

# M03. PROGRAMACIÓ BÀSICA

## UF1. Programació estructurada

# NF1. FONAMENTS DE LA PROGRAMACIÓ

# 1. INTRODUCCIÓ

Orígens de la  
programació

---

# 1. QUÈ ÉS UN PROGRAMA?

## ***Fer una pizza***

seguint la recepta escollida sabrem els ingredients necessaris i els passos a seguir per a cuinar-la

## ***Jugar a videojocs***

com a jugadors, ens podem moure pels diferents escenaris i, segons la situació, escollir quina acció ens reportarà més benefici

## ***Conduir un vehicle***

posar en marxa un vehicle, accelerar, moure el volant o frenar són les accions que ens permeten desplaçar-nos

# 1. QUÈ ÉS UN PROGRAMA?

Un **programa** és un conjunt d'instruccions (aritmètico-lògiques), executades de manera seqüencial, per tal d'obtenir el resultat esperat.

Una **instrucció** és un conjunt de dades dins d'una seqüència estructurada que el processador interpreta i executa.

# 1. QUÈ ÉS UN PROGRAMA?

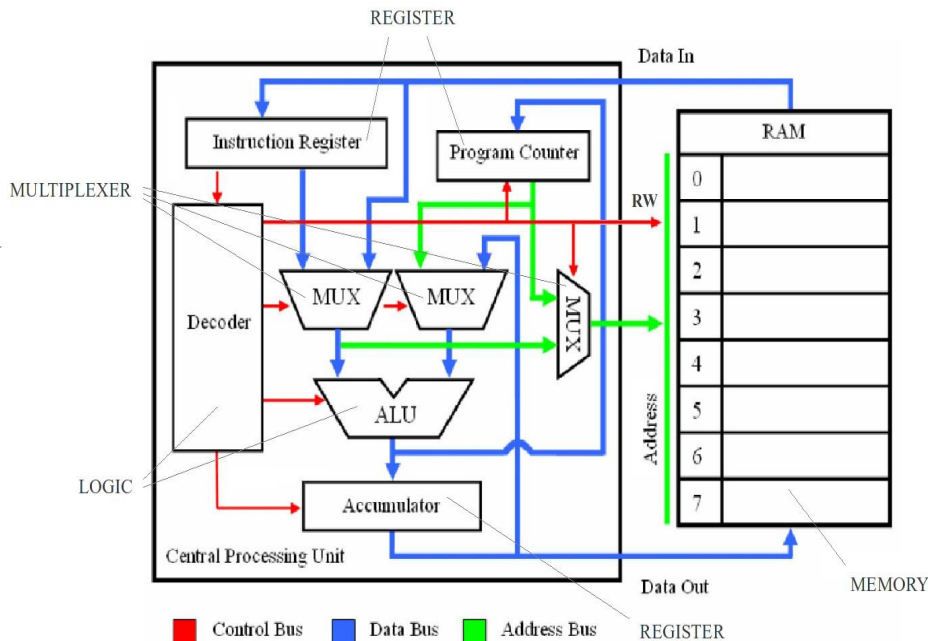
Seguint amb l'exemple de la pizza:

1. Pre-escalfar el forn a  $180^{\circ}\text{C}$  durant 5 minuts
2. Estendre la massa amb un rodet
3. Cobrir la massa amb el tomàquet fregit
4. Estendre els diferents ingredients
5. Ficar la pizza al forn, a  $180^{\circ}\text{C}$  durant 15 minuts
6. Treure la pizza i deixar que es refredi 5 minuts



# COM EXECUTA L'ORDINADOR UN PROGRAMA ?

1. S'introdueixen dades a l'ordinador
2. El processador executa les instruccions
3. Els resultats de les operacions s'emmagatzemen a la RAM
4. Els resultats finals s'emmagatzemen al disc dur



**Joseph Marie  
Jacquard**

(1752-1834)  
Inventor del teler de  
Jacquard

**Charles Babbage**

(1791- 1871)  
Creador de la màquina  
analítica

**Orígens de la  
programació**

**ADA LOVELACE**

(1815-1852)  
Primera programadora de  
la història.

**ALAN TURING**

(1912-1954)  
Pare de la informàtica  
moderna.



## 2. LLENGUATGES DE PROGRAMACIÓ

**Tipus de llenguatges i  
característiques**

---

# LLENGUATGE COMPILAT

- llenguatge d'alt nivell per a definir l'algorisme del programa (codi font)
- tradueix (compila) el codi font a un arxiu executable comprensible per a la màquina, en aquella plataforma.

Aquest arxiu pot executar el programa tantes vegades sigui necessari ,sense haver de repetir el procés de compilació. No obstant, l'arxiu només es podrà executar en el mateix SO en el que s'ha compilat (Windows, Linux, Mac).

**C, C++, Objective C, C#, Fortran, Pascal, Haskell i Visual Basic**

# LLENGUATGE INTERPRETAT

- llenguatge d'alt nivell per a definir l'algorisme del programa (codi font)
- tradueix (interpreta) el codi font pas a pas (instrucció per instrucció)
- es repeteix cada vegada que s'executa el programa el codi en qüestió.

Els llenguatges interpretats permeten el tipus dinàmic de dades, és a dir, no és necessari inicialitzar una variable amb un determinat tipus de dada, sinó que aquesta pot canviar el seu tipus en condició a la dada que emmagatzema en aquell moment, entre altres característiques més.

**Python, PHP, JavaScript, Ruby, Perl i MATLAB**

# NIVELLS D'ABSTRACCIÓ DELS LLINGUATGES DE PROGRAMACIÓ

ALT  
NIVELL

Llinguatges similars al nostre llenguatge natural

JAVA, C, C++, Python,...

Llinguatge màquina

Ensamblador (en funció de cada CPU)

Llinguatge binari

00010101010101111

BAIX  
NIVELL



# NIVELLS D'ABSTRACCIÓ DELS LLINGUATGES DE PROGRAMACIÓ

```
section .text
    global _start
_start:
    mov edx,len
    mov ecx,msg
    mov ebx,1
    mov eax,4
    int 0x80

    mov eax,1
    int 0x80

section .data
msg db 'Hello, world!', 0xa
len equ $ - msg
```

```
print ('Hello, world!')
```

PYTHON

ASSEMBLER

# TRADUCCIÓ A LENGUATGE MÀQUINA

COMPILAT

```
#include <stdio.h>
void main(){
    printf("Hola!");
}
```



INTERMEDI

```
public class Helloworld {
    public static void main(String[] args) {
        System.out.println("Hola!");
    }
}
```





VM

```
print ("Hello, world!")
```



INTERPRETAT

# TOP 10 DELS LLINGUATGES MÉS UTILITZATS

Jul 2022	Jul 2021	Change	Programming Language		Ratings	Change
1	3	▲		Python	13.44%	+2.48%
2	1	▼		C	13.13%	+1.50%
3	2	▼		Java	11.59%	+0.40%
4	4			C++	10.00%	+1.98%
5	5			C#	5.65%	+0.82%
6	6			Visual Basic	4.97%	+0.47%
7	7			JavaScript	1.78%	-0.93%
8	9	▲		Assembly language	1.65%	-0.76%
9	10	▲		SQL	1.64%	+0.11%
10	16	▲▲		Swift	1.27%	+0.20%

De tots els llenguatges existents

# 3. ALGORISMES

**Disseny i tècniques  
descriptives**

---



# ETAPES DEL DISSENY D'UN PROGRAMA

01

## Anàlisi

Estudi del problema a resoldre

02

## Disseny

Passos que haurà de seguir l'algoritme per a resoldre el problema

03

## Programació

Implementació de les instruccions de l'algorisme en un llenguatge de programació d'alt nivell

04

## Compilació

Conversió del codi d'alt nivell en llenguatge màquina

05

## Execució i proves

Execució del codi amb diferents valors d'entrada per provar que el codi funciona per a qualsevol cas

# 3. ALGORISMES

Anàlisi del problema

# ANÀLISI

En aquesta primera fase, s'estudia el problema a resoldre:

- quina és la solució demanada
- quines dades tenim
- com podem solucionar el problema
- quins errors i excepcions es poden donar

En cas de disposar de més informació relativa al problema (fòrmula, expressió matemàtica,...), també s'ha d'incloure com a informació d'anàlisi.

**Important!** Durant la fase d'anàlisi, no es busca una forma de resoldre el problema, només es tracta de comprendre'l.

# ANÀLISI

Un algorisme ben format haurà de complir les següents condicions:

- **Precís:** ha d'indicar pas a pas totes les operacions i instruccions a seguir per a resoldre el problema.
- **Ben definit:** si coneixem les entrades de l'algorisme, aquest sempre produirà la mateixa sortida i la màquina interna passarà per la mateixa seqüència d'estats (és un algorisme determinístic).
- **Finit:** l'algorisme ha d'acabar en algun moment.
- **Robust:** l'algorisme ha de tenir una resposta clara, independentment de la seva entrada i la seva execució.
- **Transportable:** un algorisme estarà dissenyat de manera que pugui executar-se, independentment del maquinari (hardware) i programari (SO) instal·lat.

# ANÀLISI. EXEMPLES

A l'escola de cuina van cada dia 234 alumnes de matí i 145 a la tarda. Quants alumnes van a l'escola de dilluns a divendres?

**Pregunta:** Quants alumnes van a l'escola de dilluns a divendres?

**Dades:**

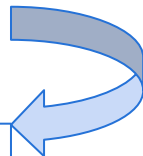
- 234 alumnes de matí
- 145 alumnes de tarda
- 5 dies

**Operacions:** una suma i una multiplicació



$$(234+145) \times 5 = 1895$$

**Sortida**



**Errors:** No hi ha (les dades són correctes).

# ANÀLISI. EXEMPLES

A l'escola de cuina van cada dia X alumnes de matí i Y a la tarda. Quants alumnes van a l'escola Z dies per setmana?

**Pregunta:** Quants alumnes van a l'escola Z dies per setmana?

**Dades:**

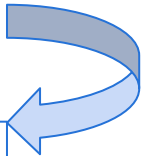
- X alumnes de matí
- Y alumnes de tarda
- Z dies

**Operacions:** una suma i una multiplicació



$$(X+Y) \times Z = \text{total}$$

**Sortida**



**Errors:** No hi ha (les dades són correctes).

# ANÀLISI. LES DADES

## Què és una dada?

Una **dada** és una **representació simbòlica** (numèrica, alfabètica, ...) d'una característica d'una entitat (objecte de la vida real). **No té valor semàntic** (sentit) per si mateixa, però convenientment tractada (processada) es pot emprar en la realització de càlculs o presa de decisions.

## I un tipus de dada?

És la definició del **conjunt de valors vàlids** que poden prendre unes dades i el **conjunt de transformacions** que s'hi pot fer.

# ANÀLISI. LES DADES. TIPUS DE DADES

- ***Simples***

- numèriques

- enters: representa un valor numèric, positiu o negatiu, sense cap decimal
    - reals: representa un valor numèric, positiu o negatiu, amb decimals

- no numèriques

- booleà: representa un valor de tipus lògic per tal d'establir la certesa o falsedat d'un estat o afirmació
    - caràcter, cadena de caràcters: representa una unitat fonamental de text usada en qualsevol alfabet, un nombre o un signe de puntuació o exclamació



# ANÀLISI. LES DADES. TIPUS DE DADES

- **Estructurades**

- internes

- estàtiques (registres, vectors, taules)
    - dinàmiques (l·listes, cues, piles, arbres)

- externes

- fitxers
    - bases de dades

# ANÀLISI. LES DADES. EXPRESSIONS

Una **expressió** és la representació de diversos **operands**, units entre ells mitjançant **operadors**, per tal de realitzar una acció sobre ells, ja sigui aritmètica o lògica.

Per exemple:

5.3 == '4'  
5 || 4.0  
!(40>25)  
10<20 && 40>25  
5 != 6

# ANÀLISI. LES DADES. OPERANDS

Una **variable** és una dada **emmagatzemada a la memòria** que **pot veure modificat el seu valor en qualsevol moment durant l'execució del programa.**

Una **constant** és un **tipus especial de variable** que té la particularitat que dins del codi del programa **el seu valor només pot ser llegit, però mai modificat.**

# ANÀLISI. LES DADES. OPERADORS

- ❏ **assignació:**  $x = y$

- ❏ **addició/resta/multiplicació/divisió/mòdul:**

  - ❏  $x = x + y$

  - ❏  $x = x - y$

  - ❏  $x = x * y$

  - ❏  $x = x / y$

  - ❏  $x = x \% y$

- ❏ **comparació:**

  - ❏  $==$

  - ❏  $!=$

  - ❏  $<, >$

  - ❏  $<=, >=$

- ❏ **aritmètics:**

  - ❏  $++$

  - ❏  $--$

  - ❏  $**$

# ANÀLISI. LES DADES. OPERACIONS I ORDRE DE PRECEDÈNCIA

Ordre	Operació	Operador
1	Parèntesi	()
2	Canvi de signe	-() !()
3	Operadors exponencials	^ **
4	Producte, divisió i mòdul	* (DIV) / (MOD) %
5	Suma i resta	+ -
6	Relacions de comparació (condicionals)	> < <= >=
7	Negació lògica	NOT (! unari)
8	Conjunció	AND (&&)
9	Disjunció	OR (  )

# ANÀLISI. LES DADES. ÀLGEBRA DE BOOLE

**Negació (NOT)**

A	$\neg A$
0	1
1	0

**Multiplicació  
lògica (AND)**

A	B	$A * B$
0	0	0
0	1	0
1	0	0
1	1	1

**Suma lògica  
(OR)**

A	B	$A + B$
0	0	0
0	1	1
1	0	1
1	1	1

# ANÀLISI. LES DADES. ÀLGEBRA DE BOOLE

Indica si són certes o falses les següents expressions:

<b>A</b>	<b>B</b>	<b>A == B</b>	<b>A &gt; B</b>	<b>A &lt; B</b>	<b>A &gt;= B</b>	<b>A &lt;= B</b>
4	3					
14	-2					
-78	34					
12	12					

# DADES. OPERACIONS I ORDRE DE PRECEDÈNCIA

`((3 + 4) == 7 ) && !(12.3 > 2.11) || ('a' == 'b')`

1. Parèntesi més interns  $\longrightarrow$  `((3 + 4) == 7 ) && !(12.3 > 2.11) || ('a' == 'b')`
2. Parèntesi més externs  $\longrightarrow$  `((7) == 7 ) && !(12.3 > 2.11) || ('a' == 'b')`
3. Negació  $\longrightarrow$  `(true) && !(true) || (false)`
4. Conjunció (multiplicació lògica)  $\longrightarrow$  `true && false || false`
5. Disjunció (suma lògica)  $\longrightarrow$  `false || false`



# ANÀLISI. LES DADES. INSTRUCCIONS

## ❑ Simples

- ❑ comentaris
- ❑ declaratives
- ❑ primitives
  - ❑ entrada
  - ❑ sortida
  - ❑ assignació
- ❑ de control
  - ❑ alternatives (condicionals)
    - ❑ simples
    - ❑ dobles
    - ❑ múltiples
  - ❑ repetitives (iteratives)

## ❑ Compostes (funcions)

# 3. ALGORISMES

Disseny de la solució

# DISSENY

- ❑ S'escriu de manera precisa les tasques a realitzar pel programa.
- ❑ S'utilitza la tècnica del disseny "top-down" (descendent), basada en el principi de "dividir i vèncer"

## **Disseny "top-down":**

1. Es planteja el problema utilitzant termes del mateix problema (nivell d'abstracció 1).
2. Es descompon en diversos sub-problemes, expressats també en termes del problema i intentant de fer-los de la manera que aquests siguin el més independents entre ells possible.
3. Es repeteix el pas 2 per a cada sub-problema tantes vegades com aquest sigui necessari, fins a arribar a una descripció del problema que utilitzi instruccions senzilles que puguin ser transformades de manera simple a codi en un llenguatge de programació.

# DISSENY. EXEMPLES

Un autobús viatja a 126 km/h per l'autopista durant 45 minuts. Quina distància ha recorregut en aquest temps?

## Descomposició del problema

<b>Nivell 1</b>	<b>1. Determinar la distància que ha recorregut en 45 minuts un autobús que viatja a 126 km/h per l'autopista.</b>
<b>Nivell 2</b>	<b>2.1. Determinar les dades necessàries. 2.2. Calcular la distància recorreguda. 2.3. Indicar a l'usuari el resultat obtingut.</b>
<b>Nivell 3</b>	<b>3.1.1. Assignar les dades de l'enunciat a les variables corresponents. 3.2.1. Convertir totes les unitats al Sistema Internacional. 3.2.2. Aplicar la fórmula <math>x_f = x_o + v \cdot t</math>, aïllant la <math>t</math>. 3.3.1. Mostrar el resultat en m.</b>

# DISSENY. EXEMPLES

A l'escola de cuina van cada dia X alumnes de matí i Y a la tarda. Quants alumnes van a l'escola cada dia?

## Descomposició del problema

<b>Nivell 1</b>	<b>1. Determinar el total d'alumnes que assisteixen al centre en 1 dia</b>
<b>Nivell 2</b>	<b>2.1. Determinar les dades necessàries. 2.2. Calcular el total d'alumnat 2.3. Indicar a l'usuari el resultat obtingut.</b>
<b>Nivell 3</b>	<b>3.1.1. Assignar les dades de l'enunciat a les variables corresponents. 3.2.1. Sumar el nombre d'alumnes de matí (X) més el nombre d'alumnes de tarda (Y) 3.3.1. Mostrar el resultat (indicant que és el total per 1 dia).</b>

# DISSENY. ESPECIFICACIÓ

Una **Precondició** és una condició que ha de satisfer-se just abans del començament de l'execució del codi.

Una **Postcondició** és una condició que ha de complir-se immediatament després de l'execució del codi.

**Pre:** Rep dos nombres enters (dividend i divisor) i el divisor ha de ser diferent de 0.

**Post:** Retorna un nombre real, resultat de dividir el dividend entre el divisor.

# DISSENY. ESPECIFICACIÓ

**Pseudocodi:** descripció informal d'alt nivell d'un algorisme.

//Pre: Rep un valor natural

function

constant

real PI = 3.141592

endconstant

var

integer radius

real area

endvar

read(radius)

area = PI\*(radius\*radius)

write(area)

endfunction

//Post: Retorna l'àrea calculada segons el valor rebut

declaració de constants

declaració de variables

instruccions

cos del  
programa

# DISSENY. ESPECIFICACIÓ

## Tipus de dades

- **integer**: per a representar valors numèrics, positius o negatius, sense cap decimal
- **real**: per a representar valors numèrics, positius o negatius, amb decimals
- **boolean**: per a representar valors de tipus lògic. Retornarà true o false
- **char**: per a representar una unitat fonamental de text usada en qualsevol alfabet, un nombre o un signe de puntuació o exclamació
- **string**: per a representar una cadena de caràcters (literals o textos)



# DISSENY. ESPECIFICACIÓ

//Pre: Rep un valor natural

```
function
    constant
        string MSG_1 = "Continua així!"
        string MSG_2 = "T'has d'esforçar una mica més"
        string MSG_3 = "A treballar més!"
    endconstant
    var
        integer mark
    endvar
    read(mark)
    if mark >= 5 then
        write(MSG_1)
    else
        if mark >=2 && mark <=5 then
            write(MSG_2)
        else
            write(MSG_3)
        endif
    endif
endfunction
```

//Post: Retorna l'àrea calculada segons el valor rebut

# ESPECIFICACIÓ. ESTRUCTURES DE REPETICIÓ

Els bucles o iteracions es repeteixen un cert nombre de vegades dins d'un algorisme (segons la condició de repetició).

**while** (mentre): les instruccions s'executaran mentre es compleixi la condició (mentre aquesta sigui certa)

```
function
  var
    integer count = 1
  endvar
  while count <= 10 do
    write(count)
    count = count+1
  endwhile
endfunction
```

# ESPECIFICACIÓ. ESTRUCTURES DE REPETICIÓ

**do-while** (repetir fins): s'executaran les instruccions una primera vegada i després s'avaluarà si es compleix la condició

```
function
  var
    integer count = 1
  endvar
  do
    write(count)
    count = count + 1
  while count <= 10
endfunction
```

# ESPECIFICACIÓ. ESTRUCTURES DE REPETICIÓ

**for** (per): s'executaran les instruccions tantes vegades com indica el comptador

```
function
  var
    integer count
  endvar
  for count = 1 to 10 do
    write(count)
    count = count + 1
  endfor
endfunction
```

no és necessari indicar-ho  
quan l'increment és 1

# ESPECIFICACIÓ. DADES ESTRUCTURADES

Un tipus de **variable estructurada** és aquella formada per **diferents valors de tipus simples i/o estructurats**.

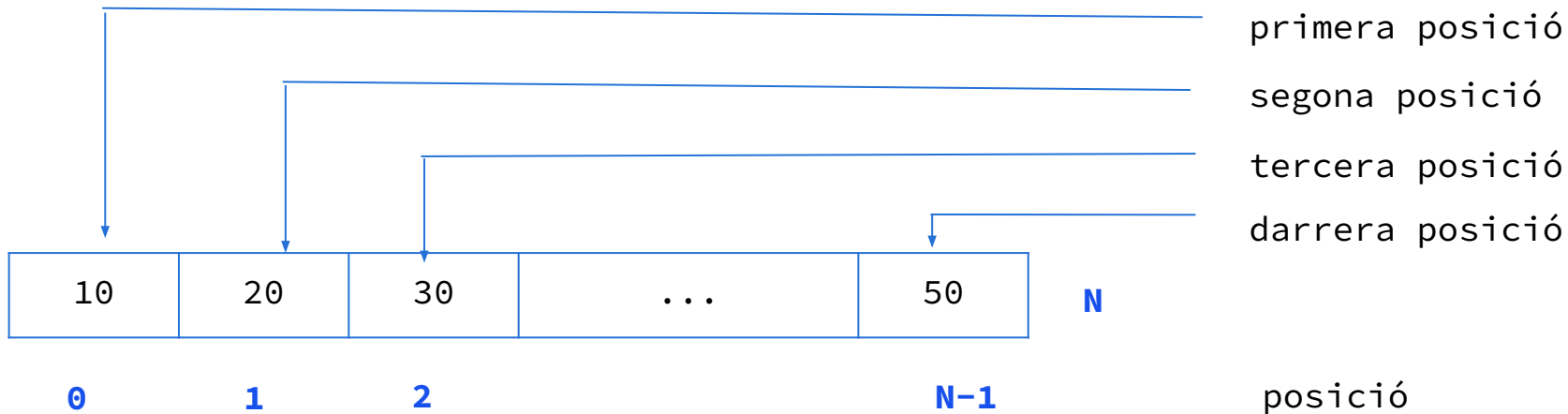
Disposem de diferents tipus de variables estructurades (internes, externes). Dins les internes, tenim dos tipus:

- ❑ arrays
- ❑ matrius

Ambdues són estàtiques, ja que a l'hora de declarar-les, s'assigna una mida i aquesta no pot variar.

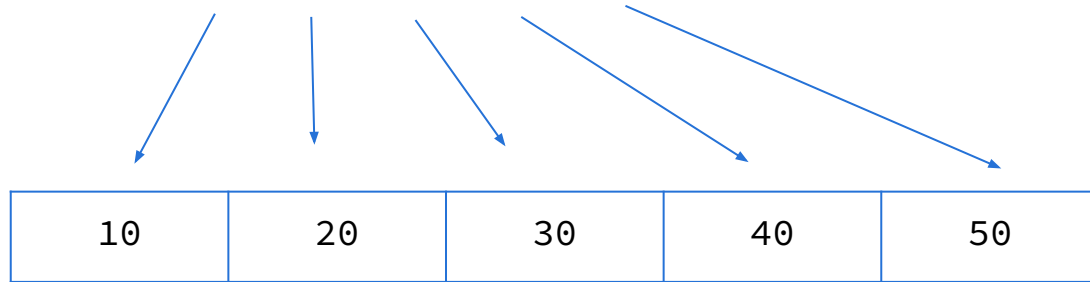
# ESPECIFICACIÓ. DADES COMPOSTES. ARRAYS

L'array (vector) permet **emmagatzemar** en forma de **seqüència**, una **quantitat predeterminada de valors pertanyents al mateix tipus de dades**.



# ESPECIFICACIÓ. DADES COMPOSTES. ARRAYS

```
integer arrayInt[5] //sense inicialitzar  
integer arrayInt[5]= {10, 20, 30, 40, 50}
```



# ESPECIFICACIÓ. DADES COMPOSTES. ARRAYS. EXEMPLE (I)

Algorisme que emmagatzema 6 valors naturals introduïts per teclat en un array:

```
function
  constant
    integer SIZE = 6
    string MSG_1 = "Introdueix un nombre"
  endconstant
  var
    integer arrayInt[SIZE], i=0
  endvar
  while i<SIZE do
    write(MSG_1)
    read(arrayInt[i])
    i=i+1
  endwhile
endfunction
```



# ESPECIFICACIÓ. DADES COMPOSTES. ARRAYS. EXEMPLE (II)

Algorisme que mostra 6 valors naturals introduïts per teclat dins un array:

```
function
  constant
    integer SIZE = 6
    string MSG_1 = "Introdueix un nombre"
  endconstant
  var
    integer arrayInt[SIZE], i=0
  endvar
  while i<SIZE do
    write(MSG_1)
    read(arrayInt[i])
    i=i+1
  endwhile
  write(arrayInt)
endfunction
```

així no es mostren els  
valors de l'array!

# ESPECIFICACIÓ. DADES COMPOSTES. ARRAYS. EXEMPLE (II)

Algorisme que mostra 6 valors naturals introduïts per teclat en un array:

```
function
  constant
    integer SIZE = 6
    string MSG_1 = "Introdueix un nombre"
  endconstant
  var
    integer arrayInt[SIZE], i=0
  endvar
  while i<SIZE do
    write(MSG_1)
    read(arrayInt[i])
    i=i+1
  endwhile
  i = 0
  while i<SIZE do
    write(arrayInt[i])
    i=i+1
  endwhile
endfunction
```

així sí!!!

# ESPECIFICACIÓ. DADES COMPOSTES. ARRAYS. EXEMPLE (II)

Algorisme que mostra 6 valors naturals introduïts per teclat en un array:

```
function
  constant
    integer SIZE = 6
    string MSG_1 = "Introdueix un nombre"
  endconstant
  var
    integer arrayInt[SIZE], i=0
  endvar
  while i<SIZE do
    write(MSG_1)
    read(arrayInt[i])
    i=i+1
  endwhile
  while i<SIZE do
    write(arrayInt[i])
    i=i+1
  endwhile
endfunction
```

**Iteracions amb while**

# ESPECIFICACIÓ. DADES COMPOSTES. ARRAYS. EXEMPLE (III)

Algorisme que mostra 6 valors naturals introduïts per teclat en un array:

```
function
  constant
    integer SIZE = 6
    string MSG_1 = "Introdueix un nombre"
  endconstant
  var
    integer arrayInt[SIZE], i
  endvar
  for i=0 to SIZE do
    write(MSG_1)
    read(arrayInt[i])
    i = i + 1
  endfor
  for i=0 to SIZE do
    write(arrayInt[i])
    i = i + 1
  endfor
endfunction
```

**Iteracions amb for**

# ESPECIFICACIÓ. DADES COMPOSTES. MATRIUS

Una **matriu** permet emmagatzemar, en forma de **taula**, una **quantitat predeterminada de valors pertanyents al mateix tipus de dades**.

```
integer matInt[2][5] //sense inicialitzar  
integer matInt[2][5] = {{1, 2, 3, 4, 5},{4,5,6,7,8}}
```

1	2	3	4	5
4	5	6	7	8

# ESPECIFICACIÓ. DADES COMPOSTES. MATRIUS. EXEMPLE (I)

Algorisme que emmagatzema les notes dels 6 mòduls professionals de 4 alumnes per teclat:

```
function
  constant
    integer ROWS = 4, COLS = 6
  endconstant
  var
    integer notes[ROWS][COLS], i, j
  endvar
  for i=0 to ROWS do
    for j=0 to COLS do
      read(notes[i][j])
      j = j +1
    endfor
    i = i +1
  endfor
endfunction
```

# ESPECIFICACIÓ. DADES COMPOSTES. MATRIUS. EXEMPLE (II)

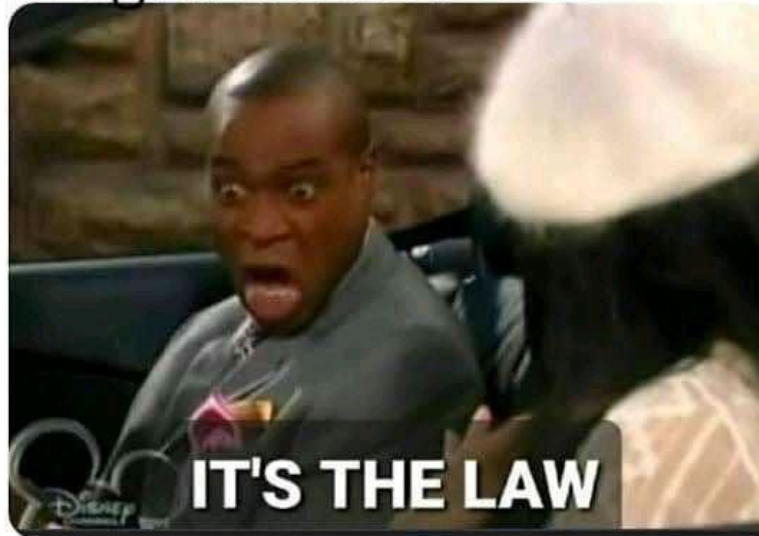
Algorisme que mostra les notes dels 6 mòduls professionals de 4 alumnes per teclat:

```
function
  constant
    integer ROWS = 4, COLS = 6
  endconstant
  var
    integer notes[ROWS][COLS]={ {1,2,3,4,6,5},
                                   {4,5,6,7,8,9}, {6,5,6,7,8,5}, {8,5,7,7,9,8}}, i, j
  endvar
  for i=0 to ROWS do
    for j=0 to COLS do
      write(notes[i][j])
      j = j +1
    endfor
    i = i +1
  endfor
endfunction
```

# ESPECIFICACIÓ. DADES COMPOSTES. MATRIUS

Others : why do you always use i,j variabes in loops?

Programmers :

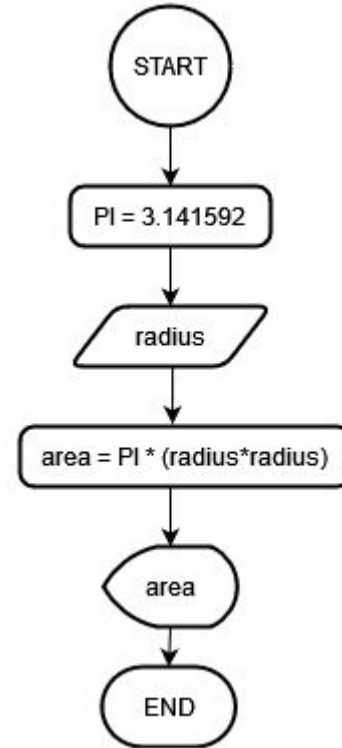
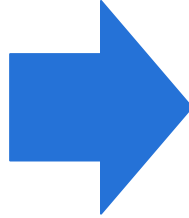








# ESPECIFICACIÓ. DIAGRAMES DE FLUX

**Diagrama de flux:** representació gràfica d'un algorisme.

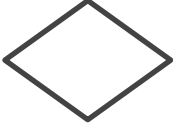

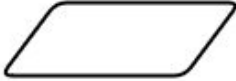


```
function
  constant
    real PI = 3.141592
  endconstant
  var
    integer radius
    real area
  endvar
  read(radius)
  area = PI * (radius*radius)
  write(area)
endfunction
```



# DISSENY. ESPECIFICACIÓ

Símbol	Nom	Descripció
	Flowline	Mostra l'ordre de funcionament del procés
	Start	Indica l'inici d'un programa o sub-procés
	Terminator	Indica la finalització d'un programa o sub-procés
	Process	Representa un conjunt d'operacions que canvien el valor, la forma o la ubicació de les dades

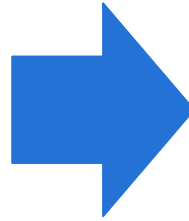
# DISSENY. ESPECIFICACIÓ

Símbol	Nom	Descripció
	Decision	Mostra una operació condicional que determina quina de les dues rutes del programa haurà de realitzar
	Repeating	Mostra una operació repetitiva que es realitzarà mentre no s'acompleixi la condició de sortida de la iteració
	Input	Indica el procés d'entrada de dades (introducció de dades per teclat)
	Output	Indica el procés de sortida de dades (mostra de resultats per pantalla)
	Sub-program	Indica l'existència d'un conjunt d'instruccions simples, agrupades sota un nom.

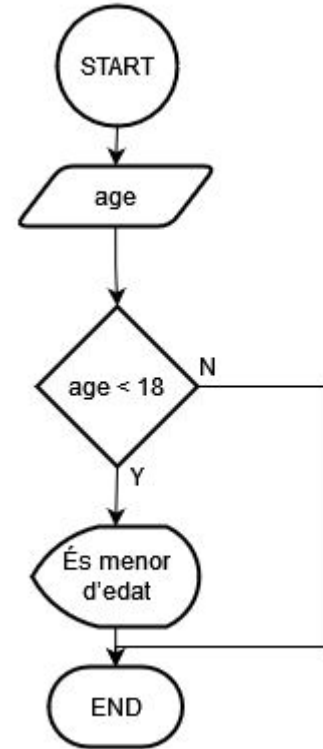
# ESPECIFICACIÓ. ESTRUCTURES DE SELECCIÓ

Controlen la seqüència d'execució d'altres instruccions, segons una determinada condició. Poden ser simples, dobles o múltiples:

```
function
  var
    integer age
  endvar
  read(age)
  if age < 18 then
    write("És menor d'edat")
  endif
endfunction
```



**selecció simple**



# ESPECIFICACIÓ. ESTRUCTURES DE SELECCIÓ

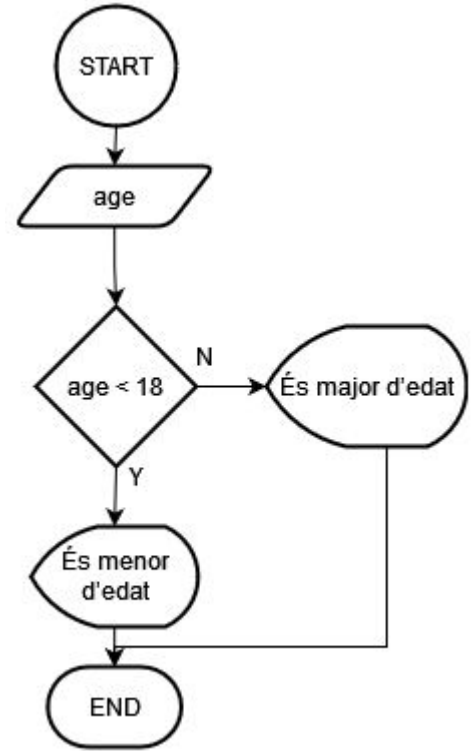
```
function
  var
    integer age
  endvar
  read(age)
  if age < 18 then
    write("És menor")
  else
    write("És major d'edat")
  endif
endfunction
```

## Compte!

En una decisió, si no es compleix  
la condició "sí", definir la  
condició

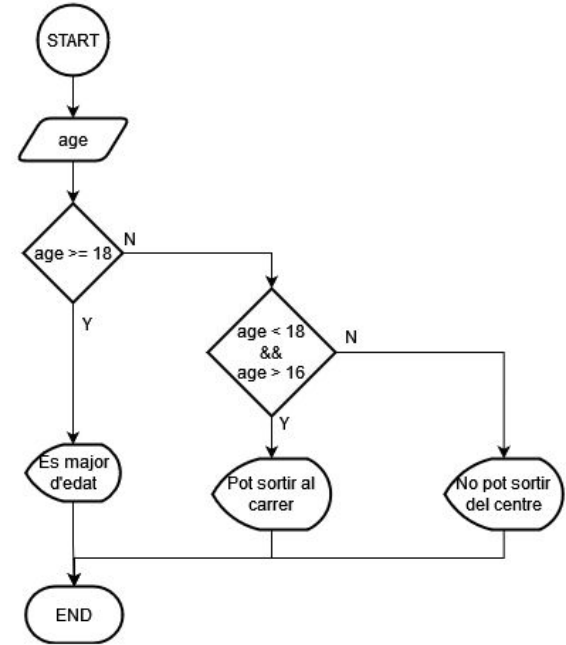



**selecció doble**



# ESPECIFICACIÓ. ESTRUCTURES DE SELECCIÓ

```
function
  var
    integer age
  endvar
  read(age)
  if age >= 18 then
    write("És major d'edat")
  else
    if age < 18 AND age > 16 then
      write("Pot sortir al carrer")
    else
      write("No pot sortir del centre")
    endif
  endif
endfunction
```



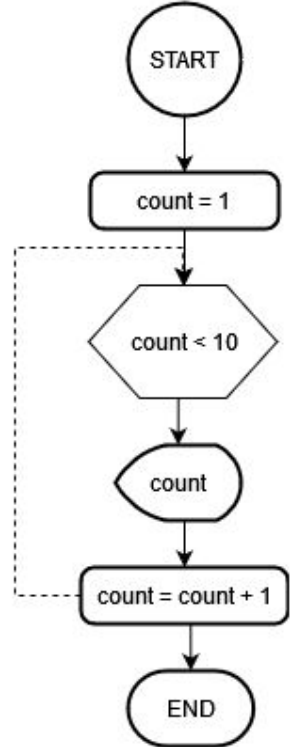
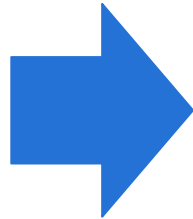
**selecció múltiple**

# ESPECIFICACIÓ. ESTRUCTURES DE REPETICIÓ

Els bucles o iteracions es repeteixen un cert nombre de vegades dins d'un algorisme (segons la condició de repetició).

**while** (mentre): les operacions s'executaran mentre es compleixi la condició (sigui certa):

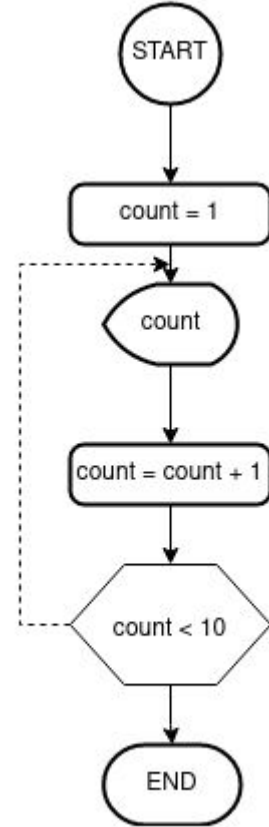
```
function
  var
    integer count = 1
  endvar
  while count < 10 do
    write(count)
    count = count+1
  endwhile
endfunction
```



# ESPECIFICACIÓ. ESTRUCTURES DE REPETICIÓ

**do-while** (repetir fins): s'executaran les operacions una primera vegada i després s'avaluarà si es compleix la condició:

```
function
  var
    integer count = 1
  endvar
  do
    write(count)
    count = count+1
  while count < 10
endfunction
```

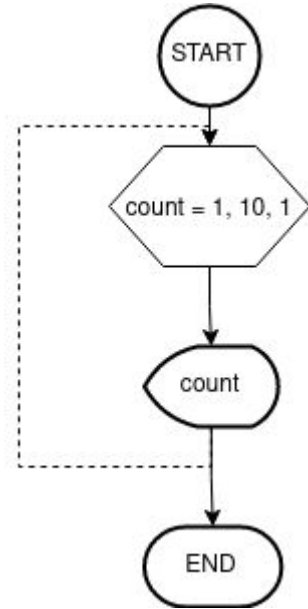




# ESPECIFICACIÓ. ESTRUCTURES DE REPETICIÓ

**for** (per): s'executarà tantes vegades com indica el comptador:

```
function
  var
    integer count
  endvar
  for count = 1 to 10 do
    write(count)
    count = count + 1
  endfor
endfunction
```



# DISSENY. ESPECIFICACIÓ

## Diagrames de Nassi-Shneiderman:

- representació gràfica del disseny per a la programació estructurada
- desenvolupats el 1972 per Isaac Nassi i Ben Shneiderman
- anomenats també estructogrames, diagrames N-S o diagrames de Chapin (que els va perfeccionar)
- basats en la representació de les diferents instruccions en forma de caixes, dins de les quals es descriuen les accions.

# DISSENY. ESPECIFICACIÓ

Descripció d'una tasca:

## Instruccions alternatives

condició
si
accions a realitzar si es compleix la condició

simples

condició	
no	si
accions a realitzar si no es compleix la condició	accions a realitzar si es compleix la condició

dobles

# DISSENY. ESPECIFICACIÓ

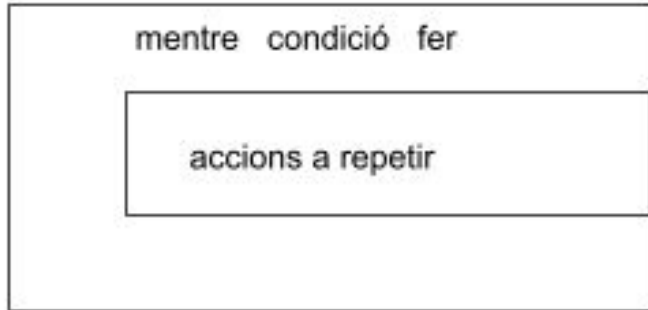
## Instruccions alternatives

variable=								
valor1	valor2	valor3	valor4	valor5	valor6	valor7	...	Alternati va
acció	acció	acció	acció	acció	acció	acció		acció

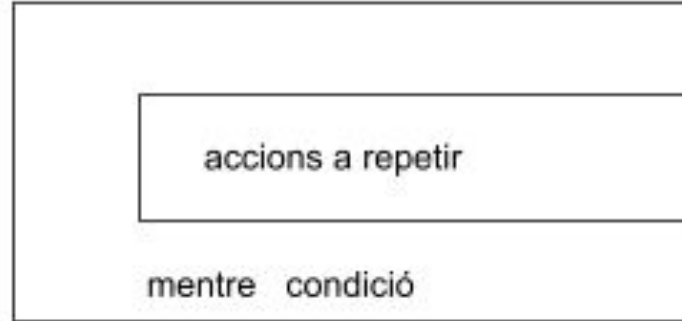
múltiples

# DISSENY. ESPECIFICACIÓ

## Instruccions repetitives



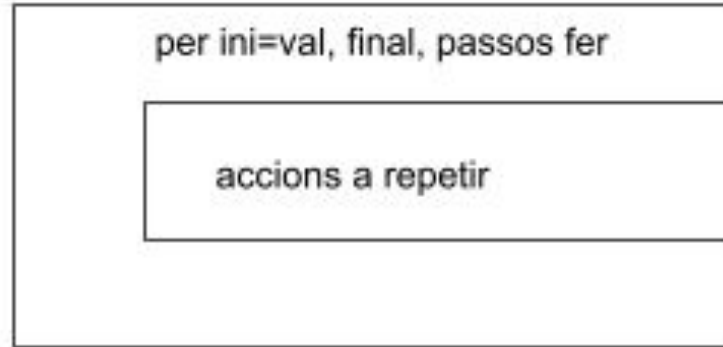
while (mentre)



do...while (repetir fins)

# DISSENY. ESPECIFICACIÓ

## Instruccions repetitives

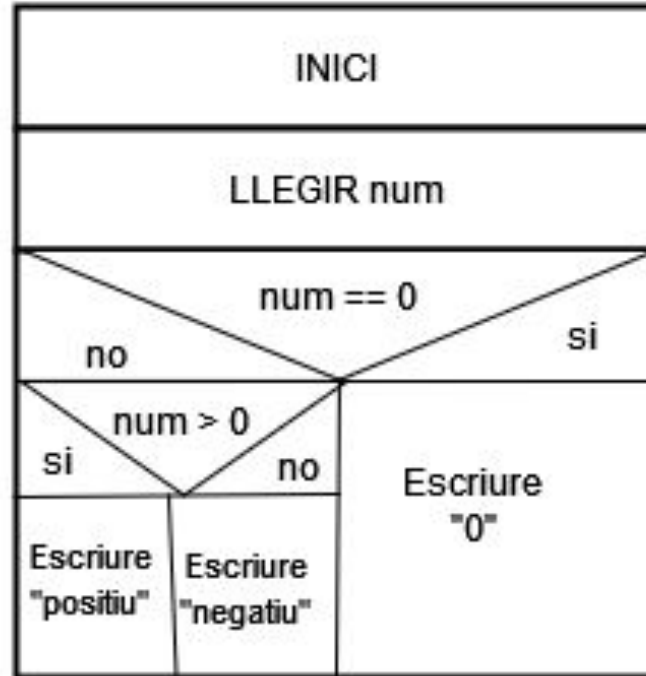


for (per)

# DISSENY. ESPECIFICACIÓ

## Exemple

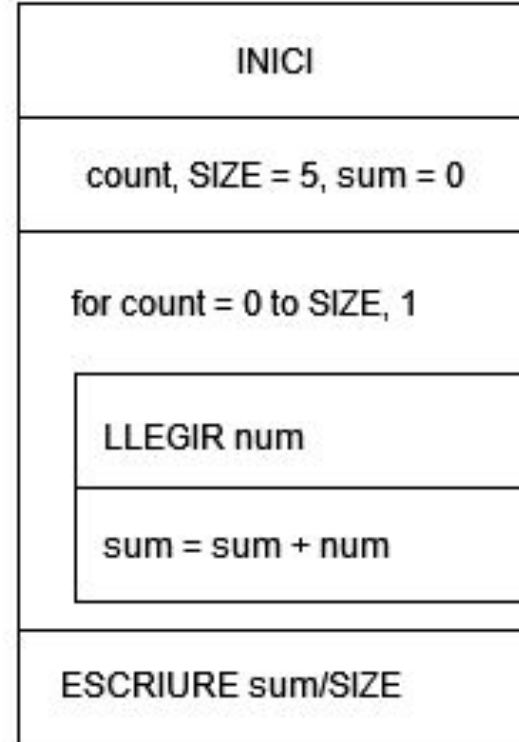
Donat un nombre introduït per teclat, mostrar per pantalla si és positiu, negatiu o 0.



# DISSENY. ESPECIFICACIÓ

## Exemple

Calcular la mitjana de 5  
nombres naturals  
introduïts per teclat.





# ESPECIFICACIÓ. JOCS DE PROVES

A l'hora de dissenyar un algorisme, cal realitzar les proves necessàries per a avaluar la seva validesa.

Els jocs de proves documenten els casos provats i els resultats obtinguts, per tal de corregir i millorar els programes.

# ESPECIFICACIÓ. JOCS DE PROVES

```
function
  constant
    string MSG_1 = "Introdueix un nombre natural"
    string MSG_2 = "El seu valor en binari és"
  endconstant
  var
    integer num, desp = 1, binNum = 0
  endvar
  write(MSG_1)
  read(num)
  while num > 0 do
    binNum = binNum+(num%2)*desp
    desp = desp*10
    num = num/2
  endwhile
  write(MSG_2, binaryNum)
endfunction
```

# ESPECIFICACIÓ. JOCS DE PROVES

# Instrucció	# Iteració	variables
1	-	num = - desp = 1 binNum = 0
2	-	num = 4 desp = 1 binNum = 0
3	1	num = 2 desp = 10 binNum = 0
3	2	num = 1 desp = 100 binNum = 0
3	3	num = 0 desp = 1000 binNum = 100
4	-	num = 0 desp = 1000 binNum = 100

# 5. ALGORISMES

**Implementació en Java**

# CARACTERÍSTICAS PRINCIPALES

Java is one of the most popular and widely used programming language.

- Java has been one of the most popular programming language for many years.
- Java is Object Oriented. However it is not considered as pure object oriented as it provides support for primitive data types (like int, char, etc)
- The Java codes are first compiled into byte code (machine independent code). Then the byte code is run on **Java Virtual Machine (JVM)** regardless of the underlying architecture.
- Java syntax is similar to C/C++. But Java does not provide low level programming functionalities like pointers. Also, Java codes are always written in the form of classes and objects.

# CARACTERÍSTICAS PRINCIPALES

- Java is used in all kind of applications like Mobile Applications (Android is Java based), desktop applications, web applications, client server applications, enterprise applications and many more.
- When compared with C++, Java codes are generally more maintainable because Java does not allow many things which may lead bad/inefficient programming if used incorrectly. For example, non-primitives are always references in Java. So we cannot pass large objects (like we can do in C++) to functions, we always pass references in Java. One more example, since there are no pointers, bad memory access is also not possible.
- When compared with Python, Java kind of fits between C++ and Python. The programs written in Java typically run faster than corresponding Python programs and slower than C++. Like C++, Java does static type checking, but Python does not.

# ENTORNS INTEGRATS DE DESENVOLUPAMENT (IDE)



**Java**



**Eclipse IDE**



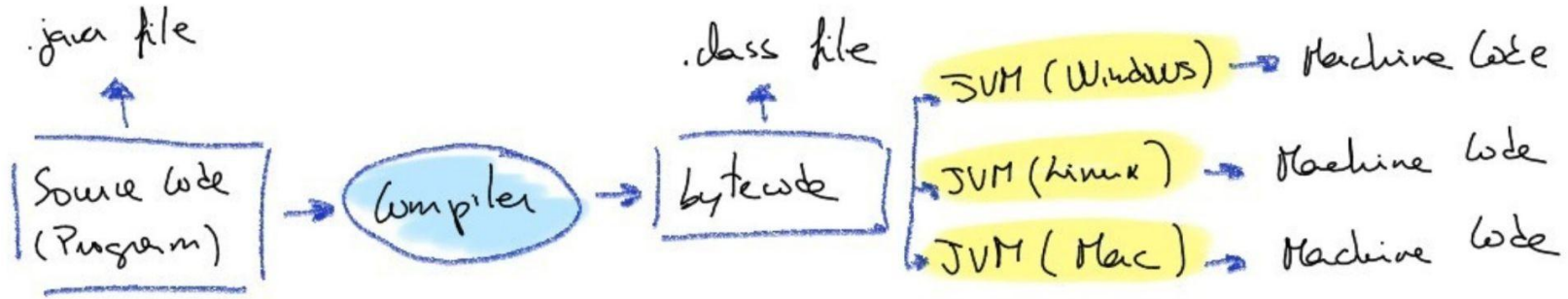
**IntelliJ**



**Netbeans**

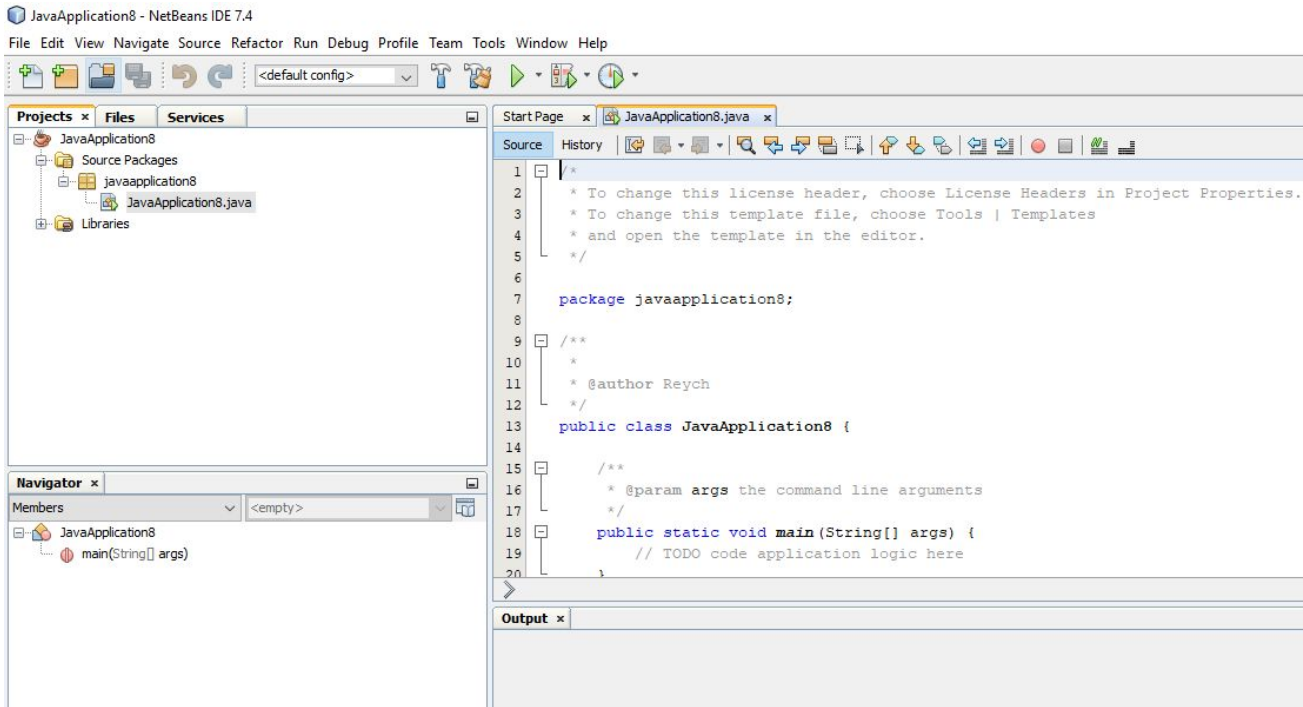
En aquest curs, treballarem amb l'IDE Netbeans, ja que és l'utilitzat en la [formació oficial d'Oracle](#) (per a l'obtenció dels certificats de desenvolupador Java).

# JAVA VIRTUAL MACHINE (JVM)





# ENTORNS INTEGRATS DE DESENVOLUPAMENT (IDE)



# ENTORNS INTEGRATS DE DESENVOLUPAMENT (IDE)



# PACKAGES

Java disposa d'un mecanisme per encapsular les implementacions del programa (grups de classes, subpaquets i interfícies), anomenat package. Els packages (paquets) són utilitzats per:

- prevenir conflictes de nomenclatura: per exemple hi pot haver dues classes amb el nom Empleat en dos paquets diferents: college.staff.cse.employee i college.staff.ee.employee
- permetre la cerca, la localització i l'ús de classes, interfícies, les enumeracions i les anotacions fàcilment
- proporcionar l'accés controlat: protected i default tenen control d'accés a nivell de paquet. Un membre protected és accessible per classes i les seves subclasses del mateix paquet. Un membre default (sense qualsevol especificador d'accés) és accessible per classes només del mateix paquet.
- encapsulació de dades (o ofuscació de dades)

**Totes les classes relacionades han d'estar dins del mateix paquet.**

En aquest MP, treballarem amb la nomenclatura de packages i amb un únic codi font per projecte.

# HELLO WORLD!

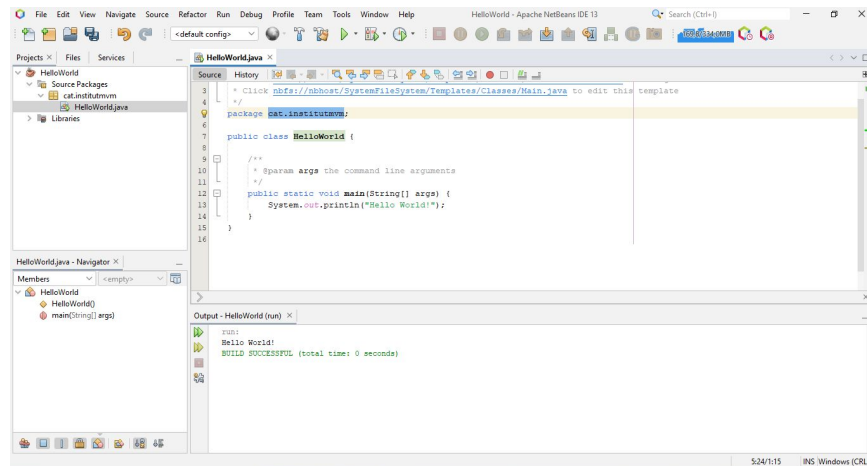
Primer programa en Java:

```
package cat.institutmvm;
```

```
public class HelloWorld {
```

```
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }
```

```
}
```



# COMENTARIS

```
package cat.institutmvm;

public class HelloWorld{
    //comentari en 1 línia
    public static void main(String[] args) {
        System.out.println("Hello World");
        /*
         * dues línies de comentari
         */
    }
}
```

```
5 package cat.institutmvm;
6
7 public class HelloWorld {
8
9     /**
10      * @param args the command line arguments
11      */
12     public static void main(String[] args) {
13         //comentari en 1 línia
14         System.out.println("Hello World!");
15         /*
16          * dues línies de comentari
17          */
18     }
19 }
```

# CONSTANTS

Les best practices indiquem que les constants s'han de declarar en majúscules i, si estan formades per dues o més paraules, separades per guió baix (underscore):

```
private static final tipus IDENTIFICADOR_VARIABLE = valor;
```

```
public class CalculaPreu {  
    private static final double IVA = 0.21;  
  
    public static void main(String[] args) {  
        int preuInicial = 25;  
        double preuFinal = 0.0;  
        preuFinal = preuInicial + (preuInicial * IVA);  
        System.out.println(preuFinal);  
    }  
}
```

# CONSTANTS

Les best practices indiquem que les constants s'han de declarar en majúscules i, si estan formades per dues o més paraules, separades per guió baix (underscore):

```
11 public class CalculaPreu {  
12     private static final double IVA = 0.21;  
13  
14     public static void main(String[] args) {  
15         int preuInicial = 25;  
16         double preuFinal = 0.0;  
17         preuFinal = preuInicial + (preuInicial * IVA);  
18         System.out.println(preuFinal);  
19     }  
20 }
```

# NOMENCLATURA DE LES VARIABLES

El nom d'una variable es una etiqueta que ens permet accedir a l'espai de memòria assignat. Tot i que el nostre programa s'executarà si aquestes tenen qualsevol valor (excepte excepcions), és recomanable seguir les següents convencions:

- el nom ha de tenir un significat (context)
- s'ha d'utilitzar la nomenclatura lowerCamelCase
- els noms habituals de les variables temporals són i, j, k, m, i n per a enters; c, d, i e per a caràcters.
- Java és case-sensitive (no és la mateixa variable JAVA que Java ni que Java)
- pot ser qualsevol identificador legal - una seqüència de longitud il·limitada de lletres i díigits Unicode, començant amb una lletra, el signe del dòlar "\$", o el caràcter guió baix " \_ "
- no es poden utilitzar paraules reservades (*for, int, case, ...*) pròpies del llenguatge

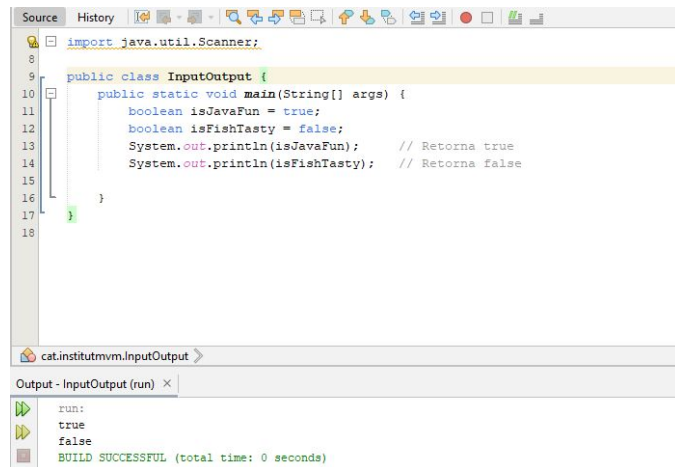


# TIPUS DE DADES SIMPLS PREDEFINIDES

TYPE	SIZE	RANGE	NOTES
int	4 bytes	−2,147,483,648 to 2,147,483,647 (just over 2 billion)	The wrapper type is Integer. Use BigInteger for arbitrary precision integers
short	2 bytes	−32,768 to 32,767	
long	8 bytes	−9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	Literals end with L (e.g. 1L)
byte	1 byte	−128 to 127	Note that the range is not 0 to 255
float	4 bytes	approximately $\pm 3.40282347\text{E}+38\text{F}$ (6–7 significant decimal digits)	Literals end with F (e.g. 0.5F)
double	8 bytes	approximately $\pm 1.79769313486231570\text{E}+308$ (15 significant decimal digits)	Use BigDecimal for arbitrary precision floating-point numbers
char	2 bytes	\u0000 to \uFFFF	The wrapper type is Character. Unicode characters > U+FFFF require two char values
boolean		true or false	

# TIPUS DE DADES. BOOLEANS

```
boolean isJavaFun = true;  
boolean isFishTasty = false;  
System.out.println(isJavaFun);    // Retorna true  
System.out.println(isFishTasty);  // Retorna false
```



The screenshot shows an IDE window with a Java source file. The code defines a class `InputOutput` with a `main` method that prints the values of `isJavaFun` and `isFishTasty`. Below the source code, the output window shows the results of the program execution: `true` and `false`. The status bar at the bottom indicates that the build was successful.

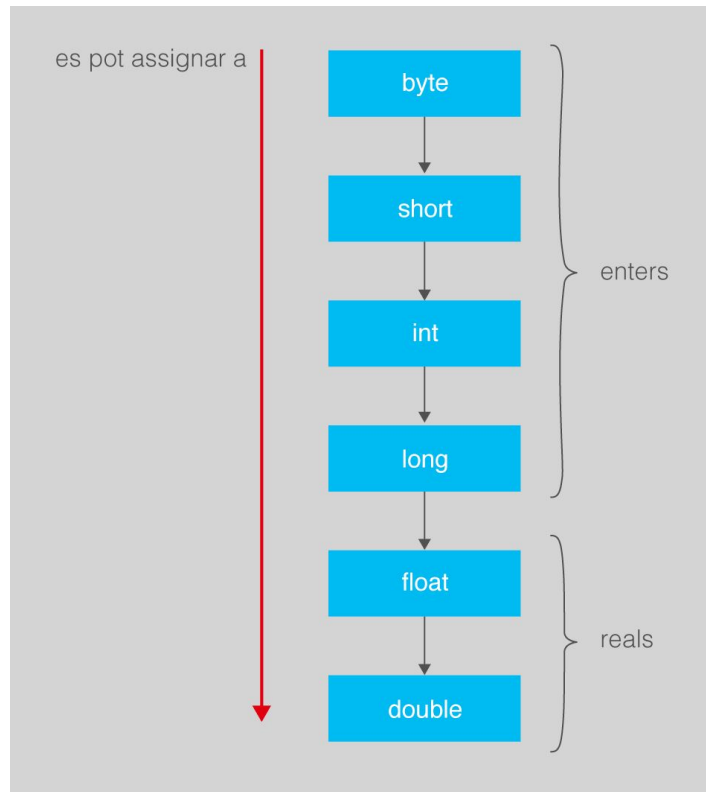
```
Source History  
import java.util.Scanner;  
8  
9 public class InputOutput {  
10     public static void main(String[] args) {  
11         boolean isJavaFun = true;  
12         boolean isFishTasty = false;  
13         System.out.println(isJavaFun);    // Retorna true  
14         System.out.println(isFishTasty);  // Retorna false  
15     }  
16 }  
17  
18  
cat.institutmvm.InputOutput  
Output - InputOutput (run) X  
run:  
true  
false  
BUILD SUCCESSFUL (total time: 0 seconds)
```

# CONVERSIÓ DE TIPUS (CASTING)

El casting ens permet fer una conversió explícita d'un tipus de dades a una altra, sempre que aquests siguin compatibles.

Hi ha dos tipus de conversions:

- explícites
- implícites



# CONVERSIÓ DE TIPUS (CASTING)

## Explícita

```
identificador = (paraulaClauTipus)expressió;
```

```
double realLlarg = 300.0;  
long enterLlarg = (long)realLlarg;
```

## Implícita

```
float real = 100;  
double doble = real;  
System.out.println(doble);
```

# LECTURA DE DADES I IMPRESSIÓ PER PANTALLA

```
package cat.institutmvm;

public class InputOutput{
    private static final String MSG_1 = "Introdueix codi: ";

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println(MSG_1);
        int code = sc.nextInt();
        System.out.println("Codi: "+ code);
        sc.close(); //és recomanable tancar el stream
    }
}
```

[Exemple amb diversos tipus de dades](#)

# SEQÜÈNCIES D'ESCAPAMENT

Seqüència d'escapament	Descripció
\t	Insereix una tabulació
\b	Insereix un backspace
\n	Insereix un salt de línia
\r	Insereix un retorn de carro
\'	Insereix el caràcter de cometa simple
\"	Insereix el caràcter de cometes dobles
\\	Insereix el caràcter de contrabarra (backslash)

# LECTURA DE DADES (DES DE TECLAT)

Mètode	Descripció
<code>nextBoolean()</code>	Llegeix un valor de tipus booleà
<code>next()</code>	Llegeix un valor de tipus String fins al primer “”
<code>nextLine()</code>	Llegeix un valor de tipus String
<code>nextByte()</code>	Llegeix un valor de tipus byte
<code>nextShort()</code>	Llegeix un valor de tipus short
<code>nextInt()</code>	Llegeix un valor de tipus int des
<code>nextLong()</code>	Llegeix un valor de tipus long
<code>nextFloat()</code>	Llegeix un valor de tipus float
<code>nextDouble()</code>	Llegeix un valor de tipus double

# OPERACIONS ARITMÈTIQUES

Operador	Acció	Exemple
+	Addició	$x+y$
-	Sostracció	$x-y$
*	Multiplicació	$x*y$
/	Divisió entera	$x/y$
%	Mòdul	$x\%y$



# OPERACIONS ARITMÈTIQUES

Operador	Acció		Exemple
++	x++	Primer assigna i després incrementa	<pre>int r = 6; System.out.println("r=: " + r++); System.out.println("r=: " + r);</pre>
++	++x	Primer incrementa i després assigna	<pre>int r = 6; System.out.println("r=: " + ++r);</pre>
--	x--	Primer assigna i després decrementa	<pre>int x = 6; System.out.println("x=: " + x--); System.out.println("x=: " + x);</pre>
--	--x	Primer decrementa i després assigna	<pre>int x = 6; System.out.println("p=: " + --x);</pre>

# OPERACIONS ARITMÈTIQUES COMPOSTES

Operador	Operació	És equivalent a
<code>+=</code>	<code>x += y</code>	<code>x = x + y</code>
<code>-=</code>	<code>x -= y</code>	<code>x = x - y</code>
<code>*=</code>	<code>x *= y</code>	<code>x = x * y</code>
<code>/=</code>	<code>x /= y</code>	<code>x = x / y</code>

# OPERADORS DE COMPARACIÓ (RELACIÓ) I OPERADORS LÒGICS

Operador	Signe
<	<
>	>
>=	>=
<=	<=
==	==
!=	!=

Operador	Signe
AND	&&
OR	
NOT	!

# ESTRUCTURES DE SELECCIÓ (CONDITIONALS)

## Selecció simple

```
if(a>b){  
    System.out.println("A és més gran");  
}
```

## Selecció múltiple

```
if(a>b){  
    System.out.println("A és més gran");  
}  
else {  
    if(a<b){  
        System.out.println("B és més gran");  
    }  
    else{  
        System.out.println("Són iguals");  
    }  
}
```

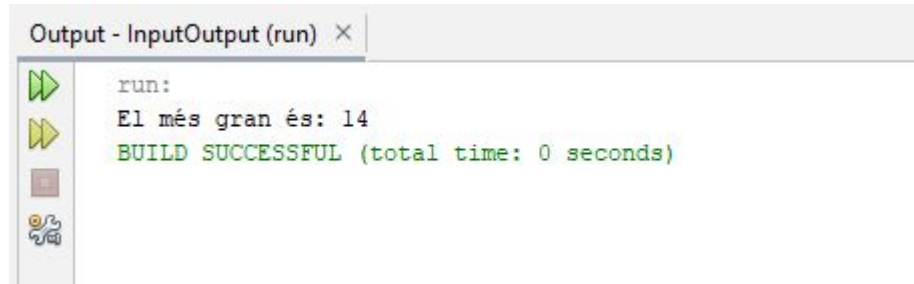
## Selecció doble

```
if(a>b){  
    System.out.println("A és més gran");  
}  
else {  
    System.out.println("B és més gran");  
}
```

# ESTRUCTURES DE SELECCIÓ (CONDITIONALS).

## OPERADOR TERNARI

```
int a=3, b=14;  
System.out.println("El més gran és: " + ((a>b)? a:b));
```



```
Output - InputOutput (run) ×  
run:  
El més gran és: 14  
BUILD SUCCESSFUL (total time: 0 seconds)
```

# ESTRUCTURES DE SELECCIÓ (CONDITIONALS). SWITCH-CASE

```
int numMonth;  
Scanner sc = new Scanner(System.in);  
System.out.println("Introdueix un número: ");  
numMonth = sc.nextInt();  
switch (numMonth){  
    case 1: System.out.println("Gener");  
            break;  
    case 2: System.out.println("Febrer");  
            break;  
    case 3: System.out.println("Març");  
            break;  
    default:  
        System.out.println("Error"); //opcional  
}
```

# ESTRUCTURES DE SELECCIÓ (CONDITIONALS). SWITCH-CASE

```
int number=5;
switch (number){
    case 1:
    case 2:
    case 3:
        System.out.println("One, Two, or Three.");
        break;
    case 4:
    case 5:
    case 6:
        System.out.println("Four, Five, or Six.");
        break;
    default:
        System.out.println("Greater than Six"); //optional
}
```

# ESTRUCTURES DE SELECCIÓ (CONDITIONALS). SWITCH-CASE

```
int numMonth;  
Scanner sc = new Scanner(System.in);  
System.out.println("Introdueix un número: ");  
numMonth = sc.nextInt();  
switch (numMonth){  
    case 1 -> System.out.println("Gener");  
    case 2 -> System.out.println("Febrer");  
    case 3 -> System.out.println("Març");  
    default -> System.out.println("Error"); //opcional  
}
```



# ESTRUCTURES DE SELECCIÓ. CONTROL D'ERRORS

Abans de llegir qualsevol dada introduïda pel teclat, podem establir si la dada que tot just s'ha escrit per teclat és d'un tipus concret o no.

```
package cat.institutmvm;

import java.util.Scanner;

public class HelloWorld {
    public static void main(String[] args) {
        float num;
        boolean tipusCorrecte;
        Scanner sc = new Scanner(System.in);
        tipusCorrecte = sc.hasNextFloat();
        if (tipusCorrecte) {
            num = sc.nextFloat();
            System.out.println(num);
        }
        else {
            System.out.println("El tipus no és correcte");
        }
    }
}
```

# ESTRUCTURES DE SELECCIÓ. CONTROL D'ERRORS

Mètode	Descripció
<code>hasNextBoolean()</code>	Valida si el valor rebut és de tipus booleà
<code>hasNextByte()</code>	Valida si el valor rebut és de tipus byte
<code>hasNextShort()</code>	Valida si el valor rebut és de tipus byte
<code>hasNextInt()</code>	Valida si el valor rebut és de tipus int
<code>hasNextLong()</code>	Valida si el valor rebut és de tipus long
<code>hasNextFloat()</code>	Valida si el valor rebut és de tipus float
<code>hasNextDouble()</code>	Valida si el valor rebut és de tipus double
<code>hasNext()</code>	Valida si el valor rebut és un text (caràcter o cadena)

# ESTRUCTURES DE REPETICIÓ (ITERACIONS)

## WHILE

```
int i=0;
while(i<10){
    System.out.println("Valor "+i);
    i++; //i=i+1
}
```

## DO...WHILE

```
int i=0;
do{
    System.out.println("Valor "+i);
    i++;
}while(i<10);
```

## FOR

```
int i;
for(i=0;i<3;i++){
    System.out.println("Versió " + i);
}
```

# DADES COMPOSTES. ARRAYS

L'array permet **emmagatzemar** en forma de **seqüència**, una **quantitat predeterminada de valors pertanyents al mateix tipus de dades**.

```
paraulaClauTipus[] identificadorVariable = new paraulaClauTipus[mida];
```

```
//declaració i inicialització  
int arrayEnters [] = {10, 20, 30, 40, 50}
```

```
//declaració, cal indicar la mida  
int[] arrayEnters = new int[20];  
double[] arrayReals = new double[10];
```

# DADES COMPOSTES. ARRAYS

```
public class IniciaArray{  
  
    public static void main(String[] args) {  
        int[] arrayEnters = new int[20];  
        for (int i = 0; i < arrayEnters.length; i++) {  
            arrayEnters[i] = i + 2;  
        }  
        for (int i = 0; i < arrayEnters.length; i++) {  
            System.out.print(arrayEnters[i] + " ");  
        }  
        System.out.println();  
    }  
}
```

Exemples [tractament d'arrays](#)

# DADES COMPOSTES. ARRAYS

**Cerca:** es recorre un array fins a trobar (o no) un valor dins seu. Si arribem al final de l'array significa que no l'hem trobat.

```
boolean trobat = false;
//comptador de posicions.
int i = 0;
while ((i < arrayNotes.length)&&(!trobat)) {
    if (arrayNotes[i] == 10) {
        trobat = true;
    }
    i = i + 1;
}
//s'ha trobat?
if (trobat) {
    System.out.println("Algú ha tret un 10.");
} else {
    System.out.println("Ningú no ha tret un 10");
}
```

# DADES COMPOSTES. ARRAYS

**Ordenació:** es reorganitza els valors que hi ha dins l'array de manera que estiguin ordenats de manera ascendent o descendent.

L'algorisme de la bombolla (**BubbleSort**) serveix per ordenar elements mitjançant recorreguts successius al llarg de la seqüència, en què es van comparant parelles de valors. Cada cop que es troben dues parelles en ordre incorrecte, s'intercanvia la seva posició.

# DADES COMPOSTES. ARRAYS

```
for (int i = 0; i < llistaNotes.length - 1; i++) {  
    //bucle intern, se cerca entre la resta de posicions quin és el  
    valor més baix.  
    for(int j = i + 1; j < llistaNotes.length; j++) {  
        //si la posició tractada té un valor més alt que el de la cerca,  
        //els intercanviem  
        if (llistaNotes[i] > llistaNotes[j]) {  
            //per intercanviar valors cal una variable auxiliar  
            float canvi = llistaNotes[i];  
            llistaNotes[i] = llistaNotes[j];  
            llistaNotes[j] = canvi;  
        }  
    }  
}
```