TP 14 Java

Exercice 1 Utiliser le code ci-dessous pour écrire un fichier `Exo1.java` qui permet d'afficher la valeur minimale du tableau `tab`. Il faut écrire que la fonction principale `main`.

```
class ....{
    public ...  void main  (String[]...){
        int[] tab = {3, 9, 23,1,29,134,12,667,43,2,120,32,123};
        int min = tab[0];
        for(int i ;   i<tab.length; ...)
            if(.....)
                min=tab[i];
    }
}
```

N'hésitez pas à consulter la dernière page de ce document pour vous habituer avec les règles d'indentation à utiliser pendant ce TP, à l'examen ou au TP noté de la semaine prochaine.

Exercice 2 Continuer le programme précédent dans un nouveau fichier `Exo2.java` pour ajouter deux fonctionnalités qui devraient donner le même résultat :

1. Utiliser la même boucle for (dans le `main`) pour calculer pas seulement la valeur minimale mais aussi le nombre d'apparitions de la valeur minimale.
2. Ajouter une fonction `int nbVals(int[] tab, int valeur)` qui renvoie le nombre d'apparitions de la valeur `valeur` dans le tableau `tab`. Utiliser cette fonction pour afficher le nombre d'apparitions de la valeur `min` calculée dans le `main`.

Exercice 3 Écrire une classe `Exo3` avec une fonction une fonction `i_puissance_i(int i)` qui renvoie i^i à l'aide d'une boucle `for`.

Exercice 4 Écrire une classe `Exo4` avec une fonction `sommePuissances(int n)` qui renvoie la somme $1^1+2^2+3^3+\ldots n^n$. Par exemple, pour $n = 5$, il faut renvoyer $1^2+2^2+3^3+4^4+5^5 = 1+4+27+256+3125 = 3413$. N'hésitez pas à utiliser la fonction `i_puissance_i(int i)` de l'exercice précédent pour écrire la nouvelle fonction.

Exercice 5 Écrire une classe `Exo5` avec une fonction `cubeParfait(int x)` qui renvoie `true` so `x` est un cube parfait (ex, $8 = 2^3$, $1000 = 10^3$) ou `false` sinon.

Indication Il y a un exercice très similaire dans le TP précédent, voir les corrections en-ligne pour la fonction `carreParfait(...)`. Vous pouvez faire une boucle `for` pour parcourir les valeurs $i = 1, 2, 3 \ldots,$ `x`. S'il y a un seul i tel que $i^3 = x$, alors `x` est un cube parfait.

Exercice 6 Écrire un programme `Exo6.java` avec une fonction `int compterCubes(int[] t)` qui calcule le nombre de cubes parfaits dans le tableau `t`. Cette fonction `compterCubes` doit faire appel à la fonction `cubeParfait(int x)` écrite à l'exercice précédent. Tester la fonction dans le `main` sur un tableau comme :

```
int[] tab = {8, 1, 27, 9,-1,2,7,-43,1000};
```

Exercice 7 Écrire un programme `Exo7.java` qui demande à l'utilisateur de saisir 5 valeurs ; vous pouvez vous inspirez du code ci-dessous. Pour chaque saisie, il faut indiquer à l'utilisateur s'il a saisit un cube parfait ou pas.

```
double[] vals = new double[5];
Scanner scan = new Scanner(System.in);
System.out.println("Entrer 5 valeurs");
for(int i=0;i<5;i++)
    vals[i] = scan.nextDouble();
```

Exercice 8 Modifier le code précédent pour lire les 5 revenus à partir d'un fichier `vals.txt`. Cette fois ci, on doit appeler la fonction `compterCubes(...)` pour afficher à la fin de la lecture (une seule fois) le nombre de cubes parfaits dans le fichier. Exemple de fichier `vals.txt` :

```
10000
27
200000
70
1000
```

Pour faire la lecture, modifier/utiliser le code ci-après (voir aussi le dernier cours).

```
import java.io.*;
class ....{
    public ... void main(...){
        int[] tab = new int[5];
        BufferedReader objLecteur;
        try{
            objLecteur = new BufferedReader (new FileReader("vals.txt"));
            for(int i=0;i<5;i++){
                String ligne = objLecteur.readLine();
                tab[i] =Integer.parseInt(ligne);
            }
        }catch(Exception e){
            System.out.println("Exception pb lecture:"+e);
        }
    }
}
```

Exercice 9 Finir les exercices du TP précédent. Vous pouvez aussi vous entrainer sur les exercices des annales d'examen disponibles en ligne.

Important : Le niveau de décalage/indentation d'une ligne doit indiquer son niveau d'imbrication

```
class Test{
    public static void main(String[] args){
        //Imbrication 2
        for(int i=0;i<4;i++){
            //Imbrication 3
            for(int j=0;j<4;j++)
                //imbrication 4
                System.out.print("␣"+i*j);
            //Imbrication 3
            System.out.println("");
        }
    //^cette colonne doit rester vide!!!
    }
}
```

Si on monte sur la colonne d'une accolade fermante on trouve la ligne de l'accolade ouvrante!!!

- L'accolade fermante doit rester seule sur sa ligne!!!