GRAU D'ENGINYERIA INFORMÀTICA

PROGRAMACIÓ II

Bloc 2:

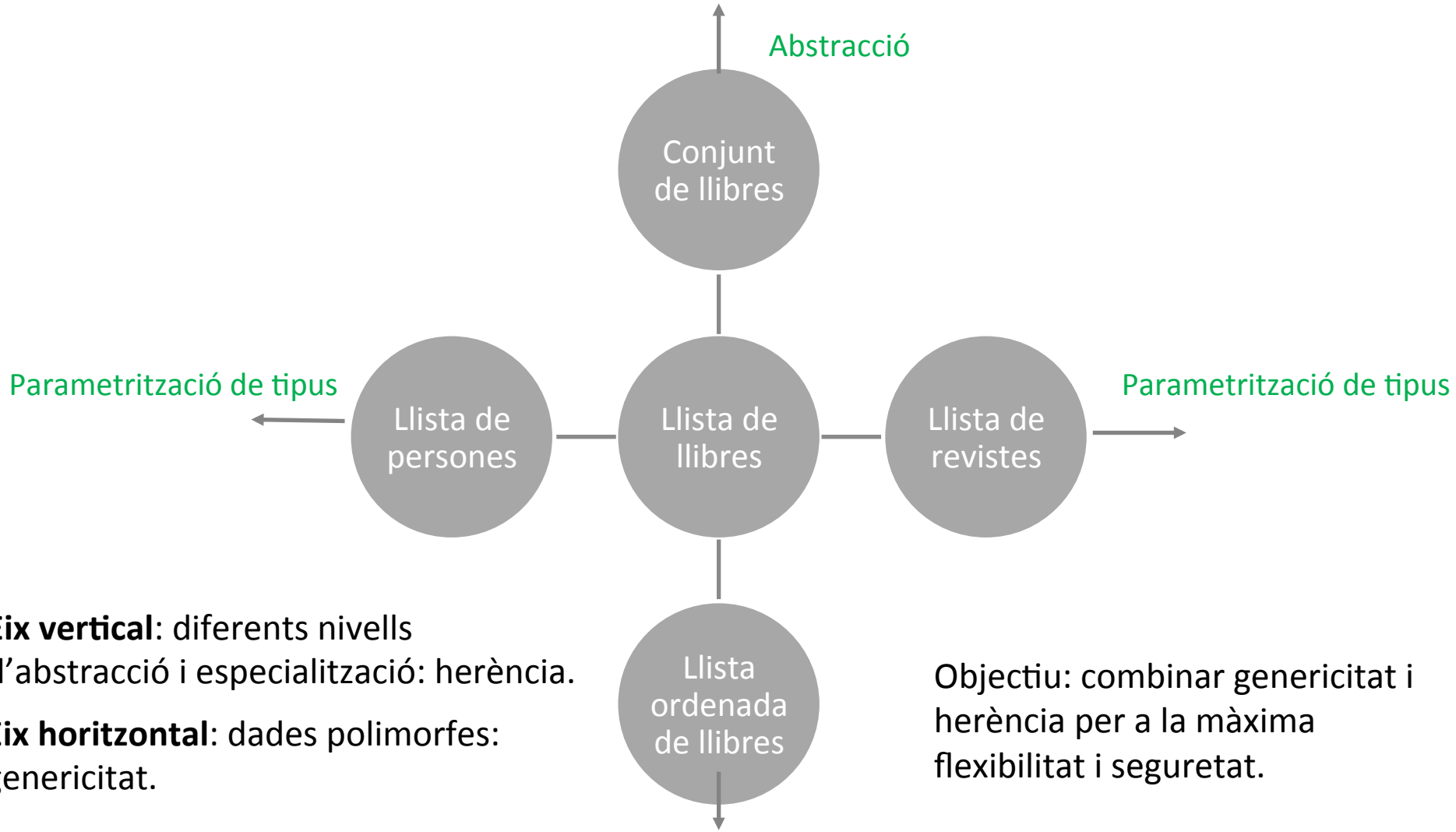
Programació Orientada a Objectes (2)

Sergio Sayago (basat en material de Laura Igual)
Departament de Matemàtiques i Informàtica
Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

Bloc 2:
Programació orientada a objectes

- Abstracció en el desenvolupament del *software*
- Conceptes fonamentals: classes i objectes
- Característiques de l'orientació a objectes
- Ús de classes i objectes
- Constructors i destructors
- Encapsulació
- **Herència i jerarquia de classes**
- Polimorfisme
- Lligadures
- Interfícies
- col·leccions

Generalització



Eix vertical: diferents nivells d'abstracció i especialització: herència.

Eix horitzontal: dades polimorfes: genericitat.

Objectiu: combinar genericitat i herència per a la màxima flexibilitat i seguretat.

Herència

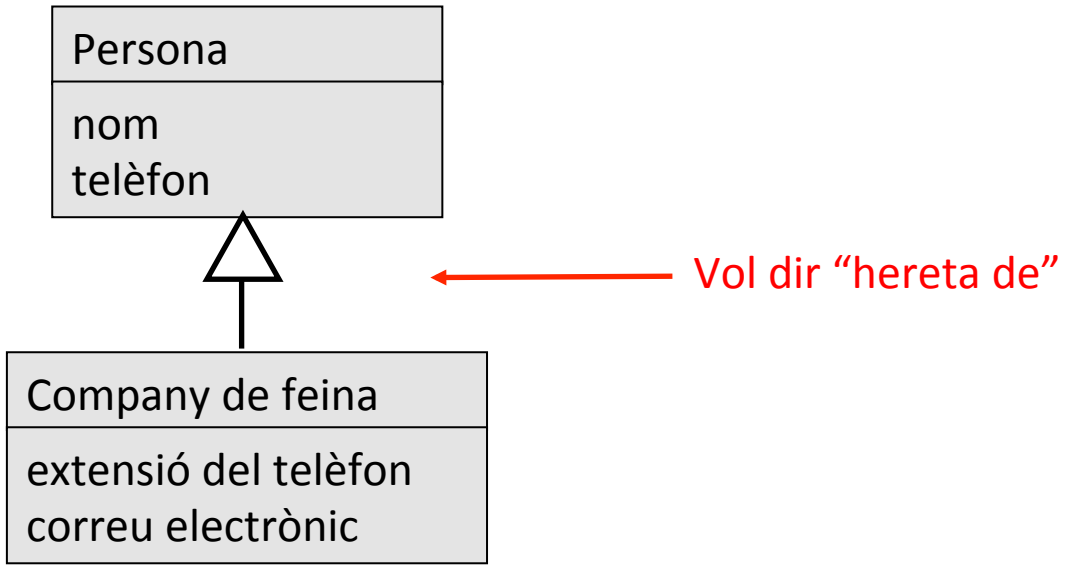
- L'herència és un mecanisme que permet definir una classe nova a partir d'una d'anterior descrivint les diferències entre elles.
- **Característica pròpia de la programació orientada a objectes**
- Facilita la reutilització
- Concepte de relacions de generalització i especialització

Tipologies d'herència

- Depenent de la manera d'arribar-hi a l'herència, s'anomenen:
  - Herència per especialització
  - Herència per generalització

Herència per especialització

- Sorgeix de la necessitat de crear una classe nova que afegeixi unes propietats i un comportament a una altra classe del domini ja existent.
- Quan afegim funcionalitat a un disseny ja donat.
- Exemple:



Herència per generalització

- Apareix amb la finalitat d'homogeneïtzar el comportament de les parts comunes a certes classes.
- Quan es crea el disseny de classes.

Cotxe
rodes potencia seients

Moto
rodes potencia requereix casco

Camió
rodes potencia carrega

Millor utilitzar herència

Vehicle
rodes potencia

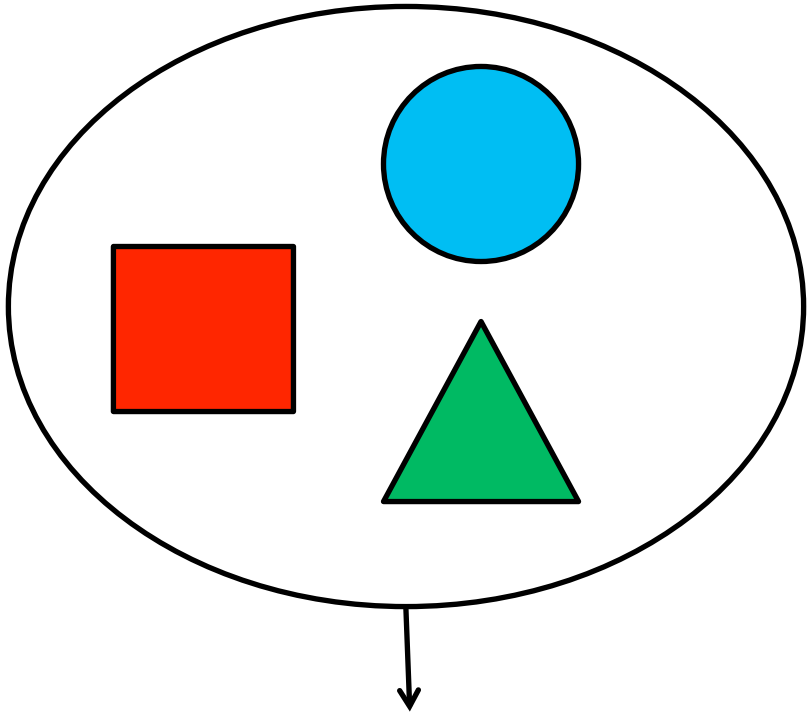
Cotxe
seients

Moto
requereix casco

Camió
carrega

Exemple

- Herència → Jerarquia de classes
- Classe : Figura geomètrica
- Classe: Triangle, quadrat, cercle, ...



Figures geomètriques → Dibuixar

Figura geomètrica
color posició a pantalla àrea perímetre
calcula àrea calcula perímetre retorna color assigna color

Quadrat
color posició a pantalla àrea perímetre dimensió costats

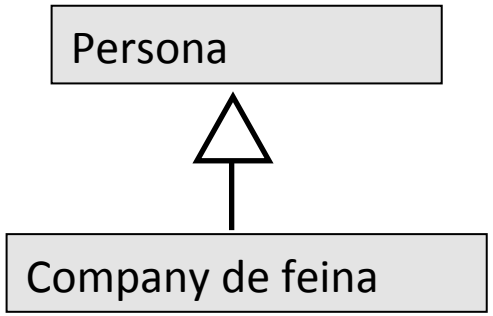
Circumferència
color posició a pantalla àrea perímetre Radi

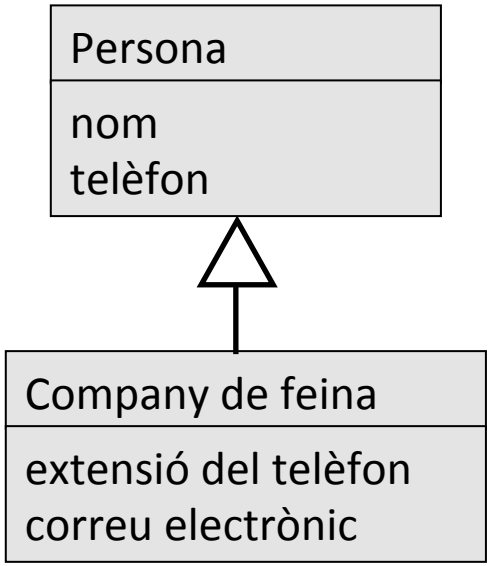
Triangle
color posició a pantalla àrea perímetre dimensió 3costats

Herència

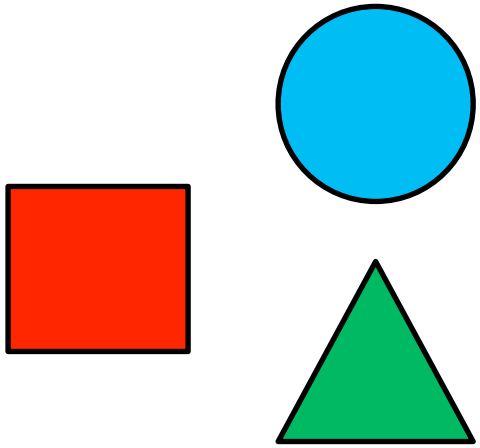
- Terminologia:

Hereta de





Exemple



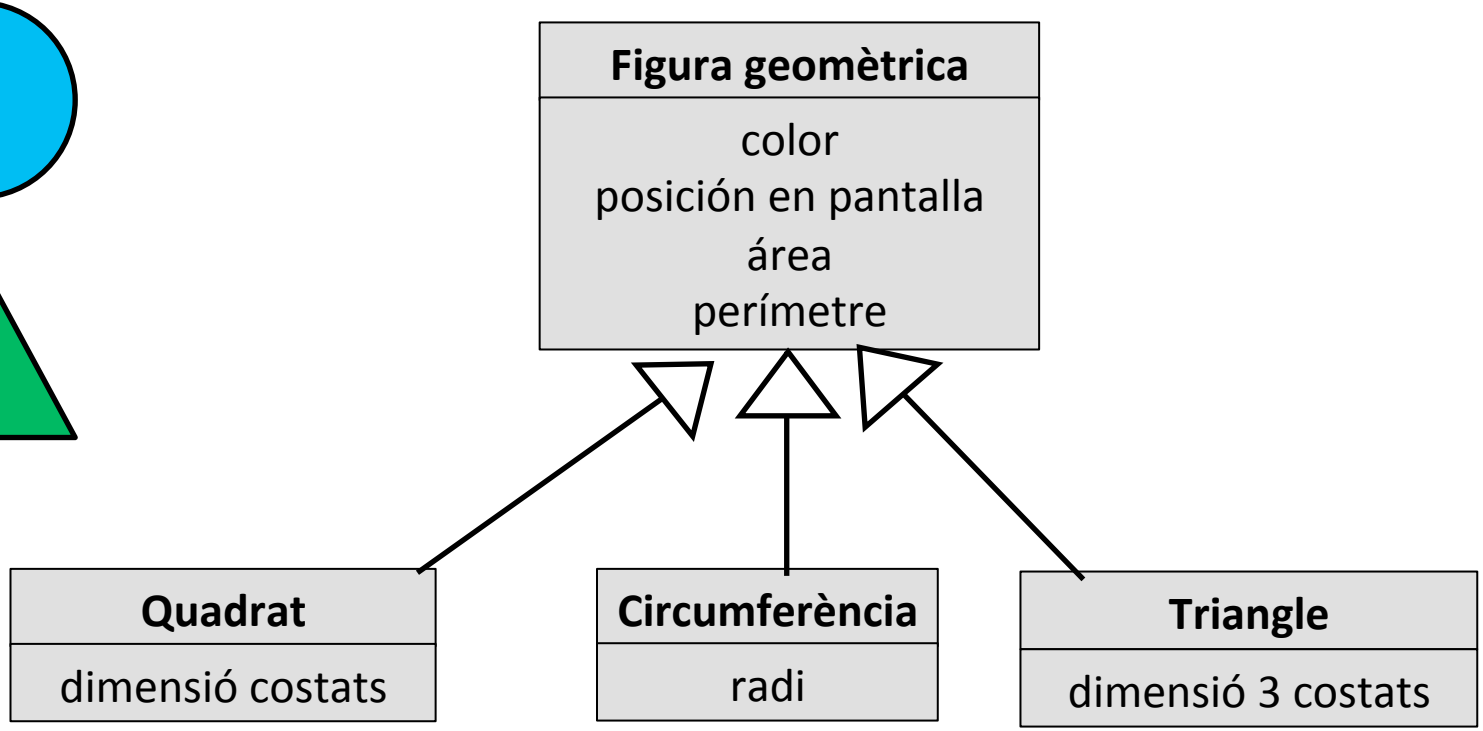
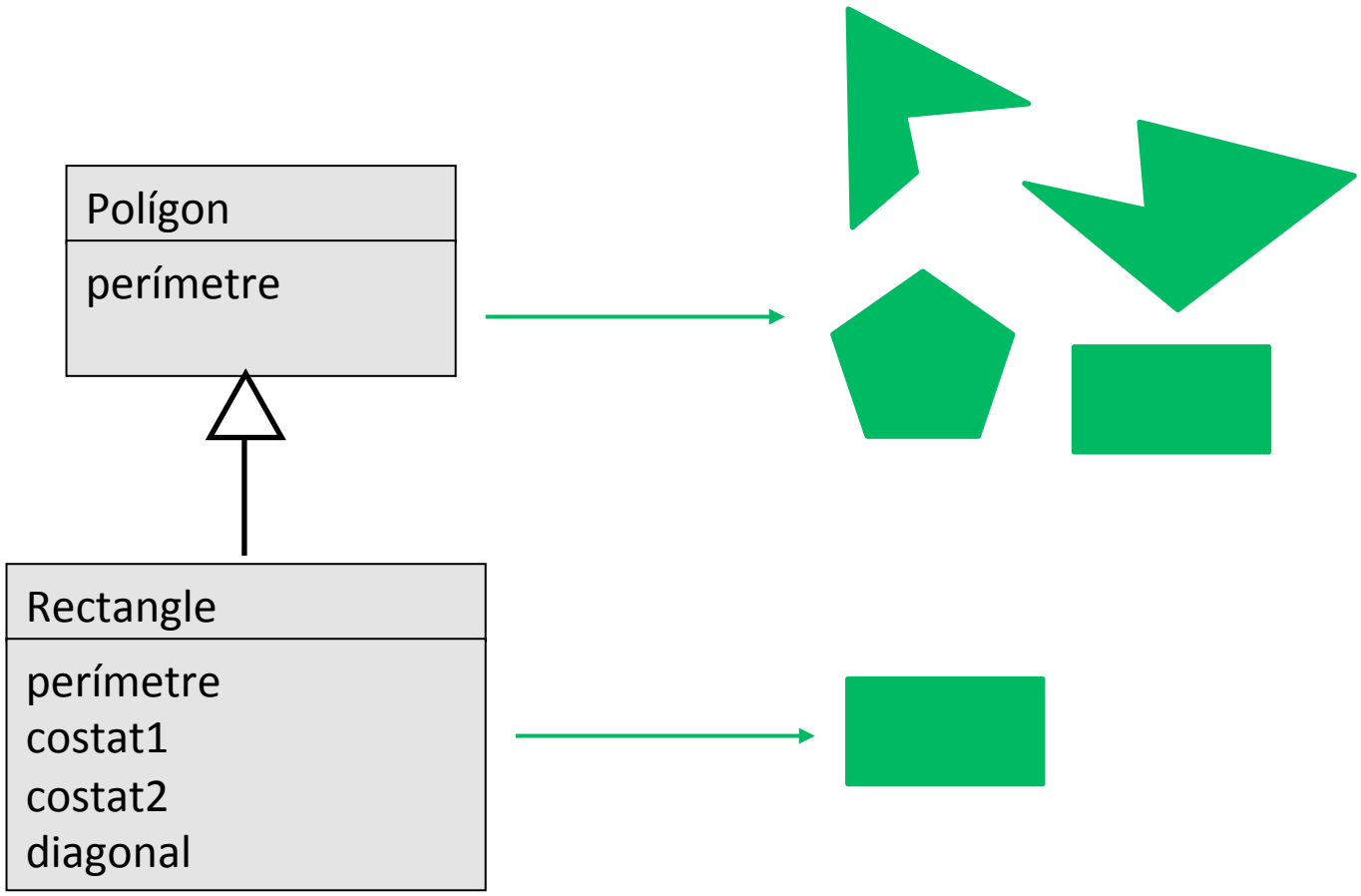


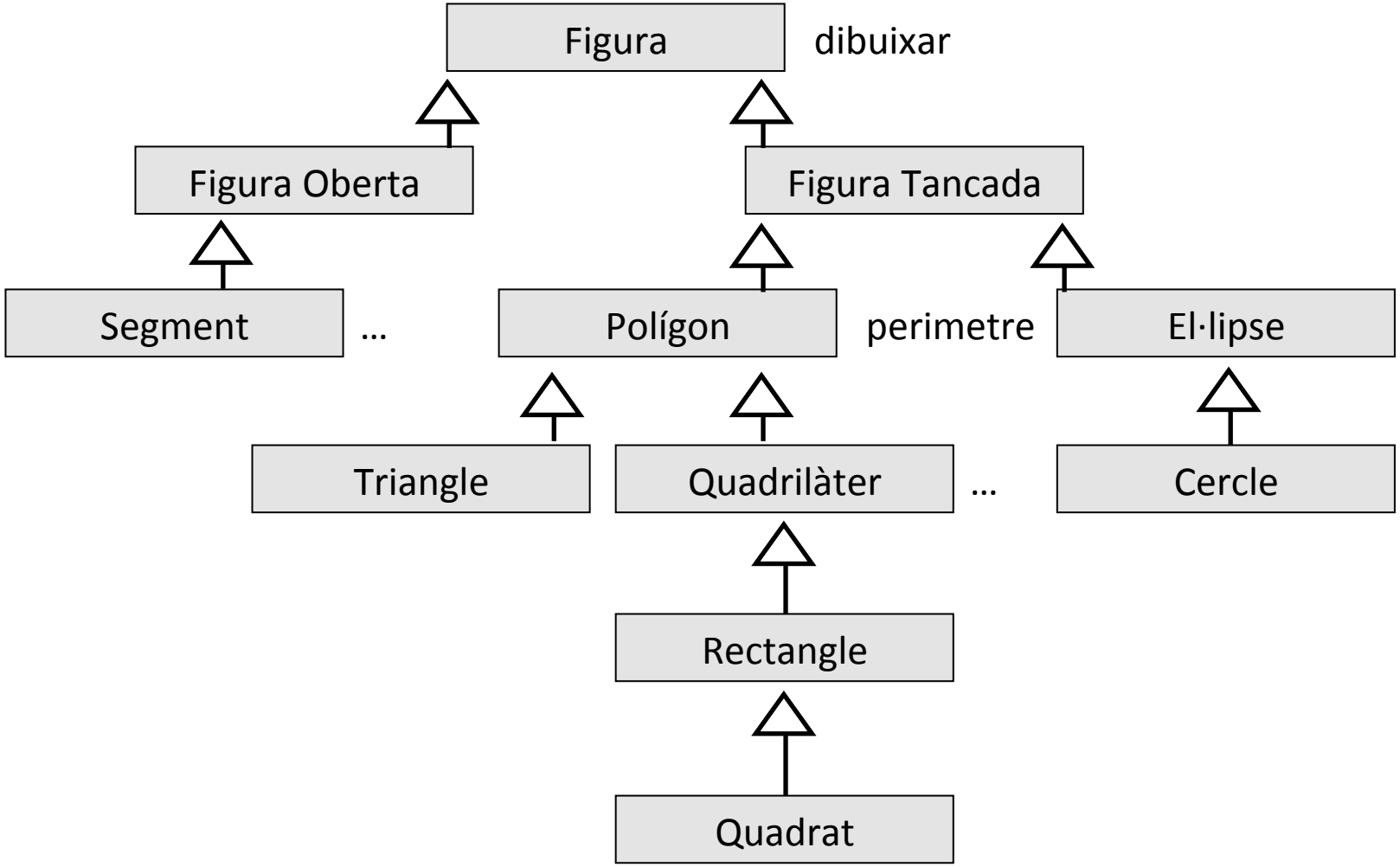
Figura geomètrica
color
posición en pantalla
área
perímetre
Quadrat
dimensió costats
Circumferència
radi
Triangle
dimensió 3 costats

Herència

- Exemple



Jerarquia de classes



Herència amb Java

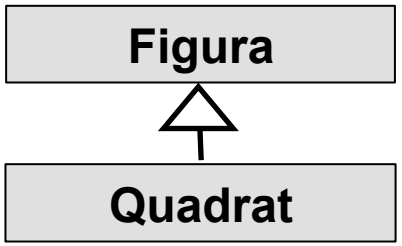
- Per definir una herència:

  paraula reservada ***extends***

  \+ nom de la classe de la qual s'hereta

- Per accedir als mètodes definits a la classe mare:

  paraula reservada ***super***

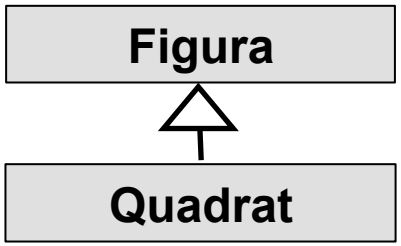



```
public class Figura{
        …
}
```

```
public class Quadrat extends Figura{
        …
}
```

Herència amb C++

- class <Nom_Classe_Derivada>: {private, protected, public} <Nom_Classe_Base>



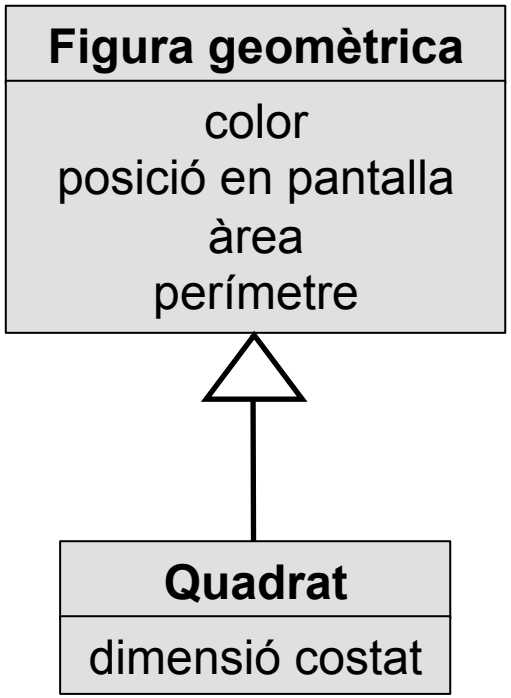
CONSIDERACIONS SOBRE L'HERÈNCIA

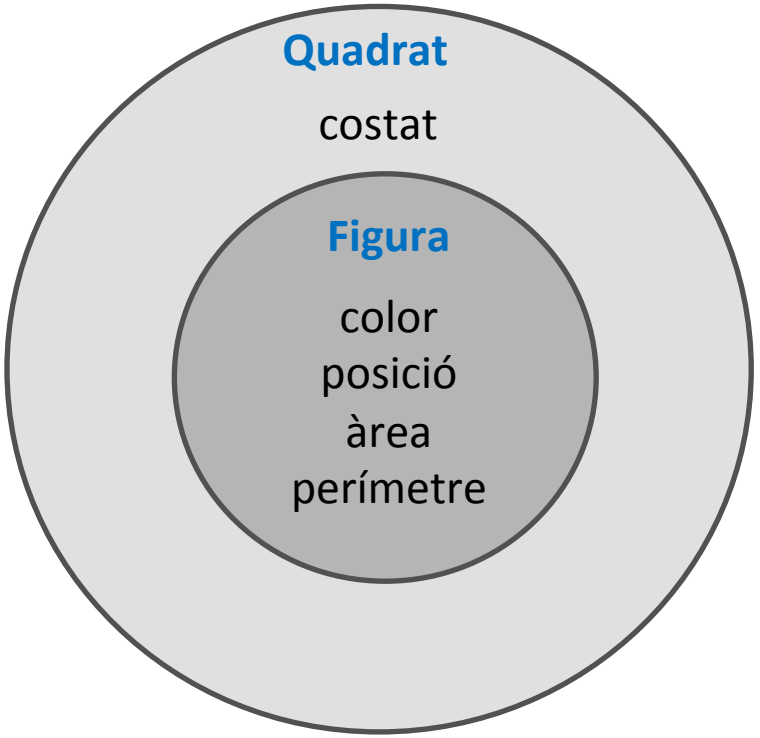
Consideracions sobre l'herència

- Els atributs i mètodes de la superclasse estaran **sempre definits en la subclasse.**
  - Aquesta restricció només s'aplica als atributs i mètodes definits amb la **visibilitat public o protected.**
  - Les classes filles no tenen accés als atributs i mètodes definits com a **private**.

Herència

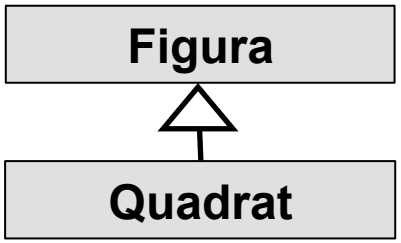
- Representació:





Consideracions sobre l'herència

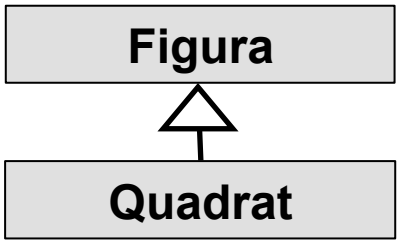
```
public class Figura{
 private String color;
 public Figura() {
    this.color = "Vermell";
 }
 public String getColor(){
    return color;
  }
}
```




```
public class Quadrat extends Figura{
 public void mostrarInfo(){
    System.out.println("Color:" + getColor() + "\n");
 }
}
```

Consideracions sobre l'herència

```
public class Figura{
 protected String color;
 public Figura() {
    this.color = "Vermell";
 }
}
```




```
public class Quadrat extends Figura{
 public void mostrarInfo(){
    System.out.println("Color:" + color + "\n");
 }
}
```

Consideracions sobre l'herència

Utilització del **constructor** de la superclasse:

```
public class LaMevaClasse {
 int i;
 public LaMevaClasse () {
        this.i = 10;
   }
}
```

```
import LaMevaClasse;
Int cont2;
public class NovaClasse extends LaMevaClasse {
  public NovaClasse() {
      cont2 = 20;
  }
}
}
```

Consideracions sobre l'herència

Utilització del **constructor** de la superclasse:

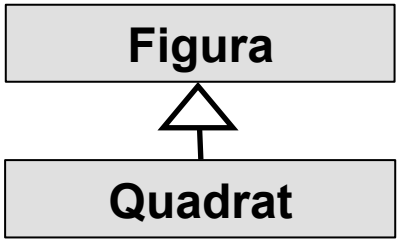
```
public class LaMevaClasse {
 int i;
 public LaMevaClasse (int i) {
        this.i = i;
   }
}
```

```
import LaMevaClasse;
public class NovaClasse extends LaMevaClasse {
  public NovaClasse(int i) {
      super(i);
  }


}
```

Consideracions sobre l'herència

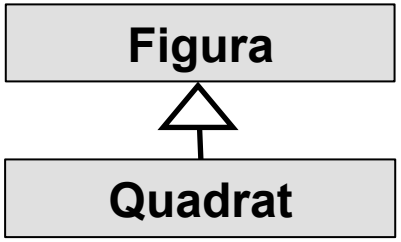
```
public class Figura{
 private String color;
 public Figura() {
    this.color = "Vermell";
 }
}
```




```
public class Quadrat extends Figura{
 private double costat;
 public Quadrat(){
        costat = 1.0;
 }
}
```

Consideracions sobre l'herència

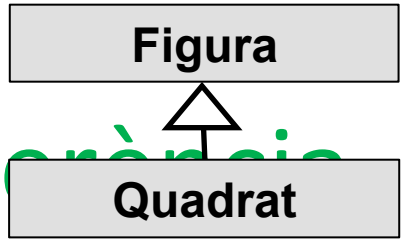
```
public class Figura{
 private String color;
 public Figura(String color)
  {
    this.color = color;
  }
}
```




```
public class Quadrat extends Figura{
 private double costat;
public Quadrat(String color){
        super(color);
        costat = 1.0;
  }
}
```

```
public class Figura{
 private String color;
 public Figura(String color)
  {
    this.color = color;
 }
}
```

bre l'h




```
public class Quadrat extends Figura{
 private double costat;
 public Quadrat(){
        String color = "Verd";
        super(color);
        costat = 1.0;
 }
 public Quadrat(String color){
        super(color);
        costat = 1.0;
 }
}
```

Consideracions sobre l'herència

Creació d'objectes en cadena o cascada:

- Quadrat elMeuQuadrat = new Quadrat();

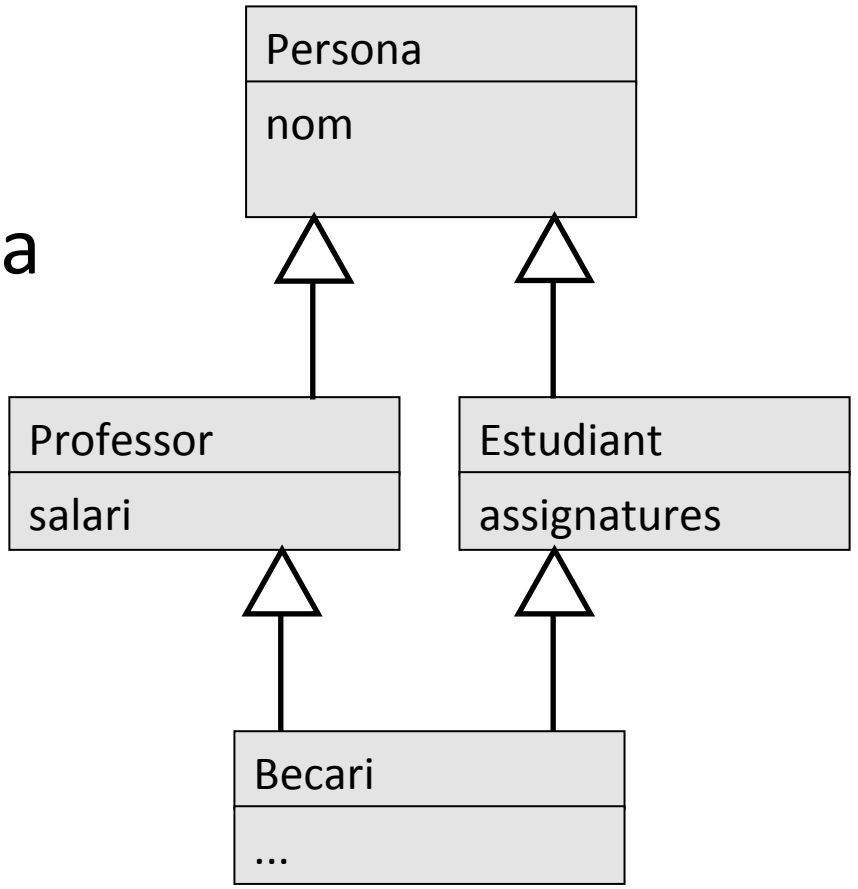
Crea dos objectes:

1) Primer, objecte de tipus Figura
2) Després, objecte de tipus Quadrat

Un Quadrat és una Figura. No podem tenir Quadrat sense Figura (pero sí Figura sense Quadrat)

Consideracions sobre l'herència

- **Herència múltiple**
- L'herència en la que la nova classe és generada a partir de dues o més classes alhora.



Quin pot ser el problema?

Problema: el becari té dos atributs nom

Consideracions sobre l'herència

... Herència múltiple

- No està soportada a Java (però si a C++)
- A Java, l'herència múltiple és soluciona amb interfaces – que veurem més endavant

Consideracions sobre l'herència

Atributs:

- En una classe filla, podem afegir **nous atributs.**
- S'ha de vigilar a l'hora de **triar els noms** dels nous atributs.

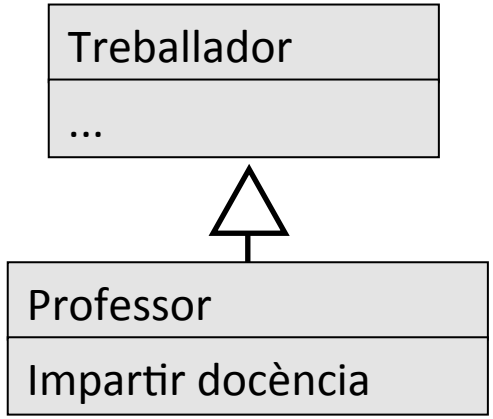
Consideracions sobre l'herència

Mètodes:

- Podem realitzar 3 tasques diferents:
  - Afegir mètodes nous
  - Implementar mètodes declarats prèviament com abstractes
  - Tornar a implementar mètodes (sobrescriptura).

Afegir mètodes nous

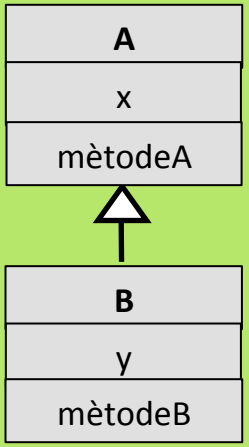
- Atributs nous → necessitem mètodes nous
  - Mètodes que realitzin tasques específiques de la classe filla.
- Els mètodes definits en la subclasse es consideraran mètodes d'aquesta i només s'hi podrà accedir des d'instàncies d'aquesta o de les seves classes filla.



Exercici: Fes de compilador!

- Donat el següent codi de la classe A i la classe B (que hereta de la classe A) i el diagrama il·lustrant la relació entre les classes:

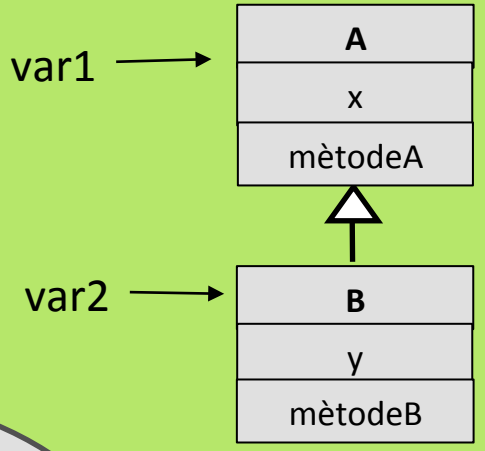
```
public class A{
	public int x;
	public void metodeA(){
		....
	}
}
public class B extends A{
	public int y;
	public void metodeB(){
		....
	}
}
```

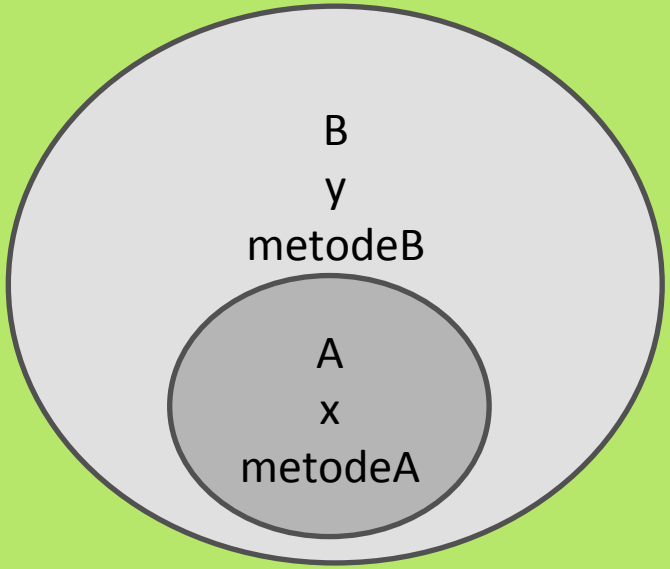


Exercici: Fes de compilador!

Indica a cada una de las línies del següent codi si haurà errors de compilació o no i explica breument perquè:

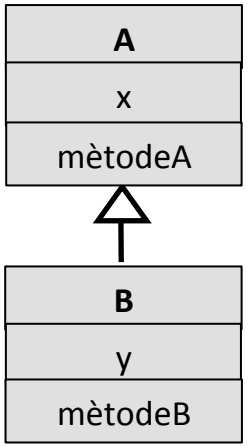
```
0 public static void main(String[] args) {
1   A var1 = new A();
2   B var2 = new B();
3   int i = var1.x;
4   int j = var2.x;
5   int k = var1.y;
6   int l = var2.y;
7var1.metodeA();
10var1.metodeB();
11var2.metodeA();
11   var2.metodeB();
12 }
```





Exercici: Fes de compilador!

Indica a cada una de las línies del següent codi si haurà errors de compilació o no i explica breument perquè:




```
0 public static void main(String[] args) {
1   A var1 = new A(); OK
2  B var2 = new B(); OK
3   int i = var1.x;    OK
4   int j = var2.x;    OK
5   int k = var1.y; Error, l'atribut y no està definit per a
A.
6   int l = var2.y;    OK
7  var1.metodeA();    OK
10 var1.metodeB(); Error, el metodeB no està definit per a A
11 var2.metodeA();  OK
11   var2.metodeB();  OK
12 }
```

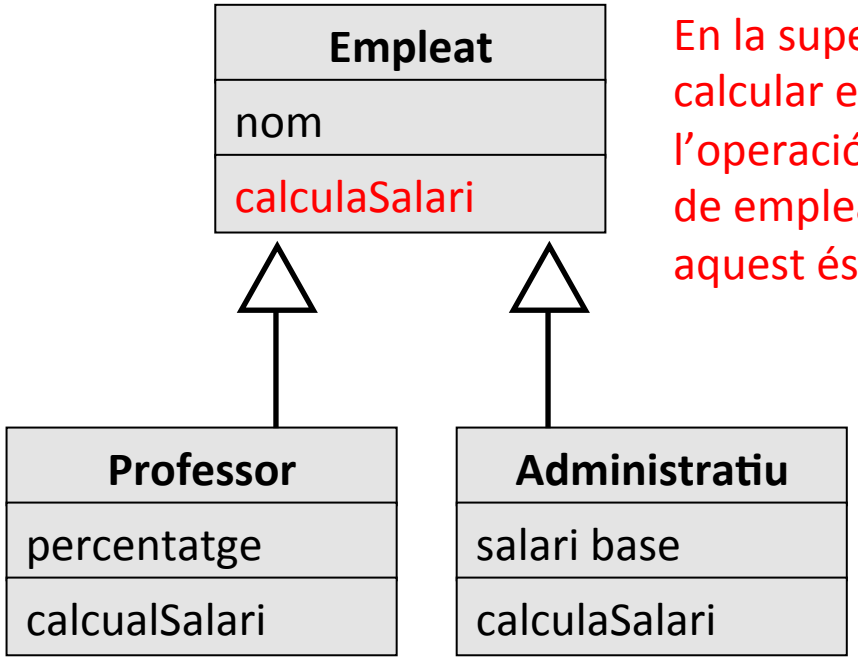
Consideracions sobre l'herència

Mètodes:

- Podem realitzar 3 tasques diferents:
  - Afegir mètodes nous
  - Implementar mètodes declarats prèviament com abstractes
  - Tornar a implementar mètodes (sobrescriptura).

Implementar mètodes abstractes

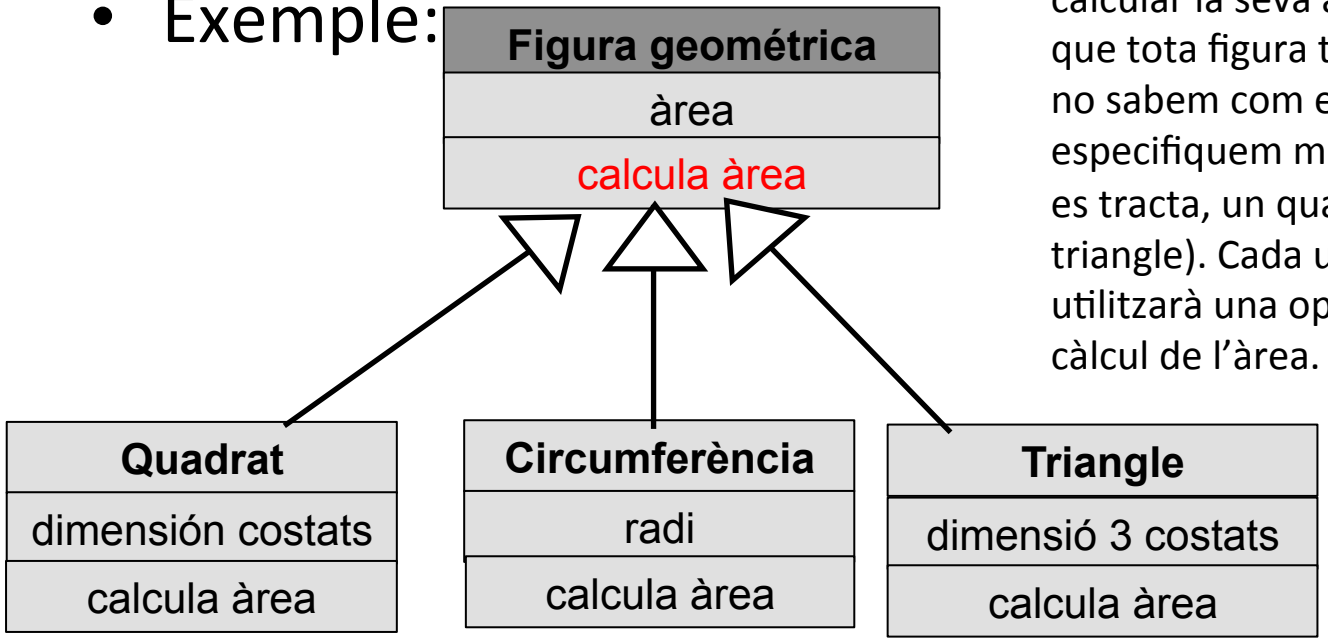
- **Mètode abstracte** és aquell que té definida la seva interfície (nom, tipus, nombre de paràmetres i valor de retorn), però no té implementat el codi que atendrà les peticions.
- Exemple:



En la superclasse no es pot calcular el salari perquè l'operació depèn de quin tipus de empleat sigui. Per tant aquest és un métode abstracte.

Implementar mètodes abstractes

- **Mètode abstracte** és aquell que té definida la seva interfície (nom, tipus, nombre de paràmetres i valor de retorn), però no té implementat el codi que atendrà les peticions.
- Exemple:



Tota figura geomètrica te que poder calcular la seva àrea, per tant obliguem a que tota figura tingui aquest mètode, però no sabem com es calcula l'àrea fins que no especifiquem més (saber de quina figura es tracta, un quadrat, circumferència o triangle). Cada una d'aquestes figures utilitzarà una operació diferent per al càlcul de l'àrea.

Implementar mètodes abstractes

- Si una classe té declarat com a mínim un mètode abstracte, es diu que la **classe** és **abstracta**.
- Les classes abstractes obliguen les classes que hereten d'aquesta a implementar els mètodes no implementats.

En cas que una classe filla continuï sense implementar un mètode abstracte, aquesta ha de ser també abstracta.

Tipus de classes

- **Abstract**
- **Final**
- **Public**
- **Synchronizable**

Tipus de classes

- **Classe abstracta:**

  No s'instancia, sinó que s'utilitza com classe base per a l'herència.

- Exemple:

Classificació animal:

  - Mamífer,
  - Bípede,
  - Quadrúpede,

  → D'aquests conjunts no hi ha instàncies concretes

  La balena és un mamífer, però de la subespècie dels cetacis

  El cavall és un quadrúpede, de la subespècie dels equins

Tipus de classes

- **Classe final**

  Se declara com la classe que termina una cadena d'herència. No es pot heretar d'ella.

- Exemple:

La classe **Math** és una classe final.

Tipus de classes

- **Classes public**

  Són accessibles des d'altres classes, o directament o per herència.
- Són accessibles dins del mateix paquet en el que s'han declarat.
- Per a accedir des d'altres paquets, primer tenen que ser importades.

Tipus de classes

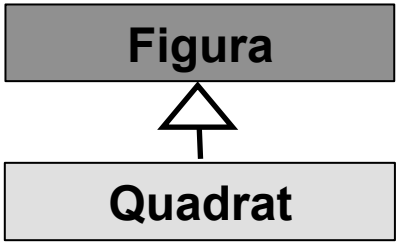
- **Classe synchronizable**
- Aquest modificador especifica que tots els mètodes definits en la classe són sincronitzats, es a dir, que no es poden accedir al mateix temps a ells des de diferents threads;
- El sistema s'encarrega de col·locar els flags necessaris per a evitar-ho.
- Aquest mecanisme fa que des de threads diferents es puguin modificar les mateixes variables sense que hagi problemes de sobreescriptura.

Classe abstracta en Java

- Per a definir una classe abstracta s'utilitza en la declaració la paraula:

abstract

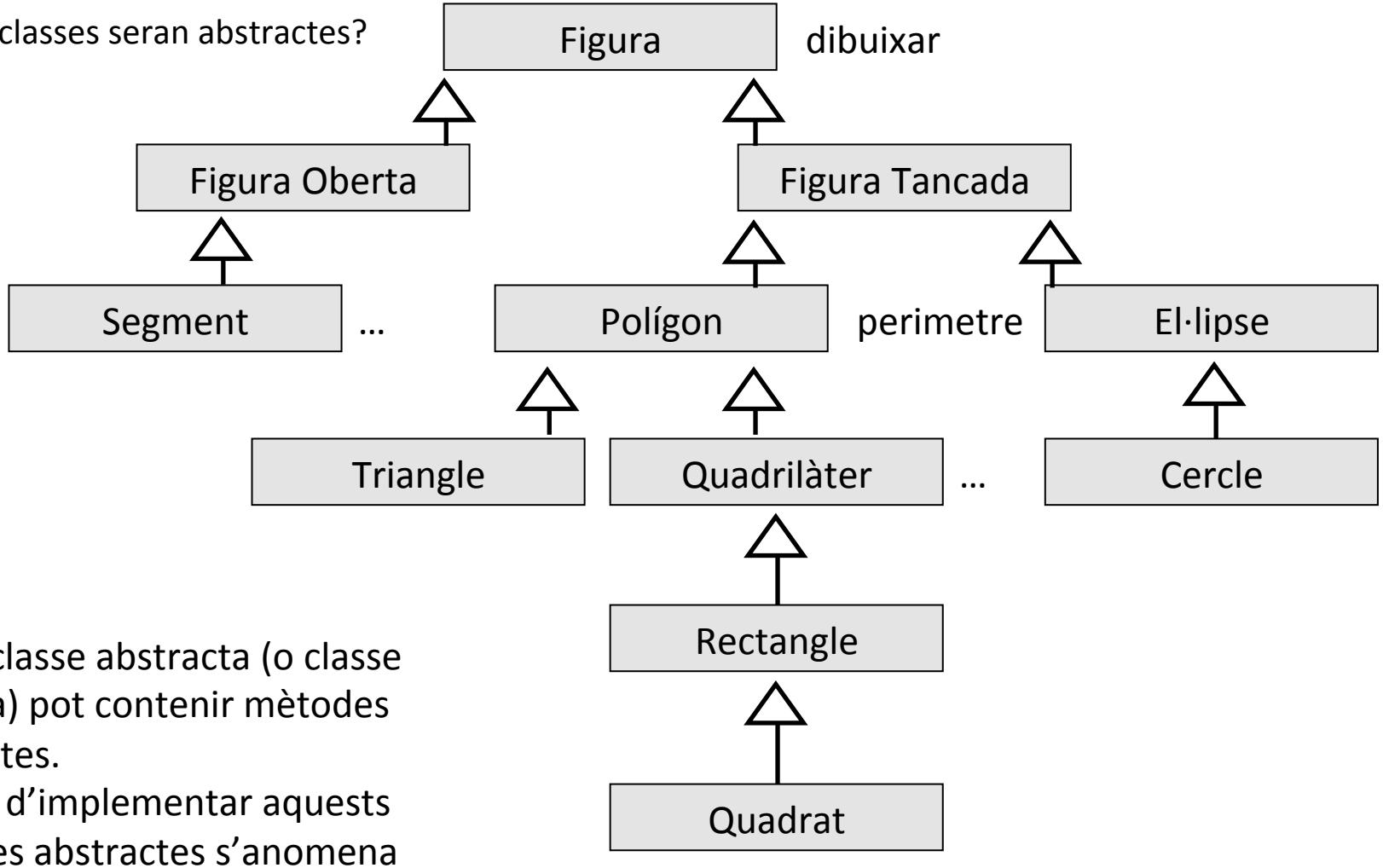
```
public abstract class Figura {
        …
}
```



En C++: classe amb funció virtual pura (per ex. virtual doble getVolume() = 0)

Exemple de jerarquia de classes

Quines classes seran abstractes?

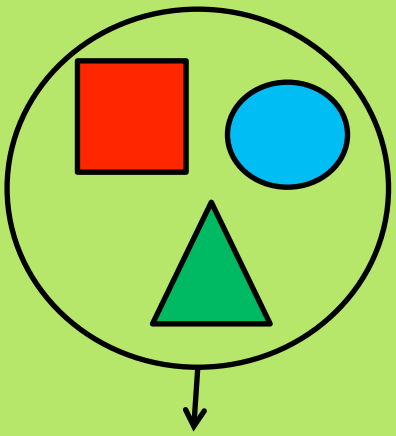


- Una classe abstracta (o classe diferida) pot contenir mètodes abstractes.
- El fet d'implementar aquests mètodes abstractes s'anomena ***fer efectiu*** un mètode.

Exercici

Implementeu el següent disseny, considerant:

- **Classe abstracta:** Figura geomètrica
- Classe: Triangle, quadrat, cercle, ...



Figures geomètriques

Solució

Figura.java

```
public abstract class Figura {
   private String color;
   protected double x, y;
   protected double area;
   protected double perimetre;

   // Mètodes abstractes:
   public abstract double calculaArea();
   public abstract double calculaPerimetre();

   //Retorna el Color
   public String getColor(){
      return color;
   }
   //Assigna el Color
    public void setColor(String color){
      this.color=color;
   }

//Retorna la posició de la Figura
    public double [] getPosicion(){
       double [] posicioxy = {x, y};
       return posicioxy;
    }
    //Assigna la posició de la Figura
    public void setPosicio(double[] posicioxy){
       x=posicioxy[1];
       y=posicioxy[2];
    }
} // Final de la classe Figura
```

Quadrat.java

```
public class Quadrat extends Figura {
    private double costat; // longitud dels costats
    // constructors
    public Quadrat() {
            costat=0.0;
    }
    public Quadrat(double costat) {
            this.costat = costat;
    }
    // Calcula l'àrea del quadrat:
    public double calculaArea() {
        area = costat * costat ;
        return area;
    }
    // Calcula el valor del perímetre:
    public double calculaPerimetre(){
        perimetre = 4 * costat;
        return perimetre;
    }
}
```

Cercle.java

```
public class Cercle extends Figura {
        public static final double PI=3.14159265358979323846;
        public double  radi;
         // constructors
           public Cercle(double x, double y, double radi) {  crearCercle(x,y,radi); }
           public Cercle (double radi) {  crearCercle(0.0,0.0,radi);  }
           public Cercle (Cercle c){ crearCercle(c.x,c.y,c.radi); }
           public Cercle() { crearCercle(0.0, 0.0, 1.0); }
           // Mètode de suport
           private void crearCercle(double x, double y, double radi) {
                 this.x=x; this.y=y; this.radi =radi;
           }
        // calcula l'area del cercle
           public double calculaArea() {
              area = PI * radi * radi;
              return area;
           }
           // calcula el valor del perímetre
           public double calculaPerimetre() {
              perimetre = 2 * PI * radi;
              return perimetre;
           }
} // fi de la classe Cercle
```

Cercle.java

```
public class Cercle extends Figura {
        public static final double PI=3.14159265358979323846;
        public double  radi;
         // constructors
           public Cercle(double x, double y, double radi) {
             crearCercle(x,y,radi);
             }
           public Cercle (double radi) {
                crearCercle(0.0,0.0,radi);
             }
           public Cercle (Cercle c){
             double [] pos = c.getPosicio();
             crearCercle( pos[0],pos[1],c.getRadi());
             }
           public Cercle() {
             crearCercle(0.0, 0.0, 1.0);
             }
            // Mètode de suport
           private void crearCercle(double x, double y, double radi) {
                 setPosico({x,y});
                  setRadi(radi);
           }
} // fi de la classe Cercle
```

O millor amb getters i setters…

Consideracions sobre l'herència

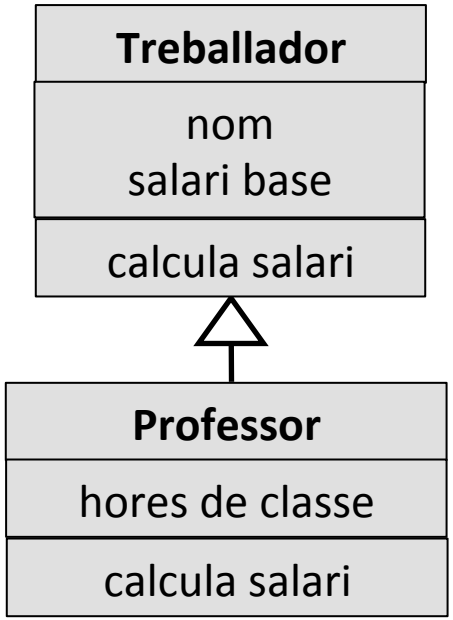
Mètodes:

- Podem realitzar 3 tasques diferents:
  - Afegir mètodes nous
  - Implementar mètodes declarats prèviament com abstractes
  - Tornar a implementar mètodes (sobrescriptura).

Sobreescriptura de mètodes

- Ens permet modificar el comportament d'un mètode definit prèviament en la classe mare per que realitzi altres tasques.
- Mateixa **signatura i tipus de retorn; diferent implementació**.
  - @Override a Netbeans

Exemple



El mètode de la superclasse calcula el salari de qualsevol treballador a partir del salari base.

La subclasse Professor ha de modificar el càlcul del salari afegint un percentatge segons el número de hores.

Ús d'herència

Utilització d'un mètode de la superclasse:

```
public class LaMevaClasse {
 int i;
 public LaMevaClasse () {
        i = 10;
   }
 public void suma_a_i( int j ) {
        i = i + j;
 }
}
```

```
import LaMevaClasse;
public class NovaClasse extends LaMevaClasse {
  public void suma_a_i( int j ) {
    j = i + ( j/2 );
    super.suma_a_i( j );
  }
}
```

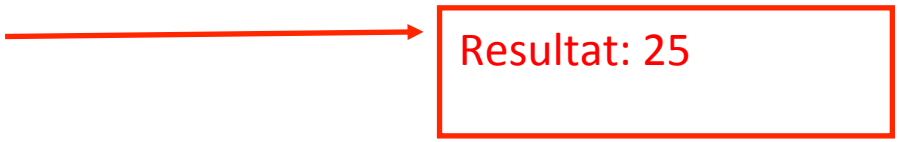
sobreescriptura

Fa referència al mètode de la classe mare

Ús d'herència

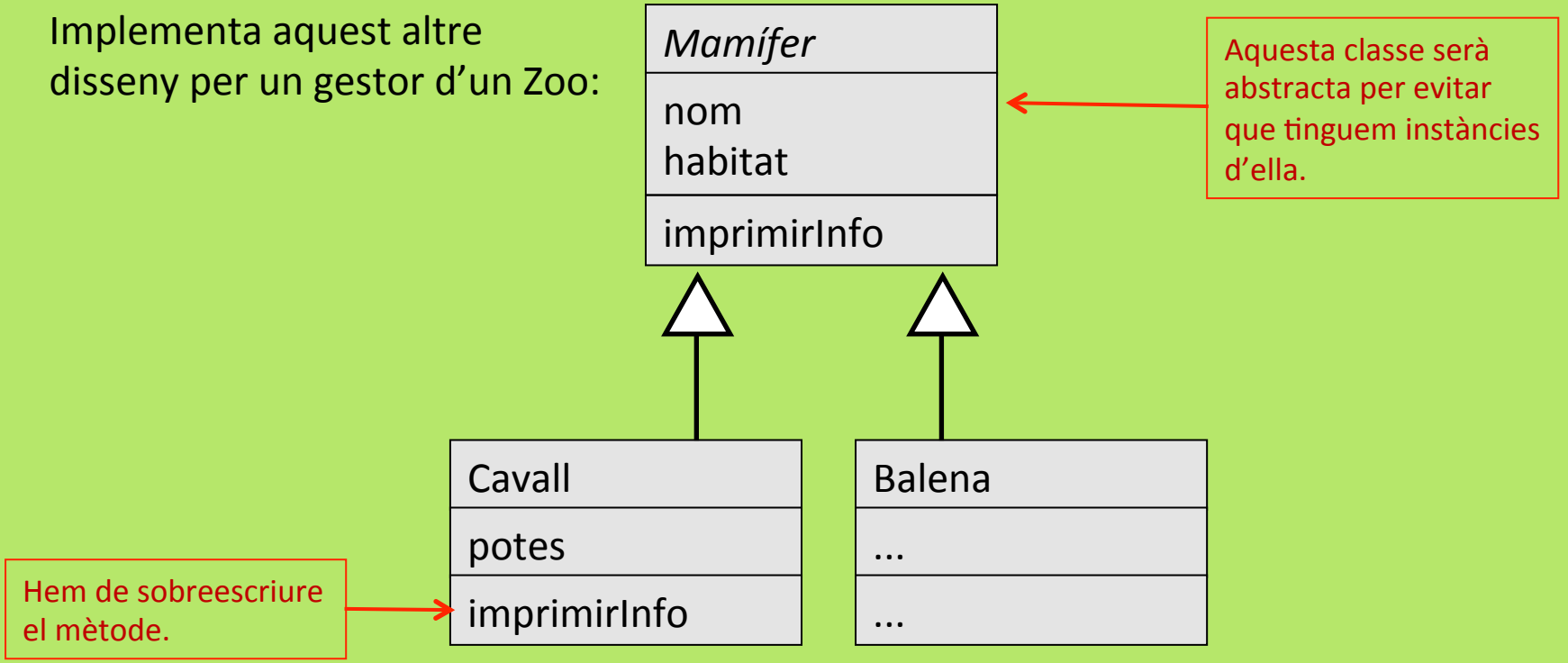
```
public static void main(String[] args) {
        NovaClasse mnc;
       mnc = new NovaClasse();
       mnc.suma_a_i( 10 );

       System.out.println(mnc.getI());
   }
```



Exercici Sobreescriptura

Implementa aquest altre disseny per un gestor d'un Zoo:



Mamífer serà una classe abstracta, ja que hi ha informació d'aquesta classe que no es pot conèixer sense especificar més (saber més sobre l'animal).
Exemple: l'habitat de l'animal: terrestre o marí.

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.