Programmation orientée objet

Cours 4 – Héritage

Attributs de classe

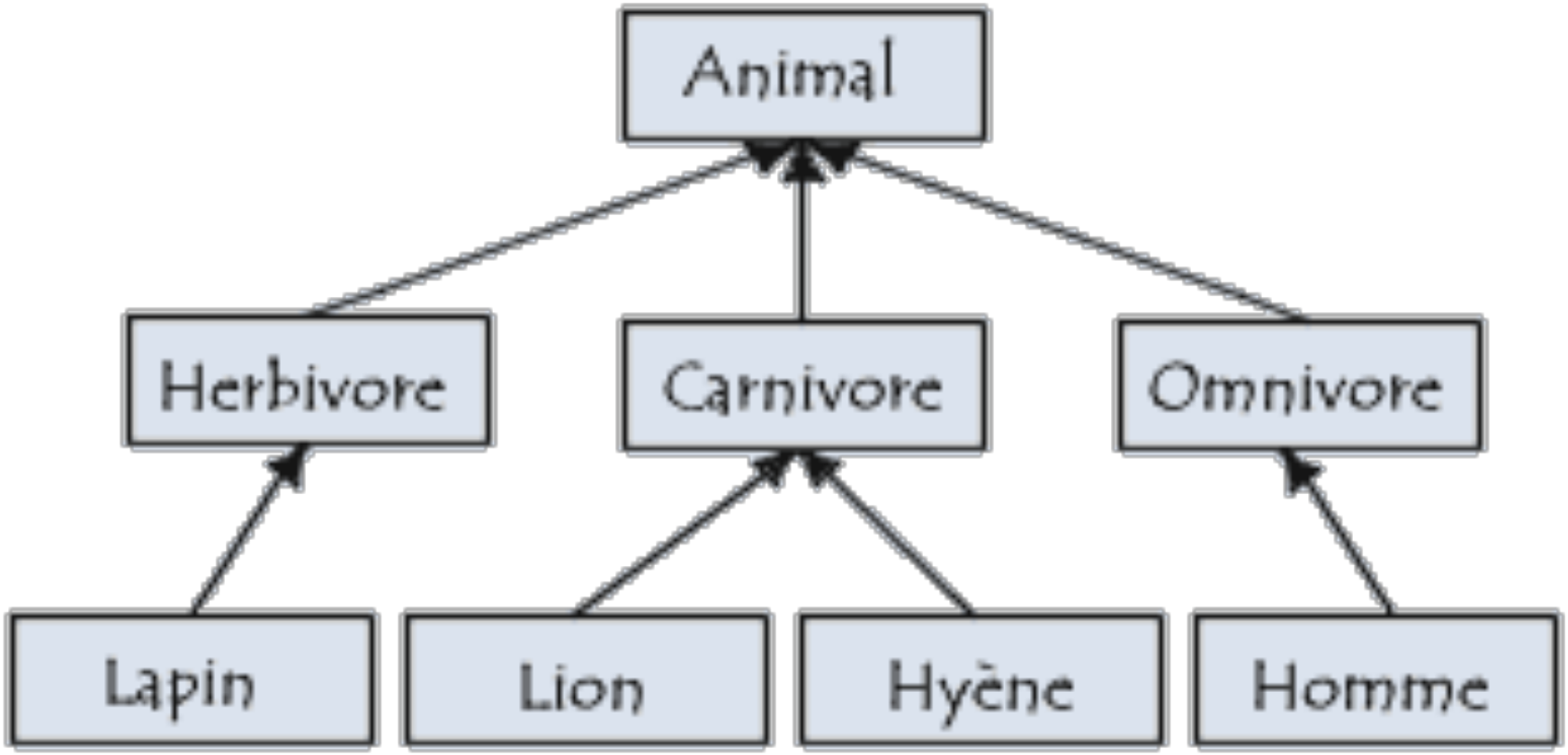
Héritage

- Parmi les raisons d'exister de la programmation orientée-objet, on retrouve la possibilité de réutiliser du code.
  - Pour rendre du code réutilisable, on doit essayer de le rendre le plus générique possible pour que l'on puisse ensuite l'utiliser dans une multitude de situations.
- Pour répondre à ce besoin en programmation orientée-objet, le concept d'héritage a été amené.
  - L'héritage permet de créer une nouvelle classe à partir d'une classe existante, ce qui permet de faire des classes plus génériques qui serviront plus tard de base à des classes plus spécifiques!

Fonctionnement de l'héritage en POO

- L'héritage permet donc de spécifier, lorsque l'on crée une classe, que cette classe "hérite" d'une autre classe.
  - Lorsque la classe B hérite de la classe A, on peut dire que la classe A est la "classe mère" ou la "classe de base" de la classe B.
  - On dit de la classe B qu'elle est une "sous-classe" de la classe A, ou une "classe fille" de la classe A.

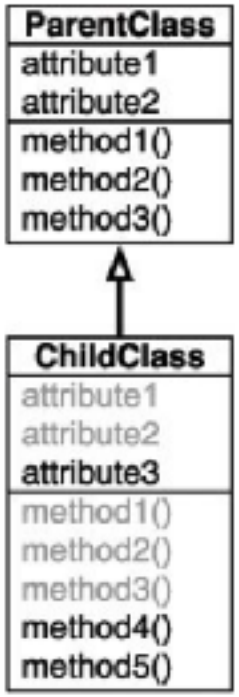
Exemple


Animal
Herbivore
Carnivore
Omnivore
Lapin
Lion
Hyène
Homme

Héritage (suite)

- Un peu comme un être humain hérite de certaines caractéristiques de ses parents, une classe qui hérite d'une autre classe contient tous les attributs et les méthodes de la classe dont elle hérite.
  - L'avantage de la classe qui hérite est qu'elle peut définir de nouveaux attributs et de nouvelles méthodes qui ne s'appliquent pas à la classe de base.

Représentation UML

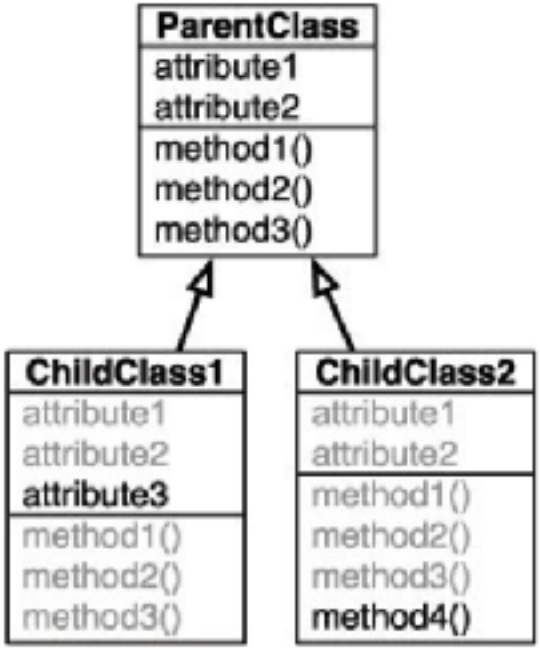


L’héritage en PHP

- Pour spécifier qu’une classe hérite d’une autre classe en PHP, il faut utiliser le mot clé *extends* lors de la définition de la *classe qui hérite*.
  - La définition de la classe mère ne change pas, ce qui est extrêmement utile!
  - Par défaut, toute classe peut devenir la classe mère d’une autre classe.
  - Voir exemple

Héritage multiple

- Il est même possible pour une même classe **parent** d'avoir plusieurs classes **enfants** ou même d'avoir des **petits-enfants!**



Encapsulation et héritage

- Si la classe B, qui hérite de la classe A, possède tous les attributs et les méthodes de la classe A, cela ne veut pas dire qu'elle peut nécessairement accéder à tous ces attributs et méthodes.
  - Pourquoi? Parce que l'attribut private que nous utilisons pour l'encapsulation restreint l'utilisation des attributs et des méthodes à *la classe qui les a défini*.
  - Cela veut dire qu'un attribut private de la classe A n'est pas accessible par la classe B, bien qu'un objet appartenant à cette classe B possède bel et bien cet attribut.

Encapsulation : Le mot clé *protected*

- Il sera toujours possible pour les méthodes de la classe B d'accéder aux méthodes public de la classe A, et donc aux setters de cette classe pour changer les attributs qui sont private.
- Dans plusieurs cas, toutefois, nous voudrons qu'un attribut défini dans la classe A puisse être accessible *directement* par les classes qui en héritent.
  - Dans ce cas, plutôt que d'utiliser l'attribut private, nous utiliserons l'attribut *protected*.
  - En utilisant protected, on obtient la même encapsulation, à l'extérieur de la classe A, qu'en utilisant *private*. Toutefois, cette encapsulation sera enlevée pour les classes *qui héritent de la classe A*.
  - Voir exemple.

Attributs et méthodes statiques

- Il est possible de déclarer des attributs et des méthodes avec le mot-clé static. Ces attributs et méthodes “statiques” sont appelés des attributs et des méthodes de classe.
- En effet, ceux-ci n’ont aucun lien avec les instances créées mais sont liées avec la classe en tant que telle.
- On les accède en utilisant le nom de la classe suivi de l’opérateur de résolution de portée (::) suivi du nom de l’attribut ou de la méthode.
- À l’intérieur d’une méthode non-statique, il est possible d’accéder aux attributs ou aux méthodes statiques en utilisant le mot clé self (qui représente la classe de l’instance), suivie de :: et du nom de l’attribut.
- Comme elles n’ont aucun lien avec une instance, il est impossible d’utiliser $this au sein d’une méthode de classe!
- Voir exemple.