

# Introduction à PHP

## Programmation objet

---

`monnerat@u-pec.fr`

2 mai 2026

IUT de Fontainebleau

Classes

Héritage

Interface

Exceptions

Objet avancé

Accès bases de données avec PDO

# Classes

# Classes

Attributs et méthodes

# Classe

Une classe encapsule des variables (appelées **attributs**) et des fonctions (appelées **méthodes**) qui fonctionnent avec ces variables. On les définit comme suit :

```
class SendMail {  
    public $destinataire;  
    public $objet;  
    public $texte;  
    public function envoyer() {  
        mail ($this->destinataire, $this->objet, $this->texte);  
    }  
}  
  
$message = new SendMail ();  
  
$message->destinataire = "monnerat@u-pec.fr";  
$message->objet = "un mail";  
$message->texte = "a moi !";  
  
$message->envoyer();
```

L'accès à un attribut, ou une méthode se fait par la syntaxe fléchée ->.

Dans l'exemple précédent, les attributs et méthodes sont publiques (accessibles en dehors des méthodes de la classe). On peut les choisir :

<code>private</code>	utilisable uniquement à l'intérieur de la classe
<code>protected</code>	utilisable uniquement dans les classes dérivées

Pour accéder à des attributs protégés en dehors de la classe, on utilise des setters et getters :

```
public function getDestinataire()  
{  
    return $this->destinataire;  
}  
public function setDestinataire($qui)  
{  
    $this->destinataire = $qui;  
}
```

# Classes

## Constructeur

# Instanciation

L'instanciation d'une classe se fait avec le mot-clé `new`

```
$monMail = new SendMail();
```

On peut ajouter à la classe un (seul) **constructeur**, avec la méthode `__construct`.

```
function __construct($qui,$objet,$texte="???"){  
    $this->destinataire = $qui;  
    $this->objet = $objet;  
    $this->texte = $texte;  
}
```

automatiquement appelé à la création

```
$monMail = new SendMail(  
    "monnerat@u-pec.fr",  
    "FA",  
    "Je suis candidat");
```



On peut déclarer un attribut ou une méthode `static`. On les utilise comme des variables ou fonctions classiques, indépendantes d'une instance quelconque (il n'y a plus de `$this`) .

```
Class SendMail{  
    static $smtp = "orion.u-pec.fr";  
}
```

On y accède avec le nom de la classe

```
echo SendMail :: $smtp;
```

# Classes

Référence et copie

# Référence

Depuis PHP5, le passage ou l'affectation d'un objet se fait par **référence**.  
(le & est présent implicitement)

```
class obj
{
    public $val;
    function __construct($x){
        $this->val = $x;
    }
    function __toString(){
        return "val = ".$this->a;
    }
}

$a = new obj(1);
$b = new obj(2);
$a=$b;
$b->val=3;
echo $a; // affiche val = 3
```

On peut cloner explicitement l'objet

```
$a = clone $b;
```

# Mots réservés

Ces mots permettent de déclarer des classes en PHP :

<code>class</code>	Déclaration de classe
<code>const</code>	Déclaration de constante de classe
<code>function</code>	Déclaration d'une méthode
<code>public/protected/private</code>	Accès
<code>self</code>	la classe elle-même
<code>parent</code>	la classe "parent"
<code>static</code>	méthode de classe (statique)
<code>extends</code>	Héritage de classe
<code>implements</code>	Implémentation d'une interface

Les mots clefs "self" et "parent" sont utiles pour accéder à une propriété ou méthode (statique ou non) de la classe elle-même ou de son parent.

# Méthodes magiques

<code>__construct()</code>	Constructeur de la classe
<code>__destruct()</code>	Destructeur de la classe
<code>__set()</code>	Déclenchée lors de l'accès en écriture à une propriété inexistante
<code>__get()</code>	Déclenchée lors de l'accès en lecture à une propriété inexistante
<code>__call()</code>	Déclenchée lors de l'appel d'une méthode inexistante de la classe
<code>__callstatic()</code>	Déclenchée lors de l'appel d'une méthode inexistante de la classe
<code>__isset()</code>	Déclenchée si on applique <code>isset()</code> à une propriété inexistante
<code>__unset()</code>	Déclenchée si on applique <code>unset()</code> à une propriété inexistante
<code>__sleep()</code>	Exécutée si la fonction <code>serialize()</code> est appliquée à l'objet
<code>__wakeup()</code>	Exécutée si la fonction <code>unserialize()</code> est appliquée à l'objet
<code>__toString()</code>	Appelée lorsque l'on essaie d'afficher directement l'objet : <code>echo \$object</code>
<code>__set_state()</code>	Méthode statique lancée lorsque l'on applique la fonction <code>var_export()</code> à l'objet
<code>__clone()</code>	Appelé lorsque l'on essaie de cloner l'objet
<code>__autoload()</code>	Cette fonction n'est pas une méthode, elle est déclarée dans le scope global et permet d'automatiser les "include/require" de classes PHP

# Fonctions et constantes utiles

## Fonctions :

<code>class_parents()</code>	Retourne un tableau de la classe parent et de tous ses parents
<code>class_implements()</code>	Retourne un tableau de toutes les interfaces implémentées par la classe et par tous ses parents ;
<code>get_class()</code>	Retourne la classe de l'objet passé en paramètre
<code>get_called_class()</code>	À utiliser dans une classe, retourne la classe appelée explicitement dans le code PHP et non au sein de la classe
<code>class_exists()</code>	Vérifie qu'une classe a été définie
<code>get_class()</code>	Retourne la classe d'un objet
<code>get_declared_classes()</code>	Liste des classes définies
<code>get_class_methods()</code>	Liste des méthodes d'une classe
<code>get_class_vars()</code>	Liste des propriétés d'une classe

Constantes :

<code>__CLASS__</code>	Donne le nom de la classe en cours
<code>__METHOD__</code>	Donne le nom de la méthode en cours

## Example

```
$classes=get_declared_classes();  
echo "<ul>";  
foreach($classes as $classe)  
{  
    echo "<li>$classe : <ul>";  
    $liste=get_class_methods($classe);  
    foreach($liste as $methode)  
    {  
        echo "<li>$methode</li>";  
    }  
    echo "</ul></li>";  
}  
echo "</ul>";
```



# Héritage

**Héritage**

Héritage

## Héritage simple uniquement

```
class point_couleur extends point { // déclaration de la sous classe
    protected $couleur="black";
    function __construct($x,$y,$couleur) { // constructeur
        parent::__construct($x,$y);      // constructeur de la classe parente
        $this->couleur=$couleur;
    }
    function affiche()
    {
        parent::affiche(); // appel de la méthode affiche de point
        echo "<p>$this->couleur</p>";
    }

    function setCouleur($couleur) {
        $this->couleur = $couleur;
    }
    function getCouleur() {
        return $this->couleur;
    }
}
```

point\_couleur est une classe dite fille, ou dérivée de point.

- On peut redéfinir une méthode dans une classe fille, à condition que le prototype de la méthode soit compatible avec celui de la méthode dans la classe parente.
- Pour interdire la redéfinition, il faut utiliser le mot-clé `final`. La redéfinition dans une classe fille est alors impossible.

# Héritage

Classe abstraite

# Classe abstraite

But : Définir une classe, non instanciable, en obligeant les classes filles à implanter certaines méthodes ou attributs

```
abstract class animal {  
    abstract function manger();  
    function dormir(){  
        echo "Zzzzz";  
    }  
}
```

Toute classe dérivée, pour être instanciable, devra obligatoirement implanter la méthode manger.

```
class chien extends animal{  
    function manger(){  
        echo "miam miam";  
    }  
}
```

**Héritage**

**Classe Finale**

Comme pour les méthodes, on peut indiquer qu'une classe est finale, c'est à dire non dérivable.



# Interface

**Interface**

**Définition**

# Interface

Notion proche de classe abstraite.

- Une interface est la déclaration d'une API ([Application Programming Interface](#)).
- Une classe implante (propose) une interface si elle définit ses fonctions.
- C'est un moyen d'être certain qu'un objet particulier possède des méthodes particulières.

```
interface estCarnivore {  
    public function mangerViande();  
}  
  
class chien implements estCarnivore {  
    function mangerVidande(){  
        echo "miam miam";  
    }  
}
```

## Exemple : interface Iterator

```
Iterator extends Traversable {  
  /* Méthodes */  
  abstract public mixed current ( void )  
  abstract public scalar key ( void )  
  abstract public void next ( void )  
  abstract public void rewind ( void )  
  abstract public boolean valid ( void )  
}
```

```

class MyString implements Iterator{
    private $s=""; // la chaîne
    private $l=0; // la longueur
    private $index=0; // index pour l'itération
    public function __construct($s) {
        $this->s = $s;
        $this->l = strlen($this->s);
    }
    function current() {
        return $this->s[$this->index];
    }
    function key() {
        return $this->index;
    }
    function next() {
        ++$this->index;
    }
    function rewind() {
        $this->index = 0;
    }
    function valid() {
        return $this->index < $this->l;
    }
    public function __toString(){
        return $this->s;
    }
}

```

```
$s = new MyString("toto");  
foreach ($s as $k => $c)  
    echo "$k = > $c";
```

# Exceptions

# Exceptions

PHP permet l'utilisation d'exceptions (**try/catch**). L'objet (l'exception) lancé (**throw**) doit être une instance de la classe **Exception**.

```
function inverse($x) {  
    if (!$x) {  
        throw new Exception('Division par zéro.');    }  
    return 1/$x;  
}  
  
try {  
    echo inverse(5) . "\n";  
    echo inverse(0) . "\n";  
} catch (Exception $e) {  
    echo 'Exception reçue : ', $e->getMessage(), "\n";  
}
```



**Objet avancé**

# Autochargement

Dans les applications objets, les scripts utilisant des classes doivent inclure les fichiers de définition : fastidieux

Solution : fonction `__autoload()` qui sera automatiquement appelée si vous essayez d'utiliser une classe ou interface qui n'est pas encore définie. Grâce à elle, vous avez une dernière chance pour inclure une définition de classe, avant que PHP n'échoue avec une erreur.

```
<?php
function __autoload($class_name) {
    include $class_name . '.php';
}

$objj  = new MaClasse1();
$objj2 = new MaClasse2();
?>
```

La fonction précédente est obsolète à partir de PHP 7.2

On peut enregistrer une (des) fonction d'autoload avec `spl_autoload_register`.

# Utilisation de chaines comme identifiant

⇒ De classe :

```
<?php
$class="SendMail";
if(class_exists($class)) {
    $sd=new $class("smtp.free.fr");
}
?>
```

⇒ De fonction

```
function barber($type){
    echo "Vous voulez une coupe $type, ok";
}
if (is_callable('barber')){
    call_user_func('barber', "au bol");
    call_user_func('barber', "au rasoir");
}
```

⇒ De méthode

```
$monobjet = new maclasse();  
$methode = array($monobjet, "dit_bonjour");  
if (is_callable($methode)){  
    call_user_func($methode);  
}
```

Une classe Database, pour organiser l'accès au sgbd, qui respecte le design pattern singleton :

```
class Database {
    static protected $_instance = null;
    protected $_db;
    static public function getInstance() {
        if( is_null(self::$_instance) )
            self::$_instance = new Database();
        return self::$_instance;
    }
    public function query($sql){
        return $this->_db->query($sql);
    }
    protected function __construct() {
        $this->_db = new PDO(
            "mysql:host=localhost;dbname=mvc;charset=utf8",
            "user",
            "password"
        );
    }
}
```

**Accès bases de données avec PDO**

- Interface commune d'accès à différents SGBD.
- Extension PHP qui est incluse dans la distribution standard depuis PHP 5.1
- Abstraction qui nécessite des drivers particuliers selon le sgbd.
- Possède trois classes :

PDO	la classe d'interaction principale avec le SGBD
PDOStatement	la classe gérant les résultats de requêtes, préparées, ou exécutées
PDOException	classe d'exception



```
<?php
$pdo = new PDO('mysql:host=localhost;dbname=tp3',
    'mylogin',
    'mypassword'
);
// iteration classique
$res = $pdo->query("SELECT nom FROM professeur");
while ($result = $res->fetch()) {
    echo $result['nom'];
}
// iteration avec l'interface traversable
foreach ($res as $result) {
    echo $result['nom'];
}

?>
```

La méthode fetch de PDOStatement est contrôlée par le fetch\_style, que l'on peut positionner en argument de cette méthode, ou avec la méthode `setFetchMode`

```
<?php
$pdo = new PDO('mysql:host=localhost;dbname=tp3',
    'mylogin', 'mypassword');
$res = $pdo->query("SELECT nom FROM professeur");
$res->setFetchMode(PDO::FETCH_OBJ);

foreach ($res as $result) {
    echo $result->nom;
}
?>
```

Il existe d'autres façons de récupérer un jeu de résultat.

En particulier, PDO::FETCH\_CLASS : retourne une nouvelle instance de la classe demandée, liant les colonnes du jeu de résultats aux noms des propriétés de la classe.

```
int PDO::exec ( string $statement )
```

exécute une requête SQL dans un appel d'une seule fonction, retourne le nombre de lignes affectées par la requête.



PDO::exec() ne retourne pas de résultat pour une requête SELECT.

- Pour une requête SELECT dont vous auriez besoin une seule fois dans le programme, utilisez plutôt la fonction PDO::query().
- Pour une requête dont vous auriez besoin plusieurs fois, préparez un objet PDOStatement avec la fonction PDO::prepare() et exécutez la requête avec la fonction PDOStatement::execute().

# Requêtes préparées

Emulées par PDO si le driver ne les supporte pas. Exemple en insert :

```
$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
$name = 'two';
$value = 2;
$stmt->execute();
```

## Remarques

- On peut utiliser ? comme marqueur de paramètre dans la requête :

```
$stmt = $dbh->prepare("INSERT INTO REGISTRY (name, value) VALUES (?,?)");
$stmt->bindParam(1, $name);
$stmt->bindParam(2, $value);

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

- La méthode PDOStatement::bindValue permet d'associer une valeur à un paramètre.
- On peut passer les paramètres de la requête dans la méthode PDOStatement::execute :

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

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

## Exemple en select

```
<?php
$stmt = $dbh->prepare("SELECT * FROM REGISTRY where name = ?")
if ($stmt->execute(array($_GET['name']))) {
    while ($row = $stmt->fetch()) {
        print_r($row);
    }
}
?>
```

- Atomicité, Consistance, Isolation et Durabilité (ACID).
- Toutes les bases de données ne supportent pas les transactions, donc, PDO doit s'exécuter en mode "autocommit" lorsque vous ouvrez pour la première fois la connexion.
- PDO::beginTransaction() pour l'initialiser.
- PDO::commit() ou la fonction PDO::rollBack() pour la terminer.



```

/* Commence une transaction, désactivation de l'auto-commit */
$dbh->beginTransaction();

/* Insérer plusieurs enregistrements sur une base tout-ou-rien */
$sql = 'INSERT INTO fruit
      (name, colour, calories)
      VALUES (?, ?, ?)';

$sth = $dbh->prepare($sql);

foreach ($fruits as $fruit) {
    $sth->execute(array(
        $fruit->name,
        $fruit->colour,
        $fruit->calories
    ));
}

/* Valider les modifications */
$dbh->commit();

/* La connexion à la base de données est maintenant
   * de retour en mode auto-commit */

```

```
<?php
/* Démarre une transaction, désactivation de l'auto-commit */
$dbh->beginTransaction();

/* Modification du schéma de la base ainsi que des données */
$sth = $dbh->exec("DROP TABLE fruit");
$sth = $dbh->exec("UPDATE dessert
    SET name = 'hamburger'");

/* On s'aperçoit d'une erreur et on annule les modifications */
$dbh->rollBack();

/* Le connexion à la base de données est maintenant de
    * retour en mode auto-commit */
?>
```