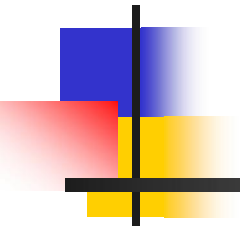
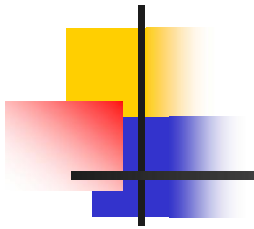


Paquetage et Héritage en ADA

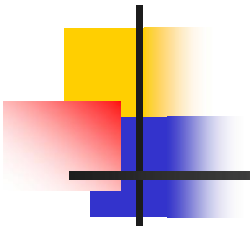
Par
Edith Roland Fotsing
et
Antoine Albanel

MCours.com



Plan

- Généralités sur les paquetages
- Description et Structure
- Usage La clause « With »
- Paquetages et types privés
- Emplacement d’un paquetage
- Notion de dérivation et héritage
- Conception Orienté Objet (C.O.O)
- Conclusion



Généralités sur les paquetages

- Les paquetage permettent d'utiliser des déclarations déjà écrites
- Les paquetages peuvent être compiler et tester séparément
- Les paquetage permettent un meilleur visibilité du programme
- Les paquetage permettent l'utilisation des types abstrait de données (TAD)
- Permet de regrouper dans une même unité de programme des déclarations de constantes,de types, d'objets et de sous-programmes

Description et structure

.ads

with (pour importer les paquetages nécessaires au déclarations)

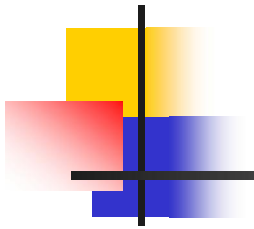
Package nom_du_paquetage ***is***
déclaration des types, variables, sous-programmes, des taches, fonctions;

end nom_du_paquetage;

.adb

With (pour importer les paquetages nécessaires aux déclarations
package body nom_du_paquetage *is*
déclarations locales au corps du paquetage;
déclaration complète des types;
réalisation complète des et des S/P et sous paquetages de la parties spécifications;
begin
end nom_du_paquetage;

- Partie Spécifications (déclarations ou profile) accessible au client (.ads)
- Partie corps ou body (implémentation) inaccessible à l'utilisateur (.adb)



Usage de la clause « With »

- La clause « with » permet de rendre visible donc utilisable le paquetage par une unité utilisatrice (programme exécutable ou un autre paquetage)

- Important: Tout objet utilisé en dehors du paquetage doit être préfixé par le nom du paquetage

Paquetage et type privés

- La déclaration d'un type privée doit se faire **absolument** dans **la parties spécifications** du paquetages

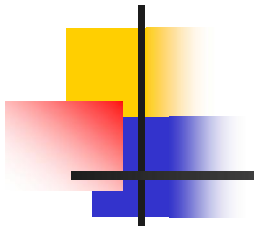
Exemple:

```
Package  P_complexe is;
Type T_nombre_complexe  is private;
-- definition des fonctions
-- definition des operations
Procedure lire (c: in out T_nombre_complexe);
Procedure ecrire (c: in out T_nombre_complexe);
```

→ Partie visible donc utilisable

```
Private
Type T_nombre_complexe is
Record
      partie_reelle: float;
     partie_immaginaire: float;
End record;
End P_cômplexe;
```

→ Partie inaccessible à l'utilisateur



Paquetage et type privés

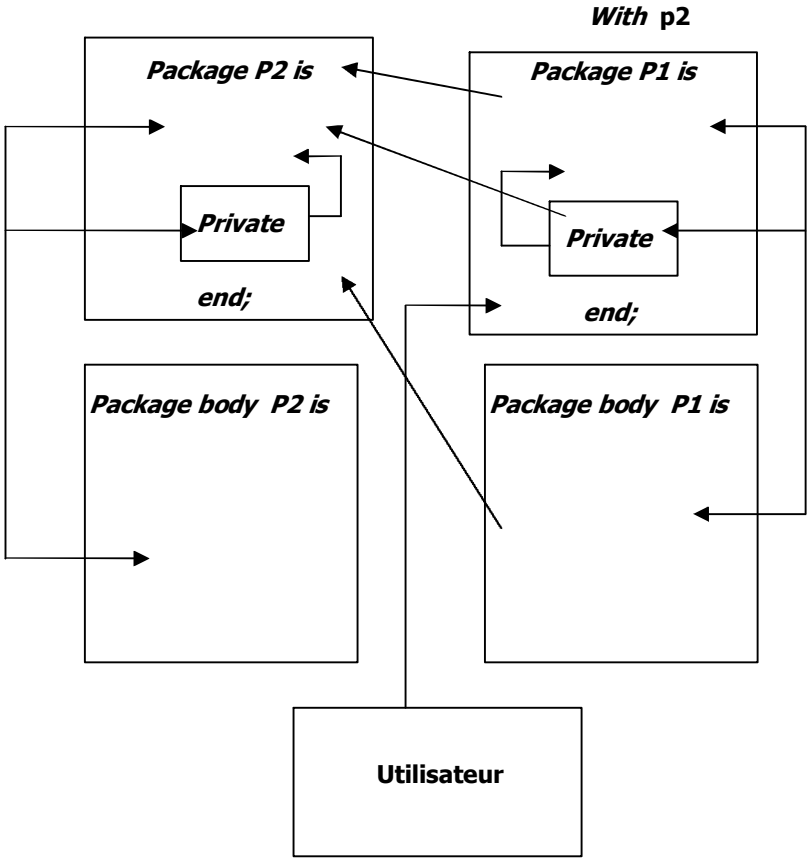
- En utilisant la clause « *With* » on peut utiliser les types à l'extérieur du paquetage

Les paquetage exporte non seulement une structure objet mais aussi un ensemble d'entités

- Les seules opérations prédéfinies sur les types privés son l'affection et la comparaison

- Les types privés limités on une utilisation encore plus restreinte et ne possèdent plus ni comparaison ni l'affection. Ils assurent le contrôle total de l'utilisation

Paquetage et type privés



- Le paquetage P1 s'appuie sur le paquetage P2 grâce à la clause ***With P2***
- Le corps de P2 a accès aux spécifications de P1 grâce à la clause ***With P2***
- Les corps des paquetages ont accès respectivement à leurs spécifications ainsi qu'à leurs parties privées.
- Les parties privées on accès a leur spécifications respectives
- L 'utilisateur n'a accès qu'aux spécifications de P2
- Aucun site n'a accès aux corps des paquetages

Emplacement d'un paquetage

- Paquetage comme unité de compilation (le plus utilisé)

```
Package nom_du_paquetage is
--Specifications;
end nom_du_paquetage;
Package body nom_du_paquetage is
begin
--implementation des specifications;
End nom_du_paquetage;
```

- Un paquetage peut être déclaré dans un bloc

```
Nom_du_bloc;
declare
package nom_du_paquetage is
--spécifications;
End nom_du_paquetage;
Package body nom_du_paquetage is
-- corps
end nom_du_paquetage;
begin --- du bloc
end nom_du bloc;
```

MCours.com

Emplacement d'un paquetage

- Un paquetage peut être déclarer dans une spécification d'un autre sous-programme

```
Procedure nom_de_la procédure is
Package nom_du_paquetage is
--spécifications;
End nom_du paquetage;
Package body nom_du_paquetage is
--implémentation;
End nom_du_paquetage;
Begin – corps de la procédure
--utilisation des fonctions et du paquetage;
End nom_de_la procédure;
```

- Un paquetage peut être déclaré dans une spécification ou une autre paquetage

```
Package body nom_du paquetage is
--declarations locales
        package locale is
-- specifications;
End local;
Package body local is
--implementations locales
End local;
End nom_du paquetage;
```

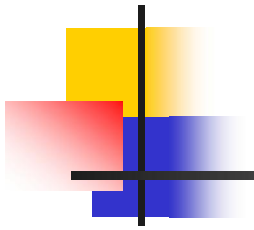
Notion de dérivation et Héritage

- Si T_père un type quelconque alors en utilisant le « is new » on peut créer un nouveau type T_nouveau dit type dérivé

  T_nouveau ***is new*** T_père

- Important: L'ensemble des valeurs d'un type dérivé est une copie de l'ensemble des valeurs du type père. Cependant les valeurs d'un type ne peuvent pas être affectées à des objets de l'autre.

```
Le_new : T_nouveau;
Le_vieux: T_père;
Le_new:=Le_vieux;  -- interdit
Le_new:= T_nouveau (le_vieux); --  possible par conversion
```

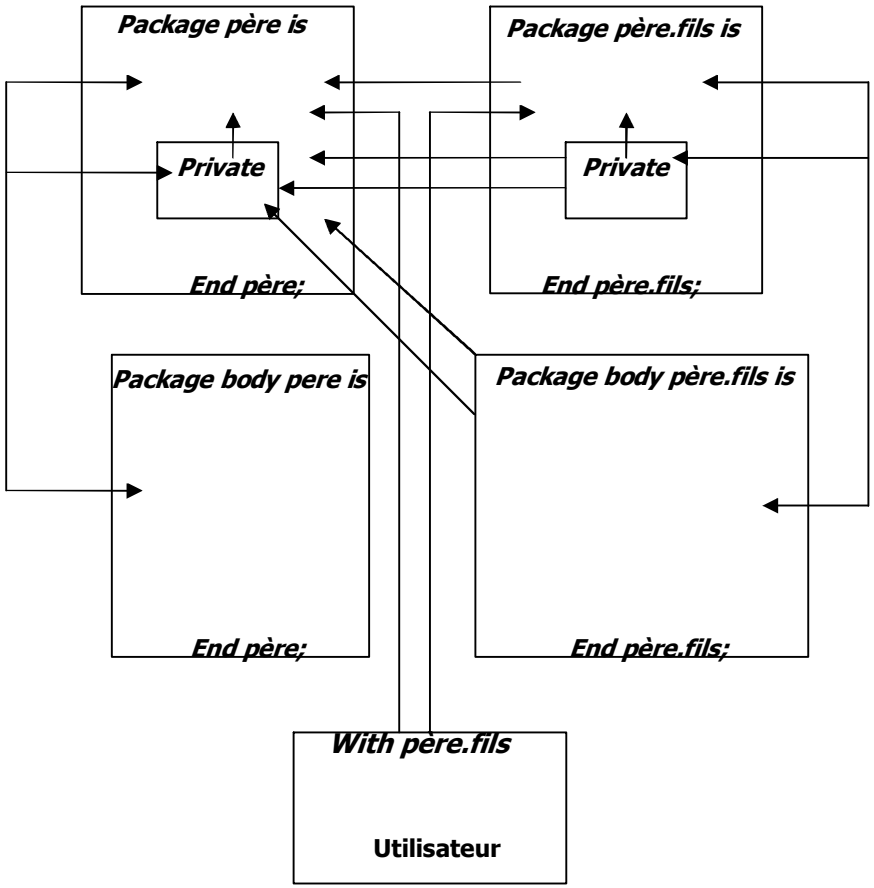


Notion de dérivation et Héritage

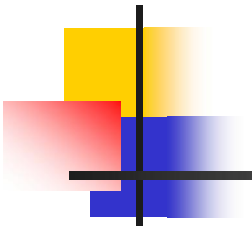
Quelques règles d'héritage pour les types dérivés

- Les opérations sur les types dérivés sont les suivantes
  - Le type dérivé possède les mêmes attributs que le type père
  - L'égalité, l'inégalité, et l' affection sont applicable sauf si le type père est limité
  - Le type dérivé héritera des sous programmes (S/P) applicables au type père
- Quand le type père est un type prédéfini alors les S/P hérités se réduisent aux S/P prédéfinis. Si le père est lui-même un type dérivé alors les S/P hérités seront de nouveau transmis
- Quand le type père est déclaré dans la partie visible d'un paquetage tout S/P déclaré dans cette partie visible sera hérité par le type dérivé dès qu'il sera déclaré.

Notion de dérivation et Héritage



- On note l'absence de la clause with
- L'utilisateur avec la clause ***with père.fils*** a accès aux spécifications du père et du fils.
- La partie privée du fils s'appuient sur les spécifications du père ainsi que sur sa partie privée du père
- Le corps du fils a accès à toutes les spécifications du père y compris la partie privée
- Les parties privées ainsi que les corps on toujours accès à leurs spécifications respectives



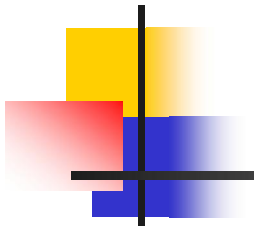
Conception Orientée Objet (C.O.O)

Définitions :

- un **objet** est un ensemble « logiciel » qui propose ses services à autrui, qui est capable de réagir à une **demande**, qui manque d'autonomie, qui a une durée de vie limitée et qui peut être **hérité**.

- Une **classe** est un ensemble d'objets (ou type pour simplifier) qui sont tous issus (ou dérivés) d'un même **ancêtre** (**racine de la classe**). En Ada cet ancêtre est un type étiqueté « *tagged* »

- En Ada **une classe se définit par un paquetage déclarant un type (étiqueté) et des sous-programmes ayant ce type en** paramètre



Conception Orientée Objet (C.O.O)

Le concept orienté objet permet:

- Une meilleur **maintenance** ainsi rien n'est plus éparpillé tout est centralisé et lié
- Une amélioration de la **sécurité** puis qu'on manipule des objets informatiques abstraction des objets réels
- Une **organisation du logiciel en couches hiérarchisées**. Il est donc possible d'établir le graphes des dépendances logiques entre les modules
- Une évolution en **parallèle** des objets

MCours.com

Conception Orientée Objet (C.O.O)

Système de réservation d'un billet d'avion avec Ada 83

- Nous partons de cet exemple inspiré du livre « Programmer en Ada 95 » de Barnes pour le concept orienté objet et l'héritage
- Pas de notion de types privés
- Codage assez complexe qui ne facilite pas la maintenance
- Pour faire un ajout on devra tout modifier et tout recompiler

```
package P_RESERVATION_83 is
type T_CATEGORIE is (TOURISME, AFFAIRE, LUXE);
type T_SITUATION is (COULOIR, FENETRE);
type T_REPAS is (VEGETARIEN, VIANDE_BLANCHE, VIANDE_ROUGE);
type T_VOITURE is (RENAULT, PEUGEOT, CITROEN);
.....
type T_RESERVATION (CAT : T_CATEGORIE) is
record
NUMERO_VOL : T_NUM_VOL;
DATE_VOYAGE : T_DATE;
NUM_SIEGE : T_NUM_SIEGE;
SIEGE : T_SITUATION;
case CAT is
when TOURISME => null;
when AFFAIRE | LUXE =>
REPAS : T_REPAS;
case CAT is
when TOURISME | AFFAIRE => null;
when LUXE => VOITURE : T_VOITURE;
end case;
end case;
end record;
.....
procedure RESERVER (R : in out T_RESERVATION);
procedure CHOIX_SIEGE (R : in out T_RESERVATION);
procedure CHOIX_REPAS (R : in out T_RESERVATION);
procedure CHOIX_VOITURE (R : in out T_RESERVATION);
.....
end P_RESERVATION_83;
```

Conception Orientée Objet (C.O.O)

Avec Ada 83….

- Le codage suit étape par étape les contours de la structure à initialiser
- Codage lourd

```
package body P_RESERVATION_83 is
procedure RESERVER (R : in out T_RESERVATION) is
begin
CHOIX_SIEGE (R);
.......
R.NUMERO_VOL := ...;
R.DATE_VOYAGE := .....;
R.NUM_SIEGE := ....;
case R.CAT is
when TOURISME => null;
when AFFAIRE | LUXE => CHOIX_REPAS (R);
case R.CAT is
when TOURISME | AFFAIRE => null;
when LUXE => CHOIX_VOITURE (R);
end case;
end case;
end RESERVER;
procedure CHOIX_SIEGE (R : in out T_RESERVATION) is
begin
....
end CHOIX_SIEGE;
procedure CHOIX_REPAS (R : in out T_RESERVATION) is
begin
....
end CHOIX_REPAS;
procedure CHOIX_VOITURE (R : in out T_RESERVATION) is
begin
....
end CHOIX_VOITURE;
........
end P_RESERVATION_83;
```

Conception Orientée Objet (C.O.O)

Avec ADA 95 ……

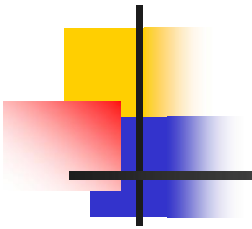
- On peut suivre la composition **incrémentale qui reprend point par point les structures héritées et n'ajoutant que** ce qui est nécessaire **au fur et à mesure et en n'utilisant que ce qui est déjà réalisé.**

```
package P_RESERVATION_95 is
type T_SITUATION is (COULOIR, FENETRE);
-- le type T_CATEGORIE n'est plus nécessaire !
type T_RESERVATION is tagged -- le discriminant a disparu!
record
NUMERO_VOL : T_NUM_VOL; -- plus de case !
DATE_VOYAGE : T_DATE;
NUM_SIEGE : T_NUM_SIEGE;
SIEGE : T_SITUATION;
end record;
procedure CHOIX_SIEGE (R : in out T_RESERVATION);
procedure RESERVER (R : in out T_RESERVATION);
type T_RESERVATION_TOURISME is new T_RESERVATION with null record;
type T_REPAS is (VEGETARIEN, VIANDE_BLANCHE, VIANDE_ROUGE);
type T_RESERVATION_AFFAIRE is new T_RESERVATION_TOURISME with
record
REPAS : T_REPAS;
end record;
procedure CHOIX_REPAS (R : in out T_RESERVATION_AFFAIRE);
procedure RESERVER (R : in out T_RESERVATION_AFFAIRE); -- surcharge
type T_VOITURE is (RENAULT, PEUGEOT, CITROEN);
type T_RESERVATION_LUXE is new T_RESERVATION_AFFAIRE with
record
VOITURE : T_VOITURE;
end record;
procedure CHOIX_VOITURE (R : in out T_RESERVATION_LUXE);
procedure RESERVER (R : in out T_RESERVATION_LUXE); -- surcharge
end P_RESERVATION_95;
```

Conception Orientée Objet (C.O.O)

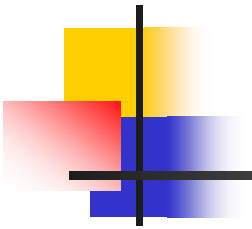
- Le type étiqueté « tagged » est extensible par dérivation (c'est la racine de la classe). Le type dérivé à nouveau ne comporte plus la mention « tagged »
- La dérivation de classe est bâtie sur le même principe classique
- T_RESERVATION_AFFAIRE déclarée après T_RESERVATION hérite des méthodes de celui-ci . Donc on pourra utiliser RESERVER et CHOIX_SIEGE avec un paramètre de type T_RESERVATION_AFFAIRE
- De même T_RESERVATION_LUXE hérite des méthodes CHOIX_REPAS et de RESERVER de T_RESERVATION_AFFAIRE et de CHOIX_SIEGE par transitivité d'avec T_RESERVATION.
- De même T_RESERVATION_LUXE hérite des méthodes CHOIX_REPAS et de RESERVER de T_RESERVATION_AFFAIRE et de CHOIX_SIEGE par transitivité d'avec T_RESERVATION.
- Les procédures RESERVER devront être **redéfinies pour les catégories « affaire » et « luxe » pour être** utilisées

```
package body P_RESERVATION_95 is
procedure RESERVER (R : in out T_RESERVATION) is
begin
CHOIX_SIEGE (R);
.....
R.NUMERO_VOL := ...;
R.DATE_VOYAGE := .....;
R.NUM_SIEGE := ....;
end RESERVER;
Ó D. Feneuille I.U.T. 2002 (cours n°10 fichier COURS10.DOC) 11/07/02
14
procedure RESERVER (R : in out T_RESERVATION_AFFAIRE) is
begin
RESERVER (T_RESERVATION(R));-- conversion sinon récursivité!
CHOIX_REPAS (R);
end RESERVER;
procedure RESERVER (R : in out T_RESERVATION_LUXE) is
begin
RESERVER (T_RESERVATION_AFFAIRE(R));
CHOIX_VOITURE (R);
end RESERVER;
procedure CHOIX_SIEGE (R : in out T_RESERVATION) is
begin
....
R.SIEGE := ......;
end CHOIX_SIEGE;
procedure CHOIX_REPAS (R : in out T_RESERVATION_AFFAIRE) is
begin
....
R.REPAS := ......;
end CHOIX_REPAS;
procedure CHOIX_VOITURE (R : in out T_RESERVATION_LUXE) is
begin
....
R.VOITURE := .....;
end CHOIX_VOITURE;
end P_RESERVATION_95;
```



Conclusion

- Les paquetages permettent une meilleur visibilités du programme par le biais de la compilation séparée.
- Permet une meilleurs gestion des types abstraits de données grâce à l'utilisation des types privés et la notion de dérivation
- L'évolution de la notion d'héritage et de la conception orientée objet depuis Ada83 jusqu'à Ada95 a été présentée.



Bibliographie

- Cours d'aix en provence
- *Programmer en Ada 95*, de John Barnes

MCours.com