PROGRAMACIÓN
Tema

FUNCIONES

ÍNDICE

1. INTRODUCCIÓN

Si el código de nuestro programa es amplio, es probable que tengamos que implementar un mismo algoritmo en momentos distintos dentro de un mismo programa, o incluso implementar el mismo algoritmo en distintos programas, lo cual nos puede llevar a tener distintos problemas:

- **Duplicidad de código.** Aumenta el tamaño del programa haciéndolo menos comprensible.
- **Dificultad de mantenimiento.** Cualquier modificación en una parte de nuestro algoritmo tendrá que aplicarse tantas veces en el código como veces se use esa funcionalidad en nuestro código.

En algunos casos los bucles pueden evitar la duplicación de código, pero para ello la ejecución repetida tiene que ser consecutiva en el tiempo. En el caso de que la funcionalidad algorítmica sea requerida en distintos sitios o momentos, esto ya no será posible.

Para evitar estos problemas se definen las funciones o métodos de las clases, aunque en este caso nos centremos en funciones genéricas no asociadas a clases, que son aquellas que tienen una similitud mayor a la de una función tal y cómo se define en matemáticas. En concreto, para nosotros:

- Una función es un conjunto de instrucciones agrupadas bajo un nombre que se ejecutan al ser invocada la función que las contiene.

- Toda función tiene una entrada, una salida, un proceso (instrucciones) y una persistencia en el tiempo asociada.

2. DEFINICIÓN EN JAVA

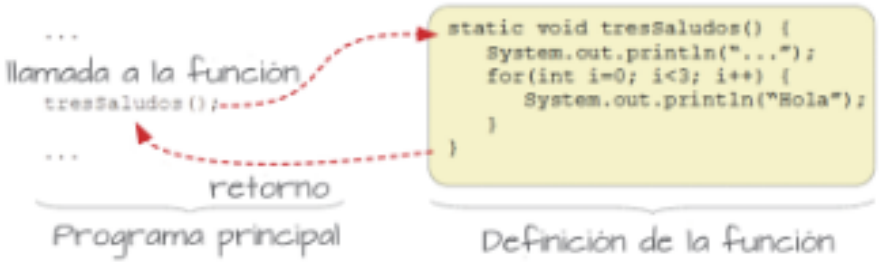
```
[acceso] [modificador] tipo nombreFuncion([tipo nombreArgumento,[tipo nombreArgumento]...])
{
    /* Instrucciones  */
    return valor;
}
```

- Acceso. Es opcional. Si no se pone se quedará por defecto. En general, las funciones suelen ser públicas.
- Modificador. Es opcional. Lo veremos en profundidad cuando avancemos en los temas de creación de clases y objetos de la POO.
- Tipo. Si no se quiere devolver nada, anulando por lo tanto la salida, se utilizará void; y si se quiere devolver un valor de salida, se utilizará el tipo del valor que devolvamos con la instrucción return.
- nombreFuncion. Es el nombre que usaremos para invocar a la función.
- Parámetros. Son opcionales. Valores que le pasamos a la función para que pueda realizar su cometido. Al igual que en matemáticas, los parámetros son los valores de entrada que utilizará la función para realizar su cometido.

- Instrucciones. Son las instrucciones del proceso algorítmico que ejecutará nuestra función. Podemos definir variables, realizar asignaciones, condiciones, bucles, invocaciones a otras funciones…

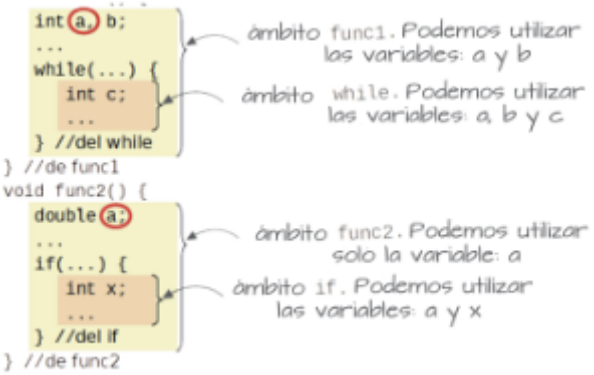
3. EJECUCIÓN

- Las instrucciones del programa principal se ejecutan hasta que encuentra la llamada a la función.
- La ejecución salta a la definición de la función.
- Se ejecuta el bloque de código de la función.
- Cuando la ejecución del código de la función termina, retornamos al punto del programa desde donde se invocó la función. Si tiene un return devolverá un valor a este punto desde donde le llamó a la función pudiendo el programador usarlo o no.
- El programa continúa su ejecución.



4. ÁMBITO DE LAS VARIABLES

Las variables definidas en una función se denominan variables locales y sólo serán visibles y utilizables dentro de la función. Sin embargo, como veremos en el desarrollo de clases, las funciones también pueden acceder otras variables.



En general, como se ve en el ejemplo, se pueden definir variables en cualquier bloque de código entre paréntesis, pero se aconseja SIEMPRE declarar las variables al principio de las clases (como atributos), al principio de las funciones (como variables locales) y en la cabecera de un bucle for (en la parte de inicialización).

Por supuesto, es una muy mala idea (de hecho nunca lo debemos de programar así) declarar una variable local con el mismo identificador usado en otro ámbito

superior, como sería un parámetro de entrada o un atributo (de clase o de objeto, que los estudiaremos con detalle en el siguiente tema)

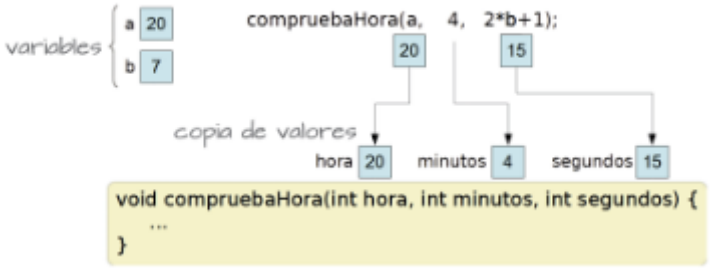
5. PARÁMETROS DE ENTRADA

Los parámetros son variables locales de la función que han sido inicializadas en la llamada a esta. Permiten comunicar datos desde el programa que invoca a la función hacía el interior del bloque de código de la función.

En java la asignación de valores a los parámetros se hace por valor, es decir, se generan unas variables locales (parámetros formales) que copiarán el valor con el qu se hizo la llamada. Si al llamar usamos variables a estas variables se les llama parámetros reales.

En java los parámetros reales se asignan uno a uno a los parámetros formales en función de su ubicación.

Cualquier cambio en un parámetro de entrada que se efectúe dentro del cuerpo de la función no repercute en la variable o expresión utilizada en la llamada, ya que lo que se modifica es una copia y no el dato original.



Cuando una función devuelve un valor, la llamada a la función se puede usar dentro de una expresión. Por ejemplo:

- int intValor = potencia(2, 3); → se asignará a intValor lo que devuelva la función potencia.
- if (intValor > media(intA, intB, intC)); → en este caso se ejecutará el if si intValor es mayor que la media que devuelva la función.

Si una función devuelve un valor, hay que cambiar el void de la definición por el tipo de valor devuelto. Dentro de la función utilizaremos la instrucción return para devolver un valor de ese tipo.

5. SOBRECARGA DE FUNCIONES

Java permite que 2 o más funciones compartan el mismo identificador, siempre que el número o el tipo de los parámetros del prototipo sean distintos. Para distinguirlas no sirve con el parámetro de vuelta.

Para invocar a cada una de las instancias de la función, se utilizarán los parámetros de la llamada.

Por ejemplo, las siguientes llamadas son a funciones distintas:

- sumar(1,2,3) → 3 valores de tipo int.
- sumar(1,2) → 2 valores de tipo int.
- sumar(1.5,2) → 2 valores, 1 de tipo double y 1 de tipo int.

6. RECURSIVIDAD

Si una función se invoca a sí misma, diremos que es recursiva directa.

Si una función invoca a otras funciones y antes de dar una respuesta a la llamada alguna de las funciones vuelve a llamar a la principal, se llamará recursividad indirecta.

En general, evitaremos la recursividad porque no hay problema resuelto recursivamente que no pueda resolverse mediante bucles. Pero en el caso de hacer alguna función que nos especifiquen que debe ser recursiva, al igual que pasa en los bucles, hay que evitar en ese caso las llamadas infinitas que se pueden producir si incluimos en la función recursiva el llamada "caso base". Un caso base es un caso especial controlado por una sentencia condicional que nos permite resolver directamente el caso más sencillo de aplicación de la función, que además tiene que ser aquel utilizado por todas las demás llamadas para resolverse aumentando gradualmente la complejidad. La función factorial es un ejemplo clásico de recursividad:

```
public static int factorial(int intNum){
    if(intNum == 0)
        return 1;
    else
        return intNum * factorial(intNum-1);
}
```