# UD6.- Programació orientada a objectes I

# CONTINGUTS

- Comportament dels objectes
- Crear una classe
- Atributs i mètodes d'una classe
- Crear instàncies d'una classe
- Els constructors
- Els destructors
- La referència *this*
- Membres (atributs i mètodes) genèrics *(static)*

# Programació estructurada vs. POO

- **Programació estructurada**
  - Dades
  - Programes i subprogrames (funcions i procediments)

- **Programació Orientada a Objectes**
  - Classes/objectes
    - Agrupació de **dades** (atributs o propietats) i **mètodes** (funcions i procediments).

# Què és una classe?(I)

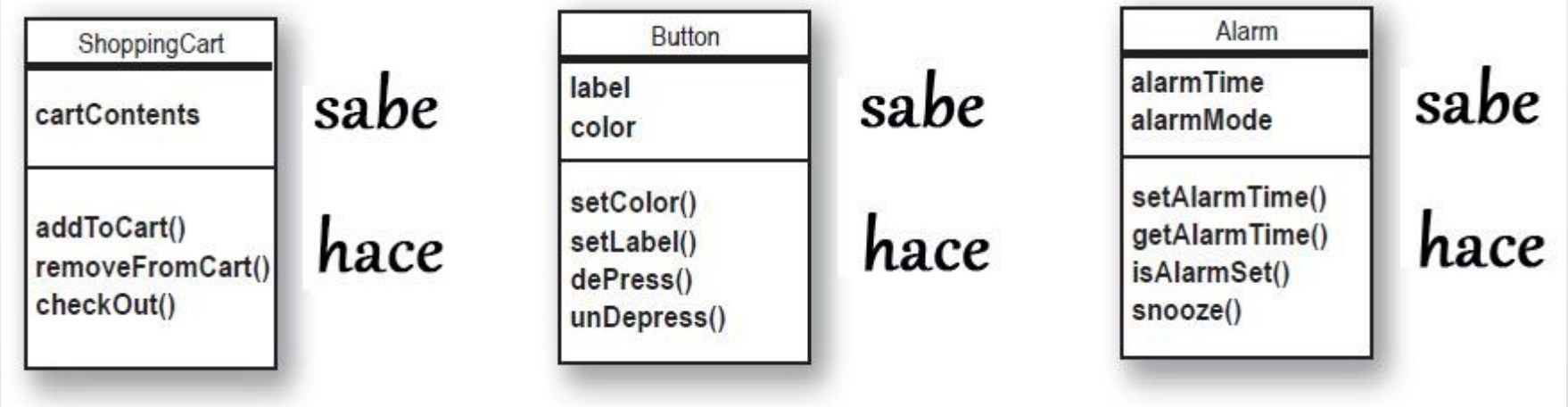
- Plantilla o estructura preliminar que descriu un objecte i definix les seues característiques (variables d'instància) i operacions (mètodes).

  - **Variables d'instància**: són les dades que tindran els objectes de la classe. Una classe pot contindre un conjunt de variables d'instància o cap. Es declaren amb un nom i un tipus de dades.

  - **Mètodes**: són les operacions que podran realitzar els objectes de la classe.

# Què és una classe?(II)

- Quan es dissenya una classe, es pensa en els objectes de la classe que es crearan:
  - Allò que l'objecte sap
  - Allò que l'objecte fa



# Exemple de classe

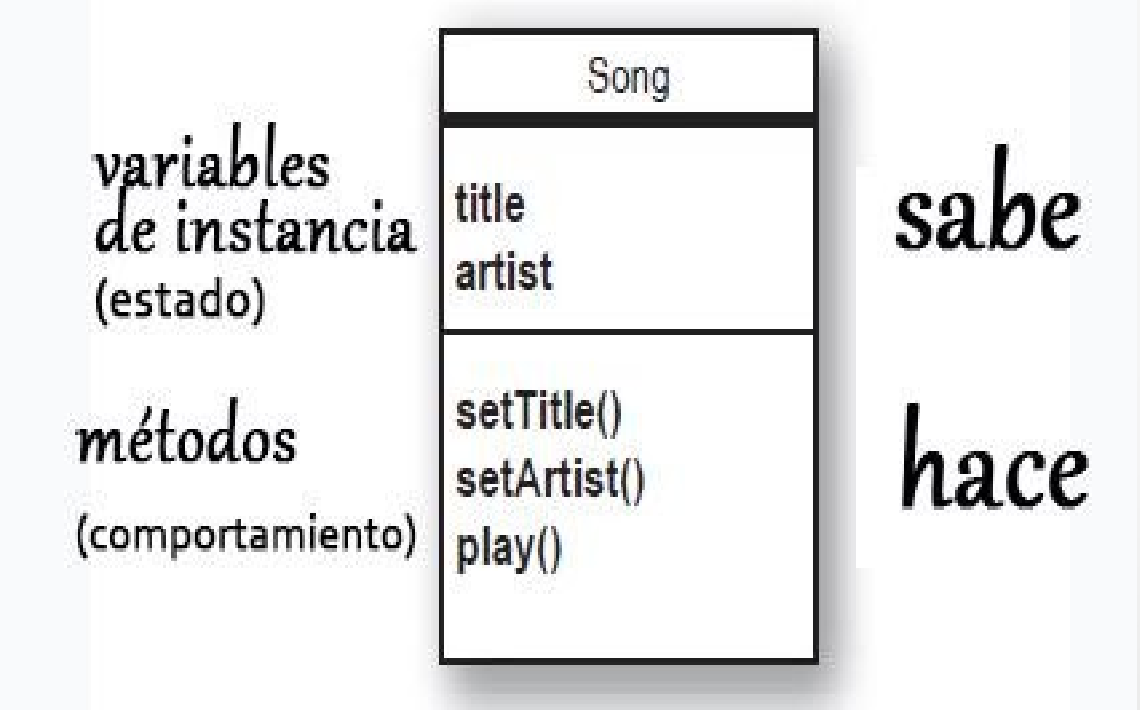
```
class Alumne{
    String nom;
    int edat;

    void setNom(String n);
    void setEdat(int e);
    String getNom();
    int getEdat();
    void visualitzarDades();
}
```

VARIABLES D'INSTÀNCIA

MÈTODES

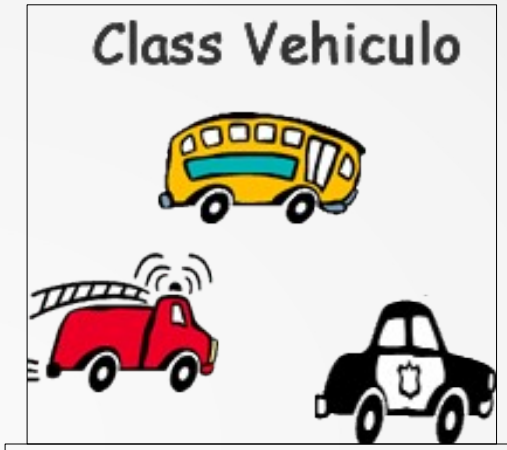
# Comportament dels objectes (I)

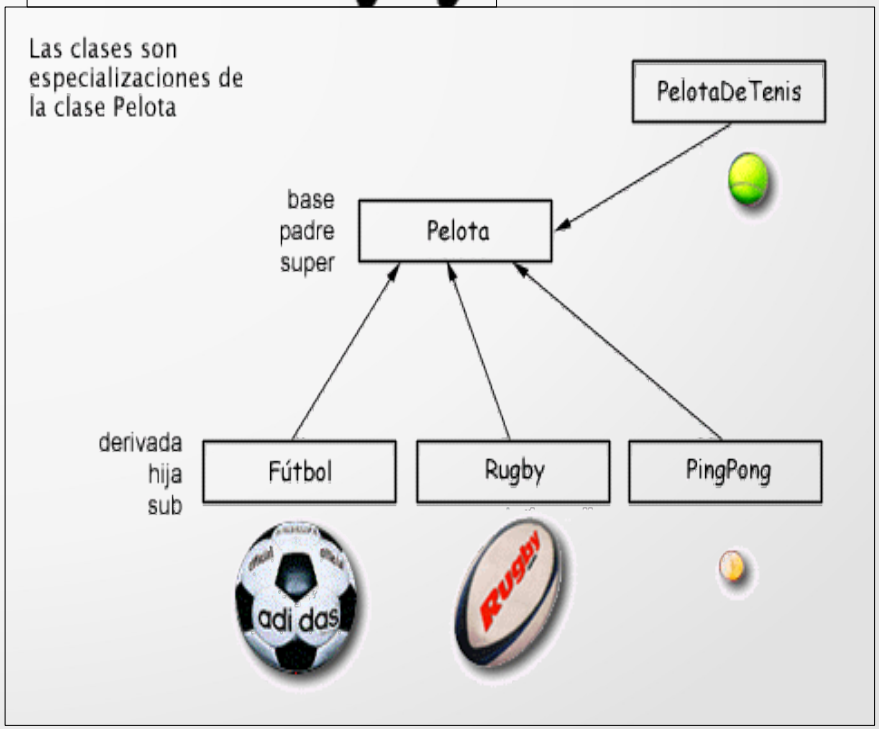
- Una classe és una “plantilla” per als objectes



# Què és un objecte?(I)

- És una **instància** d'una **classe**.
- Característiques dels objectes:
  - **Identitat**: es distingixen uns dels altres.
  - **Comportament**: poden realitzar tasques.
  - **Estat**: guarden informació que pot canviar amb el temps.



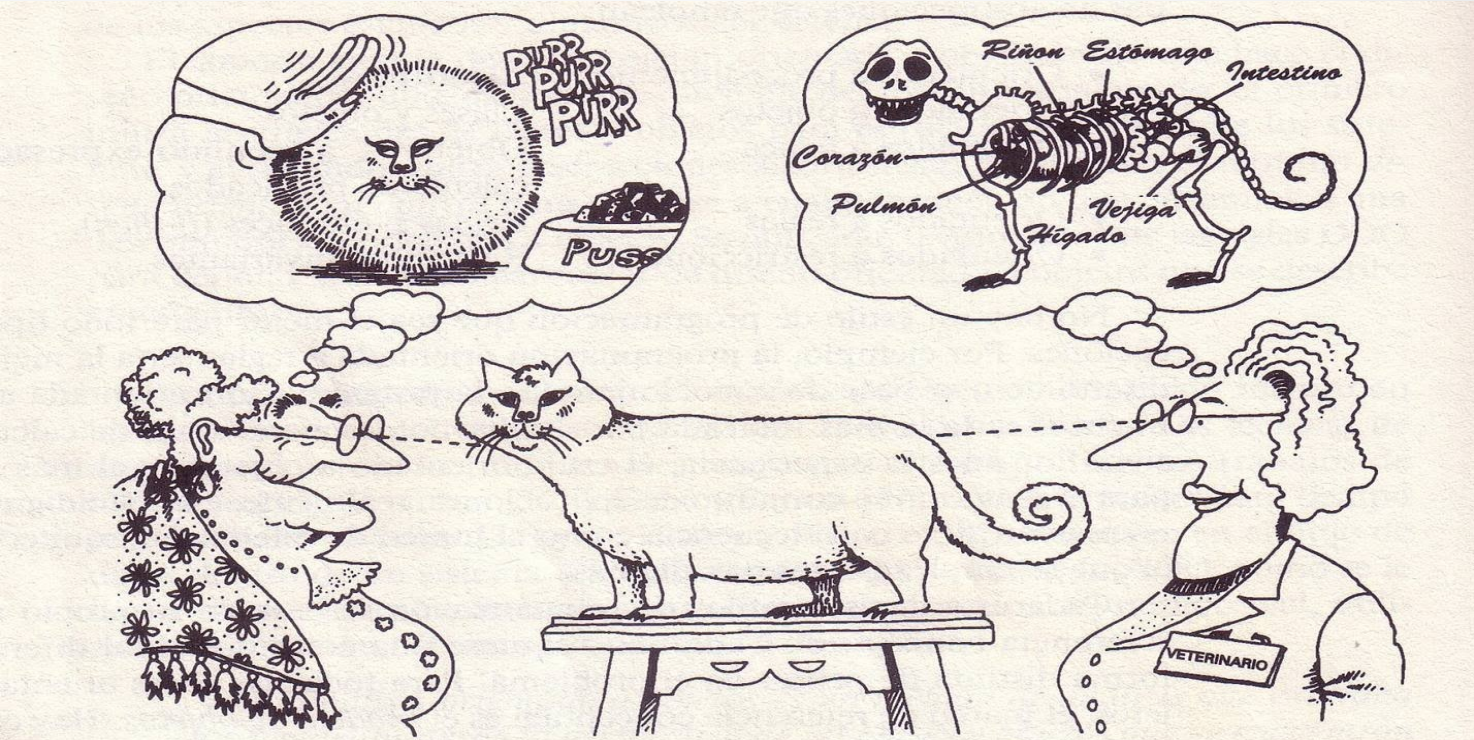



# Principis bàsics de la OO

- Abstracció
- Encapsulació
- Herència (ho vorem més endavant)
- Polimorfisme (ho vorem més endavant)

# Abstracció(I)

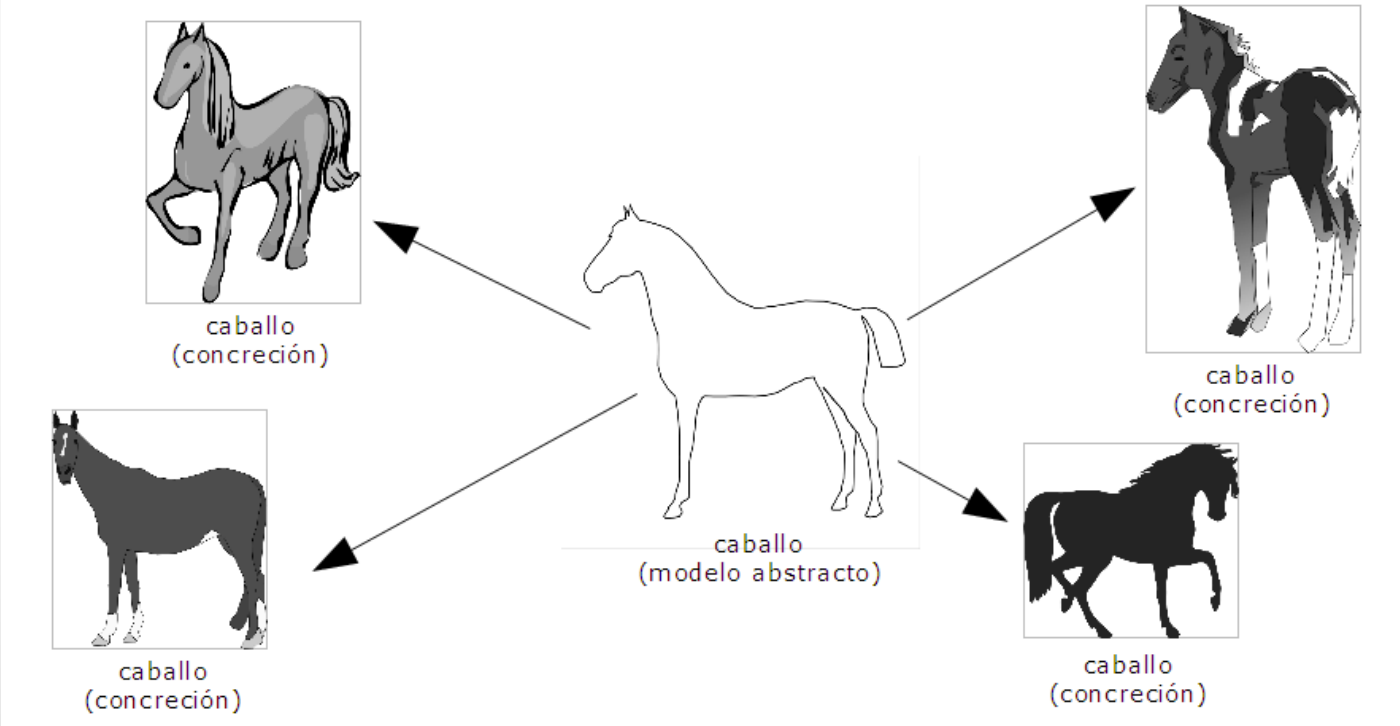
És el mecanisme per a determinar els tipus de classes, agrupant dades i funcions. No s'entra en detalls de la representació interna.




La abstracción se centra en las características esenciales de algún objeto, en relación a la perspectiva del observador.

# Abstracció(II)

Exemple: tenim cavalls de diferent raça. El seu aspecte exterior és molt diferent, però sabem que tots pertanyen  a la classe Cavall perquè realitzem una abstracció o identificació mental dels elements comuns que tenen (cola, quatre pates, són ràpids...).



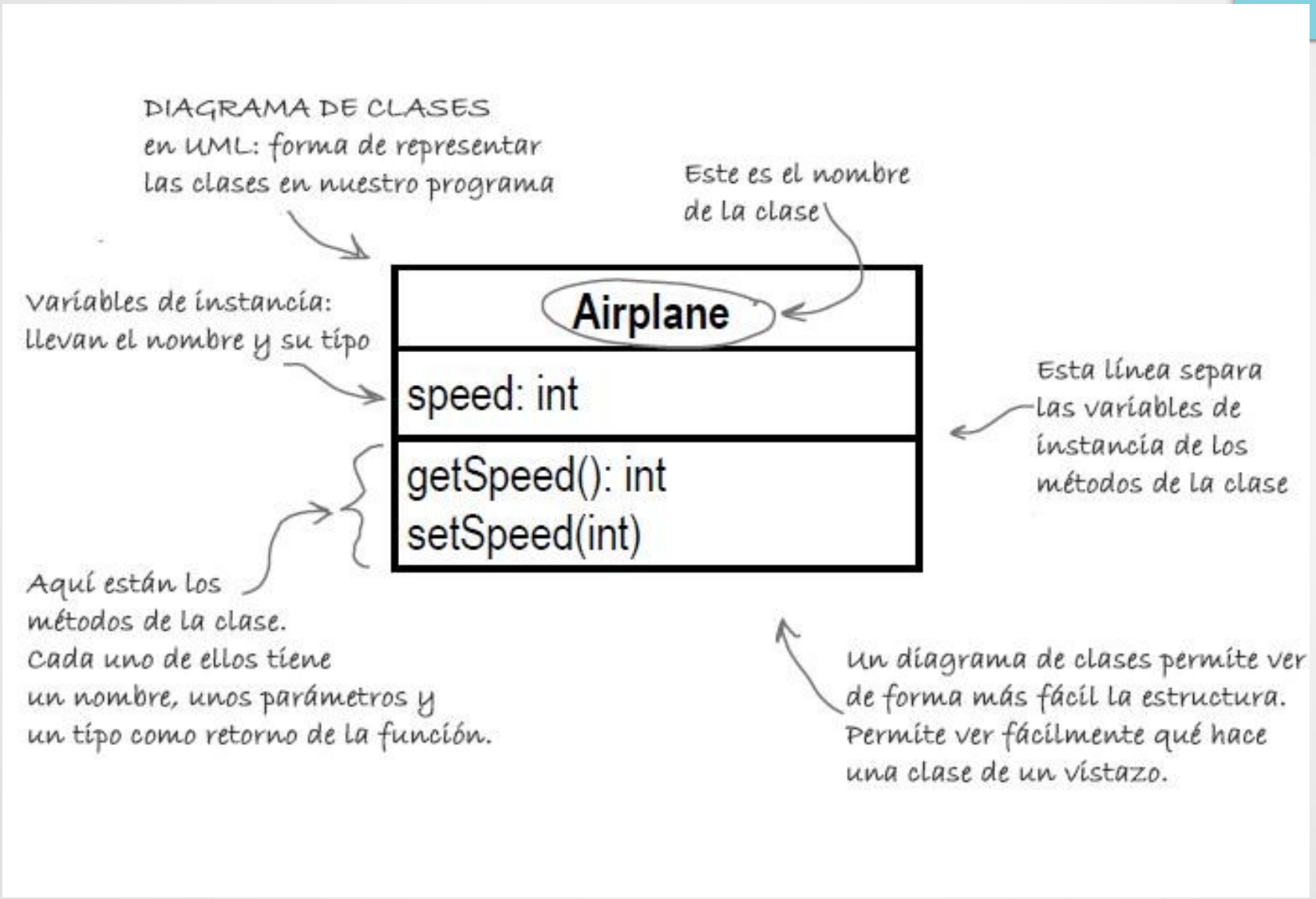
# Encapsulació(I)

Els objectes seran “caixes negres”: sabem **què fa** a través de la seua interfície **però no** sabem **com** ho fa.

# Encapsulació(II)

Exemple: un terminal d'autoservei és senzill d'utilitzar per a l'usuari. S'oculten els detalls d'implementació interna (que és complexa).

# Diagrama de classes (I)



# Treballant amb classes(I)

- Què necessitem (**de moment**)?
  - Una classe que modelarà allò que volem representar.
  - Una altra classe per provar que contindrà el mètode *main().*

# Crear una classe (I)

- Sintaxis

```
[modificador_accés] class NomClasse
    {
        //atributs de la classe
        //mètodes de la classe

    }
```

- Recordeu que una classe ha de ser creada en un fitxer amb el nom NomClasse.java
- En un fitxer pot haver més d'una classe, però sols una amb el modificador d'accés *public*.
- És preferible tindre les classes que modelen la informació del sistema separades d'aquelles que les utilitzen (en les quals s'inclou el mètode main()).

# Modificadors d'accés

| Paraula clau | Definició |
| --- | --- |
| ***public*** | La classe és accessible des d'altres *packages*. |
| ***package***<br>(per defecte) | La classe serà visible en totes les classes declarades en el mateix *package*. |

# Com agregar **atributs**?

- Sintaxis general

[modificadorAccés] [static][final][transient][volatile] tipus nom_atribut

- De moment, versió reduïda

[modificadorAccés] [static][final] tipus nom_atribut

# Com agregar **mètodes**?

- Sintaxis general

```
[modificadorAccés] [static][abstract][final][native][synchronized]
      [tipus_retornat|void] nomMètode ([llistaParàmetres])
[throws llistaExcepcions]
```

- De moment, versió reduïda

```
[modificadorAccés] [static]
      [tipus_retornat|void] nomMètode ([llistaParàmetres])
```

# Com agregar mètodes?(II)

- Els mètodes poden estar **sobrecarregats**
  - Dos o més mètodes amb el mateix nom però amb una llista de paràmetres diferent:
    - Distint número de paràmetres o
    - Almenys un paràmetre de tipus diferent
- La sobrecàrrega de mètodes fa que el mètode siga més flexible a l'hora d'utilitzar-lo.

# Modificadors d'accés de membres (I)

- Java té 4 modificadors d'accés que qualifiquen a atributs i mètodes:

| **Paraula clau** | **Definició** |
|---|---|
| *private* | L'element sols és accessible dins del fitxer en el qual està definit. |
| *package* (per defecte) | L'element sols és accessible dins del *package* on està definit. |
| *protected* | L'element és accessible dins del *package* on està definit i, a més a més, en les subclasses. |
| *public* | L'element és accessible des de qualsevol lloc. |

# Com agregar atributs. Exemples

- Els atributs sempre haurien de ser *private*

```
public class Persona{
   private String nom;
   private int edat;
   …
}
```

# Com agregar mètodes. Exemples

```
public class Persona{
   …

   public int calcularEdat(){
      ...
   }
}
```

```
public class Triangle{
   …
   int calcularArea(){
      ...
   }
   public int mostrarInfo(){
      ...
   }
}
```

# Com agregar mètodes. Exemple

```
public class Triangle{

   private int costat1;
   private int costat2;
   private int costat3;
   …

   public void esEquilater(){
      if (costat1==costat2) && (costat2==costat3){
         System.out.println("Es equilàter");
      } else {
         System.out.println("No és equilàter");
      }
   }
}
```

# Crear instàncies d'una classe (I)

- Quan creem una classe, estem definint una plantilla amb la qual es definirà allò que els objectes saben (atributs o variables d'instància) i el que fan (mètodes). Una vegada fet això, es pot instanciar la classe (crear objectes).

- Per a declarar un objecte d'una classe:

  NomClasse variable;

- Amb açò tenim un apuntador capaç d'adreçar l'objecte, però **no tenim l'objecte**:
  - De moment la variable no apunta a cap objecte
  - Es diu que conté la referència **null**

# Crear instàncies d'una classe (II)

- Per a crear un objecte, utilitzem la paraula reservada ***new*** seguida d'un mètode que es diu igual que la classe (el constructor)

- Així aconseguim una variable que apunta a l'objecte creat.

- Podem declarar l'objecte i després crear-lo:

```
Persona p1;
Persona p2;
p1=new Persona();
p2=new Persona();
```

- Podem declarar i crear l'objecte al mateix temps:

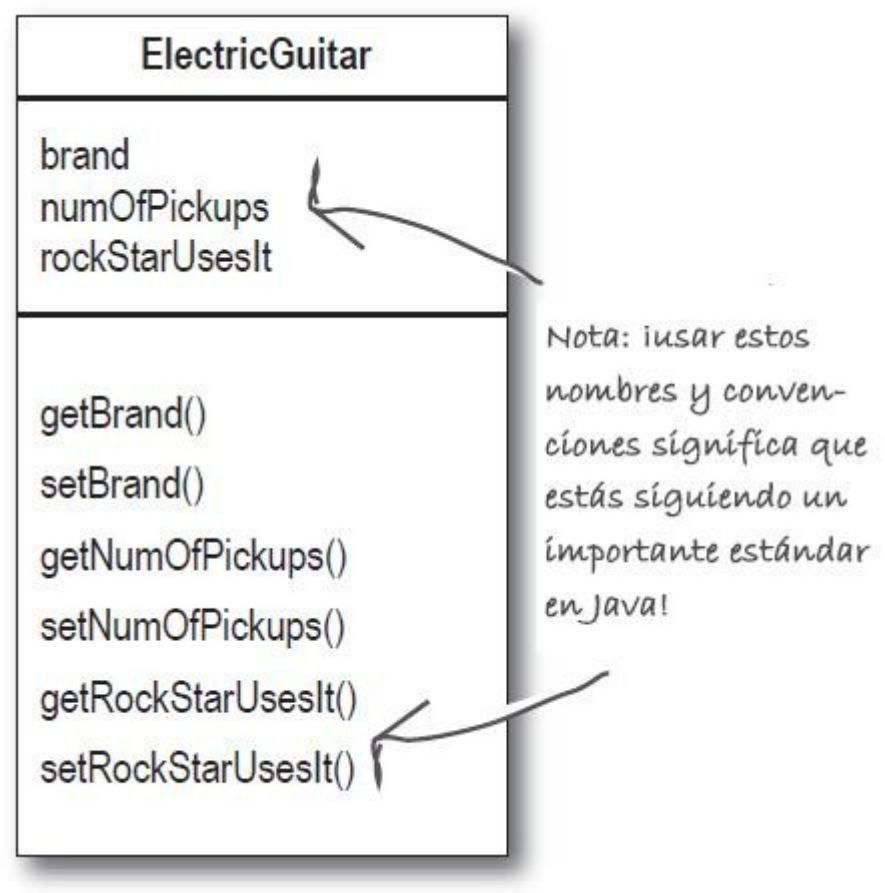
```
Persona p1=new Persona();
```

# *Setters* (modificadors) i *Getters* (consultors)

- Com s'ha de procedir amb els atributs d'una classe?
  - Crear els atributs amb el modificador *private*.
  - Crear mètodes públics per accedir als atributs:
    - Per a consultar-los (*getters*)
    - Per a modificar-los (*setters*)
  - Des d'altres classes externes no es podran ni accedir ni modificar els atributs si no es fa mitjançant els *getters* i *setters*.

- Beneficis de l'**encapsulament**:
  - Que ningú accedisca per equivocació o sobreescriga funcionalitats quan no deu.
  - Un programador que utilitze un mètode, sols necessita saber què fa, no com ho fa (caixa negra).

# *Setters* (modificadors) i *Getters* (consultors)




```
class ElectricGuitar {

    String brand;
    int numOfPickups;
    boolean rockStarUsesIt;

    String getBrand() {
        return brand;    // devuelve la variable 'brand'
    }

    void setBrand(String aBrand) {
        brand = aBrand;    // modifica la variable 'brand'
    }                      // al valor del parámetro pasado

    int getNumOfPickups() {
        return numOfPickups;
    }

    void setNumOfPickups(int num) {
        numOfPickups = num;
    }

    boolean getRockStarUsesIt() {
        return rockStarUsesIt;
    }

    void setRockStarUsesIt(boolean yesOrNo) {
        rockStarUsesIt = yesOrNo;
    }
}
```

# Encapsulament

```
class GoodDog {

  private int size;

  public int getSize() {
    return size;
  }

  public void setSize(int s) {
    size = s;
  }

  void bark() {
    if (size > 60) {
       System.out.println("Wooof! Wooof!");
    } else if (size > 14) {
       System.out.println("Ruff!  Ruff!");
    } else {
       System.out.println("Yip! Yip!");
    }
  }
}
```

la variable size será privada

los métodos getter y setter serán públicos

Aunque pensemos que los métodos no tienen nuevas funcionalidades, más tarde podremos añadirlas cambiando el código si queremos.

```
class GoodDogTestDrive {

   public static void main (String[] args) {
     GoodDog one = new GoodDog();
     one.setSize(70);
     GoodDog two = new GoodDog();
     two.setSize(8);
     System.out.println("Dog one: " + one.getSize());
     System.out.println("Dog two: " + two.getSize());
     one.bark();
     two.bark();
   }
}
```

# Encapsulament

Cualquier lugar en el que puede ser usado un valor, también puede usarse una llamada a un método que devuelve ese tipo

En vez de:

```
int x = 3 + 24;
```

puedes usar:

```
int x = 3 + one.getSize();
```

# Els constructors(I)

- Mètode especial d'una classe que és cridat automàticament sempre que es crea un objecte, és a dir, a l'emprar la instrucció new.
- La seua funció és iniciar l'objecte.
  - Es recomana que els constructors inicialitzen totes les variables d'instància de l'objecte

```
public class Rectangulo {
...
public Rectangulo(int x1, int y1, int w, int h) {
   x=x1;
   y=y1;
   ancho=w;
   alto=h;
   }
...
}
```

# Els constructors(II)

- Per a declarar un constructor, és suficient amb declarar un mètode amb el mateix nom que la classe.
  - No es declara tipus de dades retornat pel constructor, ni tan sols *void*.
- Si no n'hi ha cap constructor en la classe, Java s'inventa un que no té arguments e inicialitza tots els atributs als valors per defecte.
  - Java sols s'inventa els constructors si no n'hi ha cap.
  - Si n'hi ha algun constructor, Java es limita a fer allò que el constructor diu.

| | |
|---|---|
| enteros | 0 |
| decimales | 0.0 |
| booleanos | falso |
| referencias | null |

# Els constructors(III)

És possible declarar diferents constructors (sobrecàrrega de mètodes) a l'igual que la resta de mètodes de la classe.

```
public class Rectangulo {
     private int x;
     private int y;
     private int ancho;
     private int alto;

     public Rectangulo() {
          x=0;
          y=0;
          ancho=0;
          alto=0;
     }
     public Rectangulo(int x1, int y1, int w, int h) {
          x=x1;
          y=y1;
          ancho=w;
          alto=h;
     }
     public Rectangulo(int w, int h) {
          x=0;
          y=0;
          ancho=w;
          alto=h;
     }...
...}
```

# Els constructors. Exemple

```
class Ficha {
   private int casilla;
   Ficha() { //constructor
         casilla = 1;
   }
   public void avanzar(int n) {
         casilla += n;
   }
   public int casillaActual(){
         return casilla;
   }
}

public class app {
   public static void main(String[] args) {
         Ficha ficha1 = new Ficha();
         ficha1.avanzar(3);
         System.out.println(ficha1.casillaActual());//Da 4
```

# Destructors?

- En Java n'hi ha un recolector de basura (***garbage collector***) que s'encarrega de gestionar els objectes que es deixen d'utilitzar i alliberar l'espai que ocupen en memòria.

- Este procés és **automàtic i impredictible** i treballa en un fil (*thread*) de **baix prioritat.**

- En termes generals, este procés de recolecció de basura treballa quan detecta que algun **objecte fa molt de temps que ja no s'utilitza** en el programa.

- L'eliminació depén de la màquina virtual. Normalment, es realitza de forma periòdica.

# La referència *this*

- La paraula reservada *this* és una referència al propi objecte amb el qual estem treballant.
- Exemple:

```
class punto {
   int posX, posY;//posición del punto
   punto(posX, posY){
         this.posX=posX;
         this.posY=posY;
   }
```

  - En este exemple, cal la referència *this* per clarificar quan s'utilitzen les propietats posX i posY i quan els arguments amb el mateix nom.

# Atributs *static*

- Existeixen dos tipus d'atributs:
  - **Atributs d'objecte (variables d'instància)**
    - Són variables o objectes que guarden **valors diferents** per a instàncies diferents de la classe (per a objectes diferents).
    - Si no s'especifica de forma explícita, els atributs són d'objecte.
  - **Atributs de classe (variables de classe)**
    - Són variables o objectes que guarden el **mateix valor** per a tots els objectes instanciats a partir de la classe.
    - Es declaren amb la paraula reservada ***static***.

# Mètodes *static*

- Els mètodes static són mètodes de classe.
  - No és necessari crear un objecte de la classe (instanciar la classe) per poder cridar a un mètode static.
  - S'utilitzarà el nom de la classe per poder treballar.

- Els mètodes de classe (static) únicament poden accedir als seus atributs de classe (static) i mai als atributs d'objecte (no static).

- Fins ara, els hem utilitzat:
  - Sempre que es declarava una classe executable
    - Per poder executar el mètode main() no es declara cap objecte d'eixa classe.
    - Quan hem creat els mètodes sense crear objectes en la UD6.

# Agrupació de classes

- Els ***packages*** són una forma d'agrupar vàries classes:
  - per a estructurar les classes en grups relacionats
  - per a aprofitar la possibilitat de donar als membres d'una classe la visibilitat a nivell de *package*

- Quan no s'especifica el nom del *package* al qual pertany una classe, passa a estar en el *package* per defecte (default package en Eclipse)

- La declaració del *package* es fa a l'inici del fitxer .java:
  - package x.y.x;

    Significa que la classe definida en eixe fitxer serà del *package* x.y.z

# *Packages* i import

- El nom complet d'una classe és el nom del *package* en el qual es troba la classe, punt i després el nom de la classe.
  - Exemple: si la classe Cotxe està dins del *package* locomocio, el nom complet de Cotxe es locomocio.Cotxe
- Mitjançant el comando import s'evita haver de col·locar el nom complet.
- El comando import es col·loca abans de definir la classe:
  - Exemple: import locomocio.Cotxe;
- Gràcies a esta instrucció es pot utilitzar la classe Cotxe sense necessitat d'indicar el *package* on es troba.
- Es pot emprar el símbol * com a comodí.
  - Exemple: import locomocio.*;

  //Importa totes les classes del *package* locomocio

# Llibreries de classes(I)

- Hem vist que en Java n'hi ha una gran quantitat de classes ja definides i utilitzables. Vene agrupades en packages o llibreries estàndard:
  - **java.lang** - classes essencials, números, Strings, objectes, compilador... (es l'únic package que s'inclou automàticament en tots els programes Java)
  - **java.io** - classes que manegen entrades i eixides
  - **java.util** - classes útils, como estructures genèriques, maneig de data i hora, números aleatoris, entrada estàndard, ...
  - **java.net** – suport per a xarxes: URL, TCP, UDP, IP ...