Analyse d'une petite application avec interface texte

(NFA031 - Jour)

V. Aponte

Cnam

11 décembre 2017

1. Que contient une analyse minimale ?

Analyse d'un problème (pour concevoir une solution)

Objectif : aide à la conception d'une application

1. Liste de fonctionnalités à offrir + exemples de « scénarios » d'intéraction utilisateur/application.
2. Proposer une structures de données. Donner des exemples (via des dessins) d'actions du programme sur ces données. Etablir leurs contraintes (types, taille, intervalles, liens entre les variables).
3. Etablir une liste des opérations sur les données, sous forme de sous-programmes $\Rightarrow$ noms + paramètres+ types+ description fonctionnalité (sans dire comment) ;

Guide : pouvons nous écrire un `main` simple et court en utilisant uniquement ces données et ces opérations ?

Le problème : gestion de notes d'un élève par matière

Gérer les notes d'un élève pour un ensemble de matières, avec opérations de calcul de moyenne, recherche de la note d'une matière, modification d'une note, affichage du bulletin.

Quelques critères de qualité sur l'application à écrire :

- doit répondre au problème posé :
  - ensemble de fonctionnalités suffisant et adéquat ;
  - chaque opération fonctionne corretement ;
- doit être robuste :
  - les erreurs les plus courantes doivent être detectées et signalées, sans faire planter le tout, ni imposer de tout recommencer
- doit être facile à utiliser : clarté, ergonomie.

Les fonctionnalités de notre application

Notre application comportera deux phases. Dans la 1ère on initialise les données ; dans la 2ème on propose des opérations sur ces données.

- Phase 1 : Initialisation :
  1. lire une liste de noms de matières et une note par matière,
- Phase 2 : Boucle avec menu d'opérations (sur ensemble fixe de matières)
  1. afficher le bulletin des notes
  2. chercher la note d'une matière donnée par son nom.
  3. modifier la note d'une matière donnée par son nom.
  4. finir
- Opérations sur version ultérieure : ajouter/supprimer des matières.

Les fonctionnalités : Interface texte

Un scénario pour l'interface texte. En donner pour chaque fonctionnalité !

```
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 la note pour un nom de matiere.
  3. Mofidier la note pour un nom de matiere.
  4. Finir
Votre choix --->
```

Les fonctionnalités : erreurs à signaler sans planter

Queques exemples :

- Note invalide (on doit établir au préalable un intervalle valide),
- Numéro d'opération invalide (il y en a 4 possibles dans notre menu),
- Nom de matière inexistant ;
- Si on sait traiter les exceptions : erreur de type si saisie de caractères non numériques lors de la lecture d'un nombre.
- Le nombre de matières doit être 1 ou plus.

Représentation (structure) des données

Répresentation des données : 2 tableaux

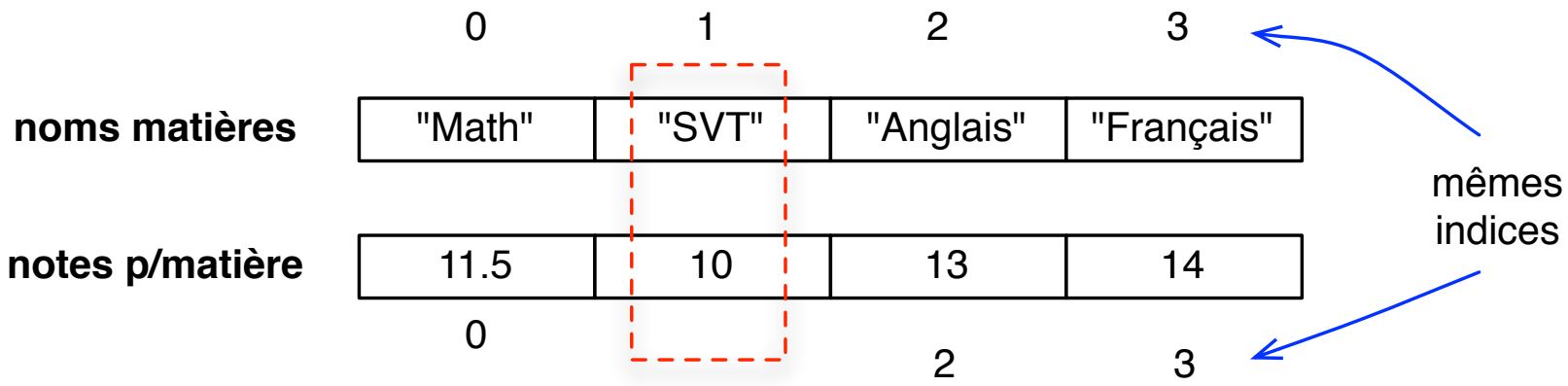
- 1 tableau de chaînes avec les noms des matières,
- 1 tableau de notes (doubles)
- Contraintes : les tableaux ont la même taille, au moins 1 composante, une matière et sa note aux mêmes indices sur les 2 tableaux.

Données auxiliaires (exemples) :

- tableau de chaînes avec les noms des opérations ;
- tableau de chaînes avec les messages d'erreur.

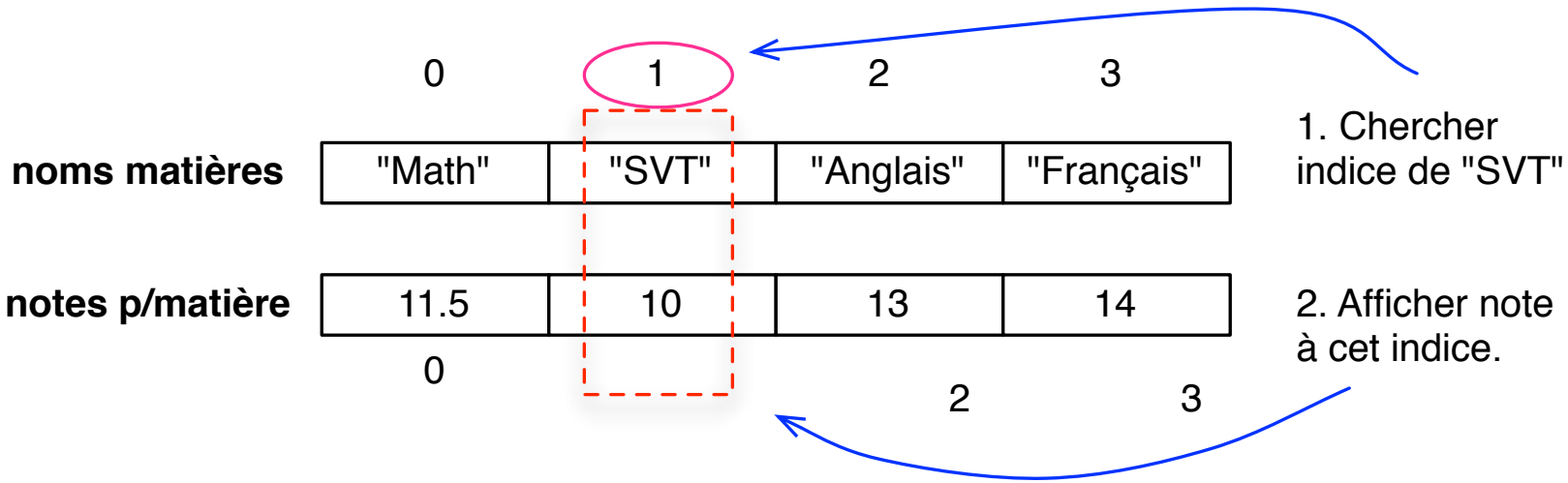
Représentation des données : un dessin

Répresentation des données : 2 tableaux de même taille avec données en relation aux mêmes indices.



Exemple de relation entre les données:
la note de SVT (indice 1, tableau matières) est 10 (indice 1, tableau notes)

Représentation des données : exemple d'opération sur les données



Exemple d'opération sur les données: "Afficher la note de SVT "

Donner au moins un exemple par fonctionnalité proposée !

Les opérations à implanter

Opérations sur les données :

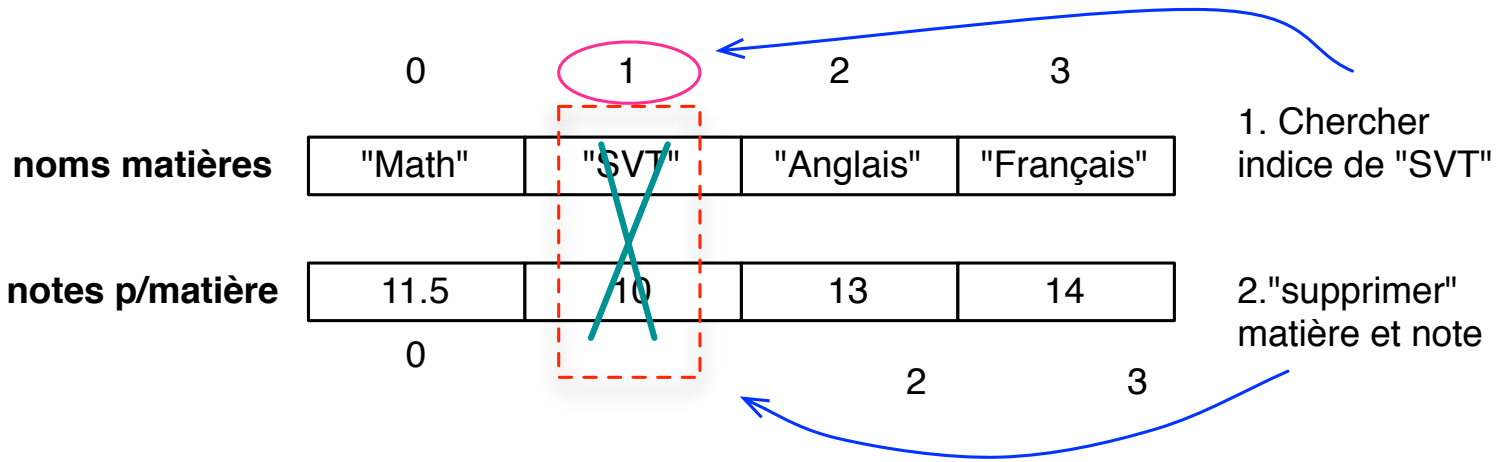
- intialisation (lecture) des données (tableaux) de la phase 1.
- afficher le contenu (matière + note) des 2 tableaux
- calculer moyenne d'un tableau de double ;
- chercher une note étant donné un nom de matière ;
- modifier une note étant donné un nom de matière ;

Opérations auxilaires (exemples) :

- obtenir l'indice d'un nom de matière dans le tableau de matières ;
- lire une note valide ;
- lire un numéro valide d'opération ;
- méthode qui implante la boucle d'opérations.

Représentation des données en cas d'ajout/suppresion

Si nous proposons des opérations pour ajouter et/ou supprimer une matière, notre tableau pourra être trop petit (en cas d'ajout) ou avoir des cases que nous voulons considérer comme « non remples ».



Exemple d'opération sur les données: "Supprimer matière SVT "

Problème : la taille du tableau change et on doit le réorganiser...

L'analyse que l'on attend de vous (avant codage)

- Au minimum : correspond à l'équivalent des transparents jusqu'ici, avec des exemples/dessins/scénarios pour chaque fonctionnalité.
- Dans l'idéal : + les profils de toutes les méthodes correspondant à des opérations sur les données. Ce qui donnerait par exemple :

```
static String [] lireTabString (int nb)
static double [] lireTabNotes (String [] mat)
static void opAfficheBulletin(double[] notes,String[] mat)
static double moyenneTab (double [] notes)
static int indiceMat(String nom, String [] mat)
static void opChercheNote(String [] mat, double [] notes)
static void opChangeNote(String [] mat, double [] notes)
```

2. Utiliser les tableaux pour représenter une liste « dynamique »

Liste dynamique

C'est quoi ? : structure de données dans laquelle on peut ajouter/supprimer des éléments sans se soucier de la taille de la liste. Nous utiliserons des tableaux pour les implanter.

- si on supprime un élément « au milieu », cette place pourra être utilisée pour stocker un nouvel ajout ;
- si on ajoute un nouvel élément le tableau interne est rédimensioné automatiquement (si nécessaire) afin de garantir l'ajout.
- on vuet accèder aux éléments via leur indice dans le tableau interne.

$\Rightarrow$ en Java : « ArrayList »

Liste dynamique : deux implantations

Nous étudions une version simplifiée pour stocker des entiers et *sans opération de redimensionnement* et avec 2 implantations :

- tableaux à trous d'entiers (TT),
- tableaux incomplet d'entiers (TI).

Propriétés importantes :

- opérations d'ajout, recherche, suppresion.
- possibilité d'accèder à un élément via son indice.
- très important : ordre des éléments ajoutés
  - TT : non préservé.
  - TI : préservé.

« Tableaux à trous »

Hypothèse : il existe une donnée « impossible » à insérer dans la liste. Ex : -1, si la liste contient des entiers positifs ; null, si elle contient des chaînes, etc.

Idée : les cases avec valeurs « impossibles » seront considérées comme « vides ».

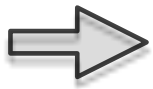
- Représentation : tableau initilaisée avec valeurs impossibles ;
- Opérations
  - *ajout* : dans 1ère case contenant valeur impossible (nous ne traitons pas le cas où il ne reste plus de place) ;
  - *suppresion* : mettre valeur impossible dans case correspondante ;
  - *énumération/recherche* : parcous en ignorant valeurs impossibles ;

Opération supplémentaire : rédimensionnement du tableau avec recopie vers tableau plus grand si plus de place.

Représentation 1 : « tableaux à trous »

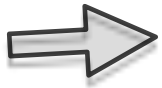
Exemple : un tableau d'entiers positifs initialisé avec -1 partout.

initialisation



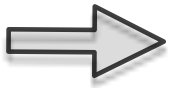
-1	-1	-1	-1	-1	-1

ajout : 1, 2, 3, 4



1	2	3	4	-1	-1

sup: 3



1	2	-1	4	-1	-1

Opérations pour tableaus à trous

Paramètre : tableau

- *affichage du tableau* : procédure ;
- *recherche* : retourne l'élément si trouvé, échoue sinon ;
- *suppresion* : fonction booléenne (élément à supprimer + tableau)
  - si non trouvé $\Rightarrow$ renvoie *false* ;
  - si trouvé $\Rightarrow$ *true* + modifie tableau paramètre ;
- *ajout* : fonction booléenne (élément à ajouter + tableau) ;
  - si plus de place $\Rightarrow$ retourne *false* ;
  - si ajout possible $\Rightarrow$ *true* + modifie tableau paramètre ;

Méthode d'affichage d'un tableau à trous

```
/* Affiche les elements d'une liste d'entiers
 * implantee par un tableau a trous.
 **/
 static void afficheListTT(int [] t) {
   for (int i=0; i<t.length; i++){
     if (t[i]<> -1) Terminal.ecrireIntln(t[i]);
   }
 }
```

Méthode de suppresion d'élément d'un TT

```
/* Supprime la 1ere occurence de a si present.
 * Renvoie true si trouve (et supprime) false sinon.
 **/
 static boolean supprListTT(int a, int [] t) {
   for (int i=0; i<t.length; i++){
     if (t[i]==a) {
           t[i] = -1;
           return true;
     }
     return false;
 }
```

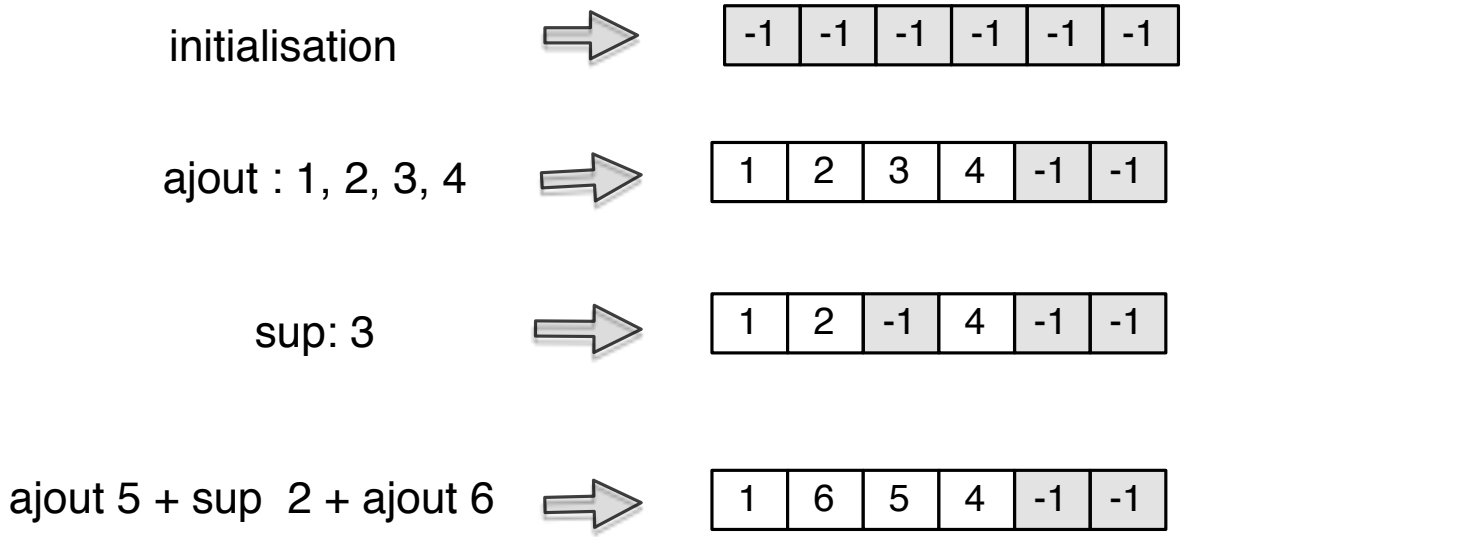
Si trouvé, tableau paramètre modifié ; retourne false sinon.
Cette opération est peu couteuse : dés que l'élément est trouvé, on met une valeur impossible à la place.

Méthode ajout élément dans TT

```
/*Ajoute a dans la liste.
 * Renvoie true si ajoute, false s'il n'y a plus de place.
 **/
 static boolean ajoutListTT(int a, int [] t) {
   for (int i=0; i<t.length; i++){
     if (t[i]==-1) {
           t[i] = a;
           return true;
     }
     return false;
 }
```

S'il reste de la place, tableau paramètre modifié ; retourne false sinon.
Cette opération est plus couteuse : il faut parcourir le tableau pour trouver une place libre.

Tableaux à trous : ordre d'ajout non preservé



Chaque élément est le numéro du ticket pris par un client dans un guichet…
Qu'observe-t-on?

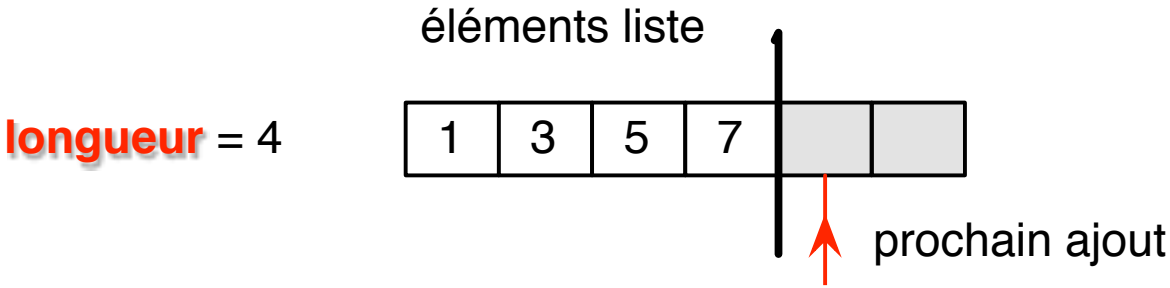
$\Rightarrow$ l'ordre des ajouts n'est pas préservé.

En préservant l'ordre on devrait avoir : 1, 4, 5, 6

Répresentation 2 : « tableau incomplet »

Idée : tableau surdimensionné où les éléments ajoutés sont rangés « à gauche » en préservant l'ordre d'ajout.

- variable longueur = nombre d'éléments présents ET indice du prochain à ajouter.
- initialement longueur = 0 ;
- tous les éléments mis à « gauche » (pas de trous)
  $\Rightarrow$ dans $t[0] \ldots t[longueur - 1]$.
- partie « vide » à « droite » $\Rightarrow$ $t[longueur] \ldots t[t.length - 1]$;



« Tableau incomplet » (2)

- Représentation interne : tableau ET variable longueur
- Opérations
  - *ajout* : dans la case t[longueur] (sauf si plus de place) +incrément de longueur.
  - *énumération/recherche* : parcous jusqu'à longueur ;
  - *suppresion* : si *a* est à supprimer, on déplace une case à gauche toutes les valeurs se trouvant à droite de a + décrément de longueur ;

Opération supplémentaire : rédimensionnement du tableau.

L'ordre des ajouts est-il préservé ? Oui

Méthodes pour opérations sur TI

Paramètres : tableau + longueur :

- *affichage* : procédure ;
- *suppresion* : fonction booléenne (élément à supprimer + tableau)
  - si non trouvé $\Rightarrow$ *false* ;
  - si trouvé $\Rightarrow$ *true* + modifie tableau paramètre ;
  - difficulté : doit décrementer *longueur* si supprimé. La méthode ne peut pas le faire $\Rightarrow$ décrementer au retour de l'appel, si résultat *true* ;
- *ajout* : fonction booléenne (élément à ajouter + tableau) ;
  - si plus de place $\Rightarrow$ *false* ;
  - si ajout possible $\Rightarrow$ *true* + modifie tableau paramètre ;
  - difficulté : doit incrémenter *longueur* si ajout effectué $\Rightarrow$ au retour de l'appel, si résultat *true* ;

Méthode d'affichage de TI

```
/* Affiche les elements d'une liste
 * implantee par un tableau incomplet
 **/
 static void afficheTI(int [] t,int longueur) {
   for (int i=0; i< longueur; i++){
     Terminal.ecrireIntln(t[i]);
   }
 }
```

Méthode ajout sur TI

```
/*Ajoute a dans la liste.
 * Renvoie true si ajoute, false s'il n'y a plus de place.
 **/
 static boolean ajoutTI(int a, int [] t, int lo) {
   if (lo >= t.length) return false;
   t[lo] = a;
   return true;
 }
```

S'il reste de la place, tableau paramètre modifié ; retourne false sinon.

Incrément longueur : *au retour* de chaque appel.

Appel ajout sur TI

L'incrément de la longueur doit se faire *au retour* de l'appel.

```
int [] t = new int [10];
int longueur = 0;
boolean r = ajoutTI(1, t, longueur);
if (r) {
  longueur ++;
}
```

Méthode ajout sur TI : solution alternative

longueur = tableau1 case $\Rightarrow$ méthode peut le modifier.

```
static boolean ajoutTI(int a, int [] t, int [] lo) {
  int n = lo[0];
  if (n >= t.length) return false;
  t[n] = a;
  lo[0]++;  // increment longueur
  return true;
}
```

Exemple d'appel :

```
int [] t = new int [10];
int [] ln = {0};    // tableau 1 case
ajoutTI(1, t, ln); // au retour: ln[0] vaut 1
```

Méthode suppresion dans TI

```
/* Supprime la 1ere occurence de a si present.
 * lo: tableau 1 case avec longueur
 * renvoie false si absent
 **/
 static boolean supprTI(int a, int [] t, int [] lo )
```