UD2.- Introducció a JAVA

"Primer resol el problema. Després, escriu el codi"

John Johnson

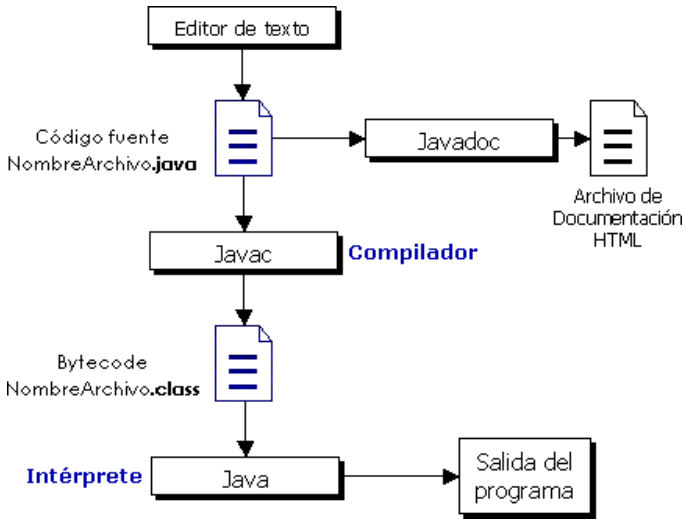
1.- Introducció.

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 anglès com WORA, o write onze, 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).

Es diu que un llenguatge de programació és concurrent si inclou les estructures necessàries per definir i gestionardiferents tasques (Fils d'Execució) dins d'un programa.

Java és un llenguatge de programació **estructurat** i, com a tal, fa ús de **variables, sentències condicionals, bucles, funcions**... Java és també un llenguatge de programació **orientat a objectes** i, per consegüent, permet **definir classes** amb els seus **mètodes** corresponents i **crear instàncies** d'aqueixes classes. Java no és un llenguatge de marques com a HTML o XML encara que pot interactuar molt bé amb ells.

Les aplicacions Java es solen compilar a **bytecode**, que és independent del sistema operatiu, no a binari (que sí que depén del sistema operatiu). D'aquesta manera, el **bytecode** generat al compilar un programa escrit en **Java** hauria de funcionar en qualsevol sistema operatiu que tinga instal·lada una **màquina virtual de Java** (JVM).



Qualsevol editor simple com Nano, GEdit o Kwrite és suficient per a escriure codi en Java encara que es recomanen IDE's com Eclipse o NetBeans ja que tenen algunes característiques que faciliten molt la programació com la revisió mèdica d'errors mentre s'escriu, l'auto completat de noms de variables i funcions i moltes més ajudes durant la tasca de programació.

Les principals 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 excepciódels tipus fonamentals de variables (int, char, long...), tot és un objecte.

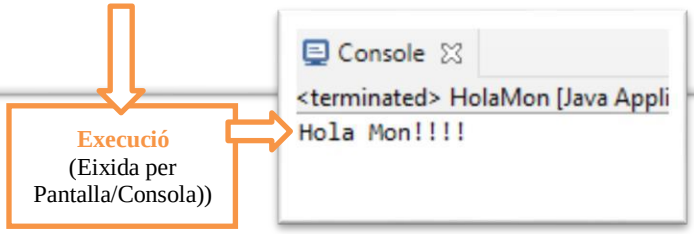
- **Distribuït:** Java està molt orientat al treball en xarxa, suportant protocols com TCP/IP, UDP, HTTP i FTP. D'altrabanda l'ús d'aquests protocols és bastant senzill comparant-los amb altres llenguatges que els suporten.
- **Robust:** El compilador Java detecta molts errors que altres compiladors només detectarien en temps d'execució o finsi tot mai.
- **Segur:** Sobretot un tipus de desenvolupament: els Applet (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 elsmateixos 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 unentorn UNIX, Mac, Windows, Mòbil, etc.
- **Rendiment mitjà:** Actualment la velocitat de processament del codi Java és semblant a la d'altres llenguatgesorientats a objectes.
- **Multithread:** Suporta de manera nativa els threads (fils d'execució), sense necessitat de l'ús de de llibreriesespecífiques.

2.- Estructura bàsica d'un programa.

L'aplicació més xicoteta possible és la que simplement imprimeix un missatge per pantalla. Tradicionalment, el missatge solser **"Hola Món!"**. (Versions de "Hola Mundo" en diferents llenguatges de programació).

Això és justament el que fa el següent fragment de codi:

```
1
2  public class HolaMon {
3
4      public static void main(String[] args) {
5
6          System.out.println("Hola Mon!!!!");
7
8          //Mètode que mostra per la pantalla (per consola) una frase.
9
10     } //del main()
11
12 } // de la classe
13
```



Analitzem línia a línia el codi anterior:

```
public class HolaMon {
…
} // de la classe
```

Esta línea **declara la classe *HolaMón***.

- El nom de la classe especificat coincidirà amb el del fitxer font (***HolaMon.java***)
- Este fitxer font **s'utilitza per a crear un fitxer *NomDeLaClasse.class*** en el directori en el qual es compila l'aplicació.En aquest cas, el compilador crearà un fitxer anomenat *HolaMon.class*. (**arxiu Bytecode**).
- Posteriorment, l'intèrpret de JAVA, executarà el programa i mostrarà per pantalla la frase **Hola Mon!!!!**

```
public static void main(String[] args) {
 …
} //del main()
```

Esta línia especifica un mètode que l'intèrpret de Java busca per a executar en primer lloc. Al igual que altres llenguatges, **Java utilitza la 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'intèrpret Java.

- **static** és una paraula clau que li diu al compilador que **main()** es refereix a la pròpia **classe** HolaMon i no a cap instànciade la classe. D'aquesta manera, si algú intenta fer una altra instància de la classe, el mètode main() no s'instanciaria.

- **void** indica que **main()** no retorna res. Això és important ja que ***Java realitza una estricta comprovació de tipus***, incloentels 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 decomandos: **java HolaMon arg1 arg2** …

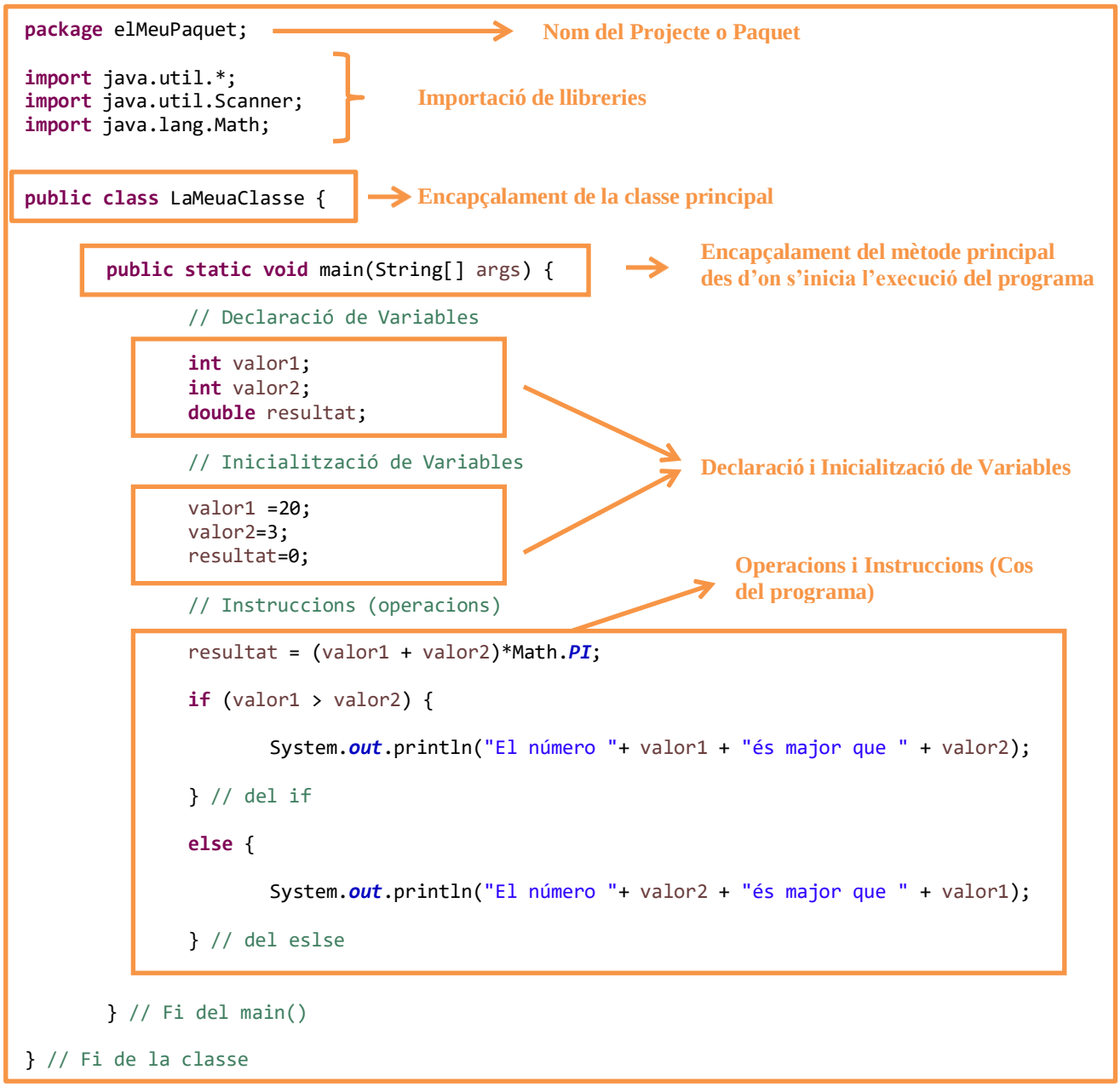
```
System.out.println("Hola Mon!!!!");
```

Esta és la funcionalitat del nostre programa. Esta 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* de l'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 mostrar en l'eixida estàndard del Sistema l'argument que se li passe. En aquest cas, s'utilitza la variable o instància de **out** per a accedir al mètode.

Totes les instruccions (creació de variables, assignacions, cridades a mètodes) han de finalitzar amb punt i coma ;

Estructura General d'un Programa Simple en Java.



3.- Elements bàsics (Comentaris i Identificadors)

• Comentaris:

A Java trobem 3 tipus de comentaris:

```
// Comentari d'una línia

/* Comentari
   de varies
   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.

• Identificadors:

Els **identificadors donen nom a variables, funcions, classes i objectes**, es a dir qualsevol cosa que el programador necessiteidentificar o usar.

Regles per a la creació d'identificadors:

- Java **diferència 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.
- El **primer** caràcter **no** pot ser un **dígit numèric** i **no** poden utilitzar-se **espais en blanc** ni símbols coincidents amboperadors (+,-, = &, etc).
- 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.

(NOTA) Per conveni:

- Els **noms** de les **variables** i els **mètodes** haurien de **començar** per una **lletra minúscula** i els de **les classes permajúscula.**
- Si l'identificador està format per **varies paraules**, la **primera** s'escriu en **minúscules** (excepte per a les classes) i la **resta** de paraules **les posem en majúscula** (per exemple: anyDeCreacio).
- 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.

Identificadors Vàl·lids	Com s'utilitzen en Java

comptador
suma
edat
souBrut
nom_usuari
esUnValorLogic
lletraDelDni
	`int comptador; // crea una variable de tipus int de nom comptador`
	`long suma; // crea una variable de tipus long de nom suma`
	`short edat; // crea una variable de tipus short de nom edat`
	`double souBrut; // crea una variable de tipus double de nom souBrut`
	`String nom_usuari; // crea una variable de tipus String de nom nom_usuari`
	`boolean esUnValorLogic; // crea una variable de tipus boolean de nom esUnValorLogic`
	`char lletraDelDni; // crea una variable de tipus char de nom lletraDelDni`

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, un contador, la edat, preu, etc.
- **Tipus de dades Compostes:** Són estructures de dades més complexes, ens permeten emmagatzemar moltes dades (vectors, objectes, llistes, etc.).

Els tipus de dades simples soportats per Java són:

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

Tipus	Descripció	Memòria	Rang de Valors
byte	Nombre enter d'1 byte	1 byte	-128 … 127
short	Nombre enter curt	2 bytes	-32.768 … 32.767
int	Nombre enter	4 bytes	-2.147.483.648 … 2.147.483.647
long	Nombre enter llarg	8 bytes	-9.223.372.036.854.775.808 … 9.223.372.036.854.775.807
float	Nombre real en coma flotant de precisió simple	32 bits	$\pm 3{,}4*10^{-38}$ … $\pm 3{,}4*10^{38}$
double	Nombre real en coma flotant de precisió doble	64 bits	$\pm 1{,}7*10^{-308}$ … $\pm 1{,}7*10^{308}$
char	Un sol caràcter	2 bytes	
boolean	Valor lògic	1 bit	true o false

Existeix també un tipus de dades compost, el tipus **String (o cadena de caràcters)**, que permet representar text, i es potutilitzar de manera pareguda als tipus de dades simples.

```
String laMeuaCadena = "Hola em diuen Vero";
```

NOTA (PERILL): Java no realitza comprobació de rangs, es a dir, si una variable de tipus short amb el valor màxim de **32.767** li sumem 1, **NO PRODUEIX UN ERROR DE DESBORDAMENT** com en altres llenguatges, el

resultat serà -**32.768**,(comença de forma cíclica).

5.- Declaració de Variables.

- La forma bàsica de **declarar** (crear) una variable és: `tipus identificador;`

Per exemple:

```
int edat;           // crea una variable de tipus int de nom edat
long suma;          // crea una variable de tipus long de nom suma
```

- Les variables poden ser **inicialitzades** en el **moment de la seua declaració**, es a dir, se'ls pot assignar un valor inicial al crear-les o també declarar primer la variable i després assignar-li un valor.

```
int edat = 45;      // declarem la variable edat i li assignem el valor 45

   és equivalent a:

int edat;           // declarem la variable edat
edat = 45;          //assignem a la variable edat el valor 45
```

- També és possible **declarar** diverses variables en una sola línia **separant-les per comes**. Per exemple, creem tres variables de tipus float anomenades preu1, preu2, preu3 :

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

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

- I també és possible inicialitzar-les.

```
float preu1 = 0.25, preu2 = 3.2, preu3 = 32.365;
```

```
float preu1 = 0.25;
float preu2 = 3.2;
float preu3 = 32.365;
```

- 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.

IMPORTANT: 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 **boolean**, el valor **false**.
- Per a les variables de tipus referencial (**objectes**), el valor **null**.

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

Java és un llenguatge de programació fortament tipat, ja que no permet violacions dels tipus de dades, és a dir, si declarem una variable amb un tipus concret, no la podem utilitzar con si fora d'un altre tipus de dades.

Estes violacions de tipus provocaran errors en temps de compilació, com els següents:

```
public class ErrorsComuns {

		public static void main(String[] args) {

	// errors en temps de compilació

			int suma=0;
			String nom;
			suma=nom; //error. Intentem assignar un tipus String en un INT

			int valor=0;
			short numero=0;
			numero=valor; //error. Intentem assignar un tipus INT en un SHORT

			String valor=""; //error. El identificador (variable) error ja existeix

			int major=1;
			int menor;

			if (true){
				major=menor;// error. La variable menor no ha sigut inicialitzada
			}

		} // Fi del main()

} // Fi de la classe
```

- **Paraules reservades:**

No es poden utilitzar com a identificadors ja que Java les reserva 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

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

cast	uture	generic	inner
operator	outer	rest	var

6.- 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 depèn 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.

• Variables locals

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

En aquest exemple (No és necessari entendre el que fa el programa) , existeixen dos variables locals:

- **int i** : únicament pot utilitzar-se dins del bloc **main()** on ha sigut creada.
- **int copiaDeI** : únicament pot utilitzar-se dins del bloc **for()** on ha sigut creada, fora de les claus de bloc {..} produeix error de compilació.

```
3
4⊖    public static void main(String[] args) {
5
6         int i;
7
8         for (i=0;i<10;i++) {
9
10            int copiaDeI=i;
11            System.out.println(i);
12        }
13
14        System.out.println(copiaDeI);
15    }
16
17 } // de la classe
18
```

• 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= 45;        // no permès, produeix un error
pi= 8;        // no permès, produeix un error
```

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

7.- Operadors.

Els **operadors** són una part indispensable de la programació ja que ens permeten realitzar **càlculs matemàtics i lògics**, entrealtres 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**: =

• 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	Divisió entera de dos operands
%	op1 % op2	Resta de la divisió entera (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.L'us 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 es increment a 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 es solen utilitzar en els bucles (estructures repetitives). Ho veurem més endavant.

- ## Relacionals

Operador	Format	Descripció
>	op1 > op2	Retorna true (cert) si op1 és major que op2
<	op1 < op2	Retorna true (cert) si op1 és menor que op2
>=	op1 >= op2	Retorna true (cert) si op1 és major o igual que op2
<=	op1<= op2	Retorna true (cert) si op1 és menor o igual que op2
==	op1 == op2	Retorna true (cert) si op1 és igual a op2
!=	op1 != op2	Retorna true (cert) si op1 és diferent de op2

Els operadors relacionals actuen sobre valors **enters**, **reals** i **caràcters** (char), i retornen un valor del tipus **boolean** (**true** o **false**).

Exemple:

```
public class OperadorsRelacionals {

	public static void main(String[] args) {

		// Declaració de Variables

		double valor1, valor2;
		char   valor3, valor4;

		// Inicialització de Variables

		valor1 = 1.34;
		valor2 = 1.35;
		valor3 = 'a';
		valor4 = 'b';

		// Instruccions (Operacions)

		System.out.println("valor1 = "+ valor1 + " , valor2 = "+ valor2);

		System.out.println("valor1 > valor2 = " + (valor1 > valor2));
		System.out.println("valor1 < valor2 = " + (valor1 < valor2));

		System.out.println("valor1 == valor2 = " + (valor1 == valor2));
		System.out.println("valor1 != valor2 = " + (valor1 != valor2));

		System.out.println("'a' > 'b' = " + (valor3 > valor4));


	} // Fi del main()

} // Fi de la classe
```

```
Console
valor1 = 1.34 , valor2 = 1.35
valor1 > valor2 = false
valor1 < valor2 = true
valor1 == valor2 = false
valor1 != valor2 = true
'a' > 'b' = false
```

- ## Lògics

Operador	Format	Descripció
&&	op1 && op2	I lògica (AND). Retorna true (cert) si són certs op1 i op2
||	op1 || op2	O lògica (OR). Retorna true (cert) si són certs op1 o op2
!	! op1	Negació lògica (NOT). Retorna true (cert) si és false op1.

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

Exemple:

```
public class OperadorsLògics {

	public static void main(String[] args) {

		// Declaració de Variables

		boolean a,b,c,d;

		// Inicialització de Variables

		a=true;
		b=true;
		c=false;
		d=false;

		// Instruccions (Operacions)

		System.out.println("true AND true = " + (a && b) );
		System.out.println("true AND false = " + (a && c) );
		System.out.println("false AND false = " + (c && d) );

		System.out.println("true OR true = " + (a || b) );
		System.out.println("true OR false = " + (a || c) );
		System.out.println("false OR false = " + (c || d) );

		System.out.println("NOT true = " + !a );
		System.out.println("NOT false = " + !c );

		System.out.println("(3 > 4 ) AND true = " + (3>4 && a) );

	} // Fi del main()

} // Fi de la classe
```

Console

```
true AND true = true
true AND false = false
false AND false = false
true OR true = true
true OR false = true
false OR false = false
NOT true = false
NOT false = true
(3 > 4 ) AND true = false
```

• De 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 auna 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

• 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.**

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 é s 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 conegues què fan.

1. Operadors postfixos: [ ] . (parèntesi)
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: =, +=, -=, *=, /=, %=, ^=, &=, |=, >>=, <<=