



UNITAT 4

INTRODUCCIÓ A JAVA

Programació
CFGS DAW

Autors:

Carlos Cacho y Raquel Torres

Revisat per:

Lionel Tarazon - lionel.tarazon@ceedcv.es

Fco. Javier Valero – franciscojavier.valero@ceedcv.es

José Manuel Martí - josemanuel.marti@ceedcv.es

2021/2022

Llicència



[CC BY-NC-SA 3.0 ES](https://creativecommons.org/licenses/by-nc-sa/3.0/es/) **Reconeixement – No Comercial – Compartir Igual (by-nc-sa)**

No es permet un ús comercial de l'obra original ni de les possibles obres derivades, la distribució de les quals s'ha de fer amb una llicència igual a la que regula l'obra original. Aquesta és una obra derivada de l'obra original de Carlos Cacho i Raquel Torres.

Nomenclatura

Al llarg d'aquest tema s'utilitzaran diferents símbols per a distingir elements importants dins del contingut. Aquests símbols són:



Important



Atenció



Interessant

ÍNDEX DE CONTINGUT

Introducció	4
Primer exemple	5
Elements bàsics	7
Comentaris	7
Identificadors	7
Tipus de dades	9
Declaració de variables	10
Àmbit d'una variable	12
Variables locals	12
Constants (final)	13
Operadors	14
Aritmètics	14
Relacionals	15
Lògics	16
D'assignació	17
Expressions	17
Precedència d'operadors	18
La classe Math	19
Literals	20
Literals lògics	20
Literals enters	20
Literals reals	21
Literals caràcter	21
Literals cadenes (String)	22
Eixida i entrada estàndard	24
Eixida estàndard	24
Entrada estàndard	25
Estructures alternatives	28
Estructura Alternativa Simple (if)	28
Estructura Alternativa Doble (if-else)	29
Estructura Alternativa Múltiple (switch)	31
exemples	33
Exemple 1	33
Exemple 2	35
Agraïments	36

1. INTRODUCCIÓ

Java és un llenguatge de programació de propòsit **general**, **concurrent** i **orientat a objectes** que va ser dissenyat específicament per a tindre tan poques dependències d'implementació com fora possible. El seu **objectiu** és permetre que els desenvolupadors d'aplicacions escriguen el programa una vegada i l'executen en qualsevol dispositiu (conegut en anglés com **WORA**, o "*write once, run anywhere*"), la qual cosa vol dir que **el codi pot escriure's una sola vegada i ser executat en qualsevol mena de dispositius** (PC, mòbil, etc.).

Les **característiques** de Java són:

- **Senzill**: És un llenguatge senzill d'aprendre.
- **Orientat a Objectes**: Possiblement és el llenguatge més orientat a objectes de tots els existents; a Java, a excepció dels tipus fonamentals de variables (int, char, long...), tot és un objecte.
- **Distribuït**: Java està molt orientat al treball en xarxa, suportant protocols com TCP/IP, UDP, HTTP i FTP. D'altra banda l'ús d'aquests protocols és bastant senzill comparant-lo amb altres llenguatges que els suporten.
- **Robust**: El compilador Java detecta molts errors que altres compiladors només detectarien en temps d'execució o fins i tot mai.
- **Assegurança**: Sobretot un tipus de desenvolupament: els Miniaplicació. Aquests són programes dissenyats per a ser executats en una pàgina web.
- **Portable**: A Java no hi ha aspectes dependents de la implementació, totes les implementacions de Java segueixen els mateixos estàndards quant a grandària i emmagatzematge de les dades.
- **Arquitectura Neutral**: El codi generat pel compilador Java és independent de l'arquitectura: podria executar-se en un entorn UNIX, Mac, Windows, Mòbil, etc.
- **Rendiment mitjà**: Actualment la velocitat de processament del codi Java és semblant a la d'altres llenguatges orientats a objectes.
- **Multithread**: Suporta de manera nativa els threads (fils d'execució), sense necessitat de l'ús de de llibreries específiques.

2. PRIMER EXAMPLE

L'aplicació més xicoteta possible és la que simplement imprimeix un missatge en la pantalla. Tradicionalment, el missatge sol ser "Hola Mundo!". Això és justament el que fa el següent fragment de codi:

```
12 public class HolaMundo {
13
14     public static void main(String[] args) {
15         // TODO code application logic here
16         System.out.println("Hola Mundo!");
17     }
18
19 }
```

Cal veure detalladament l'aplicació anterior, línia a línia. Aquelles línies de codi contenen els components mínims per a imprimir *Hola Món!* en la pantalla. És un exemple molt simple, que no instància objectes de cap altra classe; no obstant això, accedeix a una altra classe inclosa en el JDK.

public class HolaMundo

Aquesta línia **declara la classe `HolaMundo`**. El nom de la classe especificat en el fitxer font s'utilitza per a crear un fitxer ***nombredeclasse.class*** en el directori en el qual es compila l'aplicació. En aquest cas, el compilador crearà un fitxer anomenat ***HolaMundo.class***.

public static void main(String args[])

Aquesta línia **especifica un mètode** que l'interpret Java busca per a executar en primer lloc. Igual que en altres llenguatges, Java utilitza una **paraula clau `main`** per a especificar **la primera funció a executar**. En aquest exemple tan simple no es passen arguments.

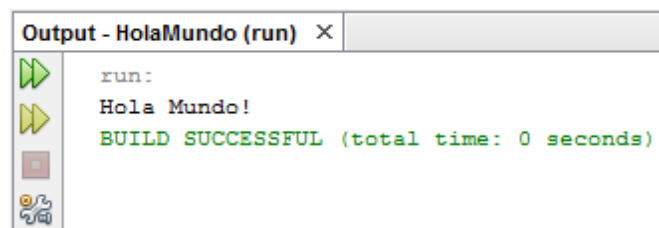
- **public** significa que el mètode `main()` pot ser cridat per qualsevol, incloent l'interpret Java.
- **static** és una paraula clau que li diu al compilador que `main` es refereix a la pròpia classe `HolaMundo` i no a cap instància de la classe. D'aquesta manera, si algú intenta fer una altra instància de la classe, el mètode `main()` no s'instanciarà.
- **void** indica que `main()` no retorna res. Això és important ja que Java realitza una estricta comprovació de tipus, incloent els tipus que s'ha declarat que retornen els mètodes.
- **args[]** és la declaració d'un array de Strings. Aquests són els arguments escrits després del nom de la classe en la línia de comandos: `java HolaMundo arg1 arg2 ...`

System.out.println("Hola Món!");

Aquesta és la funcionalitat de l'aplicació. Aquesta línia mostra l'ús d'un nom de classe i mètode. S'usa el **mètode `println()`** de la **classe `out`** que està en el **paquet `System`**.

El mètode `println()` agafa una cadena com a argument i l'escriu en el stream d'eixida estàndard; en aquest cas, la finestra on es llança l'aplicació. La classe `PrintStream` té un mètode instanciable anomenat `println()`, que el que fa és presentar en l'eixida estàndard del Sistema l'argument que se li passe. En aquest cas, s'utilitza la variable o instància d'`out` per a accedir al mètode.

El resultat seria el següent:



```
run:
Hola Mundo!
BUILD SUCCESSFUL (total time: 0 seconds)
```

⚡ Totes les instruccions (creació de variables, anomenades a mètodes, assignacions) s'han de finalitzar amb un **punt i coma**.

3. ELEMENTS BÀSICS

3.1 Comentaris

A Java hi ha tres tipus de comentaris:

```
// comentaris per a una sola línia
```

```
/*  
comentaris d'una o més línies  
*/
```

```
/** comentari de documentació, d'una o més línies  
*/
```

Els dos primers tipus de comentaris són els que tot programador coneix i s'utilitzen de la mateixa manera.

Els **comentaris de documentació**, col·locats immediatament abans d'una declaració (de variable o funció), indiquen que aqueix comentari ha de ser col·locat en la documentació que es genera automàticament quan s'utilitza l'eina de Java, **javadoc**, no disponible en altres llenguatges de programació. Aquest tipus de comentari el veurem més endavant.

3.2 Identificadors

Els **identificadors** nomenen **variables**, **funcions**, **classes** i **objectes**; qualsevol cosa que el programador o programadora necessite identificar o usar.

Regles per a la creació d'identificadors:

- **Java fa distinció entre majúscules i minúscules**, per tant, noms o identificadors com var1, Var1 i VAR1 són diferents.
- Poden estar formats per qualsevol dels caràcters del codi Unicode, per tant, es poden declarar variables amb el nom: añoDeCreación, raïm, etc., encara que això sí, el **primer caràcter no** pot ser un **dígit numèric** i **no** poden utilitzar-se **espais en blanc ni símbols coincidents amb operadors**.
- La **longitud** màxima dels identificadors és pràcticament **il·limitada**.
- **No** pot ser una **paraula reservada del llenguatge** ni els valors lògics **true** o **false**.
- **No** poden ser **iguals a un altre identificador declarat** en el mateix àmbit.
- **IMPORTANT** → Per conveni:
 - Els **noms** de les **variables** i els **mètodes** haurien de començar per una **lletra minúscula** i els de les **classes** per **majúscula**.
 - Si l'identificador està format per diverses **paraules**, la **primera** s'escriu en **minúscules** (excepte per a les classes) i la **resta** de paraules es fa **començar per majúscula** (per exemple: añoDeCreación).
 - Aquestes **regles** no són obligatòries, però són **convenients** ja que ajuden al procés de codificació d'un programa, així com a la seua llegibilitat. És més senzill distingir entre classes i mètodes o variables.

Serien identificadors vàlids, per exemple:

comptador
suma
edat
sueldoBruto
sueldoNeto
nom_usuari
nom_Complet
letraDni

i el seu ús seria, per exemple:

int comptador;	// crea variable de tipus int anomenada comptador
float sueldoNeto;	// crea variable de tipus float anomenada sueldoNeto
char letraDni;	// crea variable de tipus char anomenada letraDni

4. TIPUS DE DADES

A Java existeixen dos tipus principals de dades:

- Tipus de dades simples: Ens permeten crear variables que emmagatzemen un sol valor. Per exemple per a un comptador, edat, preu, etc. Són els que més utilitzarem ara com ara.
- Tipus de dades compostes: Estructures de dades més complexes que permeten emmagatzemar moltes dades (vectors, objectes, etc.). Les veurem en futures unitats.

Tipus de dades simples suportades per Java:

- Per a nombres enters: **byte**, **short**, **int**, **long**
- Per a nombres reals: **float**, **double**
- Per a caràcters: **char**
- Per a valors lògics: **boolean**.

Tipus	Descripció	Memòria ocupada	Rang de valors permesos
byte	Nombre enter d'1 byte	1 byte	-128 ... 127
short	Nombre enter curt	2 bytes	-32768 ... 32767
int	Nombre enter	4 bytes	-2147483648 ... 2147483647
long	Nombre enter llarg	8 bytes	-9223372036854775808 ... 9223372036854775807
float	Nombre real amb coma flotant de precisió simple	32 bits	$\pm 3,4 \cdot 10^{-38} \dots \pm 3,4 \cdot 10^{38}$
double	Nombre real amb coma flotant de precisió doble	64 bits	$\pm 1,7 \cdot 10^{-308} \dots \pm 1,7 \cdot 10^{308}$
char	Un sol caràcter	2 bytes	
boolean	Valor lògic	1 bit	true o false



Java no realitza una comprovació dels rangs.

Per exemple: si a una variable de tipus **short** amb el valor 32.767 se li suma 1, sorprenentment el resultat serà -32.768 (no produeix un error de tipus desbordament com en altres llenguatges de programació, sinó que **es comporta de manera cíclica**).

Existeix un tipus de dada composta anomenada **String** que convé conèixer ja que permet representar text. Més endavant veurem com s'utilitza.

5. DECLARACIÓ DE VARIABLES

La forma bàsica de declarar (crear) una variable és la següent:

```
tipus identificador;
```

Per exemple, creem una variable de tipus int anomenada edat:

```
int edat;
```

Les variables poden ser inicialitzades en el moment de la seua declaració, és a dir, se'ls pot donar un valor inicial en crear-les. Per exemple, creem una variable de tipus int anomenada edat i li assignem 25 com a valor inicial:

```
int edat = 25;
```

Això és equivalent a primer declarar-la i després assignar-li el valor:

```
int edat;  
edat = 25;
```

També és possible declarar diverses variables en una sola línia. Per exemple, creem tres variables de tipus float anomenades precio1, precio2 i precio3:

```
float preu1, preu2, preu3;
```

Això és equivalent a:

```
float preu1;  
float preu2;  
float preu3;
```

Al seu torn, també poden inicialitzar-se. Per exemple:

```
float preu1 = 7.0, preu2 = 7.25, preu3 = 0.5;
```

Això és equivalent a:

```
float precio1 = 7.0;  
float precio2 = 7.25;  
float precio3 = 0.5;
```

En resum la declaració de variables segueix el següent patró:



```
tipus identificador [ = valor][,identificador [= valor] ...];
```

És a dir, és **obligatori indicar el tipus i l'identificador** (a més d'acabar en punt i coma com totes les instruccions). Opcionalment (indicat entre claudàtors) es pot inicialitzar i/o es poden declarar més variables.



Si una variable no ha sigut inicialitzada, Java li assigna un valor per defecte.

Aquest valor és:

- Per a les variables de tipus **numèric**, el valor per defecte és zero (**0**),.
- Les variables de tipus **char**, el valor **'\u0000'**.
- Les variables de tipus **booleà**, el valor **false**.
- Per a les variables de tipus referencial (**objectes**), el valor **null**.

Es una **bona pràctica inicialitzar sempre totes les variables**.

Paraules clau

Les següents són paraules clau que no es poden utilitzar com a identificadors ja que Java les utilitza per a altres coses:

abstract	continue	for	new	switch
boolean	default	goto	null	synchronized
break	do	if	package	this
byte	double	implements	private	threadsafe
byvalue	else	import	protected	throw
case	extends	instanceof	public	transient
catch	false	int	return	true
char	final	interface	short	try
class	finally	long	static	void
const	float	native	super	while

Paraules reservades

A més, el llenguatge es reserva unes quantes paraules més, però que fins ara no tenen una finalitat especificada. Són:

cast	uture	generic	inner
operator	outer	rest	var

5.1 Àmbit d'una variable



L'àmbit d'una variable és la porció del programa on aquesta variable pot utilitzar-se.

L'àmbit d'una variable depén del lloc del programa on és declarada, podent pertànyer a quatre categories diferents.

1. Variable local.
2. Atribut.
3. Paràmetre d'un mètode.
4. Paràmetre d'un tractador d'excepcions.

Ara com ara utilitzarem només variables locals, les altres categories les veurem en posteriors unitats.

5.2 Variables locals



Una **variable local** es declara dins del cos d'un mètode d'una classe i és **visible únicament dins d'aquest mètode**.

Es pot declarar en qualsevol lloc del cos, fins i tot després d'instruccions executables, encara que és un **bon costum declarar-les just al principi**.

També poden declarar-se variables dins d'un bloc amb claus {...}. En aqueix cas, només seran “visibles” dins d'aquest bloc.

Per exemple (No és necessari entendre el que fa el programa):

```
14  [ ] public static void main(String[] args) {  
15  |  
16  |     int i;  
17  |  
18  |     for (i=0;i<10;i++)  
19  |         System.out.println(i);  
20  | }  
    |
```

En aquest exemple existeix una variable local: **int i**; únicament pot utilitzar-se dins del bloc **main** on es va crear.

5.3 Constants (final)

En declarar una variable pot utilitzar-se la paraula reservada **final** per a indicar que el valor de la variable no podrà modificar-se (és una constant).

Per exemple, creem variable constant tipus int anomenada x amb valor 18:

```
final int x = 18;
```

Per exemple, creem variable constant tipus float anomenada pi amb valor 3.14:

```
final float pi = 3.14;
```

Si posteriorment intentem modificar els seus valors es produirà un error i Java ens avisarà que no és possible.

```
x = 20; // no permés, produeix error
```

```
pi = 7; // no permés, produeix error
```

⚡ Per tant una variable precedida de la paraula **final** es converteix en una **constant**. O cosa que és el mateix, per a definir una constant a Java haurem de precedir la seua declaració de la paraula reservada **final**.

6. OPERADORS

Els **operadors** són una part indispensable de la programació ja que ens permeten realitzar càlculs matemàtics i lògics, entre altres coses. Els operadors poden ser:

- **Aritmètics**: sumes, restes, etc.
- **Relacionals**: menor, menor o igual, major, major o igual, etc.
- **Lògics**: and, or, not, etc.
- **Bits**: pràcticament no els utilitzarem en aquest curs.
- **Assignació**: =

6.1 Aritmètics

Operador	Format	Descripció
+	op1 + op2	Suma aritmètica de dos operands.
-	op1 - op2 -op1	Resta aritmètica de dos operands. Canvi de signe.
*	op1 * op2	Multiplicació de dos operands.
/ %	op1 / op2 op1 % op2	Divisió sencera de dos operands. Resta de la divisió sencera (o mòdul).
++	++op1 op1++	Increment unitari.
--	--op1 op1--	Decrement unitari.

L'operador - pot utilitzar-se en la seua versió unària (- op1) i l'operació que realitza és la d'invertir el signe de l'operand.

Lus operadors ++ i -- realitzen un increment i un decrement unitari respectivament. És a dir:

- **x++** equival a **x = x + 1**
- **x--** equival a **x = x - 1**

Els operadors ++ i -- admeten notació postfixa i prefixa:

- **op1++**: Primer s'executa la instrucció en la qual està immers i després s'incrementa op1.
- **op1--**: Primer s'executa la instrucció en la qual està immers i després es decrementa op1.
- **++op1**: Primer s'incrementa op1 i després executa la instrucció en la qual està immers.
- **--op1**: Primer se decrementa op1 i després executa la instrucció en la qual està immers.

Els operadors incrementals solen utilitzar-se sovint en els bucles (estructures repetitives). Ho veurem més endavant.

6.2 Relacionals

Operador	Format	Descripció
>	op1 > op2	Retorna true (cert) si op1 es major que op 2
<	op1 < op2	Retorna true (cert) si op1 es menor que op2
>=	op1 >= op2	Retorna true (cert) si op1 es major o igual que op2
<=	op1 <= op2	Retorna true (cert) si op1 es menor o igual que op2
==	op1 == op2	Retorna true (cert) si op1 es igual a op2
!=	op1 != op2	Retorna true (cert) si op1 es diferente a op2

Els operadors relacionals actuen sobre valors sencers, reals i caràcters (char); i retornen un valor del tipus booleà (true o false).

Exemple:

```
15 public static void main(String[] args){
16
17     double op1,op2;
18     char op3,op4;
19
20     op1=1.34;
21     op2=1.35;
22     op3='a';
23     op4='b';
24
25     System.out.println("op1=" + op1 + " op2=" + op2);
26     System.out.println("op1>op2 = " + (op1 > op2));
27     System.out.println("op1<op2 = " + (op1 < op2));
28     System.out.println("op1==op2 = " + (op1 == op2));
29     System.out.println("op1!=op2 = " + (op1 != op2));
30     System.out.println("'a'>'b' = " + (op3 > op4));
31
32 }
```

Eixida:

```
run:
op1=1.34 op2=1.35
op1>op2 = false
op1<op2 = true
op1==op2 = false
op1!=op2 = true
'a'>'b' = false
BUILD SUCCESSFUL (total time: 0 seconds)
```

6.3 Lògics

Operador	Format	Descripció
&&	op1 && op2	I lògic (and). Retorna true (cert) si són certs op1 i op2
	op1 op2	O lògic (or) Retorna true (cert) si són certs op1 o op2
!	!op1	Negació lògica (not). Retorna true (cert) si op1 es fals.

Aquests operadors actuen sobre operadors o expressions lògiques, és a dir, aquells que s'avaluen a cert o fals (true / false).

Exemple:

```
15 public static void main(String[] args){
16
17     boolean a, b, c, d;
18
19     a=true;
20     b=true;
21     c=false;
22     d=false;
23
24     System.out.println("true Y true = " + (a && b) );
25     System.out.println("true Y false = " + (a && c) );
26     System.out.println("false Y false = " + (c && d) );
27     System.out.println("true O true = " + (a || b) );
28     System.out.println("true O false = " + (a || c) );
29     System.out.println("false O false = " + (c || d) );
30     System.out.println("NO true = " + !a);
31     System.out.println("NO false = " + !c);
32     System.out.println("(3 > 4) Y true = " + ((3 >4) && a) );
33
34 }
35 }
```

Eixida:

```
run:
true Y true = true
true Y false = false
false Y false = false
true O true = true
true O false = true
false O false = false
NO true = false
NO false = true
(3 > 4) Y true = false
BUILD SUCCESSFUL (total time: 0 seconds)
```


6.4 D'assignació

L'operador d'assignació és el símbol igual: =

variable = expressió.

Assigna a la variable el resultat d'avaluar l'expressió de la dreta.

És possible combinar l'operador d'assignació amb altres operadors per a, de forma abreujada, realitzar un càlcul i assignar-lo a una variable:

Operador	Format	Equivalència
+=	op1 += op2	op1 = op1 + op2
-=	op1 -= op2	op1 = op1 - op2
*=	op1 *= op2	op1 = op1 * op2
/=	op1 /= op2	op1 = op1 / op2
%=	op1 %= op2	op1 = op1 % op2
&=	op1 &= op2	op1 = op1 & op2
=	op1 = op2	op1 = op1 op2
^=	op1 ^= op2	op1 = op1 ^ op2
>>=	op1 >>= op2	op1 = op1 >> op2
<<=	op1 <<= op2	op1 = op1 << op2
>>>=	op1 >>>= op2	op1 = op1 >>> op2

6.5 Expressions

Una expressió és la combinació de diversos operadors i operands. Per exemple, tenim les següents expressions:

7 + 5 * 4 - 2

10 + (1% 5)

(7 * x) <= N

etc.

El llenguatge **Java** avalua les expressions aplicant els operadors un a un seguint un ordre **específic**. Aquest ordre es detalla en el següent punt.

6.6 Precedència d'operadors

Indica l'ordre **en el qual s'avaluen els operadors** en una expressió. No és necessari saber tota la llista de memòria, però **és important conèixer almenys els més utilitzats**: matemàtics, relacionals, lògics i d'assignació.

Alguns d'aquests operadors els veurem en unitats posteriors, ara mateix no és necessari que sàpies què fan.

1. Operadors postfixos: `expr++`, `expr--`, `()`, `.`, `[]` `{}`
2. Operadors unaris: `++expr`, `--expr`, `-expr`, `~`, `!`
3. Creació o conversió de tipus: `new (tipus)expr`
4. **Multiplicació i divisió**: `*`, `/`, `%`
5. **Suma i resta**: `+`, `-`
6. Desplaçament de bits: `<<`, `>>`, `>>>`
7. **Relacionals**: `<`, `>`, `<=`, `>=`
8. **Igualtat i desigualtat**: `==`, `!=`
9. AND a nivell de bits: `&`
10. **AND lògic**: `&&`
11. XOR a nivell de bits: `^`
12. OR a nivell de bits: `|`
13. **OR lògic**: `||`
14. Operador condicional: `?:`
15. **Assignació**: `=`, `+=`, `-=`, `*=`, `/=`, `%=`, `^=`, `&=`, `|=`, `>>=`, `<<=`

6.7 La classe Math

Es troben a faltar operadors matemàtics més potents a Java. Per això s'ha inclòs una classe especial anomenada **Math** dins del paquet *java.lang*.

Aquesta classe posseeix molts mètodes molt interessants per a realitzar càlculs matemàtics complexos com a càlcul de potències, arrels quadrades, valors absoluts, si, cosinus, etc.


















Per exemple :

```
double x = Math.pow(3,3);    // Potència 3 ^ 3
double i = Math.sqrt(9);     // Arrel quadrada de 9
```

També posseeix constants com:

```
double PI = Math.PI --> El número  $\Pi$  (3,14159265...)
double E = Math.E --> El número e (2, 7182818245...)
```

Alguns exemples d'altres mètodes:

 E	double	^
 PI	double	
 IEEEremainder (double f1, double f2)	double	
 abs (double a)	double	
 abs (float a)	float	
 abs (int a)	int	
 abs (long a)	long	
 acos (double a)	double	
 addExact (int x, int y)	int	
 addExact (long x, long y)	long	
 asin (double a)	double	
 atan (double a)	double	
 atan2 (double y, double x)	double	
 cbrt (double a)	double	
 ceil (double a)	double	
 copySign (double magnitude, double sign)	double	
 copySign (float magnitude, float sign)	float	▼

7. LITERALS

A l'hora de tractar amb valors dels tipus de dades simples (i de tipus Strings) s'utilitza el que es denomina "literals". Els literals són elements que serveixen per a representar un valor en el codi font del programa.

A Java existeixen literals per als següents tipus de dades:

- Lògics (boolean).

- Caràcter (char).

- Enters (byte, short, int i long).

- Reals (double i float).

- Cadenes de caràcters (String).

7.1 Literals lògics

Són únicament dos, les paraules reservades *true* i *false*.

Exemple: *boolean activat = false;*

7.2 Literals enters

Els literals de tipus enters (nombres enters): *byte*, *short*, *int* i *long* poden expressar-se en decimal (base 10), octal (base 8) o hexadecimal (base 16). A més, pot afegir-se al final del mateix la lletra *L* per a indicar que l'enter és considerat com long (64bits).

En Java, el compilador identifica un enter decimal (base 10) en trobar un número el primer dígit del qual és qualsevol símbol decimal excepte el zero (de l'1 al 9). A continuació poden aparèixer dígits del 0 al 9.

La lletra *L* al final d'un literal de tipus sencer pot aplicar-se a qualsevol sistema de numeració i indica que el nombre decimal siga tractat com un enter llarg (de 64 bits). Aquesta lletra *L* pot ser majúscula o minúscula, encara que és aconsellable utilitzar la majúscula ja que en cas contrari pot confondre's amb el dígit un (1) en els llistats.

Exemple:

long max1 = 9223372036854775807L; //aquest és el valor màxim per a un enter llarg

7.3 Literals reals

Els literals de tipus real serveixen per a indicar valors *float* o *double*. A diferència dels literals de tipus sencer, **no poden expressar-se en octal o hexadecimal**.

Existeixen dos formats de representació: mitjançant la seua part sencera, el punt decimal (.) i la part fraccionària; o mitjançant notació exponencial o científica:

Exemples equivalents:

3.1415

0.31415e1

.31415e1

0.031415E+2

.031415e2

314.15e-2

31415E-4

Igual que els literals que representen sencers, es pot posar una lletra com a sufix. Aquesta lletra pot ser una *F* o una *D* (majúscula o minúscula indistintament).

- *F* --> Tracta el literal com de tipus *float*.
- *D* --> Tracta el literal com de tipus *double*.

Exemple:

3.1415F

.031415d

7.4 Literals caràcter

Els literals de tipus caràcter es representen sempre entre cometes simples. Entre les cometes simples pot aparéixer:

- Un **símbol** (lletra) sempre que el caràcter estiga associat a un codi Unicode.
Exemples: *'a'*, *'B'*, *'{'*, *'ñ'*, *'á'*.
- Una **"seqüència de fuga"**. Les seqüències de fuga són combinacions del símbol contrabarra \ seguit d'una lletra, i serveixen per a representar caràcters que no tenen una equivalència en forma de símbol.

Les possibles seqüències de fuga són:

`\n` ----> Nova Línia.

`\t` ----> Tabulador.

`\r` ----> Reculada de Carro.

`\f` ----> Començament de Pàgina.

`\b` ----> Esborrat a l'Esquerra.

`\\` ----> El caràcter barra inversa (`\`).

`\'` ----> El caràcter preval simple (`'`).

`\"` ----> El caràcter preval doble o bi-prima (`"`).

Per exemple :

Per a imprimir una diagonal inversa s'utilitza: `\\`

Per a imprimir cometes dobles en un String s'utilitza: `\"`

7.5 Literals cadenes (String)

Els **Strings** o cadenes de caràcters no formen part dels tipus de dades elementals a Java, sinó que són instanciats a partir de la classe `java.lang.String`, però accepten la seua inicialització a partir de literals d'aquest tipus, per la qual cosa es tracten en aquest punt.



Un literal de tipus String va tancat entre cometes dobles (`"`) i ha d'estar inclòs completament en una sola línia del programa font (no pot dividir-se en diverses línies).

Entre les cometes dobles pot incloure's qualsevol caràcter del codi Unicode (o el seu codi precedit del caràcter `\`) a més de les seqüències de fuga vistes anteriorment en els literals de tipus caràcter. Així, per exemple, per a incloure un canvi de línia dins d'un literal de tipus string haurà de fer-se mitjançant la seqüència de fuga `\n` :

Exemple:

```
System.out.println("Primera línia\nSegunda línia del string\n");
```

```
System.out.println("Hola");
```

La visualització del *string* anterior mitjançant `println()` produiria la següent eixida per pantalla:

Primera línia

Segunda línia del string

Hola

La manera d'incloure els caràcters cometes dobles (") i contrabarra (\) és mitjançant les seqüències de fuga \" i \\ respectivament (o mitjançant el seu codi Unicode precedit de \).



Si el String és massa llarg i ha de dividir-se en diverses línies en el fitxer font, pot utilitzar-se l'operador de concatenació de Strings (+) de la següent forma:

"Aquest String és massa llarg per a estar en una línia del " +
"fitxer font i s'ha dividit en dues."

8. EIXIDA I ENTRADA ESTÀNDARD

8.1 Eixida estàndard

Ja hem vist l'ús de **System.out** per a mostrar informació per pantalla:

- **print(...)** imprimeix text per pantalla
- **println(...)** imprimeix text per pantalla i introdueix un salt de línia.

La utilització de **System.err** seria totalment anàloga per a enviar els missatges produïts per errors en l'execució (és el canal que usa també el compilador per a notificar els errors trobats).

Per exemple , per a presentar el missatge de salutació habitual per pantalla, i després un missatge d'error, tindríem la següent classe (encara que en realitat tota la informació va a la consola de comandos on estem executant el programa):

```
14 public static void main(String[] args) {  
15  
16     System.out.print("HOLA ");  
17     System.out.println("mundo");  
18     System.err.println("Mensaje de error");  
19 }
```

I l'eixida seria la següent:

```
Output - HolaMundo (run) ×  
run:  
HOLA mundo  
Mensaje de error  
BUILD SUCCESSFUL (total time: 0 seconds)
```

També poden imprimir-se variables de qualsevol tipus, així com combinacions de text i variables concatenades amb l'operador +

```
14 public static void main(String[] args) {  
15     String nombre = "Pepito";  
16     int edad = 25;  
17     System.out.println(nombre);  
18     System.out.println(edad);  
19     System.out.println(nombre + " tiene " + edad + " años");  
20 }
```

I l'eixida seria la següent:

```
Pepito  
25  
Pepito tiene 25 años
```


8.2 Entrada estàndard

La entrada estàndard (llegir informació del teclat, escrita per l'usuari) és una mica més complexa. Hi ha diverses maneres de fer-ho però la més senzilla és utilitzar la classe **Scanner**.

Sempre que vulguem llegir informació del teclat primer haurem de declarar un objecte **Scanner** que llija de l'entrada estandar **System.in** així:

```
Scanner reader = new Scanner(System.in);
```

NOTA: En aquest exemple hem creat un objecte **Scanner** anomenat **reader** però podríem posar-li qualsevol nom.

Ara podrem utilitzar **reader** tantes vegades com vulguem per a llegir informació del teclat. Per exemple:

```
String texto = reader.nextLine();
```

El mètode **reader.nextLine()** recollirà el text que l'usuari escriga per teclat (fins a pressionar la tecla Intro) i ho guardarà en **texto** (de tipus **String**).

Existeixen molt altres mètodes segons la mena de dada que es vulga llegir:

- **nextByte()**: obté un nombre enter tipus byte.
- **nextShort()**: obté un nombre enter tipus short.
- **nextInt()**: obté un nombre enter tipus int.
- **nextLong()**: obté un nombre enter tipus long.
- **nextFloat()**: obté un nombre real float.
- **nextDouble()**: obté un nombre real double.
- **next()**: obté el següent token (text fins a un espai).



No existeixen mètodes de la classe **Scanner** per a obtindre directament booleans ni per a obtindre un sol caràcter.

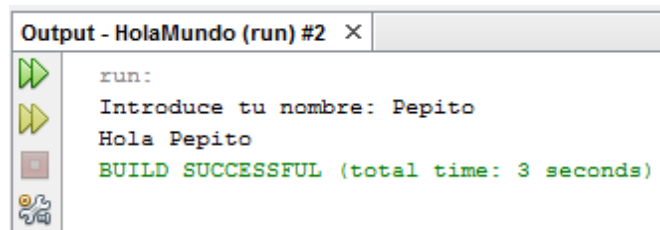
IMPORTANT: Per a poder utilitzar la classe **Scanner** és necessari importar-la des del paquet **java.util** de Java. És a dir, a dalt del tot (abans del **public class...**) cal escriure la següent sentència:

```
import java.util.Scanner;
```

Exemple en el qual llegim una cadena de text i la mostrem per pantalla:

```
12  import java.util.Scanner;
13
14  public class EjemploScanner {
15
16      public static void main(String[] args) {
17
18          String nombre;
19
20          Scanner entrada = new Scanner(System.in);
21
22          System.out.print("Introduce tu nombre: ");
23
24          nombre = entrada.nextLine();
25
26          System.out.println("Hola " + nombre);
27
28      }
29  }
```

Eixida:








```
Output - HolaMundo (run) #2 X
run:
Introduce tu nombre: Pepito
Hola Pepito
BUILD SUCCESSFUL (total time: 3 seconds)
```

Exemple en el qual llegim un valor tipus double. El programa demana a l'usuari que introduïska el radi d'un cercle, després calcula la seua àrea i circumferència, finalment el mostra per pantalla.

```
12 import java.util.Scanner;
13
14 public class EjemploScanner {
15
16     public static void main(String[] args){
17
18         double radio, area, circumferencia;
19
20         Scanner entrada = new Scanner(System.in);
21
22         System.out.print("Introduce el radio: ");
23
24         radio = entrada.nextDouble();
25
26         // Se hace uso de la librería Math para usar PI y la portencia(pow)
27         area = Math.PI * Math.pow(radio, 2);
28
29         circumferencia = 2 * Math.PI * radio;
30
31         System.out.println("El área es " + area);
32
33         System.out.println("La circumferencia es " + circumferencia);
34     }
35 }
```

Eixida:

Output - HolaMundo (run) #2 ×	
	run:
	Introduce el radio: 2,9
	El área es 26.42079421669016
	La circumferencia es 18.2212373908208
	BUILD SUCCESSFUL (total time: 4 seconds)

9. ESTRUCTURES ALTERNATIVES

Com ja vam veure, les estructures alternatives són construccions que permeten alterar el flux seqüencial d'un programa de manera que en funció d'una condició o el valor d'una expressió, el mateix pugui ser desviat en l'una o l'altra alternativa de codi.

Les estructures alternatives disponibles a Java són:

Alternativa Simple (if)

Alternativa Doble (if-else)

Alternativa Múltiple (switch)

9.1 Estructura Alternativa Simple (if)

L'alternativa simple es codifica de la següent forma:

Codi	Ordinograma
<pre>if (condició) { // Accions; }</pre> <p>El bloc d'Accions s'executa si la condició s'avalua a true (és vertadera).</p> <pre>if (cont == 0) { System.out.println("cont és 0"); // més instruccions... }</pre> <p>Si dins del if només hi ha una instrucció, no és necessari posar les claus.</p> <pre>if (cont == 0) System.out.println("cont és 0");</pre>	<pre>graph TD Entry(()) --> Condicion{Condicion} Condicion -- SI --> Acciones[Acciones] Acciones --> Merge(()) Condicion --> Merge Merge --> Exit(())</pre>

9.2 Estructura Alternativa Doble (if-else)

L'alternativa doble permet indicar quin codi executar si la condició és falsa.

Codi	Ordinograma
<pre>if (condició) { // AccionsSI; } else { // AccionsNO; }</pre> <p>El bloc AccionsSI s'executa si la condició s'avalua a true (vertadera). En cas contrari, s'executa el bloc de AccionsNO.</p> <pre>if (cont == 0) { System.out.println("cont és 0"); // més instruccions... } else { System.out.println("cont no és 0"); // més instruccions... }</pre> <p>Si dins del if o el else només hi ha una instrucció, no és necessari posar les claus.</p> <pre>if (cont == 0) System.out.println("cont és 0"); else System.out.println("cont no és 0");</pre>	<pre>graph TD Entry(()) --> Condicion{Condicion} Condicion -- NO --> Acciones1[Acciones] Condicion -- SI --> Acciones2[Acciones] Acciones1 --> Merge(()) Acciones2 --> Merge Merge --> Exit(())</pre>

⚡ Recordeu que l'operador relacional per a comprovar si són iguals és ==, no un sol = que correspon amb l'operador d'assignació. Aquest error no el detecta el compilador i és difícil d'esbrinar.

En moltes ocasions, es nien estructures alternatives if-else, de manera que es pregunte per una condició si anteriorment no s'ha complit una altra successivament.

Per exemple: suposem que realitzem un programa que mostra la nota d'un alumne en la forma (insuficient, suficient, bé, notable o excel·lent) en funció de la seua nota numèrica. Podria codificar-se de la següent forma:

```
1
2 import java.util.Scanner;
3
4 public class Nota {
5
6     public static void main(String[] args) {
7         Scanner entrada = new Scanner(System.in);
8         int nota;
9         //Suponemos que el usuario introduce el número correctamente.
10        //No hacemos comprobación
11        System.out.println("Dame un número entre 0 y 10");
12
13        nota = entrada.nextInt();
14
15        if (nota < 5) {
16            System.out.println("Insuficiente");
17        } else if (nota < 6) {
18            System.out.println("Suficiente");
19        } else if (nota < 7) {
20            System.out.println("Bien");
21        } else if (nota < 9) {
22            System.out.println("Notable");
23        } else {
24            System.out.println("Sobresaliente");
25        }
26    }
27 }
28 }
```

Sent l'eixida:

```
run:
Dame un número entre 0 y 10
8
Notable
BUILD SUCCESSFUL (total time: 11 seconds)
```

🔊 És molt recomanable usar la tecla tabulador en les instruccions de cada bloc. Com es pot veure en l'exemple, cada **else** està alineat amb el seu **if** associat, d'aquesta manera és més fàcil llegir el codi.

9.3 Estructura Alternativa Múltiple (switch)

L'alternativa múltiple es codifica de la següent forma:

Codi	Ordinograma
<pre>switch (expressió) { case valor1: // Accions1; break; case valor2: // Accions2; break; case valorN: // AccionsN; break; default: // Accions per defecte; }</pre>	

És molt important entendre que en el **switch** s'avalua una **expressió** (un valor concret com 0, 5, 1...) **no una condició** (vertadera o falsa) com en el **if** i el **ifelse**.

El programa comprova el valor de l'expressió i saltarà al 'case' que corresponga amb aquest valor (valor1 o valor2 o ...) executant el codi de dita 'case' (Accions1 o Accions2 o ...). Si no coincideix cap valor, saltarà al 'default' i executarà les accions per defecte.

És important afegir la sentència **break;** al final de cada 'case', ja que en cas contrari el programa continuarà executant el codi de les altres accions i normalment no voldrem que faci això (encara que Java permet fer-ho, és confús i per això està desaconsellat).

Un exemple seria el següent:

```
1
2 import java.util.Scanner;
3
4 public class Alternativa_Multiple {
5
6     public static void main(String[] args) {
7         Scanner entrada = new Scanner(System.in);
8         int dia;
9
10        System.out.println("Dame un número entre 1 y 7:");
11
12        dia = entrada.nextInt();
13
14        switch (dia) {
15            case 1:
16                System.out.println("Lunes");
17                break;
18            case 2:
19                System.out.println("Martes");
20                break;
21            case 3:
22                System.out.println("Miércoles");
23                break;
24            case 4:
25                System.out.println("Jueves");
26                break;
27            case 5:
28                System.out.println("Viernes");
29                break;
30            case 6:
31                System.out.println("Sábado");
32                break;
33            case 7:
34                System.out.println("Domingo");
35                break;
36            default:
37                System.out.println("Error el número debe estar entre 0 y 7");
38        }
39    }
40 }
41
42 }
```

l'execució:

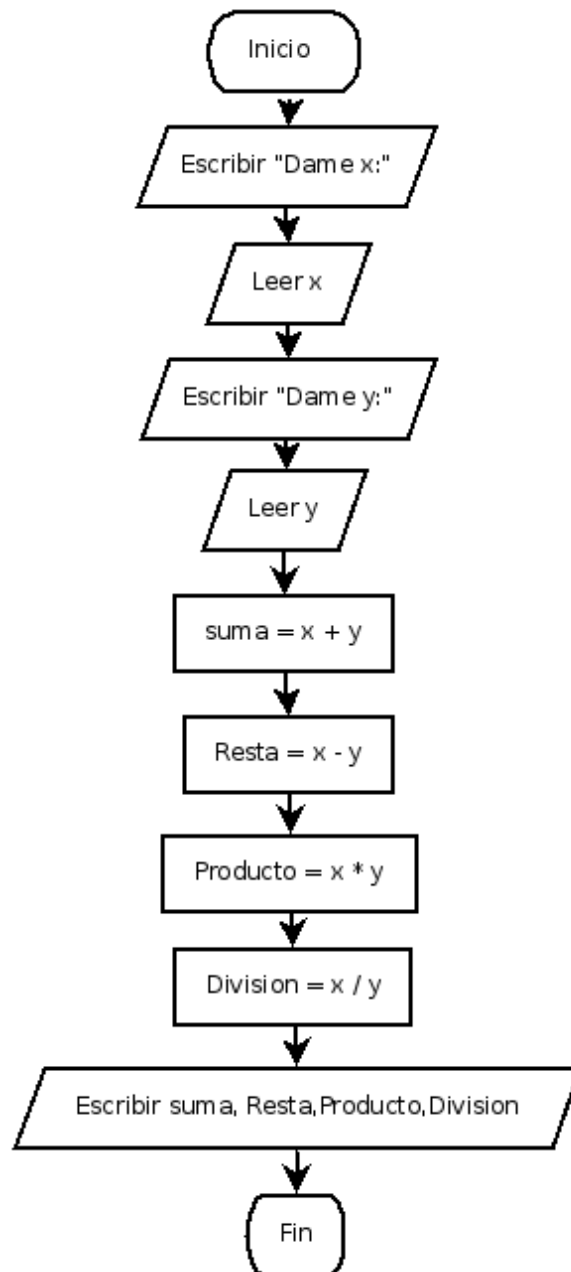
```
run:
Dame un número entre 1 y 7:
4
Jueves
BUILD SUCCESSFUL (total time: 2 seconds)
```


10. EXEMPLES

10.1 Exemple 1

Programa que llija dos números, calcule i mostre el valor de les seues suma, resta, producte i divisió.

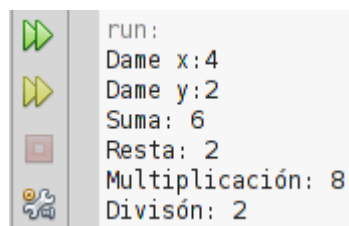
Ordinograma:



Codi:

```
1  package ejemplo1;
2
3  import java.util.Scanner; // Importamos la clase Scanner
4
5  public class Ejemplo1 {
6
7      public static void main(String[] args) {
8
9          // Declaramos las variables que vamos a necesitar
10         int x, y, suma, resta, mult, div;
11
12         // Creamos el objeto Scanner para leer por teclado
13         Scanner reader = new Scanner(System.in);
14
15         // Pedimos y leemos x
16         System.out.print("Dame x:");
17         x = reader.nextInt();
18
19         // Pedimos y leemos y
20         System.out.print("Dame y:");
21         y = reader.nextInt();
22
23         // Realizamos los cálculos necesarios
24         suma = x + y;
25         resta = x - y;
26         mult = x * y;
27         div = x / y;
28
29         // Mostramos los cálculos por pantalla
30         System.out.println("Suma: " + suma);
31         System.out.println("Resta: " + resta);
32         System.out.println("Multiplicación: " + mult);
33         System.out.println("División: " + div);
34     }
35 }
```

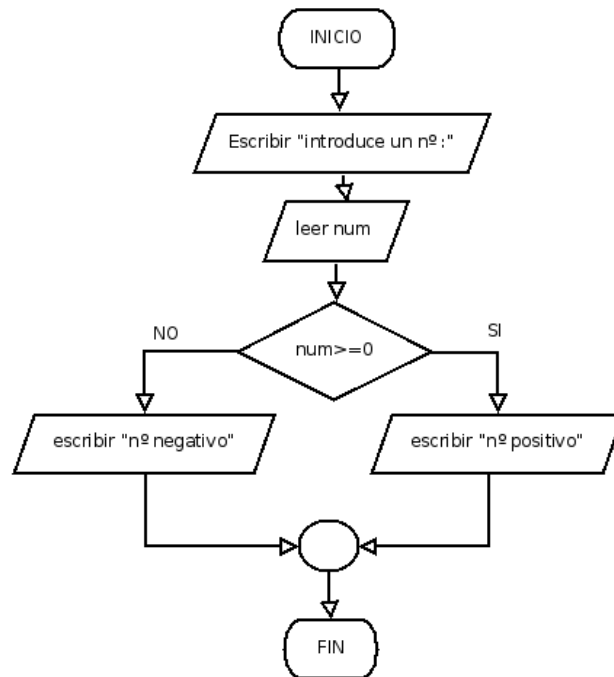
Eixida:



```
run:
Dame x:4
Dame y:2
Suma: 6
Resta: 2
Multiplicación: 8
División: 2
```

10.2 Exemple 2

Programa que llig un número i em diu si és positiu o negatiu. Considerarem el zero com a positiu.



```
1 package ejemplo2;
2
3 import java.util.Scanner; // Importamos la clase Scanner
4
5 public class Ejemplo2 {
6
7     public static void main(String[] args) {
8
9         // Declaramos la variable num
10        int num;
11
12        // Creamos el objeto Scanner para leer por teclado
13        Scanner reader = new Scanner(System.in);
14
15        // Pedimos y leemos x
16        System.out.print("Introduce un nº: ");
17        num = reader.nextInt();
18
19        // Astructura alternativa doble
20        if (num >= 0)
21            System.out.println("Número positivo");
22        else
23            System.out.println("Número negativo");
24    }
25 }
```

```
run:
Introduce un nº: 10
Número positivo
```

11. AGRAÏMENTS

Anotacions actualitzades i adaptades al CEEDCV a partir de la següent documentació:

- [1] Anotacions Programació de José Antonio Díaz-Alejo. IES Camp de Morvedre.