Exemple Analyse 1 : Opérations sur matières et notes

V. Aponte

Cnam

7 décembre 2012

Sous-programmes : outil de découpage/structuration

Principe de conception/programmation : diviser pour régner

Objectif : Concevoir un programme comme un assemblage de solutions à des sous-problèmes :

- Chaque sous-problème correspond à une partie (non triviale) du problème global ;
- Concevoir un sous-programme par sous-problème ;
- Obtenir liste sous-programmes $\Rightarrow$ noms + paramètres+ types+ description objectif (sans dire comment) ;

Guide : pouvons nous écrire une solution simple et courte par asseblage d'appels aux sous-programmes ?

Analyse descendante

Processus de découpage d'un problème en sous-problèmes :

- en identifiant les sous-programmes à écrire pour obtenir la solution ;
- chaque sous-programme utilise d'autres (éventuellement)
- chaque sous-programme reste très simple et court

Question à poser par sous-problème identifié :

- Que doit faire le sous-programme correspondant ?
- Son nom, ses paramètres, son résultat ?
- Sa signature Java $\Rightarrow$ nom, type paramètres + résultat.
- Découpable en sous-sous-problèmes ? Si oui, continuer analyse descendante sur les sous-sous-problèmes.

Analyse descendante : données

Répresentation des données :

- quelles sont les données d'entrée et/ou sortie ?
- comment les organiser ?
- quelles opérations aurons nous à leur appliquer ?
- coment les répresenter pour faciliter les opérations ?
- quelles hypothèses faisons nous sur les données ?

Lien entre sous-programmes et répresentation des données :

- chaque opération doit se traduire par une (des) méthode,
- doivent être utiles à notre problème,
- doivent maintenir les hypothèses sur les données.

Un exemple

Problème : gérer les notes d'un ensemble de matières, avec opérations de calcul de moyenne, recherche d'une note, modification d'une note, affichage du bulletin.

- Initialisation :
  1. lire une liste de noms de matières,
  2. lire une note par matière,
- Boucle avec menu d'opérations :
  1. afficher le bulletin des notes
  2. chercher la note d'une matière
  3. modifier une note
  4. finir

Interface texte : gestion de notes

```
Nombre de matieres? 4
Une matiere? Maths
Une matiere? SVT
Une matiere? Info
Une matiere? Anglais
Note de Maths ? 10
Note de SVT ?  11
Note de Info ? 12
Note de Anglais ?  13

Bonjour. Voici les operations disponibles:
  1. Afficher le bulletin
  2. Chercher une note
  3. Mofidier une note
  4. Finir
Votre choix --->
```

Gestion des notes : les données

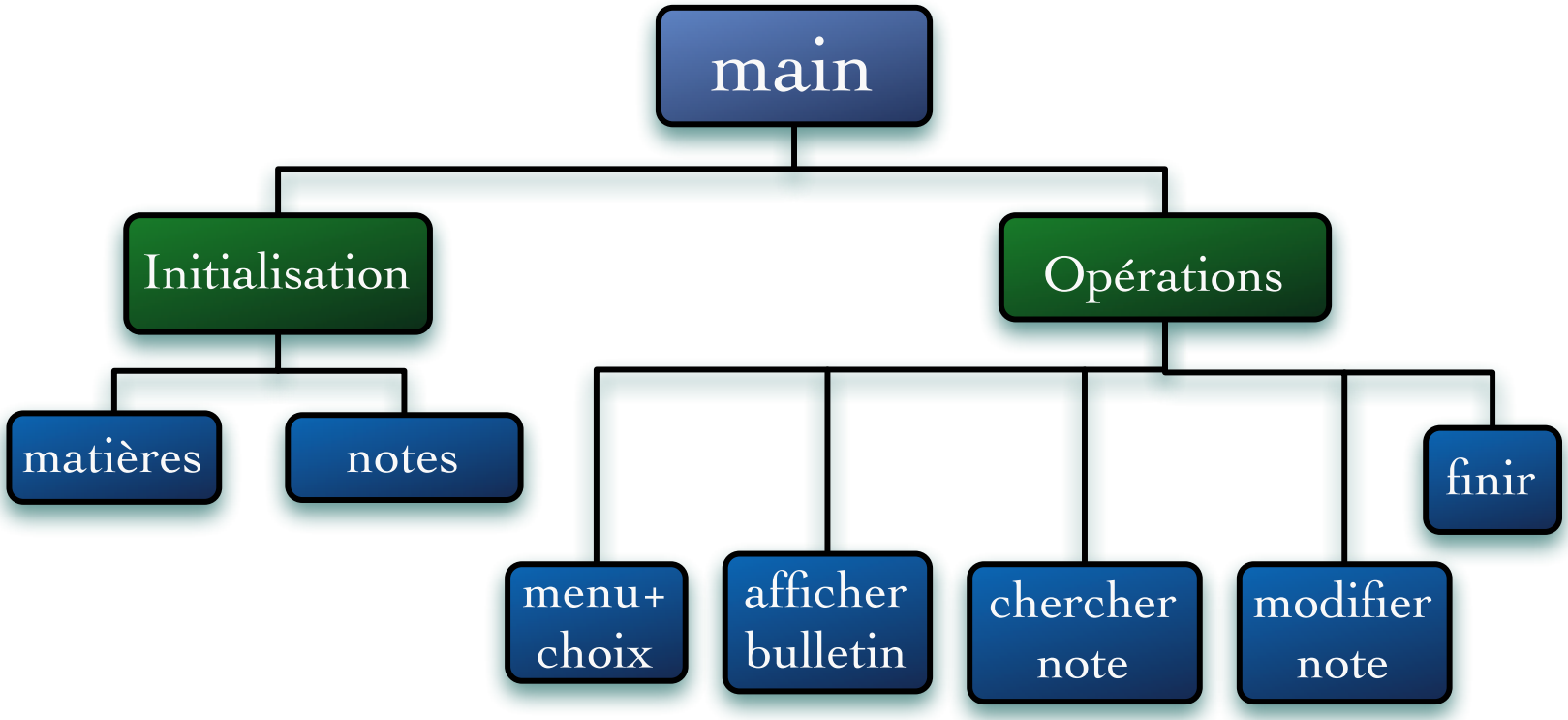
Répresentation des données : 2 tableaux

- tableau de chaînes avec noms des matières,
- tableau de notes (doubles)
- Hypothèses : tableaux de même taille, au moins 1 composante, matière et sa note aux mêmes indices.

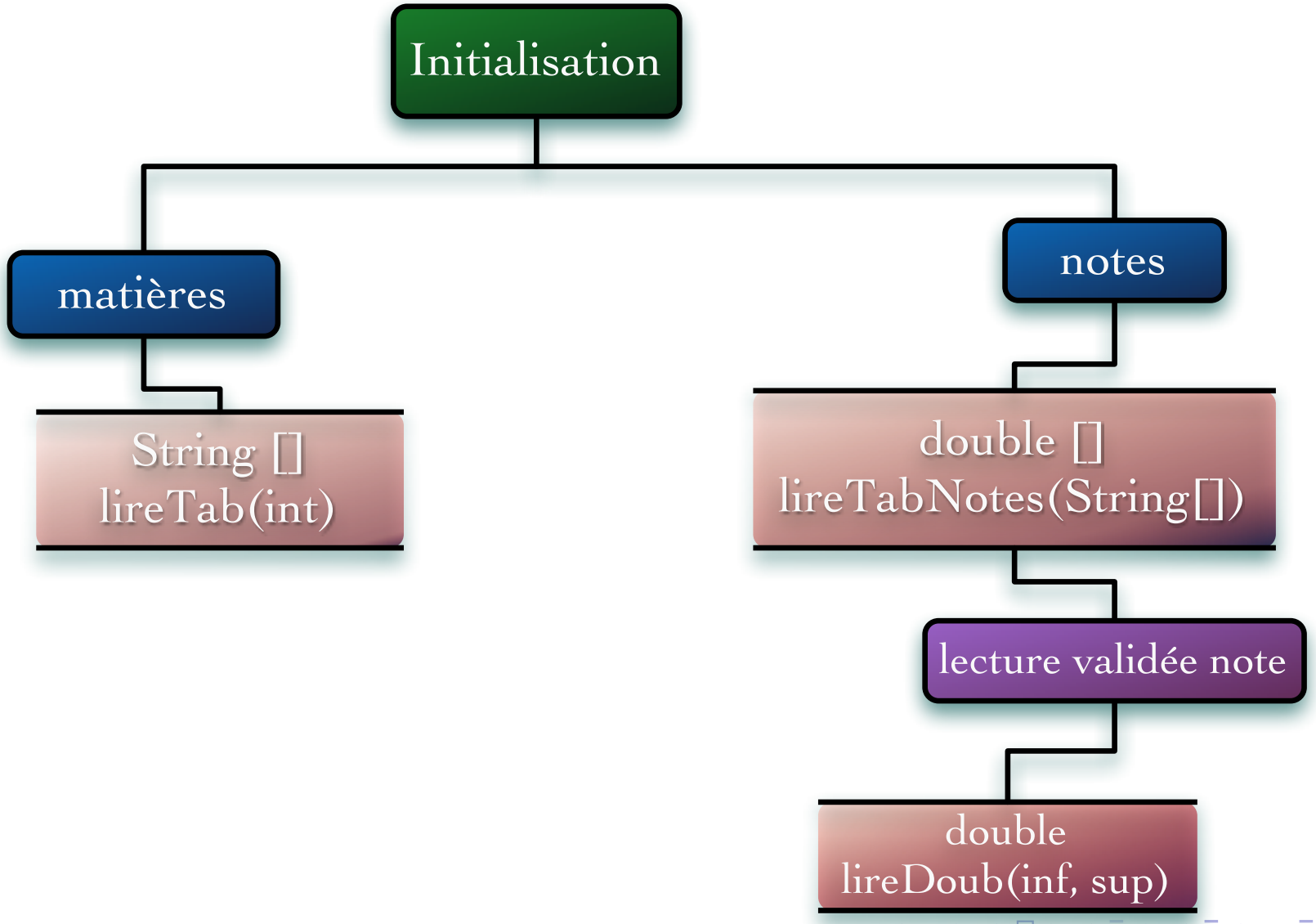
Opérations sur les données :

- afficher le contenu (matière + note) des 2 tableaux
- moyenne tableau notes ;
- chercher une note étant donné un nom de matière ;
- modifier une note étant donné un nom de matière ;
- intialiser les 2 tableaux ;

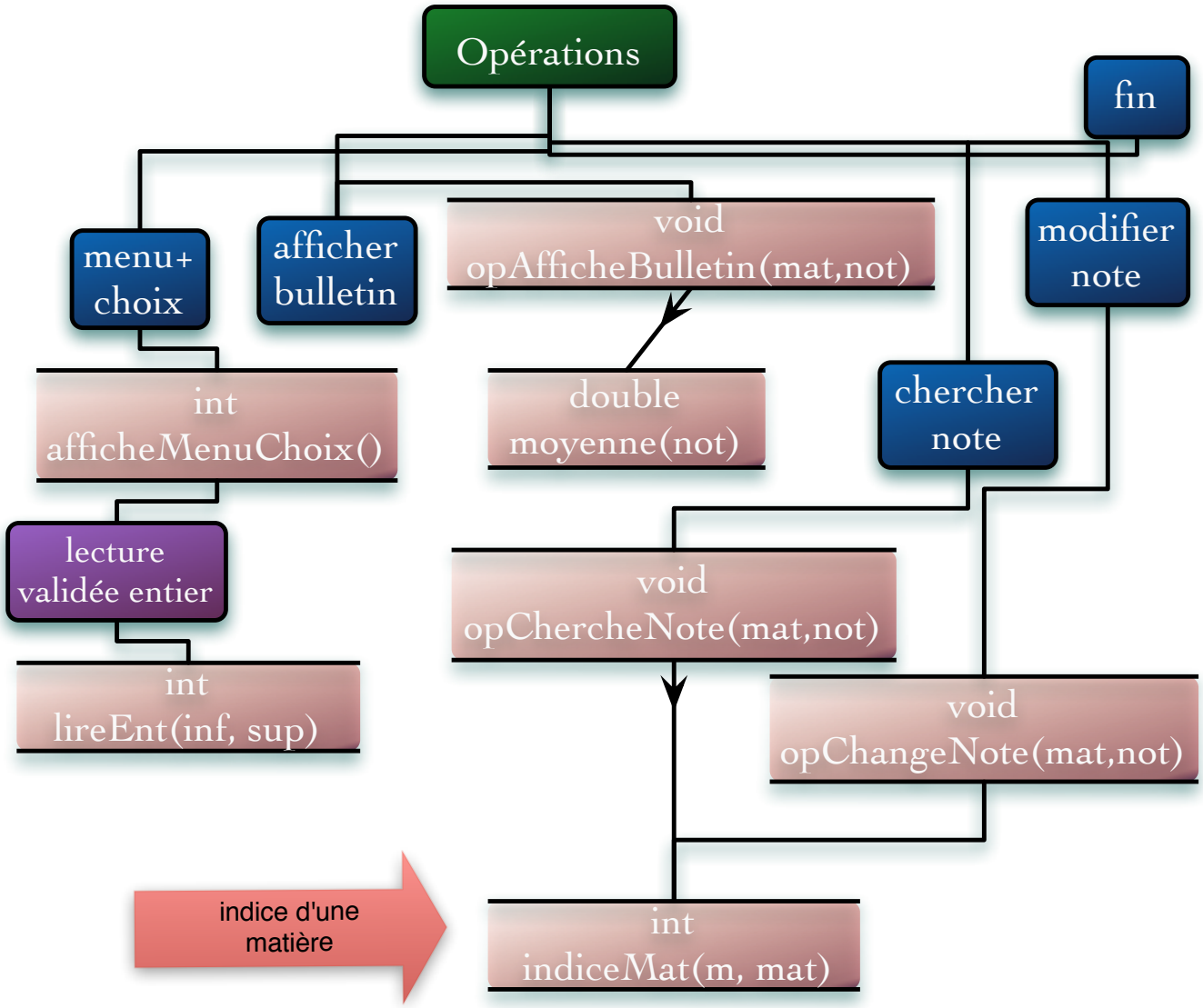
Sous-tâches du main



Sous-tâches d'initialisation des données



Sous-tâches pour les opérations



Liste détaillée de méthodes

Nous détaillons :

- entêtes des méthodes par sous-tâche identifiée,
- commentaires précis de ce qu'elle doit faire + ses hypothèses.

Intialiser les 2 tableaux

```
/* Lit un tableau de taille n */
static String [] lireTabString (int n)

/* Lit une note pour chaque matiere dans m */
static double [] lireTabNotes (String [] m)
```

Attention à lire un tableau avec au moins une matière $\Rightarrow$ appel avec paramètre $n \geq 1$.

Liste de méthodes (2)

- afficher le contenu (matière + note) des 2 tableaux

```
/* Affiche matieres de m avec notes de t
 * et leur moyenne
 */
static void opAfficheBulletin(double [] t,
                              String [] m)
```

⇒ a besoin de :

- calcul moyenne tableau notes :

```
static double moyenneTab (double [] t)
```

Préciser les hypothèses des méthodes

Important :

- préciser conditions indispensables au bon fonctionnement des méthodes $\Rightarrow$ dans commentaires
- ex : le calcul de la moyenne se fait sur un tableau non vide ;
- cela nous aide à penser par la suite à les garantir lors des appels
- $\Rightarrow$ appel avec tableau non vide

```
/*Retourne la moyenne d'un tableau t de double
 * Hypothese: t est non vide
 */
static double moyenneTab (double [] t)
```

Liste de méthodes (3)

- chercher une note étant donné un nom de matière ;

```
/* Lit un nom de matiere et cherche son indice
 * dans m. Si trouve, affiche sa note dans t,
 * sinon signale une erreur
*/
static void opChercheNote(String [] m, double [] t)
```

⇒ a besoin de :

- chercher l'indice d'une chaîne dans un tableau de chaînes

```
/* Retourne l'indice d'une matiere si existe
 * et -1 sinon.
**/
static int indiceMat(String a, String [] m)
```

Liste de méthodes (4)

- modifier une note étant donné un nom de matière ;

```
static void opChangeNote(String [] m, double [] t)
```

$\Rightarrow$ a (comme avant) besoin de :

- chercher l'indice d'une chaîne dans un tableau de chaînes

```
static int indiceMat(String a, String [] m)
```

Liste méthodes (5)

Autres :

- afficher le menu d'opérations et lire un choix,

```
/* Affiche les operations du menu
 * lit un choix (valide) de l'utilisateur
 * renvoi ce choix en resultat
 */
static int afficheMenuEtChoix()
```

- lecture validée d'une option du menu, d'une note, etc.

```
static int lireEnt(String me, int inf, int sup)
static double lireDou(String me,
                      double inf, double sup)
```

Pouvons nous écrire un main avec ces méthodes ?

```
int nb = lireEnt("Nombre␣de␣matieres?␣",1,20);
String [] matieres =  lireTabString(nb);
double [] notes = lireTabNotes(matieres);
boolean fin = false;
while (!fin) {
    int choix = afficheMenuEtChoix();
    if (choix == 1) {
        opAfficheBulletin(notes, matieres);
    } else if (choix == 2) {
        opChercheNote(matieres, notes);
    } else if (choix == 3) {
        opChangeNote(matieres, notes);
    } else if (choix == 4) {
        Terminal.ecrireStringln("␣***␣Au␣revoir...␣␣***␣
        fin=true;
    }
}
```

Bilan analyse, début du codage

Principe : l'analyse finit quand . . . commence le codage + tests.

En pratique, on se contente de :

- savoir bien expliquer comment représenter les données,
- comment s'en servir pour traiter le problème,
- et quelles sont les méthodes à écrire,

L'écriture du main permet de mettre à l'épreuve la pertinence de notre analyse.

Codage et test

Par où commencer à coder ? :

- Principe : coder en premier les méthodes « feuilles », n'ayant besoin d'autres méthodes ;
- méthodes d'affichage qui permettront de faire des tests ;
- laisser l'implantation de lecture validée pour plus tard, etc...
- chaque opération codée $\Rightarrow$ testée minucieusement : permettra de passer à autre chose sans avoir à y revenir.

Compilation et test

- Principe : déclarer toutes les méthodes, avec corps minimale permettant de compiler :

```
/* Operation affichage du bulletin de notes */
 static void opAfficheBulletin(double [] t,
                               String [] m) {
   Terminal.ecrireStringln("non implante");
  }

/* Retourne l'indice d'une matiere si elle existe
 * et -1 sinon. **/
 static int indiceMat(String a, String [] m) {
    return -1;
 }
```

- But : compiler tout le programme ; implantation incrémentale.

Codage méthodes initialisation

```
 /* Lecture d'un tableau de String de taille n */
 static String [] lireTabString (int n) {
   String [] t = new String[n];
   for (int i=0; i< t.length; i++)  {
         Terminal.ecrireString("Une matiere? ");
         t[i] = Terminal.lireString();
   }
   return t;
}
```

Méthodes initialisation (2)

```
/* Lecture d'un tableau de notes par matiere */
static double [] lireTabNotes (String [] m) {
    double [] t = new double[m.length];
    for (int i=0; i< t.length; i++)  {
      t[i] = lireDou("Note de " + m[i]+" ? ",0,20);
    }
    return t;
}
```

⇒ a besoin de `lireDou`.

Méthode lecture double entre inf et sup

```
/* Lecture d'un double compris entre inf et sup
 * avec  message
 */
static double lireDou(String message,
                      double inf, double sup) {
  while (true)  {
    Terminal.ecrireString(message);
    double n = Terminal.lireDouble();
    if (n<inf || n> sup) {
       Terminal.ecrireStringln("Doit␣etre␣entre␣"+ inf+
    } else return n;
  }
}
```

Opérations sur le menu

```
 /* Affichage menu et lecture choix operation.
  * Retourne un choix valide d'operation
  */
static int afficheMenuEtChoix(){
   Terminal.sautDeLigne();
   Terminal.ecrireStringln("Bonjour. Operations disponib
   Terminal.ecrireStringln("  1. Afficher le bulletin");
   Terminal.ecrireStringln("  2. Chercher une note");
   Terminal.ecrireStringln("  3. Mofidier une note");
   Terminal.ecrireStringln("  4. Finir. ");
   return(lireEnt("Votre choix ---> ", 1, 4));
}
```

Opération de recherche d'une note

```
/* Operation recherche d'une note */
static void opChercheNote(String [] m,
                          double [] t) {
 Terminal.ecrireStringln("Nom de la matiere a chercher? "
 String nm = Terminal.lireString();
 int ind = indiceMat(nm, m);
 if (ind <0) {
    Terminal.ecrireString("Matiere non trouvee.");
 } else {
    Terminal.ecrireString("Note de "+nm+" = "+t[ind]);
 }
}
```

⇒ a besoin de `indiceMat`.

Indice d'une matière

```
/* Retourne l'indice d'une matiere si elle existe
 * et -1 sinon.
 **/
 static int indiceMat(String a, String [] m) {
   for (int i=0; i<m.length; i++){
     if (a.equals(m[i])) return i;
     }
    return -1;
 }
```

Fin du codage

Le reste du codage est laissé en exercice !