PLAN :
INTRODUCTION

A) Piste de Documentation sur la POO et sa Réalisation en PHP

I. **Etude Comparative entre le Paradigme Objet et Paradigme procédural**
II. **POO**
    1)Namespace
    2)Classe
    3)Objets
    4) Relation entre Classe
    5) Interface

B) Piste de Documentation sur le Pattern MVC en utilisant la POO

1)Vue

2)Controller

3)Router

4)Layout ou Template

5)Avantages

6)Inconvénients

7)ORM(Object Relational Mapping)

INTRODUCTION

La programmation orientée objet, c'est un nouveau moyen de penser votre code. C'est une conception inventée dans les années 1970, qui prend de plus en plus de place aujourd'hui. Ce paradigme permet une organisation plus cohérente de vos projets, une maintenance facilitée et une distribution de votre code plus aisée !

I. Etude Comparative entre le Paradigme Objet et Paradigme procédural

Dans sa définition première, **un paradigme est une représentation du monde**, ou une manière de voir les choses. Le paradigme est un modèle, un exemple, ou encore une référence sur laquelle on se base pour illustrer une règle.

Le monde de la programmation dispose d'un bon nombre de paradigmes, dont le but est d'**aider le développeur à mieux penser son application** : Objet bien sûr, mais également MVC, programmation orientée Aspect, programmation déclarative...

C'est ainsi que le développeur programmant à l'aide d'un langage Objet doit porter sa réflexion sur un **modèle centré sur l'objet** : les programmes qu'il construit sont composé d'objets (ou instances de classe) qui communiquent entre eux, et travaillent ensemble.

Plutôt que d'écrire un programme de manière séquentielle, le développeur devra réfléchir à la **pertinence de tel objet ou telle méthode dans le cadre du programme global**, chaque objet disposant d'une responsabilité précise.

Le paradigme Objet nécessite donc des objets qui interagissent, ce qui influe profondément l'approche du code. Pour un même résultat, un code Objet sera bien différent d'un code utilisant la programmation fonctionnelle, où le programme est une séquence d'évaluations de fonctions.

Un paradigme n'est pas forcément lié à un langage : certains peuvent aussi bien utiliser du procédural, de l'orienté objet, du fonctionnel et du générique, pour une même syntaxe. En ce sens, le paradigme objet n'est pas lié directement aux interfaces Java.

Cela étant, les interfaces de Java, par l'abstraction qu'elles offrent, permettent un héritage plus poussé, par encapsulation d'objets. En cela, les interfaces de Java renforcent le paradigme Objet, même si elles ne lui sont pas nécessaires.

Peu à peu, différents **paradigmes de programmation** ont émergé.

Ainsi, on a parlé de **programmation procédurale** (permettant de regrouper des traitements au sein de procédures ou fonctions pouvant être appelées par différents composants du code), puis de **programmation orientée objet**, sur laquelle nous allons nous attarder dans cette partie.

Il faut savoir que beaucoup d'autres paradigmes de programmation existent (programmation déclarative, événementielle, orientée aspect…) : ceci dépasse le cadre de cette formation.

II. POO

La programmation orientée objet consiste à introduire un nouveau type de données qui est l'objet. Ce type de données offre plus de possibilités de contrôle et d'interactions entre les

différents composants de notre application.

Php est connu comme un langage procédural. Mais depuis sa version 4 il a introduit la possibilité de programmer en orienté objet. Cette tentative d'immersion dans le monde de la

programmation orientée objet a été mieux réussis à partir de la version 5.6.

1)Namespace

Un **espace de nom** ou **namespace** représente un moyen de séparer ses éléments au sein du code de telle sorte à éviter les conflits (ou collisions). Ces collisions sont dues à des duplications de noms (ou identifiants) d'éléments comme les fonctions, les constantes ou les classes.

2)Classe

Les classes permettent de définir un nouveau type de données avec des règles et un comportement spécifiques.
Une classe est une entité regroupant des variables et des fonctions. Chacune de ces fonctions aura accès aux variables de cette entité

a)Concrete

une classe concrète qui étend une ou plusieurs classes abstraites (indirectement), doit obligatoirement fournir une implémentation pour toutes les méthodes abstraites existantes.

b) Abstraite

Les classes abstraites s'inscrivent davantage dans la sûreté de la programmation orientée objet. La première particularité d'une classe abstraite, c'est qu'elle ne peut être instanciée (et donc créer un objet). De cette affirmation, on en déduit logiquement qu'une classe abstraite est déclarée afin d'être dérivée par des classes concrètes.Une classe abstraite se comporte comme une classe concrète typique. C'est-à-dire qu'elle peut déclarer des attributs et des méthodes traditionnels qui seront accessibles dans les classes dérivées. En fonction bien sûr de la visibilité choisie (*public*, *private* et *protected*) pour chacun des attributs et méthodes.

c) Attributs

Les attributs sont les variables membres de la classe. Ils constituent les propriétés ou les caractéristiques de l'objet (l'instance de classe) qui en sera né.
Pour déclarer un attribut il faut le précéder par sa **visibilité**. La visibilité d'un attribut indique à partir d'où on peut y avoir accès. Il existe trois types de visibilité:

-public: dans ce cas, l'attribut est accessible de partout (de l'intérieur de la classe dont il est membre comme de l'extérieur).

-**private**: dans ce cas, l'attribut est accessible seulement de l'intérieur de la classe dont il est membre.

-protected: dans ce cas, l'attribut est accessible seulement de l'intérieur de la classe dont il est membre ainsi que de l'intérieur des classes fille qui héritent de cette classe.

- Instance
  L'instanciation est le fait de créer une **instance**. Pour être précis, on parle d'une **instance de classe**. La classe étant le moule qui sert à fabriquer les objets, alors chaque objet créé correspond à une instance de la classe qui lui a donné vi

● Classe

Une classe est une structure cohérente de propriétés (attributs) et de comportements (méthodes). C'est elle qui contient la définition des objets qui vont être créés après. En général on considère une classe comme un moule à objets. Avec un seul moule on peut créer autant d'objets que l'on souhaite.

d)Methodes

Les méthodes sont des fonctions déclarées dans une classe. Les méthodes d'une classe ne sont accessibles que par les objets de la classe ou elles sont définies et par les objets enfant de cette classe.

- **Instance**
  -But : simplifier la définition des méthodes qui manipulent une structure de données
  - Principe : associer des méthodes aux instances
  Méthode d'instance = méthode sans mot clé static
  Méthode qui manipule une structure de données
  Associée à la classe dans laquelle la méthode est définie
  Reçoit un paramètre caché nommé this du type de l'instance ⇒ pas besoin de spécifier explicitement ce paramètre ⇒ simplifie le code

- **Constructeurs**
  Le constructeur d'une classe permet de définir les paramètres à renseigner au moment de l'instanciation. En effet, il est possible de déclarer une classe qui pour s'instancier a besoin que l'on définisse la valeur de ses attributs.

- **Static**
  Une **méthode static** est une **méthode** qui n'agit pas sur des variables d'**instance** mais uniquement sur des variables de classe. Ces **méthodes** peuvent être utilisées sans instancier un objet de la classe.

e)Encapsulation

L'encapsulation consiste à définir la visibilité et l'accessibilité des propriétés et méthodes d'une classe pour mieux en maîtriser leur utilisation. Pour cela, il suffit de déclarer « private » les données à encapsuler et de définir des méthodes permettant de les lire et de les modifier : on appelle ces méthodes « getter » (pour la lecture) et « setter » (pour la modification).

● Visibilité ou Portée d'un attribut ou d'une Méthode

La visibilité permet de définir de quelle manière un attribut ou une méthode d'un objet sera accessible dans le programme. Comme Java, C++ ou bien ActionScript 3, PHP introduit 3 niveaux différents de visibilité applicables aux propriétés ou méthodes de l'objet.

Il s'agit des visibilités ***publiques, privées*** ou ***protégées*** qui sont respectivement définies dans la classe au moyen des mots-clés ***public, private*** et ***protected***.

● Getters
Getter(accesseurs) : méthode « public » permettant de définir la manière de lecture d'un attribut privé. Son type de retour est celui de la donnée retournée et son nom est souvent composé de « get » et du nom de l'attribut qu'elle retourne.

● Setters
Setter(mutateurs) : méthode « public » permettant de définir la manière de modification d'une donnée. Souvent, elle ne retourne rien (« void ») et prend un paramètre du même de

type que la donnée à modifier. Son nom se compose de la mention « **set** » et du nom de l'attribut concerné.

f) Surcharge

En programmation orientée objet, la surcharge, aussi appelée « **overloading** », **consiste à déclarer, dans une même classe, deux méthodes de même nom mais avec des sémantiques différentes** :

- Même nom de méthode,
- Paramètres différents (soit sur le nombre ou le/les type(s)),
- Le type de retour n'est pas pris en compte.

L'aspect de la « surcharge » est retrouvé lors de la déclaration de plusieurs constructeurs dans une même classe.

3) Objets

Quand on utilise la POO, on cherche à représenter le domaine étudié sous la forme d'objets. C'est la phase de **modélisation orientée objet**.

Un **objet** est une entité qui représente (*modélise*) un élément du domaine étudié : une voiture, un compte bancaire, un nombre complexe, une facture, etc.

Objet = état + actions

Cette équation signifie qu'un objet rassemble à la fois :

- des **informations** (ou données) qui le caractérisent.
- des **actions** (ou traitements) qu'on peut exercer sur lui.

4) Relation entre Classe

a) Navigabilité entre Classe

Étant donné qu'en POO les objets logiciels interagissent entre eux, il y a donc des relations entre les classes.

On distingue quatre types de relations durables

Association : Représente une relation durable entre deux classes. Une personne peut posséder des voitures. La relation “possède” est une association entre les classes Personne et Voiture. Les associations peuvent donc être nommées pour donner un sens précis à la relation.

Agrégation : L’agrégation est un cas particulier d’association non symétrique exprimant une relation de contenance. Une ligne d’une commande contient l’achat d’un article. Les agrégations n’ont pas besoin d’être nommées : implicitement elles signifient “contient” ou “est composé de ”.

Composition : est une agrégation plus forte signifiant “est composé d’un” et impliquant : un composant ne peut appartenir qu’à un seul composite (agrégation non partagée) la destruction du composite entraîne la destructions de tous ses composants (il est responsable du cycle de vie de ses parties).

Héritage : permet d’ajouter des éléments (propriétés et/ou comportement) à une classe existante pour en obtenir une nouvelle plus précise.

b) **Héritage**

Les classes qui ont des attributs similaires peuvent être regroupés dans une relation hiérarchique. C'est-à-dire une relation de parent à enfant appelé héritage.

Dans cette relation nous avons une classe parente qui définit les propriétés et les méthodes. Et les classes enfants vont hériter

tout ou partie de ses attributs et méthodes. Pour définir un héritage on utilise le mot clé 'extends' lors de la création de la classe.

5) Interface

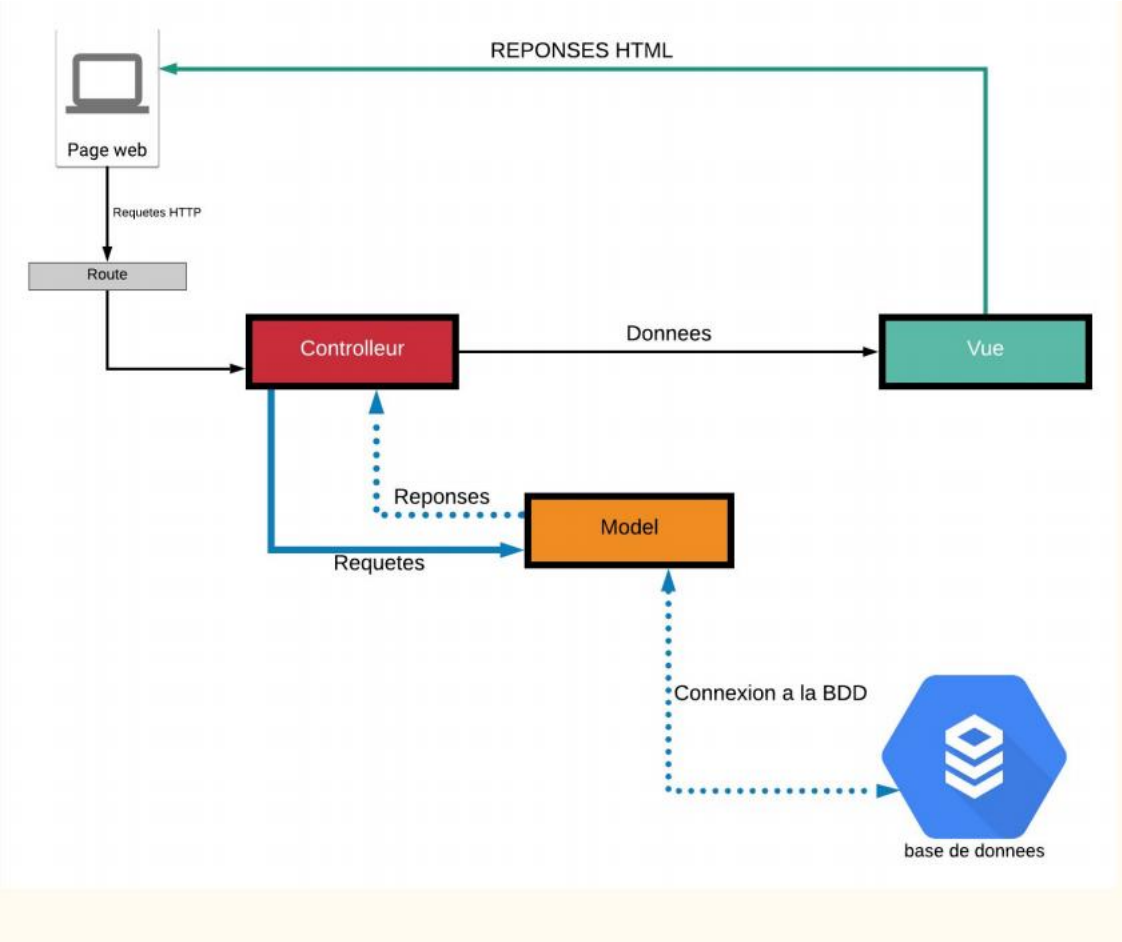
Les interfaces vont avoir un but similaire aux classes abstraites puisque l'un des intérêts principaux liés à la définition d'une interface va être de fournir un plan général pour les développeurs qui vont implémenter l'interface et de les forcer à suivre le plan donné par l'interface.
De la même manière que pour les classes abstraites, nous n'allons pas directement pouvoir instancier une interface mais devoir l'implémenter, c'est-à-dire créer des classes dérivées à partir de celle-ci pour pouvoir utiliser ses éléments.

Les deux différences majeures entre les interfaces et les classes abstraites sont les suivantes :

1. Une interface ne peut contenir que les signatures des méthodes ainsi qu'éventuellement des constantes mais pas de propriétés. Cela est dû au fait qu'aucune implémentation n'est faite dans une interface : une interface n'est véritablement qu'un plan ;
2. Une classe ne peut pas étendre plusieurs autres classes à cause des problèmes d'héritage. En revanche, une classe peut tout à fait implémenter plusieurs interfaces.

B) Piste de Documentation sur le Pattern MVC en utilisant la POO

Alors, la définition du modèle MVC, comme je vous l'ai dit, il est divisé en trois entités. Vous avez la vue, sachant que selon le type d'application que vous développez, vous pouvez avoir plusieurs vues, plusieurs modèles, plusieurs contrôleurs. Donc vous avez la vue, en gros c'est l'interface homme/machine. La vue, c'est des IHM, c'est à dire les interfaces graphiques sur laquelle votre utilisateur intervient, donc ça peut une page Web, les interfaces graphiques des jeux, ou des applications.
Et derrière vous avez le modèle. Le modèle lui, il contient toutes les données du programme, c'est à dire les algorithmes, la configuration de l'application, la logique. En gros le modèle c'est vraiment le cœur de l'application.
Et derrière, il y a le contrôleur qui lui, comme son nom l'indique, contrôle les données, c'est à dire qu'il transmet les informations entre la vue et le modèle, il peut transmettre des informations modèle ou aux vues, directement ça arrive lorsque l'utilisateur effectue une action, lorsque vous avez un retour d'événement par exemple, chargement de fichiers terminés, le contrôleur peut informer les vues en disant : "le chargement de fichiers terminés, mettez-vous à jour, dites à l'utilisateur qu'il peux cliquer sur le bouton lancer le jeu" par exemple.



1)Vue

les vues sont chargées de communiquer avec le navigateur ils gèrent l'affichage de la page demandé. Il n'y a généralement pas de traitement dans les vues on ne fait qu'afficher.

2)Controller

Les fichiers des contrôleurs sont localisés dans app/http/controllers. Ce sont des fichiers dont le nom se termine toujours par "Controller'". Ils contiennent des classes qui héritent d'une classe Controller() et qui vont avoir des méthodes appelées actions. Ces méthodes vont le plus souvent correspondre aux pages de notre site web.

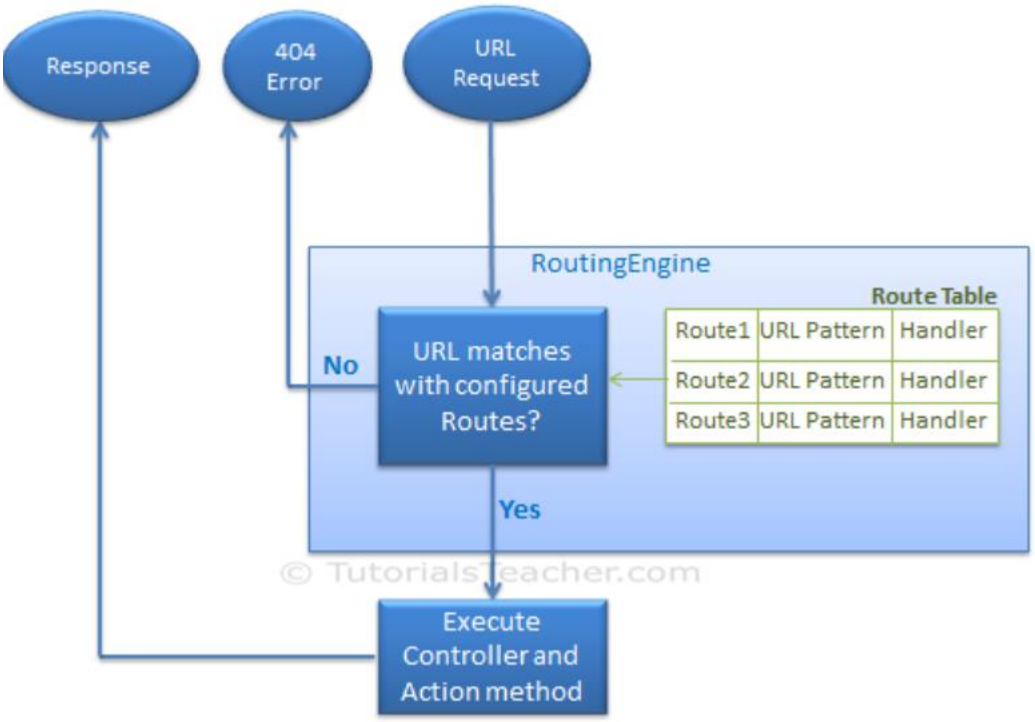
Les contrôleurs jouent un rôle intermédiaire entre les classes modèles et les vues. ils

vont recevoir les données de l'URL, faire un traitement et envoyer les valeurs aux vues. Ils sont le point central de notre application.

Pour ajouter un controller nous pouvons créer un fichier dans app/Http/Controllers. Mais nous pouvons aussi utiliser php artisan. La commande pour créer un controller est alors php artisan make:controller NomController. Cette commande va créer un fichier avec le Nom défini et la classe du controller.

3)Router

Route définit le modèle d'URL et les informations du gestionnaire. Toutes les routes configurées d'une application stockées dans RouteTable et seront utilisées par le moteur de routage pour déterminer la classe ou le fichier de gestionnaire approprié pour une demande entrante.



Routage dans MVC

4)Layout ou Template

-La `layout()` fonction vous permet de définir un modèle de mise en page qu'un modèle implémentera. C'est comme avoir des modèles d'en-tête et de pied de page séparés dans un seul fichier.

-Un **Template**, connu également sous le terme de modèle, *layout* ou Gabarit en français, représente l'ensemble des éléments graphiques de l'agencement des colonnes, passant par le choix des couleurs jusqu'à l'établissement de la structure des différents éléments enveloppant un site Internet, abstraction faite de son contenu.

Aujourd'hui la majorité des créateurs de sites web recourent à un **Template** prédéfini, pour éviter de s'occuper des dimensions graphiques et centrer plus les efforts sur tout ce qui est contenu du site. Ces derniers fonctionnent de manière très analogue aux modèles de documents Word ou Power point.

Un **Template** existe aussi bien pour les sites simples codés en HTML, que pour les CMS. En effet, un CMS tel que joomla ou Wordpress par exemple dont la **création d'un site Internet** est considérablement facilitée par son fonctionnement auquel on ajoute un Template permet l'accès à un design pro en peu de temps et rend le processus d'autant plus facile.

5) Avantage et Inconnéviens sur le Pattern MVC

Un avantage apporté par ce modèle est la clarté de l'architecture qu'il impose. Cela simplifie la tâche du développeur qui tenterait d'effectuer une maintenance ou une amélioration sur le projet. En effet, la modification des traitements ne change en rien la vue. Par exemple on peut passer d'une base de données de type SQL à XML en changeant simplement les traitements d'interaction avec la base, et les vues ne s'en trouvent pas affectées.

Le MVC montre ses limites dans le cadre des applications utilisant les technologies du web, bâties à partir de serveurs d'applications[réf. nécessaire]. Des couches supplémentaires sont alors introduites ainsi que les mécanismes d'inversion de contrôle et d'injection de dépendances[8].

6) ORM(Object Relational Mapping)

Le mappage relationnel-objet (**ORM** , **O / RM** et **outil de mappage O / R**) en informatique est une technique de programmation pour convertir des données entre des systèmes de type incompatibles à l' aide de langages de programmation orientés objet. Cela crée, en effet, une " base de données d'objets virtuels " qui peut être utilisée à partir du langage de programmation. Il existe des packages gratuits et commerciaux qui effectuent le mappage relationnel-objet, bien que certains programmeurs choisissent de construire leurs propres outils ORM.

Dans la programmation orientée objet , les tâches de gestion des données agissent sur des objets qui sont presque toujours des valeurs non scalaires . Par exemple, une entrée de carnet d'adresses qui représente une seule personne avec zéro ou plusieurs numéros de téléphone et zéro ou plusieurs adresses. Cela pourrait être modélisé dans une implémentation orientée objet par un " objet Personne " avec des attributs / champspour contenir chaque élément de données que l'entrée comprend: le nom de la personne, une liste de numéros de téléphone et une liste d'adresses. La liste des numéros de téléphone contiendrait elle-même des "objets PhoneNumber" et ainsi de suite. L'entrée du carnet d'adresses est traitée comme un seul objet par le langage de programmation (elle peut être référencée par une seule variable contenant un pointeur vers l'objet, par exemple). Diverses méthodes peuvent être associées à l'objet, comme une méthode pour renvoyer le numéro de téléphone préféré, l'adresse du domicile, etc.

Cependant, de nombreux produits de base de données populaires tels que les systèmes de gestion de base de données SQL (SGBD) ne peuvent stocker et manipuler que des valeurs scalaires telles que des entiers et des chaînes organisées dans des tables . Le programmeur doit soit convertir les valeurs des objets en groupes de valeurs plus simples pour le stockage dans la base de données (et les reconvertir lors de la récupération), soit uniquement utiliser des valeurs scalaires simples dans le programme. La cartographie relationnelle objet implémente la première approche. [1]

Le cœur du problème consiste à traduire la représentation logique des objets en une forme atomisée qui peut être stockée dans la base de données tout en préservant les propriétés des objets et leurs relations afin qu'ils puissent être rechargés en tant qu'objets en cas de besoin. Si cette fonctionnalité de stockage et de récupération est implémentée, les objets sont réputés persistants . [1]