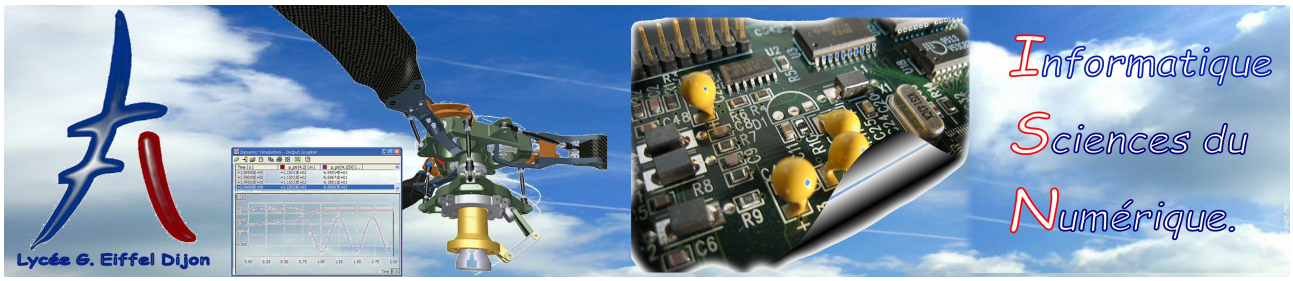

Les fonctions en Java : cours

Objectifs :
- ➢ **Savoir définir et utiliser une fonction**
- ➢ **Maîtriser la portée des variables**
- ➢ **Connaître le passage par valeurs en Java**

1 Fonctions : définition, intérêt

1.1 Exemple

On a un programme qui présente des répétitions :

```
System.out.println("Vous êtes monsieur Martin.");
System.out.println("Vous habitez à Dijon.");
System.out.println("Votre compte en banque s'élève à 1 552 000 euros.");
System.out.println("————————————————————————");
System.out.println("Vous êtes monsieur Durand.");
System.out.println("Vous habitez à Marsannay.");
System.out.println("Votre compte en banque s'élève à 3204.25 euros.");
System.out.println("————————————————————————");
System.out.println("Vous êtes monsieur Brunet.");
System.out.println("Vous habitez à Chenôve.");
System.out.println("Votre compte en banque s'élève à 24788,32 euros.");
System.out.println("————————————————————————");
```

Inconvénients de ces répétitions :
- C'est long à écrire.
- C'est difficile à lire.
- Si on veut modifier une phrase, il faut la modifier à 3 endroits dans le code source.

On utilise donc une fonction à la place.

1.2 Définition, passage d'arguments

Une fonction est une instruction isolée du reste du programme, qui possède un nom, et qui peut être appelée par ce nom à n'importe quel endroit du programme et autant de fois que l'on veut.
Les trois blocs de texte précédents ne sont pas totalement identiques : le nom, le lieu et le montant du compte diffèrent : il faut paramétrer l'instruction qu'on isole pour pouvoir lui passer le nom, le lieu et le montant : on passe donc des arguments à la fonction.
Exemple : proposer une fonction et un programme principal qui permettent d'afficher les données de Martin, Durand et Brunet.
Avantages des fonctions :
- Pour utiliser une fonction dans un programme principal, il est inutile de savoir comment la fonction est programmée, connaître son utilisation est suffisant.

– Les fonctions permettent d'organiser le travail de développement : un programmeur peut mettre au point une fonction, pendant qu'un autre crée le programme principal.
– En cas de changement, s'il y a x appels de la fonction dans le programme principal, il suffit de modifier une fois la fonction au lieu de modifier x fois le programme principal.

1.3 Retour d'une valeur

Les arguments permettent de passer des informations du programme principal vers une fonction. Inversement, une fonction peut passer une valeur qu'elle a calculée au programme principal. On utilise alors l'instruction *return*.
Par exemple, la fonction suivante prend en argument un entier n et renvoie au programme principal la somme des cubes des n premiers entiers non nuls :

```
int sommeCubes(int n) {
int s=0;
for (int k=1;k<=n;k++) s=s+Math.pow(k,3);
return s;
}
```

Remarques :
– l'instruction *return* interrompt l'exécution de la fonction.
– La première ligne de la fonction ci-dessus s'appelle l'en-tête de la fonction.
– Si la fonction ne renvoie pas de valeur, on écrit le mot clé *void* devant.
– L'instruction écrite après l'en-tête s'appelle le corps de la fonction.

1.4 Conclusion

Pour écrire une fonction :

1. On détermine l'en-tête :
   – Type de la valeur renvoyée (*void* s'il n'y a pas de valeur renvoyée).
   – Nom (explicite) de la fonction (par convention, il commence par une minuscule puis les différents mots commencent par une majuscule).
   – Liste des arguments entre parenthèses (avec un nom explicite), avec le type devant chaque argument.
2. On écrit le corps de la fonction, avec l'instruction *return* si nécessaire.
3. On s'assure par des tests que la fonction fonctionne quels que soient les valeurs passées en arguments.

Remarques :
– Lors de l'appel d'une fonction depuis le programme principal, l'ordre des arguments est important.
– En Java, le programme principal est lui-même une fonction qui s'appelle *main*.

2 Portée des variables

2.1 Variables globales, variables locales

Une variable globale est une variable déclarée en début de programme, en dehors de toute fonction, y compris en dehors de la fonction *main*. Elle est connue et utilisable dans tout le programme. On dit que sa portée est le programme entier.
Une variable locale est déclarée à l'intérieur d'une fonction et sa portée est réduite à cette fonction : elle n'est pas utilisable à l'extérieur de la fonction.
Remarque :
– Si une variable fait partie des arguments d'une fonction, sa portée est limitée à cette fonction.
– Si deux variables de même nom sont déclarées dans deux fonctions différentes, la portée de chacune est limitée à la fonction correspondante.

– Si une variable locale (définie dans une fonction) et une variable globale ont le même nom (surtout à éviter), alors la variable globale est inaccessible dans la fonction, et devient accessible à l'extérieur de la fonction.

Exemple : On considère le programme suivant :

```
int a;
void f (int x) {
System.out.println(2*x);
a=2*x;
}
void main () {
int n;
a=3;
n=4;
f(a+n);
f(a+n);
}
```

Qu'affiche-t-il ?

De manière générale, une variable utilisée dans plusieurs fonctions doit être globale. Une variable utilisée dans une seule fonction est de préférence locale.

2.2 Le passage par valeur

Quand on appelle une fonction depuis le programme principal en lui passant des paramètres, les valeurs de ceux-ci sont recopiées ailleurs dans la mémoire : dans la fonction, on travaille alors à ces nouvelles adresses. On ne peut donc pas, dans le corps de la fonction, faire une modification de ces paramètres qui se répercuterait dans le programme principal. On dit que le passage des arguments se fait par valeur.

Exemple : on considère le programme suivant :

```
void main(){
int a=3;
modifie(a);
print(a);
}

void modifie(int x) {
x=2;
}
```

Que s'est-il passé ?

Si on veut néanmoins modifier dans le corps d'une fonction une valeur qui se répercute dans le programme principal, alors il faut utiliser une variable globale (ce n'est pas possible grâce aux paramètres).

2.3 Le passage par valeur et les tableaux

On considère le programme suivant :

```
void main(){
int[] tab =new int[1];
tab[0]=3;
modifie(tab);
print(tab[0]);
}
void modifie(int[] tab2) {
tab2[0]=2;
}
```

Que s'est-il passé ?