Analyse et programmation langage ADA

Informatique 1ère année

Date : 16/02/2006

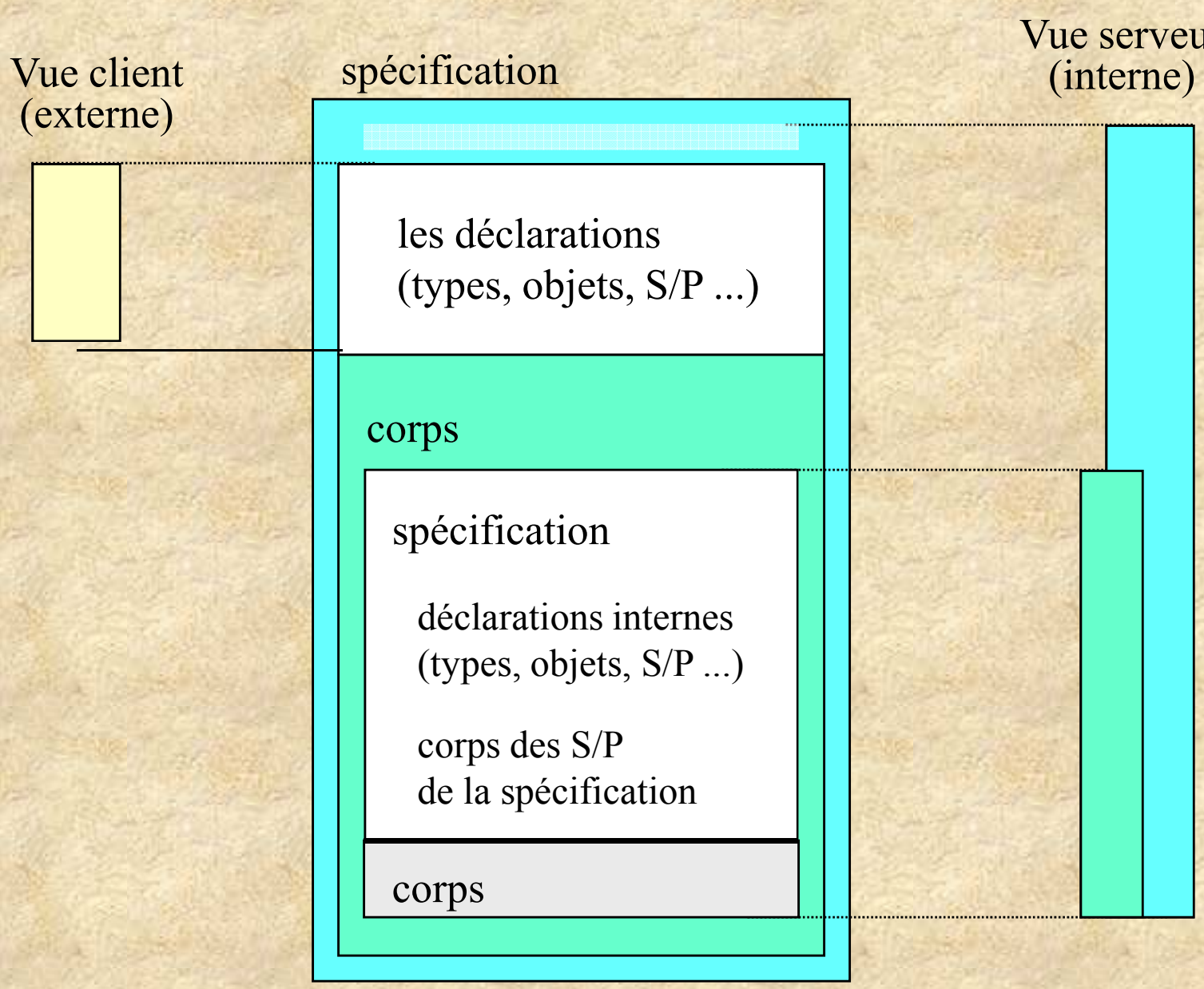
Paquetage simple

- 1 Introduction
  - 1.1 Modularité
  - 1.2 Présentation d'un paquetage
  - 1.3 Exemple
  - 1.2 Appel d'une action (procédure)
  - 1.3 Appel d'une fonction
  - 1.4 Propriétés des paquetages
- 2 Mise en place
  - 2.1 Spécification de paquetage
  - 2.2 Définition du corps de paquetage
  - 2.3 Utilisation de paquetage : Clause use
- 3 Exemple de programmation modulaire
  - 3.1 Paquetage maths (spécification, corps, utilisation)
  - 3.2 Nombres rationnels (spécification, corps et utilisation)
  - 3.3 Paquetage nombres complexes
  - 3.4 Utilisation du paquetage complexe avec la clause **use type**
  - 3.5 Partie privée

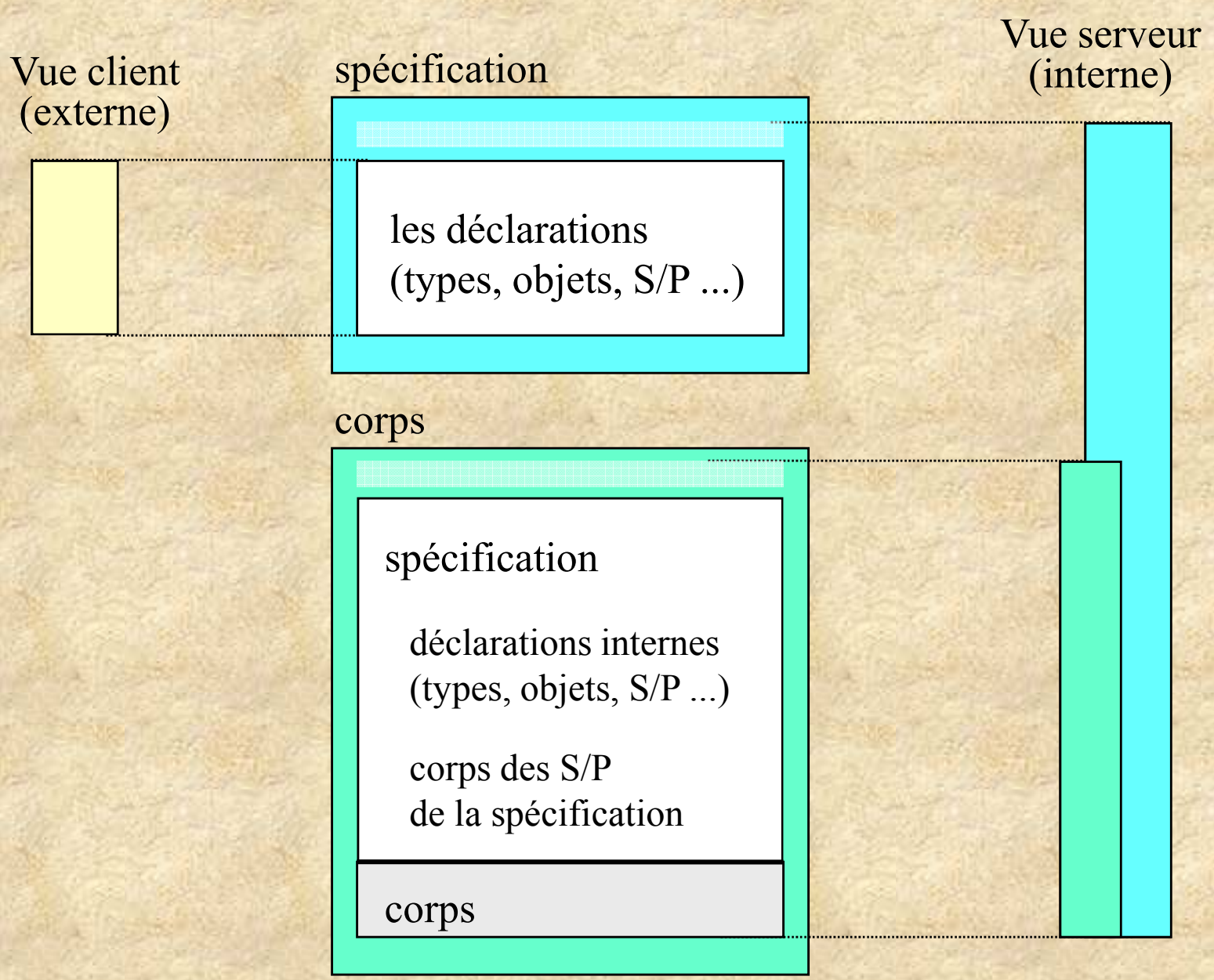
1.1 Présentation d'un paquetage

- il est composé de deux parties :
  - spécification
    - ✓ partie visible
    - ✓ interface avec le monde extérieur, le client
    - ✓ contient les déclarations (types, objets, S/P ...)
  - corps
    - ✓ partie cachée
    - ✓ réalisation du service, le serveur
    - ✓ contient le corps des S/P de la spécification plus des déclarations et des S/P internes
- La spécification et le corps peuvent se trouver :
  - dans le même fichier
  - dans des fichiers séparés

1.2 Présentation d'un paquetage (même fichier)



1.2 Présentation d'un paquetage (fichier séparé)



1.3 Exemple

```
procedure Exemple is
   package Nb_Complexe is
      type T_Complexe is ...
      function Plus (X,Y : T_Complexe) return T_Complexe;
   end Nb_Complexe;
   package body Nb_Complexe is
      Valeur_Initiale : T_Complexe ...
      function Plus ... is
   end Nb_Complexe;
   Nb_A, Nb_B, Nb_C : Nb_Complexe.T_Complexe;
   use Nb_Complexe;

begin -- Exemple
   Nb_C := Plus (Nb_A, Nb_B);
end Exemple;
```

Partie visible

Partie cachée

1.3 Exemple (suite)

```
with Nb_Complexe;
procedure Exemple is
    Nb_A, Nb_B, Nb_C : Nb_Complexe.T_Complexe;
begin -- Exemple
    Nb_C := Nb_Complexe.Plus(Nb_A,Nb_B);
end Exemple;
```

```
with Nb_Complexe; use Nb_Complexe;
procedure Exemple is
    Nb_A, Nb_B, Nb_C : T_Complexe;
begin -- Exemple
    Nb_C := Plus(Nb_A,Nb_B);
end Exemple;
```

1.4 Appel d'une action

```
with NomPackage;                                    -- en entete
…
NomPackage.NomProcedure( liste param effectifs );
```

Ou

```
with NomPackage;                                    -- en entete
use NomPackage;
…
NomProcedure(liste param effectifs);
```

1.5 Appel d'une fonction

```
with NomPackage;                                    -- en entete
…
A :=  NomPackage.NomFonction(liste param effectifs);
```

Ou

```
with NomPackage;                                    -- en entete
use NomPackage;
…
A :=  NomFonction(liste param effectifs);
```

1.6 Propriétés des paquetages

- **Modularité : Un package regroupe un ensemble de fonctions, procédures et types qui sont liés entre eux.**

  Ceci est équivalent à la notion d'unité en Pascal ou à la notion de projet en C++

- **Réutilisation du code**

- **Lisibilité du code**

- **ADA impose une unique entité (fonction, procédure ou package) par fichier.**

⇨ **Obligation d'écrire des packages**

Introduction

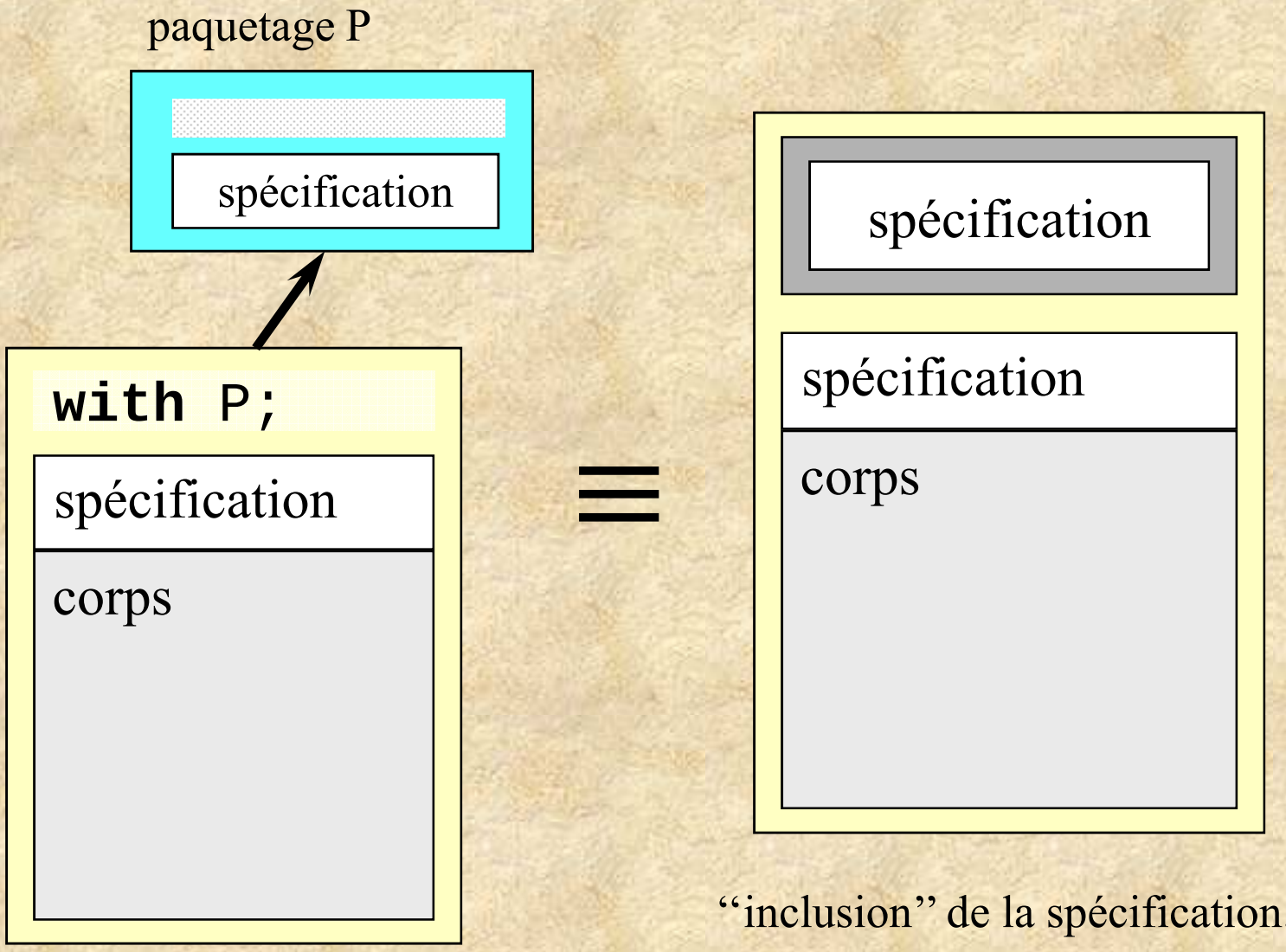
- La clause **`with`**
  - appel à une unité de bibliothèque
  - dépendance directe à une unité de bibliothèque
  - placée dans la partie déclarative
  - visible aussi dans le corps

- La clause **`use`**

- Visibilité

Clauses **`with`** et **`use`** : ‘‘inclusion’’ de la spécification.

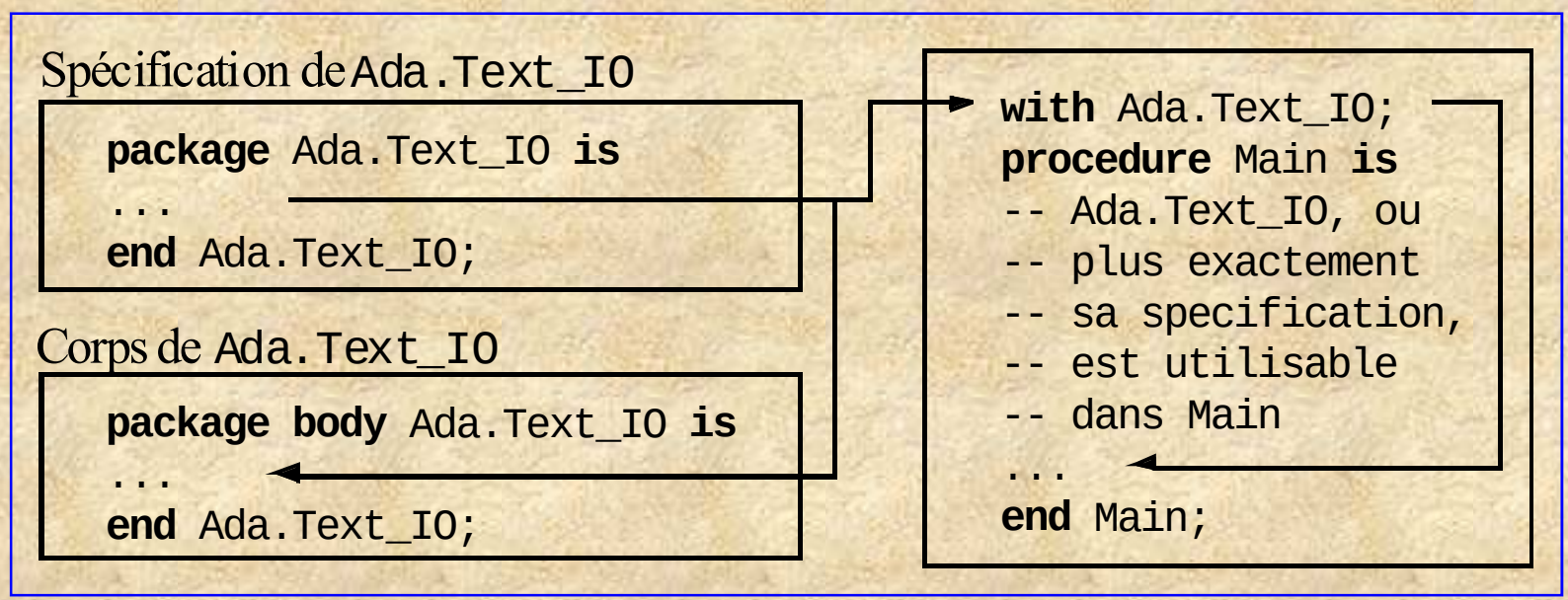
Visibilité



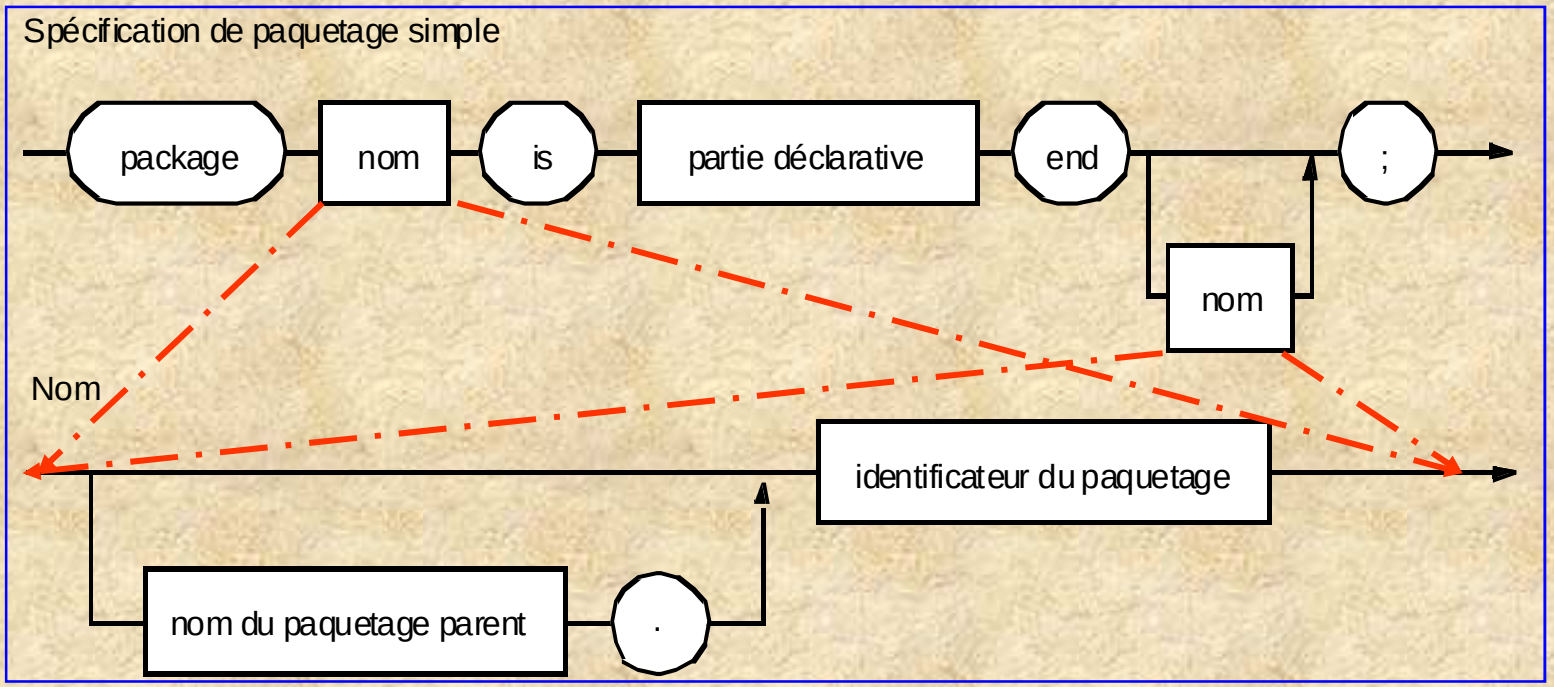
“inclusion” de la spécification.

2 Mise en en place

- La **spécification** regroupe des déclarations visibles et utilisables dans et hors du paquetage .
- Le **corps** (*body*) englobe des éléments connus uniquement à l'intérieur du paquetage.
- Les déclarations utilisables à l'extérieur du paquetage sont appelés **exportées**.
- La durée de vie des variables déclarées dans un paquetage est celle du paquetage lui-même. Elles sont parfois appelées **variables rémanentes**.
- Les éléments choisis pour faire partie d'un paquetage doivent toujours tendre à constituer un tout cohérent.



2.1 Structure d'une spécification de paquetage



- **La spécification d'un paquetage peut contenir n'importe quelle déclaration sauf des corps. Il peut être de simples déclarations de type**

  exemple : Ada.Characters.Latin_1

  Ada.IO_Exceptions

3.1 Paquetage maths (spécification)

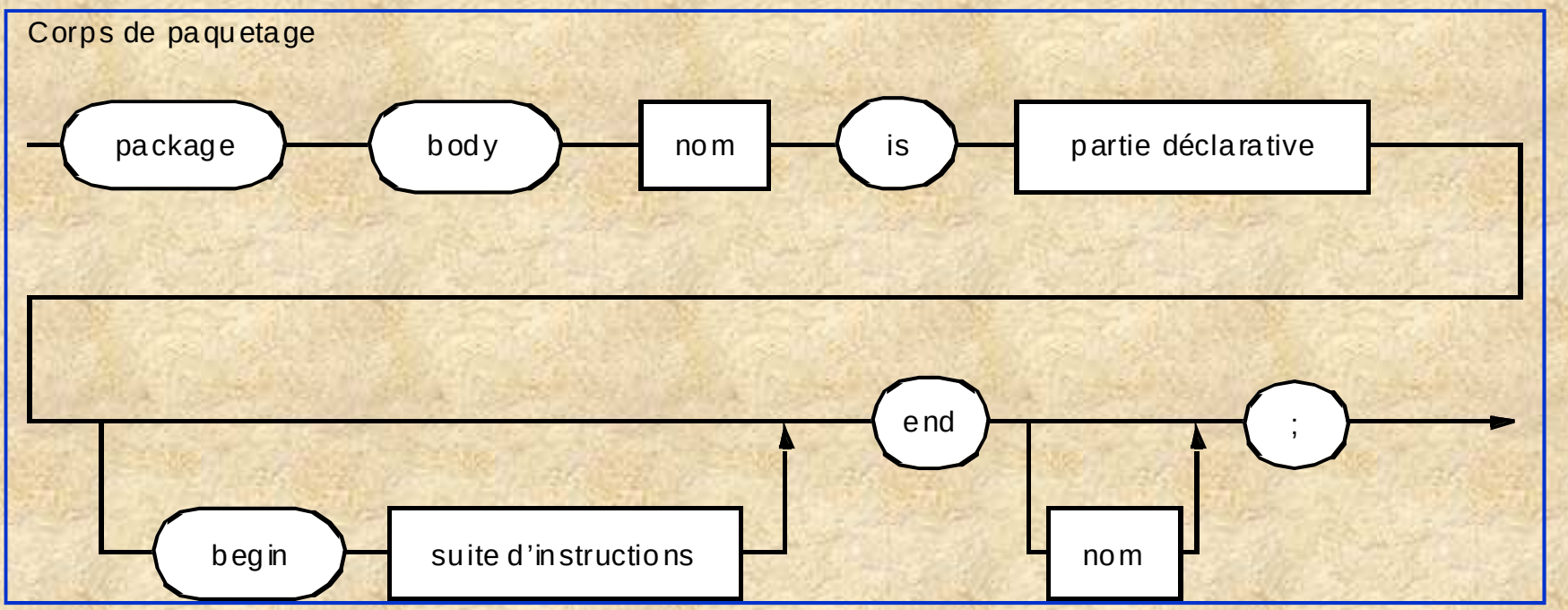
Exemple : le fichier maths.ads

```
-- maths.ads
-- Spécification du package maths : diverses fonctions mathématiques
-- Auteur :

-- Spécification du package maths

package maths is
    procedure Permuter(X,Y : in out Integer);        -- permute les valeurs de X et Y
    function Max(A,B : in Integer) return Integer;   -- renvoie le max de A et B
private
    function  Renvoie(A : in integer) return Integer; -- renvoie la valeur en entree
end maths;
```

2.2 Syntaxe du corps de paquetage



package body Nom **is**

… -- *Declarations locales et corps*

begin

… -- *Instructions: code d'initilisation.*

end Nom;

3.1 Paquetage maths (corps)

Exemple : début du fichier maths.adb

```
-- maths.adb : Corps du package maths
-- Auteur :

package body maths is
    procedure Permuter(X,Y : in out Integer) is    -- permute les valeurs de X et Y
        Tmp : Integer;                             -- sauvegarde temporaire de X
    begin
        Tmp := X;
        X := Y;
        Y := Tmp;
    end Permuter;
 -- voir la suite dans le slide suivant
```

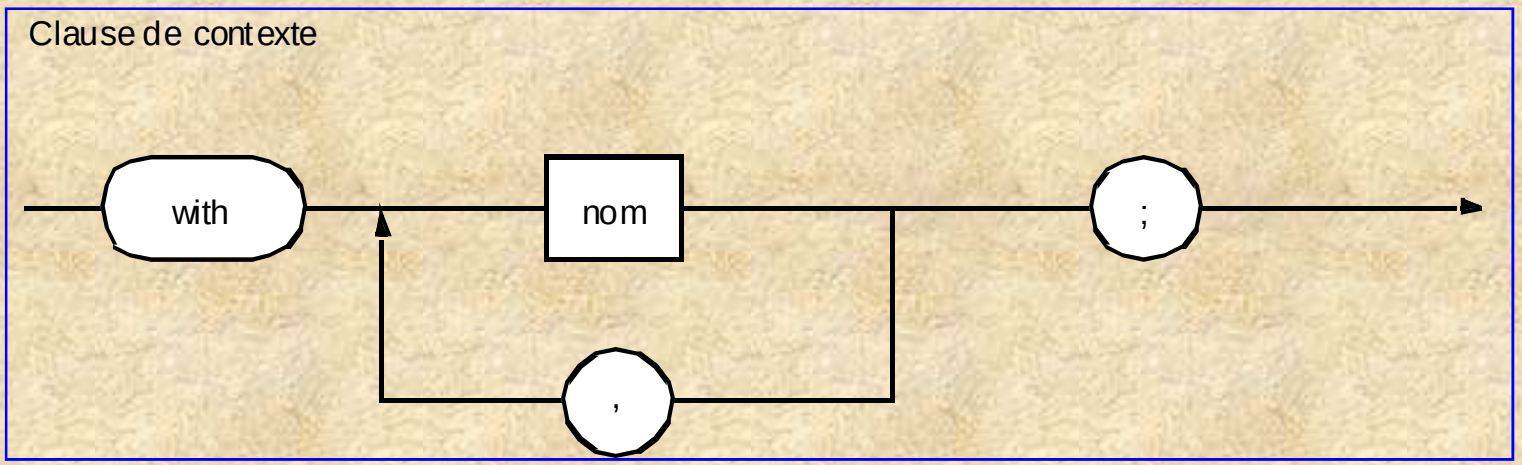
3.1 Paquetage maths (corps)

Exemple : fin du fichier maths.adb

```
function Max(A, B : in Integer) return Integer is
        -- renvoie le max de A et B
        R : Integer;                                -- résultat : le max de A et B
begin
        if A>B then R := A;
        else R := B;
        end if;
        return Renvoie(R);
end Max;
-------------------------------------------------------------------------------
function Renvoie(A: in Integer) return Integer is
begin
          return A;
end Renvoie;

end maths;                                          -- fin du package math
```

2.3 Utilisation de paquetage : Clause use



- Un identificateur déclaré dans une spécification de paquetage est rendu directement visible par une clause **use** pourvu que le même identificateur ne figure pas dans un autre paquetage mentionné avec une clause **use**, et que cet identificateur ne soit pas déjà directement visible. Si ces conditions ne sont pas remplies, le préfixage est indispensable.

- Si tous les identificateurs en présence sont des sous-programmes ou des valeurs énumérées, alors ils se surchargent et deviennent tous directement visibles. Il est donc possible que des cas d'ambiguïté surviennent, qu'il faudra éliminer par exemple en préfixant chaque identificateur ambigu par un nom de paquetage.

3.1 Paquetage maths (utilisation)

Exemple : Princ.adb – utilisation du package maths

```
-- Princ.ada : Essaie du package maths
-- Auteur :

with maths,  Ada.Text_IO, Ada.Integer_text_IO;        -- avec le package maths
use maths, Ada.Text_IO, Ada.Integer_text_IO;          -- on entre dans son domaine
procedure Principale is
   A, B : Integer;
begin
      Put("Entrer deux nombres : ");
      Get(A);
      Get(B);
      Skip_Line;
      Put("Le plus grand est : ");
      Put( Max(A,B) );
      New_Line;
end Principale;
```

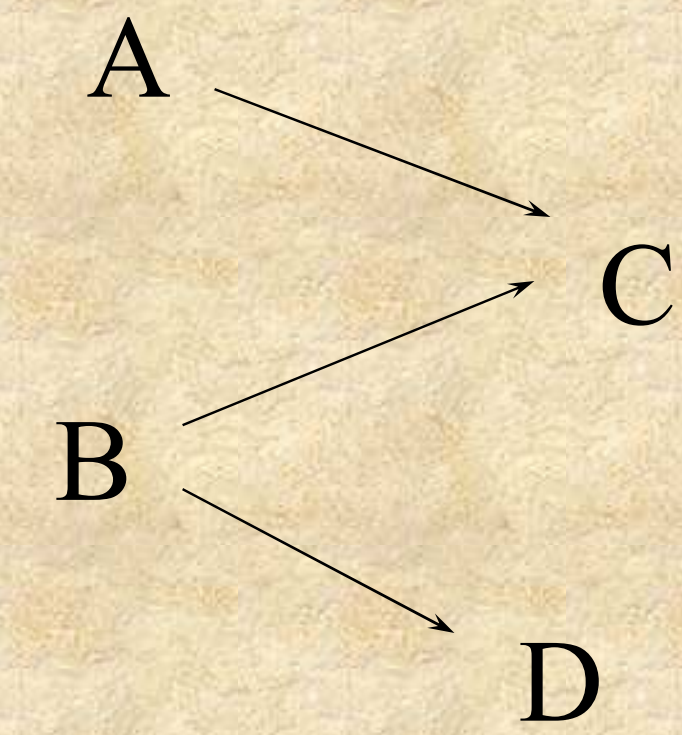
Éléments importants sur les paquetages

- **Unités de compilation et unités de bibliothèque**
  - **compilation séparée** (*separate compilation*)
  - **unités de compilation** (*compilation units*)
  - **bibliothèque** (*library*)
  - **unités de bibliothèque**

- Toute spécification (ou sous-programme) mentionnée dans une clause de contexte doit être compilée avant l'unité mentionnant cette clause.
- Une spécification de paquetage doit être compilée avant le corps.

Compilation séparée

On ne compile que ce qui est nécessaire.



	A	B	C	D
A	✓			
B		✓		
C	✓	✓	✓	
D		✓		✓

dépend de

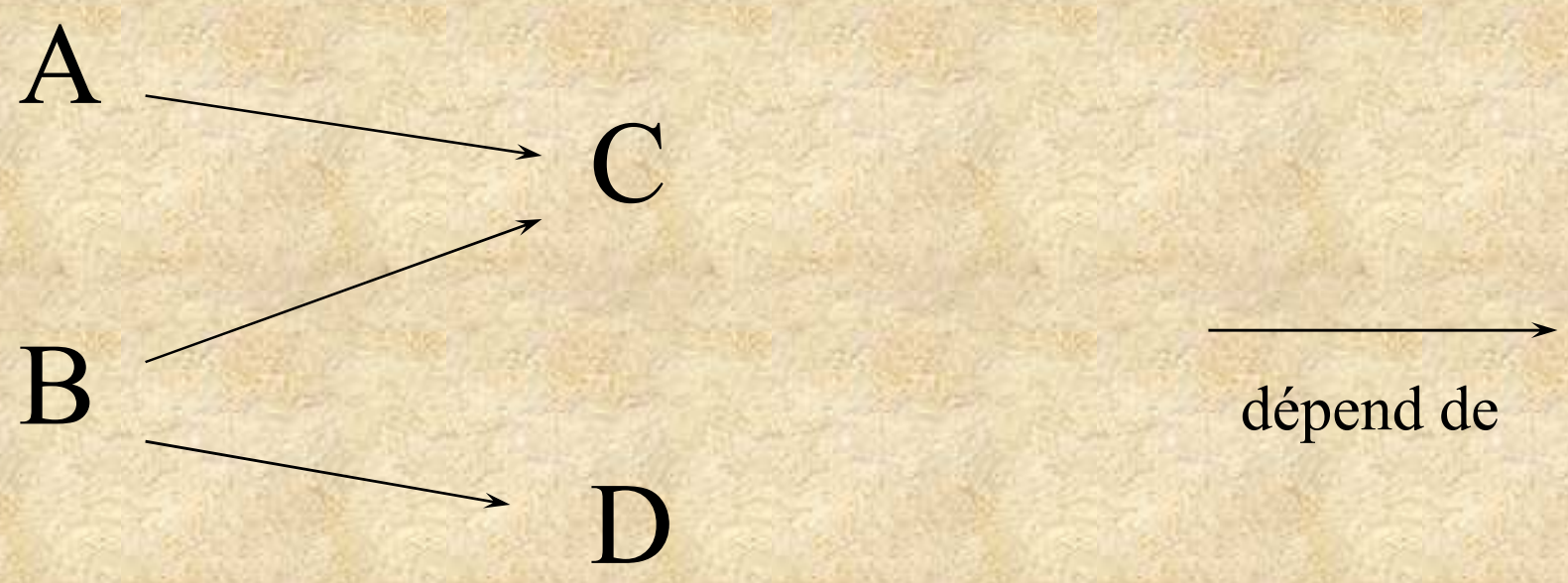
Dépendance vis-à-vis de la spécification seulement.

Bs : spécification de B

Bc : corps de B

dépend de

	A	B s	B c
A	✓		
B s	✓	✓	✓
B c			✓



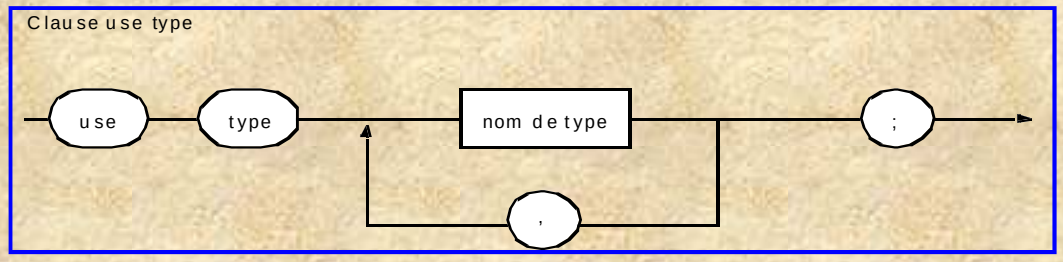
	A	B	C	D
A	✓			
B		✓		
Cs	✓	✓	✓	
Cc				
Ds		✓		✓
Dc				

Package II

3.1 *Visibilité et clauses use.*

```
with Nombres_Rationnels;
use Nombres_Rationnels;
procedure Exemple_11_5 is
        -- La declaration de la constante Zero ici cache celle  du paquetage
        Zero : constant Integer := 0;
        -- Il faut donc utiliser le nom developpe
        Nombre_1 : T_Rationnel := Nombres_Rationnels.Zero;
        -- Surcharge de la fonction-operateur de multiplication du paquetage
        function "*" (      X : Integer;
                            Y : T_Rationnel) return T_Rationnel is ... end "*";
begin -- Exemple_11_5
        -- Zero est la constante declaree ci-dessus, la premiere multiplication
        -- l'est aussi, la seconde provient du paquetage Nombres_Rationnels
        Nombre_1 := Zero * (Nombres_Rationnels.Zero * Nombre_1);
        ...
```

3.1 *Utilisation du paquetage Nombres_Rationnels avec clause use type*



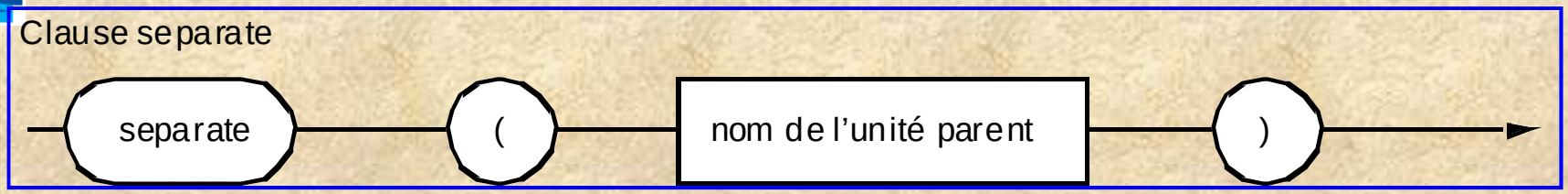

```
with Nombres_Rationnels;
use type Nombres_Rationnels.T_Rationnel;
-- La clause use type permet l'utilisation directe
-- sans prefixe, des fonctions operateurs du type T_Rationnel
procedure Exemple_11_6 is
        Une_Demi : constant Nombres_Rationnels.T_Rationnel := (1, 2);
        Nombre_1 : Nombres_Rationnels. T_Rationnel;
        -- Le prefixe est cependant toujours possible pour T_Rationnel
        Nombre_2 : Nombres_Rationnels.T_Rationnel;
begin -- Exemple_11_6
        -- Affectations
        Nombre_1 := Une_Demi;
        Nombre_2 := Nombres_Rationnels.Zero;
        -- Addition de deux nombres rationnels
        Nombre_2 := Nombre_1 + (3, 12);
        -- Division de deux nombres rationnels
        Nombre_1 := Nombre_1 / Nombre_2;
```

Sous-unités

Corps souche Puissance dans le corps du paquetage Nombres_Rationnels.

```
-- Ce paquetage permet le calcul avec les nombres rationnels
package body Nombres_Rationnels is
-------------------------------------------------------------------------
   -- Addition de deux nombres rationnels
   function "+" ( X, Y : T_Rationnel ) return T_Rationnel is ... end "+";
                ……..
   function Puissance (X : T_Rationnel; Exposant:Natural ) return T_Rationnel is separate;
end Nombres_Rationnels;
```

Syntaxe d'une clause separate



- *Sous-unité Puissance de l'unité parent Nombres_Rationnels(Nombres_Rationnels-Puissance.adb)*

Si nécessaire, une clause de contexte peut être présente!

separate (Nombres_Rationnels) ou (nom de la procédure principale) si la fonction est dans le programme principale. Pas de point-virgule ici!

```
procedure Puissance ( X:T_Rationnel; Exposant:Natural) return T_Rationnel is
begin -- Puissance
    return ( X.Numerateur ** Exposant, X.Denominateur ** Exposant );
end Puissance;
```

- Les identificateurs visibles dans une sous-unité sont ceux qui sont visibles là où le corps souche est placé, et ceux rendus visibles par les clauses de contexte mentionnées pour la sous-unité.
- Le principe pouvant se répéter à plusieurs nivaux:

  separate (Ancetre.Parent.Enfant)

Conception d'un paquetage

- il doit fournir une solution d'un problème bien défini et, si possible, de faible complexité;
- il doit constituer une brique réutilisable, voire facilement extensible;
- la spécification doit être cohérente, former un tout;
- la spécification doit être simple, et faciliter la tâche de l'utilisateur du paquetage et non celle de son concepteur;
- le corps doit réaliser complètement et précisément la spécification.

3.3 Nombres rationnels (spécification)

Spécification cohérente du paquetage Nombres_Rationnels.

```
-- Ce paquetage permet le calcul avec les nombres rationnels
package Nombres_Rationnels is
    type T_Rationnel is                          -- Le type d'un nombre rationnel
      record
            Numerateur : Integer;
            Denominateur : Positive;             -- Le signe est au numerateur
      end record;
    Zero : constant T_Rationnel := (0, 1);       -- Le nombre rationnel 0
-------------------------------------------------------------------------------
-- Construction d'un nombre rationnel
function "/" (Numerateur : Integer; Denominateur : Positive ) return T_Rationnel;
-------------------------------------------------------------------
-- Addition de deux nombres rationnels
function "+" ( X, Y : T_Rationnel ) return T_Rationnel;
-------------------------------------------------------------
```

3.3 Nombres rationnels (spécification)

```
   -- Soustraction de deux nombres rationnels
   function "-" ( X, Y : T_Rationnel ) return T_Rationnel;
   ---------------------------------------------------------------
   -- Multiplication de deux nombres rationnels
   function "*" ( X, Y : T_Rationnel ) return T_Rationnel;
   ---------------------------------------------------------------
   -- Division de deux nombres rationnels
   function "/" ( X, Y : T_Rationnel ) return T_Rationnel;
   ---------------------------------------------------------------
   -- Puissance d'un nombre rationnel
  procedure Puissance ( X:T_Rationnel; Exposant:Natural)
                                          return T_Rationnel is  separate;
  ---------------------------------------------------------------
  -- Comparaisons de deux nombres rationnels
  function "=" ( X, Y : T_Rationnel ) return Boolean;
  function "<" ( X, Y : T_Rationnel ) return Boolean;
  function "<=" ( X, Y : T_Rationnel ) return Boolean;
  function ">" ( X, Y : T_Rationnel ) return Boolean;
  function ">=" ( X, Y : T_Rationnel ) return Boolean;

end Nombres_Rationnels;
```

3.3 Nombres rationnels (corps)

```
-- Ce paquetage permet le calcul avec les nombres rationnels
package body Nombres_Rationnels is
   ------------------------------------------------------------------
   -- Construction d'un nombre rationnel
   function "/" (Numerateur : Integer; Denominateur : Positive ) return T_Rationnel is
   begin -- "/"
     return ( Numerateur, Denominateur );
   end "/";
   ------------------------------------------------------------------
   -- Pour l'addition et la soustraction afin de normaliser le resultat
   function P_P_M_C ( X, Y : Positive ) return Positive is
     Multiple_X : Positive := X;                    -- Pour les multiples
     Multiple_Y : Positive := Y;
   begin -- P_P_M_C
     while Multiple_X /= Multiple_Y loop            -- PPMC trouve?
           if Multiple_X < Multiple_Y then
                 Multiple_X := Multiple_X + X;      -- Multiple suivant
           else
                 Multiple_Y := Multiple_Y + Y;      -- Multiple suivant
           end if;
     end loop;
     return Multiple_X;                             -- C'est le PPMC
   end P_P_M_C;
   ------------------------------------------------------------------
```

3.3 Nombres rationnels (corps)

```
-- Addition de deux nombres rationnels
function "+" ( X, Y : T_Rationnel ) return T_Rationnel is
      Le_P_P_M_C : Positive := P_P_M_C (X.Denominateur, Y.Denominateur);
begin -- "+"
   return (   X.Numerateur * (Le_P_P_M_C/X.Denominateur) +
           Y.Numerateur*(Le_P_P_M_C/Y.Denominateur), Le_P_P_M_C );
end "+";
-----------------------------------------------------------------
-----------------------------------------------------------------
-- Soustraction de deux nombres rationnels
function "-" ( X, Y : T_Rationnel ) return T_Rationnel is
         Le_P_P_M_C : Positive := P_P_M_C (X.Denominateur, Y.Denominateur);
begin -- "-"
   return (   X.Numerateur * (Le_P_P_M_C/X.Denominateur) -
           Y.Numerateur*(Le_P_P_M_C/Y.Denominateur), Le_P_P_M_C);
end "-";
```

3.3 Nombres rationnels (corps)

```
-- Pour la reduction en un nombre rationnel irreductible
function P_G_C_D ( X, Y : Positive ) return Positive is
    Diviseur_X : Positive := X;          -- Pour les soustractions
    Diviseur_Y : Positive := Y;
begin -- P_G_C_D
    while Diviseur_X /= Diviseur_Y loop
  -- PGCD trouve?
       if Diviseur_X > Diviseur_Y then
            Diviseur_X := Diviseur_X – Diviseur_Y;
       else
            Diviseur_Y := Diviseur_Y – Diviseur_X;
       end if;
    end loop;
    return Diviseur_X;        -- C'est le PGCD
end P_G_C_D;
-----------------------------------------------------------------
```

3.3 Nombres rationnels (corps)

```
-- Rendre un nombre rationnel irreductible apres multiplication et  division des deux
nombres rationnels
function Irreductible ( X : T_Rationnel) return T_Rationnel is
          Le_P_G_C_D : Positive;
begin -- Irreductible
     if X.Numerateur = 0 then
           return (0, 1);
     else
           Le_P_G_C_D := P_G_C_D (abs X.Numerateur, X.Denominateur);
           return ( X.Numerateur / Le_P_G_C_D, X.Denominateur/Le_P_G_C_D);
     end if;
end Irreductible;

   -- Multiplication de deux nombres rationnels. Le resultat est un
   -- nombre rationnel irreductible
function "*" ( X, Y : T_Rationnel ) return T_Rationnel is
begin -- "*"
 return Irreductible ( X.Numerateur * Y.Numerateur, X.Denominateur * Y.Denominateur );
end "*";
```

3.3 Nombres rationnels (corps)

```
   -- Division de deux nombres rationnels. Le resultat est irreductible
function "/" ( X, Y : T_Rationnel ) return T_Rationnel is
begin -- "/"
   if Y.Numerateur > 0 then            -- Diviseur positif
      return Irreductible ( X.Numerateur * Y.Denominateur,   X.Denominateur * Y.Numerateur );
   elsif Y.Numerateur < 0 then         -- Diviseur negatif, changer de signe
      return Irreductible (– X.Numerateur * Y.Denominateur, X.Denominateur * abs Y.Numerateur);
   else -- Division par zero!
      ...              -- Lever une exception
   end if;
end "/";
-- Comparaisons entre deux nombres rationnels: egalite
function "=" ( X, Y : T_Rationnel ) return Boolean is
begin -- "="
      return     X.Numerateur * Y.Denominateur  = X.Denominateur * Y.Numerateur;
end "=";
-- Comparaisons entre deux nombres rationnels: inferieur
function "<" ( X, Y : T_Rationnel ) return Boolean is
begin -- "<"
     returnX.Numerateur * Y.Denominateur < X.Denominateur * Y.Numerateur;
end "<";
```

3.3 Nombres rationnels (corps)

```
   ----------------------------------------------------------------
   -- Comparaisons entre deux nombres rationnels: inferieur ou egal
   function "<=" ( X, Y : T_Rationnel ) return Boolean is
   begin -- "<="
      return    X.Numerateur * Y.Denominateur <= X.Denominateur * Y.Numerateur;
   end "<=";
   ----------------------------------------------------------------
   -- Comparaisons entre deux nombres rationnels: superieur
   function ">" ( X, Y : T_Rationnel ) return Boolean is
   begin -- ">"
      return    X.Numerateur * Y.Denominateur > X.Denominateur * Y.Numerateur;
   end ">";
   ----------------------------------------------------------------
   -- Comparaisons entre deux nombres rationnels: superieur ou egal
   function ">=" ( X, Y : T_Rationnel ) return Boolean is
   begin -- ">="
      return    X.Numerateur * Y.Denominateur >= X.Denominateur * Y.Numerateur;
   end ">=";
   ----------------------------------------------------------------
end Nombres_Rationnels;
```

3.3 Nombres rationnels (corps)

Remarque : 1

on compile la spécification du paquetage

on compile le corps du paquetage

on compile programme principale (procedure principale)

on compile les procédures (separate)

on fait l'édition de liens

Puis, quelque soit ce que l'on modifie, on le recompile et on fait l'édition de lien.

Remarque 2 :

- Un fichier par procedure separate
- Nom du fichier Nombres_Rationnels-Puissance
- Dans le même répertoire que le programme principale

Remarque 3 :

Tout paquetage doit être accompagné de son mode d'emploi (règle d'utilisation des opérations exportées ainsi que les sources possibles d'erreurs générées lors de l'exécution de tout sous programme du paquetage

4 Paquetage enfant

```
 -- Ce paquetage complete le calcul avec les nombres rationnels
package Nombres_Rationnels.Utilitaires is
 ------------------------------------------------------------------
   -- Valeur absolue d'un nombre rationnel
   function "abs" ( X : T_Rationnel ) return T_Rationnel;
   ------------------------------------------------------------------
   -- Multiplication d'un nombre rationnel par un nombre entier
   function "*" (       N : Integer; X : T_Rationnel ) return T_Rationnel;
   function "*" (       X : T_Rationnel; N : Integer ) return T_Rationnel;
   ------------------------------------------------------------------
   -- Division d'un nombre rationnel par un nombre entier
   function "/" (X : T_Rationnel; N : Integer ) return T_Rationnel;
   ------------------------------------------------------------------
end  Nombres_Rationnels.Utilitaires;
```

4 Paquetage enfant

```
-- Ce paquetage complete le calcul avec les nombres rationnels
package body Nombres_Rationnels.Utilitaires  is
   ------------------------------------------------------------------
   -- Valeur absolue d'un nombre rationnel
    function "abs" ( X : T_Rationnel ) return T_Rationnel is
    begin
            return ( abs X.Numerateur, X.Denominateur);
    end "abs";
  ------------------------------------------------------------------
  -- Multiplication d'un nombre rationnel par un nombre entier
  function "*" (  N : Integer; X : T_Rationnel ) return T_Rationnel is
  begin
      return ( N * X.Numerateur, X.Denominateur );
  end "*";
  ------------------------------------------------------------------
  -- Multiplication d'un nombre rationnel par un nombre entier
  function "*" (  X : T_Rationnel; N : Integer ) return T_Rationnel is
  begin
      return ( N * X.Numerateur, X.Denominateur );
  end "*";
```

4 Paquetage enfant

```
package body Nombres_Rationnels.Utilitaires  is
      -- Division d'un nombre rationnel par un nombre entier
   function "/" (   X : T_Rationnel; N : Integer ) return T_Rationnel is
   begin
        if N > 0 then                              -- Diviseur positif
                return ( X.Numerateur, N * X.Denominateur );
        elsif N < 0 then                    -- Diviseur negatif, le signe au numerateur
                return ( - X.Numerateur, abs N * X.Denominateur );
        else -- Division par zero!
                ...                          -- Lever une exception
        end if;
   end "/";
end Nombres_Rationnels.Utilitaires;
```

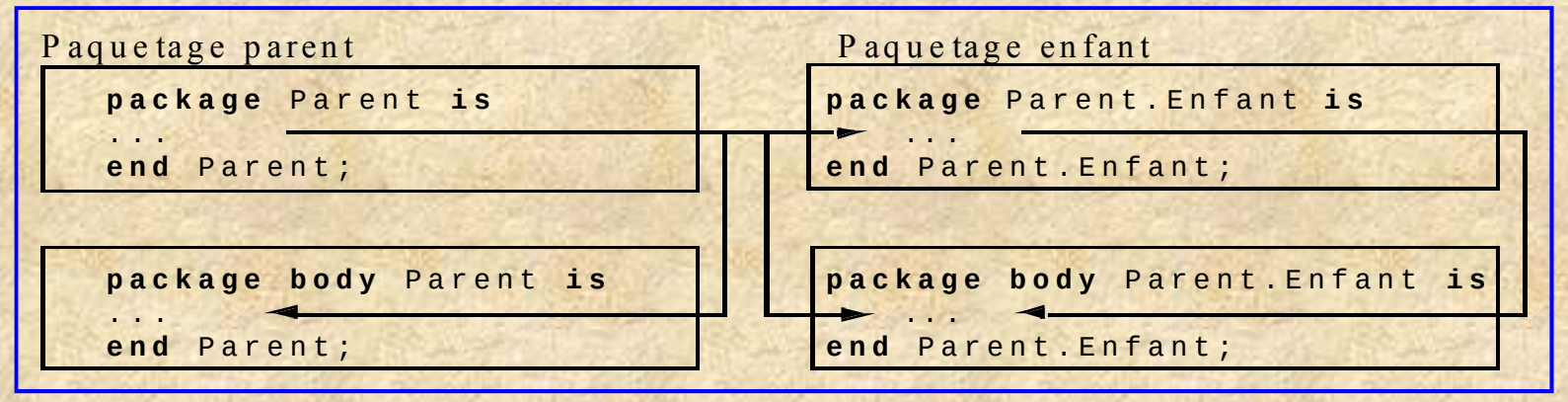
Remarque : 1

- Les éléments de la spécifications du paquetage parent sont visibles dans la spécification et le corps d'un paquetage enfant
- Les éléments du corps du paquetage parent ne sont jamais visibles pour le paquetage enfant
- Nom du fichier du paquetage enfant : Nombres_Rationnels-Utilitaires.adb

Remarque : 2

- la clause de contexte **with** d'un paquetage enfant comprend implicitement celle du paquetage parent, mais pas la clause **use**

Visibilité entre parents et enfants et Utilisation d'un paquetage enfant



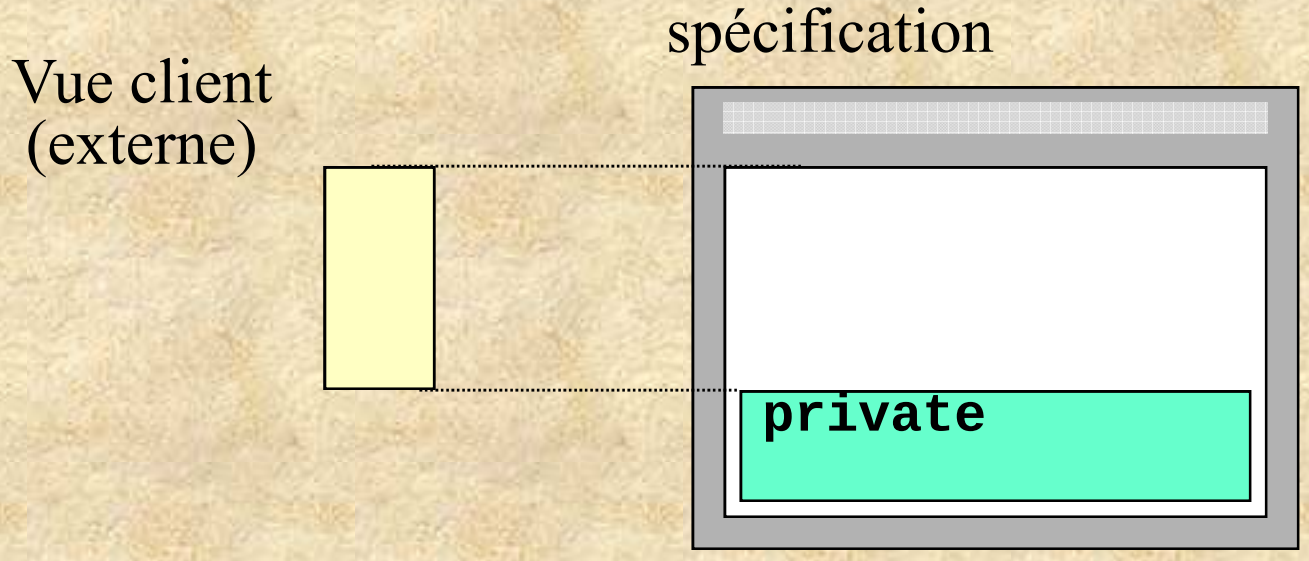

```
-- Premier exemple
with Nombres_Rationnels.Utilitaires;
-- Implique automatiquement with Nombres_Rationnels
procedure Exemple_11_13 is
        use Nombres_Rationnels.Utilitaires;
        ... -- Visibilite directe sur les elements des deux specifications
-- Deuxieme exemple
with Nombres_Rationnels.Utilitaires;
use Nombres_Rationnels;
procedure Exemple_11_13 is
        use Utilitaires;
        ... -- Visibilite directe sur les elements des deux specifications
```

Utilisation d'un paquetage enfant

```
-------------------------------------------------------------------
-- Troisieme exemple
with Nombres_Rationnels.Utilitaires;
package body Nombres_Rationnels is
-- La specification d'un enfant peut s'utiliser dans le corps du parent
            ...
-------------------------------------------------------------------
-- Quatrieme exemple
with Nombres_Rationnels.Utilitaires;
package Nombres_Rationnels.Autre_Enfant is
-- La specification d'un enfant peut s'utiliser dans la specification
-- d'un autre enfant
            ...
-------------------------------------------------------------------
-- Cinquieme exemple
with Nombres_Rationnels.Utilitaires;
package body Nombres_Rationnels.Autre_Enfant is
-- La specification d'un enfant peut s'utiliser dans le corps d'un
-- autre enfant
```

5 Partie privée

- Partie privée
  - Abstraction, encapsulation.
  - Une partie de la spécification invisible pour le client.
  - Type de données abstrait.



Type limité privé

Seules les opérations de la partie visible sont accessibles.

Conséquence : pas d'affectation.

5.1 Paquetage nombres complexes (spécification)

```
package Nb_Complexe is
    type T_Reel is new Float;
    type T_Complexe is private;
    I : constant T_Complexe;
    function ‘‘+’’ (X,Y : T_Complexe) return T_Complexe;

private  -- les détails du type sont cachés.
    type T_Complexe is
        record
            Re : T_Reel := 0.0
            Im : T_Reel := 0.0;
        end record;
    I : constant T_Complexe := (0.0,1.0);
end Nb_Complexe;
```

Paquetages prédéfinis

- Standard (cf. [ARM A.1])
- Ada (cf. [ARM A.2])
- System et ses enfants (cf. [ARM 13.7])
- Interfaces et ses enfants (cf. [ARM B.2])
- Ada.Characters.Handling (cf. [ARM A.3.2])
- Ada.Strings (cf. [ARM A.4])
- Ada.Numerics et ses enfants (cf. [ARM A.5])
- Ada.Command_Line (cf. [ARM A.15])