Programación con TAD

PROGRAMAS = DATOS + ALG. DATOS + ALG. CONTROL

Diseño Modular ----> *PROGRAMAS = TADs + ALG. CONTROL*

Pasos en la programación con TAD:

1. Especificación del tipo

Lección 2

- resultante de lo identificado en la **fase de diseño**
  - Un TAD tiene 1 especificación (única)
  - Una especificación tiene 2 partes:
    - La parte sintáctica (SIGNATURA)
    - La parte semántica

2. Implementación del tipo

Lección 3

  - Dada la especificación de un TAD, pueden realizarse de 0 .. N implementaciones del TAD
  - En una implementación de un TAD, habrá 3 etapas…

3. Uso del tipo *(objetivo: reutilización)*

Implementación de TAD

Lección 3

Esquema

- Pasos en la implementación de un TAD
- De la especificación a la implementación:
  - En *Programación Modular* →
    - En esta asignatura: seudocódigo y C++
    - Utilizaremos un lenguaje de programación abstracto o *seudocódigo* para la implementación de los TADs en las clases y exámenes de la asignatura
      - Corresponde a un paradigma de *Programación Modular*
      - Ver resumen de su sintaxis en el material de clase publicado
  - Tras estudiar *Programación OO* en otras asignaturas → Java, C++ (utilizando clases), etc.

Implementación de un TAD

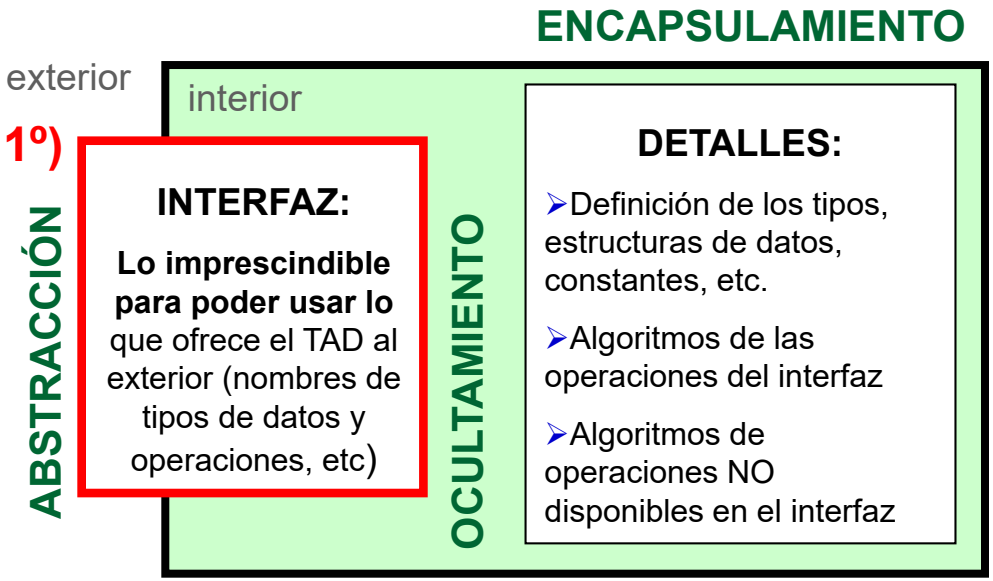
- La implementación de un TAD consiste en:
  - determinar una representación para los valores del tipo
  - codificar sus operaciones a partir de esa representación
- Una implementación ha de ser:
  - Estructurada (para facilitar su desarrollo)
  - Eficiente (optimizar el uso de recursos: tiempo, espacio)
  - Legible (para facilitar su modificación y mantenimiento)
  - Segura y robusta
  - Correcta, verificable, fácil de usar….
  - **Garantizar la encapsulación**:
    - privacidad de la representación y protección del tipo (frente a usos o accesos indebidos)→ *tipo opaco*

Implementación de un TAD

- **Para implementar un TAD (3 etapas):**

  1º) Definir todo lo que aparecerá en la ***parte pública*** o ***interfaz*** del **módulo** que implemente el TAD:
  - identificadores válidos
  - perfiles o cabeceras de cada operación:
    - parámetros de entrada, salida o entrada-salida
    - comunicación de situaciones de error
  - **documentación pública** (lo que necesitan saber los posibles usuarios, sin ningún detalle de implementación)

→Este 1er paso es requisito previo para poder distribuir el trabajo de implementación en el equipo de programadores
(implementación del TAD en paralelo, o independiente, de la implementación del código que usará el TAD)



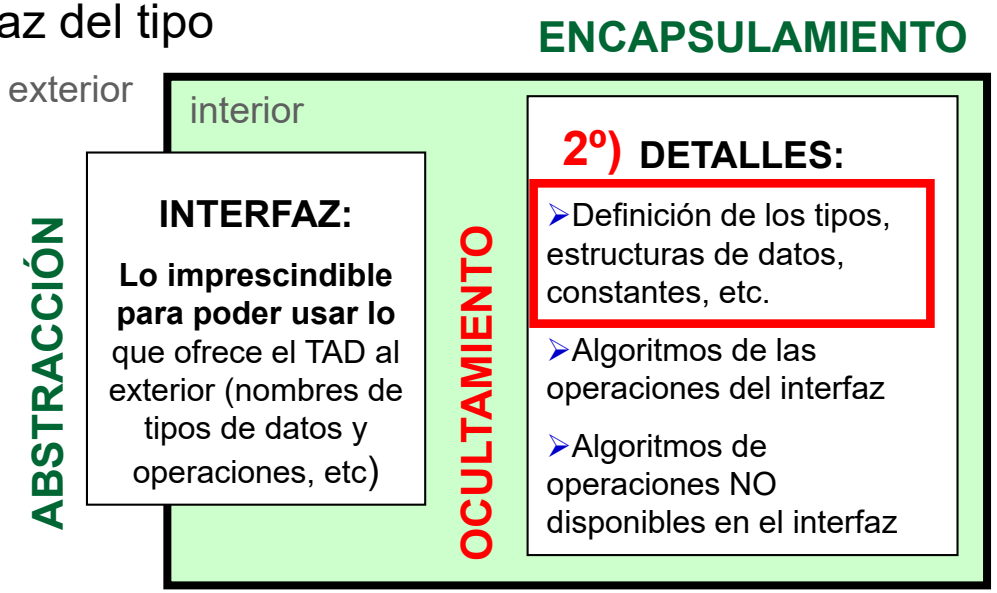
Implementación de un TAD

2º) Decidir cómo representar el tipo de datos a definir, en base a tipos básicos predefinidos, construcciones básicas como vectores y registros, u otros tipos definidos previamente

→ ***representación interna*** concreta del nuevo TAD

- deberá permitir implementar las operaciones definidas para el tipo, de forma eficiente tanto en coste en memoria como en tiempo
- La **representación interna** deberá permanecer **oculta**
  - El uso del nuevo tipo solo deberá ser posible mediante las operaciones definidas en la interfaz del tipo

IMPORTANTE:
Debe estar documentada, desde el principio
documentación de la parte privada: está destinada a programadores que desarrollan los detalles y código internos de esta implementación, y a quienes necesitarán mantenerla



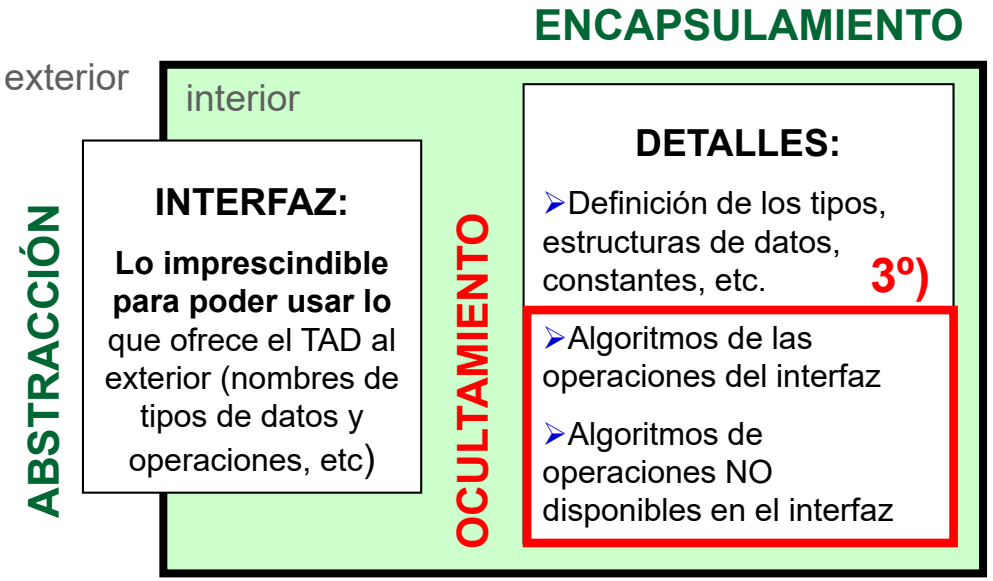
(encapsulación = privacidad + protección)

Implementación de un TAD

3º) Implementar cada operación de la interfaz del TAD, y las operaciones auxiliares que resulten de interés, de acuerdo a la representación interna definida

- las operaciones de la interfaz: serán accesibles (para su uso, pero no su implementación) para los usuarios del nuevo tipo
- el resto de las operaciones: sólo serán utilizables en la implementación del propio TAD (ocultas)

Debe estar documentada
(documentación de la parte privada: cabeceras de operaciones, algoritmos…)



(encapsulación = privacidad + protección)

Implementación en Programación Modular

Guía para pasar de la especificación a una implementación en seudocódigo u otro leng. de programación modular

- Las operaciones de la especificación del TAD:
  - Las operaciones 0-arias (también llamadas *constantes*):
    - Se implementarán como **constantes**, o como **procedimientos o funciones sin parámetros**
  - Las demás operaciones se implementarán como **procedimientos** o **funciones**:
    - Varias operaciones con el mismo dominio y distinto rango o resultado, podrán implementarse como un procedimiento que devuelva los resultados de dichas operaciones:
      - Interesante, hacerlo así, si dichas operaciones se van a utilizar a menudo de forma conjunta
      - Importante si de esa forma el coste de obtener los resultados se reduce
        - » Con el coste o trabajo para encontrar el resultado de una de las operaciones, encontramos el de la/s otra/s
  - Las operaciones con casos o situaciones de error (op. parciales):
    - Mecanismos de protección frente a errores dependientes del lenguaje de programación a utilizar
      - Manejo de excepciones → identificar o definir excepciones a utilizar
      - Parámetros de salida de error en cada operación

Operaciones: procedimientos o funciones

- En el lenguaje seudocódigo las operaciones de los TADs podrán implementarse como procedimientos o funciones
- Procedimientos y funciones encapsulan un bloque de acciones reutilizable (el código que implementa un algoritmo), y simplifican la lectura de los algoritmos y programas que escribimos

*{Un **procedimiento** es una **acción** o **instrucción virtual**, que una vez definido se puede utilizar como otra instrucción mas.*

*Un procedimiento puede definirse con **0 o más parámetros**, cada uno de ellos podrá ser de:*

- Entrada (ent): dato que el procedimiento recibe para ser utilizado en sus acciones

- Salida (sal): dato que el procedimiento comunica como resultado de sus acciones

- Entrada y Salida (e/s): dato que el procedimiento recibe para ser utilizado en sus acciones y cuyo valor actualizado comunica como resultado de sus acciones. }

procedimiento <nombre>(**ent** <parámetros_1>:<tipo_1>; **sal** <parámetros_2>:<tipo_2>;
e/s <parámetros_3>:<tipo_3> ...)
<*declaraciones locales* de constantes, variables, tipos de datos, proced., funciones...>
principio
<secuencia de acciones>
fin

Ejemplo de uso:

nombreProced(arg1, arg2, arg3);

- Tipos de paso de parámetros: por ***valor*** *(copia)*, o por ***referencia***
  - Los parámetros de entrada (*ent*) son pasados por valor,
    - pero parámetros de entrada cuyo tamaño haga ineficiente copiarlos, también deberán ser pasados por referencia
  - Los parámetros de salida (*sal*) y de entrada/salida (*e/s*) son pasados por referencia

Operaciones: procedimientos o funciones

*{ Una **función** es un **valor virtual**, es decir, se puede utilizar dentro de una expresión y el resultado es un valor.*

*Puede tener **0 o más parámetros, todos ellos de entrada, y** sólo puede (y debe!) devolver **un resultado**. }*

función <nombre>**(**<parám_1>:<tipo_1>; <parám_2>:<tipo_2> ...**) devuelve** <tipo_fun>
  <*declaraciones locales* de constantes, tipos, variables, proced., funciones...>
principio
  <secuencia de acciones>
  devuelve <valor_de_tipo_fun> *{tras devolver el valor, la función termina}*
  <… secuencia de acciones>
fin

Ejemplo de uso:
cálculo:= nombreFunción(2.7, x, y) + 23.5;

➢ *Al encapsular una operación del TAD, o un algoritmo, como procedimiento o función:*

- *si sólo hay que devolver **un dato resultado** (salida), podrá ser definido como **procedimiento o** como **función***
  - Si preferimos definirlo como ***valor virtual,*** para usarlo dentro de expresiones → implementarlo como ***función***
- *sino, (**devuelve 0, 2 o más resultados)** tendrá que ser necesariamente un **procedimiento***
- ***Si tiene que tener algún parámetro de e/s**, tendrá que ser necesariamente un **procedimiento***

Implementación en Programación Modular

Guía para pasar de la especificación a una implementación en seudocódigo u otro leng. de programación modular

- Las operaciones de la especificación del TAD:
  - Las operaciones 0-arias (también llamadas *constantes*):
    - Se implementarán como **constantes**, o como **procedimientos o funciones sin parámetros**
  - Las demás operaciones se implementarán como **procedimientos** o **funciones**:
    - Varias operaciones con el mismo dominio y distinto rango o resultado, podrán implementarse como *un procedimiento* que devuelva los resultados de dichas operaciones:
      - Interesante, hacerlo así, si dichas operaciones se van a utilizar a menudo de forma conjunta
      - Importante, si de esa forma el coste de obtener los resultados se reduce
        - » Con el coste o trabajo para encontrar el resultado de una de las operaciones, encontramos el de la/s otra/s

Deberá respetar la especificación conjunta de todas esas operaciones

  - Las operaciones con casos o situaciones de error (op. parciales):
    - Mecanismos de protección frente a errores dependientes del lenguaje de programación a utilizar
      - Manejo de excepciones → identificar o definir excepciones a utilizar
      - Parámetros de salida de error en cada operación

Ejemplo de fechas: Especificación no formal

espec fechas *{Vista en la lección 2}*

usa enteros, booleanos

género fecha

{DESCRIPCION del TAD: Los valores del TAD fechas representan fechas válidas según las reglas del calendario gregoriano}

operaciones

parcial crear: entero d, entero m, entero a → fecha

{Dados 3 valores enteros: d, m y a, siendo $1 \leq d \leq 31$, $1 \leq m \leq 12$, $1583 \leq a$, se obtiene una fecha compuesta con los tres valores dados usados como día, mes y año respectivamente.

***Parcial:** se producirá una situación de error (la operación no está definida) si los valores d, m y a no pueden formar un valor de fecha válido según el calendario gregoriano (fechaInválida)}*

día: fecha f → entero

{ Dada una fecha f, se obtiene el entero que corresponde al día en la fecha f}

mes: fecha f → entero

{Dada una fecha f, se obtiene el entero que corresponde al mes en la fecha f}

…