TIPO DE DATO ABSTRACTO (TAD)

Tipos Abstractos de Datos (TAD)

Sabemos que:

- ***Los tipos de datos*** son necesarios para identificar valores y operaciones posibles para variables y expresiones.

- Las nociones de ***estructuras de datos***, ***variables*** ***y constantes*** que se han estudiado son abstracciones para tratar de acercar al mundo real la especificación de los datos de problemas.

Tipos Abstractos de Datos (TAD)

Sabemos también que:

- Los lenguajes de programación y los sistemas operativos se encargan de manejar las conversiones entre las definiciones abstractas y la representación interna en memoria.

Hasta ahora se han considerado tipos de datos simples, estructurados y definidos por el usuario.

Tipos Abstractos de Datos (TAD)

La mayoría de los lenguajes de programación modernos *impone la obligación de declarar* todos los nombres de variables, *asociándoles un tipo.*

Los lenguajes también verifican que las operaciones se hagan entre datos *del mismo tipo.* La mezcla de tipos en general es detectada y considerada un error por los lenguajes de programación.

Tipos Abstractos de Datos (TAD)

También hemos visto la ventaja de descomponer (modularizar) un problema (sistema de software) en procedimientos y funciones.

De este modo se abstraen las operaciones del sistema, de modo de descomponerlo en funcionalidades (procedimientos/ funciones) relativamente independientes y más simples que el problema global.

Tipos Abstractos de Datos (TAD)

En general, el proceso de abstracción se entiende como la identificación de los conceptos esenciales, ignorando los detalles.

Se puede hacer referencia entonces:

- Abstracción de procedimientos → módulos
- Abstracción de datos → TAD

Tipos Abstractos de Datos (TAD)

¿Cómo relacionamos TAD con los módulos que utilizamos hasta el momento?

- Un módulo se puede ver como un bloque que tiene declaración de comunicación con otros módulos (interfaz) y una funcionalidad interna.
- La interface (del módulo) es una especificación de QUÉ puede hacer el módulo y puede tener declaraciones de tipos y variables que deban ser conocidas externamente.
- La funcionalidad interna (implementación del módulo) abarca el código de los procedimientos que concretan el CÓMO cumplir la función del módulo.

Tipos Abstractos de Datos (TAD)

Una abstracción del tipo de dato alumno podría verse como:

Alumno

Declaración de la estructura de datos Alumno

Operaciones relacionadas con un alumno.

- Asignar datos a un alumno
- Ver datos de un alumno
- Evaluar si un alumno tiene una cursada aprobada
- Evaluar si un alumno tiene un final aprobado
- otras

TAD - Abstracción y Encapsulamiento de Datos

Caracterizar los problemas del mundo real significa *reconocer* los objetos del mundo real y *abstraer* sus aspectos fundamentales y su comportamiento, de modo de representarlos sobre una computadora.

La *utilidad* fundamental de *abstraer y modelizar* objetos del mundo real es **re-usar** soluciones en diferentes problemas.

Relacionar estos conceptos con el tipo de dato Alumno visto.

TAD - Abstracción y Encapsulamiento de Datos

En el desarrollo de sistemas de software *es importante* trabajar con especificaciones abstractas, de modo de mejorar la calidad de los programas.

Los tipos definidos por el usuario permiten realizar *encapsulamiento o empaquetamiento* de los datos: se define un nuevo tipo y se integran en un módulo todas las operaciones que se pueden hacer con él.

¿Cómo hemos trabajado hasta ahora?

TAD - Abstracción y Encapsulamiento de Datos

¿Cuáles son los objetivos que se persiguen?

- Si el lenguaje permite separar la parte visible (interfaz) de la implementación se tendrá *ocultamiento de datos (data hiding)*.

- Si se logra que en la solución del módulo los cambios en los tipos de datos internos ***NO*** *afecten la parte visible del módulo*, se tendrá *independencia de la representación.*

TAD - Abstracción y Encapsulamiento de Datos

El concepto de ***tipo abstracto de dato*** es un tipo de dato definido por el programador que incluye:

- Especificación de la representación de los elementos del tipo.
- Especificación de las operaciones permitidas con el tipo (siempre visible)
- Desarrollo interno de cada una de las operaciones permitidas, con independencia para el usuario de cómo se implementan.
- Encapsulamiento de todo, de manera que el usuario no pueda manipular los datos del objeto excepto por el uso de las operaciones definidas en el punto anterior.

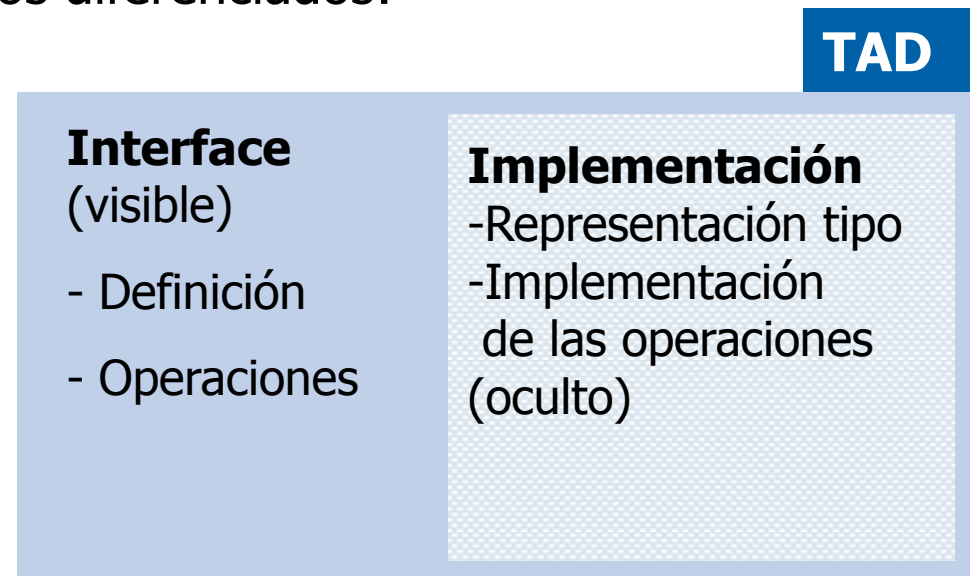
TAD - Abstracción y Encapsulamiento de Datos

Su desarrollo es independiente del lenguaje de programación utilizado, aunque este puede aportar mecanismos que faciliten su realización. Debe verse como una caja negra.

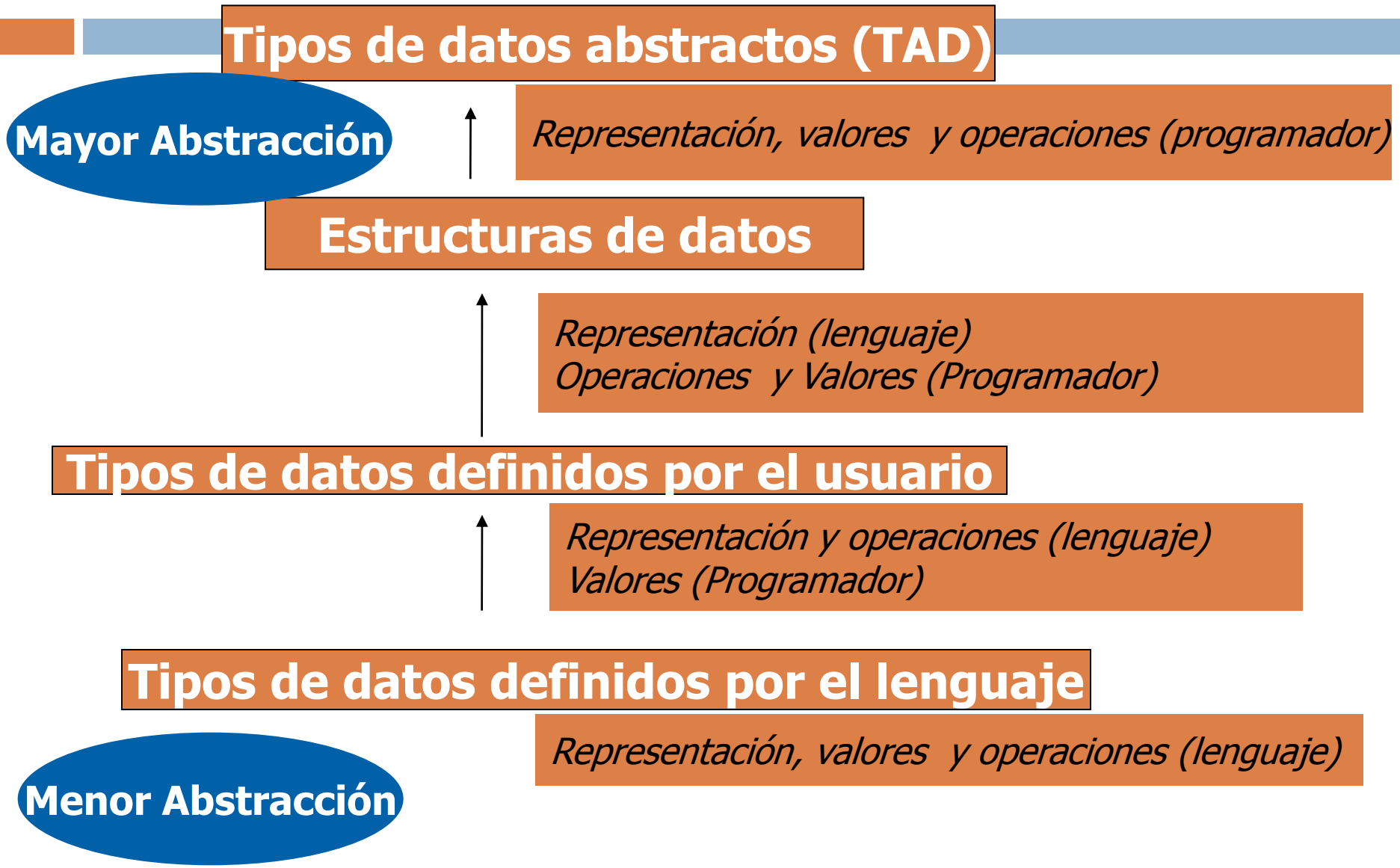
En un TDA existen dos elementos diferenciados:

- La Interface de utilización
- La Implementación

A la hora de utilizar el TDA, la representación debe permanecer oculta. Solo podremos utilizar las operaciones del tipo para trabajar con sus elementos.



Diferentes niveles de abstracción en los datos



TAD - Abstracción y Encapsulamiento de Datos

Un TAD reúne en su definición la representación y el comportamiento de los objetos del mundo real y se puede escribir la ecuación de un programa como:

Programa = TADs + Algoritmos de Control

TAD - Características

- Es un tipo de datos para representar objetos del mundo real que no existen en el lenguaje.

- Presentan una parte pública (INTERFACE) donde se declara el nombre del tipo y las operaciones permitidas y una parte privada (IMPLEMENTACION) donde se representa el tipo declarado y todas las operaciones descriptas en la interface.

- El usuario sólo conoce la parte pública, es decir, no conoce cual es la representación del tipo utilizada ni cual es la implementación las operaciones.

En el TAD identificamos dos roles:

TAD – Roles Contructor / usuario

El Constructor del Tad debe:

- Declarar la interface del TAD. (el tipo exportado y operaciones)
- Definir la representación del tipo exportado e implementar las operaciones del tipo declaradas en la interface.

El Usuario del Tad debe:

- Indicar el nombre del TAD a utilizar.
- Declarar las variables del tipo de datos exportado.
- Utilizar SOLO las operaciones permitidas para el tipo, es decir , las operaciones definidas en la interface.

TAD – Sintaxis propuesta

TAD nombre_TAD
interface
type exportado nombre_tipo_exportado
procedures y funciones visibles

implementación
representación del type exportado
procedures y funciones privadas (si las hay)
implementación de procedures y funciones visibles

TAD – Esquema general del programa

```
program nombre_programa;
uses nombre_TAD;

var
  nombre_variable: nombre_type_exportado del TAD;
begin
    ……
    nombreOperacion (parametros);
    ……
end.
```

TAD - Abstracción y Encapsulamiento de Datos

Ventajas:

- ✓Permite una mejor conceptualización y modelado del mundo real. Mejora la representación y comprensión.

- ✓La abstracción de datos provee un camino para poner juntos los componentes de software que están relacionados.

- ✓El uso de un TAD es sencillo porque no es necesario recordar detalles de implementación, solo es necesario conocer las operaciones.

- ✓El ocultamiento de la información permite garantizar la integridad de los valores de un TAD. Solo las operaciones definidas por el TAD pueden manipular la estructura de datos utilizada.

- ✓La separación de la definición del tipo de su implementación permite cambiar el módulo de implementación sin necesidad de modificar los programas que lo utilizan.

TAD – Ejemplo Alumno

TAD TipoAlumno

Interface

type exportado Alumno

procedure Crear (var a: Alumno; nombre, dni: string; edad;integer);

procedure verNombre (a: Alumno; var nombre: string);

procedure verDni (a: Alumno; var dni: string);

function verEdad (a: Alumno): integer;

procedure modNombre (var a: Alumno; nombre: string);

procedure modDni (var a: Alumno; dni: string);

procedure modEdad (var a: Alumno; edad: integer);

TAD – Ejemplo Alumno

Implementation

```
Alumno = record
                nombre,dni:string;
                edad:integer;
              end;
procedure Crear ( var a: Alumno; nombre, dni: string; edad; integer );
 begin
   a.nombre:= nombre; a.dni:=dni; a.edad:=edad;
 end;

procedure verNombre (a: Alumno; var nombre: string);
  begin
     nombre:=a.nombre;
  end;

procedure verDni (a: Alumno; var dni: string);
   begin
      dni:=a.nombre;
   end;
```

TAD – Ejemplo Alumno

```
function verEdad (a: Alumno):integer;
      begin
        verEdad:= a.edad;
       end;

 procedure modNombre (var a: Alumno; nombre: string);
       begin
         a.nombre:= nombre;
       end;

 procedure modDni (var a: Alumno; dni: string);
      begin
        a.dni:= dni;
       end;

 procedure modEdad (var a: Alumno; edad: integer);
      begin
        a.edad:= edad;
       end;
```

TAD – Ejemplo Alumno

Realice un programa que lea información de 10 personas, (utilizando el Tad TipoAlumno) y las almacene en un arreglo.

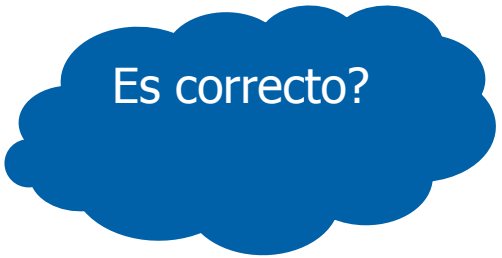
```
Program uno;
  uses TipoAlumno;

  type
    alumnos = array[1..10] of Alumno;

  var
   p: alumno;
   nom, dni:string;
   e, i:integer;
   v: alumnos;
```

```
begin
  for i:= 1 to 10 do
   begin
     read (nom, dni); read(e);
     crear (p, nom, dni, e);
     v[i]:= p;
   end;
end.
```

Esta NO es una operación definida en la interface



Qué tenemos que modificar?

TAD – Ejemplo Alumno

TAD TipoAlumno

Interface

type exportado Alumno

procedure Crear (var a: Alumno; nombre,dni:string;edad;integer);

procedure verNombre (a: Alumno; var nombre:string);

procedure verDni (a: Alumno; var dni:string);

function verEdad (a: Alumno):integer;

procedure modNombre (var a: Alumno; nombre:string);

procedure modDni (var a: Alumno; dni:string);

procedure modEdad (var a: Alumno; edad:integer);

procedure asignar (var a: Alumno; a2:alumno);

TAD – Ejemplo Alumno

```
procedure asignar (var a: alumno; a2:alumno);
 Begin
   a:= a2;
 End;
```

¿Cómo reescribimos el programa?

TAD – Ejemplo Alumno

Realice un programa que lea 10 personas, (utilizando el Tad TipoAlumno) y las almacene en un arreglo.

```
Program uno;

  uses TipoAlumno;

  type
    Alumnos = array[1..10]of Alumno;

  var
   p: alumno;
   nom,dni:string;
   e,i:integer;
   v: alumnos;
```

```
 begin
   for i:= 1 to 10 do
    begin
      read (nom, dni);
      read(e);
      crear (p, nom, dni, e);
      asignar( v[i], p);
    end;
end.
```

TAD – Consideraciones Prácticas

- Un TAD puede tener un único tipo exportado

- Los parámetros de los procedimientos y funciones de un TAD pueden ser tipos simples: integer, boolean, char, string ó el tipo exportado.

- En la implementación de las operaciones del TAD no se pueden hacer operaciones de lectura ni de escritura.

- El programa que utiliza el TAD sólo puede aplicar a las variables del tipo exportado del TAD las operaciones definidas en la interface, esto debe incluir a las operaciones de asignación y comparación en caso de ser necesarias .

TAD – Ejemplo Polinomio

TAD Polinomios

Interface

type exportado Polinomio

procedure Create (var P: Polinomio);

procedure CargarDato (var P: Polinomio; coef: integer; pot: integer);

{esta operación guarda un elemento con determinado coeficiente y potencia en el polinomio}

procedure Sumar (var S: Polinomio; P1, P2: Polinomio);

procedure Multiplicar (var M: Polinomio, P1, P2: Polinomio);

procedure Dividir (var D: Polinomio, P1, P2: Polinomio);

Function VerGrado (P: Polinomio) : integer;

TAD – Ejemplo Polinomio

TAD Polinomios
implementación

```
const
     GradoMax = .......;

 type

  Polinomio =  record
                    arreglo_coef: array [0..GradoMax]of integer;
                     potencia_mayor: integer;
                   end;
```

TAD – Ejemplo Polinomio

```
procedure Create ( var P: Polinomio);
    var  i: integer;

    begin
        for i:= 0 to GradoMax do
                P.arreglo_coef[i] := 0;
        P.potencia_mayor := 0;
    end;
```

TAD – Ejemplo Polinomio

```
procedure Sumar (var S: Polinomio; P1, P2: Polinomio);
    var  i: integer;

begin
  Create ( S );
  S.potencia_mayor:= max (P1.potencia_mayor, P2.potencia_mayor);
   for i:= S.potencia_mayor downto 0 do
       S.arreglo_coef[i]:= P1.arreglo_coef[i] +P2.arreglo_coef[i];
  end;
```

TAD – Ejemplo Polinomio

```
procedure Multiplicar (var M: Polinomio, P1, P2: Polinomio);

var  i, j: integer;

begin
  Create ( M );
  M.potencia_mayor:= P1.potencia_mayor + P2.potencia_mayor;
   for i:= 0 to P1.potencia_mayor do
        for j := 0  to P2.potencia_mayor do
             M.arreglo_coef[i+j] := M.arreglo_coef[i+j] +
                                    P1.arreglo_coef[i] *
P2.arreglo_coef[j];
end;
```

Analicemos qué se modifica si el diseñador del TAD decide representar el Polinomio con una Listas Enlazada?

TAD – Ejemplos

Discutir diferentes representaciones y operaciones para:

- **TAD** Fecha

TAD – Ejemplo Fecha

TAD TipoFecha

Interface

type exportado Fecha

Procedure Cargar (d, m, a: integer; var f: fecha);

Function dia (f: fecha): integer;

Function mes (f: fecha): integer;

Function anio (f: fecha): integer;

Function mayor (f1, f2: fecha): boolean;

Procedure SumarDias (f1: fecha; d: integer; var f2: fecha);

.....

TAD – Ejemplo Fecha

TAD Fecha
implementación

type

```
fecha =  record
            dia : integer;
            mes : integer;
             anio: integer;
          end;
```

TAD – Ejemplo Fecha

```
procedure cargar ( d,m,a: integer; var f: fecha);

    begin
        f.dia :=  d;
        f.mes := m;
        f.anio := a;
    end;

Function dia ( f: fecha) : integer;

    begin
        d := f.dia;
    end;
```

TAD – Ejemplo Fecha

```
Procedure SumarDias (f1: fecha; dias: integer; var f2: fecha);

  var diaaux,maux : integer;

  Begin {generalizando a meses de 30 dias}
      f2.dia := f1.dia + dias
      If f2.dia > 30 then begin
                              diaaux := f2.dia;
                               f2.dia := f2.dia mod 30;
                               f2.mes := f1.mes + diaaux div 30
                               if f2.mes > 12 then begin
                                                       mesaux := f2.mes;
                                                       f2.mes := f2.mes mod 12;
                                                       f2.anio := f1.anio + mesaux div 12;
                                                   end
                                                   else f2.anio := f1.anio;
                          end
                          else begin
                                 f2.mes := f1.mes;
                                 f2.anio := f1.anio;
                          end;
      End;
```