GRAU D'ENGINYERIA INFORMÀTICA

PROGRAMACIÓ II

Bloc 2:

Programació Orientada a Objectes (3)

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

...herència: algunes consideracions més

- Totes les classes de Java hereten d'Object

- Els mètodes *static* també s'hereten (però no es poden sobreescriure)

...herència: conversió de tipus

- *Casting* de tipus primitius (vist a Programació I)
- *Casting* d'instàncies (relacionat amb Herència)

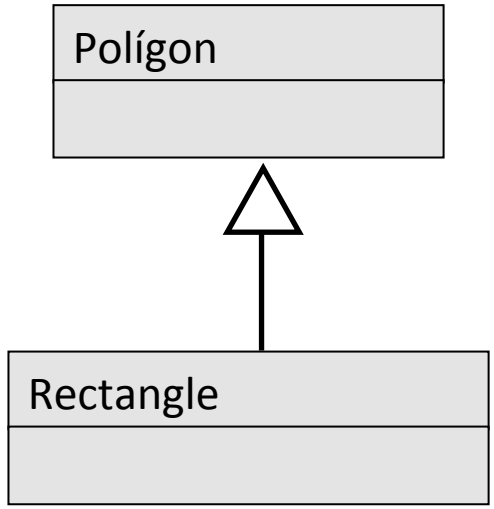
El càsting (o conversió de tipus) ens permet utilitzar una instància d'una classe com si es tractés d'una instància d'un altre tipus.

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

- **Conversió implícita:** (automàtica)
  - *Referències*: tot objecte conté una instància de les seves superclasses
    - ***cast-up***
    - sempre vàlid

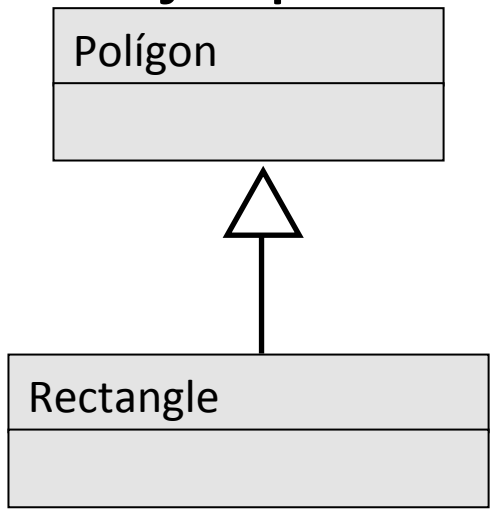
```
Poligon poligon;
Rectangle rectangle = new Rectangle();
poligon = rectangle;
```



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

- **Conversió implícita:** (automàtica)
- Poligon p = new Rectangle() funciona atès que un Rectangle és un polígon
- Rectangle r = new Poligon() no funciona ja que un polígon pot no ser un rectangle



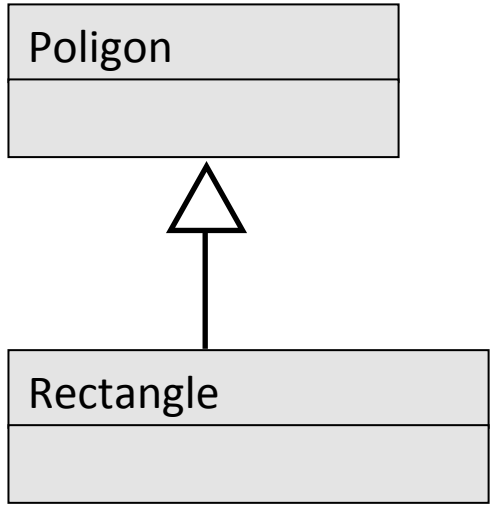
Conversió de tipus

- **<u>Conversió explícita:</u>**
  - *Referències:* assignar a un objecte d'una subclasse en 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];
 poligons[0] = new Rectangle();
 poligons[1] = new Triangle();
…
if (poligons[0] instanceof Rectangle)
     Rectangle r = (Rectangle) poligons[0];
```



- Donaria error en compilació:

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

Bloc 2:
Programació orientada a objectes

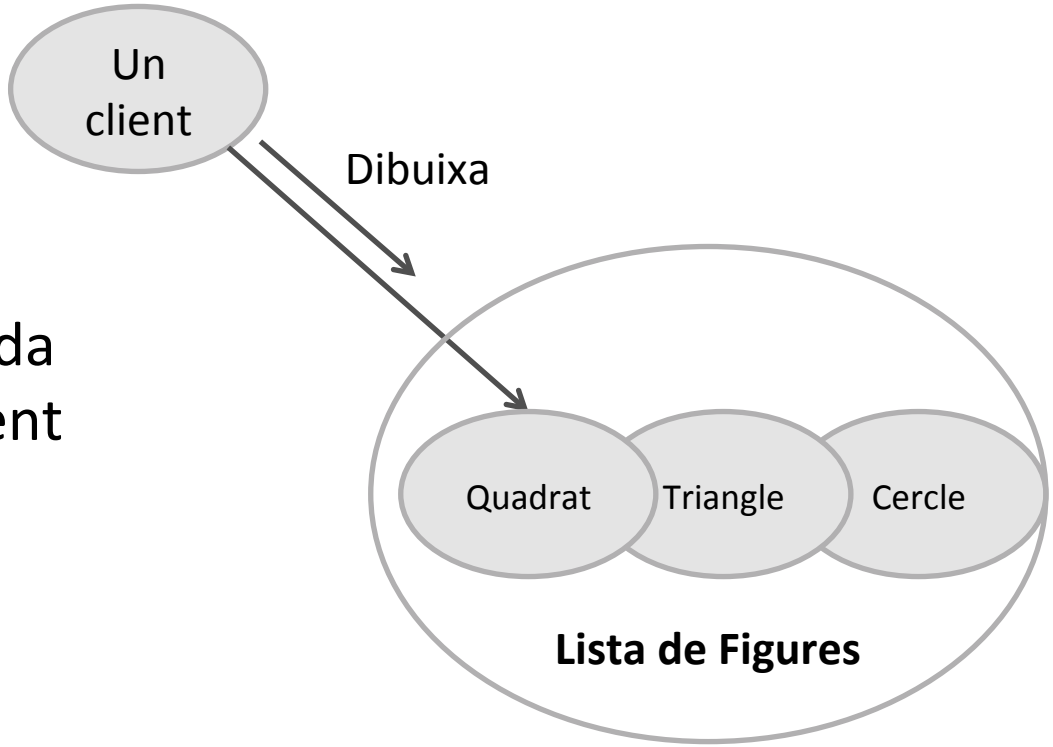
- Abstracció en el desenvolupament del *software*
- Conceptes fonamentals: classes i objectes
- Característiques de l'orientació a objectes
- Ús de classes i objectes
- Constructors i destructors
- Encapsulació
- Herència i jerarquia de classes
- **Polimorfisme**
- Lligadures
- Interfícies
- col·leccions

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.
- La reescriptura 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.

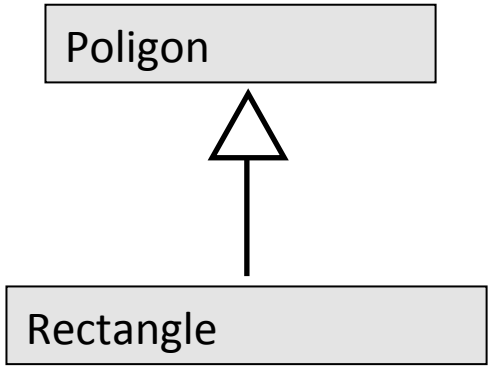
Polimorfisme

- Assignació polimorfa:

  Dues implementacions:

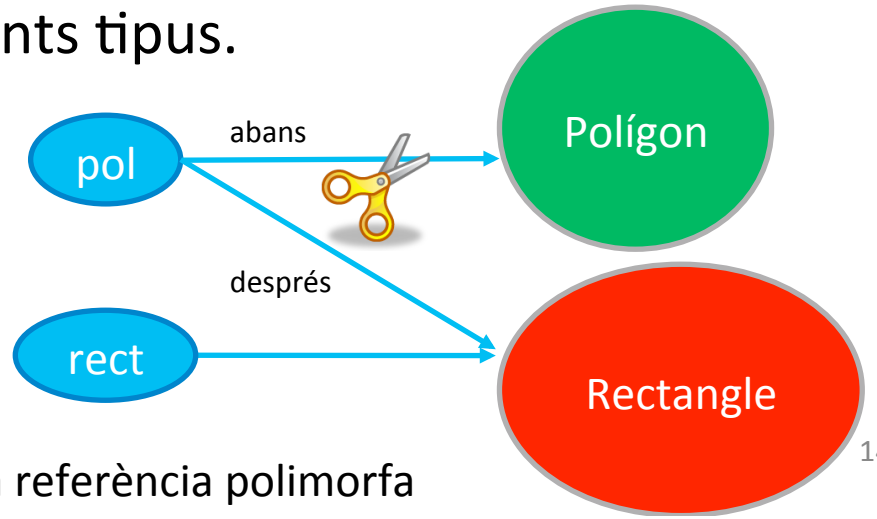
```
Poligon pol = new Poligon();
Rectangle rect = new Rectangle();
pol = rect;
```

```
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

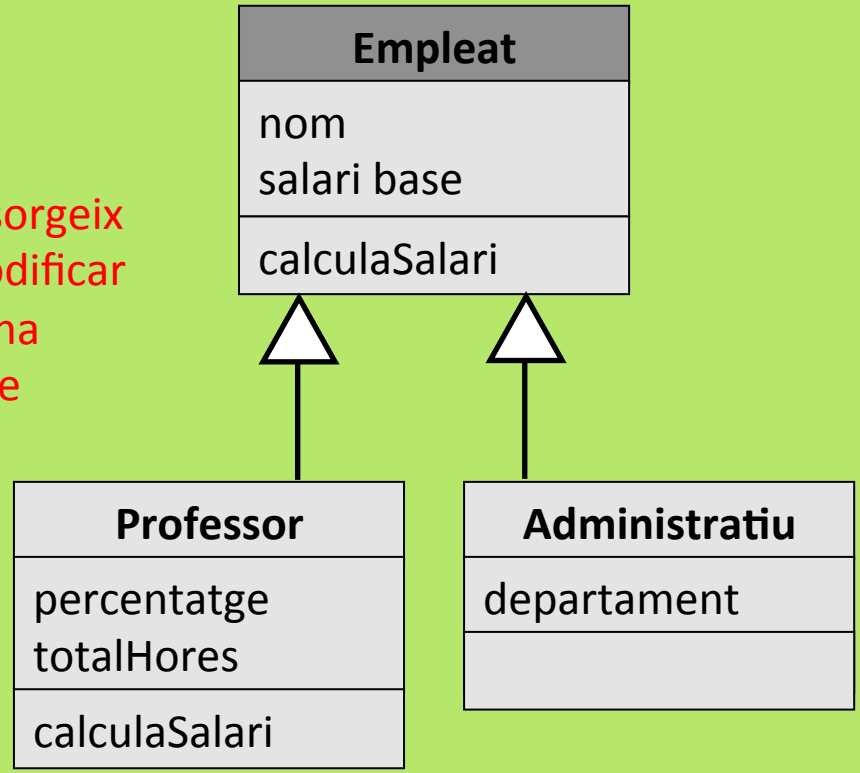
Polimorfisme

- 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.

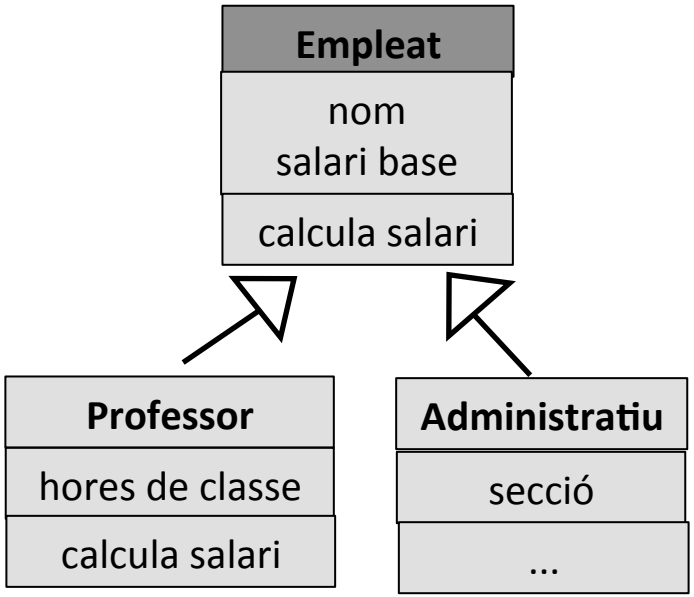
Exercici 1

- Implementació de l'exemple:

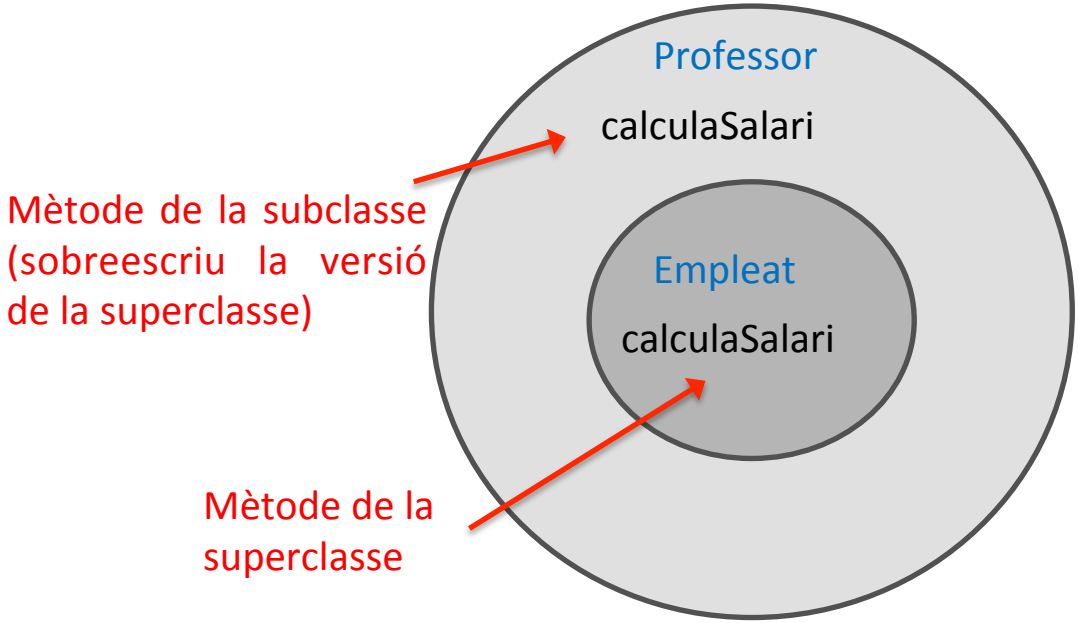
Aquí el polimorfisme sorgeix de la necessitat de modificar el comportament d'una operació per a la classe Professor



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

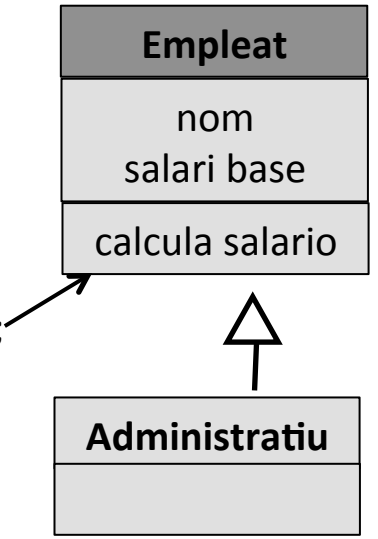
```
public class Professor extends Empleat {
        private float percentatge;
        private float totalHores;
        // constructor
        public Professor(String nom, float salariBase, float percentatge, float totalHores) {
                super(nom, salariBase);
                this.percentatge = 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

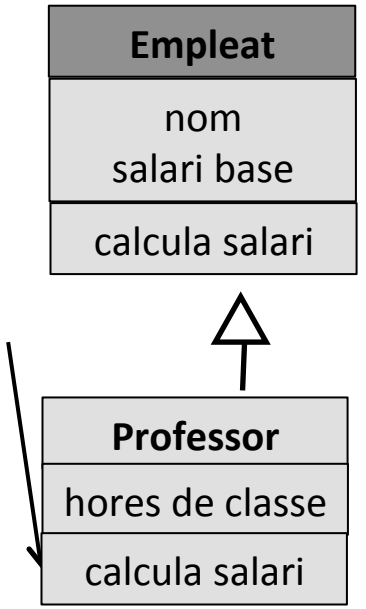
Una referència a un objecte de la subclasse Administratiu cridarà al mètode heretat.

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());

  }
}
```



→ 2.000.0

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

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.

La sobrecàrrega succeix sempre dins d'una sola classe

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

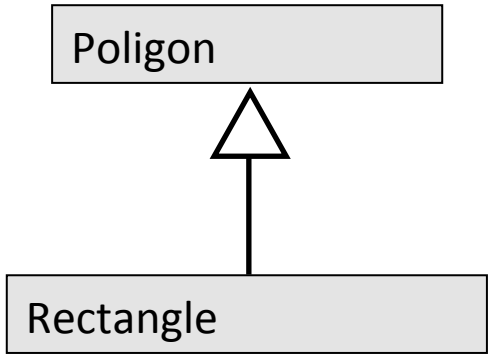
El polimorfisme es resol en temps d'execució.

El polimorfisme té sentit entre diferents classes

Exemple 2

```
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
```

Exemple 3

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

```
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);
    }
}
```

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

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

Exemple 3

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

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

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

Què surt per pantalla?

→ mètode sobreescrit

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

mètode original

mètode sobreescrit

Exemple 3

- 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");
	}
}
```

Algunes preguntes...

- Els mètodes privats es poden sobreescriure? Raona la teva resposta
  - No es poden sobreescriure. Els mètodes privats són visibles a nivell de la classe que els defineix

- Els mètodes static es poden sobreescriure? Raona la teva resposta
  - No es poden sobreescriure. Un mètode static és un mètode de classe, no d'objecte / instància.

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.

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

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.

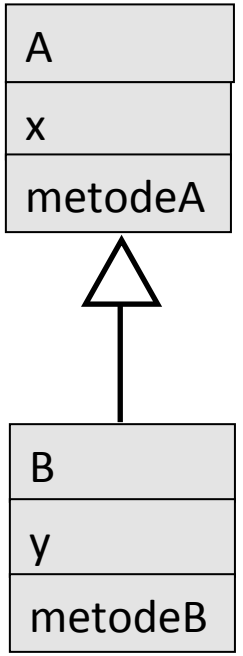
Tipus de lligadures: estàtic i dinàmic

- Java només permet invocar els mètodes i accedir a les variables conegudes pel **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
```

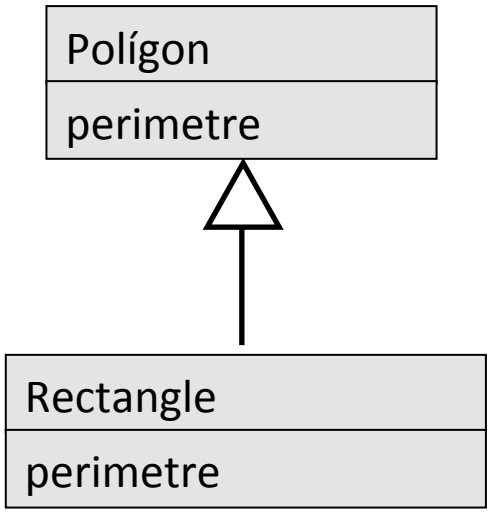


Lligadura dinàmica

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

```
// 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 **polígon**, i quina és la versió del mètode **perímetre** 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

Exemple

```
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();
   }
}
```

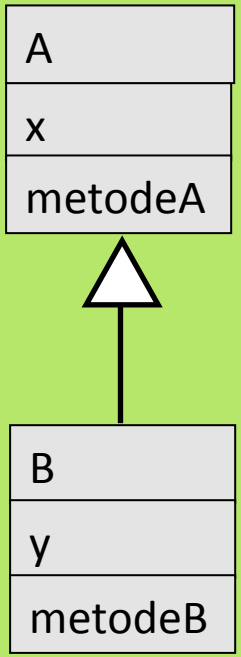
Sortida per pantalla →

```
Sóc Poligon
Sóc Rectangle
```

Exercici 1

- 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

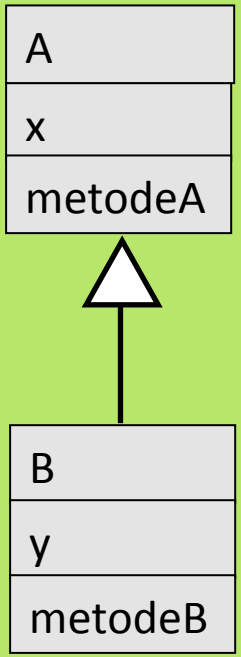
```
public class A{
        protected int x;
        public void metodeA(){
                ....
        }
}
public class B extends A{
        private int y;
        public void metodeB(){
                ....
        }
}
```



Exercici 1

Indica al costat de cadascuna de les línies del següent codi si hi haurà errors de compilació o no i explica breument perquè:

```
0 public static void main(String[] args) {
1           A var1 = new B();
2           B var2 = new B();
3           A var3;
4           var3 = var2;
5           int j = var2.x;
6           int i = var1.x;
7            int k = var1.y;
8           var1.metodeA();
11           var1.metodeB();
12          var2.metodeA();
11          var2.metodeB();
12 }
```



Exercici: Solució

Indica al costat de cadascuna de les línies del següent codi si hi haurà errors de compilació o no i explica breument perquè:

```
0 public static void main(String[] args) {
1           A var1 = new B();   OK
2           B var2 = new B(); OK
3           A var3;             OK
4           var3 = var2;        OK
5           int j = var2.x;     OK
6           int i = var1.x;     OK
7            int k = var1.y;      Error de compilació, l'atribut y no està defint per a A.
8           var1.metodeA();    OK
11         var1.metodeB();    Error de compilació, metodeB no està definit per a A
12          var2.metodeA();    OK
11          var2.metodeB();    OK
12 }
```

Exercici 2

Preguntes sobre el codi de la transparència següent.

1. Especifica si hi ha alguna **conversió** de tipus **implícita** en el codi anterior i en cas afirmatiu en quines línies.

2. Si afegim un nou mètode a la classe A anomenat imprimir que imprimeix el missatge "Missatge d'A", però no el sobreescriu a la classe B, que passa quan fem una crida d'aquesta forma:

   b.imprimir();

3. Indica com has de sobreescriure el mètode imprimir a la classe B de manera que quan fas la crida

   b.imprimir();

   La sortida sigui: "Missatge de B"

4. Ara, indica com has de sobreescriure el mètode imprimir a la classe B de manera que quan fas la crida

   b.imprimir();

   La sortida sigui: "Missatge d'A"

   "Missatge de B"

Exercici 2

```
0 public static void main(String[] args) {
1            A var1 = new B();
2            B var2 = new B();
3            A var3;
4            var3 = var2;
5            int j = var2.x;
6            int i = var1.x;
7            int k = var1.y;
8            var1.metodeA();
11          var1.metodeB();
12           var2.metodeA();
11           var2.metodeB();
12 }
```

Exercici: solució

1. Especifica si hi ha alguna **conversió** de tipus **implícita** en el codi anterior i en cas afirmatiu en quines línies.

Solució:

Sí, a la línia 1 i 4

2. Si afegim un nou mètode a la classe A anomenat imprimir que imprimeix el missatge "Missatge d'A", però no el sobreescriu a la classe B, que passa quan fem una crida d'aquesta forma:

```
b.imprimir();
```

Solució:

Apareixerà el missatge: "Missatge d'A"

Exercici: solució

3. Indica com has de sobreescriure el mètode imprimir a la classe B de manera que quan fas la crida

b.imprimir();

La sortida sigui: “Missatge de B”

Solució:

```
public void imprimir(){
        System.out.println(“Missatge de B”);
}
```

4. Ara, indica com has de sobreescriure el mètode imprimir a la classe B de manera que quan fas la crida

b.imprimir();

La sortida sigui: “Missatge d’A”

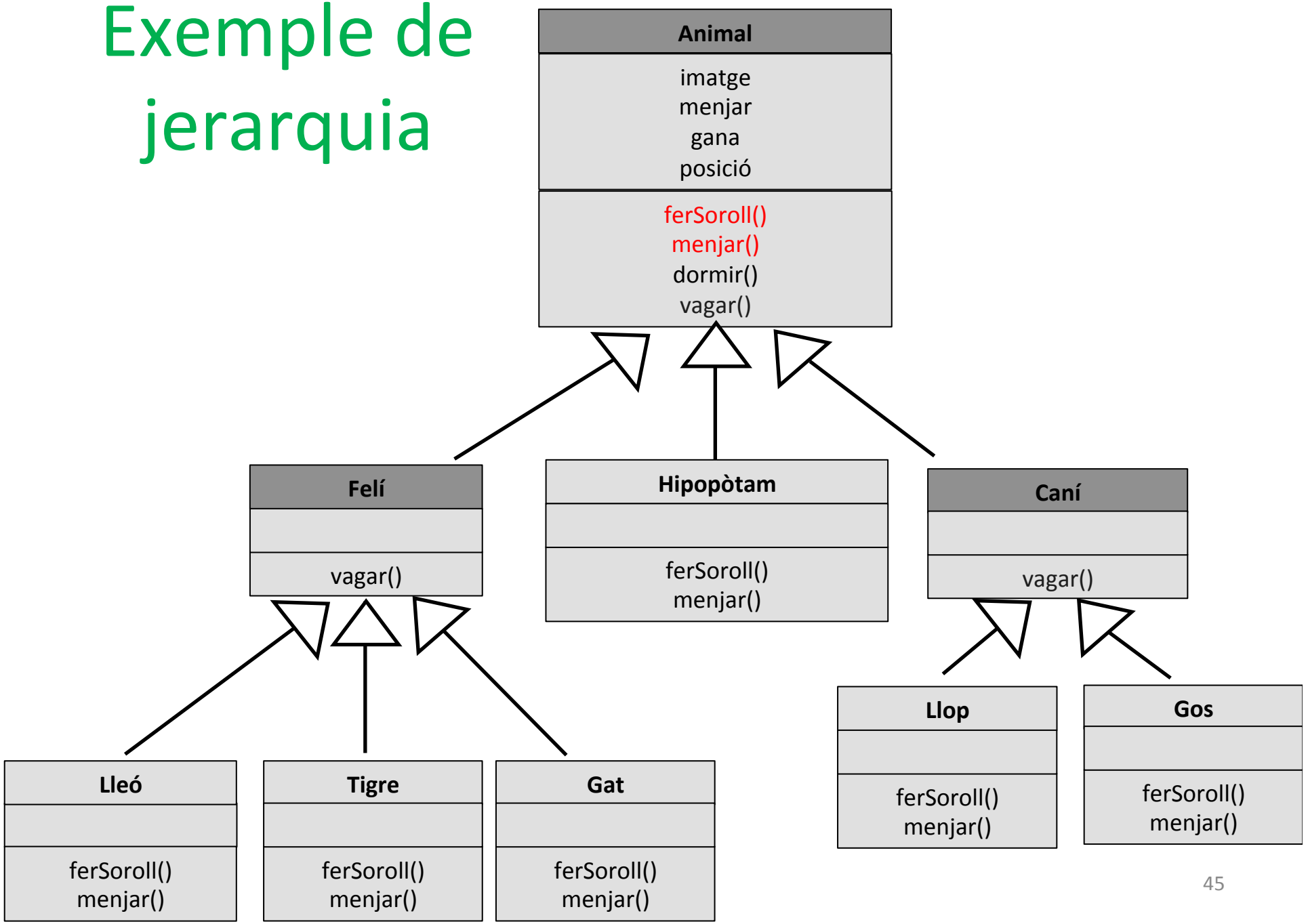
“Missatge de B”

Solució:

```
public void  imprimir(){
        super.imprimir();
        System.out.println(“Missatge de B”);
}
```

REPÀS 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);
    }
}
```

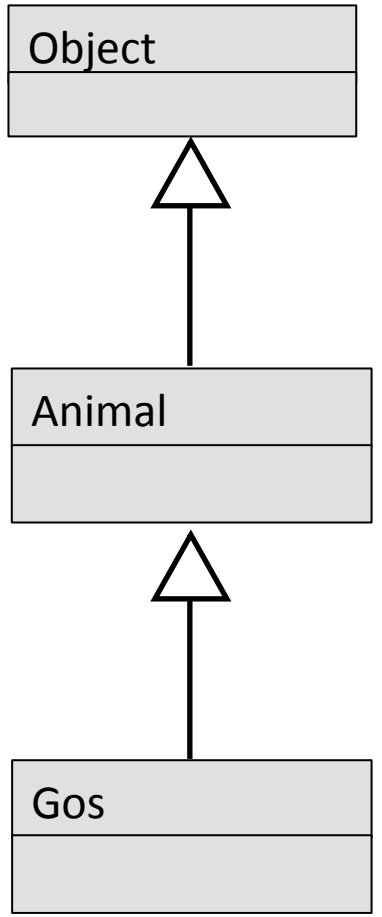
TestLlistaAnimals.java

Estem afegint tot tipus d'animals a l'array

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();
```

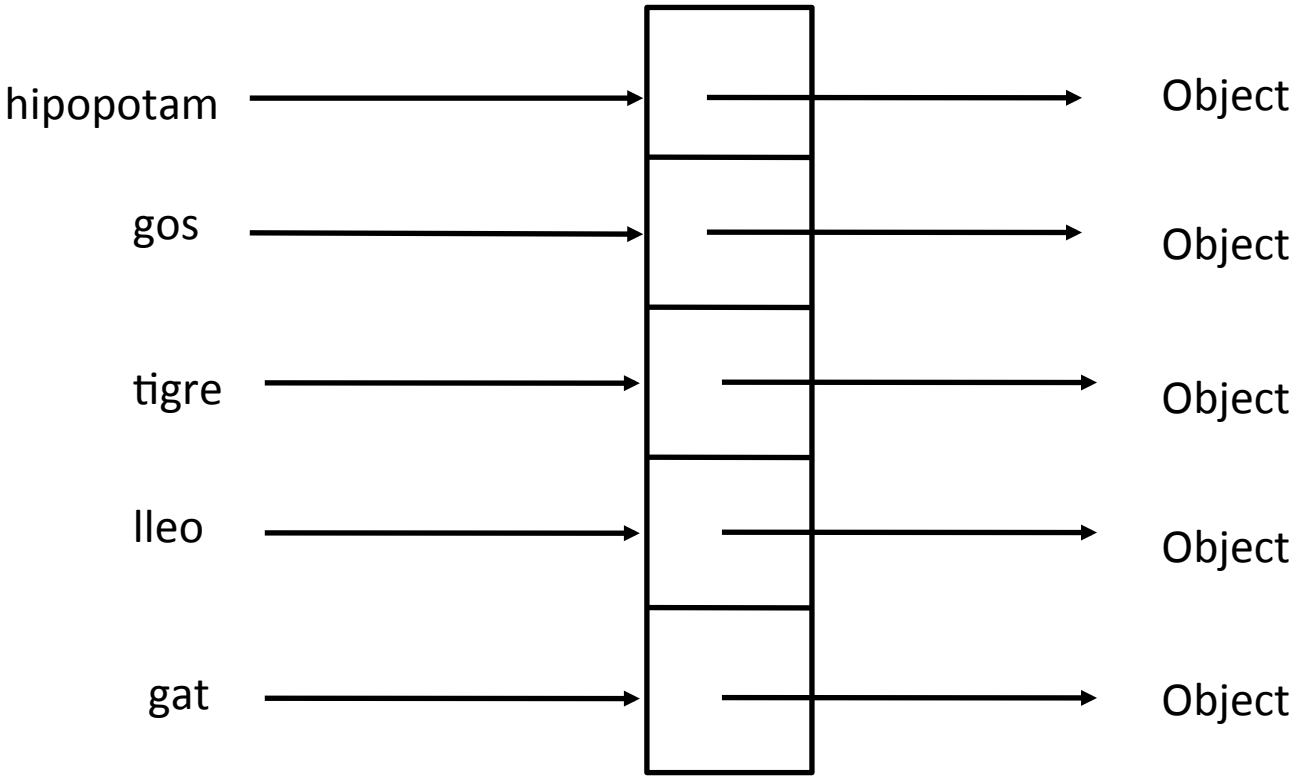
Llista per contenir tot tipus d'Objectes.

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

No compilarà!

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

No compilarà!

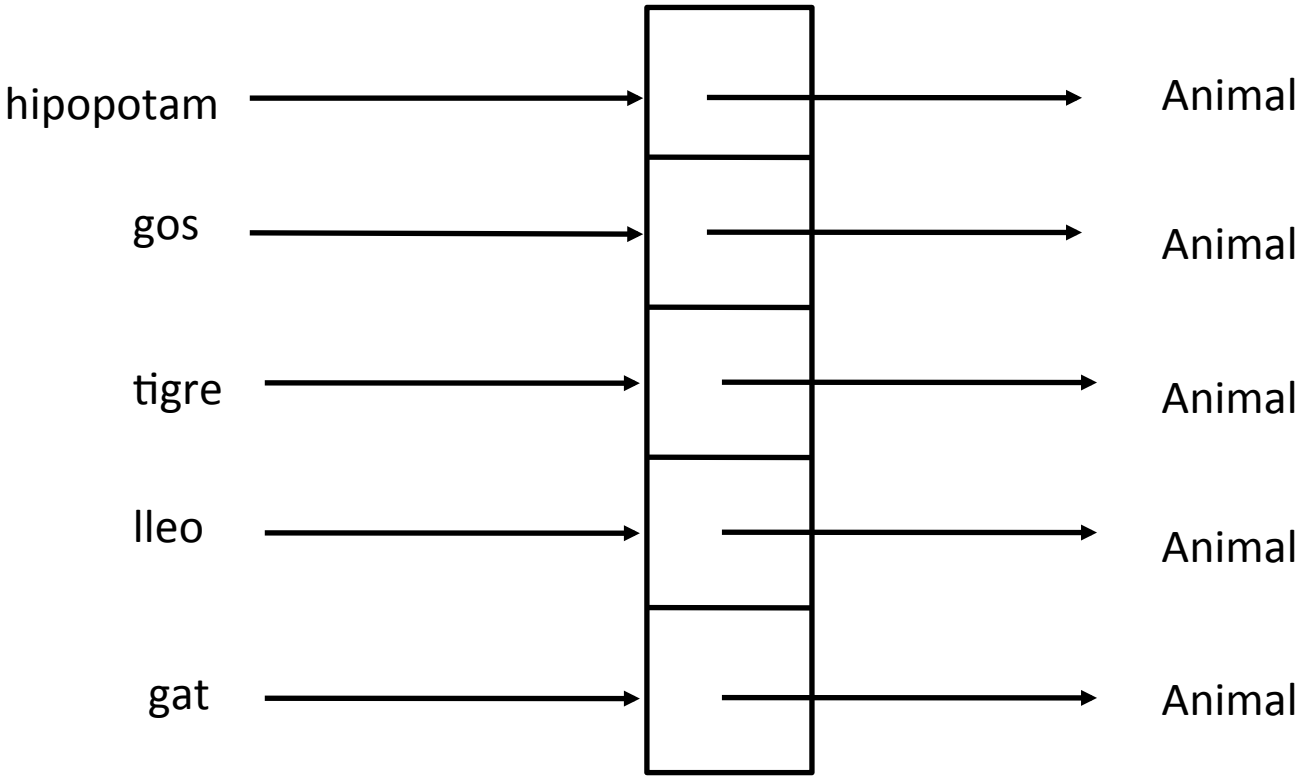


Posis el que posis en cada posició quan recuperis els objectes aquests seran de tipus Object.

Array polimòrfic

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

Quan pot ser útil?



Exemple

```
package paquetTema23;
import java.util.ArrayList;

public abstract class Animal {
    public abstract void ferSoroll();
}
```

Animal.java

Exemple

```
package paquetTema23;
import java.util.ArrayList;

public class Gat extends Animal{
 public void ferSoroll(){
       System.out.println("miau");
   }
}
```

Gat.java

```
package paquetTema23;
import java.util.ArrayList;

public class Gos extends Animal{
   public void ferSoroll(){
      System.out.println("guau");
   }
}
```

Gos.java

Exemple

```
package paquetTema23;
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

TestAnimal.java

Referències

- Bertrand Meyer, "**Construcción de software orientado a objetos", Prentice Hall, 1998.**

- Bert Bates, Kathy Sierra. **Head First Java**. O'Reilly Media, 2005.