

0101010
0100101
1101010

UD4.1- DEFINICIÓ I REFERÈNCIA A MÈTODES

0485 - Programació

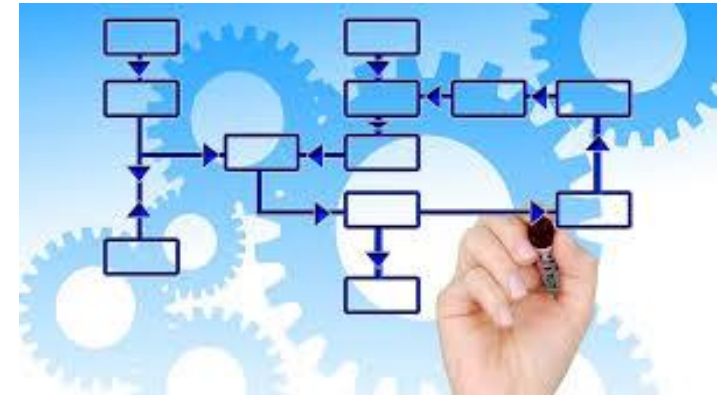
1er DAW / DAM

0. CONTINGUTS

- **QUÈ ÉS UN MÈTODE?**
- **DEFINICIÓ DE MÈTODES**
- **REFERENCIAR MÈTODES**
- **BONES PRÀCTIQUES**
- **ACTIVITATS PRÈVIES**

1. QUÈ ÉS UN MÈTODE?

- Moltes vegades el **mateix bloc de sentències** o **segment de codi** ha de ser executat múltiples vegades per solucionar un **sub-problema** amb **diferents dades**.
 - Calcular una **arrel quadrada**
 - **Demanar 1 dada** a un **usuari**
 - **Dibuixar el tauler** d'un **joc**



Repetir codi conduirà a **errors**, un **baix rendiment** i resultarà en un **codi no mantenible**.

1. QUÈ ÉS UN MÈTODE?

- Els mètodes permeten agrupar un **conjunt d'instruccions sota un identificador** comú que ens permetrà executar-les cada vegada que ho necessitem.

Identificador

```
public static void demanarEnter() {  
    int numero;  
    do {  
        System.out.println ("Introdueix un número entre [1-10]").  
        número = teclat.nextInt ();  
    } while (numero < 0 || numero>10);  
}
```

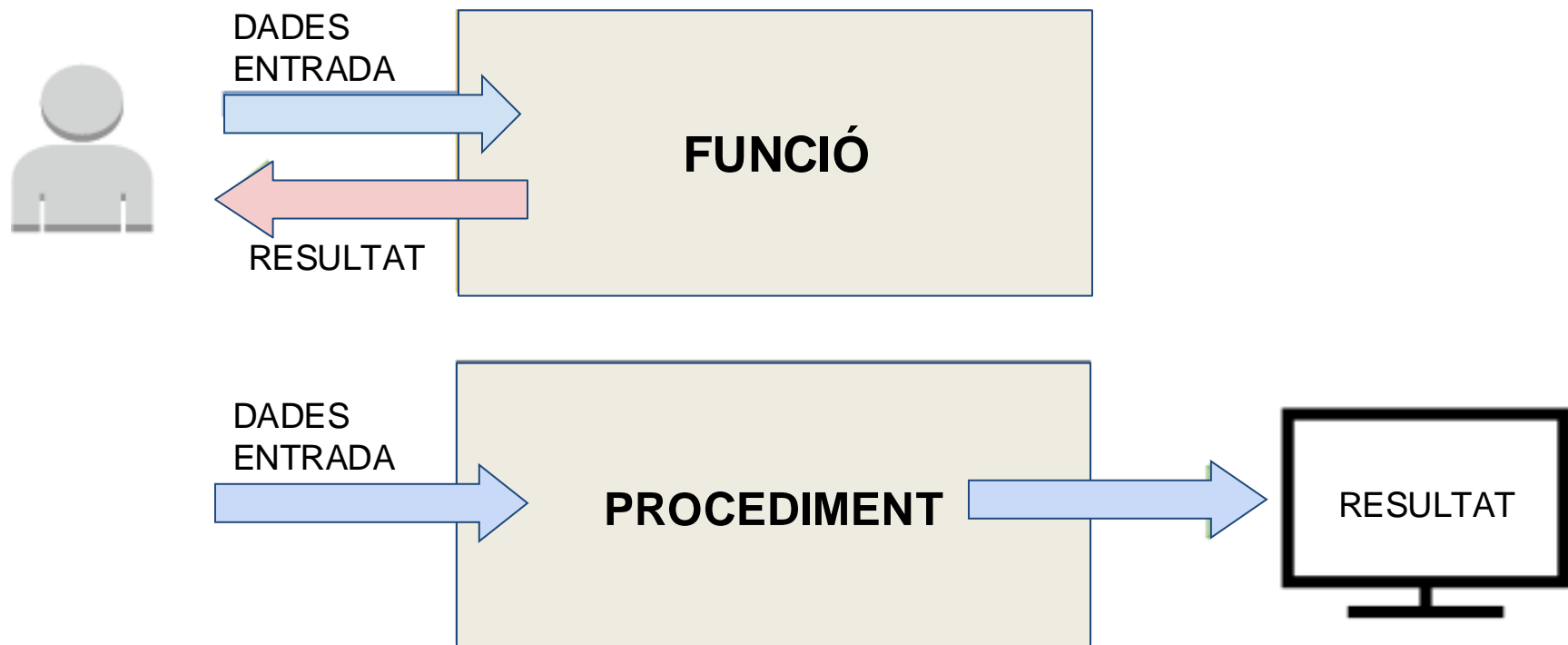
INSTRUCCIONS
a
EXECUTAR

1. QUÈ ÉS UN MÈTODE?

- Tradicionalment, en **programació estructurada** podem distingir entre:
 - **Subprogrames que no tornen cap valor (procediments):**
 - *Exemple: Subprograma encarregat de visualitzar dades per pantalla.*
 - **Subprogrames que retornen algun valor (funcions):**
 - *Exemple: Subprograma encarregat de calcular l'arrel quadrada.*

1. QUÈ ÉS UN MÈTODE?

Funcions i procediments



A *POO* no fem distinció i els anomenem mètodes

1. QUÈ ÉS UN MÈTODE?

- Fins ara hem utilitzat alguns **mètodes** definits en les **llibreries pròpies** de Java (`teclat.nextInt()`, `Math.random()`, etc)

```
int number = teclat.nextInt();  
  
double resultat = Math.pow(78, 2);  
  
System.out.println("Hola a tots");
```

Tots els llenguatges de programació proporcionen un conjunt de funcions comunes a les que anomenem biblioteca estàndard

1. QUÈ ÉS UN MÈTODE?

- Si analitzem els mètodes utilitzats podem observar que:
 - Tots els **mètodes** tenen un **identificador**: `nextInt`, `println`, `pow`.
 - **Després** de l'identificador, i **entre parèntesi**, apareixen els **paràmetres** `(78, 2)` `("Hola a tots")`. proporcionen un **mitjà de comunicació** entre el programa que el crida i el **subprograma** o **mètode**. *(Poden no tenir paràmetres)*.

```
int number = teclat.nextInt();
```

Mètode **sense** paràmetres

```
double resultat = Math.pow(78, 2);
```

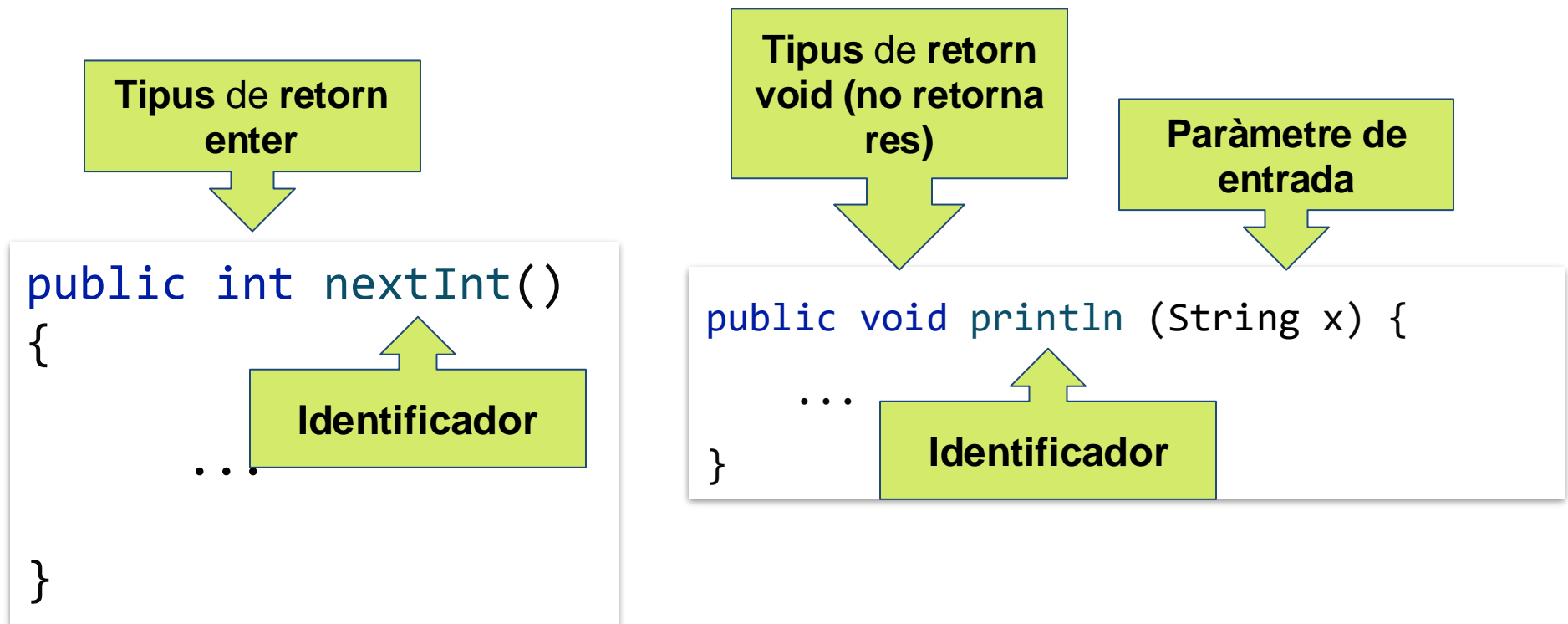
Mètode **amb 2** paràmetres
enters

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

Mètode **amb 1** paràmetre
de tipus cadena

1. QUÈ ÉS UN MÈTODE?

- Alguns mètodes retornen un **resultat** (`next`, `sqrt`), altres no (`println`).



1.1 MÈTODES PROPIS

- Quan aquests **no són suficients**, el programador **pot definir els seus propis mètodes**.
- **Avantatges:**
 - Estalvien esforç i temps quan en la resolució d'un problema es repeteix 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** del **codi font**.

1. QUÈ ÉS UN MÈTODE?

nextInt

```
public int nextInt()
```

Scans the next token of the input as an int.

An invocation of this method of the form `nextInt()` behaves in exactly the same way as the invocation `nextInt(radix)`, where `radix` is the default radix of this scanner.

Returns:

the int scanned from the input

Throws:

`InputMismatchException` - if the next token does not match the *Integer* regular expression, or is out of range

`NoSuchElementException`

`IllegalStateException`

```
Scanner teclat = new Scanner(System.in);
int num = teclat.nextInt();
```

sqrt

```
public static double sqrt(double a)
```

Returns the correctly rounded positive square root of a double value. Special cases:

- If the argument is NaN or less than zero, then the result is NaN.
- If the argument is positive infinity, then the result is positive infinity.
- If the argument is positive zero or negative zero, then the result is the same as the argument.

Otherwise, the result is the double value closest to the true mathematical square root of the argument value.

Parameters:

a - a value.

Returns:

the positive square root of a. If the argument is NaN or less than zero, the result is NaN.

Math.sqrt(25)

NaN significa
Not A Number

1.1 MÈTODES PROPIS

```
private int nextLevel()
{
    level = new Level();
    currentLevel++;
    try
    {
        level.load( fileName: Level.LEVELS_FOLDER + "level" + currentLevel + ".txt", character);
    }
}
```

```
private void choosePlayer()
{
    character = new Warrior();

    switch (PlayerSelectScene.chosenPlayer)
    {
        case 1:
            character = new Valkyrie();
            break;
        case 2:
            character = new Sorcerer();
            break;
        case 3:
            character = new Dwarf();
            break;
    }
}
```



1.2 FLUX DE CONTROL

- En Java:
 - Un programa comença a executar-se sempre pel **mètode *main()***.
 - El **mètode *main()*** pot invocar **altres mètodes** què, al seu torn, poden invocar altres.

Aquest mètode es
crïat per la JVM
al executar el
programa

Crïat pel mètode
main()

```
public class Exemple1 {  
  
    public static void main(String[] args) {  
        1. System.out.println ("--Cuadrado--");  
        2. dibujarCuadrado();  
        4. System.out.println ("--Cuadrado--");  
    }  
  
    public static void dibujarCuadrado() {  
        3. codi a executar  
    }  
  
}
```

1.2 FLUX DE CONTROL

INVOCACIÓ A UN MÈTODE

Mètode **M1** (Invocador)

1. **M1** crida a **M2**

Mètode **M2** (Invocat)

2. **M2** Executa les instruccions del seu subprograma

RETORN D'UN MÈTODE

Mètode **M1** (invocador)

3. **M2** Retorna el control a **M1**.
(Si 1 retorna resultat vindrà amb les resposta)

1.3 ACTIVITAT PRÈVIA

- **Activitat 1.** Què mostrarà per pantalla l'execució del següent codi?

```
public class Exemple {  
  
    public static void main(String[] args) {  
  
        System.out.println ("-- Forma 1 --");  
        dibujarForma();  
        System.out.println ("-- Forma 2 --");  
        dibujarForma();  
        System.out.println ("-- Forma 3 --");  
  
    }  
  
    public static void dibujarForma() {  
        for (int i = 0; i <4; i ++){  
            for (int j = 0; j <4 ; j ++){  
                System.out.print("#");  
            }  
            System.out.println();  
        }  
    }  
}
```

2. DEFINICIÓ DE MÈTODES

```
[public] [static] tipo_retorno identificador([tipo_par1 nombre_par1,  
tipo_par2 nombre_par2, ...])  
{  
    // Instruccions d'el mètode  
    // si retorna valor, s'ha d'incloure la instrucció return  
    // sino la instrucció return es opcional  
}
```

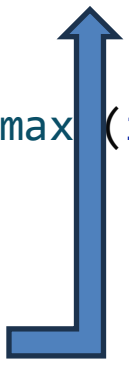
```
public class Exemple2 {  
  
    public static void greet() {  
        System.out.println ("Hola");  
    }  
  
    public static int suma (int a, int b) {  
        return a + b;  
    }  
}
```

Opcionalment, es podria
realitzar la instrucció
return a continuació

2.1 LA INSTRUCCIÓ `return`

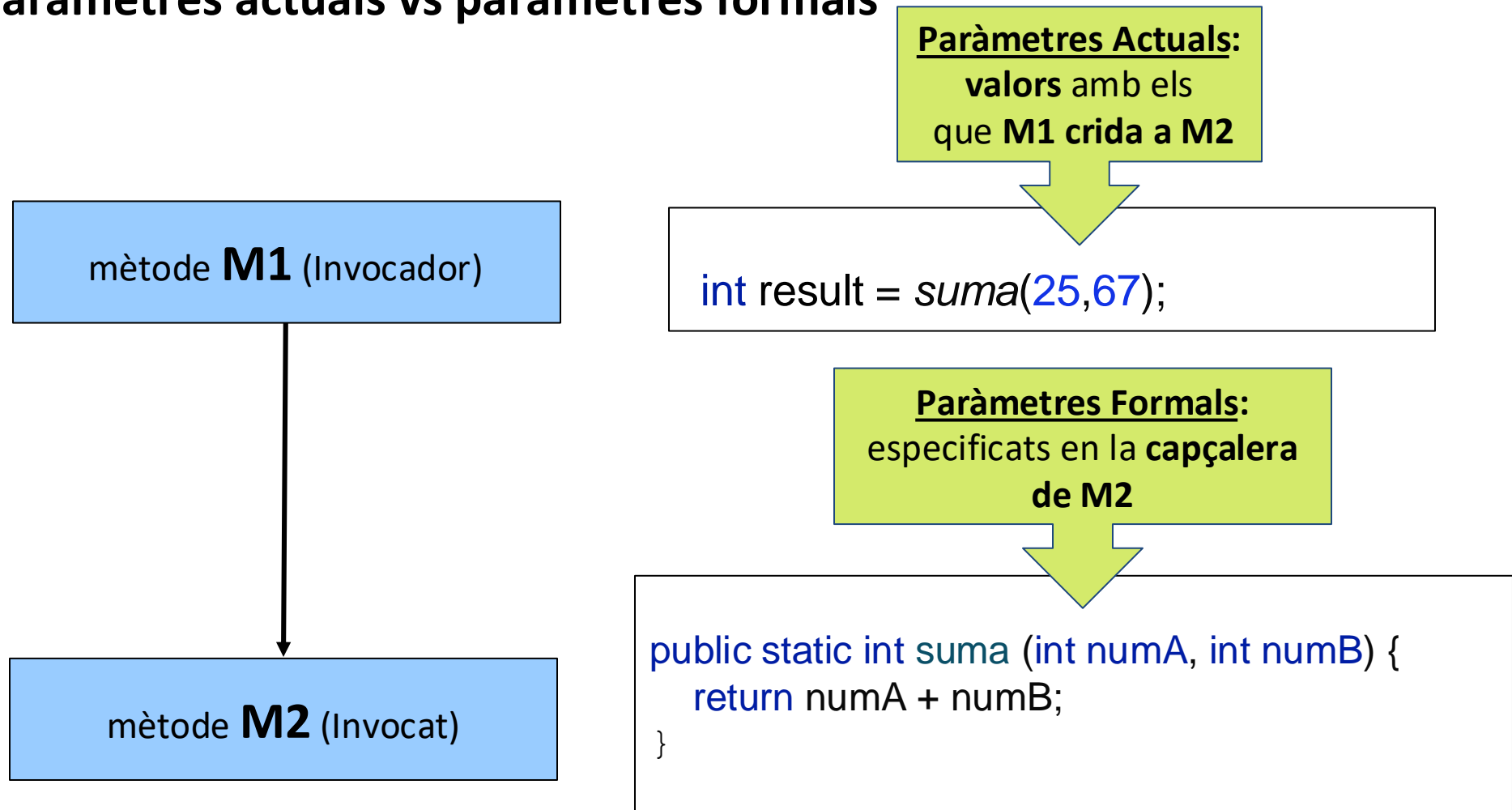
- Especifica el **valor** que **retorna** el mètode
- **Finalitza l'execució i retorna el control inmediatament al mètode invocador, finalitzant l'execució d'aquest**

```
public class Exemple3 {  
  
    public static void main(String[] args) {  
        int resultat = max(2,3);  
    }  
  
    public static int max(int x, int i) {  
        if (x >= i) {  
            return x;  
        } else {  
            return i;  
        }  
    }  
}
```



3. Referenciar MÈTODES

Paràmetres actuals vs paràmetres formals



3. Referenciar MÈTODES. EXEMPLE

Capçalera / signatura de mètodes	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) /* a i b paràmetres formals */</pre>	<pre>int num1 = 4; int num2 = 5; suma(num1, 3 + num2); /* Paràmetres actuals */</pre>
<pre>void imprimeix (int a, float b, char c) /* a b, i c paràmetres formals */</pre>	<pre>int numero = 5; imprimeix(numero, 3.14f, 'X'); /* Paràmetres actuals */</pre>

3. Referenciar MÈTODES. EXEMPLE

Activitat 2. Què mostrarà per pantalla l'execució del següent codi? Indica els **paràmetres formals** definits i els **paràmetres actuals** que s'utilitzen per a cadascuna de les cridades.

```
public class Exemple4 {  
    public static void main(String[] Args) {  
        int a = 20;  
        int b = 30;  
        int c = 40;  
        int d = 5;  
        int max1 = max(a, b);  
        int max2 = max(c, d);  
        int max = max(max1, max2);  
        System.out.println (max);  
    }  
    public static int max (int x, int i) {  
        return (x >= i)? x: i;  
    }  
}
```

3. Referenciar MÈTODES. EXEMPLE

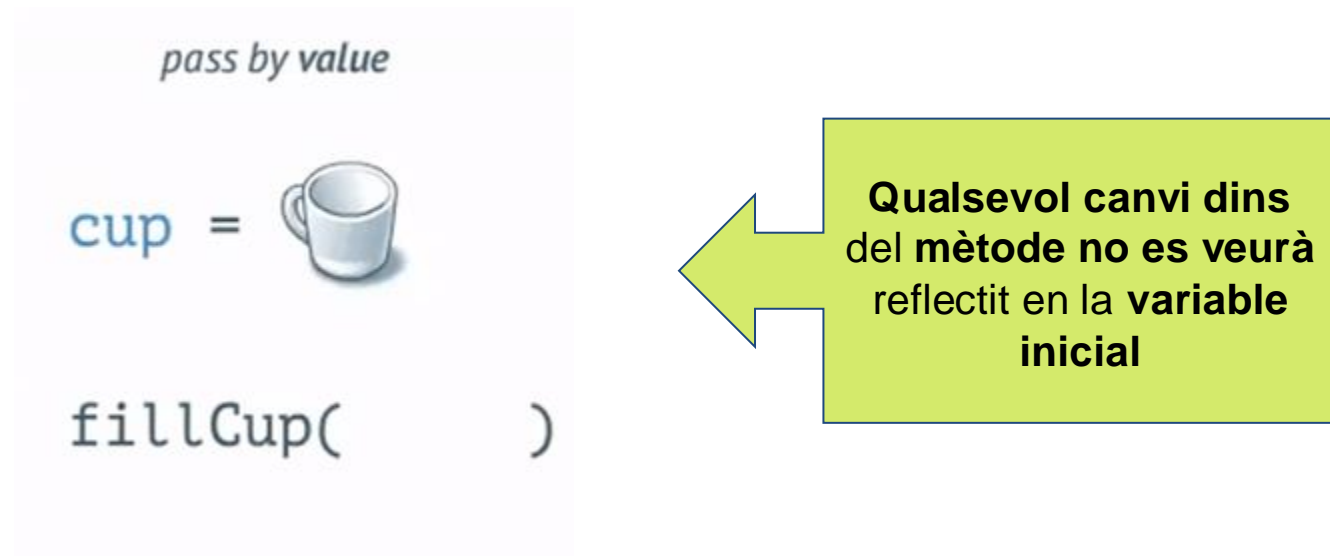
- Podem referenciar un mètode en qualsevol part del programa **on siga visible** (àmbit)
 - Ja siga dins d'una **expressió** o en una **sentència** d'assignació

```
int resultat = suma(a, 74);  
System.out.print ("El resultat és" + resultat );
```

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

4. PAS D'ARGUMENTS PER VALOR

- Quan invoquem a un mètode amb arguments de tipus primitiu, al mètode li arriba **una còpia del valor** que conté la variable.
 - Aquest valor és **independent de la variable** enviada.



Això passa amb els tipus primitius (byte, short, int, long, float, double, boolean i char)

4. PAS D'ARGUMENTS PER VALOR

```
public class Exemple5 {  
    public static void main(String[] args) {  
        int x = 1;  
        System.out.println ("Abans de la cridada x és: " + x);  
        incrementar(x);  
        System.out.println ("Després de la cridada x és: " + x);  
    }  
  
    public static void incrementar(int x) {  
        x++;  
        System.out.println ("X dins del mètode és " + x);  
    }  
}
```

X val 1

X segueix
valent 1

El paràmetre no té
perquè anomenar-se
igual que el argument

X val 2

5. BONES PRÀCTIQUES DEFINICIÓ DE MÈTODES

- Els mètodes han de:
 - Tenir una **grandària reduïda**.
 - Fer **una sola tasca i fer-la bé**
 - Tindre **noms descriptius** (Verb + substantiu)
 - `començarJoc`
 - `calcularFactorial`
 - Tindre un **nombre reduït d'arguments (<4)**

6. ACTIVITATS PRÈVIES

Activitat 3.- Crea un mètode **mostrarTaulaMultiplicar** sense arguments que al referenciar-lo mostre **totes les taules de multiplicar**. El tipus de retorn serà **void**.

Activitat 4.- Escriu un mètode **sense arguments** que retorne un enter que correspondrà al major de tres nombres enters. *Els nombres sencers els demanarem a l'usuari en el propi mètode.*

Activitat 5.- Crea un mètode **mostrarTaulaMultiplicar** amb 1 argument de tipus sencer anomenat **multiplicant**. Al cridar aquest mètode s'haurà de mostrar la taula de multiplicar corresponent. El tipus de retorn serà **void**.

Activitat 6.- Refactoriza l'**exercici 4** perquè els números sencers es demanen en la funció **main** i es passen com a argument a la funció creada.

6. ACTIVITATS PRÈVIES

Activitat 7.- Escriu un mètode `obtenirMultiplicacio` que calcule el resultat de multiplicar **3 nombres reals** passats com a argument. Referència-ho per als següents valors:

num 1	num 2	num 3
4.0	2.3	1.0
0	3.5	4.2
1	2.8	3.6

[Activitat 8.- Referència a mètodes](#)

[Activitats 9-12.- Implementació mètodes](#)

7. SOBRECÀRREGA DE MÈTODES

- La **sobrecàrrega de mètodes** (Overload) ens permet tenir **diferents mètodes** amb el **mateix nom**, Però amb **diferències** en la **quantitat, ordre o el tipus** dels paràmetres

"NameMangling" → *className :: suma :: int :: int*

ATENCIÓ: no és
suficient canviar
el tipus de retorn

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

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

Math.

- m abs (int a)
- m abs (long a)
- m abs (float a)
- m abs (double a)

7.1 ACTIVITAT PRÈVIA

- **Activitat 13.** Donat la definició d'aquests mètodes

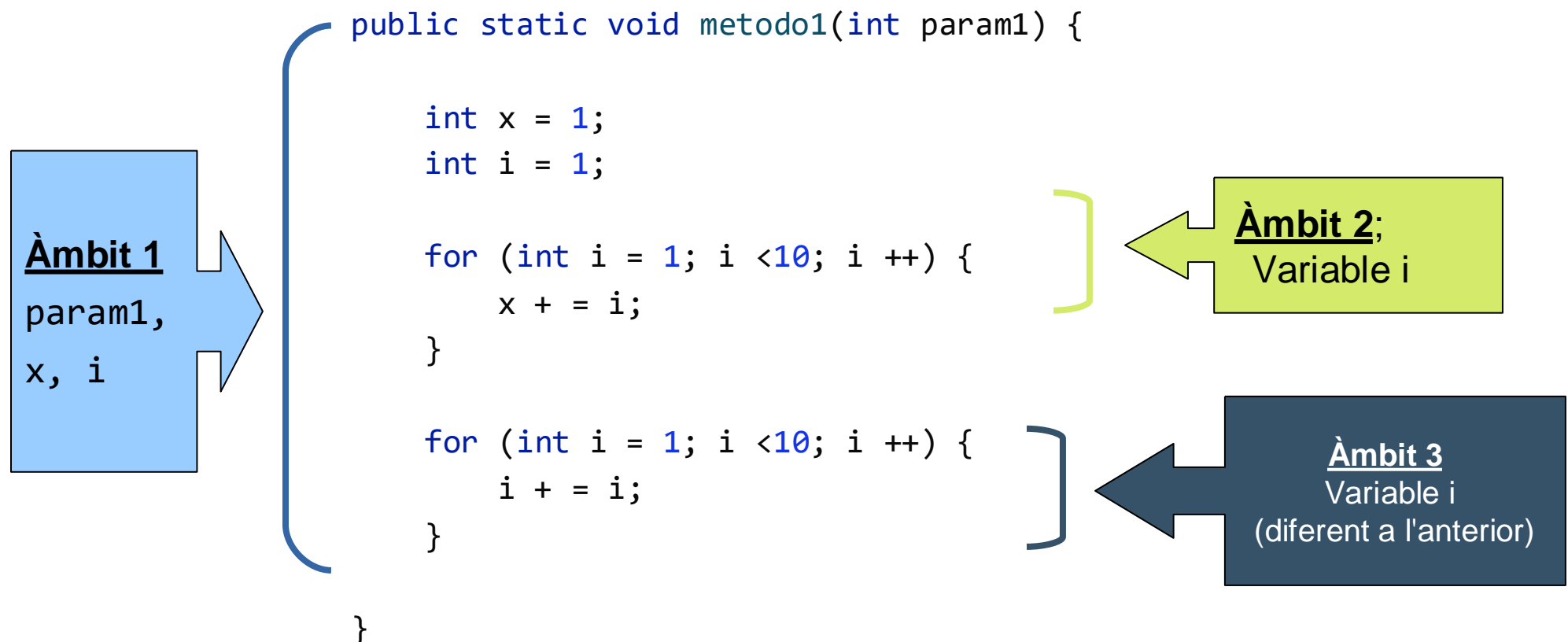
```
public class Exemple6 {  
    public static double metode(double x, double i)  
    public static double metode(int x, double i)  
}
```

Indica a quin d'ells corresponen les següents cridades realitzades des del mètode `main()`:

```
double z = metode (4, 5);  
double z = metode (4, 5.4);  
double z = metode (4.5, 5.4);
```

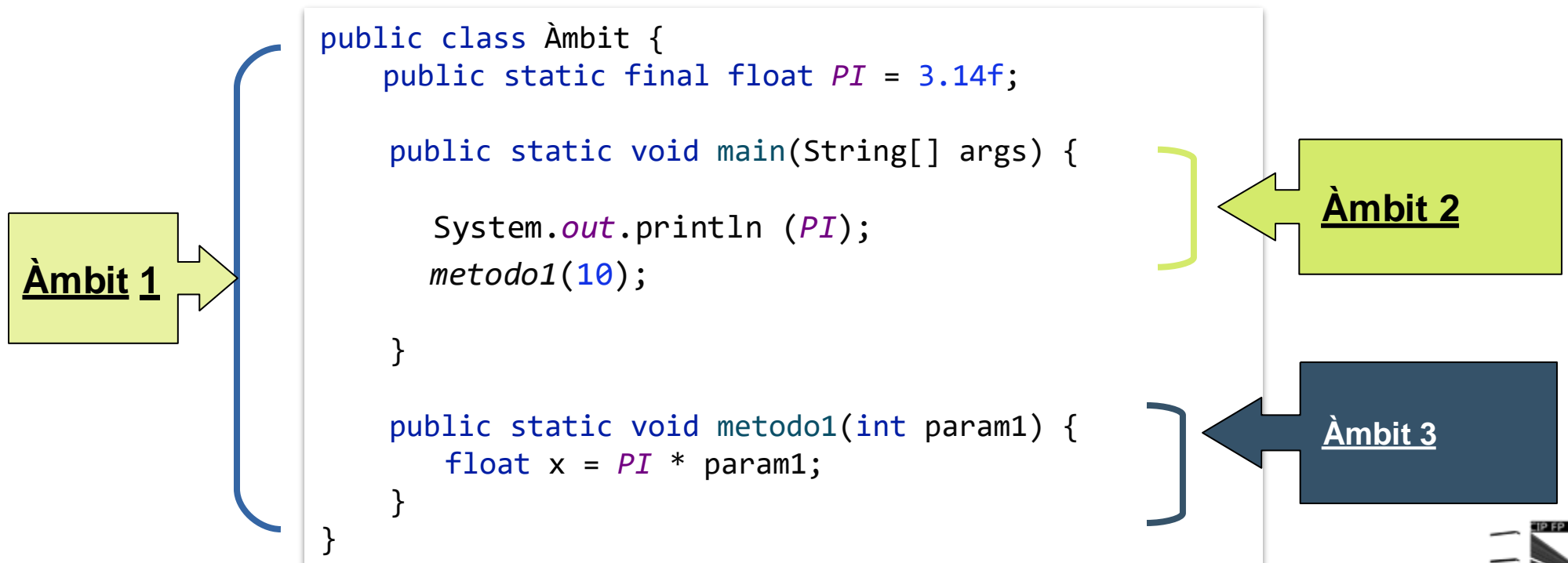
8. ÀMBIT DE VARIABLES I MÈTODES

- Les variables i els paràmetres formals d'un mètode **SÓN LOCALS** a ell, únicament **són accessibles i existeixen dins** del mètode.



8. ÀMBIT DE VARIABLES I MÈTODES

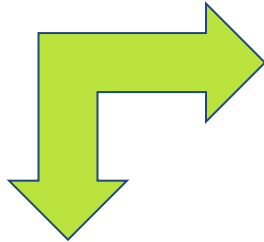
- Una **classe Java** defineix un àmbit, en ell es poden **definir i utilitzar** qualsevol dels seus mètodes.
- **No** hi ha **restriccions a l'ordre** en què s'escriuen.
- El mètode ***main()*** pot estar **abans** o **després** de qualsevol altre mètode. (El ficarem sempre abans)



8. ÀMBIT DE VARIABLES I MÈTODES

- Una classe pot emprar mètodes públiques **static** d'una altra classe.

Per fer-ho hem
d'anteposar el
nom de la classe
al mètode



```
public class Matematica {  
    public static int suma(int a, int b) {  
        return a + b;  
    }  
    public static int suma(int a, int b, int c) {  
        return a + b + c;  
    }  
}
```

```
public class Activitat14 {  
    public static void main(String[] args) {  
        int result1 = Matematica.suma(12,15);  
        int result2 = Matematica.suma(45,50,34);  
    }  
}
```

9. ACTIVITATS PRÈVIES

- **Activitat 14.** Implementa i compila **l'exemple anterior**. Afegeix un mètode a la classe `Matemàtica` anomenat `esPrim` que donat un `long` passat com a argument, ens indique si es tracta d'un nombre prim o no. *(Un nombre prim és aquell que només és divisible per ell mateix i l'unitat)*
- Crea un programa que, utilitzant la classe `Matemàtica`, demane números a l'usuari de forma indefinida e indiqueu si són prim o no.

```
PROGRAMA NOMBRES PRIMERS
```

```
-----
```

```
Introdueix un nombre: 5
```

```
El número 5 és prim
```

```
Introdueix un nombre: 10
```

```
El número 10 no és prim
```


9. ACTIVITATS PRÈVIES

- Això és tot ... de moment :-)