

Unidad 3. Funciones y Procedimientos

Fecha	Versión	Descripción
19/09/2021	1.0.0	Versión inicial
22/09/2025	2.0.0	Versión revisada.

1. ¿Por qué necesitamos Funciones?

Imagina que estás construyendo un programa complejo. Al principio, todo tu código está en el método `main`. Pero pronto, `main` empieza a crecer sin control. Repites el mismo bloque de código en varios sitios y, si encuentras un error en ese bloque, tienes que corregirlo en todos ellos. ¡Se convierte en un caos!

Aquí es donde entran las **funciones** (también conocidas como **métodos**).

Una función es como una **herramienta especializada** que guardas en tu caja de herramientas. Es un bloque de código con un nombre, diseñado para hacer **una tarea muy concreta**. En lugar de reescribir el código una y otra vez, simplemente "usas la herramienta" llamándola por su nombre.

Ventajas principales de usar funciones:

- **Reutilización:** Escribe el código una vez y lo llamas tantas veces como quieras.
- **Organización:** Dividen un problema grande en problemas más pequeños y manejables. ¡Adiós al caos en `main`!
- **Legibilidad:** Un programa bien estructurado en funciones es mucho más fácil de leer y entender.
- **Mantenimiento:** Si necesitas corregir o mejorar una tarea, solo tienes que modificar su función, y el cambio se aplicará en todos los sitios donde se use.

2. Anatomía de una Función: La Etiqueta de la Herramienta

Cada función tiene una "etiqueta" que nos dice todo lo que necesitamos saber sobre ella. Esta etiqueta es su **declaración** o **cabecera**.

```
[Modificadores] TipoDevuelto nombreDeLaFuncion(Lista de Parámetros)
```

Vamos a desglosarlo con una analogía:

Parte de la Declaración	Analogía: La Etiqueta de la Herramienta	Ejemplo: <code>public static int sumar(int a, int b)</code>
Modificadores	Las "instrucciones de uso" (pública, estática...).	<code>public static</code> (Por ahora, siempre usaremos estos).
Tipo de Retorno	Lo que la herramienta te devuelve al terminar.	<code>int</code> (Esta herramienta te devolverá un número entero).
Nombre de la Función	El nombre de la herramienta. Debe ser descriptivo.	<code>sumar</code> (Claramente, esta herramienta sirve para sumar).
Parámetros	Los materiales que necesita la herramienta para trabajar.	<code>(int a, int b)</code> (Necesita dos números enteros para poder hacer su trabajo).
Cuerpo { ... }	El mecanismo interno de la herramienta. Las instrucciones que ejecuta.	<code>{ int resultado = a + b; return resultado; }</code> (Suma los materiales y devuelve el resultado).

Ejemplo de código completo:

```
/**
 * Esta es la implementación completa de la función descrita en la tabla.
 * Recibe dos enteros y devuelve su suma.
 */
public static int sumar(int a, int b) {
    int resultado = a + b;
    return resultado;
}
```

3. Funciones vs. Procedimientos: ¿Devuelves algo o solo actúas?

En nuestra caja de herramientas, tenemos dos tipos de especialistas:

3.1. Funciones (Las que devuelven un resultado)

Una **función** es una herramienta que, después de hacer su trabajo, te **devuelve un resultado**. Utiliza la palabra clave `return` para entregar ese resultado.

```
/**
 * Esta función es como una calculadora: le das dos números
 * y te devuelve su producto.
 */
public static int multiplicar(int a, int b) {
    int resultado = a * b;
    return resultado; // Devuelve el valor calculado
}

// Cómo se usa:
int producto = multiplicar(7, 5); // La variable 'producto' ahora vale 35
```

Aquí tienes otro ejemplo clásico de una función:

```

/**
 * Esta función compara dos números y devuelve el mayor de los dos.
 */
public static double encontrarMaximo(double valor1, double valor2) {
    if (valor1 > valor2) {
        return valor1;
    } else {
        return valor2;
    }
}

// Cómo se usa:
double maximo = encontrarMaximo(15.5, 9.2); // La variable 'maximo' ahora vale 15.5

```

3.2. Procedimientos (Los que solo realizan una acción)

Un **procedimiento** es una herramienta que realiza una tarea, pero **no devuelve ningún resultado**. En Java, los identificamos porque su "Tipo de Retorno" es `void` (vacío).

```

/**
 * Este procedimiento es como un altavoz: le das un mensaje
 * y simplemente lo muestra. No te devuelve nada.
 */
public static void saludar(String nombre) {
    System.out.println("¡Hola, " + nombre + "!");
    // No hay 'return' de un valor
}

// Cómo se usa:
saludar("Álex"); // Muestra "¡Hola, Álex!" en pantalla. No se puede asignar a una variable.

```

Característica	Función	Procedimiento
Propósito	Calcular y devolver un valor.	Ejecutar una serie de acciones.
Retorno	<code>int</code> , <code>double</code> , <code>String</code> , etc.	<code>void</code>
Palabra Clave	Usa <code>return valor;</code>	No suele usar <code>return</code> (o solo <code>return;</code>).
Uso Típico	<code>variable = miFuncion();</code> o dentro de una expresión.	<code>miProcedimiento();</code> en una línea sola.

4. Ámbito de las Variables: La Regla de la Habitación de Hotel

El **ámbito** o *scope* de una variable define dónde es visible y se puede utilizar. La regla de oro es muy simple:

"Cada función es su propia habitación de hotel"

Cuando entras en la habitación de hotel (llamas a una función), puedes usar todo lo que hay dentro: la cama, la mesa, la tele (las variables locales y los parámetros). Pero cuando sales de la habitación y cierras la puerta (la función termina), ya no puedes acceder a nada de lo que había dentro. Esas variables "desaparecen" para el resto del programa.

```

public static void main(String[] args) {
    int variableMain = 10; // Una variable en el "pasillo" del hotel (main)
    miFuncion();
    // System.out.println(variableDeFuncion); // ¡ERROR! No puedes ver dentro de la habitación desde el
    pasillo.
}

public static void miFuncion() { // Entras a la "habitación"
    int variableDeFuncion = 20; // Una variable dentro de la habitación
    System.out.println(variableDeFuncion); // Correcto, puedes usarla aquí dentro.
    // System.out.println(variableMain); // ¡ERROR! Desde dentro de la habitación no ves lo que hay en
    el pasillo.
}

```

Esta **encapsulación** es fundamental, ya que garantiza que las funciones sean independientes y no interfieran unas con otras por accidente.

5. Paso de Parámetros: Fotocopias vs. Direcciones

Cuando llamamos a una función y le pasamos datos, la forma en que Java los entrega depende del tipo de dato. ¡Entender esto es crucial!

5.1. Paso por Valor: "Te paso una fotocopia" (Tipos Primitivos)

Cuando pasas un tipo primitivo (`int`, `double`, `char`, `boolean` ...), Java no te da tu variable original a la función. En su lugar, le da una **copia exacta**, una fotocopia.

La función puede hacer lo que quiera con esa fotocopia (cambiarle el valor, usarla en cálculos...), pero **tu variable original permanecerá intacta**.

```

public static void main(String[] args) {
    int miNumero = 10;
    System.out.println("Antes de llamar a la función, miNumero vale: " + miNumero); // Vale 10

    intentarModificar(miNumero); // Le pasamos una FOTOCOPIA de miNumero

    System.out.println("Después de llamar a la función, miNumero vale: " + miNumero); // Sigue valiendo
10
}

public static void intentarModificar(int numeroCopia) { // Recibe la fotocopia
    numeroCopia = numeroCopia * 2; // Modificamos la COPIA a 20
    System.out.println("Dentro de la función, la copia vale: " + numeroCopia); // Vale 20
}

```

5.2. Paso por Referencia: "Te paso la dirección" (Arrays y Objetos)

Cuando pasas un array o un objeto, Java cambia de estrategia. En lugar de copiar toda la estructura (lo cual sería muy ineficiente), le pasa a la función la **dirección de memoria** donde se encuentra el objeto original.

La función recibe la dirección, va a la "casa" original y trabaja directamente sobre ella. Por lo tanto, **cualquier cambio que la función haga en el array u objeto, afectará al original**.

```

public static void main(String[] args) {
    int[] misNotas = {7, 5, 8};
    System.out.println("Notas originales: "); // [7, 5, 8]
    for (int i = 0; i < misNotas.length; i++) {
        System.out.println(misNotas[i]);
    }

    subirNotas(misNotas); // Le pasamos la DIRECCIÓN de misNotas

    System.out.println("Notas finales: "); // ¡Han cambiado! [8, 6, 9]
    for (int i = 0; i < misNotas.length; i++) {
        System.out.println(misNotas[i]);
    }
}

public static void subirNotas(int[] notas) { // Recibe la dirección del array
    System.out.println("Recibido. Subiendo un punto a cada nota... ");
    for (int i = 0; i < notas.length; i++) {
        notas[i] = notas[i] + 1; // Modificamos el array ORIGINAL
    }
}

```

5.3. Llamada de Funciones desde otras Funciones (Composición)

La función `main` es solo el punto de entrada. El verdadero poder de la modularidad se demuestra cuando las funciones se llaman entre sí. A esto se le llama **composición de funciones**: se construyen operaciones complejas a partir de otras más simples.

Una función puede **delegar** parte de su lógica a otra función más especializada, a menudo llamada **función auxiliar** (*o 'helper function'*).

Ejemplo: Calcular el precio final con IVA, delegando el cálculo del IVA a una función auxiliar.

```

public class CalculoComplejo {

    public static void main(String[] args) {
        double precioBase = 100.0;

        // 1. 'main' llama a la función de cálculo principal
        double precioFinal = calcularPrecioConIVA(precioBase, 21.0);

        System.out.println("El precio base es: " + precioBase);
        System.out.println("El precio final con IVA es: " + precioFinal);
    }

    /**
     * Función principal de cálculo.
     * Delega la tarea específica de calcular el IVA a otra función.
     */
    public static double calcularPrecioConIVA(double base, double porcentajeIVA) {
        System.out.println("Iniciando cálculo del precio total...");

        // 2. 'calcularPrecioConIVA' llama a su función auxiliar 'calcularIVA'
        double ivaCalculado = calcularIVA(base, porcentajeIVA);

        // 3. 'calcularPrecioConIVA' usa el resultado devuelto por la auxiliar
        double total = base + ivaCalculado;
    }
}

```

```

        return total;
    }

    /**
     * Función auxiliar (helper) especializada en calcular el IVA.
     * Es llamada por 'calcularPrecioConIVA'.
     */
    public static double calcularIVA(double base, double porcentaje) {
        System.out.println("...Calculando desglose de IVA...");
        return (base * porcentaje) / 100.0;
    }
}

```

Salida:

```

Iniciando cálculo del precio total...
...Calculando desglose de IVA...
El precio base es: 100.0
El precio final con IVA es: 121.0

```

6. Buenas Prácticas al Crear Funciones

Para ser un buen programador, no basta con que las funciones funcionen. ¡También deben ser elegantes!

- Principio de Responsabilidad Única:** Una función debe hacer **una sola cosa** y hacerla bien. Si tu función se llama `calcularYMostrarNota`, probablemente debería dividirse en `calcularNota()` y `mostrarNota()`.
- Nombres Descriptivos:** El nombre de una función debe ser un verbo o una acción que deje claro su propósito. `calcularPromedio` es mucho mejor que `func1` o `calcula`.
- Pocos Parámetros:** Intenta que tus funciones no necesiten demasiados "materiales". Si una función requiere más de 3 o 4 parámetros, puede ser una señal de que está haciendo demasiadas cosas.
- Comentarios Javadoc :** Documenta tus funciones. Explica brevemente qué hacen, qué significa cada parámetro y qué devuelven. Esto es de gran ayuda para ti y para otros programadores.

```

/**
 * Calcula el área de un círculo a partir de su radio.
 *
 * @param radio El radio del círculo (debe ser un valor positivo).
 * @return El área calculada del círculo.
 */
public static double calcularAreaCirculo(double radio) {
    return Math.PI * Math.pow(radio, 2);
}

```

7. Ejemplo Práctico: Mini-Gestor de Calificaciones

Vamos a unir todo lo que hemos aprendido en un programa completo. Crearemos un pequeño gestor de notas que nos permitirá introducir las calificaciones de un alumno, calcular la media, encontrar la nota más alta y mostrar un informe final.

Este ejemplo te permitirá ver en acción:

- El `main` como organizador.

- **Procedimientos (void)** para mostrar información.
- **Funciones** que devuelven valores.
- Paso de parámetros por **valor** y por **referencia**.

Enunciado del Problema

Crear un programa que gestione las 5 notas de un alumno. El programa debe:

1. Mostrar un mensaje de bienvenida.
2. Solicitar al usuario que introduzca las 5 notas y guardarlas en un array.
3. Calcular la nota media.
4. Encontrar la nota más alta.
5. Mostrar un informe completo con todas las notas, la media y la nota más alta.

Código de la Solución

```

import java.util.Scanner;

public class GestorCalificaciones {

    // ##### PRINCIPAL (main) #####
    // Actúa como el director de orquesta. No hace los cálculos,
    // solo llama a las herramientas (funciones) adecuadas en orden.
    public static void main(String[] args) {

        mostrarBienvenida(); // Llamada a un procedimiento

        Scanner teclado = new Scanner(System.in);
        double[] notas = new double[5];

        // Llamamos a una herramienta para llenar el array.
        // Le pasamos la "dirección" del array (paso por referencia).
        llenarNotas(teclado, notas);

        // Llamamos a herramientas que calculan y nos devuelven un resultado.
        double media = calcularMedia(notas);
        double notaMaxima = encontrarMaxima(notas);

        // Finalmente, llamamos a un procedimiento para mostrarlo todo.
        mostrarInforme(notas, media, notaMaxima);

    } // Fin del main

    // ##### HERRAMIENTAS (Funciones y Procedimientos) #####
}

/**
 * PROCEDIMIENTO: Muestra un mensaje de bienvenida.
 * No necesita datos (parámetros) y no devuelve nada (void).
 */
public static void mostrarBienvenida() {
    System.out.println("--- Bienvenido al Gestor de Calificaciones ---");
    System.out.println("A continuación, deberás introducir 5 notas.");
    System.out.println("-----");
}

```

```

/**
 * PROCEDIMIENTO: Pide al usuario las notas y rellena el array.
 * Recibe el Scanner y el array de notas.
 * Como el array se pasa por REFERENCIA, los cambios hechos aquí
 * afectarán al array original del main.
 */
public static void llenarNotas(Scanner teclado, double[] arrayNotas) {
    for (int i = 0; i < arrayNotas.length; i++) {
        System.out.print("Introduce la nota " + (i + 1) + ": ");
        arrayNotas[i] = teclado.nextDouble();
    }
}

/**
 * FUNCIÓN: Calcula la media de las notas de un array.
 * Recibe un array de notas y DEVUELVE un double (la media).
 */
public static double calcularMedia(double[] arrayNotas) {
    double sumaTotal = 0;
    for (int i = 0; i < arrayNotas.length; i++) {
        sumaTotal += arrayNotas[i]; // Acumulador
    }
    return sumaTotal / arrayNotas.length;
}

/**
 * FUNCIÓN: Encuentra la nota más alta en un array.
 * Recibe un array de notas y DEVUELVE un double (la nota máxima).
 */
public static double encontrarMaxima(double[] arrayNotas) {
    double max = arrayNotas[0]; // Suponemos que la primera es la más alta
    for (int i = 1; i < arrayNotas.length; i++) {
        if (arrayNotas[i] > max) {
            max = arrayNotas[i]; // Encontramos una nueva máxima
        }
    }
    return max;
}

/**
 * PROCEDIMIENTO: Muestra el informe final de calificaciones.
 * Recibe el array de notas, la media y la máxima.
 * Los parámetros 'media' y 'maxima' se pasan por VALOR (fotocopias).
 */
public static void mostrarInforme(double[] arrayNotas, double media, double maxima) {
    System.out.println("\n--- INFORME DE CALIFICACIONES ---");
    System.out.println("Notas introducidas: " );
    System.out.print("[");
    for (int i = 0; i < arrayNotas.length; i++) {
        System.out.print(arrayNotas[i]);
        if (i != arrayNotas.length - 1) {
            System.out.print(", ");
        }
    }
    System.out.println("]");
    System.out.printf("Nota Media: %.2f\n", media);
    System.out.println("Nota más Alta: " + maxima);
}

```

```
    System.out.println("-----");
}
}
```