

## PHP et la Programmation Orientée Objet

Avec Php, les fonctions issues des bibliothèques sont accessibles « globalement » (sans préfixe pointé du nom du module). Il est impossible alors d'identifier le module auquel appartient une fonction. Pour s'y retrouver, les développeurs de PHP respectent certaines conventions d'écriture. Par exemple les fonctions MySql commencent toutes par le mot mysql\_

Exemple : mysql\_query()

Mais ce n'est pas suffisant, car les fonctions sont toutes liées à des structures de données bien précises.

Le modèle Objet est ce que l'on a trouvé de mieux pour réunir entre eux Structures de Données et Traitements. Une classe représente le comportement type d'objets de même nature. Par exemple la classe Etudiant décrit la structure type des objets de type Etudiant et leurs comportements (les fonction associées à ce type).

Php 5, bien que n'étant pas à proprement parlé un langage purement objet, intègre le concept de Classe, dispose d'un opérateur (**new**) de construction d'objets (instances) et bien autres choses.

Remarque : Dans les langages objet, une classe peut aussi représenter un module fonctionnel, c'est à dire une structure regroupant des fonctions utilitaires. C'est le cas par exemple de la classe Collections en java. Exemple : Collections.sort(...) permet de trier une collection donnée en argument.

Les règles d'écriture sont consultables ici (PHP Coding Standards) : [http://cvs.php.net/viewvc.cgi/php-src/CODING\\_STANDARDS?view=co](http://cvs.php.net/viewvc.cgi/php-src/CODING_STANDARDS?view=co)

Extrait :

nommage des méthodes :

Method names follow the 'studlyCaps' (also referred to as 'bumpy case' or 'camel caps') naming convention, with care taken to minimize the letter count. **The initial letter of the name is lowercase, and each letter that starts a new 'word' is capitalized.**

Quelques références : <http://fr2.php.net/manual/fr/language.oop5.php>

Exception : <http://fr2.php.net/manual/fr/language.exceptions.php>

Static : <http://fr2.php.net/manual/fr/language.oop5.static.php>

Variable non objet et référence : <http://fr2.php.net/manual/fr/language.references.php>

Ci-dessous un tableau comparatif de certains concepts clés de l'objet présents dans PHP et leur équivalent en Java. Un tableau à avoir sous la main quand on commence à coder en PHP Objet.

Concept Objet	Exemple	Php5	Java
Interface	<pre> &lt;&lt;interface&gt;&gt; XMLable +toXml(): String </pre>	<pre> interface XMLable {     public abstract function toXML( ); } </pre>	<pre> public interface XMLable {     public abstract String toXml(); } </pre>
Classe	<pre> Personne -nom -prenom -login </pre>	<pre> class Personne {     private \$nom;     private \$prenom;     private \$login; } </pre>	<pre> public class Personne {     private String nom;     private String prenom;     private String login; } </pre>
Constructeur		<pre> function __construct(\$nom,     \$prenom, \$login) {     \$this-&gt;nom = \$nom;     \$this-&gt;prenom = \$prenom;     \$this-&gt;login = \$login; } </pre>	<pre> public Personne(String nom,     String prenom, String login) {     this.nom = nom;     this.prenom = prenom;     this.login = login; } </pre>
Classe d'implémentation	Personne <u>réalise</u> le contrat défini par XMLable	<pre> class Personne implements XMLable {     ...     public function toXML() {         ...     } } </pre>	<pre> public class Personne implements XMLable{     ...     public String toXml(){         ...     } } </pre>
Héritage	PersonneSecure <u>est</u> une Personne disposant d'un mot de passe	<pre> class PersonneSecure extends Personne {     private \$md5pw;     ... } </pre>	<pre> public class PersonneSecure extends Personne {     private String md5pw;     ... } </pre>
Redéfinition du	Le constructeur de	<pre> class PersonneSecure extends </pre>	<pre> public class PersonneSecure </pre>

Concept Objet	Exemple	Php5	Java
constructeur	PersonneSecure personnalise la construction d'une Personne	<pre> Personne {     private \$md5pw;     function __construct(\$nom,         \$prenom, \$login, \$md5pw) {          parent::__construct(\$nom,                                 \$prenom,                                 \$login);          \$this-&gt;md5pw=\$md5pw;     } } </pre>	<pre> <b>extends</b> Personne {     <b>private</b> String md5pw;      <b>public</b> Personne(String nom,         String prenom, String login,         String md5pw) {         <b>super</b>(nom, prenom, login);         this.md5pw=md5pw;     } } </pre>
Redéfinition d'une méthode d'instance	La méthode toString est redéfinie dans PersonneSecure (suppose que la classe Personne ait correctement redéfini la méthode toString)	<pre> public function __toString() {     <b>return</b> parent::__toString() .         " \$this-&gt;md5pw"; } </pre>	<pre> <b>public String</b> toString(){     <b>return</b> <b>super</b>.toString() +         " " + <b>this</b>.md5pw; } </pre>
Création d'objets (instance)	Création d'une instance de Personne et affichage de son état.	<pre> \$p = <b>new</b> Personne('Arsac', 'Jacques',     'jarsac');  <b>echo</b> \$p; //ou <b>echo</b> \$p-&gt;__toString(); </pre>	<pre> Personne p =     <b>new</b> Personne("Arsac", "Jacques", "jarsac"); System.out.println(p.toString()); // ou System.out.println(p); </pre>
Accès aux attributs public sans les connaître		<pre> // \$p est une var. objet foreach (\$p as \$att =&gt; \$value) {     print "\$att =&gt; \$value\n"; } // voir aussi : get_class_methods // get_class_vars </pre>	utiliser l'introspection en utilisant l'API de reflection.

Concept Objet	Exemple	Php5	Java
Exception	Gérer l'exception potentielle	<pre>try {     ... }catch (Exception \$e) {     ... }</pre>	<pre>try {     ... } catch (Exception e) {     ... }</pre>
	Déclencher une exception	<code>throw new MyException("PB");</code>	<code>throw new MyException("PB");</code>
Structures «static»	Exemple d'école : un compteur d'instances	<pre>class Personne {     public static \$nbInstances = 0;     ... }</pre>	<pre>class Personne {     public static int nbInstances=0;     ... }</pre>
Définition d'une méthode de classe	Obtenir la valeur du compteur	<pre>class Personne {     private static \$nbInstances = 0;     public static function getNbInstances()     { return self::\$nbInstances; } ...}</pre>	
	Accès à un membre static	<pre>Personne::\$nbInstances++; si public ou self::\$nbInstances++; en interne Personne::getNbInstances()</pre>	<code>Personne.nbInstances++;</code>
La logique objet est implicitement par référence. La logique implicite des variables non objet est par valeur.	Un même objet peut être référencé par plusieurs variables. Exemple : \$p1 et \$p2 référencent le même objet.	<pre>\$p1 = new Personne(     'Arsac', 'Jacques', 'jarsac'); \$p2 = \$p1; echo \$p1 . "&lt;br&gt;"; echo \$p2 . "&lt;br&gt;"; <b>\$p1-&gt;setNom( 'ARSAC' );</b> echo <b>\$p2</b> . "&lt;br&gt;"; ce qui donne : Nom : Arsac, Prénom : Jacques Nom : Arsac, Prénom : Jacques Nom : <b>ARSAC</b>, Prénom : Jacques</pre>	idem
Constante	par convention les identificateurs sont en	<code>const MATCH = 1;</code> lettres MAJUSCULES.	<code>final int MATCH = 1;</code>