

# Programació estructurada

## Objectius

- S'ha escrit i provat codi que fa ús d'estructures de selecció.
- S'han utilitzat estructures de repetició.
- S'han reconegut les possibilitats de les sentències de salt.
- S'han creat programes executables utilitzant diferents estructures de control.
- S'han creat mètodes propis.
- S'han escrit cridades a mètodes estàtics.
- S'han utilitzat paràmetres en la cridada a mètodes.
- S'ha utilitzat l'entorn integrat de desenvolupament en la creació i compilació de programes.

## Teorema de la programació

El **teorema de la programació** estructurada estableix que tota funció computable pot ser implementada en un llenguatge de programació que combina només tres estructures:

- **Seqüencial**: execució d'una sentència
- **Selecció**: execució d'una sentència o d'una altra, en funció del resultat d'una expressió booleana (condició)
- **Iteració**: execució d'una sentència mentre una condició es compleix (cicle o bucle)

## Seqüencial

Una sentència seqüencial no altera el flux d'execució.

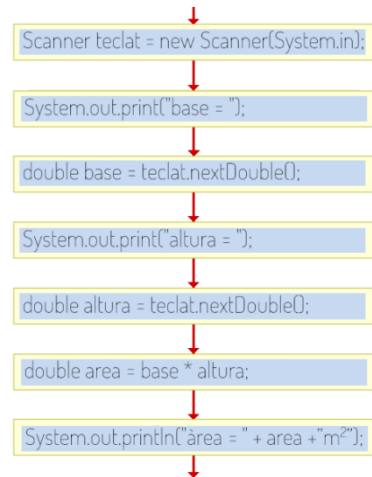
S'executa una sentència i quan termina s'executa la següent.

Totes les sentències que hem usat fins al moment (llegir, visualitzar, sumar, restar, etc.) són seqüencials, s'executen una després de l'altra.

En la figura la seqüència d'accions és:

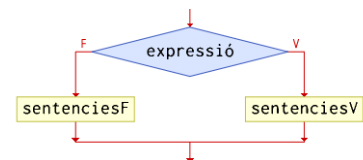


1. Crea un objecte i assigna l'objecte a una referència.
2. Visualitza un text.
3. Llig un double i defineix una referència amb un valor inicial.
4. Visualitza un text.
5. Llig un double i defineix una referència amb un valor inicial.
6. Multiplica doubles i assigna el resultat a una referència.
7. Concatena un text amb el resultat i visualitza el text resultant.



## Selecció

Les sentències de selecció permeten prendre decisions, en les quals s'avalua una condició i en funció del resultat s'executen accions diferents.



A Java una selecció s'escriu de la forma següent

```

if (expressió)
    sentènciaV;    // s'executa si expressió és vertadera
else
    sentènciaF;    // s'executa si expressió és falsa
  
```

S'avalua *expressió*, si és vertadera, llavors s'executa la *sentènciaV*, si és falsa, s'executa la *sentènciaF*.

Els parèntesis són obligatoris.

El resultat de *expressió* és booleà, pot ser una expressió simple o composta, amb operadors relacionals o lògics.

La sentència a executar pot ser única o ser un bloc de sentències.

→ Utilitza sempre un bloc, encara que la sentència siga única. Escriu les sentències del bloc sagnades.

```

int a = 3;
int b = 9;
if (a < b && a != 0) {
    b = b - a;
    a = -a;
} // l'execució continua en la sentència a++
else {
    b = a - b;
}
a++;
  
```



El bloc de la falsedat (**else**) és optatiu. La frase '*si tens més de 18 anys, llavors, visualitza "eres major"*', s'escriu

```
if (edat > 18) {  
    System.out.println("eres major");  
}
```

☠☠☠☠ No deixes el bloc corresponent a vertader buit, canvia la condició ☠☠☠☠

→ Una sentència de selecció és una sentència seqüencial.

Niar sentències de selecció, suposa escriure una sentència de selecció dins del bloc de veritat o falsedat d'una sentència de selecció anterior. Usa el mateix sagnat per als dos blocs del **if**.

```
descompte = 0.02f;           // descompte és un float  
if (punts > 1000) {           // punts és un enter  
    descompte = 0.05f;  
    if (tipusClient.equals("VIP")) { // tipusClient és una cadena  
        descompte = 0.07f;  
        if (diesAntiguitat > 1825) { // diesAntiguitat és un enter  
            descompte = 0.1f;  
        }  
    } else {                   // si no de tipusClient.equals("VIP")  
        if (diesAntiguitat > 1825) {  
            descompte = 0.06f;  
        }  
    }  
}
```

Cal anar amb compte a l'hora de niar els **if**, i tancar els blocs correctament perquè l'associació del **if** amb el **else** siga la correcta.

→ Un **else** tanca l'**if** immediatament anterior.

☠☠☠☠ No usar la indentació BAIXA LA NOTA ☠☠☠☠

## L'operador ?:

L'operador **?:** permet realitzar una sentència de selecció dins d'una expressió, el seu format és

```
expressióBooleana ? valorSiVertadera : valorSiFalsa
```

S'avalua l'expressió booleana, si és vertadera, llavors se substitueix l'expressió completa pel valor escrit entre el **?** i el **;**, si és falsa, se substitueix pel valor escrit entre **:** i **;**. Els valors han de ser compatibles.

```
int x = 1;  
int num = x > 0 ? 3 : 4; // defineix num i li assigna el valor 3, ja que x>0 és veritat
```



Si  $x$  és major que 0, llavors l'expressió se substitueix per 3, si és falsa, se substitueix per 4, el valor final s'assigna a `num`, en l'exemple s'assigna 3 a `num`

```
int x = 1;
boolean esMajor = false;
int num = (x > 0 ? 3 : 4) + (esMajor ? 0 : 1) + ((x > 0) || esMajor) ? 3 : 4; // defineix num i li assigna el valor 7
```

L'expressió pot estar en qualsevol lloc on s'accepta una expressió.

```
int edad = 12;
System.out.println(edad >= 18 ? "és major d'edat" : "és menor"); // visualitza "és menor"
```

## Selecció múltiple

La sentència de selecció múltiple té més de dues alternatives, a Java la sentència és `switch` i el seu format és

```
switch (expressió) { // expressió a avaluar
    case valor1 : // punt d'entrada si expressió és igual a valor1
        sentències1 // sentències a executar si expressió és igual a valor1
        break; // el break salta al final del switch
    case valor2 : // punt d'entrada si expressió és igual a valor2
        sentències2 // sentències a executar si expressió és igual a valor2
        break; // el break salta al final del switch
    ...
    default: // si no es compleix cap dels case
        sentènciesD // sentències a executar si expressió no és igual a cap valor dels case
}
```

L'expressió pot ser dels tipus següents: `byte`, `short`, `int`, `char`, `enum`, `String`

Els valors han de ser del mateix tipus que l'expressió i no es poden repetir.

→ Una sentència de selecció múltiple busca la IGUALTAT exacta de l'expressió amb el valor. No es poden usar condicions o rangs de valors.

Si no es compleix cap igualtat, s'executa l'opció `default` (és optativa), si s'escriu, és l'última. No es posa el `break`, ja que el bloc es tanca just darrere.

```
switch (tipus) { // tipus és un caràcter
    case 'A': System.out.println("pot conduir motocicletes"); break; // quan tipus val 'A'
    case 'B': System.out.println("pot conduir cotxes"); break; // quan tipus val 'B'
    case 'C': System.out.println("pot conduir camions"); break; // quan tipus val 'C'
    default: System.out.println("no pot conduir"); // quan tipus no val 'A', 'B' ni 'C'
}
```

El `switch` sols té un bloc de sentències, i els `case` defineixen diferents punts d'entrada al bloc de sentències. Els `case` i el `default` no formen part de les sentències del bloc.

El `break` defineix el conjunt de sentències associades a un `case`.



Si no es posa el **break**, s'executa la sentència següent, encara que estiga en un **case** diferent. El **break** "trenca" el bloc, és a dir, l'execució salta al final del bloc del **switch**.

Es pot posar un **case** sense codi, per tant, s'executa el codi següent.

```
switch (valor) { // valor és un enter
    case 1:
    case 2: porcen = porcen * 0.2; break; // s'executa quan valor val 1 o 2
    case 3: porcen = porcen * 0.15; break; // s'executa quan valor val 3
    case 4: porcen = porcen * 0.1; // s'executa quan valor val 4
    case 6: porcen = porcen * 1.1; // s'executa quan valor val 4 o 6
    case 7: porcen = porcen * 1.3; break; // s'executa quan valor val 4 o 6 o 7
    default: porcen = porcen * 1.18; // s'executa quan valor no val 1, 2, 3, 4, 6 ni 7
}
```

Els casos no tenen per què estar ordenats, però és la forma habitual, el primer **case** que compleix la igualtat indica la sentència on comença l'execució del bloc del **switch**.


```
switch (div != 0 ? num % div : num - div) { // el resultat de l'expressió és un enter
    case 3: System.out.println("tres"); break;
    case 2: System.out.println("dos"); break;
    case 0: System.out.println("zero"); break;
    case 4: System.out.println("quatre"); break;
    default: System.out.println("és un altre (> "+div != 0 ? num % div : num - div));
}
```

L'ordre no importa, però habitualment els valors s'escriuen ordenats i **default** és l'últim.

En l'exemple següent s'avalua una cadena de text en l'expressió

```
Scanner lector = new Scanner(System.in);
System.out.print("entra el nombre: ");
int A = lector.nextInt();
System.out.print("entra el nombre: ");
int B = lector.nextInt();
System.out.print("entra l'operació (suma, resta, multiplica, divideix): ");
String operacio = lector.next();
switch (operacio) { // operacio és una cadena de text
    case "suma": System.out.println("suma = " + (A + B)); break;
    case "divideix": System.out.println("divisió = " + (A / B)); break;
    case "resta": System.out.println("resta = " + (A - B)); break;
    case "multiplica": System.out.println("multiplicació = " + (A * B)); break;
    default: System.out.println("operació no suportada");
}
```

En l'exemple següent s'avalua una enumeració en l'expressió, una enumeració és un conjunt discret de valors (els punts cardinals, els mesos, els noms dels dies, etc.).

 Mira el tema "tv03a\_enumeració.docx".

```
public enum DiaSetmana { // L'enumeració DiaSetmana conté els noms dels dies de la setmana
    DILLUNS, DIMARTS, DIMECRES, DIJOUS, DIVENDRES, DISSABTE, DIUMENGE;
```



```
}
```

```
DiaSetmana dia = DiaSetmana.DIJOUS;           // assigna el dijous a dia

switch (dia) {                                  // dia és una enumeració de tipus DiaSetmana
    case DILLUNS:
    case DIMARTS :
    case DIJOUS:                               // punt d'entrada al switch
    case DIVENDRES:    valor = 5.90; break;
    case DIMECRES:     valor = 4.90; break;
    case DISSABTE:
    case DIUMENGE:     valor = 6.90; break;
}
```

## Format lambda

A partir de Java 12 tenim una versió de `switch` que usa expressions lambda. Els valors en un `case` se separen amb coma i les sentències a executar s'escriuen en un bloc després de ->

```
switch (expressió) {                          // expressió a avaluar
    case valor1, valor2, ... ->               //si expressió és igual a alguns dels valors, s'executen les sentències
    { sentències }
    case valor3, valor4, ... ->               //si expressió és igual a alguns dels valors, s'executen les sentències
    { sentències }
    ...
    default ->                               //si no es compleix cap dels case anteriors, s'executen les sentències
    { sentències }
}
```

Si les sentències són una sentència simple, pots eliminar les claus.

```
String operacio = lector.next();
switch (operacio) {                          // operacio és una cadena de text
    case "suma" ->                           System.out.println("suma = " + (A + B));
    case "divideix" -> {                     // aquí necessitem un bloc per a tancar les sentències
        System.out.println(A + " / " + B + " = " + (A / B) + " divisió entera");
        double res = (double) A / B;
        System.out.println(A + " / " + B + " = " + res + " divisió amb decimals");
    }
    case "resta" ->                           System.out.println(A + " - " + B + " = " + (A - B));
    case "multiplica" ->                     System.out.println("multiplicació = " + (A * B));
    default ->                               System.out.println("operació no suportada");
}
```

El `switch` es pot convertir en una expressió

```
int elNumero = lector.nextInt();
String qualificació = switch (elNumero) {    // el switch se substitueix pel valor tornat en el case
    case 7, 3, 5, 13 -> "nombre de la sort molt popular";
    case 8 ->          "nombre molt popular per als xinesos";
    case 2, 9, 6 ->    "nombre de la sort popular ";
    default ->         "No és popular";
};
```



```
System.out.println(elNumero+" és un " + qualificació);
```

El switch se substitueix pel valor que apareix en el `case`, les sentències han de substituir-se per un valor (no pot haver-hi un `return` en les sentències)

Quan s'utilitza com a expressió, el tractament dels casos ha de cobrir totes les possibilitats de l'expressió, cal posar el `default`. Si l'expressió és un `enum` i es tracten tots els valors no cal posar el `default`.

```
double valor = switch (dia) { // dia és una enumeració de tipus DiaSetmana
    case DILLUNS, DIMARTS, DIJOUS, DIVENDRES -> 5.90;
    case DIMECRES -> 4.90;
    case DISSABTE, DIUMENGE -> 6.90;
};
System.out.println("Preu pizza: " + valor + "€");
```



## Exercicis

### 1. descomptearticles

Escriu un programa que calcula l'import que cal pagar per uns articles.

Llig de teclat: el nombre d'articles, el preu d'un article, si cal aplicar o no un descompte, si hi ha descompte (llig un booleà), llavors llig el percentatge de descompte a aplicar (un nombre amb decimals). L'IVA que s'aplica als articles és del 21%.

Un cop has llegit les dades visualitza l'import brut (preu total sense descompte), el descompte aplicat, l'iva a afegir i l'import net (preu total amb iva i descompte), totes les quantitats que es visualitzen són diners, per tant, euros.

### 2. diasetmana

Escriu un programa que llig el dia de la setmana (un valor de l'1 al 7) el visualitza el seu nom. Els noms de la setmana comencen en "**Dilluns**" que correspon al valor 1 i acaben en "**Diumenge**" que correspon a 7.

Si no s'introdueix un valor de l'1 al 7, llavors es visualitza un missatge d'error.

### 3. recepta

Escriu un programa que visualitza les quantitats necessàries per a realitzar una recepta.

Llig de teclat per a quantes persones és la recepta. Si s'introdueix un valor inferior a 1, llavors es calcula les quantitats per a una persona.

La recepta per a sis persones té: 600gr de coliflor, 1 ceba tendra, 3 carlotes, 500gr de pit de pollastre, 25gr de pinyons, 75gr de panses, 220ml de caldo i 1.5 culleretes de curri.

Tin en compte que, les carlotes, sempre, són senceres, i com a mínim en la recepta hi ha mitja ceba tendra, una carlota, 10gr de pinyons, 20gr de panses i 0.5 culleretes de curri.

### 4. bitllettren

Escriu un programa que calcula el preu d'un bitllet de ferrocarril.

Si és un bitllet senzill, el preu és de 0.33€ el quilòmetre.

Si és un bitllet d'anada i tornada, el preu és de 0.26€ el quilòmetre

Si el preu és superior a 50€, hi ha un descompte del 5%

Si el preu és superior a 110€, hi ha un descompte del 10%.

Si el client presenta la targeta Interrail, hi ha un descompte del 25%.

Els descomptes no són acumulables, s'aplica el major.



## 5. marato

Escriu un programa que mostra si un corredor pot participar en una marató.

Per a seleccionar a un corredor, ha d'haver acabat una marató anterior en un temps inferior a:

- 190 minuts per a homes menors de 35 anys
- 225 minuts per a homes majors de 35 anys
- 240 minuts per a dones

Si el corredor és menor d'edat (18 anys) o no ha corregut una marató anterior, llavors, no pot participar.

Les dades del corredor a introduir són: el sexe, l'edat i el temps efectuat en la seua anterior marató. No s'han d'introduir dades innecessàries.

El programa visualitza els missatges "Seleccionat", "No seleccionat", "Seleccionada", "No seleccionada" o "No pot participar" en funció de les dades introduïdes.

## 6. fesoperacio

Escriu un programa que llig dos nombres i l'operació a realitzar amb ells, i després presenta el resultat de l'operació.

Els símbols que defineixen les operacions són els caràcters '+', '-', '\*', '/', '%', i l'operació a realitzar és l'operació aritmètica corresponent. Si s'introdueix altre símbol, es visualitza un missatge d'error. Per a la suma i la multiplicació es mostra un únic resultat, per a la resta, la divisió i el mòdul es mostren dos resultats (intercanviant la posició dels operands). Si es llig 5, 3, +, llavors es mostra  $5 + 3 = 8$ , si es llig 5, 3, -, llavors es mostra  $5 - 3 = 2$  i  $3 - 5 = -2$

## 7. qualificacioliteral

Escriu un programa que mostra una qualificació literal d'un estudiant.

El programa captura un valor numèric del teclat i visualitza la qualificació corresponent.

Els criteris de transformació de la nota són:

de 8.5 a 10, llavors és "Excel·lent"

de 7 a 8.5, llavors és "Notable"

de 6 a 7, llavors és "Bé"

de 5 a 6, llavors és "Suficient"

de 3.5 a 5, llavors és "Insuficient"

de 0 a 3.5, llavors és "Molt deficient"

Si una nota no està entre 0 i 10, llavors és "incorrecta".

## Bucles

Per a repetir l'execució d'un bloc de sentències, s'usa una sentència d'iteració o de bucle.

Una execució del bloc de sentències d'un bucle s'anomena iteració.

## Elements d'un bucle

Tots els bucles tenen els elements següents





- **inicialització**: que assigna els valors inicials necessaris perquè la condició de final tinga sentit la primera vegada que es comprova. És prèvia al bucle i s'executa una única vegada.
- **condició de final**: que finalitza el bucle, es pot utilitzar qualsevol expressió que tinga un resultat booleà. S'executa en cada iteració.
- **variació**: que canvia els valors dels elements que participen en la condició de final, perquè es puga complir i així finalitzar el bucle. S'executa en cada iteració.
- **sentències a repetir**: s'escriuen dins del bloc del bucle, s'executen en cada iteració. S'escriuen indentadas per a indicar la seua pertinença al bucle.

Aquest ordre dels elements pot canviar, depén de l'estructura del bucle.

→ La condició de final s'ha de complir SEMPRE, si no, el bucle no acaba (és infinit) i el programa es penja.

Procura que la variació estiga el més a prop possible de la condició, no poses codi entre la sentència de variació i la sentència de condició, així t'assegures que no canvia el valor de la variació.

En Java tenim les sentències `while`, `do while` i `for` per a realitzar bucles

→ En Java tots els bucles es repeteixen mentre la condició és vertadera. Associa fer un bucle amb la frase en REPETIR MENTRE.

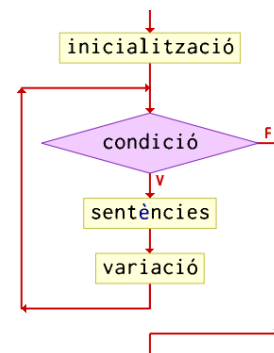
## Condició a l'inici

A Java els bucles amb la condició a l'inici són `while` i `for`

### while

El format del `while` és el següent

```
inicialització;
while (expressió) {
    sentències
    variació
}
```



→ L'expressió és booleana, el bloc de sentències es repeteix mentre l'expressió és vertadera.

La inicialització defineix el valor de l'expressió en la primera iteració.

La variació ha de fer que l'expressió passe a ser falsa, per a finalitzar el bucle.

```
int num = 1;           // inicialització
while (num < 100){     // condició de final, mentre num és menor que 100
```



```

if (num % 3 == 0) {           // sentència a repetir
    System.out.print(num + ","; // sentència a repetir
}                             // sentència a repetir
num++;                       // variació de num
}

```

En l'exemple, es visualitzen tots els múltiples de 3 des de l'1 al 99. La repetició es realitza mentre `num` és menor que 100, `num` comença en 1 i se li suma 1 en cada iteració. En cada iteració si `num` és divisible per 3, llavors es visualitza `num`.

Els bucles es poden niar, és a dir, escriure un bucle dins d'un altre, ja que el `while` és una sentència única.

## Niar

En bucles niats el nombre de repeticions de cada bucle es multipliquen.

→ El bucle intern necessita la inicialització en cada iteració del bucle extern.

En l'exemple següent, el bucle extern es repeteix 3 vegades i el bucle intern es repeteix 5 vegades, per tant la sentència `System.out.print("(" + i + "," + j + ")");` s'executa 15 vegades (3\*5).

```

int i = 0;                // inicialització del bucle extern
while (i < 3){             // condició de final del bucle extern
    int j = 0;             // inicialització del bucle intern
    while (j < 5){         // condició de final del bucle intern
        System.out.print("(" + i + "," + j + ")"); // sentència a repetir
        j++;              // variació del bucle intern
    }
    i++;                  // variació del bucle extern
}
// visualitza

```

```

(0,0)(0,1)(0,2)(0,3)(0,4)(1,0)(1,1)(1,2)(1,3)(1,4)(2,0)(2,1)(2,2)(2,3)(2,4)

```

## for

El `for` és una forma abreujada d'escriure un bucle, tots els elements que intervenen en el bucle, inicialització, expressió i variació s'escriuen en una sola línia, el seu format és:

```

for (inicialització; expressió; variació) {
    sentències
}

```

La inicialització s'executa una vegada, a l'inici del `for`. La inicialització del `for` segueix les normes de declaració de variables locals, per tant, poden declarar diverses variables del mateix tipus separades per comes.

A l'inici de cada iteració, s'avalua expressió, si és vertadera, llavors s'executen les sentències.



Al final de cada iteració es realitza la variació, es pot expressar diverses variacions separant-les per comes.

El `for` és una sentència única i finalitza en la clau de tancament del bloc de sentències.

```
for (int i = 3; i < 7; i++){           // visualitza els enters del 3 al 6
    System.out.println(i);
} // destrueix la variable local i
```

- **Inicialització:** `int i = 3` (defineix `i` com a enter local al `for` i li dona valor 3)
- **Condicció:** `i < 7` (mentre que `i` siga menor que 7, el 7 finalitza el bucle)
- **Variació:** `i++` (la variació del nombre és d'1 en 1)
- **Sentència:** `System.out.println(i);` (visualitza `i`)

```
double k = 5.571;
for (int i = 0, j = 33; i < 10 && k < 8; i++, j--, k += 0.339) { // les variables i i j són locals
    System.out.printf("i = %d j = %d k=%f\n", i, j, k);
}
```

- **Inicialització:** `int i = 0, j=33` (defineix `i` amb el valor 0 i `j` amb el valor 33, són dos enters locals al `for`)
- **Condicció:** `i < 10 && k < 8` (mentre que `i` siga menor que 10 i `k` menor que 8)
- **Variació:** `i++` (la variació del nombre és d'1 en 1)
- **Sentència:** `System.out.println(i);` (visualitza `i`)

L'element de variació es pot modificar en el bloc de sentències. La variació s'executa, sempre, després de les sentències del bloc

```
for (int i = 3; i < 7; i++){           // variació
    System.out.println(i);
    i++;                               // variació
}
```

Aquest bucle mostra 3 i 5, ja que `i` s'incrementa en la variació i en la sentència.

```
for (int i=3; i < 7; i++){           // variació
    System.out.println(i);
    i--;                               // contraresta la variació, per tant i no canvia de valor
} // ERROR és un bucle infinit
```

Aquest exemple produeix un bucle infinit, ja que `i++` suma 1 i `i--` resta 1, per tant, `i` sempre val 3 que és menor que 7.

En el `for` tots els elements són optatius, però cal escriure sempre els parèntesis i els punts i comes que separen els diferents elements.

L'inicialització pot estar definida abans del `for`, la variable ja no és local al `for`.

```
int i = 3;                           // inicialització
for (; i < 7; ){
    System.out.println(i);
    i++;                               // variació
}
```



```
}
```

El `for` sense informació crea un bucle infinit.

```
int i = 3;
for (;){                // és un bucle infinit
    System.out.println(i);
    i++;
    if(i==7) break;      // break trenca el bucle
}
```

En el tema de matriu veurem el bucle per a cada (*for each*) que oculta l'ús i el control de la variació.

Niar

El `for` és una sentència única i el bloc de sentències, pot al seu torn contindre altres sentències `for` (bucles niats).

```
for (int i = 3; i < 7; i++) {           // i comença en 3, mentre i és menor que 7, varia sumant 1
    System.out.print(i + " >> ");      // sentència del bucle extern
    for (int j = -5; j < 5; j++) {      // j comença en -5, mentre j és menor que 5, varia sumant 1
        System.out.print(j + " ");
    }
    System.out.println("");            // sentència del bucle extern, escriu un salt de línia
}
// visualitza
3 >> -5 -4 -3 -2 -1 0 1 2 3 4
4 >> -5 -4 -3 -2 -1 0 1 2 3 4
5 >> -5 -4 -3 -2 -1 0 1 2 3 4
6 >> -5 -4 -3 -2 -1 0 1 2 3 4
```

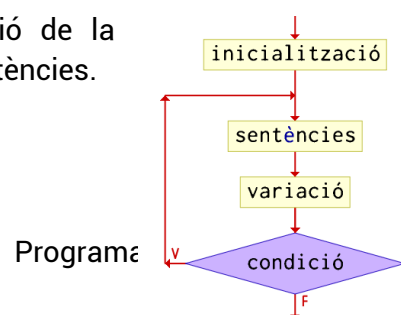
Per a cada valor de *i*, es repeteix tot el bucle sobre la *j*.

Es poden niar `while` dins de `for` i al revés.

```
int i = 0;                          // inicialització del bucle extern
while (i < 3){                       // condició de final del bucle extern
    for (int j = i; j < 5; j++) {     // j comença en i, mentre j és menor que 5, varia sumant 1
        System.out.print("(" + i + "," + j + ")"); // sentència a repetir
    }
    System.out.print("#");           // sentència del bucle extern
    i++;                             // variació del bucle extern
}
// visualitza
(0,0)(0,1)(0,2)(0,3)(0,4)#(1,1)(1,2)(1,3)(1,4)#(2,2)(2,3)(2,4)#
```

## Condició al final

En un bucle amb la condició al final, l'avaluació de la condició de final és posterior a l'execució de les sentències.



→ Les sentències s'executen com a mínim una vegada.

La inicialització pot desaparèixer, llavors en les sentències s'ha d'actuar sobre la variació per a proporcionar els valors necessaris per a l'avaluació de la condició de final.

El bucle es repeteix mentre la condició és vertadera.

A Java el bucle amb la condició al final és `do while`

### do while

El format del `do while` és

```
inicialització;  
do {  
    sentències  
    variació  
} while (expressió);
```

El `do while` és una sentència única, necessita un punt i coma al final.

La inicialització es realitza una sola vegada, l'expressió, les sentències i la variació es realitzen en cada iteració.

→ L'expressió està fora del bloc de sentències, per tant, les variables que s'usen en l'expressió i que es modifiquen en les sentències s'han de definir prèviament al bucle

```
int num = 99;                //inicialització  
do {  
    System.out.print( num + " ");    // sentències  
    num--;                          //variació  
} while (num > 0);            // la condició de final
```

La inicialització prèvia al `do while` és optativa, cal assegurar que l'expressió tinga tots els seus elements amb un valor en el moment de la seua avaluació. La variable per a la l'expressió s'ha de definir prèviament al bloc de sentències del bucle.

```
Scanner lector = new Scanner(System.in);  
int suma = 0;    // suma s'usa en la condició  
int num;        // num s'usa en la condició  
do {  
    System.out.print("entra el nombre");  
    num = lector.nextInt();    //variació de num  
    suma += num;              //variació de suma  
} while (num != 33 && suma <= 1098);    // la condició es pot avaluar, num i suma tenen valor
```

## Salts

Un salt és l'alteració de l'ordre seqüencial d'execució de les sentències. Podem salt en el flux d'execució amb les instruccions: `break`, `continue` i `return`.



## break

El `break` trenca un bucle o un `switch`, és a dir, l'execució salta al final del bucle o `switch`.

Ja hem vist, l'ús del `break` en el `switch`. En un bucle, quan s'executa un `break`, l'execució continua després del final del bloc de repetició del bucle, finalitza el bucle.

```
int i = 4;
while (i > 0) {
    System.out.println("un");
    for (int n = 3; n > 0; n--) {
        System.out.println("dos");
        if (i > 2) {
            break;
        }
        System.out.println("i = " + i + " n = " + n);
    }
    i--;
}
```

// bucle que es repeteix 4 vegades

// bucle que es repeteix 3 vegades

// si i > 2, es finalitza el bucle sobre la n

// salta després del bloc del for

El `break`, només trenca el bucle on es troba, en l'exemple anterior trenca el bucle `for`, però segueix en el bucle `while`.

→ Evita l'ús de `break`, elegeix bé la condició de final del bucle.

💀💀💀💀 l'ús indiscriminat de `break` baixa la nota 💀💀💀💀

## continue

Amb el `continue`, l'execució salta al final del bloc del bucle, és a dir, finalitza una iteració del bucle, però l'execució del bucle segueix.

El `continue` només s'usa en bucles.

→ No uses el `continue`, escriu bé la condició de final del bucle

💀💀💀💀 l'ús de `continue` baixa la nota (MOLT) 💀💀💀💀

```
int i = 4;
while (i > 0) {
    System.out.println("un");
    for (int n = 3; n > 0; n--) {
        System.out.println("dos");
        if (i > 2) {
            continue;
        }
        System.out.println("i = " + i + " n = " + n);
    }
    i--;
}
```

// bucle que es repeteix 4 vegades

// bucle que es repeteix 3 vegades

// si i > 2, es finalitza el bucle sobre la n

// salta al final del bloc de repetició



## return

El `return` finalitza l'execució d'un mètode i salta on es va cridar al mètode, ho veurem més detalladament en el següent apartat.



### Exercicis

#### 8. suma3

Escriu un programa que calcula la suma dels termes de la sèrie 1, 4, 7, 10, ..., els termes varien sumant 3.

Llig de teclat V, el valor del terme en el qual finalitza la sèrie, ha de ser un enter major que 10. V entra en la sèrie. Si V és menor que 10, no es fa res.

Es visualitzen els termes de la sèrie separats pel caràcter "+", excepte per a l'últim terme que no porta el "+" sinó el caràcter "=" seguit de la suma de la sèrie.

Per a V=14 es visualitza 1+4+7+10+13=35

Per a V=16 es visualitza 1+4+7+10+13+16=51

Per a V=30 es visualitza 1+4+7+10+13+16+19+22+25+28=145

Per a V=31 es visualitza 1+4+7+10+13+16+19+22+25+28+31=176

#### 9. serie234

Escriu un programa que calcula la suma dels termes de la sèrie 1, 2, 6, 24, 120, 720, 5040... mentre el terme de la sèrie siga menor de 10.000.000.

Es visualitzen els termes de la sèrie i la suma.

El pas d'un terme de la sèrie al següent és multiplicant el terme actual per 2, 3, 4, 5, 6,...

#### 10. interes

Escriu un programa que calcula els interessos generats per una quantitat de diners.

Llig de teclat: la quantitat de diners depositada inicialment, el nombre d'anys del depòsit i el percentatge d'interés anual.

El programa visualitza els interessos generats i els diners acumulats, per a cada any.

La fórmula per al càlcul és  $\text{interesAnual} = \text{diners} * \text{percentatge} / 100$

L'interés anual generat s'acumula als diners.

#### 11. adivina

Escriu un programa que permet jugar a endevinar un nombre generat de manera aleatòria entre 1000 i 9999.

El jugador introdueix un nombre de teclat, el programa indica si el nombre introduït és major o menor que el que cal encertar. Finalitza quan s'encerta el nombre.

Variació: el jugador només té 12 intents per a encertar el nombre, si el jugador no encerta el nombre, llavors aquest es visualitza.

#### 12. multiples

Escriu un programa que llig dos nombres enters majors que 0, i presenta els múltiples del valor més xicotet, inferiors o iguals al valor més gran. Has d'obligar a l'usuari a introduir valors major que 0.

#### 13. imparellsale

Escriu un programa que llig quants nombres imparells aleatoris entre 2222 i 22222 vols visualitzar i visualitza'ls. Has d'obligar a l'usuari a introduir un valor major que 0. Quan



finalitza el bucle mostra quants números s'han generat en total. Per a facilitar el recompte visualitza 10 imparells per línia com a màxim.

El mètode `Math.random()` genera un valor  $X$  double aleatori dins del rang  $0 \leq X < 1$

## Mètodes

Un mètode és un bloc de codi que s'executa quan es crida, la crida d'un mètode es realitza escrivint el nom que té el mètode. Hi ha dues raons per a crear mètodes

- Evitar la repetició de codi
- Millorar la llegibilitat del codi

→ Si tens línies iguals en el codi, crea un mètode amb elles, i substitueix les línies per la crida al mètode.

→ Si tens línies paregudes en el codi, crea un mètode amb paràmetres.

→ Si pots posar un nom a l'acció que realitza un conjunt de línies de codi, separa-les i crea un mètode amb aquest nom.

Si vols comprovar si un nombre enter és primer, queda més clar el codi

```
if (esPrimer(nombre)){  
...  
}
```

on es crida al mètode `esPrimer`, que escriure el codi complet que realitza la comprovació que un nombre és primer en aquest punt del programa.

L'ús de mètodes divideix el problema en subproblemes més específics, que es resolen més fàcilment.

Un mètode té una capçalera i un bloc de sentències. La capçalera defineix com s'ha de cridar al mètode i el bloc conté el codi que implementa el que fa el mètode.

```
private int sumar (int num1, int num2) {           // capçalera del mètode sumar  
    return num1 + num2;                           // cos del mètode sumar  
}  
  
int suma = sumar(8, 19);                          // crida al mètode sumar
```

## Capçalera

El format general de la capçalera d'un mètode és:

```
modificador tipusRetornat nomMètode( paràmetres ) throws excepcions{  
    // codi del mètode  
}
```

- modificador és el tipus d'accés al mètode
- tipusRetornat és el tipus de dada retornada pel mètode, o `void` si no retorna res
- nomMètode és el nom que s'utilitzarà per a cridar al mètode





- paràmetres és la informació externa que necessita el mètode per a fer la seua tasca
- excepcions són les diferents excepcions que pot llançar el mètode

→ Els parèntesis són obligatoris, és el que diferencia un mètode d'una referència.

Una excepció és un fet que altera l'execució d'un programa. Un mètode pot llançar zero o moltes excepcions, els noms de les excepcions s'escriuen després del `throws` separades per coma.

☞ Tractarem les excepcions en el tema dedicat als objectes.

## Modificador d'accés

Els modificadors d'accés són:

- `private` defineix un mètode privat, que només pot ser accedit per la classe on està escrit (és el més restrictiu).
- `no escriure res` defineix un mètode de paquet, que pot ser accedit per la classe on està escrit i les classes definides en el mateix paquet.
- `protected` defineix un mètode protegit, que pot ser accedit per la classe on està escrit, les classes del mateix paquet i les classes que hereten d'ella encara que estiguen en paquets diferents.
- `public` defineix un mètode públic, que pot ser accedit per qualsevol classe o instància sense importar el paquet (és el més permissiu).

Els exercicis que estem fent sols usen una classe (la principal), per tant, tots els mètodes que creem són privats.

☞ Veurem els modificadors d'accés en el tema dedicat als objectes.

## Bloc de sentències

El bloc de sentències del mètode està tancat en les `{ }` posteriors a la capçalera, aquest bloc funciona de la mateixa forma que qualsevol bloc de sentències.

Les variables que es defineixen en el bloc del mètode són locals al bloc, per tant són privades.

## Cridar

Cridar un mètode suposa la seua execució i es realitza amb l'operador punt, es fa amb una referència a un objecte o amb el nom d'una classe si el mètode és estàtic.

Si s'escriu únicament el nom del mètode, llavors s'està cridant un mètode escrit en la mateixa classe. Els exercicis que estem fent sols usen una classe (la principal), per tant, no cal posar el nom de la classe per a cridar al mètode.



```
Scanner teclat = new Scanner(System.in);
int valor = teclat.nextInt();           // crida al mètode nextInt amb la referència teclat
valor = sumar(8, valor);                 // crida al mètode sumar d'aquest objecte (on està escrit)
System.out.println(valor);              // crida al mètode println des de System.out que és estàtic
```

## Arguments i paràmetres

En la crida al mètode, entre parèntesis, estan els **arguments** que es copiaran en els **paràmetres** de la capçalera del mètode.

```
...                               // codi anterior
valor = sumar(8, edat);           // crida al mètode sumar, 8 i edat són els arguments
...                               // codi posterior

private int sumar(int num1, int num2) {
    ...
}                                // en la capçalera de sumar, num1 i num2 són els paràmetres
                                // num1 adquireix el valor 8 i num2 el valor de edat
                                // codi del mètode
```

La crida al mètode, provoca un salt de l'execució a l'inici del bloc de codi del mètode, després continua l'execució del codi del mètode fins a acabar-lo. Després, l'execució torna al punt posterior a la crida al mètode. Les fletxes roges indiquen la seqüència d'execució, les altres fletxes indiquen la còpia dels arguments de la crida en els paràmetres de la capçalera del mètode.

## Signatura

La **signatura** d'un mètode està formada pel seu **nom** i els **tipus dels seus paràmetres**. Dos mètodes són iguals si tenen la mateixa signatura.

Si diversos mètodes tenen el mateix nom i paràmetres diferents, la seua signatura és diferent, per tant, són mètodes diferents, es diu que estan **sobrecarregats**.

```
private int sumar(int num1, int num2) {           // la signatura del mètode és sumar(int, int)
    return num1 + num2;
}

private int sumar(int num1, int num2, int num3) { // la signatura del mètode és sumar(int, int, int)
    return num1 + num2 + num3;
}

private int sumar(double num1, int num2) {        // la signatura del mètode és sumar(double, int)
    return num1 + num2;
}

private int sumar(String num1, String num2) {     // la signatura del mètode és sumar(String, String)
    return Integer.parseInt(num1) + Integer.parseInt(num2);
}
```

A l'hora de cridar un mètode, els arguments han de coincidir en nombre i en tipus, la signatura del mètode determina quin mètode s'executa.

```
int resultat = sumar(3, 5);           // crida al mètode amb la signatura sumar(int, int)
int resultat = sumar(3.0, 5);         // crida al mètode amb la signatura sumar(double, int)
int resultat = sumar("301", "517");   // crida al mètode amb la signatura sumar(String, String)
```



## Retornar un valor

Si un mètode retorna un valor, llavors s'ha d'indicar el tipus de dada del valor retornat en la capçalera (és una primitiva o una classe).

La instrucció per a retornar un valor és `return`.

→ El `return` sols retorna un valor (un objecte).

→ El valor retornat substitueix a la crida al mètode.

→ El `return` finalitza l'execució del mètode (trenca el bloc de sentències).

El mètode següent indica si `num` és parell o no, retorna un booleà.

```
private boolean esParell(int num) { // el mètode retorna un valor de tipus booleà
    return num % 2 == 0;           // el return retorna un valor booleà
}
```

El `if` crida al mètode `esParell`, aquest s'executa i retorna un booleà.

```
int valor = 32;
if (esParell(valor)) { // la crida a esParell(valor) se substitueix per un true
    System.out.println(valor + " és parell");
}

if (true) { // després de la crida al mètode esParell(valor) tinc el valor true
    System.out.println(valor + " és parell");
}
```

Un mètode pot tindre més d'un `return`, però sols s'executa un, ja que finalitza l'execució del mètode.

```
private boolean esParell (int num) {
    if (num % 2 == 0) {
        return true; // retorna true si és parell
    }
    return false; // retorna false si no és parell (imparell)
}
```

Si un mètode no retorna cap valor, el seu tipus és `void`, però pot tindre `return`, normalment l'execució d'un mètode finalitza en la clau de tancament del bloc de sentències del mètode.

```
private void escriuSou(double sou) {
    if (sou < 0) { // si sou és negatiu
        return; // torna al mètode que el va cridar
    }
    System.out.printf("Sou = %8.2f", sou);
} // torna al mètode que el va cridar
```

Si es crea un objecte en un mètode, cal retornar-lo, si no es perd, després cal assignar-lo a una referència per a poder manejar-lo.



```

public void principal() {
    String nom="Eduardo";
    String saluda = saludaA(nom);           // la cadena de text retornada s'assigna a saluda
    System.out.println(saluda);
}
private String saludaA (String nom) {      // retorna un objecte de tipus cadena de text
    String res = "";                       // crea la cadena de text res per a guardar el resultat
    for (int i = 0; i < nom.length(); i++){ // repeteix el bucle, el nombre de caràcters de nom
        res += "hola ";                   // concatena "hola " a res
    }
    res += nom;                            // concatena nom a res
    return res;                            // retorna l'objecte creat, la cadena de text res
}                                           // la referència res es destrueix i es perd l'objecte associat

```

En l'àmbit del mètode `saludaA` es crea `res` un objecte de tipus `String` i es modifica, el `return` retorna la referència `res`, per tant s'assigna el valor de `res` a `saluda`, així la cadena té una referència en l'àmbit del mètode principal i no es perd.

## Nom

El nom del mètode és l'element usat per a cridar el mètode.

→ Posa un nom relacionat amb l'acció que realitza el mètode.

→ Elegeix bé el nom del mètode, és fonamental per a tindre un codi clar.

☠☠☠☠ un nom mal elegit baixa la nota ☠☠☠☠

→ No uses el mateix nom per a un mètode i una variable, dona lloc a confusions.

→ Posa un verb a l'inici del nom d'un mètode (representa una acció).

→ El nom d'un mètode comença amb minúscula i usa la notació CamelCase.

El nom d'un mètode que retorna un valor booleà comença per `es` (en espanyol, en valencià) o `is` (en anglés): `esMajorDEdad`, `esParell`, `isWorking`, `isDone`.

## Paràmetres

Els paràmetres són la informació que necessita el mètode per fer la seua tasca.

→ Els paràmetres permeten passar informació del bloc on està la cridada al bloc del mètode.

Els paràmetres s'escriuen entre els parèntesis de la capçalera i separats amb comes. Els parèntesis s'escriuen sempre, encara que el mètode no tinga paràmetres, és el que distingeix el nom d'un mètode del nom d'una referència.

→ Un paràmetre és la declaració d'una referència local al mètode amb un valor inicial.

En la capçalera del mètode estan els paràmetres (tipus i referència), en la crida del mètode els arguments (el valor).



Els paràmetres reben una còpia dels arguments que estan definits en la crida al mètode. Aquest tipus de pas d'informació s'anomena **pas per còpia**, en Java sols existeix aquest tipus de pas d'informació.

```
int n1 = -2;
int n2 = 4;
int suma = sumar(n1, n2);
System.out.println("suma = " + suma);

private int sumar(int num1, int num2) { // crea les referències num1 i num2 de tipus int amb els valors n1 i n2
    return num1 + num2;
}                                     // destrueix les referències num1 i num2
```

En la crida a `sumar`, `n1` i `n2` són els arguments, quan s'executa del mètode `sumar` es crea la referència `num1` de tipus `int` amb el valor de `n1` (-2) i la referència `num2` de tipus `int` amb el valor de `n2` (4), `num1` i `num2` són variables locals al mètode (només són accessible en el bloc del mètode) quan finalitza el mètode es destrueixen.

La signatura del mètode defineix els tipus i l'ordre dels paràmetres que s'esperen en la crida, si el tipus de l'argument és diferent del tipus esperat pel paràmetre, llavors dona un error. Si hi ha algun tipus repetit, llavors, cal anar amb compte a l'hora de realitzar la crida i passar els arguments en l'ordre correcte.

→ Els paràmetres s'associen per ordre d'escriptura dels arguments en la crida.

```
private int restar(int num1, int num2) {
    return num1 - num2;
}
```

Aquest mètode necessita dos enters, i l'ordre dels arguments canvia el resultat.

```
int valor = 3;
valor = restar(12, valor); // assigna 9 a valor
valor = restar(valor, 12); // assigna -9 a valor
```

## Primitives o objectes immutables

Un objecte immutable no pot canviar després de la seua creació. Les primitives i les seues classes embolcall són immutables.

Quan es passa una primitiva o un objecte immutable, llavors el paràmetre té una còpia de l'argument. El mètode usa la còpia de l'objecte i l'original no es veu afectat.

```
int n1 = -2;
int n2 = 4;
System.out.println("suma = " + sumar(n1, n2));
System.out.println("n1 = " + n1 + " n2 = " + n2); // visualitza n1 = -2 n2 = 4

private int sumar(int num1, int num2) { // crea una còpia de les primitives i treballa amb les còpies
    num1 = num1 + num2;
    return num1;
}
```



Els arguments `n1` i `n2` no es veuen afectat pel canvi realitzat en el mètode, ja que s'ha efectuat sobre `num1`, que és un objecte diferent de `n1`.

☞ Veurem el tractament dels objectes mutables en el tema dedicat als objectes.

## Recursivitat

Es parla de **recursivitat** quan un mètode es crida a si mateix, és una mena de bucle.

→ Tots els mètodes recursius han de tindre una condició d'eixida (o de parada) que finalitza la recursivitat.

La condició d'eixida no crida al mètode, si no es compleix aquesta condició d'eixida, llavors es crea un bucle infinit i es produeix un desbordament de la pila. També, es pot produir un desbordament de la pila si el nombre d'iteracions és molt gran (la pila té una grandària limitada).

Cal modificar el valor dels arguments en la crida al mètode, perquè es complisca la condició d'eixida, aquesta proporciona un valor fix que no crida al mètode.

```
private static String repiteix(String text, int i) {
    if (i < 1) { // condició d'eixida
        return ""; // finalitza el recursiu, no es crida al mètode
    }
    return i + "/" + text + repiteix(text, i - 1); // el mètode es crida a si mateix canviant el valor de l'argument
                                                    // el canvi ha de buscar que es complisca la condició d'eixida
}
```

```
System.out.println(repiteix("Ola", 3)); // crida inicial al mètode repiteix que torna un text
// visualitza
3/Ola2/Ola1/Ola
```

La taula següent conté la traça de l'execució de `repiteix("Ola", 3)`.

<code>repiteix("Ola", 3)</code>	no coneix <code>repiteix("Ola", 3)</code> , el guarda en la pila
<code>return 3 + "/" + "Ola" + repiteix("Ola", 2)</code>	no coneix <code>repiteix("Ola", 2)</code> , el guarda en la pila
<code>return 2 + "/" + "Ola" + repiteix("Ola", 1)</code>	no coneix <code>repiteix("Ola", 1)</code> , el guarda en la pila
<code>return 1 + "/" + "Ola" + repiteix("Ola", 0)</code>	coneix <code>repiteix("Ola", 0)</code>
<code>return ""</code>	retorna el valor de <code>repiteix("Ola", 0)</code>
<code>return 1 + "/" + "Ola" + ""</code>	recupera <code>repiteix("Ola", 1)</code> i retorna el seu valor
<code>return 2 + "/" + "Ola" + "1/Ola"</code>	recupera <code>repiteix("Ola", 2)</code> i retorna el seu valor
<code>return 3 + "/" + "Ola" + "2/Ola1/Ola"</code>	recupera <code>repiteix("Ola", 3)</code> i retorna el seu valor
<code>"3/Ola2/Ola1/Ola"</code>	

La crida al mètode no es pot avaluar, per tant, es guarda en la pila, es torna a cridar el mètode canviant els arguments, i així successivament, fins a arribar a la condició d'eixida, que retorna un resultat fix, amb aquest resultat es recuperen les expressions anteriors de la pila que ja es poden avaluar.

```
private static void posaAlrevés(int num) { // escriu els dígit d'un nombre enter positiu a l'inrevés
    if (num < 10) { // condició d'eixida
```



```

    System.out.println(num);           // visualitza el nombre (el dígit que queda)
    return;                           // finalitza el recursiu
}
else {
    System.out.print(num % 10);        // visualitza l'últim dígit
    posaAlrevés(num/10);              // elimina el dígit visualitzat i crida el mètode amb el nombre resultant
}
}

```

La crida a **posaAlrevés** visualitza l'últim dígit del nombre rebut com a paràmetre i crida al mètode **posaAlrevés** passant-li el nombre dividit per 10 (el canvi en l'argument), quan el nombre rebut com a paràmetre és inferior a 10 (condició d'eixida), només visualitza el dígit.

→ Si pots realitzar un mètode amb un bucle, no uses la recursivitat.

Els bucles són més ràpids i usen menys recursos que els mètodes recursius.

☞ En els exercicis, crea mètodes, no et cenyisques als sol·licitats de manera explícita en els enunciats.



## Exercicis

14.

Què visualitza el codi següent? No crees cap projecte, analitza el codi.

```

public static void main (String [ ] args) {
    int compt = 3;
    pinta(compt);
    System.out.println("compt = " + compt);
}
private static void pinta(int compt){
    System.out.print("compt: ");
    while (compt > 0) {
        System.out.print(compt + ": ");
        compt--;
    }
    System.out.println("");
}

```

15. canvia2mtdr

Què fa el codi següent? Usa'l en un programa (funciona bé per a enters majors que 0).

```

public String canvia(int num) {
    return num > 0 ? canvia(num / 2) + (num % 2) : "";
}

```

16. suma2

Escriu el mètode **suma2(int num)** que suma 2 al paràmetre que rep i retorna la suma. L'execució del codi següent ha de visualitzar **suma = 34** en la consola.

```

int num = 10;
int altreNum = 2;
int suma = suma2(num) + suma2(altreNum) + suma2(suma2(num) + suma2(altreNum));
System.out.println("suma = " + suma);

```



#### 17. diasSetmanaM

Escriu el mètode `getDiaSetmana(int nDia)` que rep el dia de la setmana i torna el nom del dia de la setmana (cadena de text). El valor 1 és el “Dilluns”, el 2 és el “Dimarts”, ... fins al 7 que és el “Diumenge”. Si no es passa un valor de l'1 al 7, llavors es torna “nombre de setmana incorrecte”.

Usa el mètode en un programa.

#### 18. qualificacioliteralM

Escriu el mètode `getNotaLiteral(double nota)` que rep una nota i torna la qualificació literal (cadena de text). Si nota val de 8.5 a 10 torna “Excel·lent”, de 7 a 8.5 torna “Notable”, de 6 a 7 torna “Bé”, de 5 a 6 torna “Suficient”, de 3.5 a 5 torna “Insuficient” i de 0 a 3.5 torna “Molt deficient”. Si la nota no està entre 0 i 10 torna “incorrecta”.

Usa el mètode en un programa.

#### 19. calculapi.

Escriu un mètode que calcula el valor de PI, rep el nombre de termes a usar en el càlcul.

La sèrie  $PI = 4 * (1 - 1/3 + 1/5 - 1/7 + 1/9 \dots)$  permet realitzar el càlcul.

Escriu un programa que llig de teclat el nombre de termes (un terme és una divisió) a usar i visualitza el valor de PI obtingut. Finalitza quan s'introdueix un 0.

#### 20. validadni

Escriu un mètode que valida un DNI, rep una cadena de text (el DNI) i torna vertader o fals. Un DNI té 8 dígits del '0' al '9' i una lletra de “TRWAGMYFPDXBNJZSQVHLCKE”, el mòdul 23 aplicat al nombre dona la posició de la lletra dins de la cadena de text anterior.

“00000023T” és correcte, ja que  $23 \% 23 = 0$  que correspon a “T”

“73721994X” és incorrecte, ja que  $73721994 \% 23 = 2$  que correspon a “W” no a “X”

Escriu un programa que valida DNIs llegits de teclat.

#### 21. mcd.

Escriu un mètode que rep dos nombres enters majors que 0, i torna el màxim comú divisor. Si algun dels nombres és negatiu, llavors es passa a positiu.

**Algoritme 1:** el divisor comença en el menor dels nombres i baixa cap a l'1, el primer divisor, el mòdul del qual és 0 per als dos números, és el màxim comú divisor.

**Algoritme 2:** obté la resta de la divisió del primer nombre pel segon nombre, al primer nombre se li assigna el segon nombre i al segon nombre se li assigna la resta, això es repeteix mentre el segon nombre (o la resta, ja que són iguals) és diferent de 0. Quan finalitza el bucle el primer nombre és el màxim comú divisor.

Escriu el mètode `lligSuperiorA(Scanner kb, String missatge, int mínim)` que mostra `missatge` i llig un nombre enter superior a `mínim`.

Escriu un programa que llig de teclat dos nombres i visualitza el el màxim comú divisor.

#### 22. factorial.

Escriu un mètode que rep un nombre enter n, i torna el seu factorial. Si n és negatiu, llavors es passa a positiu. El factorial de n s'escriu `n!` i es calcula de la forma següent:

$$0! = 1$$

$$n! = 1 * 2 * 3 * 4 * \dots * (n - 2) * (n - 1) * n$$

També es pot definir com.





$0! = 1$

$n! = n * (n - 1)!$

El factorial creix molt de pressa, elegeix bé el tipus de dada.

Escriu un programa que visualitza els factorials del 0 al 21.

### 23. trianglefloyd

Escriu un mètode que rep el nombre de files d'un triangle de Floyd i el presenta en pantalla el triangle de Floyd amb aquest nombre de files. Si el nombre de files es negatiu, llavors es passa a positiu.

Un triangle de Floyd de 5 files és el següent

```
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
```

Escriu el mètode `llog0iSuperiorA(Scanner kb, String missatge, int mínim)` que mostra `missatge` i llig el zero o un nombre enter superior a `mínim`.

Escriu un programa que llig de teclat el nombre de files d'un triangle de Floyd (mínim 2) i el visualitza. Finalitza quan s'introdueix un 0.

### 24. esprimer.

Escriu un mètode que rep un nombre enter positiu i indica si és primer o no.

Un nombre és primer si, únicament, és divisible per 1 i per si mateix. Per a reduir les comparacions, en lloc d'arribar al nombre, pots arribar al nombre dividit per 2 més 1 (tots els valors superiors donen 1 coma alguna cosa, no són divisibles)

Escriu un programa que llig de teclat un nombre i visualitza si és primer. Finalitza quan s'introdueix un 0 o un nombre negatiu.

### 25. calculasqrt

Escriu el mètode que calcula el valor de l'arrel quadrada d'un nombre real positiu. Si el nombre que rep el mètode és negatiu, llavors es passa a positiu.

El càlcul de l'arrel quadrada de num és el següent

$a_1 = \text{num} / 2$	el 1 indica que és el primer càlcul de a
$a_n = (a_{n-1} + \text{num} / a_{n-1}) / 2$	el n indica que és el càlcul enèsim de a
mentre $a_n$ és diferent de $a_{n-1}$	el $a_{n-1}$ és el valor anterior al $a_n$

El valor de a és l'arrel de num.

Escriu un programa que llig de teclat num (un nombre enter positiu) i visualitza la seua arrel quadrada. Finalitza quan s'introdueix un 0.

Variació: el mètode és recursiu.

### 26. sinusx.

Escriu un programa que calcula el sinus d'un angle llegit de teclat. El valor llegit està en graus, ha de ser un valor positiu.

El càlcul del sinus es realitza amb la sèrie

$\sin(x) = x - x^3 / 3! + x^5 / 5! - x^7 / 7! + \dots$

La variable `x` és l'angle en radians (180 graus són  $\pi$  radians). Calcula 84 termes de la sèrie, com a màxim, ja que el factorial creix molt de pressa. Comprova el resultat amb `Math.sin(radians)`.



Crea el mètode que converteix graus a radians (equivalent a `Math.toRadians(graus)`), rep un valor enter i torna un double, si el valor rebut és negatiu, llavors es passa a positiu.

Crea el mètode que eleva un nombre a una potència (equivalent a `Math.pow(nombre, potència)`), rep el nombre (double) i potència (enter) i torna un double.

Copia el mètode que calcula el factorial i modifica-ho perquè funcione amb double.

### 27. triangles.

Escriu un programa que visualitza un triangle de “\*”

```
****
***
**
*
```

Llig de teclat el nombre de “\*” de la primera línia, un valor entre 3 i 10, en l'exemple anterior s'ha llegit 4. Has d'obligar a l'usuari a introduir un valor entre 3 i 10.

Variació: llig de teclat el nombre de “\*” de la primera línia i el nombre de repeticions de la figura (un valor entre 1 i 7), entre cada figura escriu “//”, no es visualitzaran els “//” al final de cada línia.

```
*** // *** // *** // ***
**  // **  // **  // **
*   // *   // *   // *
```

En l'exemple anterior s'ha llegit 3 “\*” i 4 figures (cal visualitzar els espais en blanc).

### 28. password.

Escriu un programa que llig de teclat un password, si és vàlid, es visualitza “**el password és vàlid**” i s'acaba el programa. Si el password és invàlid, es visualitza “**el password és invàlid\نها de tindre almenys 8 caràcters, lletres minúscules i majúscules i conté dos dígitos com a mínim**” i es repeteix la lectura del password. Per a ser vàlid, un password ha de complir: té almenys 8 caràcters, conté lletres minúscules i majúscules i conté dos dígitos com a mínim. Usa mètodes de `String` i de `Character`.

### 29. sumanumreves.

Escriu un programa que llig un nombre enter positiu. Suma al nombre llegit aquest mateix nombre, però amb les xifres a l'inrevés, mostra la suma, si la suma és capicua, el programa finalitza, si no la suma passa a ser el nou nombre i es repeteix el procés.

El mòdul 10 d'un nombre ens dona la xifra de les unitats.

El nombre dividit per 10 elimina les unitats.

Un nombre es construeix sumant cada xifra per la potència de 10 corresponent, per exemple:  $123 = 1 \cdot 100 + 2 \cdot 10 + 1 \cdot 1$ ,  $78987 = 7 \cdot 10000 + 8 \cdot 1000 + 9 \cdot 100 + 8 \cdot 10 + 7 \cdot 1$

Precaució amb el 0, el 1100 invertit dona 11.

### 30. numnom

Escriu un programa que calcula el nombre associat al nom d'una persona llegit de teclat i el seu significat. Per a poder llegir correctament caràcters amb accents o la ñ, has de posar `Scanner(System.in, "ISO-8859-1")` en la sentència de creació del scanner.

El text es passa a majúscules. Sols es té en compte les lletres. Se sumen els codis de les lletres. La suma s'ha de reduir a un sol dígit, és a dir, se sumen els dígitos que formen la suma fins a obtindre un valor d'un sol dígit.



Per exemple, el nom “**Jesús García Báñez**”, les lletres sumen 1709, la suma dels dígit és 17, com té més d'un dígit es tornen a sumar, el resultat és 8 que té un únic dígit per tant es pot obtenir el seu significat de la taula següent (8 correspon a “el poder”).

valor	significat
1	el lideratge pur, l'energia l'individualisme i l'egoisme material
2	la dualitat, la parella que pot veure els dos costats de qualsevol situació
3	l'expansió
4	el constructor, de ment i consciència
5	la llibertat, la vitalitat, l'aventura, la polèmica i la controvèrsia
6	la justícia i l'ordre
7	la seguretat i la protecció
8	el poder
9	el misticisme

### 31. rombe

Realitza un programa que pinta un rombe, es llig de teclat el nombre de files fins a la fila central.

En l'exemple següent el nombre de files és 4.

```
#
###
#####
##### // la fila central és la 4
#####
###
#
```

Variació: Sense usar mètodes, usant bucles niats.

Variació: Es llig de teclat el nombre de repeticions del rombe, un valor entre 3 i 9.

### 32. Fibonacci.

Escriu un programa que visualitza els 20 primers termes de la sèrie de Fibonacci “**1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765,**”

$F_1 = 1$  el primer terme de la sèrie val 1

$F_2 = 1$  el segon terme de la sèrie val 1

$F_N = F_{N-1} + F_{N-2}$  el terme N és la suma dels dos termes anteriors ( $N \geq 3$ )

Variació: llig de teclat el nombre de termes a visualitzar, és un valor major que 2. Per a facilitar el recompte, baixa de línia cada 10 termes. Atenció, la sèrie creix molt ràpidament!

Variació: llig de teclat l'interval de valors dels termes a visualitzar, són valors major que 0 (per exemple del valor 100 al valor 5000, s'han de mostrar **144, 233, 377, 610, 987, 1597, 2584, 4181,**).





si  $a \neq 0, d = 0$  llavors tenim una arrel doble, el càlcul és  $x = -b/2a$   
 si  $a \neq 0, d < 0$  llavors tenim dues arrels imaginàries, el càlcul és

$$x = \frac{-b}{2a} \pm i \frac{\sqrt{-(b^2 - 4ac)}}{2a}$$

Visualitza l'equació en format matemàtic seguida del seu resultat.

Per a escriure l'equació en format matemàtic, s'ha de tindre en compte: si el coeficient és 0, llavors no es mostra el terme, llevat que siga el tercer (c). Si el coeficient és 1, llavors no s'escriu, llevat que siga el tercer. Si el coeficient del primer terme és positiu (pot ser a, b o c), llavors no es mostra el +. Els coeficients es mostren amb dos decimals com a màxim i si són valors enters sense decimals (si  $\text{num} \% 1 == 0$ , llavors num és enter).

L'elevat a 2, és el caràcter "<sup>2</sup>", copia el caràcter que apareix en l'enunciat al teu programa.

A continuació hi ha diferents exemples, amb els coeficients llegits i el format resultant.

coeficients			equació	#	coeficients			equació
a	b	c	resultant	#	a	b	c	resultant
3	4	4	$3x^2+4x+4=0$	#	-3	-4	0	$-3x^2-4x=0$
0	4	4	$4x+4=0$	#	0	-4	1	$-4x+1=0$
3	0	4	$3x^2+4=0$	#	1	-1	1	$x^2-x+1=0$
0	0	4	$4=0$	#	0	1	-1	$x-1=0$
2.666	3.33	1.5	$2.66x^2-3.33x+1.5=0$	#	3	-4.15	1	$3x^2-4.15x+1=0$
24.9	0	3	$24.90x^2+3=0$	#	0	-4.15	-1	$-4.15x-1=0$

