Programmation Orienté Objet

avec PHP

Arnaud BODEL
https://www.linkedin.com/in/arnaudbodel/

Introduction

- Qu’est ce que c’est ?
- Un peu d’histoire
- Avantage de la POO

Qu’est ce que c’est ?

Qu’est ce que c’est ?

Un Objet est un conteneur autonome et symbolique qui contient des **informations** et des **mécanismes** concernant un "univers".

En POO, un objet est créer à partir d'un modèle que l'on nomme **Classe** ou **Prototype**.

Les Objets en POO sont inspirés des objets réel de la vie courante.

Un peu d’histoire

Un peu d’histoire

Simula

La notion d'objet est apparue avec le langage de programmation Simula, développé à Oslo entre 1962 et 1967, dans le but de faciliter la programmation de logiciel de simulation. Avec le langage Simula, les caractéristiques et les comportements des objets à simuler sont décrits dans le code source.

SmallTalk

En 1972, Xerox développe le langage de programmation orienté objet SmallTalk.

C++

En 1983, la sortie du langage C++ popularise la programmation oriente objet dont l'utilisation ressemble volontairement au langage C.

Les bases de données

En 1993, les premières base de données orienté objet sont commercialisées. Leurs technologie n'arrivera à maturité qu'à la fin du XXe siècle.

En 1995, SunMicroSystems créer le langage de programmation orienté objet JAVA. Celui-ci été avant tout destiné au développement d'application web en pleine évolution dans les années 90.

Avantages de la P.O.O.

Avantages de la P.O.O.

Un code organisé

La POO nous permet de mieux organiser notre code source, on sait à quel "univers" se réfère une fonction ou une information.

```
1  $univers_1->fait_quelque_chose();
2  $univers_2->fait_quelque_chose();
3  $univers_1->fait_autre_chose();
```

Avantages de la P.O.O.

Un code structuré

Une application écrite en objet sera plus facilement modulable qu'une application procédurale.

En séparent les "sujets" le code devient plus facile à lire, plus facile à maintenir.

Un code plus "safe"

La POO permet de protéger l'accès à certaine données ou fonctions, cela peut éviter des erreurs de développement.

En bref

En bref

La Programmation Orienté Objet permet :

- un code plus réutilisable
- un code modulaire
- un code plus compréhensible

POO vs. Procédurale

En procédurale votre code est fonctionnel, mais l'application deviendra de plus en plus difficile à maintenir au fur et à mesure que le code de celle-ci croit.

La POO vous imposera une rigueure et une organisation de travail.

Classes et Objets

- Notion de classe
- Notion d'objet
- Les propriétés
- Les méthodes
- Les constantes
- La pseudo-variable `$this`
- Constructeur et Destructeur
- Cas pratique

Notion de classe

Notion de classe

Qu'est ce qu'une classe ?

Une classe contient le code source d'un "univers".

Cet "univers" peut être considéré comme un type de donnée supplémentaire.

On peut imaginer une classe comme un prototype ou un modèle.

Une classe contient :

- les variables d'un "univers", ce sont les **Propriétés**.
- les actions d'un "univers", ce sont les **Méthodes**.
- les constantes d'un "univers"

Notion de classe

Déclarer une classe

Une classe se déclare avec le mot clé "**class**" suivi de son nom et d'une paire d'accolades.

```
class MaClass {}
```

Notion de classe

La recommandation PSR-4

La recommandation PSR-4 permet le chargement automatique des classes dans une application.

- Il faut définir un nom de classe complet, le **Namespace**.
- Le nom de la classe doit correspondre au nom du fichier.
- Le nom du fichier doit se terminer par .php.
- Le fichier ne doit contenir qu'une seule classe.

ex : `MaClass.php` ⇒ `class MaClass {...}`

https://www.php-fig.org/psr/psr-4/

Notion de classe

Pratique

→ Déclarer une classe "**personnage**".

Notion de classe

Pratique

→ Déclarer une classe "**personnage**".

Personnage.php

```php
<?php
class Personnage {}
```

Notion d’objet

Notion d’objet

Qu'est ce qu'un objet ?

Un objet est l'instance d'une classe.

On utilise une classe — un modèle — pour créer un objet.

Notion d’objet

Créer une instance

On créer une instance avec le mot clé "**new**" suivi du nom de la classe à instancier.

Pour utiliser notre objet, on affecte l'instance de la classe à une variable qui nous servira de référence.

```php
$monObjet = new MaClasse();
```

Notion d’objet

Pratique

→ Instancier la classe **Personnage** dans le fichier **index.php**

index.php

```php
<?php
include_once "Personnage.php";
$personnage = new Personnage();
```

Les propriétés

Les propriétés

Qu'est ce que c'est ?

Les propriétés (ou attributs) sont les données d'un objet.

Les propriétés reste local à un objet.

Les propriétés

Déclarer une propriété

Une propriété se déclare comme une variable dans une classe.

```php
class MaClasse {
    public $propriete_1;
    public $propriete_2 = "Valeur de la propriété 2.";
}
```

Une propriété peut être déclarée avec ou sans valeur.

Les propriétés

Accéder une propriété

Pour accéder à une propriété, l'objet auquel elle se réfère doit être instancié.

On accède à la propriété d'un objet avec l'opérateur de classe `->`.

```php
$maClasse = new MaClasse();
$maClasse->propriete_2;
```

Les propriétés

Pratique

1 - Ajouter les propriétés suivante à la classe Personnage.

- Nom : Francis
- Age: 53

2 - Afficher les propriétés sur le fichier d'index.

Les propriétés

Pratique

1 - Ajouter les propriétés suivante à la classe Personnage.

Personnage.php

```php
<?php
class Personnage {
    public $nom = "Francis";
    public $age = 53;
}
```

Les propriétés

Pratique

2 - Afficher les propriétés sur le fichier d'index.

Index.php

```php
<?php
$personnage = new Personnage();
echo "<div>Nom : ". $personnage->nom ."</div>";
echo "<div>Age : ". $personnage->age ."</div>";
```

Les méthodes

Les méthodes

Qu'est ce que c'est ?

Une méthode est un service, une action sur un objet.

Déclarer une méthode

Une méthode se déclare comme une fonction dans une classe.

Les méthodes

Déclarer une méthode

```php
class MaClass {
    public function methode_1() {}
    public function methode_2(string $argument){
        $variable = strtoupper($argument);
        return $variable;
    }
}
```

Les méthodes

Accéder à une méthode

Pour accéder à une méthode, l'objet auquel elle se réfère doit être instancié.

On accède à la méthode d'un objet avec l'opérateur de classe **->**.

```php
$maClasse = new MaClasse();
$maClasse->methode_1();
$maClasse->methode_2("Lorem ipsum"); // LOREM IPSUM
```

Les méthodes

Pratique

Déclarer une méthode permettant au personnage de dire – avec un **return** – "Bonjour Claire" et "Bonjour Doug".

Déclarer une méthode permettant au personnage de dire – avec un **return** – simplement "Au revoir".

Afficher les "Bonjour" et "Au revoir" sur le fichier d'index.

Les méthodes

Personnage.php

```php
<?php
class Personnage {
    // ...
    public function ditBonjour(string $prenom){
        return "Bonjour $prenom.";
    }
    public function ditAurevoir(){
        return "Au revoir.";
    }
}
```

Les méthodes

index.php

```php
<?php
$personnage = new Personnage();
// ...
echo "<div>". $personnage->ditBonjour("Claire") ."</div>";
echo "<div>". $personnage->ditBonjour("Doug") ."</div>";
echo "<div>". $personnage->ditAurevoir() ."</div>";
```

Les constantes

Les constantes

Qu'est ce que c'est ?

Une constante est une propriété invariable. Elle gardera la valeur définis lors de la création de la classe.

Une constante possède une portée dite "statique".

Les constantes

Déclarer une constante

On déclare une constante avec le mot clé **const**.

```
1  class MaClasse {
2      const MA_CONSTANTE = "Valeur de la constante";
3  }
```

Les constantes

Accéder à une constante

Contrairement aux propriétés et méthode, il n'est pas nécessaire d'instancier un objet pour accéder aux constantes de celui-ci.

L'accès aux constantes se fait avec l'opérateur de résolution de portée **::** appelé **Paamayim Nekudotayim**.

Les constantes

```php
class MaClass {
    const MA_CONSTANTE = "FooBar";

    public function methode(){
        // Accès à la constante à l'intérieur de la classe.
        return self::MA_CONSTANTE;
    }
}

// Accès à la constante à l'extérieur de la classe.
MaClasse::MA_CONSTANTE;
```

Les constantes

Pratique

Créer une classe Voiture et lui définir une constante pour le nombre de roues.

Voiture.php

```php
<?php
class Voiture {
    const WHEELS = 4;
}
```

La pseudo-variable

`$this`

La pseudo-variable `$this`

Qu'est ce que c'est ?

La pseudo-variable **$this** est disponible lorsqu'une méthode est appelée depuis un contexte objet — c'est à dire quand l'objet est instancié.

$this est une référence à l'objet appelant (habituellement, l'objet auquel la méthode appartient), un peu comme le mot ***moi*** qui est utilisé quand on parle de soi.

Il est utile lorsque l’objet se cible lui-même.

La pseudo-variable `$this`

Utiliser la pseudo-variable $this.

Pour faire référence à l'objet instancié, `$this` doit être utilisé dans la classe.

```php
class MaClass {
    $propriete = "Foo";
    function methode_1() {
        $this->propriete = "Bar";
    }
    function methode_2() {
        echo $this->methode_1();
    }
}
```

La pseudo-variable `$this`

Pratique

1 - Créer la méthode `afficheNom()` qui permet de retourner la valeur de la propriété `$nom`.

2 - Afficher le nom sur le fichier index en passant par la méthode `afficheNom()`.

La pseudo-variable `$this`

Personnage.php

```php
<?php
class Personnage {
    $nom = "Francis";
    function afficheNom() {
        return $this->nom;
    }
}
```

La pseudo-variable `$this`

index.php

```php
<?php
$personnage = new Personnage();
// echo "<div>Nom : " + $personnage->nom + "</div>";
echo "<div>Nom : " + $personnage->afficheNom() + "</div>";
```

Constructeur et destructeur

Constructeur et destructeur

Le constructeur de classe

Le constructeur est une méthode de la classe qui est appelée automatiquement à la création d’un objet (instance de la classe).

Cette méthode est utilisée pour initialiser les paramètres d’un objet au moment de sa création.

Constructeur et destructeur

MaClass.php

```php
class MaClass {
    $propriete;
    function __construct(string $input) {
        $this->propriete = $input;
        echo "Un objet vient d'être créé.";
    }
}
```

Constructeur et destructeur

index.php

```php
$monObjet = new MaClass("FooBar");
echo $monObjet->propriete; // FooBarr
```

Constructeur et destructeur

Le destructeur de classe

Le destructeur est une méthode de la classe qui est appelée automatiquement à la destruction d’un objet, c’est à dire lorsque la référence de cet objet est détruite.

Constructeur et destructeur

MaClass.php

```php
class MaClass {
    function __destruct() {
        echo "Un objet vient d'être détruit.";
    }
}
```

Constructeur et destructeur

index.php

```php
$monObjet = new MaClass();
unset($monObjet); // Un objet vient d'être détruit.
// ...
```

Cas pratique

Cas pratique

Pour ce TP, nous allons travailler en collectif.

Je vous invite à la réflexion, et je travail avec vous !

Cas pratique

Pour ce TP… à vous de jouer… 45 minutes de réflexion intensive.. ca devrait le faire.

Portée et Visibilitée

- La visibilitée
- La portée “statique”
- Les mutateurs
- Les accesseurs
- L’héritage
- Les abstractions
- Les interfaces
- Les espaces de nom
- Auto-chargement

Visibilité

Visibilité

Qu'est ce que c'est ?

La visibilité permet de définir comment on peut accéder aux propriétés et méthodes.

Visibilité

La visibilité "`public`"

`public` rend les propriétés ou méthodes accessibles au niveau de l’objet , de la classe et des classes étendues.

Visibilité

La visibilité "`public`"

MaClass.php

```php
class MaClass {
    public $propriete = "FooBar";
}
```

Visibilité

La visibilité "`public`"

```php
$monObjet = new MaClass();
echo $monObjet->propriete; // FooBar
```

Visibilité

La visibilité "`private`"

`private` rend les propriétés ou méthodes accessibles seulement au niveau de la classe qui les a défini, mais pas au niveau des objets, ni au niveau des classes étendues (extends).

Visibilité

La visibilité "`private`"

MaClass.php

```php
class MaClass {
    private $propriete = "FooBar";
}
```

Visibilité

La visibilité "`private`"

```php
$monObjet = new MaClass();
echo $monObjet->propriete; // Fatal error: Cannot access private property
```

Visibilité

La visibilité "`protected`"

`protected` rend les propriétés ou méthodes accessibles au niveau de la classe qui les a défini, au niveau des classes étendues (extends), mais pas au niveau des objets

Visibilité

La visibilité "protected"

```php
class A {
    protected $propriete = "FooBar";
}
class B extends A {
    public getPropriete() {
        return $this->propriete; // Lit la valeur de A::propriete
    }
}
```

Visibilité

La visibilité "protected"

```php
$a = new A();
// $a->propriete // Fatal error: Cannot access private property A::propriete

$b = new B();
echo $b->getPropriete(); // FooBar
```

Visibilité

La visibilité "`final`"

Quand une **méthode** est définie comme `final`, elle ne peut pas être surchargée par la classe fille.

Visibilité

```php
class A {
    final function action() {
        return "Foo";
    }
}
class B extends A {
    public function action() {
        return "Bar";
    }
}

// Fatal error: Cannot override final method A::action()
```

Visibilité

Quand une **classe** est définie comme `final`, elle ne peut pas être étendue.

```php
final class A { /*...*/ }
class B extends A { /*...*/ }

// Fatal error: Class B may not inherit from final class (A)
```

Visibilité

Pratique

- Protéger les propriétés marque et modèle de la classe Voiture.
- Créer une voiture de la marque Tesla, modèle s90.
- Vérifier que les valeurs soient bien affectées aux propriété dans le destructeur de classe.
- Essayer d'afficher les valeurs depuis le fichier d'index.

Visibilité

Voiture.php

```php
<?php
class voiture {
    private $brand;
    private $model;

    public function __construct(string $brand, string $model) {
        $this->brand = $brand;
        $this->model = $model;
    }

    public function __destruct() {
        echo "<div>Marque : ".$this->brand."</div>";
        echo "<div>Modèle : ".$this->model."</div>";
    }
}
```

Visibilité

index.php

```php
<?php
include "Voiture.php";
$voiture = new Voiture("Tesla", "s90");
echo "<div>Marque : ".$voiture->brand."</div>";
echo "<div>Modèle : ".$voiture->model."</div>";
```

La portée “statique”

La portée “statique”

Qu'est ce que c'est ?

Déclarer des propriétés ou des méthodes comme statiques permet d'y accéder sans avoir besoin d'instancier la classe.

On ne peut pas accéder à une propriété déclarée comme statique avec l'objet instancié, mais il est possible d'accéder à une méthode statique.

Si aucune déclaration de visibilité n'est spécifiée, la propriété ou la méthode sera automatiquement considérée comme ***public***.

Comme les méthodes statiques peuvent être appelées sans qu'une instance d'objet n'ait été créée, la pseudo-variable `$this` n'est pas disponible dans les méthodes déclarées comme statiques.

La portée “statique”

Déclarer une propriété ou méthode statique

Une propriété statique peut être déclarer en utilisant un littéral ou une constante et ne peut pas contenir d'expression.

Une méthode statique se déclare comme une méthode "classique" en ajoutant le mot clé "**`static`**".

```php
class MaClass {
    public static $propriete_statique = "foo";
    const MA_CONSTANTE = "bar";
    public static function methode_statique(){}
}
```

La portée “statique”

Accéder à une propriété ou méthode statique

On ne peut pas accéder à des propriétés statiques à travers l'objet en utilisant l'opérateur `->`.

L'opérateur de résolution de portée se note **`::`** et s'appel **`Paamayim Nekudotayim`**.

`Paamayim Nekudotayim` est un terme hébreu signifiant "Deux fois deux points".

La portée “statique”

MaClass.php

```php
class MaClass {
    const MA_CONSTANTE = "Valeur constante";
    public static $propriete = "Valeur de la propriété";
    public static function methode(){
        return "Valeur de la méthode";
    }
}
```

La portée “statique”

index.php

```php
echo "<div>".MaClass::MA_CONSTANTE."</div>";
echo "<div>".MaClass::$propriete."</div>";
echo "<div>".MaClass::methode()."</div>";
```

Les mutateurs

Les mutateurs

Les mutateurs (Setter)

L'objectif du Mutateur est de **donner une valeur** à une propriété de l'objet.

MaClass.php

```php
class MaClass {
    private $propriete;
    public function setPropriete($argument) {
        $this->propriete = $argument;
        return $this;
    }
}
```

Les mutateurs

Les mutateurs (Setter)

index.php

```php
$monObjet = new MaClass();
$monObjet->setPropriete("FooBar");
```

Les accesseurs

Les accesseurs

Les accesseurs (Getter)

L'objectif du Getter est de **rendre une valeur** d'une propriété de l'objet.

MaClass.php

```php
class MaClass {
    private $propriete;
    public function getPropriete() {
        return $this->propriete;
    }
}
```

Les accesseurs

Les accesseurs (Getter)

index.php

```php
$monObjet = new MaClass();
echo "<div>".$monObjet->getPropriete()."</div>"; // FooBar
```

Cas pratique

Cas pratique

Pratique

Créer les Setter / Getter de la classe Personnage

L’héritage

L’héritage

Qu'est ce que c'est ?

L’héritage consiste à créer un objet — l'enfant — à partir d’une classe existante — le parent.

La classe enfant hérite des propriétés et des méthodes de la classe parente.

L'héritage est utile pour définir et abstraire certaines fonctionnalités communes à plusieurs classes, tout en permettant la mise en place de fonctionnalités supplémentaires dans les classes enfants,

L’héritage

Déclarer la classe Parent et la classe Enfant

Déclaration de la classe **parent**

MaClassParent.php

```php
class MaClassParent {
    public $propriete = "Valeur de la propriété du parent";
    public function methode_commune(){}
}
```

L’héritage

Déclarer la classe Parent et la classe Enfant

Déclaration de la classe **enfant** qui hérite de la classe **parent**.

L'héritage de classe se fait avec le mot clé **extends**.

MaClassEnfant.php

```php
class MaClassEnfant extends MaClassParent {
    public function methode_supplementaire(){}
}
```

L’héritage

Utilisation de la classe Enfant

index.php

```php
include "MaClassParent.php";
include "MaClassEnfant.php";

$enfant = new MaClassEnfant();
echo "<div>".$enfant->propriete."</div>"; // Valeur de la propriété du parent
```

L’héritage

Accès aux propriétés et méthodes du parent

Une classe enfant qui hérite d'une classe parent peut accéder aux propriétés et méthodes de la classe parent avec le mot clé parent.

MaClassParent.php

```php
class MaClassParent {
    public $propriete = "Valeur de la propriété du parent";
    public function methode_commune(){}
}
```

L’héritage

Accès aux propriétés et méthodes du parent

MaClassEnfant.php

```php
class MaClassEnfant extends MaClassParent {
    public function methode_supplementaire(){
        return parent::$propriete;
    }
}
```

L’héritage

Surcharge des méthodes du parent

Une classe enfant peut surcharger les méthodes de la classe parent, en re-déclarant la méthode.

La méthode appliquée sur l'objet de la classe enfant sera la méthode de la classe enfant et non plus la méthode de la classe parent.

L’héritage

Surcharge des méthodes du parent

```php
class A {
    public function foo() {
        return "Foo";
    }
}
class B extends A {
    public function foo() {
        return "Bar";
    }
}

$a = new A;
echo $a->foo(); // Foo
$b = new B;
echo $b->foo(); // Bar
```

Les abstractions

Les abstractions

Qu'est ce que c'est ?

Les classes et méthodes abstraite sont implantées depuis PHP 5.

Une classe abstraite ne peut pas être instanciée, et toute classe contenant au moins une méthode abstraite doit elle-aussi être abstraite.

On définie une classe abstraite avec le mot clé **`abstract`**.

Les abstractions

Qu'est ce que c'est ?

Déclaration de la classe abstraite

```
1  abstract class MaClassAbstraite {
2  }
```

Les abstractions

Qu'est ce que c'est ?

Déclaration d'une méthode abstarite

```php
abstract class MaClassAbstraite {
    abstract protected function maMethodeAbstraite_a();
    abstract protected function maMethodeAbstraite_b($argument);
}
```

Les abstractions

Qu'est ce que c'est ?

Une méthode définie comme abstraite déclare simplement la signature de la méthode - elle ne contient pas de code.

Une classe abstraite oblige les classes dérivées à avoir les mêmes fonctionnalités, elle force l’héritage :

"*Si tu veux être comme moi tu dois faire comme moi*"

Les abstractions

Qu'est ce que c'est ?

Héritage de classe abstraite avec méthodes abtraites

```php
class MaClassEnfant extends MaClassAbstraite {
    protected function maMethodeAbstraite_a() {
        // ...
    }
    public function maMethodeAbstraite_b($argument) {
        // ...
    }
}
```

Les abstractions

Qu'est ce que c'est ?

Lors de l'héritage d'une classe abstraite, toutes les méthodes marquées comme abstraites dans la déclaration de la classe parente doivent être définies par l'enfant avec la même visibilité ou une visibilité moins restreinte.

Une classe abstraite n’est pas obligée d’avoir que des méthodes abstraites.

Les abstractions

Qu'est ce que c'est ?

```php
abstract class MaClassAbstraite {
    abstract protected function maMethodeAbstraite_a();
    abstract protected function maMethodeAbstraite_b($argument);
    protected function maMethode(){
        // ...
    };
}
```

La classe qui hérite de la classe abstraite n'est pas obligé de re-déclarer les méthodes concrètes.

Les interfaces

Les interfaces

Qu'est ce que c'est ?

Une interface objet permet de créer du code qui spécifie quelles méthodes une classe doit implémenter, sans avoir à définir comment ces méthodes fonctionneront.

Une interface n'a aucune importance fonctionnelle, mais apporte un intérêt conceptuel et structurel.

Les interfaces

Déclarer une interface

Les interfaces sont définies de la même façon que pour une classe, mais en utilisant le mot-clé *interface* à la place de *class*, et sans qu'aucune des méthodes n'ait son contenu de spécifié.

```php
interface MonInterface {
    public function maMethode_a();
    public function maMethode_b($argument);
}
```

Toutes les méthodes déclarées dans une interface doivent être publiques.

Les interfaces

Implémenter une interface

l'implémentation d’une interface se fait avec le mot clé **implements** lors de la déclaration d'une classe.

```php
class MaClass implements MonInterface {
    public function maMethode_a() {
        // ...
    }
    public function maMethode_b($argument) {
        // ...
    }
}
```

Les espaces de noms

Les espaces de noms

Qu'est ce que c'est ?

Problème

Avec PHP, il n'est pas possible de créer deux constantes, fonctions ou classes portant le même nom.

Les espaces de noms

Qu'est ce que c'est ?

Problème

Avec PHP, il n'est pas possible de créer deux constantes, fonctions ou classes portant le même nom.

Solution

On peut se représenter chaque constante, fonction et classe comme de fichier et les espaces de nom comme des répertoires virtuel.

Par défaut, les constantes, fonctions et classes sont dans l'espaces de nom **global** — équivalent du dossier racine.

Les espaces de noms

Pourquoi ?

L'utilisation des espaces de nom évite les conflits lors de l'utilisation de librairies de scripts par exemple.

Les espaces de noms

Déclarer un namespace

Un espace de nom se déclare avec le mot clé **namespace**.

Les espaces de noms

Déclarer un namespace

Déclaration d'un espace de nom

```
1  namespace MonEspaceDeNom;
```

Les espaces de noms

Déclarer un namespace

Déclaration d'un sous espace de nom

```
1  namespace Mon\Espace\De\Nom;
```

Les espaces de noms

Déclarer un namespace

Toutes les fonctions et classes déclarées après un espace de nom feront parties de cet espace de nom.

Les espaces de noms

```php
namespace A;
class MaClass {
    public function methode(){
        echo "Je suis la class MaClass de l'espace de nom A.<br>";
    }
}
$monObjetA = new MaClass();
$monObjetA->methode(); // Je suis la class MaClass de l'espace de nom A.

namespace B;
class MaClass {
    public function methode(){
        echo "Je suis la class MaClass de l'espace de nom B.<br>";
    }
}
$monObjetB = new MaClass();
$monObjetB->methode(); // Je suis la class MaClass de l'espace de nom B.
```

Les espaces de noms

Utiliser une classe dans un namespace

Pour utiliser une classe stockée dans un espace de nom

```
use \Mon\Espace\De\Nom\MaClass;
```

Les espaces de noms

Utiliser une classe dans un namespace

Utiliser un espace de nom de remplace pas un **include**.

```php
include "MaClass.php";

use \A\MaClass;

$monObjet = new MaClass();
$monObjet->methode();
```

Les espaces de noms

Créer un alias

```php
include "MaClassA.php";
include "MaClassB.php";

use \A\MaClass as mcA;
use \B\MaClass as mcB;

$monObjetA = new mcA();
$monObjetA->methode();
$monObjetB = new mcA();
$monObjetB->methode();
```

Chargement automatique

Chargement automatique

Pourquoi ?

- Pour éviter les inclusions manuelle de fichiers de classe souvent nombreux.
- Pour éviter les inclusions de fichiers de classe parfois inutiles.

Chargement automatique

Comment ?

Le nom du fichier doit correspondre au nom de la classe.

MaClass.php

```php
class MaClass {
    $propriete = "FooBar";
}
```

Le fichier `MaClass.php` sera chargé automatiquement lors de l'instance de de `MaClass`.

Chargement automatique

Comment ?

index.php

```php
function autochargement($nomClass) {
    include_once $nomClass . '.php';
}
spl_autoload_register('autochargement');

$monObjet  = new MaClass();
echo $monObjet->propriete;
```

La fonction **spl_autoload_register()** peut être appelé plusieurs fois pour créer plusieurs fonctions d'auto-chargement.

Chargement automatique

La fonction `__upload()`

La fonction __autoload est devenu obsolète depuis PHP 7.2

```php
function __autoload($nomClass) {
    include $nomClass . '.php';
}

$monObjet  = new MaClass();
echo $monObjet->propriete;
```

Chargement automatique

Pratique

Utiliser **`spl_autoload_register`** pour inclure les classes `Vehicule`, `Voiture` et `Moto`

Chargement automatique

index.php

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

$voiture  = new Voiture();
$moto  = new Moto();
```

Vehicule.php

```php
<?php
class Vehicule {}
```

Voiture.php

```php
<?php
class Voiture extends Vehicule {}
```

Moto.php

```php
<?php
class Moto extends Vehicule {}
```

Plus loin

- Documenter une classe
- Le chaînage
- Les méthodes “magiques”
- Polymorphisme

Documenter une classe

Documenter une classe

Commentaire de documentation

```
/**
 * Ceci est un bloc de commentaire de documentation
 */
```

Documenter une classe

Liste des mots clés

- **@access** Méthode d'accès: **public** ou **private**
- **@author** Auteur du paquet
- **@package** Nom du paquet
- **@param** Paramètre de fonction
- **@return** Valeur retourné par la fonction
- **@see** Référence
- **@since** Depuis la version...
- **@var** Variable d'objet
- **@version** Numéro de version du paquet

Documenter une classe

Exemple

MaClass.php

```php
<?php
/**
 * Ce commentaire apporte une documentation générale sur la classe MaClass
 *
 * @usage
 * $monObjet = new MaClass();
 *
 * @package  MaClass
 * @author   John Doe <john@example.com>
 * @version  $Revision: 1.3 $
 * @access   public
 * @see      http://www.example.com/maclass
 */
class MaClass {
    /**
     * Retourne la liste des mots-clés
     *
     * @access private
     * @param  string  $argument Argument de ma methode
     * @return array   Liste des mots clés
     */
    private function methode($argument) {
    }
}
```

Le chaînage

Le chaînage

Qu'est ce que c'est ?

Le chaînage des méthodes permet d’exécuter plusieurs méthodes d’affilée de façon simple et plus rapide.

Le chaînage

Comment ?

On utilise l’opérateur d’objet pour chaîner différentes méthodes.

```php
$monObjet->method1()->method2();
```

Le chaînage

Déclarer des méthodes chaînées

Pour utiliser le chaînage de méthodes, il faut que les méthodes chaînées retournent notre objet afin de pouvoir exécuter la méthode suivante. Dans le cas contraire, une erreur sera renvoyée.

Le chaînage

Déclarer des méthodes chaînées

MaClass.php

```php
class MaClass {
    private $propriete;
    public function methode1(string $argument) {
        $this->propriete = $argument;
        return $this;
    }
    public function methode2() {
        $this->propriete = ucword($this->propriete);
        return $this;
    }
    public function getPropriete() {
        return $this->propriete;
    }
}
```

Le chaînage

Utiliser des méthodes chaînées

```php
$monObjet = new MaClass;
$monObjet->method1("foo bar")->method2();
echo $monObjet->getPropriete(); // Foo Bar
```

Les méthodes “magiques”

Les méthodes “magiques”

Qu'est ce que c'est ?

Les **méthodes magiques** sont des méthodes qui, si elles sont déclarées dans une classe, ont une fonction déjà prévue par le langage.

Les méthodes “magiques”

Les méthodes "magiques"

__construct()

Le **constructeur de la classe** est déclenché dès l'instance de l'objet et à chaque instance.

Il peut être utilisé pour exécuter une procédure à l'instance de l'objet et à passer des données en entrée à l'instance de l'objet**.**

__destruct()

Le destructeur de la classe est déclenché à la fin du script PHP.

Les méthodes “magiques”

Les méthodes "magiques"

__set()

Déclenchée lors de l'accès en écriture à une propriété de l'objet ;

__get()

Déclenchée lors de l'accès en lecture à une propriété de l'objet ;

__call()

Déclenchée lors de l'appel d'une méthode inexistante de la classe (appel non statique) ;

Les méthodes “magiques”

Les méthodes "magiques"

__callstatic()

Déclenchée lors de l'appel d'une méthode inexistante de la classe (appel statique) : disponible depuis PHP 5.3 et 6.0 ;

__isset()

Déclenchée si on applique **isset()** à une propriété de l'objet ;

__unset()

Déclenchée si on applique **unset()** à une propriété de l'objet ;

Les méthodes “magiques”

Les méthodes "magiques"

__unset()

Déclenchée si on applique **unset()** à une propriété de l'objet ;

__sleep()

Exécutée si la fonction **serialize()** est appliquée à l'objet ;

__wakeup()

Exécutée si la fonction **unserialize()** est appliquée à l'objet ;

Les méthodes “magiques”

Les méthodes "magiques"

__toString()

Appelée lorsque l'on essaie d'afficher directement l'objet : **echo $object;** ;

__set_state()

Méthode statique lancée lorsque l'on applique la fonction **var_export()** à l'objet ;

__clone()

Appelés lorsque l'on essaie de cloner l'objet ;

Les méthodes “magiques”

Les méthodes "magiques"

__autoload()

Cette fonction n'est pas une méthode, elle est déclarée dans le scope global et permet d'automatiser les "include/require" de classes PHP.

Poly morphisme

Polymorphisme

La notion de polymorphisme découle de la notion d’héritage.

Supposons qu’une classe Laser et trois classes dérivées CD, DVD et BlueRay soient définies.

Les trois classes dérivées ont toutes les caractéristiques de la classe Laser.

Ainsi, des objets de type CD, DVD et BlueRay peuvent être traités comme des objets de type Laser.

Cependant, il est possible de surcharger les méthodes d’instance de la classe parente afin d’induire des comportements spécifiques aux classes dérivées.

Polymorphisme

```
class Laser {
  public function caracteristiques() {
       echo "Disque laser<br>";
  }
}
class CD extends Laser{
  public function caracteristiques() {
       parent::caracteristiques();
       echo "Capacité : 700 Mo<br>";
  }
}
class DVD extends Laser {
  public function caracteristiques() {
       parent::caracteristiques();
       echo "Capacité : 4,7 Go<br>";
  }
}
class BlueRay extends Laser {
  public function caracteristiques() {
       parent::caracteristiques();
       echo "Capacité : 25 Go<br>";
  }
}
```

Polymorphisme

Les classes héritées CD, DVD et BlueRay surchargent la méthode publique caracteristiques de la classe parente Laser.

Elles conservent le comportement de la méthode parente (parent::caracteristiques();) et la complètent avec des informations qui leur sont spécifiques.

 Vous êtes des pros de la P.O.O.