

0101010  
0100101  
1101010

# UD7.3 - INTERFÍCIES

Programació – 1er DAW/DAM

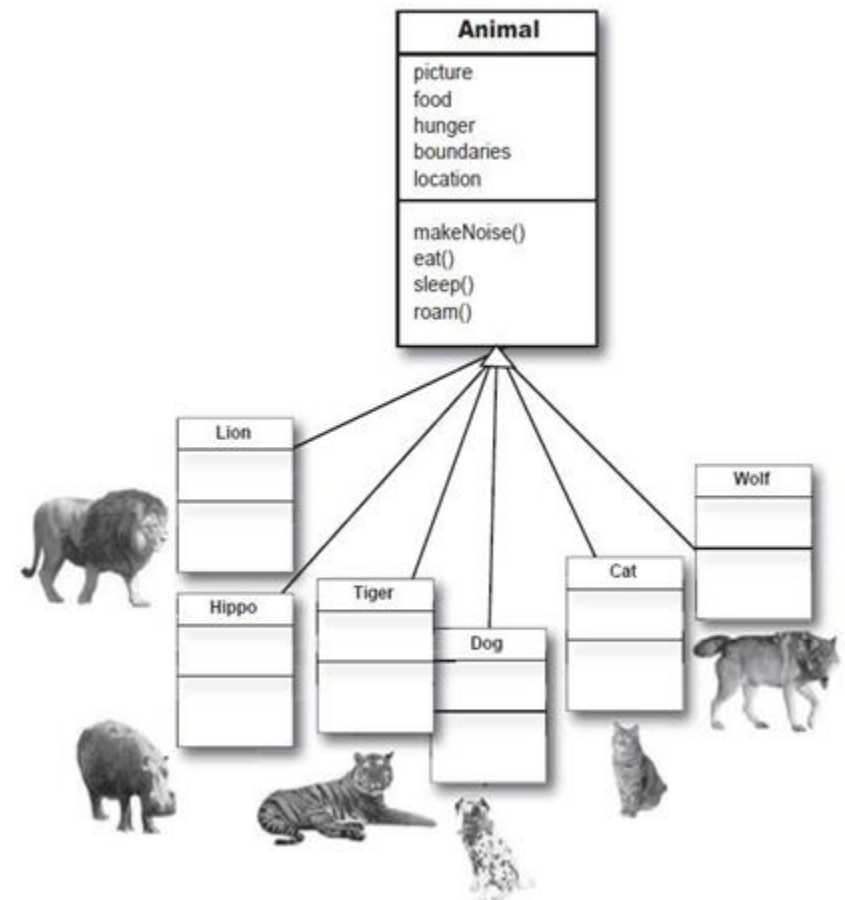
# 0. CONTINGUTS

---

- Situació d'exemple inicial
- Interfícies
  - Concepte
  - Definició i implementació
  - Característiques
  - Exemples

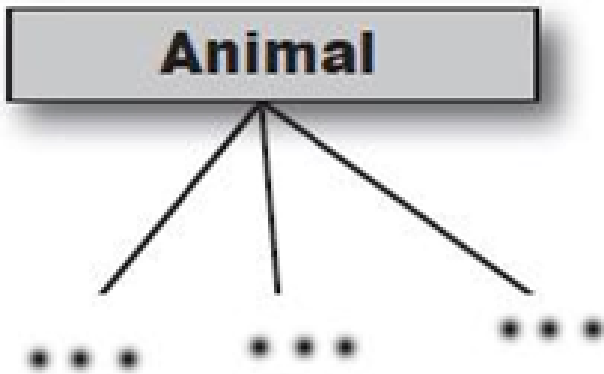
# 1. SITUACIÓ DE PARTIDA

- Imaginem que algú vol fer servir les classes d'animals que hem dissenyat per crear un programa d'una **botiga de mascotes**.
- Hi ha uns **comportaments comuns** a totes les **mascotes: gossos i gats** que no ho són a per a la resta d'animals.
- Què **opcions** tenim per descriure aquest comportament?



# DISSENY 1: DEFINICIÓ A LA CLASSE ANIMAL

- Si definim els mètodes de les mascotes a la classe **Animal**.



```
abstract public class Animal {

    public void beFriendly(){
        System.out.println("Mou la cua");
    }

    public void play() {
        System.out.println("Portar pilota");
    }

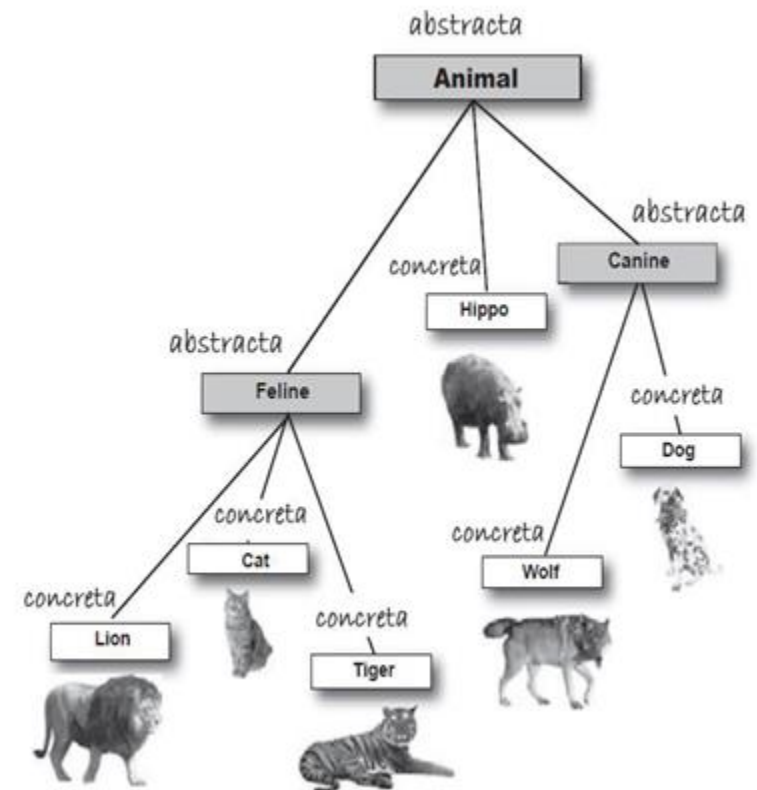
}
```

- Tots els tipus d'animals heretaran les característiques de les mascotes.

**Un lleó o un llop s'han de comportar com una mascota?**

## DISSENY 2: MÈTODES *ABSTRACTES* A LA CLASSE ANIMAL

- Definim els mètodes a la classe **Animal** com *abstract*.
  - Les classes filles han de **sobreescriure** (en realitat, implementar) els mètodes de la classe pare, encara que no siguin mascotes.
  - Les classes que no siguin mascotes tindran un mètode buit.



Principi **YAGNI**  
(*You aren't gonna need it*)

## DISSENY 3: MÈTODES EN ALGUNES CLASSES

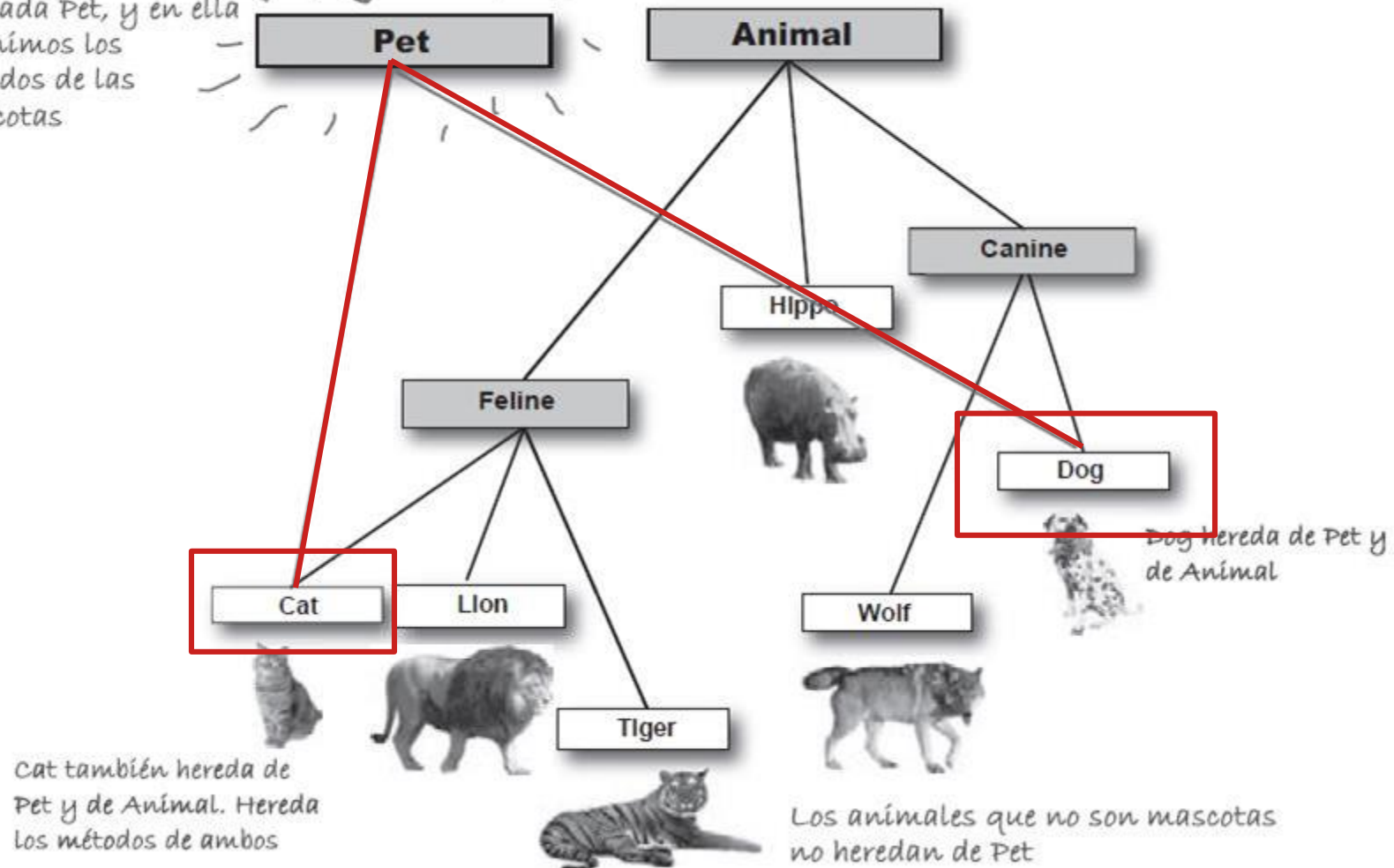
- Definim els mètodes de les mascotes a les classes que calga: **Dog** i **Cat**.
- ✓ – No tindrem animals que no siguin mascotes amb un comportament de mascotes.
- ✗ – PROBLEMA: NO podem utilitzar POLIMORFISME per a les mascotes.

```
public class Cat {  
  
    public void beFriendly(){  
        System.out.println("Ronroneja");  
    }  
  
    public void play() {  
        System.out.println("Agafant el cabdell de llana");  
    }  
}
```

```
public class Dog {  
  
    public void beFriendly(){  
        System.out.println("Meneja la cua");  
    }  
  
    public void play() {  
        System.out.println("Agafa la pilota");  
    }  
}
```

# DISSENY 4: CREEM UNA NOVA CLASSE ABSTRACTA

Creemos una nueva superclase abstracta llamada Pet, y en ella definimos los métodos de las mascotas

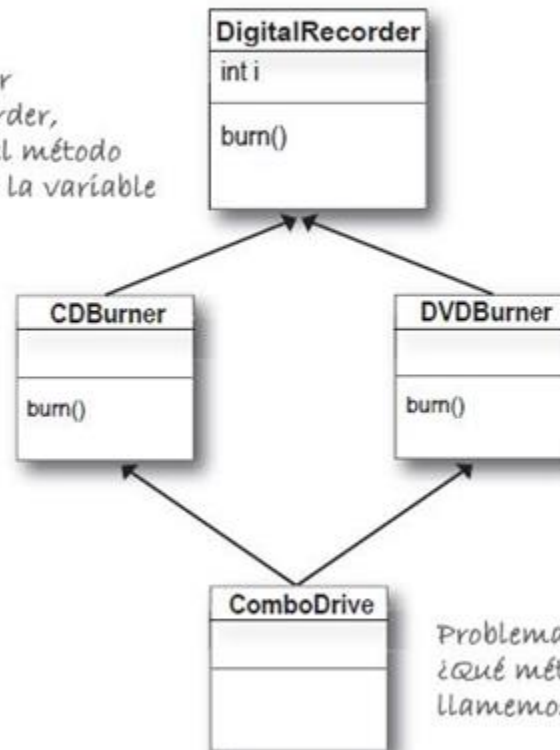


# PERÒ HI HA UN PROBLEMA ...

- Java no permet la herència múltiple.
- L'herència múltiple **provoca un problema**:

## Deadly Diamond of Death (Problema del Diamante de la muerte)

CDBurner y DVDBurner heredan de DigitalRecorder, y ambos sobrescriben el método burn(). Ambos heredan la variable de instancia "i".



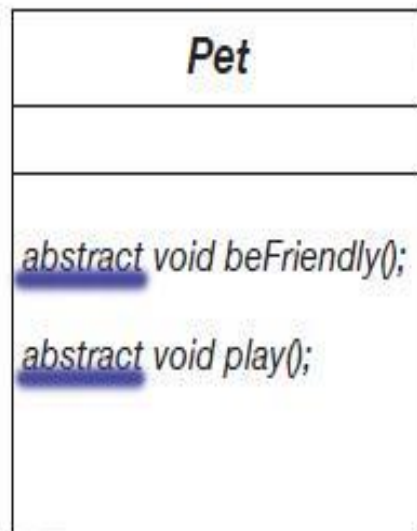
Imagina que la variable de instancia "i" es usada por ambos, CDBurner y DVDBurner, con diferentes valores. ¿Qué pasará si ComboDrive necesita usar ambos valores de "i"?

Problema de la múltiple herencia.  
¿Qué método burn() se ejecutará cuando llamemos al método burn() desde ComboDrive()?



## 2. INTERFÍCIES

- Una interfície és una **classe 100% abstracta**.
- **Tots els mètodes** d'una *interfície* són **abstractes** i han de ser sobreescrits a les classes concretes.
- Ens permeten definir una **sèrie de comportaments comuns** o **rols** a **diferents classes** sense que estiguen **relacionades jeràrquicament**. (herència)



Una interface es como una  
clase 100% abstracta

Todos los métodos en una interface  
son abstract. Cualquier clase que  
SEA una mascota, DEBE implementar  
(sobrecribir) los métodos de Pet.

## 2.1 DEFINIR I IMPLEMENTAR (I)

Para DEFINIR una interface:

```
public interface Pet {...}
```

↖ usa la palabra reservada "interface"  
en vez de "class"

Para IMPLEMENTAR una interface:

```
public class Dog extends Canine implements Pet {...}
```

↖ usa la palabra reservada "implements" seguida del nombre de la interface. Fíjate que, además de implementar la interfaz Pet, estás heredando de la clase Canine.

## 2.1 DEFINIR I IMPLEMENTAR (II)

Escribimos "interface"  
en lugar de "class"

Los métodos de la interface se definen como **public** y **abstract**,  
aunque no es necesario escribirlo, ya lo son.

```
public interface Pet {
    public abstract void beFriendly();
    public abstract void play();
}
```

Todos los métodos de la  
interface son abstractos, deben  
ir con punto y coma (;)  
Recuerda que no tienen cuerpo  
con el código.

Dog ES un Animal  
y Dog es un Pet

```
public class Dog extends Canine implements Pet {
    public void beFriendly() {...}
    public void play() {...}

    public void roam() {...}
    public void eat() {...}
}
```

Escribimos "implements"  
seguido del nombre de la  
interface

Aquí debemos implementar los  
métodos. Fíjate en los corchetes, no  
va con punto y coma.

Estos métodos son los métodos normales  
sobrescritos heredados de las superclases

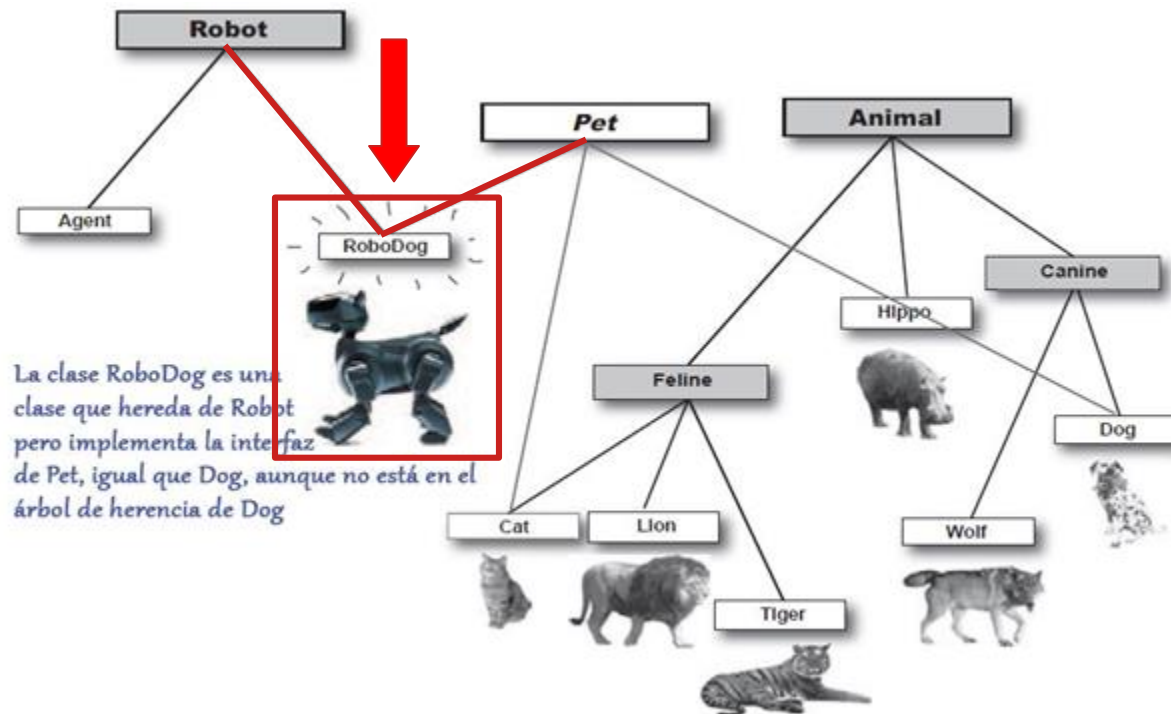
## 2.2 CARACTERÍSTIQUES

---

- Es fa servir la paraula reservada ***interface***.
- Tots els mètodes són abstract i public.
- **No** tenen **constructors**.
- **Només** poden tenir **atributs** de tipus "**public static final**", és a dir, atributs de classe, públics i constants.
- Les **classes implementen** les **interfícies** (Implements) **en lloc d'heretar** (extends) de altres classes.
- Les **classes** poden **implementar diverses interfícies**, però només **heretar d'una classe**.

## 2.2 CARACTERÍSTIQUES

- Classes que pertanyen a **diferents arbres d'herència** poden **implementar una mateixa interfície**.

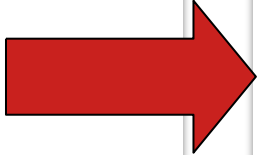


*D'aquesta manera podrem tractar com una Mascota tant a un **RoboDog** com a un **Dog***

## 2.2 CARACTERÍSTIQUES

- Qualsevol **element** del sistema que necessite interactuar amb una **mascota** només necessita **accés** a aquells mètodes que **defineixen** el **comportament** de l'objecte com **mascota**.

```
public interface Pet {  
    void beFriendly();  
    void play();  
}
```



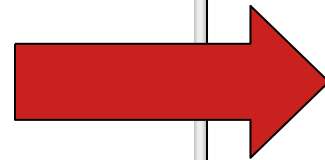
```
public class RoboDog extends Robot implements Pet {  
    public void beFriendly(){  
        System.out.println("Mou el cap");  
    }  
    public void play(){  
        ....  
    }  
}
```

```
public class Dog extends Canine implements Pet {  
    public void beFriendly(){  
        System.out.println("Mou la cua");  
    }  
    public void play(){  
        ....  
    }  
}
```

## 2.2 CARACTERÍSTIQUES

- Un objecte de tipus **Persona**, no necessita saber si es tracta d'un **gos**, un **gat** o un **robogós** per interactuar amb ell.

```
public class TestPet {  
  
    public static void main(String[] args){  
  
        Persona persona1 = new Persona();  
  
        RoboDog robot = new RoboDog();  
        Dog dog = new Dog();  
        persona1.interactuar(robot);  
        persona1.interactuar(dog);  
    }  
}
```



```
public class Persona {  
  
    public void interactuar(Pet mascota) {  
  
        mascota.beFriendly();  
        mascota.play();  
    }  
}
```



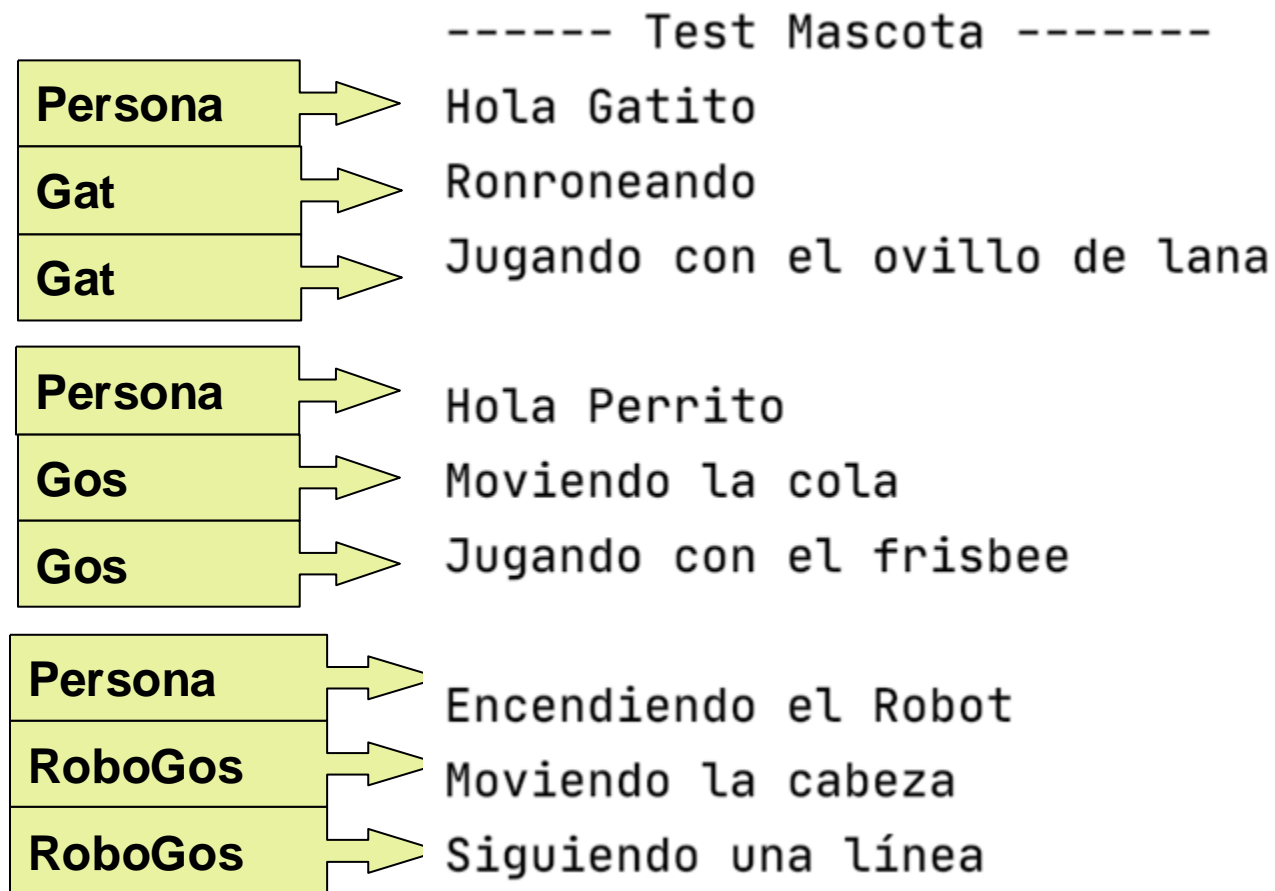
## 2.2.1 ACTIVITAT PRÈVIA

- **Activitat 12.** A partir del codi desenvolupat a l'**activitat 9** crea una interfície `Mascota` amb els mètodes `serAmigable()` (`beFriendly`) i `jugar()` (`play`) (tal com apareix a les transparències). `Gos` i `Gat` hauran d'implementar aquesta interfície.
  - Afegeix una classe `RoboGos` que implemente la interfície `Mascota`.
  - Partint de la classe `Persona` desenvolupada a l'**activitat 7** afegeix un mètode `interactuar(Mascota m)` que li permeti jugar i ser amistós amb qualsevol objecte que implemente la interfície `Mascota`.

*Per acabar, crea una classe `TestMascota` e instància diferents objectes de tipus `Gos`, `RoboGos` i `Gat`, Crea un objecte de tipus `Persona` i fes que interactúe amb totes les mascotes.*



## 2.2.1 ACTIVITAT EXEMPLE



## 2.3 HERÈNCIA A LES *INTERFÍCIES* III

- Les **interfícies** també **poden heretar** entre elles. A més permeten **herència múltiple**.

```
interface I1 {  
    void metodo1 ();  
    void metodo2 ();  
}  
interface I2 {  
    void metodo3 ();  
}
```

```
interface I3 extends I1, I2 {  
    void metodo4 ();  
}
```

Herència múltiple

metodo1(), metodo2() i metodo3() s'hereten de I1 e I2

```
class C implements I3 {  
  
}
```

ClasseC hauria d'implementar metodo1, metodo2, metodo3 i metodo4

## 2.4 HERÈNCIA VS INTERFACES

---

- **Herència**

- Les subclasses són al mateix arbre d'herència.
- Una classe només pot heretar una altra classe.

- **Interfícies**

- No cal pertànyer al mateix arbre d'herència.
- **Només defineixen rols o comportaments comuns**
- Una classe pot implementar diverses interfícies.

## 2.4 HERÈNCIA VS INTERFACES

---

Què definir a cada situació?

- Una classe no ha d'heretar d'un altra quan **no** compleix el **test SER**.
- Crea una **subclasse** quan calga una **versió més específica** de la classe i es necessita **sobreescriure** o **afegir** nous **comportaments**.
- Feu servir una **classe abstracta** per definir una **plantilla** per al conjunt de subclasses i garantir que no es puguin crear objectes d'aquest tipus.
- Crea **interfícies** per a modelar **comportaments** de **classes** que es troben a **arbres d'herència diferents**.
- Utilitza **interfícies** per «simular» l'herència múltiple.

## 2.4 HERÈNCIA VS INTERFACES

---

- Això és tot... de moment :-)