Tipus primitius:

Tipus	Ús	Mida	Rang
`byte`	sencer curt	8 bits	-128 a +127
`short`	sencer	16 bits	-32.768 a +32.787
`int`	sencer	32 bits	+-2.147.483.648
`long`	sencer llarg	64 bits	+- 9.223.372.036.854.775.808
`float`	real	32 bits	de -10^32 a +10^32
`double`	real doble	64 bits	de -10^300 a 10^300
`boolean`	lògic	1 bit	`true` o `false`
`char`	texte	16 bits	qualsevol caràcter

Operadors aritmètics:

Símbol	Descripció
+	Suma
+	Positiu
-	Resta
-	Negatiu
*	Multiplicació

Símbol	Descripció
/	Divisió
%	Mòdul
++	Increment en 1
--	Decrement en 1

Operadors aritmètics - exercicis:

1. Crear un programa que demani l'edat i mostri l'edat que tindrà l'any vinent.

2. Crear un programa que demani l'any actual i l'any de naixement, i mostri l'edat de l'usuari a dia 31 de Desembre

3. Crear un programa que calculi i mostri la mitja aritmètica de dues notes de valor sencer.

4. Crear un programa que calculi l'àrea i longitud d'un cercle.

   Recordatori:

   $Longitud = 2\pi \cdot radi$

   $Àrea = \pi \cdot radi^2$

Operadors relacionals:

Símbol	Descripció
==	Igual que
!=	Diferent
<	Menor que
<=	Menor o iqual que
>	Major que
>=	Major o igual que

Operadors lògics:

Símbol	Descripció
& &	AND (y)
| |	OR (o)
!	NOT (negació)

Taules de veritat

Operadors amb assignació:

Símbol	Descripció
+=	Suma i assigna
-=	Resta i assigna
*=	Multiplica i assigna
/=	Divideix i assigna
%=	Mòdul i assigna

Operador ternari:

```
condició ? resultat_vertader : resultat_fals;
```

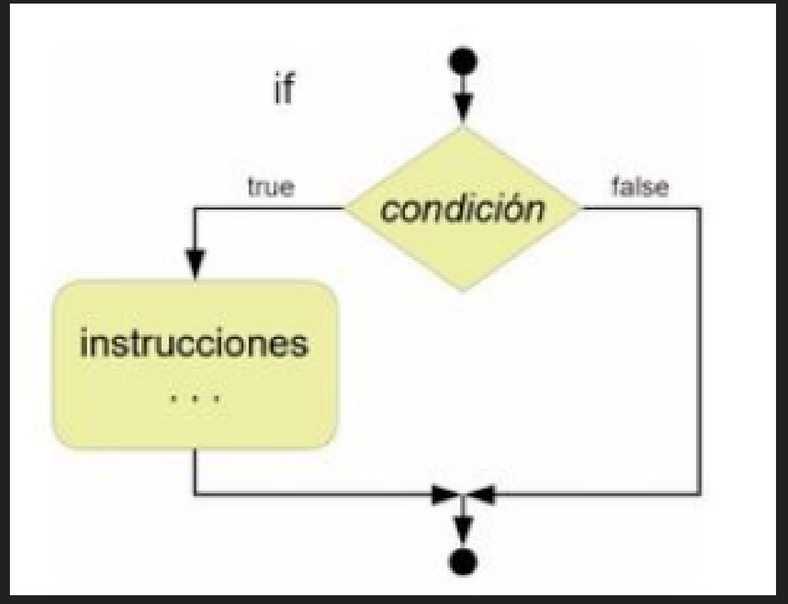
Exemple:

```
int a, b;
a = 3 < 5 ? 1 : -1;
b = a == 7 ? 10 : 20;
```

Casting de variables

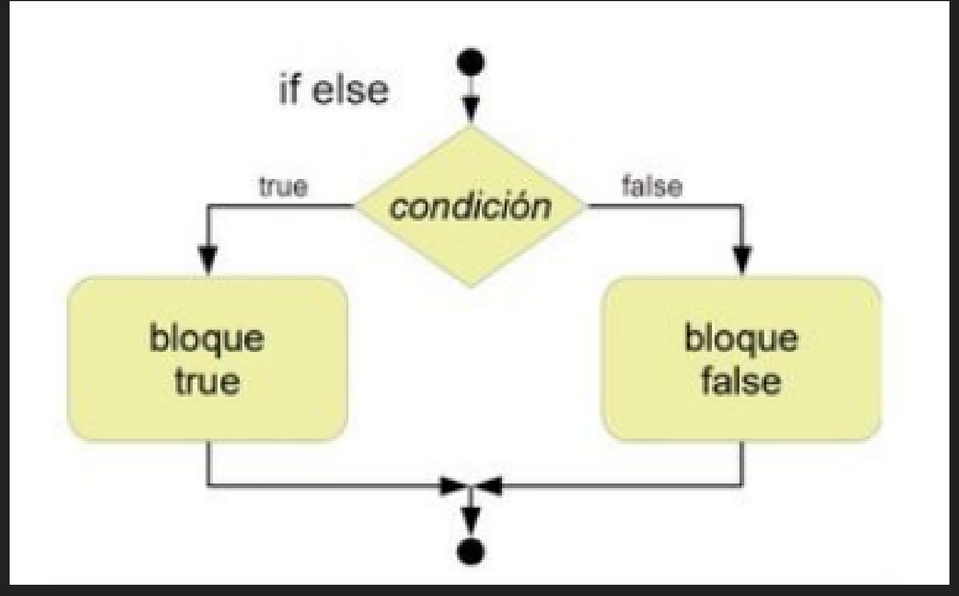
Condicional simple: `if`

```
if (condició) {
      …
      instruccions
      …
}
```



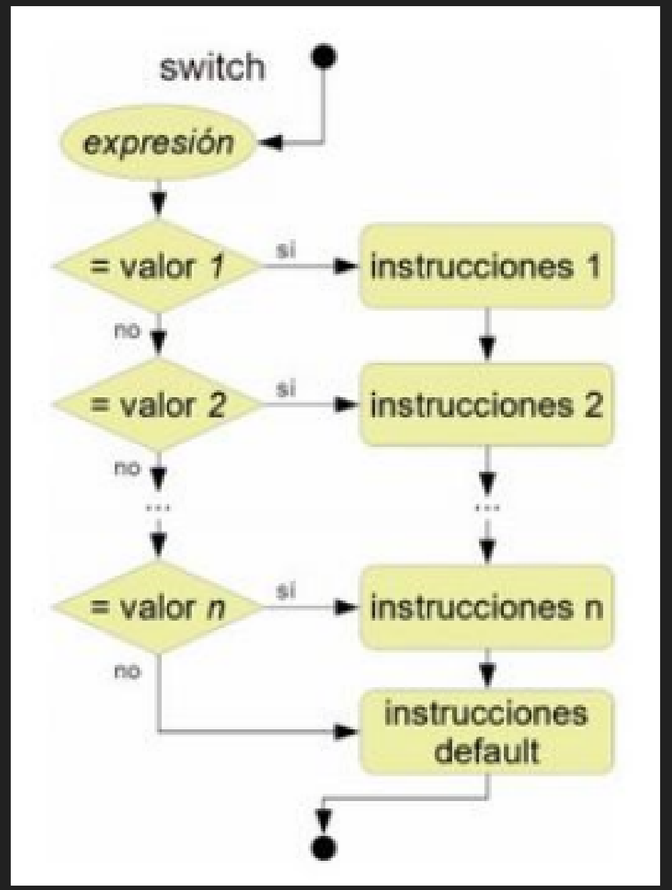
Condicional doble: `if - else`

```
if (condició) {
     …
     instruccions
     …
} else {
     …
     instruccions
     …
}
```



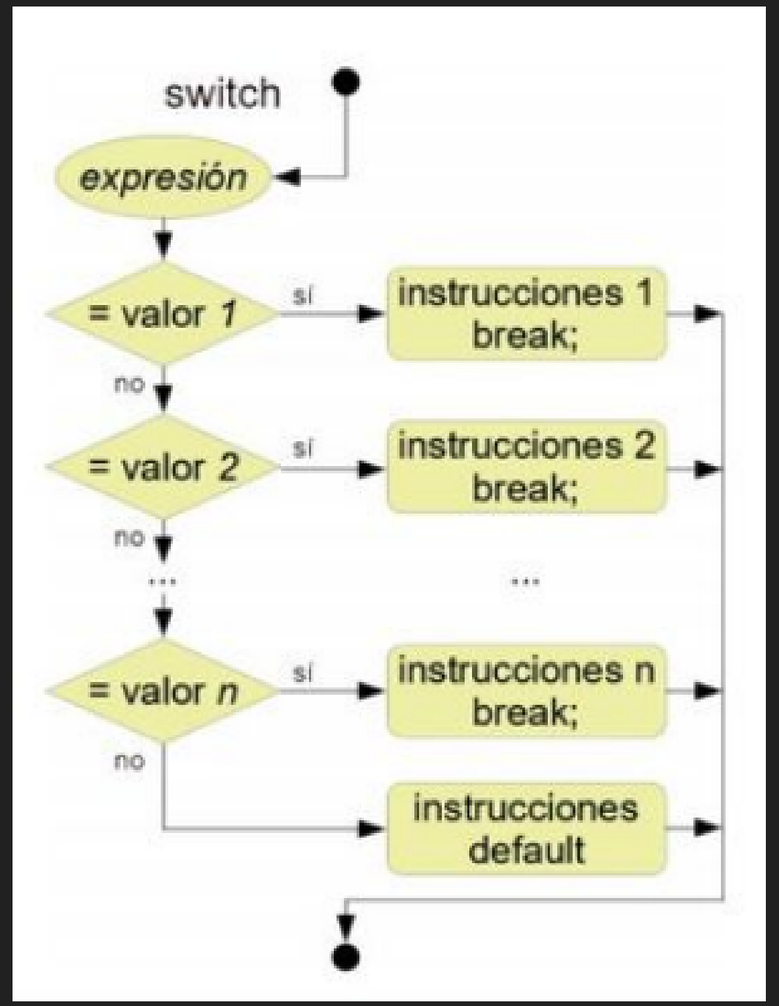
Condicional múltiple: `switch`

```
switch(expressió) {
     case valor1:
          …instruccions…
     case valor2:
          …instruccions…
     …
     case valorN:
          …instruccions…
     default:
          …instruccions…
}
```



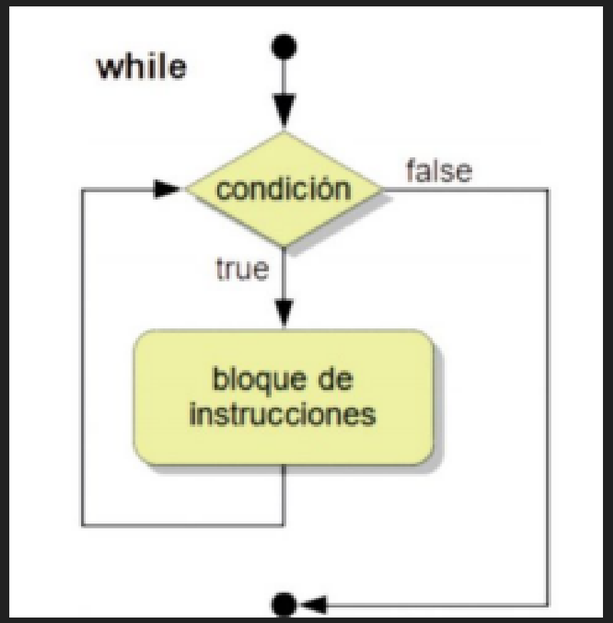
Condicional múltiple: `switch` amb `break`

```
switch(expressió) {
     case valor1:
          …instruccions…
          break;
     case valor2:
          …instruccions…
          break;
     …
     case valorN:
          …instruccions…
          break;
     default:
          …instruccions…
}
```



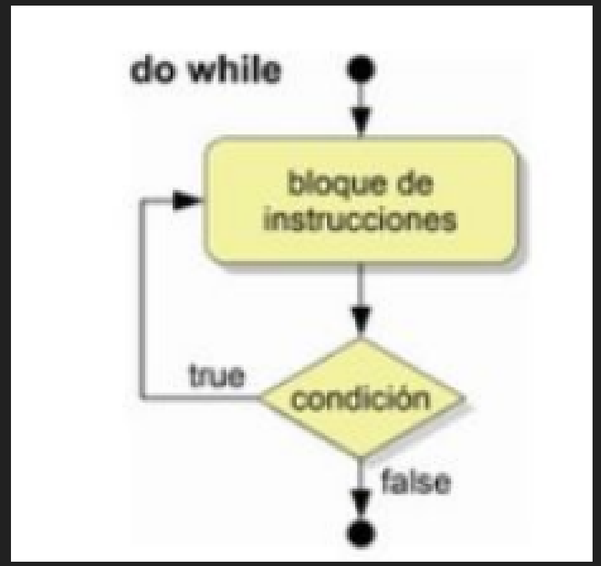
Bucle condicional: `while`

```
while (condició) {
     …
     instruccions
     …
}
```



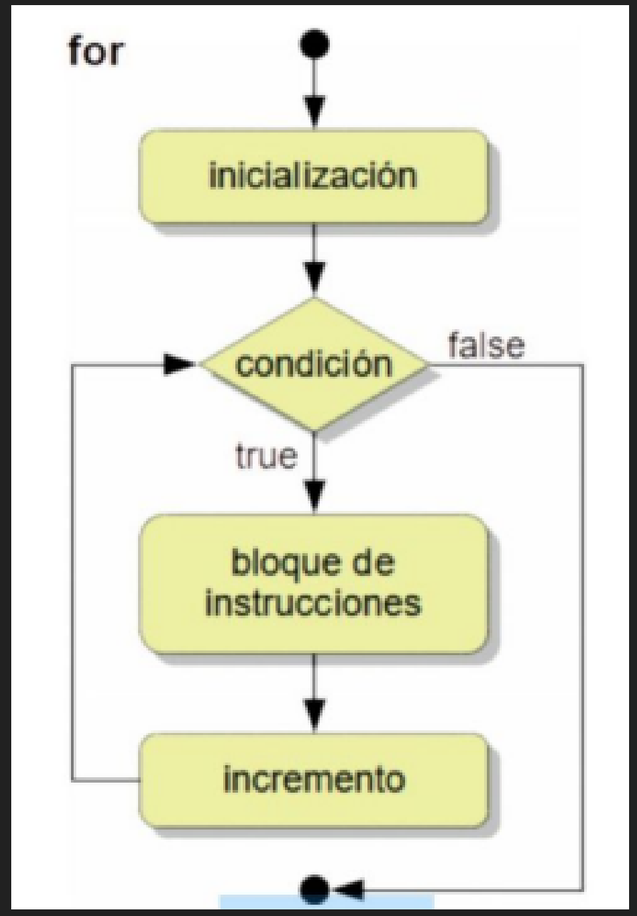
Bucle condicional: `do - while`

```
do {
     …
     instruccions
     …
} while (condició);
```



Bucle comptador: `for`

```
for (inicialització; condició; increment) {
     …
     instruccions
     …
}
```

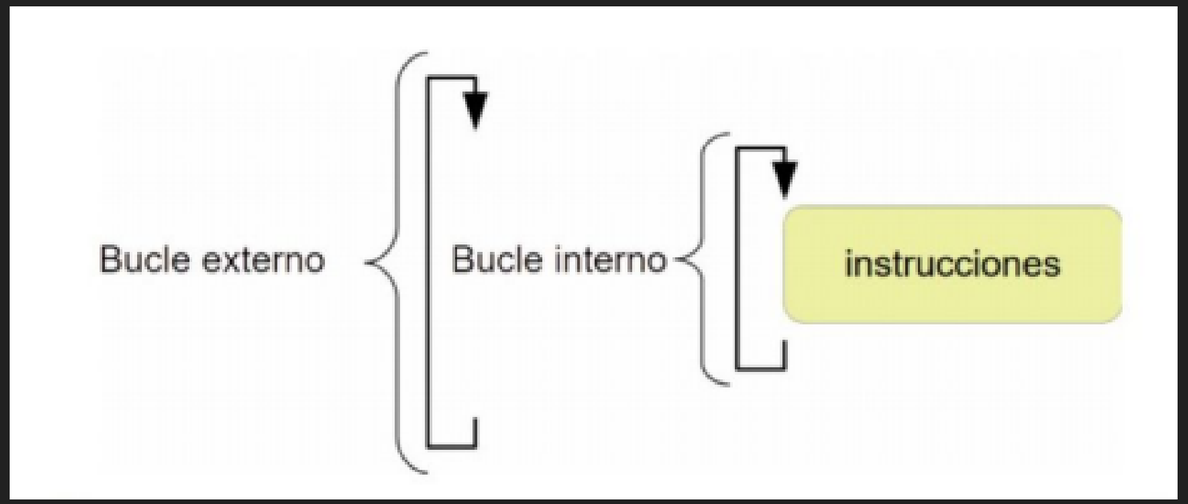


Sortides anticipades: `break` i `continue`

```
i = 1;
while (i <= 10) {
    System.out.println("La i val " + i);
    if (i == 2) {
        break;
    }
    i++;
}

i = 0;
while (i < 10) {
    i++;
    if (i % 2 == 0) {
        continue;
    }
    System.out.println("La i val " + i);
}
```

Bucles encapsulats:




```
for (int i = 1; i <= 4; i++) {
    for (int j = 1; j <= 3; j++) {
        System.out.println("Execució i="+i+" j="+j);
    }
}
```

Conceptes bàsics

- Evitar repetir estructures de codi
- Conjunt d'instruccions agrupades sota un nom i amb un objectiu comú
- Àmbit de les variables
- Notació a Java

```
static void numFunció() {
     …
     instruccions
     …
}

main() {
     nomFunció();
}
```

Pas d'informació

```
public static void numFunció(tipus nomVariable) {
    …
    instruccions usant nomVariable
    …
}
```

Retorn d'un valor

```
static tipus numFunció( … ) {
    …
    instruccions
    …
    return variable;  // Del tipus correcte
}
```

Sobrecàrrega de funcions

```
static int suma(int a, int b) {
    int suma = a + b;
    return suma;
}

static double suma(int a, float pesA, int b, float pesB) {
    double suma = a*pesA + b*pesB;
    return suma;
}

public static void main() {
    int resultat1 = suma(4, 5);
    double resultat2 = suma(4, 0.25, 3, 0.75);
    System.out.println(resultat1);
    System.out.println(resultat2);
}
```

Recursivitat

```
static int funcioRecursiva() {
    …
    funcioRecursiva();
    …
}
```

Exemple: resoldre n! = n · (n-1) · (n-2) · … · 2 · 1

```
long factorial(int n) {
    long resultat;
    if (n==1) {
        resultat = 1;
    } else {
        resultat = n * factorial(n-1);
    }
    return resultat;
}
```

Cas base: Si n és 1
el resultat és 1

Variables globals

```
public class JavaApplication {

    static tipus variable;

    public static void main(String[] args) {
        …
        instruccions
        …
    }

    static tipus funcio() {
        …
        instruccions
        …
    }
}
```

Conceptes bàsics

- Ampliar la capacitat d'una variable. De un únic valor, fins a n-valors, segons la declaració.
- Llistes i índexs:

n	0	1	2	3	4
edat[n]	15	28	72	18	64

- Índexs fora de rang ¿?
- Els arrays han de tenir longitud i tipus.

Declaracions de arrays

Declaració:

```
tipo nomVariable[]     <->     tipo[] nomVariable
```

Assignació:

```
nomVariable = new tipo[longitud];
```

Exemples:

```
int edat[];
edat = new int[5];
```

O directament:

```
int edat[] = new int[5];
```

Referències dels arrays

- La variable a on s'assigna l'array no és l'array en sí, sinó una referència a la secció de memòria a on es troba l'array.

  Prova de fer el següent:

```
int[] edat = new int[5];
System.out.println(edat);
```

- Quan es passa un array com a paràmetre d'una funció, no es fa una còpia de l'array, sinó únicament de la referència. ES MODIFICA LA TAULA A TOT EL PROGRAMA.

Utilitats dels arrays

Eines de la API Java Arrays. Requereix la importació de java.utils.Arrays.

```
import java.util.Arrays;
```

Llegir la longitud d'un array:

```
int[] array = new int[5];
System.out.println(array.length); //  Mostra 5 per pantalla.
```

Inicialització a un valor:

```
double[] mostra = double[15];
Arrays.fill(mostra, 0.5);       //  Arrays.fill(tipo array, tipo valor);
```

Utilitats dels arrays

Iterar un array:

```
for (int i = desde; i <= hasta; i++) { }
```

```
p.e.:
double[] llista = {64, 32, 3, 18, 52};
for (int i=0; i<5; i++) {
    llista[i] = llista[i] * 1.1;
}
```

Operador for-each:

```
double[] mostra = double[15];
for (double valor : mostra) {  //  Valor prendrà el valor de cada posició
    System.out.println(valor); //  A cada iteració, mostra el valor
}
```

Ordenació

- Ordenar un array és una operació computacionalment costosa. Només es fa quan és necessari executar cerques a dins la llista.

A les utilitats d'Arrays hi ha un mètode per a facilitar l'ordenació:

```
int[] edad = {85, 12, 44, 22, 52};
Arrays.sort(edad); // La llista ordenada: {12, 22, 44, 52, 85}
```

Cerca a dins un array

Cerca a un array no ordenat:

- Implica fer una cerca seqüencial a tot l'array
- No és eficient

```
/* cerca seqüencial */
int indexCerca = 0;
while (indexCerca < t.length && t[indexCerca] != termeCerca) {
    indexCerca++;
}
if (indexCerca < t.length) {
    … // s'ha trobat l'element i es troba a la posició indexCerca
} else {
    … // no s'ha trobat l'element
}
```

Cerca a dins un array

Cerca a un array ordenat:

- Pot no requerir fer una cerca seqüencial
- Cerca dicotòmica (o binària)
- Existeix una cerca dicotòmica a la classe Arrays:

```
static tipo binarySearch(tipo[] array, tipo claveBusqueda)
```

```
/* cerca no seqüencial */
int indexCerca = Arrays.binarySearch(precios, 19.95);
if (indexCerca >= 0) {
    System.out.println("Trobat a la posició: " + indexCerca);
} else {
    System.out.println("No trobat");
}
```