ugr Universidad de Granada

Estructuras de Datos

Doble Grado en Ingeniería Informática y ADE

Curso 2022/2023

Prof. Joaquin Fdez-Valdivia

DECSAI
Universidad de Granada

Dpto. Ciencias de la Computación e Inteligencia Artificial
E. T. S. de Ingenierías Informática y de Telecomunicación
Universidad de Granada (España)

Página web

https://prado.ugr.es

- Temario, Tutorías, Bibliografía, Método de evaluación
- Transparencias, Apuntes adicionales
- Prácticas y material adicional
- Noticias, Mensajería, Foros
- Material adicional

IMPORTANTE: Apuntaros en el grupo de prácticas que os ha asignado el Centro

Rellenad datos: **Nombre**, **Apellidos** (los 2) y no estaría mal una **foto** en la que estéis reconocibles.

Profesorado - Tutorías

Teoría: Joaquin Fdez-Valdivia

Dpto. Ciencias de la Computación e Inteligencia Artificial
E.T.S. de Ingeniería Informática. Despacho nº 12 (4ª planta)
Email: jfv@decsai.ugr.es
Tutorías: L (10.30-11.30)
M (11.30-13.30)
X (9.30-12.30)
(o por email o videoconferencia en cualquier otro momento)

Prácticas: Javier Poyatos *(jpoyatosamador@ugr.es): Grupo 1*
Jose Enrique Cano (eco@decsai.ugr.es): Grupo 2

Temario - Teoría

Tema 1.- Introducción a la eficiencia
Tema 2.- Abstracción en programación: TDA
Tema 3.- TDA Lineales: Vectores, pilas, colas,
Listas, colas con prioridad
Tema 4.- Generalización: Plantillas
Tema 5.- Abstracción por iteración
Tema 6.- La Standard Template Library (STL)
Tema 7.- TDA no lineales: Arboles:
Arboles binarios. APO. ABB. AVL. Otros
Tema 8.- Tablas Hash
Tema 9.- Grafos

Bibliografía

Rosa Rodriguez-Sánchez, J. Fdez-Valdivia, J.A. García. Estructuras de datos en C++: Un enfoque práctico. 2020

Antonio Garrido, Joaquín Fdez-Valdivia. "Abstracción y estructuras de datos en C++". *Delta publicaciones. 2006.*

Bjarne Stroustrup. "El lenguaje de programación C++". *Ed. especial. Addison Wesley. 2000.*

R. Musser, J. Derge y A. Saini. STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library. 3 Ed. Adisson-Wesley 2009.

Prácticas

- Desarrollo de pequeñas aplicaciones con C++ usando los TDA estudiados
- GNU/Linux

5 prácticas voluntarias (3 puntuables)

- *Eficiencia teórica vs empírica*
- *Abstracción: TDA Imagen*
- *TDA Lineales: Pila/Cola con propiedades*
- *STL: Variante del TDA Diccionario/conjunto*
- *Arboles*

1 Práctica final (voluntaria) ⇒ **"Juego de las letras"**

Muy importante: entregas periódicas/llevarlas al día

Prácticas

Grupos de prácticas:

G1: Martes de 13.30 a 14.30, Aula 3.3
G2: Martes de 11.30 a 12.30, Aula 3.1

Los configura el Centro. Cuando estén configurados, cada uno se apuntará en la página web al grupo que le corresponda

Comienzo: Semana del 12 de Septiembre)

Sistema de evaluación

Convocatoria ordinaria:
Mini prácticas periódicas (3) + Práctica final: 3 puntos
- Retos (evaluación contínua): 1 punto
- Examen escrito: 6 puntos
- **Convocatoria extraordinaria**:
- Examen escrito (teoría y prácticas): 10 puntos
- Se guarda la nota de las prácticas y retos de la convocatoria ordinaria y en ese caso el examen vale 6 puntos

... que hay ahora en el web

- **Transparencias: Información general, Presentación de la asignatura, apuntes y recordatorio de temas de MPII, Transparencias y apuntes de eficiencia.**

- **Inscripción** a los grupos de prácticas.
- Algunos **manuales** para las prácticas (GNU/Linux, g++, Make).
- **Primer guión de prácticas**

- **Foros** de discusión.
- **Recursos** en Internet sobre C++.
- Material complementario

... sobre las clases

- Debemos estar preparados para cualquier cambio de situación que se presente

- Preguntad cualquier cosa que no entendáis. Podéis interrumpir las explicaciones para preguntar.

- **Si NO asistes a clase procura estar al día de lo que se ha visto/dicho.**

- Si se explica algo en clase es para que lo aprendáis, no para que lo filtréis. Atended y entended lo que se explica.

- La página web y el material que contiene no está de adorno. Daré por supuesto que conocéis y habéis leido/estudiado cualquier cosa que aparezca en el web.

Claves de la asignatura

Conceptos

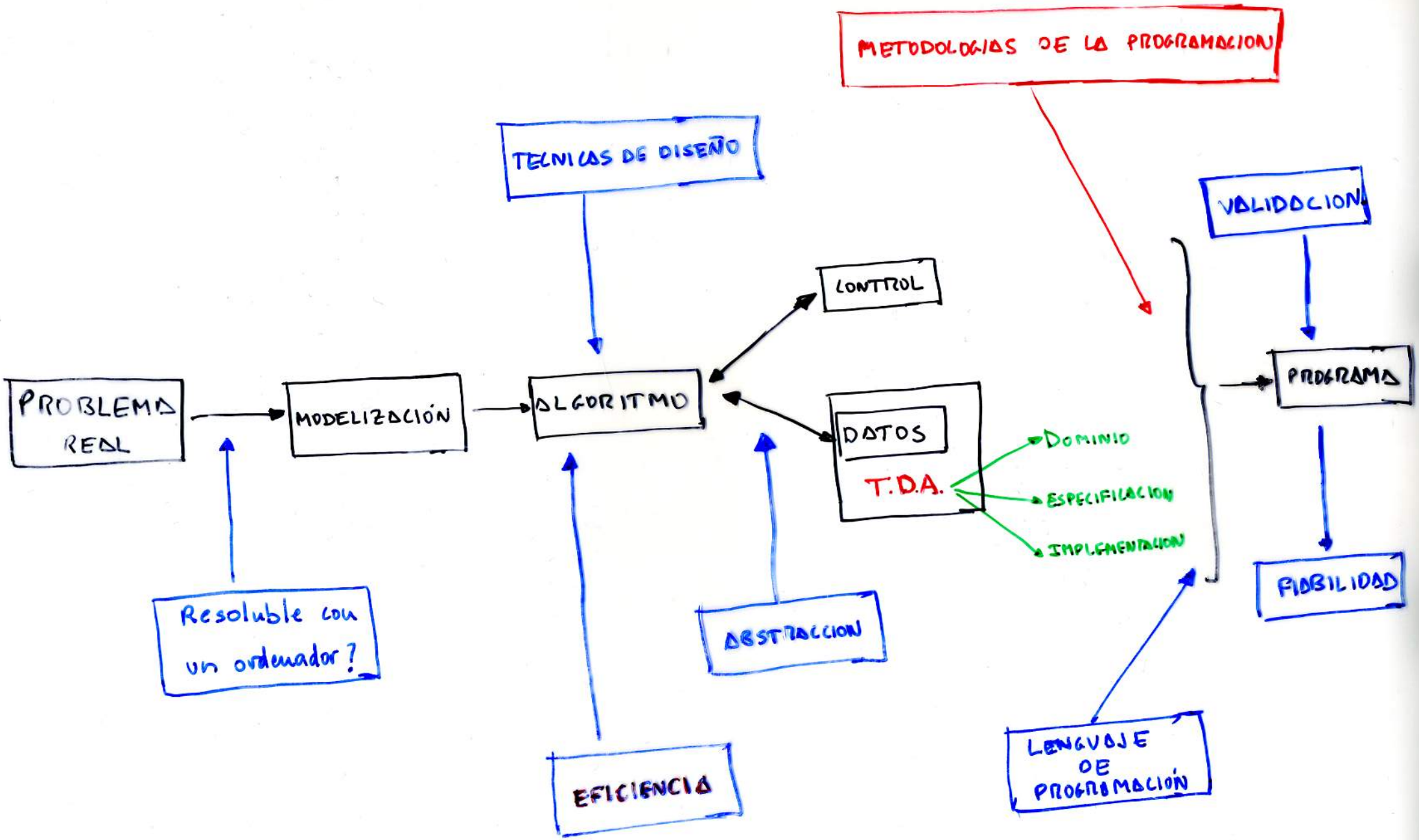
Cálculo básico de la eficiencia de un código
Conceptos básicos de abstracción y TDA
Entender el funcionamiento de los TDA Lineales/No lineales
Entender bien el funcionamiento y uso de la STL
Seguir la materia al dia y hacer retos y prácticas

Herramientas C++

Implementación y uso de **clases**
Gestión de **memoria dinámica**
Uso básico de ficheros
Recursividad básica

Herramientas de desarrollo

Compilación separada
Uso básico de g++
Uso básico de make


METODOLOGIAS DE LA PROGRAMACION
TECNICAS DE DISEÑO
VALIDACION
CONTROL
PROBLEMA REAL
MODELIZACIÓN
ALGORITMO
DATOS
T.D.A.
DOMINIO
ESPECIFICACION
IMPLEMENTACION
PROGRAMA
FIABILIDAD
Resoluble con un ordenador?
ABSTRACCION
EFICIENCIA
LENGUAJE DE PROGRAMACIÓN

Problemas, programas, algoritmos y estructuras de datos

- **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.

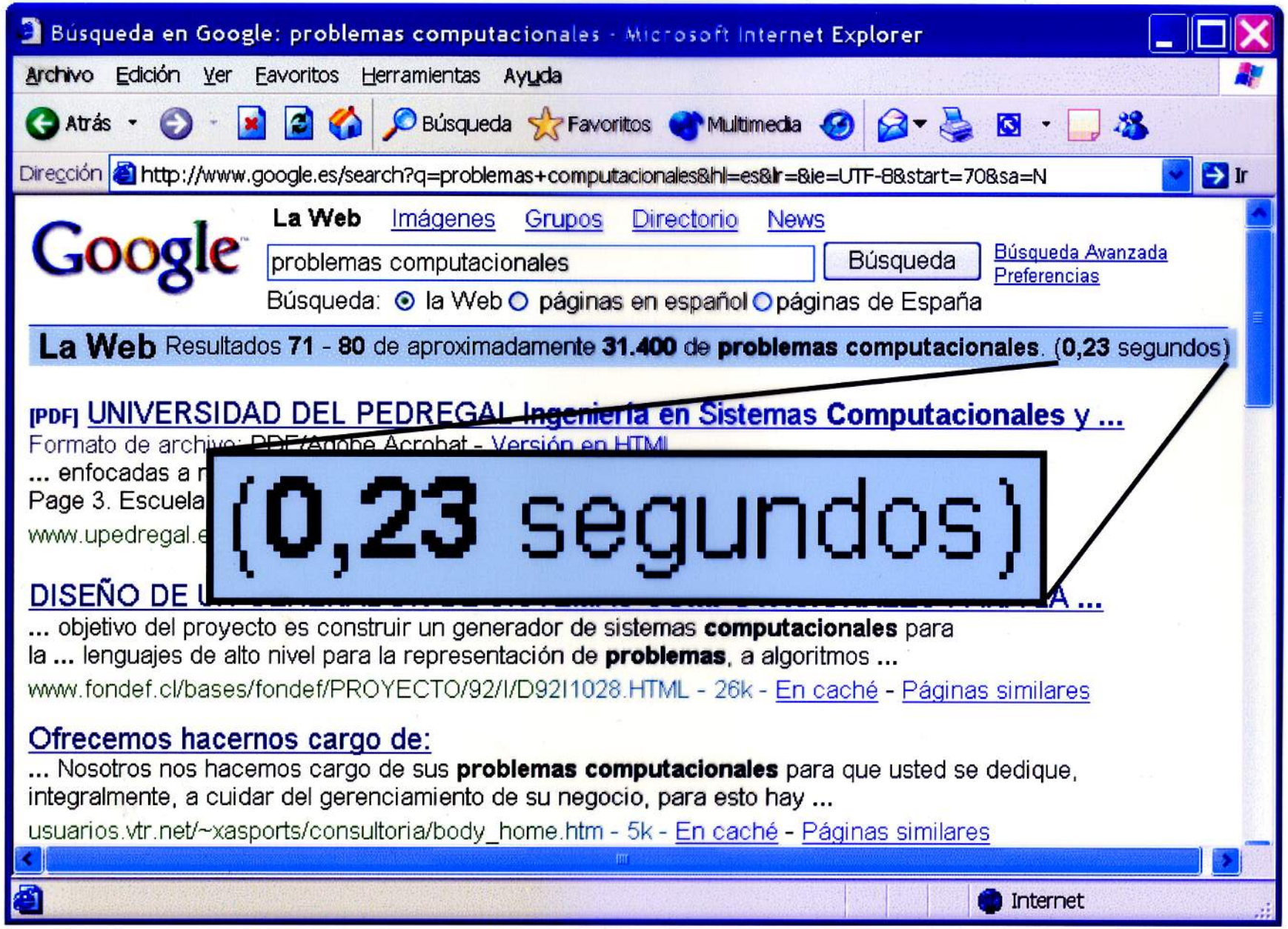
Resolución de problemas

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

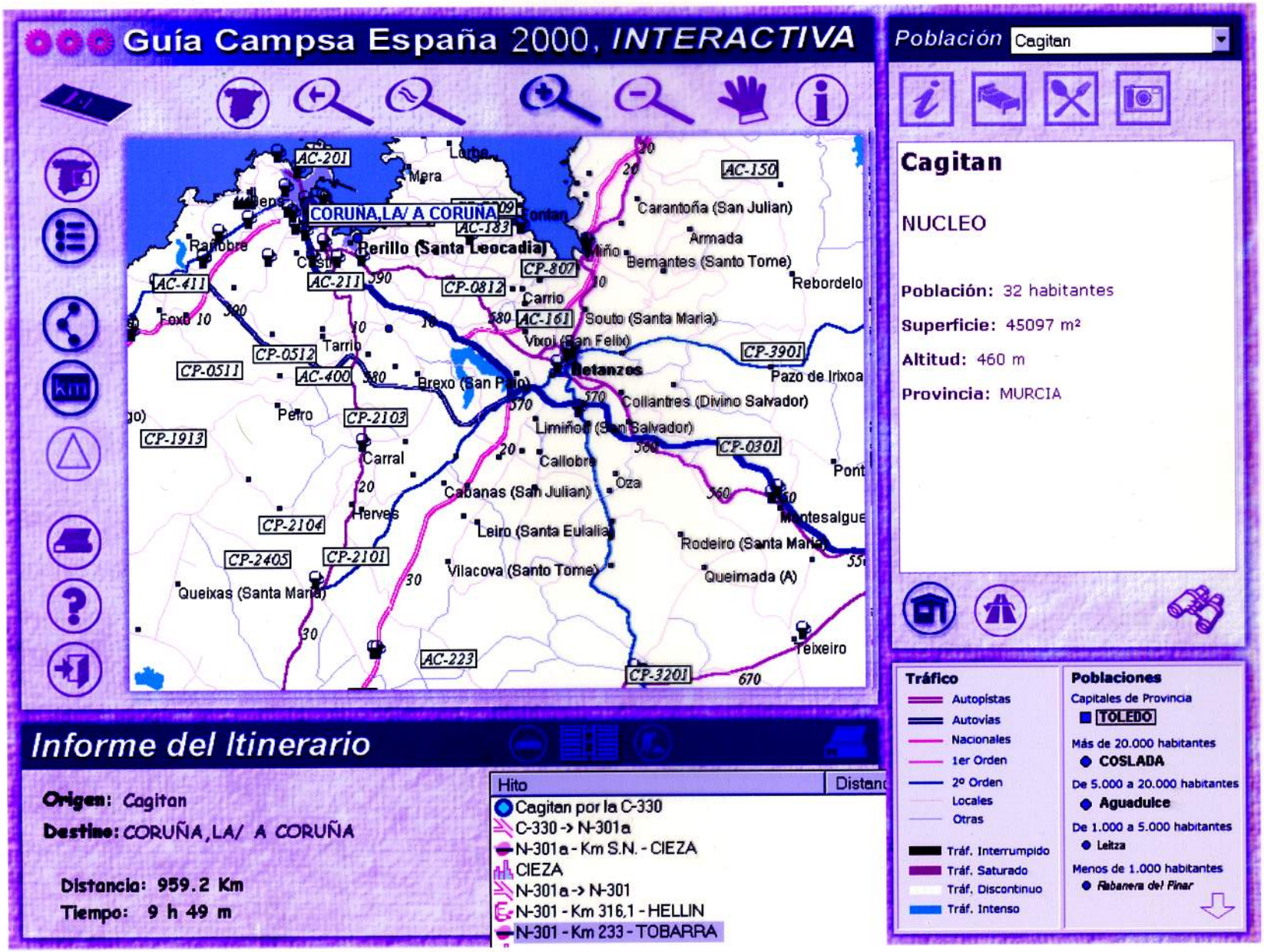
Resolución de problemas

MÉTODO CIENTÍFICO	INFORMÁTICO
1.Observación. ↔	1. **Análisis** del problema
2.Hipótesis. ↔	2. **Diseño** del programa (alg. y estr.)
3.Experimentación. ↔	3. **Implementación** (programación)
4.Verificación. ↔	4. **Verificación** y pruebas

Buscador de Internet

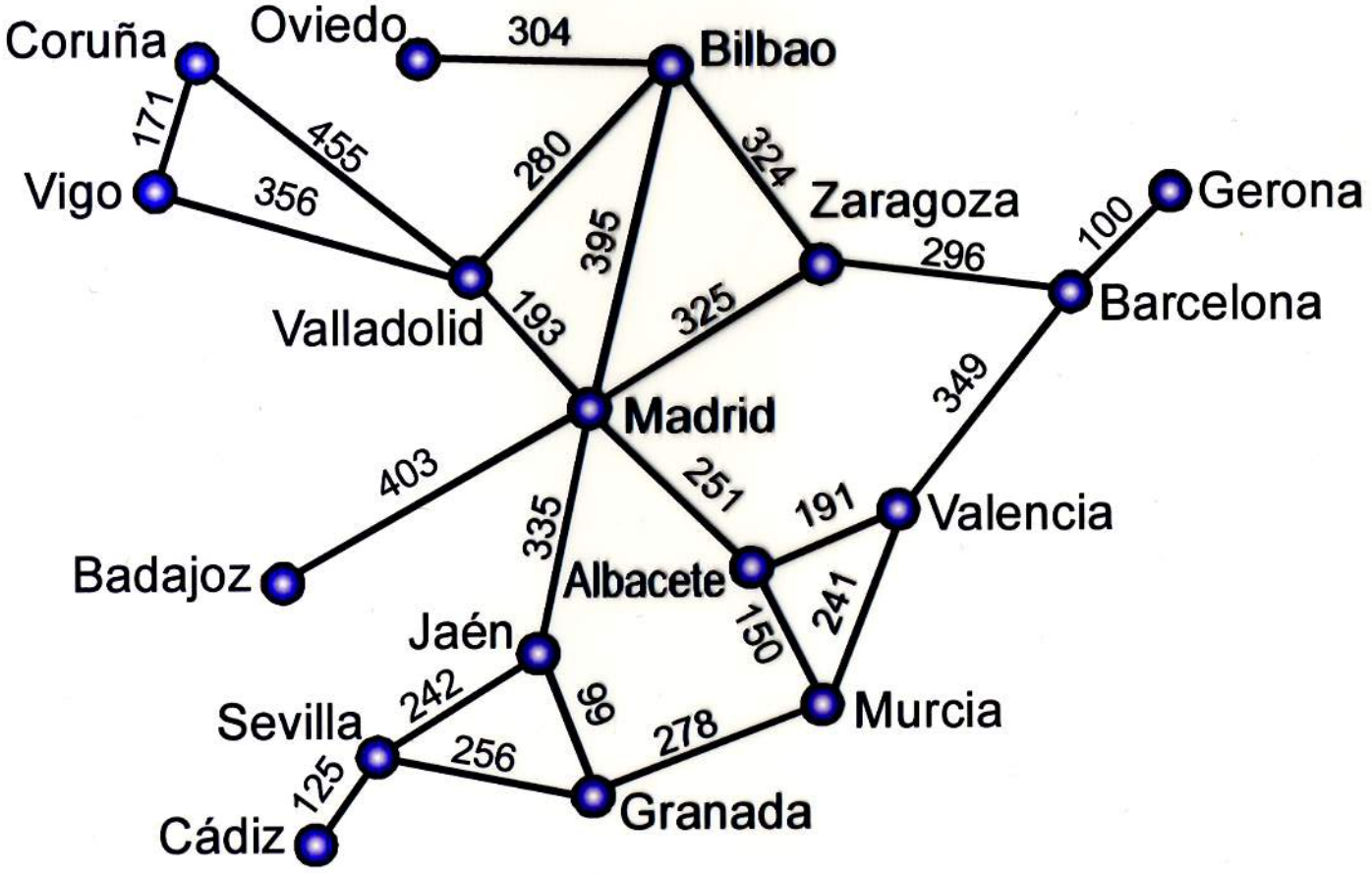


Planificador de rutas



Planificador de rutas

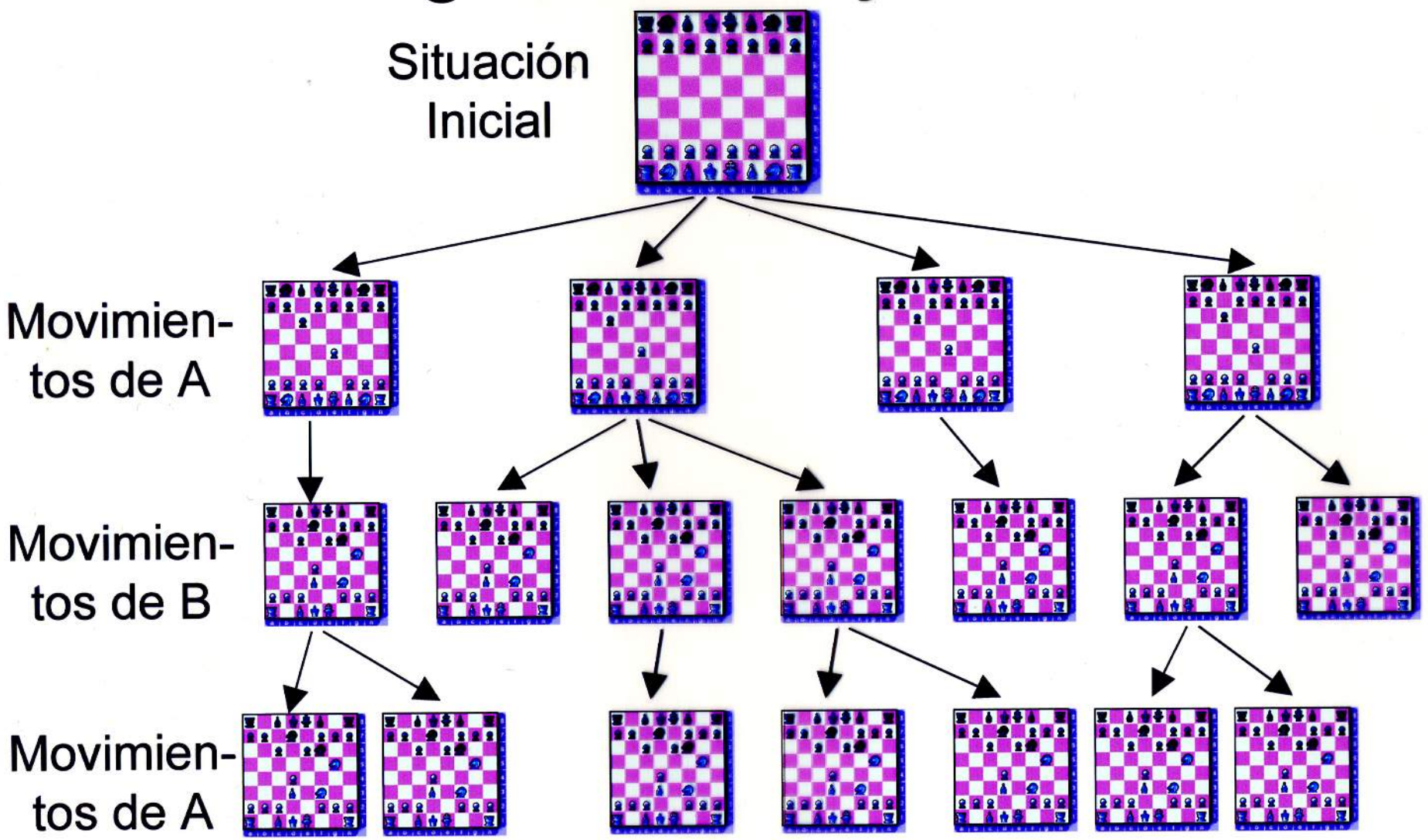
- Representación mediante un **grafo**:
  - Lugares = nodos.
  - Carreteras = arcos entre nodos.



Jugador de Ajedrez

- 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!!

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;
};

class Elipse {
  protected:
    double Fcx, Fcy;
    double Frx, Fry, Fang;
    void FsetXY (int x1, int y1, int x2, int y2);
  public:
    Elipse (int x1, int y1, int x2, int y2);
    Elipse * Clonar (void);
    void Pinta (IplImage *image, int color= 0, int ancho= -1);
};
```

Una clase es un Tipo Abstracto de Datos

Encapsulación de datos y operaciones

Los datos son privados

Las operaciones son públicas

Lenguajes orientados a objetos

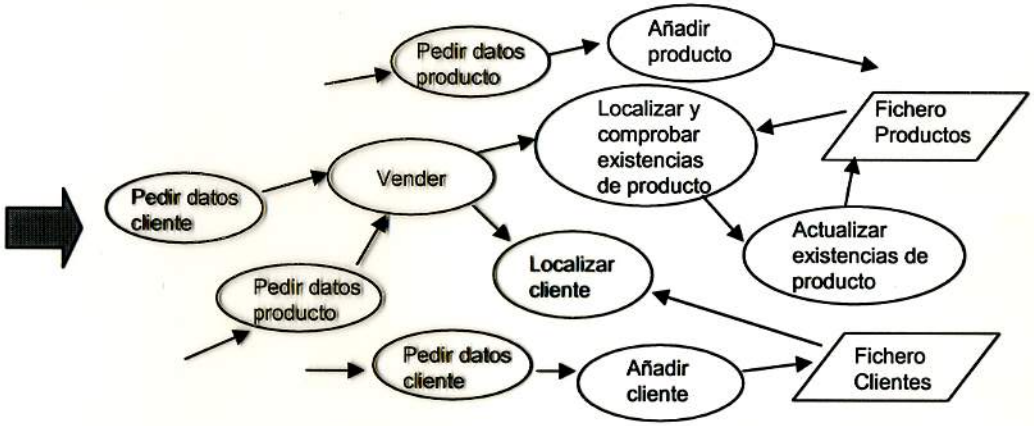
```
// Implementación

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;
}
```

Separación interface/ implementación

PDO y la simulación de la realidad

Metodologías tradicionales

Sistema real



Visión de flujos y procesos

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

Código

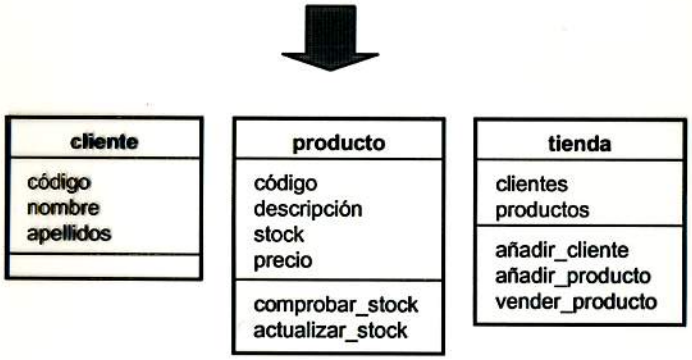
Metodología Orientada a Objetos

Sistema real

Entidades

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

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

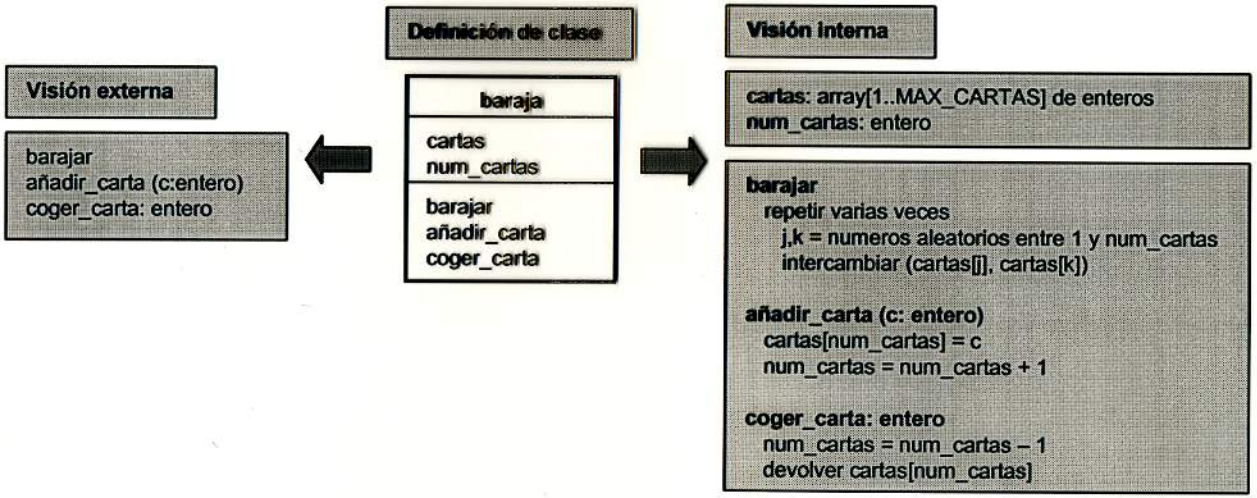
Código



Visión de objetos

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.