

Programació orientada a objectes

Possible classificació

Llenguatges de programació:

Antics (procediments, dècada dels 60 i principis dels 70)

Moderns (Orientat a Objectes. Més actuals)

Llenguatges de programació:

- Antics (Fortran, Cobol, Basic...)
 - Codi llarg en aplicacions complexes
 - Errors difícils de trobar, i molt probable que el programa caigui.
 - Ús per un altre programador complicat
 - Difícil de reutilitzar (molt difícil de canviar)
- Moderns
 - Abstracció del món real.

Exemple de codi Basic (diferent de Visual Basic)

Listado de un programa en BASIC

```
3LOAD TWO SQUARES
3LIST

30 HOME : HGR
35 PRINT CHR$(4); "BLOAD SHAPE 1,A#0300"
36 POKE 232,0: POKE 233,03
37 POKE - 16302,0
40 ST$ = "T W O S Q U A R E S "
60 A = 90:B = 5
70 HCOLOR= 7: ROT= 0
80 FOR S = 1 TO 40
90 SCALE= S
100 DRAW 2 AT A,B
101 DRAW 2 AT A + 1,B + 1
105 HPL0T 90,5 TO 10,86
110 NEXT S
120 HCOLOR= 5
130 X = 260:Y = 75
140 ROT= 5
150 FOR S = 60 TO 1 STEP - 1
160 SCALE= S
170 DRAW 2 AT X,Y: DRAW 2 AT X + 1,Y + 1
180 NEXT S
185 PRINT CHR$(4); "BLOAD SHAPE ALPHABET,A#6000"
186 VT = 21:HT = 2
187 SCALE= 2: ROT= 0
*190 GOSUB 5000
195 REM THIS ADDS LINES TO FRAME 1
200 FOR S = 200 TO 175 STEP - 4
```

Beginner's
All-Purpose
Symbolic
Instruction
Code

Programació orientada a objectes

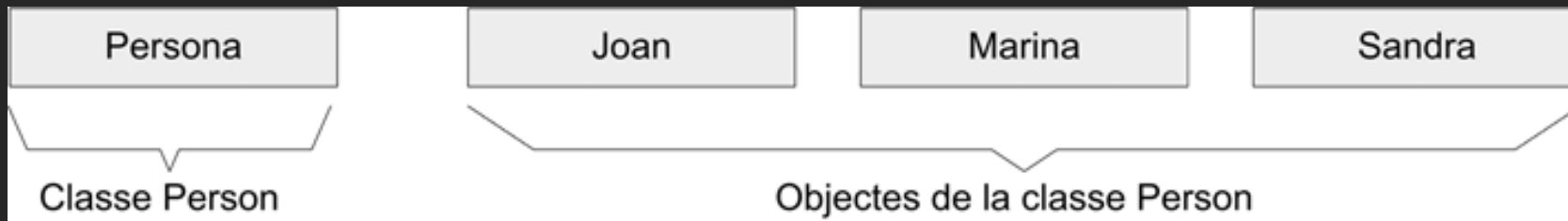
Evolució del paradigma de programació estructurada que permet la modelització d'objectes del món real, amb les seves característiques i propietats internes.

‘El llenguatge de programació Java’

La classe i l'objecte

La classe cotxe: no es refereix a una única entitat, sinó a un **conjunt d'entitats** amb unes característiques comuns.

Si parlem del cotxe d'en Josep, ens referim a un cotxe en concret, en aquest cas s'ha d'entendre com **un objecte o instància** que identifica un membre individual i concret de la classe d'objectes cotxe.



A la **classe Persona**, estan representades les propietats que caracteritzen una persona (entitat del món real), mentre que els diferents objectes representen individus concrets.

Un **atribut** és una característica o propietat de l'entitat.

El **mètode** mostra, o defineix, part del comportament dels objectes representats en la classe.

Mètodes

És una funció associada a un objecte, és una acció que executem sobre les dades de l'objecte. Es defineixen com les funcions.

Hi han **dos tipus de mètodes**; els que afecten els valors dels atributs de l'objecte i que, per tant, poden tenir efectes col·laterals (canvien l'estat, getters/setters) sobre el mateix objecte o sobre d'altres, i els que simplement calculen certs valors i després són retornats sense afectar cap altre objecte (areaCercle()).

Accés

- Tenim classes ja definides o podem crear de noves.
- Els objectes tenen les seves propietats i els seus mètodes.
- Per accedir: `objecte.Propietat`
- Per fer la crida al mètode:

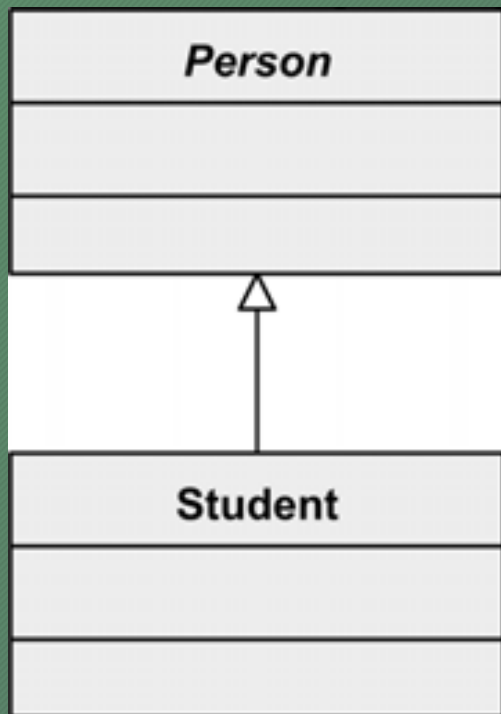
`objecte.NomMetode(parametre1,paramatre2)`

Per exemple , `System.out`

Representa una instància de `PrintStream` que es connecta a la consola estàndard de sortida. Quan s'invoca `System.out.print()` o `System.out.println()`, el text s'imprimeix en la consola. `print()` i `println()` són mètodes.

Diagrama de classes

L'objectiu principal de la metodologia d'orientació a objectes és descriure la realitat fent servir objectes.



Les relacions
amb altres objectes.
Herència.

El llenguatge unificat de modelatge (**UML**,
per les seves sigles en anglès, Unified Modeling
Language) és el llenguatge de modelatge de
sistemes de programari

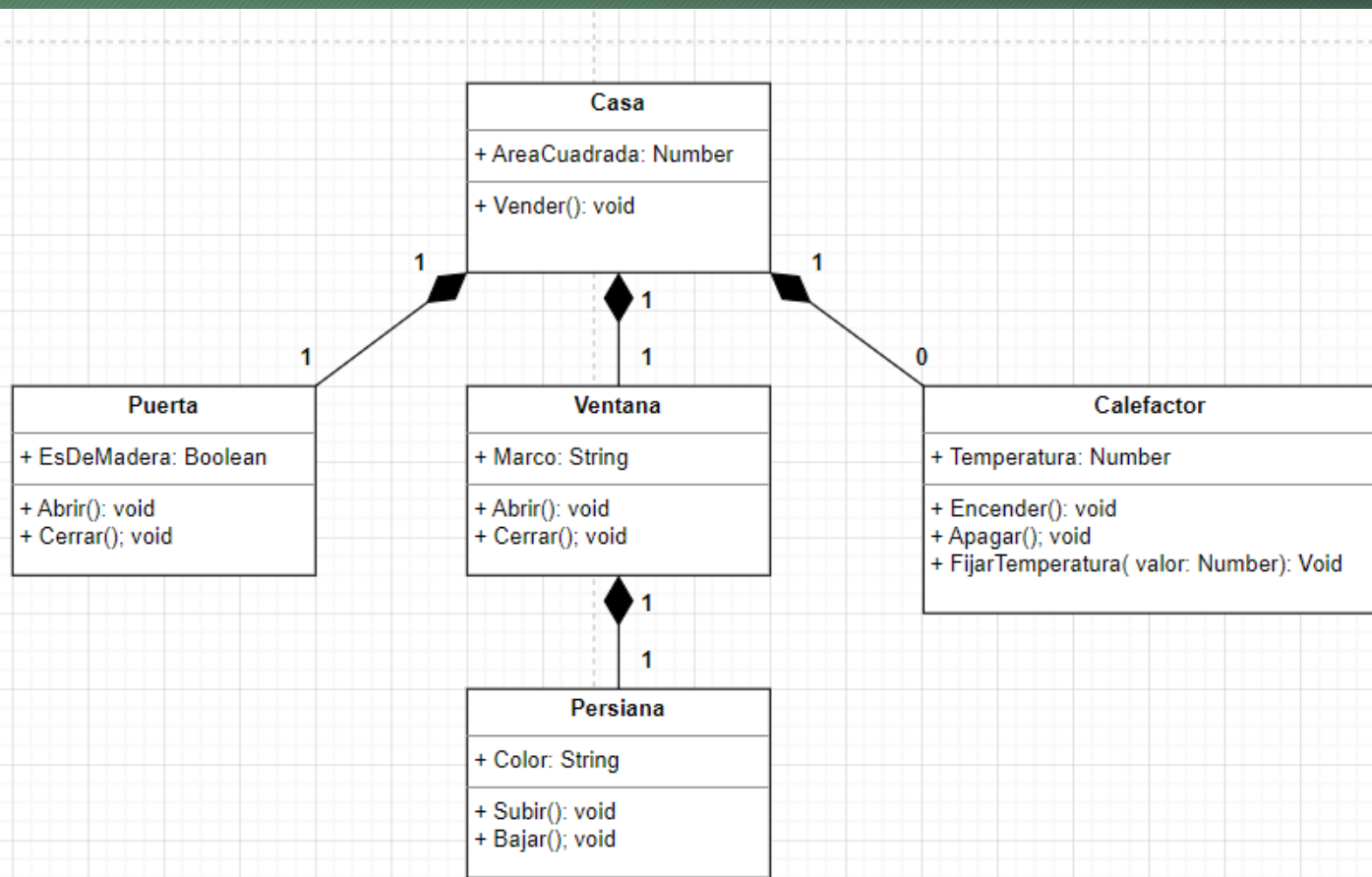
Representació de la classe Person amb UML



Dos tipus de mètodes:

```
sandra.setBirthDate("23/04/1980");  
int e = sandra.getAges();
```


Representació d'un diagrama de classes amb UML



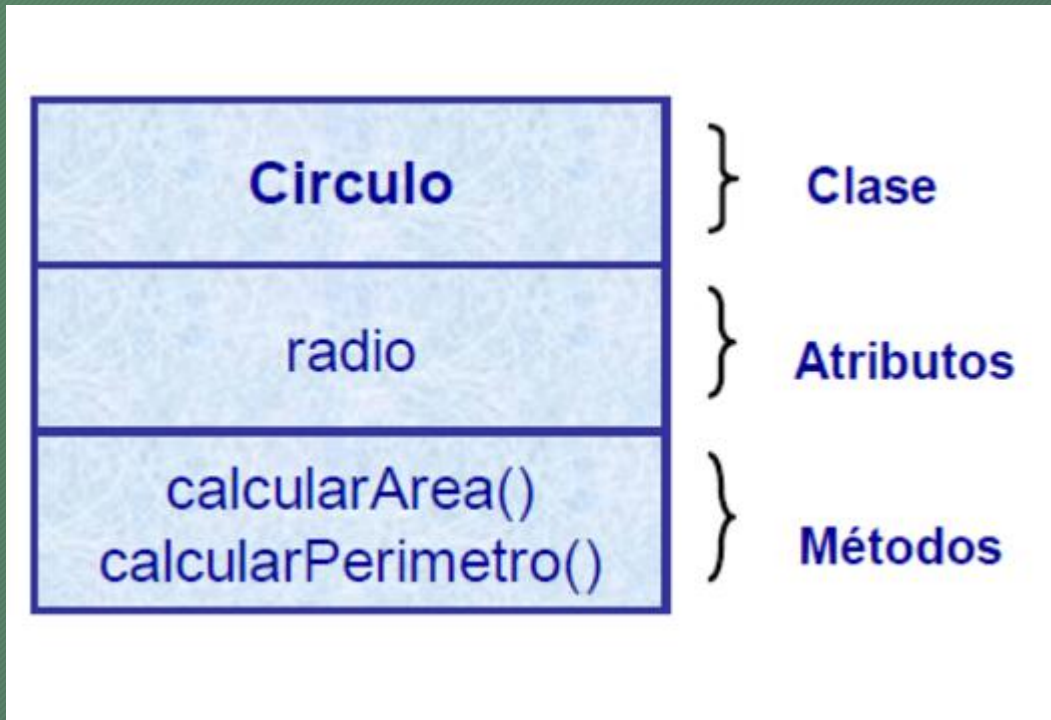
Dos tipus de llenguatges orientats a objectes:

- **Llenguatges purs:** tots els mòduls d'un programa han de ser classes (Scala).
- **Llenguatges híbrids:** permeten utilitzar classes i objectes que es poden combinar amb estructures procedimentals clàssiques (**Java**, C++, JavaScript).

Pensar en una possible classe...

Per exemple la Classe circulo:

Sol



Conceptes

1. Classe
2. Objecte
3. Instància de Classe
4. Modularització
5. Encapsulament
6. Herència
7. Polimorfisme
8. Relacions

Pilars de la POO

1. Abstracció
2. Encapsulament
3. Herència
4. Polimorfisme

Avantatges de la POO:

- Un programa es pot dividir en trossos de codi, mòdul, Classes... **Modularització**
- Molt reutilitzable. **Herència.**
- Si existeix algun error en alguna línia de codi, el programa no caurà. Es tracten els errors (**excepcions**).
- **Encapsulament.** Els objectes es poden comunicar entre si però no necessiten saber de les dades dels

Modularització

Dividir un gran programa en diferents parts que s'uneixen entre si per a formar un tot.

Avantatges

- Errors més localitzables i més fàcils de resoldre

Java: dos o més classes unides entre elles per a formar una unitat.

Equip compacte de música:



Equip modular de música:



Classe / objecte





Nomenclatura del punt

```
Objecte1.Numero=1245787;  
Objecte2.Titular="Maria del Pilar";  
Objecte3.SaberSaldo();
```

Creació d'un objecte (d'una classe):

Java ens permet crear els nostres propis objectes amb les seves pròpies propietats i els seus mètodes.

Crear un nou objecte consisteix a declarar una funció (un **constructor**).

El constructor és l'encarregat de donar un estat inicial al objecte.

Mensajes y métodos

El modelado de objetos no sólo tiene en consideración los objetos de un sistema, sino también sus interrelaciones.

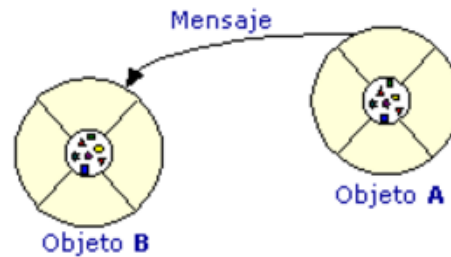
- **Mensaje**

Los objetos interactúan enviándose mensajes unos a otros. Tras la recepción de un mensaje el objeto actuará. La acción puede ser el envío de otros mensajes, el cambio de su estado, o la ejecución de cualquier otra tarea que se requiera que haga el objeto.

- **Método**

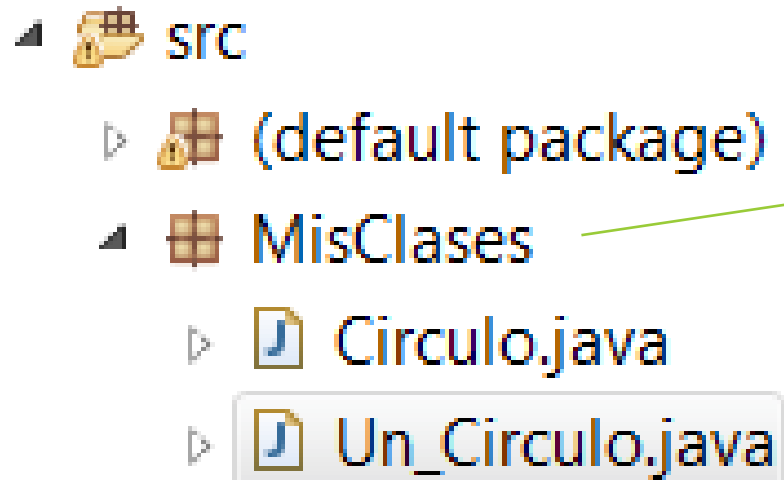
Un método se implementa en una clase, y determina cómo tiene que actuar el objeto cuando recibe un mensaje.

Cuando un objeto A necesita que el objeto B ejecute alguno de sus métodos, el objeto A le manda un mensaje al objeto B.



Al recibir el mensaje del objeto A, el objeto B ejecutará el método adecuado para el mensaje recibido.

Creació de Classes Java:



Paquet dins de src

‘MiAplicacion’ o Main

```
package MisClases; //Clase Java
```

```
public class Circulo {  
    int radio;
```

```
        //constructor;
```

```
public Circulo() {  
    radio=2;
```

```
}  
}
```

```
package MisClases;
```

```
public class Un_Circulo {
```

```
    public static void main(String[] args) {  
        // TODO Auto-generated method stub
```

```
        Circulo circulo1=new Circulo(); //instanciar una clase
```

```
        System.out.println("el radio del circulo es:"+circulo1.radio);
```

```
    }
```


```
}
```

<terminated> Un_Circulo [Java Application]
el radio del circulo es:2


```
package MisClases; //Clase Java
```

```
public class Circulo {  
    int radio;
```

```
    //constructor;  
    public Circulo() {  
        radio=2;  
    }  
}
```



Modularització:

Dos classes que no funciona una sense l'altra.

Defineix un estat inicial

```
1 package MisClases;
```

```
2  
3 public class Un_Circulo {
```

```
4  
5     public static void main(String[] args) {  
6         // TODO Auto-generated method stub
```

```
7  
8         Circulo circulo1=new Circulo(); //instanciar una classe
```

```
9  
10        System.out.println("el radio del circulo es:"+circulo1.radio);
```

```
11    }  
12  
13 }
```

Classe principal:
mètode main

Una altra manera molt menys aconsellada... sense modular

```
package MisClases;  //Clase Java

public class Circulo {
    int radio;

    //constructor;
    public Circulo() {
        radio=2;
    }

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        Circulo circulo1=new Circulo(); //instanciar una classe

        System.out.println("el radio del circulo es:"+circulo1.radio);
    }
}

class triangulo{}
class cuadrado{}
```

Tot en un únic arxiu.

Tota la definició de classes juntes.

Constructor amb pas de paràmetres

```
        //constructor;  
public Circulo() {  
    //radio=2;  
  
}        //constructor;  
public Circulo(int radio_circ) {this.radio=radio_circ;}
```


La sobrecàrrega de mètodes

És la creació de diversos mètodes amb el mateix nom però que es diferencien en algun aspecte.

En el nostre cas Java diferencia els mètodes sobrecarregats amb **base al número i tipus d'arguments** que té el mètode **i no pel tipus que retorna**.

En el moment d'invocar al mètode és quan es decideix el mètode a executar.

-

Exemple

/* Mètodes sobrecarregats */

```
public int calculaSuma(int x, int y, int z){...}
```

```
public int calculaSuma(double x, double y, double z){...}
```

```
public int calculaSuma(int x, int y){...}
```

/* Error de compilació: aquests mètodes no estàn sobrecarregats */

```
public int calculaSuma(int x, int y, int z){...}
```

```
public double calculaSuma(int x, int y, int z){ ...}
```

```
public void unMetodo(int i) {  
    System.out.println("Un metodo con argumento entero");  
}  
public void unMetodo(float i) {  
    System.out.println("Un metodo con argumento real");  
}
```


Sobrecarrega de constructors



Els constructors donen un estat inicial als objectes.

Podem triar l'estat inicial dels objectes definint diferents constructors.

A lo millor no es coneixen tots els valors inicials en el moment de crear la instància d'una classe.

```
Persona p1 = new Persona();  
Persona p2 = new Persona("Alex");  
Persona p3 = new Persona(20);  
Persona p4 = new Persona("Alex", 20);
```

Constructor per defecte, sense paràmetres



Sobrecarrega de constructors.

En aquest cas tenim
tres constructors.


Com a mínim un.

```
public Persona() {}  
public Persona(String nombre) {  
    this.nombre = nombre;  
}  
public Persona(int edad) {  
    this.edad = edad;  
}  
  
public Persona(String nombre, int edad) {  
    this.nombre = nombre;  
    this.edad = edad;  
}
```


Ens podem trobar un constructor que anomeni a un altre constructor.

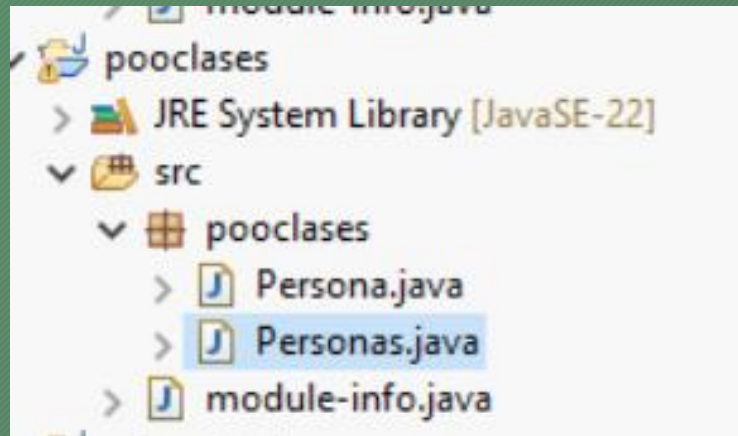
En el cas en el qual uns paràmetres ja venguen definits des de l'inici per defecte

```
public Persona() {}  
public Persona(String nombre) {  
    this(nombre, 18);  
}  
public Persona(int edad) {  
    this.edad = edad;  
}  
  
public Persona(String nombre, int edad) {  
    this.nombre = nombre;  
    this.edad = edad;  
}
```



```
Persona p3 = new Persona("Sara");
```


- 
- Crear les classes per a definir la classe Persona
La classe 'persona' es caracteritza per tenir nom, edat.
 - Crear les classes per a definir la classe Triangulo.
La classe 'triangulo' es caracteritza per tenir altura, base.



```
package pooclases;

public class Persona {
    String nom;
    int edad;
    public Persona() {

    }
    public Persona(String nombre) {
        this.nom=nombre;
    }
    public Persona(String nombre,int edat) {
        this.edad=edat;
    }
}
```

```
package pooclases;

public class Personas {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Persona persona1=new Persona("laura");
        System.out.println(persona1.nom);    }
}
```