

- 1 Introducción
- 2 Declaración de una función
- 3. Llamada a una función
- 4. Ámbito de las variables
- 5. Parámetros: Paso por valor y paso por referencia
- 6. Devolución de un valor
- 7. Funciones en Dart
 - Parámetros opcionales
 - Parámetros opcionales con nombre
 - Parámetros opcionales por posición
 - Parámetros con valor por defecto



Unión Europea

Fondo Social Europeo

El FSE invierte en tu futuro

Fecha	Versión	Descripción
19/09/2021	1.0.0	Versión inicial
07/10/2024	2.0.0	Se añade Dart

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, también conocido como **método**, 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 el fin 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, conocido como **procedimiento**.
- **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.

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 class 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'.**

```

import java.util.Scanner;

public class Main {
    public static void imprimeHolaMundo() {
        System.out.println("Hola mundo");
    }
    public static void imprimeHolaNombre(String nombre) {
        System.out.println("Hola " + nombre);
    }
    public static int doble(int a) {
        int resultado = a * 2;
        return resultado;
    }
    public static int multiplica(int a, int b) {
        int resultado = a * b;
        return resultado;
    }
    public static double maximo(double valor1, double valor2) {
        double max;
        if (valor1 > valor2)
            max = valor1;
        else
            max = valor2;
        return max;
    }
    public static int sumaVector(int v[]) {
        int suma = 0;
        for (int i = 0; i < v.length; i++)
            suma += v[i];
        return suma;
    }
    public static void main(String[] args) {
        // En el programa principal, main, invocamos las funciones declaradas fuera
        // de este.
        // Recordad: Dentro de un fichero SOLO PUEDE HABER UNA CLASE PÚBLICA

        double val1;
        double val2;

        // Se define el objeto Scanner que nos va a permitir recoger los datos de
        // teclado.
        Scanner sc = new Scanner(System.in);

        System.out.println("Introduce el valor del primer valor real");

        // Recogemos con el objeto creado sc el primer valor real y lo devolvemos
        // a la variable val1
        val1 = sc.nextDouble();

        System.out.println("Introduce el valor del segundo valor real");

        // Recogemos con el objeto creado sc el segundo valor real y lo
        // devolvemos a la variable val2
        val2 = sc.nextDouble();
    }
}

```

```

        // Invocamos a la función dentro de la función de sistema printf, la cual
        nos permite formatear con //2 decimales (%.2f), y así nos ahorramos una
        variable puesto que máximo devuelve un valor de //tipo double que
        mostramos con el printf.

        System.out.printf("El valor mayor es %.2f ",maximo(val1,val2));
        System.out.println("");

    }
}

```

En este programa tenemos 7 funciones: `imprimeHolaMundo`, `imprimeHolaNombre`, `doble`, `multiplica`, `máximo` y `main`. Sí, el 'main' donde siempre has programando 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 6 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`, como podéis observar dentro de este con **la función `maximo` que se invoca dentro de la llamada a la función de sistema `printf`.**

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 algunas de 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
System.out.println("El doble de 35 es " + doble(35) ); // "El doble de
35 es 70"
System.out.println("12 por 12 es " + multiplica(12,12) ); // "12 por 12
es 144"

}

```

Ejemplo: Programa con una función que suma dos números.

```

import java.util.*;
public class Main {

    public static void main(String[] args) {
        int n1, n2;

        Scanner sc = new Scanner(System.in);

        System.out.println("Introduce el primer número entero a sumar");
        n1 = sc.nextInt();

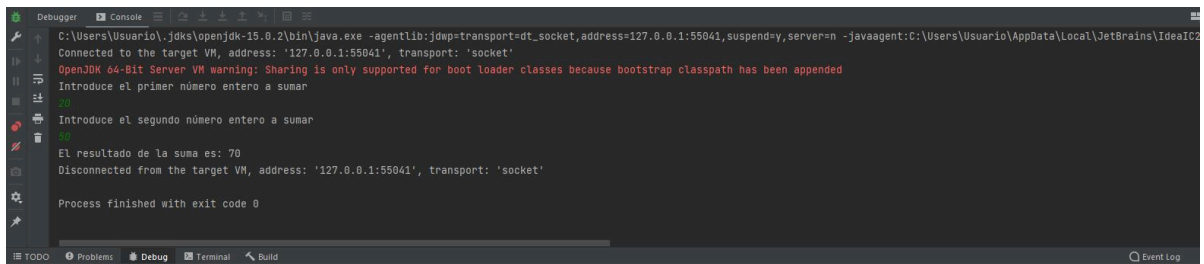
        System.out.println("Introduce el segundo número entero a sumar");
        n2 = sc.nextInt();

        System.out.println("El resultado de la suma es: " + sumar(n1, n2));

    }
    public static int sumar (int n1, int n2){
        return (n1+n2);
    }
}

```

Salida:



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

```
import java.util.*;
public class Main {

    public static void main(String[] args) {
        int n1;

        Scanner sc = new Scanner(System.in);

        System.out.println("Introduce un número entero");
        n1 = sc.nextInt();

        // Podemos utilizar un par de sentencias condicionales y en
        // función del valor devolver que es par si es true y sino impar

        if (par(n1))
            System.out.println("El número " + n1 + " es par");
        else
            System.out.println("El número " + n1 + " es impar");

        System.out.println("Ahora invocamos a la función y mostramos resultados
de una forma diferente");

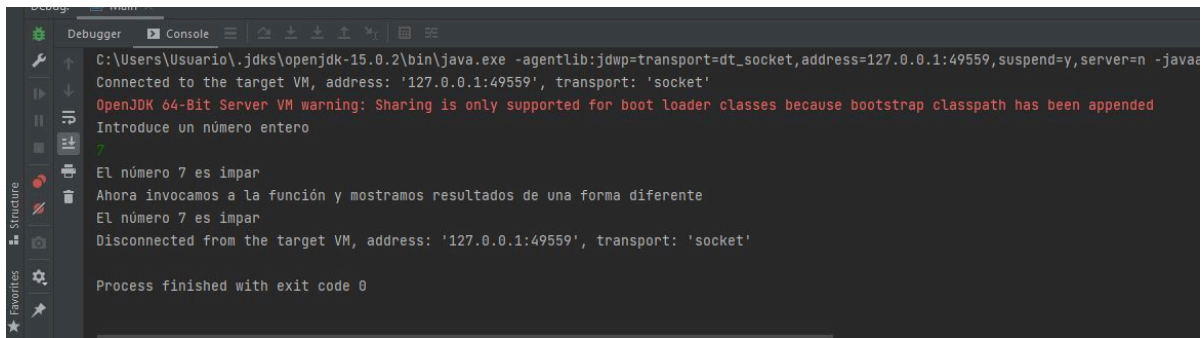
        // También podemos simplificarlo así y ahorrar código
        // Cuando invocamos a la función par(n1) sustituimos lo que va después
del interrogante ?
        // por el valor que le precede, " es par" y si es falso, por lo que
precede despues de
        // los dos puntos : , " es impar"
        System.out.println("El número " + n1 + (par(n1)?" es par":" es impar"));

    }
    public static boolean par (int n1){
        if ((n1 % 2) == 0)
            return true;

        else
            return false;

    }
}
```

Salida:



4. Ámbito 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.

Veamos un fragmento de código y el resultado de la ejecución:

```
import java.util.*;
public class Main {

    public static void main(String[] args) {
        int n1;
        // Inicializamos la variable contador a 0
        int contador=0;

        Scanner sc = new Scanner(System.in);

        System.out.println("Introduce un número entero con el que incrementamos
el contador");
        n1 = sc.nextInt();
        // Invocamos la función que se encarga de incrementar este contador con
el valor que le proporcionamos
        incrementa_contador(n1, contador);
        System.out.println("El valor de la variable global contador es: " +
contador);

    }
    public static void incrementa_contador (int n1, int contador){
        contador = contador + n1;
    }
}
```

Resultado de la ejecución para un n1 igual a 20:


```
Debug: Main x
Debugger Console
C:\Users\Usuario\.jdk\openjdk-15.0.2\bin\java.exe -agentlib:jwp=transport=dt_socket,address=127.0.0.1:63977,suspend=y,server=n -javaagent:C
Connected to the target VM, address: '127.0.0.1:63977', transport: 'socket'
OpenJDK 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended
Introduce un número entero con el que incrementamos el contador
20
El valor de la variable global contador es: 0
Disconnected from the target VM, address: '127.0.0.1:63977', transport: 'socket'
|
Process finished with exit code 0
```

⚡ 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. Parametros: Paso por valor y paso por referencia

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 class Main {

    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) { // Se copia el valor 10 a esta 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:



```
Debug: Main x
Debugger Console
C:\Users\Usuario\.jdk\openjdk-15.0.2\bin\java.exe -agentlib:jdwp=transport=dt_socket,address=127.0.0.1:51944,suspend=y,server=n -javaa
Connected to the target VM, address: '127.0.0.1:51944', transport: 'socket'
OpenJDK 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended
Valor inicial de a: 10
Valor de a en la función: 20
Valor final de a: 10
Disconnected from the target VM, address: '127.0.0.1:51944', transport: 'socket'

Process finished with exit code 0
```

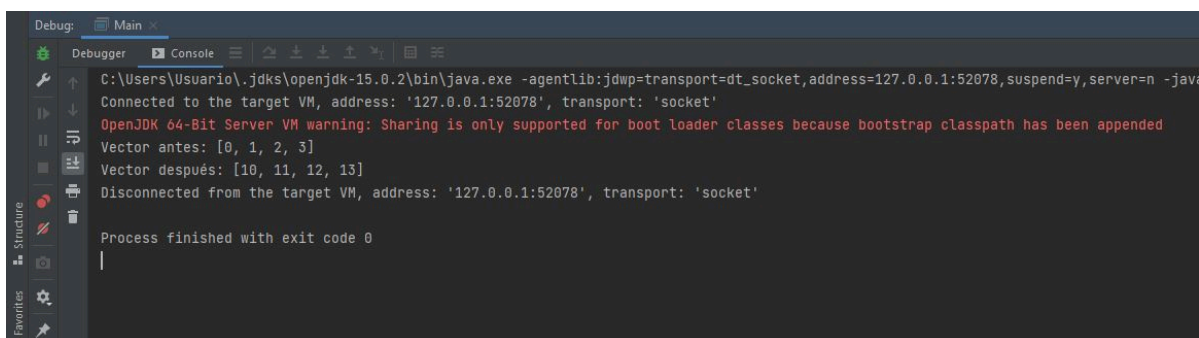
- **Parámetros de tipo objeto (paso por referencias)** : Como objetos de tipo String, los Arrays, etc. En este caso no se copia el objeto sino que **se le pasa a la función una referencia al objeto original (un puntero)**. Por ello **desde la función se accede directamente al objeto** que se encuentra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

```
import java.util.*;
public class Main {

    // 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:



```
Debug: Main x
Debugger Console
C:\Users\Usuario\.jdk\openjdk-15.0.2\bin\java.exe -agentlib:jdwp=transport=dt_socket,address=127.0.0.1:52078,suspend=y,server=n -javaa
Connected to the target VM, address: '127.0.0.1:52078', transport: 'socket'
OpenJDK 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended
Vector antes: [0, 1, 2, 3]
Vector después: [10, 11, 12, 13]
Disconnected from the target VM, address: '127.0.0.1:52078', transport: 'socket'

Process finished with exit code 0
```

🔊 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. Funciones en Dart

Dart es un verdadero lenguaje orientado a objetos, por lo que incluso las [funciones](#) son objetos y tienen un tipo, el tipo [Function](#). Esto significa que las funciones pueden asignarse a variables o pasarse como argumentos a otras funciones. También puede llamar a una instancia de una clase como si fuera una función.

```
bool isEven(int number) {
    return number % 2 == 0;
}

// Aunque se recomienda anotar el tipo en APIs públicas, se puede omitir
isEven(int number) {
    return number % 2 == 0;
}
```

Para las funciones que contienen una sola expresión, puede usar una sintaxis abreviada usando la notación `=>` también llamada '**arrow syntax**'. La sintaxis `=> expr` es una forma abreviada de `{ return expr; }`

```
bool isEven(int number) => number % 2 == 0;
```

Una función puede tener dos tipos de parámetros: **requeridos y opcionales**. Los parámetros requeridos se enumeran primero, seguidos de cualquier parámetro opcional. Los parámetros opcionales nombrados también se pueden marcar como `@required`.

Parámetros opcionales

Los parámetros opcionales pueden ser posicionales o nombrados, pero no ambos a la vez.

Parámetros opcionales con nombre

Al definir una función que tenga parámetros con nombre se usa la forma `{param1, param2, ...}` para especificar los parámetros nombrados:

```
// Named parameters
void enableFlags({bool bold, bool hidden}) {
  print("bold: $bold - hidden: $hidden");
}
```

Al llamar a una función, se especifican los parámetros con nombre usando `paramName: valor:`

```
enableFlags(); // => bold: null - hidden: null
enableFlags(bold: true); // => bold: true - hidden: null
enableFlags(bold: true, hidden: true); // => bold: true - hidden: true
enableFlags(hidden: true, bold: false); // bold: false - hidden: true
```

Para indicar que un parámetro es obligatorio usamos la anotación `@required` en el parámetro. Al usar la anotación el analizador de código del compilador pueden comprobar si no se suministra el argumento en la construcción y lanzar un aviso.

```
const Scrollbar({key key, @required widget child})
```

Parámetros opcionales por posición

Para indicar que uno o varios parámetros son posicionales se usan los corchetes `[]`:

```
String say(String from, String msg, [String device]) {
  print("from: $from - msg: $msg - device: $device");
}

String hello(String from, [String device, bool status]) {
  print("from: $from - device: $device - status: $status");
}

// Calling function
say('John', 'hi'); // => from: John - msg: hi - device: null
say('John', 'hi', 'phone'); // => from: John - msg: hi - device: phone

hello('John'); // => from: John - device: null - status: null
hello('John', 'phone'); // => from: John - device: phone - status: null
hello('John', 'phone', true); // => from: John - device: phone - status: true
```

Parámetros con valor por defecto

Para definir valores por defecto en parámetros con nombre y posicionales se usa `=`. Los valores por defecto deben ser constantes en tiempo de compilación. Si no se proporciona un valor por defecto, el valor predeterminado es `null`:

```
// Named parameters
void enableFlags({bool bold = false, bool hidden = false}) {
    //...
}

enableFlags(); // => bold: false - hidden: false
enableFlags(bold: true); // => bold: true - hidden: false
enableFlags(bold: true, hidden: true); // => bold: true - hidden: true
enableFlags(hidden: true, bold: false); // bold: false - hidden: true

// Positional parameters
String say(String from, String msg, [String device = 'carrier pigeon']) {
    // ...
}

say('John', 'hi'); // => from: John - msg: hi - device: carrier pigeon
say('John', 'hi', 'phone'); // => from: John - msg: hi - device: phone
```