

Unitat 7.

Funcions

Autors: Carlos Cacho y Raquel Torres

Revisat per: Lionel Tarazon - lionel.tarazon@ceedcv.es

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

José Manuel Martí - josemanuel.marti@ceedcv.es

Jose Cantó Alonso – j.cantoalonso@edu.gva.es

Guillermo Garrido Portes – g.garridoportes@edu.gva.es

2023/2024

Llicència



[CC BY-NC-SA 3.0 ES](https://creativecommons.org/licenses/by-nc-sa/3.0/es/) **Reconeixement – No Comercial – Compartir igual (by- nc-sa)** No es permet un ús comercial de l'obra original ni de les possibles obres derivades, la distribució de les quals s'ha de fer amb una llicència igual a la que regula l'obra original. NOTA: Aquesta és una obra derivada de l'obra original realitzada per Carlos Cacho i Raquel Torres.

Nomenclatura

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



Important



Atenció



Interessant

Contenido

1. Introducció	4
2. Declaració d'una funció	5
3. Cridar una funció	7
4. Àmbit de les variables.....	10
5. Paràmetres: pas per valor i per referència.....	11
6. Devolució d'un valor	13
7. Recursivitat	14
7.1. La recursivitat requereix sempre:.....	14
7.2. Tipus de recursivitat:	14
7.3. Avantatges:.....	14
7.4. Desventajas:.....	14
7.5. Recursivitat vs iteració.....	14
7.5.1. Exemple: Suma dels números de l'1 al n	15
8. Agraïments.....	16

1. Introducció

La millor manera de crear i mantindre un programa gran és construir-lo a partir de peces més xicotetes o mòduls. Cadascun dels quals és més manejable que el programa íntegrament.



Les **funcions** (subprogrames) són utilitzades per a evitar **la repetició de codi** en un programa en poder executar-lo des de diversos punts d'un programa amb només invocar-lo.

El concepte de funció és una manera d'encapsular un conjunt d'instruccions dins d'una declaració específica (anomenada generalment SUBPROGRAMA o FUNCÍO), permetent la descomposició funcional i la diferenciació de tasques.

Utilitat principal de les funcions:

- Agrupar codi que forma una entitat pròpia o una idea concreta.
- Agrupar codi que es necessitarà diverses vegades en un programa, amb al missió de no repetir codi.
- Dividir el codi d'un programa gran en subprogrames (funcions), cadascun d'ells especialitzats a resoldre una part del problema.

Característiques de les funcions:

- Es defineixen mitjançant un nom únic que representa el bloc de codi.
- Poden ser anomenades (executades) des de qualsevol part del codi.
- Se'ls pot passar valors perquè els processen d'alguna forma.
- Poden retornar un resultat per a ser usat des d'on se'ls haja anomenat.

2. Declaració d'una funció

Declarar una funció simplement significa crear-la perquè després pugui ser anomenada (utilitzada) des d'un altre lloc del codi del nostre programa. Una funció s'estructura en **capçalera** i **cos**.

La **capçalera** es declara en una sola línia i es compon de:

- **Modificadors de funció:** Existeixen molts però els veurem en futures unitats. (Per ara només utilitzarem public static).
- **Tipus retornat:** El tipus de dada que retornarà la funció, com per exemple int, double, char, boolean, String, etc. Si la funció no retorna res s'indica mitjançant void.
- **Nom de la funció:** Identificador únic per a cridar a la funció.
- **Llista de paràmetres:** Indica els tipus i noms de les dades que se li passaran a la funció quan sigui anomenada. Poden ser varis o cap.

El **cos** és un bloc de codi entre claus { ... } que s'executarà quan des d'una altra part del codi utilitzem la funció.

```
[*Modificador_de_funció] Tipus_retornat Nom_de_funció (llista_de_paràmetres)
{
    ...
}
```

Exemples de funcions:

```
public static void imprimeixHolaMon() {
    System.out.println("Hola mon");
}
```

Aquest és un exemple molt senzill d'una funció anomenada 'imprimeixHolaMon', que no té paràmetres d'entrada (no hi ha res entre els parèntesis) i no retorna cap valor (indicat per void). Quan la diguem l'única cosa que farà serà escriure per pantalla el missatge "Hola mon".

```
public static void imprimeixHolaNom(String nom) {
    System.out.println("Hola " + nom);
}
```

Aquesta funció es diu 'imprimeixHolaNom', té com a paràmetre d'entrada una dada String anomenat 'nom' i no retorna res. Quan la cridem ens imprimirà per pantalla el text "Hola " seguit del String nom que li'l passarem com a paràmetre .

```
public static int doble(int a) {
    int resultat = a * 2;
    return resultat;
}
```

Aquesta funció es diu 'doble', té com a paràmetre d'entrada una dada int anomenat 'a' i retorna una dada de tipus int. Quan la cridem calcularà el doble de 'a' i el retornarà (amb el return).

```
public static int multiplica(int a, int b) {
    int resultat = a * b;
    return resultat;
}
```

Aquesta funció es diu 'multiplica', té dos paràmetres d'entrada de tipus int anomenats 'a' i 'b' i retorna una dada de tipus int. Quan la cridem calcularà a*b i ho retornarà (amb el return).

```

public static double maxim(double valor1, double valor2) {
    double max;
    if (valor1 > valor2)
        max = valor1;
    else
        max = valor2;
    return max;
}

```

Aquesta funció es diu 'maxim', té dos paràmetres d'entrada de tipus double anomenats 'valor1' i 'valor2' i retorna una dada de tipus double. Quan la cridem calcularà el màxim entre 'valor1' i 'valor2' i ho retornarà.

```

public static int sumaVector(int v[]) {
    int suma = 0;
    for (int i = 0; i < v.length; i++)
        suma += v[i];
    return suma;
}

```

Aquesta funció es diu 'sumaVector', té un paràmetre d'entrada tipus int[] (un vector de int) anomenat 'v' i retorna una dada tipus int. Quan la cridem recorrerà el vector 'v', calcularà la suma de tots els seus elements i la retornarà.

És important saber que les funcions es declaren dins de 'class' però fora del 'main'.

```

1  package unidad7;
2
3  public class programadeprueba {
4
5      public static void imprimeHolaMundo() {
6          System.out.println("Hola mundo");
7      }
8
9      public static int doble(int a) {
10         int resultado = a * 2;
11         return resultado;
12     }
13
14     public static int multiplica(int a, int b) {
15         int resultado = a * b;
16         return resultado;
17     }
18
19     public static void main(String[] args) {
20
21         // Un programa siempre empieza ejecutándose por la función main.
22         // Aquí va el código principal de nuestro programa.
23     }
24
25 }
26

```

En aquest programa tenim 4 funcions: imprimeixHolaMon, doble, multiplica i main. Sí, el 'main' on sempre has programant fins ara és en efecte una funció, però una mica especial: **'main' es la funció principal, el punt d'inici d'un programa.**

És obligatori que tot programa Java tinga una funció main. Si et fixes, és una funció que rep com a paràmetre un String[] (vector de String) i no retorna res (encara que podria retornar un int). El per què d'això ho veurem més endavant.

Les 3 funcions que hem declarat a dalt del main per si soles no fan res, simplement estan ací esperant que siguin anomenades (utilitzades), normalment des del propi main.

3. Cridar una funció

Les funcions poden ser invocades o cridades des de qualsevol altra funció, inclosa ella mateixa. Sí, una funció pot cridar a qualsevol altra funció, i una funció pot cridar-se a si mateixa.

De totes maneres ara com ara cridarem funcions només des de la funció principal 'main'. Així és més senzill d'aprendre al principi.

Quan s'invoca una funció el flux d'execució salta a la funció (passant-li els paràmetres si n'hi haguera), s'executen les instruccions de la funció i finalment torna al punt que va cridar a la funció per a continuar executant-se.

Les funcions s'invoquen amb el seu nom, passant la llista de paràmetres entre parèntesis. Si no té paràmetres han de posar-se els parèntesis igualment. Si la funció retorna un valor, per a recollir-lo cal assignar-lo a una variable o utilitzar-lo d'alguna manera (poden combinar-se funcions en expressions i fins i tot passar-ho a altres funcions).

Exemple utilitzant les funcions de l'apartat anterior:

```
public static void main(String[] args) {
    // No té paràmetres ni retorna valor. Simplemente imprimeix "Hola Mon"
    imprimeixHolaMon();

    // És habitual cridar a una funció i guardar el valor retornat en una
    variable
    int a = doble(10); // a = 20 (10*2)
    int b = multiplica(3, 5); // b = 15 (3*5)

    // Poden passar-se variables com a paràmetres
    int c = doble(a); // c = 40 (20*2)
    int d = multiplica(a, b); // d = 300 (20*15)

    // Poden combinar-se funcions i expressions
    int e = doble(4) + multiplica(2,10); // e = 8 + 20
    System.out.println("El doble de 35 és " + doble(35) ); // "El doble de
35 és 70"
    System.out.println("12 per 12 és " + multiplica(12,12) ); // "12 per
12 és 144"
}
```

Exemple: Programa amb una funció que suma dos números.

```

4      public class Suma {
5
6      public static void main(String[] args) {
7          Scanner sc = new Scanner(System.in);
8          int num1, num2, suma;
9
10         System.out.print("Introduce un número: ");
11         num1 = sc.nextInt();
12
13         System.out.print("Introduce otro número: ");
14         num2 = sc.nextInt();
15
16         suma = suma(num1, num2);
17
18         System.out.println("La suma es: " + suma);
19     }
20
21     public static int suma(int n1, int n2) {
22         int suma;
23
24         suma = n1 + n2;
25
26         return suma;
27     }
28 }
29
30
31

```

Eixida:

```

run:
Introduce un número: 3
Introduce otro número: 4
La suma es: 7
BUILD SUCCESSFUL (total time: 3 seconds)

```

Exemple: Programa amb una funció que determina si un número és parell o imparell.


```

14 public class ParImpar {
15
16     public static void main(String[] args) {
17         Scanner in = new Scanner(System.in);
18         int num;
19
20         System.out.print("Introduce un número: ");
21         num = in.nextInt();
22
23         if(par(num) == true) // Llamada a la función desde la expresión
24             System.out.println(num + " es par.");
25         else
26             System.out.println(num + " es impar.");
27     }
28
29     public static boolean par(int numero)
30     {
31         boolean par = false;
32
33         if(numero % 2 == 0) // Si el resto es 0 par será 'true' sino 'false'
34             par = true;
35
36         return par;
37     }
38 }

```

Eixida:

```

run:
Introduce un número: 9
9 es impar.
BUILD SUCCESSFUL (total time: 2 seconds)

```

4. Àmbit de les variables

Una funció només pot utilitzar les variables d'àmbit local, és a dir, les seues pròpies variables (els paràmetres de la capçalera i les variables creades dins de la funció). Quan una funció s'executa es creen les seues variables, s'utilitzen i quan la funció acaba es destrueixen les variables.

Per tot això **una funció no pot utilitzar variables que estiguen fora d'ella**, i fora d'una funció no és possible utilitzar variables de la pròpia funció. A aquesta característica se'n diu **encapsulació** i permet que les funcions siguin independents entre si, facilitant el disseny de programes grans i complexos.



Tècnicament sí que és possible que una funció utilitze variables que estan fora d'ella, però això ho veurem en futures unitats quan aprenguem Programació Orientada a Objectes.

5. Paràmetres: pas per valor i per referència

Existeixen dos tipus de paràmetres i és important comprendre la diferència.

- **Paràmetres de tipus simple (pas per valor)**: Com int, double, boolean, char, etc. En aquest cas **es passen per valor**. És a dir, **el valor es copia al paràmetre** i per tant si es modifica dins de la funció això no afectarà el valor fora d'ella perquè **són variables diferents**.

```
public static void main(String[] args) {
    int a = 10;
    System.out.println("Valor inicial de a: " + a); // a val 10
    imprimeix_doble(a); // Se li passa el 10 a la funció
    System.out.println("Valor final de a: " + a); // a continua valent 10
}

// El paràmetre 'a' és independent de la 'a' del main. Són variables diferents!
public static void imprimeix_doble(int a) {
    // Es copia el valor 10 a aquesta nova 'a'
    a = 2 * a; // Es duplica el valor de la 'a' d'aquesta funció, no afecta fora
    System.out.println("Valor d'en la funció: " + a); // 'a' val 20
}
```

Eixida

```
run:
Valor inicial de a: 10
Valor de a en la funció: 20
Valor final de a: 10
```

- **Paràmetres de tipus objecte (pas per referències)** : Com a objectes de tipus String, els Arrays, etc. En aquest cas no es copia l'objecte sinó que se li passa a la funció una referència a l'objecte original (un punter). Per això des de la funció s'accedeix directament a l'objecte que es troba fora. Els canvis que fem dins de la funció afectaran l'objecte.

```
// Summa x a tots els elements del vector v
public static void summa_x_a_el_vector(int v[], int x) {
    for (int i = 0; i < v.length; i++)
        v[i] = v[i] + x;
}

public static void main(String[] args) {
    int v[] = {0, 1, 2, 3};
    System.out.println("Vector abans: " + Arrays.toString(v));
    summa_x_a_el_vector(v, 10);
    System.out.println("Vector després: " + Arrays.toString(v));
}
```

Eixida:

```
run:
Vector antes: [0, 1, 2, 3]
Vector después: [10, 11, 12, 13]
```

IMPORTANT: Com un paràmetre de tipus objecte és una referència a l'objecte String o Array que està fora d'ella, si se li assigna un altre objecte es perd la referència i ja no es pot accedir a l'objecte fora de la funció. Encara que Java permet fer-ho, no s'aconsella fer-ho.

```
// Assignem a x un nou vector, per la qual cosa x deixarà d'apuntar al vector original.
// El vector original no canvia! Simplement ja no podem accedir a ell des de x
// perquè s'ha perdut la referència a aquest objecte.
public static void funcion1(int x[]) {
    x = new int[10]; // x apuntarà a un nou vector, l'original queda intacte
    // el que fem ací amb x no afectarà el vector original
}

// El mateix succeeix en aquest exemple, perdem la referència al String original
public static void funcion2(String x) {
    x = "Hola"; // x apuntarà a un nou String, l'original queda intacte
    // el que fem ací amb x no afectarà el String original
}
```

No s'aconsella fer este tipus de coses.

6. Devolució d'un valor



Els mètodes poden retornar valors de tipus bàsic o primitiu (int, double, boolean, etc.) i també de tipus objecte (Strings, arrays, etc.).

En tots els casos és el comando ***return*** el que realitza aquesta labor. En el cas de arrays i objectes, retorna una referència a aqueix array o objecte.

7. Recursivitat

La recursivitat és la capacitat d'una funció o mètode per a cridar-se a si mateixa. Això permet dividir un problema gran en problemes més xicotets similars i resoldre'ls directament.

7.1. La recursivitat requereix sempre:

- **Cas base:** Cas més simple que es pot resoldre directament i permet parar les recursions.
- **Casos recursius:** Cridades a si mateixa per a reduir el problema en una instància més menuda i simple.

7.2. Tipus de recursivitat:

- **Recursivitat directa:** Una funció es crida directament a si mateixa.
- **Recursivitat indirecta:** Una funció crida a una altra que al seu torn crida a la primera. Es produeix un cicle cridaes recursives entre ambdues.
- **Recursivitat lineal:** Cada cridada recursiva es fa sobre un subproblema diferent. Per exemple, en recórrer un arbre binari.
- **Recursivitat en cua:** La cridada recursiva és l'última instrucció de la funció. Això permet optimitzar l'ús de la pila.

7.3. Avantatges:

- Simplicitat conceptual i llegibilitat en alguns casos.
- Permet resoldre problemes que es divideixen fàcilment en subproblemes.

7.4. Desventajas:

- Major ús de memòria per acumulació de cridaes.
- Risc de desbordament de pila (StackOverflowError).

7.5. Recursivitat vs iteració

La recursivitat i la iteració (usant bucles com while o for) són dues formes diferents de resoldre el mateix problema.

7.5.1. Exemple: Suma dels números de l'1 al n

Solució iterativa utilitzant un bucle for:

```
public static int suma(int n) {  
    int suma = 0;  
    for (int i = 1; i <= n; i++) {  
        suma += i;  
    }  
    return suma;  
}
```

Solución recursiva:

```
public static int suma(int n) {  
    if (n == 1) {  
        return 1;  
    } else {  
        return n + suma(n-1);  
    }  
}
```

Anàlisi de les diferències:

- La versió iterativa utilitza un bucle i una variable acumuladora (suma) per a anar guardant el resultat parcial.
- La versió recursiva es basa en la descomposició del problema en problemes més xicotets, fins a arribar a un cas base senzill de resoldre (quan n és 1).
- La recursivitat pot ser més elegant i fàcil d'entendre, però consumeix més recursos ja que es van acumulant anomenades en la pila.
- La iteració sol ser més eficient quant a ús de memòria i velocitat d'execució.

8. Agraïments

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

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