JAVA07 - DÉFINIR UNE CLASSE

CONCEPT DE CLASSE

Une classe définit la structure d'un objet, elle permet de déclarer l'ensemble des propriétés qui composent les futurs objets. Une classe définit :

- Les attributs : variables communes aux futurs objets de la classe.
- Les méthodes : opérations applicables aux futurs objets

Convention : Les noms de classe commence par une majuscule alors que les noms des variables faisant référence à des objets commencent par une minuscule pour le premier mot et une majuscule pour les mots suivants. Sauf pour les constantes qui sont déclarées ainsi :

ÉCRITURE D'UNE PREMIÈRE CLASSE

Une classe est en quelque sorte une structure complexe qui permet **l'encapsulation de données**.
Une classe est composée de données et de méthodes. Lorsque l'encapsulation des données est parfaite, seules les (certaines) méthodes sont accessibles. Ceci évite en principe à l'utilisateur de la classe, de se soucier de son fonctionnement et de faire des erreurs en changeant directement la valeur de certaines données.
Prenons un exemple concret et simple : l'écriture de la classe **Nombre** vue au chapitre précédent. Cet exemple va nous suivre tout au long de ce chapitre.

DÉCLARER UNE CLASSE

Une classe peut être qualifiée par les mots clefs suivants: abstract, final, private ou public.
Pour le moment, nous ne retiendrons que le terme public.

```
public class Nombre
{
 // zone de déclaration des attributs   --> partie privée
 //zone de déclarations des constructeurs -->partie publique
 // zone de déclaration et d'implémentation des méthodes --> partie publique
}
```

DÉFINIR DES ATTRIBUTS

Pour déclarer un attribut, il faut indiquer :

- S'il est public ou privé à l'aide des mots clefs : public ou private.
- Son type.
- Son nom commençant par une minuscule.

```
public class Nombre
{
 private int nb ;
  .....
}
```

Remarque : Les attributs d'une classe sont généralement privés . Ainsi on ne pourra modifier les valeurs uniquement si une méthode le prévoit (voir plus loin le rôle des « setteurs »).

DÉFINIR UN OU PLUSIEURS CONSTRUCTEURS

DÉFINITION

Le constructeur est une méthode : il porte le même nom que la classe. Il sert à construire des objets, son but est donc d'initialiser les variables contenues au sein de l'objet.

IMPLÉMENTATION DU CONSTRUCTEUR PAR DÉFAUT

```
public Nombre( )
{
 this.nb = 0 ;
 }
```

Remarque : Le mot clef this n'est pas obligatoire mais conseillé, il permet de faire référence aux attributs. This correspond à l'objet courant, c'est à dire l'objet en cours d'exécution.

IMPLÉMENTATION D'UN CONSTRUCTEUR AVEC PARAMETRES

```
 public Nombre( int nb)
    {
     this.nb=nb ;
     }
```

Remarques :

- On peut utiliser les mêmes noms pour les attributs et les paramètres, à condition d'employer this devant les attributs, il est possible de nommer les paramètres tout autrement.
- On peut déclarer autant de constructeurs que l'on désire à condition que leur signature diffère, c'est à dire que l'ordre des paramètres soit différents.

DÉFINIR DES ACCESSEURS

DÉFINITION

Les accesseurs sont des méthodes qui permettent d'accéder aux attributs privés.

IMPLÉMENTATION DES GETTERS

Les "GETTER" servent à récupérer les valeurs contenues dans un objet.Un accesseur "getter" retourne la valeur contenue dans un attribut de l'objet. Un getter :

- porte par convention le même nom que l'attribut précédé du mot get.
- est une méthode publique : utilisable ailleurs qu'au sein de la classe.
- a pour type de retour celui défini pour l'attribut en question.

```
public double getInt ( ) {
    return this.nb ;
 }
```

IMPLÉMENTATION DES SETTERS

Les "SETTER" servent à mettre à jour les valeurs contenues dans un objet. Un accesseur "setter" initialise un attribut à partir d'une donnée passée en paramètre. Un setter :

- porte par convention le même nom que l'attribut précédé du mot set.
- est une méthode publique : utilisable ailleurs qu'au sein de la classe.
- ne retourne rien (void)
- attend un paramètre de type identique à celui de l'attribut

```
public void setInt ( int nb ){
   this.nb = nb ;
}
```

DÉFINIR DES MÉTHODES

IMPLÉMENTATION DE LA MÉTHODE TOSTRING

Elle retourne les valeurs contenues dans l'objet sous forme textuelle, elle est utilisée pour l'affichage d'un objet.

```
public String toString(){
        String description = " la valeur de l'objet est de : " + this.getNb();
        return description;
}
```

IMPLÉMENTATION DES AUTRES MÉTHODES

Dans une méthode, on peut donc se servir :

- De variables ou objets :
  - paramètres (variables locales)
  - variables locales à la méthode
  - attributs de la classe (variables globales)
- Des instruction de base (if, while,...)
- Du mot clef return pour retourner des valeurs
- D'appel à d'autres méthodes définies dans la classe.

Exemples :

```
public int sommeChiffres ( ) {
        int somme = 0;
        String nombre = String.valueOf(this.nb);
        for (int i = 0 ; i < nombre.length() ; i++){
                somme +=  Character.getNumericValue(nbString.charAt(i));
        }
        return somme;
}
```

LA DÉCLARATION ET L'UTILISATION 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 méthode se fait selon la syntaxe suivante :

```
public type_de_donnee nomDeLaMethode(type1 argument1, type2 argument2, ...) {
        liste d'instructions
}
```

Remarques:

- type_de_donnee représente le type de valeur que la méthode doit retourner (char, int, float,double,String,boolean...)
- Si la méthode ne renvoie aucune valeur, on remplace le type de donnée par le mot clé ***void.***
- Une méthode doit obligatoirement porter un type de retour (sauf dans le cas des constructeurs)

Pour exécuter une méthode, il suffit de faire appel à elle en écrivant son nom (une fois de plus en respectant la casse) suivie d'une parenthèse ouverte (éventuellement des arguments) puis d'une parenthèse fermée :

IMPORTANCE DE LA DOCUMENTATION

L'utilisateur d'un objet ne voit que les **caractéristiques publiques** (*public*) de l'objet. On lui fournit la version compilée « NomObjet.class » et une **documentation des interfaces (API)**. Sans cette API vous ne pouvez pas utiliser les objets, c'est la raison pour laquelle, JAVA met à votre disposition l'outil **javadoc** qui permet de générer une telle documentation au format HTML.

UTILISATION DES COMMENTAIRES JAVADOC

Les commentaires peuvent servir au programmeur sans que ce dernier souhaite forcément qu'ils apparaissent au niveau de l'API. Pour régler ce problème, les commentaires javadoc se différencient des commentaires traditionnels. Le tableau suivant montre les types de commentaires supportés : vous conviendrez que pour ce qui est du compilateur Java, il ne voit que deux types de commentaires (les commentaires javadoc étant dérivés des commentaires à la C).

```
/* Ceci est un commentaire classique
   Il peut s'étendre sur plusieurs lignes.
*/

// Ce type de commentaire ne peut pas s'étendre sur plusieurs
lignes

/** Et là nous avons un commentaire javadoc. Comme vous le
 *  remarquez, il commence par les caractères /**.
 *  Il peut s'étendre sur plusieurs lignes
 */
```

Les commentaires javadoc peuvent contenir des sections spéciales permettant d'adjoindre des informations supplémentaires sur les paramètres des méthodes, les valeurs de retour, les exceptions, ... ainsi vous pouvez documenter chacun des paramètres d'appels de la méthode via l'instruction @*param*, ainsi que la valeur de retour de cette dernière via l'instruction @*return*.

```
/**
         * crée un nouvel objet Nombre avec une valeur par défaut à 0
         */
        public Nombre() {
                this.nb = 0;
        }
```

TRAVAIL À FAIRE :

1 – Écrire la classe Nombre
2 – Générer la documentation
3 – Tester cette classe dans un programme