AFLOKKAT
Conseil et Formation

Programmation WEB avec PHP

Matière : Programmation WEB avec PHP

Date : 13.06.2022

Formateur : Malerba Sylvain

Auteur : Malerba Sylvain

SOMMAIRE

AFLOKKAT
Conseil et Formation

AFLOKKAT
Conseil et Formation

01 Classes et objets

01 Classes et objets

AFLOKKAT
Conseil et Formation

Une classe possède:

- **Un ensemble de variable appelé « attribut »**
- **Un ensemble de fonction appelé « méthode »**
- **Un constructeur**

01 Classes et objets

AFLOKKAT
Conseil et Formation

La convention de nommage d'une classe est la suivante:

- Nom de la classe en PascalCase: ex: EquipeFoot
- Nom de la méthode en camelCase, ex: getEquipe()
- Nom du fichier identique au nom de la classe, EquipeFoot.php
- Attribut en minuscule, attribut de classe avec un « _ » en préfixe ex: $_nom
- Un fichier par classe

01 Classes et objets (Exemple)

AFLOKKAT
Conseil et Formation

```php
<?php

class Eleve
{

    function getName(){

    }

}
```

01 Classes et objets (Instanciation)

- Nous pouvons appeler le fichier d'une classe avec require ou include (cf cours précédent)
- Le mot clé pour créé un objet est new

```php
<!DOCTYPE html>
<html>
<head>
    <title>Aflokkat PHP</title>
    <meta charset="utf-8">
</head>
<body>
<?php
    require_once "classes/Eleve.php";
    $eleve = new Eleve();
?>
</body>
</html>
```

01

Classes et objets (Attributs et méthodes)

AFLOKKAT
Conseil et Formation

- Il existe 3 types d'attributs: public, private et protected
- Il est conseillé de déclarer les attributs d'une classe en private ou en protected
- Une classe utilise des Getter ou Setter pour accéder à un attribut
- Accessible grâce à l'opérateur ->
- Dans la classe, l'instance courante est accessible via $this

01

Classes et objets (Exemples)

AFLOKKAT
Conseil et Formation

```php
class Eleve {

    private $prenom;
    private $nom;
    public $age;


    public function setNom($nom) // setter
    {
        $this->nom = $nom;
    }

   public function getNom() // getter
   {
        return $this->nom;
    }
}
```

01 Classes et objets (Exemples)

AFLOKKAT
Conseil et Formation

```php
require_once "classes/Eleve.php";
$eleve = new Eleve();
$eleve->setNom("Malerba");
echo "Nom : " . $eleve->getNom() . "<br>";
// On peut modifier et accéder à la propriété public $adresse
$eleve->ville = "Boulevard Paoli";
echo "Adresse : " . $eleve->ville . "<br>";
// On ne peut pas modifier la valeur directement de la propriété private
$prenom
$eleve->prenom = "Sylvain"
```

Nom : Malerba
Adresse : Boulevard Paoli

Fatal error: Uncaught Error: Cannot access private property Eleve::$prenom in C:\wamp64\www\php4\exo1.php on line *25*

Error: Cannot access private property Eleve::$prenom in C:\wamp64\www\php4\exo1.php on line *25*

Call Stack

#	Time	Memory	Function	Location
1	0.0012	404424	{main}()	...\exo1.php:0

01 Classes et objets (Constructeur)

AFLOKKAT
Conseil et Formation

- **Méthode __construct() est appelée à chaque instanciation de la classe**

```php
public function __construct($prenom, $nom){

    $this->prenom = $prenom;

    $this->nom = $nom;

}
```

01 Classes et objets (Constructeur)

AFLOKKAT
Conseil et Formation

```php
<?php

    require_once "classes/Eleve.php";

    $eleve = new Eleve("Sylvain", "Malerba");

    echo "Bienvenu : " . $eleve->getNom() . " ".$eleve->getPrenom()."<br>";


?>
```

Bienvenu : Malerba Sylvain

01

Classes et objets (Destructeur)

AFLOKKAT
Conseil et Formation

- **Méthode __destruct() optionnelle appelée à la destruction de l'objet**

```php
public function __destruct(){

    echo $this->prenom . " " . $this->nom . " est détruit<br>";

  }
```

01 Classes et objets (Destructeur)

AFLOKKAT
Conseil et Formation

- **Méthode __destruct() optionnelle appelée à la destruction de l'objet**

```
Bienvenu : Malerba Sylvain
Bienvenu : Musk Elon
Elon Musk est détruit
Sylvain Malerba est détruit
```

```php
<?php

    require_once "classes/Eleve.php";

    $eleve = new Eleve("Sylvain", "Malerba");

    echo "Bienvenu : " . $eleve->getNom() . " ".$eleve->getPrenom()."<br>";

    $eleve2 = new Eleve("Elon", "Musk");
     echo "Bienvenu : " . $eleve2->getNom() . " ".$eleve2->getPrenom()."<br>";
     unset($eleve2);

?>
```

01 Classes et objets (Affichage)

AFLOKKAT
Conseil et Formation

- **La méthode __toString() permet d'afficher comme on le souhaite le retour de echo**

```php
public function __toString() {

    return "<b>Eleve</b><br>Prenom : " . $this->prenom . "<br>Nom : " . $this->nom . "<br>";
}
```

```php
<?php
    require_once "classes/Eleve.php";

    $eleve = new Eleve("Hermione", "Granger");


    echo $eleve;  // OK


?>
```

Eleve
Prenom : Hermione
Nom : Granger

AFLOKKAT
Conseil et Formation

02 Constantes et méthodes

02 Les constantes de classe

AFLOKKAT
Conseil et Formation

- Déclarée avec le mot-clé const
- Accessible à l'intérieur de la classe par self::
- Accessible à l'extérieur de la classe par MyClass::

```php
class Eleve {

    private $prenom;
    private $nom;
    public $ville;

    const PROMO = "2022/2023";

     public function __construct($prenom, $nom){

        $this->prenom = $prenom;

        $this->nom = $nom;

     }
```

02 Les constantes de classe

AFLOKKAT
Conseil et Formation

```php
<?php

    require_once "classes/Eleve.php";

    $eleve = new Eleve("Sylvain", "Malerba");

    echo "Bienvenu : " . $eleve->getNom() . " ".$eleve->getPrenom()."<br>";

    echo "Tu es dans la promo ".Eleve::PROMO ;

?>
```

Bienvenu : Malerba Sylvain
Tu es dans la promo 2022/2023

02 Les propriétés statiques

AFLOKKAT
Conseil et Formation

- **Propriété statique : variable partagée par tous les objets de la classe**
- **Contrairement aux constantes de classe, elles peuvent être modifiées**

```php
public static $numero_eleve = 565;

public function __construct($prenom, $nom){

    $this->prenom = $prenom;

    $this->nom = $nom;

    self::$numero_eleve++;

}
```

02 Les propriétés statiques (Exemples)

AFLOKKAT
Conseil et Formation

```php
<?php

    require_once "classes/Eleve.php";

    $eleve = new Eleve("Sylvain", "Malerba");

    echo "Bienvenu : " . $eleve->getNom() . " ".$eleve->getPrenom()." eleve n°".
        Eleve::$numero_eleve."<br>";

    echo "Tu es dans la promo ".Eleve::getPromo()."<br>" ;

    $eleve2 = new Eleve("Elon", "Musk");

    echo "Bienvenu : " . $eleve2->getNom() . " ".$eleve2->getPrenom()." eleve n°
        ".Eleve::$numero_eleve."<br>";

?>
```

Bienvenu : Malerba Sylvain eleve n°566
Tu es dans la promo 2022/2023
Bienvenu : Musk Elon eleve n°567

02 Les méthodes statiques

AFLOKKAT
Conseil et Formation

- **Appartient à la classe et non à un objet instancié**
- **Appelée avec la syntaxe MyClass::**

```php
class Eleve {

    private $prenom;
    private $nom;
    public $ville;

    const PROMO = "2022/2023";

     public function __construct($prenom, $nom){

        $this->prenom = $prenom;

        $this->nom = $nom;

     }
```

Bienvenu : Malerba Sylvain
Tu es dans la promo 2022/2023

AFLOKKAT
Conseil et Formation

03 Heritage et interface

03 Heritage

AFLOKKAT
Conseil et Formation

- Mot-clé **extends** pour étendre une classe
- Un objet hérité se comporte de la même manière que son parent.
  Elle herite de l'ensemble des méthodes (public et protected) de la classe mère.
- Un objet ne peut hériter que d'une seule classe

03 Heritage (Exemple)

AFLOKKAT
Conseil et Formation

- La classe Eleve a été renommé en Personne. Une nouvelle classe Eleve a été créée héritant de la classe Personne.

```php
<?php

class Eleve extends Personne{


}
```

```php
<?php
    require_once "classes/Personne.php";
    require_once "classes/Eleve.php";

    $eleve = new Eleve("Sylvain", "Malerba");

    echo "Bienvenu : " . $eleve->getNom() . " ".$eleve->getPrenom()." est ".
        get_class($eleve)."<br>";

    $eleve2 = new Personne("Elon", "Musk");

    echo "Bienvenu : " . $eleve2->getNom() . " ".$eleve2->getPrenom()." est ".
        get_class($eleve2)."<br>";

?>
```

Bienvenu : Malerba Sylvain est Eleve
Bienvenu : Musk Elon est Personne

03 Heritage (Visibilité)

AFLOKKAT
Conseil et Formation

- Les propriétés private de la classe mère ne sont pas accessibles
- Les propriétés protected et public de la classe mère sont accessibles
- Les propriétés protected et private ne sont pas accessibles de l'extérieur

03 Heritage (Exemples)

AFLOKKAT
Conseil et Formation

```php
<?php

class Eleve extends Personne{

    public function setNom($nom) {

        $this->nom = $nom;

    }
}
```

```php
<?php
    require_once "classes/Personne.php";
    require_once "classes/Eleve.php";

    $eleve = new Eleve("Hermione", "Granger");

    echo "Bienvenu : " . $eleve->getNom() . " ".$eleve->getPrenom()."<br>";

    $eleve->setNom("Malerba");

    echo "Bienvenu : " . $eleve->getNom() . " ".$eleve->getPrenom()."<br>";


?>
```

Bienvenu : Granger Hermione
Bienvenu : Granger Hermione

03 Heritage (Exemples)

AFLOKKAT
Conseil et Formation

```php
class Personne {

    private $prenom;
    protected $nom;
    public $ville;
```

Bienvenu : Granger Hermione
Bienvenu : Malerba Hermione

03

Heritage (Surcharge)

AFLOKKAT
Conseil et Formation

Une classe fille :

- **peut avoir des propriétés/méthodes supplémentaires**
- **peut surcharger une méthode non private**
- **peut appeler une méthode d'une classe mère avec parent::**

03 Heritage (Exemple)

AFLOKKAT
Conseil et Formation

```php
<?php

class Eleve extends Personne{

    public function setNom($nom) {

        $this->nom = $nom;

    }

    public function getPrenom() // getter
    {
        $prenom = parent::getPrenom();
        return $prenom." (En apprentissage)";

    }
}
```

```php
<?php
    require_once "classes/Personne.php";
    require_once "classes/Eleve.php";

    $eleve = new Eleve("Hermione", "Granger");

    $personne = new Personne ("Albus","Dumbledore");

    echo "Prenom de Eleve : " . $eleve->getPrenom() . "<br>";
    echo "Prenom de Personne : " . $personne->getPrenom() . "<br>";


?>
```

Prenom de Eleve : Hermione (En apprentissage)
Prenom de Personne : Albus

03 Heritage (final)

- **final class :** la classe ne peut pas être étendue
- **final function :** la méthode ne peut pas être surchargée

```php
final class Personne {

    private $prenom;
    protected $nom;
    public $ville;
```

03 Exercice

AFLOKKAT
Conseil et Formation

Créer une classe Utilisateur.php qui :

- **contient 4 attributs privés : login, password, prenom, nom.**
- **ces champs sont requis par le constructeur**
- **créer leur setter respectifs**
- **la methode setPassword encrypte le password avec password_hash($password, PASSWORD_BCRYPT)**

Créer une page inscription.php

- **contient un formulaire avec login, mdp, confirmation de mdp, prenom, nom + btn submit**
- **Vérifiez qu'aucun champ ne soit vide**
- **Vérifiez que les mdp soient similaires**
- **Si tout est OK affiché « Inscription réussie » + les informations de l'utilisateur**

03

Les classes abstraites

AFLOKKAT
Conseil et Formation

- **Classe abstraite : classe qui ne peut être instanciée**
  - **si on veut instancier un objet, il faudra créer une classe qui en hérite**
  - **on accède tout de même à ses méthodes/propriétés statiques**
  - **elle peut contenir des méthodes implémentées, des attributs comme n'importe quelle classe et même un constructeur**

03 Les classes abstraites (Exemple)

AFLOKKAT
Conseil et Formation

```php
<?php

abstract class Personne {

    protected $prenom;
    protected $nom;
    protected $ville;
```

```php
<?php
    require_once "classes/Personne.php";
    require_once "classes/Eleve.php";

    $eleve = new Eleve("Hermione", "Granger");

    $personne = new Personne ("Albus","Dumbledore");


    echo $personne;//Erreur
    echo $eleve;  // OK



?>
```

03

Les méthodes abstraites

AFLOKKAT
Conseil et Formation

- **Méthode abstraite : méthode dénie dans une classe abstraite**
  - **seulement la signature de la fonction est déclarée (pas de code)**
  - **les classes filles doivent impérativement implémenter cette méthode**

03 Les méthodes abstraites (Exemple)

AFLOKKAT
Conseil et Formation

```php
<?php

abstract class Personne {

    protected $prenom;
    protected $nom;
    protected $ville;

    const PROMO = "2022/2023";

    public static $numero_eleve = 565;

     public function __construct($prenom, $nom){

       $this->prenom = $prenom;

       $this->nom = $nom;

       self::$numero_eleve++;

     }


     abstract public static function getPromo();
```

```php
<?php

class Eleve extends Personne{

    public static function getPromo()
    {
        return self::PROMO;
    }
```

03

Les interfaces

AFLOKKAT
Conseil et Formation

- **Une interface:**
  - **Ne peuvent être instanciées**
  - **Ne contiennent pas de propriétés**
  - **Contiennent uniquement des signature de méthodes**
  - **Aucune implémentation de code**
  - **Quand une classe implémente une interface, elle s'engage à implémenter les fonctions définies**
  - **Une classe peut implémenter plusieurs interfaces**

03 Exemple

AFLOKKAT
Conseil et Formation

```php
<?php

interface CheckPassword {

    public static function getMinCaractere();

}
```

```php
<?php

abstract class User implements CheckPassword {

}
```

```php
<?php class Admin extends Utilisateur {

    public static function getMinCaractere(){

        return 6;

    }

}
```

03 Exercice

AFLOKKAT
Conseil et Formation

- **Créer une classe abstraite Produit qui contient les attributs reference, description et prix**
- **Créer des classes Vetement et Chaussure enfants de la classe Produit**
- **Implémenter le code pour obtenir le rendu ci-dessous :**

Le vêtement T-shirt bleu (REF0001) coûte 50 €.
Le vêtement Pull rouge (REF0002) coûte 80 €.
Les chaussures Adidas Stan Smith (REF0003) en pointure 42 coûtent 90 €.

03

Aller plus loin ! (le chaînage de méthode)

AFLOKKAT
Conseil et Formation

- **Intérêt : exécute plusieurs méthodes d'affilée de façon simple et plus rapide**
- **Forme :** `$objet->methode1()->methode2()`
- **Attention : chaque méthode doit retourner l'objet $this**

03 Aller plus loin ! (le chaînage de méthode)

AFLOKKAT
Conseil et Formation

- **Exemple:**

```php
<?php
require_once "classes3/Produit.php";
require_once "classes3/Vetement.php";


$item = new Vetement("REF0001", "T-shirt col V", 50);

echo "Quantite : " . $item->getQuantite() . "<br>";   $item->addQuantite(
    )->addQuantite()->addQuantite()->deleteQuantite();

echo "Quantite : " . $item->getQuantite() . "<br>";
```

Quantite : 0
Quantite : 2

```php
public function addQuantite() {

    $this->quantite++;

    return $this;

}
public function deleteQuantite() {

    $this->quantite--;

    if ($this->quantite < 0) {

        $this->quantite = 0;
    }
    return $this;
}

public function getQuantite() {

    return $this->quantite;

}
```

03

Aller plus loin ! (Auto-chargement des classes)

AFLOKKAT
Conseil et Formation

- **Intérêt : éviter un require par classe**
- **Fonction utilisée : spl_autoload_register()**

```php
<?php
/*   require_once "classes/Display.php";
require_once "classes/Produit.php";
require_once "classes/Vetement.php";
require_once "classes/Chaussure.php";   */

spl_autoload_register(function ($classe) {
    require 'classes/' . $classe . '.php';
});

$item = new Vetement("REF0001", "T-shirt col V", 50);
$item->render();
$item = new Vetement("REF0002", "Jeans Navy blue", 60);
$item->render();
```

03 Aller plus loin ! (Traits)

AFLOKKAT
Conseil et Formation

- **Rappel : une classe ne peut étendre qu'une seule classe**
- **Objectif : factoriser un code commun à 2 classes indépendante**

```php
trait Quantite {
    protected $quantite = 0;

    public function addQuantite() {
        $this->quantite++;
        return $this;
    }

    public function deleteQuantite() {
        $this->quantite--;
        $this->quantite = max($this->quantite, 0);
        return $this;
    }
    public function getQuantite() {
     return $this->quantite;
    }
}
```

```php
trait Display {

    public function renderTable()   {
        return "<table border='1'><tr><td>" . $this->
            description . "</td><td>" . $this->quantite .
            "</td></tr></table><br>";
    }
}
```

03 Aller plus loin ! (Traits)

AFLOKKAT
Conseil et Formation

- Rappel : une classe ne peut étendre qu'une seule classe
- Objectif : factoriser un code commun à 2 classes indépendante

```
abstract class Produit {
    use Display;
    use Quantite;
}
```

```
abstract class Telephone {
    use Display;
    use Quantite;
}
```

03 Aller plus loin ! (Clonage)

AFLOKKAT
Conseil et Formation

Pour dupliquer un objet, il faut utiliser le mot clé **clone**

```php
<?php

spl_autoload_register(function ($classe) {
    require 'classes3/' . $classe . '.php';
});

$item = new Vetement("REF0001", "T-shirt col V", 50);

$item2 = clone $item;
$item2->setDescription("t-shirt col rond");

echo $item2->display();

echo $item->display();
```

Le vêtement t-shirt col rond (REF0001) coûte 50 €.
Le vêtement T-shirt col V (REF0001) coûte 50 €.

03

Aller plus loin ! (Clonage)

AFLOKKAT
Conseil et Formation

Pour surcharger le comportement après clonage, on peut implémenter la méthode __clone()

```php
<?php
abstract class Produit {
    use Display;
    use Quantite;

   public function __clone()   {
    $this->description = "Copie de " . $this->description;

    $this->deleteQuantite();
   }
```

03

Aller plus loin ! (Comparaison)

AFLOKKAT
Conseil et Formation

- Comparaison se fait sur l'instance et les propriétés une à une
- Uniquement avec == et === (< et > n'ont aucun sens)

AFLOKKAT
Conseil et Formation

04 Gestion des erreurs

04

Gestion des erreurs

AFLOKKAT
Conseil et Formation

- Différents niveaux d'erreur :
  - certaines ne bloquent pas l'exécution du script (Notice ou Warning)
  - certaines sont bloquantes (Fatal)
- L'affichage des erreurs se paramètre dans le fichier **php.ini** avec la directive **error_reporting()**
- Pour le développement, on préconise d'afficher toutes les erreurs (ou analyser le fichier error.log d'apache)
- Pour un site en production, on les cachera (pour ne pas que les clients les voient)

04

Try, throw ,catch

AFLOKKAT
Conseil et Formation

- Le bloc try... catch permet de contrôler le comportement d'une instruction susceptible de renvoyer une erreur :
  - quand une erreur est déclenchée, on "lance" un objet Exception avec l'instruction throw
  - on récupère cet objet dans bloc catch et on traite l'Exception
- Plusieurs méthodes disponibles :
  - $e->getMessage()
  - $e->getLine()
  - $e->getCode()

04 Try, throw ,catch

AFLOKKAT
Conseil et Formation

```php
<?php

function division($a, $b) {
    return "Le resultat de la division de $a par $b est :
        " . ($a / $b);
}
echo division(10, 0) . "<br>";
```

Warning: Division by zero in C:\wamp64\www\php4\exo4.php on line 4

Call Stack

#	Time	Memory	Function	Location
1	0.0007	402920	{main}( )	...\exo4.php:0
2	0.0008	402920	division( )	...\exo4.php:6

Le resultat de la division de 10 par 0 est : INF

04 Try, throw ,catch

```php
<?php
function division($a, $b) {
    if ($b == 0) {
        throw new Exception("Division impossible par zéro
            !");
    }
    return "Le resultat de la division de $a par $b est :
        " . ($a / $b);
}

try {
    echo division(10, 2) . "<br>";
    echo division(10, 0) . "<br>";
} catch (Exception $e) {
    echo $e->getMessage();
}
```

Le resultat de la division de 10 par 2 est : 5
Division impossible par zéro !

04 finally

```php
<?php
    try {
     fonctionEnErreur();
    } catch (Exception $e) {

     echo $e->getMessage();

    } finally {
     echo "Tu vas y arrivé<br>";
    }
echo "Tu as réussi";//pas affiché
```

Tu vas y arrivé

Fatal error: Uncaught Error: Call to undefined function fonctionEnErreur() in C:\wamp64\www\php4\exo4.php on line *3*

Error: Call to undefined function fonctionEnErreur() in C:\wamp64\www\php4\exo4.php on line *3*

Call Stack

#	Time	Memory	Function	Location
1	0.0008	402944	{main}( )	...\exo4.php:0

04 Exercice

AFLOKKAT
Conseil et Formation

Expliquer le code suivant :

```php
<?php
spl_autoload_register(function ($classe) {
    require 'classes/' . $classe . '.php';
});

$items = [];
$items[] = new Vetement("REF0001", "T-shirt col V", 50);
$items[] = new Vetement("REF0002", "Doudoune noire", 80);
foreach ($items as $index => $item) {
    try {
        if ($index == 0) {
            $item->setPrix(-1);
        } else {
            $item->setQuantite(-1);
        }
    } catch (QuantiteException $e) {
        echo $e->formateError();
    } catch (PrixException $e) {
        echo $e->formateError();
    }
}
```

04

Exercice

- **Implémenter le code des classes pour obtenir ce résultat**

Le prix de T-shirt col V ne peut être négatif !
La quantité de Doudoune noire ne peut être négative !

04

Solution

```php
<?php
class PrixException extends Exception {
    public function formateError()   {
        return "<div style='color:blue'>" . $this->getMessage() . "</div>";
    }
}
```

```php
<?php
class QuantiteException extends Exception {

    public function formateError()   {
        return "<div style='color:red>" . $this->getMessage() . "</div>";
    }
}
```

04 Solution

AFLOKKAT
Conseil et Formation

```php
<?php
abstract class Produit {

    public function setPrix($prix)   {

        if ($prix < 0) {

            throw new PrixException("Le prix de " . $this->getDescription() . "
                ne peut être négatif !");

        }

        $this->prix = $prix;

    }
```

04 Solution

AFLOKKAT
Conseil et Formation

```php
<?php
abstract class Quantite {

    public function setQuantite($quantite)   {

        if ($quantite < 0) {

            throw new QuantiteException("La quantité de " . $this->getDescription(
                ) . " ne peut être négative !");

        }

        $this->quantite = $quantite;
    }
}
```