TP 13 Programmes Java

D'abord, démarrer un terminal
Taper les commandes suivantes pour commencer. Cela pourrait vous aider à vous mieux organiser votre travail ; vous pourriez écrire tous vos fichiers `Java` dans un dossier `tp12`.

```
cd               #se placer dans le dossier personnel
mkdir tp13       #créer un nouveau dossier
cd tp13          #se placer dans le nouveau dossier tp12
kate  Exo1.java& #editer le fichier, n'oublier pas le '&' sinon le terminal reste bloqué
```

Vous pourriez avoir besoin de faire `Kate` afficher les numéros de lignes pour corriger les erreurs : taper `F11`, ou `Configuration→Configurer Kate→Apparence→Onglet Bordures→Afficher les numéros de lignes`.

Exercice 1 Écrire dans le fichier `Exo1.java` un programme `Java` qui affiche le texte ci-dessous (3×3 étoiles) :

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

Il suffit d'ajouter trois lignes au code ci-dessous.

```
class Exo1{
    public static void main(String[] args){
        ....
    }
}
```

Note (bonus) : On peut réaliser cela en une seule ligne grâce au caractère « `\n` » qui représente un saut de ligne. Exemple : `System.out.println("un \n deux");` affiche deux lignes.

Exercice 2 Écrire une classe `Exo2` dans un fichier `Exo2.java`. Ajouter une fonction statique `somme(float x, double y)` qui renvoie la somme $x + y$ comme un double. Appeler cette fonction dans le programme principal et vérifier si le calcul est correct.

Exercice 3 Écrire une classe `Exo3` dans un fichier `Exo3.java` pour tester si un entier saisi par l'utilisateur appartient au tableau. Il faut écrire **que la fonction main(...)** ; le tableau est initialisé au début du main :

```
int[] tab = {12, 15, 13, 10, 8, 9, 13, 14};
```

Indication : Utiliser une variable booléene `valSaisieExiste` qui sera au début `false`. Mais si on trouve la valeur saisie, on met `valSaisieExiste=true`. Cette logique est illustré par le pseudo-code suivant :

```
valSaisieExiste = false
for i = 0 to n-1
    if valSaisie==tab[i]
            valSaisieExiste =true
if valSaisieExiste
    print "trouvé"
else
    print "pas trouvé"
```

Exercice 4 Écrire une classe `Exo4` dans un fichier `Exo4.java`. Ajouter une fonction `sommeTab(int[] tab)` qui calcule la somme des éléments du tableau `tab`. Il faut utiliser une boucle `for`. Tester la nouvelle fonction dans le main.

Exercice 5 Écrire une classe `Exo5` dans un fichier `Exo5.java`. Ajouter une fonction

`sommeTabVal(int val, int[] tab)`

qui vérifie si la somme des éléments du tableau `tab` est égale à `val`. Cette fonction doit envoyer un `boolean`. Dans cette fonction `sommeTabVal` il faut faire appel à la fonction `sommeTab` écrite à l'exercice précédent. Tester la nouvelle fonction dans le main.

Exercice 6 Écrire une classe `Exo6` dans un fichier `Exo6.java`. Ajouter une méthode (fonction qui renvoie `void`) nommé `toutesLesSommes(int n)` qui calcule toutes les sommes ci-dessous :

```
1
1+2
1+2+3
1+2+3+4
...
1+2+3+4...+ n
```

Il faut afficher que le résultat de chaque somme. Par exemple, pour $n = 4$, il faut afficher

```
1
3
6
10
```

Exercice 7 Écrire une classe `Exo7` dans un fichier `Exo7.java`. Ajouter une méthode (fonction qui renvoie `void`) nommé `lesProduits(int n)` qui calcule tous les produits :

```
1*2/2
2*3/2
3*4/2
4*5/2
...
n*(n+1)/2
```

Il faut afficher que le résultat de chaque produit. Par exemple, pour $n = 4$, il faut afficher

```
1
3
6
10
```

En fait, cette fonction devrait finir par afficher les même résultats que la fonction de l'exercice précédent.

Exercice 8 Un carré magique (simplifié) est une matrice 3×3 avec 9 entiers strictement positifs (`mat[i][j]`> 0) disposés de sorte que la somme sur chaque ligne et chaque colonne soit la même. Par exemple, la matrice ci-après a une somme de 10 sur chaque ligne et chaque colonne.

$$\texttt{mat} = \begin{bmatrix} 2 & 5 & 3 \\ 3 & 1 & 6 \\ 5 & 4 & 1 \end{bmatrix}$$

Écrire une fonction `magique(int[][] matrice)` qui renvoie `true` si la matrice passée comme argument représente un carré magique ou `false` sinon. On suppose que la matrice a une taille de 3×3.

Exercice 9 Continuer le programme précédent, en utilisant "copier-coller" pour copier la fonction `magique(...)` dans le nouveau programme. Appeler cette fonction après avoir lu les 9 valeurs à partir d'un fichier `mat.txt`.

Exercice 10 Écrire une classe `Exo10` avec une fonction `double noteFinale(double tp, double exam)` renvoie la note finale calculée avec les règles suivantes :

(1) La note finale vaut 0 si la note d'examen est inférieure à 10 (échec)
(2) La note finale est la note d'examen si `exam` ≥ 10 et `exam`$\geq$`tp`.
(3) La note finale est la moyenne entre la note d'examen et la note de TP si on a 10 <= `exam` < `tp`.
Indication : vous n'avez pas à tester cette relation/condition explicitement. La fonction doit pouvoir arriver au point (3) *uniquement si* elle n'appelle pas `return` au points (1) ou (2). Si elle arrive au point (3), alors la condition 10 <= `exam` < `tp` est surement vérifiée. La fonction peut s'écrire en 5-7 lignes !!

Exercice 11 Continuer le programme précédent et modifier la méthode `main(...)` pour que le programme demande à l'utilisateur de saisir la valeur des deux notes (exam et TP) ; ensuite affichez la note finale. Le programme devrait être capable de gérer le cas où l'utilisateur ne saisit pas une valeur de type `double` acceptée par le langage. Par exemple, si l'utilisateur saisit « toto » ou « 6,223 », il faut re-demander la saisie (au moins une fois suffit). Il faut aussi tester que les notes représentent des valeurs de 0 à 20.