

**Exercice 1.** Étudier le programme suivant :

```
public class Somme {  
  
    private static void usage(){  
        System.err.println("usage : Somme nombre [nombre...]");  
        System.exit(1);  
    }  
  
    private static int somme(String[] table){  
        int s=0;  
  
        for(int i=0 ; i < table.length ; i++){  
            s += Integer.parseInt(table[i]);  
        }  
        return s;  
    }  
  
    public static void main (String[] args){  
        if (args.length == 0)  
            usage();  
        else {  
            System.out.println("La somme des arguments entiers est "+  
                               somme(args));  
        }  
    }  
}
```

- Le programme `Somme` est-il correct d'un point de vue modulaire ?
- Comment réorganiser ce programme afin de disposer d'un module `TableauEntiers` qui comporte les fonctions suivantes :
  - une fonction `somme` qui retourne la somme des éléments d'un tableau d'entiers.
  - une fonction `minimum` qui retourne le minimum d'un tableau d'entiers.
  - une fonction `indiceMaxi` qui retourne l'indice du maximum d'un tableau d'entiers.
  - une fonction `opposes` qui retourne un nouveau tableau contenant les opposés des éléments du tableau d'entiers passé en paramètre.
  - une fonction `afficheTableau` qui affiche le tableau d'entiers passé en paramètre.
- Écrire un programme qui fait appel aux fonctions du module `TableauEntiers` pour calculer la somme, le minimum, l'indice du maximum et un tableau contenant les opposés des éléments d'un tableau d'entiers constitué des arguments du programme; un message d'erreur sera affiché dans le cas où ceux-ci seraient absents.

#### ARGUMENTS D'UN PROGRAMME

**Exercice 2.** Écrire un programme `Combien` qui affiche le nombre d'arguments sur la ligne de commande.

Exemple :

```
> java Combien a xx 546  
Le programme a 3 arguments
```

**Exercice 3.** Écrire un programme `HelloPerso` qui affiche "Hello <nom> !" où <nom> est passé en argument de la commande.

**Exercice 4.** Ecrire un programme **Phrase** qui affiche la liste des arguments terminée par un point.

**Exercice 5.** Ecrire un programme **Hello** qui affiche "Hello <nom>!" pour chaque <nom> passé en argument de la commande et affiche "Hello World!" pour terminer Exemple :

```
> java Hello Pierre Paul Julie
Hello Pierre !
Hello Paul !
Hello Julie !
Hello World !
```

## EDITION, COMPILATION, EXÉCUTION

### Exercice 6.

1. À votre racine créer le répertoire **programmation2**  
Ce répertoire contiendra l'ensemble de votre travail en programmation 2.
2. Se placer dans le répertoire **programmation2**  
Y créer le sous-répertoire **src**. Se placer dans le répertoire **src**.
3. Télécharger les fichiers qui accompagnent le td à l'aide de la commande  
`wget http://dept-info.labri.fr/ENSEIGNEMENT/programmation2/src/fichiers-1.tar.gz`
4. Décompacter cette archive au moyen de la commande `tar -xvzf fichiers-1.tar.gz`  
Le répertoire **td01** est créé automatiquement.
5. Éditer avec **emacs**<sup>1</sup> le fichier `programmation2/src/td01/Somme.java`
6. Compiler sous **emacs** le programme `Somme.java`; Pour cela taper  
`M-x compile`<sup>2</sup>  
puis remplacer `make -k` par  
`javac Somme.java`
7. Dans un shell, vérifier le contenu du répertoire **td01** et exécuter le programme à l'aide de la commande `java Somme`.

### Exercice 7.

1. Éditer avec **emacs** le fichier `Monnaie.java`
2. Corriger et indenter correctement ce code.
3. Lancer la compilation du programme depuis **emacs**;  
Traiter les erreurs de syntaxe en utilisant la clé `C-x` <sup>3</sup>
4. Exécuter ce programme et vérifier s'il donne tous les résultats attendus. Corriger éventuellement le code source.
5. Modifier le programme précédent afin qu'une exécution sans paramètre génère l'affichage  
`Usage: java Monnaie <somme>`

---

<sup>1</sup>il est préférable de ne pas lancer plusieurs processus emacs en parallèle

<sup>2</sup>Pour obtenir M-x (méta x), il faut appuyer successivement sur les touches "Esc" puis x.

<sup>3</sup>pour obtenir le symbole `‘` sur un clavier AZERTY, il faut taper "AltGr"-7.Cette clé permet de se positionner automatiquement dans le fichier contenant la première erreur et sur la ligne de l'erreur. Chaque nouvelle frappe de la clé `C-x` `‘` provoque le positionnement sur l'erreur suivante, et ceci jusqu'à ce que toutes les erreurs aient été balayées.

## STRUCTURES DE CONTRÔLE

**Exercice 8.** Écrire un programme **PgcdPpcm** qui calcule puis affiche le pgcd et le ppcm de deux entiers en utilisant l'algorithme d'Euclide.

Rappels :

algorithme d'Euclide : a et b sont deux entiers. Tant que le reste de la division de a par b est non nul, a prend la valeur de b, b prend la valeur du reste. Le pgcd est le dernier reste non nul.

le produit de deux entiers est égal au produit de leur pgcd par leur ppcm.

**Exercice 9.** Écrire un programme **PremiersEntre** comportant les fonctions suivantes

1. Une fonction **premier** qui retourne *true* si le nombre entier passé en paramètre est premier et *false* sinon.
2. Une fonction **premiersEntre** qui affiche tous les nombres premiers compris, au sens large, entre les 2 valeurs entières min et max passées en paramètres.
3. Une fonction **main** telle que le programme ait le comportement suivant :
  - dans le cas où il y a deux arguments, affiche tous les nombres premiers compris, au sens large, entre ceux-ci.
  - affiche le message d'erreur "**Usage : PremiersEntre <nombre1> <nombre2>**" dans tous les autres cas.

## TABLEAUX

**Exercice 10.** Soit une suite de  $n$  valeurs observées  $x_0, x_1, \dots, x_{n-1}$ ,  $x_i, i = 0 \dots n-1$ , étant des entiers.

Écrire un module **Stats** qui comporte les fonctions suivantes :

– une fonction **moyenne** qui retourne la moyenne de la suite d'entiers passée en paramètre.

$$m = \frac{1}{n} \sum_{i=0}^{n-1} x_i$$

– une fonction **ecartype** et une fonction **variance** qui retourne l'écart type **std** et la variance **var** de la suite d'entiers passée en paramètre.

$$var = \frac{1}{n} \sum_{i=0}^{n-1} (x_i - m)^2$$

$$std = \sqrt{\frac{1}{n} \sum_{i=0}^{n-1} (x_i - m)^2}$$

– une fonction **mediane** qui retourne une valeur  $M$  partageant la suite  $x_{i=0}^{n-1}$  en deux sous-ensembles ayant le même nombre d'éléments.

Par exemple pour  $int[] tabEnt = \{5, 3, 2, 4, 1\}$ ,  $M = 3$

et pour  $int[] tabEnt = \{1, 7, 5, 2\}$ ,  $M$  pourrait être 3 ou 4.

Les éléments  $\{x_i\}_{i=0}^{n-1}$  de la suite seront passés en paramètre.

Écrire un programme pour l'utilisation de ces fonctions sur une suite d'entiers constitué des arguments du programme ; un message d'erreur sera affiché dans le cas où ceux-ci seraient absents.