Packages

Presented by Quentin Ochem

university.adacore.com

The Ada Package

- A package is the base of software architecture in Ada
- It's a semantic entity checked by the compiler
- It separates clearly a specification and an implementation

```
-- p.ads

package P is
   procedure Proc;
end P;

-- p.adb

package body P is
   procedure Proc is
   begin
      null;
   end Proc;
end P;
```

```
/* p.h */

#ifndef __P_H__
#define __P_H__

void Proc ();

#endif

/* p.c */

int V;
void Proc () {
}
```

General Structure of a Package

```
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
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;
```

- **Entities should be put in the body except if they have to be exported**
- **The body is easier to change than the specification**

Example

```
package Int_Stack is

   type Int_Stack_Array is array (Integer range 1 .. 100) of Integer;

   type Stack_Int_Type is record
      Data : Int_Stack_Array;
      Last : Integer := 0;
   end record;

   procedure Push (S : in out Stack_Int_Type; Val : Integer);

   function Pop (S : in out Stack_Int_Type) return Integer;

   Empty_Stack : constant Stack_Int_Type :=
      (Data => (others => 0), Last => 0);

end Int_Stack;
```

```
package body Int_Stack is

   procedure Push (S : in out Stack_Int_Type; Val : Integer) is
   begin
      S.Last := S.Last + 1;
      S.Data (S.Last) := Val;
   end Push;

   function Pop (S : in out Stack_Int_Type) return Integer is
   begin
      S.Last := S.Last - 1;
      return S.Data (S.Last + 1);
   end Pop;

end Int_Stack;
```

Accessing components of a package

- **Only entities declared in the public part are visible**
- **Entities are referenced through the dot notation**

```
package P1 is

   procedure Pub_Proc;

end P1;
```

```
package P2 is

   procedure Proc;

end P2;
```

```
package body P1 is

  procedure Priv_Proc;
  …
end P1;
```

```
with P1;

package body P2 is

   procedure Proc is
   begin
      P1.Pub_Proc;
      P1.Priv_Proc;
   end Proc;

end P2;
```

Child units

- **A child unit is an extension of a package**
- **Can be used to organize the namespace or break big packages into pieces**
- **Child units have visibility over parents**

```
-- p.ads
package P is

end P;
```

```
-- p-child_1.ads
package P.Child_1 is

end P.Child_1;
```

```
-- p-child_2.ads
package P.Child_2 is

end P.Child_2;
```

```
-- p-child_3.ads
package P.Child_3 is

end P.Child_3;
```

```
-- p-child_2-grand_child.ads
package P.Child_2.Grand_Child is

end P.Child_2.Grand_Child;
```

Full dependencies ("with clause")

- "With clause" defines a dependency between two packages
- Gives access to all the public declarations
- Can be applied to the spec or the body
- A dependency is normally done to a specification
- "Specification with" applies to the body
- "Specification with" applies to children
- There is no other transitivity

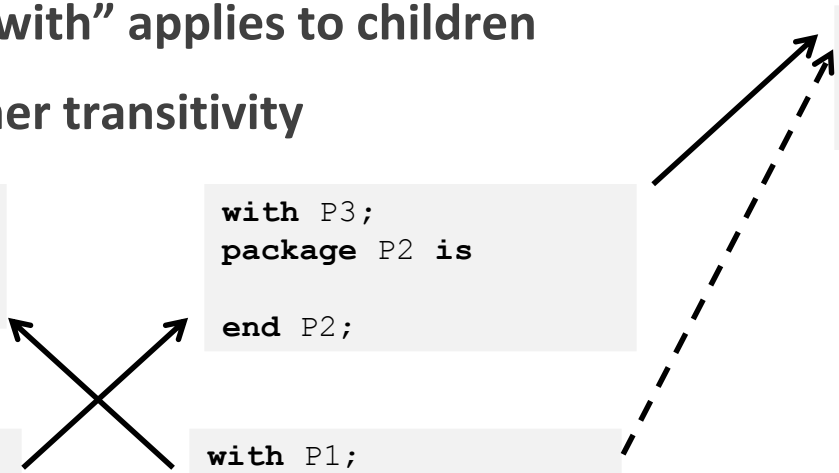
```
package P3 is

end P3;
```

```
package P1 is

end P1;
```

```
with P3;
package P2 is

end P2;
```

```
with P2;

package body P1 is

end P1;
```

```
with P1;

package body P2 is

end P2;
```



About the usage of the "with" keyword

- **With is a highly overloaded Ada reserved word**
- **It can be used to declare a dependency between two units**

```
with Ada.Text_IO;
procedure Main is …
```

- **It can be used to introduce an aspect**

```
procedure Proc;
   with Inline;
```

- **It can be used to extend a record**

```
type T2 is new T with
null record;
```

- **… and others situations as well**

Dependency shortcut ("use clause")

- **Prefix may be overkill**
- **The "use clause" allows omitting it**
- **Can introduce ambiguities**
- **Can be placed in any scope**

```
package P1 is

   procedure Proc1;
   type T is null record;

end P1;
```

```
package P2 is

   procedure Proc1;

end P2;
```

```
with P1;
with P2; use P2;

package body P3 is

   X : T;

   procedure Proc is
      use P1;
      X : T;
   begin
      Proc1;
      P1.Proc1;
      P2.Proc1;
   end Proc;

end P2;
```

A Note about Operator Symbol Visibility

- **With only a "with" clause, operators on types will not be visible!**
- **Use type will specifically allow visibility to operators of a type**
- **Prefix notation is possible too**

```
package P1 is

   type T is new Integer;

end P1;
```

```
with P1;

procedure Main is
   A, B : P1.T := 0;
begin
   A := P1."+" (A, B);
end Main;
```

```
with P1;

procedure Main is
   use type P1.T;
   A, B : P1.T := 0;
begin
   A := A + B;
end Main;
```

A Package is a High Level Semantic Entity

- **The compiler is responsible for checking structural and semantic consistency**

```
-- p.ads

package P is
   V : Integer;
   procedure Proc
      with Inline;
end P;

-- p.adb

package body P is
   procedure Proc is
   begin
      null;
   end Proc;
end P;
```

```
/* p.h */

#ifndef __P_H__
#define __P_H__

extern int V;
inline void Proc ();

#include "p.hi"
#endif

/* p.hi */

#ifndef __P_HI__
#define __P_HI__

inline void Proc () {
}

#endif

/* p.c */

int V;
```

Quiz

Is this correct? (1/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
package X_Manage is

   procedure Write (V : Integer);

   function Read return Integer;

end X_Manage ;
```

```
package body X_Manage is

   X : Integer;

   procedure Write (V : Integer) is
   begin
      X := V;
   end Write;

   procedure Read (V : out Integer) is
   begin
      V := X;
   end Read;

end X_Manage;
```

Is this correct? (1/10)

NO

```
package X_Manage is

   procedure Write (V : Integer);

   function Read return Integer;

end X_Manage ;
```

compilation error
Read is not implemented in the body

```
package body X_Manage is

   X : Integer;

   procedure Write (V : Integer) is
   begin
      X := V;
   end Write;

   procedure Read (V : out Integer) is
   begin
      V := X;
   end Read;

end X_Manage;
```

Is this correct? (2/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
package X_Manage is

   X : Integer;

   procedure Write (V : Integer);

   procedure Read (V : out Integer) is
   begin
      V := X;
   end Read;

end X_Manage ;
```

```
package body X_Manage is

   procedure Write (V : Integer) is
   begin
      X := V;
   end Write;

end X_Manage;
```

Is this correct? (2/10)

NO

```
package X_Manage is

   X : Integer;

   procedure Write (V : Integer);

   procedure Read (V : out Integer) is
   begin
      V := X;
   end Read;

end X_Manage ;
```

compilation error
a body can't be written
in a package spec

```
package body X_Manage is

   procedure Write (V : Integer) is
   begin
      X := V;
   end Write;

end X_Manage;
```

Is this correct? (3/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
package P1 is

   type T is null record;

end P1;
```

```
package P2 is

   X : P1.T;

end P2;
```

Is this correct? (3/10)

NO

```
package P1 is

   type T is null record;

end P1;
```

```
package P2 is

   X : P1.T;

end P2;
```

compilation error
"with P1;" is needed on P2 for the declaration to work

Is this correct? (4/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
package P1 is

end P1;
```

```
with P1; use P1;

package P2 is

   X : T;

end P2;
```

```
package body P1 is

   type T is null record;

end P1;
```

Is this correct? (4/10)

NO

```
package P1 is

end P1;
```

```
with P1; use P1;

package P2 is

   X : T;

end P2;
```

compilation error
T is declared in the body of P1, not reachable from the outside

```
package body P1 is

   type T is null record;

end P1;
```

Is this correct? (5/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
with P2;

package P1 is

   type T1 is null record;

   V : P2.T2;

end P1;
```

```
with P1;

package P2 is

   type T2 is null record;

   V : P1.T1;

end P2;
```

Is this correct? (5/10)

NO

```
with P2;
package P1 is
   type T1 is null record;
   V : P2.T2;
end P1;
```

```
with P1;
package P2 is
   type T2 is null record;
   V : P1.T1;
end P2;
```

compilation error
there is a circularity between P1 and P2

Is this correct? (6/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
with P2;

package P1 is

   type T1 is null record;

   V : P2.T2;

end P1;
```

```
package P2 is

   type T2 is null record;

end P2;
```

```
with P1;

package body P2 is

   X : P1.T1;

end P2;
```

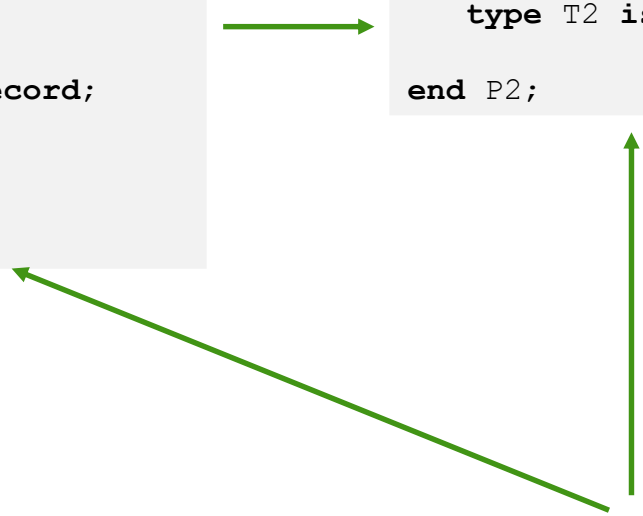
Is this correct? (6/10)

YES

```
with P2;

package P1 is

   type T1 is null record;

   V : P2.T2;

end P1;
```

```
package P2 is

   type T2 is null record;

end P2;
```

```
with P1;

package body P2 is

   X : P1.T1;

end P2;
```



there is no circularity
P1 depends only the spec of P2
the body of P2 depends on the spec of P1

Is this correct? (7/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
package Types is

   type My_Int is new Integer;

end Types;
```

```
with Types;

package Constants is

   Zero : constant P1.T := 0;
   One  : constant P1.T := 1;
   Two  : constant P1.T := One + One;

end Main;
```

Is this correct? (7/10)

NO

```
package Types is

   type My_Int is new Integer;

end Types;
```

```
with Types;

package Constants is

   Zero : constant P1.T := 0;
   One  : constant P1.T := 1;
   Two  : constant P1.T := One + One;

end Main;
```

compilation error
operators are not reachable for the lack of use clause on "Types" package

Is this correct? (8/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
package P1 is

   type T is null record;

end P1;
```

```
package P1.Child is

end P1.Child;
```

```
package body P1.Child is

   X : T;

end P1.Child;
```

Is this correct? (8/10)

YES

```
package P1 is

   type T is null record;

end P1;
```

```
package P1.Child is

end P1.Child;
```

no errors
a child package has use-visibility over its parent

```
package body P1.Child is

   X : T;

end P1.Child;
```

Is this correct? (9/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
with P1.Child;

package P1 is

   X : P1.Child.T;

end P1;
```

```
package P1.Child is

   type T is null record;

end P1.Child;
```

Is this correct? (9/10)

NO

```
with P1.Child;

package P1 is

   X : P1.Child.T;

end P1;
```

```
package P1.Child is

   type T is null record;

end P1.Child;
```

a child always depends on its parent
this create a circularity here

Is this correct? (10/10)

YES
(click on the check icon)

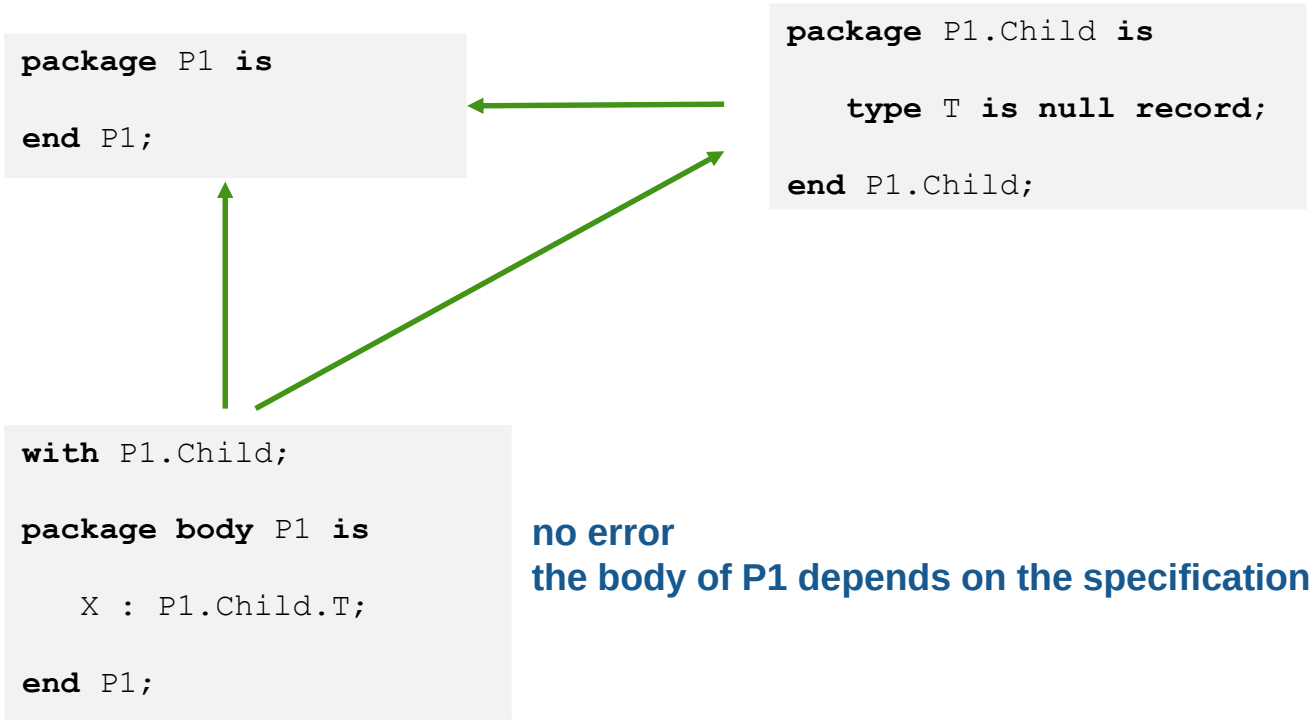
NO
(click on the error location(s))

```
package P1 is

end P1;
```

```
package P1.Child is

   type T is null record;

end P1.Child;
```

```
with P1.Child;

package body P1 is

   X : P1.Child.T;

end P1;
```

Is this correct? (10/10)



```
package P1 is

end P1;
```

```
package P1.Child is

   type T is null record;

end P1.Child;
```

```
with P1.Child;

package body P1 is

   X : P1.Child.T;

end P1;
```

no error
the body of P1 depends on the specification of P1.Child

university.adacore.com