ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS

Ing. Gutierrez Milagros

Ing. Ledesma Rodrigo

Ing. Dobler Nicolas

Tecnología de Objetos

El *Modelo Basado en Objetos* se ha convertido en uno de los impulsores de la *Industria del Software* durante la década de los noventa.

A pesar de la reciente profusión de *aplicaciones*, *herramientas y lenguajes* que ***soportan***, ***emplean*** *o* ***se han construido*** *tomando como base el* ***Modelo de Objetos***,

los *conceptos básicos* del mismo se originaron en la década del ´60.

Por ejemplo, el lenguaje SIMULA, uno de los precursores en el concepto de objetos fue desarrollado en 1967.

Modelo de Objetos

- Los conceptos básicos del ***Modelo de Objetos*** se originaron con la definición de lenguajes que incorporan conceptos tales como:
  - ***Encapsulamiento*** ***Modularidad***
  - ***Abstracción*** ***Polimorfismo***
- Si bien **SIMULA** fue el primer lenguaje en incorporar el Modelo de Objetos, lo más importante ha sido el

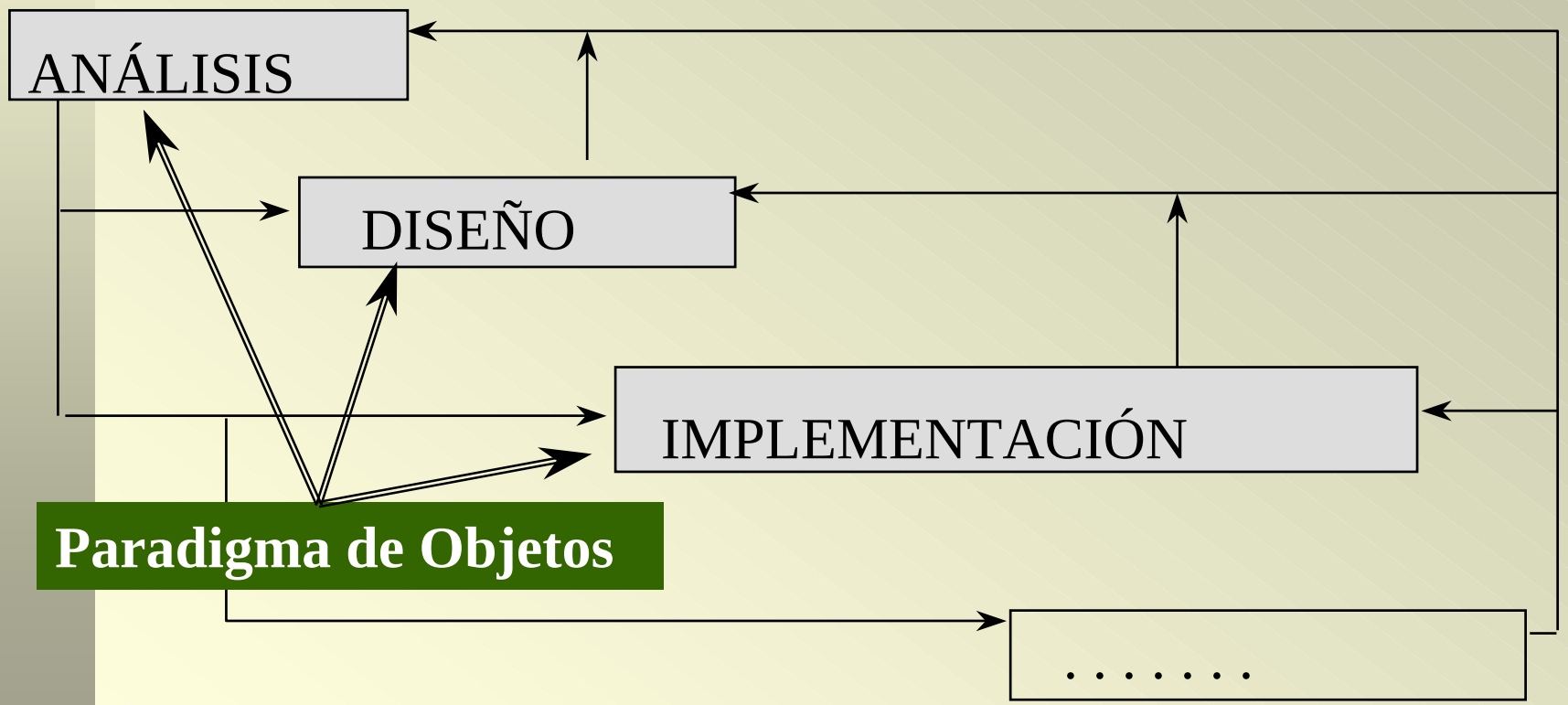
Modelo Conceptual

que ha dejado como resultado.

Modelo de Objetos

- El ***Modelo de Objetos*** ha trascendido los sistemas de información y/o software en general, y se está utilizando como una herramienta de *modelado ingenieril* de varios tipos de sistemas, por ej.:
  - distintos tipos de redes (eléctricas, de flujo, etc.)
  - ingeniería de organizaciones y métodos (re-ingeniería de empresas)
  - ingeniería de productos
  - control de tráficos
- En realidad el Modelo de Objetos se originó a partir de la necesidad de disponer de una herramienta simple para Simular Sistemas (de cualquier tipo).,

Ciclo de Vida en el desarrollo del Software



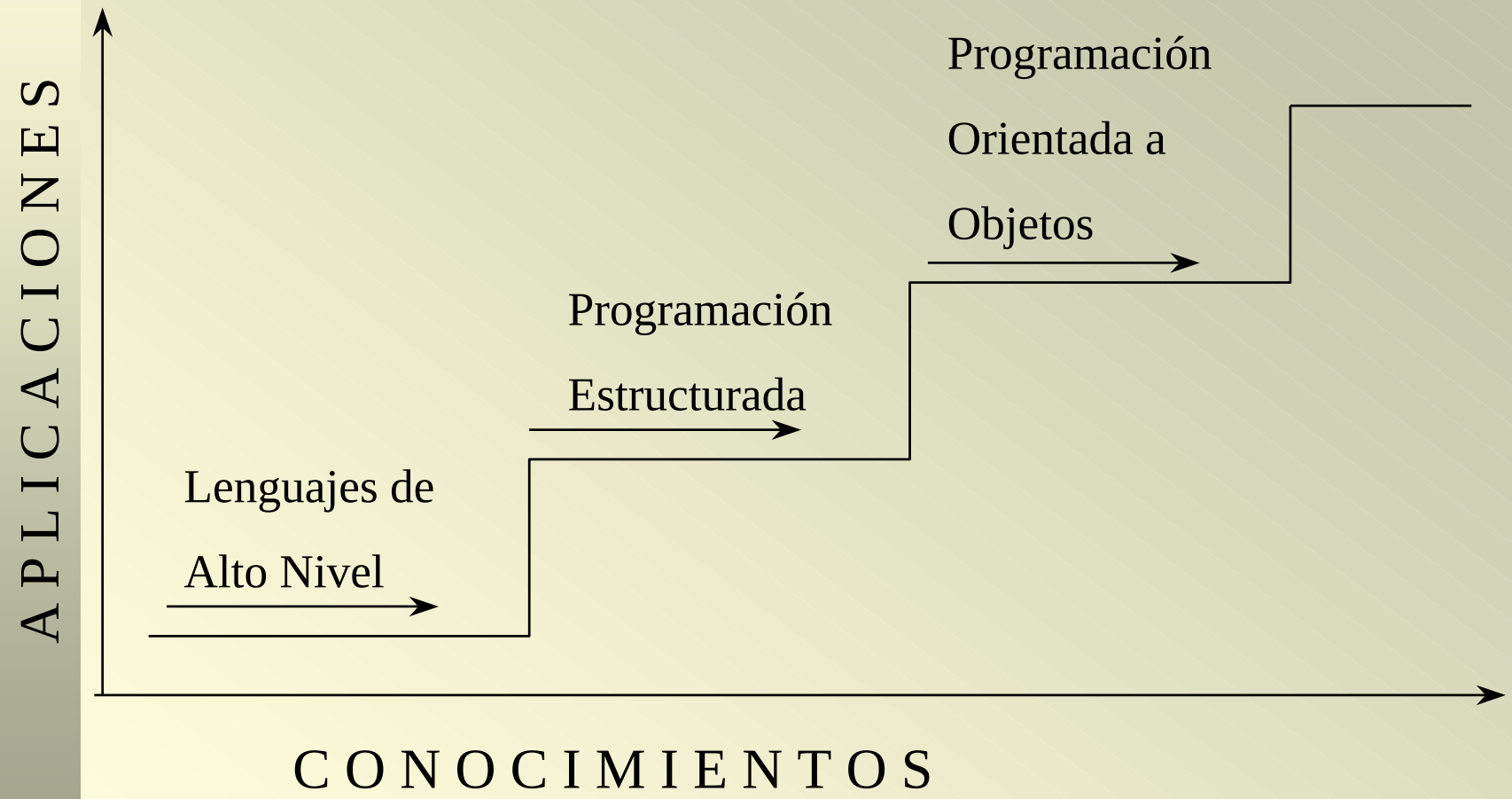
Análisis y Diseño O.O

El ***análisis orientado a objetos*** es un método de análisis que examina los **requisitos** desde las perspectivas de las clases y objetos que se encuentren en el **vocabulario del dominio del problemas**

El ***diseño orientado a objetos*** es un método de diseño que abarca el **proceso de descomposición orientado a objetos** y **una notación para describir los modelos lógicos y físico**s, así como los modelos estáticos y dinámicos del sistema que se diseña.

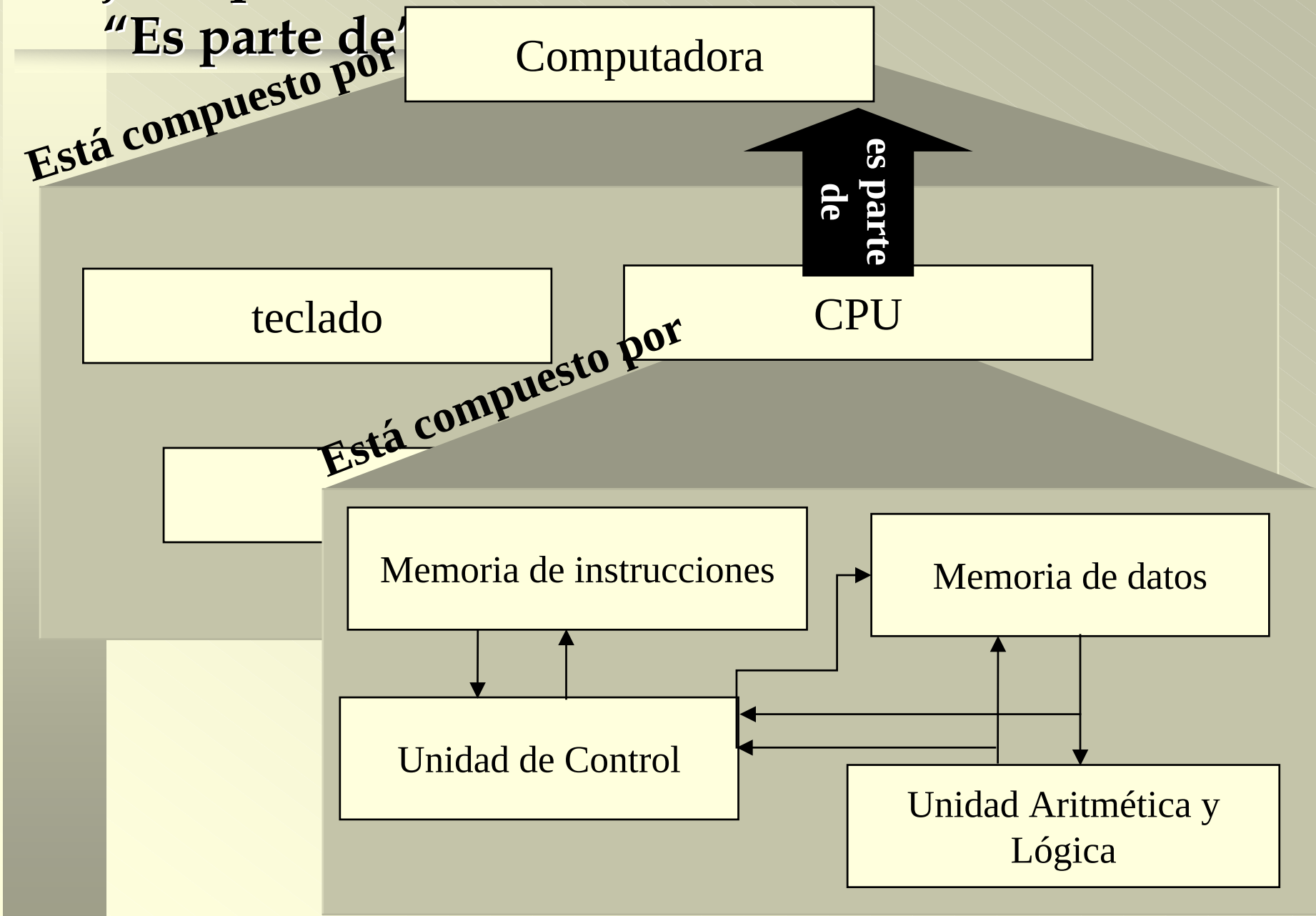
La ***programación orientado a objetos*** es un método de implementación en el que los programas se organizan como colecciones cooperativas de ***objeto***s, cada uno de los cuales representa una **instancia de alguna clas**e, y cuyas clases son, todas ellas, **miembros de una jerarquía de clases** unidas mediante relaciones de herencia.

Evolución de los Paradigmas de Programación



Sistemas Complejos: Estructura jerárquica

"Es parte de"



Sistemas Complejos: estructura Jerárquica.

Es parte de

Está compuesto por

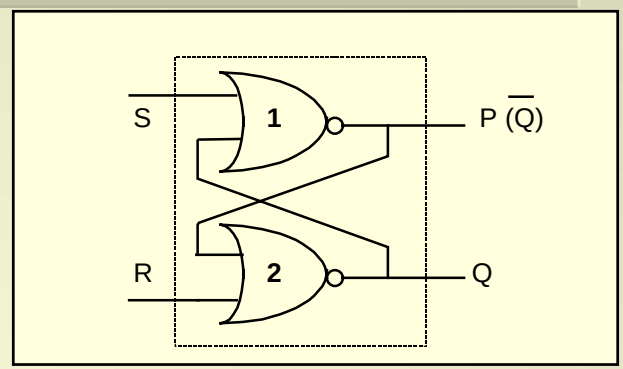
Unidad Aritmética y Lógica

+

RU

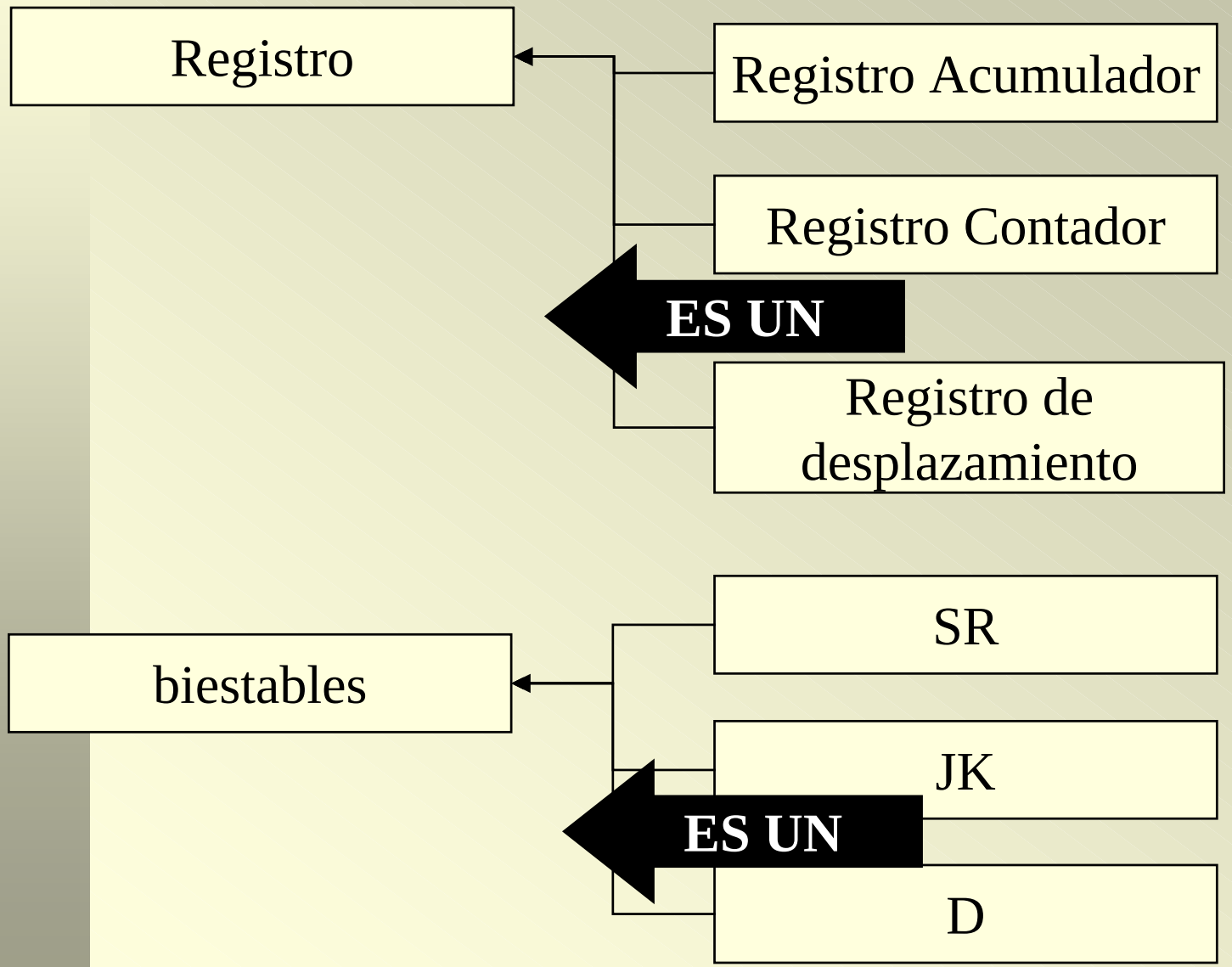
R.Acumulador

Biestables



Sistemas Complejos: estructura jerárquica.

Es-un



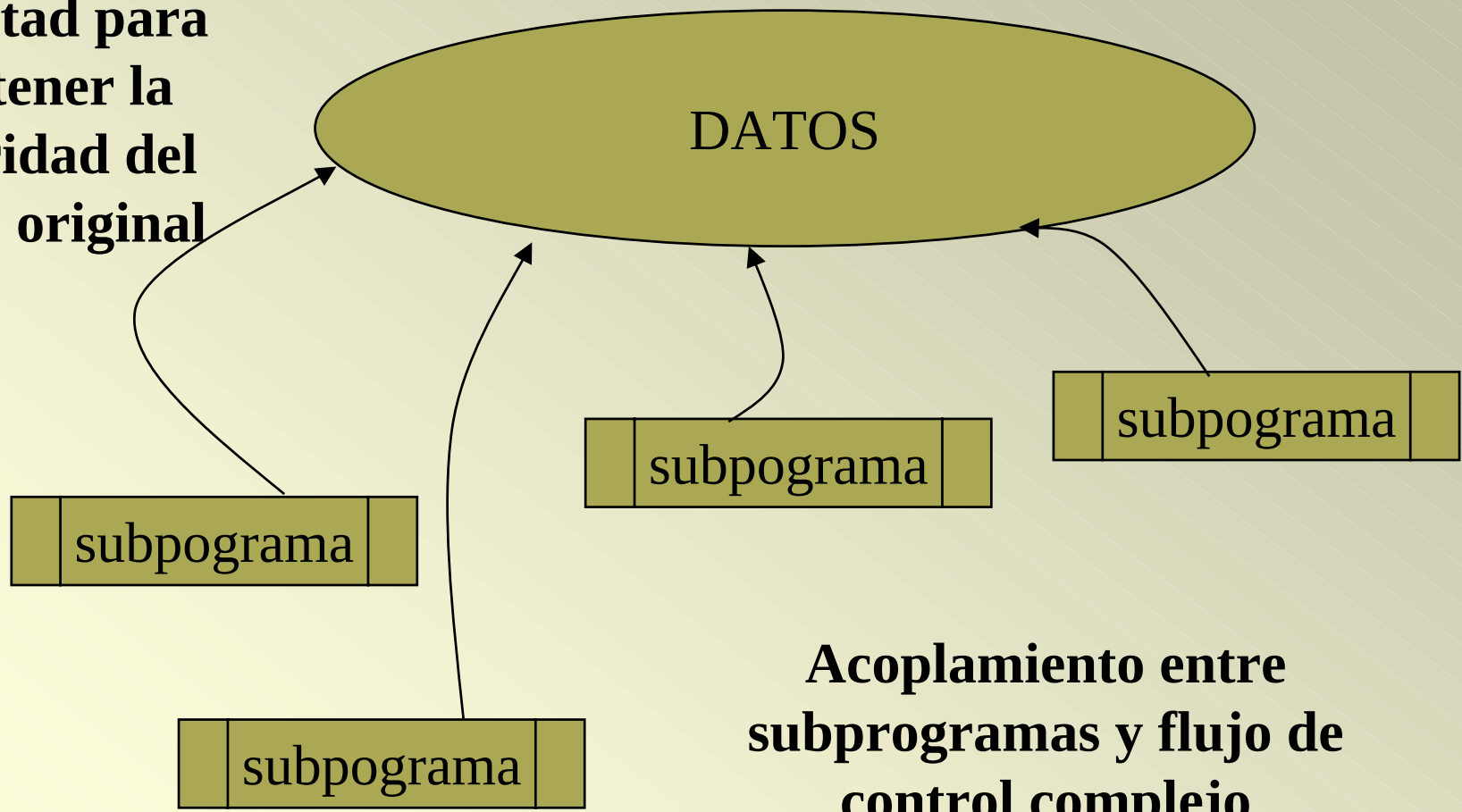
Paradigmas de Programación Procedural

- Es el paradigma de programación tradicional (y probablemente aún el más usado):
  - uso separado de los procedimientos y los datos.`
  - enfoque sobre el diseño de los procedimientos, es decir, sobre el **CÓMO.**

*Se decide qué **procedimientos** se necesitan, y luego se implementan con los **mejores algoritmos** que se puedan encontrar*

Lenguajes de primera generación

Modificaciones ⇒ dificultad para mantener la integridad del diseño original



Acoplamiento entre subprogramas y flujo de control complejo

Lenguajes de segunda generación

- El bloque físico de estos lenguajes era el subprograma. Inicialmente se lo consideró como un dispositivo para ordenar el trabajo.
- Un avance en este enfoque fue el asignar a los subprogramas la finalidad de abstraer funciones del programa.

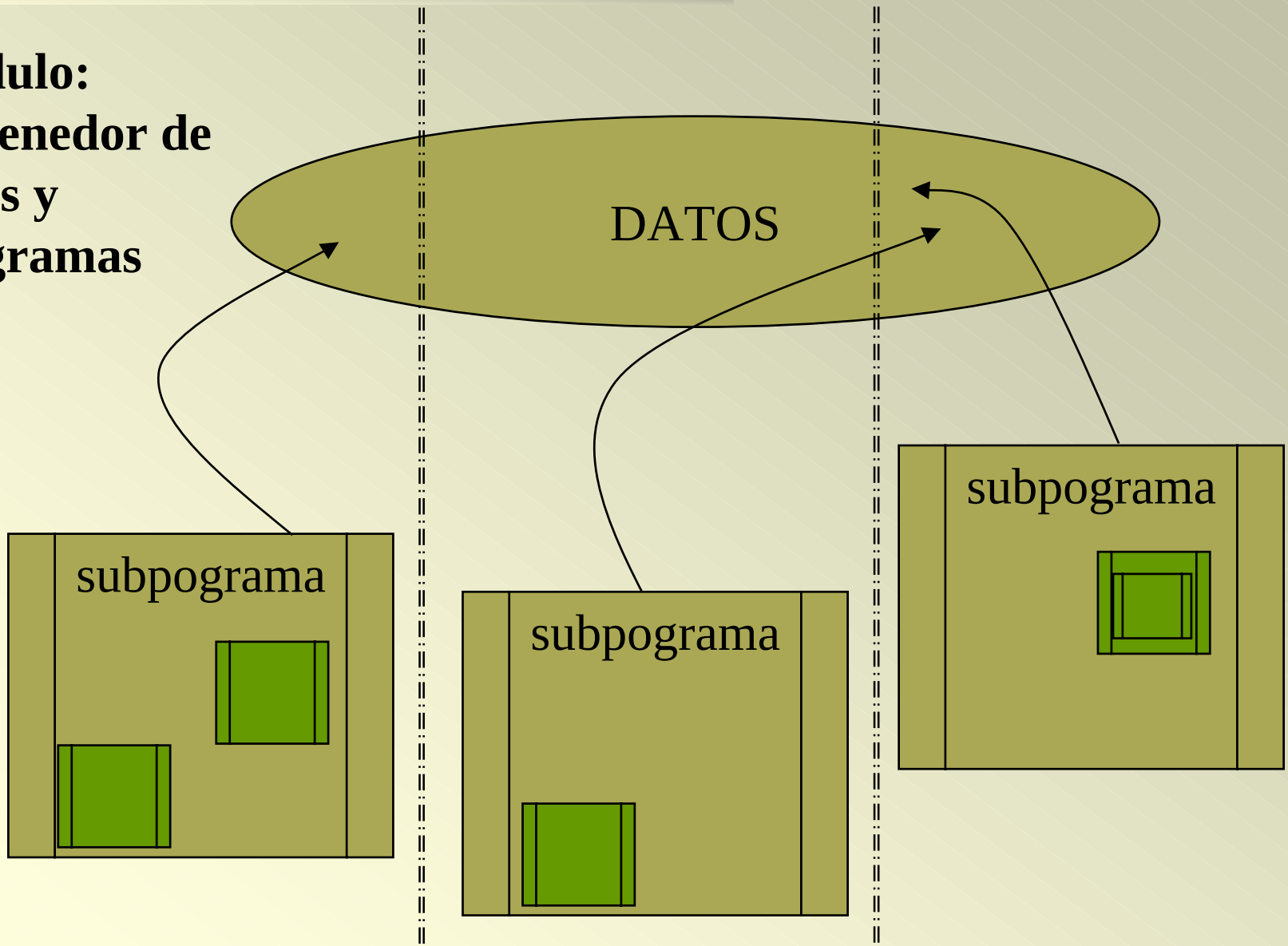
Se crearon lenguajes con diversos mecanismos de pasajes de parámetros

Se establecen los principios de la **Programación Estructurada**: anidación de programas, estructuras de control, ámbito y alcance de declaraciones, etc.

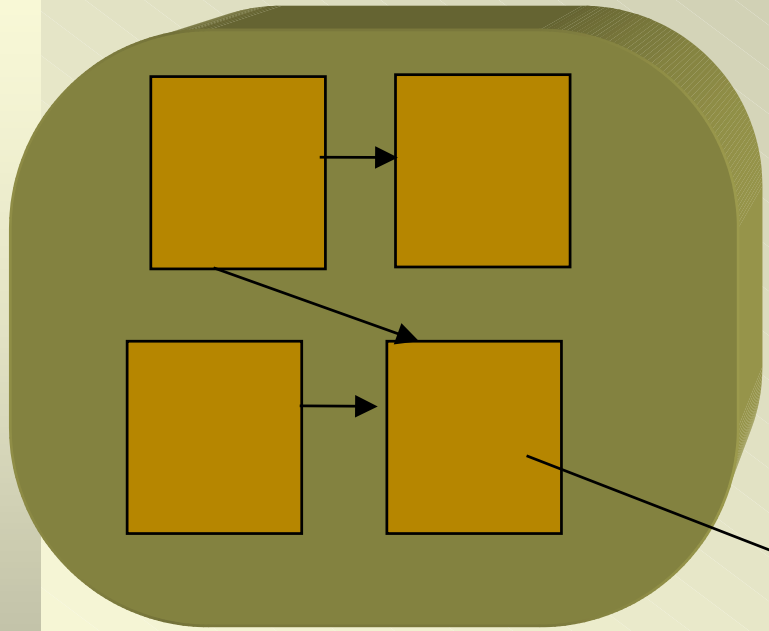
Surgen métodos de Diseño Estructurado

Lenguajes de tercera generación

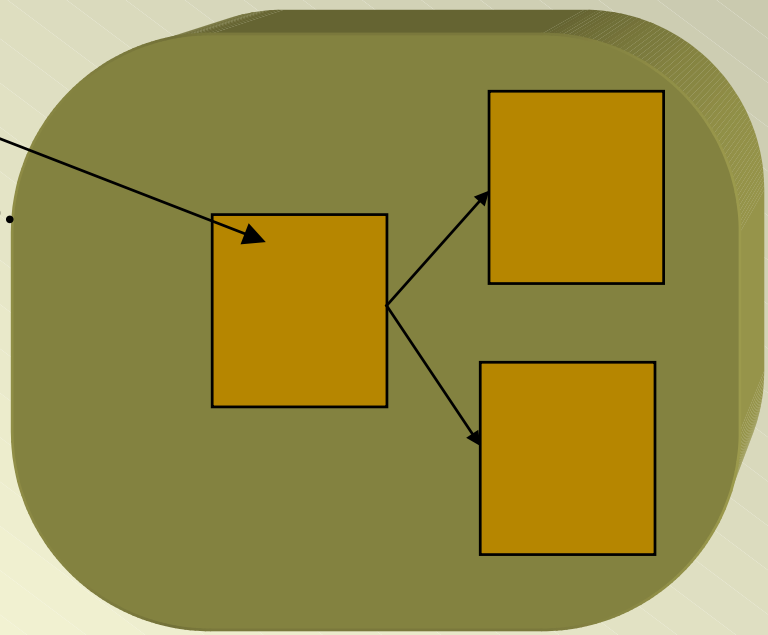
Módulo: contenedor de datos y programas



Lenguajes basados en objetos y orientados a objetos



- **Emergen métodos de diseño dirigidos por los datos.**
- **Encapsulamiento de datos y procesos, ocultos en un módulo.**
- **hay pocos datos globales.**



bloques básicos de construcción: **Clases**.

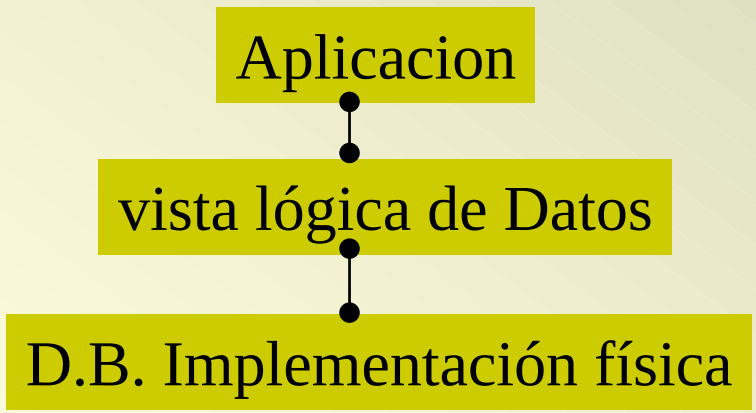
Se hace incapié en los conceptos de ***Abstracción***, ***encapsulamiento***, ***modularidad*** y ***polimorfismo***.

Encapsulamiento

⇒Es la propiedad que asegura que la información de un *objeto* esta *oculta* al mundo exterior.

⇒Es el proceso de *ocultar* todos los secretos de un objeto que no contribuyen a sus características esenciales.

Ninguna parte de un sistema complejo debe depender de los detalles de implementación de otra parte



Modularidad

Particionar un programa en componentes que puedan reducir su complejidad.

⇒ **VENTAJAS:**

- ⇒ **Centralización de todos los datos de un cierto tipo.**
- ⇒ **Programación paralela.**
- ⇒ **Facilita el Mantenimiento y Extensión.**

⇒ **En los lenguajes de programación clásicos los *módulos* se implementan mediante *subprogramas* (procedimientos, funciones y subrutinas).**

⇒ **Ésto, en general, orienta una formulación Funcional del problema, organizándolo mediante refinamientos sucesivos de los *módulos.***

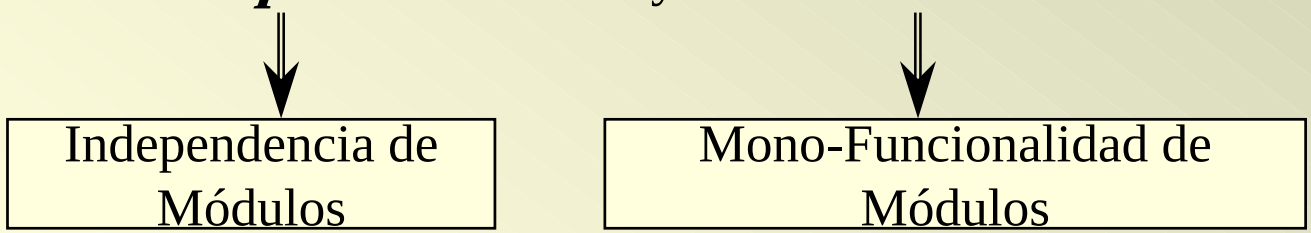
Modularidad

Cuando la **descomposición** se realiza considerando **únicamente** un *Refinamiento Progresivo* puede ocurrir que:

- ⇒ No se permitan extensiones en forma simple.
- ⇒ Las modificaciones puedan requerir un gran esfuerzo y puedan introducir errores.

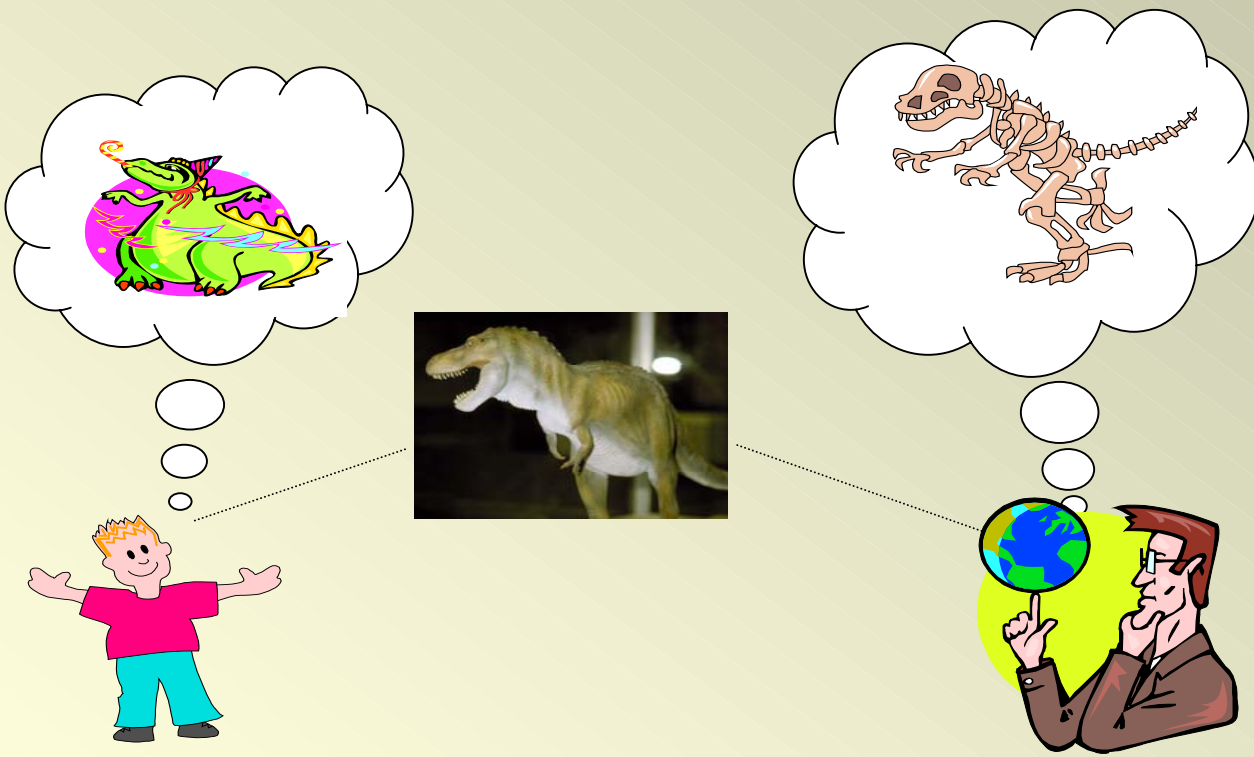
En el Diseño Clásico (estructurado) una modularización que persiga el principio de Encapsulamiento, deberá realizarse con criterios de

acoplamiento y ***cohesión***.



Abstracción

Denota las características escenciales de un objeto las cuales lo distinguen de todos los otros tipos de objetos. Provee una definición conceptual relativa a la perspectiva del observador.



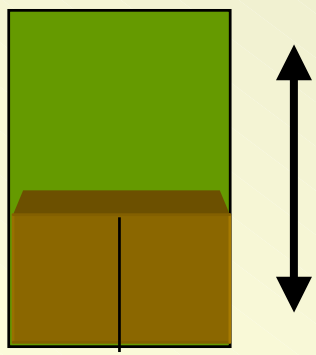
Abstracción

⇒ Enfocada sobre la vista externa del objeto que separa el comportamiento escencial del objeto de su implementación.

⇒Decidir sobre el conjunto correcto de abstracciones para un dominio dado es el problema central en Diseño Orientado a Objetos.

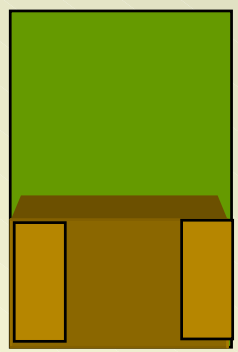
⇒Modelo de contrato: La vista exterior de un objeto define un contrato sobre la cual otros objetos dependen.

Ascensor



subir

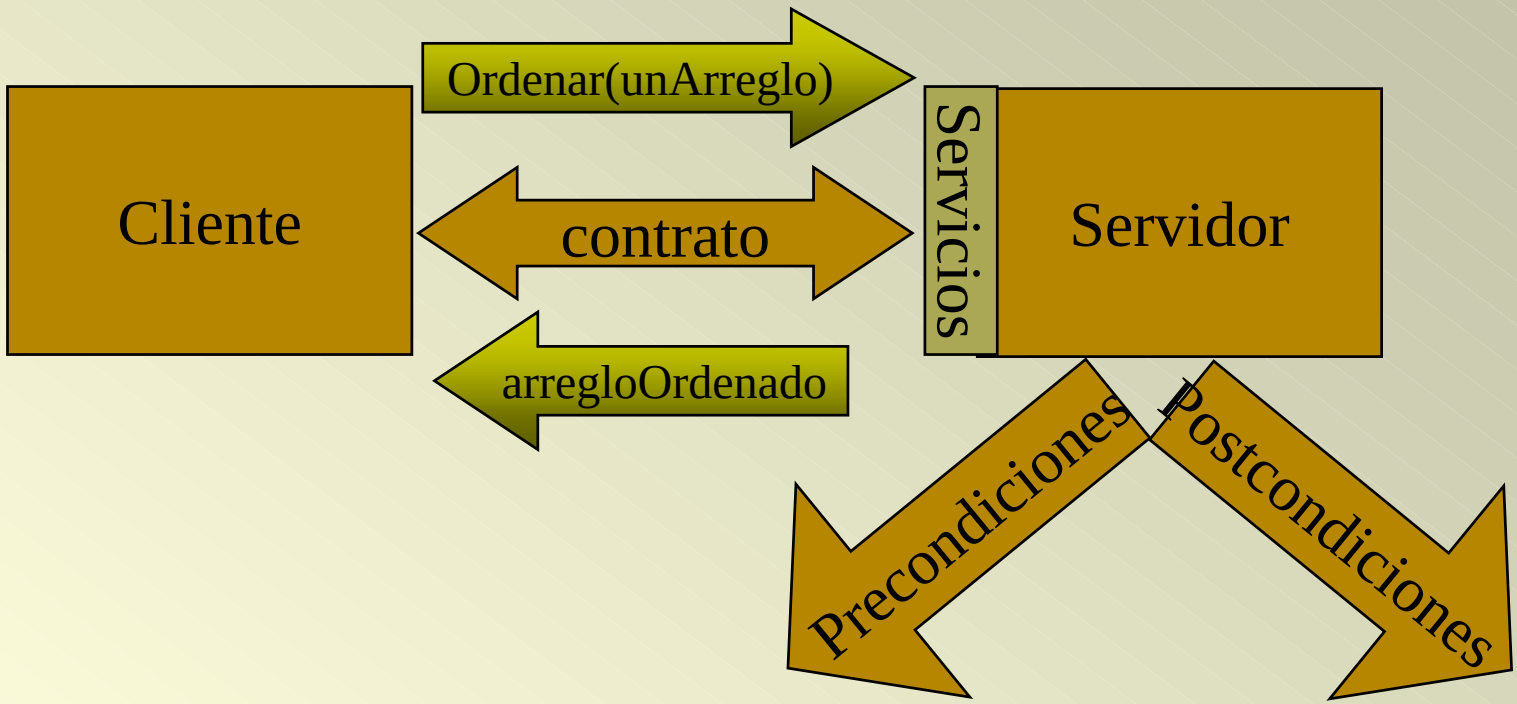
Bajar



Abrir puerta

Abstracción

- Modelo de contrato



Parte del contrato que debe ser satisfecha por el cliente

Parte del contrato que debe ser satisfecha por el servidor

Abstracción: TDA

- La *abstracción de datos* es la técnica de programación que permite definir *nuevos tipos de datos*, adecuados para la aplicación
- ***tipos de datos abstractos (TDA)***

Un lenguaje soportará la definición de ***tipos de datos abstractos***, si:

⇒el usuario puede definir nuevos ***tipos de datos*** asociando explícitamente la ***estructura de datos*** del mismo con las ***operaciones*** que lo manipulan

⇒la **representación** y las **operaciones** están ocultas en el tipo de dato abstracto.

Abstracción: TDA

Un TDA se compone de *estructuras de datos* (representación) y las *operaciones* que la manipulan

TDA = *representación + operaciones*

Para que un ***TDA*** pueda ser empleado, es necesario indicar ***cuáles son*** y ***cómo*** pueden ser usadas las *operaciones* por los clientes del mismo.

La ***especificación*** formal del TDA está compuesta por las siguientes secciones:

- ⇒Tipos
- ⇒Operaciones (funciones)
- ⇒Axiomas
- ⇒Precondiciones

Abstracción: TDA

Si quisieramos construir un TDA que represente una **Pila, que contenga elementos del tipo G**, tendríamos que establecer la siguiente especificación:

Tipos

Pila[G]

Operaciones

push: Pila[G] x G $\rightarrow$ Pila[G]

pop: Pila[G] $\rightarrow$ Pila[G]

top: Pila[G] $\rightarrow$ G

empty: Pila[G] $\rightarrow$ Boolean

new: Pila[G]

Axiomas

∇x: G , s: Pila[G]

A1- top(push(s,x)) = x

A2 - pop(push(s,x)) = s

A3 - empty(new)

A4 - **not** empty(push(s,x))

Precondiciones

pop(s: Pila[G])

requiere not empty(s)

top(s: Pila[G])

requiere not empty(s)

Podríamos afirmar que esta especificación define la *interfaz* del **TDA** Pila

Paradigmas de programación Orientado a Objetos

El Paradigma O. O. tiene como objetivo:

- escribir software fácilmente modificable y extensible.
- escribir software reusable
- Disponer de un modelo natural para representar un dominio.

DECIDIR QUE ***CLASES (O TIPOS)*** *SE REQUIEREN,*

PROVEER UN CONJUNTO COMPLETO DE OPERACIONES PARA CADA UNA DE LAS CLASES

Y

HACER EXPLICITAMENTE ACCESIBLE SU ESTRUCTURA Y COMPORTAMIENTO MEDIANTE LA HERENCIA

Beneficios del modelo de Objetos

- Ayuda a explotar las ventajas de los lenguajes de POO.
- Permite reusar tanto software como diseño.
- El software desarrollado, es más flexible al cambio.
- Es cercano a la forma de pensar de los humanos.