Encapsulation

university.adacore.com

Typical problem

- **Having the full implementation of the types accessible is error-prone**

```
package Stacks is

   type Stack_Data is array (1 .. 100) of Integer;

   type Stack_Type is record
      Max  : Integer := 0;
      Data : Stack_Data;
   end record;

   procedure Push
      (Stack : in out Stack_Type; Val : Integer);

   procedure Pop
      (Stack : in out Stack_Type; Val : out Integer);

end Stacks;
```

```
procedure Main is
   S : Stacks.Stack_Type
   V : Integer;
begin
   Push (S, 15);
   S.Max := 10;
   Pop (S, V);
end Main;
```

- **But the compiler needs to have access to the representation (needs to know how much memory is to be used)**
- **So the representation has to stay in the specification**

Private types

- **Introducing a new section in the package specification : the private section**
  - Visible by the compiler
  - Visible by the implementation and the children
  - Non visible to the user of the package
- **In Ada, private applies to a type as a whole, not on a field by field basis**
- **In Ada, privacy is managed at package level, not at class level**

```
package Stacks is

   type Stack_Type is private;

   procedure Push
      (Stack : in out Stack_Type;
       Val   : Integer);

private

  type Stack_Data is array (1 .. 100)
      of Integer;

   type Stack_Type is record
      Max  : Integer := 0;
      Data : Stack_Data;
   end record;

end Stacks;
```

```
namespace Stacks {

   class Stack_Type {
      public:
         void Push (int val);

      private:
         int [] Data;
         int Max;
   };

}
```

Who has access to the private information?

- **Body, and child unit have access to the implementation**

```
package Stacks is
   type Stack_Type is private;

   procedure Push
      (Stack : in out Stack_Type;
       Val   : Integer);
private
   type Stack_Data is array (1 .. 100)
      of Integer;

   type Stack_Type is record
      Max  : Integer := 0;
      Data : Stack_Data;
   end record;
end Stacks;

package body Stacks is
  procedure Push
      (Stack : in out Stack_Type;
       Val   : Integer)
  is
  begin
      Stack.Data (Stack.Max + 1) := Val;
      Stack.Max := Stack.Max + 1;
  end Push;
end Stacks;
```

```
package Stacks.Utils is
   procedure Empty
      (Stack : in out Stack_Type);
end Stacks.Utils;

package body Stack.Utils is
   procedure Empty
      (Stack : in out Stack_Type) is
   begin
      Stack.Max := 0;
   end Stack.Utils;
end Stack.Utils;
```

```
with Stacks;        use Stacks;
with Stacks.Utils; use Stacks.Utils;

procedure Main is
   S : Stack_Type;
begin
   Push (S, 10);
   Empty (S);
   S.Max := 0;
end Main;
```

What can you do with a private type?

- **From the user perspective, a private type is equivalent to a null record**
- **It can be used for**
  - Variables, parameters and components declarations
  - Copies
  - Comparisons

```
package Stacks is

   type Stack_Type is private;
   procedure Push
      (Stack : in out Stack_Type;
       Val   : Integer);

private

  […]

end Stacks;
```

```
procedure Main is
   S1, S2 : Stacks.Stack_Type;
begin
   Push (S1, 15);
   S2 := S1;

   Push (S2, 0);
   Push (S1, 0);

   if S1 = S2 then
      Push (S1, 1);
   end if;
end Main;
```

How can a private type be implemented?

- **A "simple" private type can be implemented by any type giving at least the same level of capabilities**
  - The type must allow variable declarations without the need for constraints, it has to be definite (e.g. no unconstrained arrays)
  - The type must allow copy and comparison (e.g. no limited types)

```
package Stacks is

   type Stack_Type is private;
```

```
private

   type Stack_Type is range 1 .. 10;

end Stacks;
```

```
private

   type Stack_Type is record
      V : Integer;
   end record;

end Stacks;
```

```
private

   type Stack_Type is array
      (Integer range 1 .. 10);
       of Integer;

end Stacks;
```

```
private

   type Stack_Type (Size : Integer) is record
      V : Integer;
   end record;
end Stacks;
```

```
private

   type Stack_Type is array (Integer range <>)
       of Integer;

end Stacks;
```

How can a private type be implemented?

- **An "indefinite" private type can be implemented by any type that can be implemented by "simple" private type as well as indefinites**
  - But the user needs to consider it as indefinite (no declaration without initialization)

```
package Stacks is

   type Stack_Type (<>) is private;
```

```
private

   type Stack_Type is range 1 .. 10;

end Stacks;
```

```
private

   type Stack_Type is record
      V : Integer;
   end record;

end Stacks;
```

```
private

   type Stack_Type is array
      (Integer range 1 .. 10);
       of Integer;

end Stacks;
```

```
private

   type Stack_Type (Size : Integer) is record
      V : Integer;
   end record;
end Stacks;
```

```
private

   type Stack_Type is array (Integer range <>)
       of Integer;

end Stacks;
```

Public Discriminants on Private Types

- **It's possible to specify the discriminants of a private type**

```
package Stacks is

  type Stack_Type (Size : Integer) is private;

private

   type Stack_Type (Size : Integer) is record
      V : Integer;
   end record;
end Stacks;
```

Deferred private constants

- It's useful to declare constants visible in the public view
- Values can't be given before the representation is accessible – so constants of private types have a public and a private view

```
package Stacks is
   type Stack_Type is private;

   Empty_Stack : constant Stack_Type;

private

   type Stack_Data is array (1 .. 100)
      of Integer;

   type Stack_Type is record
      Max  : Integer := 0;
      Data : Stack_Data;
   end record;

   Empty_Stack : constant Stack_Type :=
      (0, (others => 0));
end Stacks;
```

Private part is not only for private types

- **Any kind of declaration can be provided in the private part of the package**
- **Entities declared only in the private part are not reachable at all for the client**

```
package P is
   --  public part of the specification
   --  declaration of subprograms, variables exceptions, tasks…
   --  visible to the external user
   --  used by the compiler for all dependencies
private
   --  private part of the specification
   --  declaration of subprograms, variables exceptions, tasks…
   --  visible to the children and the implementation
   --  used by the compiler for all dependencies
end P;

package body P is
  --  body
  --  declaration of subprograms, variables exceptions, tasks…
  --  implementation of subprograms
  --  used for the compiler from P
  --  in certain cases, visible from the compiler for dependencies
end P;
```

Quiz

Is this correct? (1/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
package P is
   type T is private;

   type T is range 1 .. 10;
end P;
```

Is this correct? (1/10)

NO

```
package P is
   type T is private;

   type T is range 1 .. 10;
end P;
```

"private" keyword is missing

Is this correct? (2/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
package P is

   type T is private;

private

   type T is range 1 .. 10;

end P;
```

```
with P; use P;

procedure Main is
   V : T;
begin
   V := 0;
end Main;
```

Is this correct? (2/10)

NO

```
package P is

   type T is private;

private

   type T is range 1 .. 10;

end P;
```

```
with P; use P;

procedure Main is
   V : T;
begin
   V := 0;
end Main;
```

Main has no visibility over the fact that V is Integer

Is this correct? (3/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
package P is
   type T is private;
private
   type T is range 0 .. 10;
end P;
```

```
with P; use P;

procedure P.Main is
   V : T;
begin
   V := 0;
end P.Main;
```

Is this correct? (3/10)

YES

```
package P is
   type T is private;
private
   type T is range 0 .. 10;
end P;
```

```
with P; use P;

procedure P.Main is
   V : T;
begin
   V := 0;
end P.Main;
```

P.Main is child of P, so it has visibility over the private section

Is this correct? (4/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
package P is
   type T is private;
   Zero : constant T := 0;
private
   type T is range 0 .. 10;
end P;
```

```
with P; use P;

package P2 is
   type T2 is record
      F : T;
   end record;
end P2;
```

```
with P;  use P;
with P2; use P2;

procedure Main is
   V : T2;
begin
   V.F := Zero;
end Main;
```

Is this correct? (4/10)

NO

```
package P is
   type T is private;
   Zero : constant T := 0;
private
   type T is range 0 .. 10;
end P;
```

Compilation error.

The declaration of the constant Zero has no visibility over the representation of T, it can't be initialized. What should be done instead is:

```
package P is
  type T is private;
  Zero : constant T;
private
  type T is range 0 .. 10;
  Zero : constant T := 0;
end P;
```

```
with P; use P;

package P2 is
   type T2 is record
      F : T;
   end record;
end P2;
```

```
with P;  use P;
with P2; use P2;

procedure Main is
   V : T2;
begin
   V.F := Zero;
end Main;
```

Is this correct? (5/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
package P is
   type T is private;
private
   type T is range 0 .. 10;
   Zero : constant T := 0;
end P;
```

```
with P; use P;

procedure P.Main is
   V : T;
begin
   V := Zero;
end P.Main;
```

```
with P; use P;

procedure Main is
   V : T;
begin
   V := Zero;
end Main;
```

Is this correct? (5/10)

NO

```
package P is
   type T is private;
private
   type T is range 0 .. 10;
   Zero : constant T := 0;
end P;
```

```
with P; use P;

procedure P.Main is
   V : T;
begin
   V := Zero;
end P.Main;
```

```
with P; use P;

procedure Main is
   V : T;
begin
   V := Zero;
end Main;
```

Everything is fine in P.Main, it has visibility over P.
The with and use clauses are redundant. However, Main doesn't have visibility over the private part of P, so it can't use its entities.

Is this correct? (6/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
package P is
   type T is private;
private
   type T is array (Integer range <>) of Integer;
end P;
```

```
procedure P.Main is
   V : T (1 .. 10);
begin
   V (1) := 0;
end P.Main;
```

Is this correct? (6/10)

NO

```
package P is
   type T is private;
private
   type T is array (Integer range <>) of Integer;
end P;
```

Compilation error.

T has an indefinite full view, but a definite partial view. This is inconsistent, as clients are not aware of the fact that they should constrain the object.

```
procedure P.Main is
   V : T (1 .. 10);
begin
   V (1) := 0;
end P.Main;
```

The public view of T should be :

```
package P is
   type T (<>) is private;
private
   type T is array (Integer range <>) of Integer;
end P;
```

Is this correct? (7/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
package P is
   type T (<>) is private;
private
   type T is array (Integer range 1 .. 10) of Integer;
end P;
```

```
with P; use P;

procedure Main is
   V : T;
begin
   null;
end Main;
```

Is this correct? (7/10)

NO

```
package P is
   type T (<>) is private;
private
   type T is array (Integer range 1 .. 10) of Integer;
end P;
```

```
with P; use P;

procedure Main is
   V : T;
begin
   null;
end Main;
```

The private definition is fine, but not the declaration –
T is not constrained in Main (even if the real type doesn't have to be, the private view is unconstrained)

Is this correct? (8/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
package P is
   type T is private;

   One : constant T;
private
   type T is range 0 .. 10;
   One : constant T := 0;
end P;
```

```
with P; use P;

procedure Main is
   Val : T;
begin
   Val := One + One;
end Main;
```

Is this correct? (8/10)

NO

```
package P is
   type T is private;

   One : constant T;
private
   type T is range 0 .. 10;
   One : constant T := 0;
end P;
```

```
with P; use P;

procedure Main is
   Val : T;
begin
   Val := One + One;
end Main;
```

Compilation error.
There is no applicable operator "+" for private type "T" defined at p.ads:2

Is this correct? (9/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
package P is
   type T is private;
private
   type T is range 0 .. 10;
end P;
```

```
package P.Constants is
   Zero : constant T := 0;
   One  : constant T := 1;
end P.Constants;
```

```
with P;            use P;
with P.Constants;  use P.Constants;

procedure Main is
   V : T := One;
begin
   null;
end Main;
```

Is this correct? (9/10)

NO

```
package P is
   type T is private;
private
   type T is range 0 .. 10;
end P;
```

```
package P.Constants is
   Zero : constant T := 0;
   One  : constant T := 1;
end P.Constants;
```

```
with P;            use P;
with P.Constants;  use P.Constants;

procedure Main is
   V : T := One;
begin
   null;
end Main;
```

Compilation error.

The public view of P.Constants has only visibility of the public view of P.

It cannot reveal things that are hidden. So the constant cannot be given a literal here.

What would work is to declare them in the private part

```
package P.Constants is
   Zero : constant T;
   One  : constant T;
private
   Zero : constant T := 0;
   One  : constant T := 1;
end P.Constants;
```

Is this correct? (10/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
package P is

   type T1 is private;

   type T2 is record
      Private_Part : T1;
      F1, F2 : Integer;
   end record;

private

   type T1 is record
      F1, F2  : Float;
   end record;

end P;
```

Is this correct? (10/10)

YES

```
package P is

   type T1 is private;

   type T2 is record
      Private_Part : T1;
      F1, F2 : Integer;
   end record;

private

   type T1 is record
      F1, F2  : Float;
   end record;

end P;
```

OK, pattern to hide part of a record

university.adacore.com