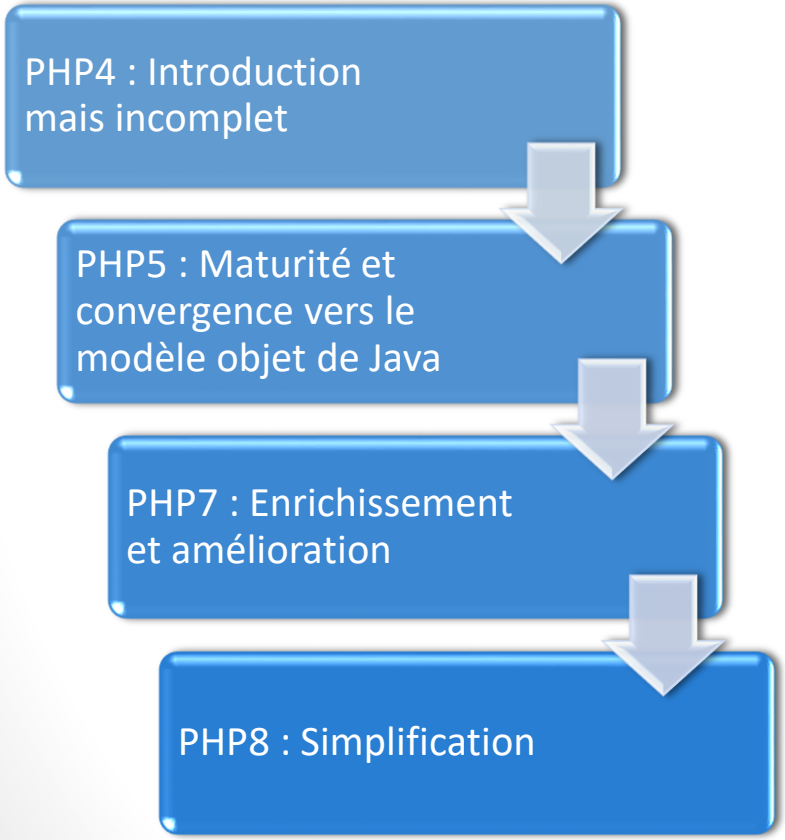
Tour d'horizon de l'objet en PHP

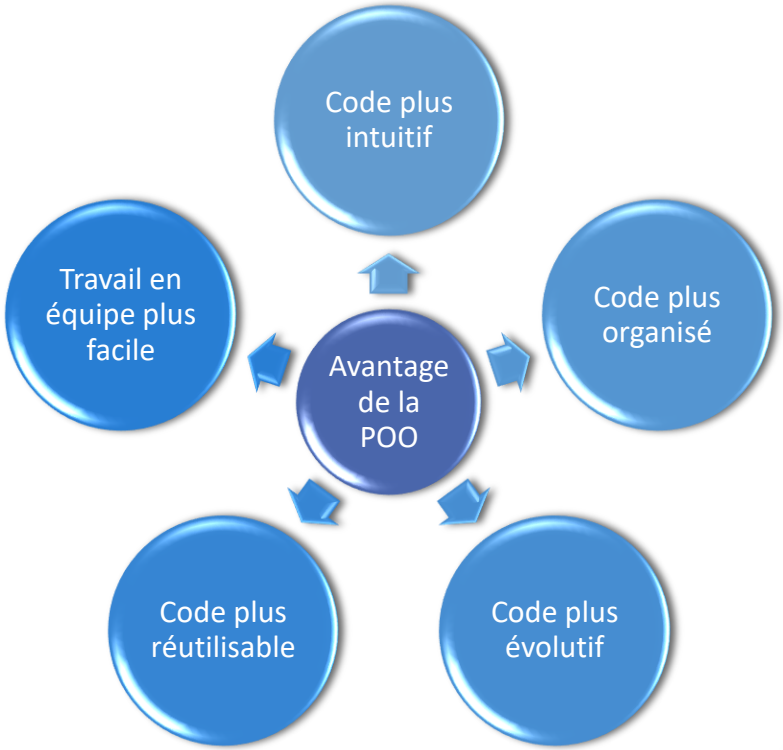
POO : PROGRAMMATION ORIENTÉE OBJET

POO en PHP

Historique



Avantages



POO ▸ Le typage

- Historiquement PHP est un langage non typé ➔ une variable prend le type de la valeur qui lui est affectée
- À partir de PHP7 : possibilité de fonctionner en mode type strict

```
// activer le contrôle du type (0 pour désactiver)
declare(strict_types = 1);
```

  - 1er ligne du code sinon erreur fatale
  - Pas d'impact sur les fichiers inclus ➔ déclarer dans chaque fichier
  - Usage
    - Les paramètres d'une fonction
    - Le type retourné par une fonction
    - Les attributs de classes
- Fonction pour connaitre le type et la valeur d'une ou plusieurs variables :

```
var_dump($var1,$var2, …);
```

10 grands types de données

- string : chaînes de caractères
- int : nombres entiers
- float : nombre décimal
- bool : true et false
- array : tableaux
- NULL : variable vide
- resource : référence à une ressource externe à PHP (connexion à une BD)
- iterable : tableau ou instance de Transversable //PHP 7.1.0
- object : objet //PHP 7.2.0
- mixed : n'importe quelle valeur //PHP 8.0.0

POO ▸ Déclaration de classe

▸ Sans typage

```php
<?php
class MyClass{
    //Attributes
    private $myAttribute;
    private $countUpdate;
    //Methods
    function getMyAttribute(){
        return $this->myAttribute;
    }
    function getNbUpdate(){
        return $this->countUpdate;
    }
    function setMyAttribute($myAttribute){
        $this->myAttribute=$myAttribute;
        $this->countUpdate++;
    }
}
?>
```

Bonnes pratiques :

- Une classe par fichier
- Même nom de classe que nom de fichier
  - MyClasse.php
  - MyClasse.class.php
- Respecter une même convention de nommage
  - camelCase : attributs, méthodes et instances
  - PascalCase : classes + interfaces
  - Snak_case (underscore) : constantes NOM_DE_CONSTANTE

POO ▸ Déclaration de classe

▸ Avec typage

```php
<?php
declare(strict_types = 1);
class MyClassType{
 private String $lastAddTime;// String attribute
 private MyClass $myClass; //Custom class attribute
 private $myClassArray=array(); // Attribute without type. It's array
 // Function return string
 function getLastAddTime():string{
    return $this->lastAddTime;
 }
 // Function return int value
 function getNbInstance():int{
     return sizeof($this->myClassArray);
 }
```

POO ▸ Déclaration de classe

▸ Avec typage

```
15. // Function receive MyClass instance
16. function add(MyClass $myClass){
17.     $this->lastAddTime=Date("l F d, Y h:m:s");
18.     $this->myClass=$myClass;
19.     $this->myClassArray[]=$myClass;
20. }
21. // Function return MyClass instance
22. function getLastAddInstance(): MyClass{
23.     return $this->myClass;
24. }
25. //Function return unspecified type
26. function buildContent(){
27.     $result="";
28.     foreach ($this->myClassArray as $instance) {
29.         $result.=$instance->getMyAttribut() ." changed ".$instance->getNbUpdate()."<br/>\n";
30.     }
31.     return $result;
32. }
33.}
34.?>
```

POO ▸ Instanciation

```php
<?php
include ("MyClass.class.php");//import class file
$myInstance= new MyClass();//create instance
$myInstance->setMyAttribut("First update");//invoke method
echo $myInstance->getMyAttribut() ." changed ".$myInstance->getNbUpdate()."<br/>\n";
$myInstance->setMyAttribut("Second update");
echo $myInstance->getMyAttribut() ." changed ".$myInstance->getNbUpdate()."<br/>\n";
?>
```

- Par défaut, il y a un constructeur sans paramètres
- -> opérateur d'accès à un attribut ou une méthode d'instance public
- Quel est le résultat du code ?

POO ▸ Encapsulation

3 niveaux d'encapsulation pour les attributs, méthodes et constantes

private	• accessible à l'intérieur de la classe
protected	• accessible à l'intérieur de la classe et à partir des classes filles
public	• accessible de partout • par défaut quand rien n'est explicité

POO ▸ Héritage

- `extends`
  - Déclarer une classe qui hérite d'une autre classe
    - ```
      class Book extends Document{
      }
      ```
  - Possibilité de redéfinir les méthodes héritées
- `abstract`
  - Appliqué sur une classe ➔ interdit son instanciation ➔ oblige l'héritage
    - ```
      abstract class Document{
      }
      ```
  - Appliqué sur une méthode ➔ classe obligatoirement abstraite, méthode sans corps ➔ surcharge obligatoire
    - ```
      abstract public function myFunction();
      ```
      ```

POO ▸ Héritage

- `final`
  - Interdit l'héritage d'une classe
    - `final class Paper extends Document { }`
      - ➔ Il n'est plus possible d'hériter de Paper
  - Interdit la surcharge de méthode
    - `final public function myFunction(){ }`
- `parent::`
  - Accéder à un attribut ou une méthode de la classe parente
- `self::`
  - Accéder à un attribut ou une méthode de la classe courante

PHP ne supporte pas l'héritage multiple

POO ▸ Attributs et méthodes statiques

- `const`
  - Déclarer une constante de classe ⇔ un attribut statique de classe ayant une valeur inchangeable
  - Déclaration `const PI=3.14;`
  - Accès: `MyClasse::PI;`
- `static`
  - Déclarer un attribut ou une méthode de classe
    - `private static $value;`
    - `public static function myFunction(){}`
  - Accès:
    - À l'extérieur de la classe
      - `MyClass::$value;`
      - `MyClass::myFunction();`
    - À l'intérieur de la classe
      - Dans une instance de classe `self::`$value;

POO ▸ Interface

- C'est une sorte de contrat que peut respecter une classe
- Permet de déclarer
  - Des constantes
  - Des signatures des méthodes qu'une classe doit implémenter pour respecter l'interface
- Une classe qui n'implémente pas toutes les méthodes d'une interface doit être déclarée abstraite
- Une classe peut implémenter plusieurs interfaces séparées par une virgule
- Une interface peut hériter d'autres interfaces

POO ▸ Interface

```php
<?php
    interface Product{
    public function getPrice():float;
    public function getweight():float;
  }
?>
```

- implements
  - Permet à une classe d'indiquer qu'elle implémente une interface
  - Exemple :

```
class Book extends Document implements
Product{
}
```

POO ▸ Trait

Définition ▸ Un concept semblable à une classe permettant de regrouper des méthodes

- Composition horizontale de comportements
- Mécanisme de réutilisation de code (méthodes) dans des classes indépendantes

Usage

- Réutiliser des méthodes de classe sans passer par l'héritage
- Pallier à l'interdiction d'héritage multiple
- Éviter de réécrire plusieurs fois la même méthode à différents endroits

Un trait peut avoir plusieurs méthodes et propriétés

POO ▸ Trait

Règles

- Il est impossible d'instancier un trait.
- Si un trait défini une propriété alors une classe qui utilise le trait ne peut pas définir une propriété ayant le même nom (sauf si la propriété a le même niveau d'encapsulation et la même valeur initiale).
- Une méthode héritée d'une classe mère est écrasée par une méthode ayant la même signature issue d'un trait.
- Une méthode d'un trait est écrasée par une méthode ayant la même de la classe courante.

POO ▸ Trait

```php
<?php
trait ConvertDate{
    public function normalizeDate( $date, $format) {
        $date=DateTime::createFromFormat($format, $date);
        return $date->format('Y-m-d_h:m');
    }
}
?>
```

- use
  - Permet à une classe d'indiquer qu'elle utilise un trait
  - Exemple :

```php
class MyClassUseTrait {
    use ConvertDate;
}
```

- Instanciation et invocation

```php
$myObject =new MyClassUseTrait();
echo $myObject->normalizeDate("15/12/2023", "d/m/Y");
```

- Plus d'exemples dans la doc officielle
  https://www.php.net/manual/fr/language.oop5.traits.php

POO ▸ Les namespaces

- namespaces ⇔ espaces de noms
- un moyen pour regrouper des classes, interfaces, fonctions ou constantes dans un même espace virtuel (répertoire abstrait regroupant tous les éléments appartenant au même namespace)
  - Éviter les collisions
  - Éviter les conflits de noms
- Sont utiles dans les gros projets et les projets qui utilisent des librairies externes
- Définir un namespace au tout début de la page avant d'écrire du contenu pour le navigateur ou toute instruction php (sauf declare):
  - ```
    <?php
        namespace mySpace;
        class MyClass {}

    ?>
    ```
- Utiliser la classe dans un autre namespace:
  - ```
    use MySpace\MyClass;
    ```

POO ▸ Méthodes magiques

- Méthodes prédéfinies invoquées automatiquement suite à une opération spéciale lors de l'exécution du programme
- Elles sont identifiables par leur préfixe: __ (double soulignement)
- Il est possible de les surcharger pour changer leur comportement par défaut en respectant
  - La signature de chaque méthode
    - Sauf pour le constructeur
    - Voir la doc https://www.php.net/manual/en/language.oop5.magic.php
  - Leur niveau d'encapsulation public
    - Sauf pour le constructeur, le destructeur et la méthode de clonage

POO ▸ Méthodes magiques et moment d'invocation

__construct()	instanciation d'un objet ➔ constructeur de classe
__destruct()	destruction d'un objet ➔ quand il n'y a plus de références vers l'objet
__call()	appel de méthodes inaccessibles dans un contexte objet
__callStatic()	appel de méthodes inaccessibles dans un contexte statique
__get()	lecture des données depuis des propriétés inaccessibles (protégées ou privées) ou inexistantes
__set()	écriture de données vers des propriétés inaccessibles (protégées ou privées) ou inexistantes
__isset()	quand isset() ou empty() sont appelées sur des propriétés inaccessibles (protégées ou privées) ou inexistantes
__unset()	quand unset() est appelée sur des propriétés inaccessibles (protégées ou privées) ou inexistantes

POO ▸ Méthodes magiques et moment d'invocation

__sleep()	exécutée avant toute linéarisation (appel de la méthode serialize)
__wakeup()	exécutée après toute dé-linéarisation (appel de la méthode unserialize)
__serialize()	méthode exécutée avant toute linéarisation (inhibe __sleep si les deux sont présentes)
__unserialize()	méthode exécutée après toute dé-linérisation (inhibe __wakeup si les deux sont présentes)
__toString()	appelée lorsqu'un script tente de traiter un objet comme une chaîne de caractères
__invoke()	appelée lorsqu'un script tente d'appeler un objet comme une fonction
__set_state()	appelée pour les classes exportées par la fonction var_export()
__**clone**()	Appelée lorsqu'un objet est clonée avec la méthode clone.
__debugInfo()	quand var_dump() est appelée sur un objet pour récupérer ses propriétés à afficher

POO ▸ Méthodes magiques ▸ Exemple du constructeur

- Invoqué lors de l'instanciation de l'objet pour
  - Initialiser les attributs de la classe
  - Allouer des ressources pour l'instance
- Un seul constructeur par classe est autorisé en PHP.
- Exemples
  - Avant PHP8.0
    - Le constructeur peut avoir le même nom que le nom de la classe (obsolète)
    - Le constructeur recommandé d'écrire

```
1.  <?php
2.  class MyClass{
3.      //Attributs
4.      private $myAttribut;
5.      private $countUpdate;
6.      //Constructor
7.      function __construct(string $attribut){
8.          $this->myAttribut=$attribut;
9.          $this->countUpdate=0;
10.     }
11. ?>
```

POO ▸ Méthodes magiques ▸ Exemple du constructeur

- Instanciation:

```
$myInstance= new MyClass("First update");
```

- Depuis PHP8.0, promotion du constructeur
  Les paramètres du constructeur peuvent être promus en attributs de classe → moins de code à écrire.
  - Le constructeur

```
1. <?php
2. class MyClass{
3.       //Constructor
4.       function __construct(private string $attribut, private int $countUpdate=1){
5.          echo "instance created<br/>";
6.       }
7. }
8. ?>
```

POO ▸ Chargement automatique de classes

- Un projet bien organisé
  - Comporte plusieurs classes
  - Code chaque classe dans un fichier à part ayant le même nom que la classe et l'extension `.class.php`

➔ Il faut inclure plusieurs classes à différents endroits avec l'instruction include:

```
include("Myclass.class.php");
```

- Utiliser une fonction d'inclusion :

```
<?php
    function load($classe){
        include($classe.".class.php");
    }
    load("MyClass");
?>
```

➔Il faut appeler plusieurs fois la fonction à différents endroits

POO ▸ Chargement automatique de classes

Comment optimiser et alléger la charge de travail?

- Solution : demander à PHP de charger dynamiquement des classes à la demande
- Depuis PHP 5, il existe la fonction `spl_autoload_register("loadFunctionName");`
- Exemple :

```php
<?php
    function load($class){
        include("slides/".$class.".class.php");
    }
    spl_autoload_register("load");
    $instance=new MyClass();//auto load class file
?>
```

Exercices de l'Activité A3 sur l'espace Moodle

PAUSE ACTIVITÉ PRATIQUE