# LANGAGES ORIENTÉS OBJETS

# SMALLTALK

***"Le savoir consiste à constituer des collections de singularités évocatrices [] Les collections les plus efficaces ne sont pas faites de réalités, mais d'emblèmes. Qui possède l'emblème agit sur la réalité. Le symbole tient lieu du réel. On se préoccupe donc des réalités et des faits, non pour remarquer des séquences et des variations quantitatives, mais pour posséder et tenir à disposition des rubriques emblématiques et des tables de récurrences constituées en songeant uniquement aux interdépendances de symboles."***

Marcel Granet
"La pensée chinoise" p.274

## Les langages orientés objets

Un *langage orienté objet* est un langage formel "parlé" dans un monde rempli d'*objets* , où ces derniers dialoguent par envois de messages. Un objet est constitué d'une enveloppe et d'un noyau; son enveloppe est son seul interface avec le reste du monde; toute transaction avec l'extérieur doit passer à travers ce filtre protecteur.

### L'instanciation

Un objet vient au monde par *instanciation* , il est généré par une ou plusieurs *classes* qui lui déterminent en quelque sorte son potentiel génétique.

Assistons à la naissance d'une nouvelle voiture baptisée "Totoche":

```
Car new Totoche.
```

Nous (le clavier: qui est l'objet qui nous représente) venons d'envoyer le message "new Totoche" à la classe des voitures "Car".
L'accouchement s'est bien passé, Totoche existe et toutes les propriétés "génétiques" d'une voiture qui fait qu'on ne peut la confondre avec un oiseau ou même une camionnette, lui sont données. En ce sens Totoche, instance de Car, fait bien partie de la classe des voitures.

Comment peut-on dialoguer avec Totoche? Oui, dialoguer avec elle, non pas lui demander si elle vous trouve beau ou si elle est sexy, mais par exemple s'il lui reste de l'essence ou si les plaquettes de frein sont usées. Avant d'essayer d'entamer le dialogue, regardons le mode de raisonnement qu'un objet quelconque sera capable de tenir lorsqu'on lui posera une question:

Soit deux objets X et Y, installés confortablement au fond d'un bar avec des amis, en pleine conversation à essayer de refaire leur monde. A un instant donné X pose une question à Y, quel peut être le comportement de Y:

Y comprend la question posée par X:
- a: Quelle méthode va-t-il prendre pour répondre à X?
- b: Où va-t-il trouver les informations nécessaires au raisonnement qu'il va tenir?
- c: Comment va-t-il appliquer ce raisonnement?
- d: A qui, et comment va-t-il transmettre la réponse?
- e: Que va-t-il faire après?

Y ne comprend pas la question posée par X:
- a: va-t-il poser la question à quelqu'un d'autre discrètement, si oui à qui?
- b: ou va-t-il dire tout simplement qu'il ne sait pas répondre, et à qui va-t-il le dire?
- c: Que va-t-il faire après?

Un langage orienté objet met en oeuvre ce type de mécanisme à chaque question posée par un objet à un autre. Les réponses aux questions posées lors de l'analyse du comportement de Y dépendent du langage orienté objet utilisé.
A chaque objet nous pouvons associer un environnement, c'est dans ce dernier que l'objet fera la recherche de méthode pour répondre aux questions posées.

Chaque objet sait de quelle race (classe, type) il est, car il connaît son ou ses géniteurs. Totoche sait qu'elle est une voiture, et de ce fait sait répondre à des questions telles que "Totoche avance de 100 mètres.", mais ne comprend pas: "Totoche décolle et vole à l'altitude de 1000 pieds.".

## Les classes et métaclasses

Les classes, qui sont des types génériques d'objets, sont un lieu privilégié de rassemblement d'informations communes à toutes leurs instances; ces dernières pour répondre à des questions vont devoir faire appel aux informations contenues dans leur(s) classe(s), ce qui nous oblige à considérer les classes comme des objets pour pouvoir répondre aux questions posées par leurs instances. Quels sont les géniteurs des classes?
Y a-t-il des classes de classe: des métaclasses en quelque sorte? Pour la beauté du raisonnement il en faut, mais alors la question se repose pour les classes des métaclasses. En fait, suivant les langages objets cette chaîne de classes existe ou non.

La hiérarchie des classes
La classification des objets étant une activité relativement répandue chez l'homme il pourrait être relativement utile de doter cet univers d'objets des possibilités de hiérarchie de classes. Chaque classe pourrait, alors, hériter des propriétés de sa ou ses surclasses. La même question que pour les classes de classe se pose pour les surclasses des surclasses d'une classe. Nous voici en face de deux récursions qui sont a priori infinies. Si nous voulons être capable de simuler un tel monde d'objets par une machine, nécessairement finie, il nous faut fermer ce monde tout en essayant de conserver sa cohérence, cet exploit est réalisé dans le premier des langage objets digne de ce nom: Smalltalk-80, chaque objet est

instance d'une seule classe, qui elle-même est instance d'une seule métaclasse; toutes les métaclasses étant les instances d'une unique classe: "METACLASSE" qui est une instance de la classe: "METACLASSE CLASSE" qui en tant que métaclasse est une instance de la classe "METACLASSE", Ouf! la boucle est bouclée, et l'univers est fermé, la genèse d'un monde d'objets n'est pas simple!

Que se passe-t-il au niveau de la chaîne des surclasses, c'est à dire de l'héritage des propriétés? Une métaclasse ayant toutes les capacités génériques d'une classe il semblerait normal que les métaclasses héritent des propriétés communes aux classes qu'elles génèrent, mais comment peut-on hériter d'enfants qui ne sont pas encore nés? Créons une chaîne de classes de telle manière que toutes les classes héritent, directement ou non, d'une classe unique: "CLASSE DESCRIPTION", y compris la métaclasse: "METACLASSE CLASSE". Pour que cet univers soit un véritable univers d'objets au sens de la classification il faut, directement ou non, que tout objet, y compris "CLASSE", hérite des propriétés de la classe sommet du monde: "OBJET". Nous pouvons dire alors que toute entité du langage est un objet car il hérite, quelque soit la façon dont il a été crée des propriétés d'une instance de la classe: "OBJET", qui est sa propre surclasse.

Le Dieu de cet univers créa la classe "OBJET", la classe "CLASSE" et la subtile liaison d'instanciation entre "METACLASSE" et "METACLASSE CLASSE"; c'est-à-dire des conditions de vie cohérentes pour ce microcosme.

## La programmation en SmallTalk-80

Contrairement aux langages LISP et PROLOG, nous devons constater que les langages orientés objets ne découlent pas directement d'une belle présentation mathématique mais plutôt d'une présentation philosophique d'un monde imaginaire. En ce sens les LOOs forment une classe différente des langages fonctionnels et relationnels.

Dans le reste de ce chapitre nous ne nous intéresserons qu'au langage SMALLTALK-80 qui nous servira de support à la présentation de la programmation en LOO.

Reprenons l'exemple de la voiture Totoche:

La classe des voitures "Car" est supposée déjà créée et posséde par héritage de la classe "CLASSE" la méthode de création d'instances "new". A la création de "Car" il a, par exemple, été précisé que ses futures instances auraient 3 variables d'instances: "réservoir", "compteur", "frein" qui correspondent respectivement à la quantité d'essence restant dans le réservoir, le nombre de kilomètres effectués depuis sa naissance et l'état d'usure des freins.

L'envoi du message "new Totoche" à "Car" est évalué de la façon suivante:

"Car" essaie de comprendre le sélecteur de message "new", pour ce faire regarde dans son dictionnaire de sélecteurs de messages qu'elle comprend directement et ne trouve pas "new"; elle fait alors appel à sa surclasse pour qu'elle fasse cette même recherche, et ainsi de suite jusqu'à la classe "CLASSE" qui comprend le sélecteur "new" et trouve dans son dictionnaire la méthode associée (le code

exécutable) de création d'instance qu'elle fait parvenir à la classe "Car".
Cette première partie d'évaluation porte le nom: recherche de méthode.

La méthode "new" crée un nouvel objet de classe "Car", de nom "Totoche", possédant 3 variables de noms: "réservoir", "compteur", et "frein", dont les valeurs ne sont pas encore définies.

Pour initialiser les variables d'instance à la création il faut une méthode "new" définie de façon locale à "Car"; c'est à partir de ce moment qu'il va falloir programmer: envoyer d'autres messages.

Création de la méthode locale "new" de "Car":

```
new Arg          Déclaration de la méthode new a un argument
        ||            qui ne nécessite pas de variable locale

super new Arg. Envoi du message new Arg a la
               superclasse de "Car"

Arg reservoir 30.   Initialisation du réservoir a 30 litres

Arg compteur 0.     Initialisation du compteur a 0 kilomètre

Arg frein "OK".     Initialisation des freins

^ Arg.           La valeur retournée du message
                 est le nom de la nouvelle voiture.
```

Super est un pseudo objet qui permet de commencer la recherche de méthode à partir de la surclasse de la classe du receveur du message. Ceci permet de redéfinir simplement des noms de sélecteurs de message. Self est un autre pseudo objet qui permet d'envoyer des messages au receveur du message qui est en cours de définition:

si au lieu de créer une méthode "new" spéciale à "Car" nous avions choisi de créer une méthode d'initialisation d'une voiture déjà créée "init":

```
init             Déclaration de la méthode init sans argument
        ||            qui ne nécessite pas de variable locale

self reservoir 30.  Initialisation du réservoir a 30 litres

self compteur 0.    Initialisation du compteur a 0 kilomètre

self frein "OK".    Initialisation des freins

^ Arg.           La valeur retournée du message est
                 le nom de la nouvelle voiture.
```

la création et l'initialisation de Totoche se feraient en deux temps:

```
Car new Totoche.
Totoche init.
```

ou plus simplement (car new Totoche) init. la méthode "new" de "CLASSE" renvoyant le nom de l'objet créé.

Programmer en LOO revient à créer un monde d'objets sur lesquels nous voulons agir et définir des méthodes qui leurs permettront de dialoguer entre eux. Ce type de programmation est manifestement bien adapté à la simulation de comportement.

Pour finir cette introduction extrêmement brève aux LOOs donnons comme exemple de fonction bien connue: la factorielle.

```
!               Déclaration de la méthode !
        ||           sans variable locale

(Self < 1)      ifTrue[^ 1.].

        ifFalse[^ Self*((Self - 1)!).].
```

**Sommet du monde SMALLTALK 80 un sujet de réflexion en soi:**