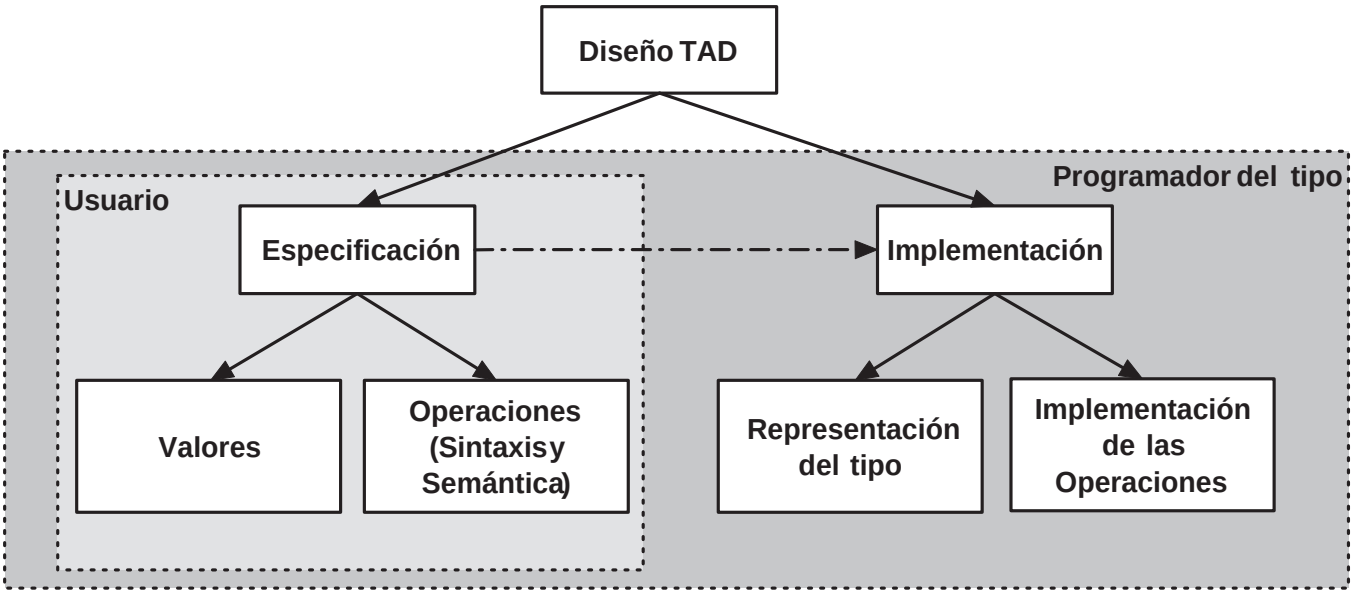
Definición de Tipo de Dato Abstracto (TDA)

John Guttag (1974):

Un tipo de dato abstracto es aquél **definido por el programador** que puede ser manipulado de forma **similar** a los **definidos por el sistema**.

Al igual que estos últimos, un tipo de dato abstracto corresponde a un **conjunto de valores** lícitos y de **operaciones** asociadas a los mismos, operaciones que se definen mediante una **especificación** que es **independiente de la implementación** de esos datos.

Niveles de especificación, implementación y uso de un TAD



Especificación informal de un TAD

TAD Nombre_del_tipo

VALORES: valores que pueden tomar los datos del tipo

OPERACIONES: nombre de las operaciones que los manipulan

Para cada operación (SINTAXIS y SEMÁNTICA):

Nombre_de_operación (tipo_de_argumento) → tipo_de_resultado

{**Objetivo:** Descripción de la operación

Entrada: Descripción de los datos de entrada

Salida: Indica qué es lo que retorna la operación al invocarla

Precondiciones: Posibles excepciones. Características que tendrán que reunir los datos de entrada para que se realice bien la tarea.

Poscondiciones: Indica un efecto lateral en la invocación a una función. Afirmaciones que podemos hacer sobre los datos después de que se ejecute la operación y que complete la información del objetivo y las salidas}

Pasos en la especificación de un TAD T

1. Seleccionar las operaciones, teniendo en cuenta para qué se va a utilizar dicho TAD T.
2. Clasificarlas:
   - *Constructoras*: Su resultado es de tipo T
     - *Generadoras*: Sólo con ellas es posible generar cualquier valor del tipo T y excluyendo cualquiera de ellas hay valores que no pueden ser generados.
     - *Modificadoras*: El resto
   - *Observadoras/Acceso*: Su resultado no es de tipo T.
   - *Destructoras*: Su resultado es de tipo T.

Ejemplo: Especificación del Tipo de Dato Abstracto *Racional*

VALORES

- Concepto matemático de números racionales, es decir, un par de números enteros tal que el primero es el numerador y el segundo el denominador.

OPERACIONES (Sintáxis y Semántica)

- Generadoras
  - `Crear_Racional (n,d:entero)` → `Racional`
    {*Objetivo*: Crear un número racional
    Entrada:
    n: numerador del nuevo racional
    d: denominador del nuevo racional
    Salida: El número racional creado}
  - `Suma (r1,r2:Racional)` → `Racional`
    {*Objetivo*: Calcula la suma de dos número racionales
    Entrada: r1, r2: números racionales a sumar
    Salida:
    Un nuevo racional que contiene la suma de los números a la entrada}
- Observadoras
  - `Numerador (Racional)` → `Entero`
    {*Objetivo*: Obtener el numerador de un número racional
    Entrada:
    Número racional del que obtener el numerador
    Salida: numerador del número a la entrada}
  - `Denominador (Racional)` → `Entero`
    {*Objetivo*: Obtener el denominador de un número racional
    Entrada:
    Número racional del que obtener el denominador
    Salida: denominador del número a la entrada}

Definición Ampliada de Tipo de Abstracto Dato (TAD)

Ghezzi (1987):

Un nuevo tipo de dato se considera un TAD sólo si:

- El lenguaje proporciona algún método para permitir **asociar** la **representación** de los datos con las **operaciones** que los manipulan.
- La representación del nuevo tipo de dato así como la implementación de las operaciones pueden permanecer **ocultas** al resto de los módulos que los utilizan.

Es decir, para construir un tipo de dato abstracto, debemos ser capaces de:

- Exportar una definición de tipo.
- Proporcionar un conjunto de operaciones que puedan usarse para manipular los ejemplares de tipo.
- Proteger los datos asociados con el tipo de tal manera que se pueda operar con ellos sólo mediante las operaciones provistas.

Estructura de una unit en Pascal

unit <nombre_unidad>;

interface

<clausula uses>

<constantes, tipos y variables publicas>

<cabeceras de procedimientos y funciones>

implementation

<clausula uses>

<constantes, tipos y variables privadas>

<procedimientos/funciones privadas>

<cuerpos procedimientos/funciones publicas>

begin

<secuencia de inicializacion >

end.

Ejemplo de `Tipo de dato` como `conjunto de valores`

```
program ejemplo1;

type
    Racional = record
                   num, den: integer
    end;

var
    r1,r2,r3,r4,s:Racional;

begin
    r1.num:=2; r1.den:=3; r2.num:=5;
    r2.den:=7;

    r3.num:=7; r3.den:=8; r4.num:=5;
    r4.den:=0;

    (* s suma de r1 y r2 *)
    s.num:=r1.num * r2.den + r2.num * r1.den;
    s.dem:=r1.den * r2.den;
    writeln('la suma es ', s.num,'/',s.den);

    (* s suma de r3 y r4 *)
    s.num:=r3.num * r4.den + r4.num * r3.den;
    s.dem:=r3.den * r4.den;
    writeln('la suma es ', s.num,'/',s.den);
end.
```

Ejemplo de `Tipo de dato` como `conjunto de valores y operaciones`

```
program ejemplo2;
type Racional = record
                    num,den:integer
                end;
var  r1,r2,s:Racional;

     function Crear_Racional (n,d:integer):Racional;
     (* Operacion para crear un racional *)
     var temp:Racional;
     begin
          if d=0 then writeln('Error: el denominador no puede ser 0')
          else begin
             temp.num:=n; temp.den:=d;
             Crear_Racional:=temp;
         end;
     end;

     function Numerador (r:Racional):integer;
     (* Operacion que retorna el numerador de un racional *)
     begin
         Numerador:=r.num
     end;

     function Denominador (r:Racional):integer;
     (* Operacion que retorna el denominador de un racional *)
     begin
         Denominador:=r.den
     end;

     function Suma (r1,r2:Racional):racional;
     (* Operacion que retorna la suma de dos racionales *)
     var s:racional;
     begin
         s.num:=r1.num * r2.den + r2.num * r1.den;
         s.den:=r1.den * r2.den;
         Suma:=s;
     end;

begin
     r1:=Crear_Racional(2,3); r2:=Crear_Racional(5,7);
     r3:=Crear_Racional(7,3); r4:=Crear_Racional(5,4);

     s:=Suma(r1,r2);
     writeln('La suma es ', Numerador(s),'/',Denominador(s));

     s:=Suma(r3,r4);
     writeln('La suma es ', Numerador(s),'/',Denominador(s));
end.
```

Encapsulando y compilando de forma independiente...utilizando como estructura de datos un registro...

```
Unit UnitRacional;
Interface
    type
        Racional = record
                num,den:integer
        end;
    function Crear_Racional (n,d:integer):Racional;
    function Numerador (r:Racional):integer;
    function Denominador (r:Racional):integer;
    function Suma (r1,r2:Racional):racional;

Implementation
    function Crear_Racional (n,d:integer):Racional;
    (* Operacion para crear un racional *)
    var temp:Racional;
    begin
        temp.num:=n; temp.den:=d; Crear_Racional:=temp
    end;

    function Numerador (r:Racional):integer;
    (* Operacion que retorna el numerador de un racional *)
    begin
        Numerador:=r.num
    end;

    function Denominador (r:Racional):integer;
    (* Operacion que retorna el denominador de un racional *)
    begin
        Denominador:=r.den
    end;

    function Suma (r1,r2:Racional):racional;
     (* Operacion que retorna la suma de dos racionales *)
     var s:racional;
     begin
          s.num:=r1.num * r2.den + r2.num * r1.den;
          s.den:=r1.den * r2.den;
          Suma:=s;
     end;
end.
```

...utilizando como estructura de datos un puntero a un registro...

```
Unit UnitRacional;
Interface
    type
        Racional = ^datos;
        datos = record
                    num,den:integer
                end;
    function Crear_Racional (n,d:integer):Racional;
    function Numerador (r:Racional):integer;
    function Denominador (r:Racional):integer;
    function Suma (r1,r2:Racional):racional;

Implementation
    function Crear_Racional (n,d:integer):Racional;
    var temp:Racional;
    begin
            new(temp);
            temp^.num:=n; temp^.den:=d; Crear_Racional:=temp
    end;

    function Numerador (r:Racional):integer;
    begin
        Numerador:=r^.num
    end;

    function Denominador (r:Racional):integer;
    begin
        Denominador:=r^.den
    end;

    function Suma (r1,r2:Racional):racional;
     var s:racional;
     begin
        new(s);
        s^.num:=r1^.num * r2^.den + r2^.num * r1^.den;
        s^.den:=r1^.den * r2^.den;
        Suma:=s;
     end;
end.
```

El programa que utiliza el tipo de datos es independiente de la representación del tipo de dato

```
program ejemplo3;

uses UnitRacional;
(*uso del tipo de dato Racional*)

var
    r1,r2,s:Racional;

begin
    r1:=Crear_Racional(2,3); r2:=Crear_Racional(5,7);

    (* s es la suma de r1 y r2 *)
     s:=Suma(r1,r2);
     writeln('La suma es ', Numerador(s),'/',Denominador(s));
end.
```