Université Paris 7 - Denis Diderot
IF1 - Examen du 3 janvier 2006 - Durée : 3 heures
Documents, calculettes, ordinateurs, téléphones non autorisés

---

Exercice 1.

1.1. Exécuter à la main le programme suivant :

```
import fr.jussieu.script.*;
class Param1 {
   static int fonc1(int a, int b, int c) {
      a = a + 1; b = b + 2; c = a + b; return 2 * c; }
   public static void main(String[ ] arg) {
      int a = 1, b = 2, c = 3, d;
      d = fonc1(a,b,c);
      Deug.println(a); Deug.println(b);
      Deug.println(c); Deug.println(d);
   }
}
```

1.2. Exécuter à la main le programme suivant :

```
import fr.jussieu.script.*;
   class Param2{
      static void fonc2(int a, int b, int[ ] t ) {
         a = a + 1; b = b + 2; t[0] = t[0] + 3; t[1] = t[1] + 4;}
      public static void main(String[ ] arg) {
         int[] t = new int[2];
         t[0] = 1; t[1] = 2;
         fonc2(t[0], t[1] , t);
         Deug.println(t[0]); Deug.println(t[1]);
      }
   }
```

Exercice 2. *Aucun tableau ne sera utilisé dans cet exercice.*

Écrire un programme Java qui lit une liste d'**au moins deux** entiers strictement positifs dont le nombre n'est pas connu a priori et affiche le deuxième plus grand entier de cette liste. Les nombres seront lus et traités successivement et la fin de la liste sera marquée par un entier négatif ou nul ne faisant pas partie de la liste. Dans le cas où l'utilisateur entre moins de deux entiers, un message d'erreur sera affiché.

Exemples :

- pour la suite de nombres $3, 5, 7, 1, 3, 8$, on obtient 7 ;
- pour la suite de nombres $8, 2, 7, 8, 3, 6$, on obtient 8 (le plus grand élément figure deux fois dans la liste).

Exercice 3.

3.1. Écrire une méthode Java, **ayant comme paramètre** une référence `a` de tableau d'entiers, **qui renvoie** `true` si et seulement si le tableau référencé par `a` est strictement croissant (c'est-à-dire, pour tout indice i, $\mathtt{a}[i] < \mathtt{a}[i+1]$).

3.2. Écrire une méthode Java, **ayant comme paramètres** deux références `a` et `b` de tableaux d'entiers strictement croissants (au sens défini en 3.1), **qui renvoie** `true` si et seulement si les ensembles de valeurs contenues dans les tableaux référencés par `a` et `b` sont disjoints, *i.e.* si étant donnés deux indices quelconques i et j, $\mathtt{a}[i] \neq \mathtt{b}[j]$.

3.3. Écrire une méthode Java, **ayant comme paramètres** deux références `a` et `b` de tableaux strictement croissants et disjoints, **qui renvoie** une référence sur l'unique tableau strictement croissant contenant exactement tous les éléments de `a` et tous ceux de `b`.

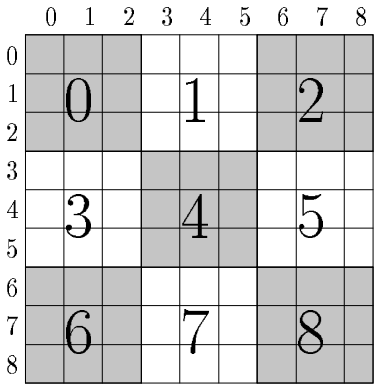
3.4. Écrire un **programme Java** qui :

- demande à l'utilisateur de rentrer deux tableaux `a` et `b` au clavier : pour chaque tableau, on demandera dans un premier temps d'entrer le nombre d'éléments qu'il contient (ce qui permettra l'allocation du tableau), puis la valeur de ses éléments ;
- vérifie qu'ils sont strictement croissants et disjoints (si ce n'est pas le cas, on affichera un message d'erreur et on terminera l'exécution du programme) ;
- affiche l'unique tableau strictement croissant contenant tous les éléments de `a` et de `b`.

On pourra bien sûr se servir des méthodes écrites aux questions précédentes.

Exercice 4. *Autour du sudoku*

On s'intéresse à quelques vérifications et manipulations de tableau de 9 lignes et 9 colonnes d'entiers. Un tel tableau peut être décomposé en 9 sous-carrés disjoints de côté 3 (3 lignes et 3 colonnes), comme indiqué et numérotés dans la partie gauche de la figure suivante :



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

On rappelle qu'une suite de 9 nombres est une permutation de l'ensemble $\{1,2,3,4,5,6,7,8,9\}$ (en abrégé $[1:9]$) si elle contient exactement un exemplaire de chacun des nombres (comme par exemple $7,4,3,8,6,2,9,1,5$).

Un tableau de 9 lignes et 9 colonnes est une solution du jeu appelé sudoku (par exemple le carré de droite de la figure) si :

- chacune des 9 lignes est une permutation de $[1:9]$;
- chacune des 9 colonnes est une permutation de $[1:9]$;
- chacun des 9 sous-carrés de côté 3 que nous avons identifiés contient exactement une fois chacun des nombres de $[1:9]$ (il en constitue donc une permutation quel que soit l'ordre dans lequel on le parcourt).

- Vous pourrez traiter une question en utilisant les méthodes des questions précédentes même si vous ne les avez pas écrites.

- Dans l'écriture des méthodes, on supposera que les paramètres transmis sont corrects. Il n'est donc pas demandé de faire de vérifications.

4.1. Quelques méthodes utiles.

4.1.1. Écrire une méthode Java `ligne`, **ayant comme paramètre** un numéro de sous-carré dans la numérotation décrite, **qui renvoie** l'indice de ligne de son coin supérieur gauche dans un tableau carré de côté 9. Par exemple, pour le carré de numéro 5 la méthode renvoie 3.

4.1.2. Écrire une méthode Java `colonne`, **ayant comme paramètre** un numéro de sous-carré, **qui renvoie** l'indice de colonne de son coin supérieur gauche dans le carré de côté 9 sous-jacent. Par exemple, pour le carré de numéro 5 la méthode renvoie 6.

4.1.3. Écrire une méthode Java `sousCarre`, **ayant comme paramètres** des indices de ligne i et de colonne j dans un carré de côté 9, **qui renvoie** le numéro du sous-carré auquel la case correspondante appartient. Ainsi, pour les valeurs 5 et 7, la méthode renvoie 5.

4.1.4. Écrire une méthode Java `dansLigne`, **ayant comme paramètres** une référence de tableau carré d'entiers, un indice de ligne et un entier val, **qui renvoie** `true` si l'entier val est dans la ligne d'indice donné et `false` sinon.

4.1.5. Écrire une méthode Java `dansColonne`, **ayant comme paramètres** une référence de tableau carré d'entiers, un indice de colonne et un entier val, **qui renvoie** `true` si l'entier val est dans la colonne d'indice donné et `false` sinon.

4.1.6. Écrire une méthode Java `dansSousCarre`, **ayant comme paramètres** une référence de tableau carré d'entiers, un numéro de sous-carré et un entier val, **qui renvoie** la valeur `true` si l'entier val est dans le sous-carré de numéro donné et `false` sinon.

4.1.7. Écrire une méthode Java `estPerm`, **ayant en paramètre** une référence de tableau d'entiers, **qui renvoie** `true` si ce tableau est une permutation de $[1:9]$ et `false` sinon.

4.2. Écrire une méthode Java `verifLig`, **ayant comme paramètres** une référence de tableau carré d'entiers et un indice de ligne i, **qui renvoie** `true` si le contenu de la ligne i est une permutation de $[1:9]$ et `false` sinon.

4.3 Écrire une méthode Java `verifCol`, **ayant comme paramètres** une référence de tableau carré d'entiers et un indice de colonne j, **qui renvoie** `true` si le contenu de la colonne j est une permutation de $[1:9]$ et `false` sinon.

4.4. Écrire une méthode Java `verifSousCarre`, **ayant comme paramètres** une référence de tableau carré d'entiers et un numéro de sous-carré x, **qui renvoie** `true` si chacun des nombres de $[1:9]$ figure exactement une fois dans le sous-carré de numéro x et `false` sinon.

4.5. Écrire une méthode Java `verifTableau`, **ayant comme paramètre** une référence de tableau carré d'entiers, **qui renvoie** `true` si le tableau est une solution du jeu et `false` sinon.

4.6. On s'intéresse maintenant à une solution partielle du jeu, c'est-à-dire un tableau de référence t dans lequel certaines cases n'ont pas encore été remplies : on supposera qu'une telle case $t[i,j]$ contient la valeur 0.

Écrire une méthode Java `placer`, **ayant comme paramètres** :

- une référence t de tableau carré d'entiers contenant des nombres compris entre 0 et 9,
- des indices de ligne i et de colonne j,
- un entier x compris entre 1 et 9,

et **qui renvoie** `true` si les conditions suivantes sont satisfaites et `false` sinon :

- l'élément $t[i,j]$ est nul,
- il est possible de lui donner la valeur x compte tenu des valeurs déjà présentes dans la ligne i, la colonne j et le sous-carré auquel l'élément repéré par i et j appartient.

4.7. Écrire une méthode Java `tabValeurs`, **ayant comme paramètres** :

- une référence t de tableau carré d'entiers contenant des nombres compris entre 0 et 9,
- des indices de ligne i et de colonne j dont on supposera qu'ils sont corrects et correspondent à une case non remplie,

et **qui renvoie** la référence d'un tableau contenant l'ensemble des valeurs qu'il est possible de donner à $t[i,j]$, compte tenu de ce que contiennent la ligne i, la colonne j et le sous-carré auquel la case repérée par i et j appartient (la fonction pourra renvoyer `null` ou une référence sur un tableau de taille 0 si la case est remplie ou si aucun entier ne peut y être écrit).