



TEMA 06

**PROGRAMACIÓN
CFGS DAM**

2023/2024

Versión: 231118.2333

FUNCIONES EN JAVA

ÍNDICE DE CONTENIDO

1. Introducción.....	3
2. Declaración de una función.....	4
3. Llamada a una función.....	7
4. Ámbito de las variables.....	10
5. Parámetros: Paso por valor y por referencia.....	11
6. Devolución de un valor.....	13
7. Paquetes.....	13
8. Licencia.....	16
8.1 Agradecimientos.....	16

TEMA 6 – FUNCIONES EN JAVA

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.

Separar una tarea grande en otras mas pequeñas es fundamental para desarrollar buenos programas puesto que facilitan la depuración, la reutilización del código y el mantenimiento de este.

⚡ 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 o FUNCIÓN), permitiendo la descomposición funcional y la diferenciación de tareas.

Utilidad principal de las funciones:

- 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.
- Dividir el código de un programa grande en subprogramas (funciones), cada uno de ellos especializados en resolver una parte del problema.

Características de las funciones:

- Se definen mediante un nombre único que representa el bloque de código.
- Pueden ser llamadas (ejecutadas) desde cualquier parte del código.
- Se les puede pasar valores para que los procesen de alguna forma.
- Pueden devolver un resultado para ser usado desde donde se les llamó.

2. DECLARACIÓN DE UNA FUNCIÓN

Declarar una función simplemente significa crearla para que luego pueda ser llamada (utilizada) desde otro lugar del código de nuestro programa. Una función se estructura en **cabecera** y **cuerpo**.

La **cabecera** se declara en una sola línea y se compone de:

- **Modificadores de función:** Existen muchos pero los veremos en futuras unidades. Por ahora solo utilizaremos public static).
- **Tipo devuelto:** El tipo de dato que devolverá la función, como por ejemplo int, double, char, boolean, String, etc. Si la función no devuelve nada se indica mediante void.
- **Nombre de la función:** Identificador único para llamar a la función.
- **Lista de parámetros:** Indica los tipos y nombres de los datos que se le pasarán a la función cuando sea llamada. Pueden ser varios o ninguno.

Ejemplo:

```
public static int abs(int x)
{
    if (x < 0) return -x;
    else return x;
}
```

El **cuerpo** es un bloque de código entre llaves { ... } que se ejecutará cuando desde otra parte del código utilicemos la función.

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

Ejemplos de funciones:

```
public static void imprimeHolaMundo() {
    System.out.println("Hola mundo");
}
```

Este es un ejemplo muy sencillo de una función llamada 'imprimeHolaMundo', que no tiene parámetros de entrada (no hay nada entre los paréntesis) y no devuelve ningún valor (indicado por void). Cuando la llamemos lo único que hará será escribir por pantalla el mensaje "Hola mundo".

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

Esta función se llama 'imprimeHolaNombre', tiene como parámetro de entrada un dato String llamado 'nombre' y no devuelve nada. Cuando la llamemos imprimirá por pantalla el texto "Hola " seguido del String nombre que se le pase como parámetro.

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

Esta función se llama 'doble', tiene como parámetro de entrada un dato int llamado 'a' y devuelve un dato de tipo int. Cuando la llamemos calculará el doble de 'a' y lo devolverá (con el return).

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

Esta función se llama 'multiplica', tiene dos parámetros de entrada de tipo int llamados 'a' y 'b' y devuelve un dato de tipo int. Cuando la llamemos calculará a*b y lo devolverá (con el return).

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

Esta función se llama 'maximo', tiene dos parámetros de entrada de tipo double llamados 'valor1' y 'valor2' y devuelve un dato de tipo double. Cuando la llamemos calculará el máximo entre 'valor1' y 'valor2' y lo devolverá.

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

Esta función se llama 'sumaVector', tiene un parámetro de entrada tipo int[] (un vector de int) llamado 'v' y devuelve un dato tipo int. Cuando la llamemos recorrerá el vector 'v', calculará la suma de todos sus elementos y la devolverá.

Es importante saber que **las funciones se declaran dentro de 'class' pero fuera 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  
27
```

En este programa tenemos 4 funciones: imprimeHolaMundo, doble, multiplica y main. Sí, el 'main' donde siempre has programado hasta ahora es en efecto una función, pero un poco especial: **'main' es la función principal, el punto de inicio de un programa.**

Es obligatorio que todo programa Java tenga una función main. Si te fijas, es una función que recibe como parámetro un String[] (vector de String) y no devuelve nada (aunque podría devolver

un int). El por qué de esto lo veremos más adelante.

Las 3 funciones que hemos declarado arriba del main por sí solas no hacen nada, simplemente están ahí esperando a que sean llamadas (utilizadas), normalmente desde el propio main.

Vamos a verlo.

3. LLAMADA A UNA FUNCIÓN

Las funciones pueden ser invocadas o llamadas desde cualquier otra función, incluida ella misma. Sí, una función puede llamar a cualquier otra función, y una función puede llamarse a sí misma.

De todos modos **por ahora llamaremos funciones solo desde la función principal 'main'**. Así es más sencillo de aprender al principio.

Cuando se invoca una función el flujo de ejecución salta a la función (pasándole los parámetros si los hubiera), se ejecutan las instrucciones de la función y por último vuelve al punto que llamó a la función para seguir ejecutándose.

Las funciones se invocan con su nombre, pasando la lista de parámetros entre paréntesis. Si no tiene parámetros han de ponerse los paréntesis igualmente. Si la función devuelve un valor, para recogerlo hay que asignarlo a una variable o utilizarlo de algún modo (pueden combinarse funciones en expresiones e incluso pasarlo a otras funciones).

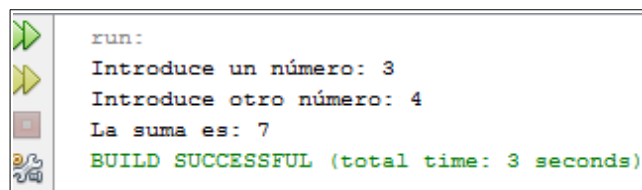
Ejemplo utilizando las funciones del apartado anterior:

```
public static void main(String[] args) {  
    // No tiene parámetros ni devuelve valor. Simplemente imprime "Hola Mundo"  
    imprimeHolaMundo();  
  
    // Es habitual llamar a una función  
    // y guardar el valor devuelto en una variable  
    int a = doble(10); // a = 20    (10*2)  
    int b = multiplica(3, 5); // b = 15    (3*5)  
  
    // Pueden pasarse variables como parámetros  
    int c = doble(a); // c = 40    (20*2)  
    int d = multiplica(a, b); // d = 300    (20*15)  
  
    // Pueden combinarse funciones y expresiones  
    int e = doble(4) + multiplica(2,10); // e = 8 + 20  
    // "El doble de 35 es 70"  
    System.out.println("El doble de 35 es " + doble(35) );  
    // "12 por 12 es 144"  
    System.out.println("12 por 12 es " + multiplica(12,12) );  
}
```

Ejemplo: Programa con una función 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  
22         public static int suma(int n1, int n2) {  
23             int suma;  
24  
25             suma = n1 + n2;  
26  
27             return suma;  
28         }  
29     }  
30  
31 }
```

Salida:

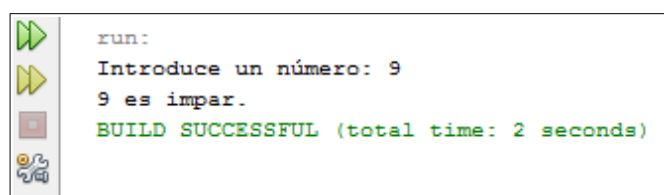


```
run:  
Introduce un número: 3  
Introduce otro número: 4  
La suma es: 7  
BUILD SUCCESSFUL (total time: 3 seconds)
```


Ejemplo: Programa con una función que determina si un número es par o impar.

```
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. ÁMBITO DE LAS VARIABLES

El ámbito de una variable es la parte del programa que puede referirse a esa variable por su nombre.

Por regla general en Java el ámbito de las variables declaradas en un bloque de sentencias se limita a las sentencias de ese bloque. En particular, el alcance de una variable declarada en un método estático se limita al cuerpo de ese método.

Por lo tanto, no se puede hacer referencia a una variable en un método estático que esté declarada en otro.

Si el método incluye bloques más pequeños -por ejemplo, el cuerpo de una sentencia if o una sentencia for, el ámbito de cualquier variable declarada en uno de esos bloques se limita a las sentencias dentro de ese bloque.

Es práctica común utilizar los mismos nombres de variables en bloques de código independientes. Cuando lo hacemos, estamos declarando variables independientes.

Por ejemplo, hemos seguido esta práctica cuando utilizamos un índice i en dos bucles for diferentes en el mismo programa.

El principio rector a la hora de diseñar software es que cada variable debe ser declarada para que su alcance sea lo más pequeño posible.

Una de las razones importantes por las que usamos métodos estáticos es que facilitan la depuración al limitar el alcance de las variables.

Una función solo puede utilizar las variables de ámbito local, es decir, sus propias variables (los parámetros de la cabecera y las variables creadas dentro de la función). Cuando una función se ejecuta se crean sus variables, se utilizan y cuando la función termina se destruyen las variables.

Por todo ello **una función no puede utilizar variables que estén fuera de ella**, y fuera de una función no es posible utilizar variables de la propia función. A esta característica se le llama **encapsulación** y permite que las funciones sean independientes entre sí, facilitando el diseño de programas grandes y complejos.

⚡ Técnicamente sí es posible que una función utilice variables que están fuera de ella, pero eso lo veremos en futuras unidades cuando aprendamos Programación Orientada a Objetos.

5. PARÁMETROS: PASO POR VALOR Y POR REFERENCIA

Es importante entender los mecanismos de paso de argumentos y retorno de valores.

Existen dos tipos de parámetros y es importante comprender la diferencia:

- **Parámetros de tipo simple (paso por valor):** Como int, double, boolean, char, etc. En este caso **se pasan por valor**. Es decir, **el valor se copia al parámetro** y por lo tanto si se modifica dentro de la función esto no afectará al valor fuera de ella porque **son variables distintas**.

```
public static void main(String[] args) {  
    int a = 10;  
    System.out.println("Valor inicial de a: " + a); // a vale 10  
    imprime_doble(a); // Se le pasa el 10 a la función  
    System.out.println("Valor final de a: " + a); // a sigue valiendo 10  
}  
  
// El parámetro 'a' es independiente de la 'a' del main.  
// ¡Son variables distintas!  
public static void imprime_doble(int a) { // copia el valor 10 a la nueva 'a'  
    a = 2 * a; // Se duplica el valor de la 'a' de esta función, no afecta fuera  
    System.out.println("Valor de a en la función: " + a); // 'a' vale 20  
}
```

SALIDA:

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

- **Parámetros de tipo objeto (paso por referencias)**: Como objetos de tipo String, los Arrays, etc. En este caso no se copia el objeto sino que **se le pasa a la función una referencia al objeto original (un puntero)**. Por ello **desde la función se accede directamente al objeto** que se encuentra fuera. Los cambios que hagamos dentro de la función afectarán al objeto.

```
// Suma x a todos los elementos del vector v
public static void suma_x_al_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 antes: " + Arrays.toString(v));
    suma_x_al_vector(v, 10);
    System.out.println("Vector después: " + Arrays.toString(v));
}
```

SALIDA:

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

IMPORTANTE: Como un parámetro de tipo objeto es una referencia al objeto String o Array que está fuera de ella, si se le asigna otro objeto se pierde la referencia y ya no se puede acceder al objeto fuera de la función. Aunque Java permite hacerlo, no se aconseja hacerlo.

```
// Asignamos a x un nuevo vector, por lo que x dejará de apuntar al vector
original.
// ¡El vector original no cambia! Simplemente ya no podemos acceder a él desde
x
// porque se ha perdido la referencia a dicho objeto.
public static void funcion1(int x[]) {
    x = new int[10]; // x apuntará a un nuevo vector, el original queda intacto
    // lo que hagamos aquí con x no afectará al vector original
}

// Lo mismo sucede en este ejemplo, perdemos la referencia al String original
public static void funcion2(String x) {
    x = "Hola"; // x apuntará a un nuevo String, el original queda intacto
    // lo que hagamos aquí con x no afectará al String original
}
```

NO SE ACONSEJA HACER ESTE TIPO DE COSAS.

6. DEVOLUCIÓN DE UN VALOR

⚡ Los métodos pueden devolver valores de tipo básico o primitivo (int, double, boolean, etc.) y también de tipo objeto (Strings, arrays, etc.).

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.

7. PAQUETES

Si una función va a ser usada en programas diferentes se puede copiar y pegar su código en cada uno de los programas, pero es más práctico agrupar para crear un paquete (package) y después importarlo desde el programa que necesite esas funciones.

Cada paquete se corresponde con un directorio. Por tanto, si hay un paquete con nombre **matematicas** debe haber un directorio llamado también **matemáticas** en la misma ubicación del programa que importa ese paquete (normalmente el programa principal).

Las funciones se pueden agrupar dentro de un paquete de dos maneras diferentes:

- Puede haber subpaquetes dentro de un paquete; por ejemplo, si quisiéramos dividir las funciones matemáticas en funciones relativas al cálculo de áreas y volúmenes de figuras geométricas y funciones relacionadas con cálculos estadísticos, podríamos crear dos directorios dentro de **matematicas** con nombres **geometria** y **estadistica** respectivamente. Estos subpaquetes se llamarían **matematicas.geometria** y **matematicas.estadistica**.
- Otra manera de agrupar las funciones dentro de un mismo paquete consiste en crear varios ficheros dentro de un mismo directorio. En este caso se podrían crear los ficheros **Geometria.java** y **Estadistica.java**.

Entenderemos mejor todos estos conceptos con un ejemplo completo. Vamos a crear un paquete con nombre **matematicas** que contenga dos clases: **Varias** (para funciones matemáticas de propósito general) y **Geometria**. Por tanto en el disco duro, tendremos una carpeta con nombre **matematicas** que contiene los ficheros **Varias.java** y **Geometria.java**.

El contenido de estos ficheros se muestra a continuación.

```
package matematicas;
/**
 * Funciones matemáticas de propósito general
 *
 * @author Luis José Sánchez
 */
public class Varias {
```

```
/**
 * Comprueba si un número entero positivo es primo o no.
 * Un número es primo cuando únicamente es divisible entre
 * él mismo y la unidad.
 *
 * @param x un número entero positivo
 * @return <code>>true</code> si el número es primo
 * @return <code>>false</code> en caso contrario
 */
public static boolean esPrimo(int x) {
    for (int i = 2; i < x; i++) {
        if ((x % i) == 0) {
            return false;
        }
    }
    return true;
}

/**
 * Devuelve el número de dígitos que contiene un número entero
 *
 * @param x un número entero
 * @return la cantidad de dígitos que contiene el número
 */
public static int digitos(int x) {
    if (x == 0) {
        return 1;
    } else {
        int n = 0;
        while (x > 0) {
            x = x / 10;
            n++;
        }
        return n;
    }
}
}
```

Se incluye a continuación Geometria.java. Recuerda que tanto Varias.java como Geometria.java se encuentran dentro del directorio [matematicas](#).

```
package matematicas;

/**
 * Funciones geométricas
 *
 * @author Luis José Sánchez
 */
public class Geometria {
    /**
     * Calcula el volumen de un cilindro.
     * Tanto el radio como la altura se deben proporcionar en las
     * mismas unidades para que el resultado sea congruente.
     *
     * @param r radio del cilindro
     * @param h altura del cilindro
     * @return volumen del cilindro
     */
}
```

```
public static double volumenCilindro(double r, double h) {
    return Math.PI * r * r * h;
}

/**
 * Calcula la longitud de una circunferencia a partir del radio.
 *
 * @param r radio de la circunferencia
 * @return longitud de la circunferencia
 */
public static double longitudCircunferencia(double r) {
    return 2 * Math.PI * r;
}
}
```

Observa que en ambos ficheros se especifica que las clases declaradas (y por tanto las funciones que se definen dentro) pertenecen al paquete **matematicas** mediante la línea `package matematicas`. Para probar las funciones desde un programa externo (`PruebaFunciones.java`) que está fuera de la carpeta **matematicas**, justo en un nivel superior en la estructura de directorios.

```
import matematicas.Varias;
import matematicas.Geometria;
/**
 * Prueba varias funciones
 *
 * @author Luis José Sánchez
 */
public class PruebaFunciones {
    public static void main(String[] args) {
        int n;
        // Prueba esPrimo()
        System.out.print("Introduzca un número entero positivo: ");
        n = Integer.parseInt(System.console().readLine());
        if (matematicas.Varias.esPrimo(n)) {
            System.out.println("El " + n + " es primo.");
        } else {
            System.out.println("El " + n + " no es primo.");
        }
        // Prueba digitos()
        System.out.print("Introduzca un número entero positivo: ");
        n = Integer.parseInt(System.console().readLine());
        System.out.println(n + " tiene " + matematicas.Varias.digitos(n) + "
digitos.");
        // Prueba volumenCilindro()
        double r, h;
        System.out.println("Cálculo del volumen de un cilindro");
        System.out.print("Introduzca el radio en metros: ");
        r = Double.parseDouble(System.console().readLine());
        System.out.print("Introduzca la altura en metros: ");
        h = Double.parseDouble(System.console().readLine());
        System.out.println("El volumen del cilindro es " +
matematicas.Geometria.volumenCilindro(r, h) + " m3");
    }
}
```

Las líneas:

```
import matematicas.Geometria;  
import matematicas.Varias;
```

Cargan las clases contenidas en el paquete `matematicas` y, por tanto, todas las funciones contenidas en ellas. No vamos a utilizar el comodín de la forma `import matematicas.*`; ya que está desaconsejado su uso en el estándar de codificación en Java de Google.

Observa que se escriben por orden alfabético; como el paquete `matematicas` coincide, se escribe antes `Geometria` y luego `Varias`.

El uso de una función es muy sencillo; hay que indicar el nombre del paquete, el **nombre de la clase** y finalmente el **nombre de la función** con los parámetros adecuados. Por ejemplo, como vemos en el programa anterior, `matematicas.Varias.digitos(n)` devuelve el número de dígitos de `n`; siendo `matematicas` el paquete, `Varias` la clase, `digitos` la función y `n` el parámetro que se le pasa a la función.

8. LICENCIA



[CC BY-NC-SA 3.0 ES](https://creativecommons.org/licenses/by-nc-sa/3.0/es/) Reconocimiento – No Comercial – Compartir Igual (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. Esta es una obra derivada de la obra original de Carlos Cacho y Raquel Torres.

8.1 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.