

0101010  
0100101  
1101010

# UD7.1 - HERÈNCIA

Programació –1er DAW/DAM

# 0. CONTINGUTS

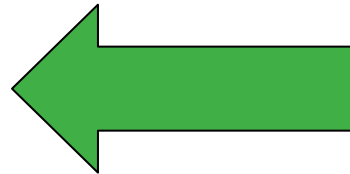
---

- Principis bàsics de la POO
- Herència a Java
  - Concepte d'herència
  - Mecanismes d'herència
  - modificador ***final***
  - referència ***super***
  - Herència i constructors

# 1. PRINCIPIS BÀSICS POO

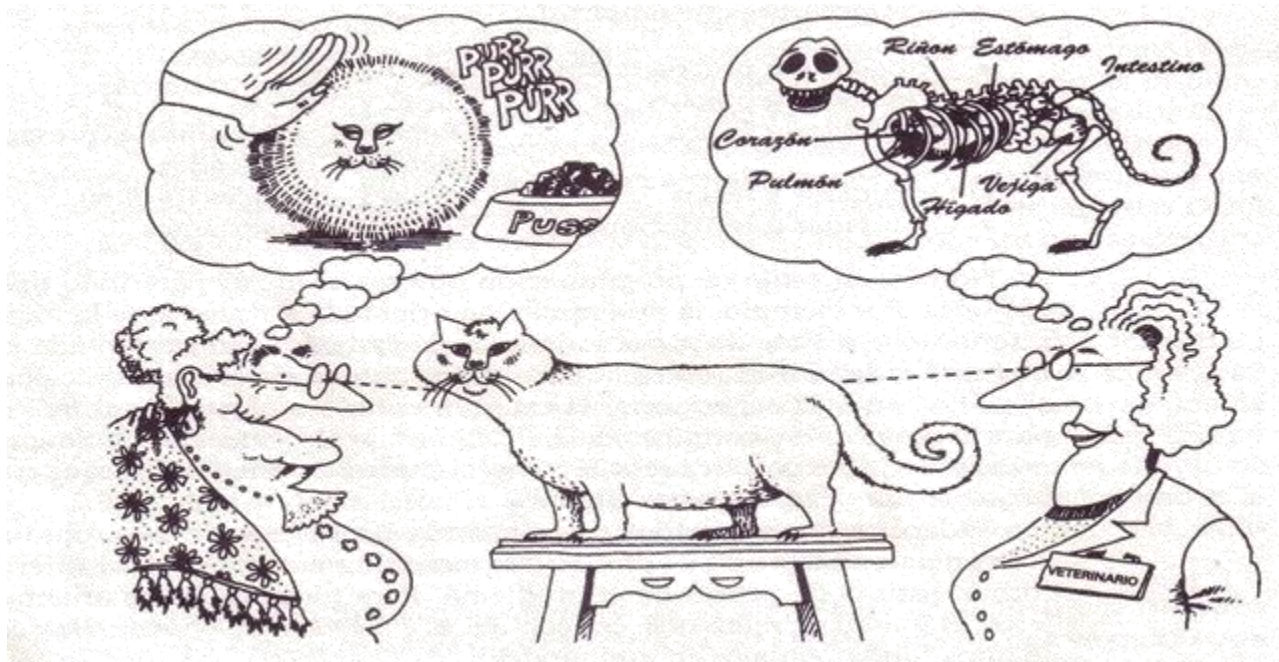
---

- Tot llenguatge orientat a objectes és fonamenta en 4 pilars:
  - Abstracció
  - Encapsulació / Ocultació
  - **Herència**
  - Polimorfisme



# 1.1 ABSTRACCIÓ

- **Mecanisme** que ens permet **determinar**, a partir de l'observació de la realitat, les **classes** i les seves característiques: **DADES i COMPORTAMENT**
  - **Diferents segons l'observador** (*Domini de l'aplicació*)



## 1.2 ENCAPSULACIÓ / OCULTACIÓ

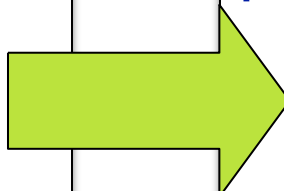
- Els objectes seran “caixes negres”: sabem **que fan** a través de la interfície **però no** sabem **com** ho fan.
  - La **interfície** quedarà definida per el seu **comportament** (accessible mitjançant els **mètodes públics**)



## 1.2 ENCAPSULACIÓ / OCULTACIÓ

- **No necessitem conèixer la implementació interna per poder utilitzar una classe.**
  - Aquesta **pot variar** sense afectar el sistema, sempre que **mantinguem la interfície pública.**

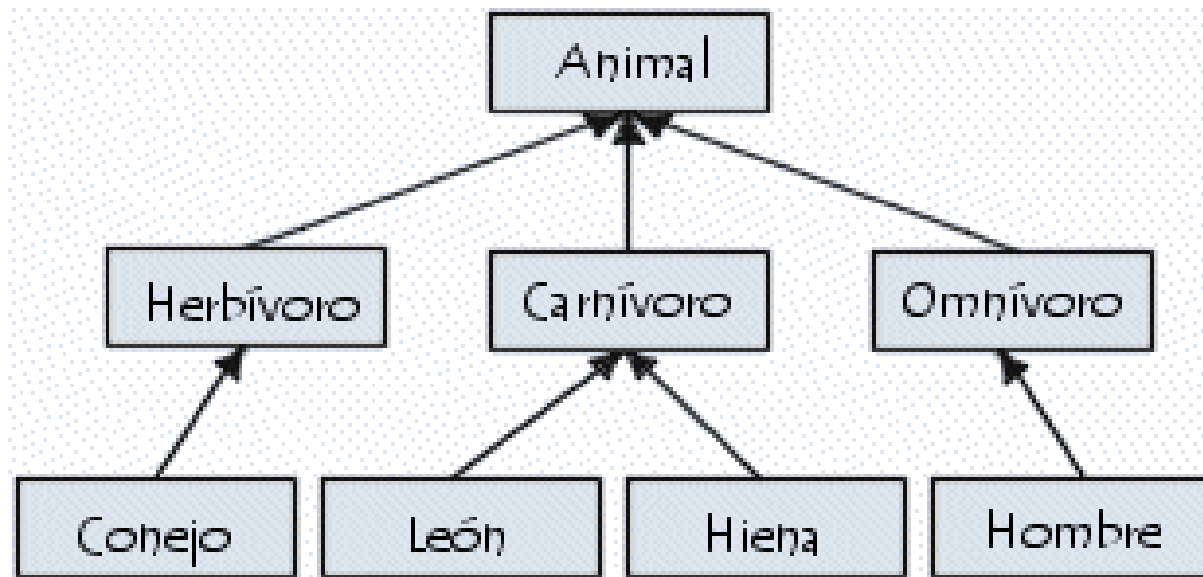
```
public class Data {  
    private int dia;  
    private int mes;  
    private int anyo;  
}
```



```
public class Data {  
    private Long timeEpoch;  
}
```

## 1.3 HERÈNCIA

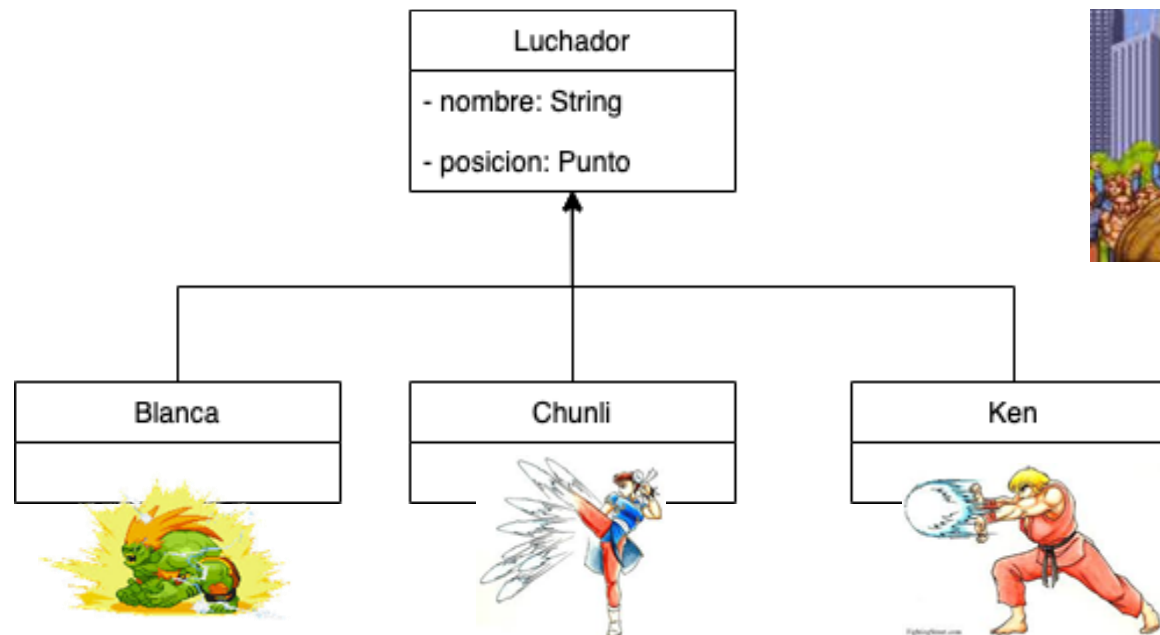
- En moltes ocasions necessitem modelar classes que presenten **característiques i/o comportaments comuns**:



Qualsevol Animal (Conill, Lleó) tindrà **una raça, nom, nº pates,...** No obstant això s'**alimentarà / caminarà** de forma diferent

## 1.3 HERÈNCIA

- **Mecanisme** que ens **permet** crear **noves classes** a partir d'altres **ja existents**.
  - La classe filla (**subclasse**) **hereta** els membres (**variables d'instància i mètodes**) de la classe pare (**superclasse**).
  - Es tracta del principal **mecanisme** de **reutilització** de codi a POO.





## 1.5 IMAGINEM UN PROBLEMA...

---

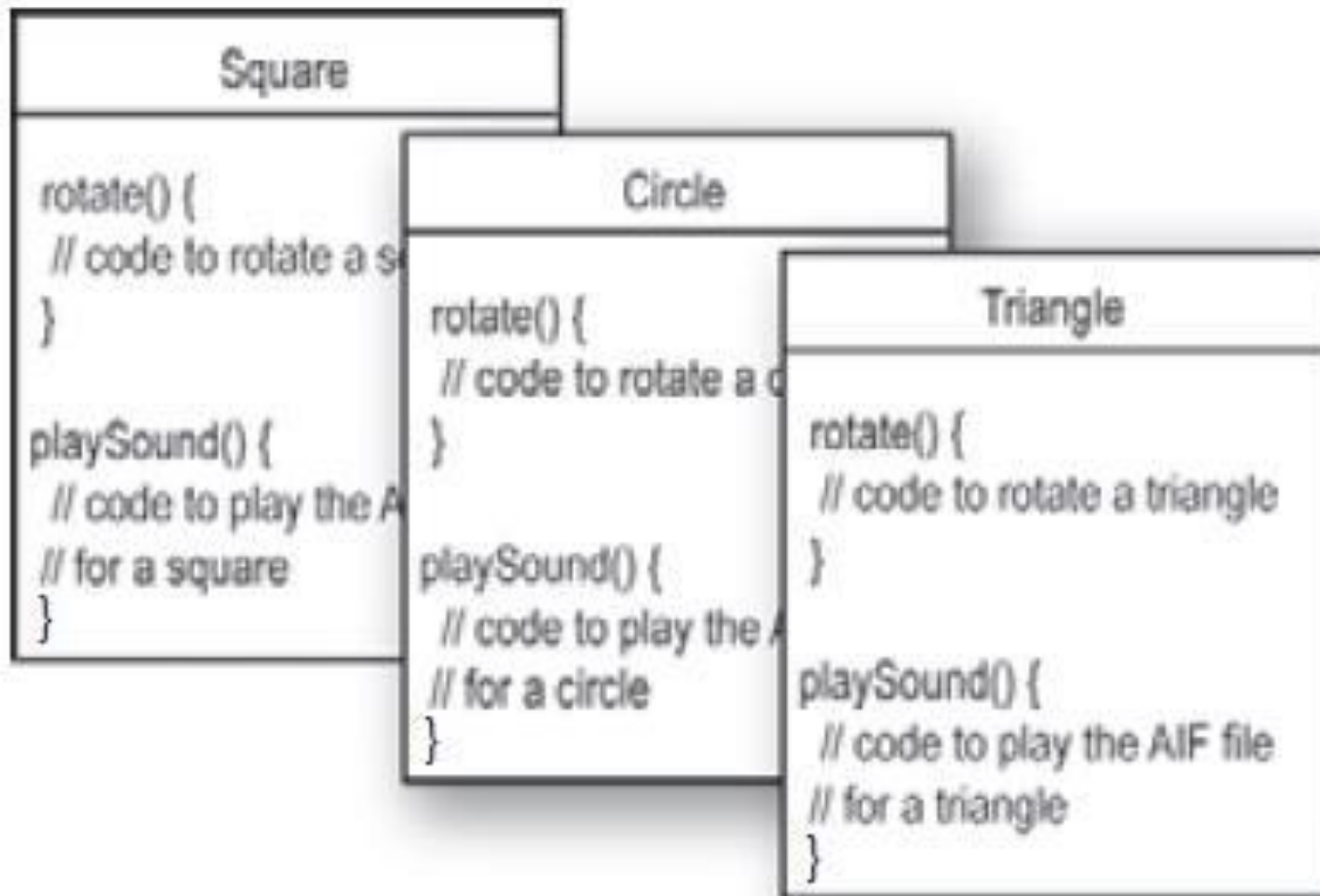
### 1. Especificació del problema...

- Necessitem **representar** diferents **formes** en una aplicació: un **quadrat**, un **cercle** i un **triangle**.
  - Quan l'usuari **fa click en la forma**, aquesta **rotarà 360°** i **emetran** un **só** característic de la forma que representen.



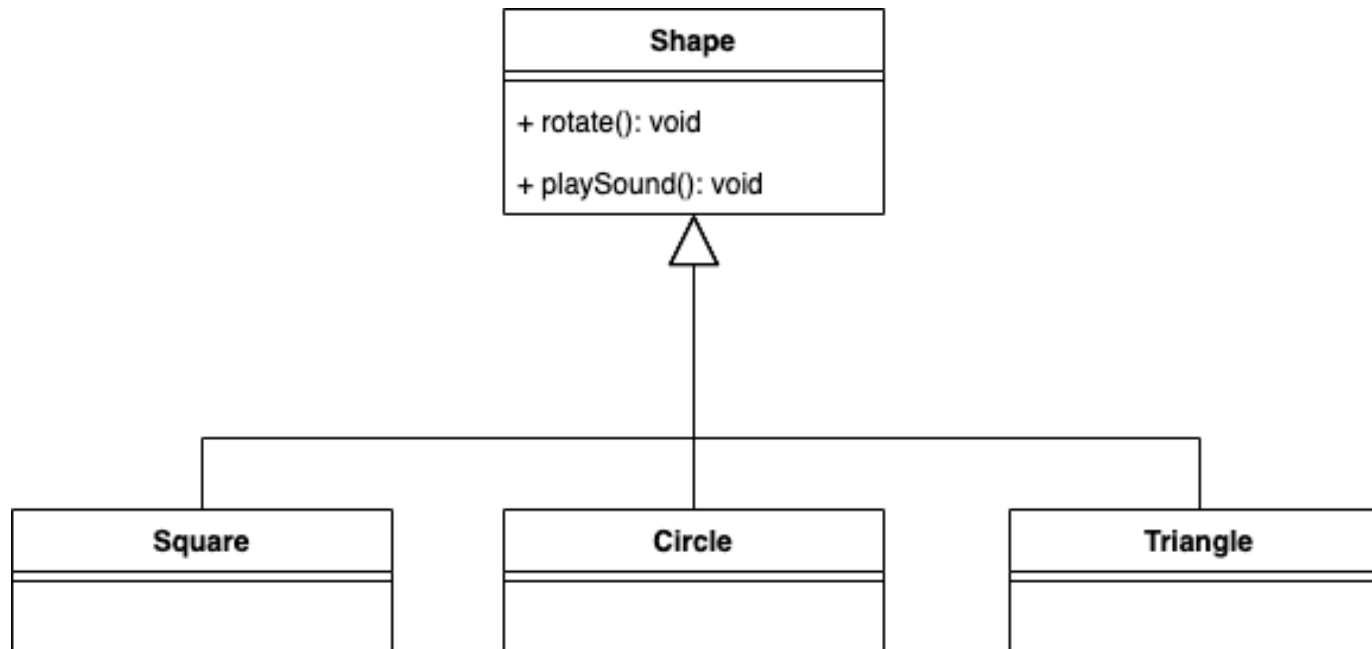
## .1.5 IMAGINEM UN PROBLEMA...

2. Creem una classe per a cada forma...



## .1.5 IMAGINEM UN PROBLEMA...

### 3. Determinació de característiques comunes (I)

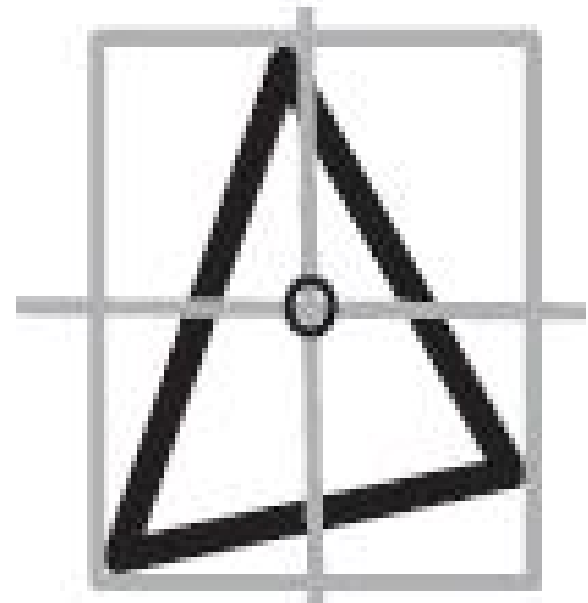


Tenim **3 classes**, totes elles han de **rotar** i **fer sonar** una melodia. Per això, resumim les **característiques comunes** en una **nova classe anomenada Shape** (forma)

## .1.5 IMAGINEM UN PROBLEMA...

4. Definim la implementació del mètode **rotate()** a la classe pare.

- El **comportament** serà **comú** a totes les formes:
  - **Determinar el rectangle** que envolta la forma.
  - **Calcular el centre** de la figura i **rotar** la figura **sobre aquest punt**.

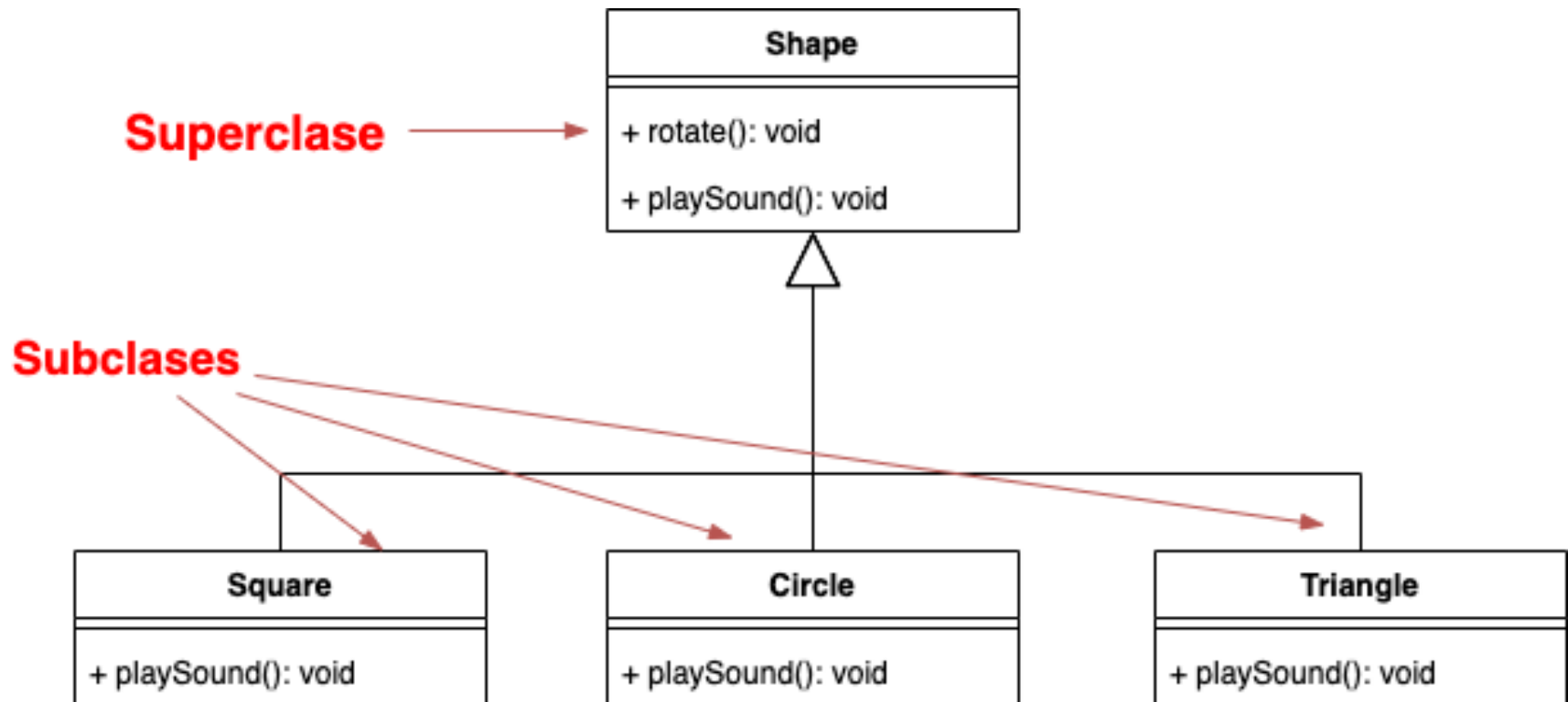


5. Definim la implementació del mètode `playsound()` a la classe pare amb un so per defecte

## .1.5 IMAGINEM UN PROBLEMA...

### 6. Sobreescrivim els mètodes necessaris a les classes filles...

- La implementació del mètode `playSound()` ha de proveir d'un so característic per a cada forma



## 2. HERÈNCIA A JAVA. SINTAXIS

- Per a la **definició** de la **herència** entre 2 classes, es fa servir la paraula reservada ***extends*** a la classe filla seguida del nom de la classe pare:

```
class Persona {  
    private String DNI;  
  
    public void saluda() {  
        System.out.println("Hola soc una persona");  
    }  
  
    public String getDNI() {  
        return DNI;  
    }  
}
```

L'anotació @Override ens assegura que la **definició del mètode en la línia posterior es exactament la mateixa que aquell que es vol sobre escriure** de la superclasse. És sempre recomanable escriure-la

```
class PersonaAnglesa extends Persona {  
    @Override  
    public void saluda() {  
        System.out.println("Hello, I'm an english person");  
    }  
}
```

## 2.1 SINTÀXI. EXEMPLE SOBRE-ESCRITURA

- La classe **PersonaAnglesa**:
  - Hereta el **atribut** nom de la classe **Persona**, així com el **mètode** **getDNI()**
  - Sobre-escriu (*overrides*) el mètode **saluda()** de manera que la salutació siga en anglès

```
public static void main(String[] args) {  
  
    Persona persona = new Persona();  
    PersonaInglesa personaInglesa = new PersonaInglesa();  
  
    persona.saluda(); //Hola soc una persona  
    personaInglesa.saluda(); //Hi I'm an english person  
    System.out.print(persona.getDNI()); // NULL  
  
}
```

## 2.2 ACTIVITAT PRÈVIA

---

- **Activitat 1.** A partir de la classe `Persona` definida a les transparències anteriors, defineix **2 noves classes**, `PersonaRusa` i `PersonaFrancesa`. Redefineix el mètode `saluda()`, de manera que cadascuna mostre les seues dades en l'idioma corresponent.
- *Per acabar, crea una classe `TestPersona` que cree un objecte de cada tipus i saluden tots.*



## 2.3 SINTÀXI. EXEMPLE ADDICIONAL

```
public class Vehicle {

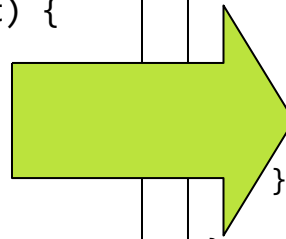
    private int velocitat;

    private int rodes;

    public void accelerar(int velocitat) {
        this.velocitat += velocitat;
    }

    public void frenar() {
        this.velocitat = 0;
    }

}
```



```
public class Cotxe extends Vehicle {

    private int carburant;

    public void repostar(int quantitat) {
        carburant += quantitat;
    }

}
```

Una **bicicleta** també seria un **vehicle** i no necessita carburant

```
public class Garatge {

    public static void main(String[] args) {
        Cotxe coche = new Cotxe();
        coche.accelerar(500); // Mètode heretat
        coche.repostar(12); // Mètode propi
    }

}
```

## 2.4 MODIFICADORS D'ACCÉS

- La **subclasse** té (o **hereta**) **TOTS** els **membres** (variables d'instància i mètodes) de la **superclasse**:
  - Seran **accessibles** els membres *protected*, *public* i *package*.
  - No** seran **accessibles** els membres *private*.

Visibilidad	Public	Protected	Default	Private
Desde la misma Clase	SI	SI	SI	SI
Desde cualquier Clase del mismo Paquete	SI	SI	SI	NO
Desde una SubClase del mismo Paquete	SI	SI	SI	NO
Desde una SubClase fuera del mismo Paquete	SI	SI, a través de la herencia	NO	NO
Desde cualquier Clase fuera del Paquete	SI	NO	NO	NO

## 2.5 LA PARAULA RESERVADA **super**

- Podem definir **variables** de **instancia** a la subclasse amb el **mateix nom** que a la superclasse:
  - Les **variables d'instància** de la **superclasse** queden "**ocultes**".
- Podem definir un **mètode** a la subclasse amb el **mateix nom** i la **mateixa capçalera** que a la superclasse.
  - Duem a terme una **redefinició de l'acció** que porta cap el mètode.

Podem fer **referència al mètode o variable de la superclasse**, mitjançant la paraula reservada **super**

## 2.5 LA PARAULA RESERVADA *super*

- La paraula **reservada** *super* ens permet accedir a una **variable d'instància** o **mètode** de la superclasse:
  - *this* → fa referència a la classe actual
  - *super* → fa referència a la superclasse respecte a la classe actual

```
public class Vehicle {  
  
    private int velocitat;  
  
    private int rodes;  
  
    public void accelerar() {  
        this.velocitat += 10;  
    }  
  
    public void frenar() {  
        this.velocitat = 0;  
    }  
  
}
```

```
public class Cotxe extends Vehicle {  
  
    private float carburant;  
  
    @Override  
    public void accelerar() {  
        super.accelerar();  
        carburant -= 0.5;  
    }  
  
    public void proveir(float quantitat) {  
        carburant += quantitat;  
    }  
  
}
```

## 2.6. ACTIVITATS PRÈVIES

---

**Activitat 2.** Donada la següent jerarquia de classes:

```
public class Medico {  
    protected boolean trabajaEnHospital;  
    public void atiendeAPaciente(){  
    }  
}  
  
public class MedicoFamilia extends Medico{  
    protected int numPacientesAconsejados;  
    public void daConsejo(){  
    }  
}  
  
public class Cirujano extends Medico{  
    @Override  
    public void atiendeAPaciente() {  
        super.atiendeAPaciente();  
    }  
    public void hazIncision(){  
    }  
}
```

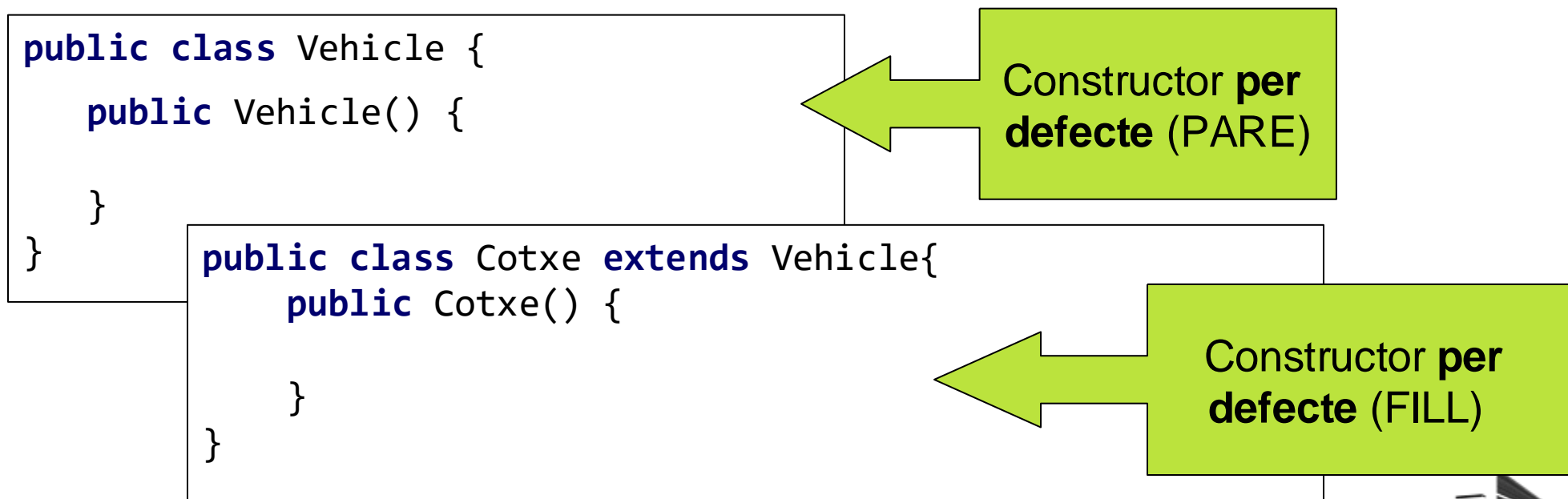
## 2.6. ACTIVITATS PRÈVIES

---

- **Contesta les preguntes següents:**
  - Quantes variables d'instància té la classe Cirujano?
  - Quantes variables té la classe MedicoFamilia?
  - Quants mètodes té la classe Medico?
  - Quants mètodes té la classe Cirujano?
  - Quants mètodes té la classe MedicoFamilia?
  - Pot un MedicoFamilia atendre un pacient? Quin mètode s'executarà? I un Cirujano?
  - Pot un MedicoFamilia fer una incisió? Per què?

## 2.7 HERÈNCIA I CONSTRUCTORS

- La subclasse no hereta els constructors:
  - Cada **nova classe** (fins i tot les derivades), deu **definir** els seus **constructors**.
  - Si no s'implementa cap constructor, es genera un **predeterminat sense arguments**.



## 2.7 HERÈNCIA I CONSTRUCTORS

- Si la **primera instrucció** d'un **constructor** d'una subclasse és una sentència que **no és ni super ni this** ...
  - Java, realitza implícitament una crida **super()** al constructor per defecte de la superclasse
  - Si el **constructor per defecte** (sense paràmetres) no existeix (ja hi ha un amb paràmetres) → **excepció (error)**.

Això **no implica** que sempre haguem de definir un **constructor per defecte**.



## 2.7 HERÈNCIA I CONSTRUCTORS

---

- Ja saps, que els **constructors** tenen la possibilitat d'**invocar** a un **altre constructor** de la seva pròpia classe amb la **sentència `this(...)`**.
- Per això, els constructors de les **subclasses** tenen la possibilitat d'**invocar** als **constructors** de les **superclasses** amb la sentència **`super(...)`**.
  - Tant la crida **`super(...)`**, com **`this (...)`** ha de ser **obligatòriament** la **primera sentència** del constructor.

## 2.7 HERÈNCIA I CONSTRUCTORS

### Vehiculo.java

```
public class Vehicle {
    private int velocitat;
    private int rodes;

    public Vehicle(int velocitat) {
        this.velocitat = velocitat;
        this.rodes = 4;
    }

    protected void accelerar() {
        this.velocitat += 10;
    }

    public void frenar() {
        this.velocitat = 0;
    }
}
```

### Cotxe.java

```
public class Cotxe extends Vehicle {
    private float carburant;

    public Cotxe(int velocitat, float carburant) {
        super(velocitat);
        this.carburant = carburant;
    }

    @Override
    public void accelerar() {
        super.accelerar();
        carburant -= 0.5;
    }

    public void proveir(int quantitat) {
        carburant += quantitat;
    }
}
```



## 2.8. ACTIVITATS PRÈVIES

---

**Activitat 3.** Du a terme la implementació de les classes `Vehicle` i `Cotxe` definides a la **transparència anterior**.

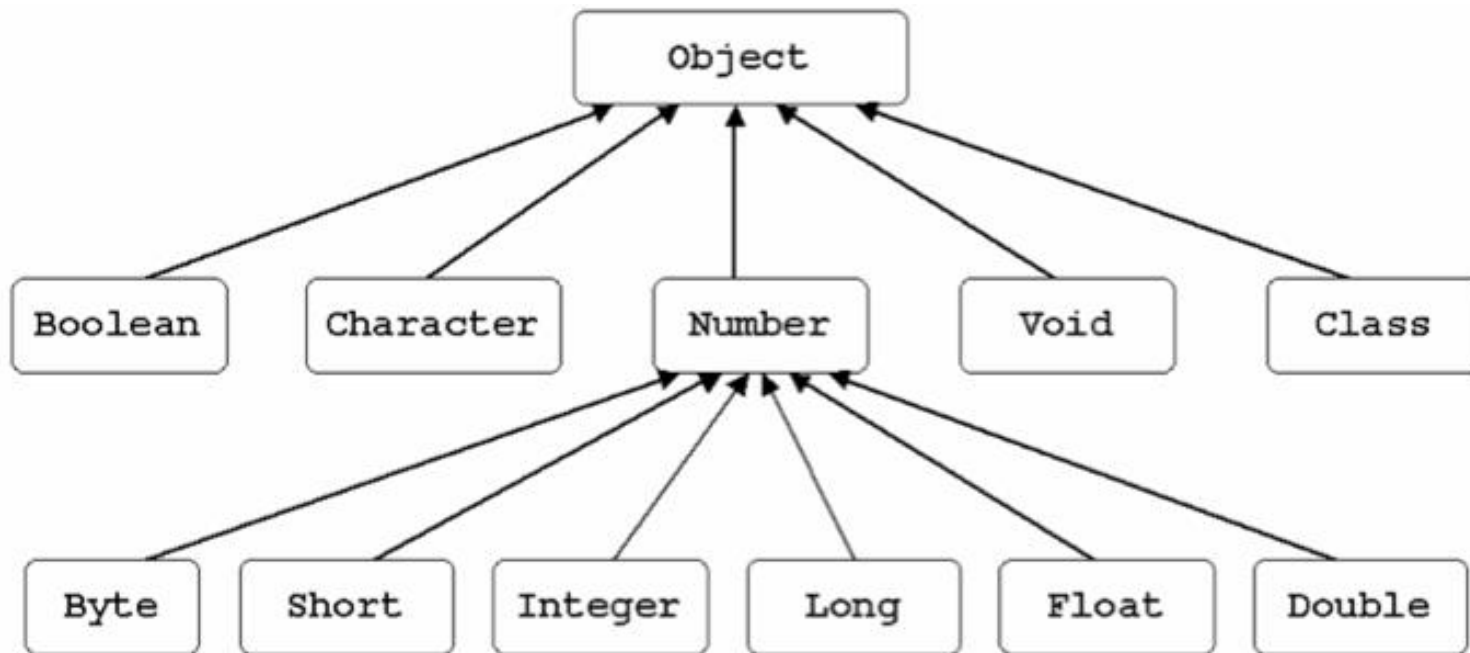
- Implementa una nova classe `CotxeEsportiu` que herete de `Cotxe`. Aquesta nova classe disposarà d'un atribut propi de **tipus booleà** `descapotable`. Quan accelere consumirà **1.5 litres** de carburant.
- Afegeix un nou **atribut** `matricula`. Aquest ha de pertànyer als 2 tipus de cotxes. Afegeix-ho també al constructor corresponent.

Per acabar, crea una classe `TestCotxe` i realitza les accions següents:

- **Defineix 2 cotxes**, un haurà de ser esportiu.
- **Reposta** als dos cotxes **50 litres**.
- Accelera els cotxes fins **50 km/hora** i fes que es paren.
- **Mostra el nivell de carburant** de cadascun d'ells

### 3. HERÈNCIA A JAVA

- A **Java** és **especialment rellevant** l'herència perquè s'utilitza (intrínsecament) en el propi llenguatge a partir del conjunt de llibreries que proporciona
  - **Tota classe a Java hereta de la classe Object.**



### 3. HERÈNCIA A JAVA

- La classe `Object` implementa una sèrie de mètodes. De tots ells, ens centrarem en el mètode: `public String toString()`

```
public class Robot {  
    private String nombre;  
    private int numSerie;  
    //.....  
}
```

```
public class TestRobot {  
    public static void main(String[] args) {  
        Robot robot = new Robot();  
        System.out.println(robot);  
    }  
}
```

*Robot@bc197456*

L'implementació  
d'`Object` retorna el següent  
String

Es pot omitir l'invocació a  
`toString()`

### 3. HERÈNCIA A JAVA

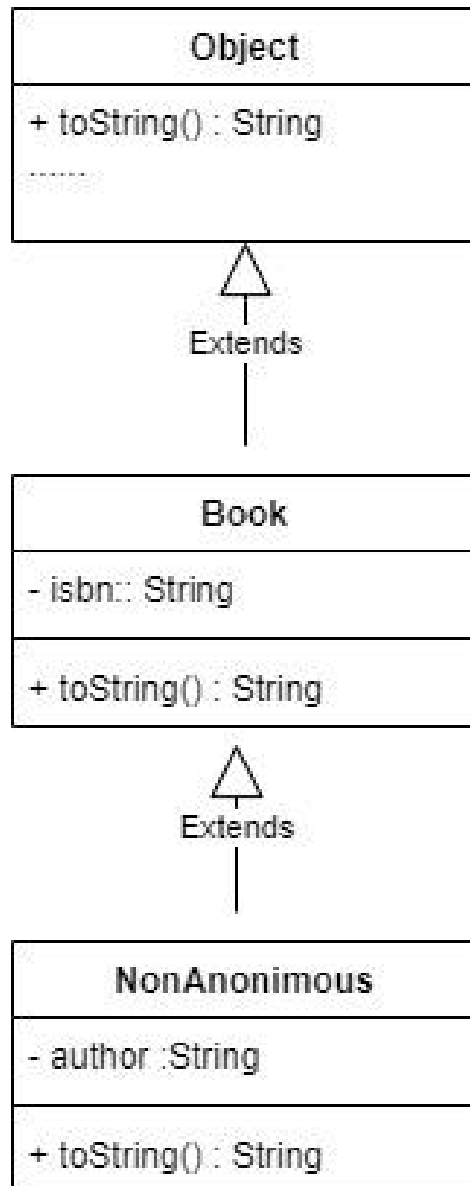
- Normalment, es sobreescriu el mètode toString de Object per a obtenir informació d'un objecte

```
public class Robot {  
    private String nombre;  
    private int numSerie;  
  
    //...  
  
    @Override  
    public String toString() {  
        return "nombre='" + nombre + '\\'  
            + ", numSerie=" + numSerie;  
    }  
}
```

Quan s'executa el  
mètode main

nombre=Pepe, numSerie=123456

### 3. HERÈNCIA A JAVA



- Si volem que el mètode `toString` de **NonAnonymous** genere un `String` que només continga la informació del autor del llibre, haurem de definir-lo d'aquesta manera:

```
public String toString() {
    return "Author: " + this.author;
}
```

- Per contra, si volem que continga la informació del ISBN, a més de l'autor:

```
public String toString() {
    return super.toString()
        + "Author: " + this.author;
}
```

## 2.8. ACTIVITATS PRÈVIES

---


**Activitat 4:** Traça d'un exemple de herència pare-  
fill-avi



## 2.9 MODIFICADOR *final*

- Per impedir que es puga **redefinir una variable** de instància o un **mètode** anteposarem el **modificador *final***.
  - El **modificador *final*** aplicat a una classe fa que no es puguin definir classes derivades.

```
class Persona {  
  
    private String nom;  
    private String cognoms;  
  
    final public void mostrarInfo () {  
        System.out.println(nom + " " + cognoms);  
    }  
  
}
```



```
class PersonaAnglesa extends Persona {  
    @Override  
    public void mostrarInfo () {  
        System.out.println(nom + " " + cognoms);  
    }  
  
}
```

### 3. COMPOSICIÓ VS HERÈNCIA

#### Test SER o NO SER

- La **herència** és un **mecanisme molt potent**, no obstant això, la seua sobre-utilització pot portar a codi altament ineficient i de baixa qualitat.
- Sempre que ens plantegem utilitzar herència **hauríem de passar el test ÉS UN**
  - La subclasse **ÉS UNA** superclasse.
  - un cirurgià **ÉS UN** metge.
  - Una banyera **NO ÉS** un bany, però un **bany TÉ** una banyera.
    - En aquest cas hauríem de **utilitzar composició**

La subclasse deu poder fer qualsevol cosa que faci la superclasse  
(o fins i tot, més coses)

### 3. COMPOSICIÓ VS HERÈNCIA

---

#### Quan fer ús de l'herència?

- UTILITZA herència quan vulgues **modelar** un **comportament més específic** que aquell definit a la superclasse. (*Especialització*).
  - Ex. A partir d'Alumne generem AlumnePractiques
- UTILITZA herència quan el **comportament** de diverses classes sigui **igual** i **general**. (*Generalització*)
  - Ex. A partir d'animals generem la Classe Animal
- **NO UTILITZES** herència quan podem reutilitzar codi però **no es compleix** la regla **SER**.

## 4. RESUM PUNTS IMPORTANTS

---

- Si una subclasse **ESTÉN** d'una superclasse.
  - La subclasse hereta tot de la superclasse.
- Els **mètodes heretats poden** ser **sobreescrits** per la subclasse.
  - Les **variables d'instància (atributs)** no es **sobreescriuen**; si es **tornen a definir**, no seran les mateixes.
- Emprarem el **test SER** per "**verificar**" la jerarquia d'**herència**.
- La **relació SER** treballa en una **única direcció** (el fill hereta del pare però no al revés).
- Els **constructors no són heretats** per les **classes filles**

## 4. RESUM PUNTS IMPORTANTS

---

Això és tot... de moment :-)