Introduction à la Programmation 1

Séance 5 de cours/TD

Université Paris-Diderot

Objectifs:

- Bilan après cc, questions/réponses.
- Comprendre les tableaux de chaînes de caractères.
- Comprendre les tableaux de tableaux.
- Boucles while.
- Variables booléennes.

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";
```

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 au 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 } } » :

```
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][3] = 3;
tPascal[3][3] = 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 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
```

(On pourra utiliser la procédure « void printString (String s) ».)

□

Exercice 2 (Table de multiplication, ⋆)

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 3 (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.* □

2 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) {
  printString ("Hello\n");
  a = a - 1;
}
```

- Une boucle non bornée peut ne jamais terminée 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 « printString ("Bonjour\n"); » n'est donc jamais exécutée :

```
int b = 0;
while (b == 0) {
  b = b / 2;
}
```

```
printString ("Bonjour\n");
```

Exercice 4 (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 5 (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;
}
printInt (a);
```

□

Exercice 6 (Affichage maîtrisé, ⋆)

Écrire, à l'aide d'une boucle, un programme qui affiche plusieurs fois de suite la chaîne `"bla"` *et s'arrête quand il a affiché* 80 *caractères (longueur standard d'une ligne dans un terminal).*

□

Exercice 7 (Terminaison, ⋆)

Comment se comportent les deux séquences d'instructions suivantes ?

```
int n = 2;
while (n > 0) {
  printInt (n);
}
```

```
int n = 2;
while (n > 0){
  n = n * 2;
  printInt(n);
}
```

□

3 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 false, 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 argument à une fonction ou une procédure et renvoyée par une fonction.

Exercice 8 (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 9 (Parité, ⋆)
Écrire une fonction « boolean isEven (int a) *» qui prend en paramètre un entier et retourne* 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 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 10 (Drapeau et recherche, ⋆)

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

```
int[] t = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
int i = 0;
boolean flag = false;
while (i < intArrayLength (t)) {
  if (t[i] == 3) {
    flag = 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 flag = false;
while (i < intArrayLength (t) && !flag) {
  if (t[i] == 3) {
    flag = 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 11 (Paramètre booléen, ⋆)
Écrire une fonction qui compte jusqu'à n, où n est un entier donné en paramètre et qui prend également un booléen en paramètre. Si le paramètre booléen ascendant est vrai, on compte de 1 à n, sinon on compte en commençant à n, en descendant, jusqu'à 1. □

4 Fonctions utilisées

```
/* Affiche une chaîne de caractères. */
public static void printString(String s) {
  System.out.print(s);
}

/* Affiche un entier. */
public static void printInt(int x) {
   System.out.print (x);
}

/* Affiche un booléen. */
public static void printBoolean(boolean b) {
   System.out.print (b);
}

/* Retourne le nombre de cases d'un tableau d'entiers. */
public static int intArrayLength (int [] t) {
   return t.length;
```

```java
}

/* Retourne le nombre de cases d'un tableau de tableau d'entiers. */
public static int intArrayArrayLength (int[][] t) {
  return t.length;
}


/* Retourne la longueur d'une chaîne de caractères. */
public static int stringLength (String s) {
    return s.length ();
}

/*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);
}

/* Affiche le contenu d'un tableau de tableaux d'entiers */
public static void printIntArrayArray (int[][] t) {
  for (int i = 0; i < t.length; i++) {
    for (int j = 0; j < t[i].length; j++) {
      System.out.print (t[i][j] + " ");
    }
    System.out.println ();
  }
  System.out.println ();
}
```

5 DIY

Exercice 12 (Comptine, ⋆)
Modifier les instructions suivantes pour que l'accord de kilomètre(s) soit correct.

```java
  int n = 10;
  while (n > 0) {
    printInt (n);
    printString (" kilometre a pied ca use les souliers");
    n = n - 1;
  }
```

□

Exercice 13 (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 14 (Logarithme, ⋆)
Écrire une fonction qui prend en paramètre un entier n, et retourne le nombre de fois qu'il faut diviser celui-ci par deux avant que le résultat soit inférieur ou égal à 1. □

Exercice 15 (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 16 (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 17 (Horloge, ⋆⋆⋆)
Reprendre le compteur du td1 en remplaçant la boucle for *par une boucle* while. □

Exercice 18 (lastOcc, ⋆⋆)
Reprendre l'exercice lastOcc du TD 4 en remplaçant la boucle for *par une boucle* while. □

Exercice 19 (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 20 (Maxima locaux, ⋆⋆)
Écrire une fonction qui prend en argument 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 retourner 2 *car il y a deux éléments de la matrice (*8 *et* 7*) qui ont uniquement des voisins plus petits.* □

Exercice 21 (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 22 (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 23 (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 4. □

Exercice 24 (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 $a^n \geq x$. □