# Qualité de développement

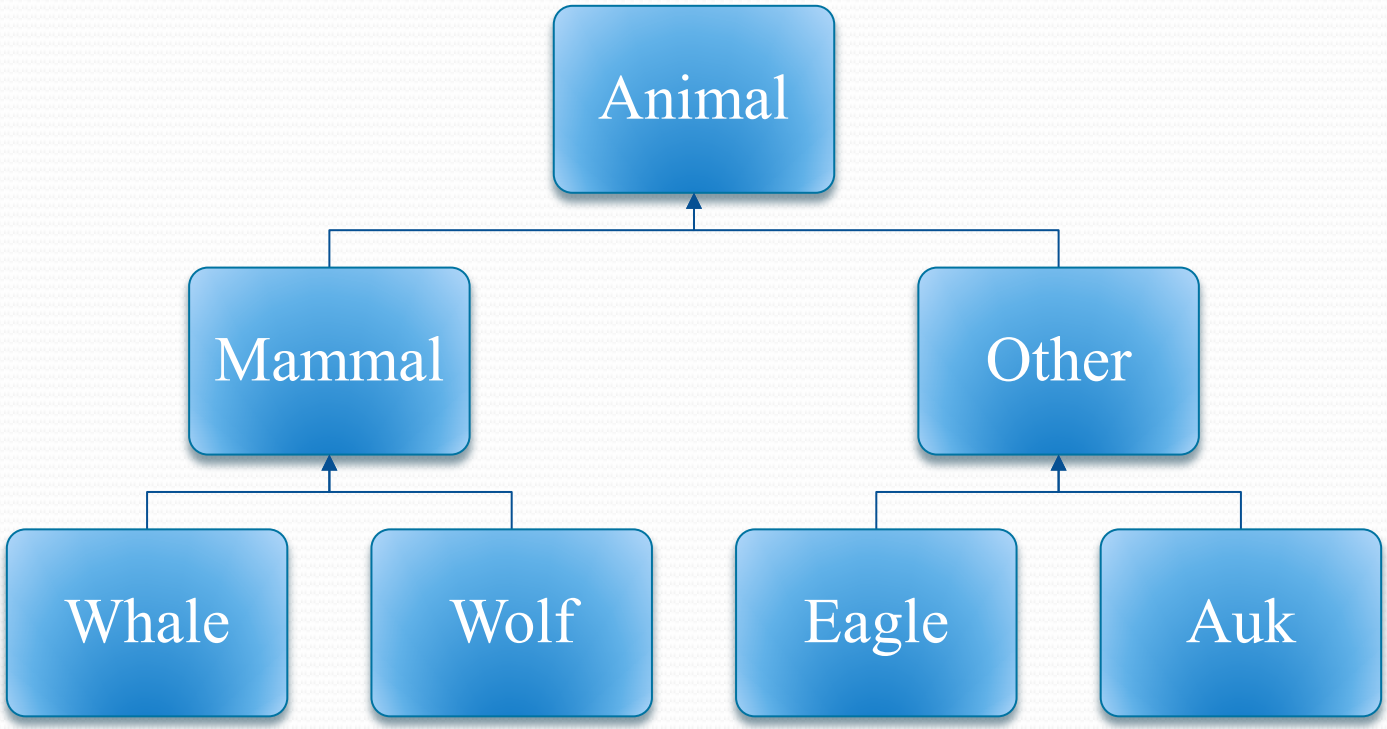
## CM3-1 : Programmation orientée objet

Mickaël Martin Nevot

V2.2.1

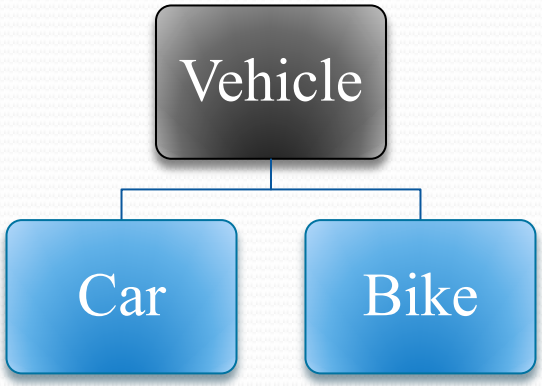
# Qualité de développement

# Héritage



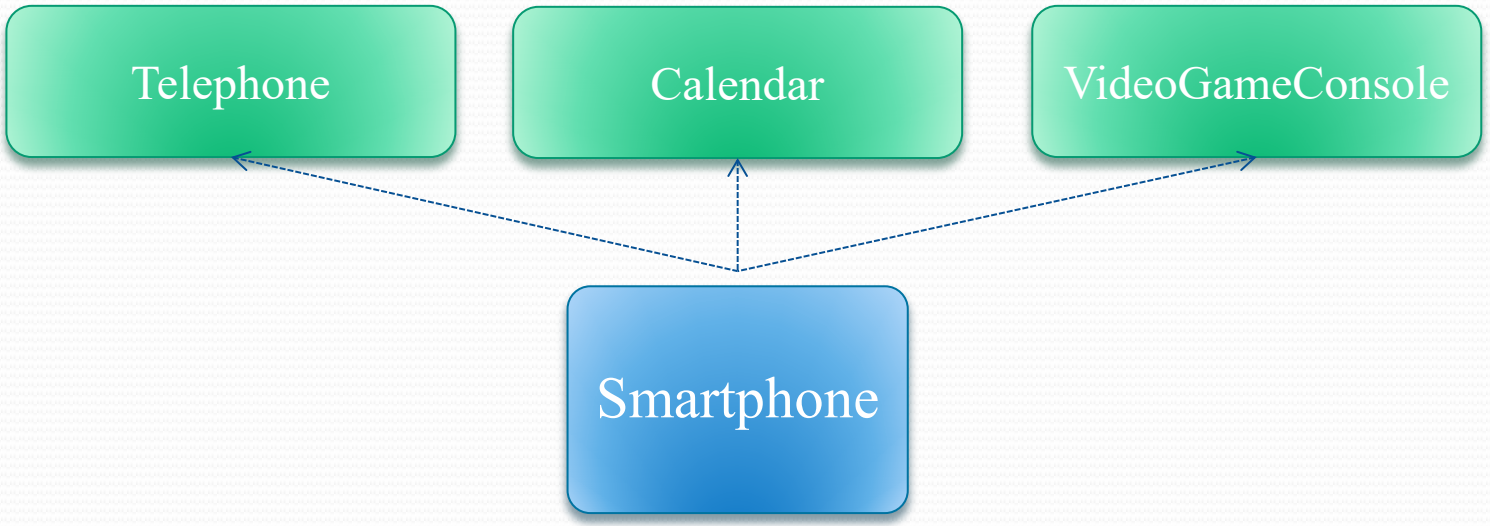
# Abstraction

- Classe abstraite :
  - **Ne peut pas être instanciée** (mais constructeur[s] possible[s])
  - Méthode abstraite :
    - Aucun service offert par la méthode mais une sémantique d'utilisation offerte
    - **Si une seule méthode est abstraite, la classe l'est aussi**

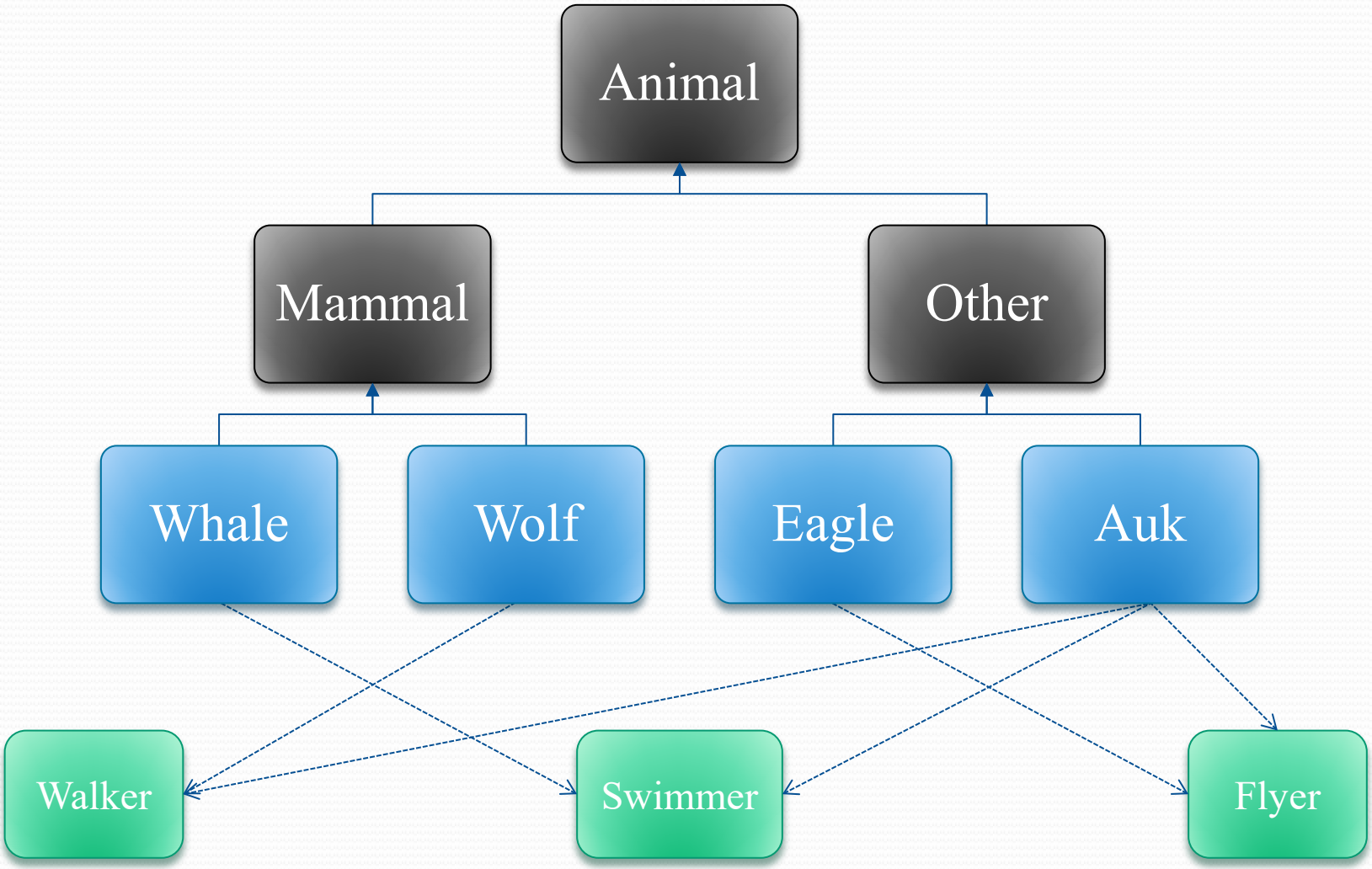


# Interface

- Modèle pour une classe
- **Classe totalement abstraite** sans attribut (non constant)
- Une classe implémentant une interface doit implanter ***(implémenter)*** toutes les méthodes déclarées par l'interface



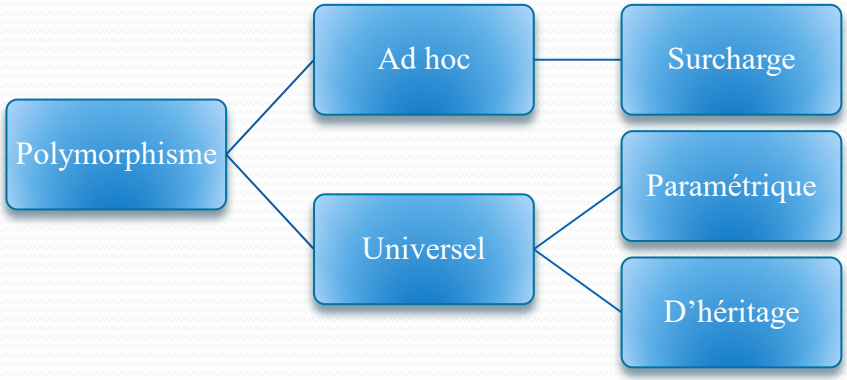
# Héritage



# Type et polymorphisme

- **Type** :
  - Défini les valeurs qu'une donnée peut prendre
  - Défini les opérateurs qui peuvent lui être appliqués
  - Défini la syntaxe : « comment l'appeler ? »
  - Défini la sémantique : « qu'est ce qu'il fait ? »
  - Une classe est un type (composé), une interface aussi…
- **Polymorphisme** :
  - Capacité d'un objet à avoir plusieurs types
  - Permet d'utiliser une classe héritière comme une classe héritée

# Catégories de polymorphisme

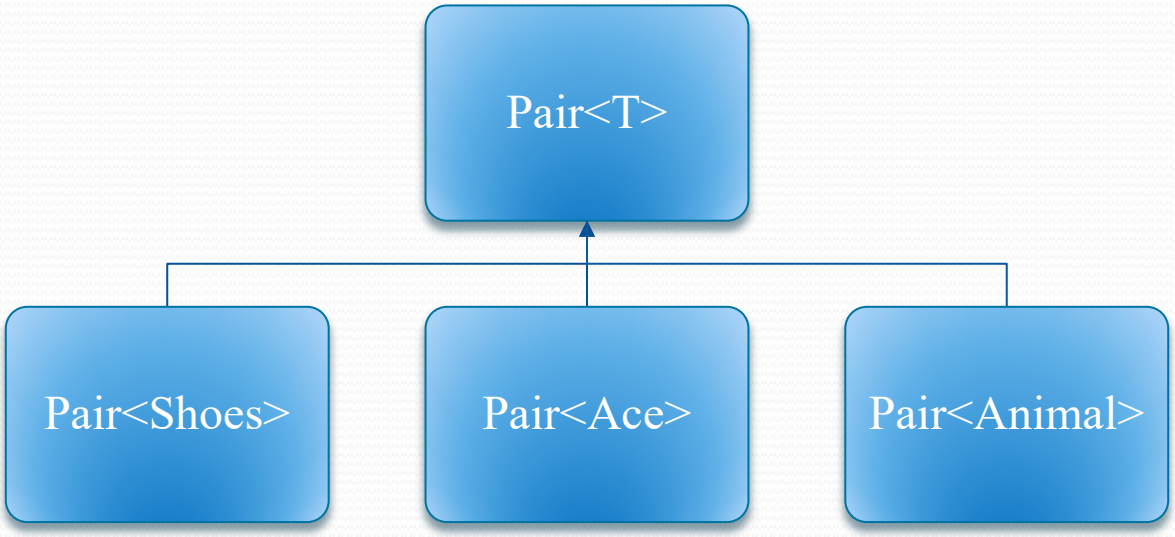



- **Surcharge** :
  - Permet d'avoir des méthodes de même nom, avec des fonctionnalités similaires
- **Paramétrique** :
  - Permet de définir plusieurs méthodes de même nom mais possédant des paramètres différents
- **D'héritage** :
  - Permet de redéfinir une méthode dans des classes héritières (l'appel se fait sans se soucier de son type intrinsèque)

# Généricité

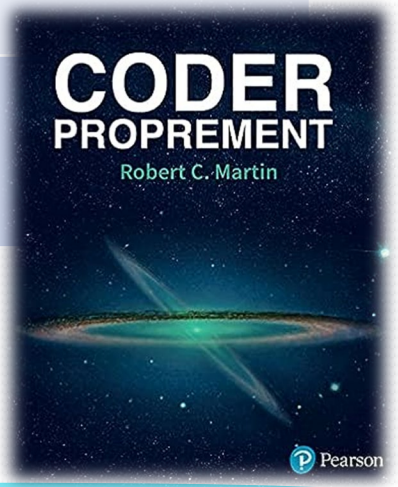
Aussi appelé type abstrait de données

- Généricité (polymorphisme paramétrique de type) :
  - **Polymorphisme** : objet pouvant prendre plusieurs types
  - **Paramétrique** : le type est paramétré par un autre type
  - **Type** : définition de la syntaxe et de la sémantique d'utilisation
- Comportement unique pour des types polymorphes



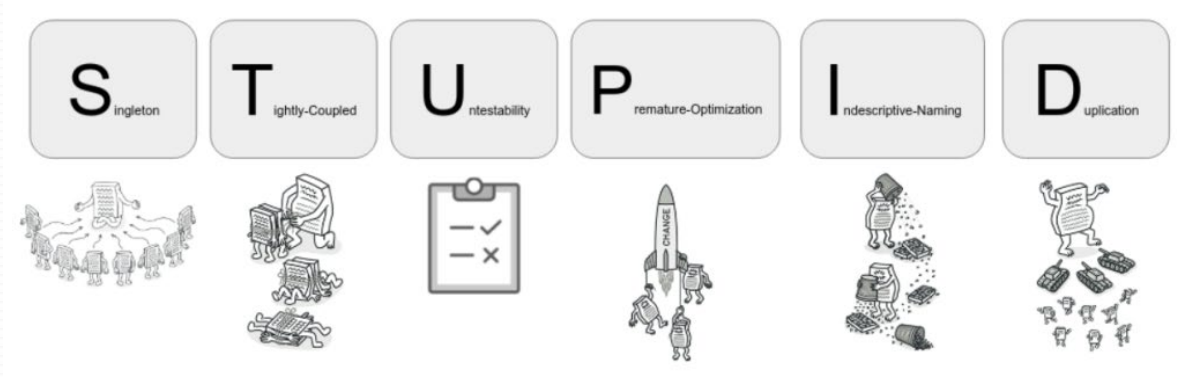
# S.O.L.I.D

| | | |
|---|---|---|
| S | **Responsabilité unique** (*single responsibility principle - SRP*) | Une classe doit avoir une et une seule responsabilité |
| O | **Ouvert/fermé** (*open/closed principle - OCP*) | Une classe doit être ouverte à l'extension, mais fermée à la modification |
| L | **Substitution de Liskov** (*Liskov substitution principle - SLP*) | Une classe doit pouvoir être remplacée par une instance d'un de ses sous-types, sans modifier la cohérence du programme |
| I | **Ségrégation des interfaces** (*interface segregation principle - ISP*) | Préférer plusieurs interfaces spécifiques pour chaque client plutôt qu'une seule interface générale |
| D | **Inversion des dépendances** (*dependency inversion principle - DIP*) | Il faut dépendre des abstractions, pas des implémentations |

# S.T.U.P.I.D

| | | |
|---|---|---|
| S | **Singleton** (*singleton*) | Singleton à utiliser avec parcimonie, et dans des cas bien précis |
| T | **Couplage fort (***Tight-Coupling*) | Privilégier le couplage faible entre les objets |
| U | **Incontrôlabilité** (*untestability*) | Avoir un code facile à tester |
| P | **Optimisation prématurée** (*premature-optimization*) | L'optimisation doit être la plus tardive possible : *Make it work, make it pretty and then, make it fast!* |
| I | **Nom indescriptible** *(indescriptible-naming)* | KISS : *keep it simple and stupid*<br>Ne pas nommer, décrire |
| D | **Duplication** (*duplication*) | DRY : *do not repeat yourself* |



# Loi de Démeter

- Loi de Déméter pour les fonctions et les méthodes **: LoD-F**
- **Principe de connaissance minimale**
- **Ne parlez qu'à ses amis immédiats**
- Toute méthode d'un objet peut simplement invoquer les méthodes des types suivants d'objets :
  - Lui-même
  - Ses paramètres
  - Les objets qu'il crée/instancie
  - Ses objets composants

# Loi de Démeter

- Avantages :
  - Résultat plus **maintenable** et plus **adaptable**
  - Faible dépendance entre les objets
  - Réduit la probabilité d'erreurs logicielles
- Inconvénients :
  - Multiplication des ***wrappers*** :
    - Augmenter le temps de développement initial
    - Accroître l'espace mémoire utilisé
    - Diminuer les performances

Méthode permettant la propagation d'appels de méthodes à leurs composants

# Aller plus loin

- Prototype et slot
- Mixin
- Trait
- Réflexion
- λ-calcul
- Programmation orientée aspect (POA)
- Programmation orientée acteur
- Programmation par contrat
- Espace de noms
- Métaclasse
- Object Calisthenics

# Crédits

## Auteur

Mickaël Martin Nevot
mmartin.nevot@gmail.com

Carte de visite électronique

## Relecteurs

Cours en ligne sur : www.mickael-martin-nevot.com