Les classes

1. Généralités

En fonction des applications que l'on veut concevoir, on rencontre différents concepts comme par exemple : des clients, des factures, des produits, des livres, des véhicules, etc. Pour représenter ces concepts au niveau informatique on utilise des classes.

Une classe représente donc un concept du problème à informatiser.

Dans une classe, on trouve :

1. des champs (appelés aussi variables d'instance, ou aussi attributs) qui représentent les caractéristiques du concept (par exemple pour une personne on trouvera comme attributs le nom, le prénom, l'adresse, etc.);
2. des méthodes qui représentent les opérations (les traitements, les actions) que l'on peut réaliser sur le concept (pour une personne, on peut avoir : afficher la personne, changer l'adresse, etc.).

Remarques :

- une bonne pratique consiste à stocker une classe par fichier source. Ce fichier doit porter exactement le même nom que la classe avec l'extension .java).
- Le nom d'une classe commence par une majuscule alors que le nom d'une variable référençant un objet commence par une minuscule.

Voir l'exemple 1.

2. Points de vue

Lorsque l'on programme, il faut distinguer deux points de vue :

- le point de vue du concepteur de la classe (celui qui écrit le code source) qui voit tout de la classe (partie publique et partie privée). La totalité du code source d'une classe s'appelle son implémentation.
- le point de vue de l'utilisateur de la classe qui ne voie que l'interface de celle-ci c'est-à-dire la partie publique. L'interface d'une classe décrit ce que fait la classe et la manière d'utiliser la classe sans montrer l'implémentation.

Une fois le code source d'une classe écrit dans un fichier, on peut compiler la classe. Une fois la compilation réussie, on peut créer des objets (aussi appelés instances) à partir de la classe.

Remarque : les mots clés *public* et *private* sont des appelés modificateurs d'accès.

3. Objet

C'est l'opérateur *new* suivi d'un constructeur avec éventuellement des paramètres qui va permettre de créer un objet. L'opérateur *new* renvoie une référence sur l'objet créer.

À partir d'une classe, on construit des objets qui sont des représentants concrets des concepts. Par exemple {« Jean », « DUPONT », « 12, rue de la Gare, Reims »} et

{« Anne », « MERCIER », « 15, avenue Robespierre, Epernay » } sont des représentants concrets de la classe Personne.

Pour créer un objet, il faut appelé un constructeur qui a pour but principal d'initialiser l'objet à l'aide de l'opérateur *new* qui renvoie une référence sur l'objet créé. On y initialisera entre autre les champs. Un constructeur est appelé au moment de la création de l'objet. Il existe toujours au moins un constructeur (si aucun constructeur n'est défini explicitement, un constructeur par défaut (sans paramètre) est généré).

Un constructeur n'a pas de valeur de retour. Il doit porté le même nom que la classe à laquelle il appartient.

Exemple :

```
public Personne(String pNom, String pPrenom) {
    nom = pNom;
    prenom = pPrenom;
    adresse = null; // pas de valeur connue
}
```

4. Démarche pour concevoir des classes

Quand on a une application à créer, il faut repérer les différents concepts utilisées (client, adresse, produit, etc.). Ces concepts seront modélisés par des classes.

Il faut ensuite déterminer les caractéristiques de chaque concept. Ces caractéristiques nous donneront les champs des classes. Par exemple, pour un produit, on peut avoir les caractéristiques suivantes : référence, libellé, poids, prix.

Enfin, il faut déterminer les actions que l'on peut faire sur les objets. Ces actions deviendrons les méthodes. Par exemple, pour un produit, on peut avoir les actions suivantes :

- créer un produit : *Produit(String reference, String libelle, int poids, double prix)* ;
- afficher le produit : *void afficher()* ;
- connaître le prix : *double getPrix()* ;
- modifier le poids : *void setPoids(int valeur)* ;
- savoir si un produit est plus lourd q'un autre : *boolean estPlusLourd(Produit p)* ;
- etc.

5. Variable – référence

Les variables ne contiennent pas les objets mais des références sur les objets.

Voir exemple 2.

Avant de pouvoir être utilisées, les variables doivent être déclarées c'est-à-dire que l'on doit préciser leur type. Il faut toujours qu'il y ait cohérence entre le type d'une variable et l'objet qu'elle référence sinon une erreur sera signalée.

Pour dire qu'une variable ne référence plus un objet, il faut utiliser le mot clé *null*.

Ainsi, dans l'exemple pour détruire le premier objet :

```
p1 = null; // la variable p1 ne référence aucun objet
```

L'objet précédemment référencé par *p1* n'est plus référencé, l'espace qu'il occupait dans le tas sera libéré lorsque le ramasse-miette se déclenchera.

Et pour le second objet :
```
p2 = null;
p3 = null;
```

6. Gestion de la mémoire - Ramasse-miettes

Une petite portion d'espace mémoire est réservé pour l'objet au moment de sa création dans un grand espace qu'on appelle le tas (*heap*). Le tas peut avoir une taille plus ou moins grande. Il peut arriver que cet espace mémoire soit complètement rempli. Dans ce cas une exception est levée qui signale la saturation de la mémoire.

Quand un objet n'est plus référencé, il a vocation à disparaître de la mémoire. Un mécanisme appelé ramasse-miettes (*garbage collector*) se déclenche périodiquement et est chargé de supprimer du tas les objets qui ne sont plus référencés libérant ainsi de l'espace mémoire qui sera à nouveau disponible pour de nouveaux objets.

7. Méthode

Les méthodes peuvent avoir des paramètres et renvoyer ou non une valeur.

Exemple :

```
public void setAdresse(String pNumero, String pVoie, String pVille) {
  adresse = new Adresse(pNumero, pVoie, pVille);
}
```

Parmi ces méthodes, il en existe des particulières que l'on nomme constructeur (voir plus haut).

Les valeurs des champs vont représenter l'état d'un objet.

Un objet (du point de vue de l'utilisateur de l'objet) doit être manipulé par l'intermédiaire de méthodes d'instance (méthodes qui ne sont pas déclarées *static*) c'est pourquoi les champs seront toujours déclarés privés (sauf quelques très très rares exceptions comme par exemple la classe *Point*). On ne veut pas qu'un utilisateur d'une classe manipule les champs directement (voir la partie Points de vue plus haut).

Pour consulter un champ on utilise des méthodes dites d'accès (ou accesseur ou *getter*). Pour modifier un champ on utilise des méthodes dites de modification (ou modificateur ou *setter*).

8. Encapsulation

Le principe de l'encapsulation est un grand principe de la programmation objet. Il consiste à cacher les informations liées à l'implémentation.

Ce principe veut que seules les informations au sujet de ce que peut faire une classe

doivent être visibles de l'extérieur alors que les informations montrant comment la classe rend ses services ne sont pas divulguées. Ainsi, si aucune classe ne sait comment les informations sont stockées, il est alors facile de modifier la façon dont sont stockées ces informations sans perturber les autres classes.

Exemple 3 : La classe *MonAppli* utilisatrice de *Personne* n'a pas été modifiée bien que *Personne* ait été modifiée.