Tipos Abstractos de Datos (TAD)
Introducción y Justificación
Informática II

Departamento de Sistemas Telemáticos y Computación (GSyC)

Octubre de 2019

©2016-2019 Grupo de Sistemas y Comunicaciones.
Algunos derechos reservados.
Este trabajo se distribuye bajo la licencia
Creative Commons Attribution Share-Alike
disponible en http://creativecommons.org/licenses/by-sa/2.1/es

Ejemplo: .ads tradicional

```
with Ada.Strings.Unbounded;

package Lists is

   package ASU renames Ada.Strings.Unbounded;

   type Cell;
   type Cell_A is access Cell;

   type Cell is record
      Name : ASU.Unbounded_String;
      Count: Natural := 0;
      Next : Cell_A;
   end record;


   -- Interfaz
   procedure Add (List   : in out Cell_A;
                  A_Name : in     ASU.Unbounded_String;
                  A_Count: in Natural);

   procedure Print_All (List: in Cell_A);

end Lists;
```

Ejemplo: Implementación

```
with Ada.Text_IO;

package body Lists is
   procedure Add (List   : in out Cell_A;
                  A_Name : in     ASU.Unbounded_String;
                  A_Count: in Natural) is
      P_Aux : Cell_A;
   begin
      P_Aux := new Cell;
      P_Aux.all.Name  := A_Name;
      P_Aux.all.Count := A_Count;
      P_Aux.all.Next  := List;
      List := P_Aux;
   end Add;

   procedure Print_All (List: in Cell_A) is
      P_Aux : Cell_A;
   begin
      P_Aux := List;
      while P_Aux /= null loop
         Ada.Text_IO.Put_Line (ASU.To_String(P_Aux.all.Name) &
                               ": " &
                               P_Aux.all.Count'Img);
         P_Aux := P_Aux.all.Next;
      end loop;
   end Print_All;
end Lists;
```

Ejemplo: Uso del paquete en un programa

```
with Ada.Text_IO;
with Ada.Strings.Unbounded;

with Lists;

procedure Cliente is
  package ASU renames Ada.Strings.Unbounded;

  Mi_Lista: Lists.Cell_A;
begin

  Lists.Add (Mi_Lista, ASU.To_Unbounded_String("Pepe"), 39);
  Lists.Add (Mi_Lista, ASU.To_Unbounded_String("Juan"), 22);
  Lists.Add (Mi_Lista, ASU.To_Unbounded_String("Lola"), 19);

  Ada.Text_IO.Put_Line ("Contenidos de la lista:");
  Lists.Print_All (Mi_Lista);

  Mi_Lista.Next := null;
  Ada.Text_IO.New_Line;

  Ada.Text_IO.Put_Line ("Contenidos de la lista:");
  Lists.Print_All (Mi_Lista);

end Cliente;
```

Ejemplo: Uso del paquete en un programa

```
with Ada.Text_IO;
with Ada.Strings.Unbounded;

with Lists;

procedure Cliente is
  package ASU renames Ada.Strings.Unbounded;

  Mi_Lista: Lists.Cell_A;
begin

  Lists.Add (Mi_Lista, ASU.To_Unbounded_String("Pepe"), 39);
  Lists.Add (Mi_Lista, ASU.To_Unbounded_String("Juan"), 22);
  Lists.Add (Mi_Lista, ASU.To_Unbounded_String("Lola"), 19);

  Ada.Text_IO.Put_Line ("Contenidos de la lista:");
  Lists.Print_All (Mi_Lista);

  Mi_Lista.Next := null;
  Ada.Text_IO.New_Line;

  Ada.Text_IO.Put_Line ("Contenidos de la lista:");
  Lists.Print_All (Mi_Lista);

end Cliente;
```

Ejemplo: Uso erróneo paquete en un programa

```
with Ada.Text_IO;
with Ada.Strings.Unbounded;

with Lists;

procedure Cliente is
  package ASU renames Ada.Strings.Unbounded;

  Mi_Lista: Lists.Cell_A;
begin

  Lists.Add (Mi_Lista, ASU.To_Unbounded_String("Pepe"), 39);
  Lists.Add (Mi_Lista, ASU.To_Unbounded_String("Juan"), 22);
  Lists.Add (Mi_Lista, ASU.To_Unbounded_String("Lola"), 19);

  Ada.Text_IO.Put_Line ("Contenidos de la lista:");
  Lists.Print_All (Mi_Lista);

  Mi_Lista.Next := null;
  Ada.Text_IO.New_Line;

  Ada.Text_IO.Put_Line ("Contenidos de la lista:");
  Lists.Print_All (Mi_Lista);

end Cliente;
```

Ejemplo: Compilación y ejecución

```
$ gnatmake cliente.adb
gcc -c cliente.adb
gcc -c lists.adb
gnatbind -x cliente.ali
gnatlink cliente.ali
$ ./cliente
Contenidos de la lista:
Lola:  19
Juan:  22
Pepe:  39

Contenidos de la lista:
Lola:  19
$
```

¿Qué ha pasado?

El programador ha destrozado la lista accediendo a su estructura interna.

¿Por qué?

- Incompetencia.
- Error inadvertido: copy-paste, etc.
- Falta de conocimiento.
- Falta de comprensión del .ads.
  - El programador del paquete y del programa suelen ser diferentes.
  - Un mismo programador, pero 6 meses más tarde, es como si fuera otro programador.

¿Qué ha pasado?

El programador ha destrozado la lista accediendo a su estructura interna.

¿Por qué?

- Incompetencia.
- Error inadvertido: copy-paste, etc.
- Falta de conocimiento.
- Falta de comprensión del .ads.
  - El programador del paquete y del programa suelen ser diferentes.
  - Un mismo programador, pero 6 meses más tarde, es como si fuera otro programador.
- **Porque se puede.**

¿Cómo evitarlo?

Los errores humanos no se pueden evitar, pero podemos hacer que el compilador nos impida cometerlos.

Ejemplo: Definición de un TAD con private (1)

```
with Ada.Strings.Unbounded;

package Lists is

   package ASU renames Ada.Strings.Unbounded;

   type Cell_A is private;

   -- Interfaz
   procedure Add (List   : in out Cell_A;
                  A_Name : in     ASU.Unbounded_String;
                  A_Count: in Natural);

   procedure Print_All (List: in Cell_A);

private
   type Cell;
   type Cell_A is access Cell;

   type Cell is record
      Name : ASU.Unbounded_String;
      Count: Natural := 0;
      Next : Cell_A;
   end record;
end Lists;
```

Ejemplo: Definición de un TAD con private (2)

```
$ gnatmake cliente.adb
gcc -c cliente.adb
cliente.adb:19:03: invalid prefix in selected component "Mi_Lista"
gnatmake: "cliente.adb" compilation error
```

- Al haber declarado como privado el tipo `Cell_A`, el cliente no puede usar variables del tipo `Cell_A` como punteros, ¡aunque en realidad lo son!.
  - De hecho, podríamos cambiar el nombre del TAD para que refleje cómo se pretende usar ese tipo y no cómo está implementado:
    `type List_Type is private;`
    mejor que
    `type Cell_A is private;`
- Ahora los clientes sólo pueden hacer lo siguiente con un objeto de tipo Cell_A:
  - Asignarlo `:=`
  - Compararlo `=`, `/=`
  - Lo que nos deje la interfaz (`Add`, `Print_All`)

Tipos privados en Ada (1)

Tipos privados

En la especificación de un paquete se puede declarar un tipo como private.
La declaración completa del tipo se hace al final del paquete, en la sección private del paquete.

```
package P is
   -- Representación del TAD NO visible para clientes del paquete
   type T is private;
   -- Interfaz definida para el TAD
   procedure P (e: T);
   function  F (e1: T; e2: T) return T;
   ...

private
   -- Representación del TAD visible en el cuerpo del paquete
   type T is new Integer range 1..10;
end P;
```

Tipos privados en Ada (2)

- La definición completa del tipo NO puede ser utilizada por los **clientes** del paquete en el que se define el tipo privado. Tan sólo son accesibles las declaraciones que aparecen en la parte pública de la especificación.
- Los tipos privados tienen sólo 3 operaciones predefinidas: asignación/copia `:=` y los comparadores de igualdad `=` y desigualdad `/=`
  - Ejemplo: Si es un entero, el cliente no puede usar los operadores aritméticos
  - Ejemplo: Si es un registro/array, el cliente no puede acceder a sus componentes/elementos, salvo que algún subprograma de la interfaz del TAD lo permita
- ¿Qué más operaciones se puede realizar con los objetos del tipo privado? ¡Sólo las que se definan en la interfaz!

Tipos limited private

Tipos limited private

En ocasiones ni siquiera queremos que esté definida la asignación ni las comparaciones de igualdad y desigualdad.
Para ello se define el tipo como limited private en lugar de definirlo como private.

Ahora sólo podemos hacer lo siguiente con un tipo *limited private*:

- Lo que nos deje la interfaz (`Add, Print_All`, en nuestro ejemplo)

¿Por qué?

- La asignación de punteros sólo copia la dirección de memoria del puntero. El programador podría pensar que se está copiando la lista entera, pero no es cierto.
- La comparación de punteros sólo compara la dirección de memoria de los punteros. El programador podría pensar que se están comparando todos y cada uno de los elementos de la lista, pero no es cierto.

Ejemplo: Definición de un TAD con limited private (1)

```
with Ada.Strings.Unbounded;

package Lists is

   package ASU renames Ada.Strings.Unbounded;

   type Cell_A is limited private;

   -- Interfaz
   procedure Add (List   : in out Cell_A;
                  A_Name : in     ASU.Unbounded_String;
                  A_Count: in Natural);

   procedure Print_All (List: in Cell_A);

private
   type Cell;
   type Cell_A is access Cell;

   type Cell is record
      Name : ASU.Unbounded_String;
      Count: Natural := 0;
      Next : Cell_A;
   end record;
end Lists;
```

Ejemplo: Definición de un TAD con limited private (2)

Para hacer copias tenemos que añadir el procedimiento `Lists.Copy`:

```
-- Devuelve una copia de todos los elementos de la lista List en Copy_List
procedure Copy (List: in Cell_A; Copy_List: out Cell_A) is
   P_Aux : Cell_A;
begin
   P_Aux := List;
   while P_Aux /= null loop
      Add (Copy_List, P_Aux.all.Name, P_Aux.all.Count);
      P_Aux := P_Aux.all.Next;
   end loop;
end Copy;
```

En un programa cliente:

```
with Ada.Text_IO;
with Ada.Strings.Unbounded;
with Lists;
procedure Cliente is
  package ASU renames Ada.Strings.Unbounded;
  Mi_Lista, Otra_Lista: Lists.Cell_A;
begin
  Lists.Add (Mi_Lista, ASU.To_Unbounded_String("Pepe"), 39);

  -- La siguiente sentencia no compila por ser limited private
  Otra_Lista := Mi_Lista;

  -- Con copy sí:
  Lists.Copy (Mi_Lista, Otra_Lista);
end Cliente;
```