

# Paradigme objet et modélisation UML

Nous allons présenter rapidement les différents concepts de la conception/modélisation Objet. Pour représenter ces concept, nous adopterons le formalisme UML (mais nous ne l'utiliserons que pour décrire des classes, quasiment jamais pour en représenter le fonctionnement complet du système).

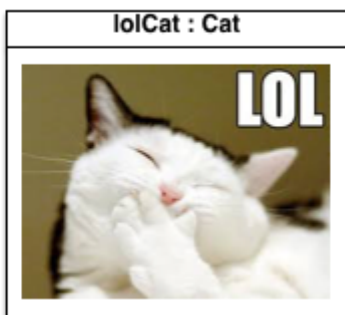
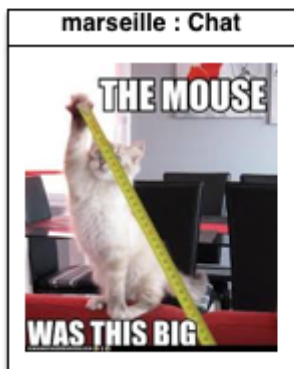
## Les objets et les classes

On ne manipule que des choses réelles, pour nous des **Objets**. Nos programmes (ou processus, projet, données, etc) consisteront à créer des objets et les faire interagir ensemble. En revanche, ces objets vont être décrits par des classes qui regroupent leurs propriétés.

Exemple : [des chats particuliers](#). Ce sont des **instances** (une réalisation concrète) de la classe *Chat*.



Tout objet fait parti d'une classe. La classe la plus générale étant souvent la classe *Object*.



Les classes sont décrite avec leurs méthodes (ce qu'ils font) et attributs :

### Chat



```
@startuml
title Chat
```

```
class Chat {  
    +bruit miaou()  
}  
@enduml
```



Pour les diagrammes UML, on utilisera <http://www.planttext.com> (qui utilise lui-même <http://plantuml.com>). Pour ce cours on mettra ainsi à gauche le résultat et à droite le code qui a permis de le produire.

## Notation & UML

### Notations

On écrira les noms en [CamelCase](#) qui est une convention Java. Chaque langage a sa propre recommandation que l'on cherchera à respecter pour augmenter la lisibilité de son code.

De plus :

- les classes commencent par une Majuscule,
- les objets commencent par une minuscule,

### UML

- un objet est une boîte dont le titre est **nomObjet : NomClasse** : ci-dessus *lolCat* est une instance de la classe *Chat*.
- les méthodes sont décrites par leur type de retour, leur nom et entre parenthèse leurs paramètres : ici la méthode *miaou* de la classe *Chat* ne prend pas de paramètre et rend un objet *bruit*.

## Relations entre les classes

On voit bien que les différentes classes d'animaux comme les [chats](#), les [chiens](#) voir même les [alpagas](#) partages des choses.

Nous avons à notre disposition essentiellement deux façons de regrouper les classes :

- par ce qu'elles **sont** (classes et héritage)
- par ce qu'elles **font** (interfaces)

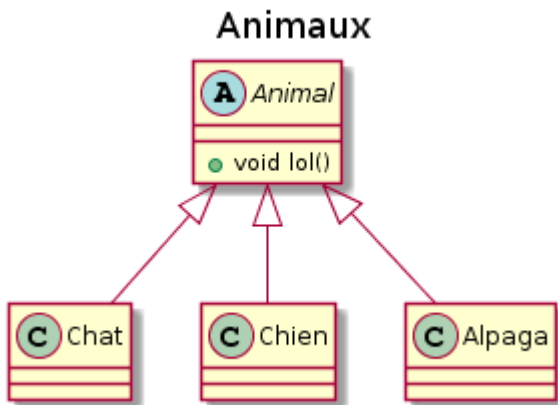
### Ce qu'ils sont

On regroupe les classes par les caractéristiques qu'elles partagent : **héritage** Avec l'héritage on

regroupe les classe par concept de plus en plus général.

Avec nos animaux :

- ces classes sont des particularités d'un concept plus général : l'"Animal"
- ici, mais ce n'est pas toujours le cas, il n'y a pas vraiment d'objet de la classe Animal, c'est une **classe abstraite** (d'où le A qui la caractérise ci-après).
- la seule chose qui les relie c'est qu'internet en fait des blagues : la méthode lol ( )



```

@startuml
title Animaux

abstract class Animal {
  +void lol()
}

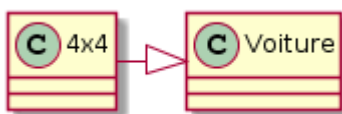
class Chat
class Chien
class Alpaga

Animal <|-- Chat
Animal <|-- Chien
Animal <|-- Alpaga

@enduml
  
```

## UML

La relation d'héritage est un "trait plein avec une flèche en triangle vide" :



```

@startuml

class 4x4
class Voiture

@enduml
  
```

```
Voiture <|-left- 4x4
```

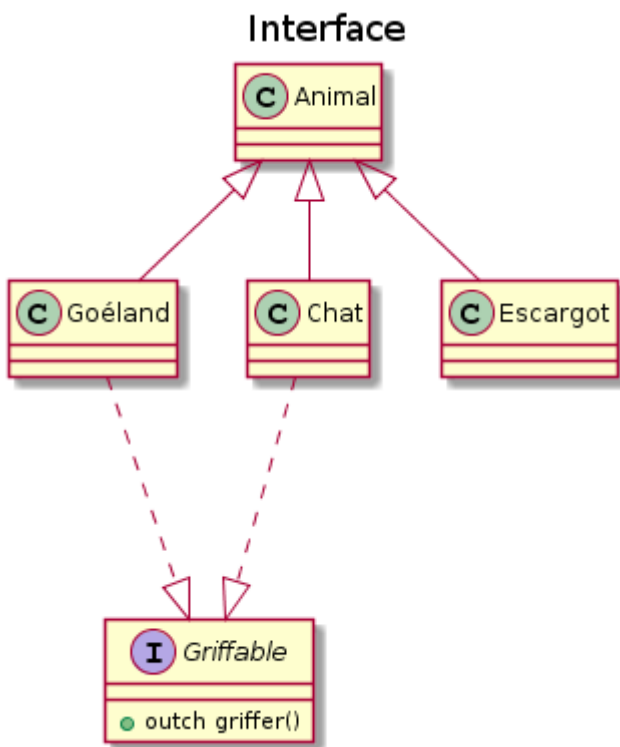
```
@enduml
```

## Ce qu'ils font

Souvent plusieurs objets de natures très différentes *font* les même choses. Ces actions communes sont liées par une **interface**.

Ainsi, un chat et un goéland peuvent tous deux griffer :

- mais pas de la même manière (l'un a des griffes, l'autre des serres),
- cela ne peut être une composante d'animal puisqu'un escargot ne peut pas griffer,
- les faire hériter d'un ancêtre commun ayant des griffes n'est pas biologiquement pertinent.



```
@startuml
title Interface

class Animal

interface Griffable {
    +outch griffer()
}

class Chat
class Goéland
```

```

Animal <|-- Chat
Animal <|-- Goéland
Animal <|-- Escargot

Griffable <|-- Chat
Griffable <|-- Goéland

@enduml

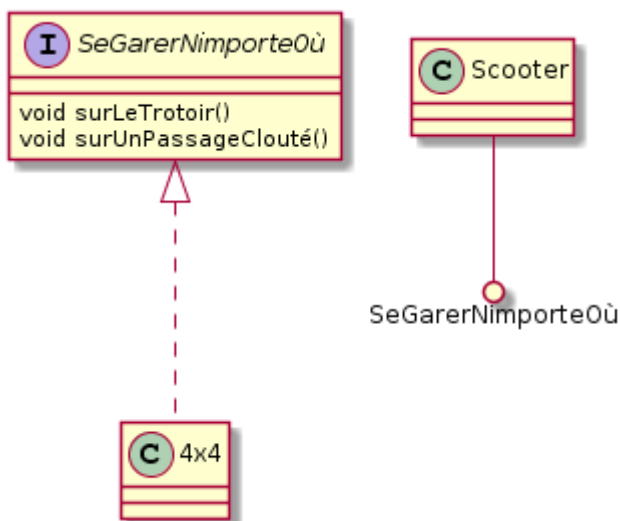
```

On dit qu'un objet **implémente une interface** s'il possède toute les méthodes de l'interface, ici une unique méthode griffer.

## UML

La relation d'implémentation d'interface peut être décrite de deux manières :

- "un trait pointillé avec une flèche en triangle vide",
- "un trait plein avec un rond vide" : utilisé pour rappeler un nom d'interface décrite précédemment



```

@startuml

class 4x4
class Scooter

interface SeGarerNimporteOù {
void surLeTrottoir()
void surUnPassageClouté()
}

SeGarerNimporteOù <|-- 4x4

Scooter -- () SeGarerNimporteOù
@enduml

```

# Attributs, méthodes et visibilité

## Visibilité

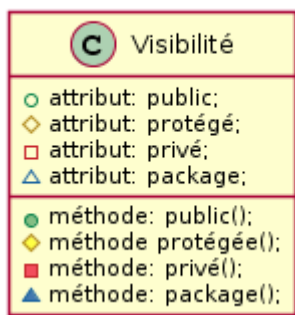
Les méthodes et attributs d'une classe peuvent avoir une visibilité limitée :

- **public** : tout le monde peut voir et utiliser la méthode/attribut.
  - rond vert (plein pour les méthodes, vide pour les attributs)
  - signe +
- **protégé** : la classe et ses descendants peuvent voir et utiliser la méthode/attribut.
  - losange orange (plein pour les méthodes, vide pour les attributs)
  - signe #
- **privé** : uniquement la classe peut voir et utiliser la méthode/attribut.
  - carré rouge (plein pour les méthodes, vide pour les attributs)
  - signe -
- **package** : les classes du package peuvent voir et utiliser la méthode/attribut.
  - triangle bleu (plein pour les méthodes, vide pour les attributs)
  - signe ~

On essaiera dans la mesure du possible de protéger les attributs d'une classe pour qu'ils ne soient pas modifiés intempestivement (portée privé ou protected).

C'est le principe d'[encapsulation](#) qui permet (entre autre) de modifier la structure interne de la classe en ajoutant des fonctionnalités sans briser le reste du code (qui utilise des méthodes publiques inchangées).

## Visibilité des attributs/classes



```
@startuml
```

```
title Visibilité des attributs/classes
```

```
class Visibilité {  
  +attribut: public;  
  #attribut: protégé;  
  -attribut: privé;  
  ~attribut: package;  
  +méthode: public();  
  #méthode protégée();  
}
```

```

-méthode: privé();
~méthode: package();

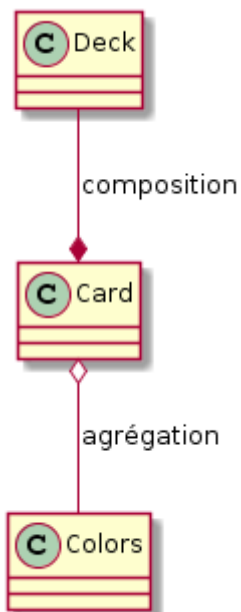
}

@enduml

```

## Composition et agrégation

Les différents types sont souvent des objets de classes que l'on a créé nous même. Selon que l'on a créé l'objet ou qu'on le partage on parlera de composition ou d'agrégation respectivement. Ce [lien](#) vous explique bien les différentes possibilités.



```

@startuml

title Composition et agrégation

class Colors
class Card
class Deck

Card o-- Colors : agrégation
Card *-up- Deck : composition

@enduml

```

## Méthodes de classes

On **souligne** les méthodes et attributs de classes (mot clé "static").

Leur intérêt est (au moins) double :

- permet de créer des objets de la classe en question ([pattern Factory](#)),
- des méthodes d'une classe utilisée seulement pour regrouper des fonctions/constantes ensemble. La classe [Math](#) de Java en est un bon exemple.

Une [bonne explication](#) pour le Java.

## Pour aller plus loin en UML

### Utilisation d'UML

[https://fr.wikipedia.org/wiki/UML\\_\(informatique\)](https://fr.wikipedia.org/wiki/UML_(informatique))

Utilisé un petit peu partout, lorsque l'on a besoin de catégoriser/regrouper des processus :

- informatique,
- définition et gestion de projets,
- création de services, brainstorming,
- ...

Très en vogue au début des années 2000, il est moins utilisé mais reste un bon outil de communication et de modélisation des données.

### Divers ressources sur le net

#### Introduction et résumé

- <http://liris.cnrs.fr/csolnon/coursUML.pdf>
- <https://people.irisa.fr/Jean-Marc.Jezequel/enseignement/ao-et-uml.pdf>

#### Série de cours

- <http://www.irisa.fr/triskell/members/pierre-alain.muller/teaching>
- <http://uml.free.fr> (ne vous laissez pas avoir par le design vieillot du site, c'est bien)
- <http://uml.developpez.com> et <http://laurent-piechocki.developpez.com/uml/tutoriel/lp/cours/>

#### Études cas UML

- [http://www.irisa.fr/manifestations/2002/jobim/papiers/O-p319\\_077.pdf](http://www.irisa.fr/manifestations/2002/jobim/papiers/O-p319_077.pdf)
- <http://www.epi.asso.fr/revue/articles/a0509b.htm>

From:

<https://wiki.centrale-med.fr/informatique/> - **WiKi informatique**

Permanent link:

[https://wiki.centrale-med.fr/informatique/public:mco-2:paradigme\\_objet\\_et\\_modelisation\\_uml](https://wiki.centrale-med.fr/informatique/public:mco-2:paradigme_objet_et_modelisation_uml)

Last update: **2016/02/11 22:13**

