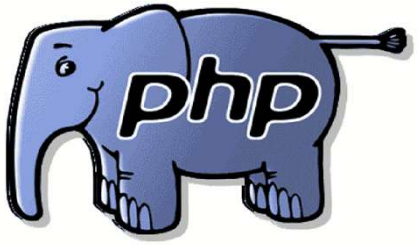
Développeur Web et Web Mobile

Le langage



Partie 3 : Le PhP avancé
La programmation orientée objet

PLAN DU COURS

Les concepts de bases

Les concepts avancés

PLAN DU COURS
suite

Les concepts avancés

Les fonctions

INTRODUCTION

- PhP offre un véritable modèle objet.
- Parmi les fonctionnalités disponibles on trouve :
  - la visibilité dans les classes
  - la notion de classe abstraite
  - les classes « final »
  - les méthodes magiques
  - les interfaces
  - le typage
  - les espaces de nommage

LA NOTION DE CLASSE
syntaxe de base

- Une classe se définie par le mot clé « **class** » suivi du **nom** de cette classe.
- Le **nom** de la classe doit avoir son **premier caractère en majuscule**.
- Les accolades « {} » constituent le corps de la classe.

Exemple :

```
<?php
class MaClasse {
    définition des propriétés ou attributs
    définition des mèthodes
}
?>
```

LA NOTION DE CLASSE
les propriétés

- Les variables d'une classe s'appellent des « **propriétés** » (ou encore des attributs).
- Elles sont déclarées en utilisant un « **modificateur d'accès** » pouvant prendre les valeurs :

  « **public** » « **protected** » ou « **private** »

  (voir le chapitre sur la visibilité)

  En PhP 4 seul le mot clé « var » existait qui est compris en PhP 5 comme « public »
- On distingue deux types **deux types** de propriétés :
  - les propriétés **d'instance**
  - les propriétés **de classe**

LA NOTION DE CLASSE
les propriétés d'instance

- Après avoir défini une classe, l'utilisation que l'on peut en faire est de créer des objets.

  c'est « l'**instanciation** »

- On instancie une classe en utilisant le mot clé « new » suivi du nom de la classe et de parenthèses « () ».

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

- Chaque objet créé possèdera sa propre copie des propriétés d'instance de la classe :

```
public $maPropriete;
```

- Les valeurs de ces propriétés représentent l'état de l'objet.

LA NOTION DE CLASSE
les propriétés de classe

- Il peut arriver que certaines propriétés soient commune à l'ensemble des objets instanciés.
- C'est ce que l'on appelle les propriétés de classe.
- Elles se déclarent au niveau de la classe en faisant précéder la définition de la propriété par le mot clé « **static** ».

Exemple :

```
public static $maStatic;
```

LA NOTION DE CLASSE
accès aux propriétés

- **A l'intérieur de la classe**, à l'intérieur d'une méthode, l'accès à une propriété :
  - Propriété d'instance

    Utilisation de « $ », du mot clé « **this** » (signifiant cet objet là) suivi d'un flèche « **->** » et du **nom** de la propriété

    ```
    $var = $this->section;
    ```

  - Propriété de classe

    Utilisation du mot clé « **self** » suivi de l'opérateur de portée « **::** » et de la propriété

    ```
    $var = self::$maStatic;
    ```

- **A l'extérieur de la classe**, l'accès à une propriété dépend de sa visibilité
  (voir chapitre sur la visibilité)

LA NOTION DE CLASSE
les propriétés : exemple

```php
<?php
class Stagiaire {
    public static $nbrStag;
    public $nom;
    public $numIdentification;
    public $section;

    public function infosStagiaire() {
        echo 'numéro : ' . self::$nbrStag;
        echo 'stagiaire : ' . $this->nom;
        echo 'section : ' . $this->section;
        echo 'num ident : ' . $this->numIdentification;
    }
}
```

nom de la classe avec majuscule

déclaration propriété de classe

déclaration de trois propriétés

une méthode

accès à la propriété de classe

accès aux propriétés d'instance

LA NOTION DE CLASSE
initialisation des propriétés

- Lors de la déclaration des propriétés, il est possible de les **initialiser**.
- Ceci n'est possible que par des valeurs constantes et ne peut pas être le résultat d'un calcul.

Exemple :

```
class Stagiaire {
    public $nom = 'inconnu';
    public $numIdentification = 0;
    public $section = 'inconnue';
```

LA NOTION DE CLASSE
les constantes

- Il peut être intéressant de définir des valeurs constantes accessibles en interne par les méthodes ou en externe.
- Syntaxiquement, les constantes se déclarent par le mot clé « **const** » suivi du **nom sans l'utilisation du « $ »**.
- La valeur affectée à cette constante **doit être une valeur immédiate** (pas le fruit d'un résultat ou d'une propriété).
- Par convention, on utilise les majuscules.

Exemple :

```
class maClasse {
    const MA_CONSTANTE = 'bonjour';
```

LA NOTION DE CLASSE
accès aux constantes

- A partir d'une méthode de la classe, l'accès à une constante se fait par le mot clé « **self** » suivi de l'opérateur de portée « **::** ».
- De l'extérieur de la classe, plusieurs possibilités nous sont offertes :

1. En utilisant le **nom de la classe** suivi de « **::** »
```
MaClasse::MA_CONSTANTE;
```
2. En référençant la classe par une variable contenant son nom (depuis PhP 5.3)
```
$nomClasse = 'MaClasse';
$nomClasse::MA_CONSTANTE;
```
3. Par la variable référençant directement un objet
```
$objet = new MaClasse();
$objet::MA_CONSTANTE;
```

LA NOTION DE CLASSE
les méthodes

- Ce sont des fonctions propres à une classe donnée.
- Là aussi, une méthode peut être d'instance ou de classe (static).
- Les modificateurs « public », « protected » et « private » s'appliquent aux méthodes (voir chapitre sur la visibilité).
- Depuis php 5, le typage des paramètres de méthode est possible.

Exemple :

```
function changeCouleur(Couleur $coul){ … }
```

$coul : objet de la classe « Couleur »

LA NOTION DE CLASSE
les constructeurs / destructeurs

- Le constructeur d'une classe est une méthode particulière et unique en PhP.
- Il est appelé lors de l'instanciation d'un nouvel objet (utilisation de new).
- Il porte le nom **« __construct() »** et est généralement « public ». Avant Php 5, le constructeur portait le nom que la classe.
- Son rôle est d'initialiser les propriétés d'instance avec les éventuels paramètres qui lui sont passés.
- Php 5 introduit la notion de destructeur appelé lorsqu'un objet n'est plus référencé. Il porte le nom « __destruct() ».

LA NOTION DE CLASSE
plusieurs constructeurs

- Certain langage objet autorise la déclaration de plusieurs constructeurs (surcharge de méthodes : arguments différents).
- Comme **c'est impossible en Php**, il faut faire appel aux fonctions spécifiques aux arguments de fonction :
  - « **func_get_args()** » : retourne un tableau de tous les arguments
  - « **func_num_args()** » : retourne le nombre d'arguments
  - « **funct_get_arg($numArg)** » : retourne la valeur d'un argument spécifié (de 0 à func_num_args() – 1)

LA VISIBILITE
« public »

- Le modificateur « **public** » s'applique aux propriétés ou aux méthodes.
- Il signifie qu'à partir d'un objet la propriété (ou méthode) sera accessible via l'opérateur « **->** ».

Exemple :

```
<?php
class Stagiaire {
    public $nom;
}

$stag1 = new Stagiaire();
$stag1->nom = "Dupond";
?>
```

LA VISIBILITE
« private »

- Contrairement au modificateur « public », « **private** » va garantir l'encapsulation des données de la P.O.O.
- Seules les méthodes de la classe auront accès aux propriétés (ou méthodes) privées.

Exemple :

```
<?php
class Stagiaire {
  private $nom;
}

$stag1 = new Stagiaire();
$stag1->nom = "Dupond";
?>
```

- Soit $nom est non visible de l'extérieur
- Soit il peut être vu : dans ce cas il faut passer par les accesseurs pour lire ou modifier sa valeur

LA VISIBILITE
« protected »

- Un troisième modificateur existe qui permet de protéger les propriétés d'une classe tout en laissant aux classes dérivées y accéder (voir chapitre sur l'héritage).
- C'est le modificateur « protected ».

Exemple :

Classe principale

```
<?php
class Personne {
        protected $nom;
}
?>
```

Classe Dérivée ou héritée

```
<?php
class Stagiaire {
    public function __construct($ident) {
        $this->nom = $ident;
    }
}
?>
```

Propriété accessible

```
<?php
$stag1 = new Stagiaire("Dupond");
?>
```

LES ACCESSEURS
« getteur » et « setteur »

- Il est fortement conseillé de définir les propriétés en « private » et de définir les accesseurs nécessaires :

Exemple :

```
class Stagiaire {
    private $nom;
    // définition des getteurs
    public function getNom() {
        return $this->nom;
    }
    // définition des setteurs
    public function setNom($ident) {
        $this->nom = strtoupper($ident);
    }
}
```

```
$stag = new Stagiaire();
echo $stag->getNom();
```

```
$stag = new Stagiaire();
$stag->setNom("Dupont");
```

L'HERITAGE
introduction

- Un des grands principes de la P.O.O.
- Une classe « fille » hérite d'une classe « mère ».
- La déclaration de cette classe fille se fait en faisant suivre le nom de sa classe par le mot clé « **extends** » suivi du nom de la classe mère.
- La classe mère doit être connue avant.
- La classe fille **hérite** des toutes les **propriétés** et **méthodes « public »** ou **« protected »** de la classe mère.

- Que ce passe t-il lors de l'héritage ?
- Deux cas se présentent :
  1. soit la classe fille ne définie pas de constructeur -> le constructeur de la classe mère est appelé.
  2. soit elle définie le constructeur et celui de la classe mère est ignoré. Il y a surcharge, la signature peut être différente.
     Elle peut néanmoins appeler explicitement le constructeur de la classe mère en utilisant :

```
parent::__construct()
```

L'HERITAGE
surcharge / redéfinition

- Une classe fille peut **surcharger** une méthode de la classe mère si :
  - celle de la classe mère n'est pas « abstract » ni « final » (voir chapitre correspondant).
  - **même nom, paramètres différents**

- Une classe fille peut **redéfinir** une méthode de la classe mère si :
  - celle de la classe mère est « abstract » (voir chapitre correspondant).
  - **même nom, même paramètres**

L'HERITAGE (suite)

- Si on veut interdire la surcharge ou la redéfinition d'une fonction, il faut déclarer celle-ci en « **final** ».
- Une classe fille peut étendre la portée d'une méthode de la classe mère :

mère	fille
public	public
protected	public protected
private	public protected private

LES CLASSES ABSTRAITES

- Une classe « abstraite » est une classe qui contient des propriétés et des méthodes mais qui ne peut pas être instanciée.
(Pas de réalité réelle)
- Elle est déclarée par le mot clé « **abstract** »
- Elle doit obligatoirement être dérivée
- Ces méthodes peuvent être :
  - Réelles : définies en public, protected, private

  -> communes à toutes les classes filles
  - Abstraites : (mot clé « abstract ») pas de définition du corps de la fonction

  -> les classes filles ont obligation de les redéfinir

LES CLASSES ABSTRAITES (suite)

- Si une classe possède une méthode abstraite, alors la classe est abstraite et doit être définie comme telle.

Exemple :

```
abstract class Vehicule {
        private numIdent;

    public function getIdent(){
        return $this->numIdent;
    }
    abstract protected function
        seDeplace() ;
}
```

```
class Voiture extends Vehicule {
        function seDeplace() {
                ….;
        }
}

class Bateau extends Vehicule {
        function seDeplace() {
                ….;
        }
}
```

LES INTERFACES

- Une interface permet de créer un « **modèle** » que les classes pourront « **implémenter** ».
- Ce modèle est en fait un **ensemble de méthodes** (public ou protected, static ou non) et de **constantes**.
- Les classes implémentant cette interface **auront obligation** de redéfinir **toutes** ses méthodes.
- Contrairement à l'héritage, une classe peut **implémenter plusieurs** interfaces.
- Elle se déclare par le mot clé « **interface** » en lieu et place du mot clé « class ».

LES INTERFACES (suite)

- Elle se déclare par le mot clé « **interface** » en lieu et place du mot clé « class ».
- Pour la classe fille, on utilise le mot clé « **implements** » suivi du nom de l'interface.
- Une interface peut servir à typer les paramètres d'une méthode.

Exemple :

```
interface IReparable {

   public function repare();

}
```

```
abstract class Vehicule
            implements IReparable {

            public function repare() {
                // todo;
    }
}
```

LES TRAITS

- PhP 5.4 défini la notion de « **trait** » qui servent à mettre en commun des méthodes (voir des propriétés) à plusieurs classes sans rapport entre elles.
- C'est presque équivalent à la notion d'héritage.
- Une classe peut « **utiliser** » plusieurs « traits ».
- La déclaration d'un « trait » se fait en utilisant le mot clé « **trait** » en lieu et place du mot clé « class ».

LES TRAITS (suite)

- Une classe utilise un « traits » en le déclarant par le mot clé « use ».

Exemple :

```
trait achetable {

public function acheter(){
        // todo;

}
```

```
class Voiture {
        use achetable;
        …..
          }
```

```
class Bateau {
          use achetable;
          ….
}
```

```
$voit1 = new Voiture();
$voit1->acheter();
```

LES SURCHARGES MAGIQUES

- En PhP, la surcharge magique permet de créer dynamiquement des propriétés ou des méthodes.
- Ces méthodes sont appelées par divers types d'actions :
  - accès à des propriétés inaccessibles
  - appel de méthodes inaccessibles

- Elles doivent être déclarées en « public ».

LES SURCHARGES MAGIQUES
sur les propriétés

- Elles sont maintenant au nombre de 4:

1. __set($nomPropriete, $valeurPropriete)

   On récupère le nom et la valeur de la propriété inaccessible (n'existant pas ou private)

2. __get(string $nomPropriete)

   Retourne la valeur de la propriété

3. __isset(string $nomPropriete)

   Sollicitée lorsque les méthodes isset() ou empty() sont appelées sur une propriété inaccessible

4. __unset(string $nomPropriete)

   Sollicitée lorsque la méthode unset() est appelée sur une propriété inaccessible

LES SURCHARGES MAGIQUES
sur les méthodes

- Elles sont maintenant au nombre de 2:

1. __call()

Appelée lorsqu'on évoque une méthode inaccessible dans un contexte objet.

```
$obj = new MyClasse();
$obj->methInconnu('appel une methode inconnue');
```

⇒ Appel de la méthode magic « __call($nom, $arguments) »

2. __callStatic()

Appelée lorsqu'on évoque une méthode inaccessible dans un contexte de classe.

```
MyClasse:: methInconnu('appel une methode inconnue');
```

⇒ Appel de la méthode magic static « __callStatic($nom, $arguments) »

LES METHODES MAGIQUES

- D'autres méthodes magiques existent. On retiendra :
  - __tostring()

    Méthode appelée lorsqu'un objet est traité comme une chaine de caractère. Elle doit donc retourner une chaine.

    Ex. => echo $monObjet;

  - __sleep() et __wakeup()

    Méthodes utilisées respectivement lors de la sérialisation et la désérialisation d'objet.
    - « sleep » doit retourner un tableau des propriétés à sérialiser.
    - « wakeup » permet d'effectuer un traitement lors de la désérialisation.

LE CLONAGE

- Il est parfois nécessaire de faire une copie d'un objet et non simplement référencer l'objet par une autre variable.
  - ⇒ $obj1 = new MonObjet();
  - ⇒ $obj2 = $ obj1;

$obj2 pointe sur le même objet que $obj1

- Pour créer une réelle copie, utiliser le mot clé « clone »
  - ⇒ $obj2 = clone $obj1;

$obj2 pointe sur un nouvel objet

- PhP effectue une copie superficielle :
  - une propriété standard est clonée
  - Une propriété qui fait référence à un autre objet garde sa valeur (pointe toujours sur le même objet)

LE CLONAGE
suite

- Une méthode magique « __clone() » est appelée automatiquement, si elle est présente dans la définition de l'objet, lors du clonage.
- Elle permet' par exemple, de cloner les objets référencés par les propriétés.

Exemple :

```
$obj = new MonObjetClonable();

$obj->object1 = new AutreObject();

$obj2 = clone $obj;
```

```
class MonObjetClonable
{
  public $objet1;

  function __clone()
  {
    // Force la copie de this->object
   $this->object1 = clone $this->object1;
  }
}
```

LA COMPARAISON

- Le résultat de la comparaison d'objets est différent suivant l'opérateur :
  - Opérateur de comparaison « **==** »
    - ✓ **mêmes propriétés** et **même valeurs**
    - ✓ **instances de la même classe**

⇒ $obj1 = new MonObjet();
⇒ $obj2 = new MonObjet();

$obj1 == $obj2 FAUX
$obj1 === $obj2 FAUX

  - Opérateur d'identité « **===** »
    - ✓ ils font **référence à la même instance** de la même classe

⇒ $obj1 = new MonObjet();
⇒ $obj2 = $obj1;

$obj1 == $obj2 VRAI
$obj1 === $obj2 VRAI

LA SERIALISATION

- Il est quelquefois utile de pouvoir sérialiser (transformer un objet en chaine de caractères comprenant toutes ses propriétés et valeurs) un objet.
- Pour cela utiliser la fonction « serialize() ».
- Pour récupérer l'objet, utiliser la fonction « unserialize() » (attention la connaissance de la classe est obligatoire.
- Bien sur, les propriétés « static » ne sont pas sérialisables.

Fin de la partie 3