#1.fr - Premiers pas en Java, chaînes de caractères, tableaux, boucles

Exercice 1 - Hello Groland

On rappelle qu'en Java chaque classe publique doit être définie dans un fichier qui lui est propre. Le nom du fichier doit être le nom de la classe qu'il contient, auquel on ajoute le suffixe **.java**. Les noms des classes doivent être constitués de mots accolés dont la première lettre est une majuscule (style *CamelCase*).
Dans un premier temps nous allons écrire des petits programmes permettant de se familiariser avec le compilateur, la machine virtuelle et les méthodes.

1. Écrire le programme suivant :

```
public class HelloGroland {
  public static void main(String[] args) {
    System.out.println("Hello Groland");
  }
}
```

   dans votre éditeur de texte préféré et sauvegarder celui-ci sous le nom `HelloGroland.java`
2. Compiler le programme en utilisant le commande `javac` puis vérifier que le fichier `.class` correspondant existe bien.

```
javac HelloGroland.java
```

3. Exécuter le programme avec la commande `java`

```
java HelloGroland
```

   On ne met pas ".class" parce que la machine virtuelle le rajoute toute seule.

Exercice 2 - Afficher les arguments de la ligne de commande

Écrire une classe `PrintArgs` qui affiche les arguments de la ligne de commande.

```
$ java PrintArgs Voici des arguments
Voici
des
arguments
```

Les arguments de la ligne de commande sont stockés dans le tableau de chaînes de caractères passé en argument à la méthode `public static void main(String[] args)`.

- Dans un premier temps, afficher le premier argument de la ligne de commande (dans notre exemple `Voici`).
  Que se passe-t'il si l'on ne passe pas d'argument lors de l'exécution du programme ?
- Écrire une boucle affichant le contenu du tableau en sachant qu'en Java les tableaux possèdent un champ (un attribut) `length` qui renvoie la taille du tableau.
- Changer votre programme pour utiliser la syntaxe dite 'foreach' `for(Type value: array)`

Exercice 3 - Calculette simple

Écrire un programme prenant un nombre sur l'entrée standard et affichant celui-ci
Pour cela, on utilisera un objet `Scanner` et particulièrement sa méthode `nextInt()`.

Pour comprendre le programme, il est utile de regarder la documentation **disponible**, et même de mettre les liens en bookmark (signet, favoris, etc.)

1. Recopier le programme précédent et le compléter pour qu'il affiche le nombre saisi par l'utilisateur.
2. Indiquer dans le programme où sont les variables et quel est leur type associé.
   Modifier le programme pour déclarer et initialiser les variables en une seule ligne.
3. Pourquoi `nextInt()` n'est pas une fonction ?
   Qu'est `nextInt()` alors ?
4. Expliquer la ligne :

```
import java.util.Scanner;
```

5. Modifier le programme pour qu'il demande deux entiers et affiche la somme de ceux-ci.
6. Afficher en plus de la somme, la différence, le produit, le quotient et le reste.

Exercice 4 - Conversion de String en entier

On souhaite écrire un programme affichant la somme d'entiers pris en paramètres sur la ligne de commande.
Voici un exemple d'exécution :

```
$ java Sum 15 5 231
integers: 15 5 231
sum: 251
```

Ce programme est décomposé en plusieurs fonctions :

1. Écrire une méthode qui prend un tableau de chaînes de caractères en argument et renvoie un tableau d'entiers de même taille contenant les entiers issus des chaînes de caractères.
   La méthode statique `parseInt(String s)` de la classe `java.lang.Integer` permet de récupérer la valeur d'un entier stockée dans une chaîne de caractères.
2. Que veut dire statique pour une méthode ?
3. Que se passe-t'il lorsqu'un mot pris en argument n'est pas un nombre ?
4. Écrire une méthode qui prend un tableau d'entiers en argument et renvoie la somme de ceux-ci.
5. Écrire la méthode `main` qui utilise les deux méthodes précédentes pour afficher le tableau d'entiers ainsi que sa somme.
   Il y a un petit piège pour afficher les tableaux vous pouvez vous aider de la classe `java.util.Arrays`.

Exercice 5 - De C vers Java

Cet exemple a pour but de montrer les différences de performance entre un programme en C et le même en Java.

1. Compiler (`gcc pascal.c`) et exécuter le programme `a.out` en demandant au système le temps d'exécution du programme. (`time a.out`).
2. Écrire le programme (`Pascal.java`) équivalent en Java. Pour une fois, servez-vous du copier/coller. Compiler le programme puis l'exécuter en mesurant le temps (toujours avec `time`).

Comment peut-on expliquer la différence de vitesse ?
