Cours de

Programmation Orientée Objet
POO

CONCEPTION ORIENTEE OBJET
COO

Par :
M. Khalifa MANSOURI

Sommaire

APPROCHE ORIENTEE OBJET

1. Introduction

Dans le monde de développement des applications informatiques, on peut noter, en général, deux types de programmation :

- programmation procédurale : faite à l'aide des langages traditionnels, linéaires, par exemple Fortran, Pascal, C, etc.
- programmation orientée objets faite à l'aide des langages objets, par exemple C ++, Java, etc. Les concepts de développement sont cependant très différents par rapport au premier cas.

La différence entre un langage objet et un langage normal est donc qu'un langage objet possède toutes les caractéristiques d'un langage traditionnel, avec en plus un nouvel aspect : l'utilisation des objets.

Un langage objet peut, par conséquent, être utilisé comme un langage procédural, ce qui pousse à négliger ce nouvel aspect très intéressant pour l'exploitation efficaces des capacités de la machine dans le développement d'applications interactive et ergonomiques.

Dans l'élaboration d'une application informatique, en général, la programmation structurée s'intéresse aux traitements (programme) puis aux données alors que la programmation orientée objet s'intéresse d'abord aux données, auxquelles elle associe ensuite des traitements. Le programmeur considère, dans ce dernier cas, des données en premier lieu et cherche à élaborer ensuite un programme autour de ces données.

L'élaboration d'un programme objet est très différente de celle d'un programme procédural. Ce dernier set constitué d'une succession d'instructions à exécuter progressivement les unes après les autres, avec des affectations, des opérations, des appels de procédures et/ou de fonctions effectuant chacune certaines taches précises. Un programme objet consiste en la modélisation des entités définies dans l'analyse du problème par des objets.

On note, cependant, que toutes les notions fondamentales et les techniques que le programmeur utilisait en algorithmique ou en programmation dans un langage traditionnel restent toujours valables dans un langage objet. La maîtrise de ces notions lui permettra de passer très vite aux nouveautés proposées dans ce dernier.

Dans tout langage objet, comme dans les langages traditionnels, on trouvera en général :

- des constantes, des variables simples de types standards et des variables indicées : tableaux,
- des instructions simples telles que l'affectation, les instructions d'entrées / sortie,
- des instructions composées telles que les instructions conditionnelles et alternatives, les instructions répétitives,

2. Notion d'objets

La particularité d'un langage objet est de donner au programmeur la possibilité d'utiliser des **objets**.

Dans un langage traditionnel, on utilisait une variable possédant un nom symbolique, un seul type donné (entier, réel, caractère, booléen, etc.) et occupant un emplacement mémoire réservé et ayant une adresse précise. Par conséquent une variable ne peut recevoir qu'une seule donnée.

Pour parler d'objets, on va essayer de généraliser la notion de variables en construisant une super-variable sous forme d'une structure de données formée par beaucoup de variables de différents types. Toutes ces variables caractérisent le même objet et tous les objets similaires à ce dernier doivent avoir les mêmes caractéristiques.

Chaque variable caractérisant un objet est désigné par un nom qui lui est propre.

Les noms des différentes variables caractérisant un objet s'appellent : **propriétés de l'objet**.

Par conséquent, toute propriété d'objet doit obéir aux même règles qui s'appliquent aux variables normales dans les langages traditionnels (déclaration, identificateur, type, taille, règles d'affectation, utilisation ... etc.)

Pour compléter la notion d'objets, on doit intégrer au sein de la structure de l'objet un certain nombre de procédures agissant sur ses propriétés appelées : **méthodes**.

Dans un objet on distingue donc deux parties :

- une partie statique : les propriétés (Données),
- une partie dynamique : les méthodes (Traitements).

3. Notion de classe

Dans un langage normal, toutes les variables de même nature sont dites de même Type.

Dans un langage objet, tous les objets qui possèdent les mêmes propriétés sont du même **type**, ou bien de même **classe**.

Les classes sont des modules autonomes accomplissant certaines tâches. Ces classes permettent de créer des objets par l'instanciation de classes.

La programmation orientée objet permet de développer des applications à partir d'objets, constituant des entités possédant des propriétés, des méthodes et des événements

La classification signifie que les objets ayant la même structure de donnée « attributs » et le même comportement « méthodes » sont regroupés en une classe. Les objets d'une classe ont donc le même type de comportement et les mêmes attributs.

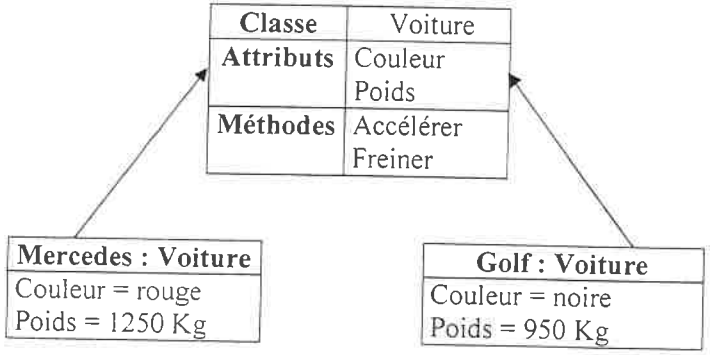
En groupant les objets en classe, on abstrait un problème. Les définitions communes par exemple le nom de la classe et les noms d'attributs sont stockées une fois par classe plutôt qu'une fois par instance.

Les méthodes peuvent être écrites une fois par classe, de telle façon que tous les objets de la classe bénéficient de la réutilisation du code. Par exemple, tous les cercles partagent les mêmes procédures de dessin, de calcul de périmètre ou de test d'intersection avec une ligne.

Une classe est un modèle utilisé pour créer plusieurs objets présentant des caractéristiques communes.

Chaque objet possède ses propres valeurs pour chaque attribut mais partage les noms d'attributs et méthodes avec les autres objets de la classe.

Par exemple une classe **Voiture** peut être définie comme ayant un des attributs **couleur** et comme une des méthodes **freiner** ; les objets associés à la classe voiture peuvent être : Mercedes rouge, Golf noire, etc. Ces objets ont le même comportement (freiner).



4. Etude d'un exemple d'application

4.1. Présentation

Prenons comme exemple d'objet : un athlète.
Les caractéristiques d'un athlète sont : sa taille, son poids, son âge, sa spécialité, son record personnel, son nom, sa situation familiale, etc.

Les types de données de ces caractéristiques sont donnés comme suit :

- la spécialité, le nom, sont de type **chaînes de caractères,**
- la taille, le poids, l'âge, le record personnel, sont de type **numérique,**

- la situation familiale (marié ou non) est de type **booléenne.**

4.2. Propriétés

Les propriétés représentent les manières qui permettent d'accéder aux valeurs des champs d'un objet.

Les propriétés (ou attributs) d'un objet contiennent les valeurs relatives à cet objet, et réagissent comme des variables normales pouvant recevoir un nombre, une chaîne de caractère, etc..

Pour référencer une propriété à un objet, on utilise un point séparateur, la syntaxe est la suivante :

```
objet.propriété
```

Les propriétés d'un objet peuvent être, lues, affectées ou affichées.

Prenons par exemple un premier athlète qu'on va identifier par : **Coureur**.

Coureur est donc un objet de type (de classe) Athlète et doit avoir toutes les propriétés d'un athlète.

Pour dire que le nom de cet athlète coureur est « Ali », on considère que la propriété désignant le nom au sein de l'objet est : la propriété **Nom** et on écrira l'instruction suivante :

```
Coureur.Nom = « Ali »
```

Considérons encore que la propriété désignant le record personnel des objets de type (de classe) athlète est la propriété **RecordPersonnel**. Pour dire que le record personnel de l'athlète Coureur est 12,45 mn, on écrira donc l'instruction suivante :

```
Coureur.RecordPesonnel = 12.45
```

Pour affecter à une variable V le record actuel de l'athlète Coureur, on écrira :

```
V = Coureur.RecordPesonnel
```

Pour diminuer de 0,15 mn le record personnel de l'athlète Coureur, on écrira :

```
Coureur.RecordPesonnel =
                    coureur.RecordPesonnel - 0.15
```

Pour dire que cet athlète est marié, on utilise une propriété booléenne appelée **Situationfamiliale** de la manière suivante :

```
Coureur.SituationFamiliale = True
```

En ce qui concerne la syntaxe, on remarque qu'il n'y a pas de différence entre l'utilisation des propriétés des objets et les variables classiques.

4.3. Méthodes

Une **méthode** est une manière d'agir sur une ou plusieurs propriétés d'un objet.

On peut considérer une méthode comme une procédure ou une fonction ayant un certain nombre d'**arguments**. On trouvera donc des méthodes à un argument, des méthodes à deux arguments et aussi des méthodes sans arguments.

Un méthode peut avoir comme mission de réaliser une tache indépendante ou faciliter la programmation en modifiant automatiquement certaines propriétés de l'objet en question.

Reprenons l'exemple de notre athlète Coureur et considérons une méthode permettant de diminuer son record personnel appelée **DiminuerRecord**. Cette méthode va utiliser un argument de type numérique : la valeur de diminution du record

personnel. L'appel à cette méthode se fait de la manière suivante :

```
Coureur.DiminuerRecord ( 0.15 )
```

Dans ce cas le record personnel de notre athlète va diminuer de 0.15 mn. De point de vue programmation, cette méthode peut remplacer avantageusement la ligne suivante agissant directement sur la propriété **RecordPersonnel** de l'objet :

```
Couruer.RecordPesonnel =
                    coureur.RecordPesonnel - 0.15
```

5. Utilisation des objets

Au sein d'un langage orienté objet, on note deux type d'objets :

- **Objets standards :** ils sont fournis par le langage de programmation lui-même. On peut donner comme exemples les « **contrôles** », qui permettent de réaliser des interfaces graphiques pour différentes applications, pour utiliser ces objets, le programmeur doit modifier et personnaliser leurs **propriétés** et leurs **méthodes** car elles sont déjà **fixées** à l'avance.

  Chaque classe d'objets possède ses propres méthodes, il faut donc s'y conformer pour bien utiliser un objet de cette classe.

  La programmation avec des objets standards se classe comme niveau d'utilisation simple du langage objet.

- **Objets non-standards** : un langage objet nous donne la possibilité de créer de nouveaux objets non-standards appelés objets propres au programmeur qui doit programmer leurs propriétés et méthodes en partant d'une modélisation de son cahier des charges.

La création des objets se classe comme vraie programmation orientée objet.

6. Les concepts théoriques de base utilisé en POO

On note trois type de concepts :

- Le concept qui consiste à modéliser des exigences issus du cahier des charges et essayer de réunir des données et des traitements dans une seule **classe d'objets.**
- Le concept qui consiste à créer des objets à partir du modèle qui est la classe. Ces objets s'appèlent instances et permettent de représenter individuellement la classe. La création s'appèle instanciation.
- Le concept qui consiste à gérer et organiser les classes, on l'appelle préfixage qui peut servir pour la création de certaines classes à partir d'autres classes, ce phénomène est appelé héritage.

7. L'encapsulation de données

7.1. Définition

L'encapsulation est le principe qui permet de regrouper les attributs et méthodes au sein d'une classe. Elle instaure un dispositif de protection permettant de contrôler la visibilité d'un attribut ou d'une méthode. Ce qui implique que lorsqu'on définit un membre d'une classe (attribut ou méthode), on doit indiquer les droits d'accès pour l'utilisation de ce membre.

L'encapsulation empêche les utilisateurs clients de connaître les détails de l'implémentation d'un objet en ne fournit qu'une vue externe (correspondant à un masquage d'information). Seule l'interface (ou protocole) d'un objet doit paraître visible aux yeux d'un client potentiel.

Ce mécanisme d'encapsulation permet surtout de protéger l'objet de toute malveillance externe. Pour cela, la plupart du temps, il faut interdire l'accès direct aux attributs et passer systématiquement par les méthodes.

Par exemple : si l'on désire changer la couleur d'une voiture, cela ne se fait pas directement, il est nécessaire de passer par tout un processus (décaper, poncer, passer plusieurs couches, ... etc.), et dans ce cas de figure l'attribut couleur ne doit pas être accessible directement.

7.2. Niveaux de protection

Il existe trois niveaux de protection :

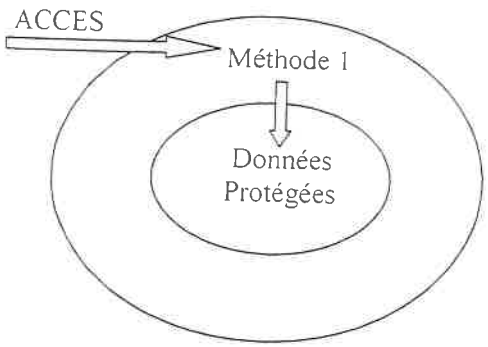
public : Tous les attributs ou méthodes d'une classe définies avec le mot clé public sont utilisables par tous les objets. Il s'agit du niveau le plus bas de protection. Ce type de protection est employé pour indiquer qu'on peut utiliser sans contrainte les attributs et les méthodes d'une classe.

private : Tous les membres d'une classe définis avec le mot clé private sont utilisables uniquement par les méthodes de la classe. Cette étiquette de protection constitue le niveau le plus fort de protection. Généralement les attributs doivent être déclarés comme privés.

protected : Tous les membres d'une classe définis avec le mot clé protected sont utilisables uniquement par les méthodes de la classe et par les méthodes des classes dérivées (par les enfants). Cette technique de protection est fortement associée à la notion d'héritage .

7.3. Possibilités d'encapsulation

7.3.1. Première possibilité : Encapsulation de données



Exemple :

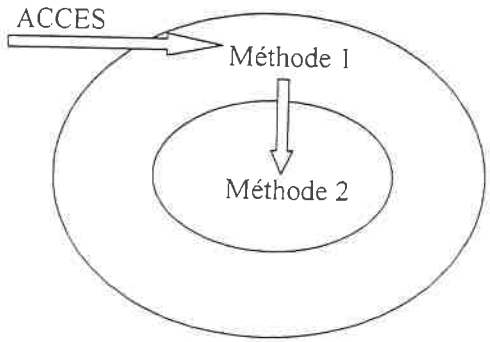
ACCES : à la base de données des employés à travers Internet

Méthode 1 : Avoir sa situation administrative, en introduisant un code d'accès personnel.

Objet : Employé (Dans le secteur public)

Donnée protégée : Codes d'accès personnels des autres employés.

7.3.2. Deuxième possibilité : Encapsulation de méthodes



Exemple :

ACCES : retirer l'argent liquide sur un guichet bancaire automatique.

Objet : Compte bancaire.

Méthode 1 : (Visible depuis l'interface du guichet) Introduire son code confidentiel.

Méthode 2 : (non visible par l'utilisateur mais appelée par la méthode 1) Validation du code confidentiel.

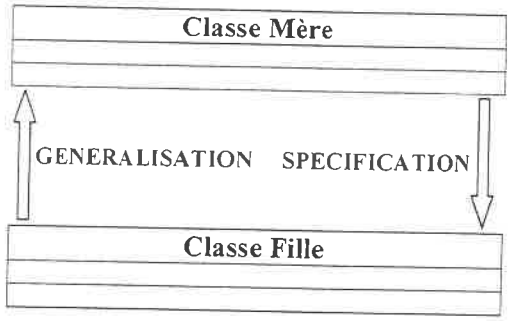
Remarque :

Des méthodes encapsulées peuvent faire appel à des données qui sont elles-mêmes protégées.

8. Concepts d'héritage

8.1. Qu'est-ce qu'un héritage ?

L'héritage est un mécanisme destiné à exprimer les similitudes entre classes. Il met en oeuvre les principes de généralisation et de spécialisation en partageant explicitement les attributs et méthodes communs au moyen d'une hiérarchie de classes.



L'héritage est une notion fondamentale de la programmation orientée objet, il est à la base des possibilités de réutilisation des composants logiciels existants en particulier les classes.

8.2. Généralisation

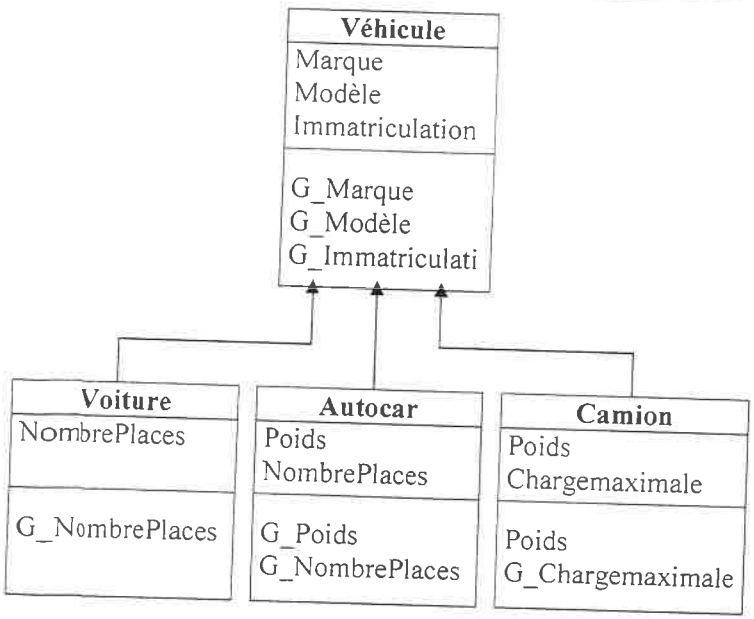
Supposons que nous voulons développer une application informatique permettant de gérer le parc autos d'une entreprise. Cette dernière utilise plusieurs types de véhicules : des voitures, des camions et les autocars.

Une première analyse nous conduit à représenter les trois grandes familles de véhicules par trois (types) classes différentes : Voiture, Camion et Autocar représentées des la manière suivante :

	Voiture	Autocar	Camion
Propriétés	Marque Modèle Immatriculation NombrePlaces	Marque Modèle Immatriculation Poids NombrePlaces	Marque Modèle Immatriculation Poids Chargemaximale
Méthodes	G_Marque G_Modèle G_Immatriculati G_NombrePlaces	G_Marque G_Modèle G_Immatriculati G_Poids G_NombrePlaces	G_Marque G_Modèle G_Immatriculation G_Poids G_Chargemaximale

Dans la définition des trois types de véhicules, il y a un certain nombre de caractéristiques qui se répètent. Pour éviter cette répétition, on va construire une nouvelle classe plus généraliste qui va factoriser toutes les caractéristiques communes.

D'une façon générale, on peut dire que quel que soit le type de véhicule, il comporte au moins une marque, un modèle, ...etc. Cette nouvelle classe à construire va s'appeler véhicule. On va ensuite instaurer la filiation par la création de relations entre les différentes classes. En précisant par exemple q 'une voiture est un véhicule.



On matérialise la généralisation par une flèche partant de la classe fille vers la classe mère.

Exemple :

Une Voiture possède, un nombre de places, mais d 'après la relation de généralisation, elle possède également une marque, un modèle et une immatriculation.

On dit que la classe Voiture hérite de toutes les caractéristiques de la classe Véhicule.

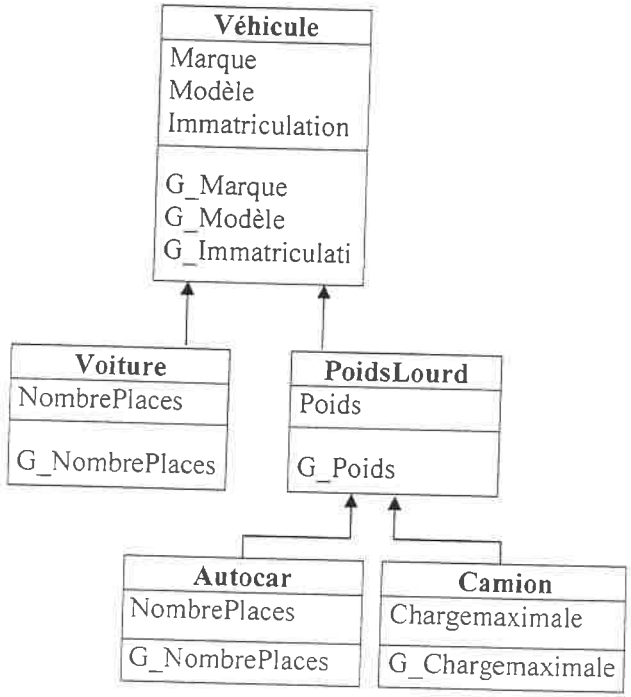
Cette description est la même que celle faite précédemment en évitant toutes les répétitions. Toutes les caractéristiques communes aux classes filles n'apparaissent plus qu'une fois dans la nouvelle classe Véhicule et s'introduisent automatiquement dans ces classes filles à travers les relations de généralisation.

Remarque :

Dans cet exemple, on a utilisé un seul niveau d'héritage. On peut cependant utiliser plusieurs niveaux selon le cas.

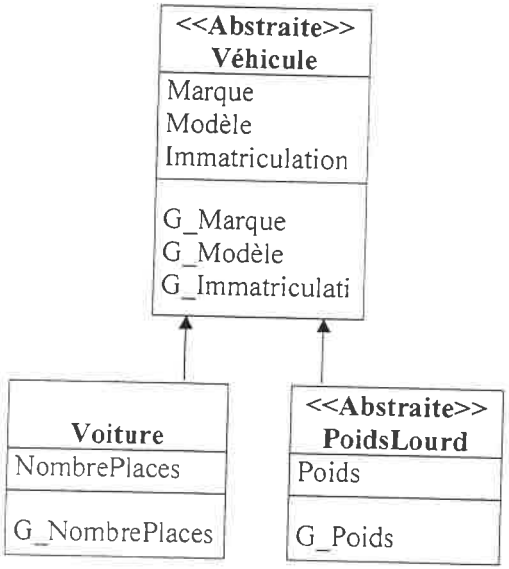
Exemple :

Dans le même exemple ci dessus, on remarque que les deux classes Autocar et Camion possèdent un chose en commun, le poids. On appliquera alors une nouvelle généralisation en factorisant le poids du camion et de l'autocar.
On va créer alors une nouvelle classe appelée : PoidLourd.



Remarque :

Dans l'application de gestion des véhicules, on n'aura aucun objet relatif à la classe Véhicule. Notre gestion concerne en fait, les voitures, les autocars et les camions. Déclarer un objet de la classe Véhicule n'aurait pas de sens. Il s'agit d'une abstraction. Les classes filles correspondent à quelque chose de concret.
C'est aussi le cas pour la classe PoidsLourd.
Il ne faut donc pas autoriser ces classes à produire des objets. En spécifiant qu'elles sont abstraites alors que les classes filles sont concrètes, et ceci de la manière suivante :



Les objets à utiliser ne peuvent se déclarer que sur des classes concrètes.

8.3. Spécification

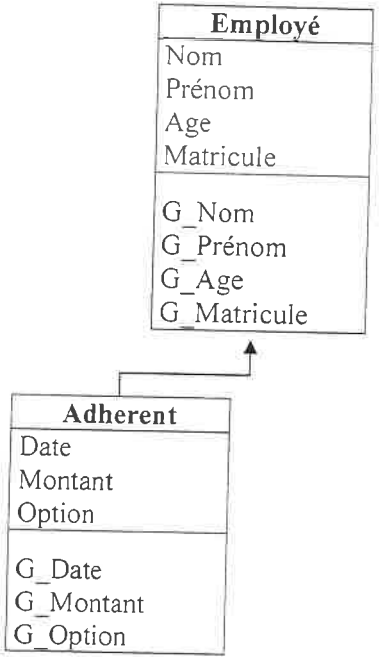
La spécification est le fait de partir d'une classe mère pour aboutir ensuite aux classes filles.
On peut définir alors, une nouvelle classe, dite dérivée, à partir d'une classe existante dite de base. La classe dérivée héritera certaines caractéristiques de la classe de base, tout en s'ajoutant de nouvelles caractéristiques spécifiques sans influencer la classe de base. On n'aura pas besoin de recompiler la nouvelle classe, ni même de disposer du son programme source correspondant.

On peut alors développer de nouveaux outils en se basant sur d'autres outils déjà acquis à l'avance, d'où l'appellation : d'héritage.

Dans la technique d'héritage, plusieurs classes peuvent être dérivées d'une même classe de base. Et une classe dérivée peut devenir à son tour classe de base pour d'autres classes dérivées. On dit alors que l'héritage est outil de spécialisation croissante.

Exemple :

Dans la gestion d'une entreprise, on considère q'un dispose déjà d'une classe concrète Employé. Parmi tous les employés, il y a ceux qui désirent adhérer à une assurance privée. Nous pouvons alors spécialiser la classe Employé pour obtenir une classe Adhérent qui représente bien des employés mais spécifiques en possédant en plus un certain nombre de caractéristiques comme, par exemple, la date d'adhérence, le montant de participation, les options préférées, ...etc.



8.4. Les différents types d'héritage

On distingue deux types d'héritage : l'héritage simple et l'héritage multiple.

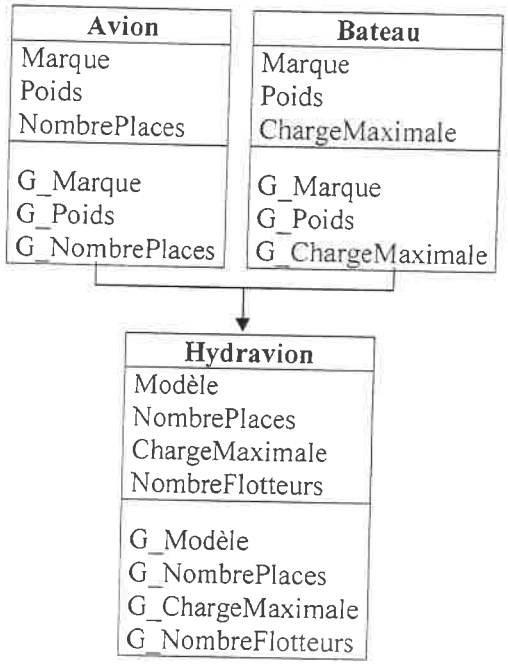
8.4.1. Héritage simple

Une classe fille n'a qu'une seule classe mère et elle hérite d'une partie des attributs et méthodes de celle-ci en ayant ses spécifications propres. C'est ce qu'on a vu dans les paragraphes ci dessus.

8.4.2. Héritage multiple :

Une classe fille hérite de plusieurs classes mères. Elle hérite d'une partie des attributs et des méthodes de chacune de ses classes mères, en plus de ses spécifications propres.

Exemple :



Remarque :

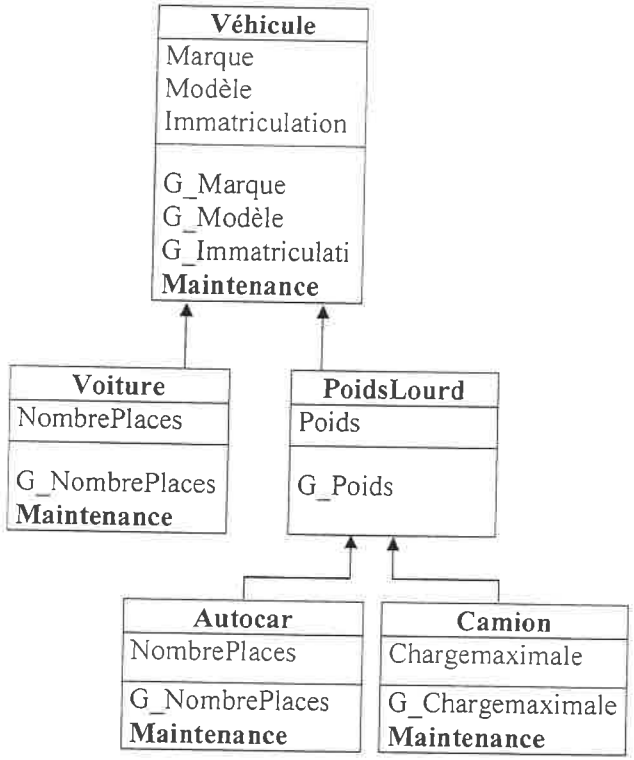
Beaucoup de langages orientés objets à objets n'utilisent pas l'héritage multiple.

9. Le polymorphisme

Le polymorphisme est un moyen permettant d'utiliser une méthode commune à une hiérarchie d'objets sous plusieurs formes.
Le forme à utiliser est déterminée lors de l'exécution de l'application suivant la classe de l'objet courant.

Exemple :

Reprenons l'exemple des véhicules étudié précédemment en considérant que chaque type de véhicule peut subir une maintenance pour le garder en bon état.
Il s'agit donc d'ajouter une nouvelle méthode qui va porter le même nom « Maintenance » et qui va apparaître sur toutes les classes faisant partie de la hiérarchie de véhicules.



Le principe de l'héritage est que lorsqu'une méthode est décrite sur une classe parente, elle est automatiquement hérité par les classes filles, ce qui nous permettra de ne pas la re-déclarer forcément au sein de ces dernières.

Le fait de ré-utiliser le même nom de la méthode dans les classes filles, veut dire qu'elles vont se présenter sous d'autres formes et qu'elles auront d'autres comportements différents selon la classe de l'objet utilisé. En réalité la maintenance d'une voiture ne se fait pas de la même manière que celle d'un camion.

L'exemple des véhicules modifié présente bien un polymorphisme.

La description générale de la maintenance d'un véhicule ne suffit pas pour maintenir un camion par exemple.

Dans la classe fille Camion, on va redéfinir la méthode maintenance qui va porter le même nom avec une signature identique (polymorphisme) que la classe de base. On dit qu'on a fait une **redéfinition** de la méthode.

Remarque :

Si dans une application, on n'utilise pas cette notion de polymorphisme, alors que c'est nécessaire, il faut utiliser une instruction à choix multiple (cas) suivant la classe des objets rencontrés.

En conséquence, à chaque fois qu'on ajoute ou on supprime une classe, on serait obligé de modifier le programme de l'application.

10. Exercices

Exercice 1 : Construction d'une classe « VOITURE »

Enoncé de l'exercice :

Considérons une classe « VOITURE », représentant un modèle de l'ensemble des voitures du tourisme en circulation.

a- Citer quelques propriétés de cette classe,
b- Citer quelques méthodes de cette classe,
c- Citer deux objets de classe voiture. Donner des valeurs aux propriétés.

Solution de l'exercice :

Considérons une classe « VOITURE »,

a- Les attribues de cette classe peuvent être résumées dans le tableau suivant :

Classe « VOITURE »
PROPRIETES
Marque
Modèle
Immatriculation
N° de châssis
Puissance fiscale
Poids
Vitesse maximale

b- Les méthodes de cette classe peuvent être résumées dans le tableau suivant :

Classe « VOITURE »
METHODES
Démarrer
Accélérer
Freiner
Vidanger
Changer pneu
Régler parallélisme
Allumer fars

c- Un OBJET est une instance de la classe « VOITURE ». C'est un élément construit selon le modèle de cette classe.

Exemples d'objets de la classe « VOITURE »	
Voiture 1	**Voiture 2**
Volkswagen	Peugeot
Vento	406
14 A 1345	2 A 4567
Z 123576899886	Z 6798766654
8 Ch	7 Ch
1324	1534
240 Km/H	200 Km/H

Exercice 2 : Construction d'une classe « COMPLEXE »

Enoncé de l'exercice :

Un nombre complexe est connu par sa partie réelle et sa partie imaginaire sous la forme suivante : **Z = PR + i.PI,**

Le module d'un nombre complexe se calcule de la manière suivante : $\mathbf{M = (PR^2 + PI^2)^{1/2}}$

L'argument d'un nombre complexe se calcule de la manière suivante : **A = ArcTg(PI/PR)**

On désire construire une classe « COMPLEXE », intégrant les parties réelles et imaginaires du nombre complexe ainsi que les calculs de son module et son argument.

a- Citer les propriétés de cette classe et donner leurs types,
b- Citer les méthodes de cette classe,
c- Ecrire les algorithmes-codes de ces propriétés.

Solution de l'exercice :

Soit la classe « COMPLEXE ».

a- On distingue deux attribues de cette classe :

- La partie réelle du nombre complexe : PR qui est de type réel,
- La partie imaginaire du nombre complexe : PI qui est aussi de type réel,

b- On distingue deux méthodes de cette classe :

- La méthode « Module » qui peut se présenter comme une fonction ayant un paramètre d'entée, le nombre complexe et retournant un nombre réel, le module du nombre complexe.
- La méthode « Argument » qui peut se présenter comme une fonction ayant un paramètre d'entée, le nombre complexe et retournant un nombre réel, l'argument du nombre complexe.

c- Les algorithmes-codes de ces propriétés peuvent se présenter comme suit :

```
Fonction Argument( )( ) : réel
Début
    Const
       PI = 3.14
    Var
       A : Réel
    Si PI = 0 alors
       Retourner(0)
    Sinon
       Si PR = 0 alors
             Si PI > 0 alors
                   Retourner(PI/2)
             Sinon
                   Retourner(-PI/2)
             Fin Si
       Sinon
             Retourner(Atan(PI/PR))
       Fin Si
    Fin SI
Fin
```

```
Fonction Module( ) : réel
Début
    Var
      M : Réel

    A =  Sqrt(PR*PR + PI *PI)
    Retourner(M)
Fin
```

Exercice 3 : Construction d'une classe « ELEVE »

Enoncé de l'exercice :

Dans une classe, un élève set connu par :

- son nom,
- son prénom,
- son adresse,
- sa date de naissance,
- son numéro d'inscription,
- sa note de math à l'examen,
- sa note de physique à l'examen.
- sa note d'informatique à l'examen.

a- Définir les propriétés de la classe ELEVE en précisant leurs types,

b- Soit un objet « Inscris » de classe ELEVE, affecter des valeurs aux propriétés de cet objet,

c- Définir une méthode permettant de calculer et d'afficher sa moyenne générale sachant que toutes les matières ont un coefficient 1.

Solution de l'exercice :

Considérons une classe « ELEVE »,

a- Les propriétés de cette classe et leurs types peuvent être résumés dans le tableau suivant :

Classe « ELEVE »	
PROPRIETES	**TYPES**
Nom	Chaîne de 12 caractères
Prenom	Chaîne de 15 caractères
Adresse	Chaîne de 35 caractères
DateNaissance	Chaîne de 10 caractères
NumeroIncription	Entier long
NoteMath	Réel
NotePhysique	Réel
NoteInformatique	Réel

b- Soit un objet « Inscris » de classe ELEVE, voilà un exemple de valeurs des propriétés de cet objet,

```
Inscris.Nom = « SAMADI »
Inscris.Prenom = « ALI »
Inscris.Adresse = « 40, Lotossement AMAL, Casa »
Inscris.DateNaissance = « 12/05/78 »
Inscris.NumeroInscription = 12305
Inscris.NoteMath = 14.5
Inscris.NotePhysique = 17.25
Inscris.NoteInformatique = 15.5
```

c- Une méthode permettant de calculer la moyenne générale de l'objet « Inscris »

```
Procédure Moyenne(NM, NP, NI : Réels)( )
Début
      MG : Réel
      MG = (NM + NP + NI)/3
      Ecrire('La moyenne générale est : ', MG)
Fin
```

COO

- Classe : variable globale ayant des propriétés et des méthodes
- Objet
- propriété
- Méthode
- Instanciation
- Abstraction : Passage du monde réel vers un Σ de classes
- Encapsulation : Protection de données (3 niveaux)
  - Public
  - Protected
  - Private
- Héritage (Simple, multiple)
- Généralisation
- Spécification
- Abstraite
- Polymorphisme
- Redéfinition