1. Chapitre 1.

- Rappels de la technologie orientée objet
- Principes fondamentaux de l'orienté objet

2. Concept orienté objet :

2.1. Notion d'objet et de classe

Qu'est-ce qu'un objet? Il est impossible de parler de Programmation ou de conception Orientée Objet sans parler d'objet. Concrètement, un objet est une structure de données valuées et cachées qui répond à un ensemble de messages. Cette *structure de données* définit son *état* tandis que l'ensemble des *messages* qu'il comprend décrit son *comportement.* En cela, rien ne distingue un objet d'une quelconque autre structure de données. La principale différence vient du fait que l'objet *regroupe les données et les moyens de traitement de ces données*.

Donc, un objet est une entité cohérente rassemblant des données et du code travaillant sur des données. Chaque chaque objet est désigné par une identité, des attributs et des opérations.

L'identité : l'objet possède une identité, qui permet de le distinguer des autres objets, indépendamment de son état. On construit généralement cette identité grâce à un identifiant découlant naturellement du problème (par exemple un produit pourra être repéré par un code, une voiture par un numéro de série, etc.) ;

Les attributs : il s'agit des données caractérisant l'objet. Ce sont des variables stockant des informations sur l'état de l'objet ;

Les méthodes : les méthodes d'un objet *caractérisent* son comportement, c'est-à-dire l'ensemble des actions (appelées opérations) que l'objet est amené à réaliser. Ces opérations permettent de faire réagir l'objet aux sollicitations extérieures (ou d'agir sur les autres objets). De plus, les opérations sont étroitement liées aux attributs, car leurs actions peuvent dépendre des valeurs des attributs, ou bien les modifier.

En résumé, un objet est un concept servant à représenter, modéliser **toute entité** décrite selon ses propriétés et son comportement (notion de classe). Ce comportement s'observe via les interactions que le monde extérieur peut avoir avec cette entité. Les interactions sont sollicitées à travers les opérations de l'objet.

Qu'est-ce qu'une classe? Avec la notion d'objet, il convient d'amener la notion de classe. Cette notion de classe n'est apparue qu'avec l'avènement de la nouvelle approche de la Programmation Orientée Objet.

La classe est une structure informatique particulière dans le langage objet qui est destiné à ***modéliser formellement des objets de même type***. Elle ***décrit*** la structure interne des données et elle définit les méthodes qui s'appliqueront ces objets. Le plus souvent les classes sont les ***descriptions*** des objets (on dit que les classes sont les méta-données des objets), lesquels sont des instances de leur classe.

Une classe propose des méthodes de création des objets dont la représentation sera donc celle donnée par la classe génératrice. Les objets sont dits alors **instances de la classe.** C'est pourquoi les attributs d'un objet sont aussi appelés *variables d'instance* et les messages *opération*

d'instance ou encore *méthodes d'instance*. L'interface de la classe (l'ensemble des opérations visibles) forme les types des objets.

En résumé : Classe (description de l'objet): [propriétés de classe ; comportement = ensemble d'opérations de la classe].

Objet (Instance de classe) : [attributs, variables d'instance ; opérations ou méthodes d'instance].

Exemple : La fabrication des costumes chez un tailleur. Le cabarie en carton représente la classe et les coustumes (différentes taille et couleurs représente les instances.)

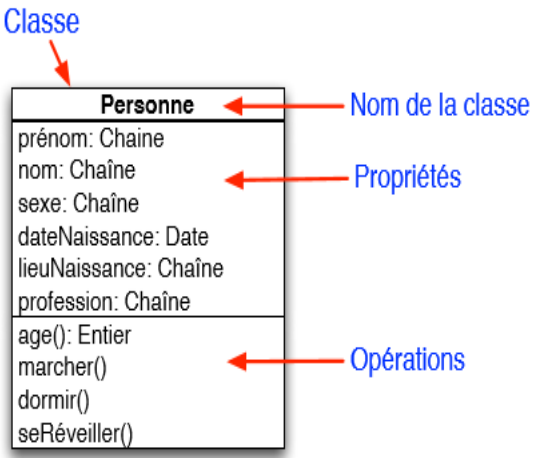
2.2. Exemples

2.2.1. Caractérisation simple d'une personne

Une personne est caractérisée par son prénom, son nom de famille, son sexe, sa date de naissance, son lieu de naissance, sa profession : ce sont ses propriétés.

Une personne peut exécuter les opérations suivantes : se réveiller, se lever, marcher, courir, sauter, s'asseoir, dormir, manger, lire, écrire, parler (dire son âge), se taire,...

Notation graphique utilisée est le diagramme de classe avec trois compartiments : nom de la classe, propriétés et opérations.



Classe : *Déclaration de type*

• nom = Personne

Propriétés : Déclaration *de propriétés*

Nom de propriétés	Type de données
Prénom, nom de famille, sexe, lieu de naissance, profession	Chaîne de caractères
date de naissance	Date

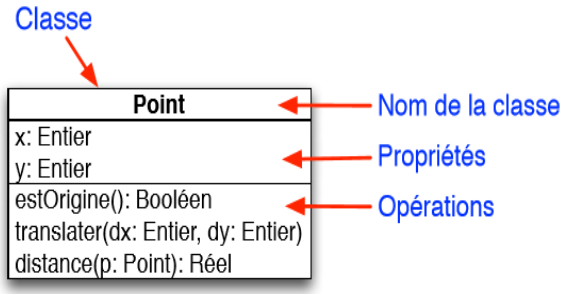
Opérations : *comportement des objets de type Personne*

Nom de l'opération	Type de retour
Age	Entier
Marcher, dormir, se reveiller	Rien (aucun type de retour : void)

2.2.2. Caractérisation simple d'un point du plan cartésien

Un point est caractérisé par son abscisse (x) et son ordonnée (y) : ce sont ses **propriétés**.

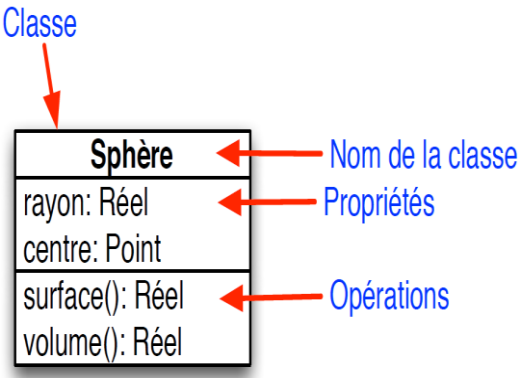
Nous peuvons exécuter les **opérations** suivantes sur un point : est-il à l'origine des axes? translation vers un autre point selon un delta (dx, dy), distance d'un autre point.



Classe : *Déclaration de type*
• nom = Point
Propriétés : *Déclaration de propriétés*
• nom = x; type de donnée = Entier
• nom = y; type de donnée = Entier
Opérations : *comportement des objets de type Point*
• nom = estOrigine; type de retour = Booléen
• nom = translater(dx : Entier, dy : Entier); type de retour = Rien • nom = distance(p : Point); type de retour = Réel

2.2.3. Caractérisation simple d'une sphère

Une sphère Possède deux propriétés : son rayon r et son centre O, qui est un objet de type Point
Connaissant r, on peut calculer sa surface (S = 4πR2) et son volume (V = 4/3πR3)

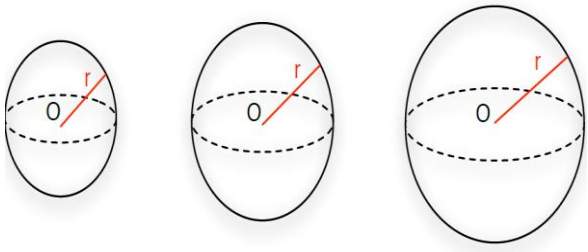


Classe : *Déclaration de type*
• nom = Sphère
Propriétés : *Déclaration de propriétés*
• nom = rayon; type de donnée = Réel; valeur par défaut = 0
• nom = centre; type de donnée = Point; valeur par défaut = Point(0, 0)
Opérations : *comportement des objets de type Sphère*
• nom = surface; type de retour = Réel • nom = volume; type de retour = Réel.
Surface et volume ne possèdent pas de paramètres formels (sous l'hypothèse que r est connu au sein des objets de type « Sphère »).

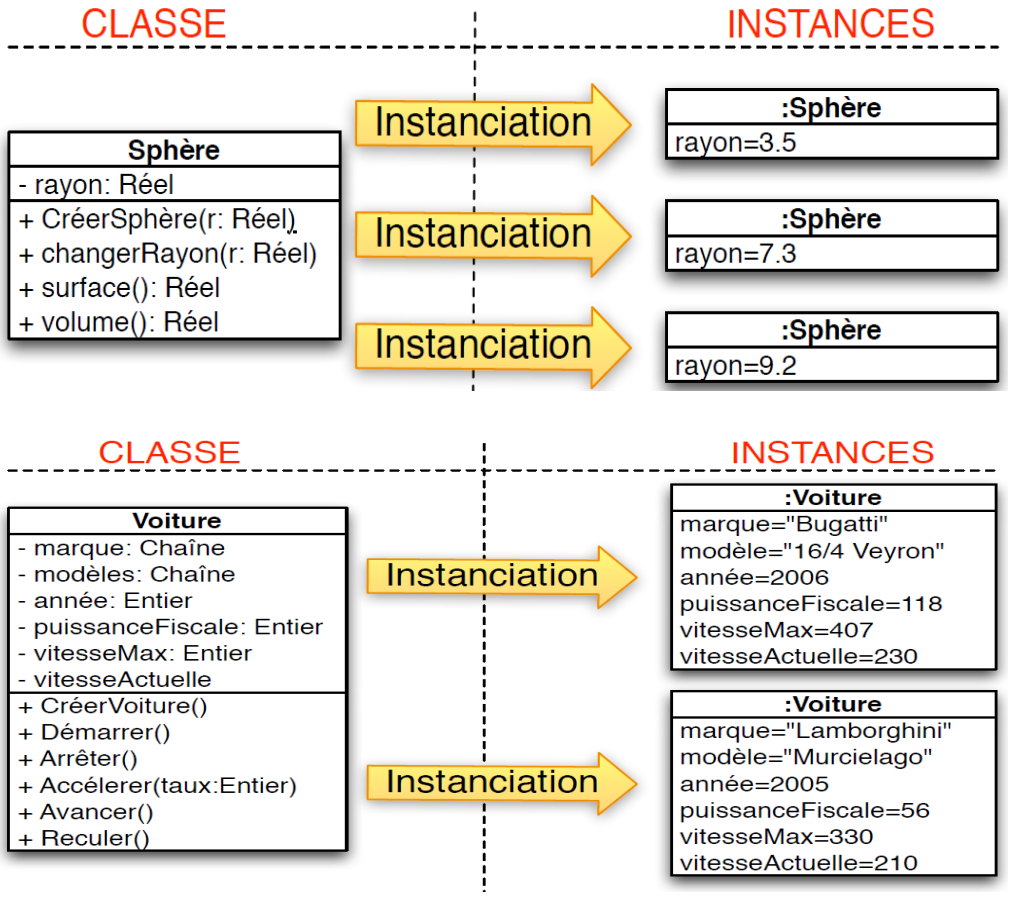
2.2.4. Instanciation de la classe Sphère

La classe Sphère dont la description graphique a été donnée précédemment décrit les objets de type < **Sphère** >. Chaque objet est une instance de cette classe. Nous pouvons en avoir d'autres **instances** de la classe Sphère.

Objet : *Déclaration d'une instance*
type = Sphère
Propriétés : *Initialisation des propriétés*
rayon = 3,5 ; (Positionné à la création de cet objet)
centre = Point(2,6) ; (Positionné à la création de cet objet)
A l'invocation des opérations : (on suppose que r n'a pas changé)
surface = 153,938
volume = 179,594



Instanciation d'une classe Sphère et d'une classe Voiture:



2.3. Caractéristiques essentielles de la POO

- Tout est objet ! : chaque objet encapsule des données et des méthodes agissant sur ces données.
- L'encapsulation réalise une abstraction des données : vu de l'extérieur de l'objet, les détails d'implémentation sont cachés.
- Le concept de classe généralise la notion de type : tous les objets d'une certaine classe sont des instances de cette classe.
- Des classes peuvent hériter d'autres classes (classe mère -classes filles). La notion d'héritage permet d'établir une hiérarchie entre les classes.
- Avec l'héritage, il devient possible de redéfinir des méthodes au sein des classes filles. nous parlons de polymorphisme.
- Tous les objets d'une même classe possèdent la même structure interne
- Un objet est une instance de la classe qui définit son type
- Les données membres de chaque objet prennent une valeur significative pour cet objet

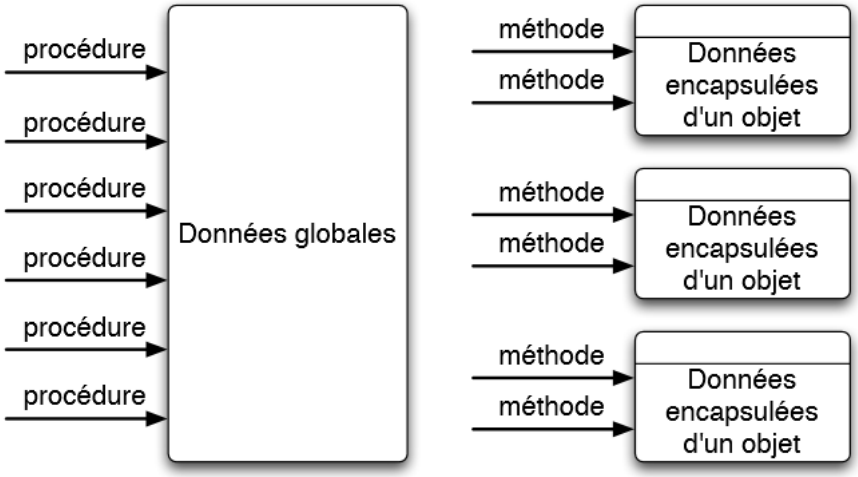
- Instanciation : création d'un objet avec ses données propres

2.4. POO vs. Programmation structurée :

Prog. structurée : détermination d'abord des procédures pour manipuler des données, puis de la structure à imposer aux données pour faciliter leur manipulation.

POO réalise l'inverse : détermination d'abord des données de façon cohérente dans des entités structurantes (objets), puis des opérations sur ces données.

POO est de ce point de vue plus modulaire, réutilisable, flexible et plus facile à maintenir



Considérant une application composée de 2000 procédures, développée selon la prog. structurée ou POO :

en prog. structurée, organisation du code très aléatoire avec du code lourd, difficilement réutilisable, variables internes exposées

en POO, on pourrait avoir 100 classes, 20 méthodes en moyenne par classe. Dont le code est vraiment modulaire, regroupements cohérents, contrôle d'accès aux variables internes. En l'avantage des bugs sont plus faciles à découvrir et corriger.

3. Les trois fondamentaux du paradigme orienté objet

Le paradigme objet est une nouvelle façon de concevoir un logiciel. L'accent est mis sur les données et les actions qu'elles supportent plutôt que sur une vision totalement procédurale. Le paradigme objet est basé sur trois principes fondamentaux : l'encapsulation, l'héritage et le polymorphisme.

3.1. Encapsulation

L'encapsulation introduit donc une nouvelle manière de gérer des données. Il ne s'agit plus de déclarer des données générales puis un ensemble de procédures et fonctions destinées à les gérer de manière séparée, mais bien de réunir le tout sous le couvert d'une seule et même entité.

L'encapsulation consiste à ***masquer les détails*** d'implémentation d'un objet, en définissant une *interface*. L'interface est la vue externe d'un objet, elle définit les services accessibles (offerts) aux utilisateurs de l'objet.

L'encapsulation facilite l'évolution d'une application, car elle stabilise l'utilisation des objets : on peut modifier l'implémentation des attributs d'un objet sans modifier son interface, et donc la façon dont l'objet est utilisé.

Pour conclure, l'encapsulation permet de garder une cohérence dans la gestion de l'objet, tout en assurant l'intégrité des données qui ne pourront être accédées qu'au travers des méthodes visibles (interdire ou restreindre l'accès direct aux attributs des objets.).

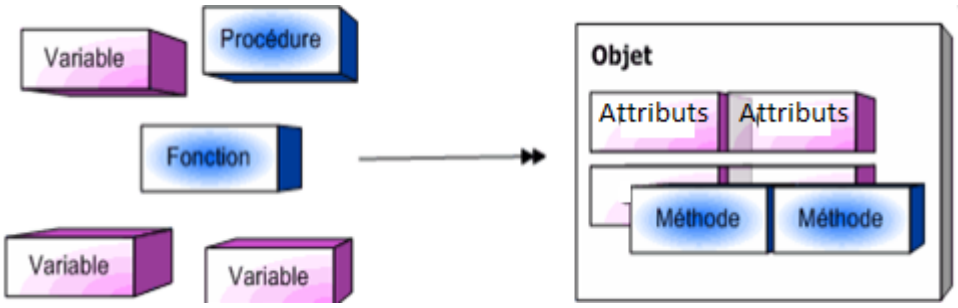


Figure 1: Encapsulation.

3.2. Héritage

L'héritage est le second des trois principes fondamentaux du paradigme orienté objet. Il est chargé de traduire le principe naturel de Généralisation / Spécialisation.

En effet, le terme d'héritage est basé sur l'idée qu'un objet spécialisé bénéficie ou hérite des caractéristiques de l'objet le plus général auquel il rajoute ses éléments propres.

En terme de concepts objets cela se traduit de la manière suivante :

- On associe une classe au concept le plus général, nous l'appellerons *classe de base* ou *classe mère* ou *super - classe*.
- Pour chaque concept spécialisé, on dérive une classe du concept de base. La nouvelle classe est dite *classe dérivée* ou *classe fille* ou *sous-classe*

L'héritage dénotant une relation de généralisation / spécialisation, on peut traduire toute relation d'héritage par la phrase :« La classe dérivée **est une** version spécialisée de sa classe de base »

En résumé, l'héritage est un mécanisme de transmission des caractéristiques d'une classe (ses propriétés et méthodes) vers une sous-classe. Une classe peut être spécialisée en d'autres classes, afin d'y ajouter des caractéristiques spécifiques ou d'en adapter certaines. Plusieurs classes peuvent être généralisées en une classe qui les factorise, afin de regrouper les caractéristiques communes d'un ensemble de classes. D'où l'héritage évite la *duplication* et encourage la *réutilisation*.

3.3. polymorphisme

Le terme polymorphisme est certainement celui que l'on appréhende le plus. Mais il ne faut pas s'arrêter à cela. Afin de mieux le cerner, il suffit d'analyser la structure du mot : **poly** comme plusieurs et **morphisme** comme forme. Le polymorphisme traite la capacité de l'objet à posséder plusieurs formes.

Cette capacité **dérive** directement du principe d'héritage vu précédemment. En effet, comme on le sait déjà, un objet va hériter des attributs et méthodes de ses ancêtres. Mais un objet garde toujours la capacité de pouvoir redéfinir une méthode afin de la réécrire, ou de la compléter. On voit donc apparaître ici ce concept de polymorphisme : choisir en fonction des besoins quelle méthode ancêtre appeler au cours de *l'exécution*. Le comportement de l'objet devient donc modifiable à volonté.

Le **polymorphisme**, en d'autres termes, est donc la **capacité du système à choisir dynamiquement** la méthode qui correspond au type réel de l'objet en cours. Ainsi, si l'on considère un objet Véhicule et ses descendants Bateau, Avion et Voiture possédant tous une méthode Avancer, le système appellera la fonction spécifique *Avancer()* selon le type de véhicule :un Bateau, un Avion ou bien une Voiture.

En résumé, le polymorphisme représente la faculté d'une méthode à pouvoir s'appliquer à des objets de classes différentes afin d'augmente la généricité et la qualité du code.

Note 1 : Le concept de polymorphisme ne doit pas être confondu avec celui d'héritage multiple. En effet, l'héritage multiple permet à un objet d'hériter des membres (attributs et méthodes) de plusieurs objets à la fois, alors que le polymorphisme réside dans la capacité d'un objet à modifier son comportement propre et celui de ses descendants au cours de l'exécution.