

# PHP

## DEUXIÈME PARTIE



# OBJECTIFS DU COURS

Deuxième partie du cours sur PHP. Principaux objectifs :

- Voir les **bases** de l'orienté objet en PHP
- Utiliser PDO pour l'accès à une base de données



Comme précédemment : la présentation part du principe que vous savez déjà programmer dans un langage orienté objet, Java par exemple.

# CLASSES EN PHP

# SYNTAXE GÉNÉRALE

Un exemple

```
<?php
class Question {
    static $num=0;

    protected $id="idQuestion_";
    protected $texte="texte de la question";
    protected $info="Entrez votre réponse";
    protected $reponse="";

    public function __construct() {
        self::$num++;
        $this->id="question_".self::$num;
    }

    public function afficher() {
        echo "<br/><span class='texteQuestion'>". $this->texte. "</span><br/>";
        echo "<input id='$this->id' type='text' placeholder='$this->info'>";
    }
}
```



# SYNTAXE GÉNÉRALE

- Mot clé `class` suivi du nom de la classe puis de la définition des propriétés et des méthodes
- Notion habituelle de **visibilité** :
  - `public`
  - `private`
  - `protected` : fonctionnement « standard », différent de Java, accès pour la classe et les sous-classes.
- `$this` :
  - référence à l'objet appelant comme habituellement, mais ici **obligatoire** pour l'accès aux propriétés et méthodes de la classe
  - opérateur `->` pour accéder aux membres

# SYNTAXE GÉNÉRALE

Un exemple

```
<?php
    require_once "Question.php";
    $q1 = new Question();
    $q1->afficher();
```

- Utilisation d'une classe :
  - `require_once` pour l'importation du fichier
  - `new` pour l'instanciation
  - accès aux membres (propriétés et méthodes) avec l'opérateur `->`, avec respect des niveaux de visibilité

# CONSTRUCTEURS

Un exemple

```
<?php
class Question {
    // .....
    public function __construct() {
        self::$num++;
        $this->id="question_".self::$num;
    }

    // .....
}
```

- **Constructeur** : méthode magique `__construct` (double souligné en préfixe)
- Définition facultative
- Il existe également une méthode destructeur, comme en C++ par exemple : `__destruct()`



# HÉRITAGE

Un exemple

```
<?php
    class QuestionCB extends Question {
        // .....
    }
```

- Mot clé `extends`
- Méthodes et propriétés peuvent être redéfinies (« surcharge »), sans mot clé particulier
- Mot clé « `final` » sur une méthode de la classe mère pour interdire la redéfinition
- Opérateur `parent::` pour accéder à un membre de la classe mère



# PROPRIÉTÉS ET MÉTHODES STATIQUES - SELF

Un exemple

```
<?php
class Question {
    static $num=0;

    public function __construct() {
        self::$num++;
        $this->id="question_".self::$num;
    }
}
```

- Rappels :
  - propriété statique : appartient à la classe, partagée par toutes les instances (« mémoire partagée »)
  - méthode statique : invoquée sans instantiation, par exemple  
`echo MaClasse::MaMethodeStatiqueAfficher();`
- Accès à la classe elle-même : `self::`
- Accès à une propriété statique depuis la classe, un exemple : `self::$num++;`



# CONSTANTE

Un exemple, doc officielle

```
<?php
class MyClass
{
    const CONSTANT = 'valeur constante';

    function showConstant() {
        echo self::CONSTANT . "\n";
    }
}

echo MyClass::CONSTANT . "\n";
```

source : <https://www.php.net/manual/fr/language.oop5.constants.php>

- Une particularité : **pas de \$**. Cela peut éclaircir les messages d'erreur obtenus lorsqu'on oublie le symbole \$ devant une variable.

# CLASSES ABSTRAITES, INTERFACES

- Classes abstraites. Syntaxe :

```
abstract class ClasseAbstraite
{
    // ....
}
```

- Interfaces. Syntaxe de la définition et de l'utilisation :

```
interface iMonInterface
{
    // ....
}
```

```
class UneClasse implements iMonInterface
{
    // ....
}
```



# DES PARTICULARITÉS

- Travail avec des chaînes de caractères

```
<?php
    $instance = new SimpleClass();

    // Ceci peut également être réalisé avec une variable :
    $className = 'SimpleClass';
    $instance = new $className(); // new SimpleClass()
?>
```

<https://www.php.net/manual/fr/language.oop5.basic.php>

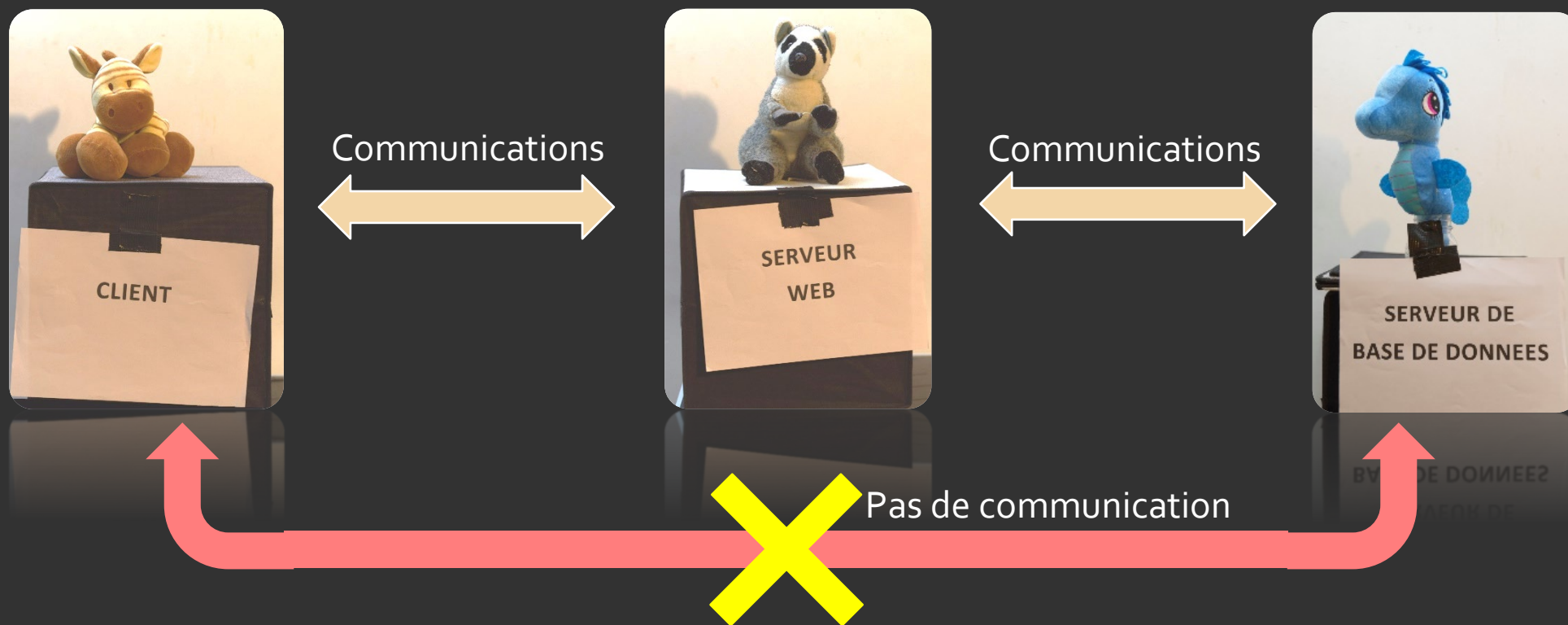
# ACCÈS AUX DONNÉES

# RAPPEL DE L'ARCHITECTURE

- Des clients
- Un serveur web (serveur : au sens logiciel-serveur)
- Un serveur de base de données (idem pour « serveur »)
- Pas de communication directe entre les clients et le serveur de base de données.
- **Attention** : en TP à l'école TOUS les services (les logiciels) sont sur la même machine.



# RAPPEL DE L'ARCHITECTURE



*Rappel : Peluche : le logiciel – Boîte : la machine qui héberge le logiciel.*



# PDO : PHP DATA OBJECTS

- PDO : une **couche d'abstraction** pour l'accès à une base de données :
  - un pilote pour chaque base
  - des fonctions identiques quelle que soit la base (d'où l'abstraction)
- Une des deux principales solutions d'accès aux données pour mysql, l'autre étant mysqli.
- Ne surtout plus utiliser mysql\_





# PDO : CONNEXION

- Se connecter = créer une instance de la classe PDO.
- Exemple pour connexion à MySQL (ou MariaDB) :

```
$maConnexion = new PDO('mysql:host=localhost;dbname=MaBase', 'monLoginBDD', 'monPassBDD');
```

- Remarques :
  - si la même machine héberge le serveur web et le serveur de base de données : **localhost**
  - l'accès via un nom de serveur ou une adresse IP peut être désactivé
  - le login et le mot de passe sont en clair dans le code



# PDO : CONNEXION

- Exceptions : il est de bonne pratique de récupérer les exceptions de PDO lors de la connexion, pour qu'une erreur non traitée ne fasse pas apparaître en clair les paramètres de connexion à la base.

```
try
{
    $maCo = new PDO('mysql:host=localhost;dbname=MaBase', 'login', 'pass');
}
catch (PDOException $e)
{
    echo "Erreur PDO : ".$e->getMessage()."<br/>";
    die();
}
```



# PDO : DÉCONNEXION

- Pour se déconnecter, il faut détruire l'objet PDO (\$maCo). En php cela se fait en s'assurant qu'il n'y a plus de référence sur lui (mécanisme de ramasse-miettes, garbage collector).
- On peut :
  - attendre que l'objet PDO soit détruit, par exemple en fin de script
  - ou bien mettre cet objet à NULL, ainsi que tous les objets qui le référençaient (en particulier les requêtes, de type PDOStatement)

```
$maCo = new PDO('mysql:host=localhost;dbname=MaBase', 'login', 'pass');  
  
//... ..  
  
$maCo = NULL; // déconnexion... sauf s'il reste des références sur $maCo
```



# PDO : REQUÊTES NON PRÉPARÉES

- Deux approches : requêtes préparées et requêtes non préparées
- Requête non préparée : `query`

Un exemple, doc officielle

```
<?php
$sql = 'SELECT name, color, calories FROM fruit ORDER BY name';
foreach ($maCo->query($sql) as $row) {
    print $row['name'] . "\t";
    print $row['color'] . "\t";
    print $row['calories'] . "\n";
}
?>
```

source : <https://www.php.net/manual/fr/pdo.query.php>



# PDO : REQUÊTES NON PRÉPARÉES

- `PDO::query` retourne :
  - un objet de type `PDOStatement`
  - false s'il y a une erreur
- Pour utiliser plusieurs requêtes : fermer le curseur après chacune (dépend des drivers), méthode `closeCursor` de la classe `PDOStatement`.
- Doc de la classe `PDOStatement` :  
<https://www.php.net/manual/fr/class.pdostatement.php>



# PDO : REQUÊTES PRÉPARÉES

- Requêtes préparées : la méthode à privilégier.

Un exemple, doc officielle

```
<?php
$stmt = $dbh->prepare("INSERT INTO REGISTRY (name, value) VALUES (:name, :value)");
$stmt->bindParam(':name', $name);
$stmt->bindParam(':value', $value);

// insertion d'une ligne
$name = 'one';
$value = 1;
$stmt->execute();

// insertion d'une autre ligne avec des valeurs différentes
$name = 'two';
$value = 2;
$stmt->execute();

?>
source : https://www.php.net/manual/fr/pdo.prepared-statements.php
```



# PDO : REQUÊTES PRÉPARÉES

- Principe d'utilisation :
  - Préparation d'un code de requête, avec des paramètres
  - Liaison des paramètres à des variables
  - Autant de fois que nécessaire :
    - Affectation de valeurs dans les variables
    - Exécution

```
<?php
$stmt = $dbh->prepare("INSERT INTO REGISTRY (name, value) VALUES (:name, :value)");
$stmt->bindParam(':name', $name);
$stmt->bindParam(':value', $value);

// insertion d'une ligne
$name = 'one';
$value = 1;
$stmt->execute();

// source : https://www.php.net/manual/fr/pdo.prepared-statements.php
```

Un exemple, doc officielle



# PDO : REQUÊTES PRÉPARÉES

- Récupération des données : méthodes fetch
- On peut choisir la nature de la structure de données retournée, par exemple un tableau associatif : chaque ligne du tableau est un tableau dont les colonnes ont pour noms les noms des colonnes récupérées.

```
<?php
    $sth = $dbh->prepare("SELECT nom, couleur FROM fruit");
    $sth->execute();

    /* Récupération de toutes les lignes d'un jeu de résultats */
    print("Récupération de toutes les lignes d'un jeu de résultats :\n");
    $result = $sth->fetchAll(PDO::FETCH_ASSOC);
    print_r($result);
```

// source : <https://www.php.net/manual/fr/pdostatement.fetchall.php>

Un exemple, doc officielle





# PDO : REQUÊTES PRÉPARÉES

- Exploitation des données (cas où fetch est paramétré pour retourner un tableau associatif)

```
<?php
// en réalité : utiliser un try-catch ici
$maCo = new PDO("mysql:host=localhost;dbname=VOTRE_BASE","VOTRE_UTILISATEUR","VOTRE_PASSWORD");

$texteRequete = "select id, nom, courriel from personnes";
$requete = $maCo->prepare($texteRequete);
$requete->execute();

// récupération du résultat dans un tableau associatif
$tabRes = $requete->fetchAll(PDO::FETCH_ASSOC);

// Si vous voulez mieux comprendre la structure de données retournée :
//var_dump($tabRes);

foreach($tabRes as $uneLigne)
{
    echo $uneLigne['id']. " : ".$uneLigne['nom']. " --> ".$uneLigne['courriel']. "<br>";
}
```

accesDonneesLecture.php  
script de création de la base :  
accesDonneesTest.sql



# PDO : REQUÊTES PRÉPARÉES

- Bénéfices
  - Meilleures performances pour le SGBD
  - Pas besoin de gérer les guillemets selon le type des données
  - Pas d'injection SQL possible

# COMPLEMENTS

# CHIFFREMENT DES MOTS DE PASSE

- Pour chiffrer un mot de passe :

```
password_hash("123456",PASSWORD_DEFAULT);
```

Le deuxième paramètre est l'algorithme employé, consultez la documentation :

<https://www.php.net/manual/en/function.password-hash.php>

- Pour vérifier si une chaîne correspond à un mot de passe :

```
password_verify("123456", $passChiffré)
```

La fonction retourne un booléen, vrai si les mots de passe correspondent, faux sinon.

Doc : <https://www.php.net/manual/en/function.password-verify.php>

**Attention** : aucun algorithme de chiffrement ne peut rendre sûr un mot de passe comme celui de l'exemple...



# CHIFFREMENT DES MOTS DE PASSE

- Exemple d'utilisation :

Création du mot de passe :

1. Préparation d'un mot de passe en clair, via un formulaire ou aléatoirement par exemple, avec respect de règles de qualité
2. Chiffrement avec `password_hash()`
3. Stockage dans la BDD du mot de passe chiffré (hashed password)

Utilisation, sur une demande de connexion par exemple :

1. Saisie d'informations de connexion : login/pass
2. Lecture du mot de passe chiffré dans la base
3. Vérification de la concordance entre le pass saisi et celui extrait de la base avec `password_verify()`



# COOKIES

- Un cookie est un fichier contenant des informations stockées « par le serveur, sur le système de fichiers du client ».
- Les cookies servent pour la gestion des sessions, de façon transparentes, ou, par exemple, pour identifier un visiteur et analyser son activité
- Créer un cookie : `setcookie($nom, $valeur, $dureeVie)`. Exemple, avec une durée de vie d'une heure :

```
setcookie("CookieWebDev", "guili guili", time()+3600);
```



# COOKIES

- Accéder à un cookie se fait avec un tableau associatif :

```
$valeurStockee = $_COOKIE["CookieWebDev"];
```

- Les cookies circulent dans les entêtes HTTP (cf cookie de session) : la commande de création d'un cookie doit être exécutée avant toute autre sortie (texte).



# UPLOAD DE FICHIERS

- php permet de traiter l'upload de fichiers assez simplement,
- Deux références :
- <https://www.php.net/manual/fr/features.file-upload.post-method.php>
- [https://www.w3schools.com/php/php\\_file\\_upload.asp](https://www.w3schools.com/php/php_file_upload.asp)
- **Attention à la sécurité** (par exemple ne pas permettre d'uploader un script php dans un répertoire accessible via une url...)





# QUELQUES FONCTIONS UTILES

- Liste complète des fonctions : <https://www.php.net/manual/fr/indexes.functions.php>
- `phpinfo()` : affiche une page de « diagnostic »
- `var_dump()` : affichage de « debug » d'une variable
- `die()` : arrêt du script
- `max($unTab), min($unTab), count($unTab)`
- `in_array($valeur, $unTab)` : vérifie la présence d'une valeur dans un tableau
- `$unTabModifié = array_unique($unTab)` : supprime les doublons
- Fonctions de tri d'un tableau : <https://www.php.net/manual/fr/array.sorting.php>



# QUELQUES FONCTIONS UTILES

- `$tabRes = explode($unSéparateur, $uneChaine)` : crée un tableau en découpant une chaîne selon un séparateur donné. Une case par « tranche »
- `substr ($chaine , $indiceDebut, $longueur)` : extrait une sous-chaîne
- `date ($format)` et classe `DateTime` : manipulation des dates, voir les docs

<https://www.php.net/manual/fr/function.date.php>

<https://www.php.net/manual/fr/class.datetime.php>

- `rand ()` : valeurs aléatoires, voir la doc selon vos besoins  
<https://www.php.net/manual/fr/function.rand.php>
- `json_encode ($uneVar)` : encodage en json de la variable (sera utilisé dans le cours sur AJAX)



# QUELQUES FONCTIONS UTILES

- fopen, fclose, ... : fonctions de gestion de fichier, voir la documentation <https://www.php.net/manual/fr/ref.filesystem.php>




# ANNEXE : SERVEUR DE TRAVAIL

- Solutions proposées en salles de TP à l'école :
  - **XAMPP**, avec pour racine web C:\WEB
  - **uWamp**, avec pour racine web Z:\
- **Une autre solution** : serveur linux hébergé par VirtualBox
- Un « vrai serveur linux » :
  - pas d'interface graphique de contrôle...
  - ...mais une liberté **d'administration** complète
  - **documentation** à volonté

# ANNEXE : SERVEUR DE TRAVAIL

- VirtualBox :
  - dans le pack installé avec les administrateurs
  - ou à télécharger ici : <https://www.virtualbox.org/>
- Créer le serveur linux :
  - à partir d'images iso d'installation de linux
  - ou utiliser une **V**irtual**M**achine déjà préparée
- Un site qui fournit des VM : <https://www.turnkeylinux.org/>
  - Par exemple LAMP stack : <https://www.turnkeylinux.org/lamp>
  - Télécharger le fichier OVA (appelé VM)

### LAMP Stack



#### Web Stack (MySQL)

LAMP stack is a popular open source web platform commonly used to run dynamic web sites and servers. It includes Linux, Apache, MariaDB (MySQL drop-in replacement), and PHP/Python/Perl. It is considered by many, as the platform of choice for development and deployment of high performance web applications which require a solid and reliable foundation.

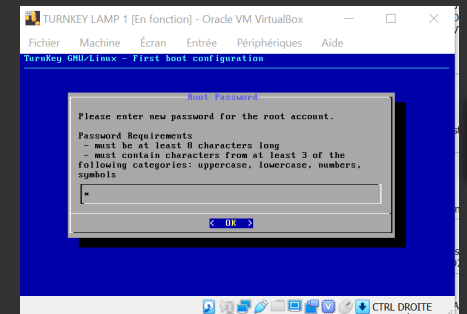
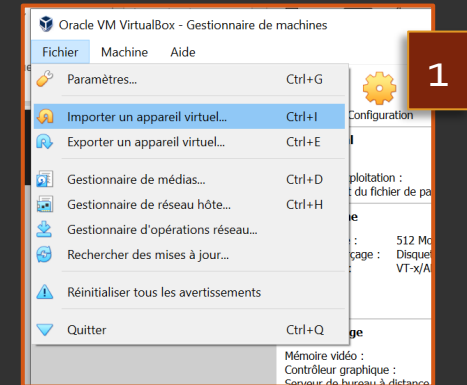
[Run From Browser](#)  
[Deploy Lamp on Amazon EC2](#)

**Builds**  
416MB VM | 327MB ISO | VMDK, OpenStack, Xen, Docker (?) | Manifest & Sigs



# ANNEXE : SERVEUR DE TRAVAIL

- Installation de la VM « LAMP Stack » de TurnkeyLinux (29/11/2019) :
  1. Lancer VirtualBox, puis fichier/Importer VM (1) :
    - aller chercher le fichier OVA
    - définir le répertoire de stockage de la VM : choisir un disque qui dispose de place
    - cliquer sur importer
  2. Démarrer la VM (double clic ou bouton démarrer)
  3. Si la VM capture la souris et le clavier, la touche qui permet de sortir est indiquée en bas à droite (2)
  4. Entrer les mots des passes : ATTENTION, la VM est en QWERTY
  5. Inutile d'activer les mises à jour ou autres services



# ANNEXE : SERVEUR DE TRAVAIL

## ▪ Utilisation :

- lancer la VM : un écran de configuration affiche des IP et ports
- se connecter en SFTP (par exemple avec FileZilla, cf support 1<sup>er</sup> TP) :
  - ip : cf écran de la VM
  - login : root
  - pass : celui que vous avez choisi
  - port : 22
  - (protocole : SFTP)
- racine web : /var/www

