Méthodes : projet *ReHello*

Définition

Nous avons créé des objets. Nous avons assigné des propriétés à ces objets. Il ne nous reste plus qu'à leur associer un comportement, car s'il se contentaient de rester plantés sur place on tomberait vite dans le très descriptif HTML ;-)
On peut aussi appeler les méthodes des *fonctions* ou des *routines* (ce terme est peu usité en Java, il vient du C++).

Note concernant ce chapitre : plusieurs paragraphes de ce chapitre concernant des fonctions plus ou moins évoluées ne sont pas développés ; je trouve que c'est préférable ainsi, et qu'après une brève définition pour vous informer sur l'existence de la chose, il vaut mieux ne pas vous noyer sous des exemples qui devraient faire appel à de concepts qui ne vous sont pas encore familiers et dont vous n'aurez pas d'utilité immédiate.

Généralités

La création de la méthode elle-même n'est pas compliquée, ce qui est plus délicat est de lui assigner un comportement. Les méthodes sont reconnaissables à leurs parenthèses, elles sont délimitées comme les classes par des accolades. Vous êtes libre du choix de leur nom (choisissez quand même un truc intelligent…).

Si une méthode ne retourne pas de valeur , (les System.out.print() ne sont pas considérés comme retournant des valeurs), il faut la déclarer à l'aide du mot clé *void*. Si elle retourne une valeur on la déclare avec le mot clé du type de valeur qu'elle retourne. Pour qu'une méthode retourne une valeur, on emploie return. On peut assigner à une méthode des arguments (sous la forme de variables classiques, par opposition aux variables de classe), on place ces éléments entre les parenthèses. Ainsi dans une classe, 2 méthodes peuvent avoir la même nom, seuls leurs arguments respectifs les différencieront aux yeux du compilateur. On appelle ce concept surcharge de méthodes.

```
// ne retourne pas de valeur
void imprime() {
    int i = 10 + 20 ;
    System.out.println(" Résultat : "  + i) ;
}

// retourne une valeur
int imprime() {
    int i = 10 + 20 ;
    return i ;
}

// admet un argument
int imprime(int i);
i = 10 + i ;
return i ;
}
```

On notera que l'habituel *public static void main(String args[]) {…}* déclare en fait une méthode publique (voir chapitre sur les *modificateurs*), statique (voir aussi ce très utile chapitre), qui ne retourne pas de valeur, qui se nomme *main* et qui a pour argument une matrice de chaînes nommée *args*. Ouf…

Constructeurs

Les *constructeurs* sont des méthodes particulières qui servent à créer des objets à partir s'une classe. Elles portent exactement (toujours la casse…) le même nom que la classes qui les contient. Elles n'ont pas de type de retour.

Le mot clé *this*

Ce mot clé est utile pour la lisibilité du code. Il sert dans le cas où une variable d'instance et une variable de classe portent le même nom. *this* est alors employé pour faire référence à la variable de classe. Vous pouvez ainsi attribuer plusieurs fois le même nom de variable.

- this.nomDeLaVariable = nomDeLaVariable ;

Le mot clé *super*

Si une méthode *beta* est imbriquée dans une méthode *alpha*, pour rajouter une instruction à *alpha* dans le corps de *beta*, on utilise le mot clé *super*.

Finaliser

On finalise quand on appelle le *garbage collector* sur un objet dont on n'a plus besoin. Il est alors retiré de la mémoire vive. C'est une des seules opérations que vous devez faire sous Java pour gérer la mémoire.

- Protected void finalize() throws Throwable {
- Super.finalize();
- }

Application: ReHello

Le programme ReHello est la seconde mouture de HelloWorld, il crée un méthode qui sert à afficher du texte. Ce très court programme est plus simple que plusieurs de chapitres précédents et il n'utilise pas de concept évolué. C'est donc un jeu d'enfant…

```
class ReHello { // déclaration de la classe principale

  public static void imprime() { // création de la méthode qui affiche le texte
    // ci-dessous les instructions de la méthode
    System.out.println("HelloWorld ! Encore une fois...");
  } // fin de la méthode imprime()

  public static void main(String args[]) { // la méthode main()...
    imprime(); // ... qui fait appel à la méthode imprime()
  }
}
```

On note que notre méthode imprime() est statique (déclarée ainsi par le mot clé static), car on ne peut faire référence dans une première méthode déclarée statique à une seconde méthode que si cette dernière est statique. C'est le cas de la méthode main() qui est statique est qui ne peut contenir que des références à des méthodes statiques. (voir le chapitre sur les modificateurs pour plus amples informations)