CHAPITRE I

LES CONCEPTS OBJETS

A l'issue de ce chapitre I, vous saurez :

- Quels sont les **objectifs** de la Programmation Objet,
- Quels sont tous les **domaines** d'un projet informatique touchés par la Technologie Objet,
- Donner une définition au terme de **classe**,
- Expliquer ce qu'est l'**encapsulation** et ce qu'elle apporte à la maintenance des applications,
- Expliquer ce qu'est un **objet**.

SOMMAIRE

1. PRESENTATION GENERALE

Du binaire aux langages de quatrième génération, de l'ENIAC aux micro-ordinateurs sous Windows, l'évolution de l'informatique suit un même chemin qui se rapproche de l'utilisateur final pour s'éloigner de l'architecture interne de la machine.

Loin d'être une révolution, l'orienté objet n'est qu'une étape de cette évolution.

Celle-ci touche toutefois la plupart des domaines de la technique informatique :

- la **démarche de projets**, en implémentant un cycle itératif et incrémental ;
- la **modélisation**, en réconciliant les données et les traitements et en apportant des possibilités de liens supplémentaires entre objets ;
- le **stockage**, en autorisant la sauvegarde des objets dans des structures adéquates, les bases de données objets (SGBDO) ;
- la **programmation**, évolution logique et naturelle de la programmation structurée à travers des langages spécialisés.

Aborder l'orienté objet par la phase de programmation est donc un contresens. Un projet peut être qualifié d'orienté objet lorsque la modélisation de l'application développée s'est attachée à isoler des classes et à les ordonnancer en utilisant des liens d'héritage ou de composition spécifiques au modèle objet, et que l'accent a été porté sur la réutilisabilité des classes conçues.

Systèmes de gestion de bases de données et langages de programmation ne sont que des outils.

```
LOO (Langage orienté objet) :
- C++                                                          - Python
- Java ==> Le plus représenté en informatique de gestion       - R
- C# ==> le plus représenté en informatique de gestion
```

Si l'orienté objet fait tout juste son apparition dans le monde de la gestion au début des années 90, porté par la vague déferlante des micro-ordinateurs et de leurs interfaces graphiques, c'est néanmoins loin d'être une technologie naissante :

- le premier langage objets, SIMULA, date de 1967 ;
- modélisation et programmation orientées objets ont déjà fait leurs preuves en intelligence artificielle.

```
 Différents langages de la L3G (langage de 3ème génération) :
 - Cobol ==> informatique de gestion
 - Pascal ==> universitaire
 - PL/1
 - Fortron ==> scientifique
 - Basic
 - C
```

2. LES OBJECTIFS

2.1. LA MAITRISE DE LA COMPLEXITE

La maîtrise de la complexité fait appel a deux notions souvent considérées comme antagonistes, la modularisation et l'abstraction.

La **modularisation** est le découpage d'un problème général en sous-problèmes, puis en sous-sous-problèmes..., jusqu'à l'obtention d'entités facilement appréhendables et conduisant à une **hiérarchie** des composants. Elle est couramment mise en oeuvre lors de l'analyse top-down d'un programme par exemple.

L'**abstraction** consiste à isoler certaines caractéristiques d'un problème de manière à en construire une image représentative. L'abstraction opère par simplification, généralisation, sélection :

- la simplification dégrossit un acte en écartant les cas particuliers,
- la généralisation universalise des éléments simples ou complexes en des propriétés plus générales,
- la sélection distingue un point de vue qui semble approprié.

L'orienté objet a pour but de concilier ces deux approches.

2.2. LA REUTILISABILITE

La **réutilisabilité** est une notion ancienne, implémentée jusqu'alors à travers les différentes versions de la programmation structurée.

Cette implémentation est toutefois extrêmement limitée :

- elle induit surtout une réutilisation des composants au sein d'un module applicatif, dans le meilleur des cas au sein de tout un applicatif, mais jamais entre plusieurs applicatifs.
- elle permet principalement une réutilisation de composants basiques, utilitaires de bas niveau tels que les traitements des erreurs graves, les calculs de différence de dates,...
- elle est mise en oeuvre à travers la modularisation, outil imparfait qui garantit le découpage en composants mais en aucun cas leur aptitude à être réutilisés.

3. LA CLASSE

3.1. DEFINITION

Plus que l'objet, c'est la notion de classe qui caractérise la technologie orientée objet.

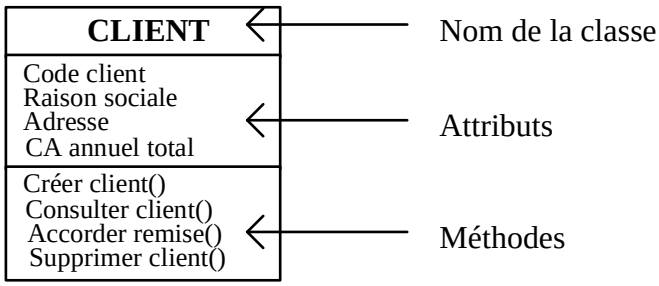
La classe peut-être définie comme un **modèle caractérisant un ensemble d'éléments du système d'information** :

- **modèle** signifie que la classe n'a aucune existence concrète ; elle ne représente que le moule à partir duquel seront créés les éléments manipulés par le système d'information ;
- **caractérisant** (au sens de *"définir par un caractère distinctif"*) indique que la classe n'a pas forcément pour vocation de décrire de manière exhaustive les propriétés d'un ensemble d'éléments, mais d'en permettre une abstraction plus ou moins élevée suivant l'avancement du processus de modélisation.
- un **élément du système d'information** peut-être décrit exhaustivement par la connaissance de son **état** et de son **comportement**, ces deux caractéristiques constituant l'**identité** de l'élément :
  - l'**état** d'un élément est représenté par la liste de ses propriétés (habituellement statiques), plus les valeurs courantes de chacune de ces propriétés ;
  - le **comportement** d'un élément est la façon dont celui-ci agit et réagit, en termes de changement de ses états et de circulation de messages.

La classe déclare donc :

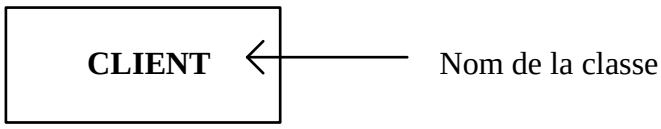
- la liste des **attributs** (ou des **connaissances**) permettant de connaître précisément l'état des éléments du système d'information qu'elle caractérise ;
- la liste des **méthodes** permettant de décrire de manière exhaustive le comportement des éléments du système d'information qu'elle caractérise ; les trois principaux types de méthodes sont :
  - les **constructeurs** qui interviennent lors de la création d'un élément ; `(instance de classe)`
  - les **accesseurs** qui permettent d'accéder à l'élément soit en consultation (on parle alors de sélecteur), en mise à jour (modificateur) ou dans un ordre défini (itérateur) ; `getter et setter`
  - les **destructeurs** qui interviennent lors de la suppression d'un élément.

Exemple :



NB1 : Le notation adoptée ci-dessus ainsi que dans le reste de ce stage est le formalisme UML (Unified Modeling Language) standardisé par l'OMG (Object Management Group) en 1997.

NB2 : On peut utiliser la notation simplifiée suivante :



NB3 : En tout état de cause, les attributs et les méthodes figurant sur les schémas de ce support de formation ne prétendent jamais à l'exhaustivité mais ont été choisis en fonction de leur intérêt pédagogique.

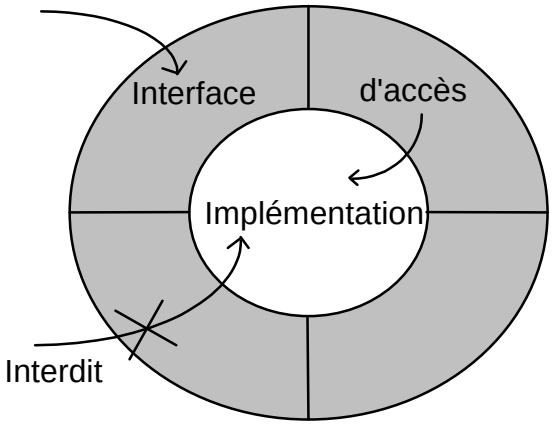
3.2. L'ENCAPSULATION (privé, protected et public)

Pour garantir la cohérence des valeurs d'attributs des occurrences d'une classe, il est intéressant de pouvoir forcer l'utilisateur de l'occurrence (le développeur d'applications) à n'accéder à son état que par l'intermédiaire de ses méthodes.

Ainsi si l'on veut garantir le respect de la règle de gestion du *Code client* de l'exemple page précédente (incrémentation de 10 en 10 par exemple), on protège cette zone qui devient inaccessible directement depuis l'environnement extérieur à la classe, sa gestion étant prise en charge uniquement par la méthode *Créer client()*.

On fera de même avec les autres attributs de la classe si l'on souhaite s'assurer que pour tout client les zones *Raison sociale* et *Adresse* ont été valorisées et que la zone *CA annuel total* est initialisée à 0.

Ce procédé de masquage de tous les attributs d'une classe qui ne seront donc accessibles que par le filtre des méthodes, aussi bien en consultation qu'en mise à jour, est connu sous le nom d'**encapsulation**.



Loin d'être une caractéristique anodine du modèle objet, l'encapsulation en est un des points forts puisqu'elle induit un couplage faible entre les programmes applicatifs et les éléments du système d'information, ce qui est particulièrement appréciable, entre autres, dans un contexte de maintenance.

Remarque :

L'implémentation du modèle objet varie fortement d'un langage de programmation à l'autre. En Java, en C# et en C++ par exemple, l'encapsulation n'est qu'une possibilité offerte alors qu'en Smalltalk elle est incontournable.

Quoiqu'il en soit, il est fortement conseillé de la mettre en oeuvre dans tous les contextes.

4. L'OBJET

L'objet est l'**occurrence ou l'instance d'une classe**.

Le procédé qui consiste à créer un objet à partir d'une classe est appelé **instanciation**.

L'instanciation d'un objet est effectuée par l'activation d'un de ses constructeurs. De même, sa suppression ne pourra être obtenue que par l'activation de son destructeur.

Exemple :