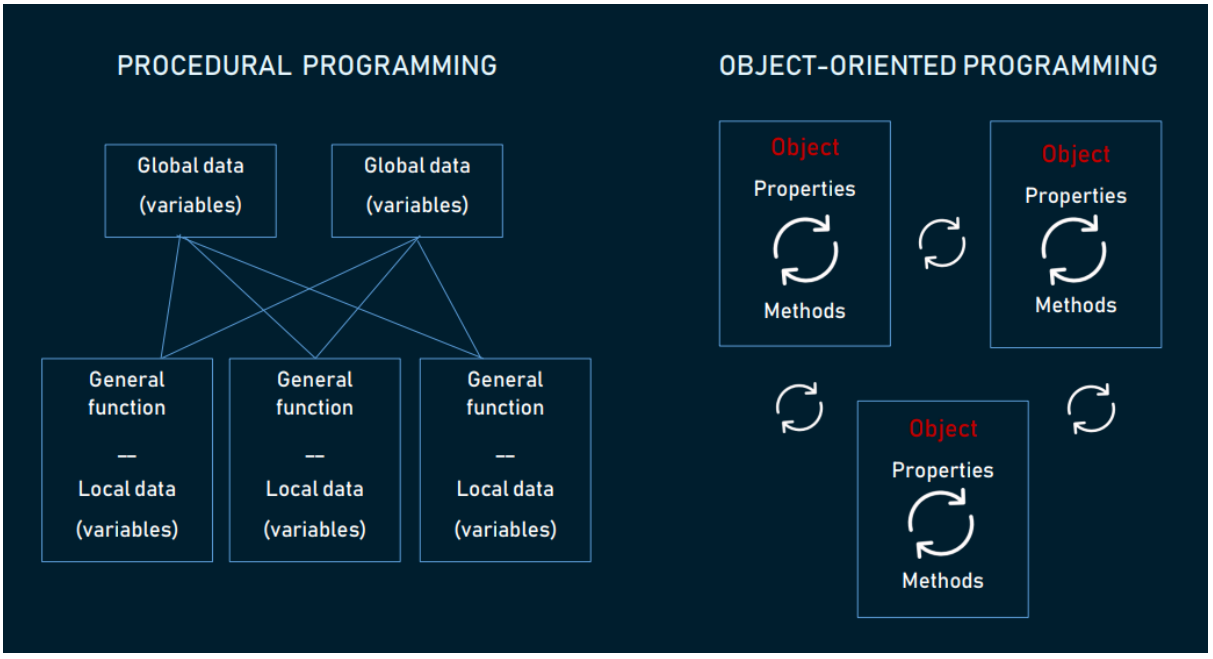
Programmation orientée objet

côté serveur (POO)

La **programmation orientée objet (POO)**, ou **programmation par objet**, est une méthode de programmation informatique. Elle consiste en la définition et l'interaction de composantes logicielles appelées **objets.** Un objet représente un concept, une idée ou toute **entité** du monde physique, comme une personne, un article ou … un objet (attention à ne pas confondre en français, en anglais la distinction entre *object* et *item* est claire).

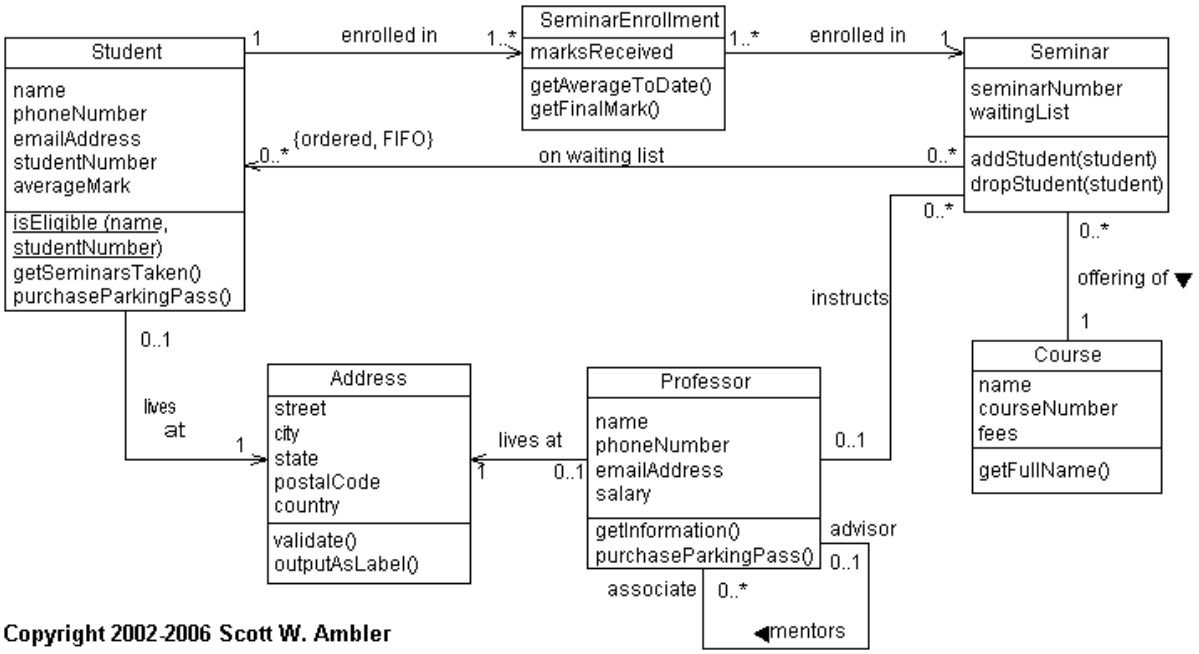
L'objet est un ensemble de données possédant une structure interne composée de différents éléments uniques appelés **attributs**. Il est capable d'interagir avec les autres objets via des appels à des fonctions internes appelées **méthodes**. Ce comportement permet aux objets de recevoir un message et d'envoyer une réponse. L'interaction entre les objets permet de concevoir et réaliser les fonctionnalités attendues afin de résoudre les problèmes posés par les algorithmes découlant des fonctionnalités nécessaires au programme.

La programmation procédurale, que vous connaissez déjà, se concentre sur la logique et l'utilisation de fonctions pour manipuler des données. En programmation orientée objet, les développeurs peuvent considérer le programme comme une collection d'objets en interaction.



Dès lors, l'étape de **modélisation** revêt une importance majeure et nécessaire pour la POO. C'est elle qui permet de transcrire les éléments réels sous forme virtuelle. La modélisation peut s'exécuter sous forme de schéma, en utilisant notamment un langage de modélisation comme UML (*Unified Modeling Language*)

Schéma de modélisation UML présentant des objets, leurs attributs, leurs méthodes et leurs relations :



Objet (attributs et méthodes)

Comme nous l'avons vu précédemment, un objet est une structure de données qui répond à un ensemble de messages. Cette structure de données définit l'**état de l'objet** tandis que l'ensemble des messages qu'il comprend décrit son **comportement** :

- les données, ou champs, qui décrivent sa structure interne sont appelées ses **attributs.**
- l'ensemble des messages forme ce que l'on appelle l'interface de l'objet ; c'est seulement au travers de celle-ci que les objets interagissent entre eux. La réponse à la réception d'un message par un objet est appelée une **méthode** (méthode de mise en œuvre du message) ; elle décrit quelle réponse doit être donnée au message.

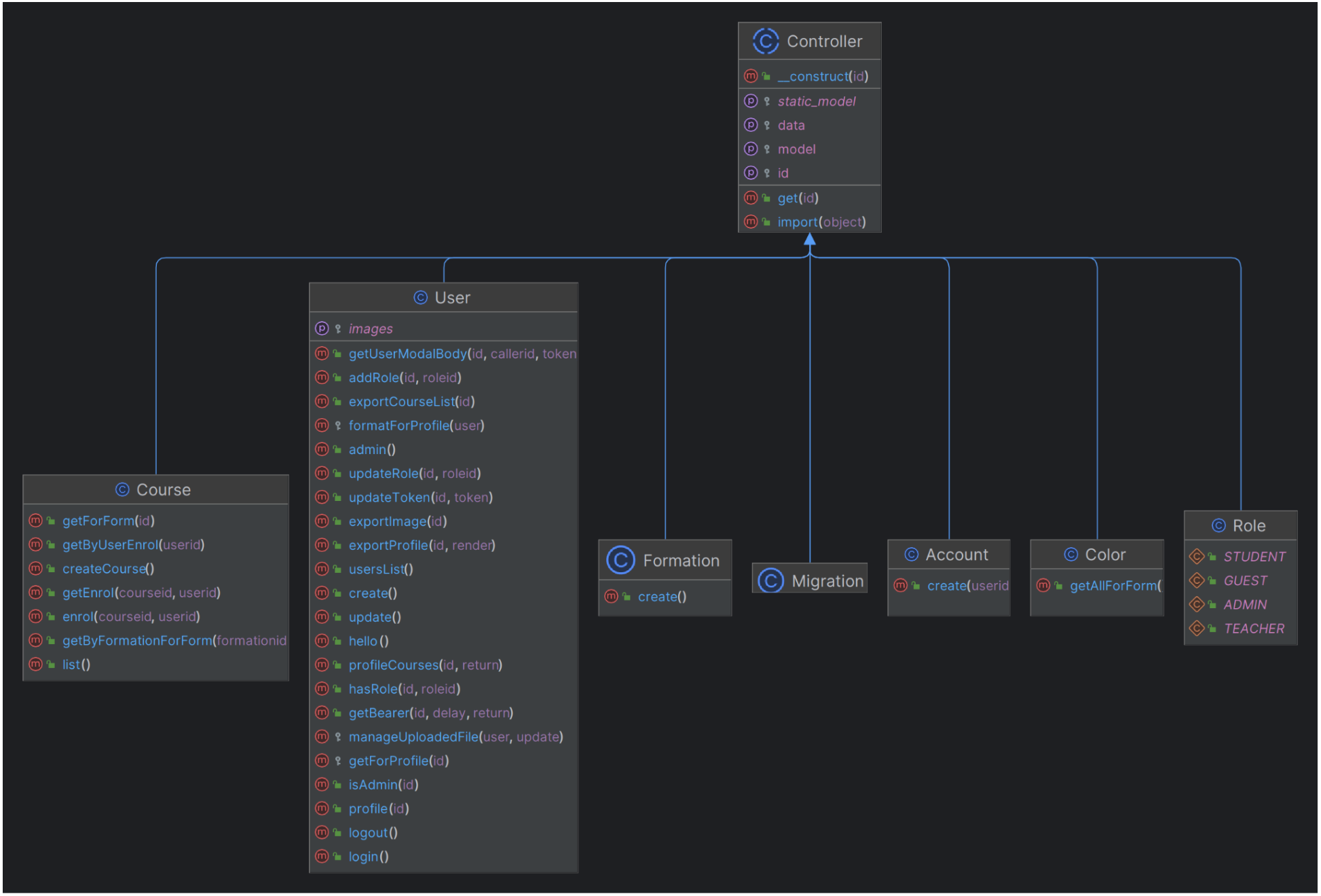
Certains attributs et/ou méthodes (ou plus exactement leur représentation informatique) sont cachés : c'est le principe d'encapsulation. Ainsi, le programme peut modifier la structure interne des objets ou leurs méthodes associées sans avoir d'impact sur les utilisateurs de l'objet.

L'encapsulation désigne le regroupement de données (les attributs) avec un ensemble de routines (les méthodes) qui en permettent la lecture et la manipulation. Ce principe est souvent accompagné du masquage de ces données brutes afin de s'assurer que l'utilisateur ne contourne pas l'interface qui lui est destinée (Typiquement, on doit passer par les méthodes pour lire ou modifier la valeur des attributs). L'ensemble se considère alors comme une boîte noire ayant un comportement et des propriétés spécifiés.
L'encapsulation est un pilier de la programmation orientée objet, où chaque classe définit des **méthodes** ou des **propriétés** pour interagir avec les données membres.

Le typage

Dans la programmation par objets, chaque objet est typé. En PHP comme en Python, le type des objets est déterminé à l'exécution lors de la création des objets, on parle alors de typage dynamique. Dans d'autres langages (C, Java), le typage est explicitement indiqué lors de la déclaration de l'objet, on parle de typage statique.

PHP inclut un modèle objet complet. Certaines de ses fonctionnalités sont : la **visibilité**, les **classes** et **méthodes abstraites** et **finales** mais aussi les **méthodes magiques**, les **interfaces**, et le **clonage**. PHP gère les objets de la même façon, qu'ils soient des références ou des gestionnaires, ce qui signifie que chaque variable contient une **référence** vers l'objet plutôt qu'une copie de l'objet complet.

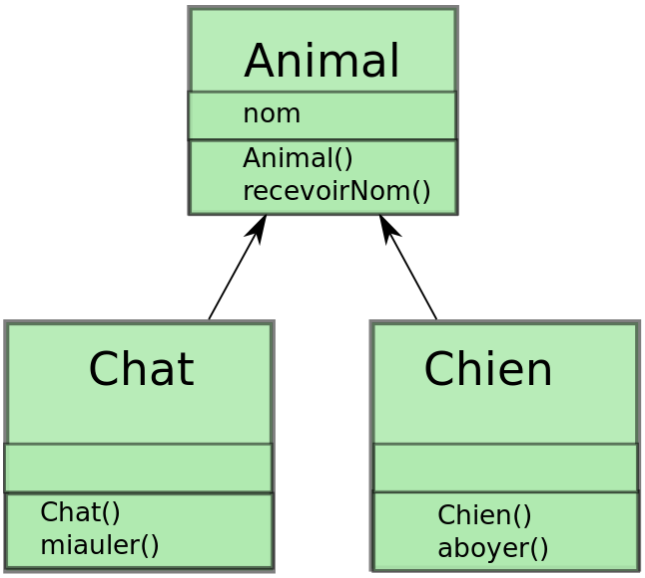

Controller
__construct(id)
static_model
data
model
id
get(id)
import(object)
User
images
getUserModalBody(id, callerid, token
addRole(id, roleid)
exportCourseList(id)
formatForProfile(user)
admin()
updateRole(id, roleid)
updateToken(id, token)
exportImage(id)
exportProfile(id, render)
usersList()
create()
update()
hello()
profileCourses(id, return)
hasRole(id, roleid)
getBearer(id, delay, return)
manageUploadedFile(user, update)
getForProfile(id)
isAdmin(id)
profile(id)
logout()
login()
Course
getForForm(id)
getByUserEnrol(userid)
createCourse()
getEnrol(courseid, userid)
enrol(courseid, userid)
getByFormationForForm(formationid
list()
Formation
create()
Migration
Account
create(userid
Color
getAllForForm(
Role
STUDENT
GUEST
ADMIN
TEACHER

La classe

Chaque objet est défini par une classe qui décrit la structure interne des données et qui définit les méthodes qui s'appliqueront aux objets de la même “famille”, c'est-à-dire les objets qui seront issus de la même définition, autrement dit de la même classe.
Lors de la création d’un objet, on va donc utiliser une méthode spécifique (appelée **constructeur**) qui va **instancier** l’objet, autrement dit créer un objet **unique** basé sur la définition de la classe génératrice. On peut donc créer plusieurs instances d’une même classe, mais chacune sera un objet différent, même s’ils sont les mêmes méthodes et même si leurs attributs ont les mêmes valeurs !

L’héritage

En programmation orientée objet, l’héritage est un mécanisme qui permet, lors de la déclaration d’une nouvelle classe, d'y inclure les caractéristiques d’une autre classe. On parle alors de relation **parent-enfant**, la classe qui hérite désignant la classe enfant.



L’héritage peut être simple (une classe hérite uniquement d’une autre classe) ou multiple (une classe peut hériter de plusieurs classes). En PHP, seul l’héritage simple est autorisé, bien qu’il existe le concept d’implémentation d’interfaces permettant d’inclure plusieurs définitions de méthodes héritées de plusieurs interfaces. Dans d’autres langages comme Python, l’héritage multiple est possible. L’héritage multiple peut présenter des problèmes lorsqu’une classe hérite de plusieurs classes possédant des définitions identiques d’attributs ou de méthodes, bien qu’il puisse exister des options de prioritisation.