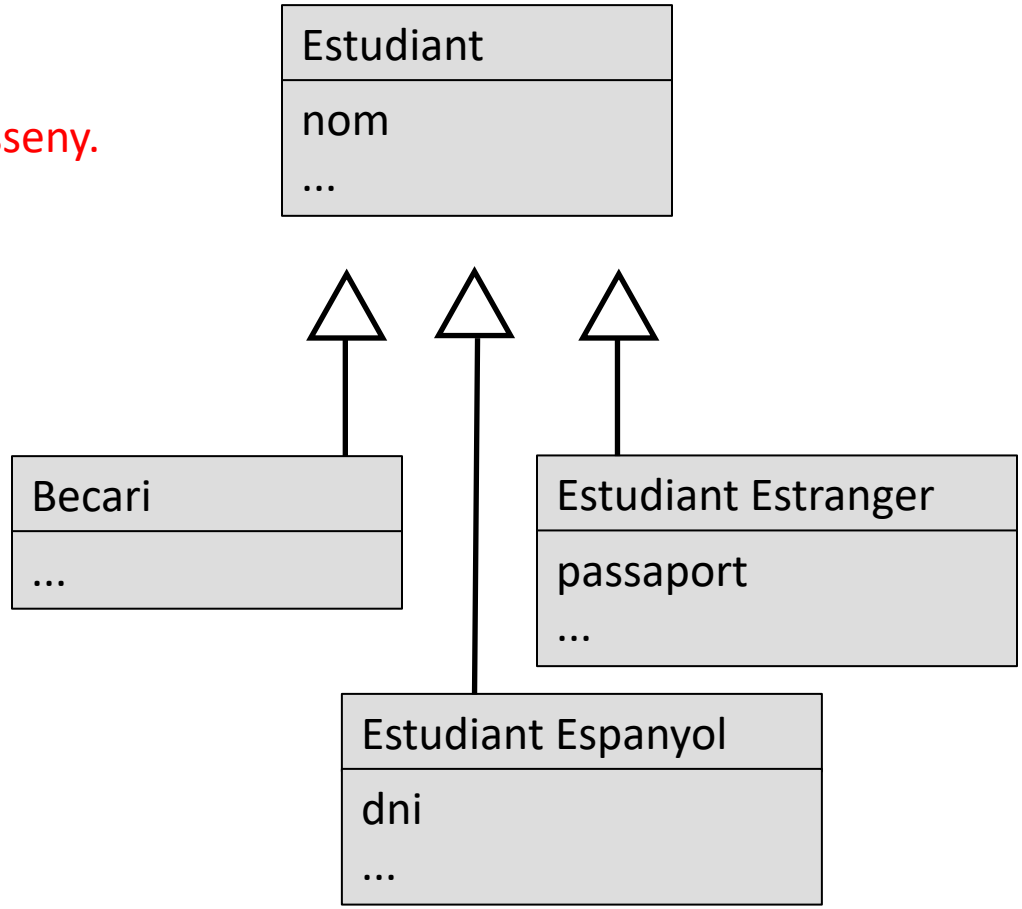
Creació de superclasses poc generals



Evitar-ho al disseny.

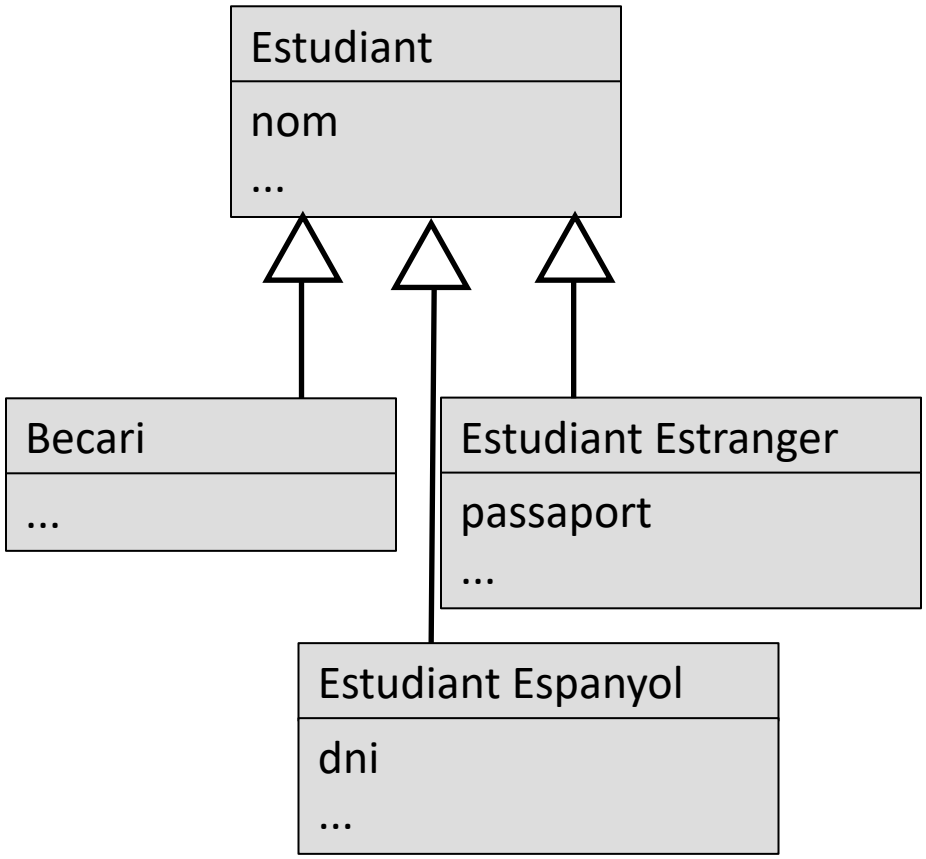
Creació de superclasses poc generals

Canvi de disseny.



Ús de subclasses en comptes d'una superclasse

- Si volem emmagatzemar una relació d'estudiants, podem definir un vector per a emmagatzemar les instàncies de diferents tipus d'estudiants: Becari, Estudiant Estranger,...
- En recuperar-los, com que poden estar barrejats, hem d'utilitzar la superclasse Estudiant, però si volem accedir a mètodes definits en la subclasse hem d'utilitzar el càsting.



CONVERSIÓ DE DADES

Conversió de dades

- Algunes vegades es convenient convertir dades d'un tipus a un altre.
- Aquestes conversions no canvien el tipus de la variable o el valor que s'emmagatzema. Només converteixen el valor en aquella part del càlcul.

Conversió de dades

- Les conversions s'han de gestionar amb cura per evitar perdre informació.
- Widening conversions (conversions d'ampliació) són les més segures, ja que van de tipus de dades petit a gran.
- Narrowing conversions (conversions per reducció) poden perdre informació, ja que van de tipus gran a petit.

Conversió de dades: Casting

- *Casting* és la tècnica de conversió més poderosa i perillosa, ja que permet conversions d'ampliació i reducció.
- Per realitzar un ***cast***, el tipus es col·loca entre parèntesis davant del valor que es desitja convertir
- Exemple (Casting de tipus primitius):

```
int total=11;
int compt=2;
float resultat;
resultat = (float) total / compt;
```

Conversió de tipus

- Casting d'instàncies:

El càsting (o conversió de tipus) ens permet utilitzar una instància d'una classe com si es tractés d'una instància d'un altre tipus.

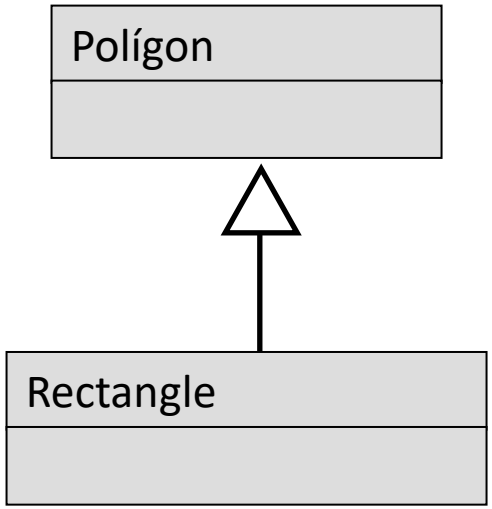
Tot i que la definició anterior es completament certa, cal matitzar-la, ja que podrem realitzar el procés de càsting sempre que la conversió sigui possible.

Conversió de tipus

- **<u>Conversió implícita:</u>** (automàtica)
  - *Tipus primitius* a un que suporti un rang major de valors

```
float saldo = 300;    //podem assignar-li un enter
int codi = 3.7;       //Donarà ERROR
```

  - *Referències*: tot objecte conté una instància de les seves superclasses
    - ***cast-up***
    - sempre vàlid




```
Poligon poligon;
Rectangle rectangle = new Rectangle();
poligon = rectangle;
```

Només podrem utilitzar com a tipus de classe de destinació, aquelles que siguin més genèriques que el tipus de classe d'origen.

Conversió de tipus

- **<u>Conversió explícita:</u>**
  - *Tipus primitius:* perdent informació

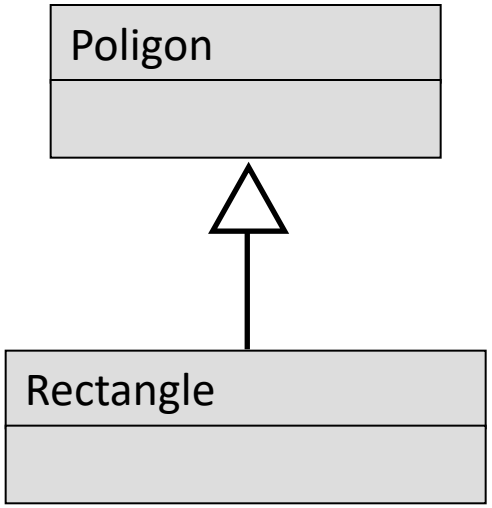
```
float f = 200,35;
int i = (int)f;
```

  - *Referències:* assignar a un objecte d'una subclasse un de la superclasse
    - ***cast-down*** o *narrowing*
    - No sempre vàlid
    - L'error es pot produir:
      - en temps d'execució (**`ClassCastException`**)
      - en temps de compilació si no és ni tan sols una subclasse.

Conversió explícita de referències

- Pot donar un error en execució:

```
Poligon [] poligons = new Poligon [30];
…
Rectangle r = (Rectangle)poligons[i];
```



- Donaria error en compilació:

```
CompteBancari c = (CompteBancari)poligons[i];
```

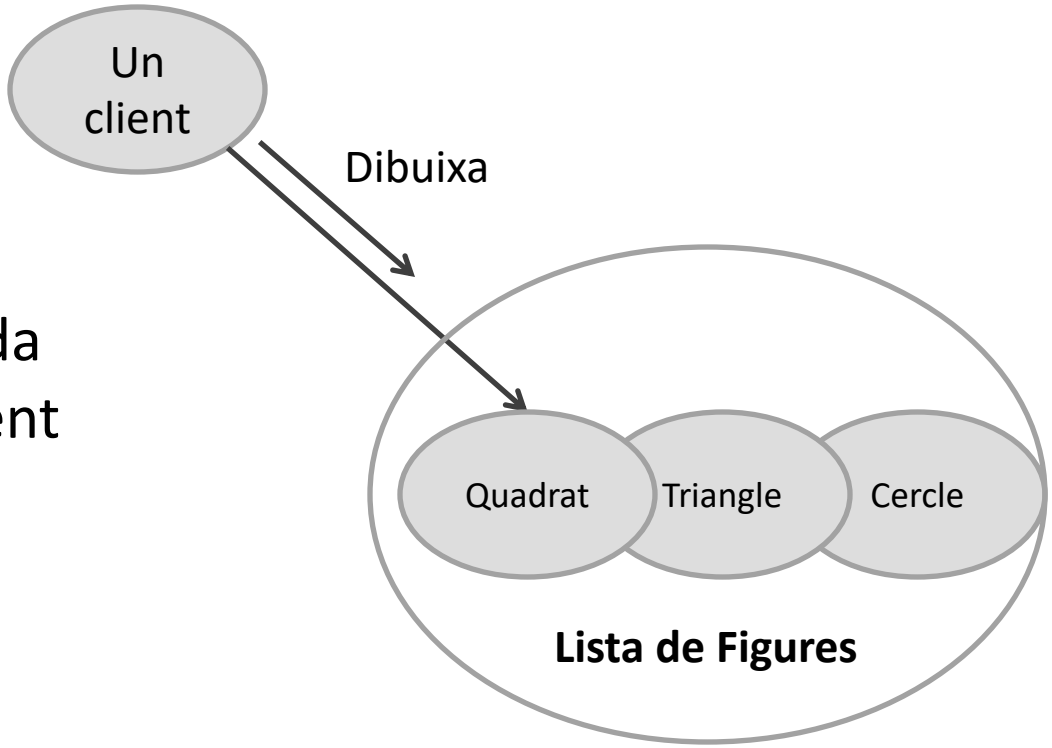
POLIMORFISME

Polimorfisme

- *Origen: poli (‘diversos’) i morfos (‘forma’)*
- “Capacitat per a adoptar diverses formes”
- El polimorfisme està lligat estretament amb l’herència
- És la propietat per la qual es poden realitzar tasques diferents invocant la mateixa operació, segons el tipus d’objecte sobre el qual s'invoca.

Polimorfisme

- El Polimorfisme provocarà un canvi de comportament d'una operació depenent de l'objecte al qual s'aplica.



- L'operació és única, però cada classe defineix el comportament d'aquella operació.

Polimorfisme

- És la propietat d'ocultar l'estructura interna d'una jerarquia de classes implementant un conjunt de mètodes de manera independent i diferenciada en cada classe de la jerarquia.
- El polimorfisme **apareix** quan definim un mètode en una classe de la jerarquia (generalment la superclasse) i el reescrivim en, com a mínim, alguna de les classes que formen la jerarquia.
  - Recordeu que la reescriptura o sobreescriptura de mètodes només pot existir en subclasses de la classe en què es defineix o implementa el mètode per primera vegada.

Polimorfisme

- El concepte de polimorfisme es pot aplicar tant a **mètodes** com a **tipus de dades**.
- Així neixen els conceptes de:
  - *Mètodes polimórfics,* són aquells mètodes que poden avaluar-se o ser aplicats a diferents tipus de dades de forma indistinta.
  - *Tipus polimòrfics*, són aquells tipus de dades que contenen al menys un element amb tipus no especificat.

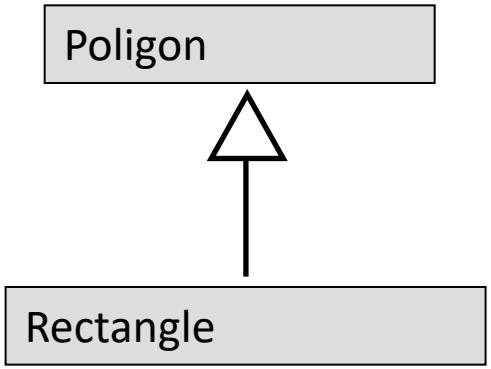
Polimorfisme

- Assignació polimorfa:

  Dues implementacions:

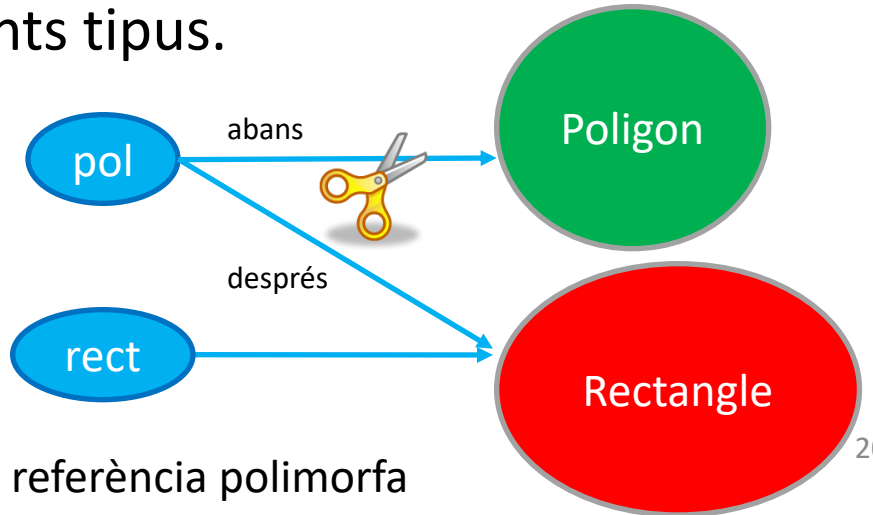
```
Poligon pol = new Poligon();
Rectangle rect = new Rectangle();
pol = rect;
```

```
Poligon pol = new Rectangle();
```



- Connexió polimorfa (assignació i passo de paràmetres): quan l'origen i el destí tenen diferents tipus.

- Que passa durant una connexió polimorfa?



Reconnexió de la referència polimorfa

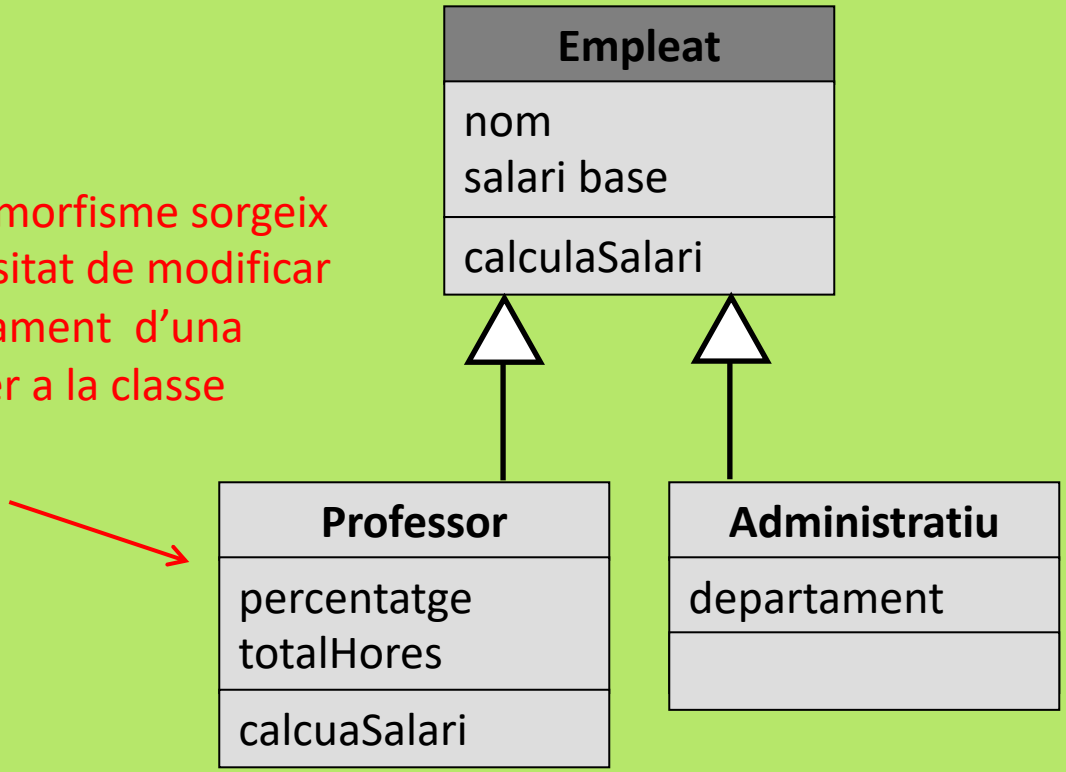
Polimorfisme

- El polimorfisme permet que es decideixi en temps d'execució i de manera automàtica quin dels mètodes cal executar: el mètode heretat o, en cas que existeixi, el mètode sobreescrit.

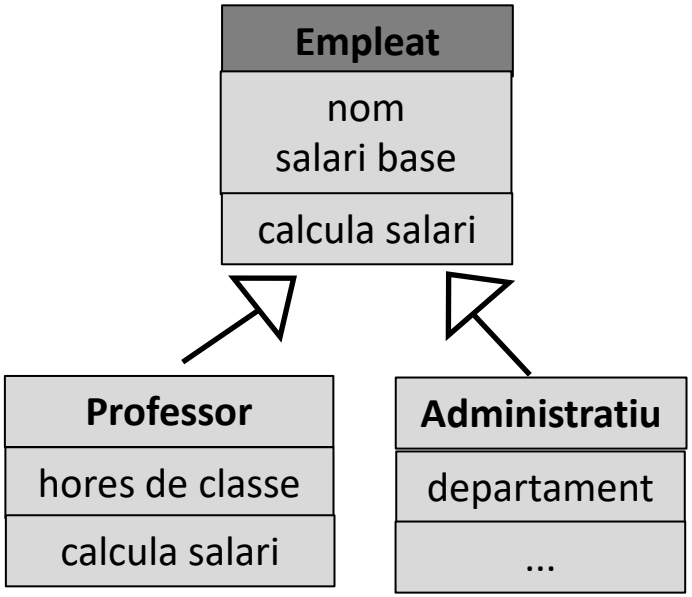
Exercici 1

- Implementació de l'exemple:

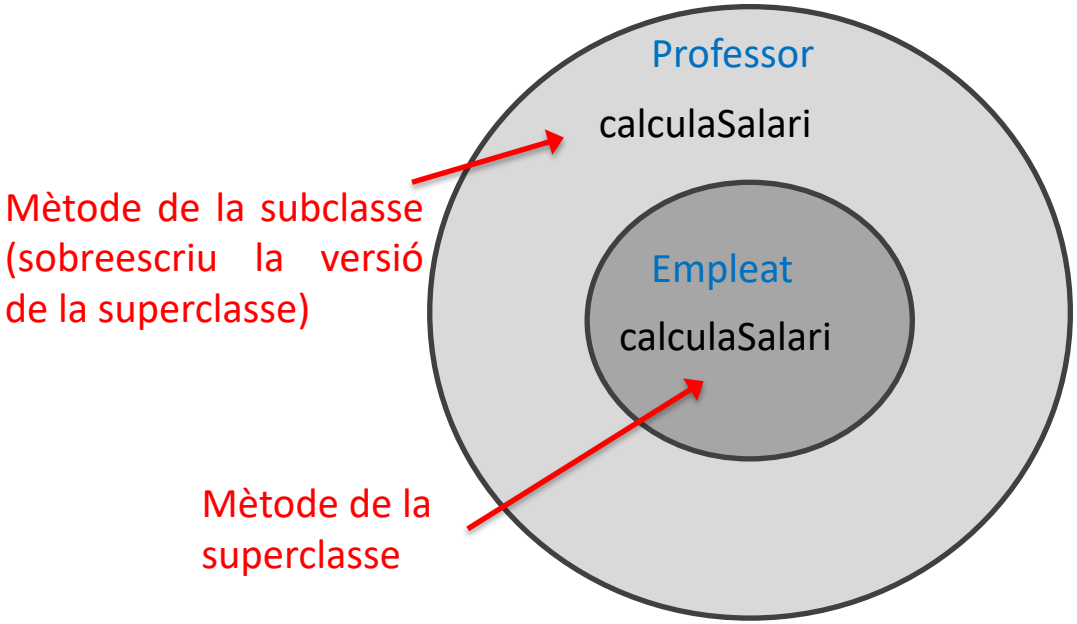
Aquí el polimorfisme sorgeix de la necessitat de modificar el comportament d'una operació per a la classe Professor



Detalls per l'exercici



Mètode sobreescrit en la subclasse Professor.
Mètode heretat en la subclasse Administratiu



Recorda que la paraula reservada **super** és una referència a la porció de la superclasse d'un objecte.

Quan el codi de la subclasse utilitza **super**, s'executarà la versió del mètode de la superclasse.

```
public abstract class Empleat{
    private String nom;
    private float salariBase;
    public Empleat( String nom, float salariBase) {
            this.nom = nom;
            this.salariBase = salariBase;
    }
    public String getNom() {
            return nom; }
    public float getSalariBase() {
            return salariBase; }
    public void setNom(String nom) {
             this.nom = nom; }
    public void setsalariBase(float salariBase ) {
            this.salariBase = salariBase; }
    public float calculaSalari() {
            float salari = (float) (salariBase * 1.5);
            return salari;
    }
}
```

Empleat.java

Administratiu.java

```
public class Administratiu extends Empleat {
    private String department;
    public Administratiu (String nom, float salariBase, String department) {
        super(nom, salariBase);
        this.department = department;
    }
    public String getDepartment() {
        return department;
    }
    public void setDepartment(String department) {
        this.department = department;
    }
}
```

Professor.java

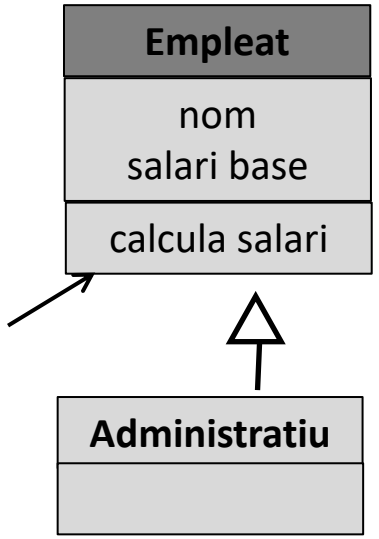
```
public class Professor extends Empleat {
    private float percentatge;
    private float totalHores;
    // constructor
    public Professor(String nom, float salariBase, float percentatge, float totalHores) {
        super(nom, salariBase);
        this.percentatge = percentatge;
        this.totalHores = totalHores;}
    // Getters i setters
    public float getPercentatge() {
        return percentatge;}
    public float getTotalHores() {
        return totalHores;}
    public void setPercentatge(float percentatge) {
        this.percentatge = percentatge;}
    public void setTotalHores(float totalHores) {
        this.totalHores = totalHores;}
    // Reescriptura del mètode calculaSalari
    public float calculaSalari() {
        return super.calculaSalari() + (percentatge * totalHores);}
}
```

Test.java

```
public class Test {
  public static void main(String[] args) {

     Empleat empleat;
     empleat = new Administratiu(“Lluís",1000, "dep");
      System.out.println("salari administratiu = " + empleat.calculaSalari());

  }
}
```



→ 1.500.0

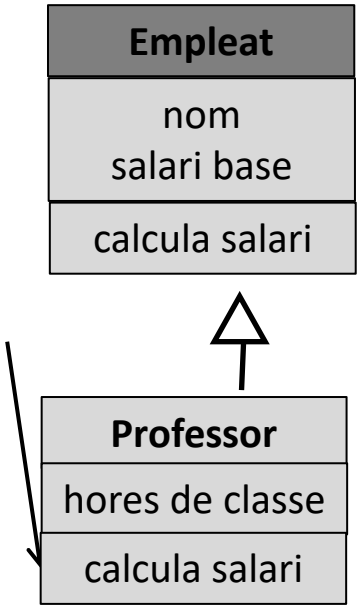
Una referència a un objecte de la subclasse Administratiu cridarà al mètode heretat.

Test.java

```
public class Test {
  public static void main(String[] args) {

     Empleat empleat;
     empleat = new Professor("Joana", 1000, 10, 20);
      System.out.println("salari administratiu = " + empleat.calculaSalari());

  }
}
```



→ 1.700.0

Una referència a un objecte de la subclasse (Professor) sempre cridarà a la versió més específica del mètode sobreescrit.

Test.java

```
public class Test {
   public static void main(String[] args) {


      Empleat empleat;
      // Preguntar a l'usuari que vol introduir a l'aplicació:
       Scanner sc=new Scanner(System.in);
       System.out.println("Indica 1 per introduir un professor i 2 per introduir un
      administratiu: ");
       int resposta = sc.nextInt();
      if(resposta==1){
          empleat = new Professor("Joana",1000, 10, 20);
       }else{
       empleat = new Administratiu("Joana",1000, "dep");
      }
       System.out.println("salari professor = " + empleat.calculaSalari());
   }
}
```

Depenent de la resposta de l'usuari, sortirà: 1.500 o 2.000

RESUM

Polimorfisme vs. Sobrecàrrega

- És important diferenciar entre sobrecàrrega i el polimorfisme (sobreescriptura).

La sobrecàrrega consisteix a definir un mètode nou amb una signatura diferent (nombre i/o tipus de paràmetres).

La sobrecàrrega es pot detectar en temps de compilació.

El polimorfisme és la substitució d'un mètode per un altre en una subclasse mantenint la signatura original.

El polimorfisme es resol en temps d'execució.

Exemples

```
public class ExempleSobrecarrega{
   public void metodeExemple(){
     System.out.println("mètode sense
     parametres");
    }
 public void metodeExemple(int x){
     System.out.println("mètode amb
     els parametres" + x);
    }
}
```

Exemples

```
public class ExPolimorfismeMare{
        public void metode(){
                System.out.println("mètode original");
        }
}
```

```
public class ExPolimorfisme extends ExPolimorfismeMare{
        public void metode(){
                System.out.println("mètode sobreescrit");
        }
}
```

```
public class Test {
        public static void main(String[] args) {
                ExPolimorfismeMare ex = new ExPolimorfisme();
                ex.metode();
        }
}
```

Què surt per pantalla?

→ mètode sobreescrit

Com has d'implementar el mètode per que aparegui per pantalla el missatge?:

mètode original

mètode sobreescrit

Exemples

- Com has d'implementar el mètode per que aparegui per pantalla el missatge?:
  mètode original
  mètode sobreescrit

Solució:

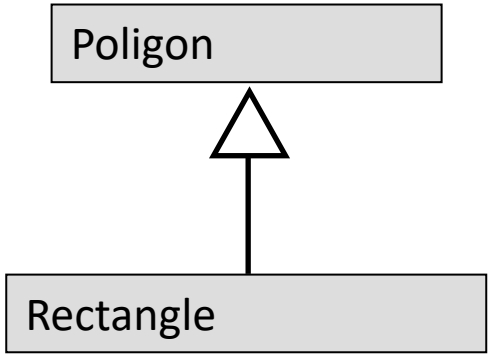
```
public class ExPolimorfismeMare{
	public void metode(){
		System.out.println("mètode original");
	}
}
```

```
public class ExemplePolimorfisme extends ExPolimorfismeMare{
	public void metode(){
		super.metode();
		System.out.println("mètode sobreescrit");
	}
}
```

Exemple de sobreescriptura

```
public class Poligon {
    public void imprimirIdentitat(){
        System.out.println("Sóc Poligon");
    }
}
```

```
public class Rectangle extends Poligon{
    @Override
    public void imprimirIdentitat(){
        System.out.println("Sóc Rectangle");
    }
}
```




```
public class Test {
    public static void main(String[] args){
        Poligon[] pol = new Poligon[2];

        Poligon elemA = new Poligon();
        Rectangle elemB = new Rectangle();

        pol[0] = elemA;
        pol[1]= elemB;

        pol[0].imprimirIdentitat();
        pol[1].imprimirIdentitat();
    }
}
```

Sortida per pantalla

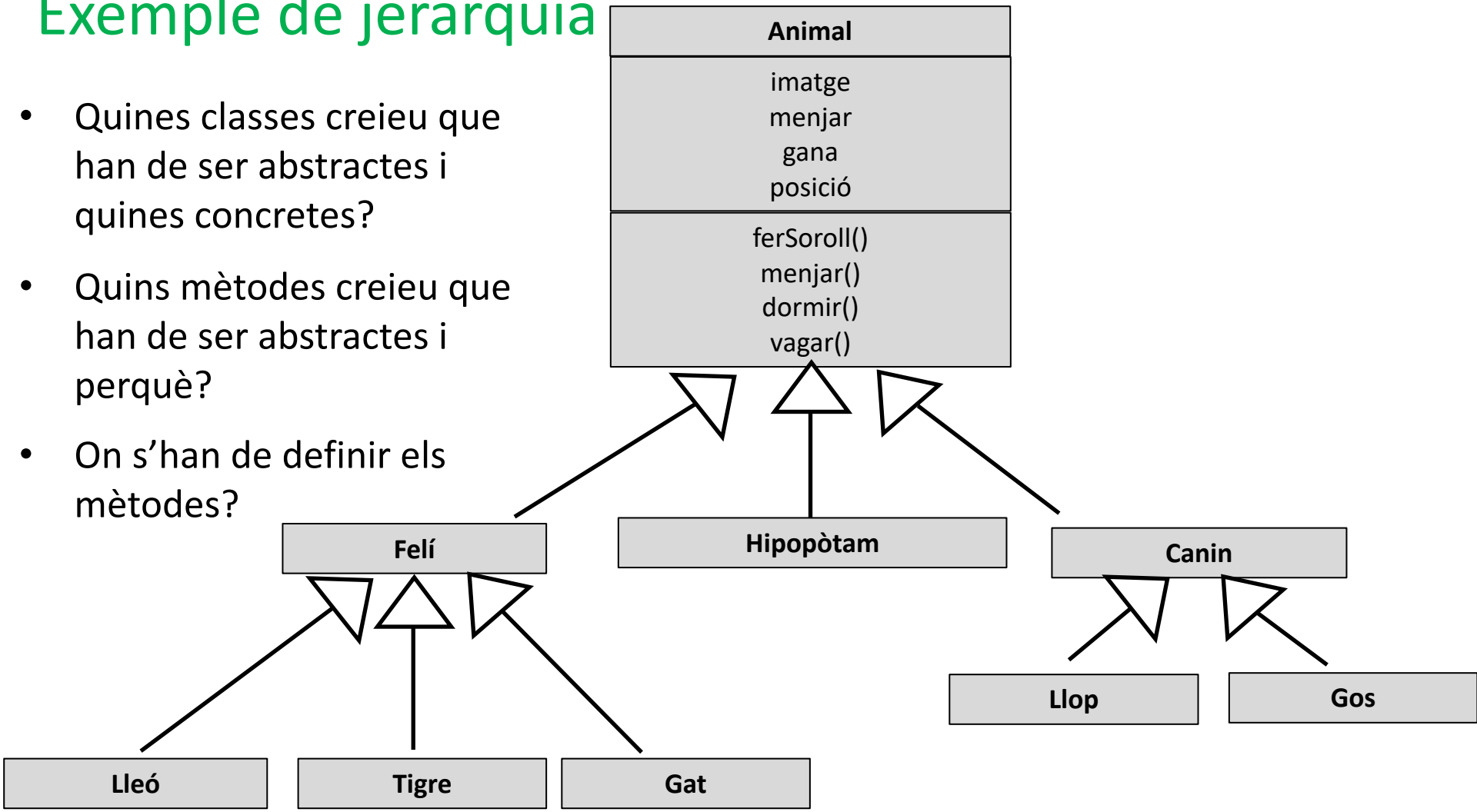
```
Sóc Poligon
Sóc Rectangle
```

Exercici de la jerarquia animal

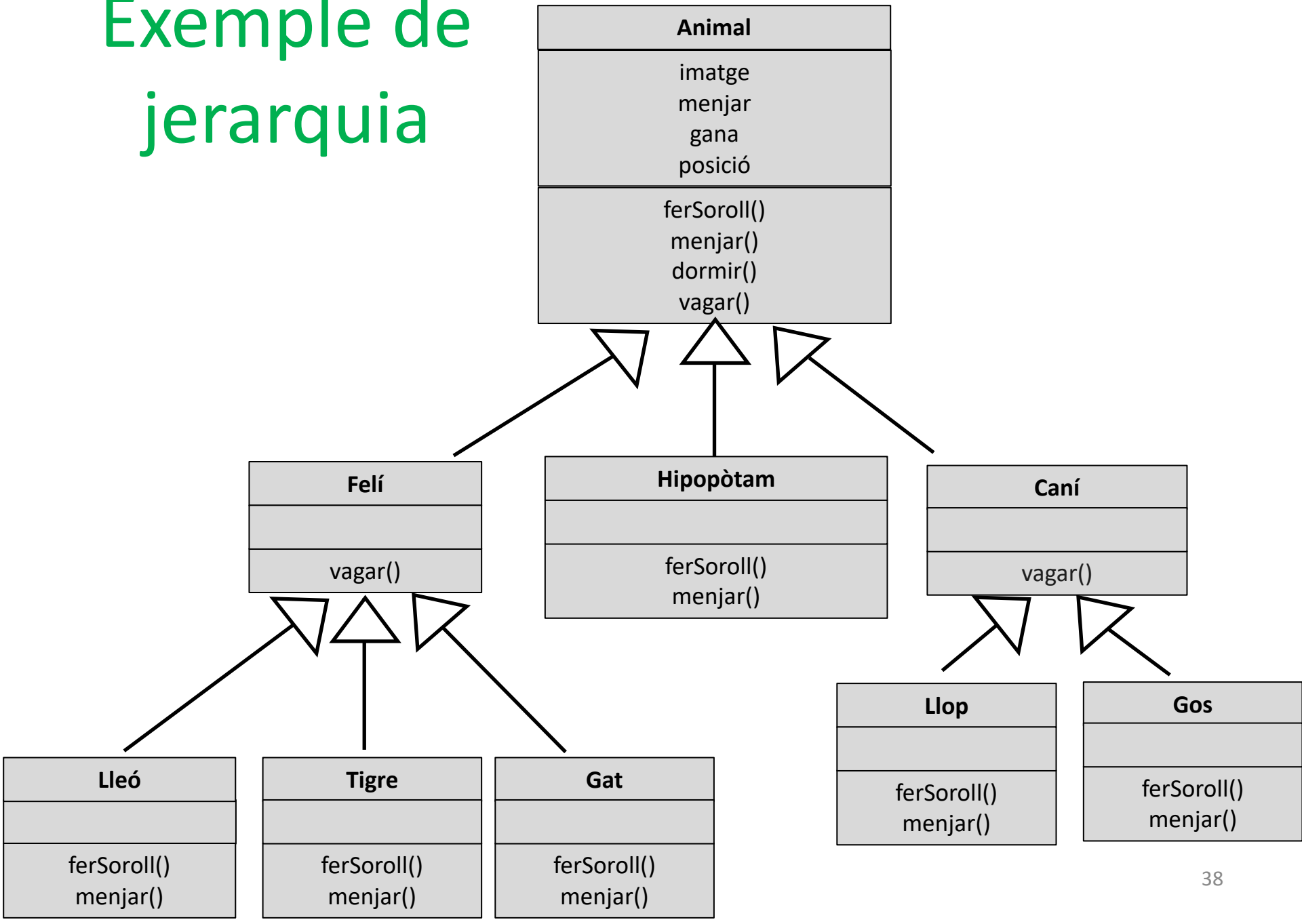
- A continuació es presenta la jerarquia d'animal.
- La superclasse Animal defineix el protocol comú per a tots els animals.
- Definim algunes de les superclasses com **abstractes** de forma que no es puguin instanciar.
- La resta de las classes seran **concretes**.

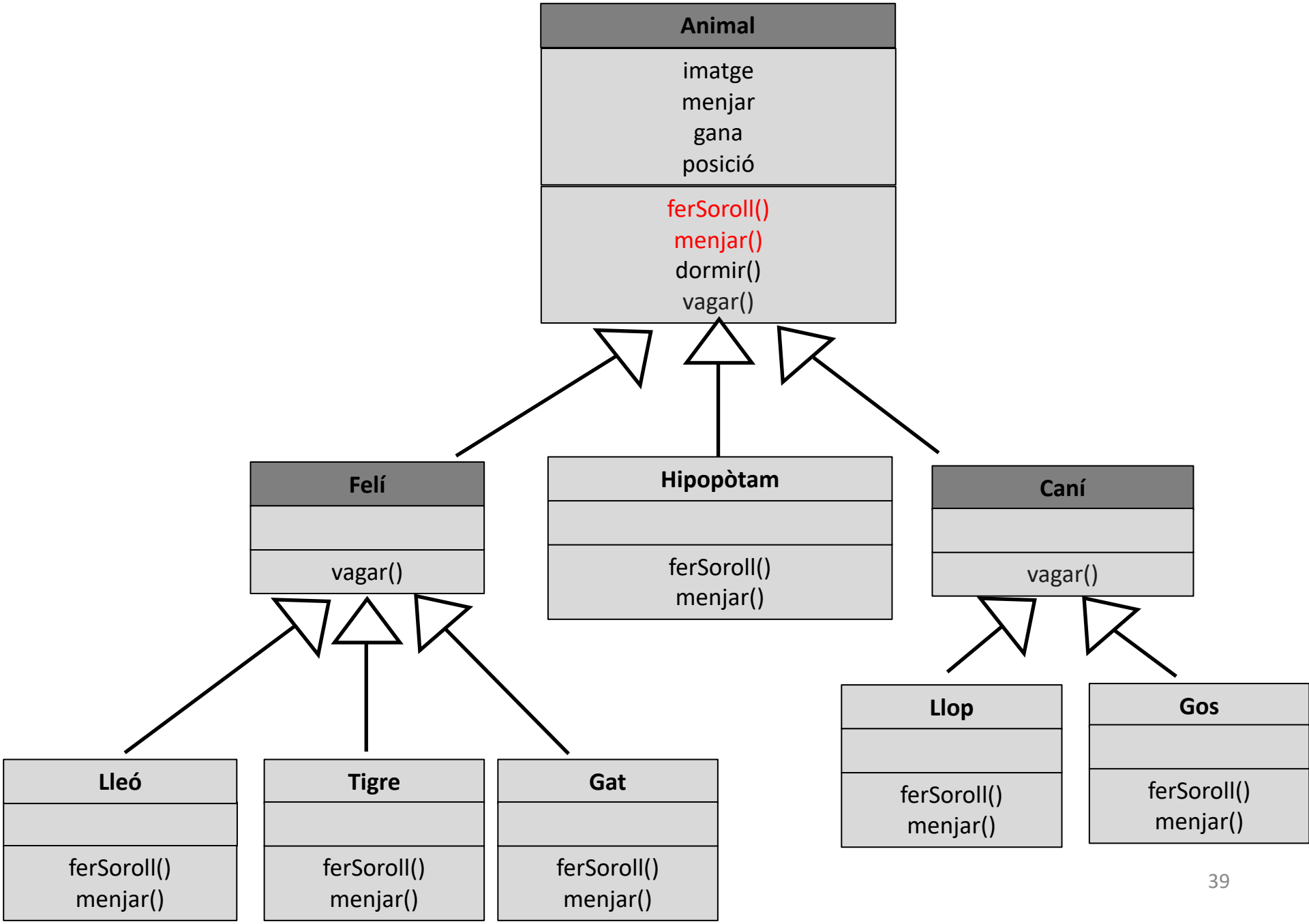
Exemple de jerarquia

- Quines classes creieu que han de ser abstractes i quines concretes?
- Quins mètodes creieu que han de ser abstractes i perquè?
- On s'han de definir els mètodes?



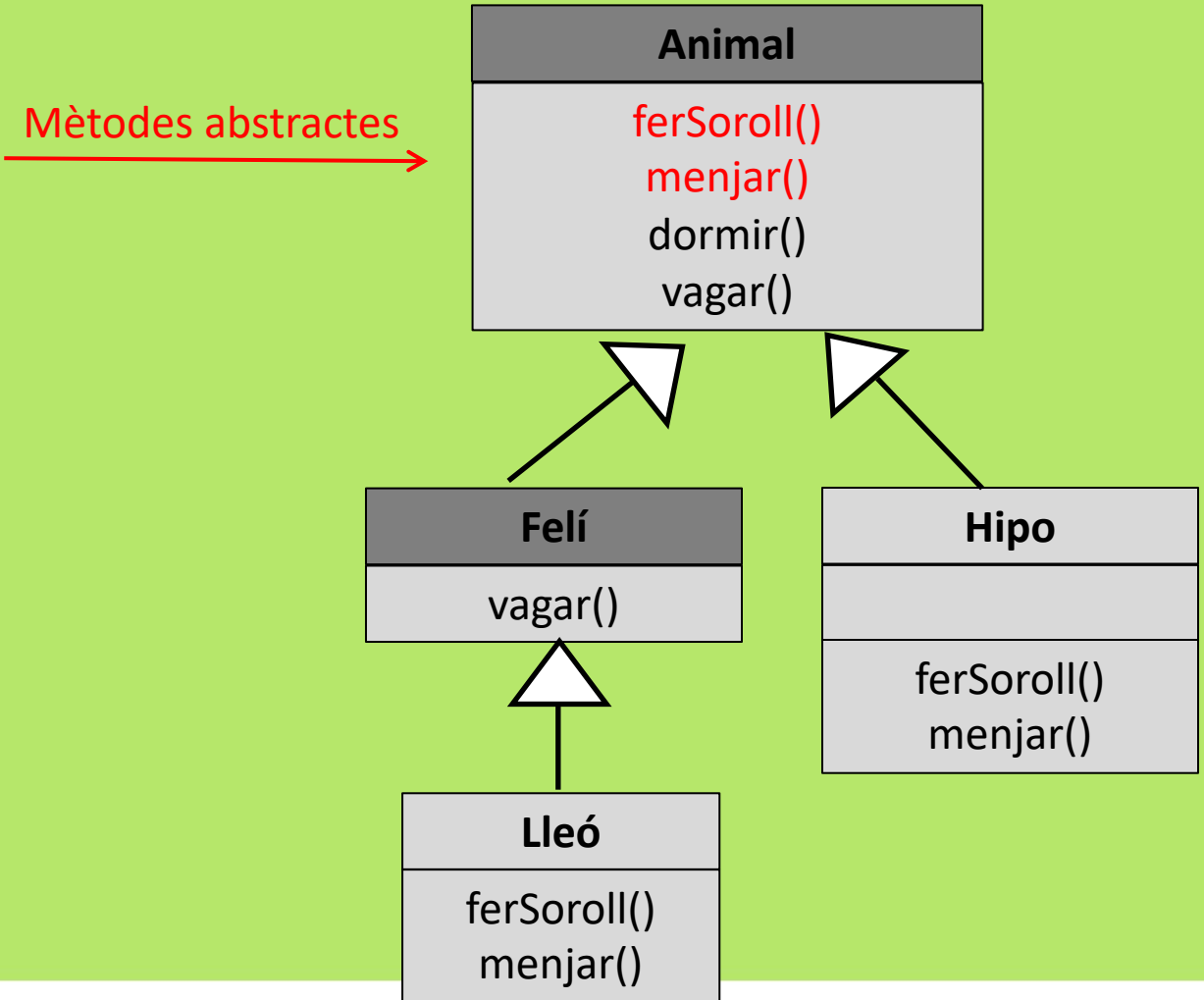
Exemple de jerarquia




Animal
imatge
menjar
gana
posició
ferSoroll()
menjar()
dormir()
vagar()
Felí
vagar()
Hipopòtam
ferSoroll()
menjar()
Caní
vagar()
Lleó
ferSoroll()
menjar()
Tigre
ferSoroll()
menjar()
Gat
ferSoroll()
menjar()
Llop
ferSoroll()
menjar()
Gos
ferSoroll()
menjar()

Exercici per fer

- Implementa les següents classes:



Referències

- Bertrand Meyer, “**Construcción de software orientado a objetos”,** Prentice Hall, 1998. Capítol 14.

- Bert Bates, Kathy Sierra. **Head First Java**. O’Reilly Media, 2005. Capítol 7.