Introduction à la Programmation 1 JAVA

51AE011F

Séance 5 de cours/TD

Université Paris-Diderot

Objectifs:

- Faire un bilan après cc, questions/réponses.
- Comprendre les tableaux de chaînes de caractères.
- Comprendre les tableaux de tableaux.
- Comprendre le statut particulier des tableaux en mémoire.

1 Tableaux d'autres types

Des tableaux de chaînes de caractères [COURS]

- Rappel : Si `T` est un type alors « `T[]` » est le type des tableaux dont le contenu des cases est de type `T`.
- Un tableau de chaînes de caractères aura pour type « `String[]` ».
- Par exemple, pour créer un tableau de chaînes de caractères `theBeatles` qui vaut initialement `{ "Paul", "John", "Ringo","Georges" }`, on peut procéder ainsi :

```
String[] theBeatles = { "Paul", "John", "Ringo", "Georges" };
```

ou bien ainsi :

```
String[] theBeatles = new String[4];
theBeatles[0] = "Paul";
theBeatles[1] = "John";
theBeatles[2] = "Ringo";
theBeatles[3] = "Georges";
```

Exercice 1 (Fonctions et tableaux de chaînes de caractères, ⋆)

1. *Que fait la fonction de signature «*`String[] funcAB (int a)` *» fournie ci-dessous en supposant que l'entier donné en paramètre est toujours strictement positif ?*

```
public static String[] funcAB (int a) {
    String[] t = new String[a];
    String s = "ab";
    for (int i = 0; i < a; i++) {
        t[i] = s;
        s = s + "ab";
    }
    return t;
}
```

2. *Utilisez la fonction précédente dans une suite d'instructions pour afficher 5 lignes de la forme suivante :*

```
ab
abab
ababab
abababab
ababababab
```

□

Exercice 2 (Tableaux de prénoms, ⋆⋆)
Écrire une fonction prenant en paramètre un tableau de prénoms `t` *et qui renvoie l'indice d'un prénom de* `t` *qui y apparaît deux fois. Si une telle valeur n'existe pas, la fonction renvoie* -1*. On pourra pour cela se servir de la fonction «* `boolean stringEquals(String st1,String st2)` *» qui permet de tester si deux chaînes de caractères sont égales.* □

Des tableaux de tableaux [Cours]

— Les tableaux sont des valeurs comme les autres. Un tableau peut donc aussi contenir un tableau dans chacune de ses cases. On parle alors de tableau de tableaux.
— Un tableau de tableaux d'entiers a pour type « `int[][]` ».
— Par exemple, un tableau de tableaux d'entiers peut valoir « `{ { 1 }, { 11, 22 }, { 111, 222, 333 } }` ».
À la première case de ce tableau, on trouve le tableau « `{ 1 }` » , puis à la deuxième case le tableau « `{ 11, 22 }` » et à la dernière case le tableau « `{ 111, 222, 333 }` ».
— Pour créer et initialiser un tableau de tableaux, il faut créer et initialiser le tableau "contenant", et les tableaux "contenus". Il y a différentes façons de procéder :

1. On peut créer et initialiser tous les tableaux au moment de la déclaration du tableau "contenant" :

```
int[][] t = { { 1, 2 }, { 11, 22 }, { 111, 222 } };
```

2. On peut créer tous les tableaux sans les initialiser :

```
int[][] t = new int[3][5];
```

crée un tableau de 3 cases, chaque case contenant un tableau de 5 cases. Remarquez qu'avec cette forme de création, tous les tableaux contenus dans le premier tableau ont la même taille (ici 5).
Pour initialiser les cases des tableaux "contenus", on peut utiliser des instructions de modification du contenu des cases en précisant leurs indices :

```
t[0][2] = 15;
```

met la valeur 15 dans la troisième case du premier tableau.

3. On peut créer le tableau "contenant" uniquement et créer les tableaux "contenus" par la suite :

```
int[][] t = new int[5][];
```

crée un tableau de 5 cases devant contenir des tableaux n'existant pas encore. On peut ainsi affecter des tableaux de taille différente à chaque case.
La déclaration suivante crée le tableau « `tPascal` » valant
« `{ { 1 }, { 1, 1 }, { 1, 2, 1 }, { 1, 3, 3, 1 } }` » :

```java
int[][] tPascal = new int[4][];
tPascal[0] = new int[1];
tPascal[0][0] = 1;
tPascal[1] = new int[2];
tPascal[1][0] = 1;
tPascal[1][1] = 1;
tPascal[2] = new int[3];
tPascal[2][0] = 1;
tPascal[2][1] = 2;
tPascal[2][2] = 1;
tPascal[3] = new int[4];
tPascal[3][0] = 1;
tPascal[3][1] = 3;
tPascal[3][2] = 3;
tPascal[3][3] = 1;
```

Attention : On ne peut pas écrire « `tPascal[0] = { 1 };` » car l'opération de création et d'initialisation simultanée d'un tableau ne peut être faite qu'au moment de la déclaration.

Exercice 3 (Table de multiplication, $\star$)

1. *Créer un tableau à deux dimensions tel que* `t[i][j]` *vaut* `(i + 1) * (j + 1)` *pour* i *et* j *allant de* 0 *à* 9.
2. *Où se trouve le résultat de la multiplication de* 3 *par* 4 *dans ce tableau ? Même question pour le résultat de la multiplication de* 7 *par* 6.

□

Exercice 4 (Carré magique, $\star\star$)

Un carré magique est une grille carrée dans laquelle des nombres sont placés de telle sorte que la somme des nombres de chaque colonne, chaque ligne et de chacune des deux diagonales soit la même. De plus, le carré doit contenir une fois chaque nombre, de 1 au nombre de cases de la grille. La grille peut être représentée comme un tableau bi-dimensionnel d'entiers. Notre objectif est d'écrire une fonction qui vérifie si une grille de nombres reçu comme paramètre est un carré magique.

1. *Écrire une fonction* `carré` *qui prend* m*, un tableau de tableaux d'entiers, en argument et qui vérifie que* m *représente bien une grille carrée.*
2. *Écrire une fonction* `aplatir` *qui prend en paramètre* m*, un tableau de tableaux d'entiers qu'on peut supposer d'être une grille carrée, et qui envoie un tableau d'entiers qui contient tous les entiers éléments de* m*. Par exemple, appliquée à* $\{\{1,2,3\},\{4,5,6\},\{7,8,9\}\}$*, la fonction doit envoyer le résultat* $\{1,2,3,4,5,6,7,8,9\}$.
3. *Écrire une fonction* `domaine` *qui prend* t*, un tableau d'entiers en arguments, et qui envoie le booléen* `true` *si tous les éléments de* t *sont des valeurs entre* 0 *et la longueur du tableau inclus, et* `false` *sinon.*
4. *Écrire une fonction* `différents` *qui prend* t*, un tableau d'entiers en arguments, et qui envoie le booléen* `true` *si tous les éléments de* t *sont des valeurs différentes, et* `false` *sinon.*
5. *Enfin, écrire une fonction* `magique` *qui prend* t*, un tableau de tableaux d'entiers, en argument et qui renvoie* `true` *si* t *représente un carré magique, et* `false` *sinon. Utilisez les fonctions demandées aux questions précédentes.*

□

Exercice 5 (Gomoku, $\star\star\star$)

Le Gomoku est un jeu de plateau à deux joueurs, dans lequel pour gagner, chaque joueur doit réussir à aligner

5 pions sur des cases consécutives d'un plateau, horizontalement, verticalement ou en diagonale. Le plateau est une grille carrée, de dimension quelconque, et il peut être représenté comme un tableau bi-dimensionnel d'entiers. L'entier vaut 0 si la case est vide, 1 si elle contient un pion du joueur 1, et 2 pour un pion du joueur 2. Ecrivez une fonction qui reçoit comme paramètre le contenu d'une partie de Gomoku et détermine si l'un des joueurs a gagné. □

2 Le statut particulier des tableaux en mémoire

Mémoire et tableau [Cours]

- Dans la mémoire, on stocke également les tableaux et leur contenu.
- Si a est une variable de type tableau référant un tableau créé, alors dans la mémoire, la variable a sera associée à la valeur $\$i$ où i est un entier positif, appelée *son adresse dans le tas*. Le tas est une mémoire auxiliaire *qui est préservée par les appels de fonction et de procédure*.
- Dans notre modèle d'exécution des programmes, le contenu d'un tableau est représenté *à côté de la mémoire des variables* dans une autre mémoire. Cette seconde mémoire, le tas, associe des tableaux à des valeurs de la forme $\$i$.
- Par exemple :

$t1$	$t2$
$1	$2

$\$1 = \{2, 3, 4, 5\}$ $\$2 = \{-9, -3, 0\}$

  indique une mémoire où la variable `t1` réfère le tableau `{2, 3, 4, 5}` et la variable `t2` le tableau `{-9, -3, 0}`
- Nous noterons □ pour le contenu des cases non initialisées d'un tableau. Cela signifie que la case contient bien une valeur mais que cette valeur n'a pas été fixée par le programme et ne peut donc pas être exploitée de façon sûre.
- Ainsi après l'instruction « `int[] t = new int[4];` », la mémoire sera :

t
$1

$\$1 = \{□, □, □, □\}$

- Si une expression accède aux valeurs du tableau (par exemple dans « `a[2]` ») alors pour il faut aller regarder dans la mémoire quel est le tableau référé par la variable a, par exemple $3 et ensuite prendre la valeur qui se trouve dans la troisième case du tableau $3. Le même procédé s'applique pour les modifications du contenu des cases du tableau.
- Si un tableau $\$i$ n'est référencé par aucune variable dans la mémoire, alors on peut le supprimer.
- Comme le contenu du tas est préservé par les appels de fonctions et de procédures, on peut écrire des fonctions et des procédures qui modifient les tableaux passés en paramètres. C'est très pratique de ne pas avoir à recopier le contenu des tableaux à chaque appel de procédure car la taille des tableaux peut être importante.

Exercice 6 (⋆⋆, Échange du contenu de deux cases)
Soit la procédure suivante :

```
public static void swap (int[] a, int i, int j) {
  int tmp = a[i];
  a[i] = a[j];
  a[j] = tmp;
}
```

Donner l'évolution de la mémoire et du tas au cours de l'évaluation des instructions suivantes :

```
int[] t = { 3, 2, 1, 0 };
swap (t, 1, 2);
swap (t, 0, 3);
```

□

3 Fonctions utilisées

```java
/*Teste si deux chaînes de caractères st1 et st2 ont le même contenu.*/
public static boolean stringEquals (String st1, String st2) {
    return st1.equals (st2);
}
```

4 DIY

Exercice 7 (Comptage, ⋆⋆)
Écrire une fonction qui, étant donné un tableau `t` *de nombres entiers à 2 dimensions et un nombre entier* `n`*, renvoie :*

- *la valeur* `null` *si le tableau donné contient (au moins) un nombre* `x` *tel que* $x < 0$ *ou* $x > n$*,*
- *un tableau* `tt` *de* $n + 1$ *entiers tel que* `tt[i]` *soit égal au nombre d'éléments de* `t` *égaux à* `i` *si le tableau ne contient que de nombres compris, au sens large, entre* `0` *et* `n`*.*

□

Exercice 8 (Suites d'entiers, ⋆⋆)
Toute suite finie d'entiers peut être décomposée de manière unique en une suite de séquences strictement croissantes maximales. En représentant une suite dans un tableau `t`*, le tableau*

$$[1\ 2\ 5\ 7\ 2\ 6\ 0\ 5\ 2\ 4\ 6\ 7\ 8\ 9\ 3\ 4\ 6\ 1\ 2\ 7\ 8\ 9\ 4\ 2\ 3\ 1\ 5\ 9\ 7\ 1\ 6\ 6\ 3]$$

se décompose ainsi en le tableau de 13 tableaux d'entiers suivant :

$$[\ [1\ 2\ 5\ 7]\ [2\ 6]\ [0\ 5]\ [2\ 4\ 6\ 7\ 8\ 9]\ [3\ 4\ 6]\ [1\ 2\ 7\ 8\ 9]\ [4]\ [2\ 3]\ [1\ 5\ 9]\ [7]\ [1\ 6]\ [6]\ [3]\ ].$$

Les premiers éléments de ces séquences sont, en plus du premier élément du tableau, les éléments (soulignés sur l'exemple) qui sont inférieurs ou égaux à leur prédécesseur dans le tableau (`t[i]`$\leq$`t[i-1]`*).*

1. *Écrire une fonction* `rupture` *qui, étant donné un tableau* `t` *d'entiers, renvoie un tableau contenant 0 en premier élément et les indices des éléments de* `t` *inférieurs ou égaux à leur prédécesseur en ordre croissant. Pour le tableau donné en exemple, la fonction renvoie le tableau :*

$$[0\ 4\ 6\ 8\ 14\ 17\ 22\ 23\ 25\ 28\ 29\ 31\ 32].$$

2. *Écrire une fonction* `factorisation` *qui, étant donné un tableau* `t` *d'entiers, renvoie un tableau bidimensionnel d'entiers, dont la* `i`*-ème ligne contient la* `i`*-ème plus longue séquence croissante de nombres adjacents dans le tableau* `t` *(résultat tel que celui donné dans l'exemple). Indication : on pourra utiliser la fonction* `rupture` *pour déterminer le nombre de lignes et la taille de chaque ligne de ce tableau bidimensionnel.*

□

Exercice 9 (Matrices, ⋆⋆)
Étant donnée une matrice (tableau à deux dimensions) `A` *avec* `n` *lignes et* `m` *colonnes, un couple d'indices (*`i`*,*`j`*) représente un* min-max *de cette matrice si la valeur* `a[i][j]` *est un minimum de la ligne* `i` *et un maximum de la colonne* `j`*, c'est-à-dire*

$$\texttt{a[i][j]} = min\{\texttt{a[i][0]}, \ldots, \texttt{a[i][m]}\} \quad \texttt{a[i][j]} = max\{\texttt{a[0][j]}, \ldots, \texttt{a[n][j]}\}.$$

Ecrivez le programme qui affiche l'ensemble de tels couples (`i`,`j`*). La méthode proposée consiste à effectuer le travail suivant pour chaque ligne* `i` *: (1) trouver les minima de la ligne* `i` *et en mémoriser les numéros de colonne et (2) pour chacun de ces rangs* `j`*, déterminer si* `a[i][j]` *est un maximum pour sa colonne.* □

Exercice 10 (Championnat, ⋆⋆⋆)
On considère un tableau à 3 dimensions stockant les scores d'un championnat de handball. Pour `n` *équipes, le tableau aura* `n` *lignes et* `n` *colonnes et dans chacune de ses cases on trouvera un tableau à une dimension de longueur 2 contenant le score d'un match (on ne tient pas compte de ce qui est stocké dans la diagonale). Ainsi, pour le tableau championnat* `ch`*, on trouvera dans* `ch[i][j]` *le score du match de l'équipe* `i+1` *contre l'équipe* `j+1` *et dans* `ch[j][i]` *le score du match de l'équipe* `j+1` *contre l'équipe* `i+1`*. De même, pour un score stocké, le premier entier de* `ch[i][j]` *sera le nombre de but(s) marqué(s) par l'équipe* `i+1` *dans le match l'opposant à l'équipe* `j+1`*. Finalement, on suppose que lorsqu'une équipe gagne un match, elle obtient 3 points, 1 seul point pour un match nul et 0 point dans le cas où elle perd le match.*

1. *Écrire une méthode* `nombrePoints` *qui prend en arguments un tableau championnat* `ch` *de côté* `n` *et le numéro d'une équipe (entre 1 à* `n`*) et qui renvoie le nombre de point(s) obtenu(s) par cette équipe pendant le championnat.*
2. *Écrire une méthode* `stockerScore` *qui prend en arguments un tableau championnat* `ch` *de côté* `n`*, le numéro* `i` *d'une équipe, le numéro* `j` *d'une autre équipe et le score du match de* `i` *contre* `j` *et qui met à jour le tableau* `ch`*.*
3. *Écrire une méthode champion qui prend en argument un tableau championnat* `ch` *et qui renvoie le numéro de l'équipe championne. Une équipe est championne si elle a strictement plus de points que toutes les autres. Si plusieurs équipes ont le même nombre de points, alors une équipe est meilleure si elle a marqué strictement plus de buts. Dans le cas d'égalité parfaite (même nombre maximum de points et même nombre maximum de buts marqués), la méthode renverra 0 pour signaler l'impossibilité de désigner un champion.*

□

Exercice 11 (Maxima locaux, ⋆⋆)
Écrire une fonction qui prend en paramètre un tableau d'entiers à deux dimensions (rectangulaire) et calcule le **nombre** *d'entrées* **intérieures** *de la matrice dont tous les voisins sont strictement plus petits. Chaque entrée intérieure de la matrice a quatre voisins (à gauche, à droite, vers le haut, vers le bas). Par exemple, pour la matrice*

```
1 4 9 1 4
4 8 1 2 5
4 1 3 4 6
5 0 4 7 6
2 4 9 1 5
```

la méthode devrait renvoyer 2 *car il y a deux éléments de la matrice (*8 *et* 7*) qui ont uniquement des voisins plus petits.* □

Exercice 12 (Entrepôt, ⋆⋆⋆)

Dans un entrepôt (divisé en cases carrées et codé par un tableau `grille` *bidimensionnel d'entiers), un magasinier (dont la position* `pos` *sur la grille est codée par un tableau de deux coordonnées entières) doit ranger des caisses sur des cases cibles. Il peut se déplacer dans les quatre directions (chacune codée par un tableau de deux coordonnées dont l'une est nulle et l'autre est 1) et pousser (mais pas tirer) une seule caisse à la fois. Dans le tableau* `grille`*, une case cible est indiquée par -1, une case libre par 0, une case cible contenant une caisse par 2, une case contenant une caisse par 3 et un mur (case infranchissable) par 4.*

1. *Écrire une fonction* `sontCorrectes` *qui prend en arguments une grille et une position et teste si l'entrepôt codé est rectangulaire, contient uniquement des cases libres, des caisses, des cases cibles et des cases infranchissables, contient autant de caisses (sur des cases non-cibles) et que de cases cibles (sans caisse) et si le magasinier se trouve à l'intérieur de l'entrepôt ailleurs que sur une caisse ou un mur.*
2. *Écrire une fonction string qui prend en arguments une grille et une position (supposées correctes) et renvoie la chaîne de caractères représentant l'entrepôt avec* `'u'` *pour une case cible,* `'.'` *pour une case libre,* `'o'` *pour une case cible contenant une caisse,* `'n'` *pour une case contenant une caisse,* `'x'` *pour un mur,* `'A'` *pour le magasinier sur une case libre et* `'B'` *pour le magasinier sur une case cible.*
3. *Écrire une fonction* `estComplete` *qui prend en argument une grille (supposée correcte) et teste si les caisses sont toutes rangées dans les cases cibles.*
4. *Écrire une fonction* `direction` *qui prend en argument un caractère et, s'il est* `'e'`*,* `'z'`*,* `'a'` *ou* `'s'`*, renvoie la direction associée (est, nord, ouest, sud) sous forme d'un tableau de deux coordonnées dont l'une est nulle et l'autre est +/- 1et renvoie* `null` *sinon.*
5. *Écrire une fonction* `coupPossible` *qui prend en arguments une grille, une position (supposées correctes) et un caractère (parmi* `'e'`*,* `'z'`*,* `'a'`*,* `'s'`*) et teste si le magasinier peut prendre la direction donnée ; auquel cas, la grille et la position seront mises à jour.*
6. *En déduire une fonction* `jouer` *qui met en œuvre ce jeu.*
7. *Proposer un (ou plusieurs) moyen(s) de prévenir certaines situations de blocage.*

```
$ java Sokoban
...xxx
ux.n.x
..nx..
.ou...
x.A.xx dir? a

...xxx
ux.n.x
..nx..
.ou...
xA..xx dir? z

...xxx
ux.n.x
.nnx..
.Bu...
x...xx dir? a

...xxx
ux.n.x
.nnx..
Auu...
x...xx dir? z

...xxx
ux.n.x
Annx..
.uu...
x...xx dir? z

...xxx
Bx.n.x
.nnx..
.uu...
x...xx dir? z

A..xxx
ux.n.x
.nnx..
.uu...
x...xx dir? e

.A.xxx
ux.n.x
.nnx..
.uu...
x...xx
    Interruption!
$
```

□

Introduction à la Programmation 1 JAVA

51AE011F

Séance 6 de cours/TD

Université Paris-Diderot

Objectifs:

- Boucles `while`.
- Variables booléennes.

1 La boucle “while”

Boucle non bornée [COURS]

La boucle non bornée permet de répéter des instructions tant qu'une expression booléenne est vraie. Elle est utile lorsqu'on ne connaît pas *a priori* le nombre d'itérations nécessaires.

```
while (expression booléenne) {
    instructions à répéter
}
```

- L'expression booléenne est appelée condition ou encore test d'arrêt de la boucle.
- Par exemple, la boucle suivante s'arrêtera quand la valeur de la variable entière `a` sera 0 après avoir affiché 50 lignes contenant la chaîne « `Hello` ».

```
int a = 50;
while (a > 0) {
    System.out.println("Hello");
    a = a - 1;
}
```

- Une boucle non bornée peut ne jamais terminer si sa condition est toujours vraie. La plupart du temps, la non terminaison du programme n'est pas le comportement escompté car on souhaite obtenir un résultat ![1]
- Par exemple, la boucle suivante ne termine jamais et l'instruction « `System.out.println ("Bonjour");` » n'est donc jamais exécutée :

```
int b = 0;
while (b == 0) {
    b = b / 2;
}
System.out.println ("Bonjour\n");
```

Exercice 1 (Première boucle, ⋆)

Quelle est la valeur de la variable `r` *à la fin de la suite des instructions suivantes ?*

```
int n = 49;
int r = 0;
while (r * r < n) {
    r = r + 1;
}
```

□

Exercice 2 (Les "for"s vus comme des "while"s, ⋆)

Réécrivez la suite d'instructions suivante pour obtenir le même comportement en utilisant un "`while`*" à la place du "*`for`*".*

```
int a = 0;
for (int i = 0; i < 32; i++) {
    a = a + i;
}
System.out.println (a);
```

□

Exercice 3 (Affichage maîtrisé, ⋆)

Écrire, à l'aide d'une boucle, un programme qui affiche la chaîne `"bla"` *plusieurs fois de suite autant que possible, sans dépasser les* 80 *caractères (longueur standard d'une ligne dans un terminal).*

□

Exercice 4 (Terminaison, ⋆)

Qu'affichent les deux séquences d'instructions suivantes ? Est-ce que leur exécution se termine ?

```
int n = 2;
while (n > 0) {
    System.out.println (n);
}
```

```
int n = 2;
while (n > 0) {
    n = n * 2;
    System.out.println(n);
}
```

□

2 Variables de type `boolean`

Présentation du type `boolean` [Cours]

— Comme toute valeur, une valeur de type `boolean` peut être stockée dans une variable.
— Rappel : Il y a deux valeurs pour le type `boolean` qui sont `true` et `false`.
— Par exemple, on peut déclarer et initialiser une variable booléenne ainsi :

```
boolean b = false;
```

ou bien encore comme cela :

```
boolean b = (x > 5);
```

Dans ce dernier cas, si la valeur contenue dans `x` est strictement plus grande que 5 alors l'expression booléenne « `x > 5` » s'évaluera en la valeur `true`, qui sera la valeur initiale de la variable `b`.
— Rappels : les opérateurs booléens sont `||` (**ou**), `&&` (**et**) et `!` (**non**).
— Par exemple, les déclarations suivantes sont valides :

```
boolean b1 = false; // b1 vaut false.
boolean b2 = (3 + 2 < 6); // b2 vaut true.
b1 = !b2; // b1 vaut la negation de true, donc false.
b2 = b1 || b2; // b2 vaut false || true, donc true.
```

— **Attention :** le symbole = est utilisée dans les instructions d'affectation tandis que le symbole == est un opérateur de test d'égalité entre entiers.
— Comme toute valeur, une valeur de type `boolean` peut être passée en paramètre à une fonction ou une procédure et renvoyée par une fonction.

Exercice 5 (Évaluer, ⋆)

Quelles sont les valeurs des variables `x`, `b`, `d`, `e` *et* `f` *après avoir exécuté les instructions suivantes ?*

```
int x = 3 + 5;
boolean b = (x == 5 + 3);
boolean d = (x > x + 1);
boolean e = b || d;
boolean f = b && d;
f = (e != b);
```

□

Exercice 6 (Parité, ⋆)

Écrire une fonction « `boolean isEven (int a)` *» qui prend en paramètre un entier et renvoie* `true` *si cet entier est pair,* `false` *sinon.* □

Exemples d'utilisation de variables booléennes [Cours]

— Les variables booléennes peuvent être utilisées pour signaler qu'une condition a été vérifiée, ce qui peut être utile pour arrêter une boucle.
— On peut aussi utiliser les variables booléennes comme paramètres de fonction pour faire varier le comportement selon des conditions.
— Les variables booléennes sont parfois appelées drapeaux (*flag* en anglais).

Exercice 7 (Drapeau et recherche, ⋆)

1. *On considère la suite d'instructions suivante qui parcourt un tableau* `t` *et affecte* `true` *à la variable* `found` *si une des cases de* `t` *vaut* 3 *:*

```
int[] t = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
int i = 0;
boolean found = false;  // flag
```

```
while (i < intArrayLength (t)) {
    if (t[i] == 3) {
        found = true;
    }
    i = i + 1;
}
```

Quelle est la valeur contenue dans la variable `i` *après ces instructions ?*

2. *Même question sur la suite d'instructions suivante :*

```
int[] t = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
int i = 0;
boolean found = false;  // flag
while (i < intArrayLength (t) && !found) {
    if (t[i] == 3) {
        found = true;
    }
    i = i + 1;
}
```

3. *Qu'en déduisez-vous ?*
4. *Écrire une fonction « `boolean findValueTab(int[] t, int v)` » qui teste si la valeur contenue dans* `v` *est présente dans une case du tableau d'entiers* `t`.

□

Exercice 8 (Paramètre booléen, ⋆)
Écrire une fonction qui énumère les entiers entre 1 *et* n. *La fonction prend deux paramètres : l'entier* n, *et un booléen* `up`. *Si le paramètre booléen est vrai, on compte de 1 à n, sinon on compte de* n *à* 1. □

3 Fonctions utilisées

```
/*Teste si deux chaînes de caractères st1 et st2 ont le même contenu.*/
public static boolean stringEquals (String st1, String st2) {
    return st1.equals (st2);
}
```

4 DIY

Exercice 9 (Comptine, ⋆)
Modifier les instructions suivantes pour que l'accord de kilomètre(s) soit correct.

```
int n = 10;
while (n > 0) {
    System.out.print (n);
    System.out.println (" kilometre a pied ca use les souliers");
    n = n - 1;
}
```

□

Exercice 10 (Palindrome, ⋆⋆)
Un mot est un palindrome si on obtient le même mot en le lisant à l'envers. Par exemple "kayak", "ressasser", ou "laval" sont des palindromes. Écrire une fonction qui renvoie le booléen `true` *si le mot* `s` *est un palindrome et* `false` *sinon.* □

Exercice 11 (Logarithme, ⋆)
Écrire une fonction qui prend en paramètre un entier n, et renvoie le nombre de fois qu'il faut diviser celui-ci par deux avant que le résultat soit inférieur ou égal à 1. □

Exercice 12 (n-ème chiffre, ⋆⋆)
Écrire une fonction qui prend en paramètres deux entiers k et n et renvoie le n-ème chiffre de k, ou 0 si k n'est pas aussi long (par exemple le 2ème chiffre de 1789 est 8). □

Exercice 13 (Binaire, ⋆⋆⋆)
Écrire une fonction qui prend en paramètre un entier `n` *et qui renvoie la représentation binaire de* `n` *sous la forme d'une chaîne de caractères formée de caractères* 0 *et* 1*.* □

Exercice 14 (lastOcc, ⋆⋆)
Écrire une fonction « `int lastOcc(int[] tab, int x)` *» qui prend en paramètre un tableau et une valeur entière. Si le tableau contient cette valeur, la fonction renvoie l'indice de la dernière occurrence de la valeur. Si le tableau ne contient pas cette valeur, la fonction renvoie* -1*.*

Cet exercice a déjà été traité au TD 4, mais on demande de remplacer la boucle `for` *par une boucle* `while`*.* □

Exercice 15 (Syracuse, ⋆⋆⋆)
La suite de Syracuse de premier terme p (entier strictement positif) est définie par $a_0 = p$ et

$$a_{n+1} = \begin{cases} \frac{a_n}{2} & \text{si } a_n \text{ est pair} \\ 3 \cdot a_n + 1 & \text{si } a_n \text{ est impair} \end{cases} \quad \text{pour } n \geq 0$$

Une conjecture (jamais prouvée à ce jour !) est que toute suite de Syracuse contient un terme $a_m = 1$. Trouver le premier m pour lequel cela arrive, étant donné p. □

Exercice 16 (Itérations, ⋆)
Combien y a-t-il d'itérations dans la boucle suivante :

```
int n = 15;
while (n >= 0) {
    n = n - 1;
}
```

Même question pour la boucle suivante :

```
static void nbIterations (int n) {
    while (n >= 0) {
        n = n - 1;
    }
}
```

□

Exercice 17 (Logarithme (itéré), ⋆⋆)

1. *Le logarithme en base 2 d'un entier $n \geq 1$ (noté $\log_2 n$) est le réel x tel que $2^x = n$. On se propose de calculer la partie entière de $\log_2 n$, c'est-à-dire le plus grand entier m tel que $2^m \leq n$. On notera l la partie entière du logarithme en base 2, c'est-à-dire que $m = l(n)$. Par exemple, $l(8) = l(9) = 3$ car $2^3 = 8 \leq 9 < 2^4$.*
   Écrire une fonction `l` *qui prend en paramètre un entier n et renvoie la partie entière $l(n)$ de son logarithme en base 2. On calculera $l(n)$ en effectuant des divisions successives par 2.*

2. *À partir de la fonction l précédente, on peut définir une fonction l^* ainsi : $l^*(n)$ est le plus petit entier i tel que la i-ème itération de l sur l'entrée n vaille 0. Par exemple, $l^*(1) = 1$ car dès la première itération, $l(1) = 0$; ou encore $l^*(2) = 2$ car $l(2) = 1$ et $l(1) = 0$. Pour prendre un exemple plus grand, on a $l^*(1500) = 4$ car $l(1500) = 10$, $l(10) = 3$, $l(3) = 1$ et $l(1) = 0$: la quatrième itération de l sur 1500 vaut 0 (en d'autres termes, $l \circ l \circ l \circ l(1500) = 0$).*

   Écrire une fonction `lstar` *qui prend en paramètre un entier n et renvoie $l^*(n)$.*

□

Exercice 18 (Racine cubique, ⋆⋆)
Écrire une fonction qui renvoie la racine cubique d'un entier x, c'est-à-dire le plus petit entier a tel que $a^3 \geq x$. On peut s'inspirer de l'exercice 1. □

Exercice 19 (Racine $n$ième, ⋆⋆⋆)
Maintenant, écrire une fonction qui renvoie la racine $n$ième d'un entier x, c'est-à-dire le plus petit entier a tel que $a^n \geq x$. □

Exercice 20 (Ecrire du Proust à l'écran, ⋆⋆⋆)
Écrire une fonction qui prend en paramètre une chaîne de caractères contenant une phrase de Proust et l'affiche à l'écran. La phrase peut être longue, et sur une ligne on ne peut afficher que 80 caractères, il faut donc revenir à la ligne quand le prochain mot à afficher ne peut tenir sur la ligne courante. Les mots sont délimités par les caractères espace.

□

Introduction à la Programmation 1 JAVA

51AE011F

Séance 7 de cours/TD

Université Paris-Diderot

Objectifs:

— Connaître ses classiques sur les tableaux.

— Apprendre à voir quand un programme est faux.

1 Devenir incollable sur les tableaux

Opérations à savoir faire sur les tableaux [COURS]

- **Rechercher**
  - Dire si un élément est présent dans un tableau
  - Renvoyer la position d'un élément dans le tableau (la première ou la dernière)
  - Compter le nombre de fois qu'un élément est présent dans un tableau
  - Savoir construire un tableau de positions d'un élément dans un tableau
- **Modifier**
  - Savoir échanger deux éléments de place dans un tableau
  - Savoir renverser un tableau
  - Savoir décaler tous les éléments d'un tableau
  - Savoir appliquer une fonction à tous les éléments d'un tableau

Exercice 1 (Présence dans un tableau, ⋆)
On considère la fonction `public static boolean isPresent (int el, int [] tab)` *qui renvoie* `true` *si* `el` *est dans le tableau* `tab`.

```java
public static boolean isPresent(int el, int[] tab){
    for(int i=0; i<tab.length; i=i+1){
        if(tab[i]==el){
            return true;
        } else{
            return false;
        }
    }
    return false;
}

public static boolean isPresent(int el, int[] tab){
    boolean b = true;
    for(int i=0; i<tab.length; i=i+1){
        if(tab[i]==el){
            b = true;
```

```
17          } else{
18              b = false;
19          }
20      }
21      return b;
22  }
23
24  public static boolean isPresent(int el, int[] tab){
25      for(int i=0; i<tab.length;i=i+1){
26          return (tab[i]==el);
27      }
28  }
29
30  public static boolean isPresent(int el, int[] tab){
31      boolean b = false;
32      int i = 0;
33      while (i<tab.length && b){
34          if(tab[i]==el){
35              b = true;
36          }
37      }
38      return b;
39  }
40
41  public static boolean isPresent(int el, int[] tab){
42      boolean b = false;
43      int i = 0;
44      while (i<tab.length && !b){
45          b = b && (tab[i]==el);
46      }
47      return b;
48  }
```

code/ExoPresent.java

1. *Pourquoi les propositions précédentes sont-elles fausses ? À chaque cas, donnez un exemple de tableaux d'entiers et d'éléments pour laquelle la fonction renvoie un mauvais résultat.*
2. *Proposez une correction pour chacun des cas.*

□

Exercice 2 (Occurences dans un tableau, ⋆)
On considère la fonction `public static int occ (int el,int[] tab)` *qui renvoie le premier indice de* `el` *dans le tableau* `tab` *et* `-1` *si* `el` *n'est pas dans le tableau.*

```
1   public static int occ(int el,int[] tab){
2       for(int i=0; i<tab.length; i=i+1){
3           if(el==tab[i]){
4               return i;
5           } else {
6               return -1;
7           }
8       }
9       return -1;
10  }
11
12  public static int occ(int el,int[] tab){
13      for(int i=0; i<tab.length; i=i+1){
```

```java
        if(el==tab[i]){
            return i;
        }
    }
    return i;
}

public static int occ(int el,int[] tab){
    int p = 0;
    for(int i=0; i<tab.length; i=i+1){
        if(el==tab[i]){
            p = i;
        } else {
            p = -1;
        }
    }
    return p;
}

public static int occ(int el,int[] tab){
    int p = 0;
    for(int i=0; i<tab.length; i=i+1){
        if(el==tab[i]){
            p = i;
        }
    }
    return p;
}

public static int occ(int el,int[] tab){
    int p = -1;
    int i = 0;
    while(i<tab.length){
        if(el==tab[i]){
            p = i;
            i = i+1;
        }
    }
    return p;
}

public static int occ(int el,int[] tab){
    int p = -1;
    int i = 0;
    while(i<tab.length && p!=-1){
        if(el==tab[i]){
            p = i;
        }
        i = i+1;
    }
    return p;
}
```

code/ExoOccurence.java

.

1. *Pourquoi les propositions précédentes sont-elles fausses ? À chaque cas, donnez un exemple de tableaux*

d'entiers et d'éléments pour laquelle la fonction renvoie un mauvais résultat.

2. *Proposez une correction pour chacun des cas.*

□

Exercice 3 (Compter dans un tableau, ⋆)
On considère la fonction `public static int count (int el, int[] tab)` *qui renvoie le nombre de fois que l'élément* `el` *apparaît dans le tableau* `tab`.

```java
public static int count(int el,int[] tab){
    int c = 0;
    for(int i=0; i<tab.length; i=i+1){
        if(el==tab[i]){
            c = c+1;
            return c;
        }
    }
    return c;
}

public static int count(int el,int[] tab){
    int c = 0;
    for(int i=0; i<tab.length; i=i+1){
        if(el==tab[i]){
            c = c+1;
        } else {
            c = c-1;
        }
    }
    return c;
}
```

code/ExoCounting.java

.

1. *Pourquoi les propositions précédentes sont-elles fausses ? À chaque cas, donnez un exemple de tableaux d'entiers et d'éléments pour laquelle la fonction renvoie un mauvais résultat.*
2. *Proposez une correction pour chacun des cas.*

□

Exercice 4 (Construire le tableau des positions, ⋆)
On considère la fonction `public static int[] pos (int el, int[] tab)` *qui renvoie un tableau contenant les positions où* `el` *apparaît dans le tableau* `tab`.

1. *Que valent les variables* `tab1`, `tab2` *et* `tab3` *après exécution du programme suivant (en supposant que la fonction* `pos` *est correctement définie).*

```java
int[] tab1 = pos (2, {3, 4, 5, 8});
int[] tab2 = pos (3, {3, 3, 4, 3, 4, 5, 6, 4, 3});
int[] tab3 = pos (4, {3, 3, 4, 3, 4, 5, 6, 4, 3});
```

2. *On considère les propositions suivantes pour la fonction* `public static int[] pos (int el, int [] li)` *(on suppose que la fonction* `count` *est celle de l'exercice précédent et qu'elle est correcte).*

```java
public static int[] pos(int el,int[] tab){
    int []tabRet={};
    for(int i=0; i<tab.length; i=i+1){
        if(el==tab[i]){
            tabRet = {i};
```

```java
            }
        }
        return tabRet;
}

public static int[] pos(int el,int[] tab){
        int n = count(el, tab);
        int []tabRet = new int[n];
        for(int i=0; i<n; i=i+1){
            tabRet[i] = 0;
        }
        for(int i=0; i<tab.length; i=i+1){
            if(el==tab[i]){
                tabRet[i] = i;
            }
        }
        return tabRet;
}

public static int[] pos(int el,int[] tab){
        int n = count(el, tab);
        int []tabRet = new int[n];
        for(int i=0; i<n; i=i+1){
            tabRet[i] = 0;
        }
        for(int i=0; i<tab.length; i=i+1){
            if(el==tab[i]){
                tabRet[el] = i;
            }
        }
        return tabRet;
}

public static int[] pos(int el,int[] tab){
        int n = count(el, tab);
        int []tabRet = new int[n];
        for(int i=0; i<n; i=i+1){
            tabRet[i] = 0;
        }
        int j = 0;
        for(int i=0; i<tab.length; i=i+1){
            if(el==tab[i]){
                tabRet[j] = i;
            }
        }
        return tabRet;
}
```

code/ExoPos.java

a Pourquoi les propositions précédentes sont-elles fausses ? À chaque cas, donnez un exemple de tableaux d'entiers et d'éléments pour laquelle la fonction renvoie un mauvais résultat.

b Proposez une correction.

□

Exercice 5 (Échanger deux éléments dans un tableau, ⋆)

Dans cet exercice, on considère la fonction `public static void swap (int i ,int j, int[] tab)` *qui*

échange les valeurs des indices `i` *et* `j` *du tableau* `tab`.

1. *Pour comprendre ce qui se passe tout en s'échauffant dîtes ce qu'affiche le programme suivant :*

```java
public class ExoSwap1{

    public static void modif(int[] t){
        t[0] = 5;
    }

    public static void main(String[] args){
        int[] tab = {1, 2, 4, 8, 16, 31};
        modif(tab);
        for(int i=0; i< tab.length; i=i+1){
            System.out.println(tab[i]);
        }
    }
}
```

code/ExoSwap1.java

2. *En déduire ce que doit retourner la fonction* `swap`.
3. *Pourquoi la définition suivante de* `swap` *n'est pas correcte ? Proposez une correction.*

```java
public static void swap(int i, int j, int[] tab){
    if(i<tab.length && j<tab.length){
        tab[i] = tab[j];
        tab[j] = tab[i];
    }
}
```

code/ExoSwap2.java

□

Exercice 6 (Inverser un tableau d'éléments, ⋆)

Dans cet exercice, on considère la fonction `public static int[] reverse (int[] tab)` *qui prend en paramètre un tableau* `tab` *et renvoie le tableau où l'ordre des valeurs est inversée.*

1. *Que renvoie donc* `reverse ({0, 1, 2, 3, 4, 5})`.

```java
public static int[] reverse(int[] tab){
    for(int i=0; i<tab.length;i++){
        tab[i] = tab[tab.length - 1 -i];
    }
    return tab;
}

public static int[] reverse(int[] tab){
    for(int i=0; i<tab.length;i++){
        int temp = tab[i];
        tab[i] = tab[tab.length - 1 -i];
        tab[tab.length - 1 -i] = temp;
    }
    return tab;
}

public static int[] reverse(int[] tab){
    int[] t = {};
    for(int i=0; i<tab.length;i++){
        t[i] = tab[tab.length - 1 -i];
    }
```

```
    return t;
}
```

code/ExoReverse.java

2. a *Pourquoi les propositions précédentes sont-elles fausses ? À chaque cas, donnez un exemple de tableaux d'entiers et d'éléments pour laquelle la fonction renvoie un mauvais résultat.*
   b *Proposez une correction.*
   c *Parmi les corrections proposées, quelles sont celles qui changent le tableau* `tab` *passée en paramètre et quelles sont celles qui le laissent inchangé ?*

□

Exercice 7 (Décaler les éléments d'un tableau, ⋆⋆)
Dans cet exercice, on considère la fonction `public static void shift (int[] li,int p)` *qui prend en paramètre un tableau* `tab` *et modifie ce tableau en décalant (de façon cyclique vers la droite) ses éléments de* `p` *positions (en supposant que* `p` *est positif).*

1. *Que valent les variables* `tab1`, `tab2` *et* `tab3` *après exécution du programme suivant (en supposant que la fonction* `shift` *est correctement définie).*

```
int [] tab1 = {3, 4, 5, 8};
shift (tab1, 1);
int[] tab2 = {3, 3, 4, 3, 4, 5};
shift (tab2, 2 );
int[] tab3 = {1, 2, 3, 4, 5, 6}
shift (li3, 7)
```

```
public static void shift(int[] tab, int p){
    for(int i=0; i<tab.length;i=i+1){
        tab[i] = tab[i-p];
    }
}

public static void shift(int[] tab, int p){
    for(int i=0; i<tab.length;i=i+1){
        int temp = tab[i];
        tab[i-p] = temp;
    }
}

public static void shift(int[] tab, int p){
    for(int i=0; i<tab.length;i=i+1){
        tab[(i+p) % tab.length] = tab[i];
    }
}
```

code/ExoShift.java

2. a *Pourquoi les propositions précédentes sont-elles fausses ? À chaque cas, donnez un exemple de tableaux d'entiers et d'éléments pour laquelle la fonction renvoie un mauvais résultat.*
   b *Proposez une solution. On pourra pour cela commencer par écrire une fonction* `shift1` *qui décale le tableau d'un élément vers la droite et utiliser ensuite cette fonction pour faire* `shift`.

□

2 DIY

Pour certains exercices, un "contrat" est proposé : ce sont des tests que nous vous fournissons pour vérifier votre réponse. Si votre réponse ne passe pas ces tests, il y a une erreur ; si votre réponse passe les tests, rien

n'est garanti, mais c'est vraisemblablement proche de la réponse. Quand aucun contrat n'est spécifié, à vous de trouver des tests à faire.

Exercice 8 (Primalité, ⋆⋆)
Un entier p est premier si $p \geq 2$ et s'il n'a pas d'autres diviseurs que 1 et lui-même.

1. *Écrire une fonction* `prime` *qui prend en paramètre un entier n et qui renvoie* `true` *si n est premier, ou* `false` *sinon.*
2. *Écrire une fonction* `next` *qui prend en entrée un entier x et qui renvoie le plus petit nombre premier $p \geq x$. On pourra bien sûr se servir de la fonction* `prime` *précédente.*
3. *Écrire une fonction* `number` *qui prend en entrée un entier y et qui renvoie le nombre de nombres premiers $p \leq y$. On pourra bien sûr se servir de la fonction* `prime`.

Contrat:
Pour la fonction `next` *:*

x = 2 → 2
x = 10 → 11
x = 20 → 23

Pour la fonction `number` *:*

x = 10 → 4
x = 20 → 8

□

Exercice 9 (Addition, ⋆⋆⋆)
Le but de cet exercice est de programmer l'addition décimale. Les deux nombres à additionner sont donnés sous forme de tableaux.

Par exemple, `x = { 7, 4, 3 }` *et* `y = { 1, 9 }`*, dont la somme doit être retournée sous forme de tableau, dans cet exemple,* `{7, 6, 2}`.

1. *Écrire une fonction* `add` *qui prend en paramètre deux tableaux et fait l'addition de gauche à droite des deux nombres représentés par les tableaux.*
   Par exemple, si les entrées sont les tableaux `t1 = { 3, 4, 7 }, t2 ={ 9, 1 }`*, cela représente la somme $743 + 19$. On calcule d'abord $9 + 3$ ce qui fait 2 avec une retenue de 1. Puis on calcule $4 + 1$, plus la retenue, ce qui donne 6, avec une retenue de 0. Enfin, le dernier chiffre est 7. À la fin, on doit renvoyer le tableau* `{ 2, 6, 7 }`.
2. *Écrire un fragment de code qui initialise deux tableaux* `t1` *et* `t2` *et fait l'addition des entiers qu'ils représentent.*

Contrat:

`t1 = {2,4,3,5}` `t2 = {4,3,6}` → *affichage :* `2 8 7 1`
`t1 = {6,1,2}` `t2 = {6,3,0}` → *affichage :* `1 2 4 2`

□

Exercice 10 (Multiplication de matrices, ⋆⋆)
Dans cet exercice, on suppose qu'une matrice A à m lignes et n colonnes est encodée par un tableau de tableaux contenant m tableaux de taille n. Par exemple, la matrice

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

est encodée par `{{1, 2, 3}, {4, 5, 6}}`.

Étant données deux matrices A (à m lignes et n colonnes) et B (à n lignes et p colonnes), le produit matriciel de A par B est une matrice C à m lignes et p colonnes telle que pour tout $1 \leq i \leq m$ et pour tout $1 \leq j \leq p$, le coefficient $C_{i,j}$ de C se trouvant à la i-ème ligne et la j-ième colonne est égal à :

$$C_{i,j} = \Sigma_{1 \leq k \leq n} A_{i,k} B_{k,j}$$

1. *Écrire une fonction* `isMatrix` *qui prend en paramètre un tableau de tableaux d'entiers et vérifie qu'il s'agit de l'encodage d'une matrice (c'est à dire que chaque sous-tableau à la même longueur) ; elle renvoie* `true` *dans ce cas et* `false` *sinon.*
2. *Écrire une fonction* `nbLines` *qui prend en paramètre un tableau de tableaux d'entiers qui encode une matrice et renvoie son nombre de lignes.*
3. *Écrire une fonction* `nbColumns` *qui prend en paramètre un tableau de tableaux d'entiers qui encode une matrice et renvoie son nombre de colonnes.*
4. *Écrire une fonction* `matriProx` *qui prend en paramètre deux tableaux de tableaux d'entiers* `A` *et* `B`*, vérifie qu'il s'agit de deux matrices, et vérifie que le nombre de colonnes de* `A` *est égal au nombre de lignes de* `B`*. Si ces conditions ne sont pas satisfaites, la fonction renvoie* `{}` *et sinon elle renvoie un tableau de tableaux contenant la matrice correspondant au produit matriciel de* `A` *par* `B`*.*

□

Exercice 11 (Pascal, ⋆⋆⋆)

Écrire un programme `ExoPascal` *qui affiche le triangle de Pascal jusqu'au rang n, passé en paramètre au moment du lancement du programme :*

```
java ExoPascal 4
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

La procédure `main` *doit faire appel à une fonction auxiliaire qui, étant donné n, renvoie le triangle de Pascal sous forme d'un tableau de tableaux d'entiers de n + 1 lignes, la i-ème ligne contenant les coefficients de la i-ème puissance du binôme $(a + b)$. On utilisera une fonction prédéfinie «* `int stringToInt(String s)` *» qui transforme une chaine de caractères* `"n"` *dans l'entier* `n` *qui contient.* □