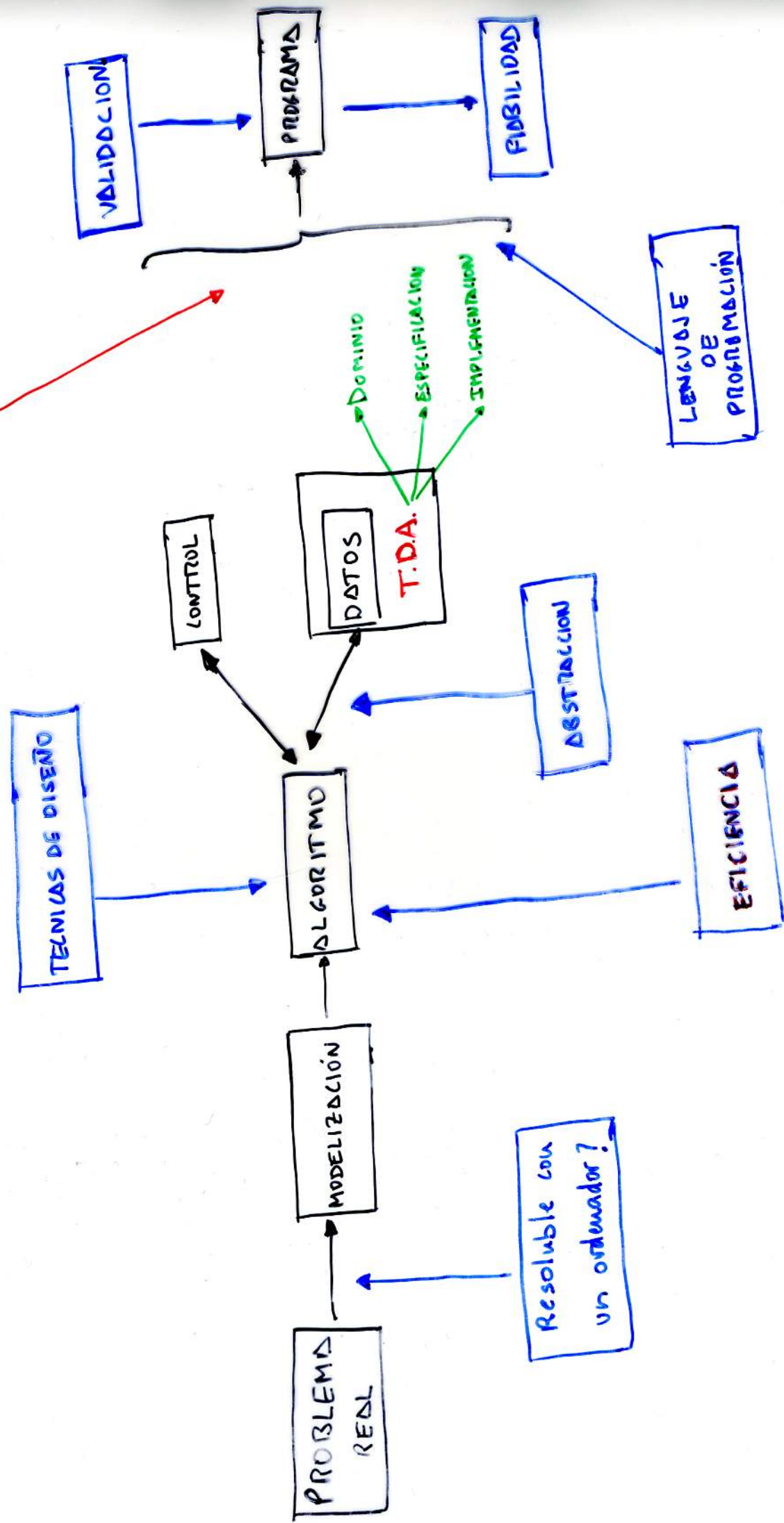


METODOLOGIAS DE LA PROGRAMACION



# Problemas, programas, algoritmos y estructuras de datos

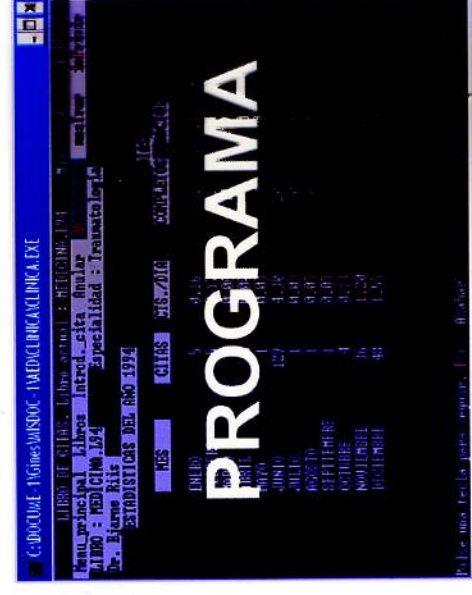


**Algoritmos**

+

**Estructuras  
de datos**

**PROBLEMA**



- **Problema:** Conjunto de hechos o circunstancias que dificultan la consecución de algún fin.
- **Algoritmo:** Conjunto de reglas finito e inambiguo.
- **Estructura de datos:** Disposición en memoria de la información.
- **Programa:** Algoritmos + Estructuras de datos.

## Resolución de problemas

### ARQUITECTO

### INFORMÁTICO

- |  |   |  |
|--|---|--|
| 1. Estudio de viabilidad, análisis del terreno, requisitos pedidos, etc. | → | 1. <b>Análisis</b> del problema              |
| 2. Diseñar los planos del puente y asignar los materiales.               | → | 2. <b>Diseño</b> del programa (alg. y estr.) |
| 3. Poner los ladrillos de acuerdo con los planos.                        | → | 3. <b>Implementación</b> (programación)      |
| 4. Supervisión técnica del puente.                                       | → | 4. <b>Verificación</b> y pruebas             |

## Resolución de problemas

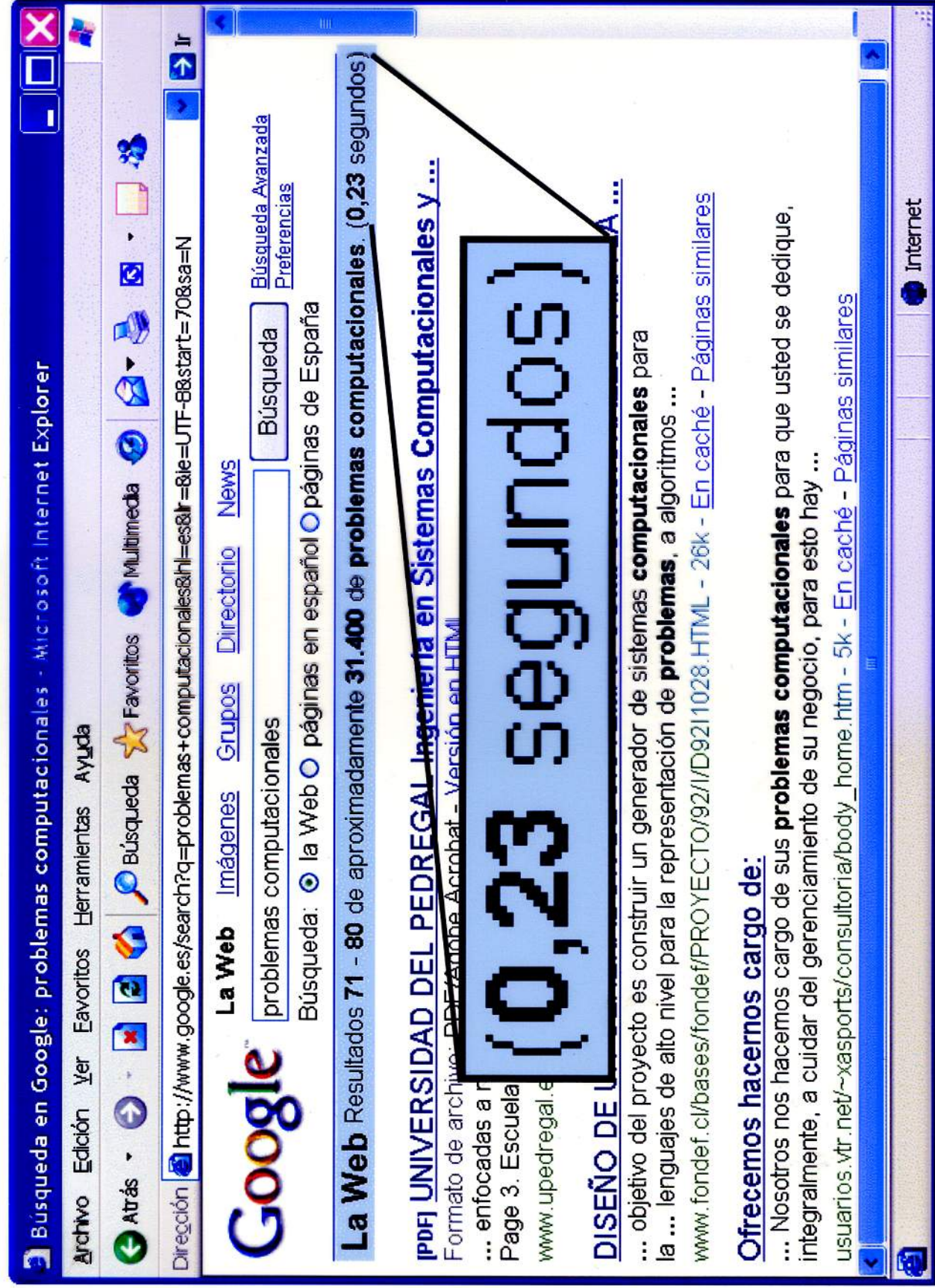
### MÉTODO CIENTÍFICO

### INFORMÁTICO

- |                     |   |  |
|---------------------|---|--|
| 1. Observación.     | ↔ | 1. <b>Análisis</b> del problema              |
| 2. Hipótesis.       | ↔ | 2. <b>Diseño</b> del programa (alg. y estr.) |
| 3. Experimentación. | ↔ | 3. <b>Implementación</b> (programación)      |
| 4. Verificación.    | ↔ | 4. <b>Verificación</b> y pruebas             |



# Buscador de Internet





# Planificador de rutas

**Guía Campsa España 2000, INTERACTIVA**

**Informe del Itinerario**

**Origen:** Cagitan

**Destino:** CORUÑA, LA/ A CORUÑA

**Distancia:** 959.2 Km

**Tiempo:** 9 h 49 m

**Población** Cagitan

**NUCLEO**

**Población:** 32 habitantes

**Superficie:** 45097 m<sup>2</sup>

**Altitud:** 460 m

**Provincia:** MURCIA

**Poblaciones**

Capitales de Provincia

Más de 20.000 habitantes

De 5.000 a 20.000 habitantes

Menos de 1.000 habitantes

● **TOLEDO**

● **COSLADA**

● **Aguadulce**

● Leiza

● Ribanera del Pinar

**Tráfico**

Autopistas

Autovías

Nacionales

1er Orden

2º Orden

Locales

Otras

Tráf. Interrumpido

Tráf. Saturado

Tráf. Discontinuo

Tráf. Intenso

**Hito**

**Distancia**

● Cagitan por la C-330

● C-330 → N-301a

● N-301a - Km S.N. - CIEZA

● CIEZA

● N-301a → N-301

● N-301 - Km 316.1 - HELLIN

● N-301 - Km 233 - TOBARRA

**Tráfico**

Autopistas

Autovías

Nacionales

1er Orden

2º Orden

Locales

Otras

Tráf. Interrumpido

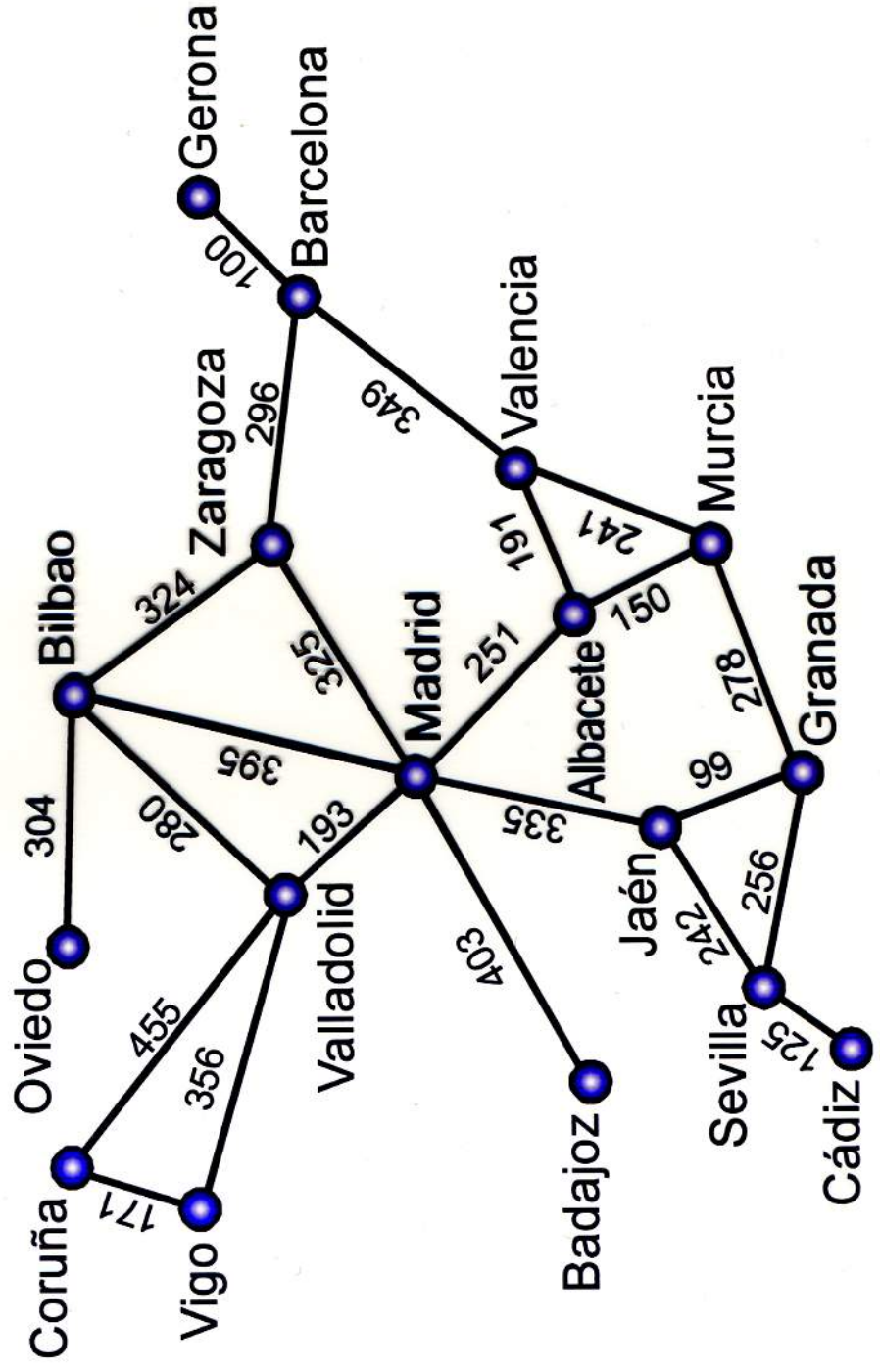
Tráf. Saturado

Tráf. Discontinuo

Tráf. Intenso

# Planificador de rutas

- Representación mediante un grafo:
  - Lugares = nodos.
  - Carreteras = arcos entre nodos.





# Jugador de Ajedrez

**Game 2  
Kasparov  
Resigns!**

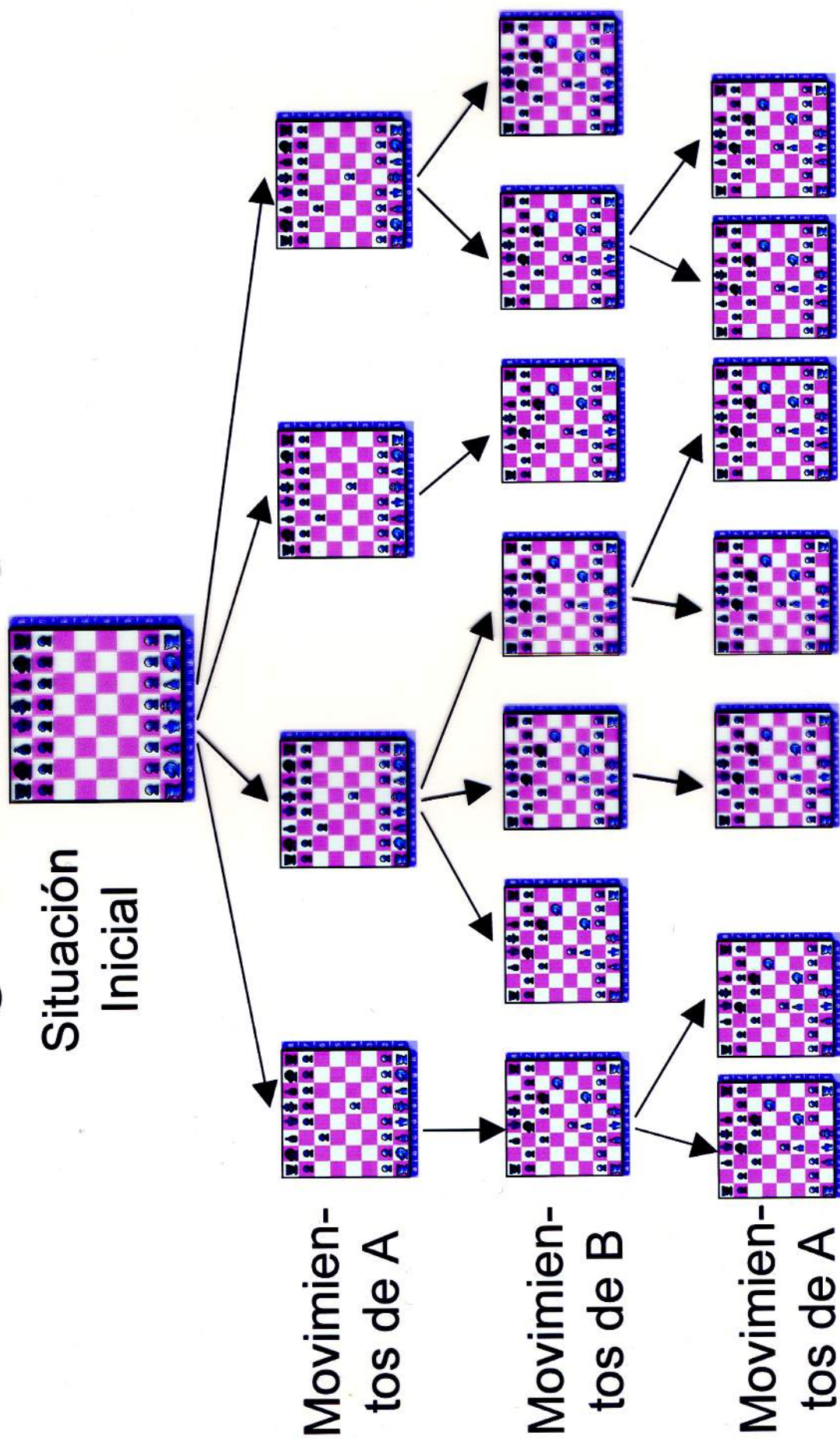


**Game 6, black  
19...Kasparov  
resigns!**



- En mayo de 1997 Deep Blue (de IBM) gana a Kasparov.

# Jugador de Ajedrez





# Jugador de Ajedrez

- Suponiendo que cada jugador hace unos 50 movimientos, el factor de ramificación medio es de 35 posibles movimientos.
- Tamaño del árbol:  $35^{100} = 2,5 \cdot 10^{154}$
- ¡¡Sólo existen  $10^{87}$  partículas subatómicas en el universo!!

# Evolución e historia de la programación

Lenguajes  
de bajo nivel



(Basic, Fortran,  
Ensamblador, ...)

## Ejemplo de programa BASIC

```
10 PAPER 7: BORDER 7: INK 0: BRIGHT 0: FLASH 0
20 DIM a$(22,20): DIM f(22): DIM c(22): DIM g$(11,2): DIM z$(22,18):
  DIM x$(22)
30 FOR n= 1 TO 22
40 READ f,c: LET b$=CHR$ 19+CHR$ 1: LET f(n)=f: LET c(n)=c
50 FOR m=0 TO 2: READ r$
60 LET b$=b$+CHR$ 22+CHR$ (f+m)+CHR$ c+ r$
70 NEXT m: LET a$(n)=b$: NEXT n: GO SUB 470
80 CLS : FOR N=1 TO 22: PRINT A$(N): NEXT N: IF x$(1)<>" " THEN LET
  g$=x$
90 PRINT AT 0,2;"■";AT 1,2;"| EBEO";AT 2,2;"■";AT 3,2;"| OBLE";AT
  4,2;"■ "; INK 3; AT 19,16;"Adaptacion para"; INK 1;AT
  20,19;"MICRO";" HOBBY"
100 PLOT 128,0: DRAW 0,170: DRAW 10,4: DRAW 24,1: DRAW 82,0
110 PLOT 128,0: DRAW 10,4: DRAW 24,1: DRAW 88,0
120 DRAW 0,164: DRAW -2,2: DRAW 0,-164: DRAW -2,2: DRAW 0,164: DRAW -
  2,2: DRAW 0,-165
130 PLOT 128,0: DRAW -10,4: DRAW -24,1: DRAW -88,0
140 DRAW 0,164: DRAW 2,2: DRAW 0,-164: DRAW 2,2: DRAW 0,164: DRAW 2,2:
  DRAW 0,-164
150 PLOT 128,170: DRAW -10,4: DRAW -24,1: DRAW -82,0
160 DATA 1,12,"| ","■","■","■","■",1,17,"| ","| ","| ","1,22,"■ ","■
  ","■","1,27,"| ","■","■","■"
170 PLOT 128,2: DRAW -10,4: DRAW -24,1: DRAW -85,0
180 PLOT 128,2: DRAW 10,4: DRAW 24,1: DRAW 85,0
```



## Ejemplo de programa BASIC

```

290 DIM b$(22,2): FOR n=1 TO 11: FOR m=1 TO 2
300 LET s=INT (RND*22)+1
310 IF b$(s,1)=" " THEN LET b$(s,1)=g$(n,1): LET b$(s,2)=g$(n,2):
    NEXT m: NEXT n: GO TO 330
320 GO TO 300
330 DIM f(22): LET di=0: LET itn=0: LET u=.001
340 PRINT AT 20,2;di: IF di=275000 THEN LET di=350000: PRINT AT
    20,2; FLASH 1;di;"CONSEGUIDO EL PLENO EN ";itn;" veces": PRINT
    #0;"Pulsa una tecla para empezar": GO SUB 440: GO SUB 440: GO SUB
    440: PAUSE 0: GO TO 80
350 INPUT n: IF n>22 OR n<1 THEN GO TO 350
360 IF r(n)=1 THEN GO TO 350
370 LET k=n: GO SUB 700
380 INPUT m: IF m>22 OR m<1 OR m=n THEN GO TO 380
390 IF r(m)=1 THEN GO TO 380
400 LET k=m: GO SUB 700
410 LET itn=itn+1: IF b$(n)=b$(m) THEN LET di=di+25000: PAPER 3: LET
    k=n: GO SUB 720: PAPER 3: LET k=m: GO SUB 720: LET r(n)=1: LET
    r(m)=1: GO SUB 440: GO SUB 450: GO TO 340
420 BRIGHT 1: PAUSE 45: LET f=f(n): LET c=c(n): PRINT AT
    f,c;a$(n,8);AT f+1,c;a$(n,14);AT f+2,c;a$(n,20): PRINT AT
    f,c;a$(n,7 TO 8);AT f+1,c;a$(n,13 TO 14);AT f+2,c;a$(n,19 TO 20):
    BEEP .01,-10: PRINT a$(n): BEEP .02,0
430 LET f=f(m): LET c=c(m): PRINT AT f,c;a$(m,8);AT
    f+1,c;a$(m,14);AT f+2,c;a$(m,20): PRINT AT f,c;a$(m,7 TO 8);AT
    f+1,c;a$(m,13 TO 14);AT f+2,c;a$(m,19 TO 20): BEEP .01,-10: PRINT
    a$(m): BEEP .02,0: BRIGHT 0: GO TO 350

```

PROGRAMA PRINCIPAL  
DATOS

PROGRAMA

## Ejemplo de programa BASIC

```

430 LET f=f(m): LET c=c(m): PRINT AT f,c;a$(m,8);AT
    f+1,c;a$(m,14);AT f+2,c;a$(m,20): PRINT AT f,c;a$(m,7 TO 8);AT
    f+1,c;a$(m,13 TO 14);AT f+2,c;a$(m,19 TO 20): BEEP .01,-10:
    PRINT a$(m): BEEP .02,0: BRIGHT 0: GO TO 350
440 BEEP .07,15: BEEP .06,25: BEEP .07,35: BEEP .07,35: BEEP
    .09,40: RETURN
450 INK 8: LET xx=c(n)*8-2: LET yy=177-(f(n)*8): PLOT xx,yy: DRAW
    27,0: DRAW 0,-27: DRAW -27,0: DRAW 0,27
460 LET xx=c(m)*8-2: LET yy=177-(f(m)*8): PLOT xx,yy: DRAW 27,0:
    DRAW 0,-27: DRAW -27,0: DRAW 0,27: INK 0: RETURN
470 RESTORE 260: FOR n=1 TO 22
475 IF n=17 THEN LET g$(6,2)=" ": GO TO 540
480 READ p$
490 FOR m=0 TO 7: READ f: POKE USR p$+m,f: NEXT m
520 IF n<12 THEN LET g$(n,1)=p$
530 IF n>11 THEN LET g$(n-11,2)=p$
540 NEXT n: RETURN
700 PAPER 5: LET y$b$(k,1): LET t$b$(k,2): LET f=f(k): LET
    c=c(k): BEEP u,25: PRINT AT f,c+2;t$:AT f+1,c+2;" ";AT
    f+2,c+2;" ": BEEP u,49: BEEP u,25
710 PRINT AT f,c+1;t$;" ";AT f+1,c+1;" ";y$:AT f+2,c+1;" v": BEEP
    u,49: BEEP u,25
720 PRINT AT f(k),c(k);b$(k,2);" ";b$(k,2);AT f(k)+1,c(k);"
    ";b$(k,1);" ";AT f(k)+2,c(k);" v ": BEEP u,49: PAPER 7: RETURN

```

## Lenguajes de bajo nivel

- No existen procedimientos ni funciones
- No existen registros ni tipos definidos por el usuario
- No existen bloques estructurados (while, repeat, etc.)
- En definitiva: no hay **abstracciones**
- Y sin embargo... funciona:

[TEBEODOBLE](http://dis.um.es/~ginesgm/museo.html)

<http://dis.um.es/~ginesgm/museo.html>

## Evolución e historia de la programación

Lenguajes  
de bajo nivel

Lenguajes  
estructurados



(Basic, Fortran,  
Ensamblador, ...)

(Pascal, C,  
Modula, ADA, ...)



## Lenguajes estructurados

**UNIT** calculo;

**INTERFACE**

**const**  
 NMAX= 10;  
 MAX\_GUARDA= 2000;

**type**  
 TDatosEnt= **array** [1..NMAX] **of** integer;  
 TDatosSal= **record**  
   NPasos: Shortint;  
   Paso: **array** [1..NMAX-1] **of** record  
     O1: byte;  
     O2: byte;  
     Fn: byte;  
**end**;  
**end**;

**procedure** Operar (var Arr: TDatosEnt; O1, O2, Func, Nivel: byte; var Vale: boolean); forward;  
**procedure** CalculaCifras (var Entrada: TDatosEnt); forward;  
**procedure** CalculaCifrasRec (var Entrada: TDatosEnt; PA, PB, Func, Nivel: byte); forward;

Concepto de módulo/unidad

Separación de interface/implementación

Tipos definidos por el usuario

Procedimientos y funciones

## Lenguajes estructurados

**IMPLEMENTATION**

**var**  
 suma, num: integer;  
 CopiaOrden: TDatosEnt;

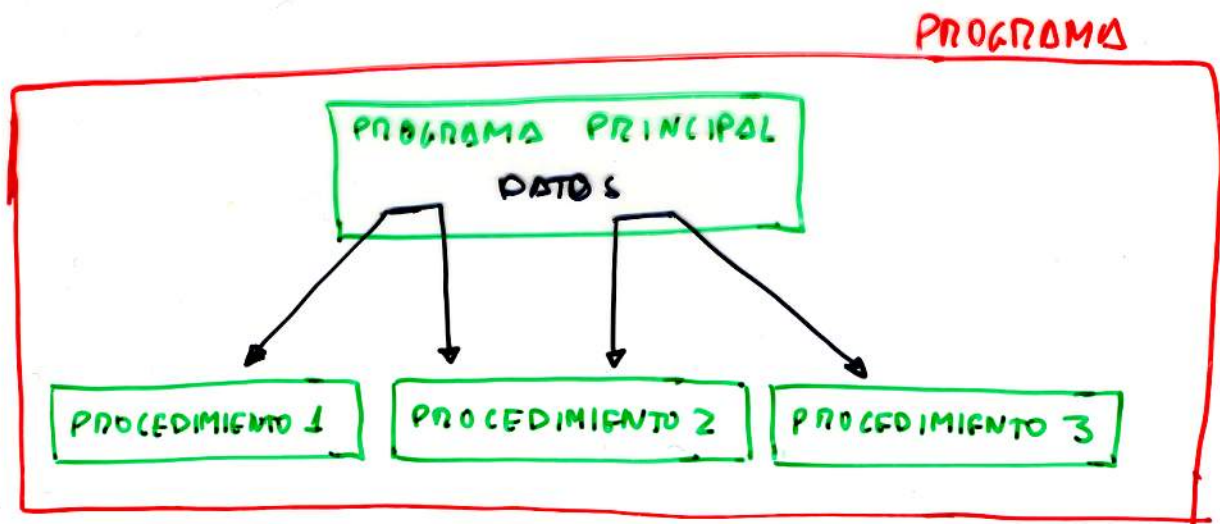
**procedure** OrdenaComb (var Entrada: TDatosEnt; Nivel: byte);  
**var**  
 i, j, maxim, pmaxim, tmp: integer;  
**begin**  
 CopiaOrden:= Entrada;  
 num:= Nivel;  
**for** i:= 1 **to** Nivel-1 **do begin**  
   maxim:= CopiaOrden[i];  
   pmaxim:= i;  
   j:= i+1;  
   **while** j<=Nivel **do begin**  
     **if** CopiaOrden[j]>maxim **then begin**  
       maxim:= CopiaOrden[j];  
     **end**;  
   **end**;  
**end**;  
**end**;

Separación interface/ implementación

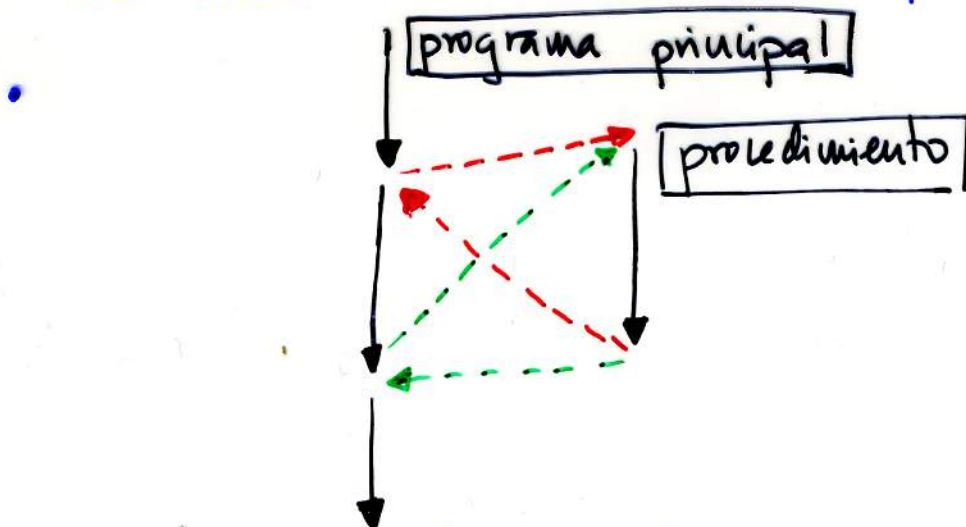
Procedimiento con parámetros

Bloques de control estructurados

# PROGRAMACION PROCEDURAL



- El programa es una secuencia de llamadas a procedimientos.
- El programa principal es responsable de pasar los datos a cada procedimiento que los procesa

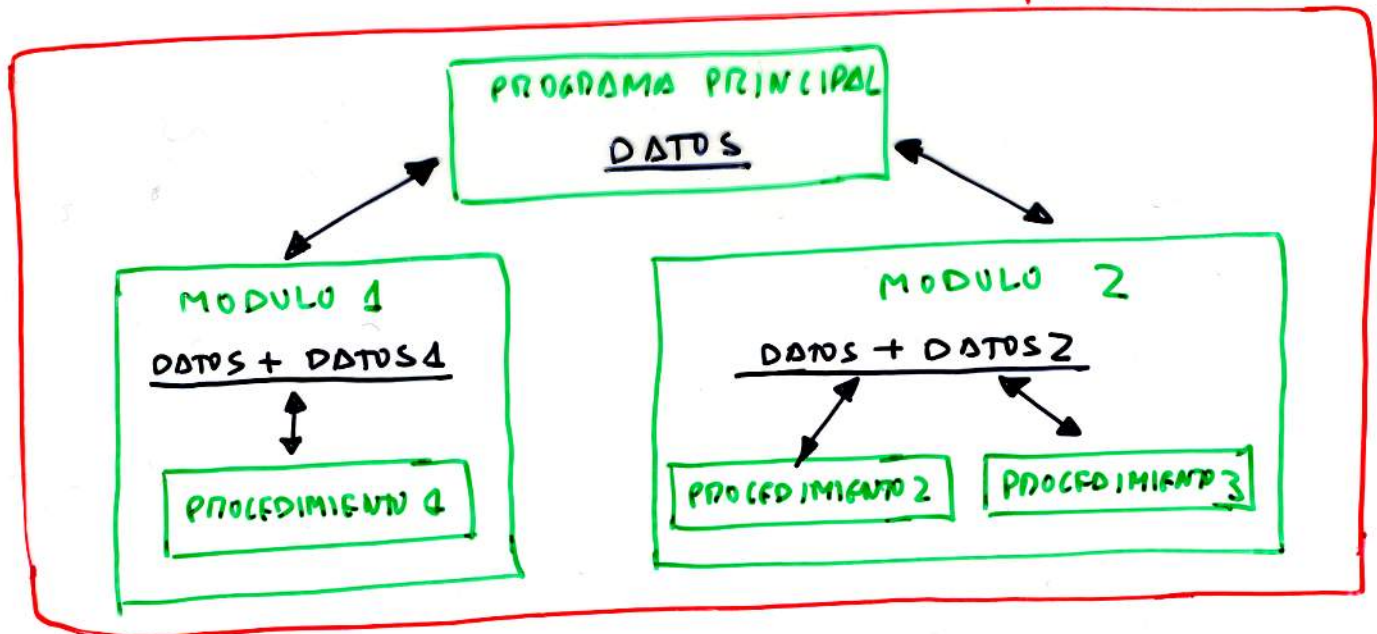


- Los procedimientos pueden anidarse: subprocedimientos
- Tenemos por tanto un programa simple que se divide en pequeñas piezas llamadas procedimientos
- Para permitir el uso de procedimientos o grupos de ellos en otros programas deben ser accesibles de forma separada: módulos → PROGRAMACION MODULAR



# PROGRAMACION MODULAR

PROGRAMA



- Los procedimientos de funcionalidad común se agrupan juntos en módulos separados
- El programa principal coordina las llamadas a los procedimientos en módulos separados. El programa se compone por tanto de diversas partes que interactúan a través de las llamadas a procedimientos.
- Cada módulo puede tener sus propios datos. Así, cada módulo tiene un estado interno que se modifica por las llamadas a procedimientos de ese módulo.  
pero:
  - solo hay un estado por módulo
  - cada módulo existe a lo más una vez en el programa completo

## ✱ INCONVENIENTES

## Lenguajes estructurados

### Inconvenientes:

- Los datos y los procedimientos de manipulación sobre los mismos van por separado.
- Es necesario garantizar la ocultación de la implementación.
- Proliferación de variables globales. ¿Qué papel juegan?
- Los programas son cada vez más complejos y difíciles de mantener.

## Evolución e historia de la programación

Lenguajes  
de bajo nivel

Lenguajes  
estructurados

Lenguajes  
orientados a objetos



(Basic, Fortran,  
Ensamblador, ...)

(Pascal, C,  
Modula, ADA, ...)

(Smalltalk, C++,  
Java, Eiffel, ...)



## Lenguajes orientados a objetos

// Interface

```
class Timer {
    private:
        double StartTime;
        double ClockRate;
    public:
        Timer (void);
        bool StartTimer (void);
        double ReadTimer (void);
        bool Exists;
};
```

Una clase es un Tipo Abstracto de Datos

Encapsulación de datos y operaciones

```
class Elipse {
    protected:
```

```
        double Fcx, Fcy;
        double Frx, Fry, Fang;
        void FsetXY (int x1, int y1, int x2, int y2);
```

Los datos son privados

```
    public:
```

```
        Elipse (int x1, int y1, int x2, int y2);
        Elipse * Clonar (void);
        void Pinta (lpImage *image, int color= 0, int ancho= -1);
```

Las operaciones son públicas

```
};
```

## Lenguajes orientados a objetos

// Implementación

Separación interface/ implementación

```
Timer::Timer (void)
```

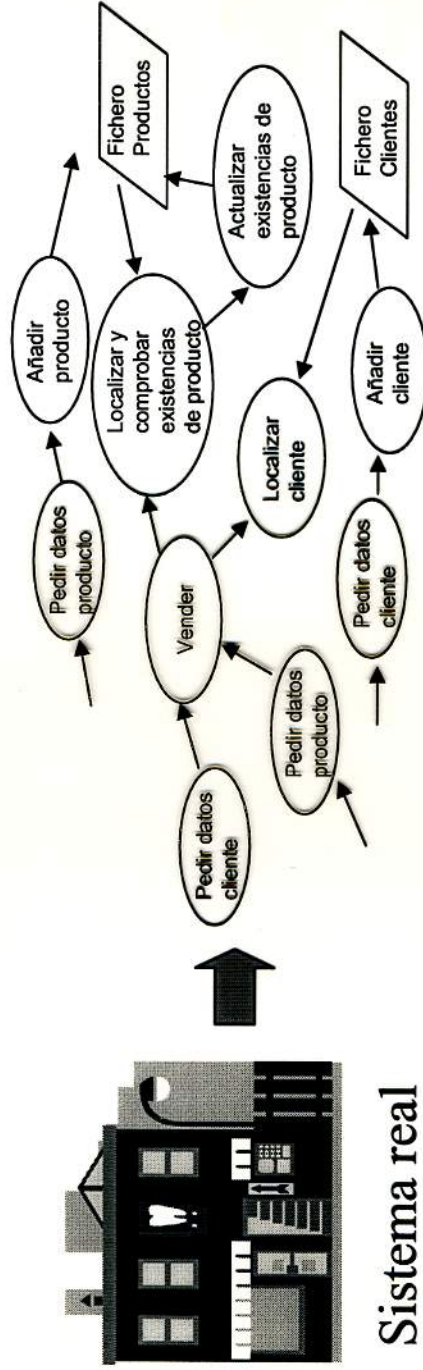
```
{
    LARGE_INTEGER *QW= new LARGE_INTEGER;
    Exists= QueryPerformanceFrequency(QW);
    ClockRate= QW->LowPart;
    delete QW;
}
```

```
bool Timer::StartTimer (void)
```

```
{
    LARGE_INTEGER *QW= new LARGE_INTEGER;
    bool res= QueryPerformanceCounter(QW);
    StartTime= QW->LowPart;
    delete QW;
    return res;
}
```

# PDO y la simulación de la realidad

## Metodologías tradicionales



## Visión de flujos y procesos

```
...  
void vender (unsigned ccliente, unsigned cproducto)  
{  
    ...  
    localizar_cliente (ccliente);  
    localizar_producto (cproducto);  
    ...  
}  
...
```

Código



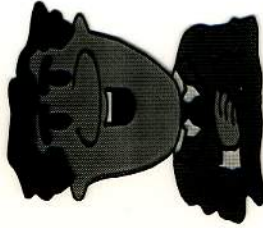
# Metodología Orientada a Objetos



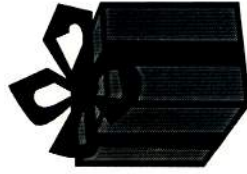
Sistema real



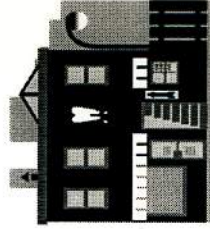
cliente



producto



tienda



Entidades



```
class cliente {
    unsigned int codigo;
    char *nombre;
    char *apellidos;

public:
    cliente (unsigned acodigo, char *anombre,
            char *apellidos);
};
```

Código

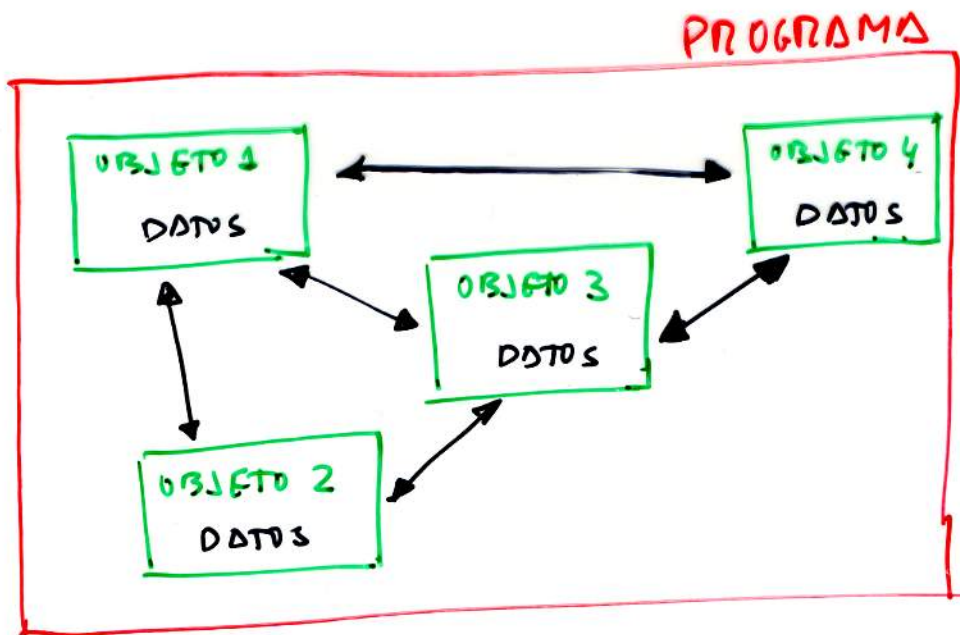
cliente
código
nombre
apellidos

producto
código
descripción
stock
precio
comprobar_stock
actualizar_stock

tienda
clientes
productos
añadir_cliente
añadir_producto
vender_producto

Visión de objetos

# PROGRAMACION ORIENTADA A OBJETOS



- El programa es una red de objetos interactuando entre sí (~~enviándose~~ se mandan mensajes unos a otros) cada uno manteniendo su propio estado.
- NO es una forma sofisticada de programación modular. su base está en los TDA

↓  
CLASES

Implementaciones de los TDA

- atributos: **datos del TDA**
- métodos: **operaciones del TDA**

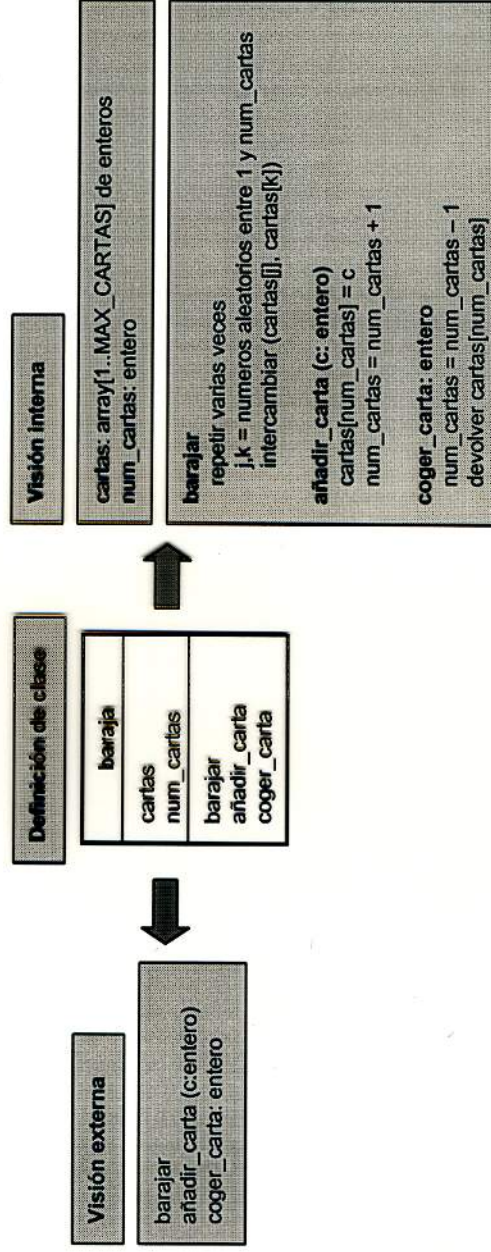
**OBJETOS**: **instancias de una clase**

- nombre: **su identificador**
- estado: **valores de sus atributos en un instante particular**



# Características de la PDO

- Una clase de objetos es un TDA. (Representación interna + operaciones).
    - Encapsulación
    - Ocultación de información.
- ↗ Visión externa o de interfaz (usuario de la clase).  
 ↘ Visión interna o de implementación (programador de la clase).



- Un objeto es usado normalmente a través de sus operaciones (paso de mensajes).

## Lenguajes orientados a objetos

- Una **clase encapsula** los datos de un tipo y las operaciones sobre el mismo
- Una clase es, al mismo tiempo, un **tipo abstracto de datos** y un **módulo** que encierra un conjunto de funciones relacionadas
- Separación clara entre **interface** (parte visible desde fuera) e **implementación** (oculta)
- ¿Qué hace? [VER](#)

## Resolución de problemas

¿Cómo resuelve un problema de programación un ingeniero?

A) Tecleando código en una máquina.

B) Siguiendo un proceso metódico.



## Conclusiones

1. **Proceso de análisis/diseño.** No empezar tecleando código como locos.
2. **Usar abstracciones**, respetando los dos principios básicos:
  - **Encapsulación:** las funciones relacionadas deben ir juntas (clases, módulos, paquetes, etc.).
  - **Ocultación de la implementación:** los aspectos de implementación no son visibles fuera del módulo, clase, etc.

## Conclusiones

3. **Reutilizar programas, librerías, tipos, etc. existentes.** Y programar pensando en la posible reutilización futura. Un nuevo programa no debe partir desde cero.
4. **No resolver casos concretos, sino el problema en general.** Si no se requiere un esfuerzo adicional, el algoritmo debe resolver un caso genérico.
5. **Repartir bien la funcionalidad.** Repartir la complejidad del problema de forma uniforme. No crear procedimientos muy largos: usar subrutinas. De esta forma se mejora la **legibilidad** del código.