

TEMA6. FUNCIONES

Programación
CFGS DAW

Profesores:

Carlos Cacho

Raquel Torres

carlos.cacho@ceedcv.es

raquel.torres@ceedcv.es


Licencia





Reconocimiento - NoComercial - CompartirIgual (by-nc-sa): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

 Importante

 Atención

 Interesante

Revisiones

- En la página 5 donde decía “Por ahora el modificador de función será siempre **static void ...**” debía decir “Por ahora el modificador de función será siempre **public static ...**”.

ÍNDICE DE CONTENIDO

1.Introducción.....	4
2.Estructura.....	4
3.Llamada a una función.....	5
4.Los parámetros.....	8
5.Devolución de un valor.....	9
6.El término void.....	9
7.Recursividad.....	10
8.Agradecimientos.....	12

UD06. FUNCIONES

1. INTRODUCCIÓN

La mejor forma de crear y mantener un programa grande, es construirlo a partir de piezas más pequeñas o módulos. Cada uno de los cuales es más manejable que el programa en su totalidad.

⚡ Las **funciones** (subprogramas) son utilizadas para **evitar la repetición de código** en un programa al poder ejecutarlo desde varios puntos de un programa con sólo invocarlo.

El concepto de función es una forma de encapsular un conjunto de instrucciones dentro de una declaración específica (llamada generalmente SUBPROGRAMA), permitiendo la descomposición funcional y la diferenciación de tareas.

Su utilidad principal es:

- Agrupar código que forma una entidad propia o una idea concreta.
- Agrupar código que se necesitará varias veces en un programa, con la misión de no repetir código.
- Se definen dándoles un nombre que agrupa una parte de código.
- Se les puede pasar valores para que los operen o procesen de alguna forma.
- Pueden devolver un resultado para ser usado en algún otro sitio.

La declaración de una función está formada por una cabecera y un cuerpo.

2. ESTRUCTURA

⚡ La codificación de una función consiste en una **cabecera** para su identificación y de un **cuerpo** que contiene las sentencias que ésta ejecuta.

La **cabecera** se compone de un nombre (identificador del método), el tipo del resultado (tipos primitivos o clases) y una lista de parámetros, que puede contener cero o más variables.

```
[Modif_de_función] Tipo_devuelto Nombre_de_función (lista_de_parámetros)
{
    ...
}
```

Por ahora el modificador de función será siempre **public static**, más adelante explicaremos los diferentes tipos que existen.

La lista de parámetros consiste en cero o más parámetros formales (variables), cada uno de ellos con un tipo. En caso que el método tenga más de un parámetro, estos deben ir separados con una coma.

Por ejemplo:

```
public static void miFuncion(int v1, int v2, float v3, String v4, ClaseObjeto v5);
```

Nótese que aunque existan varios parámetros pertenecientes al mismo tipo o clase, no pueden declararse abreviadamente, como ocurre con las variables locales y los atributos, indicando el tipo y a continuación la lista de parámetros separados por comas. Así, es ilegal la siguiente declaración del método anterior:

```
public static void miFuncion(int v1, v2, float v3, String v4, ClaseObjeto v5);
```

La declaración de un parámetro puede ir antecedita, como ocurre con las variables locales, por la palabra reservada final. En ese caso, el valor de dicho parámetro no podrá ser modificado en el cuerpo del método.

3. LLAMADA A UNA FUNCIÓN

Las funciones pueden ser invocadas o llamadas desde cualquier función de la clase, incluido ella misma.

Además, cuando se invoca, se pueden pasar un valor a cada parámetro a través de una variable o un valor constante.

⚡ La acción de pasar valores a parámetros de tipo primitivo (int, double, boolean, char..) se denomina **paso de parámetros por valor**.

En éste caso, los argumentos que se pasan, no pueden ser modificados por la función.

⚡ En caso que el parámetro sea un Objeto, lo que se está haciendo es un **paso de**

parámetros por referencia, y en este caso, los parámetros sí pueden ser modificados por la función.

El flujo de ejecución realiza un salto hasta la función y vuelve a la siguiente instrucción cuando el método haya terminado.

Se puede ver un ejemplo del flujo en el recurso de “Flujo de la función” en los recursos de esta unidad.

Las funciones se invocan con su nombre, y pasando la lista de argumentos entre paréntesis.

Por ejemplo:

```
int x;  
x = sumaEnteros(2,3);
```

Aunque la función no reciba ningún argumento, los paréntesis en la llamada son obligatorios. Por ejemplo para llamar a la función *hacerAccion*, simplemente se pondría:

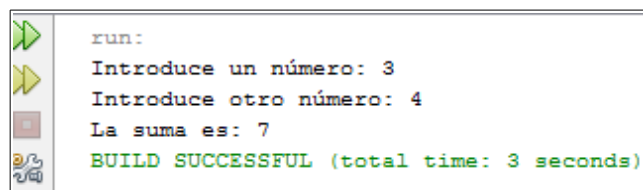
```
hacerAccion();
```

Se puede ver que como la función tampoco devuelve ningún valor, por tanto no se asigna a ninguna variable. (No hay nada que asignar).

Ejemplo1: función que suma dos números

```
14 public class Suma {
15
16     public static void main(String[] args) throws IOException {
17         BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
18         int num1, num2, suma;
19
20         System.out.print("Introduce un número: ");
21         num1 = Integer.parseInt(in.readLine());
22
23         System.out.print("Introduce otro número: ");
24         num2 = Integer.parseInt(in.readLine());
25
26         suma = suma(num1, num2);
27
28         System.out.println("La suma es: " + suma);
29     }
30     public static int suma(int n1, int n2)
31     {
32         int suma;
33
34         suma = n1 + n2;
35
36         return suma;
37     }
38
39 }
```

Salida:

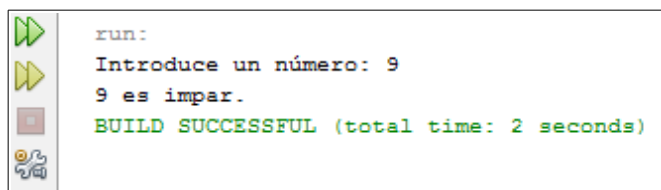


```
run:
Introduce un número: 3
Introduce otro número: 4
La suma es: 7
BUILD SUCCESSFUL (total time: 3 seconds)
```

Ejemplo2: función que determina si un número es par

```
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 }
```

Salida:



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

4. LOS PARÁMETROS

Los parámetros de un método pueden ser de dos tipos:

- Variables de tipo simple de datos: En este caso, el paso de parámetros se realiza siempre por valor. Es decir, el valor del parámetro de llamada no puede ser modificado en el cuerpo de la función. (La función trabaja con una copia del valor utilizado en la llamada).

- Variables de tipo objeto (referencias): En este caso, lo que realmente se pasa a la función es un puntero al objeto y, por lo tanto, el valor del parámetro de llamada sí que puede ser modificado dentro de la función, la función trabaja directamente con el valor utilizado en la llamada, a no ser que se anteponga la palabra reservada *final*.

5. DEVOLUCIÓN DE UN VALOR

⚡ Los métodos pueden devolver valores básicos (int, short, double, etc.), Strings, arrays (los veremos en la siguiente unidad) e incluso objetos.

En todos los casos es el comando *return* el que realiza esta labor. En el caso de arrays y objetos, devuelve una referencia a ese array u objeto.

6. EL TÉRMINO VOID

El hecho de que una función devuelva o no un valor es opcional. En caso de que devuelva un valor se declara el tipo que devuelve. Pero si no necesita ningún valor, se declara como tipo del valor devuelto, la palabra reservada *void*. Por ejemplo:

```
void hacerAlgo() {  
    ...  
}
```

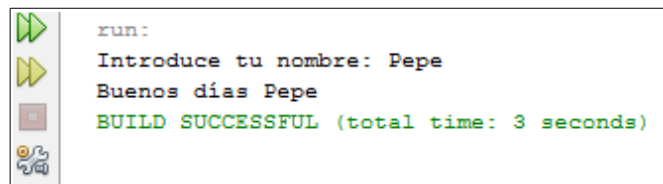
Cuando no se devuelve ningún valor, la cláusula *return* no es necesaria.

Se puede ver que en el ejemplo la función *hacerAlgo* tampoco recibe ningún parámetro. No obstante los paréntesis, son obligatorios.

Ejemplo: función que muestra un saludo

```
14 public class Saludo {  
15  
16     public static void main(String[] args) {  
17         Scanner in = new Scanner(System.in);  
18         String nombre;  
19  
20         System.out.print("Introduce tu nombre: ");  
21         nombre = in.nextLine();  
22  
23         saludo(nombre);  
24     }  
25  
26     public static void saludo(String nombre)  
27     {  
28         System.out.println("Buenos días " + nombre);  
29     }  
30 }
```

Salida:



```
run:  
Introduce tu nombre: Pepe  
Buenos días Pepe  
BUILD SUCCESSFUL (total time: 3 seconds)
```

7. RECURSIVIDAD

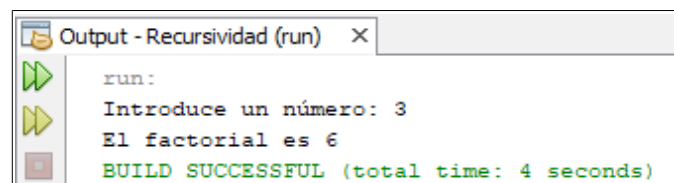
La recursividad es un método para resolver algoritmos alternativo a la iteración (bucles). En algunos casos es preferible utilizar la recursividad frente a la iteración como por ejemplo en algoritmos de ordenación de vectores como *quicksort* (los veremos en la siguiente unidad).

Una función se denomina recursiva cuando en su código se realiza una llamada a sí misma. Este tipo de funciones tienen un caso base, que será la condición de parada y salida de la función, y un caso recursivo donde se llamará de nuevo a la función.

Ejemplo: función recursiva para obtener el factorial de un número donde $N! = N * (N - 1)!$

```
14 public class Recursividad {
15
16     public static void main(String[] args) {
17         Scanner in = new Scanner(System.in);
18         int num;
19         long fact;
20
21         System.out.print("Introduce un número: ");
22         num = in.nextInt();
23
24         fact = factorial(num);
25
26         System.out.println("El factorial es " + fact);
27     }
28
29     public static long factorial(int num) {
30         long fact;
31
32         if(num > 1) // Caso recursivo
33         {
34             fact = factorial(num - 1);
35
36             fact = fact * num;
37         }
38         else // Caso base
39             fact = 1;
40
41         return fact;
42     }
43 }
```

Salida:



```
run:
Introduce un número: 3
El factorial es 6
BUILD SUCCESSFUL (total time: 4 seconds)
```

La traza de la función si el número introducido por el usuario es 4 es la siguiente:

- 1.1. Se realiza la primera llamada de la función con num=4 (línea 24).
 - 2.1. Como num es mayor que 1, se realiza la segunda llamada a la propia función pero en este caso con num=3 (línea 34).
 - 3.1. Como num es mayor que 1, se realiza la tercera llamada a la propia función pero en este caso con num=2 (línea 34).
 - 4.1. Como num es mayor que 1, se realiza la cuarta llamada a la propia función pero en este caso con num=1 (línea 34).
 - 5. Como num es igual 1 entra en el caso base y devuelve a la cuarta llamada fact=1.
 - 4.2. Ahora fact=1, se ejecuta la línea 36 y se devuelve a la tercera llamada fact=2 (fact=1*2).
 - 3.2. Ahora fact=4, se ejecuta la línea 36 y se devuelve a la segunda llamada fact=6 (fact=2*3).
 - 2.2. Ahora fact=16, se ejecuta la línea 36 y se devuelve a la primera llamada fact=24 (fact=6*4).
- 2.1. Muestra por pantalla el valor del factorial (línea 26).

8. AGRADECIMIENTOS

Apuntes actualizados y adaptados al CEEDCV a partir de la siguiente documentación:

- [1] Apuntes Programación de José Antonio Díaz-Alejo. IES Camp de Morvedre.