

Programmation Orientée Objet (Java)

TD 2 JAVA : tableaux et méthodes statiques**Exercice 1 :**

Les tableaux manipulés par la classe `Tableau` seront des tableaux à 1 dimension de nombres entiers positifs (de type `int`). Les tableaux pourront contenir des "cases vides" dont la valeur sera `-1`. Toutes les "cases vides" seront à la fin du tableau.

Ecrivez une classe `Tableau` qui contient les méthodes suivantes :

1. Méthode `initialise` qui initialise à `-1` tous les éléments d'un tableau passé en paramètre (toutes les cases du tableau sont vides).
2. Méthode `afficheTableau` qui affiche tous les éléments d'un tableau passé en paramètre (y compris les valeurs `-1`) en utilisant **une boucle `foreach`**.
3. une méthode `maxTableau` qui **renvoie** le plus grand entier d'un tableau d'entiers passé en paramètre.
4. Méthode `ajouterElement` qui ajoute un élément au tableau passé en paramètre et retourne un booléen. La méthode ne fera rien si le tableau est déjà plein ; en ce cas, elle renverra le booléen `false`. Vous pouvez tester votre méthode en utilisant `afficheTableau`. Testez les différents cas : ajout dans un tableau vide, en dernière position, dans un tableau plein.
5. Méthode `rechercher` qui recherche la position d'un entier dans un tableau. La méthode renvoie `-1` si l'élément n'est pas dans le tableau. La méthode renvoie `0` si l'entier est dans le premier élément du tableau (même si le tableau contient d'autres éléments égaux à l'entier recherché), et plus généralement, renvoie `n - 1` si l'entier est dans la $n^{\text{ème}}$ position du tableau.
6. La méthode `fusion` qui prend en paramètre deux tableaux et renvoie un nouveau tableau.

```
public static int[] fusion(int[] t1, int[] t2)
```

la fusion du tableau `[2 44 4 0 8 -1]` et du tableau `[81 -1 -1 -1]` donne le tableau `[2 44 4 0 8 81 -1 -1 -1 -1]`

7. Testez dans les méthodes de la classe `Tableau` dans la méthode `main` de la classe.

8. Réécrire une classe `TableauBis` ayant comme attribut d'instance un tableau `int T[]`. Définir toutes les méthodes de la classe `Tableau`, pour la méthode `fusion` elle prendra **un seul tableau `t1` en paramètre** et modifiera le tableau d'instance `T` en fusionnant les éléments de `t1` avec `T`.

Exercice 2: matrices et chaînes de caractères

1. Écrire, dans une classe ***Chaine*** une méthode ***toTableau()*** prenant une chaîne de caractères `s` en argument et copiant chacun de ses caractères dans un tableau (de caractères). Le programme doit en outre afficher chaque élément du tableau en séparant chacun d'eux par un espace : si c'est par exemple `toto`, le résultat sera :

`t o t o`

2. On étend maintenant la méthode ***toTableau()*** en la surchargeant de manière à ce qu'elle prenne en argument un nombre quelconque de chaînes de caractères et qu'elle manipule une matrice `m` (un tableau de tableaux) de caractères : le $i^{\text{ème}}$ caractère de la $j^{\text{ème}}$ chaîne sera placé à la position (i, j) de la matrice. Si les chaînes en argument sont Jean, Pierre et Valentine, le résultat devra être, après parcours de `m` :

mot 0 : J e a n

mot 1 : P i e r r e

mot 2 : V a l e n t i n e

Méthodes statiques

	d'instance	de classe
Attribut	Chaque objet encapsule une variable distincte (dont la valeur est propre à cet objet).	Une unique variable attachée à la classe, "partagée" par toutes les instances de la classe.
Méthode	La méthode est exécutée sur un objet donné (habituellement, ses instructions de traitement font référence à d'autres membres de l'instance)	La méthode est exécutée indépendamment de tout objet (elle peut éventuellement faire référence à d'autres membres de classe).

Exercice 3 : Indiquer l'erreur et un moyen simple de la corriger.

```
public static first(float x) { return x*2.5; }
```

Exercice 4 : Donner l'affichage produit par le programme suivant :

```
public class Modification {
    public static void modif1(int x) { x+=2; }
    public static void modif2(int x) { x=5; }

    public static void main(String[] args) {
        int x=24;
        modif1(x);
        System.out.println(x);
        x=24;
        modif2(x);
        System.out.println(x);
    } }
```

Exercice 5: On considère les deux classes suivantes :

```
public class ClassA {
    public static int f(int a) {return 2*a;}
    public static int g(int a) {return a+f(a/2);}
}

public class ClassB {
    public static int f(int a) { return 3*a; }
    public static int g(int a) {
        return ClassA.f(2*a); }
    public static void main(String[] args) {
        System.out.println(f(2));
        System.out.println(g(3));
        System.out.println(ClassA.f(2));
        System.out.println(ClassA.g(3));
    } }
```

Quel est l'affichage produit par la classe ClassB.

Exercice 6 : Coefficients binomiaux

Écrire programme Java intitulé BinomialCoefficients qui permet de calculer et afficher tous les C_n^k pour tout entier inférieur ou égal à un entier donné (saisi par l'utilisateur) en s'appuyant sur la propriété récursive des coefficients binomiaux (formule de Pascal) (voir le lien http://fr.wikipedia.org/wiki/Coefficient_binomial, la section « Propriété récursive des coefficients binomiaux d'entiers ») :

$$\binom{n}{k} + \binom{n}{k+1} = \binom{n+1}{k+1}, \forall n, k > 0$$

Les coefficients calculés doivent être stockés dans un tableau à deux dimensions C en veillant à bien respecter les notations adoptées pour les coefficients binomiaux (autrement dit, la case C[n][k] doit contenir la valeur de C_n^k). Les allocations doivent être effectuées au fur et à mesure en fonction du nombre d'éléments à stocker à chaque niveau du tableau. Un affichage possible lorsque l'utilisateur demande le calcul de tous les coefficients binomiaux pour les entiers inférieurs ou égaux à 8 est donné par l'exemple suivant :

```
Saisir un entier : 8
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
[1, 5, 10, 10, 5, 1]
[1, 6, 15, 20, 15, 6, 1]
[1, 7, 21, 35, 35, 21, 7, 1]
[1, 8, 28, 56, 70, 56, 28, 8, 1]
```

Les tableaux en Java 5

Java 5 fournit un moyen plus court de parcourir un tableau. L'exemple suivant réalise le traitement sur monTableau :

```
for (int element : monTableau){
    // traitement
}
```

Attention néanmoins, la variable element contient une copie de monTableau[i]. Avec des tableaux contenant des variables primitives, toute modification de element n'aura aucun effet sur le contenu du tableau.

```
// Vaine tentative de remplir tous les éléments du tableau avec la valeur 10
for(int element : monTableau){
    element=10;
}
```

```
// La bonne méthode :
for(int i=0; i < monTableau.length; i++){
    monTableau[i]=10;
}
```

Exercice 7: String & StringBuilder

```
public class ExoString {
    public static void main(String[] args) {
        String str = "This is text";
        String s2 = str.replace('i', 'x');
        System.out.println("- s2=" + s2);
        String s3 = str.replaceAll("is", "abc");
        System.out.println("- s3=" + s3);
        String s4 = str.replaceFirst("is", "abc");
        System.out.println("- s4=" + s4);
        String s5 = str.replaceAll("is|te", "+");
        System.out.println("- s5=" + s5);
        String s2bis = str.toLowerCase();
        System.out.println("- s2bis=" + s2bis);
        String s3bis = str.toUpperCase();
        System.out.println("- s3bis=" + s3bis);
        boolean swith = str.startsWith("This");
        System.out.println("- 'str' startsWith This
? " + swith);
        str = " \t Java is good! \t \n ";
```

```
System.out.println("- str=" + str);
String s4bis = str.trim();
System.out.println("- s4bis=" + s4bis);
StringBuilder sb= new StringBuilder(10);
System.out.println(sb+ " capacité
="+sb.capacity());
sb.append("Hello...");
System.out.println(sb+ " capacité
="+sb.capacity());
sb.append('!');
sb.insert(8," java");
System.out.println(sb+ " capacité
="+sb.capacity());
sb.delete(5, 8);
System.out.println(sb+ " capacité
="+sb.capacity());
    }}
```