Introduction aux Concepts Avancés

POO

PHP OBJECTS & CLASSES

- Définir une classe

```php
<?php

class ClassName
{
    //...
}
```

Utilisez le newmot-clé pour créer un objet à partir d'une classe.

```php
<?php

class BankAccount
{
}

$account = new BankAccount();
```

- Ajouter des propriétés à une classe

```php
<?php

class BankAccount
{
    public $accountNumber;
    public $balance;
}
```

Outre le mot-clé `public`, PHP possède également `private` `protected`

- Ajouter des méthodes à une classe

```php
public function methodName(parameter_list)
{
    // implementation
}
```

Les propriétés et les méthodes ont une visibilité.

1. **Le modificateur d'accès `public` vous permet d'accéder aux propriétés et aux méthodes depuis l'intérieur et l'extérieur de la classe.**
2. **Le modificateur d'accès private vous empêche d'accéder aux propriétés et aux méthodes depuis l'extérieur de la classe.(Utilisez des propriétés private avec une paire de méthodes publiques getter/setter.)**
3. **Le modificateur d'accès « Protected » est similaire au modificateur d'accès `Private` . La différence est que les membres de la classe déclarés comme « Protected » sont inaccessibles en dehors de la classe, mais ils peuvent être accessibles par n'importe quelle sous-classe (classe fille) de cette classe.**

La variable `$this` fait référence à l'objet actuel de la classe.

PHP CONSTRUCTORS & DESTRUCTORS

Le constructeur PHP est une méthode spéciale appelée automatiquement lors de la création d'un objet.

```php
<?php

class ClassName
{
    function __construct()
    {
        // implementation
    }
}
```

PHP appelle automatiquement le destructeur lorsque l'objet est supprimé ou que le script est terminé.

```php
<?php

class className
{
    public function __destruct()
    {
        //...
    }
}
```

HÉRITAGES

1. L'héritage permet à une classe de réutiliser le code d'une autre classe sans le dupliquer.

2. Pour définir une classe hérite d'une autre classe, vous utilisez le mot-clé `extends`.

3. Le constructeur de la classe enfant n'appelle pas automatiquement le constructeur de sa classe parent.

4. Utilisez `parent::__construct(arguments)` pour appeler le constructeur parent à partir du constructeur de la classe enfant.

Classe abstraite

- Une classe abstraite ne peut pas être instanciée. Il fournit une interface permettant à d'autres classes d'étendre.
- Une méthode abstraite n'a pas d'implémentation. Si une classe contient une ou plusieurs méthodes abstraites, elle doit être une classe abstraite.
- Une classe qui étend une classe abstraite doit implémenter les méthodes abstraites de la classe abstraite.

```php
<?php

abstract class className
{
   // ...
}
```

Interface

```php
<?php

interface MyInterface
{
    const CONSTANT_NAME = 1;

    public function methodName();
}

class MyClass implements MyInterface
{
    public function methodName()
    {
        // ...
    }
}
```

POLYMORPHISM

Le polymorphisme permet à des objets de différentes classes de répondre différemment en fonction du même message. Pour implémenter le polymorphisme en PHP, vous pouvez utiliser soit des classes abstraites , soit des interfaces .

```php
<?php

abstract class Person
{
    abstract public function greet();
}


class English extends Person
{
    public function greet()
    {
        return 'Hello!';
    }
}

class German extends Person
{
    public function greet()
    {
        return 'Hallo!';
    }
}

class French extends Person
{
    public function greet()
    {
        return 'Bonjour!';
    }
}

function greeting($people)
{
    foreach ($people as $person) {
        echo $person->greet() . '<br>';
    }
}

$people = [
    new English(),
    new German(),
    new French()
];

greeting($people);
```

TRAITS

Un trait est similaire à une classe, mais il sert uniquement à regrouper des méthodes de manière fine et cohérente. PHP ne vous permet pas de créer une instance d'un Trait comme une instance d'une classe. Et il n'existe pas de concept d'instance de trait.

Pour définir un trait, vous utilisez le mot-clé trait suivi d'un nom comme suit :

```php
<?php

trait Logger
{
    public function log($msg)
    {
        echo '<pre>';
        echo date('Y-m-d h:i:s').':'.'(' . __CLASS__ . ') '.$msg.'<br/>';
        echo '</pre>';
    }
}
```

Pour utiliser un trait dans une classe, vous utilisez le mot-clé use. Toutes les méthodes du trait sont disponibles dans la classe où il est utilisé. Appeler une méthode d'un trait est similaire à appeler une méthode d'instance.

```php
class BankAccount
{
    use Logger;

    private $accountNumber;

    public function __construct($accountNumber)
    {
        $this->accountNumber = $accountNumber;
        $this->log("A new $accountNumber bank account created");
    }
}
```

Utiliser plusieurs traits

```php
<?php

trait Preprocessor
{
    public function preprocess()
    {
        echo 'Preprocess...done' . '<br/>';
    }
```

```
}
trait Compiler
{
    public function compile()
    {
        echo 'Compile code... done' . '<br/>';
    }
}

trait Assembler
{
    public function createObjCode()
    {
        echo 'Create the object code files... done.' . '<br/>';
    }
}

trait Linker
{
    public function createExec()
    {
        echo 'Create the executable file...done' . '<br/>';
    }
}

class IDE
{
    use Preprocessor, Compiler, Assembler, Linker;

    public function run()
    {
        $this->preprocess();
        $this->compile();
        $this->createObjCode();
        $this->createExec();

        echo 'Execute the file...done' . '<br/>';
    }
}

$ide = new IDE();
$ide->run();
```

STATIC METHODS & PROPERTIES

- Les méthodes et propriétés statiques sont liées à une classe et non à des objets individuels de la classe.
- Utilisez le staticmot-clé pour définir des méthodes et des propriétés statiques.
- Utilisez le selfmot-clé pour accéder aux méthodes et propriétés statiques au sein de la classe.

Exemple de méthodes et de propriétés statiques PHP

```php
<?php

class App
{
    private static $app = null;

    private function __construct()
    {
    }

    public static function get() : App
    {
        if (!self::$app) {
            self::$app = new App();
        }

        return self::$app;
    }

    public function bootstrap(): void
    {
        echo 'App is bootstrapping...';
    }
}
```

MAGIC METHODS

Ce sont des méthodes spéciales qui peuvent être définies dans une classe pour permettre un comportement personnalisé dans des situations spécifiques.

- **__toString():**

Cette méthode est appelée lorsqu'un objet est converti en chaîne de caractères. Elle est utile pour personnaliser la représentation d'un objet lorsqu'il est utilisé dans un contexte où une chaîne de caractères est attendue.

```php
class Exemple {
    public function __toString() {
        return "Ceci est une représentation de l'objet Exemple.";
    }
}

$objet = new Exemple();
echo $objet; // Affiche : Ceci est une représentation de l'objet Exemple.
```

- **__call():**

Cette méthode est invoquée lorsque l'appel d'une méthode non déclarée ou inaccessible est effectué sur un objet. Elle permet de définir un comportement personnalisé pour les appels de méthodes qui n'existent pas dans la classe.

- **__invoke():**

Permet à un objet d'être invoqué comme une fonction. Lorsque vous tentez d'invoquer un objet qui a cette méthode définie, cette méthode est appelée.

```php
class Exemple {
    public function __invoke() {
        echo "Objet invoqué comme une fonction.";
    }
}

$objet = new Exemple();
$objet(); // Affiche : Objet invoqué comme une fonction.
```

- **__clone():**

Cette méthode est appelée lorsqu'un objet est cloné avec le mot-clé clone. Elle permet de personnaliser le processus de clonage d'un objet, notamment en réalisant des opérations spécifiques sur les propriétés de l'objet cloné.

Magic Method	Description
__call()	is triggered when invoking an inaccessible instance method
__callStatic()	is triggered when invoking an inaccessible static method
__get()	is invoked when reading the value from a non-existing or inaccessible property
__set()	is invoked when writing a value to a non-existing or inaccessible property
__isset()	is triggered by calling isset() or empty() on a non-existing or inaccessible property
__unset()	is invoked when unset() is used on a non-existing or inaccessible property.
__sleep()	The __sleep() commits the pending data
__wakeup()	is invoked when the unserialize() runs to reconstruct any resource that an object may have.
__serialize()	The serialize() calls __serialize(), if available, and construct and return an associative array of key/value pairs that represent the serialized form of the object.
__unserialize()	The unserialize() calls __unserialize(), if avaialble, and restore the properties of the object from the array returned by the __unserialize() method.
__toString()	is invoked when an object of a class is treated as a string.
__invoke()	is invoked when an object is called as a function
__set_state()	is called for a class exported by var_export()
__clone()	is called once the cloning is complete
__debugInfo()	is called by `var_dump()` when dumping an object to get the properties that should be shown.

WORKING WITH OBJECTS

Pour comparer des objets en PHP, vous pouvez utiliser soit l'opérateur de comparaison (==) soit l'opérateur d'identité (===). Cependant, chaque opérateur se comporte différemment en fonction de deux critères principaux :

- Les objets sont la même instance ou des instances différentes d'une classe
- Propriétés de l'objet et leurs valeurs.Cloning Objects

Critères	=	==
Deux objets référencent la même instance	vrai	vrai
Objets avec des propriétés correspondantes	vrai	FAUX
Objets avec des propriétés différentes	FAUX	FAUX

Exemple:

```php
<?php

class Point
{
    private $x;

    private $y;

    public function __construct($x, $y)
    {
        $this->x = $x;
        $this->y = $y;
    }
}
$p1 = new Point(10, 20);
$p2 = $p1;

if ($p1 === $p2) {
    echo 'p1 and p2 are identical.';
} else {
    echo 'p1 and p2 are not identical.';
}

$p3 = new Point(10, 20);
if ($p1 === $p3) {
    echo 'p1 and p3 are identical.';
} else {
    echo 'p1 and p3 are not identical.';
}
```

Anonymous Classes

Une classe anonyme est une classe sans nom déclaré.

```php
<?php
```

```php
$logger = new class {
    public function log(string $message): void
    {
        echo $message . '<br>';
    }
};

$logger->log('Hello');
```

En interne, PHP génère un nom pour la classe anonyme. Pour obtenir le nom généré, vous pouvez utiliser la fonction get_class().

```php
echo get_class($logger);
```

NAMESPACES

Lorsque votre projet devient complexe, vous devrez intégrer le code des autres. Tôt ou tard, vous constaterez que votre code comporte différentes classes portant le même nom. Ce problème est connu sous le nom de collision de noms .

Pour le résoudre, vous pouvez utiliser des espaces de noms.

Pour définir un espace de noms, vous placez le mot-clé namespace suivi d'un nom tout en haut de la page.

TP:

Crer un projet avec la structure de répertoires suivante.

```
.
├── index.php
└── src
    └── Model
        └── Customer.php
```

Dans Customer.php creer la classe Customer

```php
<?php

namespace Store\Model;

class Customer
{
    private $name;

    public function __construct($name)
```

```
    {
        $this->name = $name;
    }

    public function getName()
    {
        return $this->name;
    }
}
```

Importer une classe à partir d'un espace de noms

```
<?php

require 'src/Model/Customer.php';
use Store\Model\Customer;

$customer = new Customer('Bob');
echo $customer->getName();
```

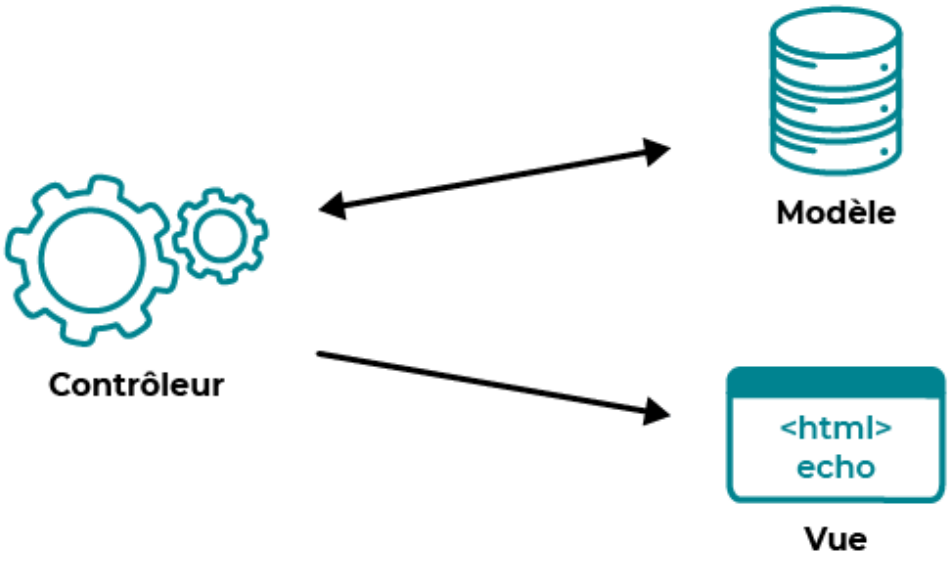
TP AUTOLOADING

MVC

Le pattern MVC permet de bien organiser son code source. Il va vous aider à savoir quels fichiers créer, mais 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.

- **Modèle** : cette partie gère ce qu'on appelle la **logique métier** de votre site. Elle comprend notamment la gestion des données qui sont stockées, mais aussi tout le code qui prend des décisions autour de ces données. Son objectif est de fournir une interface d'action la plus simple possible au contrôleur. On y trouve donc entre autres des algorithmes complexes et des requêtes SQL.
- **Vue** : cette partie 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.
- **Contrôleur** : cette partie gère les **échanges** avec l'utilisateur. C'est en quelque sorte l'intermédiaire entre l'utilisateur, le modèle et la vue. Le contrôleur va recevoir des requêtes de l'utilisateur. Pour chacune, il va demander au modèle d'effectuer certaines actions (lire des articles de blog depuis une base de données, supprimer un commentaire) et de lui renvoyer les résultats (la liste des articles, si la suppression est réussie). Puis il va *adapter* ce résultat et le donner à la vue. Enfin, il va renvoyer la nouvelle page HTML, générée par la vue, à l'utilisateur.

Il est important de bien comprendre comment ces éléments s'agencent et communiquent entre eux. Regardez bien la figure suivante.



Il faut tout d'abord retenir que le contrôleur est le chef d'orchestre : c'est lui qui reçoit la requête du visiteur et qui contacte d'autres fichiers (le modèle et la vue) pour leur demander des services.

Le fichier du contrôleur demande les données au modèle sans se soucier de la façon dont celui-ci va les récupérer. Par exemple : « Donne-moi la liste des 30 derniers messages du forum numéro 5 ». Le modèle traduit cette demande en une requête SQL, récupère les informations et les renvoie au contrôleur.

Une fois les données récupérées, le contrôleur les transmet à la vue qui se chargera d'afficher la liste des messages.

Vous pouvez retenir que le contrôleur sert presque uniquement à faire la jonction entre le modèle et la vue.

Concrètement, le visiteur demandera la page au contrôleur et c'est la vue qui lui sera retournée, comme schématisé sur la figure suivante. Bien entendu, tout cela est transparent pour lui, il ne voit pas tout ce qui

se passe sur le serveur. C'est un schéma plus complexe que ce à quoi vous avez été habitués, bien évidemment : c'est pourtant sur ce type d'architecture que repose un grand nombre de sites professionnels !