3. Funciones en Java. Paso de parámetros

La estructura de una función en Java es muy similar a la que hemos visto, en sus aspectos generales, en el apartado anterior. Tal como hemos visto, la declaración de una función debe contener, como mínimo, su nombre y el tipo de dato que devuelve. Opcionalmente, ya que no es obligatorio que una función reciba datos, también debemos indicar cuáles son los parámetros que le pasamos, y de qué tipo son. Asimismo, de manera opcional pueden añadirse más datos sobre la función, como el tipo de acceso, algunos modificadores, tratamiento de excepciones, etcétera. De momento, vamos a centrarnos en las tres partes que hemos visto hasta ahora:

- Nombre de la función
- Parámetros y tipos
- Tipo de resultado que devuelve

3.1. Funciones en Java

Si nos ceñimos a los tres elementos indicados en el apartado anterior, la estructura de una función en Java sería la siguiente:

```
tipoDevuelto nombreFuncion([lista de parámetros formales])
{
      // Instrucciones de la función
      [return resultadoADevolver];
}
```

Veamos en detalle cada punto:

- **`tipoDevuelto`:** debemos indicar de qué tipo es el dato que devuelve la función. Puede ser cualquiera de los tipos nativos de Java, o bien algún tipo de estructura específico que hayamos creado nosotros, como un objeto de una clase. En el caso de las funciones que no devuelven nada, se usa el tipo **`void`**.
- **`nombreFuncion`:** es simplemente el nombre que le damos a la función. Para asignar un nombre a una función, debemos seguir las mismas reglas que se emplean al hacer lo propio con las variables.
- **`lista de parámetros formales`**: está entre corchetes, porque es opcional. Los paréntesis, sin embargo, son obligatorios. De esa forma, si a una función no se le pasa ningún parámetro, no pondremos nada entre paréntesis, pero sí incluiremos los paréntesis. En caso de incluir parámetros, cada uno de ellos debe llevar su tipo y un nombre. Cada parámetro se separa del siguiente por una coma.
- **`return`:** también está entre paréntesis, porque es opcional. Si una función no devuelve nada, no debemos incluir la cláusula `return`. Sin embargo, sí que debemos especificar en `tipoDevuelto` la palabra `void`, tal como hemos visto anteriormente. El tipo de dato devuelto con `return` debe coincidir con el que se haya indicado delante del nombre de la función en `tipoDevuelto`.

IMPORTANTE

Una función puede tener varias instrucciones **return**, pero solo se ejecutará la primera de ellas a la que se llegue, ya que en ese momento la función terminará y se devolverá el control al módulo que la haya llamado.

Veamos como ejemplo cómo sería en Java nuestra función que calculaba y devolvía el mayor de tres números introducidos como parámetros:

```
int mayor(int numero1, int numero2, int numero3) {
      int numeroMayor;
      if (numero1 >= numero2 && numero1 >= numero3) {
            numeroMayor = numero1;
      }
      else if(numero2 >= numero1 && numero2 >= numero3) {
            numeroMayor = numero2;
      }
      else {
            numeroMayor = numero3;
      }
      return numeroMayor;
}
```

Ahora, para llamar a la función disponemos de varias posibilidades:

```
int numeroMaximo = mayor(6, 10, 8);   // devuelve 10 y se guarda en numeroMaximo
```

En este caso, nuestros parámetros formales serían numero1, numero2 y numero3 (el nombre que reciben en la función), mientras que los parámetros actuales (los que ya tienen un valor real y concreto en la llamada a la función) serían 6, 10 y 8. Las variables definidas como parámetros formales asumen el valor de los parámetros actuales en el orden en el que ambos han sido declarados. Es decir, numero1 tomaría el valor 6, numero2 el valor 10 y numero3 el valor 8.

```
System.out.println("El mayor de 10, 100 y 80 es: " + mayor(10, 100, 80));
// la función devuelve 10 de nuevo, pero el resultado se muestra por
pantalla y no se guarda
```

Supongamos que la misma función anterior, en lugar de devolver el número mayor de los tres que le pasamos, solo tenga que mostrarlo por pantalla sin devolver nada. Quedaría entonces así:

```
void mayor(int numero1, int numero2, int numero3) {
      int numeroMayor;
      if (numero1 >= numero2 && numero1 >= numero3) {
            numeroMayor = numero1;
      }
      else if(numero2 >= numero1 && numero2 >= numero3) {
            numeroMayor = numero2;
      }
```

```
        else {
                numeroMayor = numero3;
        }
        System.out.println("El mayor de los tres números es " +
numeroMayor);
}
```

En este caso, vemos que la función es de tipo void (no devuelve nada). Por el mismo motivo, tampoco dispone de una instrucción return (aunque, de todos modos, podría usarse return sin más para indicar que la función debe acabar en algún momento).

Las funciones de Java suelen declararse dentro de la public class (pero fuera del main) con acceso public, protected o private, según nos interese (veremos más adelante la diferencia entre estas opciones al tratar la programación orientada a objetos), y el modificador static (también lo explicaremos más adelante).

3.2. Paso de parámetros

Aunque ya hemos visto, en la teoría y también en la práctica, cómo se realiza el paso de parámetros a una función, hay ciertas particularidades que conviene conocer.

Parámetros por valor y por referencia

En algunos lenguajes, existe la posibilidad de pasar los parámetros por valor o por referencia. Cuando un parámetro se pasa por valor, los cambios que pueda sufrir dentro de la función no se mantienen cuando esta acaba. Esto ocurre porque el parámetro formal recibe una copia del valor del parámetro actual, pero se trata de variables distintas que ocupan diferentes posiciones de memoria.

Sin embargo, si el parámetro se pasa por referencia, los cambios que pueda sufrir en la función sí que se mantienen fuera de ella. Esto es así porque, al pasar un parámetro por referencia, en realidad lo que pasamos no es su valor sino su posición de memoria. Así, los cambios que sufra dentro de la función se están aplicando a la misma variable original que se pasó como parámetro. En este caso el parámetro formal y el parámetro actual son la misma variable, pues ocupan la misma posición en la memoria.

En Java no existe el paso de parámetros por referencia de forma explícita. Sin embargo, sí que existe de forma implícita: por defecto, el paso por parámetros siempre es por valor en los tipos primitivos de datos, pero es por referencia en el resto de tipos (estructuras de datos, objetos…).

Por ejemplo, si ejecutamos la siguiente aplicación:

```
public class App {
  public static void cambiaValor(int numeroOriginal) {
    numeroOriginal = numeroOriginal+10;
  }
  public static void main(String[] args) {
    int numeroOriginal = 5;
    cambiaValor(numeroOriginal);
```

```
    System.out.println("El número ahora vale " + numeroOriginal);
  }
}
```

emos que disponemos de una función llamada cambiaValor a la que le pasamos un número y la función le suma 10. Así pues, podríamos pensar que, dentro del main, la variable numeroOriginal, que inicialmente vale 5, al pasar por la función cambiaValor valdrá 10. Sin embargo, al mostrar su valor, aparece que sigue valiendo 5. Eso es así porque numeroOriginal se pasó por valor: el numeroOriginal de la función no es el mismo que el del main, aunque se llamen igual. Al cambiar uno, no cambia el otro. Lo mismo pasaría con cualquier otro tipo de datos primitivo y también con un String. Veamos qué pasa, sin embargo, si le pasamos un array a la función y hacemos que le sume 1 a todos sus elementos.

```
public class App {
    public static void cambiaValor(int[] arrayOriginal) {
        for(int i = 0; i < arrayOriginal.length; i++) {
            arrayOriginal[i]++;
        }
    }
    public static void main(String[] args) {
       int[] arrayOriginal = {1, 10, 100};
       cambiaValor(arrayOriginal);
       System.out.print("Contenido del array después del cambio: ");
       for(int i = 0; i < arrayOriginal.length; i++) {
           System.out.printf("%2d ", arrayOriginal[i]);
       }
    }
}
```

Nuestra función `cambiaValor` recibe como parámetro un array y le suma 1 a todos sus elementos. Sin embargo, cuando mostramos el contenido del array después de llamar a la función, vemos que sí que ha cambiado. Eso se debe a que, en este caso, al tratarse de una estructura de datos, Java pasa el dato por referencia, de forma que cualquier cambio dentro de la función permanece aun después de que esta haya terminado.

IMPORTANTE

El nombre que reciben los parámetros en la función no tiene por qué coincidir con el que tienen los datos que se pasan desde el módulo principal. Cada parámetro declarado en la función recogerá el valor del parámetro que se haya pasado en su misma posición. Es decir, si tenemos una función como esta:

```
int suma(int num1, int num2, int num3) {
      return num1 + num2 + num3;
}
```

la llamada a la función puede ser perfectamente la siguiente:

```
int resultado = suma(a, b, c);
```

Aunque los nombres de los parámetros **formales** (en la función) no coinciden con los nombres de los parámetros **reales** (los que se pasan en la llamada), no hay problema. En la función, la variable num1 tomará el valor de a, num2 el de b, y num3 el de c.