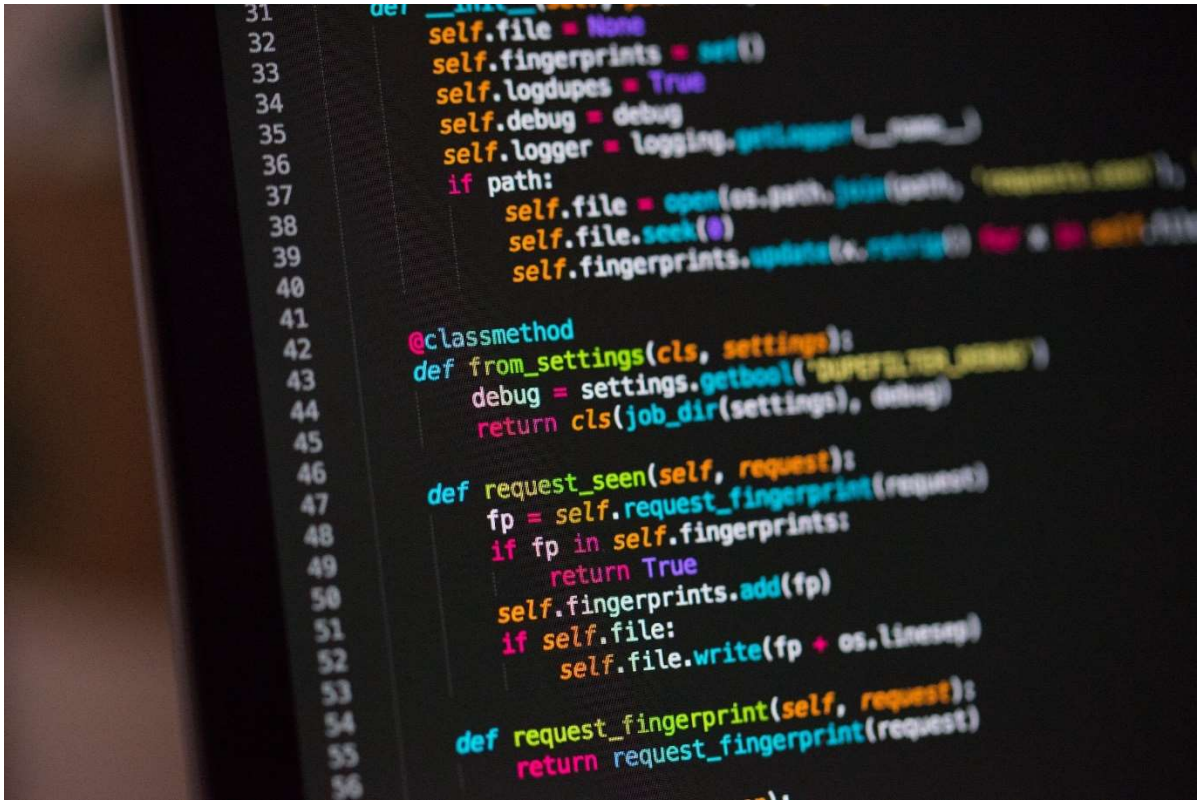
Chapitre 1 – Les notions essentielles

• A- L'orientation objet en quelques mots

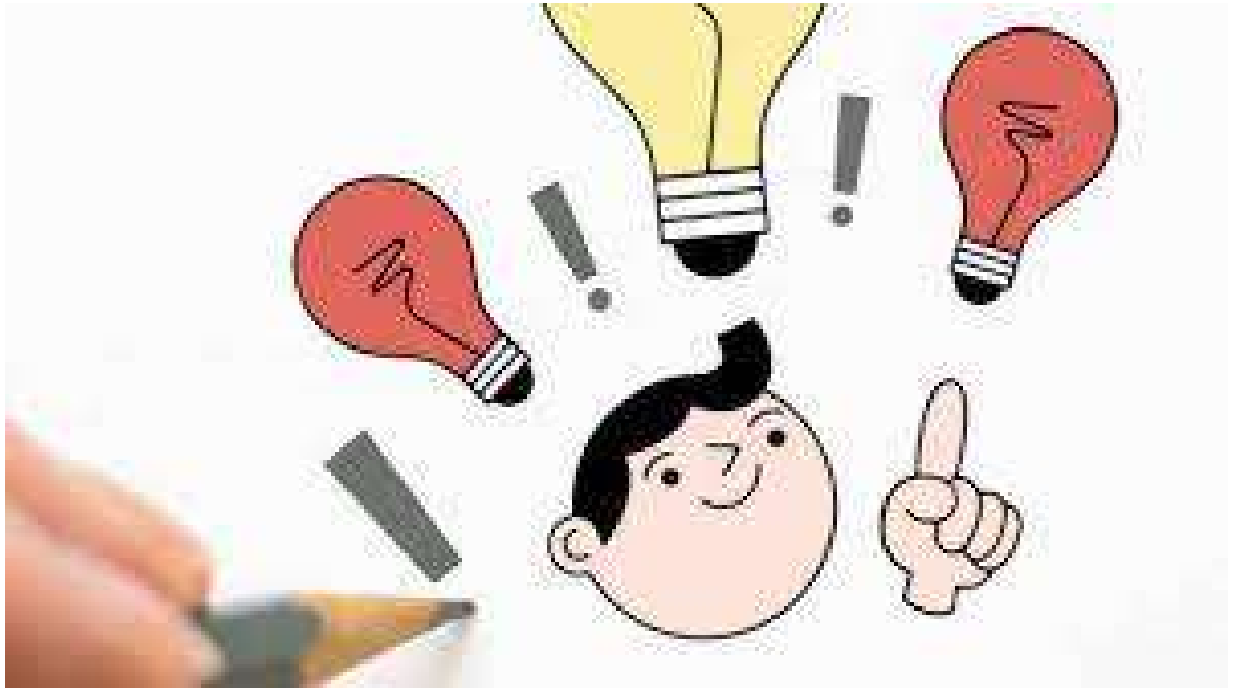
- Préambule général

À la différence de la programmation procédurale, un programme écrit dans un langage objet répartit l'effort de résolution de problèmes sur un ensemble d'objets collaborant par envoi de messages.

L'orienté objet inscrit la programmation dans une démarche somme toute très classique pour affronter la complexité de quelque problème qui soit : une découpe naturelle et intuitive en des parties plus simples. A fortiori, cette découpe sera d'autant plus intuitive qu'elle s'inspire de notre manière « cognitive » de découper la réalité qui nous entoure. L'héritage, reflet fidèle de notre organisation cognitive, en est le témoignage le plus éclatant.

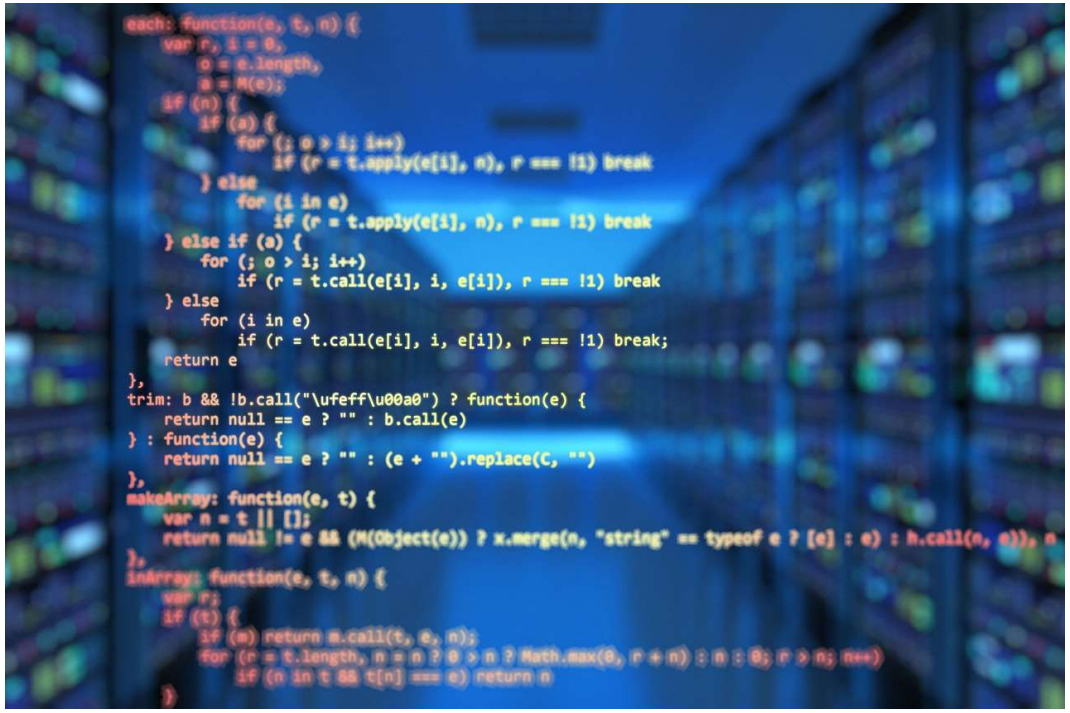


Pour chacune de ces dimensions, reproduisant fidèlement nos mécanismes cognitifs de conceptualisation, en plus de simplifier l'écriture des codes, il est important de faciliter la récupération de ces parties dans de nouveaux contextes et d'assurer la robustesse de ces parties aux changements survenus dans d'autres. Un code OO, idéalement, sera aussi simple à créer qu'à maintenir, récupérer et faire évoluer. Il n'est pas pertinent d'opposer le procédural à l'OO car, in fine, toute programmation des méthodes (c'est-à-dire la partie active des classes et des objets) reste totalement tributaire des mécanismes procéduraux.



- **Les grands acteurs de l'orienté objet**

Aujourd'hui, l'OO est omniprésent. Microsoft par exemple, a développé un nouveau langage informatique purement objet (C#). Il a très intensément contribué au développement d'un système d'informatique distribuée, basé sur des envois de messages d'ordinateur à ordinateur (les services web) et a proposé un nouveau langage d'interrogation des objets (LINQ), qui s'interface naturellement avec le monde relationnel et le monde XML.

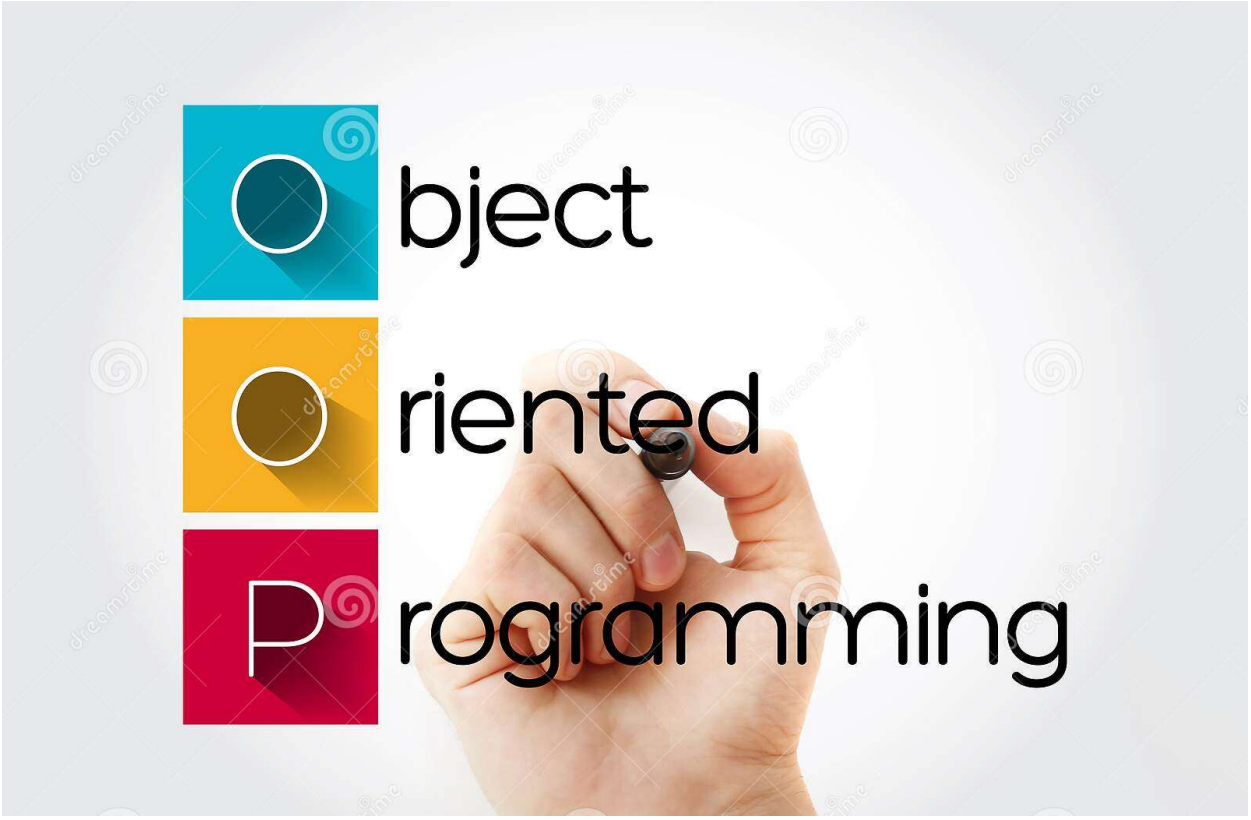


Sun a également créé RMI, Jini et sa propre version des services web, tous basés sur les technologies OO. Ces mêmes services web font l'objet de développements tout autant aboutis chez HP ou IBM. À la croisée de Java et du Web (originellement la raison, sinon du développement de Java, du moins de son succès), on découvre une importante panoplie d'outils de développement et de conception de sites web dynamiques.

De son côté, Apple encourage les développeurs à passer du langage Objective C, langage de prédilection jusqu'à présent dans l'univers Apple, au dernier né, Swift (tous deux langages OO), pour la conception d'applications censées s'exécuter tant sur les ordinateurs que les tablettes ou smartphones proposés par la célèbre marque à la pomme.

• B- Principes de base : quel objet pour l'informatique ?

- Introduction à la notion d'objet



Jetons un rapide coup d'œil par la fenêtre ; nous apercevons… des voitures, des passants, des arbres, un immeuble, un avion…

Cette simple perception est révélatrice d'un ensemble de mécanismes cognitifs des plus subtils, dont la compréhension est une porte d'entrée idéale dans le monde de l'informatique orientée objet.

- **Le trio <entité, attribut, valeur>**

Nous avons l'habitude de décrire le monde qui nous entoure à l'aide de ce trio que les informaticiens se plaisent à nommer <entité (ou objet), attribut, valeur> ; par exemple <voiture, couleur, rouge>, <voiture, marque, peugeot>, <passant, taille, grande>, <passant, âge, 50>. Les objets (la voiture et le passant) vous sautent aux yeux parce que chacun se caractérise par un ensemble d'attributs (couleur, âge, taille), prenant une valeur particulière, uniforme « sur tout l'objet».

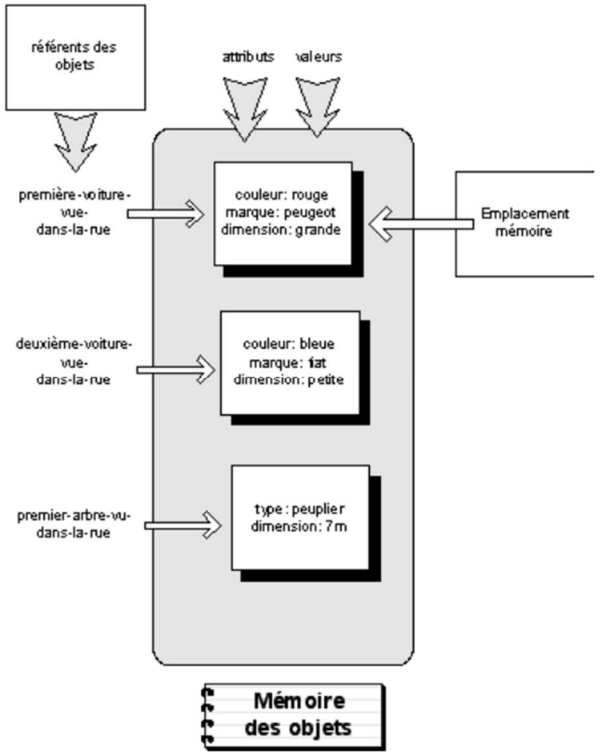
Que ce soit comme résultat de nos perceptions ou dans notre pratique langagière, les attributs et les objets jouent des rôles très différents. Les attributs structurent nos perceptions et ils servent, par exemple, sous forme d'adjectifs, à qualifier les noms qui les suivent ou les précèdent. La première conséquence de cette simple faculté de découpage cognitif sur l'informatique d'aujourd'hui est la suivante :

> **Objets, attributs, valeurs**
>
> Il est possible dans tous les langages informatiques de stocker et de manipuler des objets en mémoire, comme autant d'ensembles de couples attribut/valeur.

- **Stockage des objets en mémoire**

Dans la figure qui suit, nous voyons apparaître ces différents objets dans la mémoire de l'ordinateur.

Figure 1–1
Les objets informatiques
et leur stockage en mémoire



Dans notre exemple, les dimensions seraient typées en tant qu'entier ou réel. Tant la couleur que la marque pourraient se réduire à une valeur numérique (ce qui ramènerait l'attribut à un entier) choisie parmi un ensemble fini de valeurs possibles, indiquant chacune une couleur ou une marque.

Dès lors, le mécanisme informatique responsable du stockage de l'objet « saura », à la simple lecture structurelle de l'objet, quel est l'espace mémoire requis par son stockage.

La place de l'objet en mémoire

Les objets sont structurellement décrits par un premier ensemble d'attributs de type primitif, tels qu'entier, réel ou caractère, qui sert à déterminer précisément l'espace qu'ils occupent en mémoire.

- **Types primitifs**

À l'aide de ces types primitifs, le stockage en mémoire des objets se transforme comme reproduit sur la figure 1-2.

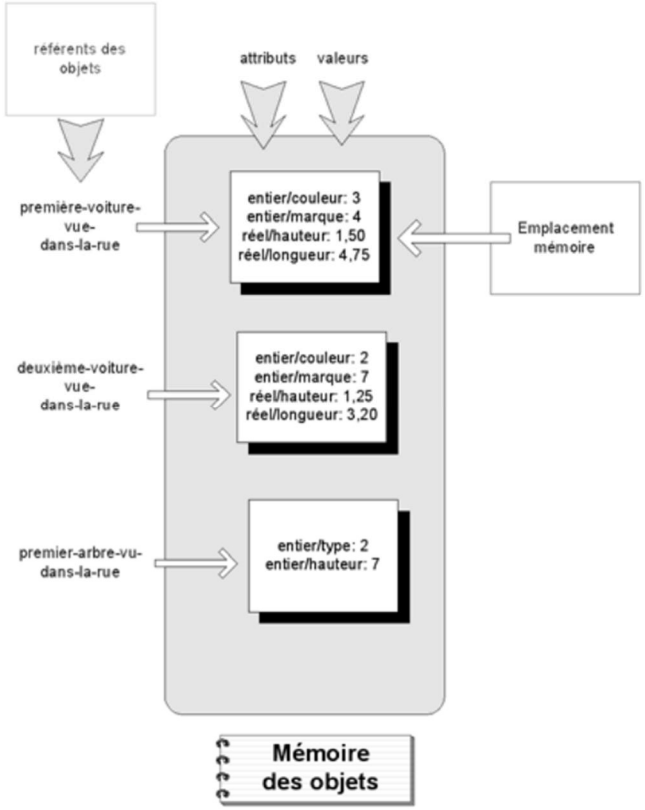


Figure 1–2
Les objets avec leur nouveau mode de stockage où chaque attribut est d'un type dit « primitif » ou « prédéfini », comme entier, réel, caractère…

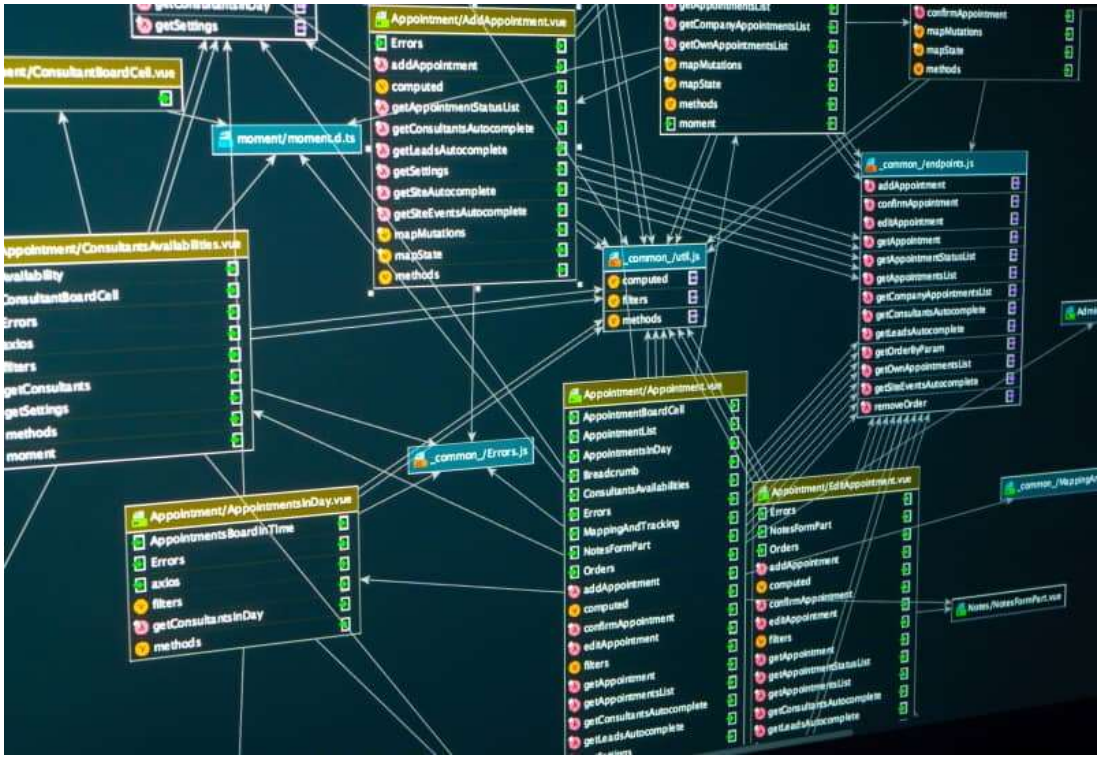
Ainsi, les voitures sont stockées dans des bases à l'aide de leurs couples attribut/valeur ; elles sont gérées par un concessionnaire automobile (comme montré à la figure 1-3).

Figure 1–3
Table d'une base de données relationnelle de voitures, avec quatre attributs et six enregistrements

Marque	Modele	Serie	Numero
Renault	18	RL	4698 SJ 45
Renault	Kangoo	RL	4568 HD 16
Renault	Kangoo	RL	6576 VE 38
Peugeot	106	KID	7845 ZS 83
Peugeot	309	chorus	7647 ABY 82
Ford	Escort	Match	8562 EV 23

Bases de données relationnelles

Il s'agit du mode de stockage des données sur support permanent le plus répandu en informatique. Les données sont stockées en tant qu'enregistrements dans des tables, par le biais d'un ensemble de couples attribut/valeur, dont une clé primaire essentielle à la singularisation de chaque enregistrement.

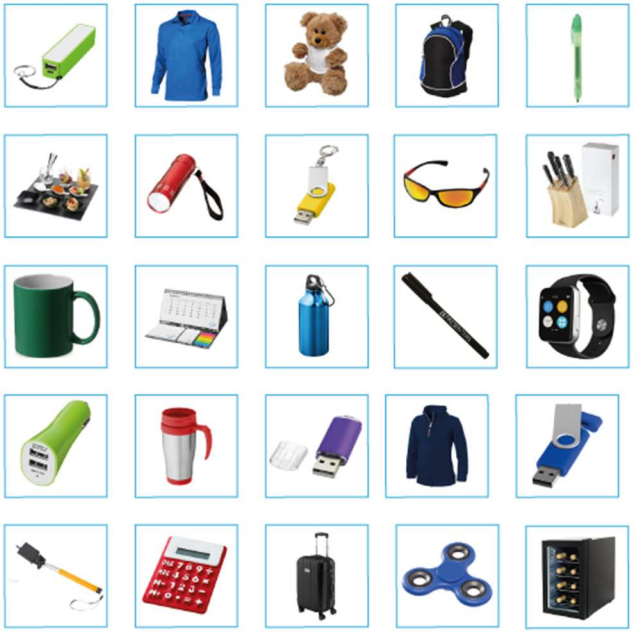


Les arbres, quant à eux, chacun également avec son couple attribut/valeur, s'enregistrent dans des bases de données gérées par un botaniste. Notez d'ores et déjà que, bien qu'il s'agisse de trois objets différents, ils peuvent être repris sous la forme de deux classes : « Voiture » comprend deux objets et « Arbre » un seul.

s

Nous la retrouvons dans pratiquement tous les langages informatiques.

- **Le référent d'un objet**



Observons à nouveau les figures 1-1 et 1-2. Chaque objet est nommé et ce nom doit être son unique identifiant. Comme c'est en le nommant que nous accédons à l'objet, il est clair que ce nom ne peut être partagé par plusieurs objets. En informatique, le nom correspondra de manière univoque à l'adresse physique de l'objet en mémoire.

En général, dans la plupart des ordinateurs aujourd'hui, l'adresse mémoire se compose de 64 bits, ce qui permet de stocker jusqu'à 264 informations différentes.

Espace mémoire

Ces dernières années, de plus en plus de processeurs ont fait le choix d'une architecture à 64 bits au lieu de 32, ce qui implique notamment une révision profonde de tous les mécanismes d'adressage dans les systèmes d'exploitation. Depuis, les informaticiens peuvent se sentir à l'aise face à l'immensité de l'espace d'adressage qui s'ouvre à eux : 264, soit 18 446 744 073 709 551 616 octets.

Référent vers un objet unique

Le nom d'un objet informatique, ce qui le rend unique, est également ce qui permet d'y accéder physiquement. Nous appellerons ce nom le « référent de l'objet ». L'information reçue et contenue par ce référent n'est rien d'autre que l'adresse mémoire où cet objet se trouve stocké.

Un référent est une variable informatique particulière, associée à un nom symbolique, codée sur 64 bits et contenant l'adresse physique d'un objet informatique.

- **Plusieurs référents pour un même objet**

Un même objet peut-il porter plusieurs noms ? Plusieurs référents, qui contiennent tous la même adresse physique, peuvent-ils désigner en mémoire un même objet ? Oui, s'il est nécessaire de nommer, donc d'accéder à l'objet, dans des contextes différents et qui s'ignorent mutuellement.

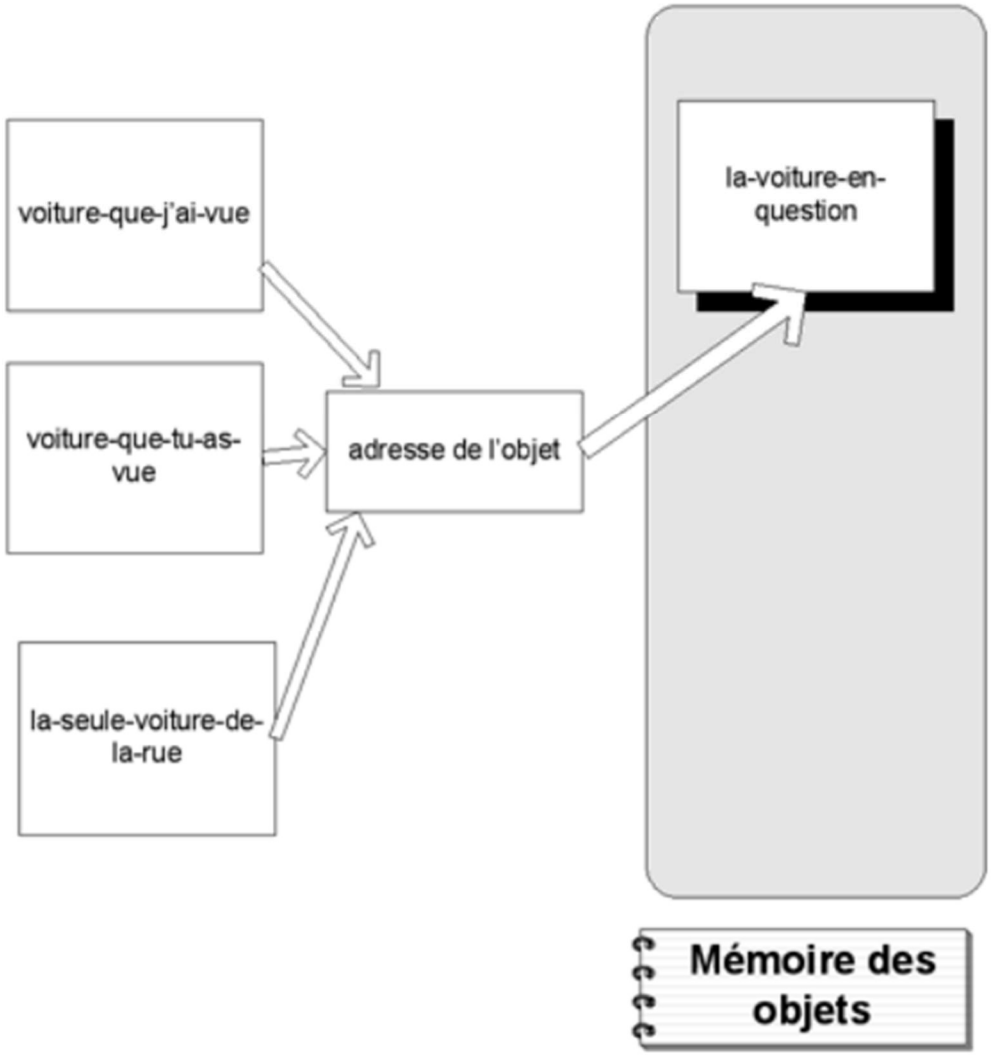
Adressage indirect

C'est la possibilité pour une variable, non pas d'être associée directement à une donnée, mais plutôt à une adresse physique d'un emplacement contenant, lui, cette donnée. Il devient possible de différer le choix de cette adresse pendant l'exécution du programme, tout en utilisant naturellement la variable.

Et plusieurs de ces variables peuvent alors pointer vers un même emplacement car partageant la même adresse. Une telle variable, dont la valeur est une adresse, est dénommée un pointeur en C et C++.

Mais est-ce qu'un attribut d'un objet peut servir de référent vers un autre objet ? Oui bien sûr ! Et il y a là un mécanisme idéal pour que ces deux objets communiquent.

Figure 1–4
Plusieurs référents désignent un même objet grâce au mécanisme informatique d'adressage indirect.



Alors, plusieurs référents pour un même objet ?

Lorsqu'on écrit un programme orienté objet, on accède couramment à un même objet par plusieurs référents, créés dans différents contextes d'utilisation. Cette multiplication des référents est un élément déterminant de la gestion mémoire associée à l'objet.

On acceptera à ce stade-ci qu'il est utile qu'un objet séjourne en mémoire tant qu'il est possible de le référer. Sans référent, un objet est inaccessible. Le jour où vous n'êtes même plus un numéro dans aucune base de données, vous êtes mort (socialement, tout au moins).

En C++, le programmeur peut effacer un objet à partir de n'importe lequel de ses référents (par exemple « delete voiture-quej'ai-vue »). On vous laisse dès lors deviner vers quoi pointeront les autres référents (taxés de « pointeurs fous »), pour que vous preniez conscience d'une des difficultés majeures et une des sources d'erreur les plus cruelles (car ses effets sont imprévisibles) inhérentes à la programmation dans ce vénérable langage.

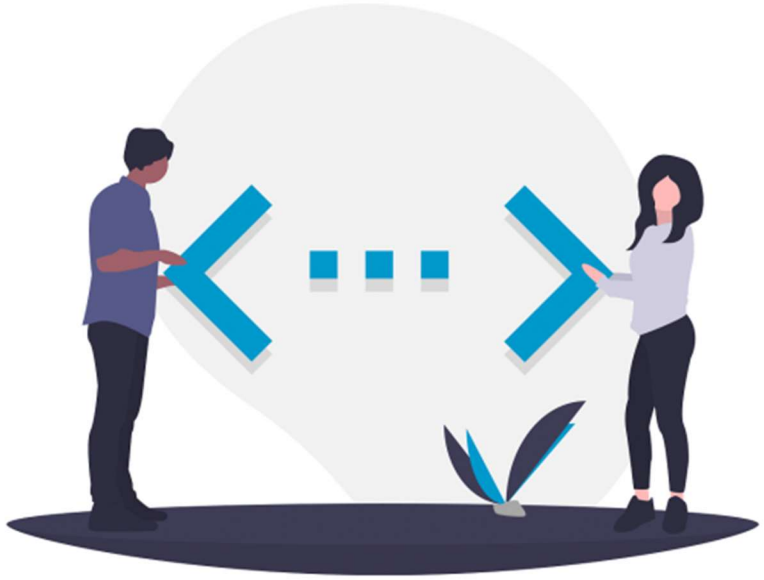
- **L’objet dans sa version passive**

L’objet et ses constituants

Voyons plus précisément ce qui amène notre perception à privilégier certains objets plutôt que d’autres. Certains d’entre eux se révèlent être une composition subtile d’autres objets, tout aussi présents que les premiers, mais que, pourtant, il ne vous est pas venu à l’idée de citer lors de notre première démonstration.

Première distinction : ce qui est utile à soi et ce qui l'est aux autres

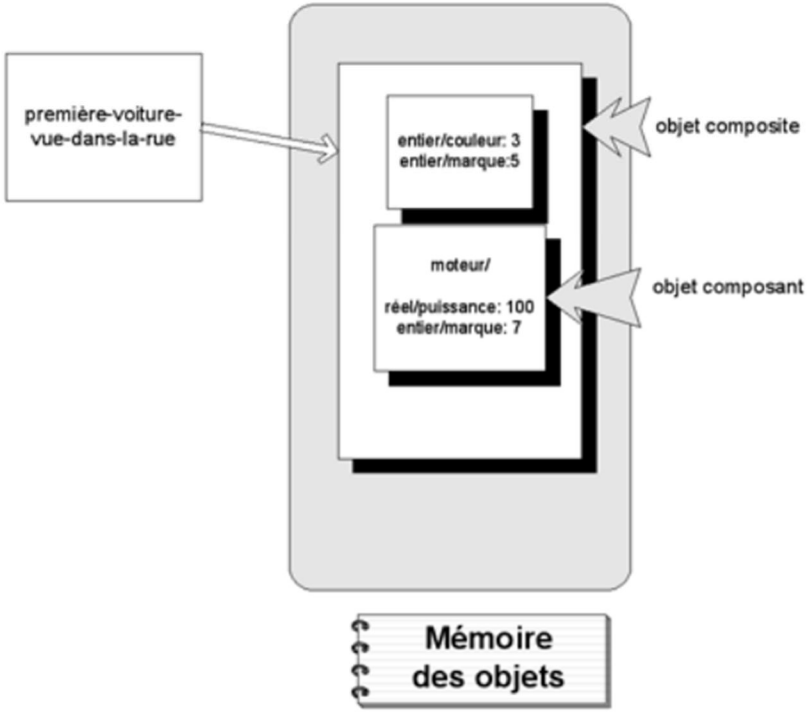
L'orienté objet, pour des raisons pratiques que nous évoquerons par la suite, encourage à séparer, dans la description de tout objet, la partie utile pour tous les autres objets qui y recourront, de la partie nécessaire à son fonctionnement propre. Il faut séparer physiquement ce que les autres objets doivent savoir d'un objet donné, afin de solliciter ses services, de ce que ce dernier requiert pour son fonctionnement, c'est-à-dire la mise en œuvre de ces mêmes services.



Objet composite

Tant que vous n'aurez pas besoin du garagiste, vous ne vous préoccuperez pas des roues ni du moteur de votre objet « voiture ». Que les objets s'organisent entre eux en composite et composants est une donnée de notre réalité que les informaticiens ont jugé important de reproduire.

Figure 1–5
L'objet moteur devient un composant de l'objet voiture.



Une composition d'objets

Entre eux, les objets peuvent entrer dans une relation de type composition, où certains se trouvent contenus dans d'autres et ne sont accessibles qu'à partir de ces autres. Leur existence dépend entièrement de celle des objets qui les contiennent.

Dépendance sans composition

Vous comprendrez aisément que ce type de relation entre objets ne suffit pas pour décrire fidèlement la réalité qui nous entoure. En effet, si la voiture possède bien un moteur, elle peut également contenir des passagers, qui n'aimeraient pas être portés disparus lors de la mise à la casse de la voiture...

- **L'objet dans sa version active**

Activité des objets

Afin de poursuivre cette petite introspection cognitive dans le monde de l'informatique orientée voiture s'arrête à un feu rouge », « Les passants traversent la route », « Un passant entre dans un magasin », « Un oiseau s'envole de l'arbre ». Que dire de toutes ces observations bouleversantes que vous venez d'énoncer ?

Les différents états d'un objet

Les objets changent donc d'état, continûment, mais tout en préservant leur identité, en restant ces mêmes objets qu'ils ont toujours été. Les objets sont dynamiques, la valeur de leurs attributs change dans le temps, soit par des mécanismes qui leur sont propres (tel le changement des feux de signalisation), soit en raison d'une interaction avec un autre objet (comme dans le cas de la voiture qui s'arrête au feu rouge).

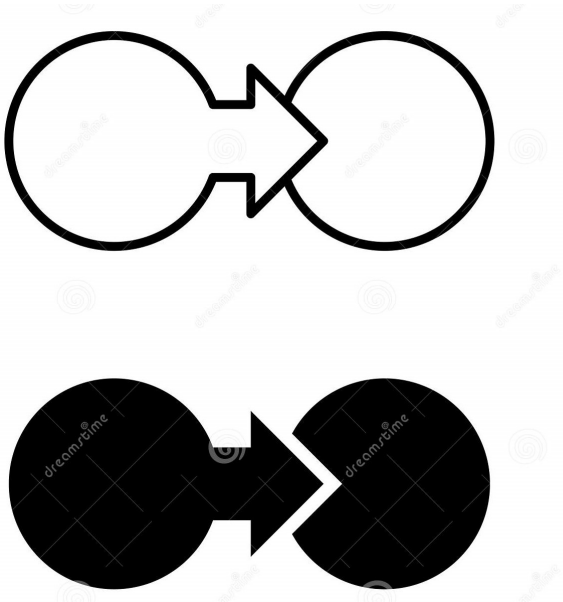
Du point de vue informatique, rien n'est plus simple que de modifier la valeur d'un attribut. Il suffit de se rendre dans la zone mémoire occupée par cet attribut et de remplacer la valeur qui s'y trouve actuellement stockée par une nouvelle valeur. La mise à jour d'une partie de sa mémoire, par l'exécution d'une instruction appropriée, est une des opérations les plus fréquemment effectuées par un ordinateur.

Changement d'états

Le cycle de vie d'un objet, lors de l'exécution d'un programme orienté objet, se limite à une succession de changements d'états, jusqu'à sa disparition pure et simple de la mémoire centrale.

Les changements d'état : qui en est la cause ?

Qui est donc responsable des changements de valeur des attributs ? Qui a la charge de rendre les objets moins inertes qu'ils n'apparaissent à première vue ? Qui se charge de les faire évoluer et, ce faisant, de les rendre un tant soit peu attrayants ?



Reprenons l'exemple des feux de signalisation et des voitures évoqué plus haut et, comme indiqué à la figure 1-6, stockons-en un de chaque sorte dans la mémoire de l'ordinateur (nous supposerons que la couleur est bien représentée par un entier ne prenant que les valeurs 1, 2 ou 3, et la vitesse par un simple entier). Installons dans cette même mémoire, mais un peu plus loin, deux opérations chargées de changer l'une la couleur du feu et l'autre la vitesse de la voiture.

Dans la mémoire dite centrale, RAM ou vive, d'un ordinateur, ne se trouvent toujours installés que ces deux types d'information : des données et des instructions qui utilisent et modifient ces données, rien d'autre.

Comme les attributs, chaque opération se doit d'être nommée afin de pouvoir y accéder ; nos deux opérations s'appelleront respectivement change et changeVitesse (int nV).

Ces deux opérations sont banales : pour le feu, il s'agit d'incrémenter l'entier couleur et de ramener sa valeur à 1 dès qu'il atteint 4 (le feu reproduit constamment le même cycle), et pour la voiture de changer sa vitesse en fonction de l'argument transmis en paramètre (l'entier nV, pour autant qu'il ne dépasse pas la limite autorisée). Ces deux opérations triviales mettront un peu d'animation dans la rue.

Comment relier les opérations et les attributs ?

Alors que nous comprenons bien l'installation en mémoire, tant de l'objet que de l'opération qui pourra le modifier, ce qui nous apparaît moins évident, c'est la liaison entre les deux. Ainsi, pour le feu, comment l'opération et les deux instructions de changement (l'incrémentation et le test), installées dans la mémoire à droite, savent-elles qu'elles portent sur le feu de signalisation installé, lui, dans la mémoire à gauche ?

Plus concrètement encore, comment l'opération change, qui incrémente et teste un entier, sait-elle que cet entier est, de fait, celui qui code la couleur du feu et non « l'âge du capitaine » ? De même, comment l'opération changeVitesse sait-elle que c'est bien sur la vitesse de la voiture qu'elle doit porter, et non pas sur un feu, qu'elle essaierait de pousser à 120. La réponse à cette question vous fait entrer de plain-pied dans le monde de l'orienté objet…

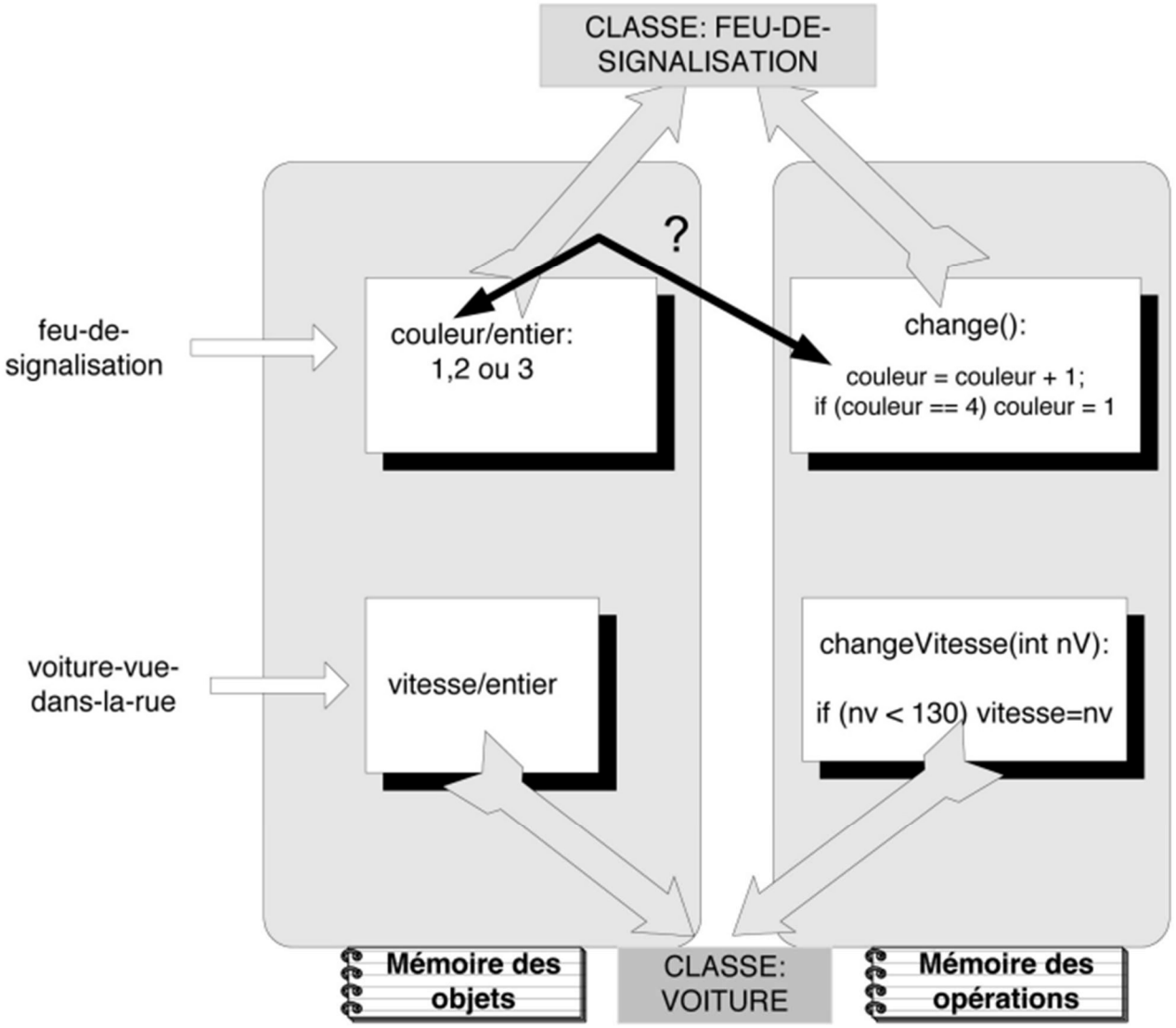


Figure 1–6 Le changement d'état du feu de signalisation par l'entremise de l'opération « change » et celui de la voiture par l'opération « changeVitesse(int nV) »

- **Introduction à la notion de classe**

Méthodes et classes

Place à la réponse. Elle s'articule en deux temps. Concentrons-nous uniquement sur le feu de signalisation (pour la voiture, le raisonnement est en tout point analogue). Dans un premier temps, il faudra que l'opération change – que nous appellerons dorénavant méthode change – ne soit attachée qu'à des objets de type feu de signalisation. Seuls ces objets possèdent l'entier sur lequel peut s'exercer cette méthode.

Appliquer la méthode change sur tout autre type d'objet, tel que la voiture ou l'arbre, n'a pas de sens, car on ne saurait de quel entier il s'agit. De surcroît, ce double incrément pour revenir à la valeur de base est totalement dénué de signification pour ces autres objets. Le subterfuge qui associe la méthode avec l'objet qui lui correspond consiste à les unir tous deux par les liens, non pas du mariage, mais de la « classe ». Deux classes sont à l'œuvre figure 1-6 : le Feu-De-Signalisation et la Voiture.

Classe

Une nouvelle structure de données voit le jour en OO : la classe, qui, de fait, a pour principal rôle d'unir en son sein tous les attributs de l'objet et toutes les opérations les concernant. Ces opérations sont appelées méthodes et regroupent un ensemble d'instructions portant sur les attributs de l'objet.

Comme c'est l'usage en informatique, s'agissant de variables manipulées, on parlera dorénavant de l'objet comme d'une instance de sa classe, et de la classe comme du type de cet objet.

On voit l'intérêt, bien sûr, de garder une définition de la classe séparée mais partagée par toutes les instances de celles-ci. Non seulement c'est la classe qui détermine les attributs (leur type) sur lesquels les méthodes peuvent opérer mais, plus encore, seules les méthodes déclarées dans la classe pourront de facto manipuler les attributs des objets typés par cette classe.

La classe Feu-de-signalisation pourrait être définie plus ou moins comme suit :

```
class Feu-de-signalisation {
  int couleur ;
  change() {
    couleur = couleur + 1 ;
    if (couleur ==4) couleur = 1 ;
  }
}
```

Un objet existe par l'entremise de ses attributs et se modifie uniquement par l'entremise de ses méthodes (et nous disons bien « ses »).

Différencier langage orienté objet et langage manipulant des objets

De nombreux langages de programmation, surtout de scripts pour le développement web (JavaScript, VB Script), rendent possible l'exécution de méthodes sur des objets dont les classes préexistent au développement. Le programmeur ne crée jamais de nouvelles classes mais se contente d'exécuter les méthodes de celles-ci sur des objets. Supposons par exemple que vous vouliez agrandir une police (*font*) particulière de votre page web. Vous écrirez `f.setSize(16)` mais jamais dans votre code vous n'aurez créé la classe `Font` (vous utilisez l'objet `f` issu de cette classe) ni sa méthode `setSize()`. Vous vous limitez à les utiliser comme vous utilisez les bibliothèques d'un quelconque langage de programmation. Les classes regroupées dans ces bibliothèques auront été développées par d'autres programmeurs et mises à votre disposition.
La programmation orientée objet débute dès que vous incombe la création de nouvelles classes, même si lors de l'écriture de vos codes, vous vous reposez largement sur des classes écrites par autrui. C'est bien de programmation orientée classe qu'il s'agirait plutôt.
Pour certains, dès lors, et au contraire de ce que nous venons d'affirmer, un langage comme Javascript en deviendrait, lui, vraiment orienté objet. Ce n'est pas tant qu'il soit impossible d'écrire de nouvelles classes, mais plutôt que tous les objets soient en même temps des classes. En effet, il est possible en Javascript de créer directement un objet à partir d'un autre, suivant un mécanisme ressemblant à l'héritage et généralement appelé prototypage. C'est également le modèle des langages Io et Self, largement reconnus comme orientés objet. Déterminer quels langages peuvent effectivement se targuer d'être orientés objet est devenu une question plutôt délicate à trancher.

- **Concept de généralisation et spécialisation de classes**

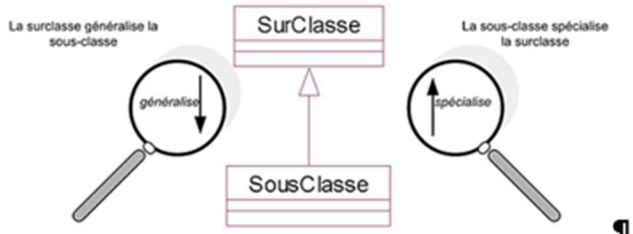
La généralisation de classes consiste à factoriser dans une classe, appelée superclasse, les attributs et/ou opérations des classes considérées. Appliquée à l'ensemble des classes, elle permet de réaliser une hiérarchie des classes.

La spécialisation représente la démarche inverse de la généralisation puisqu'elle consiste à créer à partir d'une classe, plusieurs classes spécialisées. Chaque nouvelle classe créée est dite spécialisée puisqu'elle comporte en plus des attributs ou opérations de la super-classe (disponibles par héritage) des attributs ou opérations qui lui sont propres.

Une classe spécialisée porte aussi le nom de sous-classe. La spécialisation de classe se construit en deux temps : d'abord par héritage des opérations et des attributs d'une super-classe et ensuite par ajout d'opérations et/ou d'attributs spécifiques à la sous-classe.

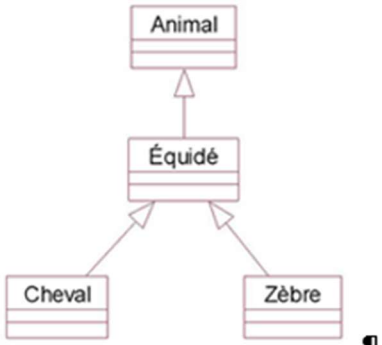
La généralisation-spécialisation est un des mécanismes les plus importants de l'approche objet qui facilite la réutilisation des classes.

Exemple¶



¶

Le·cheval·est·une·spécialisation·de·l'équidé,·elle-même·spécialisation·de·l'animal.·Le·zèbre·est·une·autre·spécialisation·de·l'équidé.¶



¶

¶

- **Concept de polymorphisme**

Le polymorphisme est la capacité donnée à une même opération de s'exécuter différemment suivant le contexte de la classe où elle se trouve. Ainsi une opération définie dans une super-classe peut s'exécuter de manière différente selon la sous-classe où elle est héritée.

En fait lors de l'exécution, l'appel de l'opération va automatiquement déclencher l'exécution de l'opération de la sous-classe concernée. Dans le déroulement de l'exécution de l'opération de la sous-classe, il est possible de faire appel à l'opération de la super-classe qui contient en général la partie commune applicable aux deux sous-classes.

- **Héritage bien reçu**

C'est avec ce mécanisme d'héritage que nous terminons notre entrée dans le monde de l'OO, muni, comme vous vous en serez rendu compte, de notre petit manuel de psychologie. En effet, les sciences cognitives et l'intelligence artificielle prennent une large place dans le faire-part de naissance de l'informatique OO.

Dans les sciences cognitives, cette idée d'objet est largement répandue, déguisée sous les traits des schémas piagétiens, des noumènes kantiens ou des paradigmes kuhniens. Tous ces auteurs se sont efforcés de nous rappeler que notre connaissance n'est pas aussi désorganisée qu'elle n'y paraît, que des blocs apparaissent, faisant de notre cognition un cheptel d'îles plutôt qu'un océan uniforme, blocs reliés entre eux de manière relationnelle et taxonomique.

Cette structuration cognitive reflète, en partie, la réalité qui nous entoure, mais surtout notre manière de la percevoir et de la communiquer, tout en se soumettant à des principes universaux d'économie, de simplicité et de stabilité.

Exemple : La classe Employé et ses deux sous-classes Cadre et NonCadre.

- Nom de classe : Employé,
  - Attributs :
    - – numéro,
    - – nom,
    - – salaire de base.
  - Opérations : calculSalaire( ).
- Nom de la sous-classe : Cadre.
  - Attributs : niveau d'encadrement,
  - Opérations : calculSalaire( ).
- Nom de la sous-classe : NonCadre.
  - Attributs : niveau de spécialisation,
  - Opérations : calculSalaire( ).

Le principe de calcul du salaire étant de calculer pour chaque type d'employé une prime spécifique en fonction soit du niveau d'encadrement, soit du niveau de spécialisation selon le type d'employé. Voyons maintenant comment se réalise l'application du polymorphisme lors de l'exécution des opérations. Dans cet exemple, lorsque l'on appelle l'opération calculSalaire( ), c'est l'opération de sous-classe Cadre ou celle de la sous-classe NonCadre qui est en fait activée selon l'objet concerné. L'opération de la sous-classe fait en général appel explicitement à l'opération calculSalaire( ) de la super-classe pour bénéficier des traitements communs aux cadres et non cadres et ensuite il y aura poursuite du traitement spécifique à la sous-classe.

- **Avantage du langage objet**

Deux avantages prépondérants sont mis en général en avant lorsque l'on choisit une approche objet :

• La modularité – Par construction, étant donné que l'on conçoit des classes représentant une entité de taille limitée en données et en opérations, il est plus aisé de construire des systèmes modulables que si l'on élabore une seule base de données d'une part et un seul logiciel d'autre part.

• La réutilisabilité – La définition d'un système à l'aide de classe ayant chacune la responsabilité d'un sous-ensemble de données et des opérations associées favorise fortement la potentialité de trouver des classes réutilisables. La réutilisation de classe se réalise soit sur le plan métier à l'intérieur d'une même entreprise dans des applications différentes, soit sur le plan technique à l'échelle de tous les développements réalisés à l'aide d'un même langage. Sur ce dernier aspect, c'est toute l'approche du développement par composant qui est en jeu.

Au-delà de ces deux avantages majeurs et compte tenu de la plus grande modularité dans la construction d'une application à l'aide d'objets, la maintenance élémentaire de chaque classe est en soi plus simple à réaliser que celle d'un logiciel unique traitant toutes les données d'un système. Il importe bien entendu dans l'approche objet de construire son système en veillant à minimiser le nombre de relations entre classes.

