Introduction à la Programmation 1 JAVA

51AE011F

Séance 1 de cours/TD

Université Paris-Diderot

Objectifs:

- Utiliser JAVA comme une calculatrice.
- Identifier et donner un sens aux différentes constructions du langage (déclaration et utilisation des variables, boucles for, conditionnelles, procédures et fonctions).
- Apporter une modification mineure à un programme existant.

1 De quoi est fait un programme ?

Qu'est-ce qu'un programme ? [COURS]

- Un **programme** est la représentation d'une séquence d'instructions à exécuter. Par exemple, une recette est un programme exécuté par un cuisinier. Un code source JAVA est un programme exécuté par un ordinateur.

Qu'est-ce qu'un ordinateur ? [COURS]

- En première approximation, un **ordinateur** est une machine à calculer munie d'une mémoire et connectée à des périphériques (comme un écran, une carte réseau, etc).
- Un ordinateur peut soit **faire des calculs** qui produisent des **valeurs** ; soit **effectuer des actions** en transmettant des valeurs à sa mémoire ou à des périphériques.
- Les calculs peuvent dépendre de valeurs récoltées par les périphériques ou lues dans la mémoire.

Les constituants d'un programme [COURS]

- Les **expressions** décrivent des calculs à faire.
- Les **instructions** décrivent des actions à effectuer.
- Les **déclarations** donnent des noms aux constituants du programme.

```
1         int stopSecs = (stopHour * 60 + stopMin) * 60;
2         int startSecs = (startHour * 60 + startMin) * 60;
3         int numberOfSecs = stopSecs - startSecs;
4         int alertCode = 0;
5         for (int i = 0; i < numberOfSecs; i++) {
6             waitUntilNextSecond ();
7             if (numberOfSecs - i < 30) {
8                 alertCode = 1;
9             }
10            drawInt (numberOfSecs - i, alertCode);
11        }
```

Listing 1 – Que fait cette séquence d'instructions?

Exercice 1 (Premier décodage, ⋆)
Dans le programme JAVA *précédent, classifiez les parties du code source correspondant aux expressions, aux instructions et aux déclarations.* □

2 Le langage des expressions arithmétiques

Sous-langage des expressions arithmétiques [COURS]

- Les expressions sont classifiées à l'aide de **types** correspondant à la forme des valeurs qu'elles calculent.
- Le type `int` est celui des expressions qui calculent des valeurs entières, les expressions arithmétiques.
- Le type `int` est l'ensemble des valeurs entre -2147483648 ($= -2^{31}$) et 2147483647 ($= 2^{31} - 1$).
- Une **expression arithmétique de type** `int` peut être :
  (a) une constante entière (`0`, `1`, 2, `-1`, `-2`, -2147483648, 2147483647 ...);
  (b) deux expressions séparées par une opération arithmétique binaire (`1 + 2`, `1 + 2 * 3 / 4`, ...);
  (c) une expression entourée de parenthèses (`(1 + 2)`, `(1 + 2 * 3 / 4)`, ...);
  (d) une expression précédée du signe moins (`-2`, `-(1 + 2)`, ...).
- Les opérateurs ont des priorités relatives, par exemple
  {`*`, `/`, `%`} $\preceq$ {`+`,`-`}
  où $\preceq$ signifie "être prioritaire sur".
- Les opérateurs binaires précédents sont associatifs à gauche.
- On peut toujours rajouter des parenthèses pour expliciter la priorité de certains calculs sur d'autres.
- L'évaluation de l'expression "`1 + 2 * 3 - 4`" est équivalente à l'expression "`1 + (2 * 3) - 4`" et elle se décompose en (i) "`2 * 3`" donne 6 ; (ii) "`1 + 6`" en 7 et (iii) "`7 - 4`" donne 3.
- Les opérations ($+$, $-$, $/$, $*$, ...) ont le sens usuel *tant que l'on reste entre les bornes du type* `int`.
- Dans le type `int`, la division $/$ est une *division entière*. Par exemple, "`31 / 7`" vaut 4.
- L'opérateur % désigne le *modulo* : "`a % b`" est le reste dans la division entière de `a` par `b`.
  Par exemple, "`31 % 7`" vaut 3 car $31 = 7 \times 4 + 3$.

Exercice 2 (Java comme une calculatrice, ⋆)
Prévoir l'évaluation des expressions arithmétiques suivantes :

```
6 * 7 + 3
6 * (7 + 3)
45 / 7
3 * 7 / 4
(3 * 7) / 4
(45 / 7) * 7 + 45 % 7
(1 + 2 - 3 + 4 - 5 + 6 - 7 + 8 - 9 + 10 - 11 + 12 - 13) / (1 - 2 + 3 -
    4 + 5 - 6 + 7 - 8 + 9 - 10 + 11 - 12 + 13)
```

□

3 Utilisation des variables

Déclaration et utilisation des variables [Cours]

— Une **variable** a un *nom* et contient une *valeur*, c'est-à-dire le résultat d'un calcul.
— On déclare une variable `x` de type `int` qui contient initialement le résultat du calcul `6 * 7` ainsi :

```
1 int x = 6 * 7;
```

— Dans l'exemple précédent, la valeur de la variable `x` est 42.
— Exemples de noms de variable : `x`, `y`, `foo`, `foobar42`...
— On peut utiliser la valeur d'une variable dans une expression en faisant référence à son nom. Ainsi, l'expression "`x + 1`" vaut 43 si `x` vaut 42.
— On peut changer la valeur d'une variable qui a été déclarée auparavant en lui *affectant* le résultat d'un nouveau calcul. L'opérateur d'affectation est « = » :

```
1 x = 2 * 10;
```

— Sur papier, une mémoire contenant les variables `x = 42`, `y = 73` et `z = 37` sera écrite :

x	y	z
42	73	37

Instructions [Cours]

— Une expression calcule une valeur tandis qu'une instruction a un effet sur la machine.
— Une affectation est un exemple d'instruction.
— On dit qu'une machine exécute une instruction.

Exercice 3 (Nommer, ⋆)
Quelle est la valeur des variables, à la suite des instructions suivantes.

```
1 int x = 6 * 7;
2 int y = x + x;
```

□

Exercice 4 (Affectations, ⋆⋆)
Quelle est la valeur des variables, à la suite des instructions suivantes.

```
1 int x = 1;
2 int y = 4;
3 x = y + 2;
```

□

Exercice 5 (Affectations, ⋆⋆)
Quelle est la valeur des variables, à la suite des instructions suivantes.

```
1 int x = 1;
2 int y = x - 1;
3 x = 2;
```

□

Exercice 6 (Affectations, ⋆⋆)
Quelle est la valeur des variables, à la suite des instructions suivantes.

```
1 int x = 1;
2 x = x - 1;
```

□

4 Fonctions et procédures

Nommer un calcul comme une fonction [Cours]

— La fonction qui attend en paramètre une valeur `x` de type `int` et calcule une valeur de type `int` qui est le triple de `x` s'écrit :

```
public static int triple (int x) {
    return (x * 3);
}
```

— On peut utiliser une fonction en écrivant son nom suivi de son paramètre entouré de parenthèses, comme par exemple dans l'expression :

```
triple (2) * 6
```

— Dans cet exemple, `triple (2)` vaut 6 donc l'expression vaut 36.
— Voici des exemples de noms de fonction : `triple`, `add3`, `fooBar`, ...
— Une fonction peut attendre plusieurs paramètres en les séparant par des virgules.
— On peut utiliser des fonctions prédéfinies dans des *bibliothèques de fonctions*.
— On fait référence à une procédure écrite dans une bibliothèque en utilisant une notation "pointée". Par exemple, `System.out.println` est le nom de la procédure `println` de `System.out`.

Exercice 7 (Utiliser une fonction, ⋆)
Étant donnée la fonction suivante, que vaut l'expression `twice(3)` *?*

```
public static int twice (int x) {
    return (2 * x);
}
```

□

Exercice 8 (Utiliser une fonction, ⋆⋆)
On considère la fonction suivante.

```
public static int cube (int x) {
    return (x * x * x);
}
```

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

```
int b = 5;
int a = 3;
a = b - a;
a = cube(a);
```

□

Exercice 9 (Écrire une fonction, ⋆⋆)
Voici une fonction

```java
public static int f(int x) {
    return (x * x + 5);
}
```

Quel est son nom ? Que calcule-t-elle ? Modifier son corps pour qu'elle calcule la division entière par 3. Modifier son nom pour qu'elle se nomme third.

□

Exercice 10 (Écrire une fonction, ⋆⋆)
Écrire des fonctions effectuant les calculs suivants :

1. *La puissance 5 d'un entier donné en paramètre.*
2. *Le produit de deux entiers donnés en paramètre moins leur somme.*
3. *Le produit de trois entiers donnés en paramètre au carré.*

□

Utiliser une procédure [Cours]

— Un **appel de procédure** est une instruction écrite à l'aide du nom de la procédure suivi d'un ou plusieurs paramètres séparés par des virgules et entourés de parenthèses. Par exemple :

```java
System.out.println (1 + 2);
```

affiche 3 à l'écran à l'aide de la procédure prédéfinie `System.out.println`. Un autre exemple :

```java
putPixel(0, 0, 255, 255, 255);
```

affiche un pixel blanc en position (0, 0) d'une image à l'aide de la procédure prédéfinie `putPixel`.

Écrire une procédure [Cours]

— Une procédure est définie en termes d'une suite d'instructions.
— Par exemple, la procédure qui attend deux entiers `x` et `y` et qui affiche successivement leur somme et leur produit s'écrit :

```java
public static void showProductAndSum (int x, int y) {
  System.out.println (x + y);
  System.out.println (x * y);
}
```

Exercice 11 (Différences syntaxiques, ⋆⋆)

1. *Quelles différences trouvez-vous entre la syntaxe de déclaration des fonctions et des procédures ?*
2. *Même question pour l'appel de fonctions et l'appel de procédures.*

□

5 Instruction conditionnelle

Instruction conditionnelle [Cours]

— Une instruction conditionnelle permet d'exécuter des instructions en fonction d'une condition (une expression qui est soit vraie, soit fausse).
— On écrit par exemple

```
int x = 10;
int y = 42;
if (x <= 0) {
    y = x;
} else {
    y = -x;
}
```

pour affecter la valeur de `x` à `y` si `x <= 0` ou pour lui affecter la valeur `-x` dans le cas contraire.
— Les conditions peuvent par exemple être des comparaisons entre deux expressions de type `int` : `e1 == e2`, `e1 != e2`, `e1 < e2`, `e1 <= e2`, `e1 > e2`, `e1 >= e2`).

Exercice 12 (Le max, ⋆)

À la suite des instructions ci-dessous, quelle est la valeur de la variable `max` *?*

```
int x = 3;
int y = 4;
int max = 0;
if (x > y) {
    max = x;
} else {
    max = y;
}
```

Transformez la suite d'instructions pour calculer le minimum de `x` *et* `y` *et le mettre dans une variable* `min`.

□

Exercice 13 (Différents tests, ⋆⋆)

Quelle est la valeur des variables `a`, `b` *et* `c` *après la suite d'instructions suivante :*

```
int a = 2;
int b = a * a + 3;
int c = b - a;
if (c == a) {
    a = 1;
} else {
    a = a + 3;
}
if (b + c < a) {
    b = 2;
} else {
    b = 4;
}
if (b != c * c) {
    c = 12;
} else {
    c = -6;
}
```

□

6 Boucles

Boucles [Cours]

— Une boucle permet de répéter plusieurs fois les mêmes instructions.
— Le numéro de l'itération est disponible dans une variable qui s'appelle le *compteur de boucle*.
— La boucle suivante affiche les entiers de 0 à 9 :

```
for (int i = 0; i <= 9; i++) {
    System.out.println (i);
}
```

— L'*en-tête* est formé du mot clé `for` suivi d'une initialisation, d'une condition, et d'une incrémentation, séparées par des points-virgules et entourées de parenthèses.
— L'*initialisation* "`int i = 0`" déclare le compteur de boucle, son type et sa valeur initiale.
— La *condition de boucle* "`i <= 9`" définit sous quelle condition l'exécution de la boucle continue.
— L'*instruction d'incrémentation* `i++`. Cette instruction est équivalente ici à l'instruction "`i = i + 1`".
— Le *corps de la boucle*, entre accolades, est exécuté à chaque itération de la boucle.
— La même boucle aurait pu être écrite comme suit :

```
for (int i = 0; i < 10; i++) {
    System.out.println (i);
}
```

— Nous verrons qu'il existe un autre type de boucle introduite par le mot-clé `while`.

Exercice 14 (Afficher des suites d'entiers, ⋆)

1. *Écrivez une boucle qui affiche les 100 premiers entiers, en commençant à 0. Quel est le dernier entier affiché ?*
2. *Écrivez une boucle qui affiche les 50 premiers entiers pairs, en commençant à 0. Quel est le dernier entier affiché ?*
3. *Écrivez une boucle qui affiche 1000 fois le nombre 3.*

□

7 Do it yourself

Exercice 15 (Valeurs d'expressions entières, ⋆)
Donnez la valeur des trois expressions suivantes : `3 + 5 / 3` `4 * 1 / 4` `2 / 3 * 3 - 2` □

Exercice 16 (Modulo 9, ⋆⋆)
Donnez la valeur des expressions suivantes :
`18 % 9` `81 + 18 % 9` `(81 + 18) % 9`

□

Exercice 17 (Valeur absolue, ⋆)
En vous inspirant de l'exercice 12, donner une suite d'instructions permettant de calculer la valeur absolue d'une variable. □

Exercice 18 (Utiliser une fonction, ⋆)
On considère la fonction suivante.

```
public static int sumsquare (int x, int y) {
    return (x * x + y * y);
}
```

Quelle est la valeur des variables `a`, `b` *et* `c` *à la suite des instructions suivantes :*

```
int a = sumsquare(2, 3);
int b = sumsquare(3, 1);
int c = sumsquare(4, 3);
```

□

Exercice 19 (Boucles simples, ⋆⋆)

1. *Quel est l'affichage produit par la suite d'instructions suivante :*

```
for (int i = 0; i < 5; i++) {
    System.out.println (8);
}
```

2. *Quelle est la valeur de* `x` *après la suite d'instructions suivante :*

```
int x = 0;
for (int i = 0; i < 5; i++) {
    x = x + 8;
}
```

3. *Quelle est la valeur de* `x` *après la suite d'instructions suivante :*

```
int x = 0;
for (int i = 0; i < 5; i++) {
    x = 10 * x + 8;
}
```

□

Exercice 20 (Des entiers qui s'ajoutent, ⋆⋆)
Quelle est la valeur de `sum` *après la suite d'instructions suivante :*

```
int i = 0;
int sum = 0;
sum = sum + i;
i = i + 1;
sum = sum + i;
i = i + 1;
sum = sum + i;
i = i + 1;
sum = sum + i;
i = i + 1;
sum = sum + i;
i = i + 1;
sum = sum + i;
i = i + 1;
```

Si on veut adapter le code ci-dessus pour aller non plus jusqu'à 5 mais jusqu'à 100, 1000 ou plus, on a un problème : on obtiendrait un programme beaucoup trop long !

On peut remplacer la longue liste d'instructions par une boucle `for` *en remarquant que l'instruction "*`sum = sum + i;`*" est exécutée 10 fois avec* `i` *prenant pour valeurs les différents entiers entre 1 et 5 car la variable est systématiquement incrémentée (sa valeur est augmentée de 1) grâce à l'instruction "*`i = i + 1;`*".*

On obtient ainsi la boucle :

```
int bound = 5;
int sum = 0;
for (int i = 0; i <= bound; i++) {
    sum = sum + i;
}
```

1. *On remplace la première ligne du code précédent par "*`int bound = 100;`*".*
   Quelle est la valeur de `sum` *après l'exécution de cette suite d'instructions ?*
   Même question si on remplace la première ligne par "`int bound = 1000;`*" ?*
2. *Modifiez le code précédent pour calculer la somme des entiers de* 10 *à* 100.
3. *Écrivez une fonction "*`int sumIntegers (int n)`*" qui renvoie la somme des entiers de* 0 *à* n.

□

Introduction à la Programmation 1 JAVA

51AE011F

Séance 2 de cours/TD

Université Paris-Diderot

Objectifs:

- Découverte du type `String`.
- Comprendre qu'il y a des types différents.
- Maîtriser les expressions booléennes dans les conditions.
- Comprendre l'utilisation des boucles avec dépendances.

1 Les chaînes de caractères : le type `String`

Les chaînes de caractères [COURS]

- Les chaînes de caractères représentent du texte.
- Les constantes de chaînes de caractères sont écrites entre guillemets doubles.
- `"Mr Robot"` est un exemple de chaîne.
- Pour introduire un guillemet dans une chaîne, on écrit `\"`.
- La chaîne vide s'écrit `""`.
- L'opérateur de concaténation de deux chaînes est +.
- Dans une chaîne, la séquence `\n` représente le passage à une nouvelle ligne.

Exercice 1 (De premiers exemples, ⋆)
Quelle est la valeur des variables après l'exécution des instructions suivantes ?

```
1 String s = " rentre chez lui.";
2 String s2 = "Paul";
3 String s3 = "Michel";
4 String s4 = s2 + s;
5 String s5 = s3 + s;
6 String s6 = s2 + " et " + s3 + " rentrent chez eux.\n";
```

□

Différence entre affichage et calcul [COURS]

- Il ne faut pas confondre l'affichage d'une chaîne de caractères sur le terminal et le calcul d'une chaîne.

```
1 System.out.println ("Omar");
```

affiche la ligne `Omar` sur le terminal tandis que le programme suivant n'affiche rien :

```
1   a = "Omar" + "Roma";
```

— mais initialise une variable a avec le résultat du calcul qui concatène les deux chaînes `"Omar"` et `"Roma"` en la chaîne `"OmarRoma"`.

2 Utilisation des types `int` et `String`

Les chaînes de caractères et les entiers sont des types différents [Cours]

— Une variable `x` de type `int` ne peut pas contenir de chaîne de caractères.
— Une variable `s` de type `String` ne peut pas contenir d'entier.
— En général, une fonction ou une procédure attend des arguments de types spécifiés lors de sa déclaration.
— Par exemple, la fonction « `void factorial (int n)` » ne peut pas être utilisée avec un argument de type `String`. Ainsi, écrire "`factorial ("Yoda")` provoquera une **erreur de typage** au moment de la compilation.

Exercice 2 (Des exemples simples avec des variables de différents type, ⋆)

1. *Quelle est la valeur de la variable* `ch` *après l'exécution des instructions suivantes ?*

```
int x = 3;
String ch = "Salut";
if (x <= 2) {
    ch = "Hallo";
} else {
    ch = "Hi";
}
```

2. *Que fait la suite d'instructions suivantes ?*

```
int x = 3;
String st = "Ca va ?";
String ch = "";
x = x / 2;
if (x == 1) {
    ch = "How are you ?";
} else {
    ch = "Comment allez-vous ?";
}
ch = ch + "\n";
System.out.print (ch);
```

□

Exercice 3 (Deux représentations très différentes de la même chose, ⋆⋆)

— *Que vaut l'expression* `"41" + "1"` *?*
— *Que vaut l'expression* `41 + 1` *?*
— *Quelle différence faites-vous donc entre la chaîne de caractères* `"42"` *et l'entier* 42 *? Dans quelles situations utiliser l'une ou l'autre de ces représentations de l'entier* 42 *?*

□

Exercice 4 (Trouver le bon type, ⋆⋆⋆)
Dans les fonctions suivantes, remplacez les `[...]` *par les bons types :*

```java

public static [...] f ([...] x) {
    [...] y = x % 2 ;
    if (y == 0) {
        System.out.println ("Argument pair");
    } else {
        System.out.println ("Argument impair");
    }
}

public static [...] g ([...] x) {
    [...] y = x % 2 ;
    if (y == 0) {
        return "Argument pair";
    } else {
        return "Argument impair";
    }
}

public static [...] h ( [...] x, [...] y) {
    return (x + y);
}

public static [...] id ( [...] x) {
    return x;
}
```

code/ExoInference.java

□

3 Fonctions avec le type `String`

Les fonctions peuvent utiliser le type `String` [Cours]

— Une fonction peut renvoyer une valeur de type `String`.
— Les valeurs de type `String` peuvent être utilisées comme paramètres d'une fonction.

On considère les deux fonctions données ci-dessous :

```java
public static String sandwich (String s, String s2) {
    return s + s2 + s;
}

public static String neymarOrNot (int x) {
    String st = "";
    if (x == 10) {
        st = "C'est le numéro de Neymar.";
    } else {
        st = "Ce n'est pas le numéro de Neymar.";
    }
    return st;
}
```

Elles peuvent être utilisées de la façon suivante :

```
String s = sandwich ("Hello", "World!\n");
String s2 = sandwich (s, "Ca va ?");
String s3 = neymarOrNot (1);
```

Voici quelques fonctions et procédures que nous vous fournirons couramment pour écrire des programmes manipulant des chaînes de caractères :

- Pour avoir la longueur d'une chaîne de caractères stockée dans une variable `s`, on écrira `s.length()`
- Obtenir le $(i+1)$-ème caractère d'une chaîne avec "`String characterAtPos (String s, int i)`" (Le premier caractère est à la position 0 et si i est plus grand que la longueur, la fonction renvoie la chaîne vide "".) ;

(De telles fonctions vous seront fournies ce semestre.)

Exercice 5 (Utilisation de fonctions données, $\star$)

1. *On considère la fonction suivante :*

```
public static String addA (String ch) {
    return (ch + "A");
}
```

Que vaut la variable `st` *après exécution du code suivant ?*

```
String s = "Ma lettre finale est ";
String st = addA (s);
```

2. *Quelle est la valeur de la variable* `a` *après les instructions suivantes où "*`int stringLength(String s)`*" est la fonction décrite ci-dessus ?*

```
String ch = "Ceci est ";
String ch2 = "une chaine";
ch = ch + ch2;
int a = stringLength (ch);
```

3. *Que vaut la variable* `cha` *dans l'exemple suivant où "*`String characterAtPos (String s, int i)`*" est la fonction décrite ci-dessus ?*

```
String ch = "Avec consonnes";
String cha = characterAtPos (ch, 0);
cha = cha + characterAtPos (ch, 2);
cha = cha + characterAtPos (ch, 6);
cha = cha + characterAtPos (ch, 9);
cha = cha + characterAtPos (ch, 12);
```

□

Exercice 6 (Modification d'une fonction, $\star\star$)

On considère la fonction suivante où la fonction "`String intToString(int x)`*" est celle décrite ci-dessus.*

```
public static String message (int x) {
    return intToString (x);
}
```

Et on considère la liste d'instructions suivante :

```
int a = 4;
int b = a * a;
a = b - a;
String s = message (a);
System.out.println (s);
```

1. *Quelles sont les valeurs des variables de ce programme à la fin de son exécution ?*
2. *Qu'affichent ces instructions ?*
3. *Est-il possible de modifier la fonction "*`String message (int x)`*" de telle sorte que le programme affiche à la fin "Le résultat du calcul est n." (où n est la valeur contenue dans l'argument transmis à* `message`*) ?*
4. *Définissez une fonction "*`int square (int n)`*" qui calcule le carré du nombre donné en paramètre et utilisez-la pour afficher une ligne du type $n * n = n^2$ (où n sera remplacé par sa valeur).*

□

4 Expressions booléennes dans les conditionnelles

Des expressions booléennes plus complexes dans les tests [Cours]

- Une expression booléenne a le type `boolean`.
- Une expression booléenne peut s'évaluer en l'une des deux valeurs `true` ou `false`.
- Une expression avec un **et** (en Java `&&`) est vraie si les expressions à gauche et à droite du **et** sont vraies.
- Une expression avec un **ou** (en Java `||`) est vraie si au moins une des expressions à gauche et à droite du **ou** est vraie.
- Une expression avec une négation (en Java `!`) est vraie si l'expression après la négation est fausse.
- L'opérateur unaire `!` est prioritaire sur l'opérateur `&&` qui est lui-même prioritaire sur l'opérateur `||`.
- Exemples d'expressions utilisant ces opérateurs :
  - `(2 < x && x <= 5)`
  - `(x == 1 || x == 2 )`
  - `!(x == 0)` (qui est équivalent à "`x != 0`")
- On peut combiner ces opérateurs, par exemple en écrivant "`(x == 1 || x == 2) && y > 5`"
- Attention, on ne peut pas tester l'égalité de deux chaînes de caractères avec `==`. Pour cela, on vous donne une fonction "`boolean stringEquals (String s1, String s2)`" à utiliser dans les tests.
- Voici un exemple :

```
public static void profiler (String name) {
  if (stringEquals (name, "Paul") || stringEquals (name, "Pierre")) {
      System.out.println ("On a trouve Paul ou Pierre.");
  } else {
      System.out.println ("Ce n'est ni Paul, ni Pierre.");
  }
}
```

Exercice 7 (Savoir évaluer une condition, ⋆)
Quelle est la valeur de la variable `x` *après la suite d'instructions suivante ?*

```
int a = 2;
a = a * a * a * a + 1;
int b = a / 2;
int c = a - 1;
int x = 0;
```

```
if ((b == 0 ) && (c == 16)) {
    x = 1;
} else {
    x = 2;
}
```

□

Exercice 8 (Équivalence d'expressions, ⋆⋆)
On dit que deux expressions booléennes sont équivalentes *quand ces deux expressions s'évaluent toujours vers la même valeur booléenne pour toutes les valeurs possibles des variables qui apparaissent dans ces expressions.*

Par exemple, "`x > 10 && x < 12`*" est équivalente à "*`x == 11`*". Par contre, "*`x > y`*" et "*`x != y`*" ne sont pas équivalentes.*

Dire si les expressions suivantes sont équivalentes :

1. *"*`x > y || x < y`*" et "*`x != y`*";*
2. *"*`x != 3 && x != 4 && x != 5`*" et "*`x <= 2 || x >= 6`*";*
3. *"*`x ==  y && x == z`*" et "*`x == z`*";*
4. *"*`x ==  y && x == z`*" et "*`x == y && y == z`*".*

□

Exercice 9 (Simplification des expressions booléennes, ⋆⋆)

Simplifier *une expression booléenne consiste à la remplacer par une autre expression booléenne équivalente qui est plus courte. Simplifier les expressions suivantes :*

- `x > 5 && x > 7`
- `x == y && x == z && y == z`
- `x == 17 || (x != 17 && x == 42)`
- `x > 5 || (x <= 5 && y > 5)`

□

5 Instructions conditionnelles imbriquées

Instructions composées [Cours]

- Certaines des instructions présentées sont des instructions *composées* : elles peuvent contenir d'autres instructions.
- Les instructions conditionnelles et les boucles sont des instructions composées.
- Par exemple :

```
int x = 1;
int y = 2;
int z = 0;
if (x == 1) {
    if (y == 2) {
        z = 3;
    } else {
        z = 5;
    }
} else {
    z = 7;
}
```

L'exécution du programme précédent a pour effet d'affecter la valeur 3 à la variable z.

— On a le droit d'imbriquer des instructions dans d'autres instructions à volonté : des conditionnelles dans des boucles dans des conditionnelles, etc ...
— Avec des instructions imbriquées sur plusieurs niveaux, il est absolument indispensable (dans l'intérêt des lecteurs humains) de suivre strictement une discipline d'indentation du code source.
— On parlera des boucles imbriquées lors de la séance 3.

Exercice 10 (Utilisation des opérateurs booléens, ⋆⋆)

Réécrire les suites d'instructions suivantes en utilisant moins de tests `if`.

1.

```
public static void f (int l) {
    int x = 0;
    if (l != 0) {
      if (l <= 10) {
        x = 1;
      } else {
        x  = 2;
      }
    } else {
        x = 2;
    }
}
```

Quelle est la valeur finale de `x` *pour des valeurs de* `l` *dans* $\{0, 5, 15\}$ *? Vérifier que votre réécriture du code est compatible.*

2.

```
public static void g (int a) {
    int b = 2 * a;
    b = a - a;
    if (b == a) {
        b = 0;
    } else {
      if (b > 43) {
        b  = 0;
      } else {
        b = 1;
      }
    }
}
```

Quelle est la valeur finale de `b` *pour des valeurs de* `a` *dans* $\{0, 25, 50\}$ *? Vérifier que votre réécriture du code est compatible.*

3.

```
public static void h (int c, int d) {
    int e = 0;
    if (d == 6) {
        e = 3;
    } else {
       if (c >= 2){
        e = 2;
      } else {
        e = 3;
```

```
        }
    }
}
```

Quelle est la valeur finale de `e` *pour de valeurs de* (`c`,`d`) *dans* $\{(0,0),(0,6),(6,0),(6,6)\}$ *? Vérifier que votre réécriture du code est compatible.*

□

6 Boucles

Exercice 11 (Pendu à l'envers, ⋆⋆)

1. *Écrire une fonction qui prend en argument une chaîne de caractères et affiche une suite de tirets (-) de même longueur.*

   Contrat:
   argument = *"amphitheatre"* → *affichage : - - - - - - - - - - -*

2. *Écrire une fonction qui prend en argument une chaîne de caractères et l'affiche à l'envers.*

   Contrat:
   argument = *"amphitheatre"* → *affichage : ertaehtihpma*

□

Boucles et conditionnelles [Cours]

Le corps d'une boucle peut contenir n'importe quelle instruction. En particulier, il peut contenir une conditionnelle. Par exemple, le programme suivant affiche `"Pair."` pour les valeurs de `i` qui sont des entiers pairs et `"Impair."` pour les valeurs de `i` qui sont des entiers impairs.

```
for (int i = 0; i < 100; i++) {
    if (i % 2 == 0) {
        System.out.println ("Pair.");
    } else {
        System.out.println ("Impair.");
    }
}
```

Dans le programme précédent les instructions qui sont effectuées à chaque itération *dépendent de la valeur du compteur de boucle* `i`. On ne se contente donc pas de répéter toujours la même chose : ce que l'on fait à chaque itération dépend du nombre d'itérations déjà effectuées, c'est-à-dire de l'indice de l'itération courante, représenté par la valeur du compteur `i`.

Exercice 12 (Effacement des espaces, ⋆)
Ecrire une fonction qui prend en argument une chaîne de caractères et affiche cette même chaîne sans les espaces.

Contrat:
argument = *"Bonjour monde !"* → *affichage : Bonjourmonde !*

□

7 Fonctions utilisées

Liste des fonctions [Cours]

```java
/* Transforme un entier x en une chaîne de caractères.
 * Renvoie la chaîne de caractères représentant la valeur de x.
 */
public static String intToString(int x) {
    return String.valueOf (x);
}

/*
 * Renvoie une chaîne constituée du caractère se trouvant à la
 * position $i$ de la chaîne s.
 * Si i est négatif ou en dehors de la chaîne, renvoie la chaîne vide.
 */
public static String characterAtPos(String s,int i) {
    String res = "";
    if(i >= 0 && i < s.length ()) {
        res = String.valueOf (s.charAt (i));
    }
    return res;
}

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

8 Do it yourself

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 13 (Concaténation, ⋆)
Écrire une fonction qui prend en paramètre une chaîne de caractères `toto` *et affiche une ligne commençant par* `bonjour,` *suivi du contenu de* `toto`.

> ***Contrat:***
> *si* `toto` *a la valeur* `"toi"`*, l'appel à la fonction doit afficher*
> `bonjour, toi`

□

Exercice 14 (Concaténation et somme, ⋆⋆)
Que valent les variables `x` *et* `s` *après les instructions suivantes ?*

```java
int x = 0;
String s = "";
for (int n = 0; n < 10; n++) {
    x = x + 1;
```

```
    s = s + "1";
}
```

□

Exercice 15 (Condition et division entière, ⋆)
Écrire une condition permettant de tester si le tiers de l'entier `x` *appartient à l'intervalle* $[2;8]$.

Contrat:
x = 0 → false
x = 6 → true
x = 9 → true
x = 25 → false

□

Exercice 16 (Condition, ⋆)
Écrire une condition permettant de tester si les entiers `a`, `b` *et* `h` *peuvent correspondre aux longueurs des côtés d'un triangle rectangle, où* `h` *serait la longueur de l'hypothénuse.*

Contrat:
(a,b,h) = (3,4,5) → true
(a,b,h) = (1,2,3) → false

□

Exercice 17 (Somme des cubes, ⋆)
Écrire une boucle permettant de calculer la somme des cubes des entiers de 1 à 100. □

Exercice 18 (Ordinaux anglais, ⋆⋆⋆)
On s'intéresse aux ordinaux anglais abrégés, *où le nombre (le cardinal) est écrit en chiffres :*

1st, 2nd, 3rd, 4th,..., 9th, 10th, 11th, 12th,..., 19th, 20th, 21st, 22nd, 23rd,...

Pour déterminer le suffixe, on regarde le dernier chiffre du nombre : si c'est 1, on ajoute le suffixe "st" ; si c'est 2, le suffixe est "nd" ; si c'est 3, le suffixe est "rd" ; sinon le suffixe est "th". Il y a cependant une exception : si l'avant-dernier chiffre du nombre est 1, le suffixe est toujours "th".

Écrire une fonction `printOrdinal` *qui prend un entier de type* `int` *en paramètre et affiche l'ordinal anglais abrégé correspondant.*

Contrat:
1 → 1st
12 → 12th
23 → 23rd
32 → 32nd
44 → 44th

□

Exercice 19 (Simplification de code, ⋆)

1. *Dire, en justifiant, ce que fait la fonction suivante selon les valeurs de* x *et* y.

```
public static int f (int x, int y) {
    if (((x <= y) || (x >= y + 1))) {
        if ((x <= y) && (x < y)) {
            if (x < -x) {
                int a = -x;
                return a;
            } else {
                return x;
            }
```

```
        } else {
            if (y * y * y < 0) {
                return -y;
            }
        }
    }
    return y;
}
```

2. *Écrire la même fonction de manière plus simple.*

□

Exercice 20 (Mention, ⋆)
Écrire une conditionnelle afin de stocker dans une chaîne de caractères `s` *la mention correspondant à la note (entière) contenue dans la variable* `n` *de type* `int`.

Rappel : entre 10 inclus et 12 exclu, la mention est passable ; assez bien *entre 12 inclus et 14 exclu ;* bien *entre 14 inclus et 16 exclu, et* très bien *au-delà de 16.* □

Exercice 21 (Lignes et pointillés, ⋆⋆)

1. *Écrivez une fonction qui prend en paramètre un entier supposé positif n et qui affiche une ligne d'astérisques « * » de longueur n puis va à la ligne. Par exemple, si n vaut 7, votre fonction affichera :*
   ****
2. *Modifiez votre fonction pour qu'elle affiche une ligne pointillée alternant astérisques et espaces. Par exemple, si n vaut 7, votre fonction affichera :*
   * * * *
   Qu'affiche votre fonction si n est pair ?

□

Exercice 22 (Nombres parfaits, ⋆⋆⋆)
Un entier $n > 1$ est dit parfait *s'il est égal à la somme de ses diviseurs propres (c'est-à-dire autres que lui-même). Par exemple, 6 est parfait car ses diviseurs propres sont 1, 2 et 3, et on a $6 = 1 + 2 + 3$.*

1. *Écrire une fonction* `divisors` *qui prend en entrée un entier* `n` *et affiche tous les entiers strictement inférieurs à* `n` *qui divisent* `n`.

   Contrat:
   n = 1 → affichage :
   n = 5 → affichage : 1
   n = 60 → affichage : 1 2 3 4 5 6 10 12 15 20 30

2. *Écrire une procédure* `void isPerfect(int n)` *qui prend en entrée un entier* `n` *et qui affiche* `"n est parfait"` *si* `n` *est parfait.*
3. *Écrire une procédure* `void enumPerfect(int m)` *qui prend en paramètre un entier* `m` *et affiche pour tous les nombres parfaits tels que* `n`≤`m`, `"n est parfait"` *(en remplaçant avec la valeur de* `n`*).*

□

Exercice 23 (Occurrences d'un caractère, ⋆⋆)
Écrire une fonction `inString` *qui prend en entrée une chaîne de caractères* `s` *et un caractère* `c` *(c'est-à-dire* `c` *est une chaîne de caractères qu'on suppose de longueur 1), et affiche toutes les positions de* `s` *où* `c` *apparaît.*

Contrat:
(s,c) = ("abracadabra","a") → affichage : 0 3 5 7 10

□

Introduction à la Programmation 1 JAVA

51AE011F

Séance 3 de cours/TD

Université Paris-Diderot

Objectifs:

— Manipuler des boucles avec accumulateurs

— Manipuler deux boucles imbriquées

1 Accumuler des valeurs grâce aux boucles

Utilisation d'accumulateur dans les boucles [COURS]

— Si une variable est déclarée avant une boucle, sa valeur persiste d'une itération de la boucle à la suivante. On peut donc modifier sa valeur à chaque itération de boucle, et donc *accumuler* une valeur dans cette variable.

— Par exemple, le fragment de code suivant calcule la somme des carrés des entiers de 1 à 100 :

```
int s = 0;
for(int i = 1; i <= 100; i++) {
    s = s + i * i;
}
```

Exercice 1 (Accumulation, ⋆)
Modifier la boucle donnée ci-dessus pour qu'elle calcule la somme des cubes des entiers de 1 à 42. □

Exercice 2 (Comprendre et modifier une boucle, ⋆)

```
String st = "";
for (int i = 1; i <= 50; i++) {
    st = st + "ab";
}
```

1. *Que contient la variable* `st` *après la suite d'instructions donnée ci-dessus ?*
2. *Modifier les instructions ci-dessus pour qu'à la fin de leur exécution la variable* `st` *contienne la chaîne de caractère "aaaaaa ... aaaa" avec la lettre "a" répétée 110 fois.*
3. *Modifier de nouveau la suite d'instructions pour imprimer à l'écran à chaque tour de boucle le contenu de la variable* `st`*. Qu'affichera l'exécution de votre code ?*

□

Exercice 3 (Accumulation booléenne, ⋆⋆)
On suppose donnée la fonction suivante qui lit un entier au clavier :

```java
public static int readInt() {
    Scanner sc = new Scanner (System.in);
    return sc.nextInt ();
}
```

Écrire un fragment de code qui lit 5 entiers au clavier, puis affiche « Gagné » si l'entier 42 se trouvait parmi ceux-là, et sinon « Perdu ». Attention, il faut lire les 5 entiers et seulement ensuite afficher le résultat. □

2 Boucles imbriquées

Boucles imbriquées [Cours]

Le corps d'une boucle peut lui-même contenir une boucle. On parle alors de *boucles imbriquées* :

```java
for (int i = 0; i < 10; i++) {
    for (int j = 1; j <= 5; j++) {
        printInt (i);
        printInt (j);
    }
}
```

- À chaque tour de la boucle externe, la boucle interne fait un tour complet. La boucle externe est donc lente, la boucle interne est rapide.
- Chaque boucle a son propre compteur.
- Pour le lecteur humain, il est essentiel d'utiliser l'indentation pour indiquer l'imbrication des boucles. Comme d'habitude, le compilateur ignore l'indentation et ne considère que les accolades.
- La variable de la boucle externe peut être utilisée dans la boucle interne. Par contre, la variable de la boucle interne (j) n'est pas accessible en dehors de celle-ci.

Exercice 4 (Cent, $\star$)
Écrire un fragment de code qui affiche les nombres de 0 à 99, à raison de 10 nombres par ligne :

```
0 1 2 3 4 5 6 7 8 9
10 11 12 13...
...
90 91 92 93 94 95 96 97 98 99
```

□

Exercice 5 (Triangle, $\star\star$)
Écrire une fonction qui prend en paramètre un entier n et affiche, pour i allant de 1 à n, i étoiles sur la i-ième ligne. Par exemple, pour $n = 5$, afficher :

```
*
* *
* * *
* * * *
* * * * *
```

□

3 Do it yourself

Exercice 6 (Puissance, ⋆)
Écrire une fonction "`int power (int x, int n)`*" qui renvoie la valeur* x^n*.* □

Exercice 7 (Factorielle, ⋆)
Écrire une fonction "`int fact (int n)`*" qui renvoie la factorielle de* n*.* □

Exercice 8 (Ça use, ⋆⋆)

1. *Écrire un fragment de code qui affiche les paroles de la chanson que vos neveux chantaient durant les vacances d'été :*
   ```
   1 kilomètre à pied, ça use les souliers.
   2 kilomètres à pied, ça use les souliers.
   ...
   100 kilomètres à pied, ça use les souliers.
   ```
   Attention, le premier vers est différent (le mot kilomètre *est au singulier).*
2. *Modifier votre code pour qu'il affiche les vers dans l'ordre inverse.*

□

Exercice 9 (Nombres premiers, ⋆⋆)

1. *Écrire une fonction "*`int isPrime (int n)`*" qui renvoie* 1 *si* n *est un nombre premier,* 0 *sinon. (On rappelle que* 1 *n'est* pas *un nombre premier ; le plus petit nombre premier est donc* 2*.)*
2. *Écrire une fonction "*`int sumPrime (int n)`*" qui renvoie la somme des nombres premiers compris (au sens large) entre* 1 *et* n*.*

□

Exercice 10 (Table de multiplication, ⋆⋆)

1. *Écrire un fragment de code qui affiche les tables de multiplication pour les entiers de* 1 *à* 10*.*
   ```
   1 2 3 4 ... 10
   2 4 6 8 ... 20
   ...
   10 20 30 ... 100
   ```
2. *Écrire une fonction qui prend un entier* n *en paramètre et renvoie* 1 *si* n *apparaît dans la table de multiplication, et 0 sinon.*

□

Exercice 11 (Carrés, ⋆⋆)

1. *Écrire une fonction qui prend en paramètre un entier positif* n *et affiche un carré d'étoiles plein de côté* n*. Par exemple, si* n *vaut* 4*, votre fonction affichera*
   ```
   ****
   ****
   ****
   ****
   ```
2. *Modifier votre fonction pour qu'elle affiche un carré creux, par exemple pour* n *valant* 4*,*
   ```
   ****
   *  *
   *  *
   ****
   ```

□

Exercice 12 (Séries, $\star$ – $\star\star\star$)
Pour chacune des séries suivantes, trouver le plus petit programme (en nombre d'appels de fonctions) qui affiche les séries de 20 nombres suivants à l'écran :

- *0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 ($\star$)*
- *1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 ($\star$)*
- *0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 ($\star\star\star$)*
- *0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 ($\star\star$)*
- *0 2 4 6 8 9 11 13 15 17 18 20 22 24 26 27 29 31 33 35 ($\star\star\star$)*
- *0 4 2 2 1 2 3 3 4 12 5 17 9 2 1 3 4 19 3 8 ($\star$)*

□

Exercice 13 (Sous-chaîne, $\star\star$)
Écrire une procédure `sousChaine` qui prend en paramètre deux chaînes de caractères `s` et `ss` non vides, et affiche toutes les positions de `s` à partir desquelles `ss` apparaît comme sous-chaîne.

Contrat:
(s,ss) = ("abracadabra","abra") $\rightarrow$ *affichage : 0 7*

□

Introduction à la Programmation 1 JAVA

51AE011F

Séance 4 de cours/TD

Université Paris-Diderot

Objectifs:

- Déclarer et initialiser un tableau de type int.
- Écrire des fonctions qui attendent des tableaux en entrée et produisent des tableaux en sortie.

1 Les tableaux d'entiers

Tableaux d'entiers [COURS]

- Un tableau est une suite de cases contenant chacune une valeur.
- Un tableau d'entiers contenant 2 dans sa première case, 4 dans sa deuxième case et 12 dans sa troisième case sera noté $\{2, 4, 12\}$.
- Les tableaux d'entiers ont le type « `int[]` ».
- Si `T` est un type alors « `T[]` » est le type des tableaux dont les valeurs sont de type `T`.
- Comme toute valeur, un tableau peut être stocké dans une variable. De même, une fonction peut prendre un tableau en paramètre et renvoyer un tableau. Ainsi, si une fonction renvoie un tableau d'entiers, son type de retour est `int[]`.
- Le nombre de cases d'un tableau `t` est appelé sa longueur et s'obtient en écrivant `t.length`.
- Si l est la longueur d'un tableau alors les cases du tableau sont numérotées de 0 à $l - 1$.
  Ces numéros sont appelés *indices*.
- Pour lire la valeur de la première case d'un tableau `t`, on écrit « `t[0]` ».
  Plus généralement, pour lire la case d'indice i, on utilise l'expression « `t[i]` ».
- Pour modifier la valeur de la case d'indice i d'un tableau `t`, on utilise l'instruction « `t[i] = e;` » où `e` est une expression qui s'évalue en une valeur du type des cases de `t`.
- **Attention** à ne jamais lire ou écrire hors des limites du tableau car cela produit une erreur. Ainsi, les indices négatifs ou plus grand que la longueur du tableau, ou égales à la longueur du tableau, sont à éviter.

Exercice 1 (Comprendre la taille et les indices, $\star$)

1. *Si un tableau* `a` *vaut* $\{1, 3, 5, 7, 9, 11, 13, 15, 17\}$. *Quelle est sa taille ? À quel indice trouve-t-on la valeur* 1 *? À quel indice trouve-t-on la valeur* 17 *? À quel indice trouve-t-on la valeur* 9 *?*
2. *Si le tableau* `b` *vaut* $\{2, 2, 3, 3, 4, 5, 7\}$. *Quelle est sa taille ? Que vaut l'expression* `b[0]` *? Que vaut l'expression* `b[4]` *? Pourquoi ne peut-on pas évaluer* `b[7]` *?*

□

Exercice 2 (Lecture, $\star$)
La déclaration suivante introduit un tableau `t` *dont les cases sont de type* `int` *et qui vaut* $\{2, 3, 4, 5, 6, 7\}$.

```java
int [] t = { 2, 3, 4, 5, 6, 7 };
```

1. *Donner une instruction modifiant la deuxième case du tableau* `t` *pour y mettre la valeur* 10.
2. *Donner les instructions permettant d'afficher la première case du tableau* `t` *et la dernière case du tableau* `t`.
3. *Quel est le contenu du tableau* `t` *après exécution des instructions suivantes :*

```java
int[] t = { 2, 3, 4, 5, 6, 7 };
for (int i = 0; i < t.length; i++) {
    t[i] = i;
}
```

□

Exercice 3 (Swap, ⋆)
On considère la déclaration suivante :

```java
int[] a = { 12, -3, 22, 55, 9 };
```

1. *Donner une suite d'instructions permettant d'inverser les contenus de la première et la deuxième case du tableau* `a`.
2. *Donner une suite d'instructions permettant d'inverser les contenus de la première et de la dernière case du tableau* `a`.

□

2 Déclarer un tableau

Déclaration d'un tableau [Cours]

— Pour déclarer qu'une variable `t` est un tableau, on la déclare avec son type de la façon suivante « `int[] t = e;` » où `e` est l'une des formes suivantes :
  — Une valeur d'initialisation pour le tableau.
    Par exemple, la déclaration « `int [] t = { 2, 6, 7 };` » introduit une variable de type tableau d'entiers valant $\{2, 6, 7\}$.
  — Une expression de création d'un tableau non initialisé, c'est-à-dire dont le contenu des cases n'est pas précisé. Dans ce cas, cette expression ne spécifie que la longueur du tableau et le type de ses cases.
    Par exemple, la déclaration « `int [] t = new int[5];` » crée un tableau `t` de taille 5, mais les valeurs que l'on trouve dans chaque case ne sont pas spécifiées. (En théorie, cela peut être n'importe quelle valeur entière.)
— Après avoir créé un tableau avec l'opérateur `new`, il faut en initialiser les cases.
  Par exemple, pour déclarer et initialiser correctement un tableau `a` pour qu'il vaille $\{-2, -1, 0, 1\}$, on peut procéder ainsi :

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

—

Exercice 4 (Création de tableau, $\star$)

1. *Donner les instructions pour créer un tableau* `a` *valant* $\{1, 2, 3, 4, 5, \ldots, 1000\}$.
   (Il est bien sûr recommandé d'utiliser une boucle `for` *pour remplir un tel tableau.)*

□

Exercice 5 (Manipulation de tableaux, $\star$)
Donner les instructions réalisant les opérations suivantes :

1. *Créer un tableau* `t` *valant* $\{2, 4, 6, 8, 10, 12, 14, 16\}$.
2. *Créer un tableau* `s` *de la même taille que* `t` *qui contient à chaque indice le double de la valeur contenue dans* `t` *à l'indice correspondant.*

□

Exercice 6 (Indices, $\star$)
Créer un tableau `a` *de longueur* 10 *et dont chaque élément se trouvant à l'indice* i *a pour valeur* $2 * i$. □

Exercice 7 (Tableau d'éléments non nuls, $\star\star$)

1. *Que fait la fonction «* `int notZero(int [] t)` *» dont le code vous est donné ci-dessous :*

```java
public static int notZero (int[] t) {
    int s = 0;
    for (int i = 0; i < t.length; i++) {
        if (t[i] != 0) {
            s = s + 1;
        }
    }
    return s;
}
```

2. *Donner une séquence d'instructions réalisant les actions suivantes :*
   (a) *Création d'un tableau* `t` *valant :* $\{4, 3, 6, 2, 0, 0, 2, 0, 4\}$.
   (b) *Création d'un tableau* `s` *contenant les éléments du tableau* `t` *qui sont différents de zéro (dans le même ordre). Pour cela vous utiliserez la fonction* `notZero`.

□

3 Fonctions utilisées

```java
/*
 * Affiche le contenu d'un tableau d'entiers
 * t est le tableau que l'on souhaite afficher
 */
public static void printIntArray (int[] t) {
    for (int i = 0; i < t.length; i++) {
        System.out.print(t[i] + " ");
    }
    System.out.print ("\n");
}
```

4 Do it yourself

Exercice 8 (Somme, ⋆)
Écrire une fonction « `int sumIntArray (int [] t)` » qui calcule la somme des éléments du tableau `t` *donné en paramètre.* □

Exercice 9 (Moyenne, ⋆)

Écrire une suite d'instructions qui crée un tableau d'entiers `t` *valant* `{ 3, 5, 2, 3, 6, 3, 4}`*, qui calcule la moyenne entière des éléments de* `t` *et qui affiche cette moyenne.*

□

Exercice 10 (Grognements, ⋆)

Écrire une fonction qui prend un paramètre entier `n` *et affiche les* `n` *premiers grognements* `brhh`, `brrhh`, `brrrhh`, *etc.*

□

Exercice 11 (Dernière occurrence, ⋆⋆)

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.

□

Exercice 12 (Première occurence, ⋆⋆)

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

Indice : on pourra tester si le résultat à renvoyer est différent de -1 pour comprendre si l'on a déjà vu la valeur dans le tableau.

□

Exercice 13 (Pyramide, ⋆⋆⋆)

Écrire une fonction qui prend en entrée un entier n *et affiche une pyramide d'étoiles de hauteur* n*. Si* n *est négatif, la pyramide devra être inversée. Par exemple, sur entrée* $n = 4$*, afficher :*

```
   *
  * *
 * * *
* * * *
```

Sur entrée $n = -3$*, afficher :*

```
* * *
 * *
  *
```

□

Exercice 14 (Suite dans un tableau, ⋆⋆)

Écrire une fonction « `int[] progression (int a, int b, int n)` » qui prend en paramètre trois entiers `a, b` et `n`, et qui renvoie un tableau de taille `n` contenant les entiers `a, a + b, a + 2b, a + 3b, ..., a + (n-1)b`.

□

Exercice 15 (Entrelace, ⋆⋆)

Écrire une fonction « `int [] interlace(int[] t1, int[] t2)` » qui prend en entrée deux tableaux `t1` et `t2` de même taille et renvoie un tableau de taille double qui contient les valeurs des tableaux de façon entrelacée, c'est-à-dire `{t1[0], t2[0], t1[1], t2[1], ..., t1[n], t2[n]}`. Par exemple, appliquée à `{ 0, 1, 6 }` et `{ 2, 4, 7 }`, elle va renvoyer le tableau `{ 0, 2, 1, 4, 6, 7}`.

□

Exercice 16 (Plagiat, ⋆⋆)

- *Écrire une fonction* `plagiarism` *prenant en paramètre deux tableaux* `t` *et* `s` *et qui renvoie l'indice d'une valeur de* `t` *qui apparaît dans* `s`*. Si une telle valeur n'existe pas, la fonction renvoie* -1*.*
- *Que se passe-t-il si on appelle la fonction* `plagiarism` *sur le même tableau* `plagiarism (s, s)` *?*
- *Écrire une fonction* `autoPlagiarism` *prenant en paramètre un tableau* `s` *et qui renvoie l'indice d'une valeur de* `s` *qui y apparaît deux fois. Si une telle valeur n'existe pas, la fonction renvoie* -1*.*

□

Exercice 17 (Récurrence, ⋆⋆⋆)

Soient U et V deux tableaux d'entiers à une dimension, de même taille arbitraire k. On considère l'équation de récurrence suivante :

$$u_i = U[i] \text{ pour } 0 \leq i < k \quad \text{et} \quad u_{n+1} = \sum_{i=0}^{k-1} V[i]u_{n-i} \text{ pour } n \geq k-1.$$

Écrire une fonction qui prend en paramètre deux tableaux d'entiers U et V (qu'on supposera de même taille) et un entier n, et qui renvoie la valeur de u_n définie ci-dessus. Attention, on ne connaît pas à l'avance la taille des tableaux. □

Exercice 18 (Mastermind, ⋆⋆⋆)

Dans cet exercice, on se propose d'écrire un programme permettant de jouer au Mastermind©.

On rappelle le principe de ce jeu. Un joueur (ici, la machine) choisit une **combinaison secrète** *de 5 couleurs parmi 8 couleurs disponibles (***on représentera durant tout l'exercice les couleurs par les entiers 0, 1, 2, 3, 4, 5, 6, 7***). Une couleur peut apparaître plusieurs fois dans une combinaison. L'adversaire de ce joueur a 10 tentatives pour deviner cette combinaison sachant qu'après chaque proposition, il bénéficie d'une indication : le premier joueur examine sa proposition et lui dit combien de pions colorés sont bien placés et combien sont présents dans la combinaison secrète, mais mal placés.*

Ainsi dans l'exemple suivant :

combinaison secrète	$\{4, 2, 6, 0, 5\}$
combinaison proposée	$\{1, 6, 4, 0, 5\}$

le joueur qui devine se verra répondre "2 pion(s) bien placé(s) et 2 pions mal placé(s)", correspondant aux couleurs 0 et 5 pour les bien placées, et aux couleurs 6 et 4 pour les mal placées.

Voici un deuxième exemple pour bien fixer les idées :

combinaison secrète	$[1, 1, 6, 0, 3]$
combinaison proposée	$[1, 6, 4, 1, 1]$

le joueur qui devine se verra répondre "1 pion(s) bien placé(s) et 2 pion(s) mal placé(s)", correspondant à la couleur 1 en position 0 bien placée et les couleurs 1 (position 3 ou 4) et 6 (position 1).

- *Écrire une fonction « `public static int wp (int[] prop, int[] sol)` » qui renvoie le nombre de pions bien placées, si `sol` représente la solution et `prop` la proposition.*
- *Écrire une fonction « `public static int[] count (int[] t)` » qui renvoie un tableau de taille 8 dont la i-ème case contient le nombre d'occurrences de i dans le tableau `t`.*
- *On veut désormais écrire une fonction permettant de compter le nombre de pions mal placés en se basant sur l'observation suivante. Pour une couleur donnée, si elle apparaît x fois dans la proposition et y fois dans la solution secrète, alors le nombre de pions mal placés de cette couleur plus le nombre de bien placés de cette couleur est égal au minimum de x et de y.*
  En déduire une fonction « `public static int bp(int[] prop, int[] sol)` » qui renvoie le nombre de couleurs mal placées. On pourra écrire une fonction « `int minInt (int x, int y)` » qui renvoie le minimum des deux entiers `x` et `y` pris en paramètre.
- *On suppose que l'on a à notre disposition une fonction « `public static int[] randSol ()` » qui renvoie un tableau aléatoire de taille 5 constitué de valeurs de 0 à 7.*
  Écrire maintenant un programme qui simule le jeu, c'est-à-dire qu'il laisse à l'utilisateur dix tentatives pour trouver la solution en lui donnant après chaque tentative le nombre de bien placées et de mal placées. Si le joueur a trouvé la bonne solution, la boucle doit s'arrêter en affichant un message de victoire, et si le joueur échoue après dix tentatives, le programme doit afficher un message d'échec en lui donnant la combinaison secrète. A chaque tour, l'utilisateur entrera sa combinaison en tapant les 5 chiffres successivement. Pour cela, on supposera donnée une fonction « `public static int inputInt ()` » qui permet à l'utilisateur de saisir un entier au clavier.

□

Exercice 19 (Réordonnancement, ⋆⋆)

Écrire une fonction qui demande un entier n à l'utilisateur, puis lui demande n fois d'entrer un entier x ainsi qu'une position $j \in \{1, \ldots, n\}$ (pour l'affichage), et qui affiche ensuite les n entiers selon l'ordre spécifié. Ainsi, si l'utilisateur entre

```
4
10
3
12
2
15
4
16
1
```

(la première ligne correspondant au nombre d'entiers à entrer, puis les lignes paires sont les entiers eux-mêmes et les lignes impaires leur position), il souhaite afficher l'entier 10 en position 3, l'entier 12 en position 2, 15 en position 4 et 16 en première position. Le programme devra alors afficher :

16 12 10 15

Pour faire cela on pourra utiliser une fonction « `public static int inputInt ()` » qui bloque tant que l'utilisateur n'a pas rentré de données au clavier et tapé sur la touche Entrée et qui renvoie alors l'entier tapé au clavier par l'utilisateur. □

Introduction à la Programmation 1 JAVA

51AE011F

Séance 5 de cours/TD

Université Paris-Diderot

Objectifs:

- 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!
$
```

□