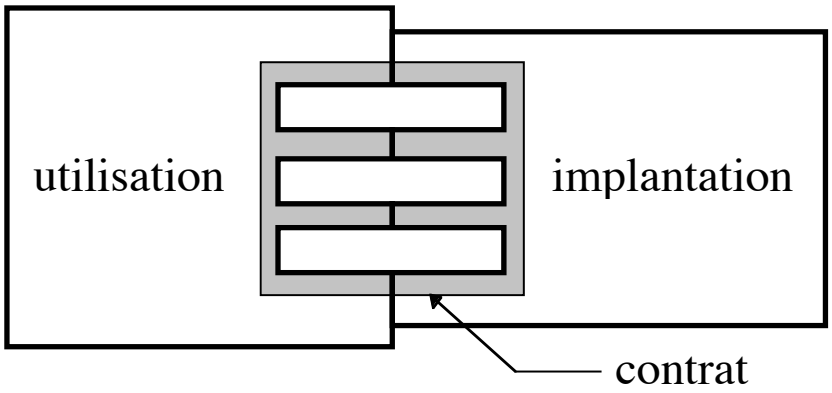
Notion de type abstrait de données

type abstrait = contrat (1)



- établissement d'un contrat entre l'implanteur et l'utilisateur
  - accord entre les deux parties
  - dire ce qu'il faut faire
  - ne pas dire comment il faut le faire
  - doit être suffisamment précis et rigoureux
  - éviter d'imposer des contraintes d'implantation

type abstrait = contrat (2)

- But: en phase de conception,
  - ne pas être encombré de détails
  - ne pas être influencé par les contraintes secondaires
  - repousser la représentation physique aux étapes ultérieures
- un type abstrait de données:
  - quelles sont les opérations
  - quelles sont les propriétés intrinsèques des opérations

type abstrait = contrat (3)

- pour l'implanteur du type,
  - vérifier que la réalisation respecte les propriétés du contrat
  - guide à la mise au point
- pour l'utilisateur du type,
  - connaître a priori les propriétés sans savoir comment il est implanté
  - vérifier la validité de son utilisation
- exemple: nombres réels avec ses 4 opérations

le type booléen

nom

type booléen

opérations

vrai	: ▯ booléen
faux	: ▯ booléen
¬_	: booléen ▯ booléen
_ ▯ _	: booléen ▯ booléen ▯ booléen
_ _	: booléen ▯ booléen ▯ booléen

signature ou syntaxe

sémantique

pour tout a, b : booléen

¬vrai = faux

¬¬a = a

vrai ▯ a = a

faux ▯ a = faux

a b = ¬(¬a ▯ ¬b)

propriétés ou sémantique

type table (1)

- qu'est-ce qu'une table?

  structure de conservation d'une quantité fixe d'informations.

  informations identifiées par un numéro appartenant à un intervalle donné.

  permet de retrouver une information, préalablement mémorisée, à partir de son numéro.

- création d'une table $i, j \xrightarrow{tab} \langle [bi \,..\, bs] \to ? \rangle$

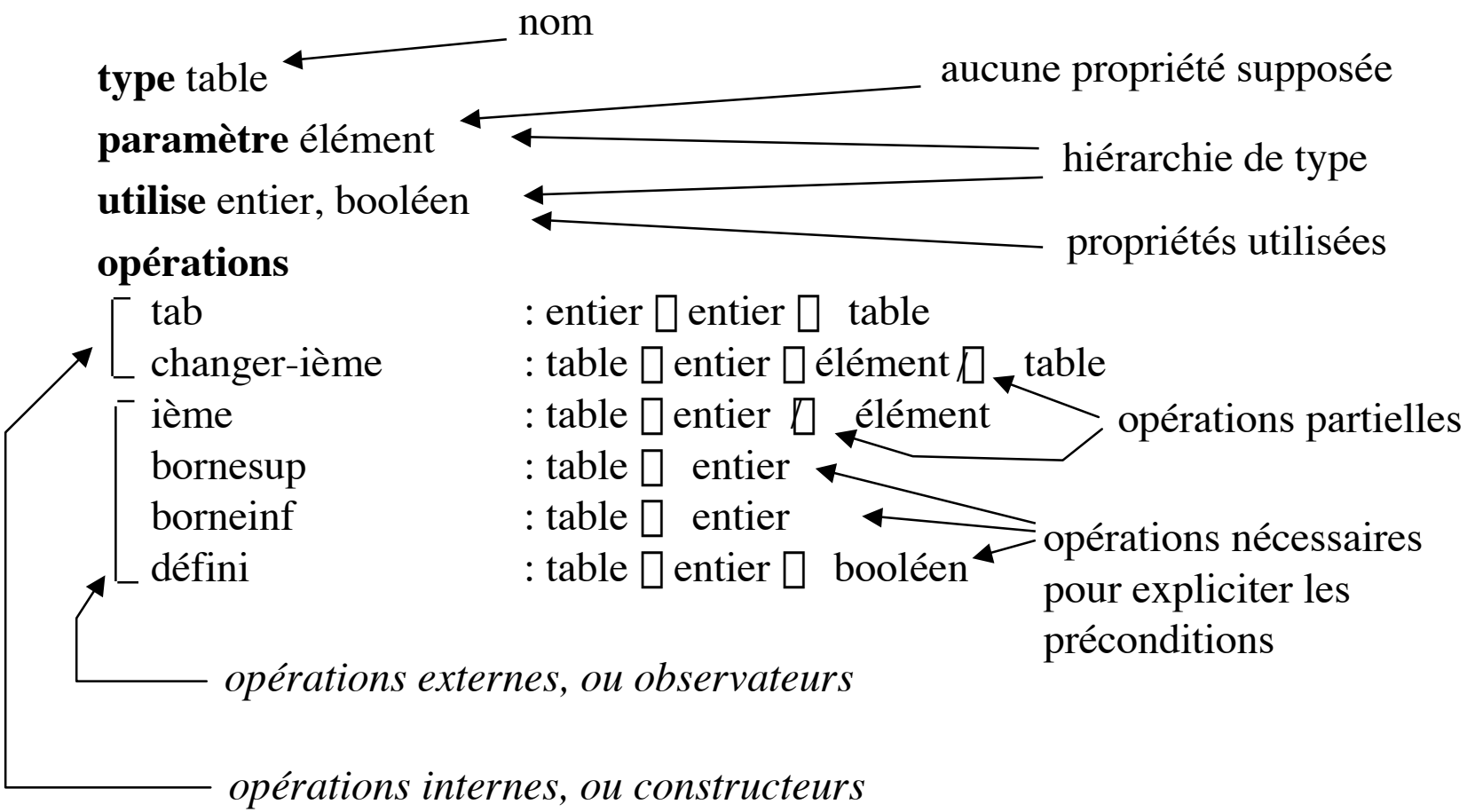
- mémorisation $\langle [bi \,..\, bs] \to ? \rangle, i, e \xrightarrow{\text{changer } i\text{ème}} \langle [bi \,..\, bs] \to ?, i \to e \rangle$

  $bi \le i \le bs$

- retrouver une information $\langle [bi \,..\, bs] \to ?, i \to e \rangle, i \xrightarrow{i\text{ème}} e$

  valeur définie en i

type table (2)


nom
type table
aucune propriété supposée
paramètre élément
hiérarchie de type
utilise entier, booléen
propriétés utilisées
opérations
tab : entier × entier → table
changer-ième : table × entier × élément ↛ table
ième : table × entier ↛ élément
bornesup : table → entier
borneinf : table → entier
défini : table × entier → booléen
opérations partielles
opérations nécessaires pour expliciter les préconditions
opérations externes, ou observateurs
opérations internes, ou constructeurs

type table (3)

préconditions

ième(t, k): défini(t, k)

changer-ième (t, k, e): borneinf(t) ≤ k ≤ bornesup(t)

sémantique

ième(changer-ième(t, i, e), j) = **si** j = i **alors** e **sinon** ième (t, j) **fsi**

borneinf(t) = **si** t = tab(i, j) **alors** i
sinon soit t = changer-ième(t', k, e); borneinf(t') **fsi**

bornesup(t) = **si** t = tab(i, j) **alors** j
sinon soit t = changer-ième(t', k, e); bornesup(t') **fsi**

défini(t, k) = **si** t = tab(i, j) **alors** faux
sinon soit t = changer-ième(t', i, e);
si k = i **alors** vrai **sinon** défini(t', k) **fsi fsi**

difficultés du choix des axiomes

- dire la vérité, rien que la vérité (c'est le contrat)
  - pas de contradiction (*consistance*)
  - suffisamment d'axiomes (*complétude*)
  - forme d'écriture tente de satisfaire ces critères
- pour les observateurs: déduire une valeur et une seule pour tous les arguments vérifiant les préconditions
- préciser le moins possible les opérations internes, pour laisser la liberté à l'implanteur
- opérations nécessaires pour expliciter les propriétés (défini) => implantation optionnelle à préciser

implantation

- transformer la notation fonctionnelle pour réintroduire les variables

  opérations de construction: T est résultat sans être argument

  opérations d'accès: T est argument sans être résultat

  opérations de transformation: T est argument et résultat

  => transformer par "effet de bord" et non par recopie

- représentation

  regroupement par contiguïté (tableaux ou articles)

  regroupement par chaînage (pointeurs, allocation/restitution dynamique)

- quelle connaissance de cette représentation donne-t-on?
- choix des opérations effectives réalisées

outils de Java (1)

- Il n'y a que des objets et pas de valeurs (sauf types élémentaires)
- arborescence de classe, avec la classe `object` à la racine
- Conséquences pour l'implanteur:

  variable = référence à un objet = pointeur
  - déclaration d'une variable => pointeur null

  affectation = affectation de pointeurs

  égalité = égalité de pointeurs

  nécessité de définir éventuellement les opérations:
  - `clone`: copie de la valeur pointée
  - `equals`: test d'égalité des valeurs pointées

outils de Java (2)

- Connaissance du type
- classe d'un paquetage

  publique (**`public`**): accessible à tous

  par défaut: accessibles aux classes du même paquetage
- composant d'une classe (objet ou méthode)

  publique (**`public`**): accessible à tous

  protégé (**`protected`**): accessible aux classes du paquetage et aux classes dérivées sous certaines conditions

  par défaut: accessibles aux classes du même paquetage

  privée (**`private`**): accessible à l'intérieur de la classe

```
paquetage
 |classes
 | |composants
 | |méthodes
 |interfaces
 | |constantes
 | |méth. abst.
```

outils de Java (3)

- Construction des objets
  - 1 constructeur par défaut si aucun fourni
  - l'implanteur définit un ou plusieurs constructeurs (selon signature)
  - peuvent être public, protégé, privé ou à visibilité par défaut
- Destruction des objets
  - automatique par ramasse-miettes
  - redéfinition éventuelle de la méthode `finalize`
- Structures "fourre tout"
  - définir une structure sur la classe `object`
  - toute classe dérivant de la classe `object` => pas de contrainte

outils de Java (4)

- Structures paramétrées à la création par un (ou des) "specimen"
  - attacher à la structure la classe du ou des specimens
  - contrôler en interne qu'un objet est de la même classe que le specimen
- Spécialisation partielle au moyen d'interfaces
  - restreindre les classes possibles du specimen,
  - fonctionnalités devant être implantées par la classe du specimen
  - contrôler qu'un objet a les fonctionnalités attendues
- Exemple (fonctionnalité `clone` attendue)

```
public interface Clonable1 {
 Object Clone() throws CloneNotSupportedException;
}
```

outils de Java (5)

- Prise en compte des erreurs au moyen des exceptions
  - classe spécifique dans l'arborescence `Throwable`
  - levée d'une exception `Throw`
  - récupération des exceptions (`try` - `catch`)
  - la signature d'une méthode doit lister les exceptions que la méthode peut lever
  - spécialisation par des sous classes

outils de Java (6)


```
package bibSD java.Les_Tables;
import bibSDjava.Erreurs_Types_Abstraits.*;
class Place { int Numero; Clonable1 Contenu; Place Suivant; }
public class Table {
  int bi, bs;                    // intervalle
  Place Tete;                    // premier couple
  Class Type_Element;            // le type des éléments de la table
  public Table (int i, int j, Clonable1 Specimen)
          throws CloneNotSupportedException {}
  public int Bi () { return bi; }
  public int Bs () { return bs; }
  public Object Ieme (int En) throws Erreur_Specification {}
  public void Changer_Ieme (int En, Clonable1 Avec)
          throws Erreur_Specification, Erreur_Typage {}
}
```

outils de C++ (1)

- connaissance du type

  toute la connaissance

  struct Table { **const** int Bi, Bs; Pt_Place Tete; }

  connaissance limitée

  class Table { **public**:Table (**const** int i, **const** int j); **const** int Bi, Bs;
  private: Pt_Place Tete; void operator= (Table &V) {}};

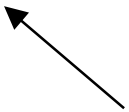
  connaissance partielle aux classes dérivées

  protected: ...

- construction des objets

  constructeurs par défaut

  l'implanteur définit un ou plusieurs constructeurs



exclusion explicite
de l'affectation

outils de C++ (2)

- destruction des objets

  destructeur par défaut

  destructeur explicite défini par l'implanteur

- généricité

  paramétrisation d'une classe par des types => "patrons" (template)

  => on peut faire des tables d'éléments sans savoir ce que sont ces éléments

  paramétrisation par des opérations

  => faire un tri sans savoir ce qu'est effectivement l'opération "<"

- prise en compte des erreurs au moyen des exceptions

  une exception retourne un objet quelconque

outils de C++ (3)


```
template <class Element> class Table {
  public:
    Table (const int i, const int j): Bi(i), Bs(j) {Tete = 0;};  ~Table ();
    const int Bi, Bs;
    Element Ieme (const int En) const throw (Erreur_Specification);
    void Changer_Ieme (const int En, const Element &Avec);
  private:
    struct Place {int  Numero; Element  Contenu; Place*  Suivant;};
    typedef Place*  Pt_Place;
    Pt_Place Tete;                    // ci-dessous, operations interdites
    void operator= (const Table<Element> &Valeur) {}
    Table (const Table<Element> &Copie) {}
  };
```

outils de Ada 95 (1)

- connaissance du type

  complète

  type Table (Bi, Bs : Integer) **is record** Tete : Pt_Place; **end record**;

  partielle (affectation et égalité)

  type Table(Bi, Bs : Integer) **is private**;

  limitée (uniquement les opérations fournies)

  type Table(Bi, Bs : Integer) **is limited private**;

  fournir la plus grande connaissance, tout en garantissant le bon fonctionnemment, quoiqu'il arrive

- hérarchie des paquetages

  les utilisateurs ont accès à la partie visible de la spécification

  les enfants ont accès à la partie privée de la spécification de leurs parents

outils de Ada 95 (2)

- construction des objets

  initialisation des objets à la création, implicitement ou explicitement

- destruction des objets

  pas de pointeur => automatique

  avec pointeurs => du ressort de l'implanteur (Finalisation)

- généricité

  paramétrisation des modules par des types

  => on peut faire des tables d'éléments sans savoir ce que sont ces éléments

  paramétrisation par des opérations

  => faire un tri sans savoir ce qu'est effectivement l'opération "<"

- prise en compte des erreurs au moyen des exceptions

outils de Ada 95 (3)


```
with Ada.Finalization;
generic
  type Element is private; -- type des éléments de la table
package Tables is
  type Table (Bi, Bs : Integer) is limited private;
  function Ieme (De : Table; En : Integer) return Element;
  procedure Changer_Ieme(De : in out Table; En : Integer; Avec : Element);
private
  type Place; type Pt_Place is access Place;
  type Table (Bi, Bs : Integer) is  new Ada.Finalization.Limited_Controlled with
    record Tete : Pt_Place := null; end record;
  procedure Finalize (T : in out Table);
end Tables;
```