Définition d 'un type abstrait

Principe du découpage en unités

Conception orientée par les Données

Conception orientée par les Traitements

Définition d 'un type abstrait

- Motivations

  Conduire la spécification d'un type "abstrait"
  Analyser les besoins des “clients” ;
  Mesurer la portée des choix d'implantation ;
  Construire les unités de compilation associées.

- Plan

1. Une étude de cas : le type "Bourre"
2. Première spécification en français
3. Analyse des besoins en types abstraits de données
   et spécification des unités correspondantes

Spécification d 'une bourre

◆ Une première spécification

Une **bourre** est une **file** d'**individus** dans laquelle
l'ordre des éléments est variable au cours du temps.

Les individus pénètrent dans une bourre par une extrémité (la fin) et en sortent soit naturellement par l'autre extrémité (le début) lorsque vient leur tour, soit renoncent et quittent la bourre en cours de route, soit encore disparaissent corps et âmes.

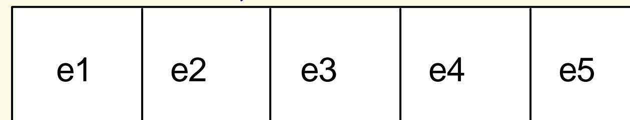
Les individus sont, entre autres, caractérisés par un **volume** et une **agressivité.** Ce sont ces paramètres, différents pour chaque individu et variables en fonction du temps, qui vont fixer leur comportement à l'intérieur de la bourre et donc la "dynamique" de la bourre.

Parmi les caractéristiques intéressantes d'une bourre on trouve le **temps moyen de séjour** et le **taux de perte**.

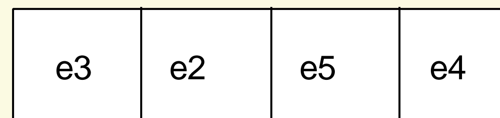
Spécification d'une bourre

◆ Un exemple de scénario

- t = 5 : arrivée de e5 ;

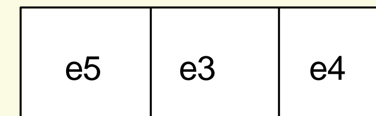


- t = 6 : sortie de e1, progression rapide de e3 et e5 ;



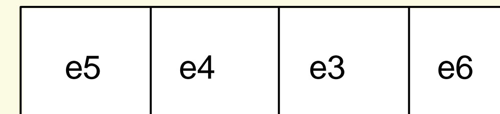
! Temps moyen de séjour

- t = 7 : e2 renonce, e5 progresse ;

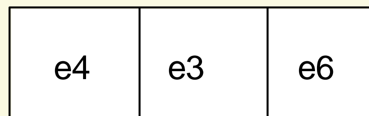


! Taux de perte

- t = 8 : arrivée de e6, e4 passe devant e3 ;



- t = 9 : sortie de e5 ;



! Temps moyen de séjour

- t = 10 ...

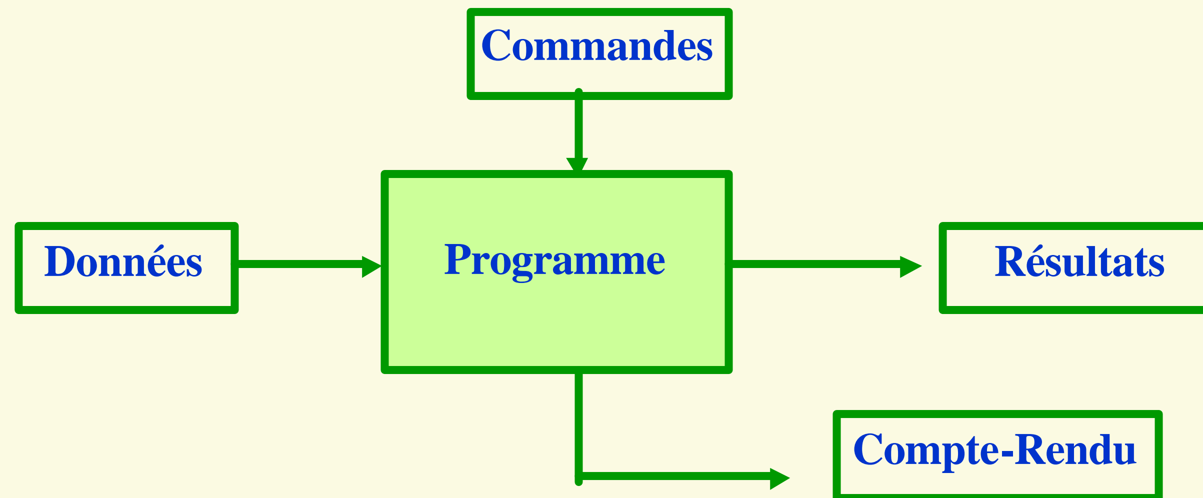
Dispositif de validation : exemple de programme

```
Program Simul ;

 {utilisation d 'unités de gestion de la bourre et des individus}
Uses u_Bourre, u_Individu ;
Var       B : t_Bourre;
          I : t_Individu;
         commande : char;
```

```
Begin
    Initialisation (B);
    Repeat
        Read (commande);
        Case commande of
            'I' :   Begin
                    creer_Individu (I) ;
                    affiche_Individu (I) ;
                    writeln (' entre à ', Heure_Locale(B)) ;
                    inserer (B,I);
                    top (B);
                    End;
            'E' :   Begin
                    extraire (B,I);
                    affiche_Individu (I) ;
                    writeln (' sort à ', Heure_Locale (B)) ;
                    top (B)
                    End;
            'T' :   top (B);
            'V' :   Informations (B);
        End;
    Until commande='X':
End.
```

Dispositif de validation : exemple de programme



Composant d ’une bourre

Le type abstrait t_Individu

```
UNIT u_Individu ;
INTERFACE
Type    Instant = Integer;
        t_Individu =    Record
                            nom : String ;
                            agr_Initiale :Real ;
                            vol_Initial : Real ;
                            f_Agressivite : Byte;
                            f_Volume : Byte;
                        End;
Procedure creer_Individu (Var I : t_Individu );
Procedure affiche_Individu (I : t_Individu) ;
Function Agressivite (I : t_Individu ; t : Instant) : Real ;
Function Volume (I : t_Individu ; t : Instant) : Real ;
```

Composant d ’une bourre

```
Implementation
Procedure creer_Individu (Var I : t_Individu ) ;
Begin
    write (‘ Nom Individu ? ’) ; readln(I.nom) ;
    write (‘ Agressivité initiale ’) ; readln(I.agr_Initiale) ;
    writeln(‘ 1 : Agressivité(t) = constante’) ;
    writeln(‘ 2 : Agressivité(t) = min(0+*t,c) ‘) ;
... Read(I.f_Agressivite) ...
End ;
Function Agressivite (I : t_Individu ; t : Instant) : Real ;
Begin
    Case I.f_agressivite Of
            1 : agressivite := I.agr_Initiale ;
            2 : agressivite := min( I.agr_Initiale ...
End ;
...
END.
```

Une première spécification de la bourre

```
UNIT u_Bourre ;
INTERFACE
Uses u_Individu ;
Type
    t_Bourre =Record
                Heure_Locale : Instant ;
                les_Individus : "liste de" Record
                                    un_Individu : t_Individu ;
                                    heure_d_Entree : Instant
                                End ;
            End;
Procedure Initialisation (Var B : t_Bourre);
Function Vide (B : t_Bourre) : Boolean;
Function Age (B : t_Bourre) : Instant;
END.
```

Une première spécification de la bourre

```
Type
    t_Bourre =Record
                Heure_Locale : Instant ;
                les_Individus : "liste de" Record
                                        un_Individu : t_Individu ;
                                        heure_d_Entree : Instant
                                End ;
            End;
```

Poursuite de l 'analyse

◆ **Définition du taux de perte**

Taux de perte = Nombre d'exclus / Nombre d'entrés

Le taux de perte est variable dans le temps.

Il faut connaître le nombre d'individus entrés depuis la création de la bourre, et le nombre de ceux qui ont disparu. A chaque nouvelle **insertion** ou nouvelle **exclusion**, on recalcule le taux de perte :

❒ Soit nb_in, le nombre des entrées ; nb_ex, le nombre d'exclus ; taux_perte, le taux de perte : taux_perte<- nb_ex / nb_in

```
Si "nouvelle entrée"
        alors "incrémenter nb_in puis calculer taux_perte"
Si "nouvelle exclusion"
        alors "incrémenter nb_ex puis calculer taux_perte"
```

Poursuite de l 'analyse

◆ **Définition du temps moyen de séjour**

Le temps moyen de séjour est calculé sur la base des individus "servis", i.e. ceux qui sortent normalement de la bourre. Le temps moyen de séjour est variable dans le temps.

A chaque **extraction**, ce temps est recalculé ; il faut donc connaître le nombre d'individus déjà sortis et la moyenne de leur temps de séjour.

❒ Soit ts, le temps de séjour du sortant ; temps_moyen, le temps moyen de séjour ; nb_out, le nombre de déjà sortis.

```
ts  <- (heure d'entrée - heure locale)
temps_moyen <-  (nb_out * temps_moyen + ts) / (nb_out + 1)
                                        ... incrémenter(nb_out)
```

Une deuxième analyse

```
UNIT u_Bourre ;
INTERFACE
Uses u_Individu ;
Type
 t_Bourre =Record
              Heure_Locale : Instant ;
              temps_moyen : Real ;
              taux_perte : Real ;
              nb_out, nb_in, nb_ex : Integer ;
              les_Individus :
                     "liste de" Record
                            un_Individu : t_Individu ;
                            heure_d_entree : Instant
                     End ;
       End;
Procedure Initialisation (Var B : t_Bourre) ;
```

Une deuxième analyse

Eléments d 'implémentation

```
Procedure Initialisation (Var B : t_Bourre) ;
Begin
    B.Heure_Locale := 0 ; B.temps_moyen := 0 ; B.taux_perte := 0 ;
    B.nb_in := 0 ; B.nb_out := 0 ; B.nb_ex := 0 ;
    B.les_Individus := ... {Liste_Vide}
End;
```

Les procédures de gestion de la bourre

Insertion

Extraction

Gestion de la dynamique d’une bourre.

Simulation : l'évolution de la bourre est provoquée par le lancement d'un "top d'horloge". Chaque top correspond à une unité de temps et incrémente donc l’heure locale de la bourre.

Procedure *Top (B : t_Bourre) ;*

L’émission d’un top est automatique ou provoquée par un événement externe e.g., l'insertion ou l'extraction d’un individu, un top manuel.

Procedure *Inserer (**Var** B : t_Bourre ; I : t_Individu) ;*

Procedure *Extraire (**Var** B : t_Bourre ; **Var** I : t_Individu) ;*

Procedure *Top (**Var** B : t_Bourre);*

Les procédures de gestion de la bourre

REMARQUES

- Ces procédures font appel à des procédures de traitement comme "Reactualiser", "Exclure un individu", "Permuter deux individus" ...
- Elles-mêmes utilisent les primitives de base de “gestion” d'une bourre.

NB. Les déclarations correspondantes peuvent être secrètes à l'unité u_Bourre (déclarations dans l’implémentation).

Procedure *Reactualiser (B : t_Bourre) ;*
Procedure *Exclure (**Var** B : t_Bourre ; **Var** I : t_Individu) ;*
Procedure *Permuter (**Var** B : t_Bourre ; **Var** i1,i2 : t_Individu) ;*
...
Function *Dans (B : t_Bourre; I : t_Individu) : Boolean;*
Function *Rang (B : t_Bourre; I : t_Individu) : Integer;*
Function *Est_Plus_Agressif (B : t_Bourre ; I1,I2 : t_Individu) : Boolean ;*
Function *Individu_En (B : t_Bourre ; rang : Integer) : t_Individu;*

Le choix des réalisations

Implantation de la "Liste de" ...

- **Liste linéaire triée par rang prévisionnel de sortie.**

  Avantages : Extraction aisée (le premier de la liste),
  Vision rapide de l'ordre actuel des individus

  Inconvénient : Réorganisation à chaque top.

- **Collection non ordonnée.**

  Avantage : Pas de réorganisation à chaque top.

  Inconvénients : Ajout d'un champ "rang" à chaque t_individu ;

  Parcours de la collection pour déterminer l'individu à extraire, ses voisins ;
  Modification du champ "rang"à chaque exclusion ou extraction.

- **Implantation Pascal.**

  e.g., Implantation d'une collection non ordonnée ?

Principes des UNITES : quelques rappels

- Définitions

  1 Unité = ensemble de procédures, fonctions et données

  1 Application = plusieurs unités

- Avantages

  plusieurs fichiers (1 unité = 1 fichier)

  ⇒ plusieurs personnes

  ⇒ compilation (donc développement séparé).

D’un type abstrait de données à l’architecture modulaire d’une application...

Syntaxe et Utilisation : quelques rappels

- Syntaxe

  Unit NomUnite;

  INTERFACE

  Déclaration des objets publics

  IMPLEMENTATION

  Déclaration des objets privés et écriture de tout le code

  END.

- Utilisation

  Instruction **Uses** NomUnite;

  { permet d'utiliser TOUT ce qui est déclaré dans l'INTERFACE }

  En principe dans la partie INTERFACE

  Peut se mettre aussi dans la partie IMPLEMENTATION (évite les références circulaires).

Delphi et les Unités

- Création d'une nouvelle fiche

  Nouvelle fiche ⇒ Nouvelle unité (automatiquement)

- Mais… l'inverse n'est pas vrai

  1 unité ne correspond pas obligatoirement à 1 fiche

  Il peut exister des unités "autonomes"

Interface et Implémentation

```
UNIT A;
INTERFACE
    Procedure SP1;
    Procedure SP2(x : Integer)          ;
IMPLEMENTATION
    Procedure SP1;
    Var I, J : Integer;
    Begin
        …I := J  + 1
    End;
    Procedure SP2(x : Integer)          ;
    Begin
        …
    End;
END.
```

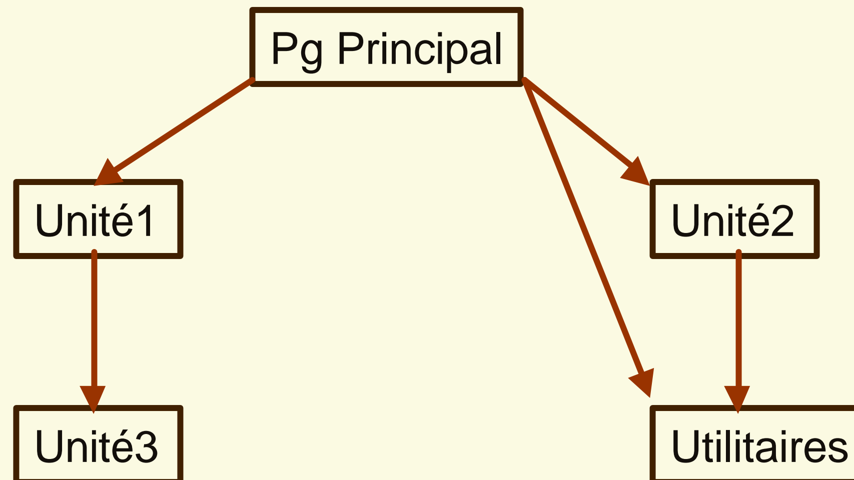
```
UNIT B;
INTERFACE
…
IMPLEMENTATION
USES A;
    Procedure SP3;
    Begin
        SP1;
        SP2( 1 )                ;
    End;
END.
```

Modif 1 : Recompilation de l'unité A, mais pas de l'unité B.

Modif 2 : Recompilation de l'unité A ET de l'unité B.

Graphe d'appel

- Chaîne d'importation ou graphe de dépendance



Conception Orientée par les Données

- Présentation

  Vers les types abstraits et / ou les objets

- Principe
  - s'intéresser aux primitives sur les données et non à leur réalisation
  - avoir des algorithmes génériques

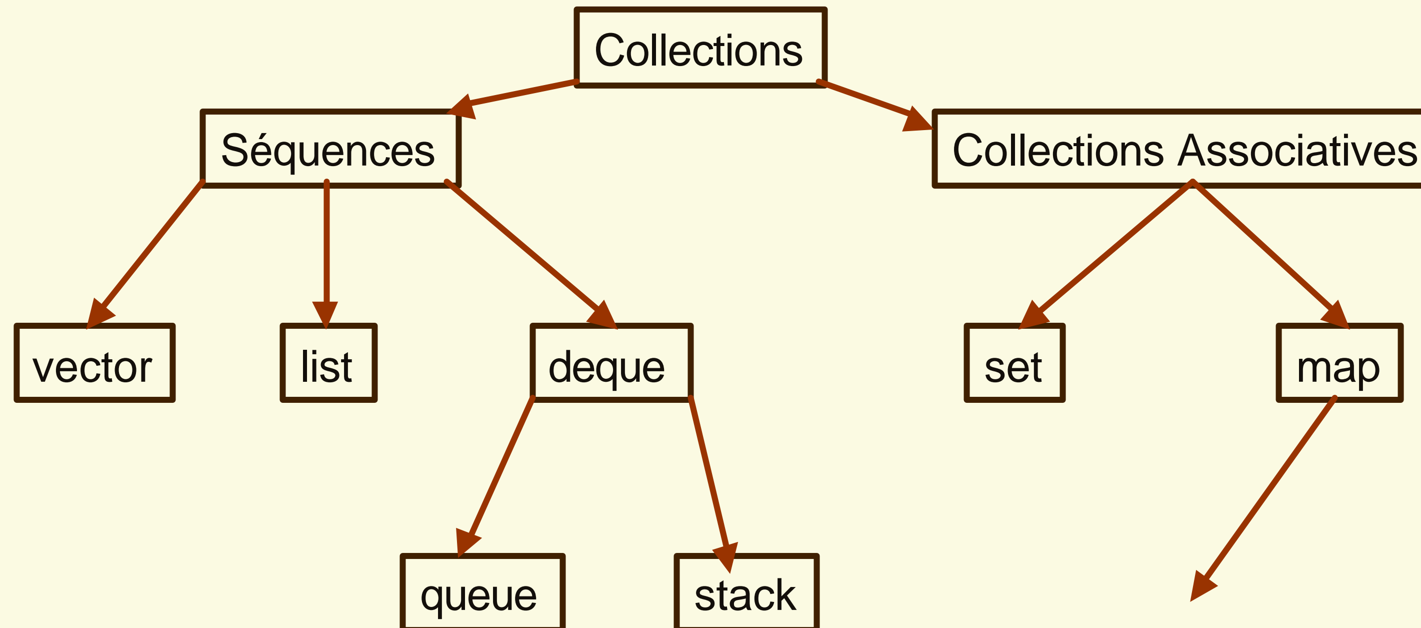
Conception Orientée par les Données

- Exemple d'algorithme générique : Balayage séquentiel d'une collection

```
Index ← Debut (collection)
BOUCLE
    SortirSi Fin (Collection)
    ElementCourant ← Prendre (Collection, Index)
    // Traitement Sur ElementCourant
    Index ← Suivant (collection)
FINBOUCLE
```

Primitives	TABLEAU	LISTE CHAINEE
Debut	Index ← 1	Index ← Collection
Fin	Index = NbElt	Index = NIL
Suivant	Index ← Index+1	Index ← Index^.suivant
Prendre	Collection[Index]	Index^

Principales Collections (Extrait de C++)



Conception Orientée par les Traitements

- Présentation

  Traitements à effectuer sur les données ⇒ coder des procédures

- Objectifs

  Liste des traitements / unités

  Graphe de dépendance entre unités

- Remarque

  Pas de solution unique

Une Méthodologie orientée application DELPHI

- **Concevoir** de façon systématique les différents **écrans** de l'appli ;
- A partir des actions possibles de l'utilisateur, **construire** les arbres **d'appel** des traitements ;
- Essayer de **factoriser** les traitements (et les écrans liés) ;
- **Construire** une unité par fenêtre (génération Delphi) ;
- **Regrouper** les autres **traitements** en unités facilitant codage et tests.

- Faire la liste des **unités**, puis le **graphe** de dépendance.