Exos d’algo en Java

Exercice 1

Définissez une fonction max() qui s'applique à 3 entiers et qui renvoie en résultat la plus grande valeur des 3 :
- version 1 : en utilisant l'instruction conditionnelle
- version 2 : en utilisant l'opérateur conditionnel

Exercice 2

Définissez une fonction ordered() qui s'applique à 3 entiers et qui renvoie en résultat un booléen qui indique si les 3 valeurs fournies sont triées par ordre croissant:
- version 1 : en utilisant l'instruction conditionnelle
- version 2: en utilisant l'opérateur conditionnel
- version 3 : en n'utilisant aucune forme de conditionnelle

Remarque : la version 3 est toujours à privilégier.

Exercice 3

Définissez une fonction entier() qui s'applique à un flottant de type double et qui renvoie en résultat un booléen qui indique si la valeur fournie correspond à une valeur entière :
- version 1 : sans tenir compte de l'approximation des flottants
- version 2 : en tenant compte d'une marge d'erreur de l'ordre d'un millionième

Par exemple, les valeurs flottantes 5.00001 et 5.0000001 ne sont pas considérées comme des valeurs entières dans la version 1. Par contre dans la version 2, la valeur 5.0000001 est considérée comme une valeur entière car ce qui la sépare de 5.0 est inférieur à un millionième.

Exercice 4

Définissez une fonction fact() qui s'applique à un entier de type int et qui renvoie en résultat la factorielle de la valeur fournie (type int aussi) :
- version 1 : en utilisant la récursivité
- version 2 : e n utilisant une itération
- version 3 : en n'utilisant ni récursivité, ni itération

Si la valeur fournie est négative, le résultat est -1. Si la valeur fournie est supérieure à 12, le résultat excède la capacité de int. Dans ce cas, le résultat est -2.

Exercice 5

Définissez une fonction maxtab() qui s'applique à un tableau d'entiers non vide et qui renvoie en résultat la plus grande valeur du tableau.

Exercice 6

Définissez une fonction inverse() qui s'applique à un tableau d'entiers et qui inverse l'ordre de ses éléments. Illustration :

```
int[] tab = { 5, -3, 8, 7, 4, -2, 4, 0 };
inverse(tab); // tab : { 0, 4, -2, 4, 7, 8, -3, 5 }
```

Attention : vous ne devez pas créer de tableau pour réaliser cette fonction.

Exercice 7

Définissez une fonction cptab() qui s'applique à un tableau d'entiers et qui renvoie en résultat une copie du tableau fourni :
- version 1 : en utilisant une boucle de parcours
- version 2 : en utilisant la méthode arraycopy() de la classe System
- version 3 : en utilisant la méthode copyOf() de la classe Arrays
- version 4 : en utilisant la méthode clone des tableaux

Exercice 8

Définissez une fonction cptab2dim() qui s'applique à un tableau à deux dimensions d'entiers et qui renvoie en résultat une copie en profondeur du tableau :
- version 1 : en utilisant une double boucle de parcours
- version 2 : en utilisant une simple boucle de parcours et en réutilisant la fonction cptab de l'exercice précédent

Exercice 9

Définissez une fonction extrairePairs() qui s'applique à un tableau à deux dimensions d'entiers tab et qui renvoie en résultat un tableau à une dimension comportant exactement les nombres pairs de tab.
Illustration :

```
int[] t1 = extrairePairs(new int[][]{{5,-3,8,7}, {4,-2,4,0}});  // t1 : [8, 4, - 2, 4, 0]
int[] t2 = extrairePairs(new int[][]{{5,-3,8,7}, {9,-3,5,1}});  // t2 : [8]
int[] t3 = extrairePairs(new int[][]{{5,-3,9,7}, {9,-3,5,1}});  // t3 : []
```

Exercice 10

Définissez une fonction contient() qui s'applique à deux chaînes de caractères et qui indique si la deuxième est un extrait de la première, indépendamment de la casse.
Exemple :

```
contient("Brocoli", "Brocolis") // faux
contient("Brocoli", "Brocoli")   // vrai
contient("Brocoli", "bro")       // vrai
contient("Brocoli", "COL")       // vrai
contient ("Brocoli", "CLO")      // faux
```

Rappel des méthodes de String utiles pour cet exercice :

```
boolean equalsIgnoreCase()  // equals()  indépendant de la casse
int length()                // taille de la chaine
String substring( int d, int f )  // extrait de d (inclus) à f (exclu)
```

Exercice 11

Définissez une fonction separe() qui s'applique à une chaîne de caractères et qui renvoie une chaîne constituée d'abord des voyelles puis de tous les autres caractères sauf des espaces. Toutes les lettres doivent apparaître en majuscules dans le résultat.
Exemple :

```
separe("Allez_les_bleus_!") // "AEEEULLZLSBLS !"
```

Rappel des méthodes de String utiles pour cet exercice :

```
String toUpperCase()  // mise en majuscules
int length            // taille de la chaine
```

char charAt (int i) // caractère à l'indice i

Exercice 12

Un palindrome est un mot que l'on peut lire indifféremment de gauche à droite ou de droite à gauche, comme par exemple ressasser. Dans un premier temps, définissez une fonction palindrome qui s'applique à une chaîne de caractères et qui indique si cette chaîne correspond à un palindrome.
Un palindrome n'est pas nécessairement réduit à un mot. Il peut s'agir d'une phrase ou même un texte complet. Dans ce cas, on ignore les espaces, les apostrophes et la ponctuation, et on ne tient pas compte de la casse, des accents, des trémas et autres cédilles.
Exemples :
- Engage le jeu que je le gagne.
- Oh, cela te perd répéta l'écho !

Adaptez la fonction palindrome() afin qu'elle puisse reconnaître les palindromes ainsi définis.
Pour cette version étendue, vous pouvez utiliser la méthode replaceAll() de String :

```
// Remplace les sous-chaines correspondant à e par r :
String replaceAll( String e, String r )  // e : expression régulière
// Exemple :
"AbCdEf". replaceAll ("[ABCDEF]", "--" ) // "--b--d--f"
```

Exercice 13

Complétez la méthode genTab1() suivante qui crée un tableau d'entiers de n cases contenant des valeurs aléatoires comprises entre 0 et 100 :

```
public static int[] genTab1( int nbCases ) { /* . . . */ }
```

Exemple d'utilisation :

```
int[] tab1Test = genTab1(10); // génération d'un tableau de 10 cases
System.out.println(Arrays.toString( tab1Test )); // affichage
```

Exemple d'affichage obtenu à l'exécution (les nombres varient à chaque exécution) :
[44, 42, 90, 65, 86, 19, 7, 69, 99, 94]

Pour définir les valeurs aléatoires, vous devez utiliser la méthode statique random de la classe Math qui fournit une valeur flottante comprise entre 0 et 1.

Exercice 14

Complétez la méthode genTab2() suivante qui fournit un tableau à deux dimensions d'entiers de nbl lignes et nbc colonnes, et dont chaque case contient des valeurs aléatoires comprises entre 0 et 100 :

```
public static int[][] genTab2 (int nbl, int nbc) { /* . . . */ }
```

Exemple d'utilisation :

```
int[][] tab2Test = genTab2( 3, 5 ); // tableau de 3x5 cases
System.out.println( Arrays.deepToString( tab2Test )); // affichage
```

Exemple d'affichage obtenu à l'exécution (les nombres varient à chaque exécution) :
[[80, 94, 72, 78, 95], [59, 50, 97, 64, 80], [34, 97, 41, 66, 95]]

Pour programmer cette méthode, vous devez utiliser la méthode genTab1() de l'exercice précédent.

Exercice 15

Complétez la méthode swap() suivante qui échange le contenu des cases d'indices i et i+1 d'un tableau fourni en paramètre :

```
public static void swap ( int[] tab, int i ) { /* * / }
```

Exemple d'utilisation :

```
int[] tab = {2, 7, 0, 5, 3, 1, 9};
System.out.println(Arrays.toString( tab )) ; // affichage avant
swap (tab, 2); // échange des cases d'indice 2 et 3
System.out.println(Arrays.toString( tab )) ; // affichage après
```

À l'exécution, on obtient l'affichage suivant (les valeurs 0 et 5 sont échangées) :

```
[2, 7, 0, 5, 3, 1, 9 ]
[2, 7, 5, 0, 3, 1, 9 ]
```

Exercice 16

Complétez la méthode bulle() suivante qui s'applique à un tableau à une dimension d'entiers et à un entier n. Cette méthode déplace dans la case d'indice n la plus grande valeur contenue dans le tableau entre les indices 0 et n (inclus) :

```
public static void bulle(int[] tab, int n) { /* . . . */ }
```

Exemple d'utilisation :

```
int[] tabBulle = {2, 7, 0, 5, 3, 1, 9};
System.out.println(Arrays.toString( tabBulle )); // affichage avant
bulle( tabBulle, 4); // maximum de [0 ; 4] placée à l'indice 4
System.out.println(Arrays.toString( tabBulle )); // affichage après
```

À l'exécution, on obtient l'affichage suivant. La valeur 7 est placée dans la case d'indice 4. Il s'agit bien de la plus grande valeur parmi les cases indicées de 0 à 4 :

```
[2, 7, 0, 5, 3, 1, 9 ]
[2, 0, 5, 3, 7, 1, 9 ]
```

Pour programmer cette méthode, vous devez utiliser la méthode swap de l'exercice précédent.

Exercice 17

Complétez la méthode triBulle1() suivante qui s'applique à un tableau à une dimension d'entiers et qui trie le tableau en appliquant le principe du tri à bulles :

```
public static void triBulle1( int[] tab ) { / *. . . * / }
```

Le but est de placer la plus grande valeur du tableau dans la dernière case, puis de placer la deuxième plus grande valeur du tableau dans l'avant-dernière case, et de continuer ainsi jusqu'à la première case qui doit alors contenir la plus petite valeur du tableau.

Exemple d'utilisation :

```
int[] tabBulle = {2, 7, 0, 5, 3, 1, 9};
System.out.println(Arrays.toString( tabBulle )); // affichage avant
triBulle1( tabBulle ); // tri
System.out.println(Arrays.toString( tabBulle )); // affichage après
```

A l'exécution, on obtient l'affichage suivant :
[2, 7, 0, 5, 3, 1, 9]
[0, 1, 2, 3, 5, 7, 9]

Pour programmer cette méthode, vous devez utiliser la méthode bulle de l'exercice précédent.

Exercice 18

Complétez la méthode triBulleLignes() suivante qui trie ligne par ligne un tableau à deux dimensions d'entiers :

```
public static void triBulleLignes( int[][] tab) { /* . . . * / }
```

Exemple d'utilisation :

```
int[][] tab2Test = genTab2(3, 5);
System.out.println(Arrays.deepToString( tab2Test )); // affichage avant
triBulleLignes( tab2Test ); // tri de chaque ligne
System.out.println(Arrays.deepToString( tab2Test )); // affichage après
```

Exemple d'affichage obtenu à l'exécution (les nombres varient à chaque exécution) :
[[39, 70, 96, 32, 55], [18, 49, 47, 65, 0], [99, 81, 24, 73, 78]]
[[32, 39, 55, 70, 96], [0, 18, 47, 49, 65], [24, 73, 78, 81, 99]]

Pour programmer cette méthode, vous devez utiliser la méthode triBulle1() de l'exercice précédent.

Exercice 19

Complétez la méthode triBulleGlobal() suivante qui s'applique à un tableau à deux dimensions d'entiers et qui trie le tableau dans sa globalité, de la première case de la première ligne à la dernière case de la dernière ligne :

```
public static void triBulleGlobal( int[][] tab ) { /* . . . */ }
```

Exemple d'utilisation :

```
int[][] tab2Test = genTab2(3, 5) ;
System.out.println(Arrays.deepToString( tab2Test )); // affichage avant
triBulleGlobal(tab2Test); // tri global
System.out.println(Arrays.deepToString( tab2Test )); // affichage après
```

Exemple d'affichage obtenu à l'exécution (les nombres varient à chaque exécution) :
[[68, 54, 16, 53, 71], [32, 68, 24, 32, 53], [71, 38, 70, 60, 49]]
[[16, 24, 32, 32, 38], [49, 53, 53, 54, 60], [68, 68, 70, 71, 71]]

Pour programmer cette méthode, vous devez adaptez les méthodes triBulle1(), bulle() et swap() de manière à ce qu'elles s'appliquent à des tableaux à deux dimensions d'entiers par l'intermédiaire d'un indice global, i.e. un indice qui permet de parcourir en séquence toutes les cases du tableau comme s'il s'agissait d'un simple tableau à une dimension.

Exercice 20

Complétez la méthode transpose() suivante qui s'applique à un tableau à deux dimensions d'entiers et qui renvoie en résultat un nouveau tableau à deux dimensions d'entiers correspondant à la transposée du tableau fourni :

```
public static int[][] transpose(int[][] tab) { /* . . . */ }
```

La transposée est obtenue en changeant les lignes et les colonnes (par symétrie axiale par rapport à la diagonale principale). Illustration :

```
int[][] tab2Test = genTab2(3 , 5);
System.out.println(Arrays.deepToString(tab2Test)); // affichage avant
tab2Test = transpose(tab2Test) ;
System.out.println(Arrays.deepToString(tab2Test)); // affichage après
```

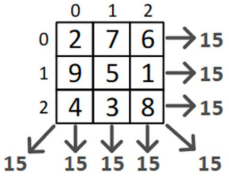
Exemple d'affichage obtenu à l'exécution (les nombres varient à chaque exécution) :

```
[[10, 85, 94, 7, 47], [90, 98, 82, 90, 53], [46, 29, 37, 14, 98]]
[[10, 90, 46], [85, 98, 29], [94, 82, 37], [7, 90, 14], [47, 53, 98]]
```

La transposée du tableau à 3 lignes et 5 colonnes comporte 5 lignes et 3 colonnes.
Chaque ligne d'indice i du tableau final correspond à la colonne d'indice i du tableau initial.

Exercice 21

Un carré magique est un tableau carré d'entiers positifs tel que les sommes de chaque ligne, les sommes de chaque colonne et les sommes des deux diagonales sont identiques. Dans l'exemple suivant, la somme de chaque ligne, de chaque colonne et de chaque diagonale est égale à 15 :



Un carré magique est dit normal s'il contient toutes les valeurs de 1 à N, avec N: nombre de cases du tableau. Par exemple, le carré ci-dessus est normal car il comporte 9 cases et contient tous les nombres allant de 1 à 9.

Complétez la méthode estCarre() suivante qui indique si un tableau à deux dimensions d'entiers comporte le même nombre de lignes que de colonnes. Si le tableau est carré, la méthode renvoie 1. Elle renvoie 0 sinon :

```
public static int estCarre( int[][] tab ) { / * . . . * / }
```

Illustration :

```
estCarre(new int[][] {{11, 20, 38}, (56, 65, 74}}); // 0 (pas carré)
estCarre(new int[][] {{11, 20}, {65, 74}});          // 1 (carré)
```

Exercice 22

Complétez la méthode estCarreMagique() suivante qui indique si un tableau d'entiers est un carré magique. Si le tableau est un carré magique, la méthode renvoie 2. Si le tableau est carré mais n'est pas magique, elle renvoie 1. Dans les autres cas, elle renvoie 0 :

```
public static int estCarreMagique( int[][] tab ) { /* . . . */ }
```

Pour programmer cette méthode, vous devez utiliser la méthode estCarre() de l'exercice précédent.

Exercice 23

Complétez la méthode estCarreMagiqueNormal() suivante qui indique si un tableau d'entiers est un carré magique normal. Si c'est le cas, la méthode renvoie 3. Si le tableau est un carré magique non normal, la méthode renvoie 2. Si le tableau est carré mais n'est pas magique, elle renvoie 1. Dans les autres cas, elle renvoie 0 :

```
public static int estCarreMagiqueNormal( int[][] tab ) { /* */ }
```

Pour programmer cette méthode, vous devez utiliser la méthode estCarreMagique() de l'exercice précédent.