

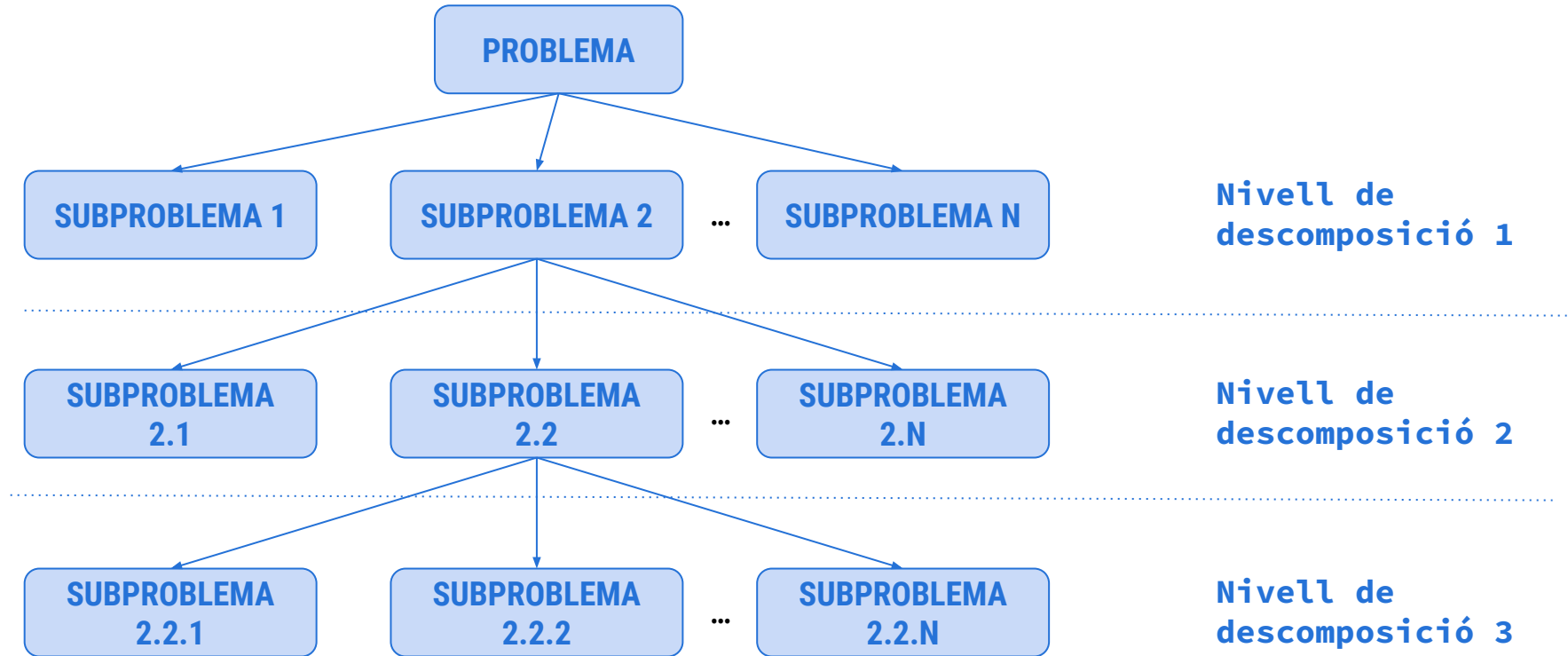
M03. PROGRAMACIÓ BÀSICA

UF2. Disseny modular

1. FUNCIONS

DISSENY MODULAR. DESCOMPOSICIÓ DEL PROBLEMA

Tal i com vam veure a la UF1, a l'hora de resoldre un problema, apliquem el disseny top-down (descendent).



QUÈ ÉS UNA FUNCIO?

Un **bloc de sentències** que executa una tasca específica i al que fem referència mitjançant un **nom**.

En java, aquesta funcionalitat s'anomena **mètode**.

QUÈ S'HA D'ESPECIFICAR A L'HORA D'ESCRIURE UN MÈTODE?

NOM

El mètode ha de tenir un nom per tal de poder-la cridar

COS

Conté les operacions (declaracion, condicions, iteracions) que ha d'executar el mètode

ARGUMENTS

Paràmetres que ha de rebre (o no) el mètode per tal d'executar les operacions del cos

RETORN

Cal indicar el tipus del resultat que retornarà (si en retorna) o si no retorna res

IMPLEMENTACIÓ DEL DISSENY MODULAR

```
function
  var
    integer num1, num2
  endvar
  num1 = validate()
  num2 = validate()
  if num1 > num2 then
    write("num1 és major")
  else
    if num1 < num2 then
      write("num2 és major")
    else
      write("Són iguals")
    endif
  endif
endfunction

function validate()
  var
    integer num
  endvar
  do
    write("Introdueix un nombre natural")
    read(num)
    while num<1
      return num
    endfunction
endfunction
```

ÀMBITS D'UNA VARIABLE

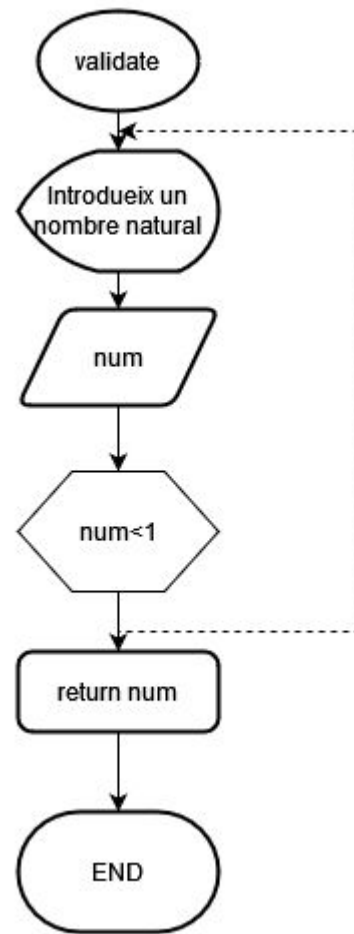
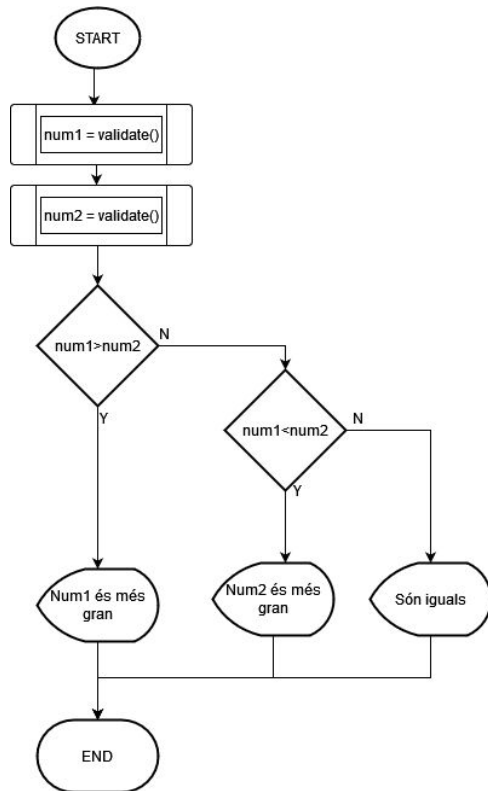
Una variable és **global** quan aquesta es declara en un programa fora de qualsevol bloc (és accessible des del seu punt de definició o declaració fins al final del codi font).

És a dir, existeix i té valor des del començament fins al final de l'execució del programa.

Una variable és **local** quan la seva declaració es fa dins d'un bloc (l'accés a aquesta variable queda limitat a aquest bloc i als blocs continguts dins d'aquest per sota del seu punt de declaració).

És a dir, és accessible només dins del bloc al qual pertany.

IMPLEMENTACIÓ DEL DISSENY MODULAR



MÉTODES EN JAVA

DEFINICIÓ D'UN MÈTODE

```
tipus nomMetode (llistaArguments){  
    (declaracions de variables locals);  
    operacions (amb els paràmetres i variables locals);  
    (retorn)  
}
```

Les variables declarades en el cos de la funció són **locals** (només són accessibles dins el mètode).

El paràmetre *tipus* indica de quin tipus serà el valor retornat pel mètode (int, float, char, objectes,...)

DEFINICIÓ D'UN MÈTODE

tipus nomMetode (llistaArguments)

```
void nomMetode ()  
void nomMetode (int)  
int nomMetode ()  
int nomMetode (int)
```

La **declaració d'un mètode (prototipus)** indica:

- ☐ el nom del mètode
- ☐ quants arguments té i de quin tipus són
- ☐ el tipus del valor retornat

Important! Podem declarar un mètode amb el mateix nom i diferent tipus d'arguments (overloading, OOP).

INVOCACIÓ DE MÈTODES. EXEMPLE

```
public class Metodes {  
    public int val1 = 3, val2 = 4;  
  
    public static void main(String[] args) {  
        Metodes method = new Metodes();  
        System.out.println(method.sumar());  
    }  
  
    public int sumar() {  
        return val1+val2;  
    }  
}
```

INVOCACIÓ DE MÈTODES. EXEMPLE II

```
//mètode que retorna la suma dels dos arguments  
public int sumar(int num1, int num2){  
    return num1+num2;  
}
```

```
//mètode que retorna la suma dels dos valors globals  
public int sumar(){  
    return val1+val2;  
}
```

INVOCACIÓ DE MÈTODES

Tenint en compte els exemples anteriors, quina de les opcions és millor a l'hora de declarar un mètode?

Cal tenir en compte els següents punts:

- **Àmbit**: els valors amb els que treballaran els mètodes seran locals o globals?
- **Retorn**: el resultat de les operacions executades dins del mètode ha de retornar al programa principal o a un altre mètode o no?

Un cop resolts els punts anteriors, podem definir els nostres mètodes segons una de les 4 combinacions possibles.

EXERCICIS I

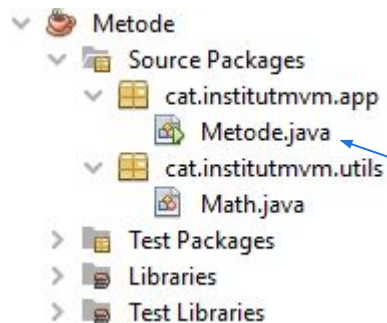
Exercici 1

Crea un programa que demani un nombre per teclat i validi si aquest és natural.

Exercici 2

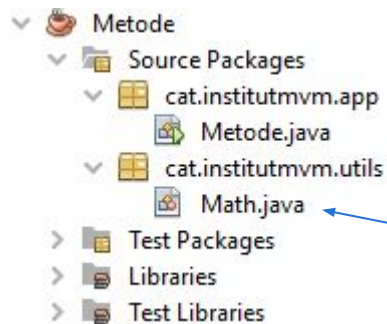
Crea un programa que demani per teclat un el radi d'una circumferència (nombre enter) i retorni la seva longitud.

ESTRUCTURA DEL PROYECTO



```
1 package cat.institutmvm.app;
2
3 import java.util.Scanner;
4 import cat.institutmvm.utils.Math;
5
6 public class Metode {
7     //variables globals
8     private int val1 = 3, val2 = 4;
9     private static final String MSG_1 = "Introdueix dos nombres: ";
10    private static final String MSG_2 = "Resultat amb valors globals: ";
11    private static final String MSG_3 = "Resultat amb pas de paràmetres: ";
12
13    public static void main(String[] args) {
14        int num1, num2;
15        Scanner sc = new Scanner(System.in);
16        Metode method = new Metode();
17        Math mt = new Math();
18        System.out.println(MSG_1);
19        num1 = sc.nextInt();
20        num2 = sc.nextInt();
21        System.out.println(MSG_3 + mt.sumar(num1, num2));
22        System.out.println(MSG_2 + method.sumar());
23    }
24
25    //mètode que retorna la suma dels dos valors globals
26    public int sumar(){
27        return val1+val2;
28    }
29 }
```

ESTRUCTURA DEL PROYECTO



```
1 package cat.institutmvm.utils;
2
3 public class Math {
4     //mètode que retorna la suma dels dos valors globals
5     public int sumar(int num1, int num2){
6         return num1+num2;
7     }
8 }
```


2. LLIBRERIES

LLIBRERIES (BIBLIOTEQUES)

Les llibreries o biblioteques són una recopilació de rutines que implementen operacions.

Als nostres programes podem incloure les llibreries pròpies del llenguatge (lectura de dades, operacions aritmètiques,...). També podem recopilar els nostres mètodes en llibreries i incloure-les en les nostres aplicacions.

En el cas de Java, aquests mètodes es defineixen dins de classes, que es classifiquen en diferents packages (segons la tipologia dels mètodes). Les classes pròpies de l'arquitectura Java estan dins l'[**API de Java**](#).

JAVA API

Alguns dels packages més importants són els següents:

- [java.lang](#): conté les classes fonamentals per a la programació en Java
- [java.util](#): conté diferents utilitats per a la conversió de cadenes i lectura de dades i el framework *collections*, entre d'altres.
- [java.nio.file](#): defineix les classes per a accedir a fitxers i fitxers del sistema
- [java.time](#): conté les classes principals per dates, temps i durades