パッケージ

Presented by Quentin Ochem

university.adacore.com

Adaパッケージ

- パッケージは，Ada言語におけるソフトウェアアーキテクチャの基礎となるものです
- コンパイラが検査を行う，意味を有したエンティティです
- 明確に，仕様と実現（implementation）に分かれます

```
-- 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 () {
}
```

パッケージの一般的な構造

```
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;
```

- エンティティは，明示的にボディ部中に置かれるべきです．エンティティを外部で利用したい場合は別です
- ボディ部は，仕様部と比べ，変更が容易です

例

```
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;
```

パッケージのコンポーネントにアクセスする

- 公開部で宣言されているエンティティのみが，（そのパッケージの利用者から）アクセスできます（可視状態）
- ドット表記を用いて，エンティティを参照することができます

```
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;
```

子単位

- 子単位は，パッケージの拡張です
- ネームスペースを組織化したり，大きなパッケージを分割するときに利用できます
- 子単位は，親に対して可視性を持ちます

```
-- 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;
```

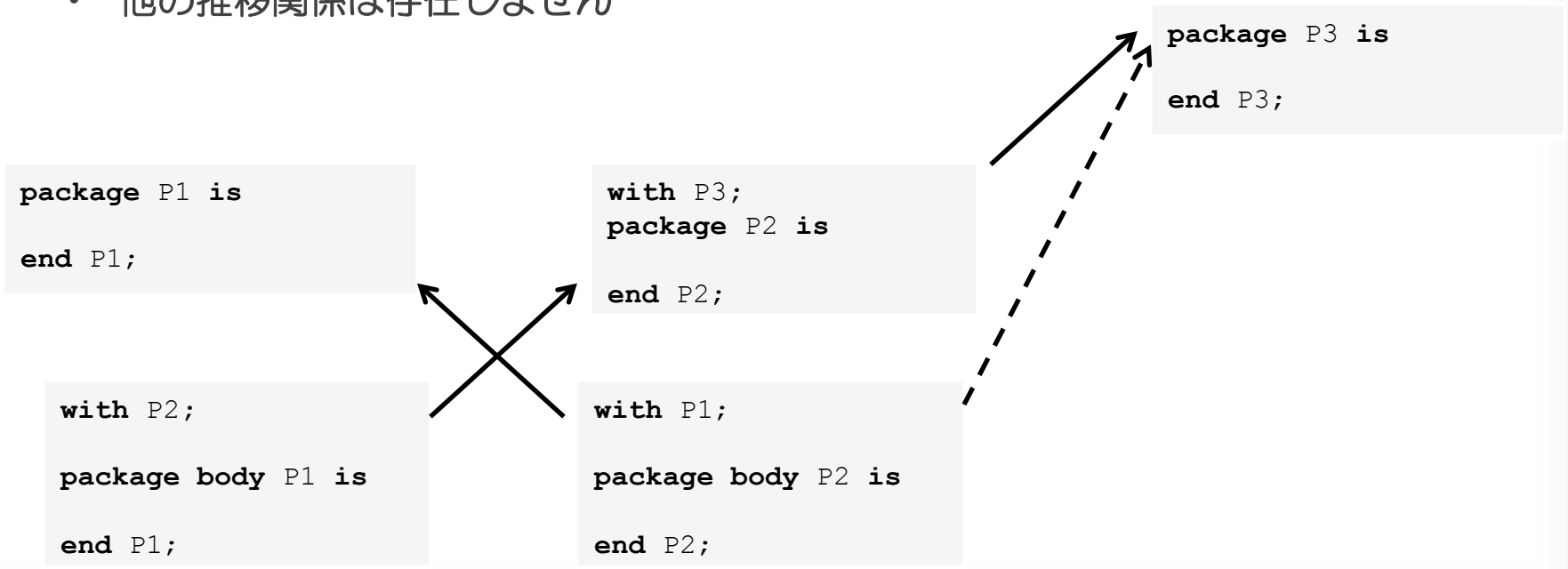
```
-- p-child_3.ads
package P.Child_3 is

end P.Child_3;
```

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

end P.Child_2.Grand_Child;
```

- 一般に，可能な限り機能を複数のパッケージに分割することは，良い習慣です

完全な依存（with 節の使用）

- with節は，2つのパッケージ間の依存を定義します
- 全ての公開宣言に対して，（パッケージ利用者は）アクセスができます
- 仕様部或いはボディ部で使用することができます
- 通常，仕様部に依存します
- 「仕様部での with節」は，そのボディ部に適用されます
- 「仕様部での with節」は，その子単位にも適用されます
- 他の推移関係は存在しません



「with」 キーワードの利用について

- 予約語 with は，Ada言語において様々な役割を持っています

- 二つのユニット間の依存性を宣言するのに用います

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

- アスペクトを導入するために用います

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

- レコード型を拡張するために用います

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

- 同様の他の場合においても利用します

依存性におけるショートカット(use 節)

- （パッケージ名の）前置にうんざりする時があります
- use 節 は，その煩雑さを取り除きます
- あいまいさが，紛れ込む可能性があります
- 全てのスコープ中に置くことがでます

```
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;
```

演算子シンボルの可視性についての注意事項

- with節だけでは，型に対する操作は，可視にはなりません
- 「use type」を用いることで，型に対する操作が可視になります．
- 前置記法も利用可能です

```
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;
```

パッケージは，高いレベルの意味を持ったエンティティである

- コンパイラは，構造的・意味的一貫性が保たれていることを確実に検査します

```
-- 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

正しいですか　(1/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
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;
```

正しいですか (1/10)

いいえ

```
package X_Manage is

   procedure Write (V : Integer);

   function Read return Integer;

end X_Manage ;
```

コンパイラエラー
Read関数はボディ部に「実現」がありません

```
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;
```

正しいですか (2/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
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;
```

正しいですか (2/10)

いいえ

```
package X_Manage is

   X : Integer;

   procedure Write (V : Integer);

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

end X_Manage ;
```

コンパイラエラー
手続きのボディ部をパッケージの仕様部に書くことはできません

```
package body X_Manage is

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

end X_Manage;
```

正しいですか (3/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
package P1 is

   type T is null record;

end P1;
```

```
package P2 is

   X : P1.T;

end P2;
```

正しいですか (3/10)

いいえ

```
package P1 is

   type T is null record;

end P1;
```

```
package P2 is

   X : P1.T;

end P2;
```

コンパイルエラー
この宣言のためには，
"with P1;" の with 節が必要になります．

正しいですか (4/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
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;
```

正しいですか (4/10)

いいえ

```
package P1 is

end P1;
```

```
with P1; use P1;

package P2 is

   X : T;

end P2;
```

コンパイルエラー
T は，P1のボディ部で宣言されています
他の場所からはアクセスすることができません

```
package body P1 is

   type T is null record;

end P1;
```

正しいですか (5/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
with P2;

package P1 is

   type T1 is null record;

   V : P2.T2;

end P1;
```

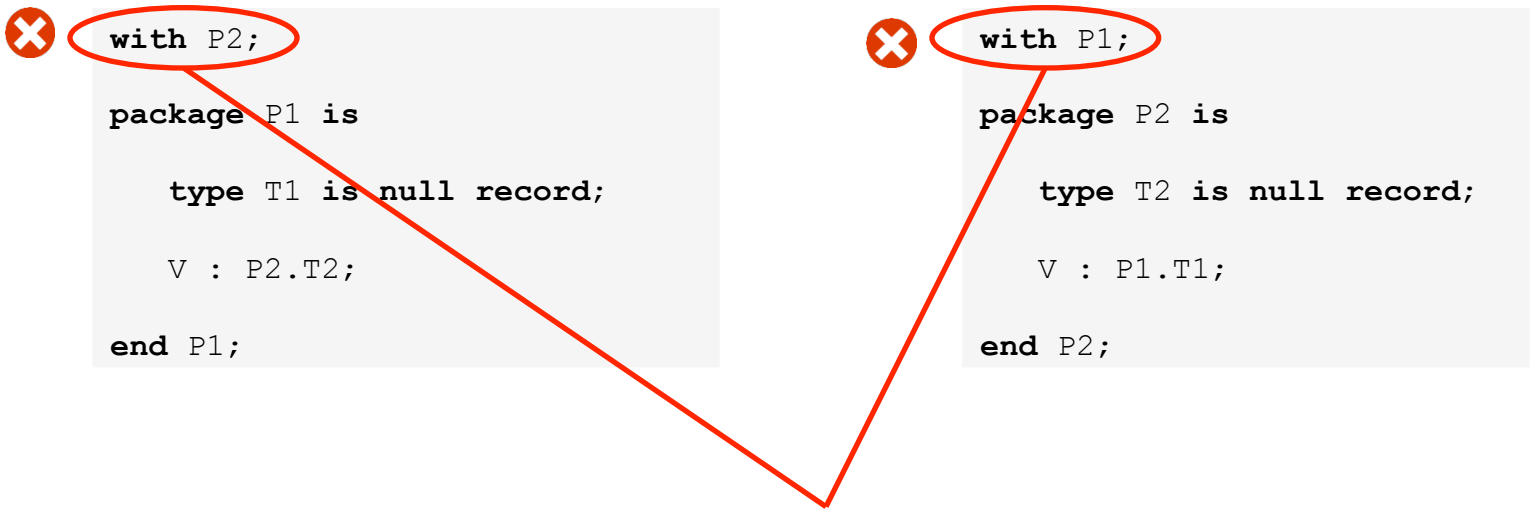
```
with P1;

package P2 is

   type T2 is null record;

   V : P1.T1;

end P2;
```

正しいですか (5/10)

いいえ

```
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;
```



コンパイルエラー
P1とP2の間で，依存が循環しています

正しいですか (6/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
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;
```

正しいですか (6/10)

はい

```
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;
```

ここでは，循環はありません
P1は，P2の仕様のみに依存しています
P2のボディ部は，P1の仕様部に依存しています

```
with P1;

package body P2 is

   X : P1.T1;

end P2;
```

正しいですか (7/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
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;
```

正しいですか (7/10)

いいえ

```
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;
```

コンパイルエラー
Types パッケージに対する Use 節がないために，この演算子を，利用できません

正しいですか (8/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
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;
```

正しいですか (8/10)

はい

```
package P1 is

   type T is null record;

end P1;
```

```
package P1.Child is

end P1.Child;
```

エラーはありません
子パッケージは，親に対して，
（Use 節と同様の）可視性を持っています

```
package body P1.Child is

   X : T;

end P1.Child;
```

正しいですか (9/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
with P1.Child;

package P1 is

   X : P1.Child.T;

end P1;
```

```
package P1.Child is

   type T is null record;

end P1.Child;
```

正しいですか (9/10)

いいえ

```
with P1.Child;

package P1 is

   X : P1.Child.T;

end P1;
```

```
package P1.Child is

   type T is null record;

end P1.Child;
```

子パッケージは，常にその親に依存しています．
従って，ここで循環が生じます

正しいですか (10/10)

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

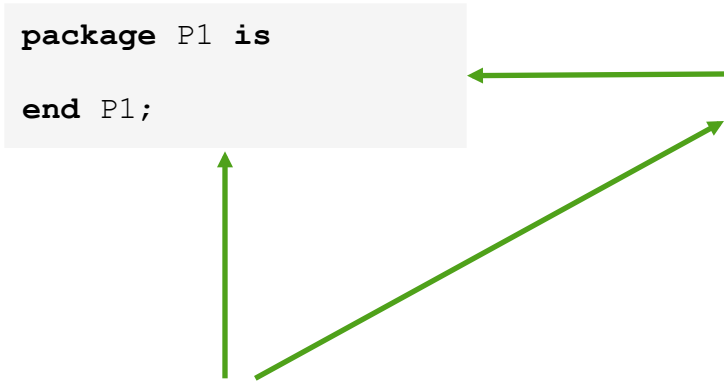
```
package P1 is

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;
```

正しいですか (10/10)

はい



```
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;
```

エラーはありません.
P1のボディ部は，P1.Childの仕様に依存しています

university.adacore.com