Unidad 1

IDEAS GENERALES SOBRE PROGRAMACIÓN

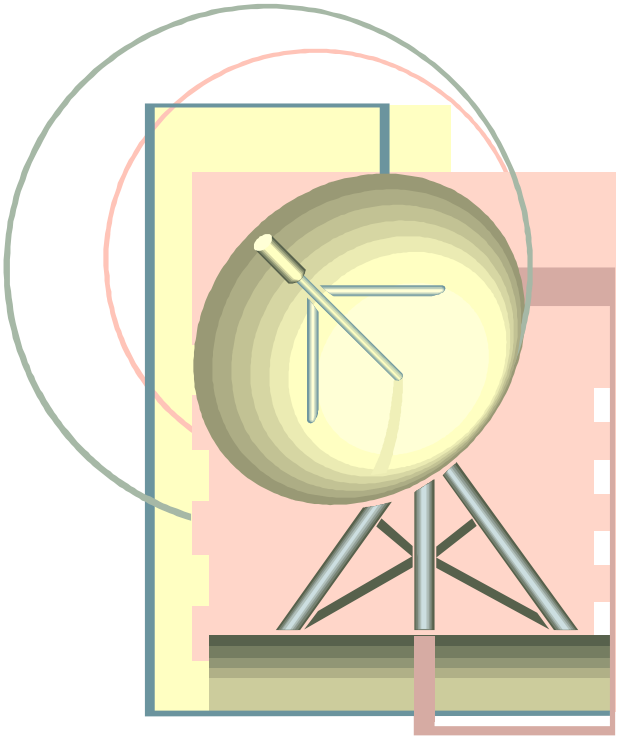
Objetivos

- Identificar los elementos básicos de un Sistema Informático.
- Conocer algunos paradigmas de programación.
- Conocer distintas metodologías de programación.
- Conocer la evolución y características de los lenguajes de programación.
- Clasificar los distintos tipos de lenguajes de programación.
- Reconocer diferencias simples entre distintos lenguajes de programación.
- Conocer los aspectos fundamentales del proceso de realización de un programa.
- Empezando a programar:
  - Entender la dificultad de expresar órdenes de forma sencilla y concreta.
  - **Valorar la importancia de realizar un buen análisis como paso previo al desarrollo de una aplicación de calidad.**
  - Conocer y valorar la importancia del control de errores.

Contenidos

- Introducción.
- ¿Qué significa programar?.
- Paradigmas de programación.
- Fases de la programación.
- Lenguajes de programación.
  - Lenguaje Máquina.
  - Lenguaje Ensamblador.
  - Lenguajes Compilados.
  - Lenguajes Interpretados.
- Fases en la creación de programas.
- Verificación, depuración y prueba de programas.
  - Errores.
  - Aprender a diseñar programas correctos
    - Asertos.
    - Precondiciones y Postcondiciones.
  - Recorrido e inspecciones de diseño y código.
  - Prueba de programas.
- Unas palabras sobre calidad del software.
- Entornos de desarrollo.
  - Como introducir, corregir y ejecutar un programa.

Introducción



- Los ordenadores están totalmente integrados en la vida diaria:
  - Usar un cajero
  - Pagar con tarjeta
  - Uso del teléfono
  - Móviles
  - Control de semáforos
  - Uso de quirófanos
  - Uso de satélites

→ **ERA DE LA INFORMACIÓN Y LA COMUNICACIÓN**

Un sistema informático es:

- El **conjunto de recursos necesarios para la elaboración y el uso de aplicaciones informáticas**

- Elementos:

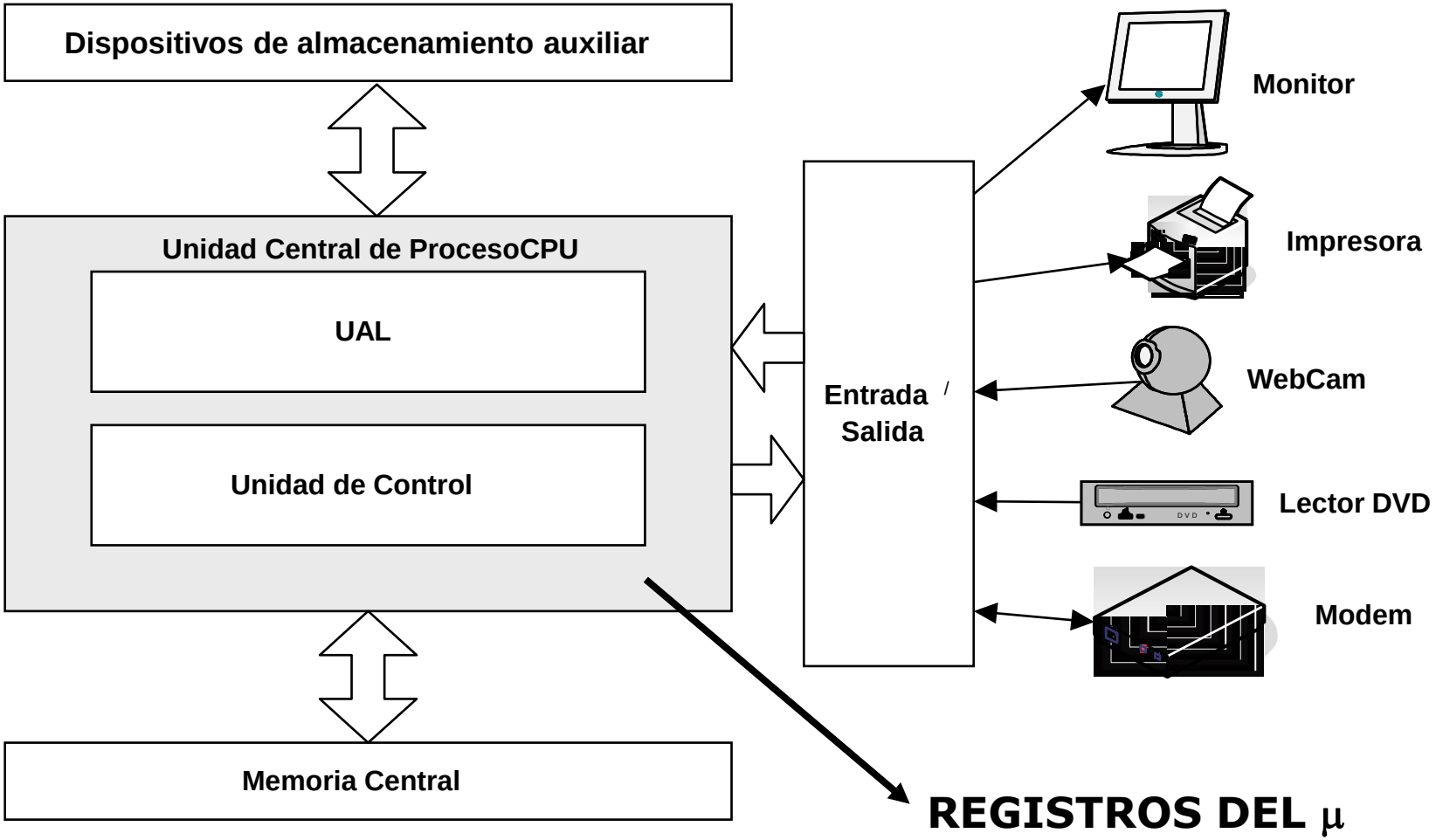
  Hardware, software y personal informático

- Interfaz

  Vía de comunicación entre elementos

USUARIO ⟷ **CAJERO AUTOMÁTICO**

Componentes del HARDWARE



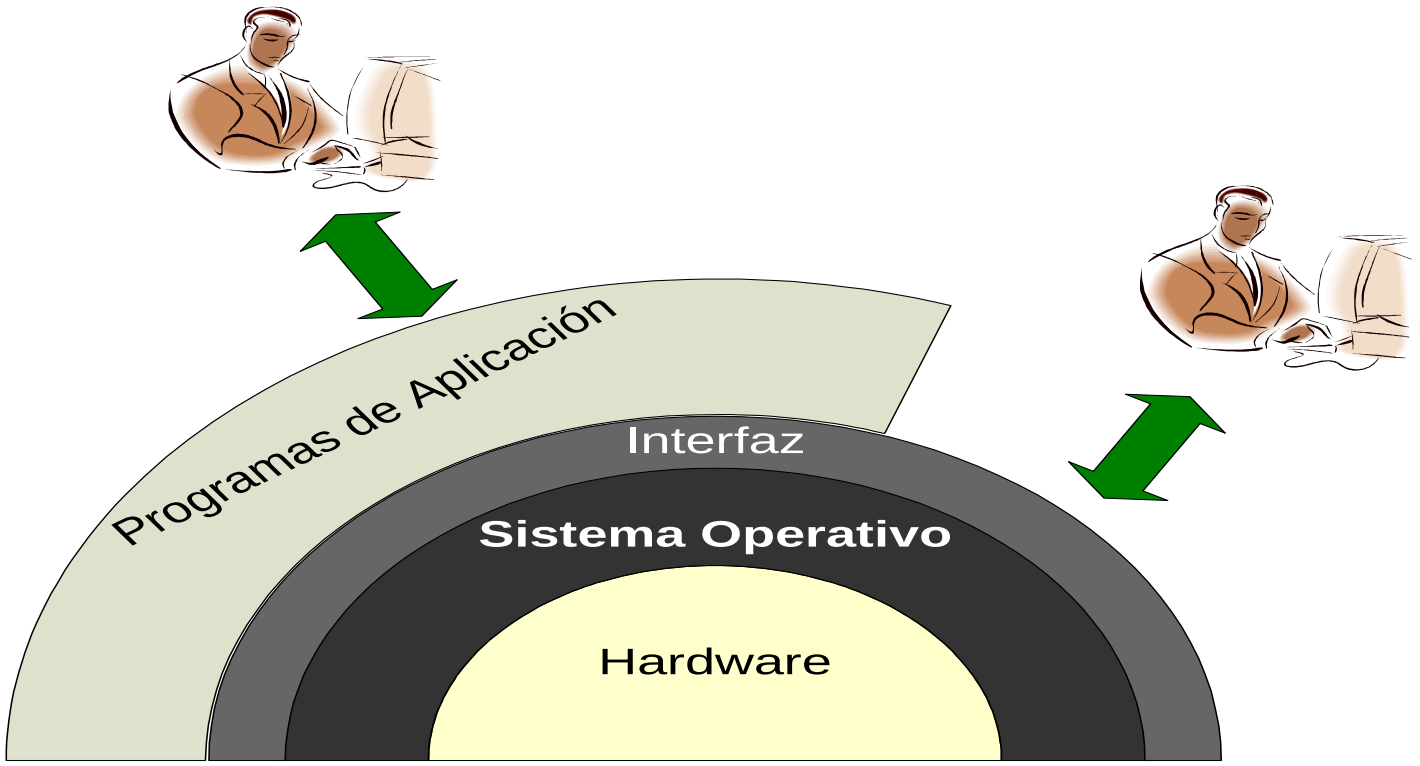
SOFTWARE

- Software Básico (Sistemas Operativos)
- Software de Aplicación
  - Editores de Texto
  - Hojas de Cálculo
  - Bases de datos
  - COMPILADORES
  - Otras aplicaciones (IDE'S)

ELEMENTO HUMANO

- Usuarios
- Personal Informático
  - Personal de Dirección
  - Analistas
  - **Programadores**
  - Operadores

Sistema informático



PROGRAMAR es:

- Planificar y concretar la secuencia de órdenes concisas para la realización de una tarea.

- Un **programa** informático es

  el conjunto de instrucciones, escritas en un **lenguaje particular**, que se da al ordenador para que realice una determinada función.

Paradigmas de programación

Un paradigma es un *modelo* básico, *esquema formal*, *marco teórico,* para la realización de algo.

Un paradigma de programación es un modelo básico para el diseño e implementación de programas.

Algunos paradigmas

- **Programación lógica**: La tarea a realizar se expresa utilizando lógica formal matemática, entorno conversacional. Ej. El lenguaje *Prolog* lo implementa
- **Programación heurística**: "buena lógica", probabilidades. Ej. Inteligencia Artificial
- Paradigmas procedimentales: órdenes paso a paso.
  - Imperativo. Ej. El lenguaje C lo implementa
  - Orientado a Objetos. Ej. C++, Java, Eiffel lo implementan
- Computación en la nube/ Programar para la nube ("Cloud computing")

Metodología

"Metodología es un modo sistemático de producir software"

Toda metodología tiene al menos las siguientes características:

- Define como se divide un proyecto en fases y las tareas a realizar en cada una.
- Para cada una de las fases está especificado cuáles son las entradas que reciben y las salidas que producen.
- Tienen alguna forma de gestionar el proyecto.

Podemos por tanto dar la siguiente definición: 

Metodología

"Conjunto de reglas, procedimientos, técnicas y herramientas para la creación de software"

A veces distintas metodologías pueden obedecer a un mismo paradigma

- De programación: met.Estructurada, met. OO
- De análisis y diseño: Yourdon, Métrica

Parad. Procedimental

(Modelo español) Est, OO

Las metodologías (nivel teórico) son implementas por

Los lenguajes de programación.

Ej. MOO

Ej, C++, JAVA

Lenguajes de Programación

Clasificación:

- Nivel
  - Alto, medio, bajo
- Propósito
  - general, específico, de sistemas…
- Evolución histórica
  - 1ª,2ª,3ª, …última generación
- Forma de ejecutarse
  - Compilados, interpretados,…
- Lugar de ejecución
  - Lenguaje de Servidor, PHP, PERL
  - Lenguaje de Cliente, HTML, JavaScript

La mayoría de los lenguajes no pueden ser clasificados en una sola categoría

Lenguajes de Programación

"Es una notación para escribir algoritmos que se ejecutarán en el ordenador"

Nivel

El nivel de un lenguaje hace referencia a **su proximidad al lenguaje natural (LN):**

- ✓ más nivel cuanto más cercanos están al **LN**
- ✓ menos nivel cuando más cerca están del lenguaje de la máquina

Tipos

- Lenguajes de Bajo nivel: Ej. Ensamblador
- Lenguajes de Medio nivel: C
- Lenguajes de Alto nivel
  - Estructurados: C++, Pascal,
  - Orientados a objetos: C++, Java
  - Orientados a eventos: Visual Basic
  - Orientada a aspectos, reflexivos,…

Lenguajes de Programación

- Lenguajes de Bajo nivel
  - ✓ **Lenguaje máquina** (El lenguaje de más **bajo nivel) :** es el que la máquina entiende, basado en el sistema binario, se organiza como secuencias de 0 y 1.
  - ✓ **Lenguajes ensambladores.** Consisten principalmente en nemotécnicos de las órdenes del lenguaje máquina.
- Lenguajes de Alto nivel
  - ✓ **Interpretados**
  - ✓ **compilados**

  Son más fáciles de aprender y permiten despreocuparse de la arquitectura del ordenador. Ejemplos son: BASIC, PASCAL, FORTRAN, COBOL, JAVA, C/C++ …

L.P. características

- **Lenguaje máquina**
  - **Programas totalmente dependientes de la máquina.**
  - **Programas muy difíciles de leer e interpretar**, por tanto, también difíciles de modificar.
  - **Los programadores de la época se veían obligados a recordar largas secuencias de ceros y unos**, correspondientes a las operaciones o instrucciones.
  - **Los programadores se encargan de introducir los códigos binarios.**
  - **Manejan directamente recursos del sistema**: memoria, puertos,..
- **Lenguaje ensamblador**
  - **Dependientes de la máquina.**
  - **"Más fáciles" de leer e interpretar.**
  - **Aparece el ensamblador (traductor)**

"Menos" propensos a errores

OPERACIONES	LENGUAJE MÁQUINA	L. ENSAMBLADOR
SUMAR	00101101	ADD
RESTAR	00010011	SUB
MOVER	00111010	MOV

Lenguajes Compilados

Los programas escritos en un LP cualquiera deben traducirse (excepto el código máquina) para que sean entendibles por el ordenador.

Los **Lenguajes compilados:**

- El **compilador** traduce el **programa entero** y lo monta (enlaza, linka), generando un **programa ejecutable**, que puede ser ejecutado en distintos entornos, sin necesidad de que el **compilador** esté presente. Más rápidos
- **Mayor independencia de la máquina.**
- El programador no necesita conocer el hardware específico de la máquina
- Programas **más fáciles de leer y modificar**

Menos propensos a errores

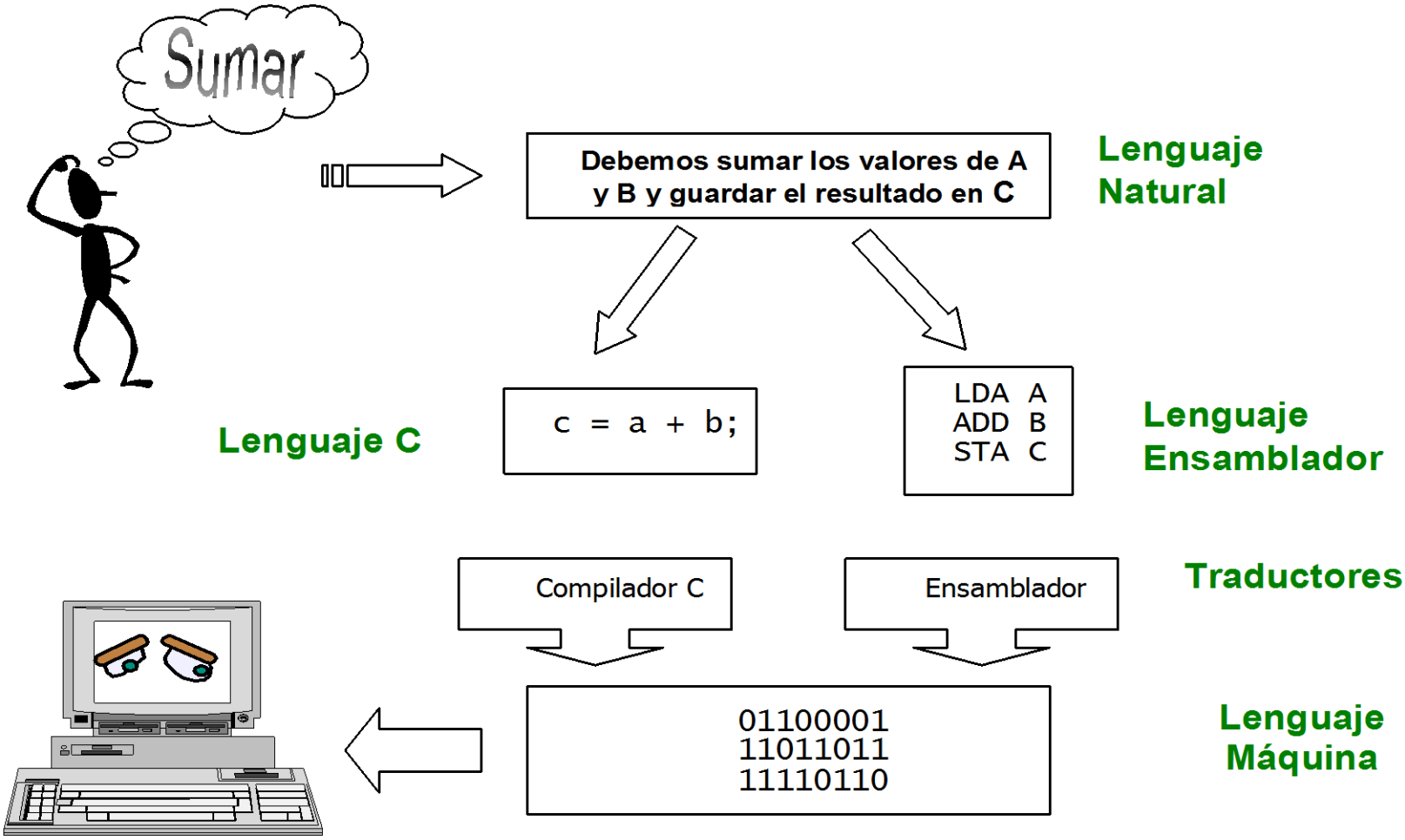
Lenguajes Interpretados

Los Lenguajes interpretados:

- **El intérprete** realiza la traducción **instrucción a instrucción** a la vez que se ejecuta el programa (son más lentos). El intérprete siempre debe estar presente mientras se ejecuta el programa.
- Más portables.
- Los programas que producen ocupan menos memoria que los producidos por los compiladores (no guardan una versión completa del programa traducido a código máquina).
- Se usan especialmente en depuradores y verificadores de código.

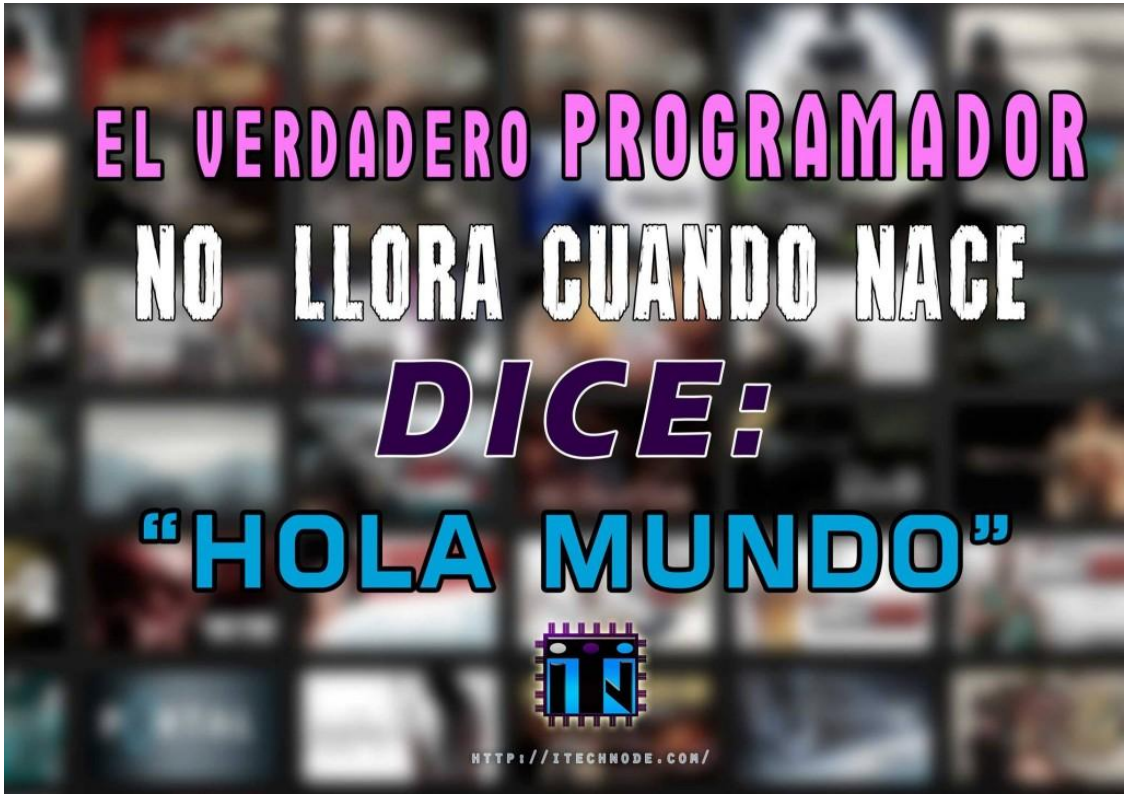
- **Lenguajes pseudo-interpretados o pseudo-compilados:** Ej. **Java**

Uso de los lenguajes de programación



Are you a real programmer?, Hello World!

How do you know that?



Vamos a ver el "Hola Mundo" en distintos lenguajes

Uso de los lenguajes de programación

mover el valor hexadecimal 61 (97 decimal) al registro 'AL'

- **Lenguaje máquina**

la instrucción de máquina

10110000 01100001

- **Lenguaje ensamblador**

microprocesadores x86

mov AL, 0x61

Registros usados en el programa ensamblador del ejemplo:

AX registro acumulador (AH byte alto AL byte bajo)

DS Puntero al Segmento de datos

DX registro de Datos de propósito general

IR: registro de instrucciones

Interrupciones:

int 21h *//para escribir en pantalla*

```
;"Hola mundo" para X86 bajo DOS
.model small
.stack
.data
   Cadena1 DB 'Hola Mundo.$'
.code
programa:
  MOV AX, @data
  MOV DS, AX
  MOV DX, offset Cadena1
  MOV AH, 9
  INT 21h
end programa
```

```
/*"Hola mundo" en C*/
#include <stdio.h>
void main (void)
{
   printf( "Hola Mundo");
}
```

Ejemplo en algunos LOO

```
//Hola mundo en C++
/* La siguiente cabecera permite usar los objetos de stdout y stdin: cout(<<)
 y cin(>>) */

#include <iostream>
 using namespace std;

int main()
{
    cout << "Hola mundo" << endl;
    return 0;
}
```

```
//Hola mundo en Java

public class Hello
{
  public static void main ()
{
     System.out.println("Hola mundo");
  }
}
```

Ejemplo en algunos LOO

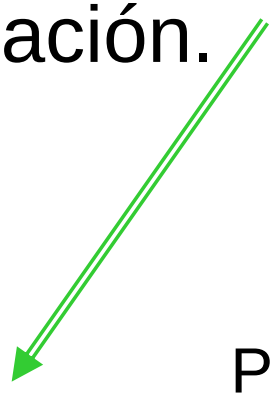
```
//Hola Mundo en C#  (# sostenido, sharp en inglés)

using System;
  class MainClass
{
    public static void Main()
    {
        System.Console.WriteLine("¡Hola, mundo!");
    }
}
```

```
-- Hola Mundo en Eiffel
class HOLA_MUNDO
 create
    make
 feature
    make is
        do
            io.put_string("%nHola mundo%N")
        end
 end  -- HOLA_MUNDO
}
```

Ejercicio: Realiza una tabla con las ventajas e inconvenientes de cada uno de estos lenguajes (C++, C#, Eiffel, Java)

Recordando

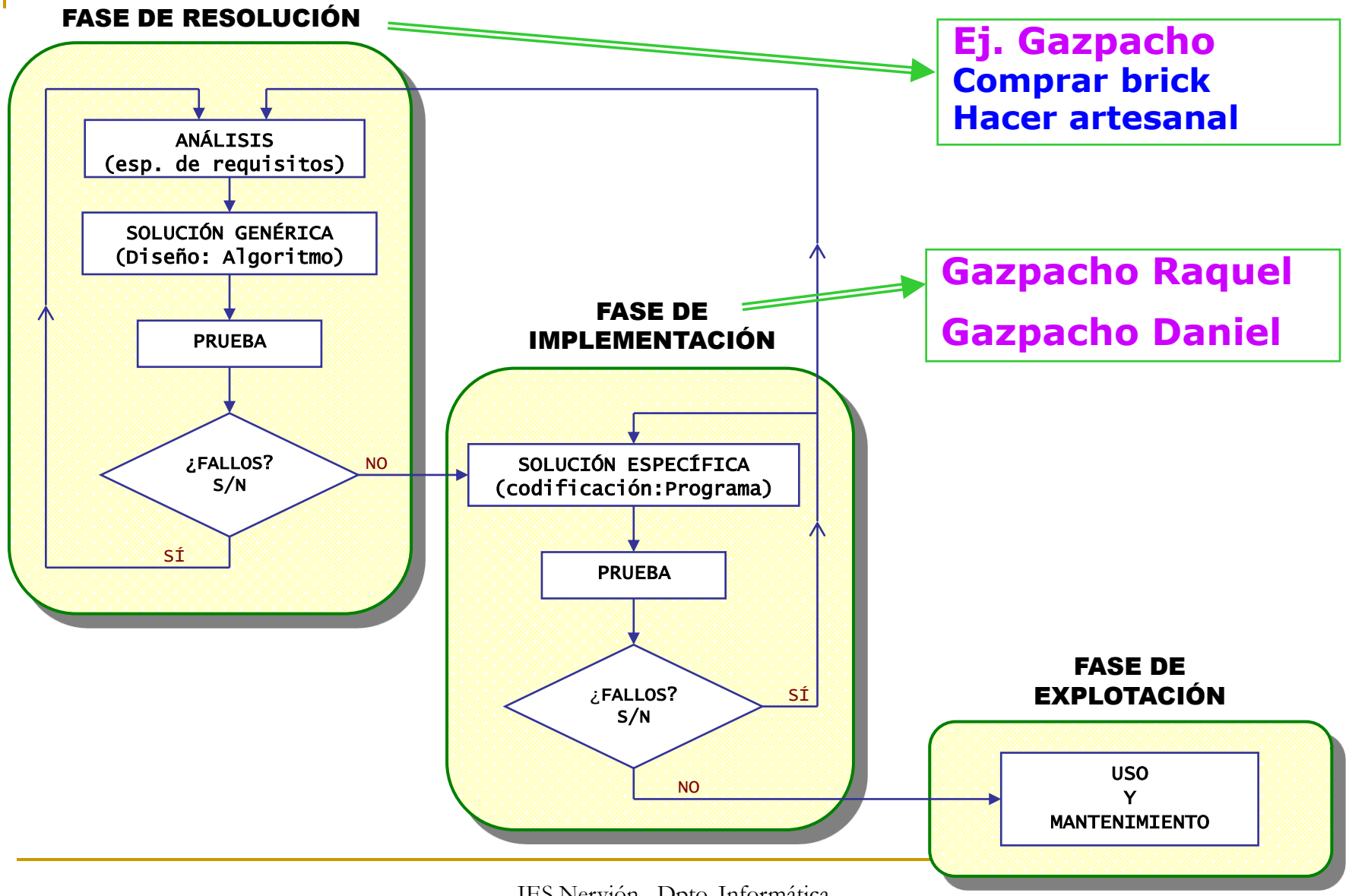
- **PROGRAMAR es**: Planificar una tarea y concretar la secuencia de órdenes que harán posible su realización.



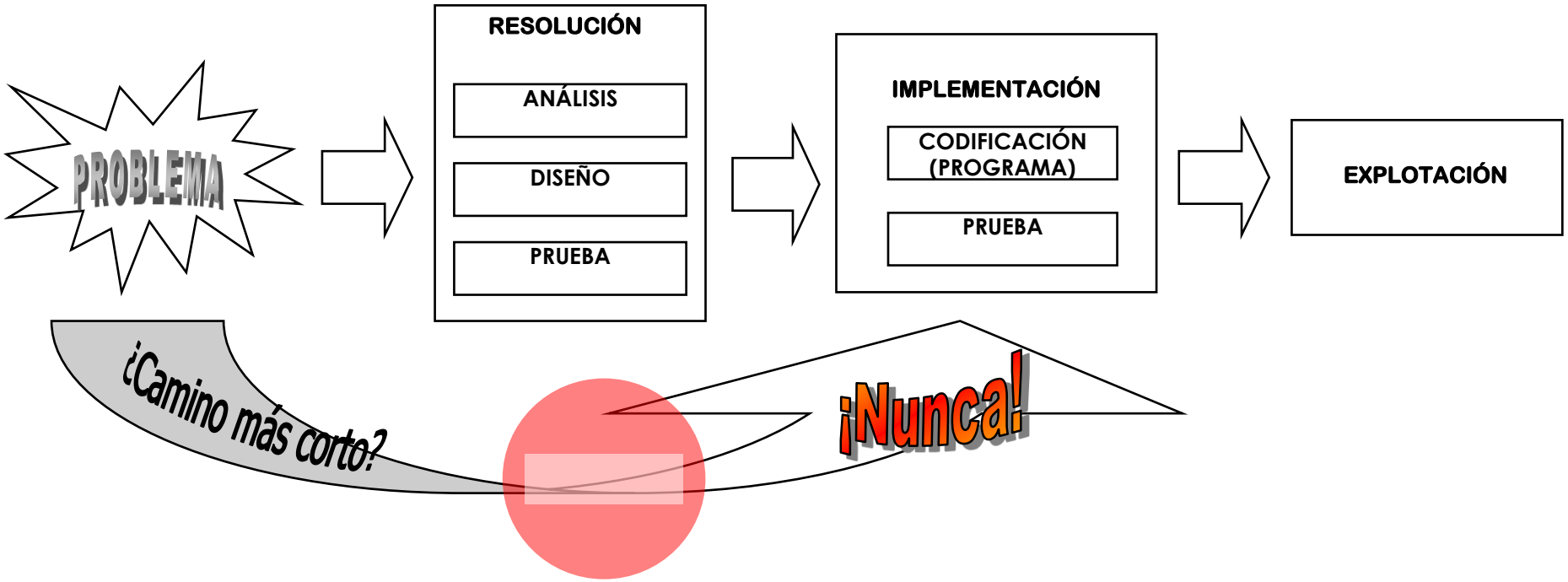
PROGRAMA

Pero, ¿Qué se hace desde que deseamos informatizar algo hasta obtener el programa

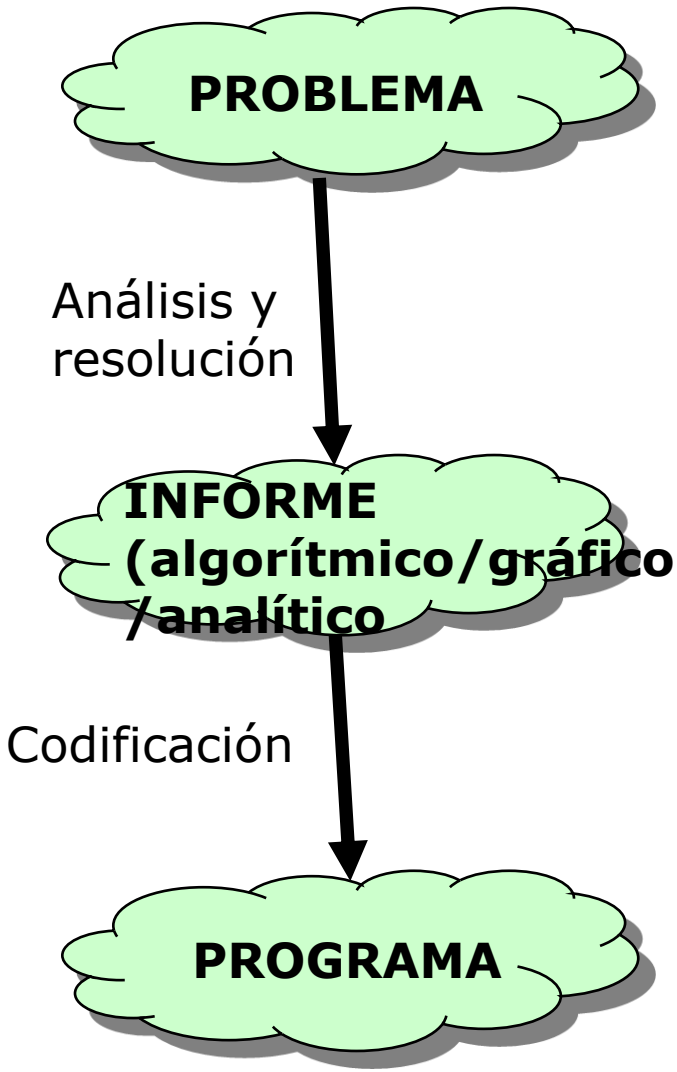
FASES PROGRAMACIÓN (similar a Ciclo vida software)



Camino equivocado



El método general para programar puede resumirse como:



1. **PROBLEMA**: actividad a informatizar.

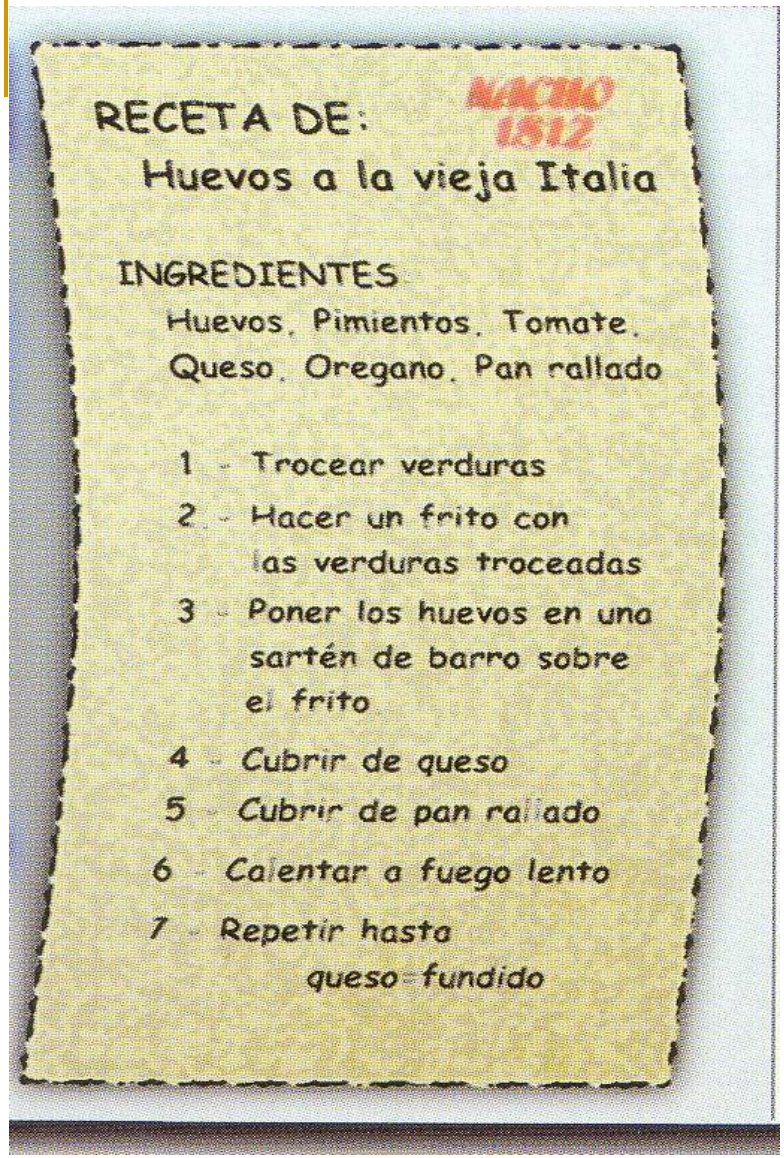
2. **INFORME**: la actividad se analiza en busca de la forma de resolución. El resultado se plasma en un informe que contiene:
   a. Descripción de objetos y tareas que realizan para conseguir los objetivos planteados
   b. Relación entre los objetos que aparecen
   c. Los recursos y elementos necesarios (entradas, salidas, suposiciones,...)
   d. **El algoritmo** (sucesión de instrucciones precisas que lleva a una solución)

3. **PROGRAMA**: el algoritmo, traducido a un lenguaje de programación específico, se convierte en un programa que el ordenador puede ejecutar

RECORDEMOS:
¿Qué es un algoritmo?

Un método para resolver un problema mediante una serie de pasos precisos, definidos y finitos

- ✓ **precisos**: indicar el orden de ejecución paso a paso
- ✓ **definidos**: si la secuencia se realiza n veces se obtiene identico resultado
- ✓ **finitos**: tiene un número determinado de pasos, es decir, termina en algún momento.



Una receta de cocina es un algoritmo para realizar un plato.

Analicemos esta receta ¿podría realizarla correctamente cualquier persona ?, un japonés, una persona de la selva amazónica, un niñ@ español de 4 años.

Universo del discurso (contexto)

Ver ejemplo tortilla pag. 10

Importancia de entender los requisitos

Algoritmo

leche=1
Si hay huevos
    leche=6

Algoritmo: Cambiar rueda al coche

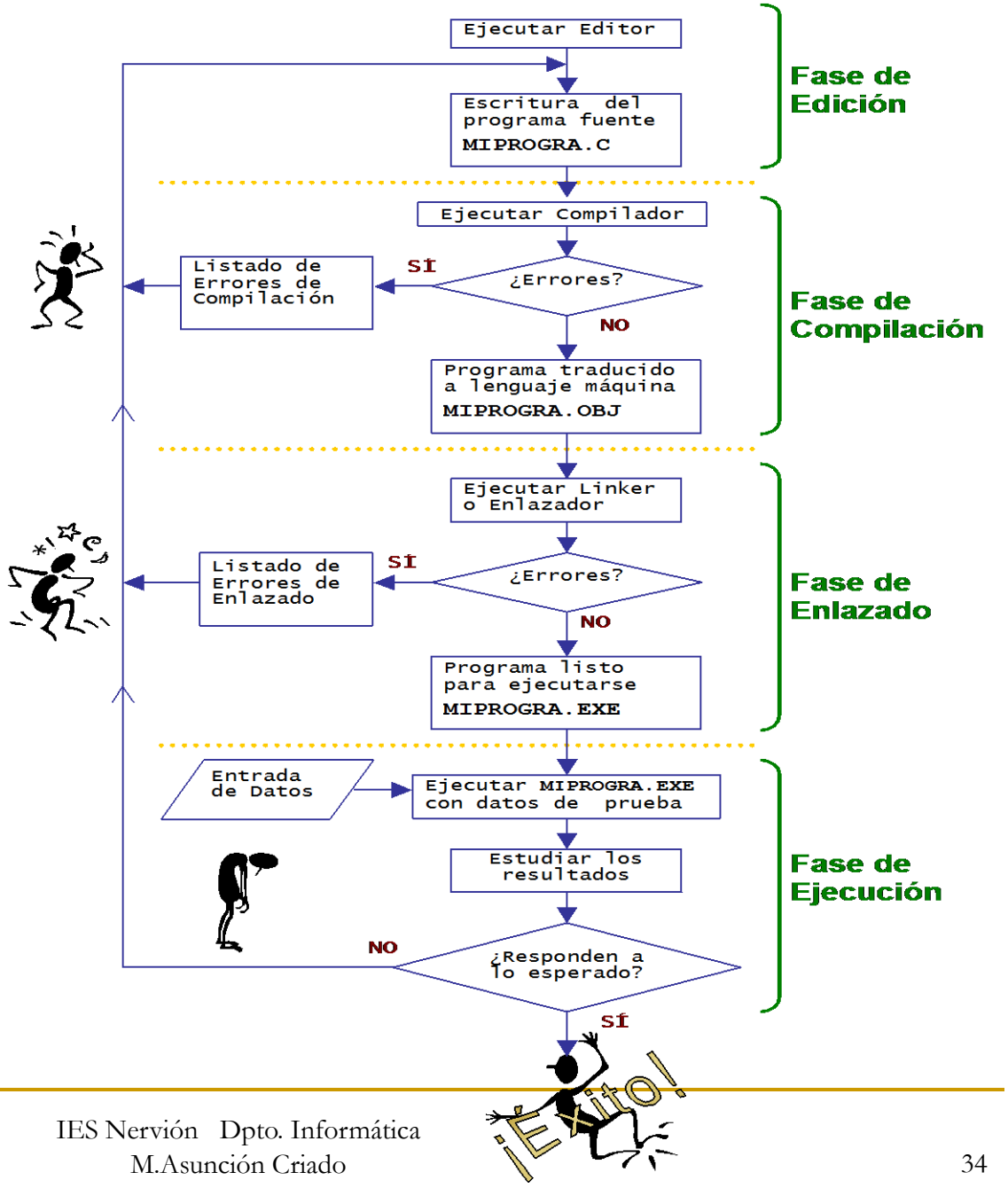
Suponemos que:
Disponemos de todos los utensilios: coche, rueda y gato
Sabemos: aflojar tornillo, apretar tornillo,….

Inicio
Inmovilizar el coche //No se explica cómo se hace y puede hacerlo otra persona
Colocar el gato //No se explica cómo se hace
Repetir
Levantar el coche accionando manivela del gato
Mientras que la rueda roce el suelo.
Repetir
Aflojar un tornillo de la rueda
Mientras queden tornillos por aflojar
Quitar los tornillos
Quitar la rueda
Poner rueda de repuesto
Repetir
Apretar un tornillo de la rueda
Mientras queden tornillos por apretar
Repetir
Bajar el coche accionando manivela del gato
Mientras que el gato soporte peso.
Fin

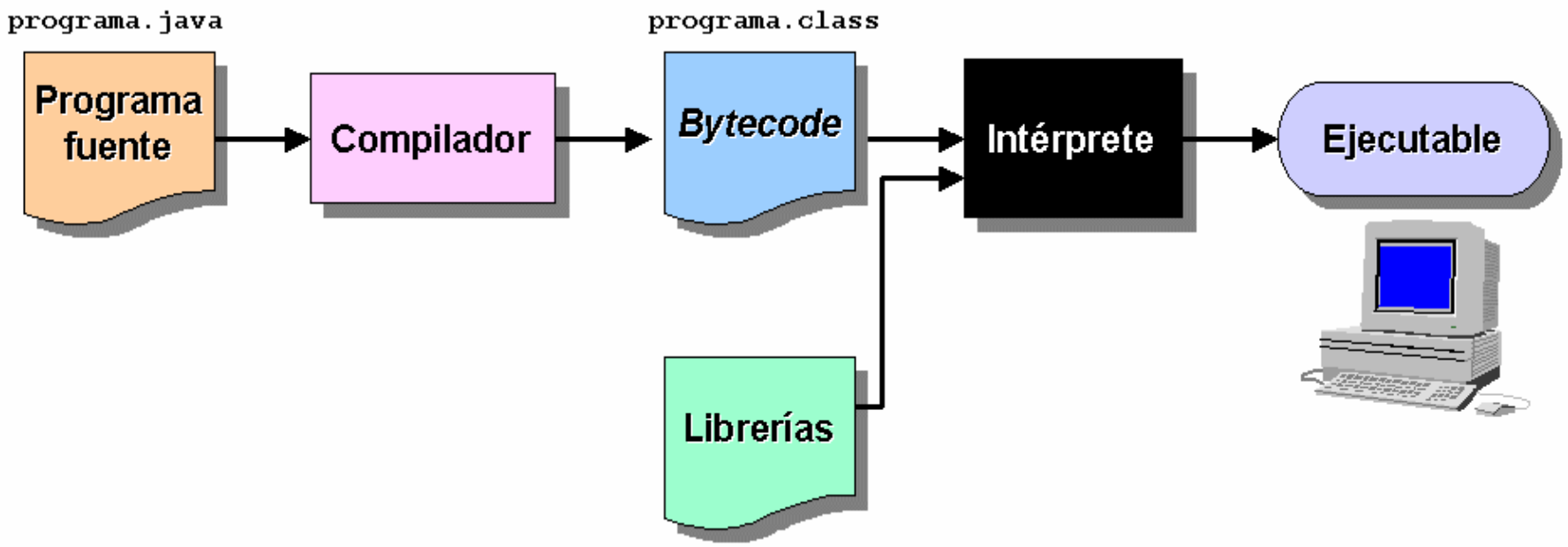
MODULOS → DISEÑO TOP-DOWN

Ejercicio: Realizar un algoritmo sobre un proceso sencillo de la vida diaria.

Fases en la creación de un programa compilado



Fases en la creación de un programa con Java



Verificación, depuración y prueba

Verificar y depurar consiste en:

Prevenir, detectar y corregir posibles errores en los programas utilizando para ello distintos mecanismos.

- Buen diseño, aplicando reglas de calidad y corrección.
- Recorrido e inspección de diseño y código.
- Establecimiento de métodos de depuración.
- Previsión de planes de prueba.
- Establecimiento de datos de prueba (juego de ensayo).

Menos análisis ⇨ más tiempo de Verificación y Depuración
⇨ software más caro

ERRORES

Buscar la robustez en los programas

- **DE ESPECIFICACIÓN Y DISEÑO:**
  - Aparecen durante el análisis o el diseño.
  - Son difíciles de corregir y costosos
- **DE COMPILACIÓN**
  - Lexicográficos: en Java escribir “mian” en lugar de “main”
  - Sintácticos: a = b + (c /)
  - Semánticos: No interrumpen el proceso de compilación
    if (a = b) ⇨usar = en lugar de ==
- **DE ENLAZADO**

  Por ejemplo, librerías no colocadas correctamente
- **DE EJECUCIÓN**

  C = a / b (b no puede valer 0. Debe controlarse).

  Overflow o desbordamiento (tipo de dato incorrecto)

Diseñar programas correctos

- Técnicas de validación y verificación de código
  - Mediante herramientas automáticas
  - Técnicas formales manuales

- Algunos conceptos sobre los que se basan las técnicas anteriores
  - ***Aserto***:
  - ***Precondición***
  - ***Postcondición***
  - Estudio de las ***invariantes de bucle*** (Capítulo 3)

Diseñar programas correctos

- ***Aserto***: Afirmación sobre el estado de un algoritmo en un punto dado.

  Ej. Resta = A-1 puedo asegurar que **Resta** siempre es menor que **A** en **1** unidad.

- ***Precondiciones:*** asertos previos a la ejecución de una instrucción, tarea, bloque de código...

  Ej. Resta debe ser positivo y mayor que cero, por tanto, A siempre será mayor que 1

  O bien,

  Resta debe ser positivo ocero, por tanto, A siempre será mayor o igual a 1

  Ej. Realizar un módulo **BuscarAlumno**

- ***Postcondiciones:*** asertos posteriores a la ejecución de una instrucción, tarea, bloque de código...

  Ej. Tras el Repetir_ hasta de la tortilla se aserta la condición de salida: “La tortilla está cuajada a nuestro gusto

DISEÑO Y DEPURACIÓN DE PROGRAMAS

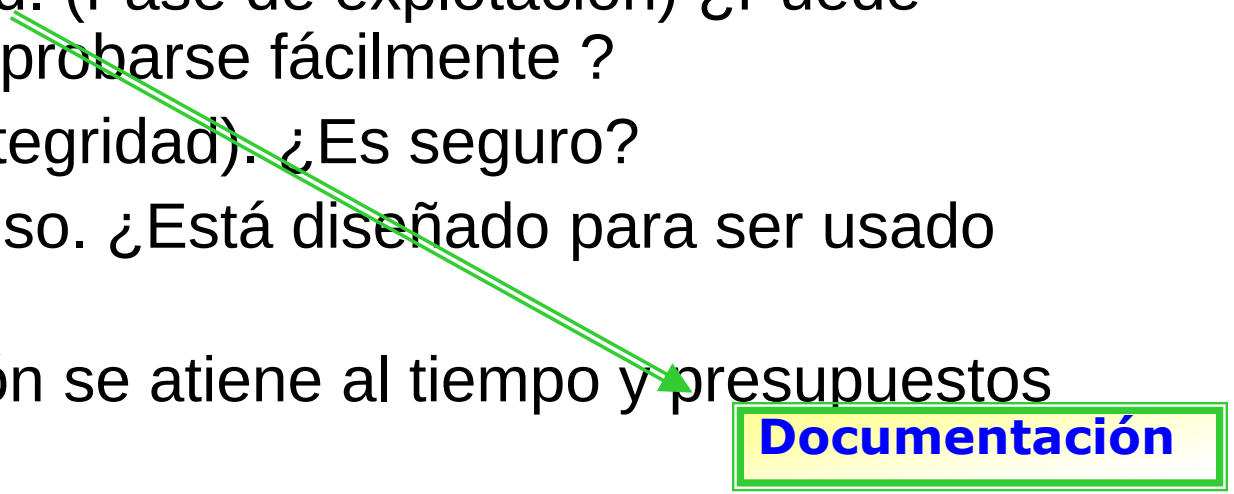
- **Inspección**: Traza o recorrido en el análisis, diseño e implementación, arrastrando valores de juego de ensayo
- **Planes de depuración**: Puntos de parada (breakpoints),visualizaciones intermedias (pintar en pantalla).
- **Prueba**: Cubrimiento de datos y validaciones de entradas

Algunas consideraciones

FASES	ACTIVIDADES DE VERIFICACIÓN Y PRUEBA
Análisis	▪Estudiar muy bien los requisitos. ▪Estudiar los requerimientos de prueba.
Diseño (Sol. General)	▪Realizar programas correctos. ▪Realizar inspecciones de código. ▪Verificar el diseño. ▪Realizar un plan de pruebas.
Codificación	▪Dominar el lenguaje de programación. ▪Realizar inspecciones del código fuente. ▪Añadir sentencias específicas al código para la depuración del programa. ▪Realizar un plan de pruebas. ▪Realizar la verificación del código.
Prueba y validación	▪Realizar la prueba de acuerdo al plan de pruebas establecido. ▪Depurar los segmentos de código que sea necesario. ▪Pruebas unitarias y de integración (Si se realizaron módulos y se depuraron por separado, integrarlos para su prueba conjunta). ▪Ejecutar nuevas pruebas después de las correcciones realizadas. ▪Realizar pruebas de aceptación del producto completo a la entrega.
Mantenimiento	▪Realizar un plan de pruebas para todo el producto cada vez que se haya cambiado o añadido nuevas funcionalidades, o bien se hayan corregido problemas detectados en la fase de explotación.

Características del software de calidad

- El programa funciona
  - Requerimientos: ¿Hace lo que se estableció como requisito?
  - Completitud ¿Lo hace completamente?
  - Corrección. ¿Lo hace bien?.
  - Eficiencia: ¿ Minimiza tiempo y recursos?
- Fiabilidad. ¿Lo hace de forma fiable todo el tiempo?
- Legibilidad y comprensión ¿Fácilmente?
- Modificabilidad. (Fase de explotación) ¿Puede modificarse y probarse fácilmente ?
- Seguridad (Integridad). ¿Es seguro?
- Facilidad de uso. ¿Está diseñado para ser usado fácilmente?
- ¿Su realización se atiene al tiempo y presupuestos establecidos?



“El comienzo de la sabiduría para un ingeniero de Software es reconocer la diferencia entre hacer que un programa funcione y conseguir que lo haga correctamente”

[Pressman, 2003]