

Amphi 4: tableaux/String

15 mai

- I. Syntaxe et premiers exemples.
- II. Organisation mémoire.
- III. Quelques exemples.
- IV. Un peu de chaînes de caractères.
- V. Entrées-sorties (suite).

I. Syntaxe et premiers exemples

Qu'est-ce qu'un tableau ? Une structure de données qui permet d'utiliser un grand nombre de variables de même type. C'est l'un des objets qui permettent le traitement massif des données.

À quoi servent les tableaux ?

- **Stocker** des données de même type (dictionnaires, etc.);
- **modéliser** vecteurs, matrices, polynômes; images, etc.;
- **implanter** de manière efficace de multiples structures de données (piles, files, arbres, graphes).

0. Rappels des épisodes précédents

- **Amphi 1: introduction.**
- **Amphi 2: programmer en Java.**
- **Amphi 3: fonctions/fonctions récursives.**
- **Amphi 4: tableaux/String.**
- **Amphi 5: classes.**

Le premier exemple

```
int x = 1;
```

x
1

```
int[] t = new int[5];  
for(int i = 0; i < 5; i++)  
    t[i] = i*i;
```

t.length	t[0]	t[1]	t[2]	t[3]	t[4]
5	0	1	4	9	16

```
for(int i = 0; i < t.length; i++)  
    System.out.println(t[i]);
```

Déclaration:

```
typ[] t;
```

Par défaut, un tableau est initialisé à la valeur `null`. Il faut donc le construire:

```
t = new typ[taille];
```

`new` demande de la mémoire au système pour créer un tableau de `taille` éléments de type `typ`.

Les éléments du tableau sont initialisés à la valeur par défaut du type (zéro pour les `int`) à la création.

Exemple: polynômes

$$P(X) = p_d X^d + p_{d-1} X^{d-1} + \dots + p_0$$

Mathématiquement: d est le degré; si $P \neq 0$, $p_d \neq 0$; si $P \equiv 0$, $d = -\infty$.

Pour simplifier: si $P \equiv 0$, $d = 0$, $p_0 = 0$.

⇒ on représente P par le tableau de ses $d + 1$ coefficients.

```
public class Polynomes{
    public static void main(String[] args){
        double[] P = {0.0, 1.0, 2.0, 1.0};

        System.out.print("P=");
        for(int i = 0; i < P.length; i++){
            System.out.print("+"+P[i]+"*X^"+i);
            System.out.println();
        }
    }
}
```

```
P=+(0.0)*X^0+(1.0)*X^1+(2.0)*X^2+(1.0)*X^3
```

Propriétés:

- Les cases du tableau sont **numérotées à partir de 0**: elles sont désignées par `t[0]`, `t[1]`, ..., `t[taille-1]`.
- On peut récupérer la taille de `t` à l'aide de `t.length`.
- `t[0]`, `t[1]`, ... se manipulent comme des variables ordinaires.
- **Java** teste les **débordements** de tableau : interdit `t[-1]`, `t[t.length]`, etc.

Déclarer, construire et initialiser en une seule instruction:

```
int[] t = {6, -4, 7};
```

Java déduit et fixe la taille de `t`; `t[0]` contient 6, `t.length` contient 3, etc.

Les deux erreurs les plus fréquentes

```
int[] t;
t[0] = 2;
```

Bug1.java:4:

```
variable t might not have been initialized
    t[0] = 2;
    ^
```

1 error

```
int[] t = {1, 2, 3};
t[3] = 10;
```

Exception in thread "main"

```
java.lang.ArrayIndexOutOfBoundsException: 3
    at Bug2.main(Bug2.java:4)
```

Exception: permet à une fonction de très bas niveau d'avertir une fonction de haut niveau qu'un problème est apparu, lui laissant le soin de résoudre le problème éventuellement.

Tableaux bi-dimensionnels

En **Java**, un tableau à deux dimensions est un tableau de tableaux (lignes).

Déclaration: `typ[][] tab;`

Construction d'un tableau de n lignes à m colonnes:

```
tab = new typ[n][m];
```

`tab.length` retourne le nombre de lignes et `tab[i].length` le nombre de colonnes (de `tab[i]`).

Ex. Création d'un tableau de 3 lignes et 2 colonnes initialisé avec des valeurs numériques :

```
int[][] tab={{1, 2}, {4, 5}, {7, 8}};
```

Exemple simple

```
public static void main(String[] args){
    int[][] tab = {{1,2},{4,5},{7,8}};

    for(int i = 0; i < tab.length; i++){
        for(int j = 0; j < tab[i].length; j++){
            System.out.print(" "+tab[i][j]);
            System.out.println();
        }
    }
}
```

```
1 2
4 5
7 8
```

Vecteurs, matrices

A matrice $n \times m$, v un vecteur $m \times 1$; $w = Av$ est un vecteur $n \times 1$ tq:
 $\forall i \in [0, n-1], w_i = \sum_{k=0}^{m-1} A_{i,k} v_k$.

```
public class Matrices{
    public static void main(String[] args){
        int n = 3, m = 4;
        double[][] A = matriceAleatoire(n, m);
        double[] v = vecteurAleatoire(m);
        double[] w = multiplier(A, v);

        afficherVecteur(w);
    }
}

public static void afficherVecteur(double[] w){
    for(int i = 0; i < w.length; i++)
        System.out.println("w["+i+"]="+w[i]);
}
```

```
public static double[][] matriceAleatoire(int n,
                                           int m){
    double[][] A = new double[n][m];

    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            A[i][j] = Math.random();
        }
        return A; // le tableau créé est retourné
    }

    public static double[] vecteurAleatoire(int m){
        double[] v = new double[m];

        for(int i = 0; i < m; i++){
            v[i] = Math.random();
        }
        return v;
    }
}
```

```
public static double[] multiplier(double[][] A,
                                double[] v){
    int n = A.length, m = A[0].length;
    double[] w = new double[n];
    for(int i = 0; i < n; i++){
        w[i] = 0;
        for(int k = 0; k < m; k++){
            w[i] += A[i][k]*v[k];
        }
    }
    return w;
}
```

Pour tester, on prend des matrices/vecteurs simples.

Que se passe-t-il pour un tableau?

tableau = référence

```
int[] t;           // bon de commande
                  // pour la commode
t = new int[3];   // on construit la commode
t[0] = 2;         // on remplit des tiroirs
t[1] = 1;
t[2] = t[0]+t[1]; // on les utilise
```

II. Organisation mémoire

La mémoire d'un ordinateur peut être vue comme un tableau, dont l'organisation exacte n'a pas besoin d'être connue du programmeur. Tous les acteurs présents dans un programme (variables, tableaux, objets) sont stockés dans ce tableau, et repérés par un indice qu'on appelle référence (ou adresse).

```
int x = 3, y = 4, z;
```

@100	...	@200	...	@300
x	...	y	...	z
3	...	4	...	?

```
z = x + y;
```

@100	...	@200	...	@300
x	...	y	...	z
3	...	4	...	7

```
int[] t;
```

@10
t
@0 (null)

```
t = new int[3];
```

@10	@500	@504	@508	@512
t	t.length	t[0]	t[1]	t[2]
@500	3	0	0	0

```
t[0] = 2;
```

@10	@500	@504	@508	@512
t	t.length	t[0]	t[1]	t[2]
@500	3	2	0	0

Une conséquence non triviale

```
int[] t = {1, 2, 3}; // t = @500

int[] u = t; // u = @500
u[0] = 0;
System.out.println(t[0]);
```

t et u font référence à la même suite d'emplacements:

	t			
	↓			
u →	t.length	t[0]	t[1]	t[2]
	3	0	2	3

Tableau bi-dimensionnel (suite)

```
int[][] t = {{1,2}, {4,5}, {7,8}}; // t = @500
```

@10	@500	@504	@508	@512
t	t.length	t[0]	t[1]	t[2]
@500	3	@600	@700	@800

@600	@604	@608
t[0].length	t[0][0]	t[0][1]
2	1	2

@700	@704	@708
t[1].length	t[1][0]	t[1][1]
2	4	5

@800	@804	@808
t[2].length	t[2][0]	t[2][1]
2	7	8

Pour recopier un tableau

Création d'une copie différente de l'original:

```
int[] t = {1, 2, 3}; // t = @500

int[] u = new int[t.length]; // u = @600
for(int i = 0; i < t.length; i++)
    u[i] = t[i];
```

@500	@504	@508	@512
t.length	t[0]	t[1]	t[2]
3	1	2	3

@600	@604	@608	@612
u.length	u[0]	u[1]	u[2]
3	1	2	3

Les tableaux comme argument de fonction)*()

Rappel: Java passe toujours les arguments des fonctions par valeur (recopie).

```
public static void f(int n){
    n = -10;
}

public static void main(String[] args){
    int n = 1;

    f(n);
    System.out.println(n);
}
```

Le programme affiche 1.

Pour les tableaux)*(*

On passe par valeur la référence du tableau.

```
public static void f(int[] w){ // w = @100
    w[0] = -10;
}
public static void main(String[] args){
    int[] t = new int[5]; // t = @100

    t[0] = 1;
    f(t);
    System.out.println(t[0]);
}
```

Le programme affiche -10 , car on a modifié la mémoire globale (du processus).

En récursif

```
// remplit les cases de i à t.length-1
public static void remplirAux(int[] t, int i){
    if(i < t.length){
        t[i] = (int)(Math.random()*M);
        remplirAux(t, i+1);
    }
}

public static void remplir(int[] t){
    remplirAux(t, 0);
}
```

III. Quelques exemples

Remplissage aléatoire

Pour remplir un tableau avec des entiers aléatoires de $[0, M[$, on écrit:

```
public class Tableau{
    // définition de constantes
    // connues de toute la classe
    final static int M = 128, N = 100;

    public static void main(String[] args){
        int[] t = new int[N];

        for(int i = 0; i < N; i++){
            t[i] = (int)(Math.random()*M);
        }
    }
}
```

Rem. il faut convertir `Math.random()*M` en entier avec une conversion explicite.

Recherche d'un élément

Question: le nombre n est-il dans le tableau, si oui dans quelle(s) case(s) ?

```
public class Recherche{
    public static void main(String[] args){
        int[] t = {1, 0, -10, 17, 20, 17};

        System.out.print("Entrer un nombre=");
        int n = TC.lireInt();
        for(int i = 0; i < t.length; i++){
            if(t[i] == n)
                System.out.println(i);
        }
        return;
    }
}
```

Recherche du plus petit élément

Principe: on utilise une variable `mint` qui va contenir la plus petite valeur trouvée jusqu'à présent:

```
int mint;

mint = t[0];
for(int i = 1; i < t.length; i++)
    // invariant: mint = min(t[j], j = 0..i-1)
    if(t[i] < mint)
        mint = t[i];
System.out.println("Le minimum est "+mint);
```

18	3	10	25	9	3	11	13	23	8
----	---	----	----	---	---	----	----	----	---

Combien de nombres sont présents ?

Pb: pour chaque $n \in [0, M[$, on veut afficher combien de fois n est présent dans t si cette valeur est > 0 .

Première idée: pour chaque $n \in [0, M[$, on parcourt t .

```
int cpt;

for(int n = 0; n < M; n++){
    cpt = 0;
    for(int i = 0; i < t.length; i++)
        if(t[i] == n)
            cpt++;
    if(cpt > 0)
        System.out.println(n+" "+cpt);
}
```

Jouons à la bataille rangée

Seconde idée: on alloue un tableau `cpt` de taille M et on parcourt une seule fois t ; `cpt[i]` contiendra le nombre de fois où i est présent.

```
int[] cpt = new int[M];

for(int i = 0; i < M; i++)
    cpt[i] = 0;
for(int i = 0; i < t.length; i++)
    cpt[t[i]] += 1;
for(int n = 0; n < M; n++)
    if(cpt[n] > 0)
        System.out.print(n+" "+cpt[n]);
```

Règle: on distribue un paquet de cartes à deux joueurs. À chaque tour, le gagnant est celui qui a la plus forte carte. On demande les scores des joueurs à la fin.

Programmation:

- on bat les cartes;
- on distribue le jeu;
- on joue;
- on affiche le vainqueur.

On modélise un paquet de n cartes, numérotées de 0 à $n - 1$, par un tableau t de taille n (par exemple, 32).

```

public class BatailleRangee{
    public static void main(String[] args){
        int[] donne, jeuA, jeuB;

        donne = creerEtBattre(32);
        afficher(donne);
        jeuA = new int[16]; // jeu de A
        jeuB = new int[16]; // jeu de B
        distribuer(jeuA, jeuB, donne);
        int gainA = jouerAB(jeuA, jeuB);
        if(gainA > 0)
            System.out.println("A gagne");
        else if(gainA < 0)
            System.out.println("B gagne");
        else
            System.out.println("ex-aequo");
        }
    }
}

```

Comment battre les cartes?

Première idée: on considère les cartes rangées dans l'ordre 0..31.

Pour déterminer la première carte du jeu, on tire un numéro j au hasard entre 0 et 31 et on prend la carte j qu'on retire du premier paquet pour la mettre dans le second.

Pour la i -ème carte, on procède de la même manière, mais on ne peut plus prendre une carte qui a déjà été prise.

```

public static int[] creerEtBattre(int n){
    int[] jeu = new int[n];

    for(int i = 0; i < n; i++){
        jeu[i] = i;
        battre(jeu);
    }
    return jeu;
}

// pratique pour le debug
public static void afficher(int[] t){
    System.out.print("[");
    for(int j = 0; j < t.length; j++){
        System.out.print(" "+t[j]);
    }
    System.out.println("]");
}

```

```

public static void battre(int[] t){
    int n = t.length, j;
    boolean[] utilisee = new boolean[n];

    for(int i = 0; i < n; i++){
        utilisee[i] = false;
        for(int i = 0; i < n; i++){
            do{ // on tire un indice au hasard
                j = (int)(Math.random() * n);
            } while(utilisee[j]);
            utilisee[j] = true;
            t[i] = j;
        }
    }
}

```

Un algorithme moins coûteux: on tire au sort un indice $j \in [0..n[$ et on permute $t[n-1]$ et $t[j]$; on recommence pour $t[0..n-1[$, etc.

```
public static void battre(int[] t){
    int n = t.length, i, j, tmp;

    for(i = n-1; i > 0; i--){
        // on choisit un entier j de [0..i]
        j = (int)(Math.random() * (i+1));
        // on permute t[i] et t[j]
        tmp = t[i]; t[i] = t[j]; t[j] = tmp;
    }
}
```

Admis: cela génère les permutations de manière uniforme.

```
public static int jouerAB(int[] jeuA, int[] jeuB){
    int gainA = 0;

    for(int i = jeuA.length-1; i >= 0; i--){
        if(jeuA[i] > jeuB[i])
            gainA++;
        else
            gainA--;
    }
    return gainA;
}
```

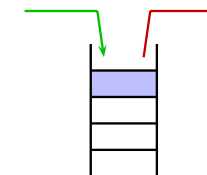
```
// donne[] contient les cartes qu'on
// distribue à partir de la fin
// on remplit jeuA et jeuB à partir de 0
public static void distribuer(int[] jeuA,
                              int[] jeuB,
                              int[] donne){
    int iA = 0, iB = 0;

    for(int i = donne.length-1; i >= 0; i--){
        if((i % 2) == 0)
            jeuA[iA++] = donne[i];
        else
            jeuB[iB++] = donne[i];
    }
}
```

Pile

Last In, First Out (LIFO)

Objet fondamental de l'informatique, ici utilisé pour stocker des cartes; premier élément arrivé se trouve dans le fond. Le dernier arrivé peut sortir.



On l'avait déjà utilisé implicitement pour expliquer les appels de fonctions.

Ex. tube d'aspirine.

IV. Un peu de chaînes de caractères

Le type `char` : les constantes s'écrivent `'a'`, `'A'`, `'0'`, `'+'`, Il existe des caractères spéciaux :

`''`, `'\n'`, `'\r'`, `'\t'`, `'\'`.

- **Unicode**: codage des caractères sur 16 bits (vs. 8 bits en ASCII) qui permet le codage de tous les types d'alphabet.
- Les caractères utiles pour le français sont de code compris entre `\u0020` (pour espace) et `\u00FC` (pour ü), le tout en hexadécimal.

Les chaînes de caractères : `String` est une classe, ce n'est pas un type primitif.

`String s ≈ char[] s`

- `s.length()` retourne la longueur de `s` (au lieu de `t.length` pour un tableau);
- `s.charAt(i)` retourne le *i*-ième caractère de `s` (compté à partir de 0);
- `s.equals(t)`: retourne `true` si `s` et `t` contiennent les mêmes caractères; plus précis `s.compareTo(t)`;
- **Immutable**: on ne peut pas écrire

```
String s = "Bonjour!";
s.charAt(0) = 'S';
```

```
Bug.java:4: unexpected type
required: variable
found   : value
        s.charAt(0) = 'S';
                ^
```

1 error

Comment créer une `String`:

- `String s = "Bonjour!";`
- par concaténation: `s = "Bonjour" + "!";`
- par conversion: `s = String.valueOf(i);`
- `...main(String[] args)` : `args` contient les arguments passés sur la ligne de commande. Par exemple:

```
public static void main(String[] args){
    System.out.println(args[0]+" "+args[1]);
}
```

affiche les deux premiers arguments. De même:

```
int n = Integer.parseInt(args[0]);
String s = args[1];
```

permet de récupérer un entier passé en argument sur la ligne de commande Unix:

```
java Essai 10 oui
```

V. Entrées-sorties (suite)

Rappel: redirection dans un fichier

```
public class Essai{
    public static void main(String[] args){
        for(int i = 0; i < 100; i++)
            System.out.println(i);
    }
}
```

```
unix% java Essai > fic
```

Les messages affichés sont mis dans le fichier `fic`.

On utilise `TC.lireInt()` ou bien dans ce cas particulier:

```
public class Relire{
    public static void main(String[] args){
        int[] t = TC.intDeFichier(args[0]);

        for(int i = 0; i < t.length; i++)
            System.out.println(t[i]);
    }
}
```

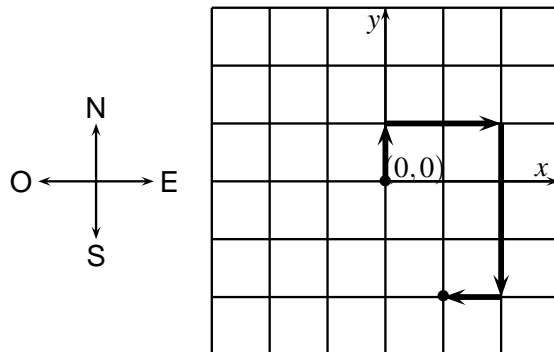
Pour relire, on tape:

```
unix% java Relire fic
```

Il existe des variantes, cf. `TC.java`.

Chasse au trésor (d'après le vrai ACM)

Présentation: Une carte au trésor donne des indications permettant de trouver un trésor à partir d'un point de départ initial et d'une suite de déplacements. Pour se simplifier la vie, on cherche à écrire un programme qui calcule tout seul les coordonnées du trésor. On modélise le problème de la façon suivante. On se place sur une grille à coordonnées entières et le point de départ des indications est toujours le point $(0,0)$.



- Énoncé simple;
- entrées et sorties complètement spécifiées;
- un programme à écrire et soumettre en trois fois;
- les programmes à écrire tiennent sur [une page d'écran](#).

Cf. les exercices des années précédentes.

Les indications pour trouver le trésor comprennent deux types d'information : une direction (**N**ord, **S**ud, **E**st, **O**uest) suivie d'un entier indiquant le nombre de pas à effectuer dans la direction en question. Par exemple, les indications

```
N 1
E 2
S 3
O 1
```

conduisent au point de coordonnées $(1, -2)$ comme indiqué sur le dessin.

Le format en entrée: la première ligne contient le nombre de cartes `nc`. Chaque carte commence par un entier contenant le nombre d'instructions `ni`. Suivent alors `ni` lignes dont le premier champ est la direction, codé sous forme d'un caractère parmi N, E, S et O, suivi d'un espace, suivi d'un entier.

Par exemple, `inputa` contient:

```
2
4
N 1
E 2
S 3
O 1
1
O 1
```

```
class Tresor{
    public static void main(String[] args){
        int nc = TC.lireInt();
        for(int c = 1; c <= nc; c++){
            System.out.println("Carte "+c);
            int ni = TC.lireInt();
            int x = 0, y = 0;
            for(int i = 0; i < ni; i++){
                String ligne = TC.lireLigne();
                String[] mots = TC.motsDeChaine(ligne);
                int d = Integer.parseInt(mots[1]);
                if(mots[0].equals("N"))
                    y += d;
                else if(mots[0].equals("S"))
                    y -= d;
                else if(mots[0].equals("E"))
                    x += d;
                else if(mots[0].equals("O"))
                    x -= d;
            }
            System.out.println(x+" "+y);
        }
    }
}
```

L'exercice: on demande d'écrire une classe `Tresor`

```
public class Tresor{
    public static void main(String[] args){
    }
}
```

qui prend en entrée un fichier contenant un certain nombre de cartes. Le programme doit afficher le numéro de la carte sur une ligne, suivie d'une ligne contenant les coordonnées du trésor, c'est-à-dire l'abscisse et l'ordonnée du trésor séparés par un espace. Pour l'exemple précédent, la sortie serait

```
Carte 1
1 -2
Carte 2
-1 0
```

Les commandes à utiliser sont:

```
java Tresor < inputa > res
diff outputa res
```

Derniers mots

Cours:

- tableaux;
- chaînes de caractères;
- entrées-sorties.

Prochains rendez-vous:

Groupes	TD 4
1-6	15h45-17h45
7-12	13h30-15h30

Prochain amphi: lundi 22 mai à 10h30.

Tutorat: jeudi 18 mai, [inscriptions avant ce soir.](#)