TP 14 Programmes Java

Informations techniques PC Suse :

(a) Pour démarrer une session : utilisateur **licencep** et mot de passe **7002n***. Vous trouverez : une `icône lézard`  en haut à droite pour accéder au menu.

(b) Pour démarrer *Processing* : clic sur `l'icône lézard` en haut à droit → Dévelopemment → Processing.

(c) Pour démarrer un *terminal* : `l'icône lézard` → Terminal → Konsole.

(d) Pour ouvrir un gestionaire/navigateur de fichiers : `l'icône lézard` → Système → Dolphin, ou cliquer sur «Dossier Personnel» en haut à gauche.

(e) Pour modifier un fichier, clic droit sur le fichier → Ouvrir avec Kate

---

Exercice 1 Démarrer un terminal et taper le commandes suivantes pour commencer à écrire des fichiers `Java` dans un dossier `tp13`. À la fin de la séance, vous allez pouvoir copier le dossier `tp13` sur une clé USB.

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

Utiliser le code ci-dessous pour écrire un fichier `Exo1.java` qui permet d'afficher la valeur minimale du tableau `tab`.

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

Pour compiler et exécuter, il faut utiliser `javac` et `java`, voir les indications fournies dans les diapos du cours 12 ou au TP7, TP8, TP9 et TP11.

Exercice 2 Taper la commande suivante pour télécharger le fichier `Bazar.java`.
wget `cedric.cnam.fr/~porumbed/Bazar.java`

Essayer de comprendre l'objectif de ce programme. C'est un code bien fonctionnel mais il n'est pas clair du tout. Tel que il est écrit, ce programme est inutile. Il est important d'écrire des programme faciles à comprendre, avec une bonne lisibilité. Corriger les problèmes suivantes :

1. Régler l'indentation avec la commande suivante décrite dans le cours 13 (consulter aussi `man indent`)
   `indent -kr -i4 -nut`
   - `-kr` indentation dans le style Kernighan & Ritchie (légendaires concepteurs du `C`)
   - `-i` chaque niveau d'indentation utilise un *décalage de 4 caractères*
   - `-nut` ne pas utiliser de tabulation
2. La fonction s'appelle `calculer(...)` ce qui ne donne pas d'indice réel sur son objectif. On peut deviner que la fonction calcule un impôt, parce que son argument s'appelle `revenu`. Donner un nom plus pertinent/intuitif à cette fonction.
3. Les variables locales de la fonction `main` ne doivent plus s'appeler x et y mais `revenuFiscalRef` et `impots`, par exemple.

Écrire dans un fichier `Exo2.java` un programme lisible qui permet de calculer les impôts à payer pour un revenu fiscal de référence de 30000. Vérifier si vous trouvez la même valeur qu'à l'exercice 3 du TP précédent. **Attention :** on doit obtenir un code Java qui **respecte les règles indiquées à la dernière page de ce document**.

Exercice 3 Écrire une classe `Exo3` avec une méthode statique `conclureNote(int note)` qui ne renvoie rien et qui affiche :

- `échec` si la note est inférieure à 10 ;
- `succès` si la note appartient à l'intervalle $[10, 19]$
- `vous êtes en génie` pour une note de 20

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

Exercice 4 Écrire une classe `Exo4` avec une fonction `valeurAbsolue(double x)` qui renvoie la valeur absolue de `x`.

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 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 une classe `Exo6` avec une fonction `sommeImpairs(int n)` qui renvoie la somme des premiers n impairs. Par exemple, pour $n = 5$ il faut renvoyer $1+3+5+7+9 = 25$. Une propriété mathématique intéressante : une telle somme est toujours un carré parfait. Écrire dans le `main` le code qui permet de vérifier que la valeur renvoyée par `sommeImpairs(int n)` est bien en carré parfait pour tout n

Exercice 7 Écrire un programme `Exo7.java` avec une fonction `void compter(int[] t)` qui calcule le nombre de valeurs négatives, le nombre de valeurs positives et le nombre de zéros dans le tableau `t`. La fonction doit stocker ces nombres dans les variables globales (attention statiques !) `nbPosit`, `nbNegat` et `nbZeros`. Tester votre fonction avec un tableau comme :

```
int[] tab = {-3, 0, -8,-1,9,-1,2,7,-43,0};
```

Exercice 8 Modifier le programme ci-après pour le faire afficher le taux moyen d'imposition pour un revenu saisi par l'utilisateur. Le taux moyen d'imposition est la part que l'impôt représente par rapport au revenu fiscal de référence. C'est le résultat du rapport entre l'impot et le revenu. Pour un revenu de 20000 euros, il faut payer un impot de (20000-9710)*0.14=1440.6. Le taux moyen d'imposit. est 1440.6/20000=0.72=7.2%.

```
import java.util.*;
class ....{
  static double calculerImpots(....){
    //utiliser le code de l'exercice 2
    //ou l'exercice 3 du TP précédent
  }
  public ... void main (String[]...){
    Scanner scan = new Scanner(System.in);
    System.out.println("Entrer votre revenu fiscal de référence:");
    double revenu = scan.nextDouble();
    //à remplir
  }
}
```

Exercice 9 Écrire une fonction d'en-tête

```
int nbPersonnesImposables(double[] revenus)
```

qui reçoit en entré un tableau (de) `revenus` de plusieurs personnes et qui renvoie le nombre de personnes imposables. Une personne est imposable si son impôt dépasse 0, vous pouvez utiliser une fonction `calculerImpots()` qu'on a déjà vue plusieurs fois (voir exo précédent). Le programme principal devrait demander à l'utilisateur de saisir 5 valeurs de revenu avec un code comme :

```
double[] revenus = new double[5];
Scanner scan = new Scanner(System.in);
System.out.println("Entrer les revenus de 5 personnes:");
for(int i=0;i<5;i++)
    revenus[i] = scan.nextDouble();
```

Exercice 10 Modifier le code précédent pour lire les 5 revenus à partir d'un fichier `revenus.txt`. Exemple de fichier `revenus.txt`

```
10000
19000
200000
70000
100000
```

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

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

Exercice 11 Écrire un programme qui affiche si une chaine de caractères se trouve dans un tableau de `String`. Vous pouvez tester votre programme sur l'exemple ci-après. Remarquer que pour la comparaison (test d'égalité) des chaînes de caractères on utilise la méthode `equals(...)` et non pas l'opérateur ==.

```
String[] tabStr={"abc","toto","titi"};
String valQuOnCherche = "toto";
if(valQuOnCherche.equals(tabStr[0]))
    ....
if(valQuOnCherche.equals(tabStr[1]))
    ....
```

Exercice 12 Écrire une fonction statique d'en-tête

```
int trouveMinMat(int[][] m)
```

qui renvoie la valeur minimale d'une matrice `m`. Tester cette méthode sur une matrice comme par exemple :

```
public static void main(...){
    int[][] mat = { {2, 8, 9},
                    {90,2, 70},
                    {0, 1, 3}};
    ..println(trouveMinMat(...));
}
```

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!!!