

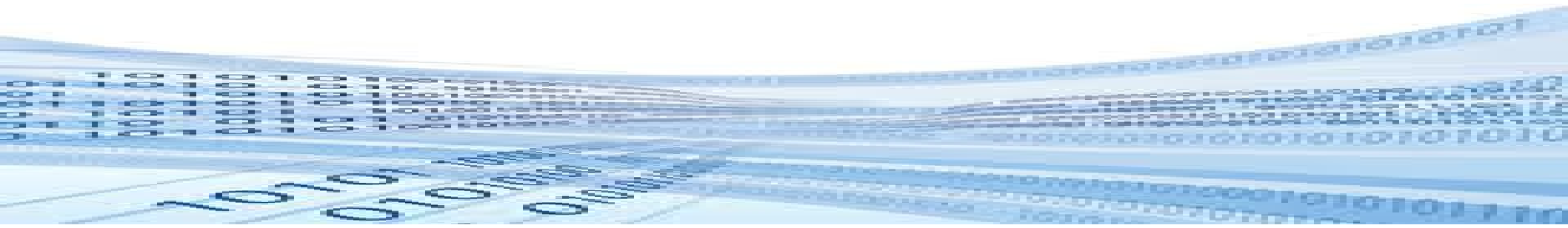
UD6.- Mètodes

Mòdul: Programació
1r DAM
Curs 2018-2019



CONTINGUTS

- Què és un mètode?
- Com crear un mètode?
- Com cridar a un mètode?
- Pas d'arguments
- Mètodes sobrecarregats
- Àmbit de definició de mètodes



Què és un mètode?(I)

- Conjunt d'instrucciones agrupades baix un identificador.
- Pot ser cridat des de diferents punts d'un programa.
- Opcionalment pot retornar un valor. Tradicionalment (programació estructurada):
 - Els mètodes que retornen un valor s'anomenen funcions
 - Els mètodes que no retornen cap valor s'anomenen procediments

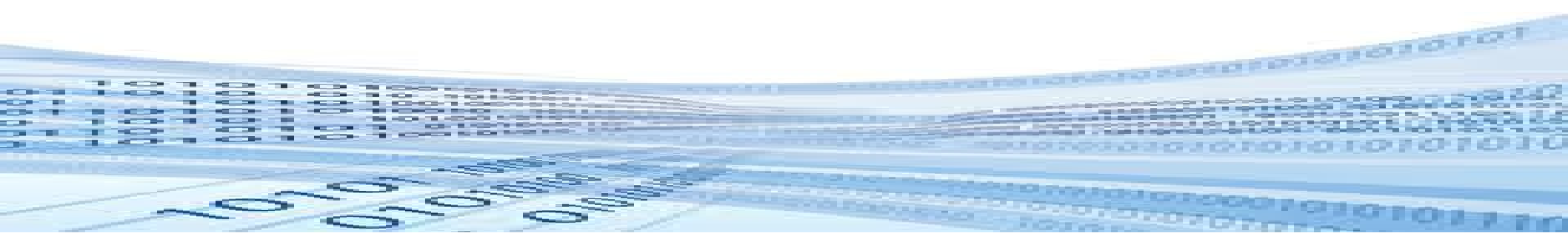


Què és un mètode?(II)

- Fins ara hem utilitzat alguns mètodes definits en les llibreries pròpies de Java. Per exemple:
 - `int i=entrada.nextInt();`
 - `double rx=Math.sqrt(78);`
 - `System.out.println("Hola a tots");`
- Podem observar que:
 - Tots els mètodes tenen un identificador: `sqrt`, `nextInt`, `println`
 - Després de l'identificador, i entre parèntesi, apareixen els paràmetres (78, "Hola a tots"). Poden no tindre paràmetres.
 - Alguns mètodes retornen un resultat (`nextInt`, `sqrt`), altres mètodes no explícitament (`println`).

Què és un mètode?(III)

- El programador també pot definir els seus mètodes propis.
- Avantatges:
 - Estalvia esforç i temps quan en la resolució d'un problema es repetix amb freqüència una mateixa seqüència d'accions: reutilització de codi.
 - Facilita la resolució de problemes grans a través de la descomposició en problemes més senzills.
 - Incrementa la llegibilitat dels programes.

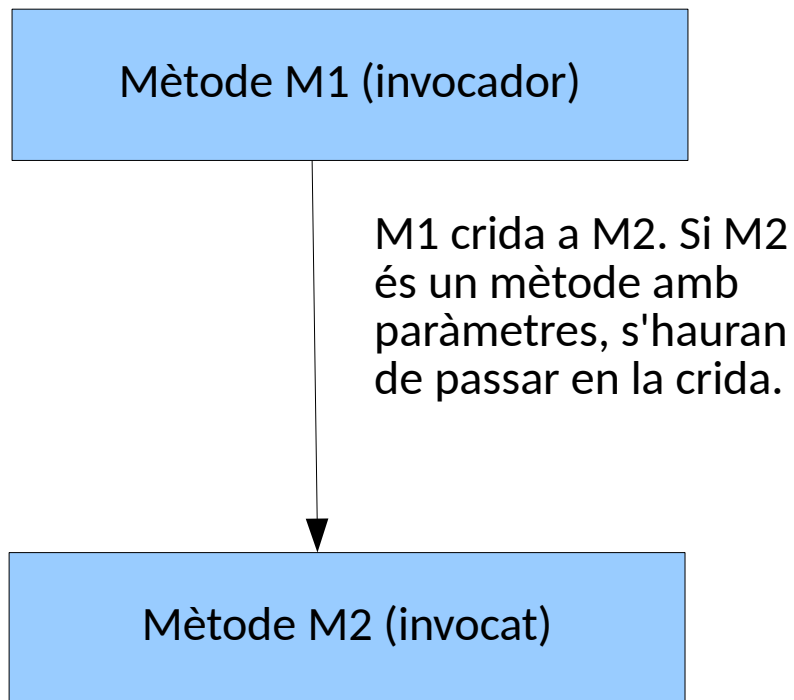


Què és un mètode?(IV)

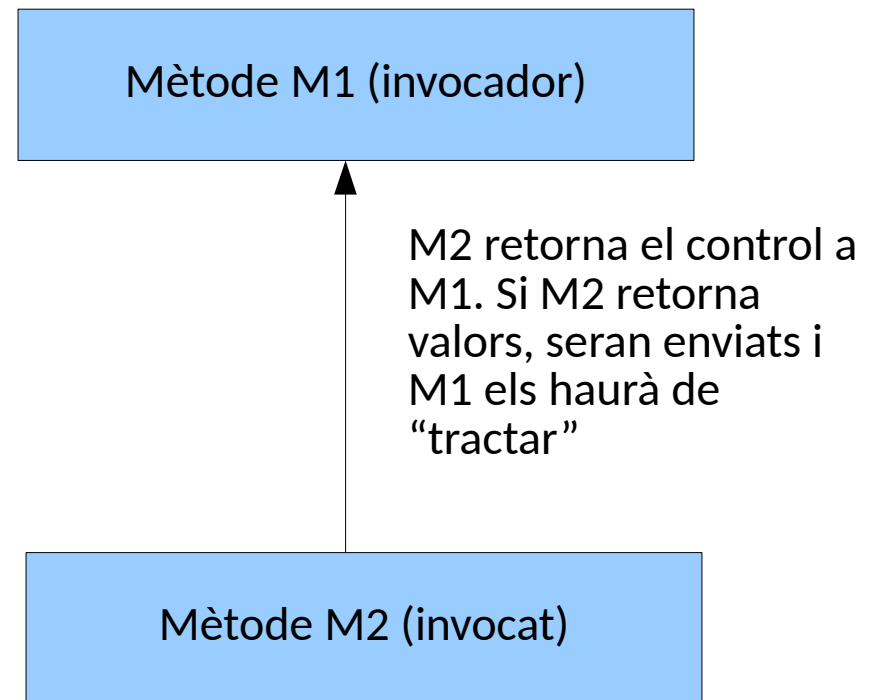
- Un mètode pot ser invocat (cradat) des d'un altre mètode:
 - Quan un mètode(M1) crida a un altre mètode(M2), el mètode invocador(M1) transferix el control al mètode invocat(M2).
 - Quan el mètode invocat(M2) finalitza l'execució del seu codi, retorna el control al mètode invocador(M1).
- En Java:
 - Un programa comença a executar-se sempre pel mètode *main()*.
 - El mètode *main()* pot invocar a altres mètodes que, a la vegada, en poden invocar a altres.
- Un mètode també es pot invocar a ell mateix: recursivitat.

Què és un mètode?(V)

CRIDA A UN MÈTODE



RETORN D'UN MÈTODE



Com crear un mètode?(I)

```
[static] tipus_retornat | void nom([tipus_par1 par1, tipus_par2 par2, ...]) {  
    //Instruccions del mètode  
    //si retorna algun valor, s'ha d'incloure la instrucció return  
}
```

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


Com crear un mètode?(II)

```
[static] tipus_retornat | void nom([tipus_par1 par1, tipus_par2 par2, ...]) {  
    //Instruccions del mètode  
    //si retorna algun valor, s'ha d'incloure la instrucció return  
}
```

- **tipus_retornat**
 - tipus de dades que retorna el mètode
 - si no retorna res, s'ha d'indicar amb void
- **paràmetres** (arguments)
 - són opcionals
 - si apareixen, s'ha d'indicar, per a cadascun d'ells, el seu tipus i el seu nom

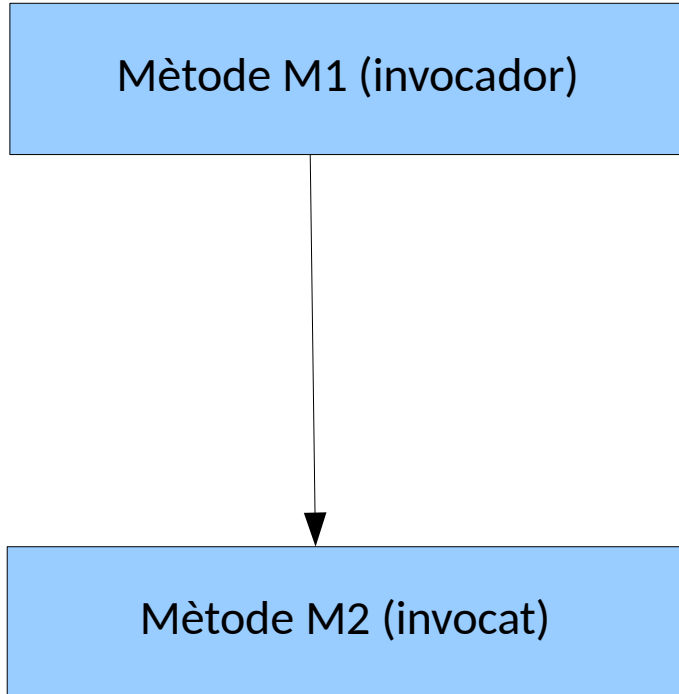
Com crear un mètode?(III)

- Instrucció `return`

- especifica el valor que retorna el mètode
- retorna el control immediatament al mètode invocador

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

Com cridar a un mètode?(I)



En M1, quan es fa la crida:

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

Paràmetres actuals: contenen els valors amb els quals M1 crida a M2.

Ha de coincidir la quantitat de paràmetres i el tipus amb:

Paràmetres formals: especificats en la capçalera d'M2

M2 (tipus_par1 par1, tipus_par2 par2, ..., tipus_parN parN)

Cridar un mètode. Exemple(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) ;  
}
```

Cridar un mètode. Exemple(II)

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

```
int a,b,c,d;  
int max1,max2,max;
```

//Assignació de valors a les variables

```
max1=maxim(a,b);  
max2=maxim(c,d);  
max=maxim(max1,max2);
```

Cridar un mètode. Exemple(III)

Capçalera del mètode	Crida al mètode
<pre>int suma (int a, int b) /* a i b són paràmetres formals */</pre>	<pre>suma(2 , 4); /* 2 i 4 són paràmetres actuals */</pre>
<pre>int suma (int a, int b) /* paràmetres formals */</pre>	<pre>suma(num1, 3+num2); /* paràmetres actuals */</pre>
<pre>void imprime (int a, float b, char c) /* paràmetres formals */</pre>	<pre>imprime (numero, 3.14 , 'x'); /* paràmetres actuals */</pre>

Cridar un mètode. Exemple(IV)

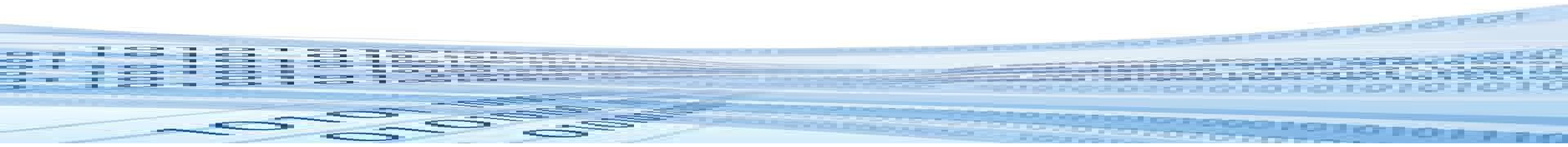
```
System.out.print("El resultado es"+ suma(a, 74);
```

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

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

Pas d'arguments(I)

- En Java, tota la informació que se li passa a un mètode quan es crida es passa per valor. Però s'ha de distingir si els paràmetres són de tipus **primitius** o de tipus **referència**.
 - Quan els arguments són de tipus **primitius** (tipus bàsics), al mètode li arriba un **valor** que es guardarà dins d'una variable. Este valor és independent de la variable que es va enviar en la crida.
 - Els paràmetres de tipus byte, short, int, long, float, double, boolean i char mai es modifiquen en el mètode invocador, encara que les seues còpies varien en el mètode invocat.
 - Quan els arguments són de tipus **referència** (array, objecte...), al mètode li arriba una **adreça de memòria** (referència). El mètode no podrà modificar la referència, però si els valors que n'hi ha en l'adreça de memòria. ***
- ***: ho vorem amb més detall en l'orientació a objectes.



Pas d'arguments. Exemple

```
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);  
    }  
}
```

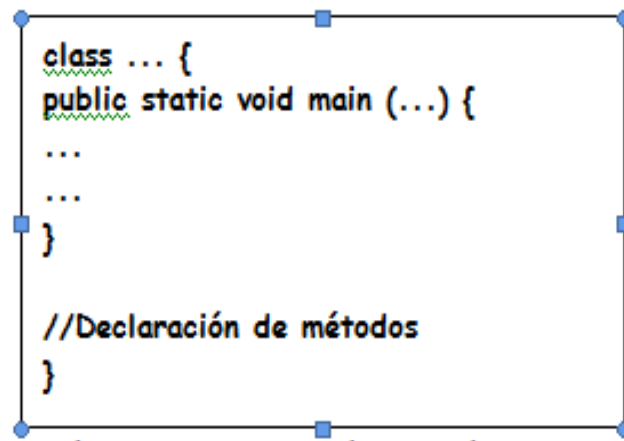
Mètodes sobrecarregats

- Quan dos mètodes tenen el mateix nom, però són diferents en la quantitat, ordre o tipus dels paràmetres, es diu que estan sobrecarregats
 - en anglès *overloaded*
- No n'hi ha prou diferència amb el tipus de valor retornat o en les excepcions (errors a controlar) que es puguin llançar.

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

Àmbit de definició de mètodes(I)

- Les variables i els paràmetres formals que es definixen dins d'un mètode són locals a ell.
 - Únicament són accessibles dins del mètode.
- Una classe Java pot definir i emprar els seus propis mètodes:
 - No n'hi ha restriccions en l'ordre en el qual s'escriuen els mètodes.
 - El mètode *main()* pot estar abans o després de qualsevol altre mètode.



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

The diagram shows a rectangular box representing a Java class. Inside the box, the code is as follows: `class ... {`, `public static void main (...) {`, `...`, `...`, `}`, `//Declaración de métodos`, and `}`. The box has blue square handles at the corners and midpoints of the edges.

Àmbit de definició de mètodes(II)

- Una classe pot emprar mètodes **static** (de moment) d'una altra classe.

```
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) )  
            return false;  
        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+" - ");  
            }  
        }  
    }  
}
```