Contexte

PHP est à la base un langage de script conçu pour le Web.

Au fil du temps, PHP s’est enrichi d’outils complets pour développer selon l’approche objet, le paradigme POO, plutôt que selon l’approche procédurale standard du socle de PHP.

Les avantages de la POO sont :

* Meilleure réutilisabilité du code : les types d’objet peuvent servir de base pour de nouvelles classe
* Un code plus proche des modèles de conception comme UML, permettant de mieux analyser, décrire et communiquer les besoins fonctionnels et métier
* Un code modulaire facilitant l’isolation, et donc la clarté, le travail d’équipe, la sécurité des applications : chaque type ne communique avec d’autres types que via des interfaces

Objets, classes et instances

## Objet

**Wikipedia** : « un objet est un conteneur symbolique et autonome qui contient des informations et des mécanismes concernant un sujet, manipulés dans un programme. Le sujet est souvent quelque chose de tangible appartenant au monde réel »

## Attributs et méthodes

Chaque objet dispose de ses propres ***attributs*** et ***méthodes***. Les attributs sont les caractéristiques propres d’un objet.

Les méthodes sont les actions qu’on peut appliquer à un objet.

## Classe

Une classe est un modèle de données. C’est en quelque sorte l’équivalent d’un patron en couture qui permet de fabriquer un ensemble de vêtements similaires. Tous les objets d’une même classe utilisent et sont dotés des mêmes attributs et méthodes.

## Instance

Une ***instance*** est une représentation particulière d’une classe. Un ***objet*** est donc une instance de classe.

## Syntaxe et déclaration de classe

## Déclaration de classe

Le mot clef class permet de définir une nouvelle classe. Il est suivi du nom de la classe, puis de la définition de la classe, à savoir ses attributs et méthodes, entre accolades { }

## Définition d’attributs

Chaque attribut peut être défini de la manière suivante :

***portée***  $nomAttribut = valeurInitiale ;

La portée peut être :

* public : l’attribut est accessible depuis toute l’application. C’est la portée par défaut, mais il est très recommandé d’être explicite
* private : l’attribut n’est utilisable qu’à ***l’intérieur*** de la classe
* protected : l’attribut est utilisable à ***l’intérieur*** de la classe et de toute classe ***dérivée***

L’initialisation est optionnelle.

## Définition de méthodes

Chaque méthode est définie de la manière suivante :

***portée***  function ***nomMethode***( [$param1,$param2…])

{

// Instruction de la méthode ***nomMethode***

}

La définition de la portée est la même que pour les attributs. Les paramètres à la méthode sont optionnels (indication de leur caractère optionnel avec les [] )

## Instanciation d’un objet

Un objet est instancié avec le mot clef ***new*** .

Exemple :

$voiture = new Voiture() ;

Dans l’exemple, $voiture est un objet instance de la classe Voiture. Les () après le nom de la classe sont optionnelles mais recommandées systématiquement, même si il n’y a aucune paramètre dans le constructeur.

## Utilisation d’un attribut

Les attributs s’utilisent comme des variables. On y accède grâce à l’opérateur -> (moins inférieur).

Exemple :

Soit une classe Compte, avec un attribut $solde , initialisé à 0.

$compteProfesseur = new Compte() ;

// on crée un objet $compteProfesseur instance de la classe Compte

$ compteProfesseur->solde = 6000 ;

// l’attribut solde de l’objet $compteProfesseur prend la valeur 6000

**Remarque** : lorsque on accède à l’attribut d’un objet via l’opérateur -> , **on ne met pas de $**. L’objet est une variable mais le nom de l’attribut est fixe.

## Utilisation d’une méthode

L’appel d’une méthode est similaire à celui d’une fonction, et on utilise comme pour les attributs d’objet l’opérateur ->

Exemple : récupérez et exécutez le code du script EXEMPLE1\_eleve.php.

$eleveSerieux->travaille() ;

$eleveSerieux est une instance de la classe Eleve, pour laquelle on appelle la méthode travaille().

# Constructeur et destructeur

## Constructeurs

Un constructeur est une méthode particulière qui a les 3 propriétés suivantes :

* Son nom est toujours \_\_construct() (2 underscores devant construct)
* Elle est appelée automatiquement et immédiatement à chaque fois qu’un objet de la classe est instancié
* Elle ne peut comporter d’instruction return

Syntaxe

Class NomDeClasse {

public $attributClasse ;

function \_\_construct()

{

// Code du constructeur

}

}

Un constructeur peut être utilisé pour définir une connexion à une base de données, établir des cookies ou initialiser un certain nombre de valeurs.

**Question** : comment est nommée la méthode de construction pour les classes Java ?

## Chargement des classes

Une classe définie dans un fichier php séparé peut être appelée via :

* la fonction require (cf cours sur les fonctions) ou bien via
* une fonction spéciale **spl\_autoload\_register** ainsi définie à ajouter dans le programme appelant.

## Destructeur

Un destructeur est une méthode particulière nommée \_\_destruct(), automatiquement appelée lorsque l’objet est détruit.

# Héritage

Lorsqu’on catégorise les objets, on souhaite créer des sous catégories, pour caractériser des objets qui partagent certains attributs et méthodes.

En PHP, l’héritage est simple, c'est-à-dire qu’une classe ne peut hériter que d’une seule classe.

Pour compléter l’héritage, PHP propose :

* **Des interfaces** : il s’agit de définir une API (Application Programming Interface) utilisée par un composant. Dans une interface sont déclarées des fonctions que doivent implémenter les classes qui implémentent cette interface
* **Des traits** : les traits permettent une réutilisation du code sans passer par la notion de hiérarchie de code.

## Définition d’une classe héritée – fille

L’héritage est défini par le mot clef **extends** , après le nom de la classe.

Exemple d’héritage :

Si on définit une classe VehiculeAMoteur, avec pour méthodes (fonctions) freine() et avance(), on peut ensuite définir une classe Voiture qui étend la classe VehiculeAMoteur et qui dispose en plus de nouveaux attributs, par exemple marque, modele, annee et de nouvelles méthodes klaxonne() et demarre().

### Redéfinition

Si un attribut ou une méthode est redéfinie dans la classe fille, c’est cet attribut ou cette méthode qui sera utilisée.

Si on souhaite tout de même explicitement accéder à la méthode de la classe parente, on y accède via l’opérateur :: sous la forme parent::methode()

Exemple : TP sur Héritage avec accès aux méthodes parentes.

# Interfaces et Traits

## Les interfaces

Les interfaces sont utilisées afin de déclarer qu’un objet implémente un certain nombre de méthodes : c’est une notion de contrat. Une classe qui implémente une interface promet d’implémenter les méthodes (fonctions) déclarées dans l’interface. Si elle ne le fait pas d’ailleurs, PHP génère une erreur fatale.

Une classe peut implémenter autant d’interfaces qu’elle le souhaite.

Les méthodes (fonctions) d’une ***interface*** sont juste une déclaration d’intention. Le code réel est écrit dans les différentes classes qui implémentent cette interface.

## Les traits

Les traits permettent une réutilisation horizontale (c'est-à-dire sans utiliser l’héritage, qui est vertical) du code.

Ils peuvent contenir aussi bien des attributs que des méthodes (fonctions). Le code ***effectif*** de ces fonctions est écrit dans le trait (contrairement aux interfaces qui délèguent l’implémentation aux classes).

Pour créer un trait :

Trait NomTrait

{

// Attributs et fonctions implémentées dans le trait

}

Pour indiquer qu’une classe utilise un trait, on utilise la syntaxe : use NomTrait ; où NomTrait est le nom du trait déclaré.

## Traits ou Interfaces ?

Il est considéré comme une meilleure pratique de design d’utiliser ***à la fois*** les traits et les interfaces :

* L’interface assure le respect du contrat : une classe implémentant une interface doit nécessairement remplir le contrat, c'est-à-dire implémenter les méthodes (fonctions) définies dans l’interface
* Le trait permet la réutilisation et l’homogénéité du code, sans avoir à le recopier partout pour des classes qui ne sont pas sur une même arborescence

# Débat théorique et pratique : Composition ou Héritage

Le choix de modélisation et d’implémentation entre une Composition et un Héritage est complexe et peut donner lieu à débats, essais, refactoring.

Une règle de base pour aider à choisir entre Composition et Héritage est la suivante :

Soit 2 objets ObjetA et ObjetB.

Si ObjetB **est une sorte** de ObjetA, alors on préférera une modélisation de type Héritage : B extends A

Si ObjetB **a, contient tout ou partie** de ObjetA, alors on préférera B contient A (concrètement on définit parmi les attributs de B un objet instance de A)

Dans le monde anglo saxon, c’est le débat IS or HAS.

Pour être à l’aise sur ces sujets, rien ne remplace la pratique et l’expérience.