La POO démystifiée

Introduction aux Objets

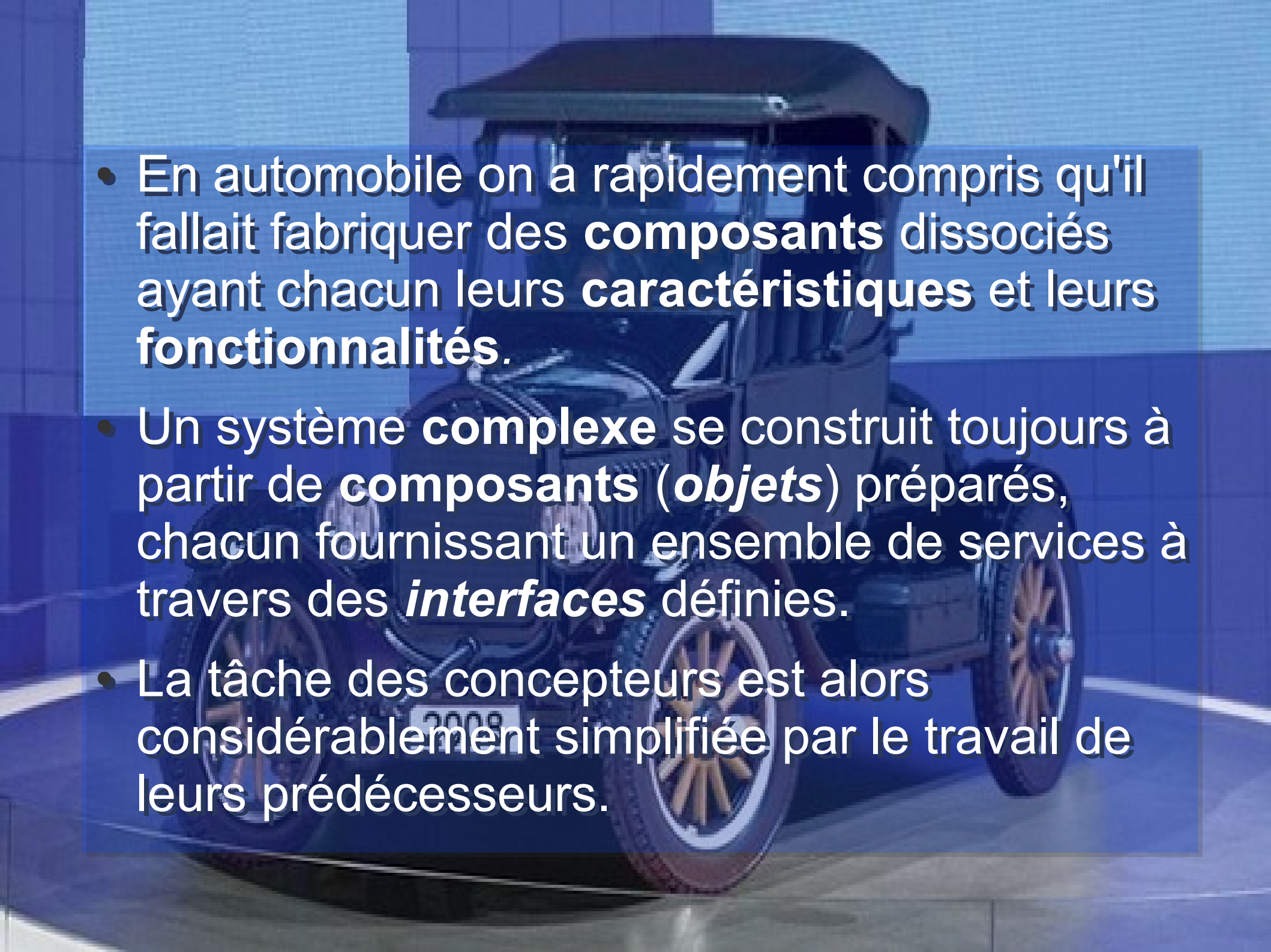
„Aujourd'hui quasiment tous les langages permettent une approche orientés objets. Une connaissance minimale des principes de la POO est donc indispensable à tout informaticien, qu'il soit développeur ou non.“

Dominique CARON

Oct 2014

Sur la plage, quand on est petit, on construit des grands chateaux avec du sable...

Mais devenu plus grand nous viendrait t'il à l'idée de construire une maison à partir de tas de sable, de cailloux et de terre?



- En automobile on a rapidement compris qu'il fallait fabriquer des **composants** dissociés ayant chacun leurs **caractéristiques** et leurs **fonctionnalités**.
- Un système **complexe** se construit toujours à partir de **composants** (***objets***) préparés, chacun fournissant un ensemble de services à travers des ***interfaces*** définies.
- La tâche des concepteurs est alors considérablement simplifiée par le travail de leurs prédécesseurs.

Cas du logiciel: les limites de la programmation procédurale

- La programmation procédurale montre ses limites dans les projets complexes (plus de 10 000 lignes de codes)
- Le code procédurale est difficile à débugguer, difficile à tester et donc moins fiable.
- La qualité du programme se dégrade dans le temps. Il devient très difficile voir impossible de faire des mises à jour intégrant de nouvelles fonctionnalités.
- La reprise du code par d'autres développeurs est problèmatique.

Naissance de la POO

- Les concepts de la Programmation Orientée Objet naissent dans les labos de recherche des années 70.
- Simula (1967), SmallTalk(1973) sont les premiers langages objet.
- Les années 80 voient naître le C++ (extension du C) et ObjectiveC.
- Java apparait dans les années 90 ...

Définir un objet du monde réel?

- Par ses caractéristiques (taille, poids, nom, marque, longueur, vitesse, force, age, couleur, position etc ...) qu'on appelera ***attributs***.

*Note: un **attribut** d'**objet** peut être un autre **objet** (ex: force, position).*

- Par ses fonctions ou actions possibles (tourne, vole, chauffe, mange, grandit etc...) qu'on appelera ***méthodes***.

*Note: Les **méthodes** peuvent agir sur les **attributs** de l'objet et/ou renvoyer des **messages**.*

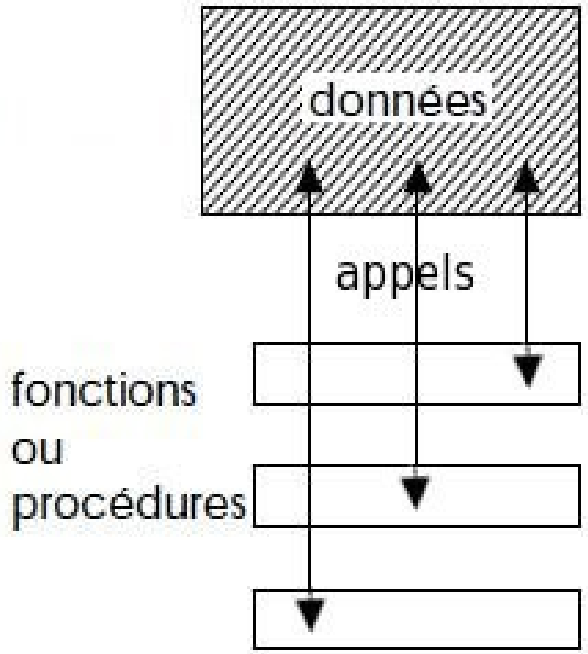
- Définir un objet du monde réel dans son ensemble peut s'avérer impossible (les objets sont trop complexes)

- La ***modélisation*** objet consiste à créer une représentation informatique des **éléments du monde réel auxquels on s'interesse**.

  - Il s'agit donc de déterminer les objets présents dans **notre problème** et d'isoler les caractéristiques (***attributs***) et les fonctions (***méthodes***) **minimales** permettant de le résoudre.

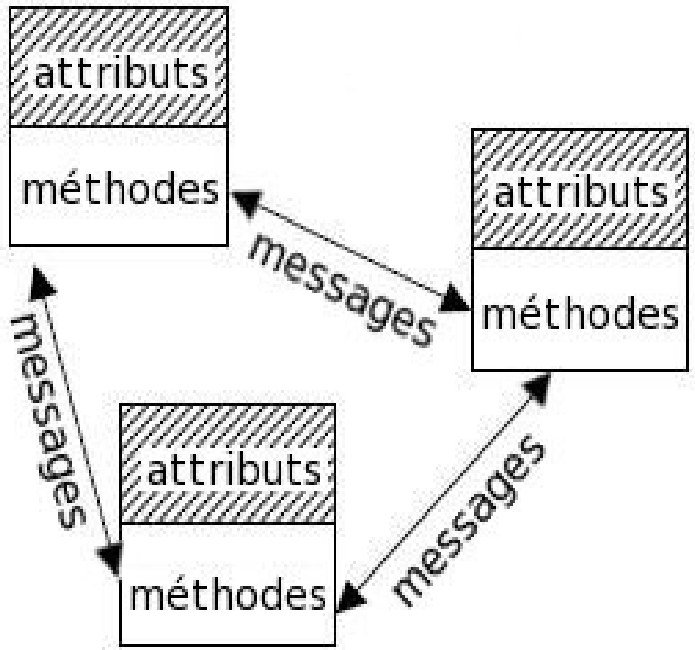
Une nouvelle approche des problèmes

Programmation procédurale



Que doit faire le programme?

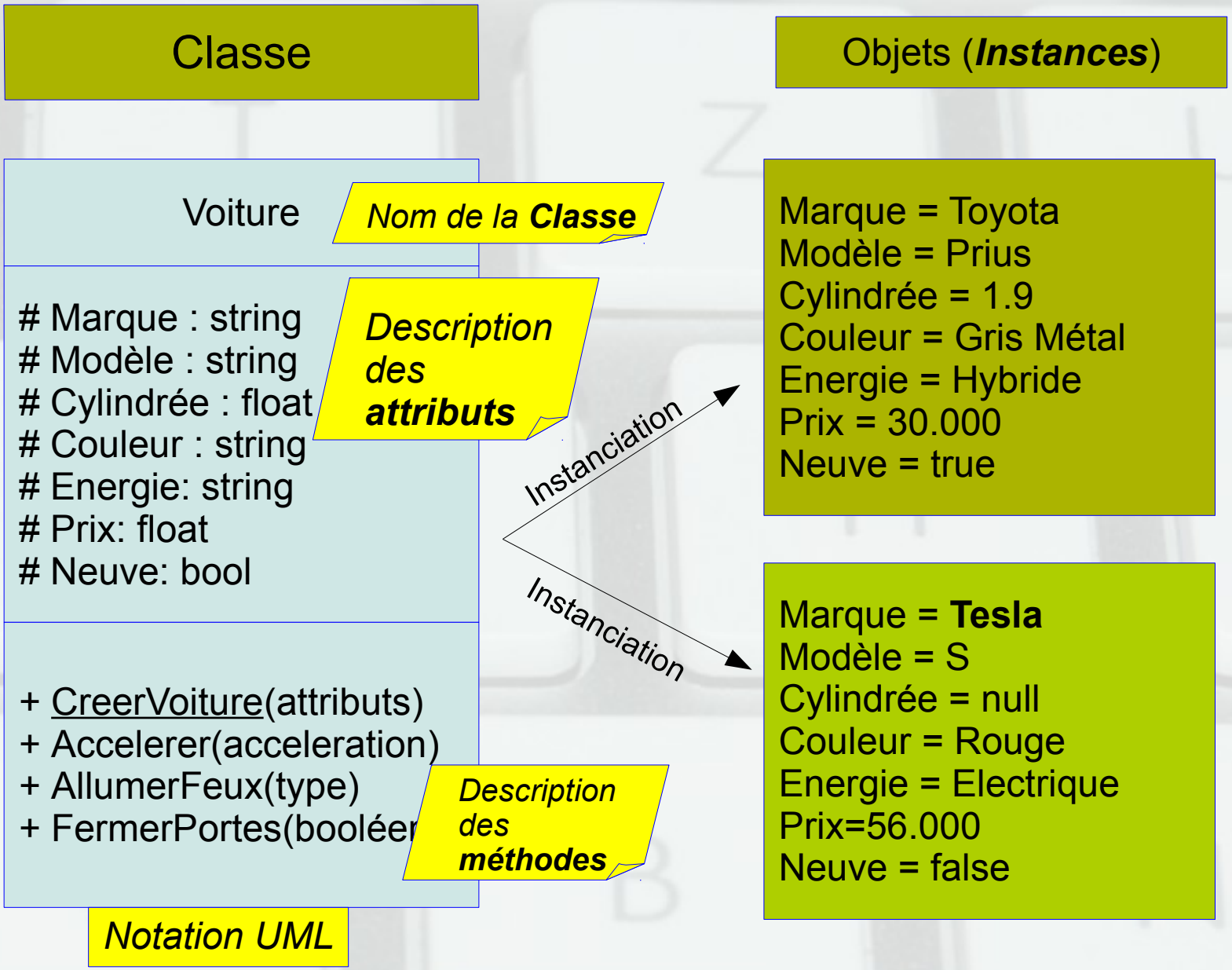
Programmation objet



Sur quoi porte le programme?

Classons les objets....

- La description d'un objet est appelée une ***classe*** d'objet.
- On dit qu'un ***objet*** est une ***instance*** de la ***classe*** ou un ***état*** de celle-ci. ***L'instanciation*** d'une ***classe*** est un ***objet*** **unique**.
- La ***classe*** décrit donc la structure interne d'un ***objet***, c'est à dire: les données qu'il regroupe (***attributs***) et les actions qu'il est capable d'assurer (***méthodes***).

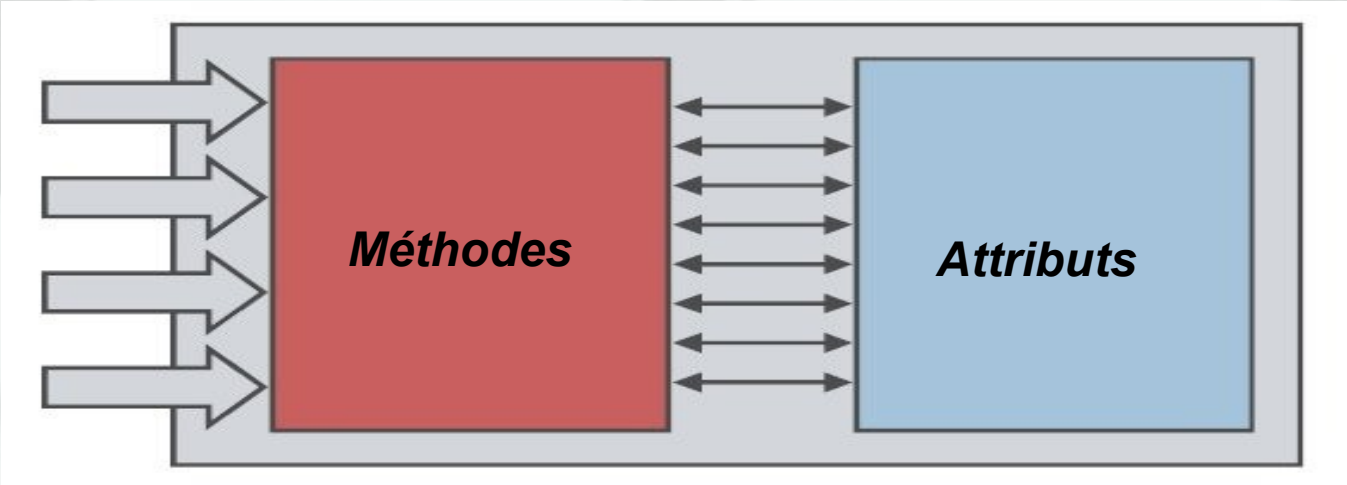

Classe
Objets (*Instances*)
Voiture
*Nom de la **Classe***
Marque : string
Modèle : string
Cylindrée : float
Couleur : string
Energie: string
Prix: float
Neuve: bool
*Description des **attributs***
+ CreerVoiture(attributs)
+ Accelerer(acceleration)
+ AllumerFeux(type)
+ FermerPortes(booléen
*Description des **méthodes***
Notation UML
Instanciation
Instanciation
Marque = Toyota
Modèle = Prius
Cylindrée = 1.9
Couleur = Gris Métal
Energie = Hybride
Prix = 30.000
Neuve = true
Marque = **Tesla**
Modèle = S
Cylindrée = null
Couleur = Rouge
Energie = Electrique
Prix=56.000
Neuve = false

Les trois grands paradigmes de la POO

- L'Encapsulation
- L'Héritage
- Le Polymorphisme

1) L'Encapsulation

- Le principe est d'interdire l'accès direct aux ***attributs*** d'un objet. On ne dialogue avec l'objet qu'à travers une interface définissant les services accessibles aux utilisateurs de l'objet.
- C'est alors le rôle des ***méthodes*** de modifier ou de retourner la valeur des ***attributs***.



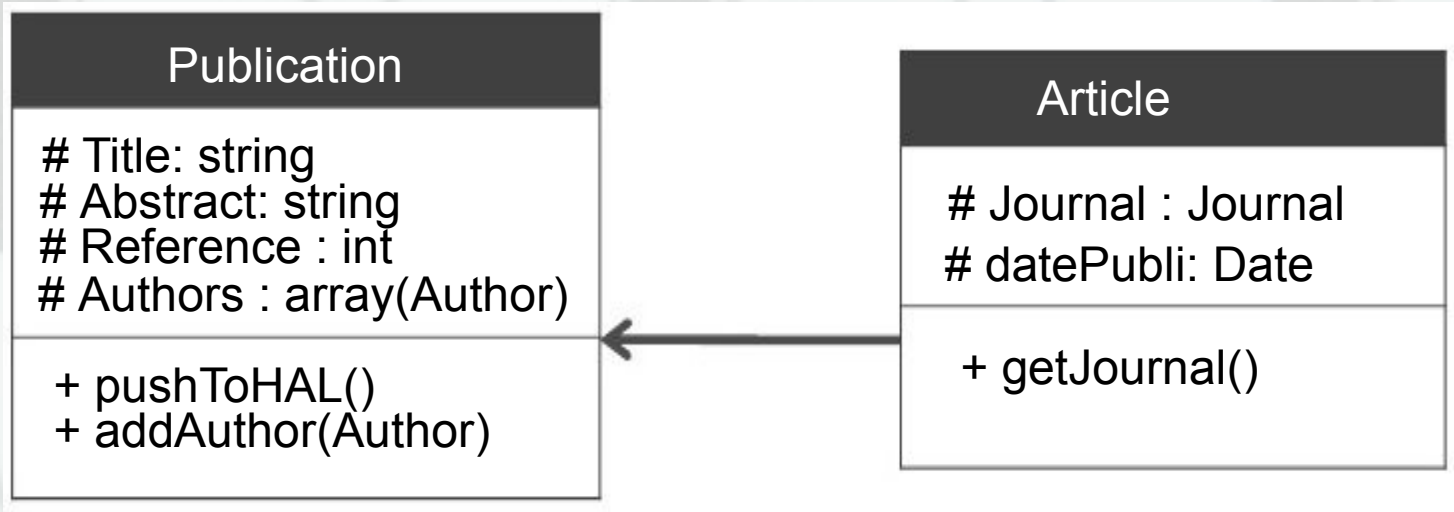
Intérêt de l'encapsulation

Facilite l'évolution d'une application

- 1. On peut modifier les ***attributs*** d'un objet sans modifier la façon dont il est utilisé.
- 2.Garantie l'intégrité des données, car leur accès direct est interdit (limité et/ou contrôlé)

2) L'héritage

- Relation de ***spécialisation/généralisation*** entre deux classes. Elle indique qu'une classe est une sous-classe d'une autre, c.à.d qu'elle possède les mêmes attributs et méthodes **plus d'autres** qui lui sont propres.



Intérêt de l'héritage

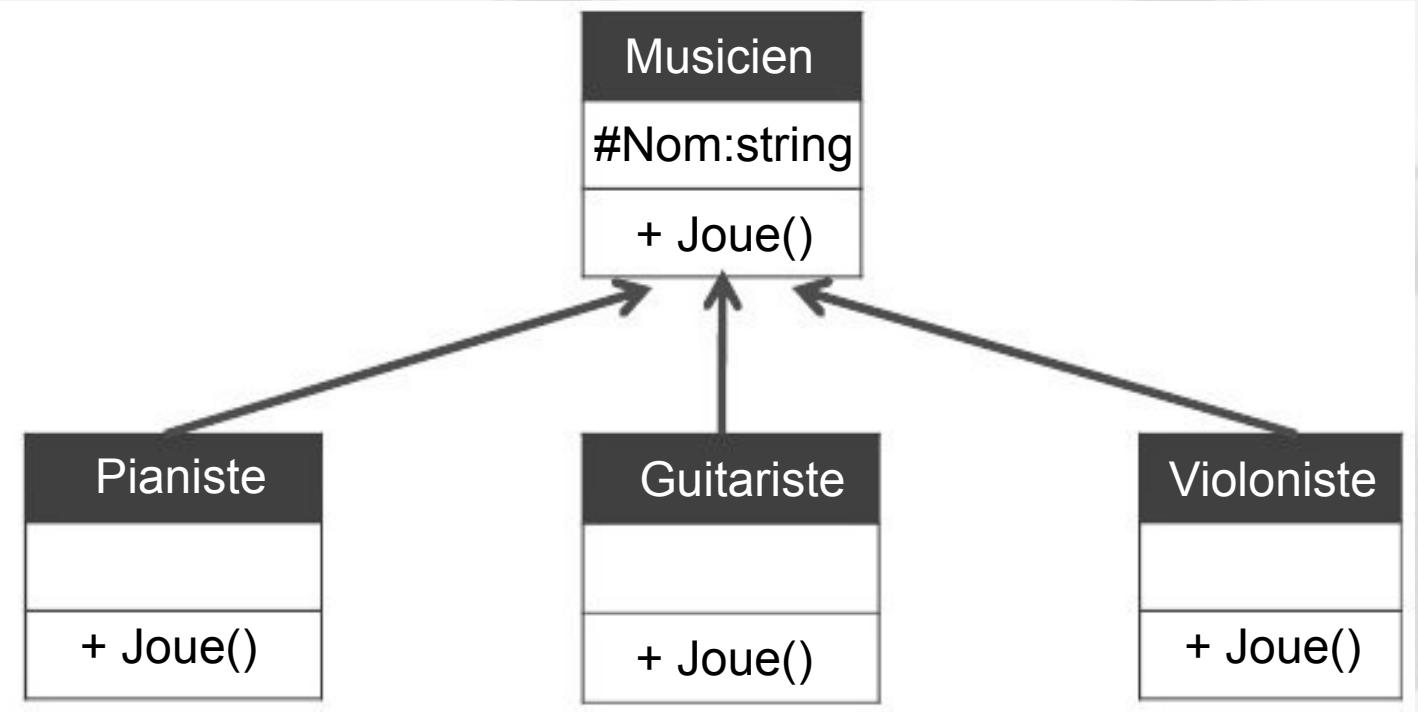
- En construisant une hiérarchie de classe, on évite des répétitions dans le code (factorisation), en encourageant la réutilisation de classes déjà existantes.
- Cela permet également de simplifier la conception de la modélisation.
- Héritage multiple : Tous les languages ne le permettent pas et il n'est pas conseillé. Il peut introduire des bugs difficiles à décelés.

3) Polymorphisme

- Littéralement : c'est la faculté de prendre plusieurs formes.

- En POO, c'est un mécanisme permettant à une ***sous-classe*** de redéfinir une ***méthode*** dont elle a ***hérité*** **tout en gardant la même signature**.

Polymorphisme



L'appel de la ***méthode*** Joue() sur les objets ***héritant*** de la ***classe*** Musicien produira un résultat différent selon la ***sous-classe***.
Intérêt: permet une meilleur factorisation du code (généricité).

Les Classes Abstraites

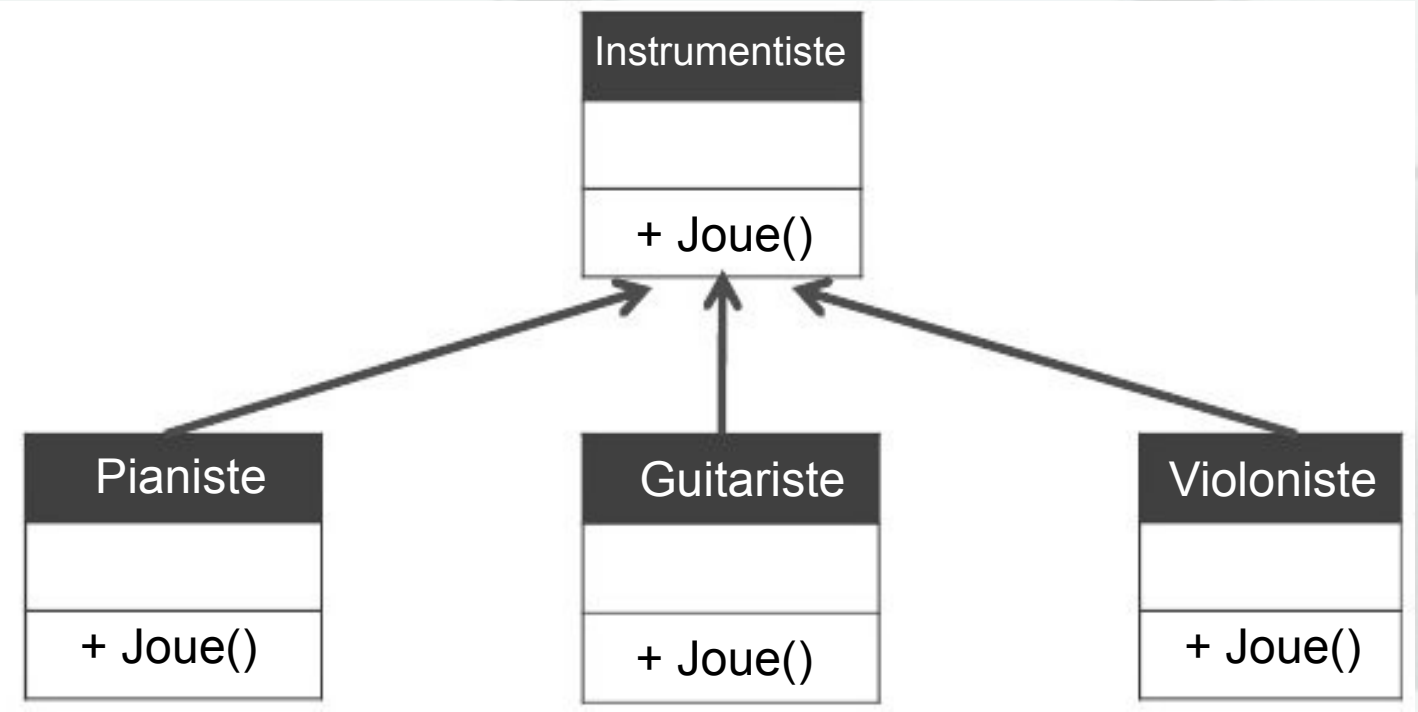
- Une ***classe abstraite*** est une classe dont les ***méthodes*** n'ont pas toutes été implémentées.
- Elle n'est donc pas instanciable (*devenir objet*) mais sert avant tout à factoriser du code.
- Une classe qui ***hérite*** d'une classe abstraite doit obligatoirement implémenter les ***méthodes*** manquantes (elles sont déclarées « abstraites » dans la classe parente). En revanche, elle n'est pas obligée de réimplémenter les ***méthodes*** déjà implémentées dans la classe parente (d'où une maintenance du code plus facile).

Les Interfaces

- Une ***interface*** est une ***classe*** destinée à définir des ***méthodes* publiques** des ***classes*** qui l'implémente (API). **Aucune *méthode*** n'est implémentée

- Une ***interface*** regroupe la **signature des *méthodes*** qui devront être implémentées par les ***classes*** l'implémentant. C.à.d : en **implémentant** une ***interface***, une ***classe*** s'oblige à définir l'ensemble des ***méthodes*** de ***l'interface***.
- On peut implémenter plus d'une ***interface*** par ***classe***, à condition que celles-ci n'aient aucune ***méthode*** portant le même nom.

Interfaces



L'appel de la ***méthode*** Joue() sur les objets ***héritant*** de la ***classe*** Musicien produira un résultat différent selon la ***sous-classe***.

Intérêt: permet une meilleur factorisation du code (généricité).

- Références

Introduction aux objets - Ours Blanc Des Carpathes - http://www.mcours.net/cours/pdf/info/Cours_introduction_aux_objets.pdf

Introduction à la modélisation objet - Laurent Godefroy - http://fr.scribd.com/doc/77622472/01-Introduction-a-la-modelisation-objet

Les Questions...