

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Tema 3 – Métodos (Funciones)

Introducción y definición

Es evidente que con lo visto hasta el momento cuando un programa se hace demasiado grande van a aparecer zonas de código repetidas y todo bastante desorganizado. Esto lo hemos visto por ejemplo en menús a la hora de pedir datos. Lo ideal sería poder realizar pequeños fragmentos de código que funcionaran como “nuevas instrucciones” y que pudiéramos ejecutarlas desde distintas partes del programa. Con esto conseguiríamos:

- No repetir código.
- Mantener el código más fácilmente ya que al cambiar/arreglar en un sitio lo cambiamos para todo el programa.
- Organizar el código incluso con la idea de repartir la realización del programa entre un equipo de programadores de forma que cada uno se ocupe de cierta parte lo más independiente posible del resto.
- Reutilizar código. Si hago un “comando nuevo” para un programa puede que me sirva para futuros programas.

Por supuesto que estamos en un lenguaje de POO y todo lo anterior lo vamos a conseguir de una forma muy eficiente con esta filosofía que veremos más adelante, pero la división de nuestro programa en los módulos denominados métodos o funciones conlleva una primera estructuración y organización modular que se ha usado como paradigma durante muchos años.

Una **función o método** es una subrutina (fragmento de código útil y específico) que puede ser invocada desde distintas partes de un programa y que puede además de realizar cierta tarea, devolver un valor de cualquier tipo de dato.

En los temas pasados ya hemos estado usando funciones que han construido otros programadores. Algunos ejemplos:

print(datos): Función dentro de *System.out* que toma el contenido de *datos* y lo imprime en pantalla. Esta función “no devuelve” nada, simplemente realiza una acción: muestra por pantalla.

sqrt(numero) : Función dentro de *Math* que devuelve la raíz cuadrada de un número. Es decir, que cuando la llamamos se ejecuta cierto código que calcula la raíz de *numero* y luego lo devuelve. Con devolver quiero decir que la propia función es sustituida por el valor que calcula al finalizar. Por ejemplo si tengo el siguiente código:

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

```
double resultado, a=36,b=6;
```

```
resultado = 2*Math.sqrt(a)+25/b;
```

```
System.out.printf("Resultado = %.3f", resultado);
```

la segunda línea comienza ejecutando la función *Math.sqrt(a)* y por tanto se ejecuta un código que calcula la raíz de 36 (ya que ese es el valor de *a*) y devuelve 6 de forma que la línea para el ordenador quedaría (después de sustituir también *b*):

```
resultado = 2*6+25/6;
```

En ambos ejemplos (print y sqrt) se llama a un código escrito previamente y se ejecuta. La diferencia es que *sqrt* devuelve un valor y por tanto puede formar parte de una expresión sin embargo *print* no devuelve nada, solo realiza acciones.

Devolver un valor significa que tras hacer los cálculos, la función deja el resultado en una zona de memoria que luego puede ser recogida desde la función llamante por ejemplo en una variable:

```
resultado = Math.sqrt(a);
```

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Métodos que no devuelven ningún valor

Es un subprograma que realiza una tarea específica pero no toma ningún valor como resultado de sus operaciones internas. La estructura básica es:

```
[public static] void nombreMetodo([lista de parámetros]) {
    cuerpo del método
}
```

Las partes entre corchetes son opcionales o dependerán de la situación. En nuestro caso, como aún no hemos profundizado en las características de orientación a objetos del lenguaje, es imprescindible que indiquemos todas las funciones con el modificador `static` y opcionalmente con `public`.

La lista de parámetros es opcional y es necesaria para darle datos a la función para poder actuar. Veamos un ejemplo:

```
public class Prueba1 {
    static void estrellas1(){
        System.out.print("*****\n");
    }

    static void estrellas2(int cantidad){
        for (int i=0; i<cantidad; i++) {
            System.out.print("*");
        }
        System.out.println();
    }

    //Método principal
    public static void main(String args[]){
        estrellas1();
        System.out.println(" Pirámide de asteriscos");
        estrellas1();
        System.out.println();
        for (int i=1; i<=10; i++) {
            estrellas2(i);
        }
        System.out.println();
        Prueba1.estrellas2(10);
        System.out.println(" Fin ");
        Prueba1.estrellas2(10);
    }
}
```

Para llamar a funciones dentro de la misma clase no es necesario poner el nombre de la clase. Si las funciones estuvieran en otra clase sí sería necesario hacerlo y además necesitaríamos anteponerle el modificador `public`.

Actividad: Prueba a crear otra clase (sin `public` delante) y mete ahí las funciones.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Paso de parámetros

El uso de parámetros es la forma que tenemos para darle información a una función para que pueda realizar tareas con datos externos a la misma. Por ejemplo si hacemos un método que sume dos números, lógicamente deberíamos de poder darle los números a sumar.

A una función se le puede pasar como parámetro un literal, una constante, una variable o una expresión.

Ejemplos:

```
public static void estrellas2(int cantidad) {
    for (int i=0; i<cantidad; i++) {
        System.out.print("*");
    }
    System.out.println();
}

public static void suma(double sumando1, double sumando2) {
    System.out.printf("El resultado de sumar %.2f+%.2f es%.2f",
        sumando1, sumando2, sumando1+sumando2);
}
```

La forma en que funcionan los parámetros es la siguiente: cuando se llama a una función se le da en la posición del parámetro (y en el mismo orden si son varios) un valor o variable. Por ejemplo:

```
suma (4, 3) ;
```

En el momento que se va a ejecutar esa línea, como el 4 está en la posición de *sumando1* este valor es sustituido en toda la función y como 3 está en la posición de *sumando 2* se sustituye en toda la función de la misma manera. Como si antes de ejecutar la función se ejecutaran las líneas:

```
sumando1=4;
sumando2=3;
```

Es decir, que al llamar a la función se ejecuta la siguiente línea:

```
System.out.printf("El resultado de sumar %.2f+%.2f es%.2f",
    4, 3, 4+3);
```

Es importante resaltar que en el caso de que se le pase una variable **esta no cambiará de valor** a lo largo de la función. Cuando profundicemos en el tema de objetos veremos que existe la posibilidad de que parezca que sí cambia.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Veámoslo:

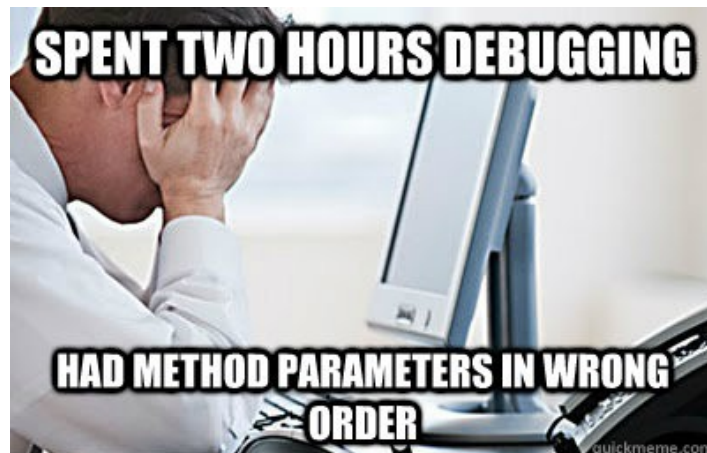
```
public static void pruebaCambio(int dato) {
    System.out.println("Dentro de la función antes del incremento: "+dato);
    dato++; // Es lo mismo que poner dato = dato + 1;
    System.out.println("Dentro de la función después del incremento: "+dato);
}
```

Para llamarlo hagamos lo siguiente:

```
int valor=10;

pruebaCambio(valor);
System.out.println("Tras acabar la función: "+valor);
```

Comprueba el resultado.



COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Métodos que devuelven valores

En los casos anteriores hemos visto funciones que realizan cierto proceso y, a lo sumo, sacan datos y resultados por pantalla, sin embargo habrá muchos casos en los que nos interese que una función tras hacer algún tipo de operación nos deje un valor de resultado para poder seguir trabajando con él. Es lo que se denomina “valor devuelto por una función”.

Para que una función devuelva un valor se necesitan dos cambios, por un lado en lugar de usar *void* en la cabecera se indica el tipo de dato que se quiere devolver. Además se usará el comando *return* para realizar la devolución. En el momento que se ejecuta *return* se termina el método aunque se esté dentro de un bucle.

Realmente cuando una función llega a *return* lo que sucede es que deja el resultado en una zona temporal de memoria y el programa principal recoge el dato de esa posición.

Veamos algunos ejemplos:

```
public static int suma(int sumando1, int sumando2) {
    return sumando1+sumando2;
}

public static int signo(int numero) {
    if (numero<0) {
        return -1;
    } else {
        if (numero>0) {
            return 1;
        }
    }
    return 0; //Puedo hacerlo sin else ¿Por qué?
}
```

¿Por qué crees que esta función suma que devuelve el resultado podría ser mejor que la anterior que además lo muestra?

En el programa principal:

```
int a,b=5,c,d;

a=suma(12,b)*10;
b=(4+a)*signo(a-300);
c=signo(suma(a,b));
d=suma(c,signo(c));
System.out.printf("a:%d\nb:%d\nc:%d\nd:%d\n",a,b,c,d);
```

En la materia de ED, en el apartado de refactorización se explicó como crear automáticamente un método a partir de una serie de líneas.

Actividades:

1. Realizar una función que sume 1+2+3+...+N y devuelva el resultado. N será un parámetro. En el programa principal ejecútala dos veces para luego mostrar la suma hasta 10 y hasta 10000.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Ámbito de variables

Hasta ahora el único sitio donde hemos declarado variables es dentro de métodos (incluido el método *main*), sin embargo las variables también se pueden declarar dentro de la clase pero fuera de todas las funciones. Esto hace que la variable pueda ser utilizada por todas las funciones o incluso como “intercambio de información” entre las funciones en lugar de los parámetros.

Tendríamos por tanto dos ámbitos de trabajo para variables:

Variables de clase: declaradas fuera de las funciones, son vistas por todas las funciones definidas en la clase. Hay que anteponerles el modificador **static** (ya entenderemos por qué).

Variables locales: declaradas dentro de las funciones y por tanto son visibles sólo dentro de la función en la que fue declarada. Es posible, por tanto, usar los mismos nombres en variables dentro de distintas funciones.

Vimos también que una **variable definida dentro de una estructura** es visible sólo dentro de la estructura (zona entre llaves).

Por ahora **vamos a usar sólo los dos últimos tipos aquí definidos** ya que el primero (*static*) tiene una visión más amplia que estudiaremos en el tema de POO.

En particular una variable definida dentro de la zona de inicialización del **for**, solo puede ser vista dentro del **for**.

Ejemplo de código incorrecto:


```
for(int i=0;i<10;i++){
    System.out.println("i dentro del bucle"+i);
}
System.out.println("i fuera del bucle"+i);
```

Esto nos permite usar la misma variable como contador en distintos bucles **for**.

También hay que tener cuidado porque una variable definida dentro de un **do-while** NO puede ser usada como parte de la condición del **while**.

Ejemplo de código incorrecto:

```
do {
    System.out.println("Introduce un n°");
    int num=sc.nextInt();
} while (num<10);
```

	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Algunas funciones de interés

Funciones de conversión: Las vistas en el segundo tema.

```
String → byte:    Byte.parseByte(str)
String → double:  Double.parseDouble(str)
String → float:   Float.parseFloat(str)
String → int:     Integer.parseInt(str)
String → long:    Long.parseLong(str)
String → short:   Short.parseShort(str)
```

Funciones matemáticas: Se encuentran en la clase Math. Algunas de ellas son:

```
Math.abs(x): Valor absoluto. Devuelve el mismo tipo que el parámetro.
Math.ceil(x): Aproxima al entero mayor más cercano. El parámetro y el
valor devuelto son double. P.ej. devuelve 5 si se le pasa 4.2.
Math.floor(x): Aproxima al entero menor más cercano. P.ej. devuelve 4 si
se le pasa 4.8.
Math.cos(x): Devuelve el coseno de x (x en radianes).
Math.sin(x): Devuelve el seno de x (x en radianes).
Math.tan(x): Devuelve la tangente de x (x en radianes).
Math.sqrt(x): Raíz cuadrada de x.
Math.pow(x,y): devuelve x elevado a y.
Math.max(x,y): Devuelve el máximo de x e y.
Math.min(x,y): Devuelve el mínimo de x e y.
```

También existen las constantes Math.PI y Math.E

Números aleatorios: Pertenece a las matemáticas pero vamos a tratarla aparte.

`Math.random()` Función que devuelve un double entre 0 y 1 (1 no incluido)

Para sacar un número aleatorio entero en cierto rango habría que hacer lo siguiente:

```
(int) (Math.random()*cantidadDeValores+valorMinimo)
```

Ejemplos:

Entre 1 y 10 (ambos incluidos): `(int) (Math.random()*10+1)`

Entre 20 y 30 (ambos incluidos): `(int) (Math.random()*11+20)`


Puedes ver otras posibilidades en:

<https://stackoverflow.com/questions/363681/how-do-i-generate-random-integers-within-a-specific-range-in-java>

También como curiosidad, si alguien quiere ver el código fuente de la clase Math puede obtenerlo aquí:

<http://developer.classpath.org/doc/java/lang/Math-source.html>

Algunas de las funciones llaman a VMMath que contiene referencias a funciones “native”, es decir, ya compiladas en binario nativo con las librerías java por temas de eficiencia.

	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Comentando las funciones

Cuando se desarrollan funciones es necesario informar de la utilidad de dichos subprogramas para lo cual se suele hacer un tipo de comentario estándar justo antes de empezar el método. En la asignatura de Entornos de Desarrollo veremos una herramienta muy potente denominada *javadoc* que permite la automatización de la creación de documentación técnica. Por el momento vamos a comentar las funciones de la siguiente manera.

- Usamos comentario de documentación: `/**`
- Descripción de lo que hace el método
- **@param** por cada parámetro y una descripción si fuera necesaria
- **@returns** indicando el valor que devuelve

Ejemplo:

```
/**
 * Resta de valores.
 *
 * @param num1 Minuendo de la resta
 * @param num3 Sustraendo de la resta
 * @return La resta de los parámetros
 */
public static int resta(int num1, int num2) {
    return num1 - num2;
}
```

La principal ventaja de hacer esto a medida que se programa es doble: Por un lado tenemos siempre a nuestra disposición o a disposición del equipo de programación una descripción de los distintos elementos que componen nuestro programa. Por otro lado, la realización del Manual Técnico de cualquier aplicación se realiza muy rápido mediante el programa *javadoc*. Para entender esto hagamos una prueba rápida:

1. Introduce la función anterior en una clase denominada Resta.
2. Guarda y en una consola ve al directorio donde tengas el archivo que contiene dicha clase.
3. Ejecuta el siguiente comando:

```
> javadoc Resta.java -d doc
```

Mediante el argumento **-d** indicamos el directorio destino de la documentación (ponemos el nombre que queramos). Si no existe lo crea.

Ahora en dicho directorio dispones de una página HTML con la documentación creada de forma estándar.

Realmente javadoc tiene muchas posibilidades y modificadores.

Si quieres ver lo anterior en el vscode no tienes más que poner el cursor del ratón encima de la función y ves cuál sería el resultado de dicha documentación.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Apéndice I: Recursividad.

Se define la recursividad como una técnica en la que una función se llama a si misma. El caso más simple de recursividad es el siguiente:

```
public static void infinito(){
    infinito();
}
```

Si lo pruebas llamándola desde un programa principal saltará un stack overflow debido a que se está llamando a si misma continuamente hasta que se llena la memoria ya que en cada llamada tiene que guardar la dirección de memoria de retorno.

Por tanto una función recursiva debe tener dos partes: una que sí es recursiva pero debe tener también una condición de salida para que termine en un momento dado.

Veamos un ejemplo con la serie de Fibonacci que es una sucesión de números definida de forma recursiva (https://es.wikipedia.org/wiki/Sucesión_de_Fibonacci):

Fib(1)=1

Fib(2)=1

Fib(3)= Fib(1)+ Fib(2)=2

Fib(4)= Fib(2)+ Fib(3)=3

Sería: 1,1,2,3,5,8,13,21,34,55, ...

La definiríamos de forma recursiva así:

1. Salida: si $n=1$ o $n=2$ (Fibonacci(n)=1)

2. Parte recursiva: Fibonacci(n)=Fibonacci($n-1$)+Fibonacci($n-2$)

```
public static int fibonacci(int n){
    if (n==1 || n==2){
        return 1;
    }
    return fibonacci(n-1)+fibonacci(n-2);
}
```

La recursividad no es siempre imprescindible pero para ciertas tareas facilita mucho la labor. Juegos como el buscaminas u operaciones como un factorial o ciertas sucesiones son un claro ejemplo.

