

## Contenido

1. CONCEPTOS: MÉTODOS, FUNCIONES, PROCEDIMIENTOS. ....	2
2. SINTAXIS. ....	4
LLAMADA AL MÉTODO O FUNCIÓN .....	5
3. PASO DE INFORMACIÓN A UNA FUNCIÓN. ....	9
4. VALOR DEVUELTO POR UNA FUNCIÓN. ....	11
5. ÁMBITO DE LAS VARIABLES. ....	12
6. SOBRECARGA DE FUNCIONES. ....	15
7. RECURSIVIDAD. ....	16
8. EJERCICIOS. ....	17

## 1. CONCEPTOS: MÉTODOS, FUNCIONES, PROCEDIMIENTOS.

En Java son imprescindibles los **métodos o funciones**, un programa Java estará formado por un conjunto de ellos. Ya hemos visto el método **main()** en todos los programas que hemos realizado hasta ahora:

```
public static void main(String[] args) {
    instrucciones;
}
```

Los métodos son indispensables para programar ya que nos van a facilitar el mantenimiento del programa. Son como un **pequeño programa o subprograma** dentro de otro. Tiene los mismos componentes que un programa y se ejecuta cuando lo ordena el programa principal.

A los **métodos** en Java, también se les llama **procedimientos o funciones**. Veamos sus diferencias:

- Las **funciones** y los **métodos** son un conjunto de líneas de código (instrucciones), encapsulados en un bloque, usualmente reciben parámetros (aunque pueden no recibir nada) cuyos valores se utilizan para efectuar operaciones y adicionalmente retornan un valor.

Funciones y métodos realizan las mismas tareas. La diferencia entre uno y otro está en el contexto, es decir en la manera en que hacemos uso de uno u otro:

- El método SIEMPRE está asociado a un objeto.

```
String cad = "Hola";
System.out.println("Cadena mayuscula: " + cad.toUpperCase());
```

```
Scanner sc = new Scanner(System.in);
int n = sc.nextInt();
```

- La función existe por sí sola, sin necesidad de crear un objeto para ser usada, debe ser **static**.

```
System.out.println("Ejemplo Función: " + Math.ceil(6.7));
```

- Un **procedimiento** es básicamente un método cuyo tipo de retorno es **void** que no nos obliga a utilizar una sentencia **return**.

### Resumiendo: Un método o función es

- Un grupo de sentencias que realizan una tarea concreta dentro de un programa y que están identificadas con un nombre. Pueden devolver o no un valor y recibir o no valores.

## PORQUÉ USAR FUNCIONES:

EVITAR DUPLICIDAD DE CÓDIGO.

FACILITAR EL MANTENIMIENTO DEL PROGRAMA.

CUANDO LA MISMA FUNCIONALIDAD SE REALIZA EN DISTINTOS LUGARES DE NUESTRO PROGRAMA. SE ETIQUETA CON UN NOMBRE ESE TROZO DE CÓDIGO.

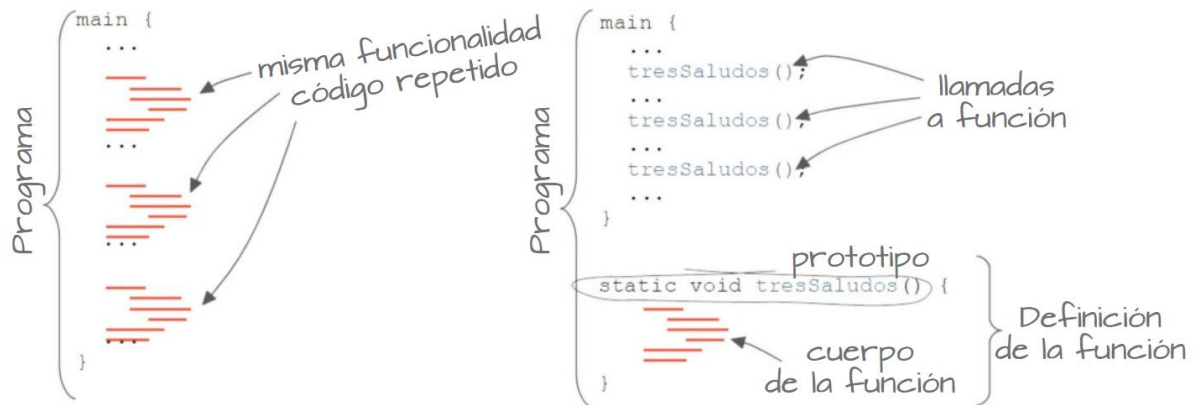


Figura 4.1. Representación de un programa sin y con funciones.

```
public class EjemploInicial1 {
```

```
    public static void main(String[] args) {
```

```
        System.out.println ("Llamada1:");
```

```
        tresSaludos();
```

```
        System.out.println ("Llamada2:");
```

```
        tresSaludos();
```

```
        System.out.println ("Llamada3:");
```

```
        tresSaludos();
```

```
    }
```

```
    static void tresSaludos() {
```

```
        System.out.println ("Voy a saludar tres veces:");
```

```
        for (int i=0; i<3; i++) {
```

```
            System.out.println ("Hola.");
```

```
        }
```

```
    } //
```

```
} //fin
```

Llamada

Declaración

CUERPO DE LA FUNCIÓN

1. Las instrucciones del programa principal se ejecutan hasta que encuentra la llamada a la función.
2. La ejecución salta a la definición de la función.
3. Se ejecuta el cuerpo de la función.
4. Cuando la ejecución del cuerpo termina, retornamos al punto del programa desde donde se invoca la función.
5. El programa continua su ejecución.

```
import java.util.Scanner;

public class EjemploInicial2 {
    static Scanner sc = new Scanner(System.in);

    public static void main(String[] args) {

        int dep = LeerEntero("Dame el número de Departamento: ");
        int emp = LeerEntero("Dame el número de Empleado: ");
        System.out.println("Dep: " + dep + ", Empleado: " + emp);

    }

    private static int LeerEntero(String cad) {
        System.out.print(cad);
        int n = sc.nextInt();
        sc.nextLine();
        return n;
    }
}

// fin
```

**Llamada** (punta a la llamada a `LeerEntero` en `main`)

**Declaración** (punta a la declaración de `LeerEntero`)

**CUERPO DE LA FUNCIÓN** (punta al cuerpo de `LeerEntero`)

**Valor de retorno** (punta al `return n;` en `LeerEntero`)

## 2. SINTAXIS.

### Declaración de un método o una función:

```
[acceso][modificador] tipoDato identificador (lista de parámetros){
    sentencia_1;
    sentencia_2;
    ...
    return valor;
}
```

**CUERPO DE LA FUNCIÓN** (punta al cuerpo de la función)

```
public static void main(String[] args) {
}
```

La sentencia **return** (no es obligatoria) se emplea para salir de un método y **opcionalmente** devolver un **valor**.

Donde **[acceso][modificador]**

**acceso:** **public** o **private** éste es opcional, si no ponemos nada, se asume el modificador de acceso por defecto.

**modificador:** el segundo componente es el modificador que puede ser **final** o **static** (o ambas), también es opcional. (*final en la declaración de un método: en este contexto una clase que herede un método final no puede redefinirlo en la clase hija, no se puede hacer un **override**. En la declaración de una clase impide crear clases derivadas de dicha clase – Se usa en la **herencia***)

Donde **tipoDato**: Es el tipo de dato del **valor** devuelto por la función o método. Si no devuelve nada se indica **void**. Por ejemplo, Si devuelve un valor numérico entero ponemos: **int**, Si devuelve una cadena de caracteres ponemos **String**, etc.

Donde la **lista de parámetros** tiene este formato:

```
tipoDato param1, tipoDato param2, tipoDato param3, . . .
```

Ejemplo de declaración de la función **main()**:

```
public static void main(String[] args) {
    . . . . .
}
```

Ejemplo de declaración de la función **LeerEntero()**:

```
private static int LeerEntero(String cad) {
    . . . . .
}
```

**Ejemplos de declaración de funciones** (EjemploInicial3.java):

```
// Función sin parámetros, visualiza 2 saludos en pantalla
// No recibe ni devuelve ningún valor
private static void Saludar() {
    System.out.println("Buenos días");
    System.out.println("Saludos desde Guadalajara");
}

// Función que recibe 2 parámetros enteros y muestra la suma en pantalla
private static void SumaNumeros(int a, int b) {
    int suma = a + b;
    System.out.println("La suma es: " + suma);
}

// Función que recibe 2 parámetros enteros y devuelve la suma
private static int SumaDosNumeros(int a, int b) {
    int suma = a + b;
    return suma;
}
```

## LLAMADA AL MÉTODO O FUNCIÓN

El formato general para realizar la llamada a un método o función cuando devuelve un valor es el siguiente:

```
tipoDato var = identificador(lista de valores);

    int sum = SumaDosNumeros(20, 30);
//el método devuelve un valor entero, enviamos dos valores enteros al método

    int emp = LeerEntero("Dame el número de Empleado: ");
//el método devuelve un entero, enviamos una cadena (un String)
```

Si el método o función no devuelve nada y recibe valores hacemos así la llamada:

```
identificador(lista de valores);  
  
SumaNumeros(100, 200);  
//el método no devuelve nada, enviamos dos valores al método
```

Si el método no recibe ningún valor ni devuelve nada hacemos así la llamada:

```
identificador();  
  
Saludar();  
//el método no devuelve nada, no enviamos valores al método  
  
tresSaludos();
```

Donde la **lista de valores** tiene este formato:

```
expresion1, expresion2, expresion3, . . . .  
  
expresión puede ser: una variable, una constante, cualquier expresión, etc. Ejemplos:
```

```
SumaNumeros(a * b, b * 5);  
SumaNumeros(10, 20);  
  
int x = 20, y = 30;  
int s = SumaDosNumeros(SumaDosNumeros(30,40), y*y);
```

Ejemplos de llamadas a las funciones anteriores:

```
Saludar(); //llamada a la función, se ejecuta código de la función  
  
(EjemploInicial3.java)  
int a = 2, b = 9, c = 4;  
SumaNumeros(100, 200); //llamada a la función, se ejecuta código de la función  
SumaNumeros(a * b, b * c);  
  
int sum = SumaDosNumeros(20, 30);  
System.out.println("La suma es: " + sum);  
  
System.out.println("La suma es: " + SumaDosNumeros(20 * a, 30 * b));
```

**OTRO EJEMPLO, la función recibe y devuelve un valor:**

The diagram illustrates the relationship between a function call and its declaration. A box labeled "Llamada" (Call) points to the `cubo(7.5)` in the `main` method. Another box labeled "Declaración" (Declaration) points to the `cubo` method definition. A third box labeled "Valor de retorno" (Return value) points to the `return` statement in the `cubo` method.

```
public class PruebaCubo {
    public static void main (String [] args){
        System.out.println("El cubo de 7.5 es: " + cubo(7.5));
    }

    public static double cubo (double x) {
        return x*x*x;
    }
}
```

Otro ejemplo de la necesidad de definir una función. Tenemos un programa que lee 4 edades por teclado y antes de leer cada edad muestra un mensaje en consola indicando lo que se pide:

```
import java.util.Scanner;
public class LeerEdades1 {

    static Scanner sc;

    public static void main(String[] args) {
        sc = new Scanner(System.in);

        System.out.print("Dame primera edad: ");
        short ed1 = sc.nextShort();

        System.out.print("Dame segunda edad: ");
        short ed2 = sc.nextShort();

        System.out.print("Dame tercera edad: ");
        short ed3 = sc.nextShort();

        System.out.print("Dame cuarta edad: ");
        short ed4 = sc.nextShort();

        //para que salgan decimales hacer cast
        float media = (float) (ed1 + ed2 + ed3 + ed4) / 4;

        System.out.printf("\nLa edad media es: %,5.2f %n", media);
    }
}
```

The screenshot shows the output of the program in a console window. The user has entered four ages: 12, 15, 10, and 17. The program calculates the average and displays it as 13,50.

```
<terminated> LeerEdades1 [Java Application] C
Dame primera edad: 12
Dame segunda edad: 15
Dame tercera edad: 10
Dame cuarta edad: 17
La edad media es: 13,50
```

Hay dos líneas que se repiten en el código: pintar en pantalla y la lectura del entero por teclado. Son estas dos líneas:

```
System.out.print("Dame primera edad: ");
short ed1 = sc.nextShort();
```

Se puede hacer una función que reciba el mensaje a visualizar en pantalla y devuelva el número entero leído por teclado, y en lugar de escribir esas dos líneas escribiríamos la llamada a la función, el programa quedaría así:

```
import java.util.Scanner;

public class LeerEdades2 {

    static Scanner sc;

    public static void main(String[] args) {
        sc = new Scanner(System.in);

        short ed1 = LeerNumero("Dame primera edad: ");
        short ed2 = LeerNumero("Dame segunda edad: ");
        short ed3 = LeerNumero("Dame tercera edad: ");
        short ed4 = LeerNumero("Dame cuarta edad: ");

        //para que salgan decimales hacer cast
        float media = (float) (ed1 + ed2 + ed3 + ed4) / 4;
        System.out.printf("\nLa edad media es: %,5.2f %n", media);
    }

    private static short LeerNumero(String m) {
        System.out.print(m);
        short ed = sc.nextShort();
        return ed;
    }
}
```


## Normas de construcción de un método

- La situación relativa en el código fuente de la llamada y la declaración de un método no es importante
- Los modificadores permiten construir diferentes tipos de métodos
- El tipo de dato indica el tipo de valor que devuelve un método
- Parámetros ó argumentos:
  - Sirven para intercambiar información con el método
  - El número de parámetros puede ser 0 o más
  - Es necesario indicar el tipo de dato e identificador de cada uno de los parámetros del método
  - El tipo y número de parámetros en la llamada (**parámetros reales**) ha de coincidir con el tipo y número de parámetros en la cabecera de declaración del método (**parámetros formales**)
- La sentencia **return** se incluye en el cuerpo del método (es opcional).



### 3. PASO DE INFORMACIÓN A UNA FUNCIÓN.

UN **PARÁMETRO DE ENTRADA** DE UNA FUNCIÓN ES UNA VARIABLE LOCAL A LA QUE SE LE ASIGNAN VALORES EN CADA LLAMADA:



```
static void variosSaludos(int veces) {
    for(int i = 0; i < veces; i++) {
        System.out.println("Hola.");
    }
}
```

(EjemploInicial4.java)

VALORES EN LA LLAMADA, por ejemplo un valor constante y el valor de una variable:

```
variosSaludos(2);
int n = 3;
variosSaludos(n);
```

LOS PARÁMETROS TOMAN SU VALOR DE LA LLAMADA A LA FUNCIÓN. ASÍ EN DISTINTAS LLAMADAS SE PUEDEN ASIGNAR DISTINTOS VALORES.

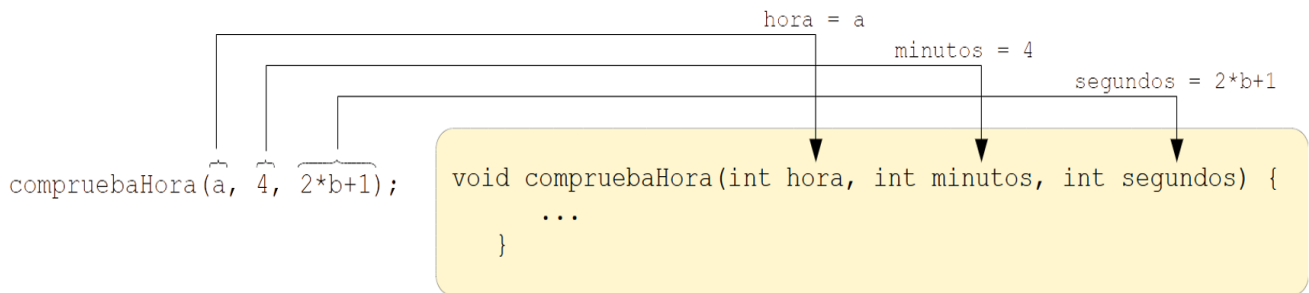
EL NÚMERO DE PARÁMETROS O ARGUMENTOS DE LOS MÉTODOS PUEDE SER 0, 1, 2..

Cada dato utilizado en la llamada a una función será asignado a un parámetro de entrada, especificado en la definición de la función con la siguiente sintaxis:

```
tipo nombreFuncion(tipo1 parametro1, tipo2 parametro2...) {
    cuerpo de la función
}
```

Mecanismo de paso de parámetros en una función:

**Parámetros:** son variables locales de la función que han sido inicializadas en la llamada.



En Java los parámetros toman su valor como una copia del valor de la expresión o variable utilizada en la llamada; este mecanismo de paso de parámetros se denomina **paso de parámetros por valor o por copia**.

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.

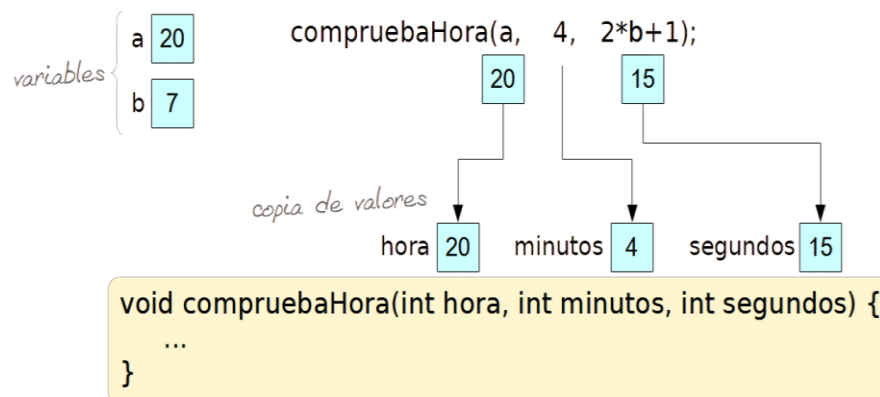


Figura 4.5. Mecanismo de copia de valores a los parámetros.

Ver Actividades Resueltas 4.1, 4.2. 4.3.

#### Actividades:

- Método que reciba dos números leídos de teclado; y muestre en pantalla los números pares comprendidos entre ellos. (variación del 4.2)
- Método que reciba un número leído por teclado y muestre en pantalla mediante \* un cuadrado. Por ejemplo si recibe:

4 debe pintar ***** ***** ***** *****	5 debe pintar ***** ***** ***** ***** *****	2 debe pintar ** **
---	--	---------------------------

El proceso se repetirá hasta leer un numero sea  $\leq 0$ .

- Igual que el anterior pero ahora el método también recibe el carácter con el que se pintará el cuadrado. El carácter se leerá por teclado.

## 4. VALOR DEVUELTO POR UNA FUNCIÓN.

Si un método devuelve un valor hay que indicar en su declaración el tipo de dato del valor devuelto:

```
[modificadores] tipoDato identificador (lista de parámetros) { }
```

### Modificadores (se verán en el Capítulo 7)

El modificador **public** indica que el método es accesible fuera del código de la clase a la que pertenece a través del operador punto. El modificador **private** indica que solamente es accesible a través de los métodos de la propia clase. El modificador **protected** indica que solo podrán tener acceso al método las clases y objetos que pertenezcan al mismo paquete.

Y en el cuerpo del método debe aparecer la sentencia **return** devolviendo el valor de ese tipo **de dato**. Por ejemplo, el método **suma** devuelve un valor entero (la variable *resul* es int):

```
static int suma (int x, int y) {
    int resul = x + y;
    return resul;
}
```

Si el método no devuelve nada se debe declarar de tipo **void** y no es necesario que aparezca la sentencia **return**:

```
static void pintaMensaje (String m) {
    System.out.println(m);
}
```

(EjemploInicial4.java)

### return y métodos void

- No es obligatorio que la llamada a un método implique la devolución de un valor.
- Métodos con el modificador **void**:
  - en lugar del identificador de un tipo de dato
  - no devuelven un valor al ejecutarse la sentencia **return**;
  - Son equivalentes a los procedimientos de Pascal, PL/SQL, etc

**Ver Actividades Resueltas 4.4 a 4.8**

## 5. ÁMBITO DE LAS VARIABLES.

VARIABLES LOCALES. Visibilidad en el método donde se declaran.

VARIABLES DE BLOQUE. Visibilidad en el bloque donde se declaran.

VARIABLES DE CLASE (GLOBALES). Visibilidad en toda la clase.

```
void func1() {
    int a, b;
    ...
    while(...) {
        int c;
        ...
    } //del while
} //de func1
void func2() {
    double a;
    ...
    if(...) {
        int x;
        ...
    } //del if
} //de func2
```

*ámbito func1. Podemos utilizar las variables: a y b*  
*ámbito while. Podemos utilizar las variables: a, b y c*  
*ámbito func2. Podemos usar solo la variable: a*  
*ámbito if. Podemos usar las variables: a y x*

- Variables locales
- Variables de bloque

**Figura 4.3.** Ámbitos de distintas variables. Aunque las variables a de func1() y de func2() (marcadas en rojo) compartan identificador, son variables distintas.



© Ediciones

```
public class ambitovariables {

    static int a = 4; // variable de clase

    public static void main(String[] args) {
        int b = 20; // local
        funcion1();
        funcion2(a, b);
    }

    static void funcion1() {
        int a = 5, b = 1; // locales
        while (a > 0) {
            int c = b; // de bloque
            System.out.println(a + c);
            a--;
            c++;
        }
    }

    static void funcion2(int a, int b) {
        int z; //local
        z = a + b; //a y b locales
        System.out.println(z);
    }
} //
```

## Ejercicios ámbito de variables. (en papel)

Dada esta clase (sin errores), ¿Cuál es el resultado de salida por pantalla al compilar y ejecutar después?

```
public class Ejer23_AmbitoVariables {
    static int z; //valor inicial 0
    public static void main(String[] args) {
        int z = 6;
        System.out.println(z);
        funcion1();
        System.out.println(z);
        z+=8;
        funcion2();
    }
}
```

a)	b)	c)	d)
6	6	6	0
10	4	9	10
5	5	9	5
6	6	6	10
18	8	14	18

```
static void funcion1() {
    int z = 5;
    funcion2();
    System.out.println(z);
}
```

```
static void funcion2() {
    z = z+4;
    System.out.println(z);
}
```

```
} }
```

Dada esta clase (sin errores), ¿Cuál es el resultado de salida por pantalla al compilar y ejecutar después?

```
public class Ejer24_AmbitoVariables {
    static int x; // valor inicial 0

    public static void main(String[] args) {
        int z = x + 10;

        System.out.println("x= " + (++x) + ", z = " + z);

        z = funcion1(x, (z + x));

        System.out.println("x= " + (++x) + ", z = " + z);
    }

    static int funcion1(int xx, int zx) {
        x = zx + funcion2(xx);
        return zx * x;
    }

    static int funcion2(int xx) {
        x += 4 + xx;
        System.out.println(x);
        return x * 2;
    }
}
```

EJERCICIO: Hacer una clase Java con las siguientes especificaciones  
(EjercicioVariosMetodosIniciales.java):

- Realiza una función que reciba dos números y devuelva el producto.
- Realiza una función que reciba dos números y muestre en pantalla el producto.
- Realiza una función que reciba una variable **short** y devuelva **true** si la variable está comprendida entre 1 y 12; y **false** si no lo está.
- Realiza una función que reciba una variable **short** y devuelva el nombre del día de la semana: “Lunes” si la variable vale 1, “Martes” si la variable vale 2, “Miércoles” si la variable vale 3, y así hasta 7. Si el día no está comprendido entre 1 y 7 debe devolver “INCORRECTO”.
- Realiza una función que lea un número por teclado en un rango de valores. La función recibirá un String, similar a la función *LeerNumero(String m)* del ejemplo anterior, y los valores máximo y mínimo que debe tener el entero (de tipo int). La función devolverá el valor entero (tipo int) que se debe ajustar al rango. Dentro de la función si el número leído no está en el rango se debe leer de nuevo.
- Realiza el método **main()** que haga uso de las funciones anteriores. Los números que se envían a la función se leerán por teclado.

## EJERCICIOS FECHAS EjercicioFechas.java:

- 1) Escribe una función que reciba un número (**int**) y devuelva el nombre del mes. Por ejemplo, si recibe un 3 debe devolver Marzo.
- 2) Función que recibe un día del mes, un mes y un año (**int**) y devuelva **true** o **false** indicando si la fecha es correcta o no. Debe considerarse el caso de año bisiesto. Año válido desde 1900 Hasta 2030.
- 3) Escribe una función a la que le pasamos un mes y un año (de tipo **int**) y nos devuelva el número de días que tiene ese mes de tipo **int**. Año válido desde 1900 Hasta 2030. Considera los años bisiestos. En un año bisiesto el mes de Febrero tiene 29 días en lugar de 28. Si el año no es válido devuelve 0.
- 4) Escribe la función **main()** que haga uso de las funciones anteriores.

## FUNCION QUE RECIBA UN NÚMERO ENTERO Y DEVUELVA EL NÚMERO DE CIFRAS DEL NÚMERO.

### CONDICIONES:

NÚMERO MAYOR QUE 0.

Y NÚMERO COMPRENDIDO ENTRE 1 Y 999.999

SI NO SE CUMPLE LO ANTERIOR LA FUNCIÓN DEBE DEVOLVER 0  
SI SE CUMPLEN LAS CONDICIONES DEBE DEVOLVER EL NÚMERO DE CIFRAS

Hacer **EJERCICIO01** DEL APARTADO DE EJERCICIOS.

## 6. SOBRECARGA DE FUNCIONES.

- Varios métodos pueden compartir el mismo identificador
- El tipo y/o número de parámetros de los métodos que compartan identificador ha de ser diferente
- El compilador se encarga de averiguar cuál de los métodos se ejecuta al realizar la llamada

Función sobrecargada que calcula el máximo de 3 números

### Ejemplo:

```
public class PruebaSobrecarga {
    public static void main(String[] args) {
        int a = 34;
        int b = 12;
        int c = 56;
        System.out.println("a = " + a + "; b = " + b + "; c = " + c);
        System.out.println("El mayor de a y b es: " + mayor(a, b));
        System.out.println("El mayor de a, b y c es: " +
                           mayor(a, b, c));
    }

    // Definicion de mayor de dos numeros enteros
    public static int mayor(int x, int y) {
        return x > y ? x : y;
    }

    // Definicion de mayor de tres numeros enteros
    public static int mayor(int x, int y, int z) {
        return mayor(mayor(x, y), z);
    }

    // No se permite, da error
    public static String mayor(int x, int y) {
        return null;
    }

    public static int mayor(String x, int y) {
        return 0;
    }
}
```

### Ver Actividad Resuelta 4.9

HACER UN METODO SOBRECARGADO DE NOMBRE **SUMA** QUE RECIBA 2 ENTEROS Y DEVUELVA LA SUMA DE LOS ENTEROS, OTRO QUE RECIBA DOS DECIMALES, DEVUELVA LA SUMA DECIMAL

Hacer **EJERCICIO2** y **EJERCICIO3** DEL APARTADO DE EJERCICIOS.

## 7. RECURSIVIDAD.

- Java admite la recursión o recurrencia en la construcción de métodos
- Dentro del método se realiza una llamada al propio método
- Peligro: posibilidad de recursión infinita.
- Ejemplo de función recursiva: el factorial de un número entero.

```
public static long factorialR(int n) {  
    if (n == 0)  
        return 1;  
    else  
        return n * factorialR(n - 1);  
}
```

Ver Actividades Resueltas 4.10 a 4.12

### Actividades:

- Construir de forma recursiva un método potencia que devuelva el resultado de elevar una base a un exponente entero n. Recibe la base y el exponente.  
$$A \text{ elevado a } N = A * A * A \dots N \text{ VECES}$$
- Construir de forma recursiva un método que devuelva la suma de las cifras de un número. Recibe el número. Es decir si el número es 12345, se debe obtener la suma de las cifras: 1+2+3+4+5.
- Construir de forma recursiva un método que reciba un **entero positivo** N y devuelva la suma de 1 a N. Es decir si el número es 4, se debe devolver 10 la suma de los números del 1 al 4 (1 +2 +3 +4).

Hacer **EJERCICIO4** DEL APARTADO DE EJERCICIOS.



## 8. EJERCICIOS.

### EJERCICIO1:

#### REALIZA LOS SIGUIENTES MÉTODOS EN UNA CLASE JAVA:

- 1) Método que reciba un número double y un entero y devuelva el número de tipo double elevado al número de tipo entero.
- 2) Método que devuelva el entero mayor de tres enteros dados como parámetros.
- 3) Método que devuelva el entero menor de tres enteros dados como parámetros.
- 4) Método que devuelva el entero central de tres enteros dados como parámetros.
- 5) Método que devuelve verdadero o falso dependiendo de si el carácter que recibe es una letra minúscula o no.
- 6) Método que devuelve verdadero o falso dependiendo de si una fecha (dando el día, el mes y el año como parámetros) es válida o no.
- 7) Método que devuelve el número de cifras de un número entero que recibe como parámetro.
- 8) Método que recibe una letra y devuelve true si se trata de una vocal, en caso contrario devuelve false.
- 9) Método que recibe dos números enteros y un carácter indicando la operación a realizar con esos números: suma, resta, multiplica o divide ( +, -, \*, / ) y devuelve un entero con el resultado de la operación. En una división por 0 debe devolver 0.

#### REALIZA UN MÉTODO main QUE LLAME A LOS MÉTODOS ANTERIORES Y VISUALICE LA INFORMACIÓN PERTINENTE DEPENDIENDO DE LO QUE DEVUELVA CADA LLAMADA.

### EJERCICIO2:

Realiza un programa JAVA que pinte un menú de opciones e introduzca por teclado la opción deseada. Según sea la opción introducida se realizará el ejercicio correspondiente del menú. El proceso se repite hasta que la opción introducida sea 5. Si se introduce una distinta se muestra mensaje, se vuelve a pintar el menú y se pide de nuevo la opción.

Cada ejercicio se realizará en un método (estos no devuelven nada). Los ejercicios presentarán una cabecera anunciando el mismo.

El menú es el siguiente, el **método que pinta el menú no devuelve nada**:

```
*****
MENÚ DE EJERCICIOS
1 - NUMEROS PRIMOS.
```

2 - DIBUJO DEL ROMBO.

3 - TABLA DE MULTIPLICAR.

4 - DIBUJO DEL TABLERO DE AJEDREZ.

5 - FIN.

\*\*\*\*\*

Introduce opción (1 a 5):

- **Opción 1:** Método que calcule y visualice los N primeros números primos. N es un valor entero que se leerá de teclado. Deberá ser > 0.
- **Opción 2,** Método que haga el ejercicio del rombo donde, los caracteres del rombo se piden por teclado.
- **Opción 3:** Método que lea un número entero de teclado y muestre su tabla de multiplicar.
- **Opción 4:** Método que pinte un tablero de ajedrez. Carácter negro para el tablero: **char negro = '\u2588';** Para que al ejecutar salga ese carácter en consola acceder a las propiedades del fichero Java: Properties -> Resource -> en "Text file encoding" de la lista Other seleccionar: **UTF-8** y click en OK.

PARA INTRODUCIR DATOS ENTEROS POR TECLADO REALIZAR UN MÉTODO QUE RECIBA EL MENSAJE QUE APARECE ANTES DE LEER EL ENTERO, VALOR MINIMO Y VALOR MAXIMO QUE PUEDE TENER ESE ENTERO:

```
private static int LeerEntero(String mensaje, int min, int max) {}
```

Y DEVUELVA EL ENTERO SOLICITADO. CONTROLAR EXCEPCIONES.

PODEIS SOBRECARGAR ESTE MÉTODO SI EL ENTERO SOLO TIENE QUE SER > QUE 0, POR EJEMPLO:

```
private static int LeerEntero(String mensaje, int min) {}
```

EN CADA MÉTODO SE DEBE MOSTRAR EN PANTALLA UNA CABECERA DE LO QUE HACE.

### EJERCICIO3:

Realiza un **programa Java** de nombre **lecturaProductos** que nos permita introducir por teclado datos de las ventas de una serie de productos. Los datos son los siguientes:

```
String producto, int cantidad, int pvp, int tipoiva
```

Estos se introducirán en un proceso repetitivo que finalizará cuando la **cantidad introducida sea <= 0**.

Condiciones de los datos de entrada:

La **cantidad** no puede ser > a 9999, si supera ese valor se debe leer de nuevo.

En la entrada de datos **SI** se controlarán excepciones.

NO se utilizarán variables globales en el código (excepto la variable **Scanner**).

El **pvp** leído debe tener un valor entre 1 y 99999, si no está en ese rango se vuelve a leer de nuevo.

El **tipoiva** puede tener los siguientes valores (1, 2, 3):

1 => IVA General 21%.

3=> IVA Superreducido 4%

2=> IVA Reducido 10%.

Define **un único método para leer por teclado** la *cantidad*, el *pvp* y el *tipoiva*. Este método recibirá los valores máximo y mínimo que pueden tomar y un mensaje que indique lo que se va a leer. El método devolverá el dato deseado que tendrá que tener un valor correcto entre los límites definidos.

**Una vez leídos los datos del producto** se mostrará el importe total por pantalla. El importe total se calcula así:  $ImporteTotal = cantidad * pvp + (cantidad * pvp) \text{ multiplicado por } \%IVA \text{ correspondiente}$ .

#### EJERCICIO4:

Realiza **un método Java** con nombre **ultimo** que reciba un número N entero positivo > 0 y muestre en pantalla el último **dígito del factorial** de dicho número.

Por ejemplo Para N = 2, debe mostrar un 2 (1\*2 = 2)

Por ejemplo Para N = 4, debe mostrar un 4 (1\*2\*3\*4 = 24)

Por ejemplo Para N = 5, debe mostrar un 0 (1\*2\*3\*4 \*5 = 120)

Probar el método para pintar el factorial del 1 al 15 y mostrar el ultimo dígito:

Ultimo dígito de factorial de 1, (	1) 1
Ultimo dígito de factorial de 2, (	2) 2
Ultimo dígito de factorial de 3, (	6) 6
Ultimo dígito de factorial de 4, (	24) 4
Ultimo dígito de factorial de 5, (	120) 0
Ultimo dígito de factorial de 6, (	720) 0
Ultimo dígito de factorial de 7, (	5.040) 0
Ultimo dígito de factorial de 8, (	40.320) 0
Ultimo dígito de factorial de 9, (	362.880) 0
Ultimo dígito de factorial de 10, (	3.628.800) 0
Ultimo dígito de factorial de 11, (	39.916.800) 0
Ultimo dígito de factorial de 12, (	479.001.600) 0
Ultimo dígito de factorial de 13, (	6.227.020.800) 0
Ultimo dígito de factorial de 14, (	87.178.291.200) 0

**HACER ACTIVIDADES DE APLICACIÓN PAGINA 119 - 120**