TP 13 Programmes Java

Informations techniques PC Suse :

(a) Pour démarrer une session : utilisateur **licencep** et mot de passe **7002n***. Vous trouverez :
— un gestionnaire de fichiers en haut à gauche placé dans le dossier personnel HOME
— une `icône lézard`  en haut à droite pour accéder au menu.

(b) Pour démarrer une *console* : clic sur `l'icône lézard` en haut à droit → Terminal → Konsole.

(c) Pour ouvrir un gestionaire/navigateur de fichiers : clic sur `l'icône lézard` → Utilitaires → Dolphin.

(d) Pour modifier un fichier, clic droit sur le fichier → Ouvrir avec → Kate (ou autre éditeur de votre choix).

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 TP6, TP7, TP8, TP9 et TP10.

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 c'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 fonction statique `conclureNotes(int note)` 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 un programme `Exo4.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 5 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
    //à remplir
  }
}
```

Exercice 6 Écrire une fonction
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 7 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 :

```
import java.io.*;
...
...static double calculerImpots(..) ...
...public static void main(...) ....
....

double[] tab = new double[5];
....

BufferedReader lecteur;//objLecteur fic
try{
   lecteur = new BufferedReader
             (new FileReader("in.txt"));
   while(true){
     String ligne = lecteur.readLine();
     if(ligne==null)
          break;
     tab[i] =Double.parseDouble(ligne);
     i++;
   }
}catch(Exception e){
    System.out.println("Exception:"+e);
}
```

Exercice 8 Écrire une classe `Exo8` avec une fonction `double noteFinale(double tp, double exam)` qui permet de calculer la note finale 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` $\geq$ 10 et `exam`$\geq$`tp`. **Indication** : il n'est plus nécessaire de tester `exam>=10`. Il suffit d'appeler `return 0` au point (1) si `note<10`. Si la note est inférieure à 10, on n'arrive pas au point (2).

(3) La note finale est la moyenne de la note d'examen et de TP si on a les relations 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.

Exercice 9 Continuer le programme pour afficher le nombre de personnes qui ont validé (note finale $\geq$ 10). Les notes d'examen et de TP sont stockées dans deux tableaux `notesexam` et `notestp` de longueur 5. Il faut récupérer ces notes à partir des fichiers `notesexam.txt` et `notestp.txt`. Faire une lecture de fichier comme à l'exercice précédent. On doit utiliser deux objets `BufferredReader`, un pour le fichier `notesexam.txt` et un autre pour le fichier `notestp.txt`.

Exercice 10 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 11 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(...));
}
```

Exercice 12 Écrire un programme qui transpose et affiche une matrice saisie en dur dans le code comme à l'exercice précédent. Sur cet exemple, le programme devrait afficher.

```
2  90  0
8  2   1
9  70  3
```

Exercice 13 Écrire une fonction qui affiche un sapin de Noël avec une taille n (nombre de couches au-dessus du tronc) saisie par l'utilisateur. Voici un exemple pour $n = 5$:

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

Exercice 14 Écrire un programme qui permet de lire un fichier texte et d'afficher le texte codé avec le chiffre de César (par décalage) présenté au Cours 3 sur les réseaux. La clé de décalage est saisie par l'utilisateur. Par exemple, si la clé vaut 3 alors on a les transformations $a \to d$, $b \to e$, $c \to f, \ldots, x \to a$, $y \to b$, $z \to c$. Les espaces et les virgules ne sont pas modifiés.

Vous allez avoir besoin de récupérer chaque caractère individuel d'une ligne. Si vous utilisez un appel comme `ligne=lecteur.readLine()`, alors `ligne.charAt(0)` renvoie un `char` qui correspond à la première lettre. Une variable de type `char` est stockée sur un octet qui représente le code ASCII de la lettre. Par exemple, la lettre `A` a le code 65, `B` a le code 66, etc (taper "table ASCII" sur internet pour visualiser tous les codes). Dans une première étape on peut travailler que avec des majuscules. Pour décaler une lettre de 3, il suffit d'écrire par exemple `lettreDecale=(char)(lettre+3)`, on ignore les accents, les virgules, les apostrophes, les points.

Exercice 15 Une méthode pour casser le chiffre de César est la suivante. Trouver le caractère qui apparaît le plus souvent dans le code et on a de bonnes chances de trouver le code de "e" (la lettre la plus utilisée en français), voir le Cours 3 sur les réseaux. Écrire un programme capable de trouver le message derrière le fichier suivant : wget `cedric.cnam.fr/~porumbed/cesar.txt`

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!!!