

# UD6.- Métodos

Módulo: Programación  
1.º DAM



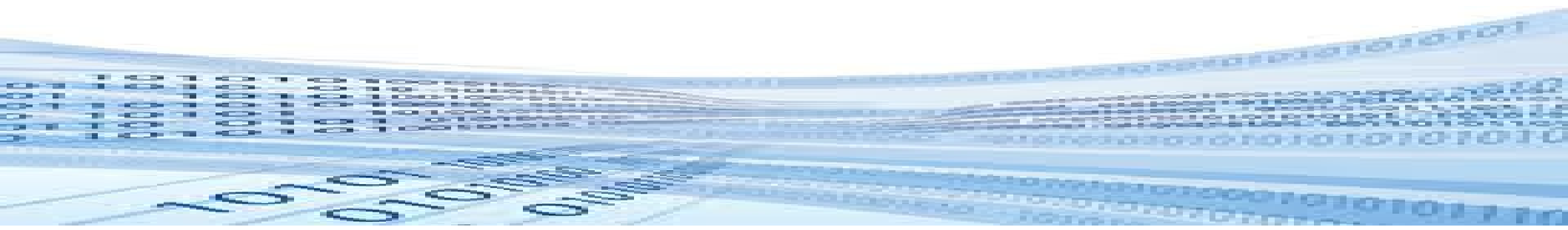
**Unión Europea**

Fondo Social Europeo

*El FSE invierte en tu futuro*

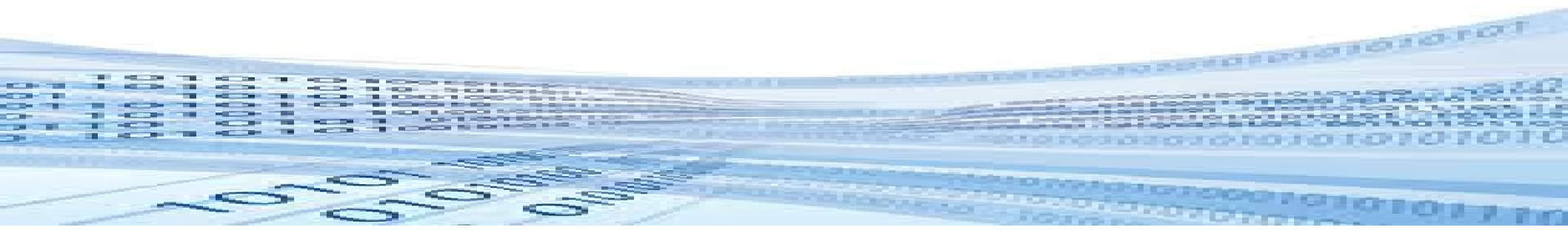
# CONTENIDOS

- ¿Qué es un método?
- ¿Cómo crear un método?
- ¿Como invocar a un método?
- Paso de argumentos
- Métodos sobrecargados
- Ámbito de definición de métodos



# ¿Qué es un método? (I)

- Conjunto de instrucciones agrupadas bajo un identificador.
- Puede ser invocado desde diferentes puntos de un programa.
- Opcionalmente puede devolver un valor. Tradicionalmente (programación estructurada):
  - Los métodos que devuelven un valor se denominan funciones
  - Los métodos que no devuelven ningún valor se denominan procedimientos

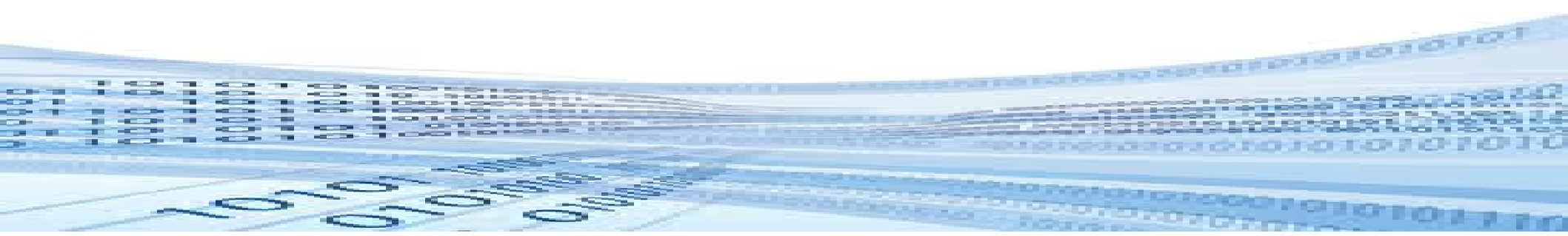


# ¿Qué es un método? (II)

- Hasta ahora hemos utilizado algunos métodos definidos en las librerías propias de Java. Por ejemplo:
  - `int i = entrada.nextInt();`
  - `double rx = Math.sqrt(78);`
  - `System.out.println("Hola a todos");`
- Podemos observar que:
  - Todos los métodos tienen un identificador: `sqrt`, `nextInt`, `println`
  - Después del identificador, y entre paréntesis, aparecen los parámetros (78, "Hola a todos"). Pueden no tener parámetros.
  - Algunos métodos devuelven un resultado (`nextInt`, `sqrt`), otros métodos no explícitamente (`println`).

# ¿Qué es un método? (III)

- El programador también puede definir sus métodos propios.
- Ventajas:
  - Ahorra esfuerzo y tiempo cuando en la resolución de un problema se repite con frecuencia una misma secuencia de acciones: reutilización de código.
  - Facilita la resolución de problemas grandes a través de la descomposición en problemas más sencillos.
  - Incrementa la legibilidad de los programas.

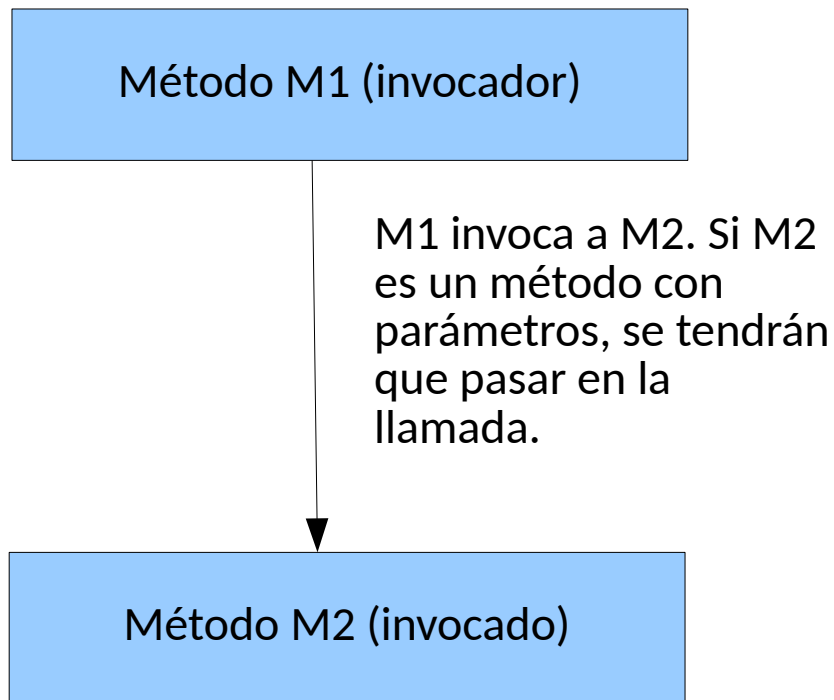


# ¿Qué es un método? (IV)

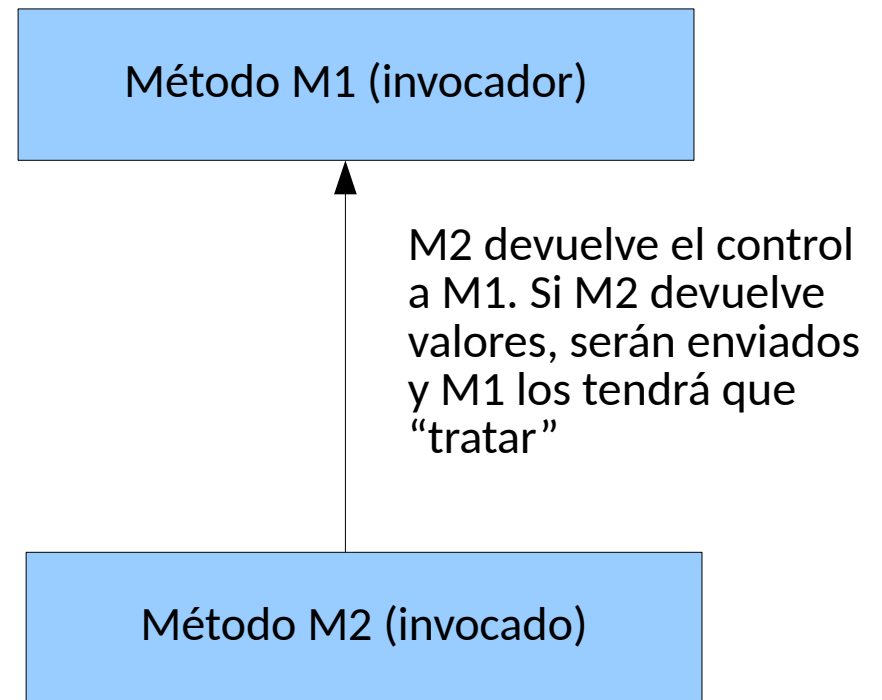
- Un método puede ser invocado (llamado) desde otro método:
  - Cuando un método(M1) llama a otro método(M2), el método invocador(M1) transfiere el control al método invocado(M2).
  - Cuando el método invocado(M2) finaliza la ejecución de su código, devuelve el control al método invocador(M1).
- En Java:
  - Un programa empieza a ejecutarse siempre por el método *main()*.
  - El método *main()* puede invocar a otros métodos que, a su vez, pueden invocar a otros.
- Un método también se puede invocarse así mismo: recursividad.

# ¿Qué es un método? (V)

## LLAMADA A UN MÉTODO



## RETORNO DE UN MÉTODO



# ¿Cómo crear un método? (I)

```
[static] tipo_devuelto | void nombre([tipo_par1 par1, tipo_par2 par2, ...]) {  
    //Instrucciones del método  
    //si devuelve algún valor, se tiene que incluir la instrucción return  
}
```

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



# ¿Cómo crear un método? (II)

```
[static] tipo_devuelto | void nombre([tipo_par1 par1, tipo_par2 par2, ...]) {  
    //Instrucciones del método  
    //si devuelve algún valor, se tiene que incluir la instrucción return  
}
```

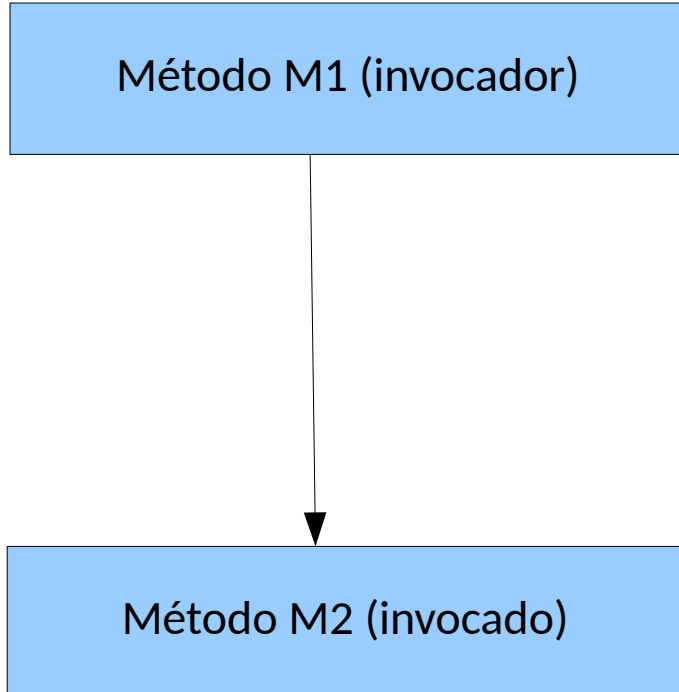
- **tipo\_devuelto**
  - tipo de datos que devuelve el método
  - si no devuelve nada, se tiene que indicar con void
- **parámetros** (argumentos)
  - son opcionales
  - si aparecen, se tiene que indicar, para cada uno de ellos, su tipo y su nombre

# ¿Cómo crear un método? (III)

- Instrucción `return`
  - especifica el valor que devuelve el método
  - devuelve el control inmediatamente al método invocador

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

# ¿Cómo invocar a un método? (I)



En M1, cuando se hace la llamada:

M2 (par1, par2, ..., parN)

Parámetros actuales: contienen los valores con los cuales M1 llama a M2.

Tienen que coincidir la cantidad de parámetros y el tipo con:

Parámetros formales: especificados en la cabecera de M2

M2 (tipo\_par1 par1, tipo\_par2 par2, ..., tipo\_parN parN)

# Invocar a un método. Ejemplo (I)

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

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

# Invocar a un método. Ejemplo(II)

```
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);
```

# Invocar a un método. Ejemplo (III)

Cabecera del método	Llamamiento al método
<pre>int suma (int a, int b)  /*a y b son parámetros formales*/</pre>	<pre>suma( 2 , 4 );  /*2 y 4 son parámetros actuales */</pre>
<pre>int suma (int a, int b)  /*parámetros formales*/</pre>	<pre>suma( num1, 3+num2 );  /*parámetros actuales */</pre>
<pre>void imprime (int a, float b, char c)  /*parámetros formales*/</pre>	<pre>imprime ( numero, 3.14 , 'x' );  /*parámetros actuales */</pre>

# Invocar a un método. Ejemplo(IV)

```
System.out.print("El resultado es"+ suma(a, 74);
```

```
if(suma(a,b) < 20 )  
...
```

```
imprime('a', 40);
```

# Paso de parámetros (I)

- En Java, toda la información que se le pasa a un método cuando se invoca se pasa por valor. Pero se tiene que distinguir si los parámetros son de tipos **primitivos** o de tipos **referencia**.
    - Cuando los argumentos son de tipos **primitivos** (tipos básicos), al método le llega un **valor** que se guardará dentro de una variable. Este valor es independiente de la variable que se envió en la llamada.
      - Los parámetros de tipos byte, short, int, long, float, double, boolean y char nunca se modifican en el método invocador, aunque sus copias varían en el método invocado.
    - Cuando los argumentos son de tipos **referencia** (array, objeto...), al método le llega una **dirección de memoria** (referencia). El método no podrá modificar la referencia, pero si los valores que hay en la dirección de memoria. \*\*\*
- \*\*\*: lo veremos con más detalle en la orientación a objetos.





# Paso de parámetros. Ejemplo

```
public class ValorApp {  
  
    public static void main(String[] args) {  
  
        int a=3;  
        System.out.println("antes de la llamada: a="+a);  
        funcion(a);  
        System.out.println("después de la llamada: a="+a);  
    }  
  
    public static void funcion(int x){  
        x=x+3;  
        System.out.println("dentro de la función: a="+x);  
    }  
}
```

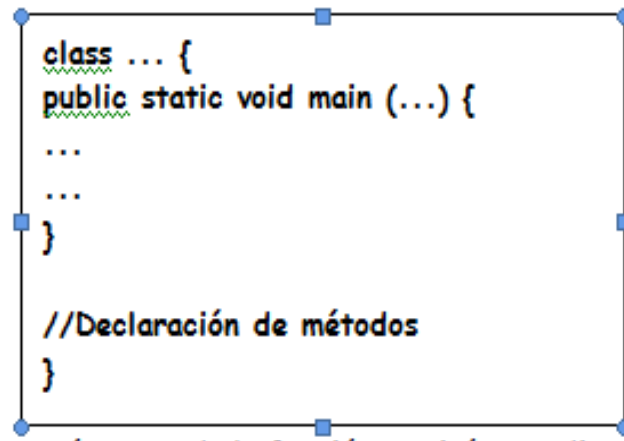
# Métodos sobrecargados

- Cuando dos métodos tienen el mismo nombre, pero son diferentes en la cantidad, orden o tipo de los parámetros, se dice que están sobrecargados
  - en inglés *overloaded*
- No hay suficiente diferencia con el tipo de valor devuelto o en las excepciones (errores a controlar) que se puedan lanzar.

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

# Ámbito de definición de métodos (I)

- Las variables y los parámetros formales que se definen dentro de un método son locales a él.
  - Únicamente son accesibles dentro del método.
- Una clase Java puede definir y emplear sus propios métodos:
  - No hay restricciones en el orden en el que se escriben los métodos.
  - El método *main()* puede estar antes o después de cualquier otro método.



The diagram shows a rectangular box representing a Java class. Inside the box, the following code is written:

```
class ... {  
  public static void main (...) {  
    ...  
    ...  
  }  
  
  //Declaración de métodos  
}
```

The box has small blue squares at its corners and midpoints, suggesting it is a diagrammatic representation of a code block or a window.

# Ámbito de definición de métodos (II)

- Una clase puede emplear métodos **static** (de momento) de otra clase.

```
public class Matematicas {  
    static long factorial(int num) {  
        long resultado=1;  
        while(num>0) {  
            resultado*=num;  
            num--;  
        }  
        return resultado;  
    }  
    static boolean esPrimo(int numero) {  
        if((numero!=2) && (numero%2==0))  
            for(int i=3; i<numero/2; i+=2) {  
                if(numero%i==0) {  
                    return false;  
                }  
            }  
        return true;  
    }  
}
```

```
public class MatesApp {  
    public static void main(String[] args) {  
        //Número combinatorio de m sobre n  
        int m=8, n=3;  
  
        long numerador=Matematicas.factorial(m);  
        long denominador=Matematicas.factorial(m-n);  
  
        System.out.println(" vale "+numerador + " / " + denominador);  
        System.out.println("*****");  
  
        //números primos comprendidos entre 100 y 200  
        System.out.println("Números primos comprendidos entre 100 y 200");  
        for(int num=100; num<200; num++) {  
            if(Matematicas.esPrimo(num)) {  
                System.out.print(num+" - ");  
            }  
        }  
    }  
}
```

return false;

# Recapitulando (I)

- Nuestros programas van a consistir, a partir de ahora, en una serie de métodos, donde cada uno de ellos va a aportar una pequeña parte de la solución que queremos implementar.

## **¿En qué circunstancias debemos definir métodos?**

- Cuando detectemos que tenemos código repetido en varias partes de nuestro programa.
- Cuando una porción de código sea susceptible de ser reutilizada.
- Los métodos deberían realizar tareas simples. Cuando identifiquemos métodos que realizan “demasiadas tareas” deberemos dividirlos en varios métodos.

# Recapitulando (II)

## ¿Cómo definir los métodos?

- Al definir un método siempre debemos identificar:
  - ¿qué tarea necesitamos que realice el método?
  - ¿qué datos de entrada necesita?
  - ¿qué resultado/s producirá?
- Una vez identificados los requisitos anteriores, puede que se nos ocurran diferentes variantes de métodos.
  - La norma es: elegir siempre aquel método que sea susceptible de poder ser utilizado en más situaciones, siempre y cuando, el hecho de hacerlo más reutilizable no suponga tener que hacer mucho más código y complicar demasiado el método.