

Java

CM2 : Programmation orientée objet

Mickaël Martin Nevot

V2.2.0

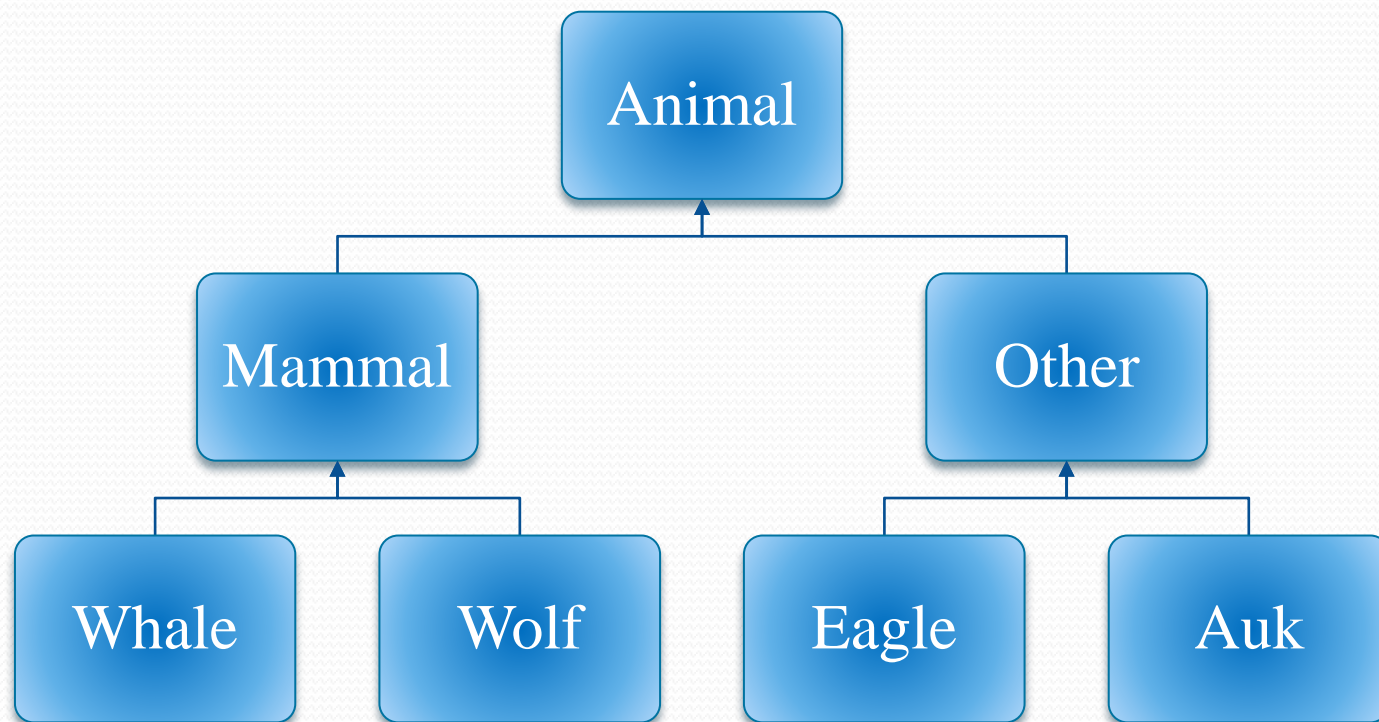


Cette œuvre de [Mickaël Martin Nevot](#) est mise à disposition selon les termes de la [licence Creative Commons Attribution - Pas d'Utilisation Commerciale - Partage à l'Identique 3.0 non transposé](#).

Java

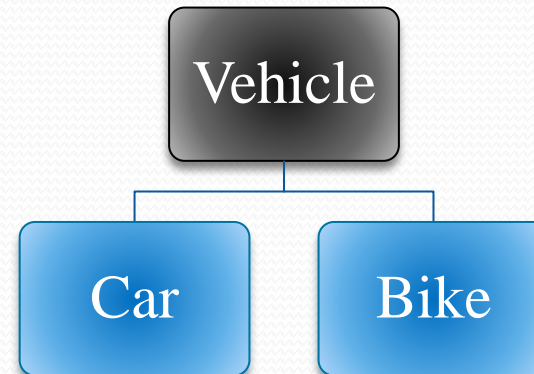
- I. Prés.
- II. POO
- III. **Objet**
- IV. Java
- V. Types
- VI. Héritage
- VII. Outils
- VIII. Exceptions
- IX. Polymorphisme
- X. Thread
- XI. Avancé

Héritage



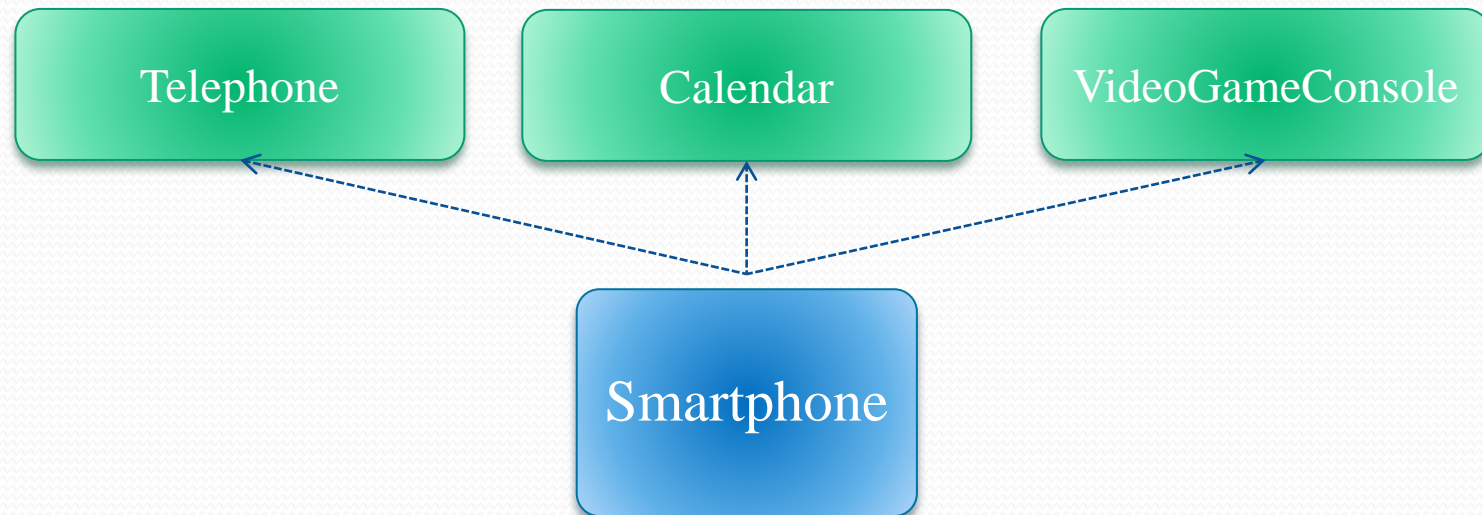
Abstraction

- 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**

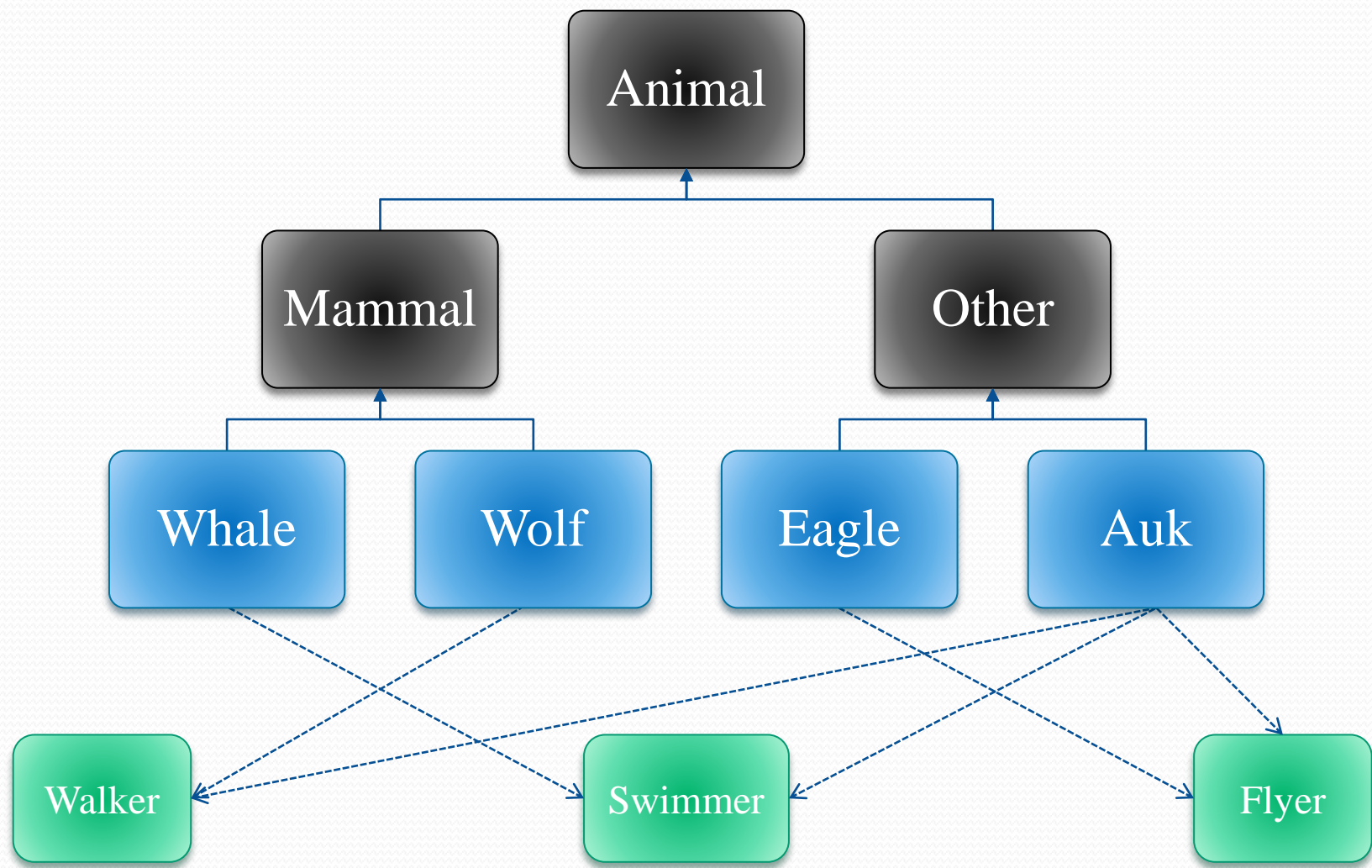


Interface

- 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



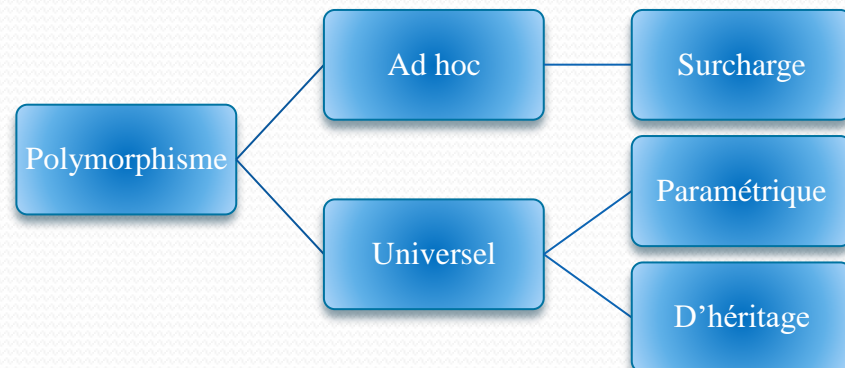
Héritage



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

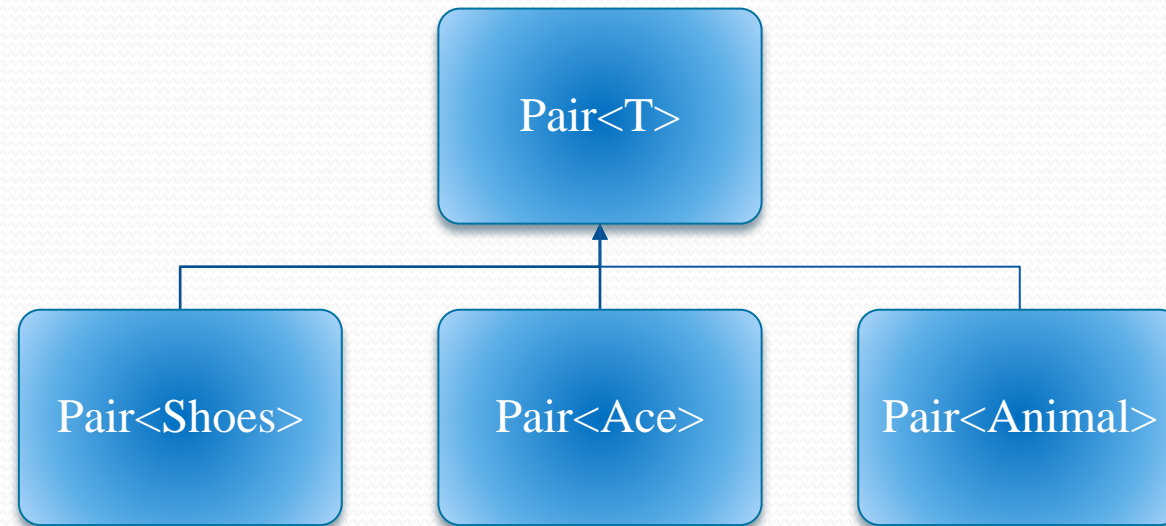


- **Surcharge :**
 - Permet d'avoir des méthodes de même nom, avec des fonctionnalités similaires
- **Paramétrique :**
 - Permet de définir plusieurs méthodes de même nom mais possédant des paramètres différents
- **D'héritage :**
 - Permet de redéfinir une méthode dans des classes héritières (l'appel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



Module et paquetage

- Regroupement de codes sources en **unités de travail logiques**
- **Encapsulation**
- **Réutilisabilité** et partage du code facilités
- Facilitation de réalisation de bibliothèques
- **Généricité** possible
- Paquetage : organisation sémantique de classes en répertoires



Couplage et cohésion

- **Couplage :**

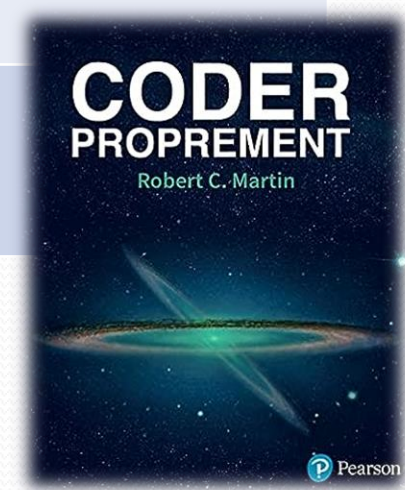
- Interconnexion de deux classes (modules)
- **Couplage fort** : une modification d'une classe nécessite la modification de l'autre
- **Couplage faible** : communication de données uniquement grâce à des « interfaces »

- **Cohésion :**

- Relations des données au sein d'une même classe
- Centré sur un but : se concentre sur tâche unique et précise

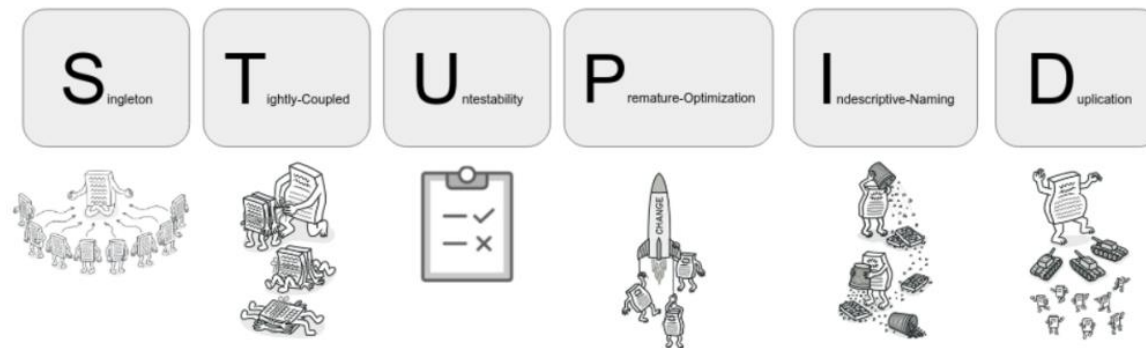
S.O.L.I.D

S	Responsabilité unique (<i>single responsibility principle - SRP</i>)	Une classe doit avoir une et une seule responsabilité
O	Ouvert/fermé (<i>open/closed principle - OCP</i>)	Une classe doit être ouverte à l'extension, mais fermée à la modification
L	Substitution de Liskov (<i>Liskov substitution principle - SLP</i>)	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 (<i>interface segregation principle - ISP</i>)	Préférer plusieurs interfaces spécifiques pour chaque client plutôt qu'une seule interface générale
D	Inversion des dépendances (<i>dependency inversion principle - DIP</i>)	Il faut dépendre des abstractions, pas des implémentations



S.T.U.P.I.D

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



Loi de Déméter

- 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



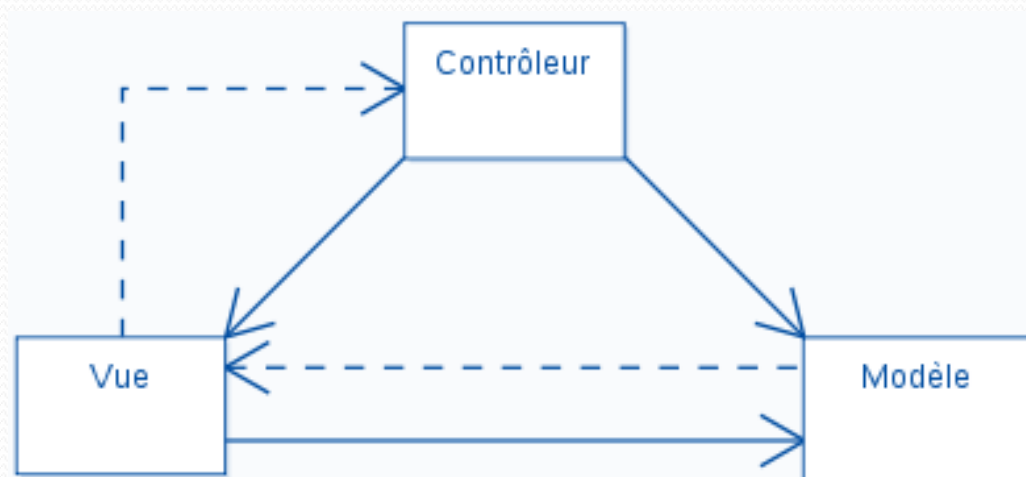
Loi de Démeter

- 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

Modèle-Vue-Contrôleur

- **Architecture et méthode de conception** d'IHM
- **Modèle** : données et leur manipulation
- **Vue** : élément de l'interface graphique
- **Contrôleur** : orchestre les actions, synchronise
- **MVC 2** : un seul contrôleur



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

Liens

- Documents électroniques :

- <http://hdd34.developpez.com/cours/artpoo/>
- <http://fabrique-jeu-video.blogspot.fr/2013/08/poo.html>

- Document classique :

- Hugues Fauconnier. *Programmation orientée objet en Java.*
- Guillaume Revy. *Introduction à la Programmation Orientée Objet... et son application au C++.*
- Craig Larman. *UML2 et les design patterns.*
- Mathieu Nebra. *La programmation orientée objet.*

Crédits

Auteur

Mickaël Martin Nevot

mmartin.nevot@gmail.com



Carte de visite électronique

Relecteurs

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

