

# Java

CM1 : Introduction aux objets

Mickaël Martin Nevot

V1.1.1



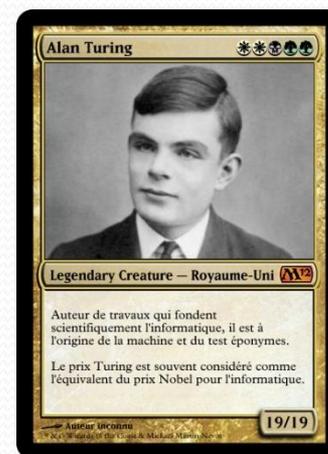
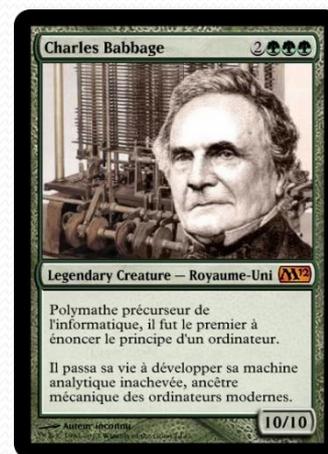
Cette œuvre est mise à disposition selon les termes de la [licence Creative Commons Attribution - Pas d'Utilisation Commerciale - Partage à l'Identique 3.0 non transposé](https://creativecommons.org/licenses/by-nc-sa/3.0/).

# Java

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

# Programmation : historique

- IX<sup>e</sup> siècle : Al Khuwarizmi (algorithmique)
- 1842 : Charles Babbage et Ada Lovelace (programmation)
- 1847 : Boole (logique)
- 1936 : Turing (machine de)



# Programmation : historique

- **1954** : système d'exploitation et Fortan
- **1959** : Grace Murray Hopper (compilateur/bogue)
- **1960 - 1980** : cartes perforées
- **1970** : programmation structurée
- **1980 – 1990** : programmation orientée objet (POO)



**John von Neumann** (4 icons)

Legendary Creature — Hongrie (A12)

Genie moderne ayant donné son nom à l'architecture de von Neumann utilisée dans la quasi-totalité des ordinateurs modernes.

Il est aussi le premier à envisager la notion de singularité technologique.

La médaille John von Neumann est nommée en son honneur.

— Alan Richards (16/16)



**Grace Hopper** (3 icons)

Legendary Creature — USA (A12)

Vice-amiral de la marine américaine.

Conceptrice du premier compilateur et du langage de programmation COBOL.

On lui attribue à tort le terme de bogue bien qu'elle soit effectivement à l'origine du concept de débogage.

— Courtesy of Vassar (12/12)



**Dennis Ritchie** (5 icons)

Legendary Creature — USA (A12)

Codéveloppeur d'Unix.

Cocréateur du langage de programmation C.

— Astor Urciano (12/12)



**Brian Kernighan** (4 icons)

Legendary Creature — Canada (A12)

Cocréateur du langage de programmation C.

Il imagina également les heuristiques pour les problèmes du partitionnement de graphe et du voyageur de commerce.

— Ben Lowe (11/11)



**Barbara Liskov** (3 icons)

Legendary Creature — USA (A12)

Elle a posé les bases de la programmation orienté objet (POO).

Elle a notamment donné son nom au principe de substitution de Liskov utilisé en POO.

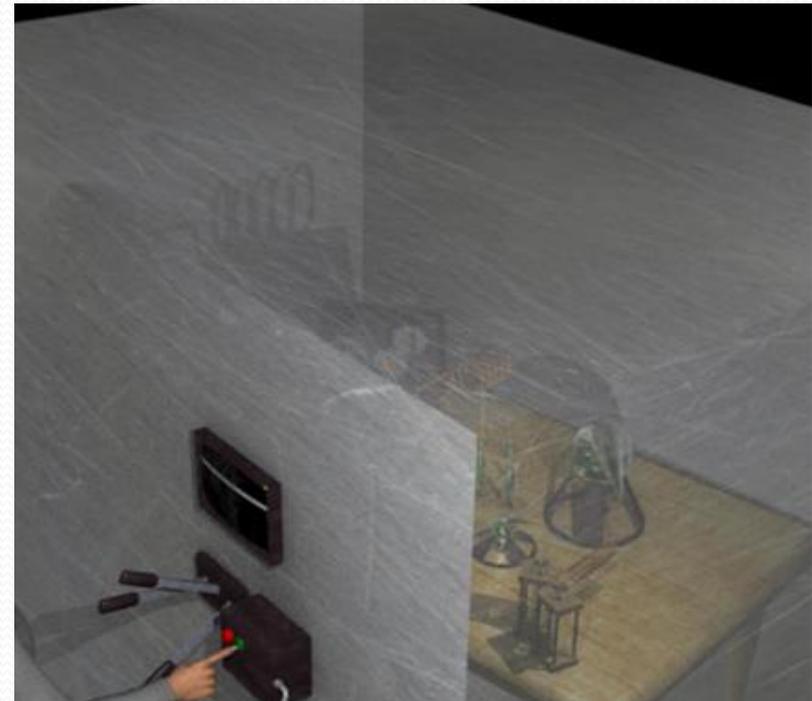
— Donna Coveney (10/10)

# Qu'est-ce qu'un objet ?

## Code source traditionnel



## Code source orienté objet



# Qu'est-ce qu'un objet ?

## Code source traditionnel



## Code source orienté objet

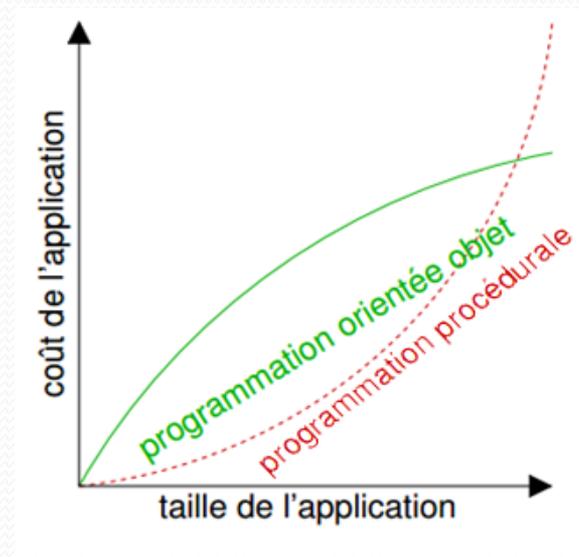


*« Un objet est une capsule logicielle oblatrice avec un tropisme conatif dont l'hétéronomie est la marque de la durée, de l'éphémère et de la hoirie »*

— Serge Miranda

# Pourquoi la POO ?

- **Diminuer le coût** d'un logiciel
- Faciliter la **conception** et l'**évolution** du code
- **Augmenter la durée de vie** d'un logiciel
- Augmenter la **réutilisabilité** d'un logiciel
- Augmenter la **facilité de maintenance** d'un logiciel



Conception d'un logiciel à la manière de la fabrication d'une voiture

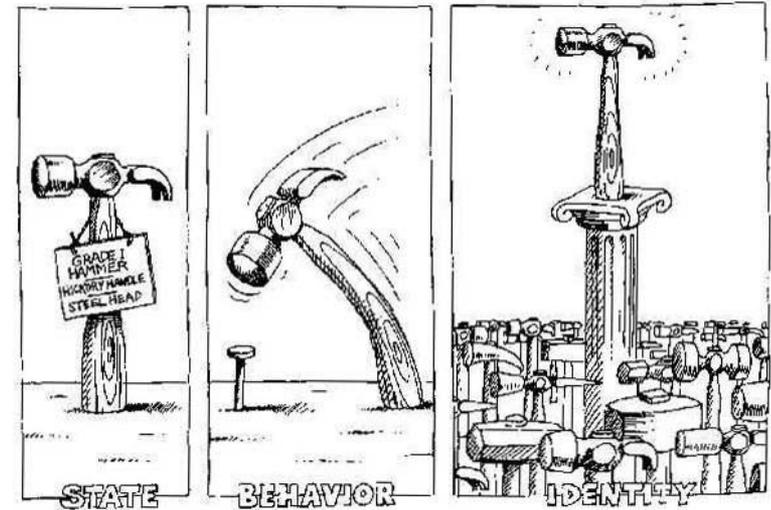
# Principe de la POO

- Communication inter-objets par messages
- À la réception d'un message :
  - Appel d'une méthode qui modifie son état
  - Appel d'une méthode qui envoie un message à son tour



# Programmation objet

- **Paradigme** ( $\neq$  méthodologie)
- **Concept** :
  - **Objet** :
    - **État** (attributs)
    - **Comportement** (méthodes)
    - **Identité**
  - **Encapsulation**
  - **Héritage**
  - **Polymorphisme**
- **Langage de programmation** :
  - **Java**, C++, Ada, PHP, C#, Objective C, Python, etc.



# Objet et classe

- **Objet** : un concept, une idée/entité du monde physique
  - Exemples : voiture, étudiant, chat, fenêtre, forme, etc.
- **Classe** : regroupe les objets de mêmes comportements
- **Instancier** : fabriquer un exemplaire d'un élément à partir d'un modèle qui lui sert en quelque sorte de moule
- Un objet est une instance de **classe**
- **Réification** : permet de transformer ou à transposer un concept en une entité informatique

Une classe représente une responsabilité

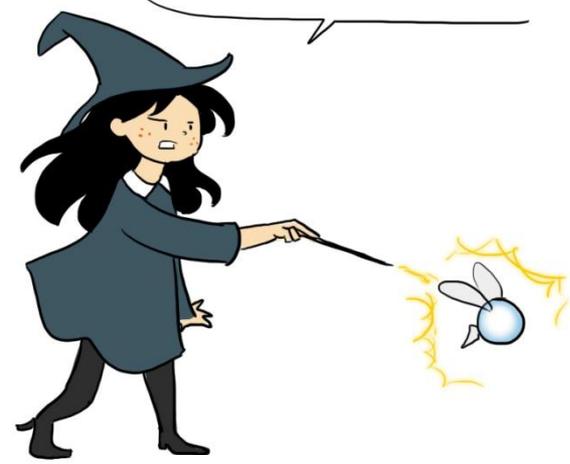
# État et comportement

- État :
  - Défini par l'ensemble des attributs
  - **Attribut**, variable d'instance, donnée membre :
    - Variable spécifique à l'objet
- Comportement :
  - Défini par l'ensemble de méthodes
  - **Méthode** : fonctions spécifiques à l'objet
  - Méthodes :
    - **Constructeur** : appelé à la création de l'objet
    - **Destructeur** : appelé à la destruction d'un objet
    - Méthode abstraite : méthode sans code
    - Accesseurs et mutateurs : sert de mandataire d'accès à l'état de l'objet (depuis l'extérieur de celui-ci)



# POO et magie

```
var navi : Fée = new Fée();
```



```
var taille : Number = 80 cm;  
var densitéPoils : Number = 67 %;  
var poids : Number = 26 kg;  
var paroles : String = "Hey!"
```

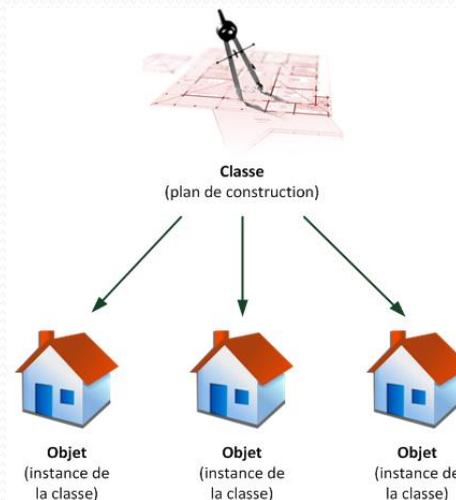


```
fonction voler () : void  
{  
  battre des ailes ;  
  se déplacer ;  
  if (rencontre un mur)  
  {  
    faire demi-tour ;  
  }  
}
```



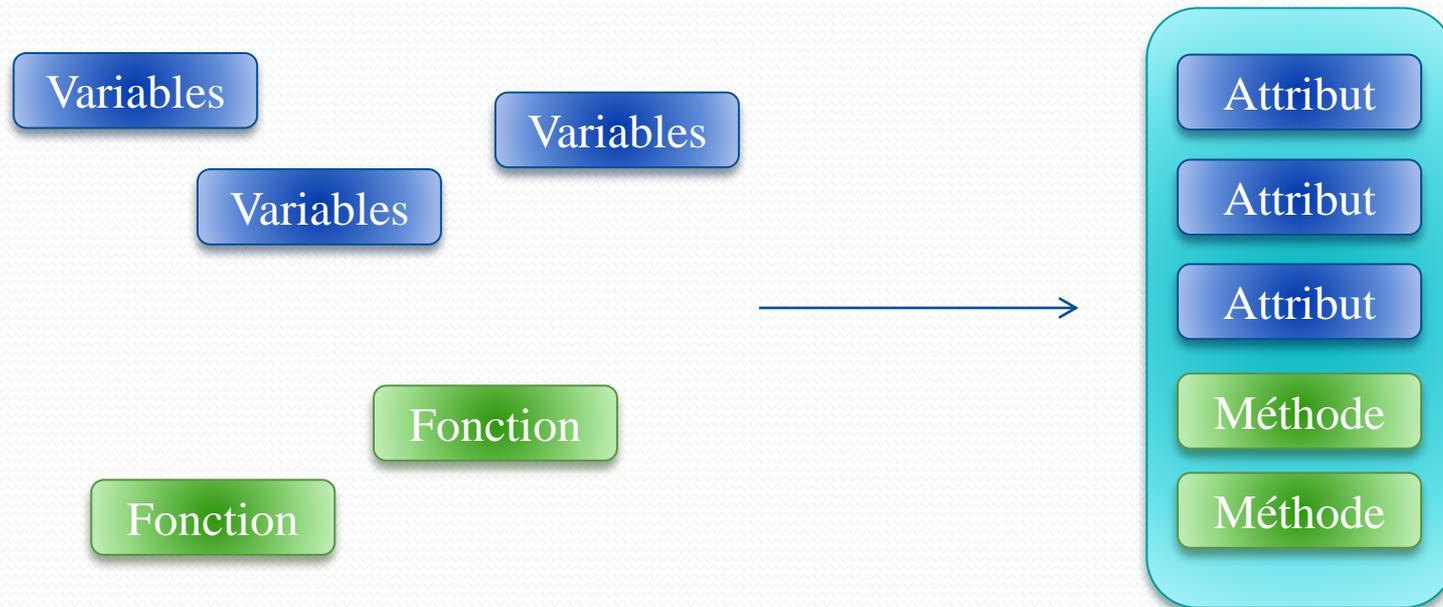
# POO : deux aspects

- Programme en cours d'écriture :
  - Ensemble de **classes**
  - Chaque classe a des **attributs** et des **méthodes**
- Programme en cours d'exécution (processus) :
  - Ensemble d'**objets**
  - Chaque **objet** a son **état** courant et un **comportement**

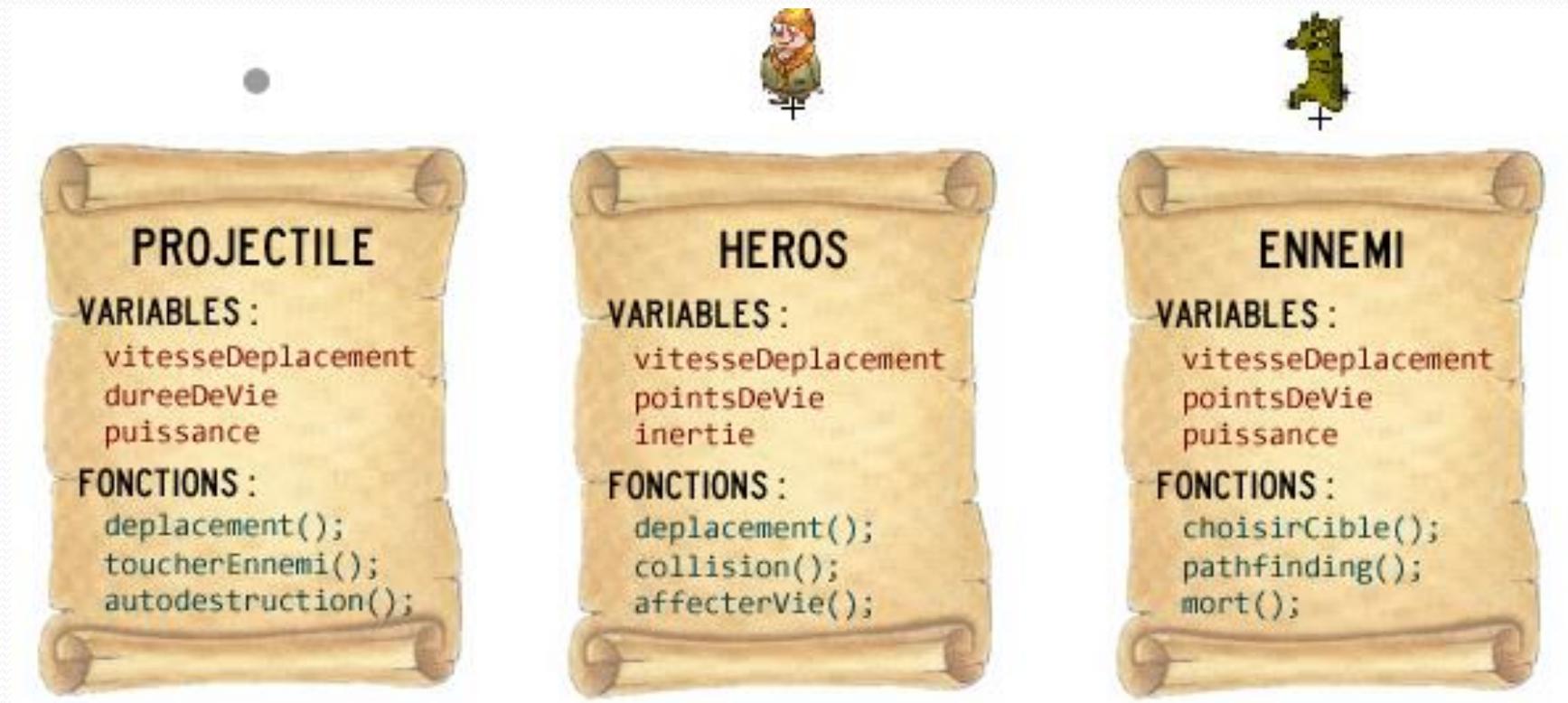


# Encapsulation

- Association de variables et fonctions dans une même entité
- L'objet est vu de l'extérieur comme une boîte noire ayant certaines propriétés et ayant un comportement spécifié



# POO et magie

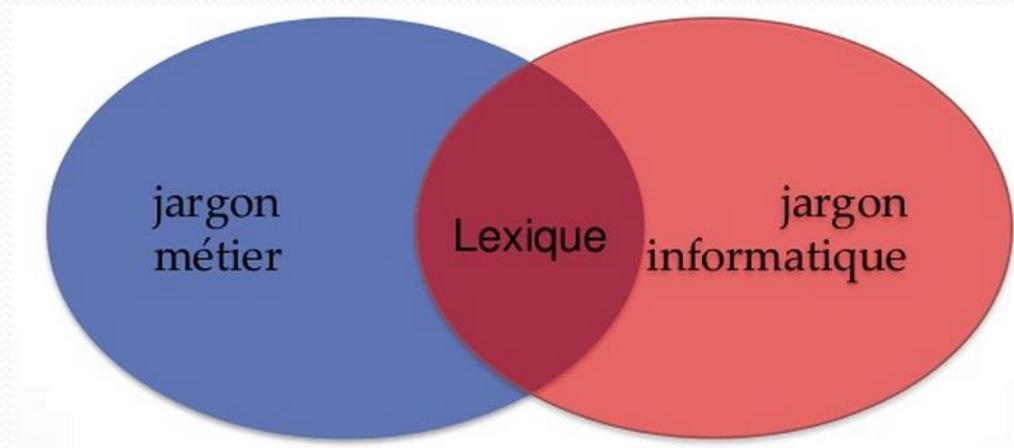


# Encapsulation : accessibilité

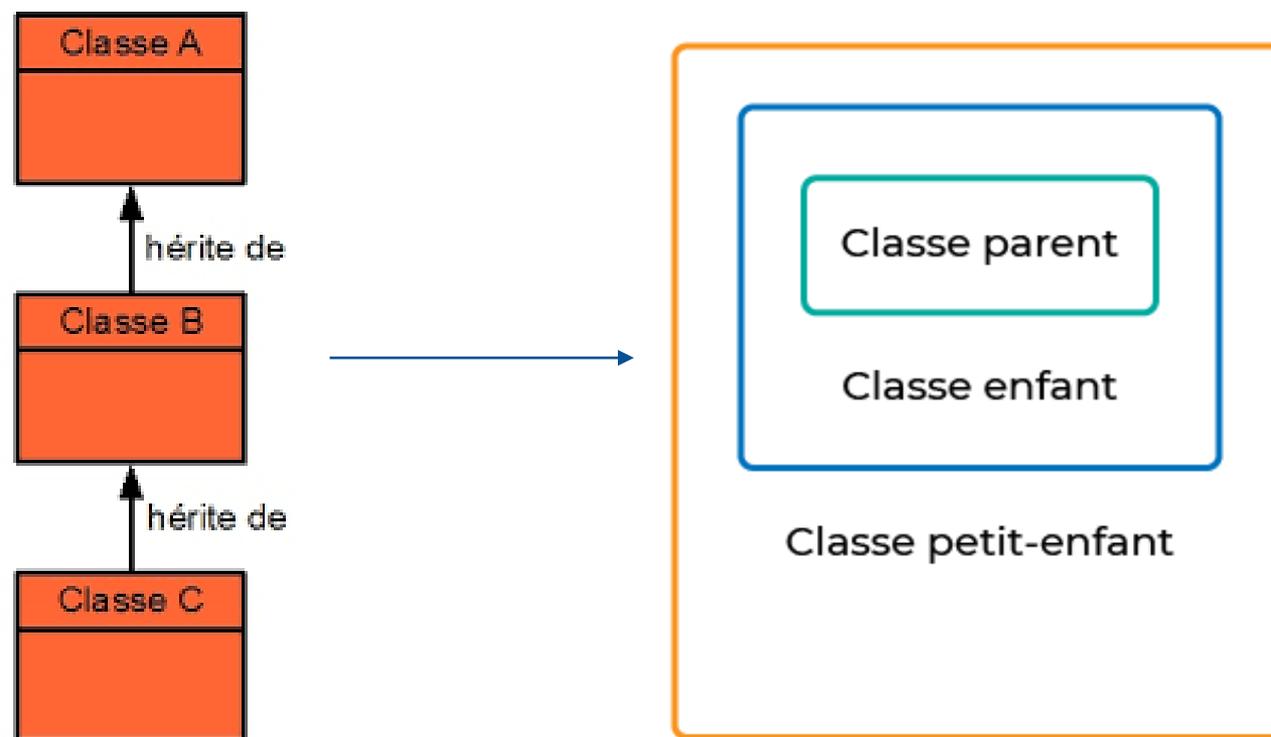
- Concerne : classe, constructeur, attribut et méthode
- Visibilité :
  - **Publique** : accessible de partout et sans aucune restriction
  - **Protégée** : accessible aux classes du module (ou paquetage) et à ses classes filles
  - **Privée** : accessible uniquement au sein de sa classe
- Accesseurs (*getters*) / mutateurs (*setters*) :
  - Permet de récupérer/modifier la valeur d'un attribut avec une visibilité restreinte grâce à une méthode

# Typologie des objets

- Objets **techniques** qui rendent des services
- Objets **métiers** qui implémentent les aspects métiers du système (s'appuie sur un lexique) :
  - Aspect pragmatique
  - Aspect sémantique
  - Aspect logique

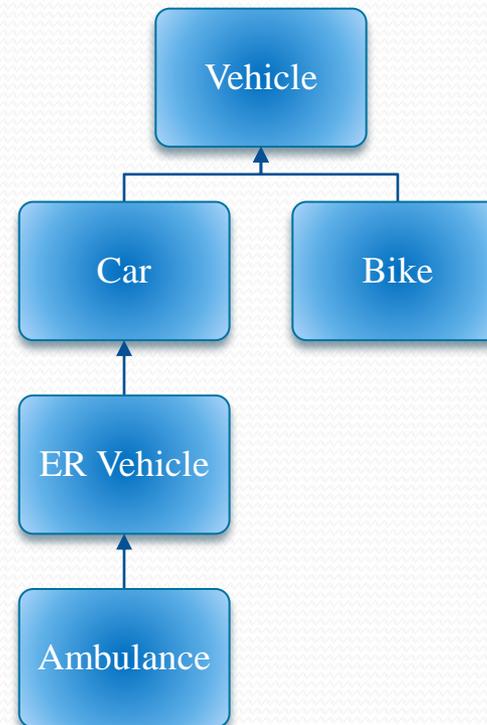


# Héritage



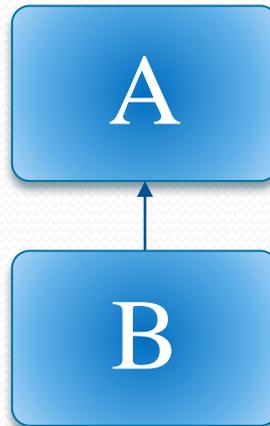
# Héritage

- Construction d'une classe à partir d'autres classes en partageant leurs attributs et méthodes
- **Spécialisation** (ou généralisation), **enrichissement**
- **Réutilisation**
- **Redéfinition**



# Classes et sous-classes

- B **hérite** de A
- A est la **classe mère**, B la **classe fille**
- A est la **super-classe** de B
- B est une **sous-classe** de A
- Un objet de type B **est** aussi un objet de type A
- Un objet de type A **n'est pas** un objet de type B



# Notions générales

- **Opérateurs**
- **Données :**
  - **Variables  $\neq$  attributs**
  - Constantes
  - **Valeurs littérales** ("hello!", 3, 2.75, true)
  - Expressions (et expressions « parenthésées »)
- **Affectation  $\neq$  condition**
- **Instructions**
- **Fonction  $\neq$  procédure  $\neq$  méthode** (signature)
- **Paramètres** (formels)  $\neq$  **arguments** (paramètres effectifs)
- **Exécution séquentielle**

# Aller plus loin

- Prototype
- Mixin
- Trait

# Crédits

## Auteur

Mickaël Martin Nevot

[mmartin.nevot@gmail.com](mailto:mmartin.nevot@gmail.com)



Carte de visite électronique

## Relecteurs

Cours en ligne sur : [www.mickaël-martin-nevot.com](http://www.mickaël-martin-nevot.com)

