

Documentation sur la Programmation Orientée Objet (POO)

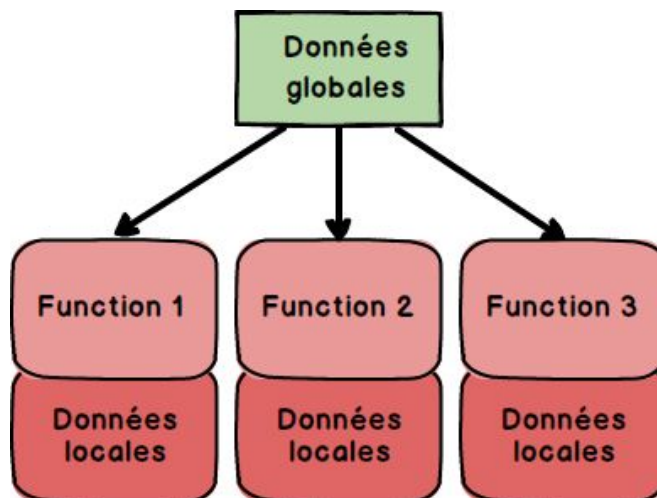
I. Etude Comparative entre la POO et la programmation procédurale

1. La Programmation Procédurale

a) Définition

En **informatique**, la **programmation procédurale** est un **paradigme** qui se fonde sur le concept d'appel procédural.

Dans la **programmation procédurale** le programme est divisé en petites parties appelées **procédures** ou **fonctions**. Comme son nom l'indique, elle contient une procédure étape par étape à exécuter. Ici, les problèmes sont décomposés en petites parties et ensuite, pour résoudre chaque partie, une ou plusieurs fonctions sont utilisées.



Exemple de code procédural:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
```

```
// Create connection
$conn = mysqli_connect($servername, $username,
$password);

// Check connection
if (!$conn) {
    die("Connection failed: " .
mysqli_connect_error());
}
echo "Connected successfully";
?>
```

b) Avantages

La programmation procédurale est un meilleur choix qu'une simple **programmation séquentielle**. Les avantages sont en effet les suivants :

- la possibilité de réutiliser le même **code** à différents emplacements dans le programme sans avoir à le retaper (factorisation), ce qui a pour effet la réduction de la taille du code source et un gain en localité des modifications, donc une amélioration de la maintenabilité (compréhension plus rapide, réduction du risque de régression) ;
- une façon plus simple de suivre l'exécution du programme : la programmation procédurale permet de se passer d'**instructions** telles que **goto**, évitant ainsi bien souvent de se retrouver avec un programme compliqué qui part dans toutes les directions (appelé souvent « **programmation spaghetti** »)

c) inconvénients

- L'évolution d'une application développée suivant ce modèle n'est pas évidente car la moindre modification des structures de données d'un programme conduit à la révision de toutes les procédures manipulant ces données.
- Pour de très grosses applications, le développement peut être très long

2 .Programmation Orienté Objet

a) Définition

Les inconvénients de la programmation procédurales et en particulier les temps de programmation trop longs ont contribués à développer une nouvelle méthode de programmation appelée POO.

Le concept objet logiciel est né du besoin de modéliser des objets du monde réel.

Tout objet du mode réel peut être représenter par un objet, qu'il s'agisse d'un objet vivant, ou d'un objet construit.

Par exemple, prenons un véhicule industriel à automatiser.

- Un véhicule possède plusieurs comportements : avancer, reculer, charger des éléments, ...
- Un véhicule possède des caractéristiques propres : Vitesse, poids, taille, résistance, ...

Pour représenter ce véhicule dans l'approche objet :

- Les comportements seront représentés par des méthodes.
- Les caractéristiques seront représentées par des variables.

b) avantages

Le premier avantage du concept d'objets est que tout le code qui n'a rien à voir avec les vaisseaux spatiaux sont situés dans un seul endroit. Un autre avantage est que les objets peuvent posséder les attributs inhérents de la classe qu'ils appartiennent; par exemple, des vaisseaux spatiaux et des astéroïdes les deux pourraient avoir une position XY car tous les objets qui appartiennent à la classe d'objets en mouvement ont une position XY. Ecrire du code est souvent plus facile parce que vous pouvez conceptualiser comme quelque chose qui se passe à un objet. Un autre avantage est que la POO fait de grands programmes plus

faciles à gérer. Si toutes les fenêtres appartiennent à une hiérarchie des fenêtres des classes et tout le code qui fait référence à une fenêtre particulière est avec cet objet de la fenêtre, toutes les manipulations de fenêtres peuvent être écrites que le transfert de messages simples.

c) inconvénients

Tous les programmes ne peuvent être modélisés avec précision par le modèle des objets. Si vous voulez juste de lire certaines données, faire quelque chose de simple à elle et à écrire en arrière, vous avez pas besoin de définir des classes et des objets. Cependant, dans certains langages de POO, vous pouvez avoir à effectuer cette étape supplémentaire. Un autre inconvénient est que si vous forcez la langue dans le concept de la POO, vous perdez quelques-unes des caractéristiques des langues utiles comme les «langages fonctionnels." Un autre inconvénient est que le concept de l'un programmeur de ce qui constitue un objet abstrait pourrait ne pas correspondre à la vision d'un autre programmeur. Les objets ont souvent besoin d'une documentation complète.

3) Leurs Différences

	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

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.

Les namespaces font alors en sorte de créer comme des répertoires abstraits qui permettent d'encapsuler les éléments et prévenir ainsi tout risque de collision. Toutefois, ils ont aussi un autre rôle qui consiste à créer des alias (des noms alternatifs) plus simples pour les éléments qui ont des noms trop longs ou trop compliqués.

Les namespaces ne sont pas propres qu'au PHP, mais de nombreux langages les prennent en charge. D'ailleurs l'implémentation des namespaces en PHP est inspirée du C++.

On peut se servir des espaces de noms quand on veut, mais les développeurs expérimentés les déconseillent sur des petits projets, car on risque de se retrouver avec de nombreux namespaces qui encapsulent chacun un ou deux éléments, ce qui peut donner lieu à du travail supplémentaire qui consiste à se déplacer entre ces espaces là bien que le risque de collision n'était pas vraiment considérable. Donc, le jeu n'en vaut pas vraiment la chandelle. Cependant, on peut y recourir dès les cas suivants:

- **Dans un gros projet:** où le nombre de fonctions, de constantes et de classes mises en jeu est énorme, donc le risque de collision est élevé.
- **Dans des projets qui utilisent des librairies externes:** dans ce cas, il se peut que l'on utilise des éléments qui peuvent avoir le même nom que leur homologues qui sont embarquées dans la librairie.

Toutefois, il n'existe pas une règle fixe qui dicte quand est ce qu'on doit vraiment se servir des namespaces. Alors, ça revient à vous d'estimer quand est ce qu'ils pourront vous être utiles.

Pour définir un namespace on utilise le mot-clé **namespace** suivi du nom de celui-ci. Le nom peut être choisi de la même façon qu'une fonction (lettres, chiffres et caractères souligné).

Exemple:

```
<?php
    namespace monEspace;
?>
```

2. Classe

Une classe est une entité regroupant des variables et des fonctions. Chacune de ces fonctions aura accès aux variables de cette entité. Une classe représente donc une catégorie d'objets. Elle apparaît aussi comme un *moule* ou une *usine* à partir de laquelle il est possible de créer des objets ; c'est en quelque sorte une « boîte à outils » qui permet de fabriquer un objet. On parle alors d'un objet en tant qu'*instance* d'une classe (création d'un objet ayant les propriétés de la classe).

a) Instanciation

Une *instanciation*, c'est le fait d'*instancier* une classe. *Instancier* une classe, c'est se servir d'une classe afin qu'elle nous crée un objet. En gros, une instance est un objet.

b) Classe abstraite

Les classes abstraites représentent généralement un concept abstrait ou une entité avec une implémentation partielle ou nulle. Par conséquent, les classes abstraites agissent comme des classes parent desquelles les classes enfant sont dérivées, de sorte que la classe enfant partage les caractéristiques incomplètes de la classe parent et que des fonctionnalités puissent être ajoutées pour les compléter.

c) Classe concrète

Une classe concrète est une classe qui a une implémentation pour toutes ses méthodes. En outre, il ne définit aucune méthode abstraite en soi. Cela signifie qu'une instance de la classe peut être créée/allouée avec le mot-clé new sans avoir à implémenter de méthode au préalable. Par conséquent, on peut en déduire que toute classe qui n'est pas une classe ou une interface abstraite est une classe concrète.

c) Objet

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.

d) attribut

Les attributs (aussi appelés données membres») sont les caractères propres à un objet.

Une personne, par exemple, possède différents attributs qui lui sont propres comme le nom, le prénom, la couleur des yeux, le sexe, la couleur des cheveux, la taille...

e) méthode

Les méthodes décrivent les opérations qui sont applicables aux instances de la classe.

Un exemple : les étudiants

Supposons que chaque étudiant soit caractérisé par sa note en mathématiques (NoteMath) et sa note en informatique (NoteInfo). Un étudiant doit pouvoir effectuer éventuellement des opérations de calcul de ses moyennes dans ces deux matières (MoyMath, MoyInfo) et connaître sa moyenne générale calculée à partir de ces deux notes (MoyTotale).

- Constructeur

Le constructeur est une méthode particulière. C'est elle qui est appelée implicitement à la création de l'objet (instanciation).

Dans notre exemple, le constructeur n'a ni paramètre ni instruction. Le programmeur est libre de définir des paramètres obligatoires à passer au constructeur ainsi qu'un groupe d'instructions à exécuter à l'instanciation de la classe. Nous nous en passerons pour simplifier notre exemple.

- Static

Une **variable statique** est une **variable** qui a été **allouée** "statiquement", ce qui signifie que sa **durée de vie** (ou "étendue") correspond à l'exécution complète du programme.

f) Création et Accès

- Déclaration d'une classe, attributs et méthodes

```
Déclaration d'une classe PHP 5
<?php

class NomDeMaClasse
{
    // Attributs

    // Constantes

    // Méthodes
}

?>
```

```
<?php

class Personne
{
    // Attributs
    public $nom;
    public $prenom;
    public $dateDeNaissance;
    public $taille;
    public $sexe;

    // Constantes
    const NOMBRE_DE_BRAS = 2;
    const NOMBRE_DE_JAMBES = 2;
    const NOMBRE_DE_YEUX = 2;
    const NOMBRE_DE_PIEDS = 2;
    const NOMBRE_DE_MAINS = 2;

    // Méthodes
    public function __construct() { }

    public function boire()
    {
        echo 'La personne boit<br/>';
    }

    public function manger()
    {
        echo 'La personne mange<br/>';
    }
}
```


- Instanciation d'un objet

Création d'objets de type Personne

```
<?php

$personne1 = new Personne();
$personne2 = new Personne();
$personne3 = new Personne();
$personne4 = new Personne();
```

- Accès aux attributs

Utilisation des attributs d'un objet

```
<?php

// Définition des attributs de la personne 1
$personne1->nom = 'Hamon';
$personne1->prenom = 'Hugo';
$personne1->dateDeNaissance = '02-07-1987';
$personne1->taille = '180';
$personne1->sexe = 'M';

// Définition des attributs de la personne 2
$personne2->nom = 'Dubois';
$personne2->prenom = 'Michelle';
$personne2->dateDeNaissance = '18-11-1968';
$personne2->taille = '166';
$personne2->sexe = 'F';

// Définition des attributs de la personne 3
$personne3->nom = 'Durand';
$personne3->prenom = 'Béatrice';
$personne3->dateDeNaissance = '02-08-1975';
$personne3->taille = '160';
$personne3->sexe = 'F';

// Définition des attributs de la personne 4
$personne4->nom = 'Martin';
$personne4->prenom = 'Pierre';
$personne4->dateDeNaissance = '23-05-1993';
$personne4->taille = '155';
$personne4->sexe = 'M';
```

Affichage du nom et du prénom de la personne 1

```
<?php

echo 'Personne 1 :<br/><br/>';
echo 'Nom : ', $personne1->nom , '<br/>';
echo 'Prénom : ', $personne1->prenom;
```

Résultat d'exécution du code

```
Personne 1 :

Nom : Hamon
Prénom : Hugo
```

- Accès aux méthodes

Appel de méthode sur des objets

```
<?php

$personne1->boire();
$personne2->boire();
$personne3->manger();
$personne4->manger();

?>
```

Résultat de l'exécution du code

```
La personne boit
La personne boit
La personne mange
La personne mange
```

3) Encapsulation

L'**encapsulation** fait référence au regroupement de données avec les méthodes qui opèrent sur ces données, ou à la restriction de l'accès direct à certains des composants d'un objet. L'encapsulation est utilisée pour masquer les valeurs ou l'état d'un objet de données structuré à l'intérieur d'une **classe**, empêchant les parties non autorisées d'y accéder directement. Des méthodes accessibles au public sont généralement fournies dans la classe (appelées "**getters**" et "**setters**") pour accéder aux valeurs, et d'autres classes clientes appellent ces méthodes pour récupérer et modifier les valeurs dans l'objet.

- **Getters**

La **get** méthode renvoie la valeur de la variable.

- **Setters**

La **set** méthode définit la valeur de la variable.

La **get** méthode renvoie la valeur de la variable **name**.

La **set** méthode prend un paramètre (**newName**) et l'affecte à la **name** variable. Le **this** mot-clé est utilisé pour faire référence à l'objet courant.

```
public class Person {  
    private String name; // private = restricted access  
  
    // Getter  
    public String getName() {  
        return name;  
    }  
  
    // Setter  
    public void setName(String newName) {  
        this.name = newName;  
    }  
}
```

4) Surcharge

La **surcharge** (également connue sous le nom de **surdéfinition**, **polymorphisme ad hoc** ou **overloading** en anglais) est une possibilité offerte par certains **langages de programmation** qui permet de choisir entre différentes versions d'une même fonction ou méthode selon le nombre et le **type** des arguments fournis.

La surcharge peut être **statique** (le choix de la version est alors fait en fonction du nombre d'arguments et de leur type statique **déclaré** à la **compilation**) ou **dynamique** (le choix de la version est alors fait en fonction du type dynamique des arguments constaté à l'exécution). La surcharge dynamique est également appelée **dispatch multiple** et une méthode surchargée dynamiquement **multiméthode** .

5) Relation entre classe

a) Navigabilité entre classe

- **OneToMany**

Une relation OneToMany est l'endroit où l'objet source a un attribut qui stocke une collection d'objets cibles et *si* ces objets cibles avaient la relation inverse avec l'objet source, ce serait une relation ManyToOne. Toutes les relations en Java et JPA sont unidirectionnelles, en ce sens que si un objet source fait référence à un objet cible, rien ne garantit que l'objet cible a également une relation avec l'objet source. Ceci est différent d'une base de données relationnelle, dans laquelle les relations sont définies via des clés étrangères et des requêtes telles que la requête inverse existe toujours.

- **ManyToOne**

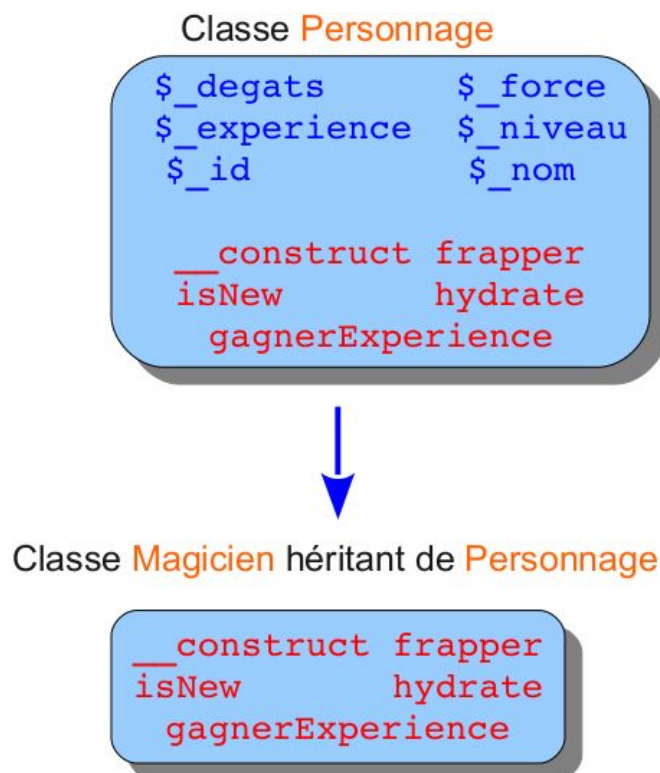
Cette relation est l'inverse de celle **OneToMany**

- **ManyToMany**

Une relation ManyToMany, qui est similaire à une OneToMany, sauf que la relation inverse (si elle était définie) est une relation ManyToMany. La principale différence entre une OneToMany et une ManyToMany est qu'une ManyToMany utilise toujours une table de jointure relationnelle intermédiaire pour stocker la relation, tandis qu'une OneToMany peut utiliser une table de jointure ou une clé étrangère dans la table de l'objet cible référençant la table primaire de la table d'objet source clé.

b) Héritage

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 **filles**.



- Syntaxe

Pour procéder à un héritage (c'est-à-dire faire en sorte qu'une classe hérite des attributs et méthodes d'une autre classe), il suffit d'utiliser le mot-clé `extends`. Vous déclarez

vosre classe comme d'habitude (class MaClasse) en ajoutant extends NomDeLaClasseAHeriter comme ceci :

```
1 <?php
2 class Personnage // Création d'une classe simple.
3 {
4
5 }
6
7 class Magicien extends Personnage // Notre classe Magicien hérite des attributs et méthodes de
  Personnage.
8 {
9
10 }
```

- Redéfinition

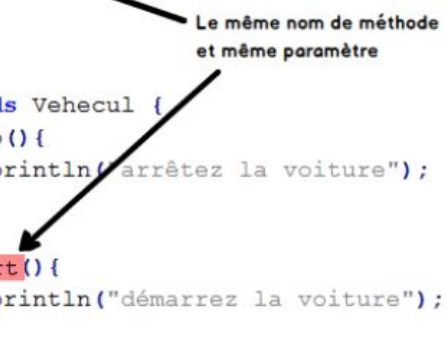
La **redéfinition de fonction** est un concept que l'on rencontre lors de la création de sous-classes. Ici, si vous déclarez une sous-classe et créez une fonction avec le même nom et les mêmes arguments qu'une fonction dans la classe de base, la fonction associée à la sous-classe sera exécutée. C'est parce que vous allez probablement déclarer un objet de la sous-classe. D'une manière générale, la fonction associée à l'objet créé sera exécutée.

Redéfinition

```
class Vehicul {
    public void start() {
        System.out.println("démarrez la véhecul");
    }
}

class Voiture extends Vehicul {
    public void stop() {
        System.out.println("arrêtez la voiture");
    }

    public void start() {
        System.out.println("démarrez la voiture");
    }
}
```



- Polymorphisme

Il y a un concept essentiel en POO désignant la capacité d'une hiérarchie de classes à fournir différentes implémentations de méthodes portant le même nom et par corollaire la capacité qu'ont des objets enfants de modifier les comportements hérités de leur parents. Ce concept d'adaptation à différentes "situations" se dénomme le **polymorphisme**.

Le concept actuel de polymorphisme en POO peut être implémenté à plusieurs niveaux :

❖ Polymorphisme d'Objet:

C'est une interchangeabilité entre variables d'objets de classes de la même hiérarchie sous certaines conditions, que nommons le polymorphisme d'objet.

❖ Polymorphisme par héritage de méthode:

Lorsqu'une classe enfant hérite d'une classe mère, des méthodes supplémentaires nouvelles peuvent être implémentées dans la classe enfant mais aussi des méthodes des parents substituées pour obtenir des implémentations différentes.

❖ Polymorphisme par héritage de classes abstraites:

Une classe abstraite est une classe qui ne peut pas s'instancier elle-même ; elle doit être héritée. Certains membres de la classe peuvent ne pas être implémentés, et c'est à la classe qui hérite de fournir cette implémentation.

❖ Polymorphisme par implémentation d'interfaces:

Une interface décrit la signature complète des membres qu'une classe doit implémenter, mais elle laisse l'implémentation de tous ces membres à la charge de la classe d'implémentation de l'interface.

6) Interfaces

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). Ainsi, une voiture et un personnage n'ont aucune raison d'hériter d'une même classe. Par contre, une voiture et un personnage peuvent tous les deux se déplacer, donc une interface représentant ce point commun pourra être créée.

Une interface se déclare avec le mot-clé `interface`, 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 :

```
1 <?php
2 interface Movable
3 {
4     public function move($dest);
5 }
```

Une interface doit respecter ces conditions :

1. Toutes les méthodes présentes dans une interface doivent être publiques.
2. Une interface ne peut pas lister de méthodes abstraites ou finales.
3. Une interface ne peut pas avoir le même nom qu'une classe et vice-versa.

III. Pattern MVC

Model – view – controller (généralement connu sous le nom de **MVC**) est un **modèle de conception logicielle** couramment utilisé pour

développer **des interfaces utilisateur** qui divise la logique de programme associée en trois éléments interconnectés. Cela permet de séparer les représentations internes des informations de la manière dont les informations sont présentées et acceptées par l'utilisateur. Ce type de motif est utilisé pour concevoir la mise en page de la page.

- La Vue

Ce que l'on nomme la vue est en fait une IHM. Elle représente ce que l'utilisateur a sous les yeux. La vue peut donc être :

- ❖ une application graphique `Swing`, `AWT`, `SWT` pour Java (`Form` pour C#...);
- ❖ une page web ;
- ❖ un terminal Linux ou une console Windows ;

- Contrôleur

Accepte l'entrée et la convertit en commandes pour le modèle ou la vue.

En plus de diviser l'application en ces composants, la conception modèle-vue-contrôleur définit les interactions entre eux.

Le modèle est responsable de la gestion des données de l'application.

Il reçoit l'entrée utilisateur du contrôleur.

Le contrôleur répond à l'entrée utilisateur et effectue des interactions sur les objets du modèle de données. Le contrôleur reçoit l'entrée, la valide éventuellement, puis transmet l'entrée au modèle.

Comme avec d'autres modèles de logiciels, MVC exprime le «cœur de la solution» à un problème tout en permettant son adaptation à chaque système. Les conceptions MVC particulières peuvent différer considérablement de la description traditionnelle ici.

- Model

L'élément central du motif. Il s'agit de la structure de données dynamique de l'application, indépendante de l'interface utilisateur. Il gère directement les données, la logique et les règles de l'application.

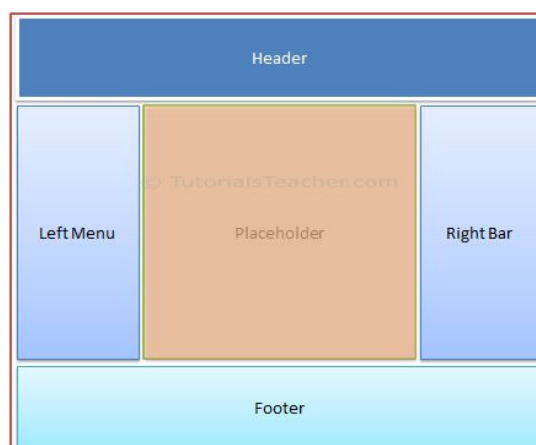
- Router

Le routage est l'acte de faire correspondre une demande à un contrôleur donné.

En règle générale, le routage examinera l'URI de la demande et tentera de faire correspondre le segment de chemin URI aux contraintes fournies. Si les contraintes correspondent, un ensemble de «correspondances» est renvoyé, dont l'un doit être le nom du contrôleur à exécuter. Le routage peut également utiliser d'autres parties de l'URI ou de l'environnement de requête - par exemple, l'hôte ou le schéma, les paramètres de requête, les en-têtes, la méthode de requête, etc.

- Layout ou Template

Celui-ci permet de définir un modèle de site commun, qui peut être hérité dans plusieurs vues pour fournir une apparence cohérente dans plusieurs pages d'une application. La vue de présentation élimine le codage en double et améliore la vitesse de développement et la maintenance facile. La vue de mise en page de l'exemple d'interface ci-dessus contiendrait des sections En-tête, Menu gauche, Barre droite et Pied de page. Il contient un espace réservé pour la section centrale qui change dynamiquement comme indiqué ci-dessous.



- Avantages

- ❖ **La Séparation des tâches**, séparer la logique métier, l'interface utilisateur et la dynamique du système.
- ❖ **La Spécialisation**, une définition claire des zones d'intervention des développeurs. Les développeurs de l'UI peuvent se concentrer exclusivement sur l'interface, sans être gênés par le reste de l'application. De même pour les développeurs de la logique métier ou dynamique du système.
- ❖ **Le Développement**, les *CI(continuous Integration)*, *CD(continuous development)* **en parallèle**, la possibilité pour les développeurs de travailler en parallèle, en même temps sans se marcher dessus.
- ❖ **La Réutilisabilité et gain en temps**, cette option fournit aux développeurs la possibilité d'utiliser et de substituer les Contrôleurs et Vues pour le même Modèle afin de fournir une autre interface. Avec ceci le MVC est très flexible et est un outil puissant pour la représentation des données. Le MVC permet de fabriquer des composants UI qui encouragent la réutilisation et réduisent le nombre de sous-classes spéciales. Il apporte les bénéfices qu'apportent les bonnes pratiques de la POO, il élimine les répétitions inutiles de code et améliore sa réutilisation et sa maintenabilité.

- Inconvénients

- ❖ **Difficultés de conception** cependant, le prix à payer est une augmentation de la complexité de l'architecture. Il est important

de savoir que cette architecture présente des difficultés de conception qui peuvent se présenter au moment de l'implémentation lorsque les différentes équipes ne sont pas coordonnées surtout au niveau de l'interfacage des couches.

- ❖ **Un nombre énorme de fichiers** à manipuler, en fait la séparation des différentes couches nécessite la création de plus de fichiers (3 fois pour les différentes couches). Mais pour des projets de grandes envergures, ce détail semble moins pertinent, si la conception a été bien faite.

- ORM (Object Relational 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.