CENTRALE
L Y O N

ADA et le Génie Logiciel

CENTRALE
L Y O N

Bertrand DAVID : Génie Logiciel

ADA: Unités génériques (1)

- **Problème :**
  - **Dans la réalisation de systèmes informatiques nous trouvons toujours des sous-programmes ou des modules qui ont des objectifs similaires.**

  Tri d'un tableau d'entiers :

  ```
  type TABLEAU_ENTIER is array (NATURAL range <>) of INTEGER;
  procedure TRIER (MON_TABLEAU : in out TABLEAU_ENTIER);
  ```

  Tri d'un tableau de réels : même traitement sur des réels. Nous devons réécrire la procédure TRIER

  ```
  type TABLEAU_REEL is array (NATURAL range <>) of FLOAT;
  procedure TRIER (MON_TABLEAU : in out TABLEAU_REEL);
  ```

  - **Le composant du logiciel écrit en ADA peut être paramétré : il s'agit d'une unité générique.**

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA: Unités génériques (2)

- **Soit le sous-programme qui permute deux éléments de type INTEGER .**

```
procedure PERMUTE_INTEGRE (PREMIER, SECOND: in out
   INTEGER ) is
          TEMPORAIRE : INTEGRE;
   begin
                  TEMPORAIRE  := PREMIER;
          PREMIER := SECOND;
           SECOND  := TEMPORAIRE;
  end PERMUTE_INTEGER;
```

- **Problème :** il n'est pas possible de permuter des éléments d'un autre type

BTD/GL-ADA

3

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA: Unités génériques (3)

- **Solution : on définit l'unité générique suivante**

```
generic
   type ELEMENT is private ;
   procedure PERMUTE (PREMIER, SECOND: in out ELEMENT );
```

  - **nous pouvant maintenant écrire le corps:**

```
 procedure PERMUTE (PREMIER, SECOND: in out ELEMENT ) is
   TEMPORAIRE :ELEMENT;
   begin   TEMPORAIRE  := PREMIER;
           PREMIER := SECOND;
           SECOND  := TEMPORAIRE;
   end PERMUTE;
```

Instanciation : création d'une instance du paquetage ou du sous-programme génériques.

```
procedure PERMUTE_INTEGER is new PERMUTE(INTEGER);
procedure PERMUTE_FLOAT is
          new PERMUTE (ELEMENT=>FLOAT);
procedure PERMUTE_CHARACTER is
          new  PERMUTE(ELEMENT=>CHARACTER);
```

BTD/GL-ADA

4

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA: Unités génériques (4)

- **Paramètres génériques :**

  Les différentes sortes de paramètres génériques sont possibles :

  – **Paramètres types génériques**

  – **Paramètres valeurs et objets génériques**

  – **Paramètres sous-programmes génériques**

BTD/GL-ADA

5

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA: Unités génériques (5)

1/ Paramètres types génériques :

Les formes de tous les paramètres types génériques sont :

type USAGE_GENERAL **is limited private ;**

Le paramètre réel peut être de n'importe quel type, seule opération autorisée: instanciation.

type ELEMENT **is private ;**

Le paramètre réel peut être de n'importe quel type, avec comme opérations possibles: l'instanciation, l'affectation et le test d'(in) égalité.

type LIEN **is access** UN_OBJECT**;**

Le paramètre réel peut être de n'importe quel type accès désignant le même type objet.

type ENUMERATION **is (<>);**

Le paramètre réel peut être tout type discret (type entier et énumération).

type ELEMENT_ENTIER **is range <> ;**

Le paramètre réel peut être tout type entier.

BTD/GL-ADA

6

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA: Unités génériques (6)

type ELEMENT_FIXE **is delta <>;**

Le paramètre réel peut être tout type point fixe.

type ELEMENT_FLOTTANT **is digits <>;**

Le paramètre réel peut être tout type point flottant.

type TABLEAU _CONTRAINT **is array (**INDEX**) of** ELEMENT**;**

Le paramètre réel peut être tout type tableau contraint de mêmes dimensions, types d'index, et type de composant

type TABLEAU _NON_CONTRAINT **is array (**INDEX **range <>) of** ELEMENT**;**

Le paramètre réel peut être tout type tableau non contraint de mêmes dimensions, types d'index, et type de composant

BTD/GL-ADA

7

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA: Unités génériques (7)

2/ Paramètres valeur et objet générique :

– **ADA permet également de définir des valeurs et des objets en tant que paramètres formels génériques.**

```
generic
   RANGEES : in INTEGER := 24;
   COLONNES : in INTEGER := 80;
   package TERMINAL is .....
```

– **On pout créer plusieurs instances de ce paquetage générique:**

```
package MICRO_TERMINAL is new TERMINAL (24, 40);

package TERMINAL_VAX is new TERMINAL (RANGEE => 66,
   COLONNES => 132);

package TERMINAL_DE_PROGRAMMATION is new TERMINAL ;
```

BTD/GL-ADA

8

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA: Unités génériques (8)

3/ Paramètre sous-programme générique:

– **ADA permet également de passer des sous-programmes comme paramètres à une unité générique.**

```
generic
   RANGEES : in INTEGER := 24;
   COLONNES : in INTEGER := 80;
   with procedure ENVOYER (valeur : in CHARACTER);
   with procedure RECEVOIR (valeur : out CHARACTER);
package TERMINAL is .....
```

A l'instanciation :

```
procedure MICRO_ENVOYER (valeur : in CHARACTER) is...
procedure MICRO_RECEVOIR (valeur : out CHARACTER) is...
package MICRO_TERMINAL is new
   TERMINAL (RANGEE =>24,
   COLONNES => 40,
   ENVOYER => MICRO_ENVOYER,
   RECEVOIR => MICRO_RECEVOIR);
```

BTD/GL-ADA

9

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA: Unités génériques (9)

Remarque :

le sous-programme effectif doit être compatible avec le sous-programme générique formel, c.à.d : il doit avoir des paramètres de même type, mode, ordre et contraintes que le sous-programme générique formel, et en plus une valeur retournée conforme dans le cas des fonctions.

```
generic
   type ELEM is private;
   with function "+" (A, B : ELEM) return ELEM;
function DOUBLER (R, "+" : ELEM) return ELEM is
   begin   return R+R ;   end ;
```

A l'instanciation il faut préciser le type remplaçant ELEM et la fonction remplaçant "+" associée au type effectif:

```
function AD_MAT (X, Y : MATRICE) return MATRICE.
function DEUX_FOIS is new DOUBLER (MATRICE, AD_MAT) ;
```

Les unités génériques permettent la paramétrisation des procédures, des paquetages et des fonctions, évitent la multiplication d'erreurs tout en donnant une meilleure lisibilité.

BTD/GL-ADA

10

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : entrées - sorties (1)

- **ADA fournit des paquetages prédéfinis pour trois niveaux d'entrées-sorties, qui sont:**
  - **Deux paquetages génériques pour tout type d'élément.**

    SEQUENTIAL_IO E/S sur fichiers séquentiels.
    DIRECT_IO E/S sur fichiers à accès direct.

  - **Deux paquetages de base :**

    TEXT_IO Paquetage pour les E/S de caractères.
    LOW_LEVEL_IO Paquetage défini par l'implémentation, pour les E/S primitives.

BTD/GL-ADA 11

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : entrées - sorties (2)

1/ SEQUENTIAL_IO et DIRECT_IO fournissent toutes les fonctions primitives nécessaires aux E/S d'un type d'élément donné.

les opérations définies :

CLOSE (fermer),
CREATE (créer),
DELETE (effacer),
OPEN (ouvrir) ,
READ (lire),
RESET (reprendre),
WRITE (écrire)
...

Ils comprennent également les fonctions suivantes :

END_OF_FILE (fin de fichier),
IS_OPEN (est-ouvert),
MODE,
NAME
...

BTD/GL-ADA 12

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : entrées - sorties (3)

– **DIRECT_IO définit en plus des procédures qui permettent d'accéder directement aux éléments**

SET-INDEX, SIZE, INDEX ...

– **Ces paquetages sont génériques, il faut donc les instancier avec un type particulier (via le paramètre générique** ELEMENT_TYPE**).**

```
with DIRECT_IO, SEQUENTIAL_IO;
procedure PRINCIPALE is
   package INTEGER_IO is
         new SEQUENTIAL_IO (TYPE_ELEMENT => INTEGER);
   package FLOAT_IO is
         new SEQUENTIAL_IO (TYPE_ELEMENT => FLOAT);
   begin ... end;
end PRINCIPALE;
```

Remarque: on peut également utiliser des types composites (tableaux, articles) du moment qu'ils sont contraints.

BTD/GL-ADA 13

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : entrées - sorties (4)

- **Structure et traitement de fichiers :**
  – Un fichier en ADA possède un type particulier (FILE_TYPE). Tous les éléments du fichier doivent appartenir au même type: ELEMENT_TYPE .
  – Un fichier est associé à un périphérique physique comme un disque, un terminal, une imprimante ou une boite noire (un capteur, par exemple).

Le mode d'un fichier est déterminé lors de son ouverture, ou sa création.

IN_FILE	**(fichier d'entrée)**
OUT_FILE	**(fichier de sortie)**
INOUT_FILE	**(fichier d'E/S)**

Déclaration de fichiers:

```
FICHIER_INTEGER : INTEGER_IO.FILE_TYPE;
FICHIER_FLOAT : FLOAT_IO.FILE_TYPE;
```

BTD/GL-ADA 14

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : entrées - sorties (5)

- **Utilisation:**

```
- CREATE (FILE_FICHIER_INTEGER,
                MODE => OUT_FILE,
                NAME => "MON-FICHIER-DISQUE",
                FORM => "DISQUE");

- OPEN (FILE_FICHIER_FLOAT,
                MODE => IN_FILE,
                NAME => "MA_BOITE_NOIRE",
                FORM => "CAPTEUR");

- CLOSE (FICHIER_FIXED);
- DELETE (FICHIER_INTEGER);
- SET_INDEX (FICHIER_FIXED, TO => 137);
- READ (FILE => FICHIER_INTEGER, ITEM => MON_INTGER);
- WRITE (FILE =>FICHIER_FLOAT, ITEM => 3.14);
```

BTD/GL-ADA

15

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : entrées - sorties (6)

2/ Entrées-sorties de texte :

ADA définit un paquetage séparé, pour les données composées que de caractères ASCII.

les procédures offertes par TEXT_IO :

PUT : écrire
GET : lire

COL : renvoie la valeur de colonne courante.
SET_COL : change la colonne courante.

LINE : renvoie la valeur de ligne courante.
SET_LINE : change la ligne courante.

NEW_LINE : uniquement pour les fichiers OUT_FILE.
SKIP_LINE : uniquement pour les fichiers IN_FILE.

PAGE : renvoie le numéro de page courante.
NEW_PAGE: uniquement pour les fichiers OUT_FILE, saute SPACING pages.

```
1/ NEW-LINE (SPACING =>7);   SET_COL (TO => 26);
2/ PUT ( " hello , it is me");     PUT ( "  salut");     NEW_LINE ;     PUT (" bof ");
```

BTD/GL-ADA

16

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Exceptions (1)

- **Une exception désigne un événement qui provoque la suspension de l'exécution normale du programme. Cet événement peut être:**
  - Une erreur
  - Une condition exceptionnelle qui demande un traitement spécial.
- **Utilisation des exceptions :**
  - Abandonner l'exécution de l'unité.
  - Essayer l'opération de nouveau.
  - Utiliser une méthode différente.
  - Réparer la cause de l'erreur.

BTD/GL-ADA

17

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Exceptions (2)

- **ADA fournit un mécanisme en trois temps :**
  - déclaration d'une exception,
  - signalement d'une exception,
  - traitement d'une exception
- Une **déclaration d'exception** est similaire dans sa forme à une déclaration d'objets.

```
HORS_LIMITE1, HORS_LIMITE2: exception;
ERREUR_DE_PARITE : exception;
ERREUR_DISQUE : exception;
```

Il existe des exceptions prédéfinies :
- NUMERIC_ERROR;
- PROGRAM_ERROR;
- STORAGE_ERROR;
- TASKING_ERROR;

BTD/GL-ADA

18

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Exceptions (3)

- **Signalement d'une exception** a pour effet **de** suspendre le traitement séquentiel normal et passer le contrôle à un traitement-exception approprié. L'instruction **raise** permet de lever une exception :

```
raise HOR_LIMITE1 ; ou raise;
```

- **Traitement des exceptions** s'effectue dans un bloc particulier "exception".

```
Declare NIVEAU_BAS_DU_LIQUIDE : exception ;
begin ....
exception
    when NIVEAU_BAS_DU_LIQUIDE => OUVRIR_VANNE;
    DECLENCHER_ALARME;
    when NUMERIC_ERROR =>
        FERMER_VANNE;
    when others => ENREGISTRER_ERREUR;
end ;
```

Remarque : Après avoir terminé l'exécution du traitement d'exception le contrôle ne retourne pas où l'exécution a été levée; il continue

BTD/GL-ADA 19

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Exceptions (4)

```
procedure REMPLIR_BOUILLOTTE (X : INTEGER) is
   DEBORDEMENT : exception          -- déclaration
   ....
   begin
   ....
   if X > MAXIMUM then raise DEBORDEMENT
                                    -- signalement
   end if;
   ....
exception
   when DEBORDEMENT => ESSUYER
                                    -- traitement
   when others => ....

end REMPLIR_BOUILLOTTE ;
```

BTD/GL-ADA 20

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Compilations séparées (1)

- **Intérêt : Découpage de gros programmes en plusieurs parties dans un souci de simplicité et d'efficacité :**
  - **les unités de compilation sont plus simples et plus faciles à gérer.**
  - **organisation du travail de plusieurs programmeurs travaillant sur le même projet.**
- **Unité de compilation est constituée d'une ou plusieurs unités de compilation, qui comprennent :**

Déclaration générique.
Instanciations de génériques.
Déclaration de sous programmes.
Corps de sous programmes.
Déclaration de paquetages.
Corps de paquetages.
Sous Unité.

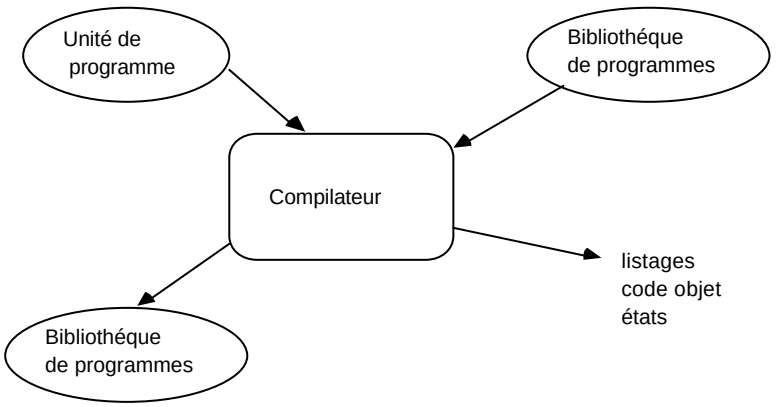
BTD/GL-ADA

21

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Compilations séparées (2)



BTD/GL-ADA

22

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Compilations séparées (3)

- **La clause WITH indique utilisation d'un package**
- **la clause SEPARATE définit une dépendance entre sous-unités de programme.**

```
procedure PRINCIPALE is
   package TRANSFORMATION is
         procedure DECRYPTER (MESSAGE : in out STRING);
         procedure ENCRYPTER (MESSAGE : in out STRING);
   end TRANSFORMATION ;
package body is separate ;
begin
--  corps de PRINCIPALE avec
      procedure DECRYPTER (MESSAGE : in out STRING) is
   separate ;
end PRINCIPALE;
```

Nous pouvons compiler les sous-unités séparées en utilisant :

```
separate ( DECRYPTER)
procedure DECRYPTER (MESSAGE : in out STRING) is  ... end ;
```

BTD/GL-ADA

23

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Compilations séparées (4)

Remarque:

- **with** est utilisée dans le développement ascendant (on réutilise les composants déjà faits).
- **separate** est utilisée dans le développement descendant (on réalisera plus tard).

- **Ordre de compilation :**
  - Une unité ne peut être compilée que si les spécifications des packages, interfaces de l'unité déclarée par la clause with, l'ont été.
  - Les corps de ces packages (package body) peuvent être compilés ou recompilés après, séparément des spécifications.

BTD/GL-ADA

24

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Compilations séparées (5)

- **Règles de recompilation :**
  - ➔ **Spécifications puis corps**
  - ➔ **Corps librement**
  - ➔ **des séparés (clause « separate ») librement**
  - ➔ **des réutilisations (clause « with ») d'abord, des modules qui réutilisent doivent être recompilés si ces modules changent**

BTD/GL-ADA

25

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA 95

- La langage ADA, normalisé en février 1983, se devait de rester stable pendant dix ans.
- La modification de la norme a été faite pour s'adapter à l'évolution de l'état de l'art en programmation a été conduite dans le projet appelé ADA-9X.
- Le résultat, officiellement adopté par l'ISO le 15 février 1995 a conduit au langage ADA-95.

BTD/GL-ADA

26