POO
Approche Objet

SOMMAIRE

Edition janvier 2019

L'APPROCHE OBJET

0-1 - Les 5 paradigmes de programmation (d'après Stroustrup, créateur du C++)

Paradigme 0 : pdg. du débutant : « Tout dans le main »

Codez tout dans le programme principal.

Débrouillez-vous comme vous pouvez.

Faîtes des tests jusqu'à ce que ça marche !

Ce paradigme est celui du débutant. C'est ce qu'on fait quand on commence la programmation.

Paradigme 1 : pdg. de la programmation procédurale

Choisissez les procédures (=fonctions).

Utiliser les meilleurs algorithmes que vous pourrez trouver.

Le problème est celui du bon découpage du main en procédures.

Il faut définir les entrées et les sorties pour chaque procédure.

Un principe général est de distinguer entre l'interface utilisateur (la saisie et l'affichage) et le calcul.

Paradigme 2 : pdg. de la programmation modulaire : masquer l'information

Choisissez vos modules (fichiers avec des fonctions et regroupements de fichiers)

Découpez le programme de telle sorte

que les données soient masquées par les modules

Le paradigme de la programmation modulaire est un **aboutissement de la programmation procédurale**.

Il consiste à regrouper les fonctions dans des fichiers et à **organiser ces fichiers en modules** (un module est un regroupement de fichiers) qui permettent de **masquer les données** du programme.

Techniquement, il conduit à l'utilisation de variables globales, de static, d'extern, de compilation conditionnelle, d'espace des noms, d'organisation du type : outils.c, outils.h (interface), utilisation.c.

Les parties « masquage de l'information » et « regroupement des fonctions dans un fichier » du paradigme est rendue obsolète par l'usage de la programmation objet.

La partie « organisation des fichiers » reste présente.

Paradigme 3 : pdg. de l'abstraction des données : type structurés = classe

Choisissez les types dont vous avez besoin.

Fournissez un ensemble complet d'opérations pour chaque type.

Une classe est un type, en général structuré, auquel on ajoute des procédures appelées méthodes.

Une classe en tant que type qui peut donner lieu à la fabrication d'une variable appelée « objet ».

Fabriquer un type structuré consiste à regrouper des types (simples ou déjà structurés) dans un même type structuré : c'est déjà un mécanisme d'abstraction.

Principe d'encapsulation

Le principe d'encapsulation est le principe de fonctionnement du paradigme d'abstraction.

Principe d'encapsulation :

- Les méthodes sont l'interface obligatoire d'accès aux données d'un objet.

L'encapsulation consiste à :

- Cacher l'état de l'objet.
- On ne peut dès lors alors accéder à cet état, pour le consulter ou le modifier, que en utilisant les méthodes de l'objet.

Y sont associées les notions de : **visibilité** des attributs et des méthodes, **constructeur**, **destructeur**, **surcharge**.

Paradigme 4 : pdg. de l'héritage

Choisissez vos classes.

Fournissez un ensemble complet d'opérations pour chaque classe.

Rendez toute similitude explicite à l'aide de l'héritage.

Principe de substitution

Le principe de substitution est le principe de fonctionnement de l'héritage.

On peut substituer :

- n'importe quel objet d'une super-classe (ou classe de base ou classe mère)
- par n'importe quel objet d'une sous-classe (ou classe dérivée ou classe enfant).

Y sont associées les notions de : **polymorphisme**, **redéfinition**, **interface**, **implémentation**, **classe et méthode abstraites**.

Paradigme 5 : pdg. de la généricité (type variable)

Choisissez vos algorithmes.

Paramétrez-les de façon qu'ils soient en mesure de fonctionner

avec une variété de types et de structures de donnée.

La généricité au sens stricte consiste à ce que le type des paramètres des algorithmes devienne un paramètre lui aussi.

Par exemple, on a un algorithme qui permet de trier un tableau lignes-colonnes selon une colonne particulière. On va faire adapter cet algorithme pour qu'il puisse fonctionner quelque soit la colonne et même avec plusieurs colonnes.

On peut aussi adapter l'algorithme pour qu'il puisse trier un tableau, mais aussi n'importe quelle sorte de collection (une liste, un array-liste, etc.).

La généricité s'appuie sur la notion d'interface.

Paradigme final : pdg. de la productivité : factoriser encore plus le code

- **Les interfaces :** elles permettent d'élargir la possibilité de coder de la généricité stricte et sont largement utilisées dans les design patterns. Une interface est un type abstrait (tandis que la classe est un type concret). Ce type abstrait est utilisé de façon générique dans le code. Seule l'instanciation concrète différencie ensuite les comportements.
- **Les design patterns :** ce sont des solutions classiques à des petits problèmes de codage. Ils s'appuient souvent sur les interfaces.
- **Les patterns d'architecture :** ce sont des solutions classiques d'architecture. Le MVC est un pattern d'architecture.
- **Les bibliothèques :** elles offrent des méthodes permettant d'éviter de les réécrire et organisant même la façon de réfléchir à la solution des problèmes à résoudre.
- **Les framework :** ce sont des architecture semi-finies qu'on peut va ensuite paramétrer et compléter en fonction des spécificités du produit à réaliser.

0-2 - Programmation objet vs. Programmation procédurale

Principe de la distinction

Approche procédurale ou fonctionnelle : ce que le système fait

- L'approche procédurale propose une méthode de décomposition basée uniquement sur **ce que le système fait**, c'est-à-dire aux fonctions du système **On s'intéresse aux fonctions puis aux données manipulées par les fonctions**. Les fonctions sont identifiées puis décomposées en sous-fonctions, et ainsi de suite jusqu'à l'obtention d'éléments simples programmables.
- C'est la méthode cartésienne ou analytique.
- L'architecture du système montre finalement un **système unique** avec une **architecture hiérarchique des fonctions**.

Approche objet : démarche systémique : ce que le système est

- L'approche objet propose une méthode de décomposition non pas basée sur ce que le système fait, mais sur **ce que le système est.**
- Ce que le système est, ce sont les objets, au sens commun, qui interviennent dans le système. Par exemple des utilisateurs, des produits, des paniers, etc. Un objet correspond en informatique à une donnée de type « structure ».
- **On s'intéresse donc à à un ensemble de données formant un objet (par exemple le nom, le prénom, la date de naissance l'adresse mail d'un utilisateur) puis aux fonctions au sens de la programmation procédurale associées à ces objets (on les appelle des « méthodes ».** L'objet est une unité de services rendus (ses fonctions). Les fonctions associées à leur objet et sont identifiées par leur objet, pour faire des actions à partir des informations qu'il porte ou pour le modifier. Chaque objet est un sous-système dans le système, portant ses données et ses méthodes de façon indépendante du système complet alors qu'en programmation procédurale classiques les fonctions dépendent très vite des objets du système.
- C'est la **méthode systémique** : les objets collaborent entre eux pour constituer le système. C'est ce qu'on appelle la « **synergie** » entre les objets. On peut en rajouter ou en supprimer sans perturber les collaborations entre les objets non concernés. A noter que la méthode MERISE était déjà une méthode systémique s'appuyant tout particulièrement sur une analyse des données (le MCD).
- L'architecture du système montre finalement un **ensemble de sous-systèmes considérés comme des objets**. C'est la description de ce que le système est. **Chaque sous-système ou objet intègre sont jeu de fonctions**. C'est la description de ce que le système fait à travers ce que font chacun des sous-systèmes ou objets.

Avantages de l'approche objet

- **Stabilité de la modélisation** par rapport aux entités du monde réel.
- **Construction itérative facilitée** par le **couplage faible** entre les composants (les sous-systèmes ou les objets).
- Possibilité d'une **réutilisation les composants** d'un développement à un autre.
- Accessoirement et en conséquences des avantages précédents : limitation des variables globales.

> ⇨ La programmation objet a répondu à la « **crise du logiciel** » de la fin des années 80, quand les systèmes informatiques sont devenus trop gros à gérer avec le paradigme procédural.

5 concepts fondateurs

La programmation objet est basée sur **5 concepts fondateurs** :

1. **Objet** et **encapsulation**
2. **Message** (l'échange entre deux objets – la **synergie** entre objets).
3. **Classe** (la généralisation de l'objet)
4. **Héritage** (la factorisation de propriétés et de fonctions entre 2 classes)
5. **Polymorphisme**

1 - Objet et encapsulation

Présentation

Objet = **données** (état) + **méthodes** (comportement, rôles, responsabilités)

D'un point de vue abstrait, un objet informatique est une représentation d'un objet (une réalité) du monde extérieur. Cette représentation est caractérisée par des valeurs et des rôles à jouer.

D'un point de vue informatique, un objet informatique est une **variable de type structure** avec une ou plusieurs valeurs (ses attributs) qui seront manipulés (en lecture ou en écriture) par **les fonctions associées** à l'objet (les méthodes). Cette variable est aussi associée à des fonctions de plus haut niveau : les responsabilités ou rôles.

Etat d'un objet - attributs

L'état d'un objet c'est **l'ensemble des valeurs instantanées** des attributs de l'objet.

Certaines parties de l'état peuvent **évoluer au cours du temps**.

D'autres parties de l'état peuvent **être constantes**.

Comportement - méthodes

Le comportement d'un objet regroupe des fonctions au sens de la programmation procédurale, qu'on appelle **méthodes** (ou **compétences** ou **responsabilités** ou **rôles**) d'un objet.

Les méthodes sont des fonctions qui permettent d'**accéder aux valeurs des attributs** d'un objet (getter et setter) mais aussi des **fonctions de plus haut niveau (responsabilités et rôles).**

Ces méthodes sont déclenchées par des stimulations externes : des messages envoyés par d'autres objets (c'est-à-dire des appels de méthodes).

On parle de **méthode** pour l'en-tête et le code de la fonction. On parle d'**opération** pour la seule en-tête des méthodes.

Identité de l'objet

Chaque objet possède une **identité** attribuée de manière implicite à la création de l'objet et qui n'est jamais modifiée (c'est son **adresse en mémoire**).

De ce fait, **chaque objet a forcément une identité distincte d'un autre objet** et est donc forcément différent d'un autre objet.

Toutefois, deux objets différents ayant une identité différente peuvent avoir le même état.

Techniquement, l'objet c'est la mémoire qui permet de stocker l'état et les comportements de l'objet. L'identité de l'objet c'est cette mémoire.

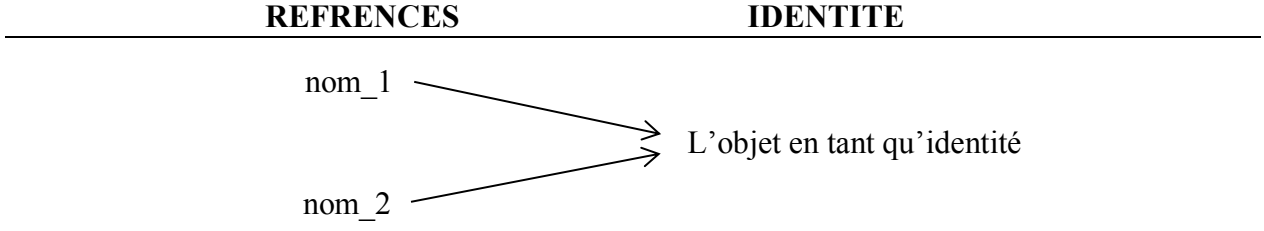
Identité de l'objet et référence

Un objet peut être nommé ou pas. Il est parfois difficile de nommer tous les objets : on peut donc les nommer du nom de leur classe, avec « : » devant.

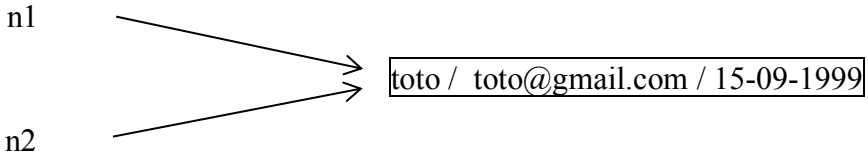
Le nom de l'objet correspond en théorie à l'identité de l'objet : à la mémoire de stockage de l'objet.

Toutefois techniquement, pour accéder à un objet, on passe toujours par une référence : une variable qui permet d'accéder à l'identité de l'objet. Techniquement et à l'usage, le nom de cette référence est le nom de l'objet.

Donc deux références avec 2 noms différents peuvent faire référence au même objet. Un même objet, en tant qu'identité, peut avoir plusieurs noms en tant que référence.



Par exemple, on a deux références n1 et n2 pour un même objet, une même identité :



Encapsulation

Principes

L'encapsulation consiste à cacher une partie de l'état de l'objet.

On ne peut alors accéder à cet état, pour le consulter ou le modifier, que en utilisant les méthodes de l'objet.

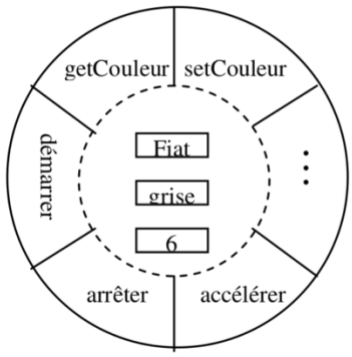
C'est la « **visibilité** » qui définit si les attributs et les méthodes sont cachées ou non.

Les méthodes sont l'interface obligatoire d'accès aux données d'un objet.

Exemple

Les données sont encapsulées par les méthodes :

Exemple : l'objet maVoiture



Visibilité

La visibilité des attributs et des méthodes est précisée par des mot-clés ou des symboles.

Il y a essentiellement **3 niveaux de visibilité : private, protected et public**

Symbole	Mot-clé	Signification
-	private	Visible uniquement **dans la classe**
#	protected	Visible dans la classe, **dans ses sous-classes**
+	public	Visible **partout**

Syntaxe UML

Les objets sont soulignés et placés dans un rectangle.

Le nom de l'objet commence par une minuscule.

Persistance et activation des objets

Un objet persistant sauvegarde son état dans un système de stockage permanent, de sorte qu'il est possible d'arrêter le processus qui l'a créé sans perdre l'information représentée par l'objet. C'est la passivation de l'objet.

L'activation de l'objet consiste à reconstruire l'objet dans l'état dans lequel on l'avait sauvegardé.

Par défaut, les objets ne sont pas persistants.

2 - Le Message

Méthode de communication

Pour accéder à une méthode d'un objet, que ce soit une méthode de consultation ou modification de son état, ou une méthode-responsabilité, il faut passer par l'objet.

On écrit :

```
objet.methode()
```

Par exemple

```
user.setSurnom("toto")
```

Ou encore

```
objet.afficher()
```

Communication

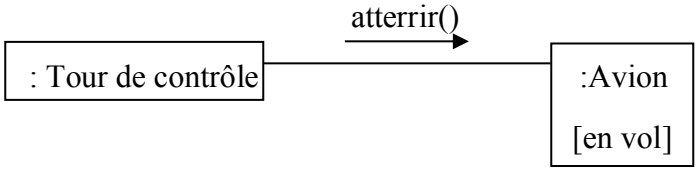
Quand on fait appel à une méthode d'un objet 2 à partir d'un objet 1, on dit que **l'objet 1 envoie un « message » à l'objet 2**. Ce message, c'est la méthode appelée.

Envoyer un message, c'est communiquer.

L'envoi de message s'apparente à un appel de fonction.

Scénario de communication

L'objet révèle son vrai rôle et sa vraie responsabilité lorsque, par l'intermédiaire de l'envoi de messages, il s'insère dans un scénario de communication (c'est-à-dire un cas d'utilisation concret du système).



L'objet « :Avion » a dans ses méthodes la fonction « atterrir() ».

L'objet « :Avion » est dans l'état « en vol ». Cela veut dire qu'un de ses attributs vaut « en vol ».

On dit que l'objet « Tour de contrôle » envoie un message à l'objet « avion » : celui d'atterrir. Concrètement, cela veut dire que l'objet « Tour de contrôle » a une méthode m() qui exécute un code comme : avion.atterrir(). C'est ce que le représentation ci-dessus veut dire.

Ainsi, la tour de contrôle ordonne à l'avion d'atterrir. L'avion va exécuter la méthode.

Communication entre objets

Un système informatique peut être vu comme une société d'objets qui communiquent entre eux pour réaliser les fonctionnalités de l'application.

Le comportement global d'une application repose sur la communication entre les objets qui la composent. C'est ce qu'on appelle la synergie entre les objets.

De ce fait, l'étude des relations entre les objets du domaine est de première importance dans la modélisation objet.

De plus, la première communication étant la communication entre les utilisateurs (acteurs externes) et le système, on peut aussi partir de l'analyse fonctionnelle pour trouver les classes.

Collaboration et communication

On peut parler indifféremment de communication ou de collaboration entre objet.

Cependant, on parle plutôt de collaboration quand on décrit les communications nécessaires pour réaliser une fonctionnalité. La collaboration est plutôt finalisée (on collabore pour quelque chose).

Subtilités techniques : liaison dynamique et polymorphisme

- Dans la programmation procédurale, l'appel de fonction correspond à un branchement sur la fonction à partir d'une table d'adresses. La fonction à appeler est définie une fois pour toute. Au moment de la compilation, on sait quelle fonction sera appelée.
- En POO, il n'y a **pas de correspondance statique prédéfinie** entre le nom de l'appel et le code effectivement exécuté. **La liaison est dynamique**

  Le mécanisme d'envoi de message consiste à remonter à la classe pour trouver la fonction et à remonter la hiérarchie des classes jusqu'à trouver la fonction correspondante tant au niveau du nom qu'au niveau des paramètres. Si aucune fonction n'est trouvée, alors il y aura un message d'erreur.

  Ce mécanisme permet le **polymorphisme**, autrement dit la possibilité d'avoir des fonctions différentes et donc des comportements différents (polymorphe), pour un même nom de fonction.

Synchronisation des messages et UML

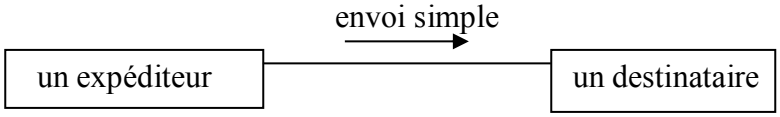
La synchronisation précise la nature de la communication et les règles qui régissent le passage des messages.

Message synchrone

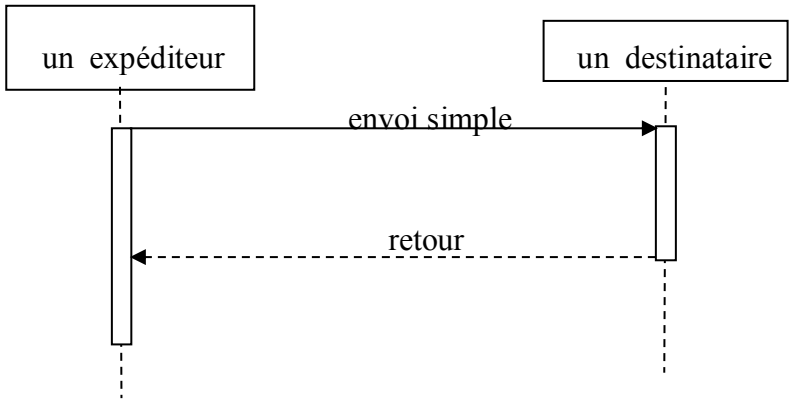
Une fois le message envoyé, l'expéditeur est bloqué jusqu'à ce que le destinataire accepte le message.

Un appel de procédure est un message synchrone.

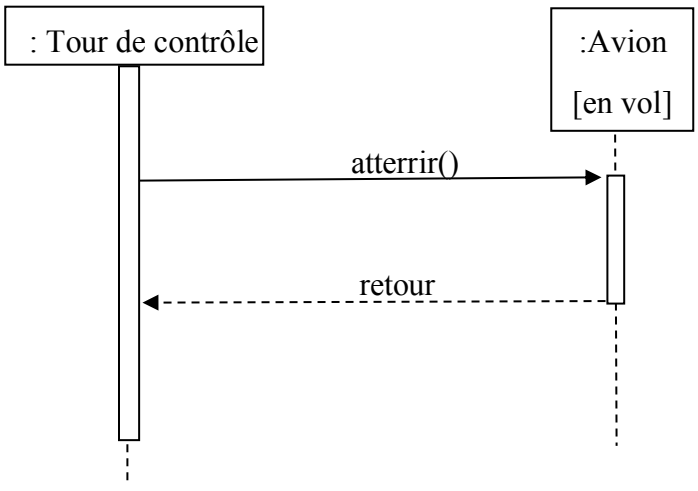
- ***Diagramme de collaboration correspondant :***



- ***Diagramme de séquence correspondant :***



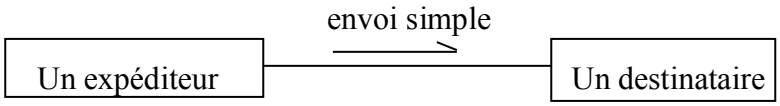
Des flèches à pointes noires ou en simples V peuvent représenter les messages synchrones.



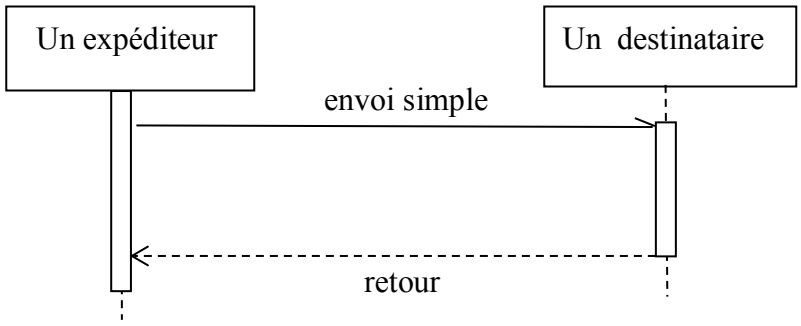
Message asynchrone

Il n'interrompt pas l'exécution de l'expéditeur. L'expéditeur envoie le message sans savoir quand ni même si le message sera traité par le destinataire.

- ***Diagramme de collaboration correspondant :***



- ***Diagramme de séquence correspondant :***



3 catégories d'objet en fonction de leur mode de communication

Un objet peut être :

1. **émetteur** de message sans jamais en recevoir,
2. **destinataire** de message sans jamais en émettre,
3. **émetteur et destinataire** de message.

Les acteurs : client, thread

Ce sont des objets à l'origine d'une interaction. Ce sont des objets actifs. Il possède un « fil d'exécution » : un thread. Ils passent la main aux autres objets. On peut les appeler « client ».

Les serveurs

Ce sont des objets qui ne sont jamais à l'origine d'une interaction. Ils sont toujours destinataire des messages et ensuite ils répondent. Ce sont des objets passifs.

Les agents

Ce sont des objets qui cumulent les caractéristiques des acteurs et des serveurs. Ils peuvent interagir avec les autres objets à tout moment, de leur propre initiative ou suite à une sollicitation externe.

Les agents permettent le mécanisme de délégation. Un client peut communiquer avec un serveur qu'il ne connaît pas via un agent. Les agents découplent les acteurs des serveurs en introduisant une indirection dans le mécanisme de propagation des messages.

3 - La Classe

Classe, objet, instance

Une classe est la description d'un ensemble d'objets ayant les mêmes méthodes et les mêmes types de données.

La classe peut être vue comme une extension de la notion de type. C'est un modèle qui permet ensuite de créer des objets concrets.

L'objet est la réalisation concrète de la classe : **un objet est une instance d'une classe**.

Les objets apparaissent alors comme des variables de type classe.

Une classe est donc définie par ses attributs et ses méthodes.

Syntaxe UML

La classe et ses attributs

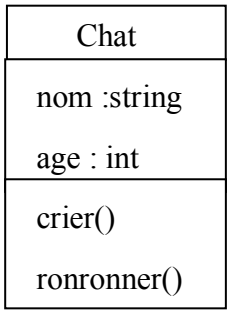
La classe est représentée par une **bloc rectangulaire à 3 parties horizontales**.

Le nom de la classe commence par une **Majuscule** et est au **singulier** si une instance de la classe est un objet unique (ici, une instance de Chat est un chat unique).

En dessous on met les attributs. Par défaut ils sont tous « **private** ».

En dessous on met les méthodes. Par défaut, ils sont tous « **public** »

La classe "Chat" :

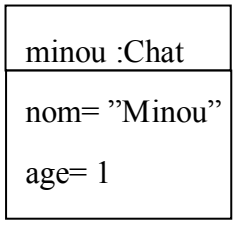


Instanciation d'un objet appelé «minou »

- ***Code***

```
Chat minou = new Chat(« Minou », 1) ;
```

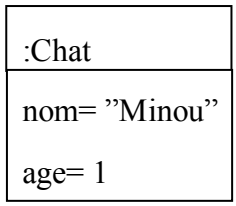
- ***UML***



L'objet Chat appelé minou

On ne représente pas les méthodes de l'objet. Elles sont dans la classe. L'objet peut y accéder.

- ***objet sans nom en UML***



:Chat représente un objet de la classe Chat

- ***attribut d'instance***

Les attributs d'un objet sont appelés attributs d'instance par opposition aux attributs de classe (cf. ci-dessous).

Les principales catégories de méthodes

Les méthodes peuvent être classées en différents types :

- **Les constructeurs :** pour créer les objets (**création**).
- **Les destructeurs :** pour détruire les objets (**destruction**).
- **Les getter:** (ou sélecteurs) pour renvoyer tout ou partie de l'état d'un objet (**consultation**).
- **Les setter :** (ou modificateurs) pour modifier tout ou partie de l'état d'un objet (**modification**).
- **Les itérateurs :** pour consulter le contenu d'une structure de données qui contient plusieurs objets.
- **Les responsabilités ou rôles :** ces méthodes correspondent aux fonction de haut niveau permettant la réalisation des fonctionnalités du système. En phase de conception, on s'intéresse surtout à ces méthodes.

Attributs et méthodes de classe

Attribut de classe

Les attributs d'instance sont des attributs qui définissent l'état d'un objet.

Les attributs de classe caractérisent la classe en général et pas un objet en particulier.

Les attributs d'instance sont accessibles avec l'objet qui les porte : **objet.attribut**

Les attributs de classe sont accessibles avec la classe qui les porte : **Classe.attribut**

Le nom d'un objet commence toujours par une minuscule. Le nom d'une classe commence toujours par une majuscule.

Exemple

Le nombre d'objets instanciés de la classe.

Méthode de classe

C'est la même chose pour les méthodes. En général, les méthodes sont des méthodes d'instance. Elles sont accessibles avec l'objet qui les porte : **objet.methode()**

Il arrive que ce soit des méthodes de classe. Elles sont alors accessibles avec la classe qui les porte : **Classe.methode()**

Exemple

Si on veut accéder au nombre d'objets instanciés, il faudra un getter(). Mais ce getter est une méthode de classe : elle concerne la classe et pas un objet en particulier.

On écrira : Chats.getNbChats()

4 - L'Héritage

Héritage et représentation ensembliste

Inclusion d'un ensemble dans un autre

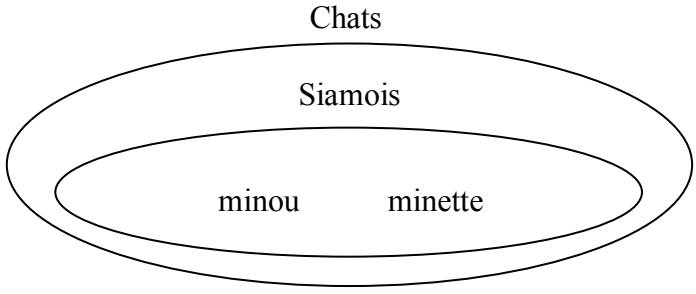
La représentation ensembliste permet de concevoir aisément la notion héritage : il suffit de concevoir l'inclusion d'un ensemble dans un autre.

L'ensemble inférieur hérite des propriétés et des associations de l'ensemble supérieur : l'espèce hérite des attributs du genre.

Un ensemble fait hériter ses attributs à tous les ensembles de niveaux inférieurs qu'il contient. Le genre fait hériter à toutes ses espèces.

Cette organisation forme une hiérarchie, c'est la hiérarchie de spécialisation/généralisation, encore appelée hiérarchie « is-a », « est-un ».

Formalisme ensembliste



Chaque élément d'un ensemble est aussi élément des ensembles qui le contiennent : « Minou », c'est mon chat siamois : il appartient à l'ensemble des Siamois, mais aussi à l'ensemble des Chats. Minette c'est le chat siamois de la voisine.

Vocabulaire ensembliste

Classe : ensemble, entité, table

On peut considérer les classes comme des ensembles. Les attributs de la classe sont les attributs de l'ensemble. Chaque objet est un élément de l'ensemble.

La notion rejoint celle d'entité et de table, mais au sens de la définition de la table, sans les éléments dans la table. La Classe rejoint la notion de table au sens d'un « CREATE TABLE » SQL.

A noter qu'on dit « entité » par facilité, mais il s'agit en réalité d'entité-type et l'entité est en réalité un objet (ou un tuple) !

Objet : élément, tuple, composé

L'objet correspond à un élément d'un ensemble.

La notion rejoint celle de tuple (ligne de la table). Un tuple est composé de chacun de ses attributs.

Sous-ensemble : espèce, classe-enfant, classe dérivée

L'espèce est un sous-ensemble de l'ensemble dont on parle : le siamois est une **espèce** de chat. Quand on parle d'un sous-ensemble, on dit aussi : « classe-enfant » ou « classe dérivée » ou « sous-classe ».

Sur-ensemble : genre, classe-mère, classe de base

Le genre est un « sur-ensemble » de l'ensemble dont on parle : le chat est le **genre** du siamois. Quand on parle d'un « sur-ensemble », on dit aussi : « classe-parent » ou « classe-mère » ou de « classe de base ».

Abstrait / Concret – Abstraction - Abstraire

L'abstraction, c'est la classe !

Tous les noms d'ensemble sont abstraits : ce sont des abstractions. Une classe est donc toujours abstraite d'un certain point de vue : ce qui est concret, c'est l'objet instance de la classe.

Les seules choses concrètes, ce sont les éléments de l'ensemble, c'est-à-dire les objets (Minou, mon chat siamois).

Abstraire

Abstraire, c'est remonter du concret à l'abstrait, donc des objets à la classe qui les englobe. Par exemple de Minou et Minette au Siamois, ou de Minou et Minette au Chat.

Abstraire, c'est aussi remonter de l'espèce au genre, c'est-à-dire d'une classe-enfant à une classe-parent. Par exemple du Siamois et de l'Angora au Chat

Abstraire consiste à trouver des attributs communs à plusieurs ensembles ou à plusieurs choses concrètes pour définir un ensemble qui portera ces attributs et qui inclura les ensembles ou les choses concrètes en question.

Concrétiser

Concrétiser, c'est descendre de l'abstrait au concret, donc de la classe aux objets. Par exemple de la classe Chat aux objets concret que sont Minou et Minette.

Concrétiser c'est aussi aller de la classe Chat aux classes enfants Siamois ou Européen. Toutefois, la concrétisation finale doit aboutir à des objets concrets.

Notion de classe abstraite

En POO, peut définir des classes comme étant abstraite.

Cela veut dire qu'elles ne pourront pas être instanciées : il n'existera pas d'objet de cette classe.

Une classe abstraite est forcément une classe de base pour une ou plusieurs classes dérivées qui elles pourront être instanciées.

L'héritage en programmation objet

Principe : mettre le commun dans une classe de base

Dès que 2 classes ont des attributs ou des méthodes en commun, on retire ces attributs et méthodes de leurs classes de départ pour créer une nouvelle classe avec les attributs et méthodes communes. On fait ensuite « hériter » les classes de départ de la nouvelle classe.

La nouvelle classe est appelée : classe de base (ou classe-mère ou classe-parent).

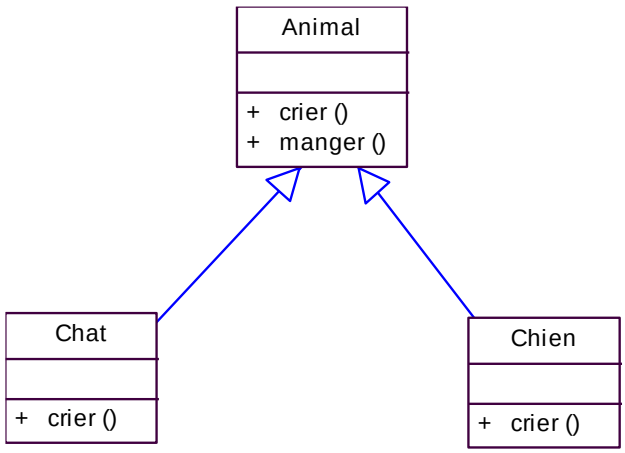
Les classes de départ sont appelées : classe dérivée (ou classe-fille ou classe-enfant).

On dit que les classes enfants « héritent » de la classe parent. En effet, tous les attributs et les méthodes de la classe parents seront accessibles par la classe enfant.

Utilisation

L'héritage est utilisé pour :

- **Factoriser le code** : ça évite d'avoir à écrire deux fois la même chose !
- **Utiliser un code existant déjà** : c'est une forme de factorisation.
- **Le polymorphisme** (possibilité qu'une même méthode se réalise différemment).
- La conception abstraite (conception de « haut niveau », éloignée du codage concret).



héritage de avec polymorphisme de crier()

Héritage avec polymorphisme : la méthode crier est définie dans les classes filles en plus de la classes mère. Elle pourrait même être abstraite dans la classe mère. Ainsi, si on appelle la méthode crier à partir d'un animal, tout dépend de quel animal il s'agit pour savoir quelle méthode se déclenche réellement.

Principe de substitution

On peut substituer :

- n'importe quel objet d'une super-classe (ou classe de base ou classe mère)
- par n'importe quel objet d'une sous-classe (ou classe dérivée ou classe enfant).

C'est ce qui permettra le polymorphisme.

L'abstraction dans la programmation objet : un nouveau paradigme de programmation

- L'abstraction consiste à regrouper les éléments qui se ressemblent et à distinguer des structures de plus haut niveau d'abstraction, débarrassées de détails inutiles.
- La classe décrit le domaine de définition d'un ensemble d'objets.
- Les généralités sont contenues dans la classe.
- Les particularités sont contenues dans les objets.
- Avec les langages-objet, le programmeur peut construire une représentation informatique des abstractions de haut niveau correspondant aux usages mêmes de l'application, sans traduction vers des concepts de plus bas niveau, comme les variables, les types abstraits de données et les fonctions des langages non objet.

5 - Le Polymorphisme

Principes

- Le polymorphisme décrit la caractéristique d'un élément qui peut **prendre plusieurs formes** (l'eau peut être à l'état solide, liquide ou gazeux).
- Le polymorphisme désigne le principe qui fait qu'**un nom d'objet peut désigner des objets de différentes classes** issues d'une même arborescence d'héritage.
- Le polymorphisme désigne surtout le **polymorphisme d'opération** : la possibilité de déclencher des opérations différentes en réponse à un même message.
- **Le polymorphisme s'appuie sur le principe de substitution**. On peut manipuler des objets dont le type est abstrait au niveau de la classe générale (abstraite). Le type deviendra concret quand l'objet deviendra concret et portera le type de sa classe spécialisée, en application du principe de substitution : un objet d'une classe de base peut être remplacé par un objet d'une classe dérivée.
- Le polymorphisme permet de manipuler des objets sans en connaître précisément le type : c'est un cas de généricité.

Remarques méthodologiques

- Le polymorphisme concerne le codage du corps des différentes méthodes. Par rapport à la conception, c'est une notion qui intervient à bas niveau.
- Il ne faut pas penser l'analyse en terme de polymorphisme, mais en terme d'abstraction (et donc d'héritage). L'analyse en terme d'abstraction rend possible le polymorphisme au niveau concret.
- Les bénéfices du polymorphisme sont un plus grand découplage entre les objets : ils sont avant tout récoltés durant la maintenance.

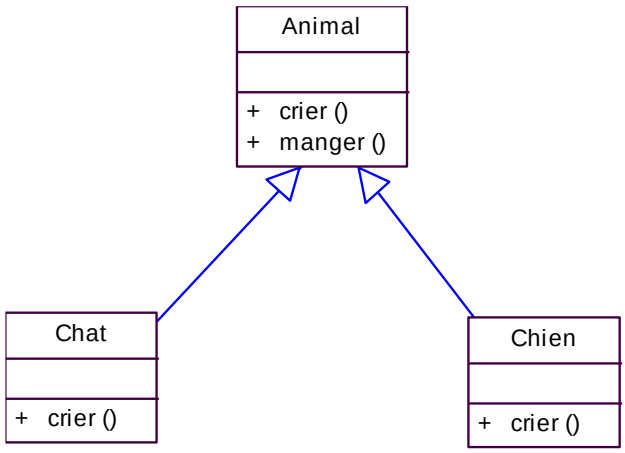
Polymorphisme - méthode abstraite

Le polymorphisme est souvent associé aux méthodes abstraites.

Une méthode abstraite est une méthode dont seule l'en-tête est définie et pas le corps.

Toute classe qui contient une méthode abstraite est une classe abstraite : elle ne peut pas être instanciées.

Souvent les méthodes abstraites sont associées au polymorphisme : le corps de la méthode est défini concrètement dans les classes dérivées.



Si on a code qui dit :

```
animal.crier() ;
```

ce code est syntaxiquement juste.

A l'éxécution, il faudra que l'animal contienne un chat ou un chien pour qu'un crier concret puisse s'exécuter. C'est le principe de substitution qui permet de faire

animal=new Chien() ;

animal.crier()

Ainsi le comportement est polymorphe.

Interface

Une interface est une sorte de classe abstraite qui ne contient que des méthodes abstraites et pas d'attributs.

De plus, une classe peut « **réaliser**» plusieurs interfaces.

La relation entre une classe et une interface ressemble à un héritage, mais on ne dit pas qu'une classe hérite d'une interface : on dit qu'elle la réalise.

L'interface est le niveau d'abstraction le plus élevé : il permet au programmeur de construire une représentation informatique des abstractions correspondant aux usages mêmes de l'application., sans traduction vers des concepts de plus bas niveau, comme les variables, les types abstraits de données et les fonctions des langages non objet.