



U04.Métodos

Curso 2020-2021



Métodos

- Qué es un método
- Ejemplos de métodos conocidos
- Ventajas de utilizar métodos
- Definición y llamada
- Conexión entre métodos
- Definición:
 - Valor devuelto:
 - Procedimientos
 - Funciones
 - Parámetros:
 - Formales: parámetros
 - Actuales: argumentos
- Tipos de parámetros
- Llamada a un método
- Ámbito de las variables
- Métodos recursivos

Qué es un método

- Un método es un **conjunto de instrucciones referenciadas por un identificador**
- Puede ser llamado desde diferentes puntos de un programa
- Opcionalmente puede devolver un valor.
Tradicionalmente:
 - A los métodos que devuelven un valor se les denominan **funciones**
 - A los métodos que no devuelven ningún valor se les llama **procedimientos**

Ejemplo de métodos de la clase Math

static double	<code>pow(double a, double b)</code> Returns the value of the first argument raised to the power of the second argument.
static double	<code>random()</code> Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.
static double	<code>rint(double a)</code> Returns the double value that is closest in value to the argument and is equal to a mathematical integer.
static long	<code>round(double a)</code> Returns the closest long to the argument, with ties rounding up.
static int	<code>round(float a)</code> Returns the closest int to the argument, with ties rounding up.
static double	<code>max(double a, double b)</code> Returns the greater of two double values.
static float	<code>max(float a, float b)</code> Returns the greater of two float values.
static int	<code>max(int a, int b)</code> Returns the greater of two int values.
static long ③	① <code>max</code> ② <code>long a, long b</code> Returns the greater of two long values.
static double	<code>min(double a, double b)</code> Returns the smaller of two double values.
static float	<code>min(float a, float b)</code> Returns the smaller of two float values.
static int	<code>min(int a, int b)</code> Returns the smaller of two int values.
static long	<code>min(long a, long b)</code> Returns the smaller of two long values.

Cuando se da información sobre un método, se especifica:

1. su nombre
2. los datos que recibe para operar
3. y el resultado que devuelve

Ejemplos de métodos conocidos

```
import java.util.Scanner;

public class PruebaMetodos {
    public static void main(String[] args) {

        double x,y,mayor;
        Scanner teclado=new Scanner(System.in);
        System.out.print("Introduce dos números");
        x=Double.parseDouble(teclado.next());
        y=Double.parseDouble(teclado.next());
        mayor=Math.max(x,y);
        System.out.println("El mayor es:"+mayor);

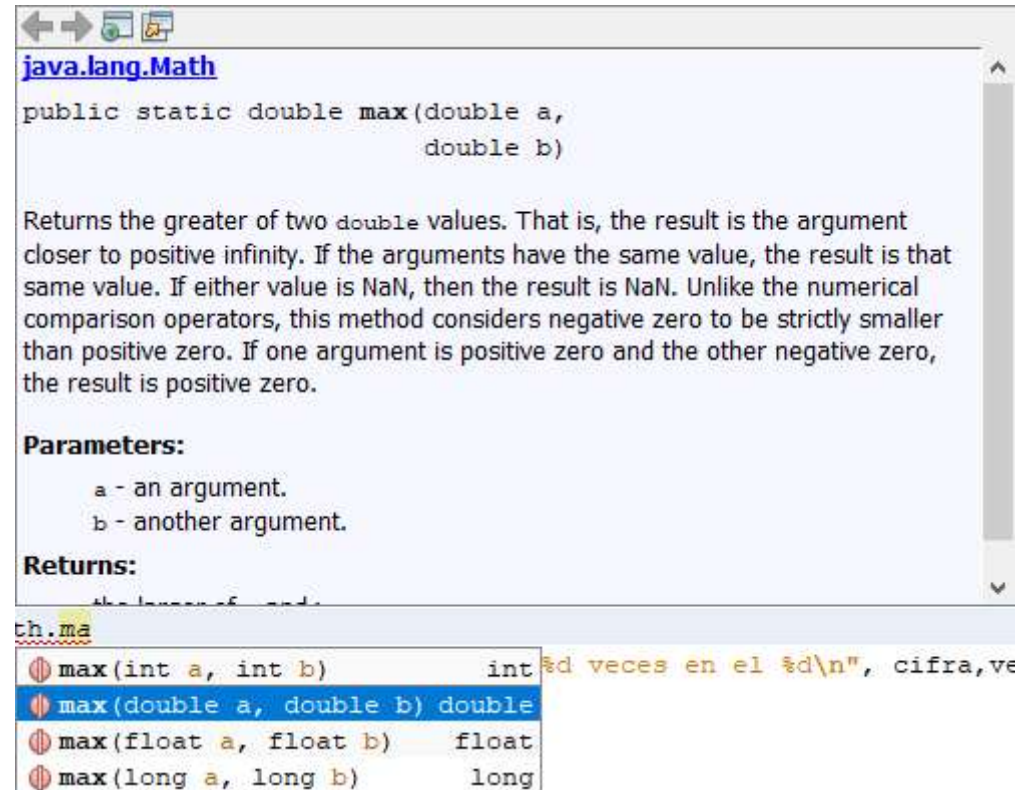
    }
```

Todos los métodos utilizados en este programa los conocemos de ejercicios anteriores excepto la llamada a `Math.max(x,y)`

Y aunque es obvio, ante la duda de cómo pueda funcionar este método en la ayuda de netbeans podemos encontrar la siguiente explicación:

```
}
```

Math.max(int a, int b))



[java.lang.Math](#)

```
public static double max(double a,
                        double b)
```

Returns the greater of two `double` values. That is, the result is the argument closer to positive infinity. If the arguments have the same value, the result is that same value. If either value is NaN, then the result is NaN. Unlike the numerical comparison operators, this method considers negative zero to be strictly smaller than positive zero. If one argument is positive zero and the other negative zero, the result is positive zero.

Parameters:

- a - an argument.
- b - another argument.

Returns:

the larger of `a` and `b`.

th.ma

- max(int a, int b) int
- max(double a, double b) double
- max(float a, float b) float
- max(long a, long b) long

- En esta explicación encontramos cómo es la cabecera de la definición de este

método: `public double max(double a, double b)`

- Y a la hora de utilizarlo sabemos que le damos dos valores de tipo double y nos devuelve el mayor que también es de tipo double

```
double x,y,mayor;
```


```
Scanner teclado=new Scanner(System.in);
```

```
System.out.print("Introduce dos números");
```


```
x=Double.parseDouble(teclado.next());
```

```
y=Double.parseDouble(teclado.next());
```

```
mayor=Math.max(x,y);
```

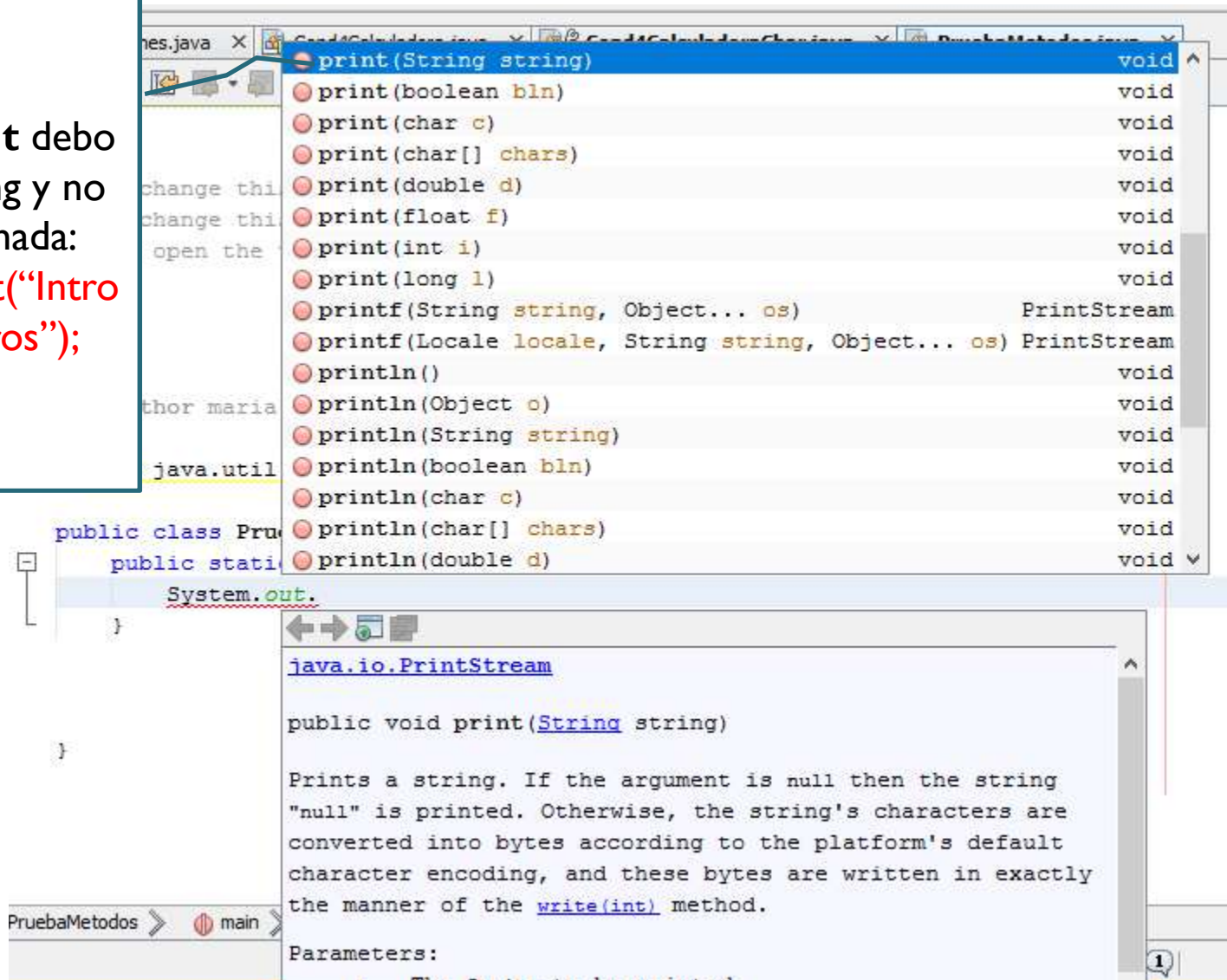
- 
- Compliquemos el ejemplo simplemente para utilizar algun otro método que ya conocemos (round)
 - Ahora en vez de hallar el real mayor, convertiremos primero los números introducidos a su entero más próximo y hallaremos el mayor de dichos enteros


```
14  import java.util.Scanner;
15
16  public class PruebaMetodos {
17      public static void main(String[] args) {
18          long a,b;
19          double x,y;
20          Scanner teclado=new Scanner(System.in);
21          System.out.print("Introduce dos números");
22          x=Double.parseDouble(teclado.next());
23          y=Double.parseDouble(teclado.next());
24          a=Math.round(x);
25          b=Math.round(y);
26          System.out.print("El entero más próximo mayor es:");
27          System.out.println(Math.max(a,b));
28      }
29
30
31
32  }
33
```

- 
- Veamos la ayuda de autocompletado que nos muestra netbeans al ir a utilizar cada método

Al método **print** debo
pasarle un String y no
me devuelve nada:

**System.out.print("Intro
duce 2 números");**



parseFloat espera un String y devuelve un double:

`x=Double.parseDouble(teclado.next());`

The screenshot shows an IDE with a Java file named `baMetodos.java`. The code includes imports for `java.util.Scanner` and `System`, and a class `P` with a static method `main`. Inside `main`, a `Scanner` object is created and the line `x=Double.parseDouble(teclado.next());` is highlighted. A tooltip for the `Double.parseDouble` method is displayed, showing its signature and documentation. The tooltip text is as follows:

`Double`
static double `parseDouble(String string)` throws `NumberFormatException`
Returns a new double initialized to the value represented by the specified String, as performed by the `valueOf` method of class `Double`.
Parameters:
 s - the string to be parsed.
Returns:
 the double value represented by the string argument.
Throws:
 `NullPointerException` - if the string is null
 `NumberFormatException` - if the string does not contain a

At the bottom of the IDE, the breadcrumb navigation shows `baMetodos > main`.

Next no recibe ningún valor y devuelve un string
`x=Double.parseDouble(teclado.next());`

```
/**
 * @author maria
 */
import java.util.Scanner;

public class PruebaMetodos {
    public static void main(String[] args) {
        int a,b;
        double x,y;
        Scanner teclado=new Scanner(System.in);
        System.out.print("Introduce un número: ");
        x=Double.parseDouble(teclado.next());
    }
}
```

next()	String
next(Pattern ptrn)	String
next(String string)	String
nextLine()	String
nextBigDecimal()	BigDecimal
nextBigInteger()	BigInteger
nextBigInteger(int i)	BigInteger
nextBoolean()	boolean
nextByte()	byte
nextByte(int i)	byte
nextDouble()	double
nextFloat()	float
nextInt()	int
nextInt(int i)	int
nextLong()	long
nextLong(int i)	long
nextShort()	short

round espera un double y devuelve un long:

```
long a,b;  
double x,y;  
a=Math.round(x);
```

```
/*  
 * @author M...  
 */  
Returns the closest long to the argument, with ties  
import java.math.RoundingMode; rounding to positive infinity.  
  
public class Special cases:  
    public static  
        int  
        double  
        Scanner  
        System  
        x=Double  
        y=Double  
        a=round  
        b=Math.round
```

- If the argument is NaN, the result is 0.
- If the argument is negative infinity or any value less than or equal to the value of Long.MIN_VALUE, the result is equal to the value of Long.MIN_VALUE.
- If the argument is positive infinity or any value greater than or equal to the value of Long.MAX_VALUE, the result is equal to the value of Long.MAX_VALUE.

round(float f)	int
random()	double
rint(double d)	double
round(double d)	long


```

public class Prue
    public stati
        long a,b
        double x
        Scanner
        System.o
        x=Double
        y=Double
        a=Math.r
        r
        out.printl

```

println()	void
println(Object o)	void
println(String string)	void
println(boolean bln)	void
println(char c)	void;
println(char[] chars)	void;
println(double d)	void
println(float f)	void
println(int i)	void
println(long l)	void

Tenemos varias opciones para utilizar el método **println**, depende de lo que queramos escribir (es un método sobrecargado)

max espera dos long y devuelve un long:

long a,b;

System.out.println(Math.max(a,b));

Utilizamos el println que recibe un long como parámetro y le pasamos el long que devuelve max(a,b)

```
/*
 * To change this license header, choose License Headers in Project
 * Properties.
 * To change this template file, choose the template in the
 * Resources folder.
 */
```

```
/**
 *
 * @author maria
 */
import java.util.Scanner;
```

```
public class PruebaMetodos {
    public static void main(
        long a,b;
        double x,y;
        Scanner teclado=new
        System.out.print("In
        x=Double.parseDouble
        y=Double.parseDouble
        a=Math.round(x);
        b=Math.round(y);
        System.out.println(Math.m
    }
```

[g.Math](#)

```
static long max(long l, long l1)
```

Returns the greater of two long values. That is, the result is the argument closer to the value of [Long.MAX_VALUE](#). If the arguments have the same value, the result is that same value.

Parameters:


a - an argument.

b - another argument.

Returns:

the larger of a and b.

max(double d, double d1)	double
max(float f, float f1)	float
max(int i, int i1)	int
max(long l, long l1)	long
min(double d, double d1)	double
min(float f, float f1)	float
min(int i, int i1)	int
min(long l, long l1)	long
multiplyExact(int i, int i1)	int
multiplyExact(long l, long l1)	long

- 
- Hasta ahora hemos utilizado ciertos métodos que se definen en librerías del propio lenguaje:
 - `double rx=Math.sqrt(78);`
 - `int i=entrada.nextInt();`
 - `System.out.println(“Hola a todos”);`
 - Podemos observar en las llamadas anteriores a métodos que:
 - Todos los métodos tienen un **identificador**: `sqrt`, `nextInt`, `println`
 - Después del identificador, y entre paréntesis, figuran los **parámetros** del método: `78`, `“Hola a todos”`. Pueden no tener parámetros
 - Algunos métodos devuelven un **resultado** (`sqrt`), otros métodos no devuelven ningún resultado explícitamente (`println`)

Ventajas

- El programador también puede definir sus métodos propios
- Ventajas:
 - Ahorra esfuerzo y tiempo cuando en la resolución de un problema se repite frecuentemente una misma secuencia de acciones: **Reutilización del código**
 - Facilita la resolución en términos de **sub-problemas más sencillos**
 - **Incrementa la legibilidad** de los programas



Definición y llamada

- **Definición:** Se establece lo que hace el método y si es necesario, los datos que utiliza y el resultado que devuelve
- **Llamada:** utilizando el identificador y pasándole los datos necesarios, se llama a un método para que realice su función

¿dónde definimos los métodos?

- Una clase está compuesta por:
 - atributos
 - Métodos
- Tanto unos como otros aparecen definidos dentro de la definición de una clase
- No importa el orden en la definición de los métodos. Aunque suele aparecer la definición antes que las llamadas
- Los métodos estáticos de una clase pueden llamarse únicamente por su nombre dentro de la clase donde se han definido.
- Una clase puede utilizar métodos definidos en otras clases. Para llamar a un método estático definido en otra clase habrá que llamarlo como:

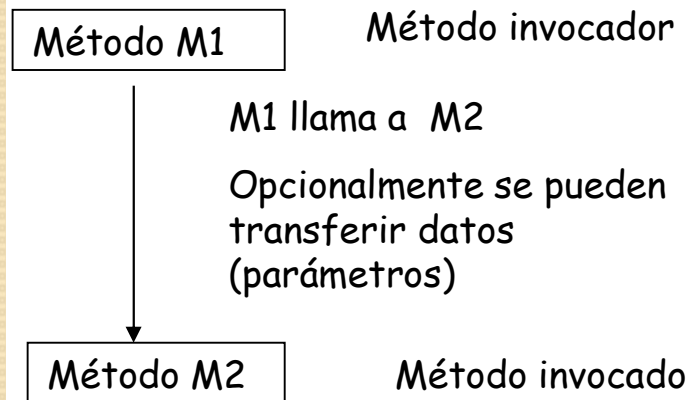
`NombreClase.nombreMetodo`

Conexión entre métodos

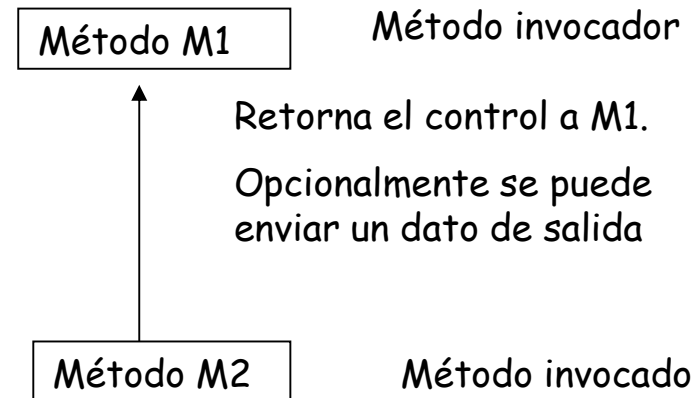
- Un método puede ser invocado, o *llamado*, desde otro método:
 - Cuando un método llama a otro método, se transfiere el control al segundo método
 - Cuando finaliza la ejecución del código del segundo método, éste devuelve el control al método que lo invocó
- En JAVA:
 - Un programa siempre empieza a ejecutarse en el método 'main' o principal
 - El método main podrá invocar a otros métodos que, a su vez, pueden invocar a otros métodos
- Un método puede invocarse a sí mismo, esto se llama recursividad

Conexión entre módulos

Llamada a una método



Retorno de un método



Ejemplo

Llamada

```
1 EjemplosTema et=new EjemplosTema();  
   int rdo;  
4   rdo=et.suma(3, 4); 2  
5   System.out.println(rdo);
```

Definición

```
public class EjemplosTema {  
  
   int suma(int num1, int num2) {  
3     return num1 + num2;  
   }
```

1. Primera instrucción del programa principal
2. Llamada al método, se le pasa el control
3. Se ejecutan las instrucciones del método y se devuelve el resultado
4. Se recoge el resultado del método
5. Continúa la ejecución del programa principal

Cómo crear un método. Definición

```
modificadores tipo_devuelto nombre( [argumentos] )  
{  
    ' Instrucciones de la función,  
    ' incluyendo opcionalmente la instrucción return  
}
```

Ejemplo:

```
static int suma (int a, int b){  
    return a+b;  
}
```

En este tema
practicamos
con métodos
static
Veremos los
diferentes
modificadores
más adelante

Podemos ver otro ejemplo de cabecera de definición de un módulo con varios modificadores en la definición del módulo main: **public static void main(String args[])**

Cómo crear un método

```
static tipo_devuelto nombre( [parámetros ] )  
{  
    ' Instrucciones de la función,  
    ' incluyendo opcionalmente la instrucción Return  
}
```

- **Modificador:** los veremos más adelante

- **tipo_devuelto:**

Es el tipo de dato del valor que devuelve la función

Si no retorna ningún valor escribimos void (en ese caso lo llamaríamos procedimiento)

- **parámetros:**

tipo1 variable1, tipo2 variable2,..., tipoN variableN

➤ entre [] significa que es opcional

Es decir puede o no tener parámetros

Tipos de métodos

- En función del valor devuelto:
 - **Procedimientos:** pueden
 - No devolver ningún valor
 - Devolver varios valores a través de los parámetros.
 - **Funciones:** devuelve un único valor que se recoge en la llamada al método.

Funciones

- Son subprogramas cuyo cometido principal es el cálculo de un único valor:
 - La línea de cabecera debe indicar también el tipo del valor devuelto por la función (En java será distinto de void)
 - Dentro del código debe haber al menos una instrucción que devuelva el valor resultante.

Cómo crear un método. Valor de retorno

- La instrucción return
 - Especifica el valor que devuelve el método (*valor de retorno*)
 - Devuelve el control inmediatamente al método que origina la llamada

```
static int maximo (int x, int y) {  
    if (x>=y)  
        return x;  
    else  
        return y;  
}
```

Procedimientos

- Se utilizan cuando el subprograma no tiene como función principal el cálculo de un único valor.
- Puede darse el caso de que el subprograma:
 - no tenga que realizar ningún cálculo (por ejemplo mostrar un menú por pantalla)
 - O que tenga que devolver varios datos, modificando sus parámetros
- En JAVA los procedimientos son los que tienen tipoDato VOID

Procedimiento

```
public static void saluda() {  
    System.out.println("Hola");  
}
```

Operaciones.saluda();

Función

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

```
EjemplosTema et=new EjemplosTema();  
int rdo;  
rdo=et.suma(3, 4);  
System.out.println(rdo);
```

Parámetros

- **Formales:**
 - Aparecen en la definición del módulo en la cabecera del mismo entre paréntesis.
 - Son variables que recogen los datos pasados en la llamada.
- **Actuales**, también llamados **argumentos**:
 - Son los datos que se le pasan a un método en la llamada al mismo
 - Pueden ser valores constantes, o variables que los contengan.
 - Una vez efectuada la llamada son volcados en los parámetros en el mismo orden en que aparecen.

Tipos de parámetros

■ POR VALOR

proporcionan información al módulo y que no son modificados por éste. El módulo crea una copia de los argumentos y es esto lo que utiliza

son parámetros de ENTRADA de datos

POR REFERENCIA

Proporciona al módulo la dirección de los datos origen. Así el módulo accede a la misma para leer o guardar los datos.

son parámetros de ENTRADA/SALIDA

Tipos de parámetros en JAVA

Por valor:

Los tipos básicos y las cadenas se pasan siempre por valor:

- String
- byte
- short
- int
- char,
- float
- double,
- boolean

Por referencia:

Los objetos y tipos referencia:

- Vectores
- Arreglos
- Objetos

Puesto que de momento sólo hemos visto los tipos básicos, hablaremos de los parámetros por referencia a partir del próximo tema

Cómo llamar a un método

Método invocador

Método M1



Método M2

Método invocado

En M1:

Parámetros actuales:

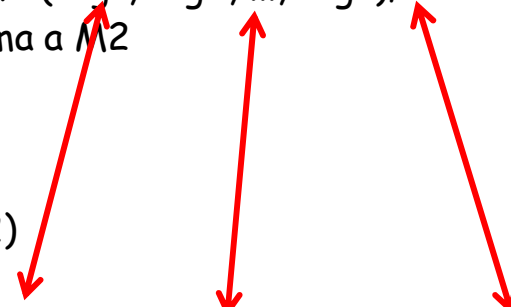
contienen los valores con que M1 llama a M2

M2 (arg1, arg2, ..., argn);

Argumentos formales

(especificados en la cabecera de M2)

M2 (tipo1 var1, tipo2 var2, ..., tipoN varN)



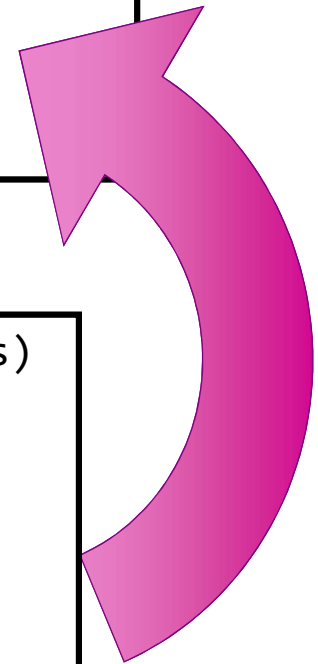
Los parámetros actuales y los formales **deben coincidir en número, orden y tipo.**

Cómo llamar a un método. Ejemplos

```
static int suma (int a, int b){  
    return a+b;  
}
```

Ambos métodos forman parte de la misma clase, por eso en la llamada no es necesario especificar el nombre de la clase

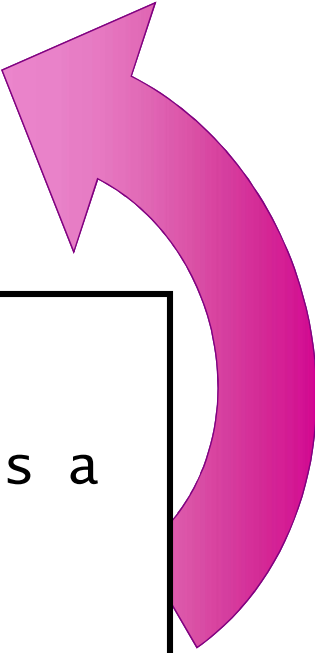
```
public static void main(String[] args)  
{  
    .....  
    y=suma( x , x*67 );  
    z=suma( x+y , 45 );  
}
```



Cómo llamar a un método. Ejemplos

```
static int maximo (int x, int y) {  
    if (x>=y)  
        return x;  
    else  
        return y;  
}
```

```
int a,b,c,d;  
int max1, max2, max;  
//Asignación de valores a  
las variables  
max1=maximo(a,b);  
max2=maximo(c,d);  
max=maximo(max1,max2);
```



Ámbito de las variables

- Se entiende por ámbito de una variable la parte del programa en que es conocida o se puede utilizar, dicha variable.
- Las variables en función de su ámbito se pueden clasificar en:
 - Globales
 - Locales.

Variables Globales

- Se definen fuera de todos los métodos, al principio de la clase.
- Su ámbito de utilización es la clase en la que está declarada.
- Su tiempo de almacenamiento está limitado al tiempo que dura la ejecución del programa

Variables locales

- Se definen al comienzo del bloque de instrucciones de un módulo.
- Su ámbito es el bloque donde está definida
- Su tiempo de almacenamiento es la duración de la ejecución del bloque donde está definida.

Precedencia variables

En caso que una variable local tenga el mismo nombre que una global tiene preferencia la más cercana, es decir, la variable local, y los cambios NO se aplicarían a la global.

Interconexión de métodos

- Los métodos pueden intercambiar información de 3 formas distintas:
 - Paso de parámetros
 - Funciones que retornan un valor
 - Uso de variables globales

Módulos recursivos

- Son módulos que se caracterizan por llamarse a si mismos
- Para crearlos es FUNDAMENTAL establecer una condición de terminación o caso base que permita la finalización de las llamadas de un módulo a sí mismo.

Ejemplo de módulo recursivo: Factorial

Cada llamada queda en suspensión hasta que se llega al caso base y se van devolviendo el control a las llamadas anteriores

Condición de salida	Proceso
$0! = 1$ $1! = 1$	<p>The diagram illustrates the recursive calculation of 6!. It shows a series of steps where each factorial is calculated by multiplying the current number by the factorial of the previous number. The steps are: $6! = 6 * 5! = 6 * 120 = 720$, $5! = 5 * 4! = 120$, $4! = 4 * 3! = 24$, $3! = 3 * 2! = 6$, $2! = 2 * 1! = 2$, and finally the base case 1. Red arrows indicate the flow of the calculation, starting from the base case and moving upwards through each step.</p>