34 - Types privés

Programmation Concurrente - LI330
Université P. & M. Curie - année scolaire 2013/2014

PrC

La notion de type privé

- Type utilisable à l'extérieur d'une unité (i.e. un paquetage)
- Type dont l'implémentation reste inaccessible
- Type pour lequel seules deux opérations sont disponibles par défaut
  - Affectation
    - Recopie de deux zones mémoires
  - Comparaison d'égalité (= et /=)
    - Comparaison bit à bit de deux zones mémoires
- Les autres opérations permettant la manipulation du type sont explicitement définies par le concepteur de l'unité

Types privés en Ada

```
spéc_paq2  {cl_visibilité }
           package identificateur is
              {déclaration_de_types_variables_ou_sous-programmes}
           private
              {déclaration_éléments_privés}
           end identificateur ;

décl_tp_pr  type identificateur [discriminant] is private;

spéc_tp_pr   type identificateur [discriminant] is définition_de_type ;
```

- Les déclarations d'éléments privés peuvent être
  - Des types déclarés privés dans la partie publique
  - Des types non déclarés dans la partie publique
  - Des variables, constantes, procédures...
    ...non déclarées dans la partie publique

} Visible de l'extérieur

} Invisible de l'extérieur

Application à l'unité Gest_Fiches


```
-- Unité de gestion de fiches
package Gest_Fiches_P is

   -- Déclaration de la structure de données "fiche"
   type Une_Fiche is private;

   -- Définition des primitives de manipulation d'une fiche
   procedure Nouvelle_Fiche (N, P : in String;
                             F : out Une_Fiche);
   function Lire_Nom (F : in Une_Fiche) return String;
   function Lire_Prenom (F : in Une_Fiche) return String;

private
   -- Définition de la structure de données "fiche"
   type Une_Fiche is record
      Nom, Prenom : String (1..20);
      L_Nom, L_Prenom : Natural := 0;
   end record;
end Gest_Fiches_P;
```

Avantages de l'utilisation d'un type privé

- L'utilisateur du paquetage peut-il créer d'autres opérations sur le type Une_Fiche?
  - Non, il ne voit que le type, pas sa définition
- Est-ce un procédé sûr?
  - Oui, car tout changement dans la structure de données sous-jacente n'impacte que l'unité concernée
    - L'usager ne peut construire d'opération sur le type Une_Fiche qu'à partir des services de base déjà définis
    - Cela impose de bien «tailler» les services (tout service manquant devrait être intégré dans l'unité Gest_Fiches_p
  - C'est bien une encapsulation du type Une_Fiche et des services liés
- Impact sur le corps du paquetage
  - Aucun, les primitives de l'unité se programment identiquement
  - Les restrictions sur la visibilité d'un type privé concerne l'«extérieur»

Effets de bord sur la déclaration des constantes

- Déclaration de constantes
  - décl_const identificateur : constant id_type := expression;
    - Il faut différer la définition
- La définition de constantes se fait en deux temps
  - Déclaration
  - Définition (dans la partie «private»)
- On parle de «constantes différées»

Autre exemple (1)


```
with Mes_Definitions_Numeriques;
use  Mes_Definitions_Numeriques;

-- Unité de gestion de nombres complexes
package Complexe_2 is

   -- Déclaration de la structure de données "complexe"
   type Un_Complexe is private;

   -- Déclaration de  quelques constantes utiles
   Un : constant Un_Complexe;
   I  : constant Un_Complexe;
   Zero : constant Un_Complexe;

   -- Définition des primitives de manipulation d'un complexe
   function Nouveau_Complexe (Re, Im : Un_Reel) return Un_Complexe;
   function "+" (C1, C2 : Un_Complexe) return Un_Complexe;
   function "-" (C1, C2 : Un_Complexe) return Un_Complexe;
   -- Définition des E/S sur un complexe
   procedure Saisir (C : out Un_Complexe);
   procedure Afficher (C : in Un_Complexe);
```

Autre exemple (2)


```
private
   -- Définition de la structure de données "complexe"
   type Un_Complexe is record
      R, I : Un_Reel := 0.0;
   end record;

   -- Définition des constantes utiles
   Un : constant Un_Complexe := Nouveau_Complexe (1.0, 0.0);
   I  : constant Un_Complexe := (0.0, 1.0);
   Zero : constant Un_Complexe := (0.0, 0.0);

end Complexe_2;
```

Possible car on est «dans» le paquetage

Remarques sur le paquetage Complexes_2

- Avec le paquetage «Complexes», on pouvait écrire

```
Un : Un_Complexe := (1.0, 0.0);
C  : Un_Complexe;
--...
C.R := X; -- X de type Un_Reel
C.I := Y; -- Y de type Un_Reel
```

- Avec «Complexes_2», c'est devenu impossible
  - Le compilateur l'interdit!
- Dans les deux cas, les opérations d'affectation et de comparaison d'égalité sont fournies