

GRAU D'ENGINYERIA INFORMÀTICA

PROGRAMACIÓ II

Bloc 2:

Programació Orientada a Objectes (4)

Laura Igual

Departament de Matemàtiques i Informàtica

Facultat de Matemàtiques i Informàtica

Universitat de Barcelona



UNIVERSITAT DE
BARCELONA

Index

- Lligadures
- Array Polimòrfic
- Més sobre us de les classes

LLIGADURES

Tipus de lligadures: estàtic i dinàmic

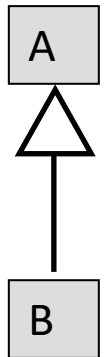
- Donada una assignació polimorfa

- Exemple:

A a;

a = new B();

- Es a dir, una variable de la classe A és una referència a un objecte de la classe B.
- Llavors, es diu que:
 - A és el **tipus estàtic** de la variable **a** i
 - B es el **tipus dinàmic** de **a**.
- El tipus estàtic sempre es determina en temps de compilació i és fix, mentre que el tipus dinàmic només es pot conèixer en temps d'execució i pot variar.



Tipus de lligadures: estàtic i dinàmic

- Java només permet invocar els mètodes i accedir a les variables conegudes per al **tipus estàtic** de a.

```
A a = new B();
```

```
a.metodeA(); // Ok
```

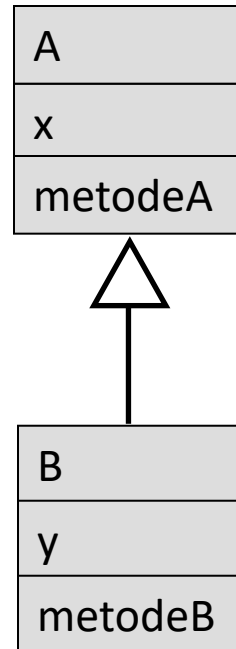
```
a.metodeB(); // error de compilació
```

// metodeB no està definit per a A

accés:

```
a.x; // Ok
```

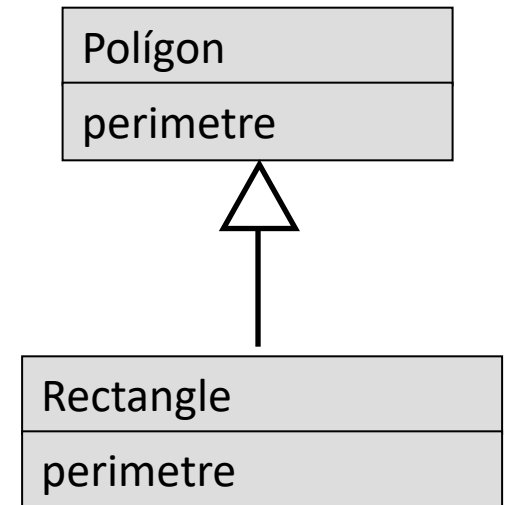
```
a.y; // error de compilació
```



Lligadura dinàmica

En POO, què passa quan realitzem una connexió polimorfa i cridem a una operació redefinida?

```
// La variable "poligon" pot referenciar a un
objecte Polígon o Rectangle
Poligon poligon;
float peri;
Rectangle rectangle = new Rectangle();
...
poligon = rectangle;
...
peri = poligon.perimetre();
```



El compilador no té informació per a resoldre la crida.

Per defecte utilitzaria el tipus de la referència, i per tant generaria una crida a `Poligon.perimetre()`

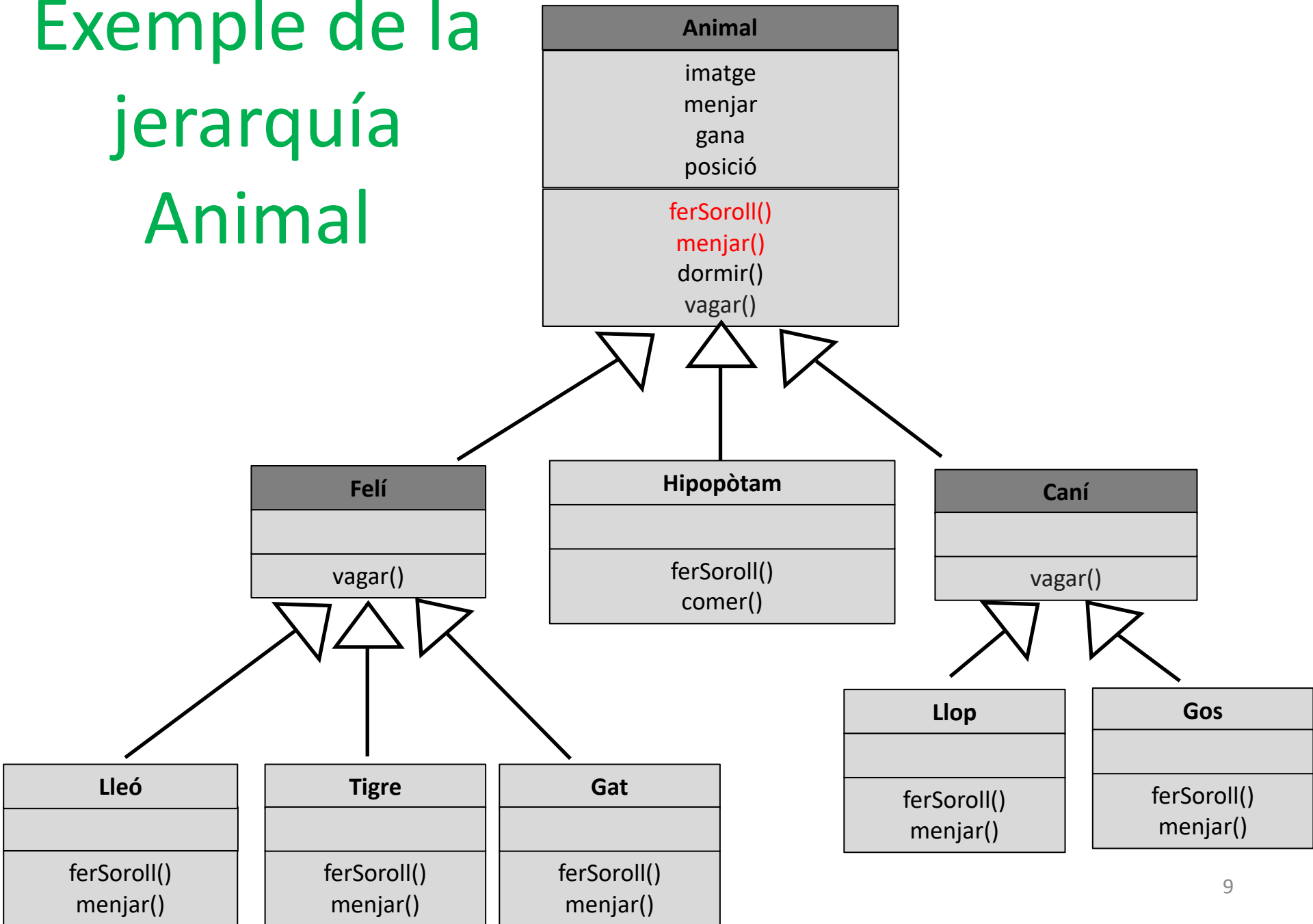
Però la referència `poligon` pot apuntar a un objecte de la classe `Rectangle` amb una versió diferent del mètode

Lligadura dinàmica

- La solució consisteix en esperar a **resoldre la crida en temps d'execució**, quan es coneix realment els objectes connectats a la variable **poligon**, i quina és la versió del mètode **perimetre** apropiada.
- Aquest enfocament de resolució de crides s'anomena **lligadura dinàmica**
- Entenem per **resolució d'una crida** el procés pel qual es substituirà una crida a una funció per un salt a la direcció que conté el codi d'aquesta funció.

SOLUCIÓ EXERCICI PENDENT

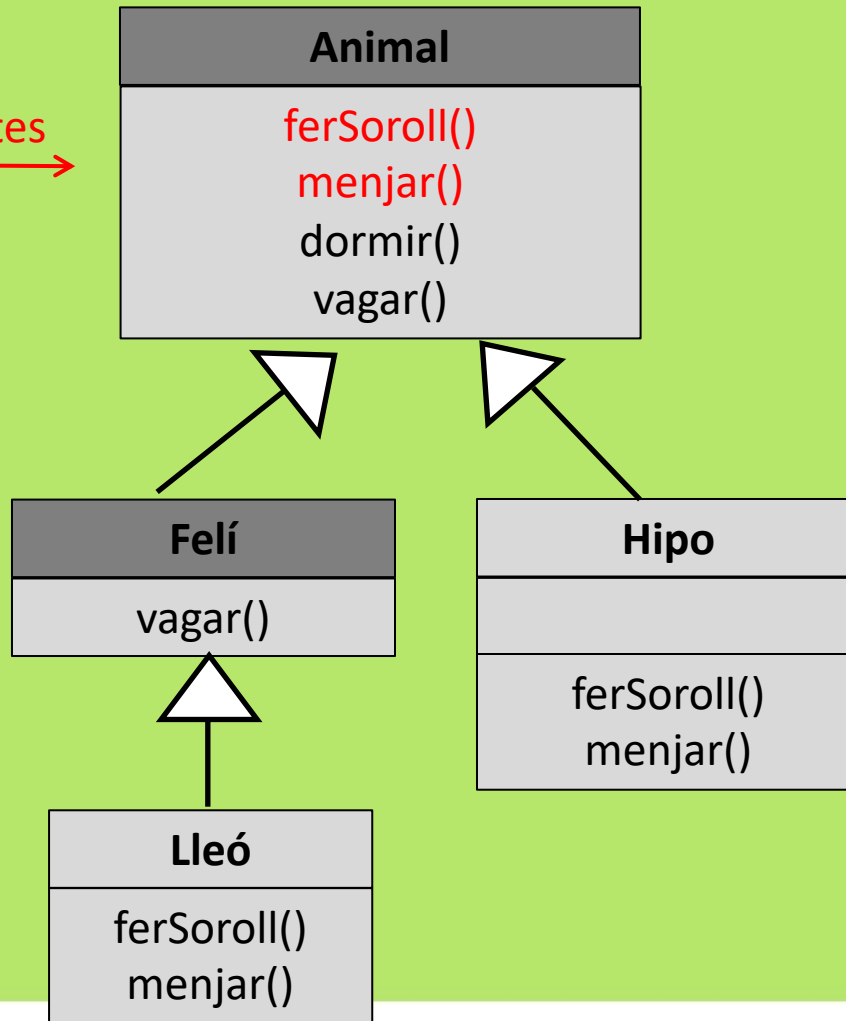
Exemple de la jerarquia Animal



Exercici

Implementa les següents classes:

Mètodes abstractes



Solució Exercici

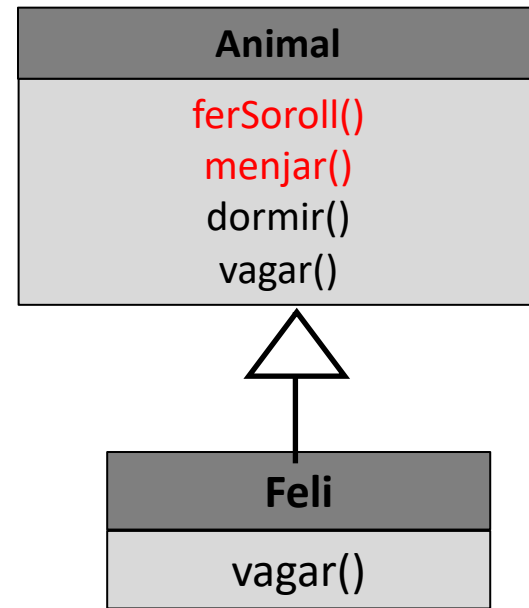
Animal
ferSoroll() menjar() dormir() vagar()

```
public abstract class Animal{  
    public abstract void ferSoroll();  
    public abstract void menjar();  
  
    public void dormir(){  
        System.out.println("zzzz");  
    }  
    public void vagar(){  
        System.out.println("Em moc a vegades");  
    }  
}
```

Mètodes
abstractes

Mètodes
efectius

Solució Exercici



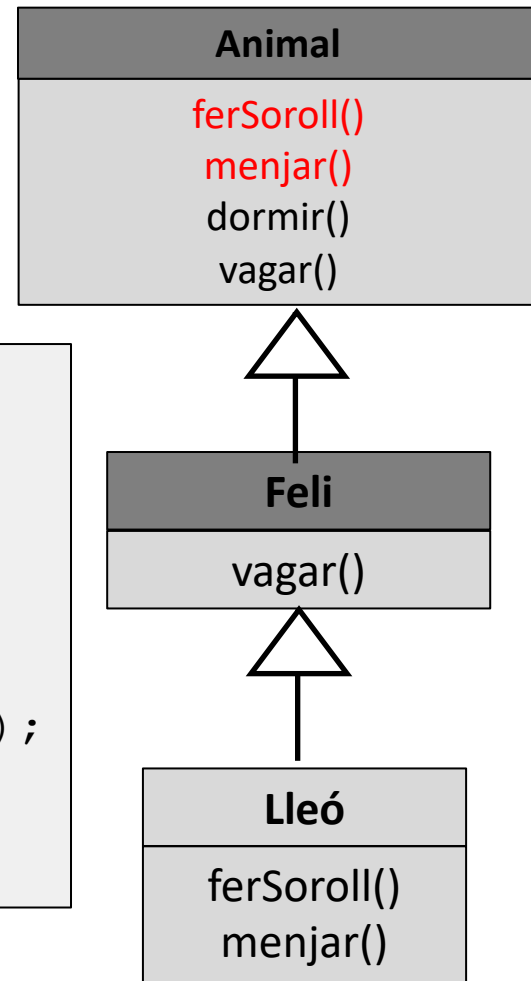
```
public abstract class Feli extends Animal{
    public void vagar(){
        System.out.println("Soc independent");
    }
}
```

Sobreescritura d'un mètode

Solució Exercici

```
public class Lleo extends Feli{  
    public void ferSoroll(){  
        System.out.println("RRRUGIT");  
    }  
    public void menjar(){  
        System.out.println("Menjo carn");  
    }  
}
```

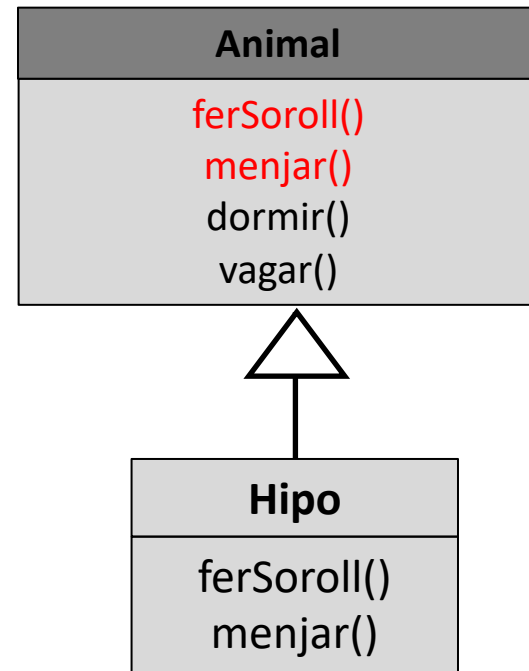
Fer efectius els mètodes abstractes



Solució Exercici

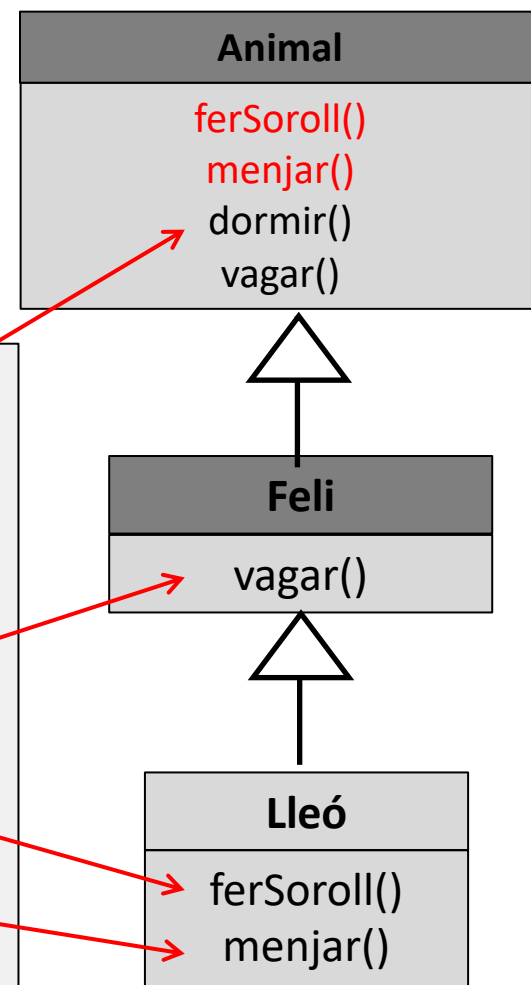
```
public class Hipo extends Animal{  
    public void ferSoroll(){  
        System.out.println("BBBRAM");  
    }  
    public void menjar(){  
        System.out.println("Menjo molt");  
    }  
}
```

Fer efectius els mètodes abstractes



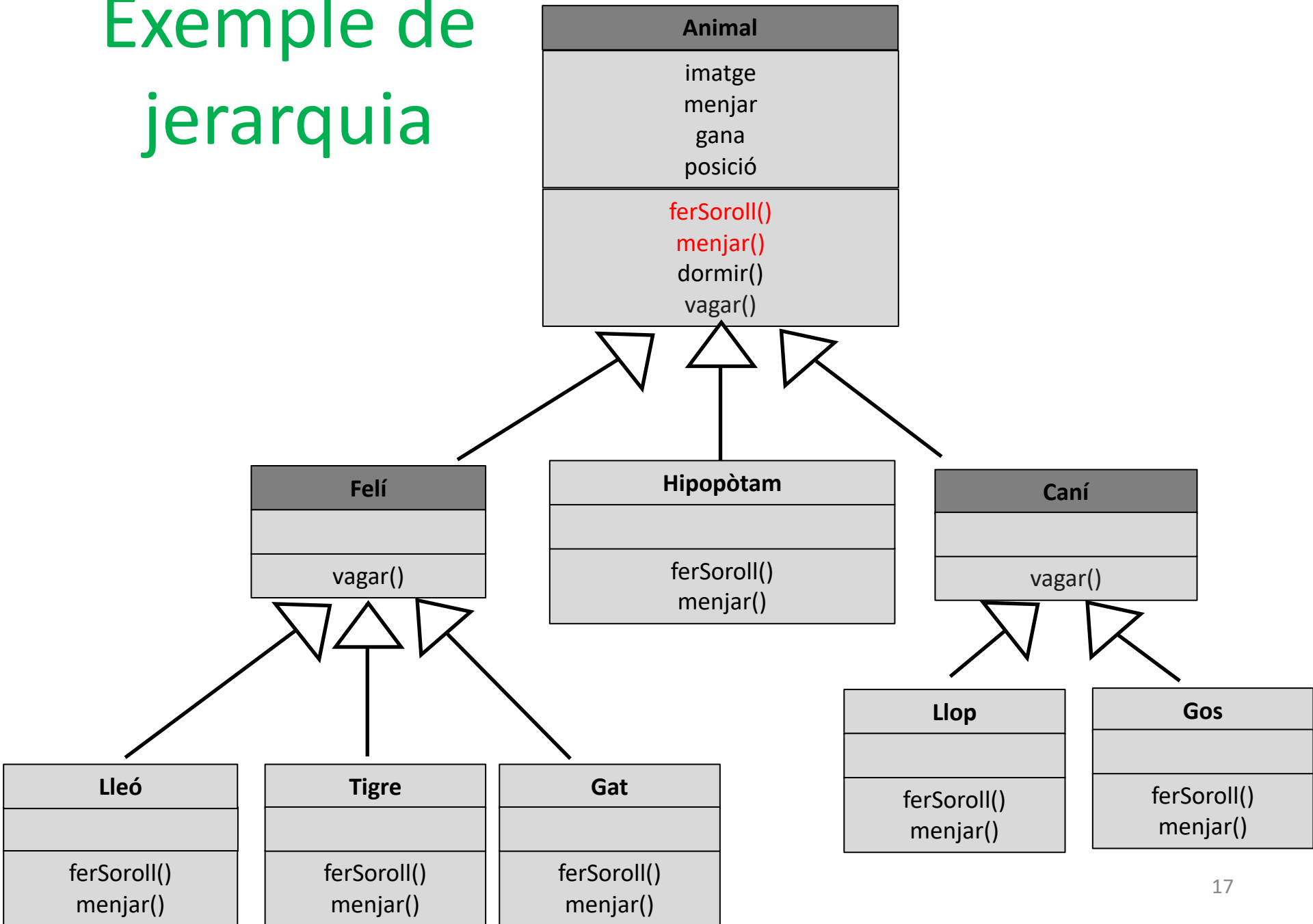
Exercici

```
public class TestAnimals{  
    public static void main(String[] args){  
        Lleo misu = new Lleo();  
  
        misu.ferSoroll();  
        misu.dormir();  
        misu.menjar();  
        misu.vagar();  
    }  
}
```



ARRAY POLIMÒRFIC

Exemple de jerarquia



Array polimòrfic

```
public class LlistaAnimals {  
    private Animal[] animals = new Animal[5];  
    private int nextIndex=0;  
  
    public void add(Animal a){  
        if (nextIndex < animals.length){  
            animals[nextIndex] = a;  
            System.out.println("Animal afegit a la posició " + nextIndex);  
            nextIndex++;  
        }  
    }  
}
```

LlistaAnimals.java

Array polimòrfic

```
public class TestLlistaAnimal {  
    public static void main(String[] args){  
        LlistaAnimals llista = new LlistaAnimals();  
        Gos gos = new Gos();  
        Gat gat = new Gat();  
        llista.add(gos);  
        llista.add(gat);  
    }  
}
```

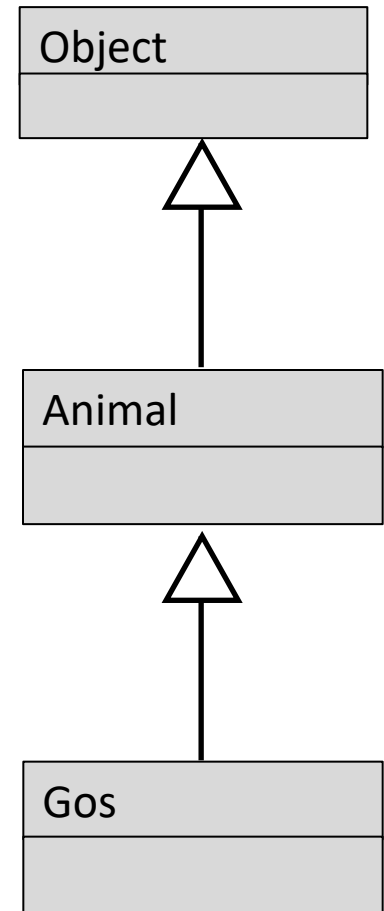
Estem afegint
tot tipus
d'animals a
l'array

TestLlistaAnimals.java

Animal afegit a la posició 0
Animal afegit a la posició 1

Llista polimòrfica

- També es podria optar per fer servir la classe Object que és encara més genèrica i referenciar a qualsevol tipus d'objectes.
- Però això porta alguns inconvenients!!!



Llista polimòrfica

```
ArrayList laLlistaAnimals = new ArrayList();
```

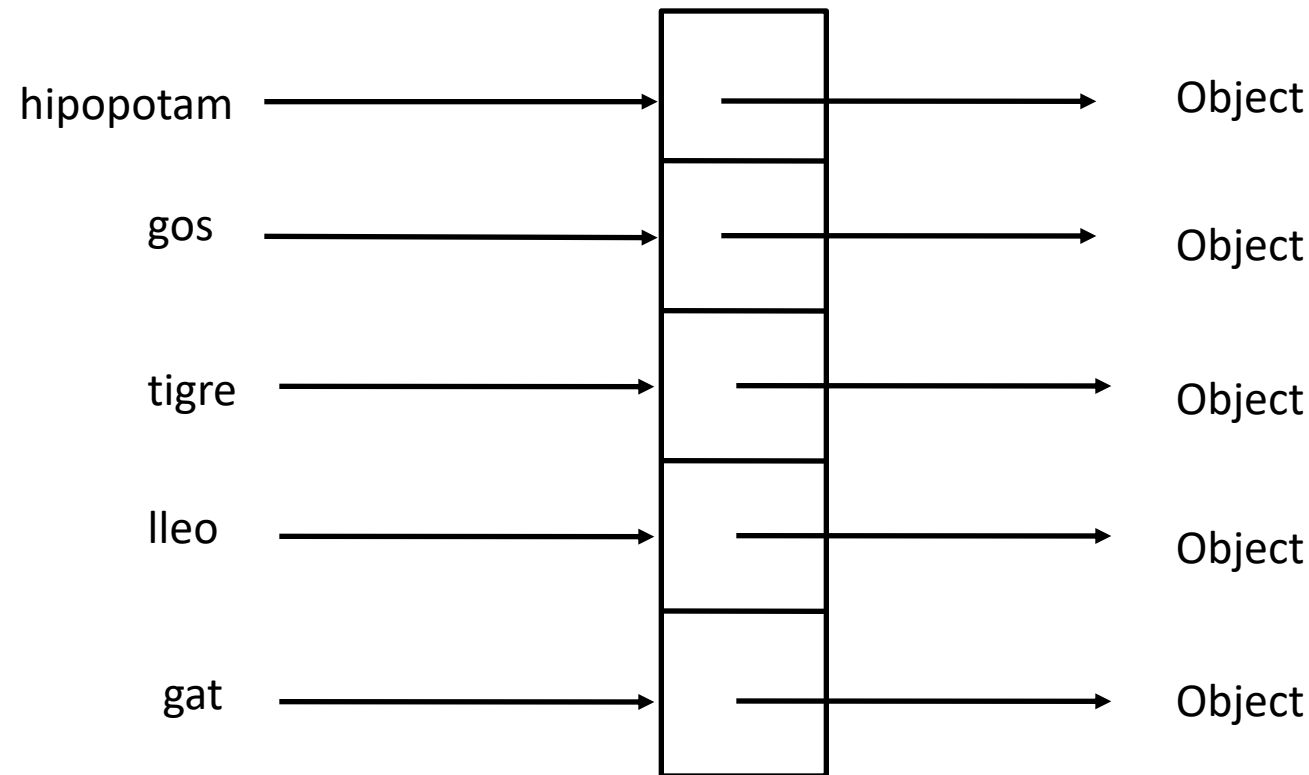
← Llista per contenir tot tipus d'Objectes.

```
Gos gos = laLlistaAnimals.get(0);
```

No compilarà!

```
laLlistaAnimals.get(0).ferSoroll;
```

No compilarà!

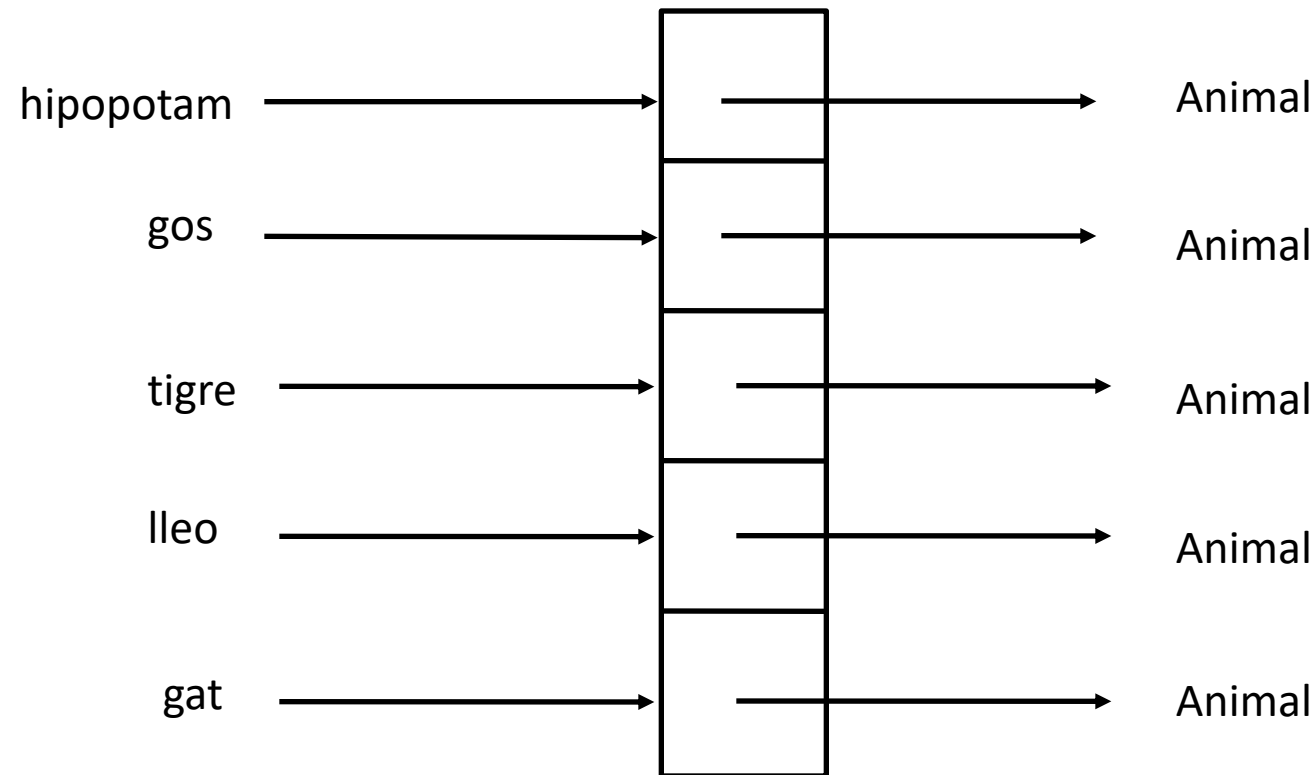


Posis el que posis en cada posició quan recuperis els objectes aquests seran de tipus Object.

Array polimòrfic

```
ArrayList<Animal> laLlistaAnimals = new ArrayList<Animal>();
```

Quan és útil?



Exemple

```
package paquetInterficies;  
import java.util.ArrayList;  
  
public abstract class Animal {  
    public abstract void ferSoroll();  
}
```

Animal.java

Exemple

```
package paquetInterficies;  
import java.util.ArrayList;  
  
public class Gat extends Animal{  
    public void ferSoroll(){  
        System.out.println("miau");  
    }  
}
```

Gat.java

```
package paquetInterficies;  
import java.util.ArrayList;  
  
public class Gos extends Animal{  
    public void ferSoroll(){  
        System.out.println("guau");  
    }  
}
```

Gos.java

Exemple

```
package paquetInterficies;
import java.util.ArrayList;

public class TestAnimals {
    public static void main(String[] args){
        ArrayList<Animal> arrayAnimals = new ArrayList<Animal>();

        Gos gos = new Gos();
        Gat gat = new Gat();
        arrayAnimals.add(gos);
        arrayAnimals.add(gat);
        arrayAnimals.get(0).ferSoroll();
        arrayAnimals.get(1).ferSoroll();
    }
}
```

ferSoroll és un
mètode polimòrfic

Sortida per pantalla:
guau
miau

Informació de classes en temps d'execució

- Després de realitzar una connexió polimorfa és freqüent la **necessitat de tornar a recuperar l'objecte original**, per a accedir a les seves operacions pròpies

- Exemple:**

```
Figura [] figures = new Figura[10];
```

```
...
```

```
Figures[0]= new Rectangle(); Connexions
```

```
Figures[1]= new Cercle();      polimorfes
```

```
....
```

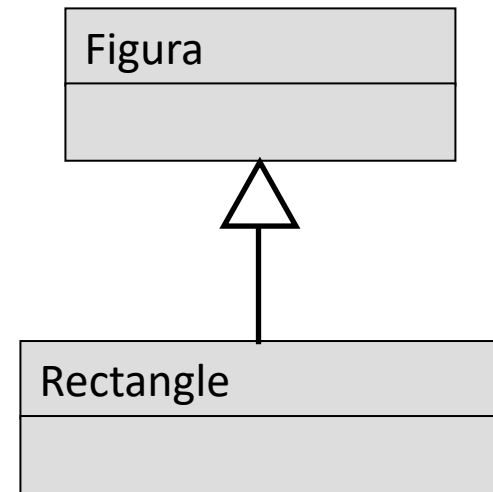
```
Figura fig;
```

```
for (int i=0; i<10; i++) {
```

```
    fig = figures[i];
```

```
}
```

Pot interessar recuperar un Rectangle o Cercle en lloc d'una Figura.



Informació de classes en temps d'execució

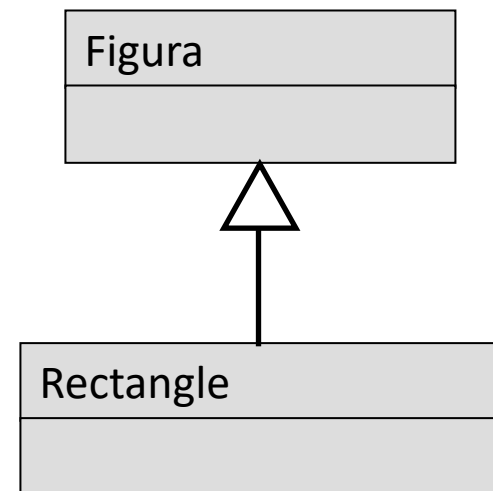
- Es tracta de l'operació inversa al polimorfisme (upcasting), denominada **downcasting**
 - Si el polimorfisme implica una generalització, el downcasting implica una especialització.
- Al contrari que el upcasting, el downcasting no pot realitzar-se directament mitjançant una connexió amb una referència de la classe de l'objecte.
- Recordatori:

Upcasting

```
Figura figura = new Rectangle();
```

Downcasting

```
Rectangle rectangle = new Figura();  
Rectangle rectangle = (Rectangle)figures[i];
```

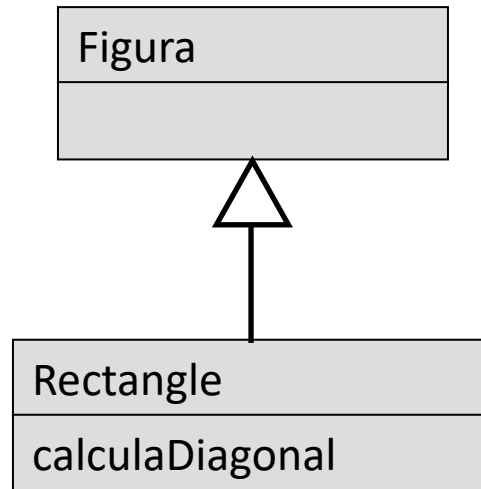


Informació de classes en temps d'execució

- Un casting permet forçar la connexió a la referència
- Un intent de casting impossible generarà una excepció ***ClassCastException*** en temps d'execució
- Possibles accions:
 - Podem capturar aquesta excepció per a determinar si l'objecte apuntat per la referència és del tipus esperat o no, realitzant accions diferents en cada cas *try catch*
 - O, podem utilitzar **instanceof** per determinar si l'objecte és de la classe esperada abans de realitzar el casting.

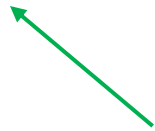
Exemple: diagonal màxima

- El mètode diagonal, és un mètode propi de la subclasse.



Exemple: diagonal màxima

```
Figura [] figures = new Figura[10];  
...  
float actual, maxDiagonal=0;  
for (int i=0; i<10; i++){  
    actual = figures[i].calculaDiagonal();  
    if (actual>maxDiagonal)  
        maxDiagonal=actual;  
}
```



Mètode propi de la
classe Rectangle

Donarà error de compilació!

¿Què passa si no és un rectangle?
Tindríem que preguntar pel tipus

Identificació del tipus en temps d'execució

- `if (figures[i] instanceof Rectangle) ...`
- `java.lang` conté la classe **Class** amb la que es pot:
 - Conèixer el nom de la classe d'un objecte amb el mètode:

String getName()

- Saber si un objecte és instància de la classe o d'una subclasse:

boolean instanceof (Object o)

Exemple:

- `if figures[i].getClass().getName().equals("Rectangle")...`

instanceof vs. equivalencia de Class

- `instanceof`
“Ets d'aquesta classe o d'una classe derivada d'aquesta?”
- `isInstance`

Comparant els objectes `Class`

Exemple: `if(someObject.getClass().isInstance(obj))`

- Exemple: `Rectangle` és una subclasse de la classe `Figura`

```
Rectangle r = new Rectangle();
```

```
(r instanceof Figura) → true
```

```
(r.getClass().equals(Figura.class)) → false
```

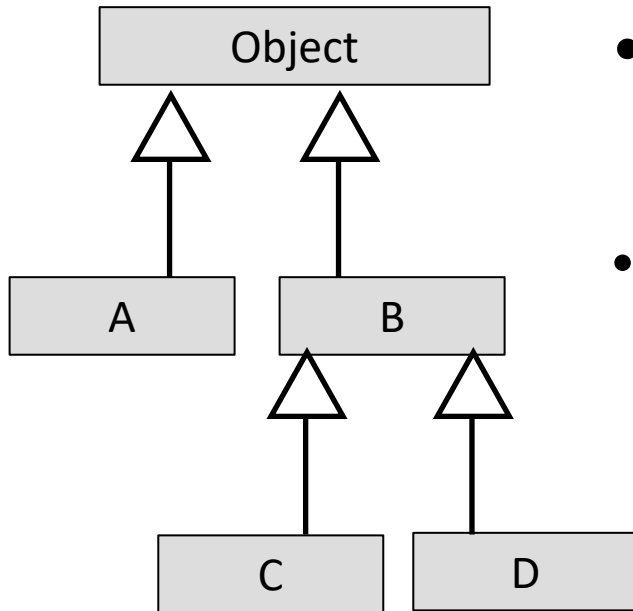

Exemple: diagonal màxima

```
Figura [] figures = new Figura[10];  
...  
float actual, maxDiagonal=0;  
for (int i=0; i<10; i++){  
    if (figures[i] instanceof Rectangle){  
        Rect r = (Rectangle) figures[i];  
        actual = r.calculaDiagonal();  
        if (actual>maxDiagonal)  
            maxDiagonal=actual;  
    }  
}
```

Implementació correcta

MES SOBRE US DE LES CLASSES

Jerarquia de Java



- **Object** és la classe arrel (paquet `java.lang`)
- **Object** descriu les propietats comunes a tots els objectes

Classe Object

- Qualsevol classe implementada per tu hereta de la classe Object.

1. equals(Object o)

```
Dog a = new Dog();
Cat c = new Cat();
if (a.equals(c)) {
    System.out.println("true");
} else {
    System.out.println("false");
}
```

```
% java TestObject
false
```

3. hashCode()

```
Cat c = new Cat();
System.out.println(c.hashCode());
```

```
% java TestObject
8202111
```

2. getClass()

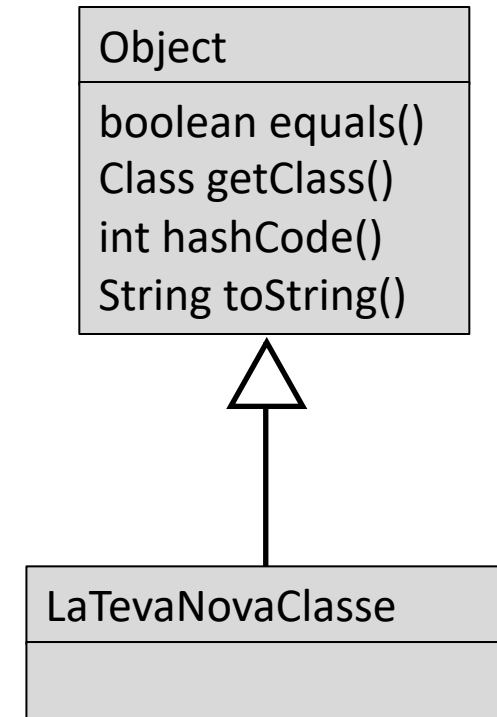
```
Cat c = new Cat();
System.out.println(c.getClass());
```

```
% java TestObject
class Cat
```

4. toString()

```
Cat c = new Cat();
System.out.println(c.toString());
```

```
% java TestObject
Cat@7d277f
```



Nom de la classe + @ +
representació hexadecimal del
hash code de l'objecte.

Ús de classes

Classes Wrapper

- El paquet java.lang conté classes wrapper (envoltori) que es corresponen amb cada tipus primitiu:

<u>Classe Wrapper</u>	<u>Tipus primitiu</u>
Byte	byte
Short	short
Integer	int
Long	long
Float	float
Double	double
Character	char
Boolean	boolean
Void	void

Ús de classes

Classes Wrapper II

- La següent declaració crea un objecte Integer que representa un enter 40 com un objecte

Integer age = new Integer(40) ;

- Un objecte d'una classe *wrapper* pot ser usat en qualsevol situació on un valor primitiu no pot. Per exemple es pot emmagatzemar en un contenidor d'objectes.

Ús de classes

Classes Wrapper III

- Les classe *wrapper* també contenen mètodes estàtics que ajuden a gestionar el tipus associat
- Per exemple, la classe `Integer` conté un mètode per convertir un enter contingut en un `String` al seu valor `int`:

```
int num = Integer.parseInt(str) ;
```

- Les classes wrapper contenen constants molt útils
- La classe `Integer` té `MIN_VALUE` i `MAX_VALUE` que contenen el major i menor nombre que es pot guardar en un `int`

Exemple

- Exemple fent servir la classe String: Com puc extreure l'extensió del nom d'un fitxer?

```
public String getExtensio() {  
    int pos;  
    String ext = null;  
    String nom = getName();  
    pos = nom.lastIndexOf('.');  
    if(pos>0) {  
        ext = nom.substring(pos+1);  
    }  
  
    return ext;  
}
```


Exemple

- Exemple fent servir la classe Float i String:

```
String line = sc.nextLine();
```

```
float durada = !line.isEmpty() ? Float.valueOf(line) : -1;
```

String

```
public boolean isEmpty()
```

Returns true if, and only if, length() is 0.

Float

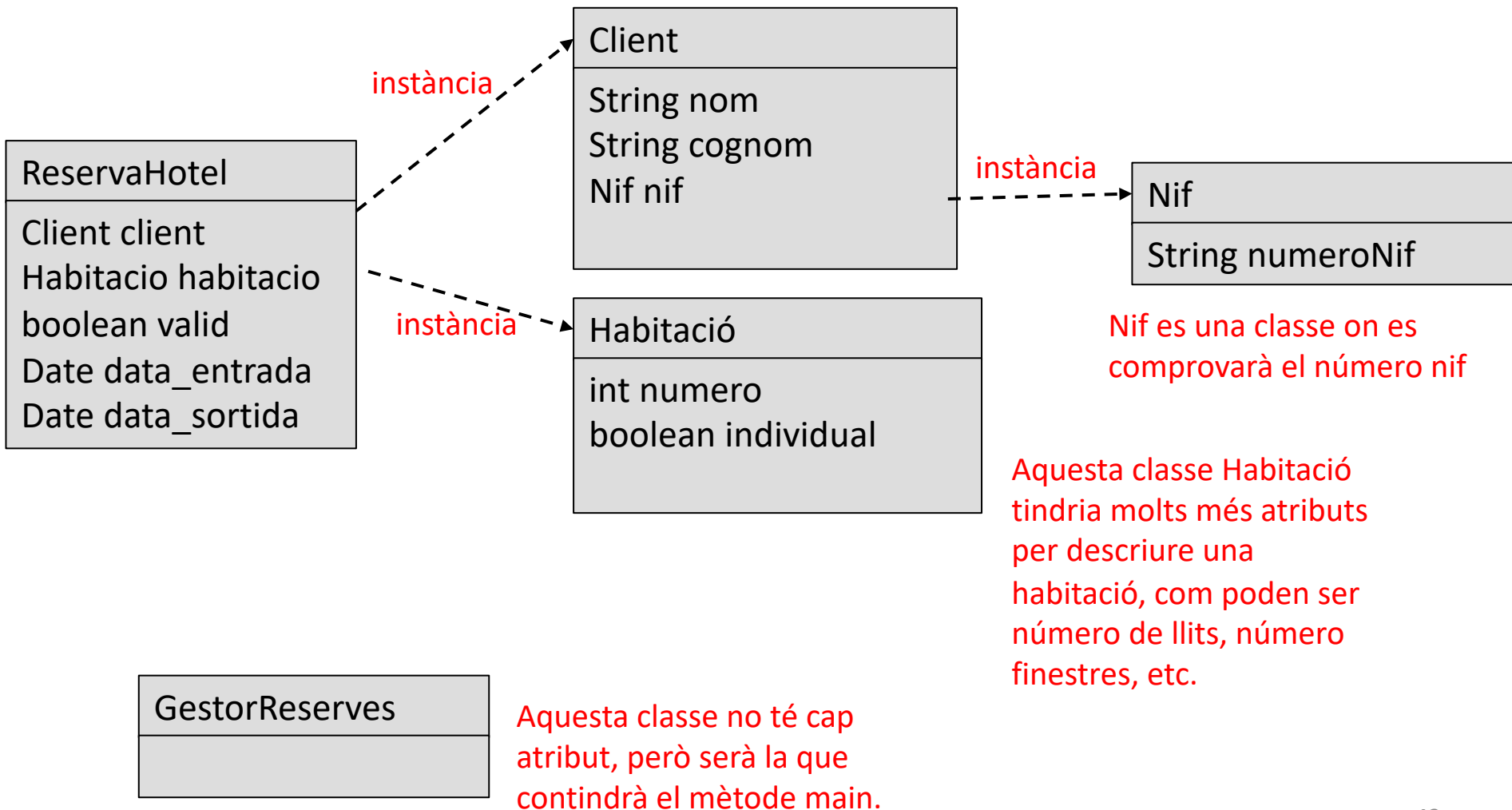
```
public static Float valueOf(String s) throws NumberFormatException
```

Returns a Float object holding the float value represented by the argument string s.

Exercici 1 per fer

- Es vol implementar una aplicació de gestió de reserves d'hotel seguint el diagrama de classes definit a continuació. On apareixen el nom de les classes i la llista dels seus atributs.
- Implementa les classes:
 - **ReservaHotel**
 - **Habitacio**
 - **Client**
 - **GestorReserves**
- Al mètode main de la classe **GestorReserves** s'ha de crear una reserva i validar-la.

Exercici 1: Diagrama de classes



Exercici 2 per fer

Donada la classe **Cotxe**, implementeu les següents classes:

1. **CotxeAmbGPS** que hereta de **Cotxe** i tindrà dos atributs nous de tipus double: latitud i longitud i un nou mètode **canviarCoordenades** que rebrà dos doubles com a paràmetres.
2. **Taxi** que hereta de **Cotxe** i redefinirà el mètode **recorrer** de manera que imprimeixi per pantalla dos missatges, abans (“iniciar carrera”) i després (“fin de carrera”) de fer el recorregut.

Exercici 2 per fer

3. **FactoriaDeCotxes** que tindrà un mètode `fabricarCotxeNou` que retorni un cotxe de nova creació que algunes vegades serà un **Cotxe** un **CotxeAmbGPS** o un **Taxi**.

```

package ub.estudiant;
public class Cotxe{
    private String propietari;
    private string matricula;
    private double compteKilometres;

    public void vendre(String elPropietari) {
        propietari = elPropietari;
    }

    public void matricular(string laMatricula) {
        matricula = laMatricula;
    }

    pulic void recorrer(double kms){
        compteKilometres = compteKilometres + kms;
    }
    public void printInfo(){
        String tmp =
        " Propietari : " + propietari + "; " +
        " Matricula : " + matricula + "; " +
        "Kms recorridos : " + contaKilometres + ";" ;
        System.out.println(tmp) ;
    }
}

```

Referències

- Bertrand Meyer, “**Construcción de software orientado a objetos**”, Prentice Hall, 1998.
- Bert Bates, Kathy Sierra. **Head First Java**. O’Reilly Media, 2005.