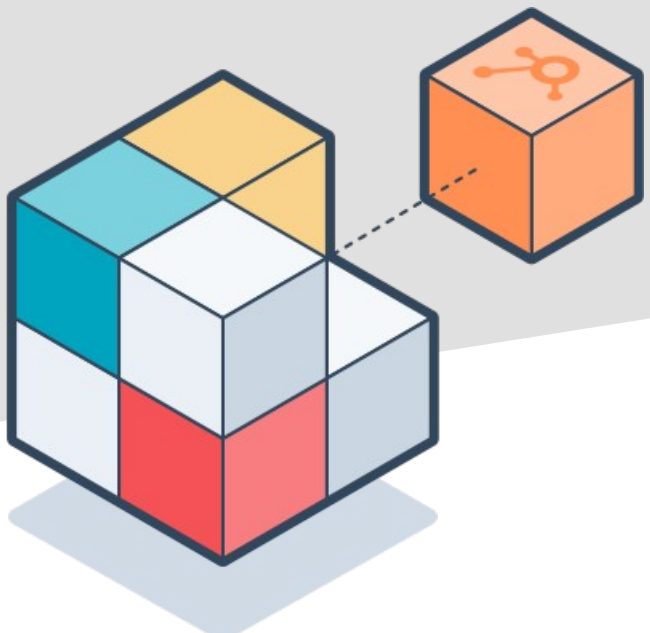
Tema 03. Programación modular



Índice

1.- Introducción.

La programación modular surge de la evolución de la programación estructurada.

Consiste en crear módulos o **partes del programa reutilizables**, que se unirán para resolver el problema original.

Se basa en la técnica "**divide y vencerás**".

Evita que en los programas exista código repetido.

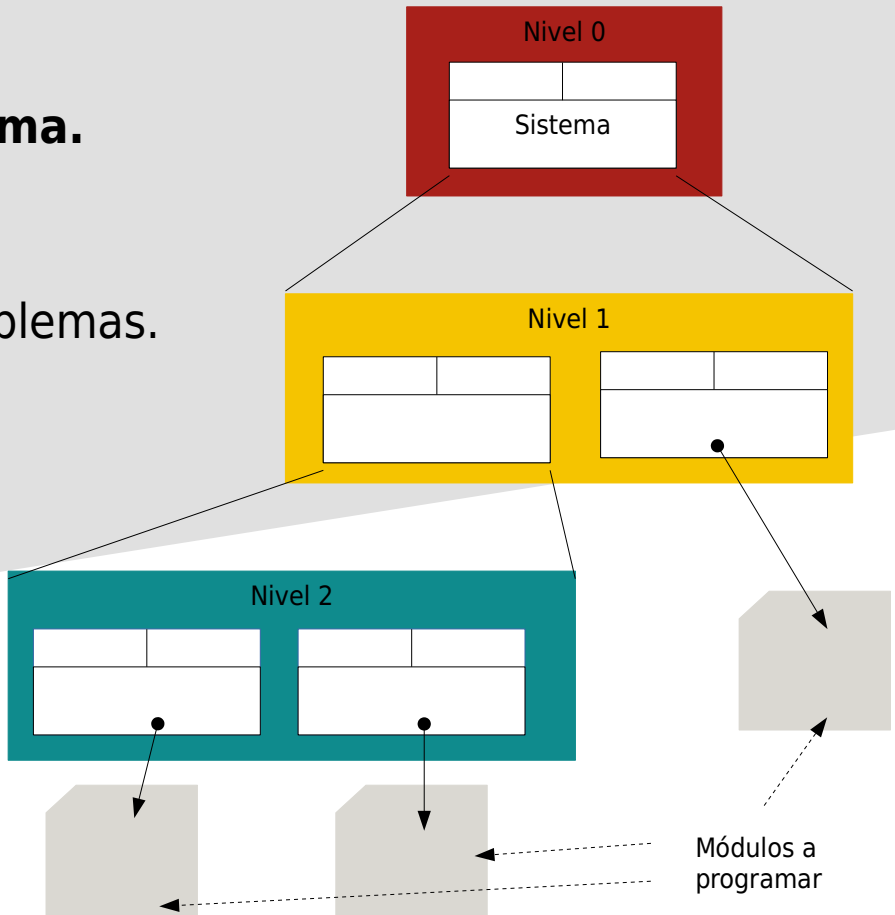
1.- Introducción.

Descomposición modular de un sistema.

Dividimos el problema en subproblemas.

A su vez pueden dividirse en más subproblemas.

Programamos los diferentes módulos.



Resolviendo los subproblemas y reutilizando los módulos, obtenemos la solución del problema original.

2.- Programación modular en Java.

En Java la programación modular se basa en el concepto de **método** (por ahora serán estáticos).

Un método es un conjunto de instrucciones de programa que realizan una determinada tarea. Tiene una serie de **datos de entrada** y pueden devolver un **valor de retorno**.

Los métodos se podrán “invocar” o “llamar” desde el main o desde otro método.

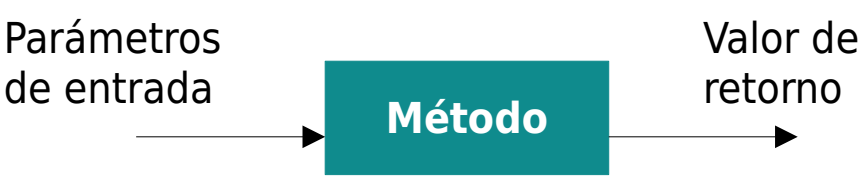
Por ahora trabajaremos con métodos que serán siempre estáticos y privados (private static).

Cada método representa un módulo del programa y contribuye a la solución del problema principal.

3.- Métodos.

Un método tiene:

- Una lista de **parámetros de entrada** al método (datos que necesita para hacer la tarea específica).
- Un **valor de retorno** (valor que devuelve el método).
- Puede tener también **variables locales** al método.



3.- Métodos.

Un método puede tener de 0 a N parámetros de entrada.

Los parámetros (formales) tienen:

- Nombre
- Tipo
- Valor (viene dado)

Un método puede o no tener un valor devuelto (asociado).

3.- Métodos.

Sintaxis de un método estático en Java

```
private static TipoDevuelto nombreMetodo (listaDeParametros) {
    // Variables locales al metodo
    // Codigo del metodo
    // Instrucción return
}
```

TipoDevuelto indica que tipo de dato devuelve el método: int, float, char…

ListaDeParametros representa los datos que van a “entrar” en el método. Para cada dato de entrada hay que definir de que **tipo** es y darle un **nombre**.

Ejemplo: cabecera de un método que calcula el área de un rectángulo

```
private static int calculaAreaRectangulo (int base, int altura) {
```

3.- Métodos.

PARÁMETROS

- TipoDevuelto es el tipo de dato que devuelve el método, **SOLO UNO**:
  - *int, float, char...* (**void**, si no devuelve nada)
- ListaDeParametros
  - (tipo param1, tipo param2, ... tipo paramN). Ejemplo *(int base, int altura)*
- La sentencia **return** aparecerá siempre salvo que el método no devuelva nada (sea void).
- Las variables locales solo son visibles desde el código del método.

  Ámbito de variables (scope)

3.- Métodos.

LLAMADAS A MÉTODOS

- La llamada a un método estático que **NO** devuelve nada (void) se realiza con:
  - `NombreClase.nombreMetodo(parametrosReales)`
- La llamada a un método estático **que devuelve un valor** se realiza con:
  - `variable = NombreClase.nombreMetodo(parametrosReales)`
  - `// La variable debe ser del tipo devuelto por el método`
- En los parámetros reales no hay que poner el tipo, solo el nombre o valor. Van separados por comas y **deben coincidir en orden, tipo y numero** con los parámetros formales.

3.- Métodos.

LLAMADAS A MÉTODOS

Cuando desde un programa P se hace una llamada o se “invoca” un método M ¿Qué ocurre?

1. Se “traspasan” los parámetros reales (los de P) en los parámetros formales (los de M).
2. A continuación se ejecuta el código del método M.
3. El método M termina la última instrucción y devuelve el valor de retorno (opcional), volviendo el control al programa P.
4. El programa P continúa por la siguiente instrucción a la llamada al método.

3.- Métodos.

LLAMADAS A MÉTODOS

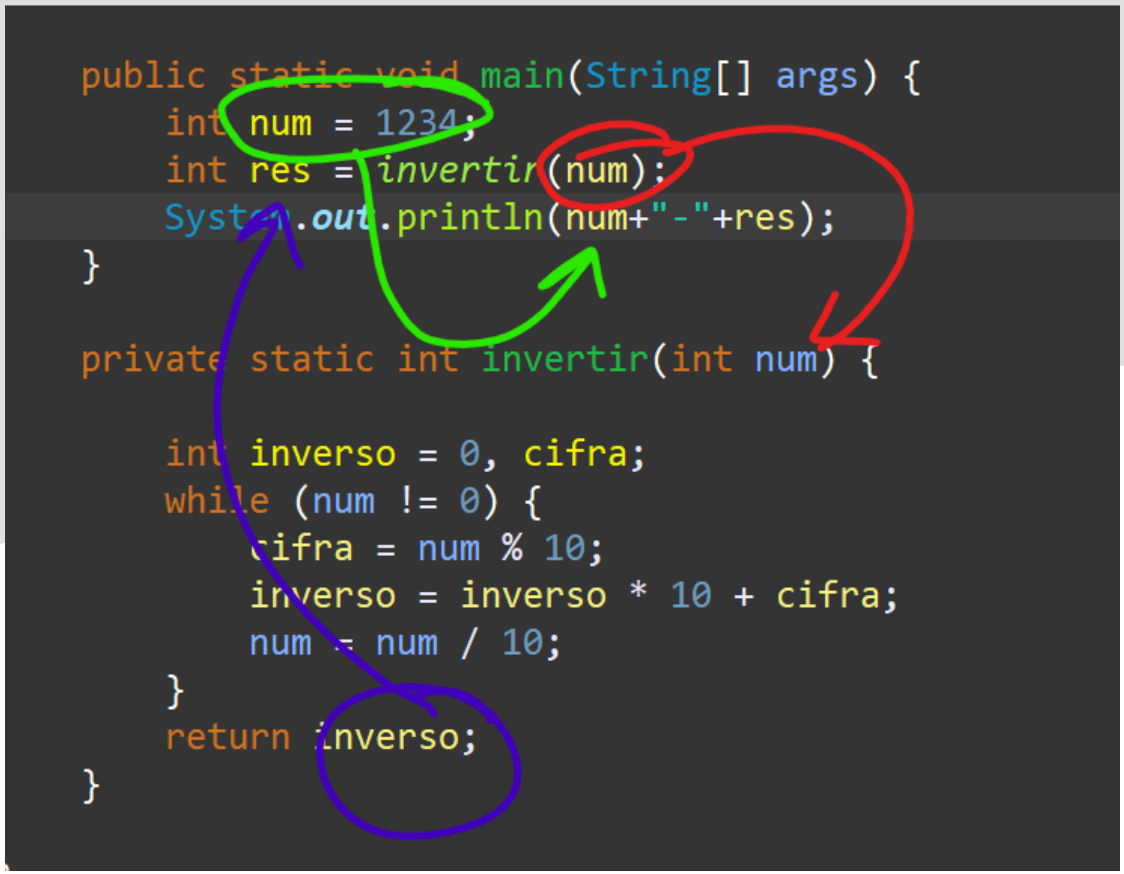
Existen 2 variable **num**

1. En el *main*
2. En el *invertir*

Desde el método "*invertir*" no se modifica la variable **num** del "*main*", aunque se llamen igual.

En consola mostrará:

1234-4321



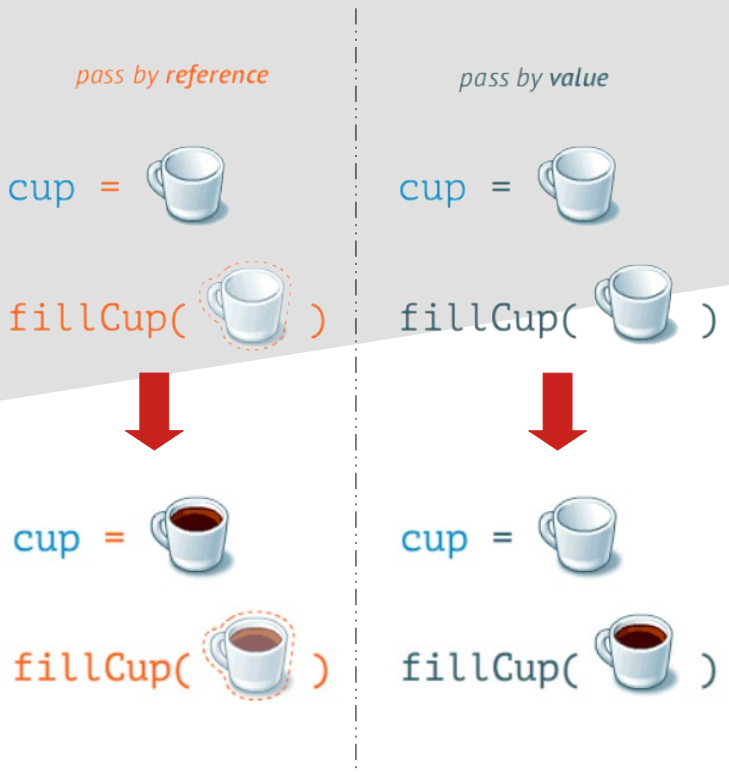
3.- Métodos.

PASO DE PARÁMETROS

Si los parámetros son de tipos básicos el paso de parámetros se hace “**por valor**” (también llamado “por copia”).

Si los parámetros son objetos (lo veremos en el próximo tema), el paso de parámetros es “**por referencia**”.

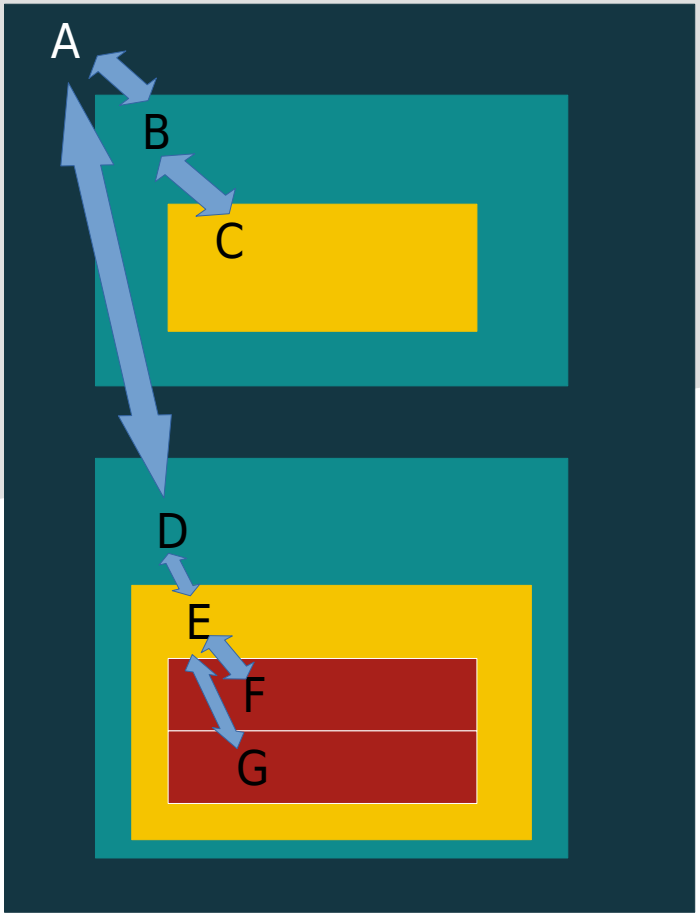
Los parámetros reales y los formales deben coincidir en **orden, número y tipo.**

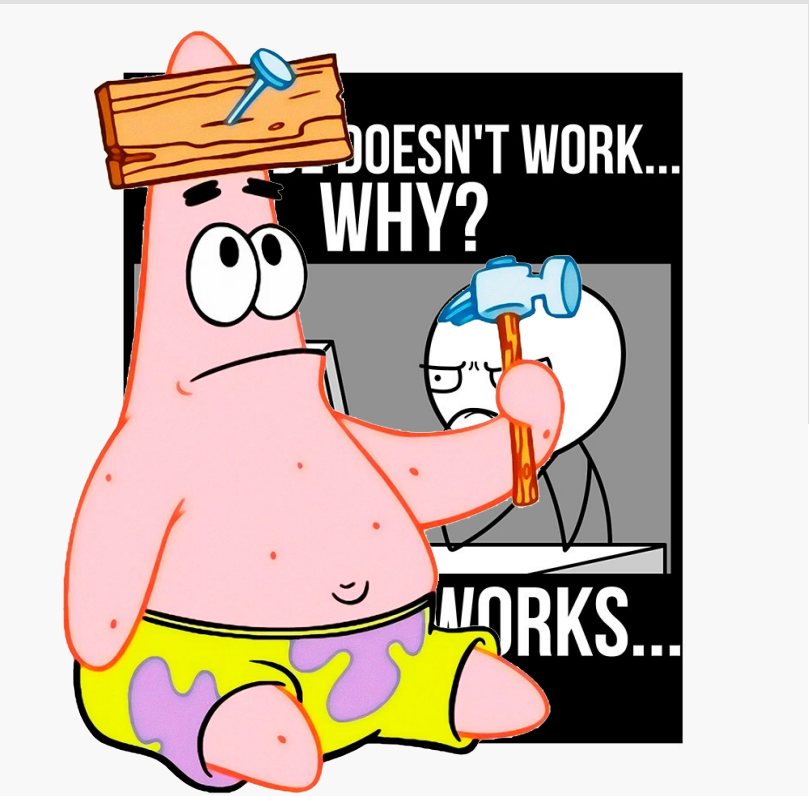


3.- Métodos.

ÁMBITOS VARIABLES

VARIABLE	ACCESIBLE DESDE
A	A, B, C, D, E, F, G
B	B, C
C	C
D	D, E, F, G
E	E, F, G
F	F
G	G




DOESN'T WORK...
WHY?
WORKS...