PISTE DE DOCUMENTATION SUR LA POO ET SA REALISATION EN PHP

I- ETUDE COMPARATIVE ENTRE LE PARADIGME OBJET ET LE PARADIGME PROCEDURAL

La différence entre la **programmation procédurale** et la **programmation orientée objet (POO)** réside dans le fait que dans la **programmation procédurale**, les programmes sont basés sur des fonctions, et les données peuvent être facilement accessibles et modifiables, alors qu'en **programmation orientée objet**, chaque programme est constitué d'entités appelées objets, qui ne sont pas facilement accessibles et modifiables.

Table de comparaison

	Programmation Procédurale	Programmation Orientée Objet
Programmes	Le programme principal est divisé en petites parties selon les fonctions.	Le programme principal est divisé en petit objet en fonction du problème.
Les données	Chaque fonction contient des données différentes.	Les données et les fonctions de chaque objet individuel agissent comme une seule unité.
Permission	Pour ajouter de nouvelles données au programme, l'utilisateur doit s'assurer que la fonction le permet.	Le passage de message garantit l'autorisation d'accéder au membre d'un objet à partir d'un autre objet.
Exemples	Pascal, Fortran	PHP5, C ++, Java.
Accès	Aucun spécificateur d'accès n'est utilisé.	Les spécificateurs d'accès public, private, et protected sont utilisés.
La communication	Les fonctions communiquent avec d'autres fonctions en gardant les règles habituelles.	Un objet communique entre eux via des messages.
Contrôle des données	La plupart des fonctions utilisent des données globales.	Chaque objet contrôle ses propres données.
Importance	Les fonctions ou les algorithmes ont plus d'importance que les données dans le programme.	Les données prennent plus d'importance que les fonctions du programme.
Masquage des données	Il n'y a pas de moyen idéal pour masquer les données.	Le masquage des données est possible, ce qui empêche l'accès illégal de la fonction depuis l'extérieur.

II- POO

La programmation par objet consiste à utiliser des techniques de programmation pour mettre en oeuvre une conception basée sur les objets. Celle-ci peut-être élaborée en utilisant des méthodologies de développement logiciel objet, dont la plus connue est le processus unifié (« *Unified Software Development Process* » en anglais), et exprimée à l'aide de langages de modélisation tels que le *Unified Modeling Language* (UML). La programmation orientée objet est facilitée par un ensemble de technologies dédiés comme les langages de programmation java, php, c++, python, ruby.

1- NAMESPACE

Dans leur définition la plus large, ils représentent un moyen d'encapsuler des éléments. Cela peut être conçu comme un concept abstrait, pour plusieurs raisons. Par exemple,

dans un système de fichiers, les dossiers représentent un groupe de fichiers associés et servent d'espace de noms pour les fichiers qu'ils contiennent. Dans le monde PHP, les espaces de noms sont conçus pour résoudre deux problèmes que rencontrent les auteurs de bibliothèques et d'applications lors de la réutilisation d'éléments tels que des classes ou des bibliothèques de fonctions

2- CLASSE

En PHP, la création d'une classe se fait en utilisant le mot-clé class suivit du nom de la classe puis d'un bloc d'instruction présentant les attributs et les méthodes de l'objet.

La déclaration d'une classe en PHP ressemble à ça :

```
class MaClasse {
   var $un_attribut;

   function une_methode() {
      return true;
   }
}
```

a. CONCRETE

Une classe concrète est une classe qui peut être instancié directement mais ne peut pas avoir d'opération abstraite.

b. ABSTRAITE

L'abstraction de classe est un moyen pour une classe de forcer toute autre classe qui l'étend à implémenter des méthodes spécifiques. La classe parente est appelée une classe abstraite, la classe enfant que nous appellerons une classe concrète et les méthodes qu'une classe concrète doit implémenter sont appelées méthodes abstraites.

Pour nos besoins, l'abstrait est un concept généralisé et le concret est une implémentation spécifique de l'abstrait.

c. ATTRIBUTS ou PROPRIETE

Les variables au sein d'une classe sont appelées "propriétés". On peut également les retrouver sous les dénominations "attributs", "membres" ou "champs. Elles sont définies en utilisant un des mots-clés *public*, *protected*, ou *private*, suivi optionellement d'une déclaration de type, suivi d'une déclaration classique de variable. Cette déclaration peut comprendre une initialisation, mais celle-ci doit être une valeur constante, c'est à dire qu'elle doit pouvoir être évaluée pendant la compilation du code, et qu'elle ne peut pas dépendre d'informations déterminées lors de l'exécution de celui-ci pour pouvoir être évaluée.

d. METHODES

Une méthode est une fonction dans un objet. Elle permet d'effectuer des opérations précises sur les propriétés de votre objet ou autres instruction. L'accès au méthodes d'un objet se fait comme pour les propriétés : écrire le nom de la variable contenant l'objet instancié, suivi d'une flèche ->et du nom de la méthode (). La méthode doit impérativement être suivi d'une parenthèse, contenant les arguments s'il y'en a.

- CONSTRUCTEUR

Le constructeur c'est la méthode qui est appelée lorsque l'on instancie un objet (avec l'instruction new). En PHP, le constructeur est une méthode qui doit s'appeler __construct().

```
<?php
class MaClasseAMoi
{
   public function __construct()
   {
      echo 'Ca y est je commence a comprendre';
   }
}
```

e. ENCAPSULATION

L'encapsulation c'est un groupement des données (propriétés, etc.) et des données permettant de les manipuler au sein d'une classe. L'encapsulation c'est pour empêcher que certaines propriétés ne soient manipulées depuis l'extérieur de la classe. Pour définir qui va pouvoir accéder aux différentes propriétés, méthodes et constantes de nos classes, nous allons utiliser des limiteurs d'accès ou des niveaux de visibilité qui vont être représentés par les mots clefs public, private et protected.

- VISIBILITE OU PORTEE D'UN ATTRIBUT OU D'UNE METHODE

Public : Les propriétés, méthodes ou constantes seront accessibles partout, c'est-à-dire depuis l'intérieur ou l'extérieur de la classe.

Private : Les propriétés, méthodes ou constantes ne seront accessibles que depuis l'intérieur de la classe qui les a définies.

protected : Les propriétés, méthodes ou constantes seront accessibles que depuis l'intérieur de la classe qui les a définies ainsi que depuis les classes qui en héritent ou la classe parente.

- GETTERS

La méthode __get() permet, quant à elle, de lire la valeur d'une propriété inexistante de la classe. Au même titre que la méthode __set(), la méthode __get() doit être redéfinie dans la classe pour exécuter du code personnalisé lorsque PHP appelle implicitement

cette méthode. Là encore, cela permet de réaliser un contrôle d'accès sur les propriétés dont on essaie de lire les valeurs.

• SETTERS

La méthode __set() permet de faire ce que l'on appelle de la surcharge de propriétés d'une classe. En effet, lorsque l'on essaie de fixer la valeur d'une propriété inexistante de la classe, PHP appelle automatiquement cette méthode de manière implicite. En redéfinissant explicitement cette méthode dans le corps de la classe, cela permet au développeur de réaliser des contrôles d'accès et de s'assurer que seules quelques propriétés peuvent être mises à jour. Voici sa structure générale.

```php
<?php

class MyObject
{
 /**
  * Methode magique __set()
  *
  * @param string $property Nom de la propriété
  * @param mixed $value Valeur à affecter à la propriété
  * @return void
  */
 public function __set($property, $value)
 {
  // Code personnalisé à exécuter
 }
}

?>
```

f. SURCHARGE

La surcharge magique en PHP permet de "créer" dynamiquement des propriétés et des méthodes. Ces entités dynamiques sont traitées via des méthodes magiques établies que l'on peut positionner dans une classe pour divers types d'actions.

Les méthodes magiques de surcharge sont appelées lors de l'interaction avec des propriétés ou des méthodes qui n'ont pas été déclarées ou ne sont pas visibles dans le contexte courant. Le reste de cette section utilise les termes de propriétés inaccessibles et de méthodes inaccessibles pour se référer à cette combinaison de déclaration et de visibilité.

Toutes les méthodes magiques de surcharge doivent être définies comme *public*.

3- OBJETS

Pour faire simple, un objet est une chose. Il est possible de définir par exemple un objet télévision, voiture, bureau, musique… Un objet peut donc être vu comme une donnée à manipuler. Chaque objet est défini par des attributs et des méthodes qui lui sont propres. Un attribut est une caractéristique de l'objet. Par exemple, un objet voiture peut avoir un attribut couleur. Une méthode est une action qui peut être appliquée à un objet. Par exemple, un objet voiture peut avoir une méthode accélérer. En resumé , un objet est une instance de classe.

4- RELATION ENTRE CLASSE

a. NAVIGABILITE ENTRE CLASSE

- OneToMany
- ManyToOne
- OneToOne
- ManyToMany

b. HERITAGE

Quand on parle d'héritage, c'est qu'on dit qu'une classe B hérite d'une classe A. La classe A est donc considérée comme la classe mère et la classe B est considérée comme la classe fille.

Lorsqu'on dit que la classe B hérite de la classe A, c'est que la classe B hérite de tous les attributs et méthodes de la classe A. Si l'on déclare des méthodes dans la classe A, et qu'on crée une instance de la classe B, alors on pourra appeler n'importe quelle méthode déclarée dans la classe A du moment qu'elle est publique.

- REDEFINITON

La redéfiniton de méthodes permet à une sous-classe de modifier le comportement d'une méthode héritée de sa super-classe.

- POLYMORPHISME

La notion de polymorphisme est très liée à celle d'héritage. Grâce à la redéfinition, il est possible de redéfinir une méthode dans des classes héritant d'une classe de base. Par ce mécanisme, une classe qui hérite des méthodes d'une classe de base peut modifier le comportement de certaines d'entre elles pour les adapter à ses propres besoins. Contrairement à la surcharge, une méthode redéfinie doit non seulement avoir le même nom que la méthode de base, mais le type et le nombre de paramètres doivent être identiques à ceux de la méthode de base.

5- INTERFACE

Techniquement, une interface est une classe entièrement abstraite. Son rôle est de décrire un comportement à notre objet. Les interfaces ne doivent pas être confondues avec l'héritage : l'héritage représente un sous-ensemble (exemple : un magicien est un sous-ensemble d'un personnage.

u ne interface se déclare avec le mot-cléinterface, suivi du nom de l'interface, suivi d'une paire d'accolades. C'est entre ces accolades que vous listerez des méthodes. Par exemple, voici une interface pouvant représenter le point commun évoqué ci-dessus :

< ?php

Interface Movable

{

Public function mave($dest)

}

?>

PISTE DE DOCUMENTATION SUR LE PATTERN MVC EN UTILISANT LA POO

L’accronyme MVC signifie (Model Vue Controller) est un des plus célèbre design patterns. Le pattern MVC permet de bien organiser son code source. Il aide a créer des fichiers, et surtout à définir leur rôle. Le but de MVC est justement de séparer la logique du code en trois parties que l'on retrouve dans des fichiers distincts.

1- MODEL

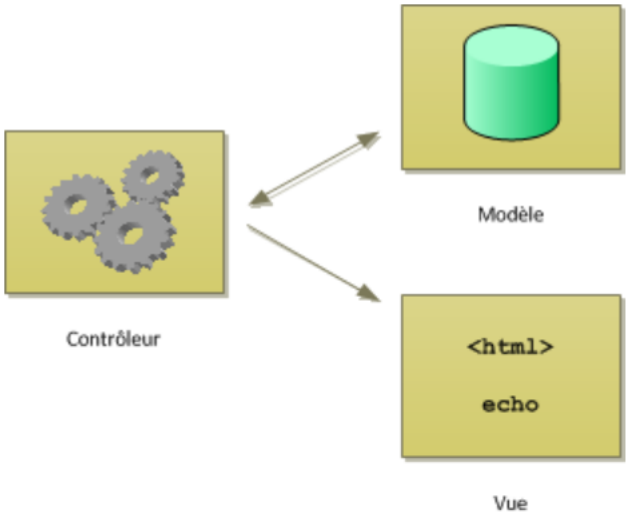
Le model gère les *données* de votre site. Son rôle est d'aller récupérer les informations « brutes » dans la base de données, de les organiser et de les assembler pour qu'elles puissent ensuite être traitées par le contrôleur. On y trouve donc entre autres les requêtes SQL.

2- VUE

La vue se concentre sur l'*affichage*. Elle ne fait presque aucun calcul et se contente de récupérer des variables pour savoir ce qu'elle doit afficher. On y trouve essentiellement du code HTML mais aussi quelques boucles et conditions PHP très simples, pour afficher par exemple une liste de messages.

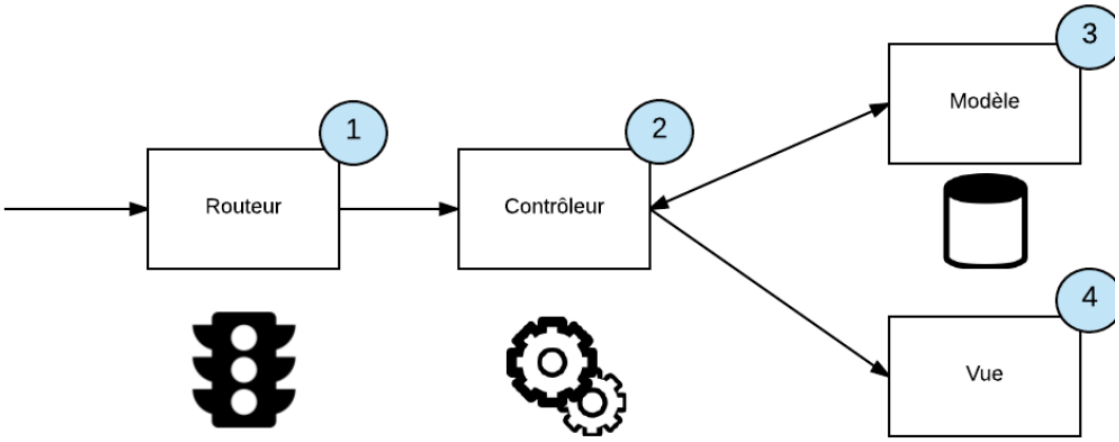
3- CONTROLLER

Le controller gère la logique du code qui prend des *décisions*. C'est en quelque sorte l'intermédiaire entre le modèle et la vue : le contrôleur va demander au modèle les données, les analyser, prendre des décisions et renvoyer le texte à afficher à la vue. Le contrôleur contient exclusivement du PHP. C'est notamment lui qui détermine si le visiteur a le droit de voir la page ou non (gestion des droits d'accès).



4- ROUTER

Pour faciliter la maintenance, il est plus simple de passer par un contrôleur *frontal*, qui va jouer le rôle de *routeur*. Son objectif va être d'appeler le bon contrôleur (on dit qu'il *route* les requêtes).



5- LAYAOUT OU TEMPLATE

On va créer un template (aussi appelé gabarit) de page. On va y retrouver toute la structure de la page, avec des "trous" à remplir.

6- AVANTAGES

L’approche MVC apporte de réels avantages:

- Une conception claire et efficace grâce à la séparation des données de la vue et du contrôleur
- Un gain de temps de maintenance et d’évolution du site

- Une plus grande souplesse pour organiser le développement du site entre différents développeurs (indépendance des données, de l'affichage (webdesign) et des actions)

7- INCONVENIENTS

L'inconvénient majeur du modèle MVC n'est visible que dans la réalisation de petits projets, de sites internet de faible envergure.
En effet, la séparation des différentes couches nécessite la création de plus de fichiers (3 fois plus exactement):

- Un fichier pour le modèle
- Un fichier pour le contrôleur
- Un fichier pour la vue

Il n'est donc pas très intéressant de recourir à ce système dans ce cas là.

8- ORM(Object Relationnal Mapping)

Un mapping objet-relationnel (en anglais object-relational mapping ou ORM) est un type de programme informatique qui se place en interface entre un programme applicatif et une base de données relationnelle pour simuler une base de données orientée objet. Ce programme définit des correspondances entre les schémas de la base de données et les classes du programme applicatif. On pourrait le désigner par là, « comme une couche d'abstraction entre le monde objet et monde relationnel ». Du fait de sa fonction, on retrouve ce type de programme dans un grand nombre de frameworks sous la forme de composant ORM qui a été soit développé, soit intégré depuis une solution externe.