

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 number0fSecs = stopSecs - startSecs;
4      int alertCode = 0;
5      for (int i = 0; i < number0fSecs; i++) {
6          waitUntilNextSecond ();
7          if (number0fSecs - i < 30) {
8              alertCode = 1;
9          }
10         drawInt (number0fSecs - 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 ($0, 1, 2, -1, -2, -2147483648, 2147483647 \dots$);
 - (b) deux expressions séparées par une opération arithmétique binaire ($1 + 2, 1 + 2 * 3 / 4, \dots$);
 - (c) une expression entourée de parenthèses ($(1 + 2), (1 + 2 * 3 / 4), \dots$);
 - (d) une expression précédée du signe moins ($-2, -(1 + 2), \dots$).
- 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 $(+, -, /, *, \dots)$ 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 :

```
1 6 * 7 + 3
2 6 * (7 + 3)
3 45 / 7
4 3 * 7 / 4
5 (3 * 7) / 4
6 (45 / 7) * 7 + 45 % 7
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 :

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

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

```
1 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)` ?

```
1 public static int twice (int x) {  
2     return (2 * x);  
3 }
```

□

Exercice 8 (Utiliser une fonction, **)

On considère la fonction suivante.

```
1 public static int cube (int x) {  
2     return (x * x * x);  
3 }
```

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

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

□

Exercice 9 (Écrire une fonction, **)

Voici une fonction

```
1 public static int f(int x) {  
2     return (x * x + 5);  
3 }
```

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 :

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

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

```
1 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 :

```
1 public static void showProductAndSum (int x, int y) {  
2     System.out.println (x + y);  
3     System.out.println (x * y);  
4 }
```

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

```
1 int x = 10;  
2 int y = 42;  
3 if (x <= 0) {  
4     y = x;  
5 } else {  
6     y = -x;  
7 }
```

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` ?

```

1 int x = 3;
2 int y = 4;
3 int max = 0;
4 if (x > y) {
5     max = x;
6 } else {
7     max = y;
8 }
```

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 :

```

1 int a = 2;
2 int b = a * a + 3;
3 int c = b - a;
4 if (c == a) {
5     a = 1;
6 } else {
7     a = a + 3;
8 }
9 if (b + c < a) {
10    b = 2;
11 } else {
12    b = 4;
13 }
14 if (b != c * c) {
15    c = 12;
16 } else {
17    c = -6;
18 }
```



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 :

```

1 for (int i = 0; i <= 9; i++) {
2     System.out.println (i);
3 }
```

- 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 :

```

1 for (int i = 0; i < 10; i++) {
2     System.out.println (i);
3 }
```

- 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.

```

1 public static int sumsquare (int x, int y) {
2     return (x * x + y * y);
3 }
```

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

```

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

□

Exercice 19 (Boucles simples, **)

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

```
1 for (int i = 0; i < 5; i++) {  
2     System.out.println (8);  
3 }
```

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

```
1 int x = 0;  
2 for (int i = 0; i < 5; i++) {  
3     x = x + 8;  
4 }
```

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

```
1 int x = 0;  
2 for (int i = 0; i < 5; i++) {  
3     x = 10 * x + 8;  
4 }
```

□

Exercice 20 (Des entiers qui s'ajoutent, **)

Quelle est la valeur de *sum* après la suite d'instructions suivante :

```
1 int i = 0;  
2 int sum = 0;  
3 sum = sum + i;  
4 i = i + 1;  
5 sum = sum + i;  
6 i = i + 1;  
7 sum = sum + i;  
8 i = i + 1;  
9 sum = sum + i;  
10 i = i + 1;  
11 sum = sum + i;  
12 i = i + 1;  
13 sum = sum + i;  
14 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 :

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

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 des types `String` et `char`.
- 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";
```



Définition d'une chaîne de caractères

[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 Manipulation de chaînes de caractères

Chaînes de caractères et caractères

[COURS]

- Une chaîne de caractère a une longueur qui correspond à son nombre de caractères (en comptant les espaces et les caractères spéciaux comme '\n').
- Pour avoir la longueur d'une chaîne de caractères stockée dans une variable s, on écrira s.length().
- Par exemple, après la suite d'instructions suivantes :

```
1 String st = "Bonjour";
2 String st2 = st + " Bob!";
3 int le = st.length();
4 int le2 = st2.length();
```

la valeur stockée dans la variable le est 7 et celle stockée dans la variable le2 est 12.

- La longueur de la chaîne de caractères "" est 0.
- Une chaîne de caractères est donc composée de caractères. Il y a aussi un type qui correspond aux caractères, c'est le type **char**.
- Un caractère est compris entre guillemets simple.
- Exemples de caractères : les lettres minuscules et majuscules (par ex. 'a', 'D'), les chiffres (par ex. '0', '8'), les signes de ponctuation (par ex. '.', '?', '!') ou encore les caractères spéciaux (par ex. '\n').
- Attention, par définition une donnée de type **char** ne contient qu'un seul caractère ! On ne peut pas écrire 'sy' par exemple.
- Les caractères d'une chaîne de caractères sont numérotés de 0 à n-1 où n est la longueur de la chaîne. Pour obtenir le (i+1)-ème caractère d'une chaîne de caractères stockée dans une variable st, on peut faire st.charAt(i). Attention il faut que i soit supérieur ou égal à 0 et strictement inférieur à la longueur de la chaîne contenue dans st.
- Par exemple, après la suite d'instructions suivantes :

```
1 String ch= "Hello!";
2 char c1=ch.charAt(0);
3 char c2=ch.charAt(2);
4 char c3=ch.charAt(ch.length()-1);
```

le caractère stocké dans la variable c1 est 'H', celui stocké dans la variable c2 est 'l' et celui stocké dans la variable c3 est '!'.

- On peut ajouter un caractère à la suite ou au début d'une chaîne de caractères en utilisant le symbole de concaténation +. Par exemple "Salu"+'t' donnera la chaîne "Salut". De même 'S'+"alut" donnera aussi la chaîne "Salut".
- Mais on NE peut PAS concaténer deux caractères. Ainsi, l'instruction 'a'+'b' ne produira pas la chaîne "ab".

Exercice 2 (Manipulation de chaînes, *)

1. Quelle est la valeur de la variable a après les instructions suivantes ?

```
1 String ch = "Ceci est ";
2 String ch2 = "une chaine";
3 ch = ch + ch2;
4 int a = ch.length();
```

2. Que vaut la variable cha dans l'exemple suivant ?

```
1 String ch = "Avec consonnes";
2 String cha = " " + ch.charAt(0);
3 cha = cha + ch.charAt(2);
```

```

4 cha = cha + ch.charAt(6);
5 cha = cha + ch.charAt(9);
6 cha = cha + ch.charAt(12);

```

□

3 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 3 (Des exemples simples avec des variables de différents type, *)

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

```

1 int x = 3;
2 String ch = "Salut";
3 if (x <= 2) {
4     ch = "Hallo";
5 } else {
6     ch = "Hi";
7 }

```

2. Que fait la suite d'instructions suivantes ?

```

1 int x = 3;
2 String st = "Ca va ?";
3 String ch = "";
4 x = x / 2;
5 if (x == 1) {
6     ch = "How are you ?";
7 } else {
8     ch = "Comment allez-vous ?";
9 }
10 ch = ch + "\n";
11 System.out.print (ch);

```

□

Exercice 4 (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 5 (Trouver le bon type, ***)

Dans les fonctions suivantes, remplacez les [...] par les bons types :

```

1  public static [...] f ([...] x) {
2      [...] y = x % 2 ;
3      if (y == 0) {
4          System.out.println ("Argument pair");
5      } else {
6          System.out.println ("Argument impair");
7      }
8  }
9 }

10 public static [...] g ([...] x) {
11     [...] y = x % 2 ;
12     if (y == 0) {
13         return "Argument pair";
14     } else {
15         return "Argument impair";
16     }
17 }
18 }

19 public static [...] h ( [...] x, [...] y) {
20     return (x + y);
21 }

22 }

23 public static [...] id ( [...] x) {
24     return x;
25 }

```

code/ExoInference.java



4 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 :

```

1  public static String sandwich (String s, String s2) {
2      return s + s2 + s;
3  }

4

5  public static String neymarOrNot (int x) {
6      String st = "";
7      if (x == 10) {
8          st = "C'est le numéro de Neymar.";
9      } else {
10         st = "Ce n'est pas le numéro de Neymar.";
11     }
12     return st;
13 }

```

Elles peuvent être utilisées de la façon suivante :

```
1 String s = sandwich ("Hello", "World!\n");
```

```
2 String s2 = sandwich (s, "Ca va ?");
3 String s3 = neymarOrNot (1);
```

Exercice 6 (Utilisation de fonctions données, *)

On considère la fonction suivante :

```
1 public static String addA (String ch) {
2     return (ch + "A");
3 }
```

Que vaut la variable st après exécution du code suivant ?

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

□

Conversion de chaînes de caractères en entier et vice-versa _____ [COURS]

- Il existe une fonction de Java pour transformer un entier de type `int` en une chaîne de caractères représentant cet entier. Cette fonction est “`String String.valueOf(int x)`”. Par exemple, `String.valueOf(123)` renverra la chaîne de caractères “`123`”.
- De même on peut transformer une chaîne de caractères représentant un entier en un entier grâce à la fonction “`int Integer.valueOf(String ch)`”. Par exemple, `Integer.valueOf("1290")` renverra l’entier 1290.
- ATTENTION : si l’on donne en argument à la fonction “`int Integer.valueOf(String ch)`” une chaîne de caractères ne correspondant pas à la représentation d’un entier alors il y aura une erreur à l’exécution du programme ! C’est ce qui se passera par exemple avec l’instruction `Integer.valueOf("Boum")`.

Exercice 7 (Modification d’une fonction, **)

On considère la fonction suivante.

```
1 public static String message (int x) {
2     String ch = String.valueOf(x);
3     return ch;
4 }
```

Et on considère la liste d’instructions suivante :

```
1 int a = 4;
2 int b = a * a;
3 a = b - a;
4 String s = message (a);
5 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).

□

5 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`"

Exercice 8 (Savoir évaluer une condition, *)

Quelle est la valeur de la variable `x` après la suite d'instructions suivante ?

```
1 int a = 2;
2 a = a * a * a * a + 1;
3 int b = a / 2;
4 int c = a - 1;
5 int x = 0;
6 if ((b == 0) && (c == 16)) {
7     x = 1;
8 } else {
9     x = 2;
10 }
```

□

Exercice 9 (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 10 (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)`

Tests sur les chaînes de caractères

[COURS]

- Attention, on ne peut pas tester l'égalité de deux chaînes de caractères avec `==`. Pour faire cela, si l'on a deux variables `st1` et `st2` stockant des chaînes de caractères et si l'on veut tester que les chaînes sont égales, on écrira `st1.equals(st2)` (ou de façon équivalente `st2.equals(st1)`). Ces expressions seront égales à `true` si les deux variables contiennent les même chaînes et à `false` sinon. On peut également utiliser directement des chaînes de caractères à la place de ces variables, ainsi si l'on veut tester que la variable `st1` contient la chaîne de caractères "`Badaboum`", on pourra écrire `st1.equals("Badaboum")`.
- Voici un exemple :

```

1 public static void profiler (String name) {
2     if (name.equals("Paul") || name.equals( "Pierre")) {
3         System.out.println ("On a trouve Paul ou Pierre.");
4     } else {
5         System.out.println ("Ce n'est ni Paul, ni Pierre.");
6     }
7 }
```

- En revanche, on peut tester l'égalité de deux caractères avec `==`.

Exercice 11 (Conditions sur chaîne de caractères, *)*Quelle est la valeur de la variable z après la suite d'instructions suivante ?*

```

1 int z = 0;
2 String st1 = "Hello";
3 String st2 = "Hallo";
4 if(st1.equals(st2) || (st1.charAt(0)==st2.charAt(0) && st1.charAt(4)==
5     st2.charAt(4))){
6     z=1;
7 }else{
8     z=-1;
}
```

6 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 :

```

1 int x = 1;
2 int y = 2;
3 int z = 0;
4 if (x == 1) {
5     if (y == 2) {
6         z = 3;
7     } else {
8         z = 5;
9     }
10 } else {
```

```

11     z = 7;
12 }
```

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 12 (Utilisation des opérateurs booléens, **)

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

1.

```

1 public static void f (int l) {
2     int x = 0;
3     if (l != 0) {
4         if (l <= 10) {
5             x = 1;
6         } else {
7             x = 2;
8         }
9     } else {
10        x = 2;
11    }
12 }
```

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.

```

1 public static void g (int a) {
2     int b = 2 * a;
3     b = a - a;
4     if (b == a) {
5         b = 0;
6     } else {
7         if (b > 43) {
8             b = 0;
9         } else {
10            b = 1;
11        }
12    }
13 }
```

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.

```

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

```

8         } else {
9             e = 3;
10            }
11        }
12    }

```

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



7 Expressions conditionnelles

Expressions conditionnelles

[COURS]

- En utilisant l'opérateur ternaire ?, on peut créer des expressions dont la valeur dépend d'une condition.
- La syntaxe est alors la suivante : condition ? expression1 : expression2 et la valeur de l'expression est expression1 si condition est évaluée à vraie et expression2 si condition est évaluée à fausse.
- Par exemple, le code suivant :

```

1 int n = 25;
2 String ch = "";
3 if(n%2 == 0) {
4     ch = "Le nombre est pair";
5 }
6 else {
7     ch = "Le nombre est impair";
8 }

```

peut se réécrire avec une expression conditionnelle de la façon suivante :

```

1 int n = 25;
2 String ch = (n%2 == 0) ? "Le nombre est pair" : "Le nombre est
   impair";

```

Exercice 13 (Fonctions avec expression conditionnelle, *)

1. On considère la fonction suivante :

```

1 public static int fonz (int x) {
2     int res= (x>=0) ? x : -x;
3     return res;
4 }

```

Que renvoient fonz(-5) et fonz(3) ? Que calcule donc la fonction fonz ?

2. On considère la fonction suivante :

```

1 public static String fonz2 (String st) {
2     String res= (st.equals("Alice") || st.equals("Bob")) ? "
   Utilisateur connu" : "Utilisateur inconnu";
3     return res;
4 }

```

Que renvoient fonz2("John") et fonz2("Alice") ?



8 Boucles

Exercice 14 (Pendu à l'envers, **)

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

Contrat:

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

- 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.

```
1 for (int i = 0; i < 100; i++) {  
2     if (i % 2 == 0) {  
3         System.out.println ("Pair.");  
4     } else {  
5         System.out.println ("Impair.");  
6     }  
7 }
```

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 15 (Effacement des espaces, *)

Écrire une fonction qui prend en argument une chaîne de caractères et affiche cette même chaîne sans les espaces.

Contrat:

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

□

9 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 16 (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 17 (Concaténation et somme, **)

Que valent les variables `x` et `s` après les instructions suivantes ?

```
1 int x = 0;
2 String s = "";
3 for (int n = 0; n < 10; n++) {
4     x = x + 1;
5     s = s + "1";
6 }
```

□

Exercice 18 (Condition et division entière, *)

Écrire une condition permettant de tester si le tiers de l'entier `x` appartient à l'intervalle [2; 8].

Contrat:

$x=0$	→	<code>false</code>
$x=6$	→	<code>true</code>
$x=9$	→	<code>true</code>
$x=25$	→	<code>false</code>

□

Exercice 19 (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:

$(a,b,h)=(3,4,5)$	→	<code>true</code>
$(a,b,h)=(1,2,3)$	→	<code>false</code>

□

Exercice 20 (Somme des cubes, *)

Écrire une boucle permettant de calculer la somme des cubes des entiers de 1 à 100.

□

Exercice 21 (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:

<code>1</code>	→	<code>1st</code>
<code>12</code>	→	<code>12th</code>
<code>23</code>	→	<code>23rd</code>
<code>32</code>	→	<code>32nd</code>
<code>44</code>	→	<code>44th</code>

□

Exercice 22 (Simplification de code, *)

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

```
1 public static int f (int x, int y) {
2     if (((x <= y) || (x >= y + 1))) {
3         if ((x <= y) && (x < y)) {
```

```

4         if (x < -x) {
5             int a = -x;
6             return a;
7         } else {
8             return x;
9         }
10        } else {
11            if (y * y * y < 0) {
12                return -y;
13            }
14        }
15    }
16    return y;
17 }
```

2. Écrire la même fonction de manière plus simple.

□

Exercice 23 (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 24 (Lignes et pointillés, **)

- 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 :

- 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 25 (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 $6 = 1 + 2 + 3$.

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

Contrat:

$n = 1$	\rightarrow	affichage :
$n = 5$	\rightarrow	affichage : 1
$n = 60$	\rightarrow	affichage : 1 2 3 4 5 6 10 12 15 20 30

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

□

Exercice 26 (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:

$(s,c) = ("abracadabra", "a") \rightarrow \text{affichage} : 0\ 3\ 5\ 7\ 10$

□

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 :

```
1 int s = 0;
2 for(int i = 1; i <= 100; i++) {
3     s = s + i * i;
4 }
```

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, *)

```
1 String st = "";
2 for (int i = 1; i <= 50; i++) {
3     st = st + "ab";
4 }
```

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 :

```

1 public static int readInt() {
2     Scanner sc = new Scanner (System.in);
3     return sc.nextInt ();
4 }
```

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* :

```

1 for (int i = 0; i < 10; i++) {
2     for (int j = 1; j <= 5; j++) {
3         System.out.println(i);
4         System.out.println(j);
5     }
6 }
```

- 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, *)

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, **)

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 “`boolean isPrime (int n)`” qui renvoie `true` si n est un nombre premier, `false` 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, * - **)**

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 (*)

— 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 (*)

— 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 (***)
— 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 (**)
— 0 2 4 6 8 9 11 13 15 17 18 20 22 24 26 27 29 31 33 35 (***)
— 0 4 2 2 1 2 3 3 4 12 5 17 9 2 1 3 4 19 3 8 (*)

□

Exercice 13 (Sous-chaîne, **)

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:

$(s,ss) = ("abracadabra", "abra") \rightarrow \text{affichage : } 0\ 7$

□

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, *)

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, *)

La déclaration suivante introduit un tableau `t` dont les cases sont de type `int` et qui vaut {2, 3, 4, 5, 6, 7}.

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

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

```

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

□

Exercice 3 (Swap, *)

On considère la déclaration suivante :

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

- Donner une suite d'instructions permettant d'inverser les contenus de la première et la deuxième case du tableau a.*
- 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 :

```

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

- **Attention :** les valeurs d'initialisation (comme par exemple { 2, 6, 7 }) ne sont pas des expressions. On ne peut les utiliser que dans les déclarations de variables et non dans les expressions.
Par exemple, on ne peut pas écrire l'affectation « `t = { 2, 6, 7 };` ».

Exercice 4 (Création de tableau, *)

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

□

Exercice 5 (Manipulation de tableaux, *)

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, *)

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, **)

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

```
1 public static int notZero (int[] t) {  
2     int s = 0;  
3     for (int i = 0; i < t.length; i++) {  
4         if (t[i] != 0) {  
5             s = s + 1;  
6         }  
7     }  
8     return s;  
9 }
```

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

```
1 /*  
2  * Affiche le contenu d'un tableau d'entiers  
3  * t est le tableau que l'on souhaite afficher  
4  */  
5 public static void printIntArray (int[] t) {  
6     for (int i = 0; i < t.length; i++) {  
7         System.out.print(t[i] + " ");  
8     }  
9     System.out.print ("\n");  
10 }
```

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 (Groggnements, *)

Écrire une fonction qui prend un paramètre entier `n` et affiche les `n` premiers grognements `brhh`, `brrhh`, `brrrh`, 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 occurrence, **)

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 $\{t1[0], t2[0], t1[1], t2[1], \dots, t1[n], t2[n]\}$. 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 :

<i>combinaison secrète</i>	$\{4, 2, 6, 0, 5\}$
<i>combinaison proposée</i>	$\{1, 6, 4, 0, 5\}$

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 un tableau de taille 8 dont la i -ème case contient le nombre d'occurrences de i 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 $j \in \{1, \dots, n\}$ (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:

- Connaître ses classiques sur les tableaux.
- Comprendre le statut particulier des tableaux en mémoire.

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`.

```
1 public static boolean isPresent(int el, int [] tab){  
2     for(int i=0; i<tab.length; i=i+1){  
3         if(tab[i]==el){  
4             return true;  
5         } else{  
6             return false;  
7         }  
8     }  
9     return false;  
10 }  
11  
12 public static boolean isPresent(int el, int [] tab){  
13     boolean b = true;  
14     for(int i=0; i<tab.length; i=i+1){  
15         if(tab[i]==el){  
16             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 (Occurrences 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){

```

```

14         if(el==tab[i]){
15             return i;
16         }
17     }
18     return i;
19 }
20
21 public static int occ(int el,int[] tab){
22     int p = 0;
23     for(int i=0; i<tab.length; i=i+1){
24         if (el==tab[i]){
25             p = i;
26         } else {
27             p = -1;
28         }
29     }
30     return p;
31 }
32
33 public static int occ(int el,int[] tab){
34     int p = 0;
35     for(int i=0; i<tab.length; i=i+1){
36         if(el==tab[i]){
37             p = i;
38         }
39     }
40     return p;
41 }
42
43 public static int occ(int el,int[] tab){
44     int p = -1;
45     int i = 0;
46     while(i<tab.length){
47         if(el==tab[i]){
48             p = i;
49             i = i+1;
50         }
51     }
52     return p;
53 }
54
55 public static int occ(int el,int[] tab){
56     int p = -1;
57     int i = 0;
58     while(i<tab.length && p!=-1){
59         if(el==tab[i]){
60             p = i;
61         }
62         i = i+1;
63     }
64     return p;
65 }

```

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`.

```
1 public static int count(int el, int[] tab){  
2     int c = 0;  
3     for(int i=0; i<tab.length; i=i+1){  
4         if(el==tab[i]) {  
5             c = c+1;  
6             return c;  
7         }  
8     }  
9     return c;  
10}  
11  
12 public static int count(int el, int[] tab){  
13     int c = 0;  
14     for(int i=0; i<tab.length; i=i+1){  
15         if(el==tab[i]) {  
16             c = c+1;  
17         } else {  
18             c = c-1;  
19         }  
20     }  
21     return c;  
22}
```

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).

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

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

```

6         }
7     }
8     return tabRet;
9 }
10
11 public static int[] pos(int el,int[] tab){
12     int n = count(el, tab);
13     int []tabRet = new int[n];
14     for(int i=0; i<n; i=i+1){
15         tabRet[i] = 0;
16     }
17     for(int i=0; i<tab.length; i=i+1){
18         if(el==tab[i]){
19             tabRet[i] = i;
20         }
21     }
22     return tabRet;
23 }
24
25 public static int[] pos(int el,int[] tab){
26     int n = count(el, tab);
27     int []tabRet = new int[n];
28     for(int i=0; i<n; i=i+1){
29         tabRet[i] = 0;
30     }
31     for(int i=0; i<tab.length; i=i+1){
32         if(el==tab[i]){
33             tabRet[el] = i;
34         }
35     }
36     return tabRet;
37 }
38
39 public static int[] pos(int el,int[] tab){
40     int n = count(el, tab);
41     int []tabRet = new int[n];
42     for(int i=0; i<n; i=i+1){
43         tabRet[i] = 0;
44     }
45     int j = 0;
46     for(int i=0; i<tab.length; i=i+1){
47         if(el==tab[i]){
48             tabRet[j] = i;
49         }
50     }
51     return tabRet;
52 }

```

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.

□

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 :

$t1$	$t2$
$\$1$	$\$2$

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

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 \square 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
$\$1$

$$\$1 = \{\square, \square, \square, \square\}$$

- 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 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 dites ce qu'affiche le programme suivant :

```
1 public class ExoSwap1{  
2  
3     public static void modif(int[] t){  
4         t[0] = 5;  
5     }  
6  
7     public static void main(String[] args){  
8         int[] tab = {1, 2, 4, 8, 16, 31};  
9         modif(tab);  
10        for(int i=0; i< tab.length; i=i+1){  
11            System.out.println(tab[i]);  
12        }  
13    }  
14 }
```

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.

```

1 public static void swap(int i, int j, int[] tab){
2     if(i<tab.length && j<tab.length){
3         tab[i] = tab[j];
4         tab[j] = tab[i];
5     }
6 }
```

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.

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

```

1 public static int[] reverse(int[] tab){
2     for(int i=0; i<tab.length;i++){
3         tab[i] = tab[tab.length - 1 -i];
4     }
5     return tab;
6 }

7
8 public static int[] reverse(int[] tab){
9     for(int i=0; i<tab.length;i++){
10        int temp = tab[i];
11        tab[i] = tab[tab.length - 1 -i];
12        tab[tab.length - 1 -i] = temp;
13    }
14    return tab;
15 }

16
17 public static int[] reverse(int[] tab){
18     int[] t = {};
19     for(int i=0; i<tab.length;i++){
20         t[i] = tab[tab.length - 1 -i];
21     }
22     return t;
23 }
```

code/ExoReverse.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.

c Parmi les corrections proposées, quelles sont celles qui changent le tableau `tab` passé 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).

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

```

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

```

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

```

```

1 public static void shift(int[] tab, int p){
2     for(int i=0; i<tab.length;i=i+1){
3         tab[i] = tab[i-p];
4     }
5 }
6
7 public static void shift(int[] tab, int p){
8     for(int i=0; i<tab.length;i=i+1){
9         int temp = tab[i];
10        tab[i-p] = temp;
11    }
12 }
13
14 public static void shift(int[] tab, int p){
15     for(int i=0; i<tab.length;i=i+1){
16         tab[(i+p) % tab.length] = tab[i];
17     }
18 }

```

- 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.

□

3 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 (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 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 renournée 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:

$t1 = \{2, 4, 3, 5\}$ $t2 = \{4, 3, 6\}$ \rightarrow affichage : 2 8 7 1
 $t1 = \{6, 1, 2\}$ $t2 = \{6, 3, 0\}$ \rightarrow affichage : 1 2 4 2

□

Introduction à la Programmation 1 JAVA

51AE011F

Séance 6 de cours/TD

Université Paris-Diderot

Objectifs:

- Comprendre les tableaux de chaînes de caractères.
- Comprendre les tableaux de tableaux.

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 :

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

ou bien ainsi :

```
1 String [] theBeatles = new String [4];
2 theBeatles [0] = "Paul";
3 theBeatles [1] = "John";
4 theBeatles [2] = "Ringo";
5 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 ?

```
1 public static String[] funcAB (int a) {
2     String[] t = new String[a];
3     String s = "ab";
4     for (int i = 0; i < a; i++) {
5         t[i] = s;
6         s = s + "ab";
7     }
8     return t;
9 }
```

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

```
1 ab
2 abab
3 ababab
4 abababab
5 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.

□

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” :

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

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

```
1 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 :

```
1 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 :

```
1 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 } }` » :

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

```

6 tPascal[1][1] = 1;
7 tPascal[2] = new int[3];
8 tPascal[2][0] = 1;
9 tPascal[2][1] = 2;
10 tPascal[2][2] = 1;
11 tPascal[3] = new int[4];
12 tPascal[3][0] = 1;
13 tPascal[3][1] = 3;
14 tPascal[3][2] = 3;
15 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, *)

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, **)

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 reçue 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, ***)

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 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 6 (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 7 (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 8 (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

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

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 9 (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.

- 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.
- 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`.
- 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 10 (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 11 (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 -ème colonne est égal à :

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

- 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.
- Écrire une fonction `nbLines` qui prend en paramètre un tableau de tableaux d'entiers qui encode une matrice et renvoie son nombre de lignes.
- Écrire une fonction `nbColumns` qui prend en paramètre un tableau de tableaux d'entiers qui encode une matrice et renvoie son nombre de colonnes.
- 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 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 +/- 1 et 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...
xA..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...xx
ux.n.x
.nnx..
.uu...
x...xx
    Interruption!
$
```

□

Introduction à la Programmation 1 JAVA

51AE011F

Séance 7 de cours/TD

Université Paris-Diderot

Objectifs:

- L'instruction `switch`
- Boucles `while`.
- Variables booléennes.

1 L'instruction “switch”

L'instruction “switch” pour distinguer des valeurs _____ [COURS]

- L'instruction `switch` prend en paramètre une expression ayant pour type `int` ou `String` et réalise une instruction selon la valeur exacte de cette expression.
- C'est donc une façon facile de faire une énumération de cas.
- Par exemple :

```
1 int num = 3;
2 String jour = "";
3 switch( n ) {
4     case 1 : jour = "Lundi";
5         break;
6     case 2 : jour = "Mardi";
7         break;
8     case 3 : jour = "Mercredi";
9         break;
10    case 4 : jour = "Jeudi";
11        break;
12    case 5 : jour = "Vendredi";
13        break;
14    case 6 : jour = "Samedi";
15        break;
16    case 7 : jour = "Dimanche";
17        break;
18    default : jour = "Invalide";
19        break;
20 }
```

Dans la partie de code précédent, la variable `jour` prendra après l'exécution la valeur "`Mercredi`" car l'entier stocké dans `n` vaut 3.

- Pour utiliser l'instruction `switch`, on liste donc avec les `case` les différentes valeurs pour lesquelles on veut tester l'égalité et le mot clef `default` est utilisé pour traiter les cas non présents.

- Ainsi dans l'exemple précédent, si `n` avait stocké par exemple l'entier 10, alors la variable `jour` aurait valu "`Invalide`".
- Lorsque l'instruction trouve un cas correspondant, elle exécute toutes les instructions suivantes jusqu'à rencontrer l'instruction `break`. Il est donc important de ne pas oublier cette instruction.
- Par exemple, si on prend les instructions suivantes :

```

1 String prenom = "Paul";
2 String nom = "";
3 switch( prenom ) {
4     case "John" : nom = "Lennon";
5
6     case "Paul" : nom = "McCartney";
7
8     case "Ringo" : nom = "Starr";
9
10    case "George" : nom = "Harrison";
11        break;
12    default : nom = "Inconnu";
13        break;
14 }
```

à la fin la variable `nom` contiendra la chaîne de caractères "`Harrison`". Si on veut qu'elle contienne la valeur "`McCartney`", alors il faut procéder de la façon suivante :

```

1 String prenom = "Paul";
2 String nom = "";
3 switch( prenom ) {
4     case "John" : nom = "Lennon";
5         break;
6     case "Paul" : nom = "McCartney";
7         break;
8     case "Ringo" : nom = "Starr";
9         break;
10    case "George" : nom = "Harrison";
11        break;
12    default : nom = "Inconnu";
13        break;
14 }
```

Exercice 1 (Simplification de conditions, *)

Réécrivez la suite d'instructions suivantes en utilisant l'instruction `switch`.

```

1 int nombre=1024;
2 String dernierChiffre="";
3 if(nombre%10 == 0){
4     dernierChiffre = "zero";
5 } else if (1024%10 == 1){
6     dernierChiffre = "un";
7 } else if (1024%10 == 2){
8     dernierChiffre = "deux";
9 } else if (1024%10 == 3){
10    dernierChiffre = "trois";
11 } else if (1024%10 == 4){
12    dernierChiffre = "quatre";
13 } else if (1024%10 == 5){
14    dernierChiffre = "cinq";
```

```

15 } else if (1024%10 == 6){
16     dernierChiffre = "six";
17 } else if (1024%10 == 7){
18     dernierChiffre = "sept";
19 } else if (1024%10 == 8){
20     dernierChiffre = "huit";
21 } else {
22     dernierChiffre = "neuf";
23 }

```

□

Exercice 2 (Utilisation de switch, *)

Écrire une fonction numeroMois qui prend en argument une chaîne de caractères et qui renvoie un entier correspondant au numero de mois (compris entre 1 et 12) si cette chaîne est un mois compris dans l'ensemble {Janvier, Fevrier, Mars, Avril, Mai, Juin, Juillet, Aout, Septembre, Octobre, Novembre, Decembre} et renvoie 0 sinon.

□

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.

```

1 while (expression booléenne) {
2     instructions à répéter
3 }

```

- 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 ».

```

1 int a = 50;
2 while (a > 0) {
3     System.out.println("Hello");
4     a = a - 1;
5 }

```

- 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 :

```

1 int b = 0;
2 while (b == 0) {
3     b = b / 2;
4 }
5 System.out.println ("Bonjour\n");

```

On peut aussi faire en sorte que la boucle soit exécutée au moins une fois et l'expression booléenne est alors évaluée après la première itération. Pour cela on utilise la syntaxe suivante :

```

1 do {
2     instructions à répéter

```

```
3 while (expression booléenne);
```

Exercice 3 (Première boucle, *)

Quelle est la valeur de la variable r à la fin de la suite d'instructions suivantes ? 7

```
1 int n = 49;
2 int r = 0;
3 while (r * r < n) {
4     r = r + 1;
5 }
```



Exercice 4 (Différence entre while et do...while, *)

1. Quelle est la valeur de la variable n à la fin de la suite d'instructions suivantes ?

```
1 int n = 10;
2 while (n < 10) {
3     n = n*2;
4 }
```



2. Même question avec la suite d'instructions suivantes ?

```
1 int n = 10;
2 do {
3     n = n*2;
4 } while (n < 10);
```



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".

```
1 int a = 0;
2 for (int i = 0; i < 32; i++) {
3     a = a + i;
4 }
5 System.out.println (a);
```



Exercice 6 (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 7 (Terminaison, *)

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

```

1 int n = 2;
2 while (n > 0) {
3     System.out.println (n);
4 }
```

```

1 int n = 2;
2 while (n > 0) {
3     n = n * 2;
4     System.out.println(n);
5 }
```

□

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 :

```

1 boolean b = false;
```

ou bien encore comme cela :

```

1 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 :

```

1 boolean b1 = false; // b1 vaut false.
2 boolean b2 = (3 + 2 < 6); // b2 vaut true.
3 b1 = !b2; // b1 vaut la negation de true, donc false.
4 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 8 (Évaluer, *)

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

```

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

□

Exercice 9 (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 10 (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 :

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

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

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

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

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 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.

□

4 Boucles de parcours

Parcourir des tableaux

[COURS]

- Pour l'instant, la seule façon que nous avons vu pour accéder aux éléments d'un tableau, c'est de passer par l'indice de ces éléments.
- JAVA offre la possibilité d'énumérer dans l'ordre les éléments d'un tableau sans passer par leur indice avec une utilisation particulière des boucles `for`.
- Voilà un exemple qui permet d'afficher les éléments d'un tableau d'entiers :

```
1 int [] tab={2,4,6,8,10};  
2 for (int x : tab){  
3     System.out.println(x);  
4 }
```

À chaque tour de boucle, un élément du tableau (de type `int`) est mis dans la variable `x`.

- Cette méthode a l'avantage d'être concise, mais toutefois elle ne permet pas de récupérer les indices des éléments, on ne peut donc pas toujours l'utiliser.
- Bien entendu cette méthode marche avec des tableaux à plusieurs dimensions. Il faut juste bien faire attention au type des éléments du tableau que l'on parcourt.
- Par exemple, le code suivant affiche tous les entiers contenus dans un tableau de tableaux d'entiers :

```
1 int [][] tab={{2,4,6,8,10},{1,3,5,7,9,11}};  
2 for (int [] t : tab){  
3     for (int x : t) {  
4         System.out.print(x);  
5         System.out.print(" ");  
6     }  
7 }
```

Il affichera ainsi la séquence 2 4 6 8 10 1 3 5 7 9 11 .

5 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 12 (Comptine, *)

Modifier les instructions suivantes pour que l'accord de kilomètre(s) soit correct.

```
1 int n = 10;  
2 while (n > 0) {  
3     System.out.print (n);  
4     System.out.println (" kilometre a pied ca use les souliers");  
5     n = n - 1;  
6 }
```

□

Exercice 13 (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 14 (Itérations, *)

Combien y a-t-il d'itérations dans la boucle suivante :

```
1 int n = 15;
```

```

2 while (n >= 0) {
3     n = n - 1;
4 }
```

Même question pour la boucle suivante :

```

1 static void nbIterations (int n) {
2     while (n >= 0) {
3         n = n - 1;
4     }
5 }
```

□

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 (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 précédemment, mais on demande ici d'utiliser une boucle `while`. □

Exercice 18 (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` :

$$\begin{aligned} x=2 &\rightarrow 2 \\ x=10 &\rightarrow 11 \\ x=20 &\rightarrow 23 \end{aligned}$$

Pour la fonction `number` :

$$\begin{aligned} x=10 &\rightarrow 4 \\ x=20 &\rightarrow 8 \end{aligned}$$

□

Exercice 19 (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 20 (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 $a^3 \geq x$.

□

Exercice 21 (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 22 (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 23 (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 8 de cours/TD

Université Paris-Diderot

Objectifs:

- Savoir écrire un programme en entier (structure du programme avec une procédure main et des déclarations de fonctions, de procédures et de variables globales).
- Savoir compiler et exécuter un programme.

```
1 public class ExoExecution {  
2  
3     public static void main (String[] args) {  
4         int p = 6;  
5         int q = 3;  
6         int u = 0;  
7         if (p % q == 0) {  
8             System.out.println ("q divise p.");  
9         }  
10        if (power (q, 2) <= p) {  
11            System.out.println ("p est plus grand que le carré de q.");  
12        }  
13        if (p / q >= 2) {  
14            System.out.println ("p est plus grand que le double de q.")  
15        }  
16    }  
17  
18    /* Retourne base élevée à la puissance exponent. */  
19    public static int power (int base, int exponent) {  
20        int result = 1;  
21        while (exponent > 0) {  
22            result = result * base;  
23            exponent = exponent - 1;  
24        }  
25        return result;  
26    }  
27}  
28}
```

Listing 1 – Un programme Java complet

1 Structure d'un programme Java

Structure d'un programme Java

[COURS]

- Le nom du programme suit le mot-clé `class`.
- Le nom du fichier doit être le nom du programme suivi du suffixe `.java`, le fichier du programme ci-dessus s'appellera donc `ExoExecution.java`.
- Le programme contient une procédure qui s'appelle `main`. L'exécution du programme consiste à exécuter la procédure `main`. L'exécution débute donc à la première ligne du corps de cette procédure.
- Pour éviter un `main` trop long et illisible, on peut bien sûr (et c'est fortement conseillé !) introduire des procédures et des fonctions auxiliaires qui permettent de regrouper des suites d'instructions similaires en des "briques" réutilisables dont les noms rendent plus clair le sens du programme.

Exercice 1 (Ordre d'exécution, *)

Écrire la séquence des lignes du code du programme ExoExecution dans l'ordre de leur exécution.



Exercice 2 (Ajouter des fonctions, *)

Ajouter la fonction `divides` suivante au programme ExoExecution et modifier le corps du main en conséquence.

```
1 public static bool divides (int q, int p) {  
2     return (p % q == 0);  
3 }
```



Structure des fonctions

[COURS]

- Une fonction est une série d'instructions *qui renvoie une valeur*.
- Les fonctions permettent de rendre un programme plus compréhensible en augmentant le vocabulaire du langage : on donne un nom à une série d'instructions calculant une certaine valeur et il n'est alors plus nécessaire de se souvenir de cette série d'instructions car on peut utiliser le nom de fonction introduit comme si il s'agissait d'une nouvelle primitive du langage.
- Les fonctions permettent de factoriser du code : plutôt que d'écrire plusieurs fois la même série d'instructions (avec éventuellement quelques variations artificielles), on écrit une seule fois ces instructions en explicitant des paramètres pour représenter les variations. Par exemple, à la place de :

```
1 int z = y * y + (1 + y) * (1 + y) + (2 + y) * (2 + y);
```

On peut introduire la fonction `square` suivante :

```
1 public static int square (int x) {  
2     return x * x;  
3 }
```

et remplacer la déclaration de la variable `z` par :

```
1 int z = square (y) + square (1 + y) + square (2 + y);
```

- Une fonction est caractérisée par :
 - **un corps** : le bloc d'instructions qui décrit ce que la fonction calcule;
 - **un nom** : par lequel on désignera cette fonction ;
 - **des paramètres** (*l'entrée*) : l'ensemble des variables extérieures à la fonction dont le corps dépend pour fonctionner et qui prennent une valeur au moment de l'appel de la fonction ;
 - **une valeur de retour** (*la sortie*) : ce que la fonction renvoie à son appelant.

Par exemple, le code du programme ExoExecution ci-dessus définit une fonction de nom `power`, qui prend en paramètre deux valeurs entières `base` et `exponent`, de type `int`, et renvoie en sortie une valeur de type `int` qui vaut $\text{base}^{\text{exponent}}$:

```

1  /* Retourne base élevée à la puissance exponent. */
2  public static int power (int base, int exponent) {
3      int result = 1;
4      while (exponent > 0) {
5          result = result * base;
6          exponent = exponent - 1;
7      }
8      return result;
9 }
```

- La première ligne qui déclare la fonction est appelée **en-tête** ou **prototype** de la fonction et précise le type de sa valeur de retour, son nom et les types et les noms de chacun de ses paramètres.
- Le sens précis de **public** et **static** seront traités dans le cours de programmation orientée objet en Java.
- Le choix des noms de la fonction et de ses paramètres fournit une documentation minimale du comportement de la fonction, il est donc souhaitable qu'ils soient parlant. Par exemple, le programme est moins compréhensible si on remplace l'en-tête de **power** par « **public static int function1 (int x, int y)** ».
- Dans le corps de la fonction se trouvent les instructions qui détaillent la manière dont la valeur de sortie est calculée. Ces instructions utilisent les paramètres de la fonction (**base** et **exponent**) et, éventuellement, de nouvelles variables déclarées localement dans le corps (dans notre exemple, on utilise la variable **result**).
- La valeur renournée par la fonction est indiquée par l'instruction

```
return expression ;
```

- où **expression** a le même type que celui de la valeur de sortie déclaré dans l'en-tête de la fonction.
- L'instruction **return** fait deux choses :
 1. elle précise la valeur qui sera fournie par la fonction en résultat,
 2. elle met fin à l'exécution des instructions de la fonction.
 - Il est possible d'avoir plusieurs occurrences de **return** dans le même corps, comme dans la fonction suivante :

```

public static int minimum (int a, int b) {
    if (a < b) {
        return a;
    } else {
        return b;
    }
}
```

- On doit s'assurer que tout chemin d'exécution possible du corps d'une fonction mène à un **return**, sinon Java rejette le programme. Par exemple :

```

public static int minimum (int a, int b) {
    if (a < b) {
        return a;
    }
}
```

produit l'erreur suivante à la compilation :

```
ExoInvalidMinimum.java:10: missing return statement
```

- Une fois définie, il est possible d'utiliser la fonction dans toute expression, on parle alors d'**appel** (ou d'**invocation**) d'une fonction. On peut par exemple écrire :

```
int a = 3 + power (4, 2);
```

Dans cette déclaration, la sous-expression « power (4, 2) » est remplacée par la valeur renournée par le corps de la fonction power pour base valant 4 et exponent valant 2, c'est-à-dire 16. On évalue ensuite $3 + 16$ pour obtenir la valeur 19 qui sera la valeur initiale de a.

- L'appel d'une fonction doit respecter le prototype de cette fonction, c'est-à-dire le nombre et les types des paramètres ainsi que le type de la valeur de sortie.

Exercice 3 (Évaluer un appel, *)

Soit power la fonction définie dans le programme ExoExecution ci-dessus, quelle est la valeur de la variable x après exécution de la suite d'instructions qui suit :

```
1 int x = 2;
2 int y = 2;
3 x = power (x, y);
4 x = power (x, y);
5 x = power (x - 8, y);
```

□

Exercice 4 (Appelle-moi bien !, *)

Soient « `public static int power(int base, int exponent)` » un prototype de fonction et x, y deux variables de type `int`. Trouver les appels incorrects parmi ceux de la liste suivante. Motiver votre réponse.

1. « `y = power (2, x);` »
2. « `int a = power ("2", 3);` »
3. « `String a = power (2, x);` »
4. « `if (power (2, x) > 5) { ... }` »
5. « `while (power (2, x)) { ... }` »
6. « `x = power (x, power (x, x));` »
7. « `x = power(x, System.out.print (y));` »

□

Exercice 5 (Mauvais en-tête, **)

Dire pourquoi aucun des en-têtes suivants n'est correct.

1. « `public static int fonc1 (int a; int b)` »
2. « `public static fonc2 (boolean c)` »
3. « `public static int, fonc3 (String w)` »
4. « `public static boolean fonc4 (int)` »
5. « `public static String fonc5 (String t, int d, int t)` »
6. « `public static int 6fonc (int x)` »

□

Exercice 6 (En-tête déduite d'une spécification, *)

Donner l'en-tête d'une fonction qui prend en paramètre une chaîne de caractères et renvoie sa longueur. □

Exercice 7 (En-tête déduite d'un corps de fonction, **)

Donner l'en-tête de la fonction dont le corps est défini comme suit. (La fonction « `public static int intArrayLength (int[] t)` » prend en paramètre un tableau d'entiers et renvoie sa longueur.) :

```
1 ... {
2     int r = 0;
3     int i = 0;
```

```

5  if (t.length != u.length) {
6      return -1;
7  }
8
9  while (i < t.length) {
10     r = r + t[i] * u[i];
11     i = i + 1;
12 }
13
14 return r;
15 }
```

□

Les procédures

[COURS]

- Les **procédures** sont des suites d'instructions qui ne renvoient pas de valeur mais qui produisent des effets sur le système qui les exécute. Par exemple, la procédure « `void System.out.print(String s)` » affiche un message à l'écran.
- Les procédures ont un en-tête et un corps, mais le type de leur valeur de retour est `void` (*vide* en français) et le corps ne contient donc pas obligatoirement l'instruction `return`. Par exemple :

```

1 public static void printRange (int from, int to) {
2     for (int i = from; i <= to; i++) {
3         System.out.print (i);
4     }
5 }
```

- La procédure `main` a un rôle particulier car elle contient les premières instructions à exécuter lorsque le programme est lancé : on dit que `main` est le *point d'entrée* du programme.
- Le prototype de `main` est obligatoirement « `public static void main(String [] args)` ». On verra par la suite à quoi sert le tableau de chaînes de caractères `args`.

2 Taper, Coder, Exécuter

Les trois étapes principales de la programmation

[COURS]

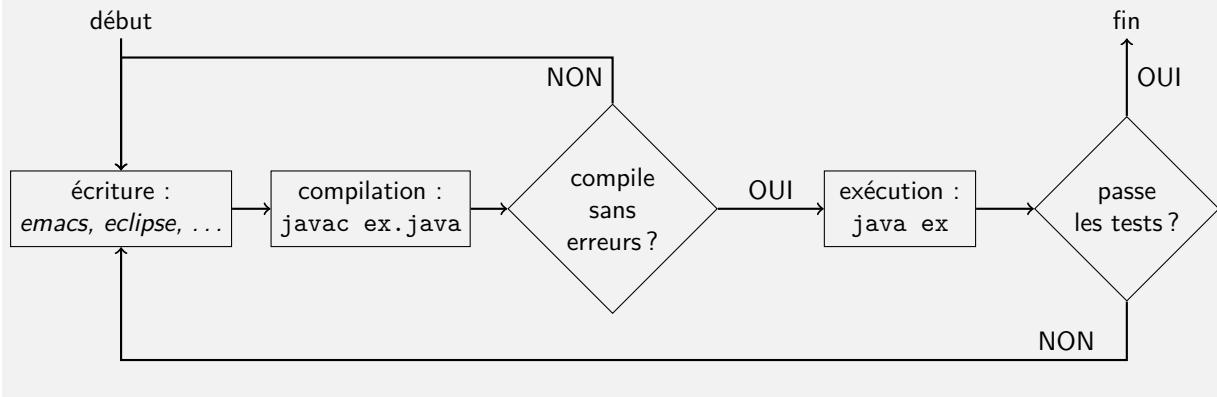
Pour obtenir un programme, on suit les trois étapes suivantes :

1. L'**écriture** du code source en utilisant un éditeur de texte (par exemple `Emacs`, `Eclipse`, ...).
 - Le nom du fichier contenant la procédure `main` doit être le nom du programme suivi du suffixe `.java`. Par exemple, pour l'exemple de la page 1, le code se trouve dans le fichier `ExoExecution.java`.
 - La procédure `main` décrit les instructions principales du programme et, éventuellement, utilise d'autres fonctions auxiliaires définies dans le même fichier ou dans des bibliothèques (comme par exemple la bibliothèque `Math` de Java).
2. La **compilation** du code source, effectuée par le programme `javac` (*java compiler*) auquel on a donné le nom de fichier contenant le code source (qui porte le suffixe `.java`).
 - Cette commande transforme le code qui est un texte lisible par un être humain en *bytecode* lisible par la machine virtuelle `JVM` (*Java Virtual Machine*). Le nom du fichier en bytecode se termine par le suffixe `.class`.
 - Au passage le compilateur `javac` vérifie que le programme est syntaxiquement correct et qu'il est bien typé, c'est-à-dire que les expressions produisent bien des valeurs du type attendu. Si le fichier ne respecte pas l'un de ces points (par exemple, si on oublie un ; après une instruction), une erreur

se produit à la compilation et aucun fichier .class n'est créé. Il faut alors revenir au point 1 et corriger le code source.

3. L'**exécution** du bytecode par la machine virtuelle JVM. Cette exécution s'effectue par la commande java suivie du nom du programme contenant la procédure main.
 - La machine virtuelle exécute les instructions de la procédure main.
 - **Attention** : une compilation qui se passe bien ne signifie pas que le programme est correct, des erreurs peuvent se produire à l'exécution. Il faut donc tester le programme et vérifier que le comportement est celui attendu. Dans le cas contraire, il faut revenir au point 1 et corriger le code source.

Le diagramme suivant résume ces trois étapes :



Exercice 8 (Compiler, exécuter, **)

Considérez le code du programme ExoExecution présenté en première page du TD.

- Quel doit être le nom du fichier contenant ce code source ?
- Quelle commande permet de le compiler ?
- Quel est le nom du fichier bytecode obtenu
- Comment réalise-t-on son exécution ?
- Que se passe-t-il si on élimine la ligne 5 ?
- Que se passe-t-il si on la remplace par l'instruction suivante ?

```
int q = 0;
```

□

Passer des paramètres à la procédure main

[COURS]

- Comme dit précédemment, l'en-tête de la procédure main est « `public static void main(String [] args)` ». Elle prend en paramètre un tableau de chaînes de caractères, où sont stockées les valeurs qui suivent le nom du programme dans la commande qui provoque son exécution.
- Par exemple, pour le programme ExoExecution, la commande

```
1 java ExoExecution a1 a2 a3
```

fixe le contenu du tableau args à :

```
1 { "a1", "a2", "a3" }
```

Exercice 9 (Utilisation args, *)

Considérons le programme suivant :

```
1 public class ExoArgs {
2
3     public static void main (String[] args) {
```

```

4     System.out.println ("Bonjour " + args [0]);
5 }
6
7 }

```

code/ExoArgs.java

Après compilation du fichier ExoArgs.java, qu'affichent à l'écran les commandes suivantes ?

1. « java ExoArgs Jean-Paul »
2. « java ExoArgs Jean Paul »
3. « java ExoArgs »

□

Exercice 10 (Option de verbosité, **)

Considérons le code source suivant :

```

1 public class ExoVerbose {
2     public static void main (String [] args) {
3         int r = 0;
4         int x = 10;
5         for (int i = 1; i <= x; i++) {
6             r = r + i;
7         }
8         System.out.print (" La somme des premiers ");
9         System.out.print (x);
10        System.out.print (" nombres entiers est ");
11        System.out.println (r);
12    }
13 }

```

Modifier le programme de façon à ce que :

- l'exécution sans option (java ExoVerbose) affiche sur l'écran la somme des dix premiers nombres entiers ;
- l'exécution avec l'option verbose (java ExoVerbose verbose) affiche sur l'écran toutes les valeurs des affectations exécutées sur i et r et ensuite la somme des 10 premiers entiers ;
- l'exécution avec toute autre option (par exemple java ExoVerbose verbose all) n'exécute pas le calcul mais affiche sur l'écran : "Ces options ne sont pas connues".

□

3 Fonctions utilisées

Liste des fonctions

[COURS]

```

1 /* Renvoie base élevée à la puissance exponent. */
2 public static int power (int base, int exponent) {
3     int result = 1;
4     while (exponent > 0) {
5         result = result * base ;
6         exponent = exponent - 1;
7     }
8     return result;
9 }

```

4 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 11 (Devine la fonction, **)

Définir la fonction `average` utilisée dans le code suivant (les variables `m` et `grades` sont supposées être définies avant cette portion de code) :

```
1  while (e < grades.length) {
2      if (grades[e].length == 0) {
3          System.out.print ("l'étudiant numéro ");
4          System.out.print (e);
5          System.out.print (" n'a aucune note !\n");
6      } else {
7          m = average (grades[e]);
8          if (m < 10) {
9              System.out.print ("l'étudiant numéro ");
10             System.out.print (e);
11             System.out.print (" n'a pas passé le semestre.\n");
12         } else {
13             System.out.print ("l'étudiant numéro ");
14             System.out.print (e);
15             System.out.print (" valide avec la moyenne de ");
16             System.out.print (m);
17             System.out.print ("\n");
18         }
19     }
20     e = e + 1;
21 }
22
23 /* Retourne la moyenne des entiers contenus dans le tableau tab. */
24 public static int average (int[] t) {
```

code/ExoGuess.java

□

Exercice 12 (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` :

$$\begin{aligned}x=2 &\rightarrow 2 \\x=10 &\rightarrow 11 \\x=20 &\rightarrow 23\end{aligned}$$

Pour la fonction `number` :

$x=10 \rightarrow 4$
 $x=20 \rightarrow 8$

□

Exercice 13 (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 renournée sous forme de tableau, dans cet exemple, $\{ 7, 6, 2 \}$.

1. Écrire une fonction `reverse` qui prend en paramètre un tableau d'entiers, renvoie le tableau qui contient les mêmes chiffres, dans l'ordre inverse. Par exemple, si l'entrée est le tableau $\{ 7, 4, 3 \}$, on doit rendre un tableau de trois chiffres $\{ 3, 4, 7 \}$.

2. 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 $\{ 2, 6, 7 \}$.

3. 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 14 (Perroquet, *)

Écrire un programme `ExoPerroquet` dont l'exécution affiche à l'écran "Je suis un Perroquet" et puis, dans une suite de lignes, tous les mots donnés en paramètre au programme :

```
java ExoPerroquet toto titi  
je suis un Perroquet  
toto  
titi
```

□

Exercice 15 (Perroquet avec options, **)

Écrire un programme `ExoPerroquetOpt` dont l'exécution affiche à l'écran "Je suis un Perroquet" sur la première ligne; sur la ligne suivante, il affiche d'abord "Options: ", puis la liste des mots donnés en paramètre au programme si et seulement si ils commencent avec le caractère "-" (tiret). La présence d'un mot en paramètre qui ne commence pas par un tiret doit engendrer l'arrêt du traitement des mots suivants et l'affichage du message "erreur: MOT n'est pas une option" où `MOT` est le premier mot qui ne commence pas avec un tiret.

Contrat:

```
java ExoPerroquetOpt -toto -titi  
je suis un Perroquet  
Options : -toto -titi
```

```
java ExoPerroquetOpt -toto -titi all any
je suis un Perroquet
Options : -toto -titi
Erreur : all n'est pas une option
```

□

Exercice 16 (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 $(a + b)$.

□

Introduction à la Programmation 1 JAVA

51AE011F

Séance 9 de cours/TD

Université Paris-Diderot

Objectifs:

- Variables locales et globales
- Tester une fonction
- Décrire (spécifier) le comportement d'une fonction.
- Réaliser une spécification.

1 Approfondissements sur les fonctions

Rappels

[COURS]

- Les fonctions et procédures peuvent prendre toutes sortes de *paramètres* : des entiers, des booléens, des couples, des tableaux, etc. En général, on écrit des fonctions qui prennent peu de paramètres, et on évite les paramètres redondants ou inutiles.
- Une fonction permet de *factoriser du code*. Par exemple, si l'on veut remplacer une lettre par une autre dans plusieurs chaînes de caractères, il est utile de créer une fonction que l'on puisse appliquer à chacune des chaînes plutôt que de faire des copier-coller comme dans le programme suivant :

```
1 String s = "abracadabra";
2 String r = "";
3 for(int i = 0; i < s.length(); i++) {
4     if (s.charAt(i)=='a') {
5         r = r + "e";
6     } else {
7         r = r + s.charAt(i);
8     }
9 }
10 System.out.println(r);
11
12 String s = "magie";
13 String r = "";
14 for(int i = 0; i < s.length(); i++) {
15     if s.charAt(i)=='a' {
16         r = r + "e";
17     } else {
18         r = r + s.charAt(i);
19     }
20 }
21 System.out.println(r)
```

- Introduire des fonctions a aussi d'autres buts : en donnant des noms bien choisis aux fonctions et à leurs arguments, et en leur assignant des tâches clairement identifiées, on améliore la lisibilité du code, et sa réutilisabilité dans d'autres programmes.
- Une fonction *doit* renvoyer un résultat du type annoncé dans l'en-tête de la fonction sauf s'il s'agit d'une fonction de type `void`, ou si la fonction ne se termine jamais. Le compilateur Java renvoie une erreur s'il n'est pas en mesure de s'assurer qu'un résultat sera bien renvoyé quelles que soient les valeurs des paramètres.
- L'instruction `return` permet de renvoyer le résultat. Cette instruction stoppe l'exécution et termine immédiatement la fonction. Il faut faire attention lorsqu'on l'utilise à ne pas stopper l'exécution trop tôt.

Exercice 1 (Bien concevoir ses fonctions, *)

Critiquez la façon dont est conçu ce code et proposez une version améliorée.

```

1 public static void f23(int []t, int n) {
2     int z = 0;
3     for (int i = 0; i<n; i++) {
4         z = z + t[i];
5     }
6     System.out.println("Somme : " + z);
7 }
8
9 public static int f24(int n, int []t, String msg) {
10    int h = 0;
11    for (int i = 0; i<n; i++) {
12        h = h + t[i];
13    }
14    return (h / n);
15 }
16
17 public static void main(String []args) {
18     [...]
19     f23(t, t.length);
20     [...]
21     System.out.println("Moyenne : " + f24(u, u.length, "Moyenne"));
22 }
```

□

Exercice 2 (Bien utiliser return, *)

1. Écrivez une fonction `membership` qui prend en argument un entier `n` et un tableau d'entiers `t`, et qui renvoie `true` si `n` apparaît dans `t`, autrement dit si `n=t[i]` pour au moins un `i`. On cherchera à renvoyer le résultat le plus tôt possible.
2. En utilisant la fonction `membership`, écrivez une fonction `inclusion` qui prend en argument deux tableaux d'entiers `t` et `u` et qui renvoie `true` si l'ensemble des entiers de `t` est inclus dans l'ensemble des entiers de `u`, autrement dit, `membership(t,u)` renvoie `true` si pour tout indice `i` de `t`, l'entier `t[i]` apparaît dans `u`.

□

Variables locales, variables globales

[COURS]

- Une variable définie dans le corps d'une fonction ou d'une procédure est **locale** à cette fonction. Les paramètres sont assimilés à des variables locales.
 - Une variable définie en dehors de toute fonction ou procédure est dite **globale**.
- Par exemple, dans le programme ci-dessous,

```
1 class myprog {  
2     public static boolean flag = false;  
3  
4     public static int f(int x) {  
5         int a = 0;  
6         if (x>=3) {  
7             a = 3;  
8         } else {  
9             a = x;  
10        }  
11        return a;  
12    }  
13  
14    public static void g() {  
15        flag = (f(5) == 3);  
16    }  
17    [...]  
18 }
```

il y a une variable *globale* : flag et deux variables *locales* à la fonction f : le paramètre x et la variable a.

- Les paramètres et variables locales d'une fonction ou d'une procédure n'appartiennent qu'à cette fonction, c'est la règle de *localité*. En particulier, une fonction (par exemple g) ne peut pas faire référence à une variable locale d'une autre fonction (par exemple la variable a de f). Les variables locales sont détruites après chaque exécution d'une fonction.
- Au contraire, une variable globale est accessible par toutes les fonctions.

```
1 public class c {  
2     public static int count = 0;  
3     public static void show() { System.out.println(count); }  
4     public static void inc() { count = count + 1; }  
5     public static void main(String []args) {  
6         while (count < 5) { inc(); }  
7         show();  
8     }  
9 }
```

Dans le programme ci-dessus, la procédure show accède en lecture à la variable globale count, la procédure inc modifie la variable globale count.

- Quand on passe un tableau en paramètre à une fonction ou à une procédure, ce n'est pas son contenu, mais son adresse dans le tas. Ainsi, on peut écrire des fonctions et procédures qui modifient des tableaux passés en paramètres.

```
1 public static void changeFst(int []t, int a) { t[0]=a; }  
2 public static main(String []args) {  
3     int []u = {1, 2, 3};  
4     changeFst(u, 0);  
5 }
```

Exercice 3 (Règle de localité, *)

Supprimez la ligne de programme qui n'est pas correcte, et renommez les variables pour lever les ambiguïtés (sans changer le comportement du programme restant). Assurez-vous que chaque déclaration de variable (globale, locale, paramètre) introduit un nom de variable distinct.

```
1 public static int a = 0;
2
3 public static int f (int a, int b) {
4     int c = 1;
5     a = a+b;
6     return a;
7 }
8
9 public static int g(int b) {
10    a = a + b;
11    int b = 1;
12    c = a;
13    return f(a,b);
14 }
```

□

Exercice 4 (Compte en banque, **)

On considère le programme suivant.

```
1 public class bankAccount {
2
3     public static int balance;
4
5     public static void deposit(int amount) {
6         balance = balance + amount;
7     }
8
9     public static int withdraw(int amount) {
10        int newbalance = balance - amount;
11        if (newbalance>0) {
12            balance = newbalance;
13        } else {
14            amount = balance;
15            balance = 0;
16        }
17        return amount
18    }
19
20    public static void main(String []args) {
21        System.out.println(balance);
22        deposit(1000);
23        System.out.println(balance);
24        int amount1 = 100;
25        System.out.println(withdraw(amount1));
26        System.out.println(balance);
27        int amount2 = 1000;
```

```

28     System.out.println(withdraw(amount2));
29     System.out.println(amount2);
30     deposit(1000);
31     int amount = 2000;
32     System.out.println(withdraw(amount));
33     System.out.println(amount);
34 }
35 }
```

1. Quelles sont les variables globales, les paramètres, et les variables locales ?
2. Qu'affiche le programme ?

□

Exercice 5 (Test, **)

1. Écrire une fonction `isInMatrix` qui prend en argument un tableau `M` de tableaux d'entiers et un entier `n`, et qui renvoie `true` si `n` apparaît dans `M`.

Contrat:

Pour $0 \leq n \leq 5$ et $0 \leq m \leq 9$.

`M1 = {{0,1,2},{3,4,5}} et M2 = {{0,5},{1,4},{2,2},{3,1}}.`

<code>isInMatrix([], n)</code>	→	<code>false</code>
<code>isInMatrix(M1, n)</code>	→	<code>true</code>
<code>isInMatrix(M1, m)</code>	→	<code>false</code>
<code>isInMatrix(M2, n)</code>	→	<code>true</code>
<code>isInMatrix(M2, m)</code>	→	<code>false</code>

2. Écrire les tests correspondants au contrat ci-dessus. Si le i -ème test n'est pas valide, le message test i non valide est affiché.
3. On veut pouvoir écrire les tests de la façon suivante :

```

1 startTests();
2 for(int n=0; i<10; i++) {
3     test(M1, n, n<6);
4     test(M2, n, n<6);
5 }
6 stopTests();
```

Écrire les fonctions `startTests`, `test`, et `stopTests`. La fonction `stopTests` affiche le nombre de tests qui ont échoué.

□

Documentation et Spécification

[COURS]

Un programme peu ou mal documenté est difficile à corriger, modifier et réutiliser. Il est donc important de documenter et de commenter systématiquement son code.

Pour chaque **fonction**, avant l'en-tête, on place quelques lignes de commentaires, au format suivant :

- une ligne pour chaque paramètre d'entrée, indiquant son type, ce que ce paramètre représente et une propriété attendue par la fonction sur l'entrée, que l'on appelle **précondition**;
- une ligne pour la valeur renvoyée, son type, ce qu'elle représente et la propriété promise par la fonction sur cette sortie, que l'on appelle **postcondition**.

Des commentaires supplémentaires liés à l'implémentation de la fonction seront placés dans son corps, comme dans l'exemple suivant :

```

1  /*
2   * Vérification de la correction d'une date.
3
4   Entrée : d un entier désignant le jour.
5   Entrée : m un entier désignant le mois.
6   Entrée : y un entier désignant l'année.
7   Sortie : true si la date est correcte, false sinon.
8 */
9  public static boolean checkDate (int d, int m, int y) {
10    if (m >= 1 && m <= 12 && d >= 1 && d <= 31) {
11      if (m == 4 || m == 6 || m == 9 || m == 11) {
12        return (d <= 30);
13      } else {
14        if (m == 2) { /* Est-ce une année bissextile? */
15          if ((y%4 == 0 && y%100 !=0 ) || y%400 == 0) {
16            return (d <= 29);
17          } else {
18            return (d <= 28);
19          }
20        } else {
21          return true;
22        }
23      }
24    } else {
25      return false;
26    }
27 }
```

Pour les **procédures**, on indiquera l'**effet** qui sera observé (affichage, modification d'un tableau, modification de variables globales, lecture d'un fichier,...) après l'exécution de la procédure à la place de la **postcondition**, qui n'a pas lieu d'être puisqu'aucune valeur n'est renvoyée par une procédure.

```

1
2 // Procedure LastFirst
3 //
4 // Entrée: t un tableau d'entier
5 //
6 // Effet1: mise à 0 du dernier élément de t
7 // Effet2: affichage du premier élément de t
8 //
9
10 public static int LastFirst(int []t) {
11   t[t.length-1] = 0;
12   print(t[0]);
13 }
```

La spécification des fonctions et procédures ainsi fournie doit permettre de les utiliser sans avoir à connaître leur code.

Exercice 6 (Somme de tableaux, *)

Complétez la spécification ci-dessous et écrivez la fonction qui implémente cette spécification.

```

1 // Somme de tableaux
2 //
3 // Entrée : un tableau d'entiers a
```

```
4 // Entrée : un tableau d'entiers b  
5 // Sortie : un tableau d'entiers c tel que c[i] = a[i] + b[i] pour tout indice i
```

□

Exercice 7 (Séquences dans un tableau, *)

Écrire une fonction qui implémente la spécification suivante :

```
1 /*  
2     Entrée : un tableau d'entiers a.  
3     Sortie : true si et seulement si a contient deux  
4             séquences consécutives identiques. Par exemple, sur  
5             {1,2,3,2,3,7} la valeur à renvoyer est true.  
6 */
```

□

2 DIY

Dans les exercices suivants, vous vous appliquerez à bien factoriser votre code.

Exercice 8 (Réaliser des spécifications, *)

Écrire une fonction qui implémente la spécification suivante :

```
1 // Somme des entiers d'un intervalle.  
2 //  
3 // Entrée: low un entier désignant le début de l'intervalle.  
4 // Entrée: high un entier désignant la fin de l'intervalle.  
5 // Sortie: la somme des entiers compris au sens large entre  
6 // low et high.
```

□

Exercice 9 (Utiliser des spécifications, **)

La conjecture de Goldbach dit que tout nombre pair plus grand que 2 peut être écrit comme la somme de deux nombres premiers.

1. Écrire une fonction `boolean isPrime(int p)` qui implémente la spécification suivante :

```
1 // Teste si un entier est premier.  
2 //  
3 // Entrée : un entier p plus grand que 1.  
4 // Sortie : true si et seulement si n est premier, false sinon.
```

2. Écrire la spécification de la fonction `isGoldbach(n)` qui renvoie True si et seulement si la conjecture est vérifiée pour l'entier n. (Si n n'est pas pair ou est strictement plus petit que 2 la fonction renvoie True.)

3. Donner le code de la fonction `isGoldbach(n)`. (On utilisera la fonction précédemment définie.)

4. Écrire une fonction `goldbach(p)` qui implémente la spécification suivante. (On utilisera la fonction précédemment définie.) :

```
1 // Vérifie la conjecture de Goldbach jusqu'à un certain rang.  
2 //  
3 // Entrée : un entier n.  
4 // Sortie : True si et seulement si tout entier inférieur ou  
5 // égal à p vérifie la conjecture de Goldbach.  
6 //
```

□

Exercice 10 (Chaîne correspondante à un tableau, *)

Écrire une fonction qui implémente la spécification suivante :

```
1 // Entrée : un tableau de tableaux d'entiers t.  
2 // Sortie : une chaîne de caractères le représentant.  
3 //  
4 // Par exemple, si t vaut {{1,2},{3},{2,3,7}}, la chaîne de caractères  
5 // sera "{{1,2},{3},{2,3,7}}".  
6 //
```

□

Exercice 11 (Tri d'un tableau, **)

1. Écrire une procédure swap qui implémente la spécification suivante :

```

1 // 
2 // Echange le contenu de deux cases d'un tableau.
3 //
4 // Entrée : un tableau d'entiers t.
5 // Entrée : un entier i compris dans les bornes du tableau.
6 // Entrée : un entier j compris dans les bornes du tableau.
7 //
8 // Après avoir exécuté cette procédure les contenus des
9 // cases i et j du tableau t sont échangés.
10 //
```

2. En utilisant la procédure précédente, écrire une procédure findmin de spécification suivante :

```

1 // 
2 // Echange le contenu de la case i avec la case j du tableau t qui
3 // contient le plus petit entier des cases d'indice compris entre
4 // i et l'indice de fin du tableau.
5 //
6 // Entrée : un tableau d'entiers t.
7 // Entrée : un entier i compris dans les bornes du tableau.
8 //
9 //
```

3. En utilisant la procédure précédente, écrire une procédure sort qui implémente la spécification suivante :

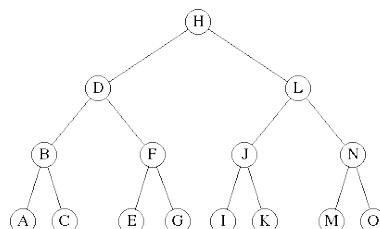
```

1 // 
2 // Tri le tableau.
3 //
4 // Entrée : un tableau d'entiers t.
5 //
```

□

Exercice 12 (Tournoi, ***)

Un tournoi est un arbre binaire (les noeuds de l'arbre contiennent toujours deux fils) dont chaque noeud interne correspond au gagnant du match entre ces deux fils. Par exemple, dans le tournoi



on a $B = \max(A, C)$, $D = \max(B, F)$, etc. Les participants au tournoi sont les entiers apparaissant sur les feuilles de l'arbre ; dans l'exemple ci-dessus, les participants sont donc les entiers A, C, E, G, I, K, M, O . Pour simplifier le problème, le nombre de participants est supposé être une puissance de 2.

Un arbre est représenté par un tableau en décidant que :

- la racine de l'arbre est à la position 0 ;
- le fils gauche d'un noeud situé dans la case i est situé dans la case $2 * i + 1$;

— le fils droit d'un nœud situé dans la case i est situé dans la case $2 * i + 2$.

Ainsi, l'arbre précédent peut être représenté par le tableau de taille 15 suivant :

```
1 { H, D, L, B, F, J, N, A, C, E, G, I, K, M, O }
```

Pour représenter un tournoi avec 2^n participants, il faut $2^{n+1} - 1$ noeuds dans l'arbre. (Montrez-le par récurrence pour vous en convaincre.) Il faut donc créer un tableau de taille $2^{n+1} - 1$ et on l'initialise avec la valeur -1 pour indiquer qu'aucun match n'a encore eu lieu.

Pour commencer le tournoi, on place les participants sur les feuilles de l'arbre. Tant qu'il ne reste pas qu'un seul participant, on fait un tour supplémentaire de jeu. Un tour de jeu consiste à faire jouer tous les matchs du niveau le plus bas de l'arbre dont les matchs ne sont pas encore joués. On fait alors "remonter" le gagnant de chaque match joué pour ce tour au niveau suivant, ce qui a pour effet de préparer le tour suivant, joué au niveau du dessus dans l'arbre.

1. Après avoir spécifié et implémenté des fonctions utiles pour :

- construire l'arbre nécessaire à la représentation d'un tournoi entre 2^n participants représentés par un tableau ;
- représenter l'indice du père d'un nœud dans l'arbre ;
- faire jouer deux participants en faisant remonter le gagnant dans l'arbre ;

Écrivez une fonction tournament de spécification suivante :

```
1 //
2 //
3 // Entrée : un entier n.
4 // Entrée : un tableau d'entiers t de taille 2 puissance n.
5 // Sortie : un tableau représentant le tournoi des participants
6 // de t.
7 //
8 //
```

2. Comment généraliser la fonction précédente à un nombre de participants qui n'est pas une puissance de 2 ?

3. Après avoir spécifié et implémenté une fonction qui calcule l'ensemble des participants qui ont perdu contre le vainqueur d'un tournoi, écrivez une fonction kbest de spécification suivante :

```
1 //
2 // Entrée : un tableau d'entiers t.
3 // Entrée : un entier k compris entre 1 et la taille du tableau.
4 // Sortie : un tableau représentant les k meilleurs participants
5 // de t.
6 //
```

□

Introduction à la Programmation 1 JAVA

51AE011F

Séance 10 de cours/TD

Université Paris-Diderot

Objectifs:

- Reconnaître les erreurs de programmation
- Débogage “scientifique” d’un programme

1 Erreurs et débogage

Typologies d’erreurs de programmation [COURS]

Dans la pratique de la programmation vous avez déjà rencontré plusieurs types d’erreurs de programmation. Classifions les :

erreurs de syntaxe (à la compilation) le code du programme ne respecte pas les règles de syntaxe du langage de programmation ; le programme n'est pas compris par le compilateur (ou l'interpréteur, le cas échéant). Vous n'arrivez pas à obtenir un programme exécutable.

erreurs de typage (à la compilation ; plus en général : erreur de non-respect de la sémantique du langage de programmation) la syntaxe du programme est correcte, mais l'usage d'une ou plusieurs expressions n'est pas compatible avec leur contexte. Par exemple, l'évaluation de ces expressions donne des valeurs dont le type n'est pas compatible avec le type attendu. Vous n'arrivez pas à obtenir un programme exécutable non plus.

Dans la catégorie plus générale de non-respect de la sémantique du langage de programmation on inclut aussi d'autres erreurs, comme par exemple l'appel d'une fonction jamais définie, l'appel d'une fonction avec un nombre incorrect de paramètres, etc.

non-respect de la spécification le programme est syntaxiquement correct et ne contient aucune erreur de type. Vous arrivez à obtenir un programme exécutable. Dans certains cas, l'exécution du programme donne des résultats (ou un comportement) différents de ce que sa spécification demande. Il s'agit d'erreurs à l'exécution.

“**échec**” un cas particulier du non-respect de la spécification est l'échec à l'exécution, non prévu par la spécification, menant à l'arrêt anticipé de l'exécution d'un programme, par exemple à cause d'une exception. Remarquez qu'une spécification peut prévoir l'arrêt d'un programme, même avec une exception. Dans ce cas il ne s'agit pas d'un non-respect de la spécification (mais il peut s'agir d'une spécification mal conçue).

Le compilateur Java aide le développeur à ne pas commettre certains types d’erreurs, notamment :

- un programme avec erreurs de syntaxe *ne compile pas* ;
- un programme avec erreurs de typage *ne compile pas*.

Si un programme compile, vous avez donc la certitude que les erreurs de syntaxe et typage ont toutes été résolues.

Exercice 1 (Bestiaire d’erreurs, *)

Trouver, classifier, et corriger (si possible) toutes les erreurs dans les fragments de code suivants.

```
1 public static void main(String[] args) {
2     for(int i = 0; i++) {
3         System.out.print(i);
4 }
```

code/SvntCompErrEx.java

```
1 public static void hello(int x) {
2     int n = "an integer";
3     System.out.print ("Hello, ");
4     System.out.print (x);
5     System.out.print (" world!\n");
6     return x;
7 }
8
9 public static void main(String[] args) {
10    hello(42);
11    hello("42", 43);
12    bonjour("42");
13 }
```

code/TvpeCompErrEx.java

```
1 /* spécification (informelle): renverser le tableau d'entiers
2  * {2, 5, -12, 8} et afficher le résultat sur la sortie standard
3  */
4 public static void main(String[] args) {
5     int[] tab = {2, 5, -12, 8};
6     int len = tab.length;
7     int[] revtab = new int[len];
8     for (int i = 0; i < len; i++) {
9         revtab[i] = tab[len - i];
10    }
11    for (int i = 0; i < len; i++) {
12        System.out.println (tab[i]);
13    }
14 }
```

code/ExcepRunTErrEx1.java

```
1 /* proportion des entiers sur la somme d'un tableau
2  * entrée: un tableau d'entiers t
3  * sortie: un tableau d'entiers dont chaque élément en position i vaut
4  * t[i]/sum ou t[i] et l'entier correspondant dans l'array t et sum la
5  * la somme des éléments de t
6  */
7 public static int[] proportion(int[] t) {
8     int sum = 0;
9     int len = t.length;
10    int[] proportion = new int[len];
11
12    for(int i = 0; i < t.length; i++) {
13        sum = sum + tab[i];
14    }
15
16    for(int i = 0; i < t.length; i++) {
17        proportion[i] = i / sum;
18    }
19 }
```

```
20     return proportion;  
21 }
```

code/ExcepRunTErrorEx2.java

□

Débogage

[COURS]

- Le *debugging* (ou *débogage*) est l'approche qu'on suit pour corriger les erreurs de programmation. Notamment, pour corriger les erreurs de non-respect de la spécification.
- Plus précisément, le débogage s'attaque aux *bugs* (ou *bogues*) : défauts d'un programme qui causent un comportement différent du comportement attendu. Pour pouvoir corriger un bug, il faut d'abord le trouver. Le trouver implique l'identification des lignes de code où l'intuition du développeur sur le comportement du programme se sépare du modèle formelle d'exécution
- Bien que pour déboguer vous pouvez toujours vous baser sur des tests empiriques, de la relecture du code, des intuitions, etc, le débogage proprement dit est une application rigoureuse de la *méthode scientifique* à la programmation.
- Le débogage “scientifique” prévoit donc :
 1. *observation* d'une erreur
 2. formulation d'une *hypothèse* qui explique l'erreur
 3. *prévision*, compatible avec l'hypothèse, d'un comportement non encore observé du programme
 4. *expérimentation* pour tester l'hypothèse
 - si l'expérience confirme la prévision (hypothèse confirmée), *raffinement* de l'hypothèse
 - si non (hypothèse réfutée), formulation d'une hypothèse alternative
 5. répétition des points 3–4 jusqu'à l'obtention d'une hypothèse valide qui ne peut pas être raffinée ultérieurement
- pour *observer une erreur* il ne suffit pas de l'avoir “vu passer” quelques fois, il faut être capable de la *reproduire* systématiquement. Vous devez identifier une série d'entrées pour votre programme qui, *d'une façon déterministe*, mènent au comportement incorrecte du programme. Cela s'applique aussi à toutes les étapes d'expérimentation prévues par l’“algorithme” de débogage ci-dessus. La reproductibilité est l'essence de la méthode scientifique !

Exercice 2 (Déboguer, **)

Tester, déboguer “scientifiquement”, et corriger le programme suivant :

```
1 class ExoRevArrayBuggy {  
2  
3     public static String[] reverseArray(String[] t) {  
4         int len = t.length;  
5         for (int i = 0; i < len; i++) {  
6             t[len - i - 1] = t[i];  
7         }  
8         return t;  
9     }  
10  
11    public static void printStringArray(String[] args) {  
12        for (int i = 0; i < args.length; i++) {  
13            System.out.print(args[i] + " ");  
14        }  
15        System.out.print("\n");  
16    }  
17  
18    public static void main(String[] args) {  
19        String[] revArgs = reverseArray(args);  
20        printStringArray(revArgs);
```

```
21 }  
22 }  
23 }
```

code/ExoRevArrayBuggy.java



Tri par insertion

[COURS]

- Un algorithme de tri est un algorithme qui permet de réordonner une collection d'éléments (par exemple un tableau d'entiers) selon un ordre déterminé (p.ex. en ordre croissant). Le tri par insertion est un des algorithmes de tri les plus simples.
- Dans le tri par insertion on maintient une collection d'éléments déjà triés. Cette collection est initialement vide ; à la fin de l'exécution elle contiendra la totalité des éléments, dans le bon ordre.
- À chaque itération, le tri par insertion prend un des éléments encore à trier et l'insère à la bonne position parmi les éléments déjà triés. Le nombre d'éléments déjà triés augmente donc de 1 à chaque itération.

Exercice 3 (Déboguer, ***)

Tester, déboguer "scientifiquement", et corriger le programme suivant :

```
1 public class ExoInsertionSortBuggy {  
2  
3     /* Insère un élément dans un tableau partiellement trié  
4      * Entrée : t un tableau d'entiers partiellement trié, t est  
5      * trié par ordre croissant jusqu'à l'indice (last - 1).  
6      * À la fin de la procédure, t est trié jusqu'à l'indice last.  
7      */  
8     public static void insert(int x, int[] t, int last) {  
9         int i = t.length - 1;  
10        while (i > 0 && t[i] >= x) {  
11            t[i] = t[i-1];  
12            i = i - 1;  
13        }  
14        t[i] = x;  
15    }  
16  
17    /* Trie le tableau d'entiers donné en entrée  
18     * Entrée : t un tableau d'entiers  
19     * À la fin de la procédure, le tableau t est trié.  
20     */  
21    public static void sort(int[] t) {  
22        for (int i = 1; i < t.length; i++) {  
23            insert(t[i], t, i);  
24        }  
25    }  
26  
27  
28    public static void main(String[] args) {  
29        int len = args.length;  
30        int[] t = new int[len];  
31        for (int i = 0; i <= args.length; i++) {  
32            t[i] = Integer.valueOf(args[i]);  
33        }  
34        sort(t);  
35        printIntArray(t);  
36    }  
37}
```

```

38 public static void printIntArray(int[] t) {
39     printString("[ ");
40     for (int i = 1; i < t.length; i++) {
41         System.out.print (t[i]);
42         System.out.print (" ");
43     }
44     System.out.print (""]\n");
45 }
46
47 }

```

code/Ex0InsertionSortBuggy.java



2 Fonctions utilisées

```

1 /*
2  *Attend que l'utilisateur tape une ligne au clavier
3  *Renvoie la chaîne de caractères correspondante
4 */
5 public static String readLine() {
6     return System.console().readLine();
7 }

```

3 DIY

Exercice 4 (Médiane d'un tableau, *)

On s'intéresse ici à des tableaux de nombres entiers à une seule dimension, possédant un nombre impair d'éléments et dont tous les éléments sont différents. On appelle médiane d'un tel tableau T l'élément m de T tel que T contienne autant d'éléments strictement inférieurs à m que d'éléments strictement supérieurs à m .

1. Écrire une fonction nbInf qui, étant donné un tableau d'entiers T et un entier v, renvoie le nombre d'éléments du tableau T strictement inférieurs à v.
2. Écrire une fonction mediane qui, étant donné un tableau T satisfaisant les conditions énoncées, renvoie la position de la médiane dans le tableau T.
3. Écrire une fonction verifArray qui, étant donné un tableau T de nombres entiers, renvoie la valeur booléenne true si le tableau T satisfait effectivement les conditions énoncées et la valeur false si ce n'est pas le cas.
4. Écrire une fonction tabInf qui, étant donné un tableau T, renvoie un tableau de taille 0 si T ne satisfait pas les conditions énoncées et un tableau contenant tous les éléments de T inférieurs à la médiane de T sinon.



Exercice 5 (Spécification, ***)

Écrire un programme Backdoor qui, en boucle infinie, lit une ligne de texte de l'utilisateur à la fois, et l'affiche sur l'écran après avoir converti en majuscule les caractères "a", "b", "c", "d", "e", et "f"; les autres caractères ne sont pas modifiés. Si, par contre, la ligne entrée est le mot secret "3XzRWo" (une "backdoor" insérée par le développeur), le programme doit terminer avec une exception après avoir affiché sur l'écran l'ASCII art suivant :

```

( _ )
( oo )
/-----\/
/ |   || 

```

* / \---/\~\~

Vous avez à disposition : String readLine() pour lire une ligne de texte de l'utilisateur.

À noter : dans ce cas terminer avec une exception (dans un cas très spécifique) fait partie de la spécification du programme. Il ne s'agit donc pas d'une erreur de programmation. Mais comment causer volontairement une exception ?

□

Exercice 6 (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 fonction *numberPoints* qui prend en paramètres 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 fonction *storeScore* qui prend en paramètres 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 fonction *champion* qui prend en paramètre 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 fonction renverra 0 pour signaler l'impossibilité de désigner un champion.

□

Exercice 7 (Recherche dichotomique dans un tableau trié, **)

On souhaite définir une fonction pour rechercher efficacement un élément dans un tableau d'entiers. La fonction prendra en paramètres un tableau d'entiers t et un entier x et renverra un indice i tel que $t[i]$ soit égal à x , si un tel indice existe, et -1 sinon.

1. Définir une fonction *searchA* qui vérifie la spécification ci-dessus et qui renvoie un résultat dès qu'un indice qui satisfait la spécification a été trouvé.
2. Dans quelles situations la fonction *searchA* effectue-t-elle le plus d'accès au tableau ? Dans ce cas, combien de fois accède-t-on au tableau ?
3. On suppose maintenant, et dans la suite de l'exercice, que les tableaux d'entiers que reçoit notre fonction sont toujours triés par ordre croissant (c'est-à-dire que $t[i] \leq t[i+1]$ tant qu'on est dans les bornes du tableau).

*Définir une nouvelle fonction *searchB* qui assure que si la valeur recherchée est hors des valeurs extrêmes du tableau, on ne fait pas plus de deux accès au tableau avant de renvoyer -1.*

4. On conserve la même hypothèse que ci-dessus (les tableaux passés à la fonction sont triés) et on demande de définir une fonction *searchC* qui tire partie du fait que le tableau est trié à partir de l'observation suivante :

Si la valeur recherchée est présente, on la trouvera dans la première moitié ou la seconde moitié du tableau ; la comparaison de cette valeur avec une seule case du tableau suffit à savoir dans quelle moitié rechercher.

*Pour cela, on déclarera dans *searchC* deux variables auxiliaires *imin* et *imax* de type *int* qui serviront à stocker les indices des extrémités de la portion du tableau dans laquelle on doit chercher la valeur.*

Initialement ces indices devront couvrir le tableau entier et on poursuit la recherche jusqu'à ce qu'on ait trouvé la valeur recherchée ou que la portion du tableau dans laquelle rechercher soit de longueur 1.

5. *(***)* Dans quelle situation searchC accède-t-elle le plus de fois aux valeurs du tableau ? Dans ce cas, combien de fois la fonction accède-t-elle aux cases du tableau ?
6. Pour aller plus loin : une fonction peut appeler une autre fonction définie dans le programme. Elle peut en fait s'appeler elle-même. (Est-ce que cela change quelque chose à l'exécution d'appels imbriqués de fonctions tel qu'on l'a vu dans cette séance ?)

On peut utiliser cela pour mieux organiser le code de la fonction de recherche. On définit searchD qui repose sur une fonction auxiliaire comme suit :

```
1 public static int searchD(int[] t, int x) {  
2     return searchAux(t,x,0,t.length - 1);  
3 }
```

où searchAux(t,x,imin,imax) recherche x entre les indices imin et imax du tableau t. Pour cela, searchAux utilisera la même observation que plus haut, mais en vue de rappeler la fonction searchAux avec des bornes de recherche mises à jour, par exemple :

searchAux(t,x,imin,(imin+imax)/2).

□