***Objet***

En POO, un objet est un ensemble d’attributs et de méthodes (définis dans une classe) qui lui sont propre : un objet est donc une sorte de variable plus complexe qui possède elle-même des attributs (= variables) et des méthodes (= fonctions).

Un objet hérite ses caractéristiques (attributs et méthodes) d’une classe : c’est donc une classe qui stock toutes les informations qui pourront par la suite permettre de créer un objet.

Un objet en POO est donc à peu près pareil qu’un objet réel. En effet, comme on l’a dit, un objet en POO possède des attributs et des méthodes. Eh bien pour un objet réel, les attributs seraient ses caractéristiques et les méthodes seraient les fonctions qu’il peut remplir.

- Prenons un exemple plus concret et plus clair : on va prendre l’exemple d’une chaise pliante :

Dans la réalité, les caractéristiques de la chaise seront ses pieds, le dossier et l’assise. On a également dit qu’elle pouvait se plier : sa fonction serait la capacité de se plier.

En POO, cette même chaise possèderait des attributs qui seraient ses pieds, son dossier et son assise. Elle possèderait également une méthode : sa capacité à se plier.

- Prenons un autre exemple : le personnage d’un jeu :

Un personnage dans un jeu est défini selon ses propres caractéristiques (physiques par exemple) : il aura de la force, une intelligence, etc…

Ce personnage pourra également frapper des ennemis, se déplacer, gagner de l’expériences, etc…

En POO, ce personnage serait un objet, tout simplement : en effet, il possède des attributs (ses caractéristiques, soit la force par exemple) et des méthodes (ce qu’il sera capable de faire, se déplacer par exemple).

En clair, les méthodes en POO sont des fonctions en procédurale et les attributs en POO sont des variables en procédurale : les attributs définissent les propriétés d’un objet et les méthodes définissent le comportement de l’objet.

Cependant, pour créer un objet, on a besoin de créer une classe qui stockera toutes les informations de l’objet.

**Comment créer un objet ?**

<?php

$NomObjet = new NomObjet(attributs nécessaires à la fabrication de l’objet : ils sont définis dans la classe);

**Nicolas Sugranes**

***Classe***

**Définition :**

Un « objet » est une représentation d'une chose matérielle ou immatérielle du réel à laquelle on associe des propriétés et des actions.

Par exemple : une voiture, une personne, un animal, un nombre ou bien un compte bancaire peuvent être vus comme des objets, par conclusion tout est objet.

**Qu’est-ce-qu'une classe ?**

Une classe est le “plan de fabrication” d’un objet : en effet, c’est la classe qui contiendra les attributs (variables) et les méthodes (fonctions). C’est donc une sorte de conteneur qui regroupe la définition d’un objet. Chacunes des méthodes qui définissent un objet, ont accès aux attributs de ce même objet. Prenons l’exemple d’un gâteau et de son moule : le moule à gâteaux est unique, cependant il permet de réaliser plusieurs gâteaux. Eh bien, en POO c’est exactement pareil : le moule à gâteaux représente la classe et le gâteau représente l’objet. Le moule à gâteaux défini la forme du gâteau : il contient donc le plan de fabrication de l’objet, que l’on peut produire à l’infini avec un moule unique.

Notons également qu’une classe ne définit pas les propriétés d’un seul objet, mais plutôt d’une catégorie d’objets similaires les uns aux autres, et qui auront par conséquent des propriétés identiques ou similaires.

**Déclaration d’une classe :**

Nous venons de définir le vocabulaire propre à la programmation orientée objet. Entrons à présent dans le vif du sujet, tout d’abord une classe doit être déclarer. Nous allons déclarer une classe **«** ***Personne******»*** qui nous permettra ensuite de créer autant d'instances (objets) de cette classe que nous le souhaitons.

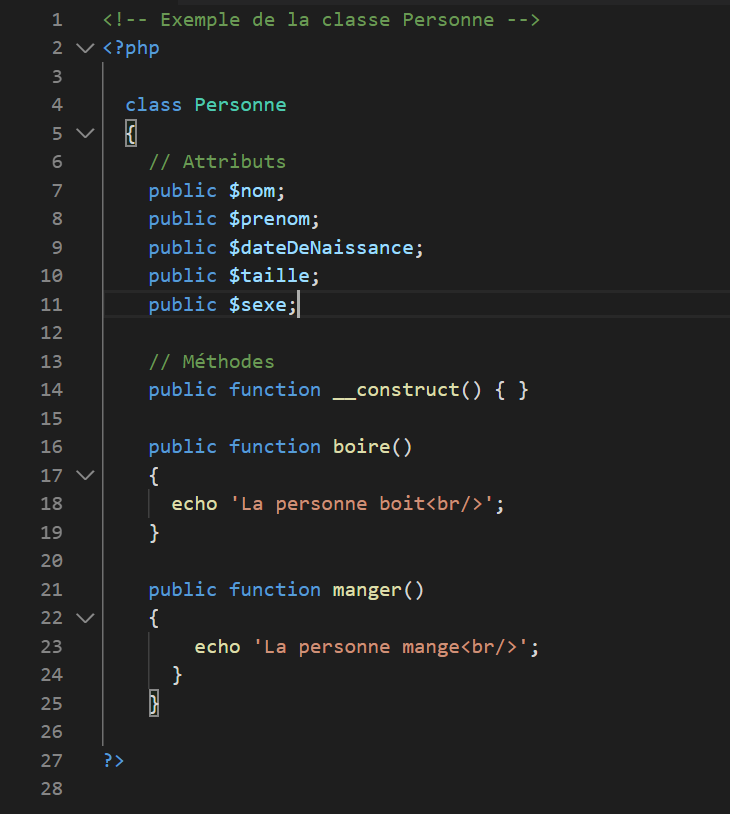
**Syntaxe de déclaration d’une classe :**

Le code suivant présente une manière de déclarer et de structurer correctement une classe. Nous vous conseillons vivement de suivre ces conventions d'écriture**.**

***Déclaration d’une classe PHP***

******

***Voici par exemple ce que cela donne avec notre exemple :***

******

Par *convention, on écrit le nom d'une classe en « Upper Camel Case »*, c'est à dire que tous les mots sont accrochés et chaque première lettre de chaque mot est écrit en majuscule.

**Killian Sieniski**

**Nicolas Sugranes**

**Les Attributs**

*Définitions :*

Les Attributs appelés aussi variables d’instances, champs, ou encore propriété. Permet de décrire la structure de l’objet.

*Comment l’utiliser :*

Pour définir un attribut on utilise un des mots-clés public, private, ou protected suivie d’une déclaration de type(optionnel), suivi de la variable et se terminant par un point virgule.

*Exemple :*

On déclare un attribut de cette façons.

mots-clé type $variable;

public $Student;

private String $Student;

protected $Student;

**AMELINE Matthias**

**Les variables**

*Définition :*

Dans un langage de programmation, une variable est un espace de stockage temporaire pour un résultat. Elle est déclarée différemment selon le langage utilisé.

En PHP, les variables sont représentées par un signe dollar "$" suivi du nom de la variable.

Les noms de variables suivent les mêmes règles de nommage que les autres entités PHP. Un nom de variable valide doit commencer par une lettre ou un souligné (\_), suivi de lettres, chiffres. Exemple : function \_\_construct().

Les variables non initialisées ont une valeur par défaut selon leur type FALSE pour les booléens, zéro pour les entiers et les réels, chaîne vide pour les chaînes de caractères

*$this* est une variable spéciale, qui ne peut pas être assignée.

*Les différents types de variables :*

les chaînes de caractères : ce type de donnée à un nom “STRING”. On y stockera du texte.

les nombres entiers : ce type de donnée à un nom “INT”. On y stockera un entier relatif.

les nombres décimaux : ce type de donnée à un nom “FLOAT”. On y stockera un nombre à virgule.

les booléens : ce type de donnée à un nom “BOOL”. On y stockera soit True soit False.

rien : ce type de donnée à un nom “NULL”.

*Constitution d’une variable :*

Une variable est toujours composé de deux éléments son Nom et sa Valeur.

**AMELINE Matthias**

***Les fonctions***

**Introduction :**

Une fonction est ce que l'on peut appeler un sous programme, une procédure. On distingue deux types de fonctions : les "fonctions intégrées" qui sont incluses par défaut avec les distributions de PHP comme **print**, **echo** et les fonctions définies par le programmeur, dites aussi "*fonctions utilisateur*".

Les fonctions ont plusieurs buts:

* Eclaircir le code en regroupant certaines fonctionnalités d'un programme qui se répètent dans une même fonction.
* Pouvoir créer des fonctions génériques qui pourront être utilisées dans d'autres programmes, ce qui évite de répéter pour chaque projet le même code
* Possibilité d'évolution du code plus facile dans la mesure où lorsque vous modifiez le contenu d'une fonction, les répercussions sont effectuées sur l'ensemble du programme sans que vous ayez à le modifier dans la plupart des cas (sauf si vous rajoutez des paramètres etc).

**Pourquoi les fonctions :**

Cela a plusieurs avantage.

**Le premiers** concerne la redondance de code et par conséquent la rapidité d'exécution.

En effet, en créant une fonction qui nous évite une redondance, le code sera moins long.

**Le second** concerne la lisibilité du code.

**Le troisième** concerne la façon de travailler :

Définir des fonctions permet de travailler avec des éléments préfabriqués.

Cela permet une plus grande flexibilité dans la façon de penser et d'écrire son code.

Vous n'allez pas écrire un programme (parfois de plusieurs centaines de lignes) si vous savez que vous avez une fonction a disposition qui fera la même chose.

Les paramètres :

Une fonction en générale prend des paramètres en entrée pour renvoyer une ou plusieurs information.

Exemple d'une fonction qui calcul l'hypoténuse d'un triangle.

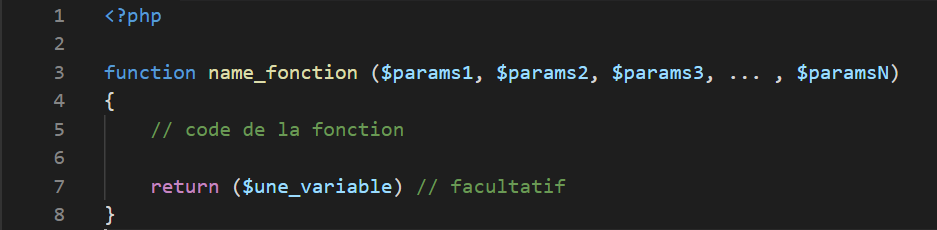
Cette fonction prendra en entrée la longueur de 2 des côtés d'un triangle rectangle et retournera en information la taille de l'hypoténuse.

Les possibilitées sont multiples et peuvent rendre le code très flexible :

Pour continuer dans cet exemple on peut imaginer que cette fonction pourra aussi prendre en entrée juste une longueur et un des angle du triangle pour retourner en informations toutes autres info manquantes : mesure des 2 autres angles, de l'autre côté de l'angle droit et de l'hypothénus.

**Syntaxe de déclaration**

Une fonction se déclare de la manière suivante :



Analysons de la syntaxe et déclaration :

* L’instruction **function** qui déclare une fonction.
* Le nom de la fonction **name\_fonction**.
* (...) Ce qui est entre parenthèse et la liste des paramètres séparés par une virgule, les arguments sont facultatifs, tout dépend du but de la fonction.
* Une instruction **return** et une fonction qui retourne/renvoie une variable.

Toutes fonctions doit être déclarée avant d'être appelée pour la première fois dans un programme.

Cela signifie que l'on signale son existence au programme.

1 - je crée mon programme.

2 - pour éviter une redondance de code, je créer une fonction.

Ainsi j'améliore la lisibilité de mon code et la rapidité d'exécution.

3 - je signale à mon programme que cette fonction existe (en général en tête du programme)

**Définition d'une fonction :**

La définition d'une fonction est propre au langage utilisé mais d'une façon générale cela s'effectuera de la façon suivante :

nomDeLaFonction(Paramètre 1, Paramètre 2,....)

..code définissant le fonctionnement de la fonction..

Puis si celle-ci retourne quelque chose :

return (information a retourner 1, information à retourner 2, information a retourner 3,...)

**Utilisation :**

Si on souhaite utiliser une fonction dans un programme, il suffit d'appeler celle-ci.

L'appel se fait suivant la syntaxe du langage utilisé.

Cependant une chose reste primordiale lors de l'appel d'une fonction.

Les paramètres et l'ordre de ceux ci :

Exemple de la fonction

calculHypoténus (angle A, longueur 1)

Je devrais de façon primordiale respecter l'ordre des informations transmisent dans les paramètres lors de l'appel de la fonction.

A contrario, le nom des arguments passés lors de l'appel importe peu.

Le programme fera lui même l'association "arguments" - "paramètres de fonction".

**Les retours :**

Une fonction peut renvoyer différente informations.

Celle ci auront été définies lors de la création de la fonction.

Cela peut être un résultat de calcul, une string, un float, un tableau, autant de type possible que peut en gérer le langage de programmation utilisé.

**Tardy Nicolazic**

**Hacault Maxence**

**Richard Théo**

**Riffi Philippe**

**Killian Sieniski**

**Quillet Olivier**

***Les méthodes***

définition wikipédia :

En programmation orientée objet (POO), une méthode est une routine membre d'une classe.

Ce que cela signifie :

Lorsque nous sommes en programmation objet, on ne parle plus de fonction mais de méthode.

On peut définir que les méthodes sont des actions applicable à un objet.

**Les méthodes** sont les actions que l'on peut appliquer à un objet.

Il s'agit en fait de **fonctions** qui peuvent prendre ou non des paramètres et retourner ou non des **valeurs / objets**. S'agissant d'actions,il est recommandé d’utiliser un verbe à l'infinitif.

Elles se déclarent de la même manière que des fonctions traditionnelles.

Au même titre que les attributs, on déclare une méthode avec un niveau de visibilité.

**Il faut ajouter trois notions :**

*public :*

Une méthode public en POO sera librement utilisable par un autre objet pas forcément de la même classe (que l'on appelle les clients de la classe).

*protégée :*

Une méthode protégée sera utilisable par les membres de la même classe et de ses classes dérivées.

*privée :*

Une méthode privée sera uniquement utilisable par les membres de la classe en cours.

**Exemple :**

Considérons

Une classe *siège* où l'on retrouve :

Un *objet fauteuil* qui possède :

Une méthode pour calculer son confort d'assise.

Une méthode pour calculer la taille de ses pieds.

Une méthode pour calculer la taille du dossier.

Un objet *tabouret* qui possède 4 pieds et une assise.

Puis une autre classe *chaise longue* qui possède :

Un objet matelas.

On peut ainsi définir la méthode taille de ses pieds en "protégée" pour que seuls les objets de la classe siège y aient accès.

C'est à dire que les objets fauteuil et tabouret y auront accès.

Concernant la méthode dossier, celle-ci pourra être "privée" parce que

seul le fauteuil a un dossier et donc ni le tabouret et le matelas en aura l'utilité.

Pour la méthode confort, celle ci pourra être "public", ainsi le matelas qui n'est pas dans la même classe pourra y avoir accès.

On peut donc noter les points suivants :

* Il existe de nombreuse méthodes déjà existante qui sont là pour simplifier la tâche du codeur, on peut en trouver notamment dans les frameworks.
* Il est parfaitement possible d'avoir une méthode qui appelle une autre méthode.

**Remarque :** Il est intéressant de noter que deux classes différentes peuvent avoir les mêmes méthodes sans risque de conflit.

**Quillet Olivier**