Développement PHP Niveau 1

SQLI INSTITUT

EXPERIENCED BY

SQLI GROUP

Présenté par
Olivier Lonzi

Chapitres

1 Introduction au langage PHP

2 Notre premier projet

3 Les bases de la programmation PHP

4 Programmation Orientée Objet

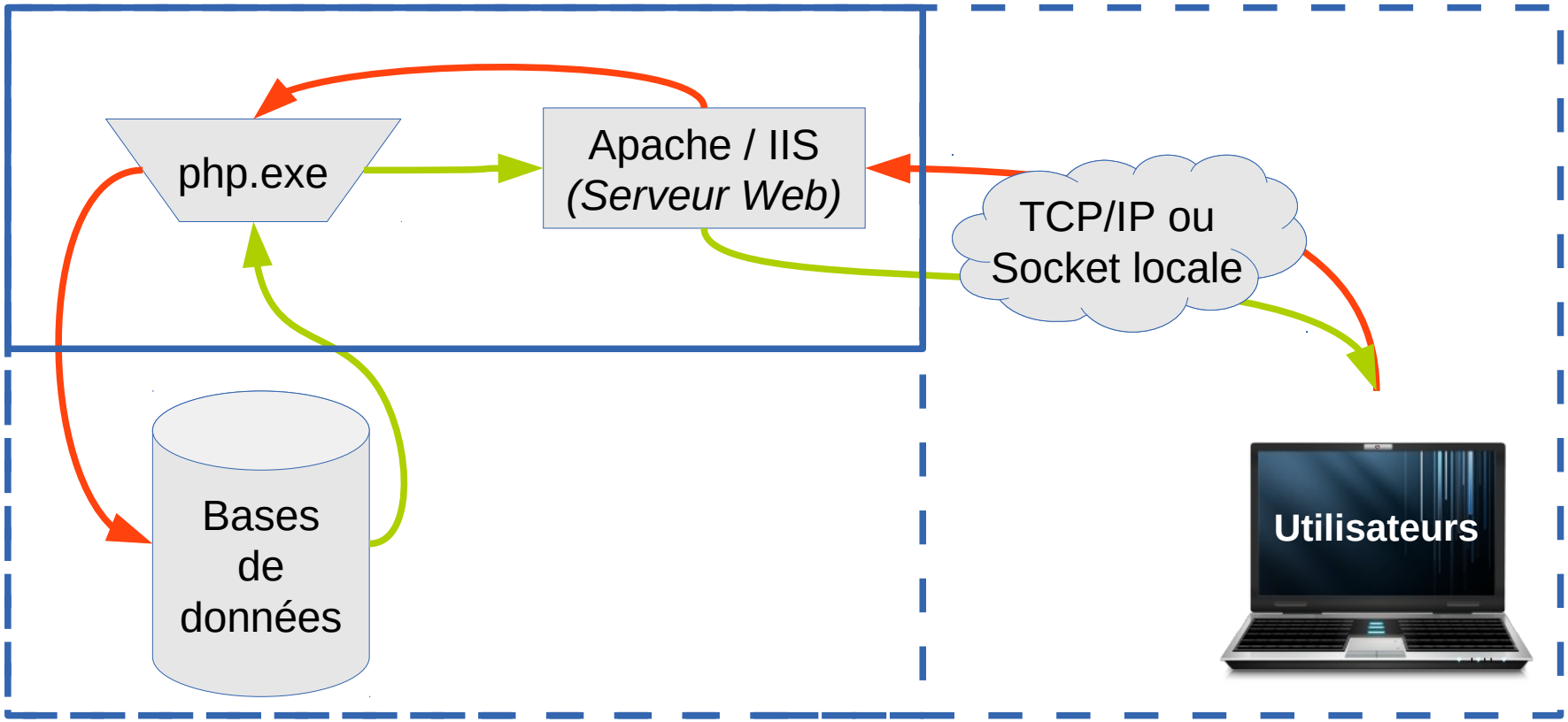
5 Interactions utilisateur et formulaires

6 Authentification et super globales

7 L'accès à une base de données MySQL

8 Validation des acquis

Architecture d'un serveur PHP



Exemple PHP et présentations

+ Fichier « hello_me.php »

```
<?php
        echo "Je suis Olivier LONZI !";
?>
```

+ En ligne de commande

```
$ php -f hello_me.php
Je suis Olivier LONZI !%
```

Les services

- Apache, MySQL, PHP

+ Apache
  - Serveur de requête HTTP,
  - Décide du besoin ou non d'appeler PHP en fonction de l'url saisie par le client,
  - Permet de gérer entre autres les réécritures d'url, qui ont pour but de ne plus voir apparaître « .php » dans l'uri, et d'être mieux référencées.

Les services

- Apache, MySQL, PHP

+ PHP
  - PHP n'est pas un service à proprement parlé, car il est exécuté au besoin, et non dans l'attente en tâche de fond,
  - Interpréteur du langage PHP,
  - Reçois du serveur Web (Apache), un fichier à interpréter, avec ou non des paramètres.

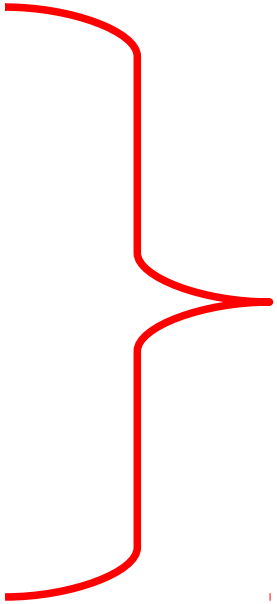
Les services

- Apache, MySQL, PHP

+ MySQL
  - **S**ystème de **G**estion de **B**ases de **D**onnées **R**elationnel,
  - Utilise le langage de requêtes SQL,
  - Gratuit pour une utilisation non commerciale,
  - L'un des SDGBR gratuits les plus populaires de nos jours.

Historique

+ 1995 – PHP/FI
+ 1997 – PHP/FI 2
+ 1998 – PHP 3
+ 2000 – PHP 4
+ 2004 – PHP 5.0
+ 2009 – PHP 5.3
+ 2012 – PHP 5.4
+ 2014 – PHP 5.6
+ 2015 – PHP 7.0
+ 2016 – PHP 7.1



Plus de mise à jour de sécurité

Changements majeurs de PHP

+ *PHP 5.3* – Plusieurs fonctionnalités sont indiquées comme dépréciées, désactivées par défaut.

+ *PHP 5.4* – Des fonctions dépréciées sont supprimées.

+ *PHP 7* – Meilleurs gestion de la mémoire et des processeurs, plus adapté pour les frameworks lourd et les CMS

+ http://php.net/manual/fr/appendices.php

Avantages de PHP

+ Opensource, avec une large communauté,
+ Installation sous de multiples systèmes,
+ Évolutions et mises à jour régulières,
+ Les dernières versions
  – La suppression dans PHP 5.4 de fonctionnalités gardées pour compatibilité permet des gains de performances,
  – Parfaitement adaptées pour une utilisation professionnelle.
  – La meilleur gestion des ressources systèmes dans PHP 7

Les modes de fonctionnements

+ PHP peut être utilisé en ligne de commande
  $ php -r 'echo "Coucou !";'

+ Ou via un serveur Web
  – http://localhost/coucou.php
  – Apache → php -f <racine>/coucou.php

Configuration de PHP

- Premier pas

+ Fichier de configuration php.ini
+ Par exemple, l'option « ***short_open_tags*** »
  - Dépréciée depuis longtemps
  - Désactivée par défaut dans PHP5.3
  - Fonctionnement modifié dans PHP5.4
+ Exemples

```
<?php     echo "Coucou";  ?>
<?        echo "Coucou";  ?>
<?="Coucou"?>
```

si ***sot***=true

PHP ≥ 5.4 *ou* si ***sot***=true

Environnements de développements

+ PHP ne nécessite aucun environnement particulier, cependant pour votre confort un EDI peut être tout indiqué.
  – Netbeans pour PHP
  – Eclipse + PDT
  – Komodo
  – Sublime Text
  – Zend Studio

+ Cependant pour les besoins du cours, un simple éditeur de texte, avec colorisation syntaxique est amplement suffisant :
  – Notepad++
  – Emacs ou Vim ou Notes

Exercice

+ Il existe une fonction PHP qui permet d'avoir toutes les informations sur la configuration du serveur.

+ En utilisant votre moteur de recherche préféré, trouvez cette fonction, et testez-la dans une page PHP visible par un navigateur, ou par une ligne de commande.

Chapitres

Introduction au langage PHP

Notre premier projet

Les bases de la programmation PHP

Programmation Orientée Objet

Interactions utilisateur et formulaires

Authentification et super globales

L'accès à une base de données MySQL

Validation des acquis

Utiliser PHP avec HTML

+ Fichier « coucou.php »

```
<html>
    <head><title>Premier projet</title></head>
    <body><?php echo "Coucou"; ?></body>
</html>
```

+ Que se passe-t-il avec l'extension « .html » ?
+ Autre possibilité d'écriture :

```
<?php
  echo "
      <html>
          <head><title>Premier projet</title></head>
          <body>Coucou</body>
      </html>
   ";
```

Ajouter des commentaires

+ Dans les balises PHP

```
<?php
    // Envoyer la chaîne « Coucou » au navigateur
    /* Attention ne pas confondre un affichage, et un envoi, « echo » peut envoyer des balises HTML, c'est le navigateur qui décide de l'affichage */
    echo "Coucou";
?>
```

+ Rappel : En HTML

```
<!-- Ceci est un commentaire HTML,
à ne pas mettre dans du code PHP -->
```

Enchaînements des instructions

+ Enchaînements d'instructions

```
<?php
    echo "Coucou";
    echo "Comment vas-tu ?";
```

Exemple « short_open_tags »

+ Simplifier avec « ***short_open_tags*** » ?

<? echo "Coucou"; ?> si ***sot***=true

<?="Coucou"?> PHP ≥ 5.4 *ou* si ***sot***=true

+ Le problème
  - « <? » N'est pas dédié à PHP !!
  - Exemple : <?xml … >

+ L'avantage est de pouvoir finalement simplifier l'écriture des courtes insertions d'un code PHP dans du HTML.

Exemple « short_open_tags »

- Exemples

+ Dans le cas de courtes insertions, deux solutions :

Je m'appelle ***<?php echo $prenom; ?>*** ***<?php echo $nom; ?>*** *et travaille à* ***<?php echo $ville; ?>***

Je m'appelle ***<?=$prenom?>*** ***<?=$nom?>*** *et travaille à* ***<?=$ville?>***

+ Attention à l'espace entre le prénom et le nom

<?php ***echo*** *$prenom, ' ', $nom; ?>*

Où trouver de l'aide

- Aide et ressource en ligne

+ *PHP est prédominant sur la toile, principalement car c'est un langage gratuit, relativement facile à apprendre par soi-même, on retrouve donc une pléthore de tutoriels en ligne, plus ou moins bien réalisés !*

+ *Attention, PHP a beaucoup évolué, mais les tutoriels pas forcément, il faudra toujours vérifier que la solution trouvée en ligne soit adaptée à une utilisation « moderne » de PHP.*

Où trouver de l'aide

- Aide et ressource en ligne

+ *Le site de référence incontournable sur le sujet de PHP est*
  - *http://www.php.net/*

+ *Il permet d'avoir des informations sur une fonction très simplement*
  - *http://php.net/phpinfo*

+ *Présente les différentes versions d'une fonctionnalité, ce qui a été ajouté / modifié / supprimé :*
  - *http://docs.php.net/manual/en/ini.core.php#ini.short-open-tag*

Notre premier projet

- Exercice de fin de chapitre

+ **Créer** au moins deux fichiers
  - page1.php
  - page2.php

+ ***Afficher*** *en PHP un lien HyperTextML :*
  - *Vers la page2 depuis la page1*
  - *Vers la page1 depuis la page2*

Chapitres

Introduction au langage PHP

Notre premier projet

Les bases de la programmation PHP

Programmation Orientée Objet

Interactions utilisateur et formulaires

Authentification et super globales

L'accès à une base de données MySQL

Validation des acquis

Les variables

+ Affectation des variables

```
$var   = 55;
$var2 = $var;
$a      = $b      = $c      = 77;
```

+ Typage des variables
  - Le type d'une variable est défini par la valeur affectée.

```
$a = 77; // entier
$a = "string"; // chaîne de caractère
```

Les variables

+ Nommer les variables
  - Les variables sont toujours préfixées du sigle dollars ($)
  - Restrictions
    - Ne commence jamais par un chiffre
    - Éviter les variables commençant par un « _ », même si cela est possible.
    - Se limiter aux lettres, chiffres, et trait-bas
  - Conseils
    - Nommer clairement vos variables, et éviter simplement $i, $o, $coucou,
    - Utiliser des conventions de nommages, comme le *CamelCase* **$plusieursMotsDansLeNom** ou le *snake_case* **$plusieurs_mots_dans_le_nom**

Les constantes

+ Les Constantes
  – Les constantes n'ont aucun préfixe

  – Restrictions
    • L'interprétation du langage PHP ne permet pas de stocker des valeurs autres que les types primitifs
      (entier, booléen, double, chaîne de caractères)

  – Conseils
    • Conventionnellement les constantes sont toujours écrites en lettres capitales,
    • Déclarer les constantes dans des fichiers spécifiques, pour les ajouter en amont de vos traitements.

  – Exemples :
    • define('DB_HOST', 'localhost');
    • define('DB_USER', 'root');

→ Exemple 3-1

Les variables et les types

+ Les types de variables
  - Booléen (boolean)
  - Numérique (integer, float, double)
  - Chaîne de caractères (string)
  - Spéciaux (Array, Object, Resources, …)

```
$a = 77;      // Entier
$b = 0.0;     // Float
$c = "c";     // String
```

+ Tester le type d'une variable

```
is_numeric($a);
is_integer($b);
...
```

+ Concevoir & déboguer avec var_dump()

```
var_dump($a, $b, $c);
```

Valeur nulle

+ Le cas particulier de la valeur nulle
  - Par définition une variable non-définie n'existe pas

    echo $nonExistant; // Génère un Warning PHP

  - On peux définir une valeur nulle, pour éviter cet avertissement PHP

    $nonExistant = null;
    echo $nonExistant; // Rien ne s'affiche (= chaîne vide ici)
    echo 100 + $nonExistant; // 100 + null = 100 (null = 0 ici)

  - /!\ Une variable ne devrait jamais être utilisée sans être définie ; utiliser la valeur passe-partout « null » est une bonne chose.

Opérateurs, et chaîne de caractères

+ Opérateurs arithmétiques (+, -, /, *, %)

```
$a = 5 % 2; // $a = 1;
```

+ Opérateur de concaténation (.)

```
$a = 5;
echo 'La valeur de $a = ['.$a.']';
```

+ Comment déclarer une chaîne de caractères
  - Avec les guillemets

```
$a = "Coucou";
```

  - Avec les apostrophes

```
$a = 'Coucou';
```

  - Avec la syntaxe Heredoc

```
$a = <<<EOT
    Coucou
EOT;
```

Opérateurs, et chaîne de caractères

+ La différence entre les apostrophes et les guillemets, lors d'une concaténation :

```
$b = 'Olivier';
```

  – Avec les guillemets

```
$a = "Coucou $b";
```

  – Avec les apostrophes

```
$a = 'Coucou '.$b;
```

  – Avec la syntaxe Heredoc

```
$a = <<<EOT
    Coucou $b
EOT;
```

  – Le caractère d'échappement (\)

```
"Aujourd'hui" = 'Aujourd\'hui';
"Coucou \$b" = 'Coucou $b';
```

Opérateurs, et chaîne de caractères

+ Heredoc et Nowdoc depuis PHP 5.3
  - La syntaxe Heredoc ne change pas, mais peut être écrite avec des guillemets, comme ceci :

```
$a = <<<"EOT"
   Coucou $b
EOT;
```

  - Nowdoc, lui s'écrira :

```
$a = <<<'EOT'
   Coucou $b
EOT;
```

    Nowdoc n'évaluera pas $b

Opérateurs, et chaîne de caractères

- Raccourcis

+ D'une manière générale on peut raccourcir les opérations sur la variable à affecter quand cette dernière est – elle-même – utilisée pour l'opération :

```
$a = 5;
$a+= 5; // $a = $a + 5
$a*= 2; // $a = $a * 2

$b = 'Bonjour';
$b.= ' le monde'; // Bonjour le monde
```

+ Incrémentation / Décrémentation

```
$c = 0;
$c++;
$c--;
```

+ Ces notations peuvent sembler perturbantes, mais avec des noms de variable longs, cela devient très intéressant de les utiliser.

Opérateurs, et chaîne de caractères

Exercice

+ **Créer** une constante MULTIPLICATEUR à 100

+ **Affecter** des variables et les multiplier par le multiplicateur de 2 façons différentes

+ **Afficher** ces variables en utilisant :
  - Un var_dump();
  - Un echo '...';
  - Une concaténation
  - La syntaxe HEREDOC

Transtypage

+ Outre le fait que PHP ne nécessite pas de déclarer une variable, on a pu constater que celles-ci avaient tout de même un type ; il peut être changer à la volée :

```
$a = 5;          // entier
$a = $a + 0.5;   // $a devient un double
```

+ Qu'en est-il lorsque l'on « transtype » dans un format qui permet de stocker moins d'informations ?

```
$b = 1 + '3 petits cochons';
$c = false + '3 petits cochons';
$d = (int) "j'ai 2 pots de 4 fraises";
```

Tests d'existence

+ Les tests d'existence
  - isset($a)
    - Renvoie vrai si la variable $a a été définie
    - Renvoie faux dans le cas contraire
  - empty($a)
    - Renvoie vrai si la variable est « vide »
    - Renvoie faux dans le cas contraire

```
isset($a);      // = false
$a = 77;
isset($a);      // = true
empty($a);      // = false
```

- http://php.net/empty

→ Exemple 3-5

Les structures de contrôle

- Les conditions

+ La structure de contrôle conditionnelle peut s'écrire de plusieurs manières :

  – if (**condition**) { *si vrai* } else { *si faux* }

```
if ($a <= 25) {
    echo '$a ≤ 25';
} else {
    echo '$a > 25';
}
```

  – **condition** ? *si vrai* : *si faux*; *(Opérateur ternaire)*

```
echo ($lang == "FR" ? 'Bonjour !' : 'Hello!');
```

Les structures de contrôle

- Selon les cas

– Selon les cas (Switch case)

```
switch ($a) {
    case 25:
        echo 'cas 25';
        break;
    case 3:
    case 4:
        echo 'cas 3 et 4';
        break;
    case 'hello':
        echo 'cas hello';
        break;
    default:
        echo 'cas par défaut';
}
```

Les opérateurs de comparaisons

+ Les structures conditionnelles ont besoin d'une **réponse booléenne** pour répondre à la condition, pour cela on utilise entre autre les **opérateurs de comparaisons**

  – Égalité (== ou ===)

```
if (25 == $a)       { echo '$a est égal à 25'; }
if (25 === $a)      { echo '$a est égal à 25 et $a est un entier'; }
if ('25' === $a)    { echo '$a est égal à 25 et $a est une chaîne'; }
```

  – Supérieur / Inférieur (> , >= , < , <=)

```
if ($a <= 25)       { echo '$a est inférieur ou égal à 25'; }
if ($a > 25)        { echo '$a est strictement supérieur à 25'; }
```

  – Différence (!= ou !==)

```
if ($a != 25)       { echo '$a est différent de 25'; }
if ($a !== 25)      { echo '$a n\'est pas un entier OU est différent de 25'; }
```

Les opérateurs de comparaisons

- == ou === ou != ou !==

+ PHP, cherchant à ne pas avoir de type, autorise certains fonctionnements qui, suivant nos habitudes de codage, peuvent sembler impies. Il faut cependant bien les comprendre.

```
if (25 == $a)      echo '$a est égal à 25';
if ($a === 25)   echo '$a est égal à 25 et $a est de type entier';
if ($a === '25') echo '$a est égal à 25 et $a est une chaîne';
```

25 == '25'	VRAI
25 === '25'	FAUX
0 == false	VRAI
0 === false	FAUX
100 == true	VRAI
"coucou" == true	VRAI
"" == true	FAUX
0 !== false	VRAI
0 !== 1	VRAI

Les opérateurs logiques

- && and || or xor !

+ PHP utilise les opérateurs logiques, qui peuvent être écrits de plusieurs façons

&& and [ET]
|| or [OU]
xor [OU exclusif]
! [Négation de la valeur booléenne qui suit]

true && true	VRAI
true || false	VRAI
!true	FAUX
!(true xor true)	VRAI
(25 == '25') && (true xor true)	FAUX
false and true || true	FAUX
isset($nonExist)	FAUX
$a = '25'; is_numeric($a);	VRAI
isset($nonExist) && $nonExist == 25	FAUX

Fonctions & arguments

+ Fonctions natives
  - isset(); // http://php.net/isset
  - is_numeric(); // http://php.net/is_numeric
  - …

+ Fonctions personalisées
  - Les fonctions servent à factoriser du code plus ou moins complexe, pour le ré-utiliser à plusieurs endroits dans un même projet, ou dans des projets différents.

```php
/**
 * Renvoyer le plus petit dénominateur commun entre deux entiers.
 */
function plus_petit_denominateur_commun(int $val1, int $val2) : int {
    // ...
}

// Faire appel à la fonction
echo 'le pdc de 13 et 99 est : ' . plus_petit_denominateur_commun(13, 99);
```

/!\ La déclaration des types n'est possible qu'en PHP 7

Les arguments passés en paramètres

+ Paramètres obligatoires

```
function add($a, $b) {
    return $a+$b;
}
echo add(4, 5);    // 9
echo add(4);       // Erreur, 2 paramètres nécessaires
```

+ Paramètres optionnels

```
function add($a, $b = 2) {
    return $a+$b;
}
echo add(4, 5);    // 9
echo add(4);       // 6 ($b prend la valeur 2, si rien n'est précisée)
```


→ Exemple 3-7

Les arguments passés en paramètres

+ Paramètres et retour avec un type

  – 1) Depuis PHP 7

```
function add(int $a, int $b = 2) : int {
    return $a+$b;
}
```

  – 2) Avant PHP 7, mais toujours valide

```
function add($a, $b = 2) {
     if (!is_int($a) || !is_int($b)) {
          // Erreur fatale
          trigger_error('Typage incorrect', E_USER_ERROR);
     }
     return $a+$b;
}
```

  – Code d'appel de la fonction, toujours le même dans les deux cas

```
echo add(4, 5);    // 9
echo add(4);       // 6
```

Portée et visibilité des variables

+ Dans le cas suivant :

```
$param = 3;
function decrementer($valeur) {
   $valeur = $valeur -1;
   echo $param;
}
decrementer($param);
echo $param;
echo $valeur;
```

+ Que vont afficher les 3 echo ?

+ Porter la variable « $param » en ajoutant cette ligne au début du traitement de la fonction :

```
global $param;
```

  – Quel(s) changement(s) cela apporte(nt) ?

→ Exemple 3-8

Opérateurs, et chaîne de caractères

Exercices

- **Créer** « Exercices/chapitre3/autres/**puissance2.php** »
- **Créer** la fonction « puissance2 », qui renverra la valeur décimale de x à la puissance 2.
  - puissance2(x); // x^2
  - puissance2('toto'); // 0 ou erreur

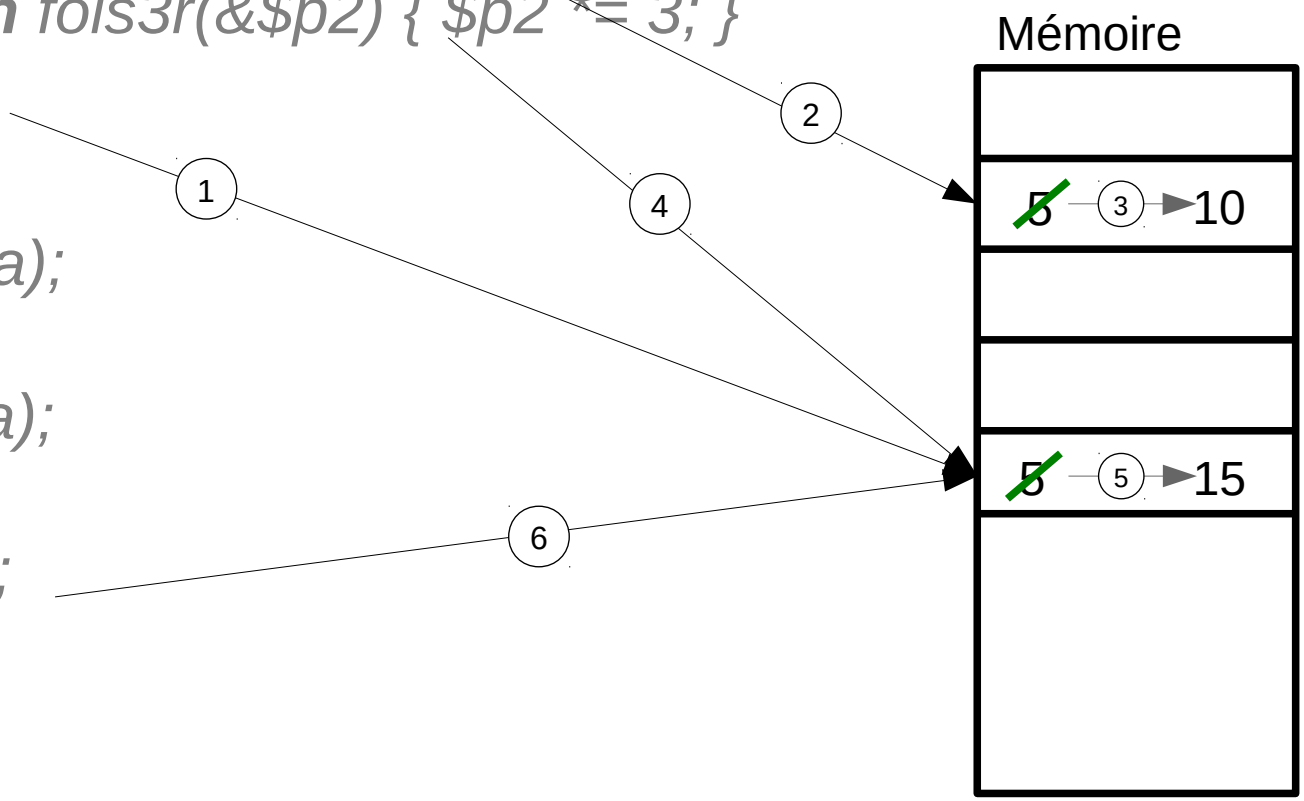
- **Afficher** le résultat concaténé au texte « La valeur [x] à la puissance de 2 vaut : »

x	**Résultat [$x^2 = x \times x$]**
0	0
1	1
2	4
3	9
'toto'	0

Les arguments passés en paramètres

+ ***Passage de paramètres par copie ou par référence***

```
function fois2c(  $p1) { $p1 *= 2; }
function fois3r(&$p2) { $p2 *= 3; }
$a = 5;

fois2c($a);

fois3r($a);

echo $a;
```



Les structures de contrôle

- Les Boucles

+ Les boucles permettent de coder une fois un traitement, que l'on poura répété autant de fois que souhaité.

– Boucle Tant que (While)

```
$a = 0;
while ($a <= 25) {
   echo ++$a, PHP_EOL;
}
```

– Boucle Jusqu'à ce que (Do While)

```
$a = 0;
do {
   echo $a++, PHP_EOL;
} while  ($a <= 25);
```

Les structures de contrôle

- Les Boucles

– Boucle Pour (For)

```
for ($a = 0 ; $a <= 25 ; $a+=2) {
   echo $a, PHP_EOL;
}
```

– Boucle Pour chacun de (Foreach)

```
foreach ($array as $val) {
   echo $val, PHP_EOL;
}
```

Les structures de contrôle

- Les instructions d'arrêts

+ Il existe des instructions « d'arrêt » qui peuvent servir dans toutes les boucles.
  - break;
    - Sort instantanément de la boucle
  - continue;
    - Reprend la boucle au début, sans exécuter la suite
  - exit(1);
    - Stop l'interprétation du code PHP avec le code d'erreur 1
  - die('Message de mort');
    - Affiche un message, et quitte comme exit(0)

+ Bien que cela peut sembler pratique, il faut utiliser ces instructions avec précaution.
+ En effet, un « continue » caché au milieu d'une grosse boucle peut ne pas être vu lors d'une maintenance par une tierce personne, et aboutir à des résultats inattendus.

Opérateurs, et chaîne de caractères

Exercice

+ **Créer** « Exercices/chapitre3/autres/**puissance.php** »
+ **Coder** la fonction « puissance », qui renverra la valeur décimale de x à la puissance y.
  – puissance(x, y); // x^y

+ **Afficher** le résultat concaténé au texte « La valeur [x] à la puissance de [$y] vaut : »

x	***y***	**Résultat [x^y]**
0	*	0
*	0	1
2	3	2×2×2 = 8
2	4	2×2×2×2 = 16
3	3	3×3×3 = 27

Manipulation des tableaux

+ Tout comme les variables primitives, les tableaux en PHP n'ont pas de déclaration, ni d'allocation mémoire, et s'agrandissent aux besoins.

  – Tableau à une dimension et à clé numérique
    - *$tab1 = **array(**'Bonjour', 'le', 'monde', '!'**)**;*

    - *$tab2[0] = 'Bonjour';*
      $tab2[1] = 'le';
      $tab2[50] = 'monde';
      $tab2[25] = 'grand';
      $tab2[] = '!';

  – *var_dump($tab, $tab2);*

Manipulation des tableaux

– Tableau à une dimension et à clé associative

```
$cfg['min']         = 4;
$cfg['max']         = 50;

$cfg2 = array(
            'min' => 4,
            'max' => 50
);

var_dump($cfg, $cfg2);
```

Manipulation des tableaux

– Tableau à plusieurs dimensions :

```
$tab[0][0] = 'case 0,0';
$tab[0][1] = 'case 0,1';
$tab[1][0] = 'case 1,0';
$tab[1][1] = 'case 1,1';
$tab[1][2] = 'case 1,2';

var_dump($tab);
```

– /!\ Contrairement à d'autres langages :

- Le **nombre de colonnes n'est pas** systématiquement le même.

Manipulation des tableaux

+ Particularité des tableaux
  – Les contenus des cellules d'un tableau ne sont pas obligatoirement de même type.
    - *Il faut imaginer chaque cellule comme une variable à part entière.*

  – Lorsque l'on crée une fonction personnalisée, il est possible d'obliger un des paramètres à être du type « tableau » (Avant PHP7).

```
function func(array $tab) { return $tab }
```

Manipulation des tableaux

+ Tableau dans un tableau, dans un tableau, …

```
$tab['fr']['btn']['back'] = 'Retour';
$tab['en']['btn']['back'] = 'Back';
```

ou

```
$tab = array('fr' =>
                array('btn' =>
                        array('back' => 'Retour)
                      )
                'en' => ...
              )
```

ou

```
$fr['btn']['back']= 'Retour';
$en['btn']['back']      = 'Back';
$tab                    = array('fr' => $fr, 'en' => $en);
```

Manipulation des tableaux

+ Fonctions natives sur les tableaux

```
is_array($tab);
[a|k|r|nat]sort($tab);
implode($glue,          $tab);
explode($separator, $string);
in_array        ($search,      $tab,          true);
shuffle         ($tab);
count           ($tab);
array_sum       ($tab);
…
```

  – http://php.net/manual/en/book.array.php

Manipulation des fichiers

+ *Écrire en une fois dans un fichier*
  ***file_put_contents**('filename.ext', $stringData);*

+ *Lire en une fois le contenu d'un fichier*
  *$stringData = **file_get_contents**('filename.ext');*

+ *Lire un fichier, ligne par ligne dans un tableau*
  *$tab = **file**('filename.ext');*

+ *Supprimer un fichier*
  ***unlink**('filename.ext');*

+ Fonctions natives sur les fichiers bruts *(Rarement utilisées)*

```
$fp = fopen('filepath/filename.ext', 'w');
for ($w = 0; $w < strlen($stringData); $w += $nbWrite) {
    $nbWrite = fwrite($fp, substr($stringData, $w));
    if ($nbWrite === false) { die('Erreur'); }
}
fclose($fp);
```

Prémices au modèle MVC

- Modèle-vue-contrôleur

+ Le modèle MVC a pour but de dissocier :
  - la vue (HTML),
  - du langage de programmation (PHP).

+ Il faudra donc développer nos applications, en deux phases minimum :
  - Phase 1 : Effectuer toute la partie dynamique via PHP
  - Phase 2 : Envoyer le résultat sous forme HTML au navigateur

Prémices au modèle MVC

- Modèle-vue-contrôleur

+ Cas sans le modèle MVC

```
<html>
  <head>
    <title>Titre identique sur toutes les pages</title>
  </head>
  <body>
  <?php
    // Récupération du titre et du contenu de la page dans une Bdd
    echo "<h1>$titre</h1>";
    echo "<article>$texte</article>";
  ?>
  </body>
</html>
```

Prémices au modèle MVC

- Modèle-vue-contrôleur

+ Avec le modèle MVC

```
<?php
    // Récupération du titre et du contenu
    //   de la page dans une Bdd
?>
<html>
  <head>
    <title><?=$titre?></titre>
  </head>
  <body>
     <h1><?=$titre?></h1>
     <article><?=$texte?></article>
  </body>
</html>
```

Prémices au modèle MVC

- Modèle-vue-contrôleur

+ On peut même envisager de scinder en plusieurs fichiers :
+ Index.php

```
<?php
    // Requiert impérativement des informations de configuration
    require 'config/config.inc.php';

    // Récupération du titre et du contenu de la page dans une Bdd

    // Affichage du résultat
    include 'templates/common_top.tpl.htm';
    include 'templates/index.tpl.htm';
    include 'templates/common_bottom.tpl.htm';
```

+ templates/**common_top**.tpl.htm

```
<html>
  <head>
    <title><?=$titre?></titre>
  </head>
  <body>
```

+ templates/**index**.tpl.htm

```
    <h1><?=$titre?></h1>
    <article><?=$texte?></article>
```

Cette solution est faiblement MVC, car on perd la logique HTML. Des balises ouvertes dans un fichier, sont fermées dans un autre.

Prémices au modèle MVC

- Modèle-vue-contrôleur

+ Solution avec les templates « **phtml** »
(Mélange PHP & HTML, **le moins de PHP possible**).

+ index.php

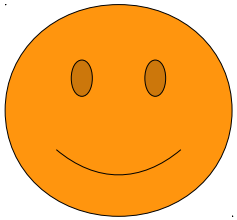
```
<?php
    // Récupération du titre et du contenu de la page dans une Bdd
  $titre = '…' ;    $texte = '...';

    // Génération de la page
   $tpl_filename = 'index';
   require 'templates/layout.phtml';
```

+ **templates/layout.phtml**

```
<html>
  <head><title><?=$titre?></title></head>
  <body><?php require 'templates/'.$tpl_filename.'.phtml'; ?></body>
</html>
```

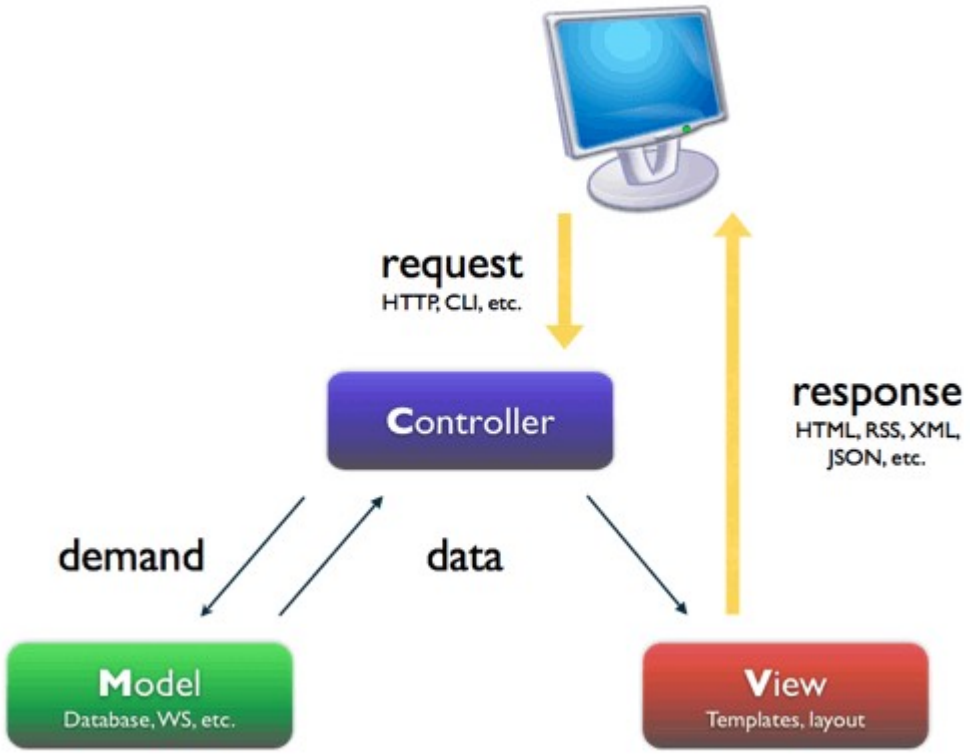
+ templates/index.phtml

```
    <h1><?=$titre?></h1>
    <article><?=$texte?></article>
```



Prémices au modèle MVC

- Modèle-vue-contrôleur



Exercice fin de chapitre

+ **Ouvrir** le projet exercice « Exercices/chapitre3/mini-projet »

+ **Créer** un tableau d'au moins 3 clients
  – Prénom,
  – Nom,
  – Numéro de téléphone,
  – Type de forfait (Fixe, Zen, Play, Jet, …),
  – Données utilisées par ce numéro téléphonique (En Octets).

+ **Parcourir** ce tableau pour afficher les informations en HTML

+ **Déterminer** en fonction du numéro de téléphone :
  – La zone géographique (01, 02, 03, 04, 05),
  – Les mobiles (06, 07),
  – Les numéros spéciaux (08, 09).

+ Pour les plus rapides :
  – **Stocker** le tableau dans un fichier « export.txt »
  – **N'oublier** pas le modèle MVC

Exercice fin de chapitre

+ Résultat attendu

Nom complet	Type forfait	Téléphone	Zone géographique	Données utilisées / max (Mio)
M. Jean ALOUET	Play	0102030405	Région parisienne	0.03 / 100
Mme Alice DELAH	Zen	0607080304	Téléphone mobile	*Pas de données*
Mlle Louise DELAH	Play	0607080305	Téléphone mobile	0.43 / 100
M. Albert DUPONT	Zen	0825010101	Numéro spéciaux	*Pas de données*
M. Durand DUPOND	Jet	0901010101	Pas de zone	32.96 / 500

Chapitres

Introduction au langage PHP

Notre premier projet

Les bases de la programmation PHP

Programmation Orientée Objet

Interactions utilisateur et formulaires

Authentification et super globales

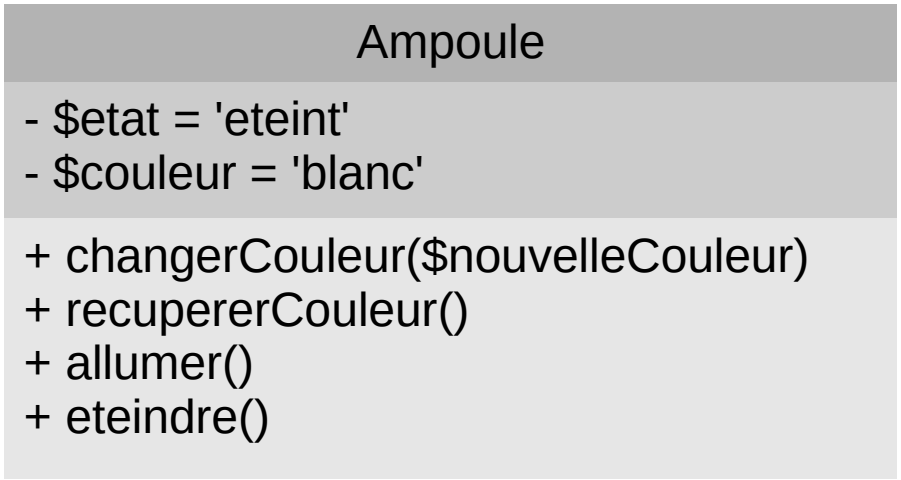
L'accès à une base de données MySQL

Validation des acquis

Programmation Orientée Objet

+ La POO, permet de grouper sous forme d'un ensemble nommée
  - Les attributs de l'ensemble
  - Les comportements de l'ensemble

+ Exemple avec une représentation UML



Programmation Orientée Objet

+ Comment aurions nous fait sans POO ?

```
function ampouleSetColor(&$ampoule, $couleur) {
        if (couleur valide) {
                $ampoule['couleur'] = $couleur;
        }
}
function ampouleGetColor($ampoule) {
        return 'La couleur est : ' . $ampoule['couleur'];
}

$ampoule1 = $ampoule2
        = array('etat' => 'eteint', 'couleur' => 'blanche');

ampouleSetColor($ampoule1, 'Coucou');
echo ampouleGetColor($ampoule2);
```

Programmation Orientée Objet

+ Syntaxe de la POO avec PHP

```
class Ampoule {
        private $etat           = 'eteint';
        private $couleur        = 'blanche';

        public function changerCouleur($couleur) {
                if (couleur valide) { $this->couleur = $couleur; }
        }
        public function recupererCouleur() {
                return 'La couleur est : ' . $this->couleur;
        }
        ...
}

$ampoule1 = new Ampoule();
$ampoule2 = new Ampoule();
$ampoule1->changerCouleur('rouge');
echo $ampoule2->recupererCouleur();
```

Ampoule
- $etat = 'eteint' - $couleur = 'blanc'
+ changerCouleur($nouvelleCouleur) + recupererCouleur() + allumer() + eteindre()

Programmation Orientée Objet

+ Quelles différences ?

  – Avec la *Programmation Orientée Objet* :
    - Nous sommes obligés de passer par les méthodes de la classe pour modifier l'attribut *$couleur*,
    - L'utilisation de l'instance créée avec « *new* » identifie l'objet courant.

  – Avec la *Programmation Procédurale* :
    - La variable globale pourrait être modifiée par n'importe qui, n'importe où dans le code
    - Impossible d'assurer qu'un attribut est lu en utilisant la fonction « *get* » associée.

Programmation Orientée Objet

+ Classe / Instance ?
  - Dans la POO, la notion de « **Classe** » est le **cahier des charges** d'un projet, cela définit le « Ce que l'on pourra faire »,
  - La notion « d'**Instance** » est la **réalisation de ce projet**, cela définit donc une concrétisation d'un cahier des charges,
+ Mais « l'Objet » dans tout ça ?
  - Un objet est l'instance d'une classe

+ Exemples

```
class MaClasse {
        public $attr = 'Coucou';
        public function faireChose() { return $this->attr; }
}

MaClasse->attr = 'Test';     // Erreur car maClasse est une Classe...
MaClasse->faireChose();  // … nous avons besoin d'une instance.


$obj = new MaClasse(); // $obj est un Objet instancié de la classe maClasse...
var_dump($obj instanceof MaClasse); // … comme le prouve ce test


$obj->attr = 'Test Coucou';
echo $obj->faireChose();   // Affiche 'Test Coucou'

$obj2 = new MaClasse();
echo $obj2->faireChose();              // Affiche ?
```


→ Exemple 4-2

Programmation Orientée Objet

+ Protection des attributs et des méthodes
  - Public
    - Tout le monde peut accéder à la ressource (attribut ou méthode)
  - Privé
    - Seules les méthodes de l'instance en cours peuvent accéder à la ressource
  - Protégé
    - Les méthodes des classes héritées pourront accéder à la ressource

```
class MaClasse {
    public          $public;
    private         $prive;
    protected       $protege;
    privatefunction fonctionPrive($param)   { $this->prive = $param; }
    public  function fonctionPublic($param)   { $this->fonctionPrive($param); }
}

$instance = new MaClasse();
$instance->public          = 'Coucou';
$instance->prive           = 'Coucou';   // Erreur fatale
$instance->protege         = 'Coucou';   // Erreur fatale, voir le chapitre sur les héritages
$instance->fonctionPrive('Coucou');  // Erreur fatale
$instance->fonctionPublic('Coucou');
```


→ Exemple 4-3

+ Exercice
  - ***Créer*** *la classe suivante*

Velo
$nbr_roue = 2 # $nbr_vitesse # $taille_roue
+ construire($vitesse, $taille) + ajouter_2_roues() + retirer_2_roues() + getInformationsVelo()

  - ***Conditionner*** *les comportements*
    - *Le nombre de roues doit être 2 ou 4*
    - *Le nombre de vitesse doit être 5,6 ou 8*
    - *La taille des roues ne peux être négative*

Programmation Orientée Objet

+ Les méthodes pré-définies (Surcharges magiques)

  – Constructeur / Destructeur (*__construct() / __destruct()*)

```
$obj = new MaClasse();
unset($obj);
```

  – *Cloneur (__clone())*

```
$obj = new MaClasse();
$obj2 = clone $obj;
```

  – *Transtypeur (__toString())*

```
$obj = new MaClasse();

echo $obj;
// Pas d'erreur si « __toString » renvoie une chaîne de caractères
```

+ Exercice
  - ***Reprendre*** *la classe* ***Velo*** *de l'exercice précédent*
  - ***Modifier*** *les comportements pour utiliser*
    - *Un constructeur*
    - *Un transtypeur*
  - ***Modifier*** *les appels aux comportements, pour prendre en comptes les nouveautés.*

Programmation Orientée Objet

– Accesseurs (*__set() / __get()*)

```
class MaClasse {
    private $params = array();

    public function __set($name, $value) {
        echo ¨Appel de la fonction __set($name, $value)¨;
        $this->params[$name] = $value;
    }
    public function __get($name) {
        echo ¨Appel de la fonction __get($name)¨;
        return $this->params[$name];
    }
}

$obj->attributNonDefini = 'Coucou';
echo $obj->attributNonDefini;
```

– Appeleurs (*__call()*)

```
class MaClasse {
   public function __call($name, $args) {
      echo 'Appel de ['.$name.'] : '
                                    .implode('/ ', $args);
   }
}
$obj = new MaClasse();
$obj->Coucou('Coucou', 35, true);
$obj->TraiterChoses();
```

Programmation Orientée Objet

+ Passage par copie ou par référence ?
  - Dans le chapitre précédent, nous avions vu que les paramètres des fonctions était passés par copie.
  - L'exception est faite avec les objets, ceux-ci sont toujours passés par référence.

```
function incrementer(MaClasse $o) {
   $o->attr++;
}

$obj = new MaClasse();
$obj->attr = 1;

incrementer($obj);

echo $obj->attr; // 2
```


→ Exemple 4-6

Programmation Orientée Objet

+ Les informations « statiques » associées à une classe :
  - Informations que l'on peut manipuler sans avoir besoin d'instancier la classe,
  - Elles ne sont donc pas liées à un objet en particulier, mais à tous les objets instanciés.

+ Exemples

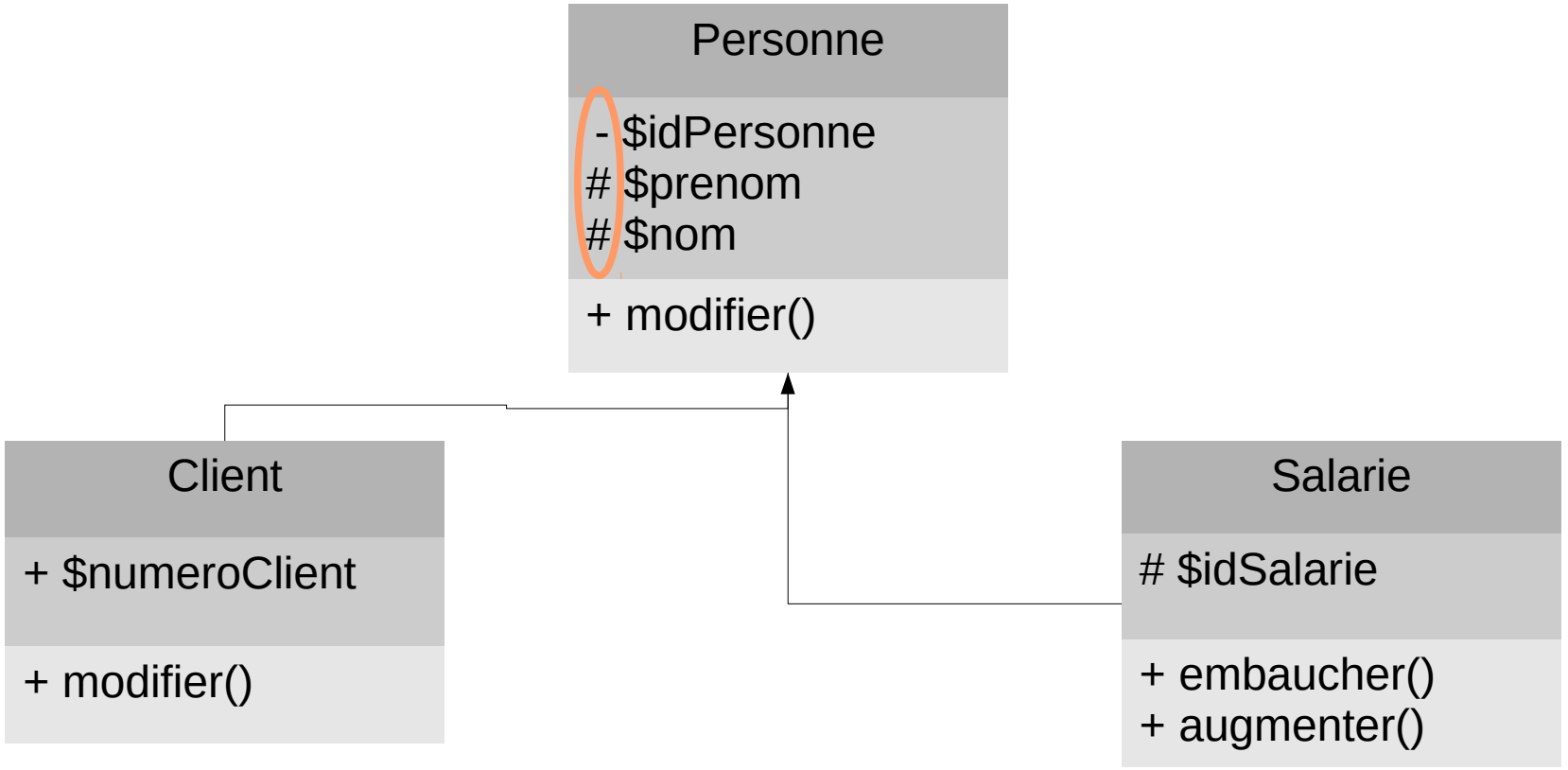
```
class MaClasse {
        static private $count = 0;

                public function __construct()  { self::$count++; }
                public function getCount()    { return self::getStaticCount(); }
        static public function getStaticCount() { return self::$count; }
}

echo MaClasse::getStaticCount();
$obj1 = new MaClasse();          echo $obj1->getCount();
$obj2 = new MaClasse();          echo $obj2->getCount();
```


→ Exemple 4-4

Programmation Orientée Objet

- L'Héritage

+ Pour ne pas ré-écrire plusieurs fois les mêmes morceaux de code qui pourraient être communs, la POO nous permet d'utiliser la notion d'héritage.



Programmation Orientée Objet

- L'Héritage

+ D'un point de vue programmation, voilà comment écrire le schéma précédent

```
class Personne {
   public $nom;
}
class Client extends Personne {
}
class Salarie extends Personne {
}

$client = new Client();
$salarie = new Salarie();
$client->nom = 'Client';
$salarie->nom = 'Salarié';
```

Programmation Orientée Objet

- Classe abstraite

+ Dans l'exemple précédent, nous pourrions vouloir ajouter une contrainte interdisant l'instanciation d'objets de type « Personnes », car toute personne doit être spécifiée via une classe héritée.
+ Pour cela, nous allons utiliser le mot clé « abstract »

```
abstract class Personne { }
class Client extends Personne { }
class Salarie extends Personne { }


$personne = new Personne();      // Erreur fatale
$client   = new Client();                // Ok
$salarie  = new Salarie();               // Ok
```

+ Outre la contrainte en place, le fonctionnement de l'instanciation avec l'héritage reste inchangé.

Programmation Orientée Objet

+ Exercice
  - ***Reprendre*** *la classe* ***Velo*** *de l'exercice précédent*
  - ***Créer*** *les classes Vehicule & Moto*
  - ***Schématiser*** *les connexions entre ces 3 classes*
  - ***Modifier*** *les classes, pour les rendre cohérentes*
    - *Nombre de roue minimum / maximum*
    - *Quelles classes s'occupe des limitations*
    - *…*

Programmation Orientée Objet

- La Composition

+ Parfois nous souhaiterions pouvoir hériter une classe depuis plusieurs autres classes, cela n'est pas possible, mais un fonctionnement similaire et plus malléable est possible avec la composition.

```
class Humain {
	public $nom;
}
class Animal {
	public function ronronner() { echo 'Ronronner !'; };
}
class LoupGarou {
	private $partieAnimale;
	private $partieHumaine;
	function __construct() {
		$this->partieAnimale = new Animal();
		$this->partieHumaine = new Humain();
	}
	function setNom($nom) { $this->partieHumaine->nom = $nom; }
	function actionAnimale() { $this->partieAnimale->ronronner(); }
}

$lg = new LoupGarou();
$lg->nom;			// Pas ok, pas d'héritage
$lg->setNom('Jean Michel');		// Ok, accès grâce à la composition
$lg->ronronner();		// Pas ok, pas d'héritage
$lg->actionAnimale();			// Ok, accès grâce à la composition
```

Programmation Orientée Objet

- Les interfaces

+ Les interfaces ont pour but de définir des méthodes, qui seront obligatoires dans la classe qui va les implémenter, afin de prototyper le développement de nos objets.

```
interface Telephone {
        function __construct($phoneNumber);
        function Appeler();
}
interface GererSms {
        function SmsEnvoyer();
        function SmsRecevoir();
}
class TelephoneFixe implements Telephone {
        public function __construct($phoneNumber) { … }
        public function Appeler()  { … }
}
class TelephoneMobile implements Telephone, GererSms {
        public function __construct($phoneNumber) { … }
        public function Appeler()  { … }
        public function SmsEnvoyer()  { … }
        public function SmsRecevoir()  { … }
}
class Beeper implements GererSms { … }

$tel    = new Telephone();                                  // Erreur fatale
$telF   = new TelephoneFixe('0606060606');          // Ok
$telM  = new TelephoneMobile('0101010101');  // Ok
```

Programmation Orientée Objet

- Les espaces de noms

- *PHP inclut depuis sa version 5.3.0 une notion d'espace de nom, cela permet de scinder notre projet ; chacune de ses parties pourra utiliser des variables, des fonctions, des classes, etc ; toutes de mêmes noms, et pourtant sans risque d'incompréhension par le système.*

- *Fichier « ns1.php »*

```
namespace espace1;
function whoAmI() { return 'espace 1'; }
```

- *Fichier « ns2.php »*

```
namespace espace2;
function whoAmI() { return 'espace 2' ; }
```

- *Fichier « test.php »*

```
require('ns1.php');  require('ns2.php');
echo \espace1\whoAmI();
echo \espace2\whoAmI();
```

Programmation Orientée Objet

- Auto-Chargement des classes

+ Lorsque l'on utilise un projet PHP avec plusieurs dizaines, voire centaines de classes, il peut être fastidieux d'importer tous ces objets lorsque l'on en a besoin.

+ Une première solution est de dire que l'on va importer tous les fichiers de déclaration des classes en amont de notre projet !
  - Cela est lourd au final car parfois nous n'avons besoin que d'un ou deux objets.

+ Une seconde solution est d'utiliser l'auto-chargement avec « spl_autoload_register »;
  *bool **spl_autoload_register** (callable $autoload_function)*

Programmation Orientée Objet

- Auto-Chargement des classes

+ Pour utiliser l'auto-chargement, il faut organiser nos déclarations de classes, le plus souvent, un sous-dossier « classes » de notre application, dans lequel on retrouve un fichier « .php » par classe.
  - classes/
    - Client.php → class Client {}
    - Salarie.php → class Salarie {}
+ On crée la fonction d'auto-chargement

```
function myAutoLoader($className) {
        require ("classes/${className}.php");
}
spl_autoload_register('myAutoLoader');
```

+ Il est parfaitement possible de complexifier la structure de rangement, avec des sous-dossiers pour ranger les sous-classes, il faudra modifier la fonction d'auto-chargement pour faire suivre cela.

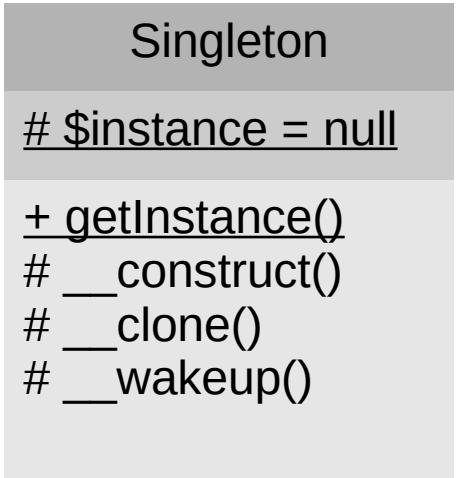
+ Le développement d'application est souvent la réutilisation de plusieurs choses déjà vues et utilisées par d'autres ; inutile de ré-inventer la roue, surtout quand celle-ci nous permet d'avancer plus vite.
+ Le prototypage
  – Le motif de conception « Prototype » est utilisé lorsque la création d'une instance est complexe ou consommatrice en temps. Plutôt que créer plusieurs instances de la classe, on copie la première instance et on modifie la copie de façon appropriée.
+ Le singleton
  – Le singleton est un motif de conception dont le but est de restreindre l'instanciation d'une classe à un seul objet. Il est utilisé lorsque l'on a besoin d'exactement un objet pour coordonner des opérations dans un système.

+ Le prototypage
  - PHP nous offre la méthode « clone() » pour simplifier le prototypage
  - Exemple si l'on souhaite créer plusieurs clients qui ont beaucoup de similitudes (Et dont l'objet généré prendrait plus de temps à créer normalement)

```
class Clients {
   private $prototype = true;
   function __clone() { $this->prototype = false; }
}


$protoClient = new Clients(); // Création du client type
$clientA = clone $protoClient; $clientA->setName(...);
$clientB = clone $protoClient; $clientB->setName(...);
$clientC = clone $protoClient; $clientC->setName(...);
```

Motifs de conception – Singleton

- Singleton
  - Le pattern Singleton a pour but :
    - D'assurer qu'une classe ne possède qu'une seule instance
    - De fournir une méthode de classe unique retournant cette instance.
- Structure



Programmation Orientée Objet

- Motifs de conception

+ Le Singleton
  – Il peut être intéressant parfois de s'assurer que notre application ne comporte qu'un unique objet d'une certaine classe, pour cela nous allons utiliser le Singleton.

```
class Singleton {
   protected static $instance;

   protected function __construct() { }
   protected function __clone() { }
   protected function __wakeup() { }

   public static function getInstance() {
      if ( is_null(self::$instance) )  {
         self::$instance = new self;
      }
      return self::$instance;
   }
}
$objet = new Singleton(); // Erreur fatale
$objet = Singleton::getInstance(); // Ok
```

Programmation Orientée Objet

- Exercice

+ **Ouvrir** le projet « Exercices/chapitre4/mini-projet/ »
  - **Constater** les modifications par rapport à la correction du chapitre précédent

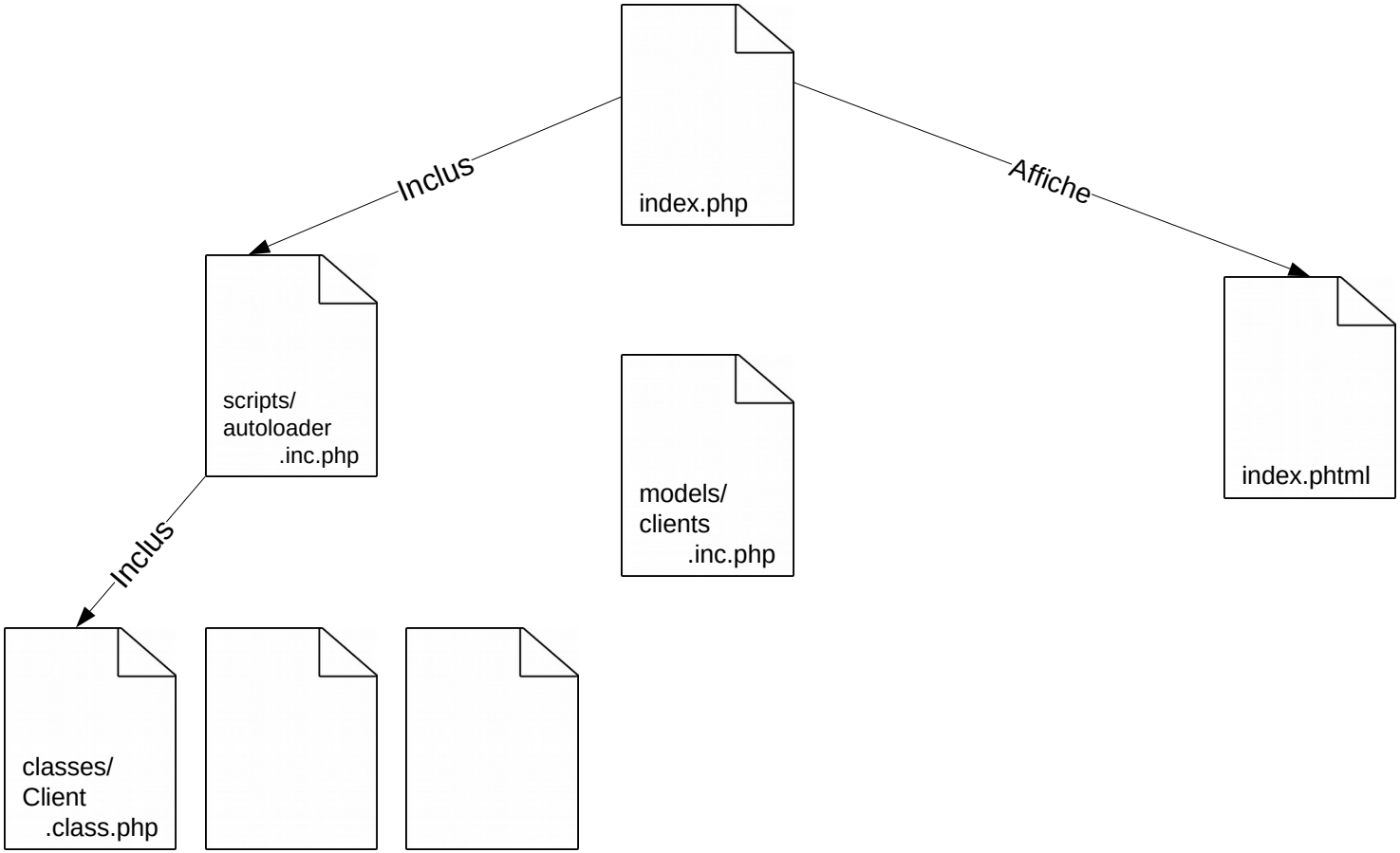
+ **Schématiser** les interactions entres les fichiers

+ **Modifier** le tableau de clients « $tabClients », pour :
  - *Prendre en compte un forfait*
  - *Prendre en compte un numéro de ligne*
  - *Prendre en compte une consommation de données*

+ **Séparer** le projet en plusieurs fichiers logiquement distinct pour respecter le modèle MVC et l'auto-chargement des classes.

Interactions utilisateur et formulaires

- Exercice



Chapitres

Introduction au langage PHP

Notre premier projet

Les bases de la programmation PHP

Programmation Orientée Objet

Interactions utilisateur et formulaires

Authentification et super globales

L'accès à une base de données MySQL

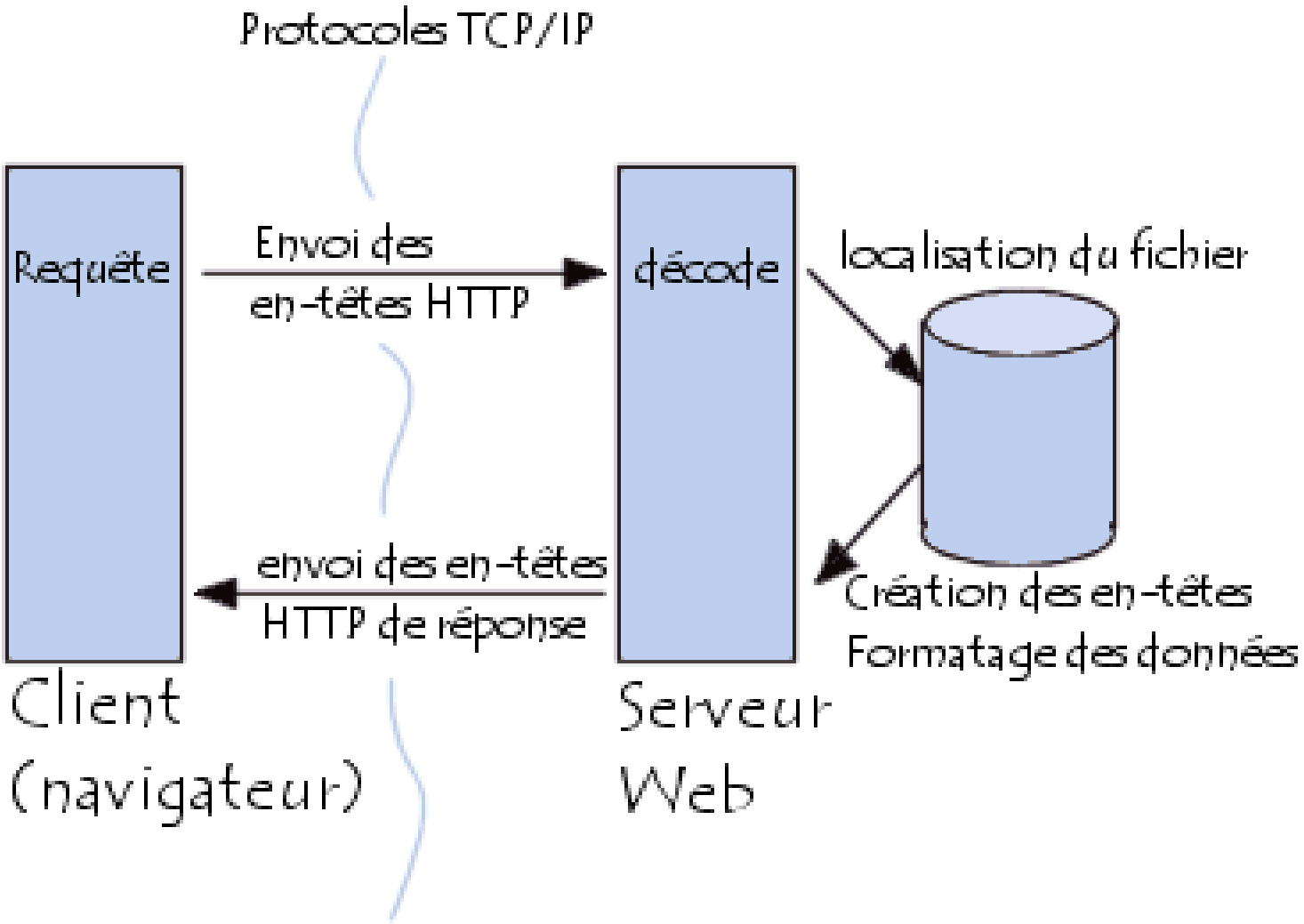
Validation des acquis

Interactions utilisateur et formulaires

+ PHP s'exécute dans un laps de temps court (sauf cas particuliers).
+ Il n'est donc pas fait pour recevoir des paramètres une fois lancé.
+ Pour que les utilisateurs interagissent avec, il faudra utiliser des méthodes associées aux requêtes HTTP, on parlera alors de GET, et POST !
  - Une requête HTTP se décompose en plusieurs lignes d'en-tête.
  - La réponse à cette requête se décompose aussi en plusieurs lignes d'en-tête, mais dispose en plus d'un corps de message, qui correspond au contenu de la page HTML.

Interactions utilisateur et formulaires

- Schéma HTTP



Interactions utilisateur et formulaires

- Protocole HTTP, et paramètres GET/POST

+ Les paramètres GET ou POST sont envoyés dans les en-têtes de la requête HTTP.
  - GET : correspond aux paramètres visibles dans l'url : http://bing.com/?q=coucou
  - POST : correspond aux paramètres « cachés » dans la requête d'en-tête HTTP, souvent après la validation d'un formulaire.

+ L'une comme l'autre, ces deux méthodes permettent l'interaction entre l'utilisateur du site et le langage PHP, soit en passant par des liens hypertextes <a> soit par des formulaires HTML!

Interactions utilisateur et formulaires

- Protocole HTTP, et paramètres GET/POST

+ /!\ Attention, en GET comme en POST, il est très facile pour l'utilisateur de changer les valeurs fournies ; il faudra toujours tester les informations qui proviennent des utilisateurs.

  – Si vous avez un lien hypertexte http://monsite.fr/articles.php?num=30&admin=0
    - N'importe qui peut changer le 0 en 1, pour essayer d'avoir les droits administrateurs.
    - À vous de vérifier s'ils disposent vraiment de ces droits.

Interactions utilisateur et formulaires

- Les formulaires

+ Les formulaires HTML et HTML5

```
<form action='traiter.php' method='post'>
  <input type='text' name='prenom' />
  <input type='submit' value='Valider' />
</form>
```

+ Changer le POST en GET, et constater la différence.

+ Utiliser les super-globales $_POST, ou $_GET pour récupérer les informations.

```
<?php var_dump($_POST['prenom']); ?>
```

Interactions utilisateur et formulaires

- HTML5

+ HTML5 apporte son lot de nouveautés dans la création des formulaires
  - Placeholder
  - Champs courriel
  - Champs date
  - …
  - Exemple : http://www.alsacreations.com/tuto/lire/1372-formulaires-html5-nouveaux-types-champs-input.html
+ /!\ Tous les navigateurs ne gèrent pas ces nouveaux champs, il s'agit donc d'une aide à la création de visuel, vous devrez **TOUJOURS vérifier** les informations fournies après réception des paramètres PHP.

```
<form action='traiter.php' method='post'>
  <input type='email' name='courriel' placeholder='Courriel'  />
  <input type='submit' value='Valider' />
</form>
```

Interactions utilisateur et formulaires

- Sécurité GET/POST

+ Comme on a pu le constater toutes les données fournies par GET ou POST provienne de l'utilisateur et potentiellement celui-ci peut les avoirs altérées (volontairement ou non), malgré des « protections » HTML (ou JS).
  – /!\ Il faudra donc **TOUJOURS** vérifier les données utilisateurs du côté serveur (PHP).

+ Pour sécuriser la transmission des données, la meilleure solution reste de passer par la couche TLS des protocoles TCP/IP.
  – https://coucou.fr/

+ Une autre solution est de chiffrer les données avant la transmission, mais cela ne sécurise que faiblement la transmission.

Interactions utilisateur et formulaires

- Téléversement

+ Le « téléversement d'un fichier » consiste à faire ce que l'on appelle communément « Uploader un fichier sur le serveur », en contraire direct avec le « Téléchargement ».

+ /!\ Le serveur PHP peut limiter la taille des fichiers, il faut regarder le php.ini
  - **file_uploads** : Autorise ou non le téléversement
  - **upload_max_filesize** : Octets max. par fichier
  - **post_max_size** : Octets max. par requête POST
  - **upload_tmp_dir** : Dossier temporaire des fichiers

+ Pour activer le téléversement il faut ajouter un attribut aux formulaires HTML
  `<form enctype='multipart/form-data' … >`

Interactions utilisateur et formulaires

- Exercice

+ **Compléter** le schéma page suivante
  – **Ajouter** les liens entre les fichiers, en fonctions l'étude du code existant

  – **Répondre** aux questions suivantes :
  (Apposer le numéro correspondant au bon fichier, directement dans le schéma)

    1) Dans quel fichier ce trouve le formulaire HTML ?
    2) Dans quel fichier le formulaire est-il traité en PHP ?
    3) Dans quel fichier ce fait l'enregistrement persistant d'un client ?
    4) Dans quel fichier ce fait la récupération persistante d'un client ?

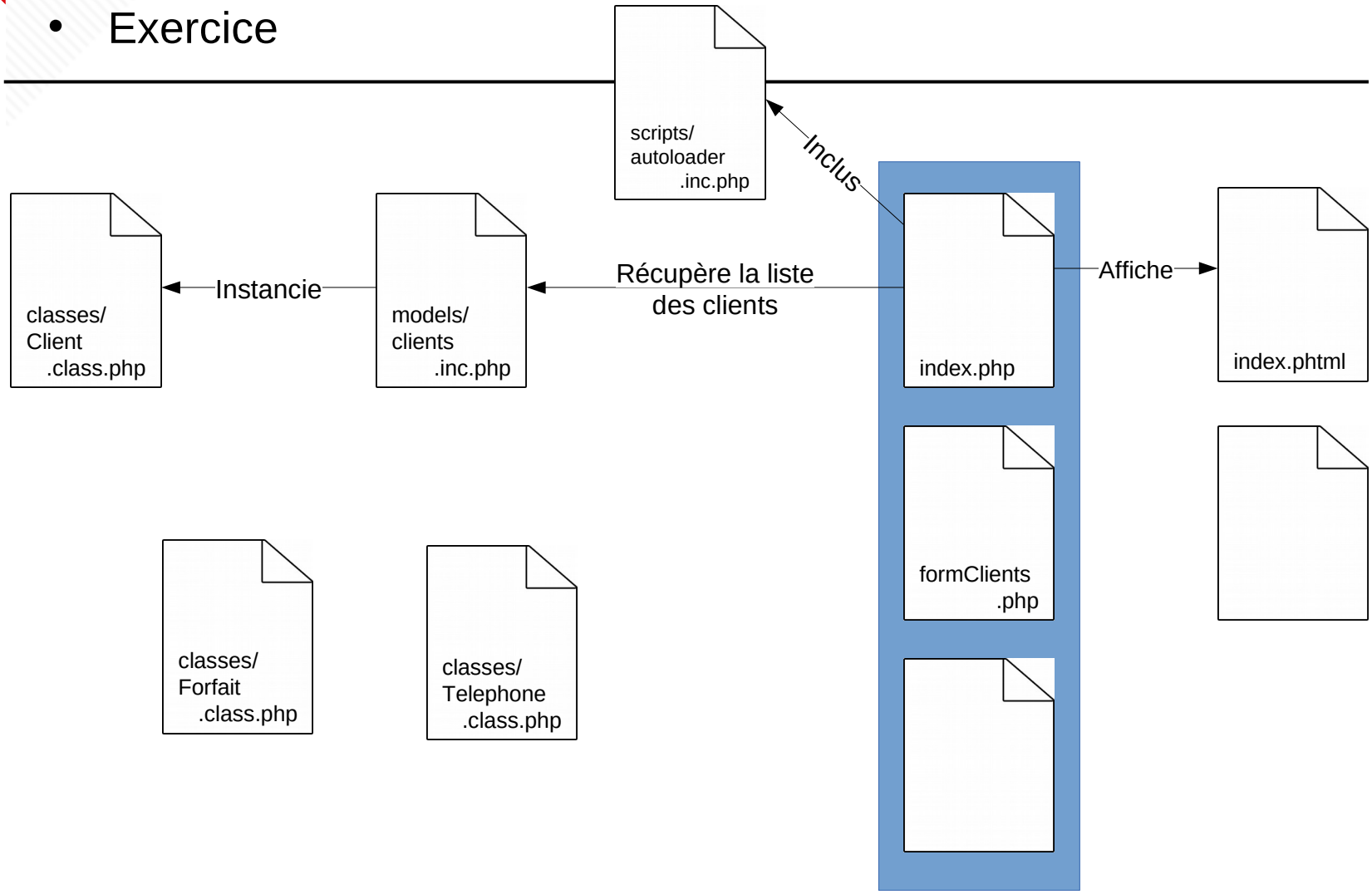
+ **Étudier** le formulaire HTML d'ajout de client

  – **Utiliser** un système de stockage persistant
    - CSV, XML, **Fichiers bruts**, …
    - Pour garder en mémoire les valeurs entre chaque exécution de PHP
    - **Astuce** : Utiliser serialize() & unserialize() dans un fichier brut, pour simplifier le traitement, et vous focaliser sur l'objectif.

+ Pour les plus rapides :
  – **Créer** un formulaire de contact,
    - Fournir la possibilité d'envoyer un fichier pour les utilisateurs ayant un compte client

Interactions utilisateur et formulaires

- Exercice



Chapitres

Introduction au langage PHP

Notre premier projet

Les bases de la programmation PHP

Programmation Orientée Objet

Interactions utilisateur et formulaires

Authentification et super globales

L'accès à une base de données MySQL

Validation des acquis

Authentification et super globales

- Super-Globales

+ En PHP, il existe la notion de variable globale,
  mais surtout de super globale,
  toutes accessibles de n'importe où dans le code source (Objets, Classes, Espace de noms, Fonctions, …) :

  – $_GET : Pour les paramètres passés dans l'url
  – $_POST : pour les paramètres passés dans les en-têtes HTTP
  – $_SERVER : Pour les informations sur le serveur
  – $_SESSION : Pour les paramètres liés à une session PHP
  – $_COOKIE : Pour les valeurs des cookies navigateurs
  – $GLOBALS : Qui regroupe toutes ces super globales

+
```php
<?php
    var_dump($_SERVER, $_COOKIE);
```

Authentification et super globales

- Super-Globale $_COOKIE

+ Les cookies
  - Un cookie n'est pas un biscuit, mais une information stockée sur le navigateur du client, à la demande du serveur.

```
<?php
  if ( !isset($_COOKIE['test']) ) {
      setcookie('test', 'Coucou');
  }
  var_dump($_COOKIE['test']);
```

+ /!\ setcookie utilise les en-têtes HTTP, il ne faut donc pas avoir commencé à envoyer des données aux navigateurs (echo, var_dump, …) !

Authentification et super globales

+ Les sessions
  - Une session définit souvent une connexion authentifiée auprès du serveur après identification par mot de passe ; ce n'est pas systématiquement le cas.

  - En PHP, deux requêtes à deux pages, par un navigateur :
    - Ne garde aucune variable en mémoire
    - Les scripts sont ré-interprétés depuis le début à chaque fois
    - Nous avons vu que pour pallier à cela nous pouvions utiliser les Cookies.

  - Les Cookies sont stockés côté navigateur.

  - Nous pourrions souhaiter avoir une information stockée au niveau du serveur, afin de s'assurer qu'aucun bidouilleur ne puisse le modifier ; c'est le principe des Sessions PHP.

Authentification et super globales

+ Exemple :

  Nous avons # étapes indissociables sur la même page (Panier, Adresse facturation, Adresse livraison, …).

  Impossible d'aller à l'étape suivante sans être passé par l'étape précédente, ou alors tout recommencer.

  – Impossible de stocker l'étape en paramètre GET ou COOKIE, trop facilement modifiable par les visiteurs.

  – Il faut d'une manière ou d'une autre stocker l'information sur le serveur.

Authentification et super globales

- Super-Globale $_COOKIE

+ Solution 1 : Full-Cookie

```
if ( empty($_COOKIE['idCook']) ) {
  $_COOKIE['idCook'] = mt_rand();
  setcookie('idCook', $_COOKIE['idCook']);
}

$idCook = $_COOKIE['idCook'];
if (file_exists("tmp-$idCook.data")) {
    $step = file_get_contents("tmp-$idCook.data") +1;
} else {
    $step = 1;
}



file_put_contents("tmp-$idCook.data", $step);
echo 'Étape '.$step. PHP_EOL;
echo '<a href="pages.php">Passer à l\'étape suivante</a>';
```

Authentification et super globales

- Super-Globale $_SESSION

+ Solution 2 : Session PHP

```
session_start(); // À lancer dès que possible !!

if ( !empty($_SESSION['step']) ) {
  $step = ++$_SESSION['step']; // Pré-incrémentation
} else {
  $step = $_SESSION['step'] = 1;
}
echo 'Étape '.$step. PHP_EOL;
echo '<a href="pages.php">Passer à l\'étape suivante</a>';
```

- On ajoute en début de script « session_start() » sur toutes les pages où nous souhaitons manipuler les sessions.
- La super globale $_SESSION s'occupe du reste, elle stocke sur le serveur les valeurs fournies sous forme de tableau associatif.
- Les sessions utilisent strictement le même principe de cookie que précédemment, si on supprime le cookie, le serveur détecte le navigateur comme un nouvel utilisateur.

Authentification et super globales

- Formulaires d'authentification

+ Le formulaire suivant

```
<form action='ident.php' method='post'>
  <input type='text' name='username' placeholder='Nom d'utilisateur' />
  <input type='password' name='userpass' placeholder='Mot de passe'/>
  <input type='submit' name='M\'authentifier' />
</form>
```

- Et le test PHP suivant

```
session_start();
if ($_POST['username'] == 'admin' and $_POST['userpass'] == 'coucou') {
   $_SESSION['username'] = $_POST['username'];
   header('Location: http://coucou.fr/admin/');
   exit();
} else {
   include ('form.html'); // Afficher le formulaire
}
```

+ La page « admin/index.php »

```
session_start();
if ( !isset($_SESSION['username']) ) require ('404andDie.php');
echo 'Vous êtes sur la page d\'administration';
```

Authentification et super globales

- Formulaires d'authentification

+ Dans l'exemple précédent, le formulaire HTML demande un identifiant et mot de passe, et les transmet en clair via la méthode POST du protocole HTTP

+ Si le couple « admin/coucou » est utilisé alors, une variable de session est initiée, ici pour stocker la valeur du nom d'utilisateur, et l'utilisateur est redirigé vers la page d'administration

+ La page d'administration vérifie avant toute chose que la variable de session est une valeur, si oui alors on affiche la page

+ Il s'agit d'un exemple minimum pour la création d'un formulaire d'authentification, la protection doit être améliorée.

Authentification et super globales

- Exercice

+ **Ajouter** un bouton de déconnexion

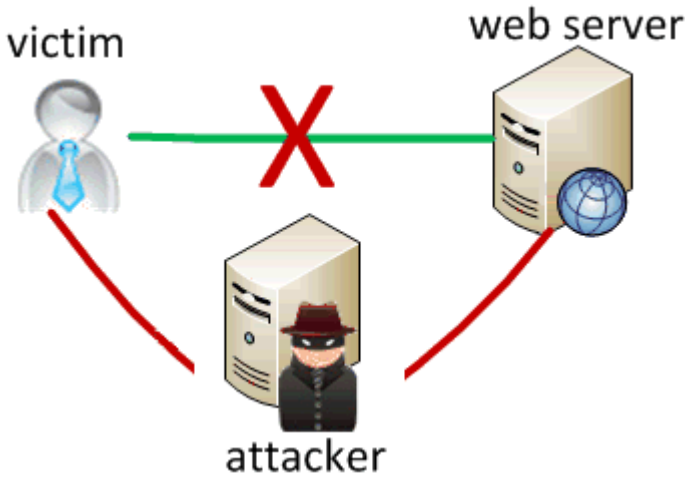
+ **Proposer** des améliorations pour sécuriser au mieux les données du formulaire, puis les informations de sessions une fois initiées.
  – Mettre en place dans la mesure du possible ces améliorations.

+ **Noter** qu'aucun système n'est infaillible.

Authentification et super globales

- Sécurité

+ Une première chose vers laquelle le Web commence fortement à tendre, depuis ces 5 dernières années, c'est de refuser les formulaires de mots de passes sans un accès TLS, et de toujours mettre en place une couche TLS, au minima lorsque vous avez un formulaire avec un mot de passe.
  – Sans la couche TLS, il est aisé pour certains de faire des attaques du type « Man-In-The-Middle », surtout si la victime est sur un réseau public.



Authentification et super globales

- Exercice de fin de chapitre

+ **Pour les plus rapide**
  - **Reprendre** le mini-projet Correction chapitre 5.
  - **Mettre** en place un système d'authentification
    - Pour les administrateurs
      - Accès à la liste des clients, avec ligne téléphonique
    - Pour les clients
      - Accès aux détails de leur fiche uniquement
    - Autres
      - Afficher un message d'erreur, en plus du formulaire pour tenter une nouvelle connexion.

Chapitres

Introduction au langage PHP

Notre premier projet

Les bases de la programmation PHP

Programmation Orientée Objet

Interactions utilisateur et formulaires

Authentification et super globales

L'accès à une base de données MySQL

Validation des acquis

L'accès à une base de données MySQL

- SGBDR

+ SGBDR – Système de Gestion de Bases de Données Relationnel

+ Le choix de MySQL
  - Licence GPL
  - SGBD populaire
  - S'interface avec des requêtes SQL
  - S'utilise nativement ou via PDO dans PHP

L'accès à une base de données MySQL

- SGBDR

+ Rappel des instructions SQL les plus courantes :
  - CREATE / DROP

    ```
    CREATE TABLE coucou (id integer);
    DROP TABLE coucou;
    ```

  - INSERT

    ```
    INSERT INTO coucou (id) VALUES (1), (2);
    ```

  - UPDATE / DELETE

    ```
    UPDATE coucou SET id=100 WHERE id=1;
    DELETE FROM coucou WHERE id=2;
    ```

  - SELECT

    ```
    SELECT * FROM coucou;
    ```

L'accès à une base de données MySQL

- SGBDR

+ Rappel des clauses SQL les plus courantes :

- WHERE

```
SELECT * FROM coucou WHERE id=100;
```

- ORDER BY

```
SELECT * FROM coucou ORDER BY id DESC;
```

- JOIN / LEFT JOIN

```
SELECT * FROM coucou JOIN coucou2 USING (colJoin);
SELECT * FROM coucou JOIN coucou2
            ON (coucou.colJoin = coucou2.colJoin);
SELECT * FROM coucou LEFT JOIN coucou2
            ON (coucou.colJoin = coucou2.colJoin);
```

- GROUP BY

```
SELECT * FROM coucou GROUP BY groupCol;
```

- AS

```
SELECT id AS identifiant FROM coucou;
```


→ Exemple 7-1

L'accès à une base de données MySQL

- SGBDR

+ Se connecter à MySQL en ligne de commande
  – mysql -u root -p

+ Se connecter à MySQL graphiquement
  – PhPMyAdmin *(Dispo dans WampServer)*
  – MySQL Workbench

L'accès à une base de données MySQL

- SGBDR

+ Se connecter à MySQL en PHP
  - Fonctions : mysql_* – Supprimées en PHP 7
  - Fonctions : mysqli_*
  - Objet : Mysqli
  - Objet : PDO – Php Database Object

+ Éxecuter des requêtes
  - mysqli_query('...')
  - Mysqli::query('...')
  - PDO::exec('...')

→ Exemple 7-2

L'accès à une base de données MySQL

- SGBDR

+ Quelle méthodes choisir
  - Fonctions : mysql_*
    - Supprimées en PHP 7
  - Fonctions : mysqli_*
    - Approche Procédural (Sans objet)
  - Objet : Mysqli
    - Approche Objet
    - Des méthodes dédiés à MySQL
  - Objet : PDO – Php Database Object
    - Approche Objet
    - Des méthodes génériques, seule le paramètre de connexions permet de savoir que l'on manipule MySQL.

+ L'injection SQL, et comment s'en prémunir

– Supposons le traitement d'un formulaire d'authentification en PHP, comme suit :

```
$sql      = "SELECT count(*) FROM clients WHERE user='$user' AND pass='$pass'";

$result   = mysqli_query($sql);
$count    = mysqli_result($result, 0, 0);

if ($count == 1) {
        echo 'Vous êtes identifié';
}
```

L'accès à une base de données MySQL

- SGBDR

+ Lors du remplissage du formulaire, il suffit à un méchant pirate, de remplir comme nom d'utilisateur, quelques choses comme cela :
  $user = « **admin' OR '** »

+ Après traitement des variables PHP, la requête deviendrais :
  SELECT count(*) FROM clients WHERE user='**admin' OR ''** AND pass=''

+ La **requête** est **valide**, et **ne demande** finalement **que** de connaître un nom d'utilisateur valide, **sans mot de passe**

+ Pour s'en prémunir, il faut protéger les apostrophes :
  SELECT count(*) FROM clients WHERE user='**admin\' OR \''** AND pass=''

```
$sql = "SELECT count(*) FROM clients WHERE
        user='".mysqli_escape_string($user)."' AND
        pass='".mysqli_escape_string($pass)."'";
```

L'accès à une base de données MySQL

- SGBDR – Requêtes préparées

+ Une solution **encore plus adaptée** pour ne jamais risquer d'oublier cette protection est l'utilisation des **requêtes préparées**.

```
$db     = new PDO('mysql:host=;dbname=', $user, $pass);
$stmt   = $db->prepare('SELECT * FROM coucou WHERE id < :id');
$stmt->execute(array('id' => 3));
```

+ L'utilisation du nommage dans la requête « :id » permet lors de l'exécution de passer une valeur en paramètre « array('id' => 3) », le système s'occupe de protéger toute apostrophe pouvant se trouver là.

```
$tab = $stmt->fetchAll();          // Récupérer un tableau PHP
var_dump($tab);                    // Afficher le tableau pour confirmer
```

+ Cette méthode est d'autant plus intéressante qu'il est possible d'**exécuter** une requête préparée **plusieurs fois de suite**, avec des **paramètres différents**, comme pour une suite d'insertion par exemple.

L'accès à une base de données MySQL

- SGBDR

+ L'utilisation de PDO, n'implique pas systématiquement de préparer les requêtes, il peut être utilisé comme suit :

```
$db = new PDO('mysql:host=;dbname=', $user, $pass);

// exec() pour les requêtes sans retour complexe
// (Ici le nombre de lignes insérés)
$db->exec('INSERT INTO coucou (id) VALUES (20)');

// query() pour les requêtes d'extraction
$stmt          = $db->query('SELECT * FROM coucou');
var_dump($stmt→fetchAll());
```

+ On note la différence du retour entre exec(), et query() !

L'accès à une base de données MySQL

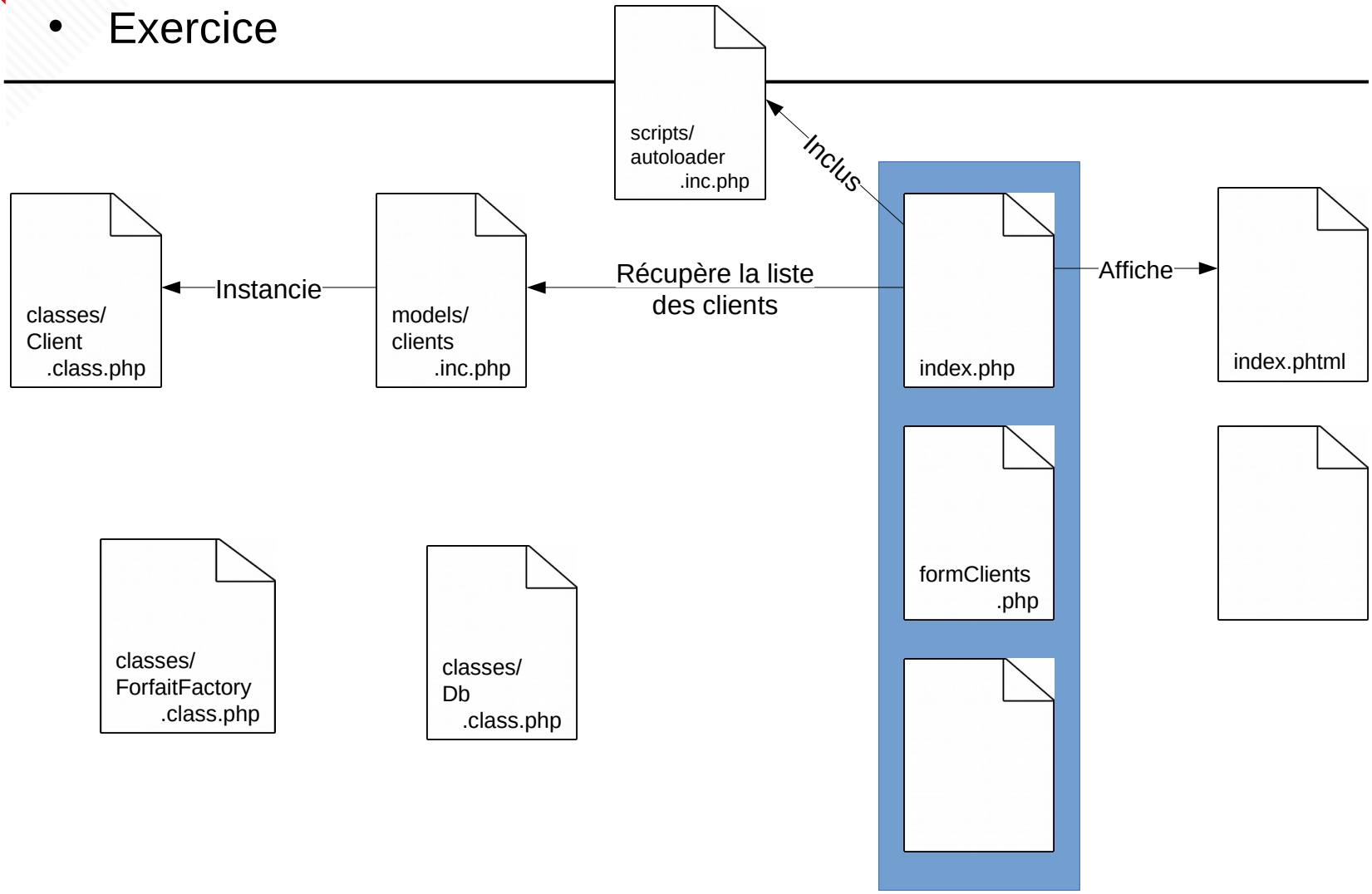
- Exercices

+ **Modifier** Exercices/chapitre7/mini-projet/
  - **Installer** les tables de données pouvant stocker les informations des clients & forfaits (***sqls/install.sql***).
  - **Constater** comment la classe *ForfaitFactory* utilise avantageusement les données provenants de la base.
    - Comprendre *Db::connect()*
  - **Constater** les modifications dans la méthodes *Client::loadAll()*
    - Comprendre *Db::connect()*
    - Ne pas hésiter à utiliser var_dump(), pour étudier les étapes.
    - Ajouter des données dans la base via *phpMyAdmin* si nécessaire.
  - **Transformer** le mini-projet afin d'utiliser la base de données pour ajouter des nouveaux clients.

+ Pour les plus rapide
  - **Ajouter** la possibilité de supprimer définitivement un client de la base.

Interactions utilisateur et formulaires

- Exercice



L'accès à une base de données MySQL

- SGBDR

+ Prenons deux tables
  - Utilisateurs
    - Prénom
    - Nom
    - CodePays
    - Téléphone
  - Pays
    - CodePays
    - Libellé

id	prenom	nom	codePays	telephone
1	Robert	Dupont	FR	0102030405
2	Jeanine	Dupond	FR	0102030407
3	Jack	O'Neil	US	001020305
4	Hiro	Nagaji	*NULL*	*NULL*

codePays	libelle
FR	France
US	United-States
RU	Russia

+ On peut sélectionner la totalité des informations de l'une ou l'autre des tables facilement.

```
SELECT * FROM utilisateurs;
SELECT * FROM pays;
```

+ On peut aussi joindre les résultats en une fois

```
SELECT * FROM utilisateurs LEFT JOIN pays USING (codePays);
```

L'accès à une base de données MySQL

- SGBDR

+ Contraintes associées
  - Avec ce genre de tables et jointures, il apparaît important de pouvoir mettre des **contraintes d'intégrité**
    - Un client peux avoir un « codePays », **seulement si** le pays existe dans la table Pays.

  - Lors de la création d'une table de données, nous pouvons choisir le « Moteur de données » :

    - MyISAM
      - Par défaut jusqu'à la version 5.5
      - Moteur le moins gourmand en mémoire disque

    - InnoDB
      - Par défaut depuis la version 5.6
      - Moteur permettant entre autre la gestion des **clés étrangères** nécessaires aux contraintes d'intégrité sur les jointures.

+ http://fr.openclassrooms.com/informatique/cours/les-moteurs-de-stockage-de-mysql-2

→ Exemple 7-5

L'accès à une base de données MySQL

- Transactions ACID

+ Le moteur InnoDB de MySQL, permet d'assurer l'acidité des transactions SQL.
+ Une transaction SQL peut être une unique requête ou un ensemble de requêtes indissociables les unes des autres.
+ Prenons l'exemple deux de comptes clients, chacun ayant des points de fidélité échangeables, si Alice souhaite donner 10 points à Bob, en SQL nous écrirons :

```
UPDATE users SET points=points+10 WHERE name='Bob'
UPDATE users SET points=points-10 WHERE name='Alice'
```

+ Il est évident que si Alice n'a pas assez de points, Bob ne doit pas avoir les 10 points supplémentaires, une solution serait de tester avant qu'Alice dispose des points, et que le plafond de points pour Bob ne soit pas atteint... Cela peut être lourd, sans prendre en compte qu'entre nos tests et les UPDATEs, d'autres requêtes peuvent influer sur ces points.

L'accès à une base de données MySQL

- Transactions ACID

+ C'est dans ce genre de cas que l'on parle de Transaction SQL ACID
  - A – Atomicité : Tout ou rien.
  - C – Cohérence : Placement et respect des contraintes.
  - I – Isolation : Deux transactions ne se voient pas.
  - D – Durabilité : Une fois validées, les modifications sont immuables.
+ Pour rendre ces deux requêtes SQL ACID, nous allons ajouter la notion de COMMIT

```
BEGIN;
  UPDATE users SET points=points+10 WHERE name='Bob' AND points<990;
  UPDATE users SET points=points-10 WHERE name='Alice' AND points>10;
COMMIT;
```

+ Avec ces simples instructions, nous ajoutons contraintes de mini/maxi, ainsi que l'assurance que si l'une des deux requêtes n'est pas satisfaite, l'autre non plus.
+ Avec PDO, nous allons utiliser les instructions suivantes :

```
$db = new PDO('mysql:host=;dbname=', $user, $pass);
$db->beginTransaction();
…
$db->commit();
```

Chapitres

Introduction au langage PHP

Notre premier projet

Les bases de la programmation PHP

Programmation Orientée Objet

Interactions utilisateur et formulaires

Authentification et super globales

L'accès à une base de données MySQL

Validation des acquis

Validation des acquis

- Comme tout projet, à la fin de celui-ci, nous devrions toujours passer par une vérification complète :
- Dans le cas d'une mise à jour d'une application existante, nous appellerions cela une vérification de non-régression.
- Il est recommandé que ce soit une personne extérieure au développement de l'application qui effectue les tests finaux.
- Encore mieux, les utilisateurs finaux lors d'une version Bêta.

Validation des acquis

- Nous allons donc dans ce chapitre, reprendre notre projet, vérifier sa conception, s'assurer que nous n'aurions pas pu améliorer des choses, ou les faire autrement, avec les nouvelles connaissances.
- Chercher à trouver des cas d'erreur non gérés, il y en a toujours, plus ou moins simples à exploiter :
  - Paramètres
  - Retour non géré
  - Conditions mal formulées (ex: $a < 15 ; $a > 15, au lieu de >=, pour gérer tous les cas)
- Chercher de nouvelles fonctionnalités sur Internet, en gardant à l'esprit que la solution trouvée pour faire quelque chose n'est pas forcément la meilleure façon de le faire ; Il faudra chercher d'autres façons, et peser les pour et les contre.

Validation des acquis

- Exercices

+ En partant des acquis et du mini-projet, rechercher de nouvelles fonctionnalités manquantes, ou peu adaptées.
+ Une liste de piste à suivre :
  - Compléter les forfaits dans la BdD,
  - Réfléchir à la possibilité d'associer une ligne téléphonique à deux clients,
  - Gestion des erreurs connues, et inconnues via les Exceptions,
  - Utiliser les « belles URLs » avec le serveur Web,
  - Afficher pour un client toutes ses lignes de téléphone après identification sur son compte ; et un formulaire pour effectuer des modifications.
  - Envisager avec le groupe d'autres améliorations possibles, et tenter de trouver les meilleures solutions pour les mettre en place.

Validation des acquis

- Formes et couleurs

+ PHP est un langage de programmation, cependant le rendu final en PHP pur, est vraiment brute, nous avons dans certains cas utilisé HTML, pour emphaser des points, ou mis en place des listes à puce, mais la forme et les couleurs, c'est le travail de CSS
+ CSS – Cascading Style Sheet !
+ Il est peut-être temps d'essayer un peu les possibilités de cette couche pour rendre notre travail plus plaisant !

Annexe

Programmation Orientée Objet

- Exercice complémentaires

+ *Supposons que vous ayez un site Internet fonctionnel en procédurale, et que vous souhaitiez y inclure de la POO progressivement ou non.*

+ *Lors d'une maintenance évolutive, ont vous demande de mieux gérer les réseaux sociaux, dans notre cas « Facebook », pour fonctionner il faut indiquer à celui-ci l'image à prendre en compte lors du partage d'une page de votre site.*

+ *La documentation officielle nous dis d'ajouter dans le <head> de la page :*

```
<meta property="og:image"          content="/images/img.jpeg" />
<meta property="og:image:type"     content="image/jpeg" />
<meta property="og:image:width"    content="1024" />
<meta property="og:image:height"content="780" />
```

+ *Sur la base de « Exercice/chapitre4/FbImage/ », modifier le code procédural en code orienté objet*