**La notion de fonction et de méthode**

On appelle **fonction** un sous-programme qui permet d'effectuer un ensemble d'instructions par simple appel de la fonction dans le **corps** du programme principal. Les fonctions permettent d'exécuter dans plusieurs parties du programme une série d'instructions, cela permet une simplicité du code et donc une taille de programme minimale. D'autre part, une fonction peut faire appel à elle-même, on parle alors de **fonction récursive** (il ne faut pas oublier de mettre une condition de sortie au risque sinon de ne pas pouvoir arrêter le programme...).

Une **méthode** est une fonction faisant partie d'une **classe**. Elle permet d'effectuer des traitements sur (ou avec) les données membres des **objets**. Puisqu'en Java on ne manipule que des classes, il n'y aura formellement que des méthodes, même si on pourra parler de fonction pour des méthodes static qui ne manipule aucune des données membres.

**La déclaration d'une méthode**

Avant d'être utilisée, une méthode doit être définie car pour l'appeler dans une classe il faut que le compilateur la connaisse, c'est-à-dire qu'il connaisse son nom, ses arguments et les instructions qu'elle contient. La définition d'une méthode s'appelle "***déclaration****"*. La déclaration d'une fonction se fait selon la syntaxe suivante :

TypeDeRetour nomDeLaMethode(Type1 argument1, Type2 argument2) {  
 liste d'instructions  
}

**Remarques:**

* TypeDeRetour représente le type de valeur que la méthode va retourner, cela peut-être un type primitif, une classe, ou alors le mot-clé *void* si la méthode ne retourne aucune valeur.
* le nom de la méthode suit les mêmes règles que les noms de variables :
  + le nom doit être écrit en minuscule, sauf pour les initiales de chaque nouveau mot
  + un nom de méthode peut comporter des chiffres, mais pas pour le premier caractère
  + les caractères spéciaux \_ et $ peuvent être utilisés mais ne devraient pas l'être
  + le nom de la méthode est sensible à la casse (différence entre les minuscules et majuscules)
* Les arguments sont facultatifs, mais s'il n'y a pas d'arguments, les parenthèses doivent rester présentes
* Il ne faut pas oublier de refermer les parenthèses et les accolades
* *Le nombre d'accolades ouvertes (méthode, boucles et autres structures) doit être égal au nombre d'accolades fermées !*
* *La même chose s'applique pour les parenthèses, les crochets ou les guillemets !*

Une fois cette étape franchie, votre méthode ne s'exécutera pas tant que l'on ne fait pas appel à elle quelque part dans la classe!

## Appel de méthode

Pour exécuter une méthode, il suffit de faire appel à elle en écrivant l'objet auquel elle s'applique (celui qui contient les données), le nom de la méthode ( en respectant la casse), suivie de ses arguments entre parenthèse :

objet.nomDeLaMethode(argument1,argument2);

**Remarques :**

* si vous exécutez une méthode sur l'objet courant (*this*), c'est à dire que vous utilisez dans une classe, une méthode de la même classe, alors il est inutile de préciser que *this* est l'objet auquel s'applique la méthode.
* le point virgule signifie la fin d'une instruction pour distinguer les différents blocs d'instructions
* si jamais vous avez défini des arguments dans la déclaration de la méthode, il faudra veiller à les inclure lors de l'appel de la méthode (le même nombre d'arguments séparés par des virgules !), mais sans avoir à préciser leur type.

## Les arguments d'une méthode

Il est possible de passer des arguments (appelés aussi *paramètres*) à une méthode, c'est-à-dire lui fournir une valeur ou le nom d'une variable afin que la méthode puisse effectuer des opérations sur ces arguments ou bien grâce à ces arguments.  
Le passage d'arguments à une méthode se fait au moyen d'une liste d'arguments (séparés par des virgules) entre parenthèses suivant immédiatement le nom de la méthode.

Le nombre et le type d'arguments dans la déclaration, le prototype et dans l'appel doit correspondre, au risque, sinon, de générer une erreur lors de la compilation...

Un argument peut être :

* une constante
* une variable
* une expression
* une autre méthode retournant une valeur

## Renvoi d'une valeur par une méthode

La méthode peut renvoyer une valeur (et donc se terminer) grâce au mot-clé *return*. Lorsque l'instruction *return* est rencontrée, la méthode évalue la valeur qui la suit, puis la renvoie au programme appelant (classe à partir de laquelle la méthode a été appelée).

Une méthode peut contenir plusieurs instructions *return*, ce sera toutefois la première instruction *return* rencontrée qui provoquera la fin de l'exécution de la méthode et le renvoi de la valeur qui la suit.

La syntaxe de l'instruction return est simple :

return valeurDeRetour;

*Le type de valeur retourné doit correspondre à celui qui a été précisé dans la définition*

## La surcharge de méthode

Un des apports les plus intéressants de la conception objet, est la possibilité d'appeler plusieurs méthodes avec le même nom, à condition que leurs arguments diffèrent (en type et/ou en nombre). Ce principe est appelé surcharge de méthode. Il permet de donner le même nom à des méthodes comportant des paramètres différents et simplifie donc l'écriture de méthodes sémantiquement similaires sur des paramètres de types différents.

En effet, une méthode est déterminé par ce que l'on appelle sa *signature*, c'est-à-dire :

* sa classe d'appartenance
* son nom
* ses paramètres

Il est ainsi possible de définir une méthode réalisant la même opération sur des variables différentes en nombre ou en type :

int somme(int p1, int p2) {  
 return p1+p2;  
}  
int somme(int p1, int p2, int p3) {  
 return p1+p2+p3;  
}  
double somme(double p1,double p2){  
 return p1+p2;  
}