



Programmation orientée objet en PHP

Accueil ► Mes cours ► Développement logiciel ► POO PHP ► La programmation orientée objet : premiers pas ► **Introduction**

Introduction

La programmation orientée objet est ce qu'on appelle un paradigme de programmation, c'est-à-dire une façon de concevoir des programmes. Jusqu'ici nous avons écrit des programmes selon un style de programmation très impératif : nos programmes sont constitués d'une suite d'instructions qui seront réalisées séquentiellement (les unes après les autres) par l'ordinateur. Nous allons à présent créer des objets qui posséderont certaines caractéristiques et compétences, et nous demanderons à notre programme de mettre ces objets en interaction.

Il existe beaucoup de langages qui permettent un style de programmation orienté objet, parmi les plus connus nous pouvons citer C++, C#, Java.

Fin

NAVIGATION



Accueil

■ Ma page

Pages du site

Mon profil

Cours actuel

POO PHP

Participants

Généralités

La programmation orientée objet : premiers pas

■ **Introduction**

■ Classes et objets

■ T.P. première classe

■ Les classes en PHP

■ T.P. Personne

■ Le constructeur et this

■ T.P. Personne v2

■ Accesseurs et mutateurs

■ Les méthodes magiques

■ T.P. Personne v3

■ Les constantes de classe

■ Le modificateur static

■ T.P. Personne v4

■ Une Personne et une Adresse : la relation has-a

■ Histoire de références

■ T.P. formation

■ T.P. formation v2

■ Une Personne et plusieurs Adresse(s)

■ T.P. formation v3

■ Bricolage et dépendance

L'héritage

Les interfaces

Le typage

Les namespaces

Les exceptions

Les bases de données avec PDO

Les tests avec PHPUnit

Petite application version 2
Petite application version 3

[Mes cours](#)

ADMINISTRATION



[Administration du cours](#)

[Réglages de mon profil](#)

Connecté sous le nom « [Arnaud Lemais](#) » ([Déconnexion](#))
[POO PHP](#)



Programmation orientée objet en PHP

[Accueil](#) ► [Mes cours](#) ► [Développement logiciel](#) ► [POO PHP](#) ► La programmation orientée objet : premiers pas ► [Classes et objets](#)

Classes et objets

Pour aborder un des concepts fondamentaux de la programmation orientée objet (POO), celui de classe, nous allons partir d'un exemple concret : celui de l'être humain.

Comment pourrions-nous décrire un être humain ? Nous pourrions par exemple dire qu'un humain possède nécessairement un age, une taille, un poids... N'importe quel être humain pourra donc être décrit selon un ensemble choisi de caractéristiques : ici l'age, la taille, le poids. Chaque être humain aura donc une valeur précise pour chaque caractéristique, ces valeurs ne seront pas forcément les mêmes d'un humain à l'autre. Par exemple un humain pourra peser 100 kg alors qu'un autre n'en pèsera que 50.

Nous pourrions aussi décrire un être humain en énumérant les compétences communes à tous les humains et la façon dont elles se réalisent ; c'est-à-dire ce que tous les humains sont capables de faire et comment ils le font. Par exemple, tous les humains savent marcher en mettant un pied devant l'autre, courir en passant d'un pied sur l'autre, parler en émettant des sons, etc... Toutes ces compétences sont communes au genre humain, tout comme les caractéristiques énumérées précédemment.

Nous pouvons représenter notre description de l'humain dans un tableau avec une case pour le nom de la classe, une pour les caractéristiques ou attributs, une autre pour les compétences ou méthodes :

Humain
age
taille
poids
marcher en mettant un pied devant l'autre
courir en passant d'un pied sur l'autre
parler en émettant des sons

La classe Humain est donc en quelque sorte un modèle, une trame ou encore une description qui énumère les attributs et méthodes que possèdent n'importe quel être humain.

Maintenant que nous avons notre "modèle" d'humain, c'est-à-dire notre classe Humain, nous allons l'utiliser pour décrire des humains concrets, par exemple Jack Sparrow et Bruce Wayne. Entre ces deux humains les compétences seront les mêmes, mais les valeurs de leurs attributs seront différentes :

sparrow : Humain
age = 42
taille = 1,84
poids = 75
marcher en mettant un pied devant l'autre
courir en passant d'un pied sur l'autre
parler en émettant des sons
wayne : Humain
age = 41
taille = 1,95
poids = 100

marcher en mettant un pied devant l'autre
courir en passant d'un pied sur l'autre
parler en émettant des sons

Nos deux humains sparrow et wayne sont des objets de type Humain, ou encore des instances de la classe Humain.

La classe représente le modèle, l'instance de classe (ou l'objet) représente la réalisation concrète de la classe.

Lorsqu'on crée un objet à partir d'une classe, on dit que l'on instancie la classe. L'action de créer un objet à partir d'une classe s'appelle l'instanciation.

Un objet est donc une instance de classe, ses propriétés sont des attributs d'instance et ses compétences des méthodes d'instance.

Fin

NAVIGATION



[Accueil](#)

■ [Ma page](#)

[Pages du site](#)

[Mon profil](#)

[Cours actuel](#)

[POO PHP](#)

[Participants](#)

[Généralités](#)

[La programmation orientée objet : premiers pas](#)

[Introduction](#)

[Classes et objets](#)

[T.P. première classe](#)

[Les classes en PHP](#)

[T.P. Personne](#)

[Le constructeur et this](#)

[T.P. Personne v2](#)

[Accesseurs et mutateurs](#)

[Les méthodes magiques](#)

[T.P. Personne v3](#)

[Les constantes de classe](#)

[Le modificateur static](#)

[T.P. Personne v4](#)

[Une Personne et une Adresse : la relation has-a](#)

[Histoire de références](#)

[T.P. formation](#)

[T.P. formation v2](#)

[Une Personne et plusieurs Adresse\(s\)](#)

[T.P. formation v3](#)

[Bricolage et dépendance](#)

[L'héritage](#)

[Les interfaces](#)

[Le typage](#)

[Les namespaces](#)

[Les exceptions](#)

[Les bases de données avec PDO](#)

[Les tests avec PHPUnit](#)

[Petite application version 2](#)

[Petite application version 3](#)

[Mes cours](#)

ADMINISTRATION



[Administration du cours](#)

Réglages de mon profil

Connecté sous le nom « [Arnaud Lemaïs](#) » ([Déconnexion](#))
POO PHP



Programmation orientée objet en PHP

Accueil ► Mes cours ► Développement logiciel ► POO PHP ► La programmation orientée objet : premiers pas ► T.P. première classe

T.P. première classe

Proposez un exemple de classe et, à partir de cette classe, instancier un objet. Vous pourrez décrire votre classe et vos instances sous forme de tableaux comme vu dans la leçon précédente.

Le devoir sera remis sous forme d'un fichier pdf dont le nom sera de la forme "premiere_classe_nom_prenom.pdf".

État du travail remis

Numéro de tentative	Ceci est la tentative 1.
Statut des travaux remis	Aucune tentative
Statut de l'évaluation	Pas évalué
Dernière modification	vendredi 27 mars 2015, 15:04

[Ajouter un travail](#)

[Modifier votre travail remis](#)

NAVIGATION



[Accueil](#)

■ [Ma page](#)

Pages du site

Mon profil

Cours actuel

[POO PHP](#)

Participants

Généralités

La programmation orientée objet : premiers pas

[Introduction](#)

[Classes et objets](#)

[T.P. première classe](#)

[Les classes en PHP](#)

[T.P. Personne](#)

[Le constructeur et this](#)

[T.P. Personne v2](#)

[Accesseurs et mutateurs](#)

[Les méthodes magiques](#)

[T.P. Personne v3](#)

[Les constantes de classe](#)

[Le modificateur static](#)

[T.P. Personne v4](#)

[Une Personne et une Adresse : la relation has-a](#)

[Histoire de références](#)

-  T.P. formation
-  T.P. formation v2
-  Une Personne et plusieurs Adresse(s)
-  T.P. formation v3
-  Bricolage et dépendance

L'héritage
Les interfaces
Le typage
Les namespaces
Les exceptions
Les bases de données avec PDO
Les tests avec PHPUnit
Petite application version 2
Petitie application version 3

Mes cours

ADMINISTRATION



Administration du cours

Réglages de mon profil

Connecté sous le nom « Arnaud Lemaïs » (Déconnexion)
POO PHP



Programmation orientée objet en PHP

Accueil ► Mes cours ► Développement logiciel ► POO PHP ► La programmation orientée objet : premiers pas ► Les classes en PHP

Les classes en PHP

Même s'il est possible en PHP de créer plusieurs classes dans un seul fichier source, il est préférable de ne créer qu'une seule classe par fichier source. Il est aussi préférable de donner au fichier le même nom qu'à la classe : une classe Humain dans un fichier "Humain.php".

Le nom de la classe commencera toujours par une majuscule et sera écrit en camelCase (les mots seront collés et chaque première lettre de mot sera écrite en majuscule).

Pour créer une classe nous utiliserons le mot clé "class" suivi du nom de la classe écrit en camelCase. Ensuite, entre accolades, nous créerons les attributs d'instance sous forme de variables puis les méthodes d'instance sous forme de fonctions. Les noms des attributs d'instance et méthodes d'instance seront écrits en camelCase mais la première lettre sera toujours en minuscule. Le mot clé public devant les attributs et méthodes a une signification particulière que nous aborderons plus tard.

```
1 <?php
2
3 // Humain.php
4 class Humain {
5
6     public $age;
7     public $taille;
8     public $poids;
9
10    public function marcher() {
11        echo 'je marche';
12    }
13
14    public function courir() {
15        echo 'je cours';
16    }
17
18    public function parler() {
19        echo 'je parle';
20    }
21
22 }
```

Pour créer une instance de classe Humain, nous allons utiliser l'opérateur "new" et une "méthode" spéciale qui s'appelle un constructeur. Pour appeler ce constructeur nous utiliserons le nom de la classe suivie de parenthèses.

```
1 <?php
2
3 // Humain.php
4 class Humain {
5
6     public $age;
7     public $taille;
8     public $poids;
9
10    public function marcher() {
11        echo 'je marche';
12    }
13
14    public function courir() {
15        echo 'je cours';
16    }
17
18    public function parler() {
19        echo 'je parle';
20    }
21
22 }
```

```

16     }
17
18     public function parler() {
19         echo 'je parle';
20     }
21
22 }
23
24 $sparrow = new Humain();

```

La variable \$sparrow est une référence à un objet de type Humain.

Pour accéder en lecture et écriture aux attributs, nous utiliserons l'opérateur "->" à partir de la référence. Pour l'instant, les attributs de sparrow n'ont pas de valeurs (ils sont null). Nous allons assigner une valeur aux attributs de sparrow puis afficher ces valeurs :

```

1 <?php
2
3 // Humain.php
4 class Humain {
5
6     public $age;
7     public $taille;
8     public $poids;
9
10    public function marcher() {
11        echo 'je marche';
12    }
13
14    public function courir() {
15        echo 'je cours';
16    }
17
18    public function parler() {
19        echo 'je parle';
20    }
21
22 }
23
24 $sparrow = new Humain();
25 $sparrow->age = 42;
26 $sparrow->taille = 1.84;
27 $sparrow->poids = 75;
28 echo "age = $sparrow->age" . PHP_EOL;
29 echo "taille = $sparrow->taille" . PHP_EOL;
30 echo "poids = $sparrow->poids" . PHP_EOL;

```

Nous pouvons aussi appeler les méthodes en utilisant l'opérateur "->" et la référence.

```

1 <?php
2
3 // Humain.php
4 class Humain {
5
6     public $age;
7     public $taille;
8     public $poids;
9
10    public function marcher() {
11        echo 'je marche';
12    }
13
14    public function courir() {
15        echo 'je cours';
16    }

```

```

17
18     public function parler() {
19         echo 'je parle';
20     }
21
22 }
23
24 $sparrow = new Humain();
25 $sparrow->age = 42;
26 $sparrow->taille = 1.84;
27 $sparrow->poids = 75;
28 echo "age = $sparrow->age" . PHP_EOL;
29 echo "taille = $sparrow->taille" . PHP_EOL;
30 echo "poids = $sparrow->poids" . PHP_EOL;
31 $sparrow->courir();
32 echo PHP_EOL;
33 $sparrow->marcher();
34 echo PHP_EOL;
35 $sparrow->parler();

```

Lors de la déclaration des attributs, il est possible de leur assigner tout de suite une valeur, si cette valeur est de type integer, double, string, boolean.

```

1 <?php
2
3 // Humain.php
4 class Humain {
5
6     public $age = 42;
7     public $taille = 1.84;
8     public $poids = 75;
9
10    public function marcher() {
11        echo 'je marche';
12    }
13
14    public function courir() {
15        echo 'je cours';
16    }
17
18    public function parler() {
19        echo 'je parle';
20    }
21
22 }
23
24 $sparrow = new Humain();
25 echo "age = $sparrow->age" . PHP_EOL;
26 echo "taille = $sparrow->taille" . PHP_EOL;

```

Fin

NAVIGATION



[Accueil](#)

■ [Ma page](#)

[Pages du site](#)

[Mon profil](#)

[Cours actuel](#)

[POO PHP](#)

[Participants](#)

[Généralités](#)

[La programmation orientée objet : premiers pas](#)

- [Introduction](#)
- [Classes et objets](#)
- [T.P. première classe](#)
- [**Les classes en PHP**](#)
- [T.P. Personne](#)
- [Le constructeur et this](#)
- [T.P. Personne v2](#)
- [Accesseurs et mutateurs](#)
- [Les méthodes magiques](#)
- [T.P. Personne v3](#)
- [Les constantes de classe](#)
- [Le modificateur static](#)
- [T.P. Personne v4](#)
- [Une Personne et une Adresse : la relation has-a](#)
- [Histoire de références](#)
- [T.P. formation](#)
- [T.P. formation v2](#)
- [Une Personne et plusieurs Adresse\(s\)](#)
- [T.P. formation v3](#)
- [Bricolage et dépendance](#)
- [L'héritage](#)
- [Les interfaces](#)
- [Le typage](#)
- [Les namespaces](#)
- [Les exceptions](#)
- [Les bases de données avec PDO](#)
- [Les tests avec PHPUnit](#)
- [Petite application version 2](#)
- [Petitie application version 3](#)

[Mes cours](#)



ADMINISTRATION

[Administration du cours](#)

[Réglages de mon profil](#)

Connecté sous le nom « [Arnaud Lemais](#) » ([Déconnexion](#))
POO PHP



Programmation orientée objet en PHP

Accueil ► Mes cours ► Développement logiciel ► POO PHP ► La programmation orientée objet : premiers pas ► T.P. Personne

T.P. Personne

Créez une classe Personne dans un fichier Personne.php qui possédera en attribut d'instance un nom et un prénom, et une méthode d'instance sePresenter() qui affichera "Bonjour". Créez une instance de Personne, affectez une valeur à ses attributs puis affichez ces valeurs, enfin appelez la méthode sePresenter().

Le fichier portera le nom Personne.php. Vous remettrez une archive zip contenant le fichier. Le nom de l'archive sera de la forme "personne_nom_prenom.zip".

État du travail remis

Numéro de tentative	Ceci est la tentative 1.
Statut des travaux remis	Aucune tentative
Statut de l'évaluation	Pas évalué
Dernière modification	vendredi 27 mars 2015, 15:05

[Ajouter un travail](#)

[Modifier votre travail remis](#)

NAVIGATION



[Accueil](#)

■ [Ma page](#)

Pages du site

Mon profil

Cours actuel

POO PHP

Participants

Généralités

La programmation orientée objet : premiers pas

[Introduction](#)

[Classes et objets](#)

[T.P. première classe](#)

[Les classes en PHP](#)

[T.P. Personne](#)

[Le constructeur et this](#)

[T.P. Personne v2](#)

[Accesseurs et mutateurs](#)

[Les méthodes magiques](#)

[T.P. Personne v3](#)

[Les constantes de classe](#)

[Le modificateur static](#)

[T.P. Personne v4](#)

[Une Personne et une Adresse : la relation has-a](#)

-  [Histoire de références](#)
-  [T.P. formation](#)
-  [T.P. formation v2](#)
-  [Une Personne et plusieurs Adresse\(s\)](#)
-  [T.P. formation v3](#)
-  [Bricolage et dépendance](#)
- L'héritage
- Les interfaces
- Le typage
- Les namespaces
- Les exceptions
- Les bases de données avec PDO
- Les tests avec PHPUnit
- Petite application version 2
- Petitie application version 3

[Mes cours](#)

ADMINISTRATION



[Administration du cours](#)

[Règlages de mon profil](#)

Connecté sous le nom « [Arnaud Lemaïs](#) » ([Déconnexion](#))
POO PHP



Programmation orientée objet en PHP

Accueil ► Mes cours ► Développement logiciel ► POO PHP ► La programmation orientée objet : premiers pas ► Le constructeur et this

Le constructeur et this

Lors de la création de notre objet de type Humain nous avons utilisé l'opérateur new et une "méthode" spéciale nommée constructeur. Par défaut chaque classe possède un constructeur, appelé constructeur par défaut ou encore constructeur "vide". Pour appeler ce constructeur nous utilisons le nom de la classe suivie de parenthèses comme pour une fonction classique et sans argument.

Le rôle du constructeur est d'initialiser les attributs de l'objet, c'est à dire leur donner une valeur lors de la création de l'objet. Mais le constructeur par défaut, lui, ne modifie pas les valeurs des attributs. Si je souhaite initialiser les attributs lors de l'appel au constructeur je vais devoir créer un constructeur personnalisé.

Le constructeur personnalisé possède un nom particulier : __construct. Comme n'importe quelle fonction, il peut posséder des paramètres, y compris des paramètres facultatifs. Je pourrai donc appeler ce constructeur en lui donnant en paramètre un age, une taille et un poids. La signature de mon constructeur sera donc __construct(\$page, \$ptaille, \$ppoids). La lettre p devant chaque nom de variable signifie "paramètre", et nous permettra d'éviter les confusions entre les attributs et les paramètres du constructeur.

Dans le corps de la fonction il va falloir que j'assigne à l'attribut age la valeur de page, à l'attribut taille la valeur de ptaille, et à l'attribut poids la valeur de ppoids. Pour effectuer cela je vais avoir besoin du mot clé "this" qui signifie l'instance courante, l'objet que je suis en train de créer. Dans le constructeur, j'écrirai que :

- la valeur de l'attribut age de l'objet que je suis en train de créer (l'instance courante) est égal à la valeur de page
- la valeur de l'attribut taille de l'objet que je suis en train de créer (l'instance courante) est égal à la valeur de ptaille
- la valeur de l'attribut poids de l'objet que je suis en train de créer (l'instance courante) est égal à la valeur de ppoids

Ce qui nous donne en PHP :

```

1 <?php
2
3 // Humain.php
4 class Humain {
5
6     public $age;
7     public $taille;
8     public $poids;
9
10    public function __construct($page, $ptaille, $ppoids) {
11        $this->age = $page;
12        $this->taille = $ptaille;
13        $this->poids = $ppoids;
14    }
15
16 }
17
18 $sparrow = new Humain(42, 1.84, 75);
19 echo "age = $sparrow->age" . PHP_EOL;
20 echo "taille = $sparrow->taille" . PHP_EOL;
21 echo "poids = $sparrow->poids" . PHP_EOL;

```

L'appel au constructeur se fait toujours en utilisant le nom de la classe. Une fois qu'un constructeur personnalisé a été défini, le constructeur par défaut n'existe plus. L'appel à un constructeur vide déclencherait une erreur puisque le constructeur personnalisé attend plusieurs arguments.

```

1 <?php
2
3 // Humain.php
4 class Humain {
5
6     public $age;
7     public $taille;
8     public $poids;
9
10    public function __construct($page, $ptaille, $ppoids) {
11        $this->age = $page;
12        $this->taille = $ptaille;
13        $this->poids = $ppoids;
14    }
15
16 }
17
18 $sparrow = new Humain(); // erreur !
19 echo "age = $sparrow->age" . PHP_EOL;
20 echo "taille = $sparrow->taille" . PHP_EOL;
21 echo "poids = $sparrow->poids" . PHP_EOL;

```

Le mot clé this peut aussi être utilisé dans les méthodes pour accéder à un attribut ou à une autre méthode, par exemple si nous ajoutons une méthode afficherPoids() qui affichera la valeur de l'attribut poids de la personne :

```

1 <?php
2
3 // Humain.php
4 class Humain {
5
6     public $age;
7     public $taille;
8     public $poids;
9
10    public function __construct($page, $ptaille, $ppoids) {
11        $this->age = $page;
12        $this->taille = $ptaille;
13        $this->poids = $ppoids;
14    }
15
16    public function afficherPoids() {
17        echo "Je pèse {$this->poids}kg.";
18    }
19
20 }
21
22 $sparrow = new Humain(42, 1.84, 75);
23 $sparrow->afficherPoids();

```

Autre exemple avec une fonction afficher() qui appelle la méthode afficherPoids() :

```

1 <?php
2
3 // Humain.php
4 class Humain {
5
6     public $age;
7     public $taille;
8     public $poids;
9
10    public function __construct($page, $ptaille, $ppoids) {
11        $this->age = $page;
12        $this->taille = $ptaille;
13        $this->poids = $ppoids;
14    }

```

```

15
16     public function afficherPoids() {
17         echo "Je pèse {$this->poids}kg.";
18     }
19
20     public function afficher() {
21         $this->afficherPoids();
22     }
23
24 }
25
26 $sparrow = new Humain(42, 1.84, 75);
27 $sparrow->afficher();

```

Fin**NAVIGATION****Accueil**

Pages du site

Mon profil

Cours actuel

POO PHP

Participants

Généralités

La programmation orientée objet : premiers pas

Introduction **Classes et objets** **T.P. première classe** **Les classes en PHP** **T.P. Personne** **Le constructeur et this** **T.P. Personne v2** **Accesseurs et mutateurs** **Les méthodes magiques** **T.P. Personne v3** **Les constantes de classe** **Le modificateur static** **T.P. Personne v4** **Une Personne et une Adresse : la relation has-a** **Histoire de références** **T.P. formation** **T.P. formation v2** **Une Personne et plusieurs Adresse(s)** **T.P. formation v3** **Bricolage et dépendance**

L'héritage

Les interfaces

Le typage

Les namespaces

Les exceptions

Les bases de données avec PDO

Les tests avec PHPUnit

Petite application version 2

Petitie application version 3

Mes cours**ADMINISTRATION**

Administration du cours

Réglages de mon profil

Connecté sous le nom « [Arnaud Lemaïs](#) » ([Déconnexion](#))
POO PHP



Programmation orientée objet en PHP

Accueil ► Mes cours ► Développement logiciel ► POO PHP ► La programmation orientée objet : premiers pas ► T.P. Personne v2

T.P. Personne v2

Ajoutez à votre classe Personne un constructeur personnalisé qui prendra en paramètre un nom et un prénom. Modifiez la méthode sePresenter() pour qu'elle affiche "Bonjour je suis " + nom de la personne + prénom de la personne. Créez une instance de Personne et appelez la méthode sePresenter().

Le nom du fichier sera "Personne.php". Vous remettrez une archive zip contenant le fichier. Le nom de l'archive sera de la forme "personnev2_nom_prenom.zip".

État du travail remis

Numéro de tentative	Ceci est la tentative 1.
Statut des travaux remis	Aucune tentative
Statut de l'évaluation	Pas évalué
Dernière modification	vendredi 27 mars 2015, 15:05

[Ajouter un travail](#)

[Modifier votre travail remis](#)

NAVIGATION



[Accueil](#)

■ [Ma page](#)

[Pages du site](#)

[Mon profil](#)

[Cours actuel](#)

[POO PHP](#)

[Participants](#)

[Généralités](#)

[La programmation orientée objet : premiers pas](#)

[Introduction](#)

[Classes et objets](#)

[T.P. première classe](#)

[Les classes en PHP](#)

[T.P. Personne](#)

[Le constructeur et this](#)

[T.P. Personne v2](#)

[Accesseurs et mutateurs](#)

[Les méthodes magiques](#)

[T.P. Personne v3](#)

[Les constantes de classe](#)

[Le modificateur static](#)

[T.P. Personne v4](#)

[Une Personne et une Adresse : la relation has-a](#)

-  [Histoire de références](#)
-  [T.P. formation](#)
-  [T.P. formation v2](#)
-  [Une Personne et plusieurs Adresse\(s\)](#)
-  [T.P. formation v3](#)
-  [Bricolage et dépendance](#)
- L'héritage
- Les interfaces
- Le typage
- Les namespaces
- Les exceptions
- Les bases de données avec PDO
- Les tests avec PHPUnit
- Petite application version 2
- Petitie application version 3

[Mes cours](#)

ADMINISTRATION



[Administration du cours](#)

[Règlages de mon profil](#)

Connecté sous le nom « [Arnaud Lemaïs](#) » ([Déconnexion](#))
POO PHP



Programmation orientée objet en PHP

Accueil ▶ Mes cours ▶ Développement logiciel ▶ POO PHP ▶ La programmation orientée objet : premiers pas ▶ Accesseurs et mutateurs

Accesseurs et mutateurs

En programmation objet il existe un grand principe qui s'appelle l'encapsulation. L'encapsulation signifie que l'état d'un objet ne peut être modifié que par les méthodes de cet objet. Par "état" il faut entendre la valeur des attributs à un moment t. Prenons un exemple avec notre classe Humain : nous allons créer une instance de la classe Humain et modifier son age en accédant directement à l'attribut age via l'opérateur ->. Ensuite nous afficherons la valeur de l'attribut age :

```

1 <?php
2
3 // Humain.php
4 class Humain {
5
6     public $age;
7     public $taille;
8     public $poids;
9
10    public function __construct($page, $ptaille, $ppoids) {
11        $this->age = $page;
12        $this->taille = $ptaille;
13        $this->poids = $ppoids;
14    }
15
16 }
17
18 $sparrow = new Humain(42, 1.84, 75);
19 $sparrow->age = 456;
20 echo "age de Sparrow = $sparrow->age";

```

Ici nous avons pu modifier directement la valeur de l'attribut sans passer par une méthode de Humain. Cela est en contradiction avec le principe d'encapsulation. Si nous voulons rendre impossible l'accès à un attribut ou une méthode en dehors de la classe, nous devons utiliser le mot clé "private", ou lieu du mot clé "public", devant les attributs et méthodes concernés. Ici nous souhaitons simplement "protéger" les trois attributs, notre classe devient donc :

```

1 <?php
2
3 // Humain.php
4 class Humain {
5
6     private $age;
7     private $taille;
8     private $poids;
9
10    public function __construct($page, $ptaille, $ppoids) {
11        $this->age = $page;
12        $this->taille = $ptaille;
13        $this->poids = $ppoids;
14    }
15
16 }
17
18 $sparrow = new Humain(42, 1.84, 75);
19 $sparrow->age = 456; // déclenche une erreur !

```

```
20 echo "age de Sparrow = $sparrow->age";
```

Les mots clés public et private sont des modificateurs d'accès, ils définissent le niveau d'accessibilité ou de visibilité d'un attribut ou d'une méthode. Le mot clé public signifie que l'attribut ou la méthode est accessible partout où la classe est accessible. Le mot clé private signifie que l'attribut ou la méthode n'est accessible qu'au sein de la classe. Il en existe un troisième : protected, dont nous reparlerons bientôt.

Mais comment allons-nous faire maintenant si nous souhaitons récupérer ou modifier un attribut dans notre programme ? Nous allons créer des méthodes spécifiques : une méthode pour récupérer la valeur d'un attribut : un getter ou accesseur, une méthode pour assigner une valeur à un attribut : un setter ou mutateur. Chaque attribut pourra donc posséder un getter et un setter, ou l'un des deux, ou aucun, c'est selon...

En POO l'usage est de nommer les getters `getAttribut` (par exemple `getNom`) et les setters `setAttribut` (par exemple `setAge`). Pour les booléens, le getter s'appellera `isAttribut` (par exemple `isValid`).

Le getter est donc une simple méthode sans paramètre qui va renvoyer la valeur d'un attribut. Le setter est une méthode qui assignera à un attribut une nouvelle valeur passée en argument. La classe Humain devient donc :

```
1 <?php
2
3 // Humain.php
4 class Humain {
5
6     private $age;
7     private $taille;
8     private $poids;
9
10    public function __construct($page, $ptaille, $ppoids) {
11        $this->age = $page;
12        $this->taille = $ptaille;
13        $this->poids = $ppoids;
14    }
15
16    public function getAge() {
17        return $this->age;
18    }
19
20    public function setAge($age) {
21        $this->age = $age;
22    }
23
24 }
25
26 $sparrow = new Humain(42, 1.84, 75);
27 $sparrow->setAge(456);
28 echo "age de Sparrow = {$sparrow->getAge()}";
```

Et maintenant nous pouvons modifier et récupérer la valeur de l'attribut en utilisant le getter et le setter.

PHP permet de définir un getter et un setter unique pour tous les attributs. Le getter sera appelé automatiquement lorsque l'on tentera d'accéder directement à un attribut inaccessible (avec l'opérateur `->`), le setter sera appelé automatiquement lorsque l'on tentera de modifier directement la valeur d'un attribut inaccessible (avec l'opérateur `->`).

Le getter prendra en argument le nom de l'attribut dont on souhaite récupérer la valeur. Le setter prendra en argument l'attribut dont on souhaite modifier la valeur et la nouvelle valeur à assigner.

```
1 <?php
2
3 // Humain.php
4 class Humain {
5
6     private $age;
7     private $taille;
8     private $poids;
9
```

```

10     public function __construct($page, $ptaille, $ppoids) {
11         $this->age = $page;
12         $this->taille = $ptaille;
13         $this->poids = $ppoids;
14     }
15
16     public function __get($name) {
17         return $this->$name;
18     }
19
20     public function __set($name, $value) {
21         $this->$name = $value;
22     }
23
24 }
25
26 $sparrow = new Humain(42, 1.84, 75);
27 $sparrow->age = 456;
28 echo "age de Sparrow = $sparrow->age" . PHP_EOL;
29 $sparrow->taille = 2;
30 echo "la taille de Sparrow est $sparrow->taille";

```

Fin

NAVIGATION[Accueil](#)[■ Ma page](#)[Pages du site](#)[Mon profil](#)[Cours actuel](#)[POO PHP](#)[Participants](#)[Généralités](#)[La programmation orientée objet : premiers pas](#) [Introduction](#) [Classes et objets](#) [T.P. première classe](#) [Les classes en PHP](#) [T.P. Personne](#) [Le constructeur et this](#) [T.P. Personne v2](#) [Accesseurs et mutateurs](#) [Les méthodes magiques](#) [T.P. Personne v3](#) [Les constantes de classe](#) [Le modificateur static](#) [T.P. Personne v4](#) [Une Personne et une Adresse : la relation has-a](#) [Histoire de références](#) [T.P. formation](#) [T.P. formation v2](#) [Une Personne et plusieurs Adresse\(s\)](#) [T.P. formation v3](#) [Bricolage et dépendance](#)[L'héritage](#)[Les interfaces](#)[Le typage](#)[Les namespaces](#)[Les exceptions](#)[Les bases de données avec PDO](#)

Les tests avec PHPUnit

Petite application version 2

Petitie application version 3

[Mes cours](#)



ADMINISTRATION

[Administration du cours](#)

[Réglages de mon profil](#)

Connecté sous le nom « [Arnaud Lemais](#) » ([Déconnexion](#))
[POO PHP](#)



Programmation orientée objet en PHP

Accueil ► Mes cours ► Développement logiciel ► POO PHP ► La programmation orientée objet : premiers pas ► Les méthodes magiques

Les méthodes magiques

PHP propose un ensemble de méthodes prédéfinies dont le nom commence par un double underscore. Ces méthodes sont appelées des **méthodes magiques**. Nous avons déjà rencontré trois de ces méthodes magiques : `__construct()`, `__get()` et `__set()`. Même si ces méthodes sont prédéfinies par PHP et possèdent un comportement par défaut, il est possible, au sein de chaque classe, de redéfinir le comportement d'une ou plusieurs méthodes magiques. Par exemple dans la classe `Humain` nous avons redéfini les méthodes `__construct()`, `__get()` et `__set()`.

Nous pouvons aussi noter que les fonctions magiques sont rarement appelées directement, en général elles sont appelées "en sous main" par PHP. Par exemple la méthode `__construct()` est appelée par PHP quand nous instancions une classe, la méthode `__get()` est appelée par PHP quand nous accédons directement à un attribut, la méthode `__set()` est appelée par PHP quand nous modifions directement la valeur d'un attribut.

Il existe d'autres méthodes magiques comme par exemple la méthode `__toString()` qui renvoie une chaîne de caractères représentant l'objet. Cette méthode est appelée lorsque l'on tente d'afficher un objet.

```
1 <?php
2
3 // Humain.php
4 class Humain {
5
6     private $age;
7     private $taille;
8     private $poids;
9
10    public function __construct($page, $ptaille, $ppoids) {
11        $this->age = $page;
12        $this->taille = $ptaille;
13        $this->poids = $ppoids;
14    }
15
16    public function __get($name) {
17        return $this->$name;
18    }
19
20    public function __set($name, $value) {
21        $this->$name = $value;
22    }
23
24    public function __toString() {
25        return "age : $this->age, taille : $this->taille, poids : $this->poids";
26    }
27
28 }
29
30 $sparrow = new Humain(42, 1.84, 75);
31 echo $sparrow; // appel en sous main de __toString()
```

Fin

NAVIGATION

Accueil



■ Ma page[Pages du site](#)[Mon profil](#)[Cours actuel](#)**POO PHP**[Participants](#)[Généralités](#)[La programmation orientée objet : premiers pas](#) [Introduction](#) [Classes et objets](#) [T.P. première classe](#) [Les classes en PHP](#) [T.P. Personne](#) [Le constructeur et this](#) [T.P. Personne v2](#) [Accesseurs et mutateurs](#) [Les méthodes magiques](#) [T.P. Personne v3](#) [Les constantes de classe](#) [Le modificateur static](#) [T.P. Personne v4](#) [Une Personne et une Adresse : la relation has-a](#) [Histoire de références](#) [T.P. formation](#) [T.P. formation v2](#) [Une Personne et plusieurs Adresse\(s\)](#) [T.P. formation v3](#) [Bricolage et dépendance](#)[L'héritage](#)[Les interfaces](#)[Le typage](#)[Les namespaces](#)[Les exceptions](#)[Les bases de données avec PDO](#)[Les tests avec PHPUnit](#)[Petite application version 2](#)[Petitie application version 3](#)**Mes cours****ADMINISTRATION**[Administration du cours](#)[Réglages de mon profil](#)

Connecté sous le nom « Arnaud Lemais » ([Déconnexion](#))
[POO PHP](#)



Programmation orientée objet en PHP

Accueil ► Mes cours ► Développement logiciel ► POO PHP ► La programmation orientée objet : premiers pas ► T.P. Personne v3

T.P. Personne v3

Dans votre classe Personne, modifiez la visibilité des attributs d'instance en private. Ajoutez des getters et des setters pour chaque attribut (aidez vous de l'IDE). Ajoutez une méthode __toString() qui renverra une chaîne de caractères de la forme "je suis nom prénom.". Instanciez la classe Personne et affichez l'objet créé.

Le nom du fichier sera "Personne.php". Vous remettrez une archive zip contenant ce fichier. Le nom de l'archive sera de la forme "personnev3_nom_prenom.zip".

État du travail remis

Numéro de tentative	Ceci est la tentative 1.
Statut des travaux remis	Aucune tentative
Statut de l'évaluation	Pas évalué
Dernière modification	vendredi 27 mars 2015, 15:06

[Ajouter un travail](#)

[Modifier votre travail remis](#)

NAVIGATION



[Accueil](#)

■ [Ma page](#)

[Pages du site](#)

[Mon profil](#)

[Cours actuel](#)

[POO PHP](#)

[Participants](#)

[Généralités](#)

[La programmation orientée objet : premiers pas](#)

[Introduction](#)

[Classes et objets](#)

[T.P. première classe](#)

[Les classes en PHP](#)

[T.P. Personne](#)

[Le constructeur et this](#)

[T.P. Personne v2](#)

[Accesseurs et mutateurs](#)

[Les méthodes magiques](#)

[T.P. Personne v3](#)

[Les constantes de classe](#)

[Le modificateur static](#)

[T.P. Personne v4](#)

[Une Personne et une Adresse : la relation has-a](#)

-  [Histoire de références](#)
-  [T.P. formation](#)
-  [T.P. formation v2](#)
-  [Une Personne et plusieurs Adresse\(s\)](#)
-  [T.P. formation v3](#)
-  [Bricolage et dépendance](#)
- L'héritage
- Les interfaces
- Le typage
- Les namespaces
- Les exceptions
- Les bases de données avec PDO
- Les tests avec PHPUnit
- Petite application version 2
- Petitie application version 3

[Mes cours](#)

ADMINISTRATION



[Administration du cours](#)

[Règlages de mon profil](#)

Connecté sous le nom « [Arnaud Lemaïs](#) » ([Déconnexion](#))
POO PHP



Programmation orientée objet en PHP

Accueil ► Mes cours ► Développement logiciel ► POO PHP ► La programmation orientée objet : premiers pas ► Les constantes de classe

Les constantes de classe

Il est possible de définir des constantes au sein des classes en utilisant le mot clé const. Ces constantes pourront être appelées à l'intérieur ou à l'extérieur de la classe en utilisant le nom de la classe et l'opérateur "::__". Il n'est donc pas nécessaire de créer une instance pour accéder à une constante de classe.

```

1 <?php
2
3 // Humain.php
4 class Humain {
5
6     const TAILLE_MAX = 2.10;
7
8     private $age;
9     private $taille;
10    private $poids;
11
12    public function __construct($page, $ptaille, $ppoids) {
13        $this->age = $page;
14        $this->taille = $ptaille;
15        $this->poids = $ppoids;
16    }
17
18    public function __get($name) {
19        return $this->$name;
20    }
21
22    public function __set($name, $value) {
23        $this->$name = $value;
24    }
25
26    public function __toString() {
27        return "age : $this->age, taille : $this->taille, poids : $this->poids";
28    }
29
30 }
31
32 echo Humain::__TAILLE_MAX;

```

[Fin](#)

NAVIGATION



[Accueil](#)

■ [Ma page](#)

Pages du site

Mon profil

Cours actuel

[POO PHP](#)

Participants

Généralités

La programmation orientée objet : premiers pas

[Introduction](#)

- [!\[\]\(8b54eb1193a30d999597474e5a23f9ed_img.jpg\) Classes et objets](#)
 - [!\[\]\(e0a24949320a739a8894abfc3bb2a05d_img.jpg\) T.P. première classe](#)
 - [!\[\]\(b2cd8c2db1f5c98c3d2f3bf9889781e8_img.jpg\) Les classes en PHP](#)
 - [!\[\]\(ffc3df84e757707de4804f88c3430ecf_img.jpg\) T.P. Personne](#)
 - [!\[\]\(f55eb32f7518b9377e5dcb394c18f4ad_img.jpg\) Le constructeur et this](#)
 - [!\[\]\(6c713c0db51857feedcbaacee99066ad_img.jpg\) T.P. Personne v2](#)
 - [!\[\]\(1c83a9e5540dbf14b06d01363376cda4_img.jpg\) Accesseurs et mutateurs](#)
 - [!\[\]\(50b0a1f8779db7ea433f4711a6466d0b_img.jpg\) Les méthodes magiques](#)
 - [!\[\]\(8d4c595a7e6902286403c24468d96bf6_img.jpg\) T.P. Personne v3](#)
 - [!\[\]\(7f5f34a60b4ab909a4791644993a44b3_img.jpg\) **Les constantes de classe**](#)
 - [!\[\]\(08f21dc3d9f737d20fcd782206e5ecd1_img.jpg\) Le modificateur static](#)
 - [!\[\]\(391e8b8e855c025b754bed1c28240e90_img.jpg\) T.P. Personne v4](#)
 - [!\[\]\(a424e549c3b262ab4c18a919fc36abcf_img.jpg\) Une Personne et une Adresse : la relation has-a](#)
 - [!\[\]\(a74f13c292fe3185c3531d99c32c0d96_img.jpg\) Histoire de références](#)
 - [!\[\]\(89d8a05286115355fd945401f997a286_img.jpg\) T.P. formation](#)
 - [!\[\]\(816517cb76780930e3668b82c09c9694_img.jpg\) T.P. formation v2](#)
 - [!\[\]\(a520fac9fb585e2bee3fd3e802c2a019_img.jpg\) Une Personne et plusieurs Adresse\(s\)](#)
 - [!\[\]\(9540a2c199d3dd7393e0ea2895af07b4_img.jpg\) T.P. formation v3](#)
 - [!\[\]\(3d25c7cc66d8fc05f04962c55ae9770f_img.jpg\) Bricolage et dépendance](#)
 - L'héritage
 - Les interfaces
 - Le typage
 - Les namespaces
 - Les exceptions
 - Les bases de données avec PDO
 - Les tests avec PHPUnit
 - Petite application version 2
 - Petitie application version 3
- [Mes cours](#)

ADMINISTRATION



[Administration du cours](#)

[Réglages de mon profil](#)

Connecté sous le nom « [Arnaud Lemais](#) » ([Déconnexion](#))
[POO PHP](#)



Programmation orientée objet en PHP

Accueil ▶ Mes cours ▶ Développement logiciel ▶ POO PHP ▶ La programmation orientée objet : premiers pas ▶ Le modificateur static

Le modificateur static

Le modificateur static permet de définir des attributs et méthodes dits "statiques" et qui appartiendront à la classe : nous les appellerons aussi des attributs de classe et des méthodes de classe. Comme pour les constantes, il ne sera pas nécessaire de posséder une instance pour accéder à ces attributs ou méthodes statiques. Pour accéder aux attributs et méthodes de classe, il faudra utiliser le nom de la classe et l'opérateur :: suivi de l'attribut ou de la méthode. Ces attributs et méthodes de classe posséderont une visibilité et seront accessibles à chaque instance. Par exemple nous allons, dans la classe Humain, créer un attribut de classe public "compteur" qui stockera le nombre d'instances que nous avons créées. Ce compteur sera incrémenté dans le constructeur de la classe : à chaque appel au constructeur nous créerons une nouvelle instance et donc nous incrémenterons le compteur.

```
1 <?php
2
3 // Humain.php
4 class Humain {
5
6     const TAILLE_MAX = 2.10;
7
8     public static $nb_instances = 0;
9     private $age;
10    private $taille;
11    private $poids;
12
13    public function __construct($page, $ptaille, $ppoids) {
14        $this->age = $page;
15        $this->taille = $ptaille;
16        $this->poids = $ppoids;
17        Humain::$nb_instances++;
18    }
19
20    public function __get($name) {
21        return $this->$name;
22    }
23
24    public function __set($name, $value) {
25        $this->$name = $value;
26    }
27
28    public function __toString() {
29        return "age : $this->age, taille : $this->taille, poids : $this->poids";
30    }
31
32 }
33
34 $h1 = new Humain(40, 1.80, 75);
35 $h2 = new Humain(35, 1.90, 80);
36 $h3 = new Humain(42, 1.75, 72);
37 $h4 = new Humain(39, 1.78, 77);
38 echo Humain::$nb_instances;
```

Voici un exemple de méthode de classe. Dans une classe Convertisseur, nous créons une constante USD dont la valeur est 1.37 (le taux de change du dollar). La méthode de classe toUSD() prendra en paramètre un double (le montant en euro) et renverra le montant en Dollar US.

```
1 <?php
2
3 // Convertisseur.php
4 class Convertisseur {
5
6     const USD = 1.37;
7
8     public static function euroToUSD($amount) {
9         return $amount * Convertisseur::USD;
10    }
11 }
12 }
13
14 $montant = 450;
15 echo Convertisseur::euroToUSD($montant);
```

Fin

NAVIGATION



[Accueil](#)

■ [Ma page](#)

[Pages du site](#)

[Mon profil](#)

[Cours actuel](#)

[POO PHP](#)

[Participants](#)

[Généralités](#)

[La programmation orientée objet : premiers pas](#)

[Introduction](#)

[Classes et objets](#)

[T.P. première classe](#)

[Les classes en PHP](#)

[T.P. Personne](#)

[Le constructeur et this](#)

[T.P. Personne v2](#)

[Accesseurs et mutateurs](#)

[Les méthodes magiques](#)

[T.P. Personne v3](#)

[Les constantes de classe](#)

[Le modificateur static](#)

[T.P. Personne v4](#)

[Une Personne et une Adresse : la relation has-a](#)

[Histoire de références](#)

[T.P. formation](#)

[T.P. formation v2](#)

[Une Personne et plusieurs Adresse\(s\)](#)

[T.P. formation v3](#)

[Bricolage et dépendance](#)

[L'héritage](#)

[Les interfaces](#)

[Le typage](#)

[Les namespaces](#)

[Les exceptions](#)

[Les bases de données avec PDO](#)

[Les tests avec PHPUnit](#)

[Petite application version 2](#)

[Petite application version 3](#)

[Mes cours](#)

ADMINISTRATION[Administration du cours](#)[Réglages de mon profil](#)

Connecté sous le nom « [Arnaud Lemaïs](#) » ([Déconnexion](#))
[POO PHP](#)



Programmation orientée objet en PHP

Accueil ► Mes cours ► Développement logiciel ► POO PHP ► La programmation orientée objet : premiers pas ► T.P. Personne v4

T.P. Personne v4

Ajoutez à votre classe Personne un attribut d'instance privé age. Modifiez le constructeur. Créez une constante MAJORITE qui aura pour valeur 18. Créez une méthode de classe isMajeur() qui renverra true si l'age de la personne est supérieur ou égal à la constante MAJORITE, false sinon. Créez une instance de Personne et testez votre méthode.

Le nom du fichier sera "Personne.php". Vous remettrez une archive zip contenant le fichier. Le nom de l'archive sera de la forme "personnev4_nom_prenom.zip".

État du travail remis

Numéro de tentative	Ceci est la tentative 1.
Statut des travaux remis	Aucune tentative
Statut de l'évaluation	Pas évalué
Dernière modification	vendredi 27 mars 2015, 15:06

[Ajouter un travail](#)

[Modifier votre travail remis](#)

NAVIGATION



[Accueil](#)

■ [Ma page](#)

[Pages du site](#)

[Mon profil](#)

[Cours actuel](#)

[POO PHP](#)

[Participants](#)

[Généralités](#)

[La programmation orientée objet : premiers pas](#)

[Introduction](#)

[Classes et objets](#)

[T.P. première classe](#)

[Les classes en PHP](#)

[T.P. Personne](#)

[Le constructeur et this](#)

[T.P. Personne v2](#)

[Accesseurs et mutateurs](#)

[Les méthodes magiques](#)

[T.P. Personne v3](#)

[Les constantes de classe](#)

[Le modificateur static](#)

[T.P. Personne v4](#)

 Une Personne et une Adresse : la relation has-a

 Histoire de références

 T.P. formation

 T.P. formation v2

 Une Personne et plusieurs Adresse(s)

 T.P. formation v3

 Bricolage et dépendance

L'héritage

Les interfaces

Le typage

Les namespaces

Les exceptions

Les bases de données avec PDO

Les tests avec PHPUnit

Petite application version 2

Petitie application version 3

[Mes cours](#)

ADMINISTRATION



[Administration du cours](#)

[Réglages de mon profil](#)

Connecté sous le nom « [Arnaud Lemaïs](#) » ([Déconnexion](#))

[POO PHP](#)



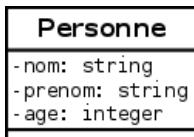
Programmation orientée objet en PHP

Accueil ▶ Mes cours ▶ Développement logiciel ▶ POO PHP ▶ La programmation orientée objet : premiers pas ▶ Une Personne et une Adresse : la relation has-a

Une Personne et une Adresse : la relation has-a

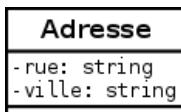
La relation has-a est une relation d'appartenance qui signifie qu'un objet va pouvoir posséder un autre objet. Par exemple une personne possède une adresse. Pour réaliser cet exemple nous allons avoir besoin de deux classes : la classe Personne et la classe Adresse.

La classe Personne possédera comme attribut un nom, un prénom et un age, tous privés :

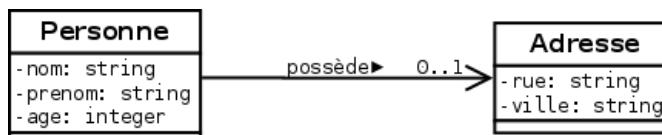


Dans le diagramme le signe "-" devant les attributs signifie "private", les ":" signifient "est de type".

La classe Adresse possédera une rue et une ville (pour aller plus vite...) :



Entre ces deux classes nous avons une relation has-a (possède un) puisque nous pouvons dire qu'une personne possède une adresse, dit autrement, une instance de Personne possède une instance d'Adresse.



La relation has-a est représentée par la ligne (l'association) qui relie les deux classes. Cette association se lit "une personne possède 0 ou 1 adresse".

Quand un objet A possède un objet B, cela signifie que l'objet A possède l'objet B en attribut d'instance. Dans le cas présent, cela signifie que Personne possédera un attribut d'instance de type Adresse. Nous appellerons cet attribut adressePrincipale.

Nous allons implémenter ces deux classes, chacune dans un fichier :

```

1 <?php
2
3 // Personne.php
4 class Personne {
5
6     private $nom;
7     private $prenom;
8     private $age;
9     private $adressePrincipale;
10
11     function __construct($nom, $prenom, $age, $adressePrincipale) {
12         $this->nom = $nom;
13         $this->prenom = $prenom;
14         $this->age = $age;
15         $this->adressePrincipale = $adressePrincipale;
16     }
17
18     public function getNom() {
19         return $this->nom;
20     }
21
  
```

```

22     public function getPrenom() {
23         return $this->prenom;
24     }
25
26     public function getAge() {
27         return $this->age;
28     }
29
30     public function getAdressePrincipale() {
31         return $this->adressePrincipale;
32     }
33
34     public function setNom($nom) {
35         $this->nom = $nom;
36     }
37
38     public function setPrenom($prenom) {
39         $this->prenom = $prenom;
40     }
41
42     public function setAge($age) {
43         $this->age = $age;
44     }
45
46     public function setAdressePrincipale($adressePrincipale) {
47         $this->adressePrincipale = $adressePrincipale;
48     }
49
50 }
```

Le constructeur de Personne prendra en argument un objet de type Adresse ; cet objet de type Adresse sera "stocké" en attribut d'instance.

```

1 <?php
2
3 // Adresse.php
4 class Adresse {
5
6     private $rue;
7     private $ville;
8
9     function __construct($rue, $ville) {
10        $this->rue = $rue;
11        $this->ville = $ville;
12    }
13
14    public function getRue() {
15        return $this->rue;
16    }
17
18    public function getVille() {
19        return $this->ville;
20    }
21
22    public function setRue($rue) {
23        $this->rue = $rue;
24    }
25
26    public function setVille($ville) {
27        $this->ville = $ville;
28    }
29
30 }
```

[Fin](#)

NAVIGATION



Accueil

■ Ma page

[Pages du site](#)[Mon profil](#)[Cours actuel](#)

POO PHP

[Participants](#)[Généralités](#)[La programmation orientée objet : premiers pas](#)

■ Introduction

■ Classes et objets

■ T.P. première classe

■ Les classes en PHP

■ T.P. Personne

■ Le constructeur et this

■ T.P. Personne v2

■ Accesseurs et mutateurs

■ Les méthodes magiques

■ T.P. Personne v3

■ Les constantes de classe

■ Le modificateur static

■ T.P. Personne v4

■ **Une Personne et une Adresse : la relation has-a**

■ Histoire de références

■ T.P. formation

■ T.P. formation v2

■ Une Personne et plusieurs Adresse(s)

■ T.P. formation v3

■ Bricolage et dépendance

[L'héritage](#)[Les interfaces](#)[Le typage](#)[Les namespaces](#)[Les exceptions](#)[Les bases de données avec PDO](#)[Les tests avec PHPUnit](#)[Petite application version 2](#)[Petite application version 3](#)[Mes cours](#)

ADMINISTRATION

[Administration du cours](#)[Régagements de mon profil](#)

Connecté sous le nom « Arnaud Lemaïs » (Déconnexion)

POO PHP



Programmation orientée objet en PHP

[Accueil](#) ▶ [Mes cours](#) ▶ [Développement logiciel](#) ▶ [POO PHP](#) ▶ La programmation orientée objet : premiers pas ▶ [Histoire de références](#)

Histoire de références

Dans un fichier lanceur.php nous allons instancier les classes Adresse et Personne.

```

1 <?php
2
3 // lanceur.php
4 include './Adresse.php';
5 include './Personne.php';
6
7 // création d'une instance d'adresse
8 $adr = new Adresse('Lignier', 'Paris');
9 // création d'une instance de Personne avec son adresse
10 $pers = new Personne('Sparrow', 'Jack', 42, $adr);

```

La méthode getAdressePrincipale() de Personne permet de récupérer l'objet Adresse, et la méthode getRue() d'Adresse permet de récupérer la rue.

```

1 <?php
2
3 // lanceur.php
4 include './Adresse.php';
5 include './Personne.php';
6
7 // création d'une instance d'adresse
8 $adr = new Adresse('Lignier', 'Paris');
9
10 // création d'une instance de Personne avec son adresse
11 $pers = new Personne('Sparrow', 'Jack', 42, $adr);
12
13 // récupération de l'adresse de pers via le getter
14 $adresse_de_la_personne = $pers->getAdressePrincipale();
15
16 // récupération et affichage de la rue via le getter
17 echo $adresse_de_la_personne->getRue();
18
19 // ou en une seule ligne
20 echo $pers->getAdressePrincipale()->getRue();

```

Dans cet exemple, l'attribut adressePrincipale de l'objet pers et la variable adresse_de_la_personne indiquent le même objet, ce sont en fait deux références au même objet, c'est-à-dire deux "flèches" qui indiquent le même objet en mémoire.

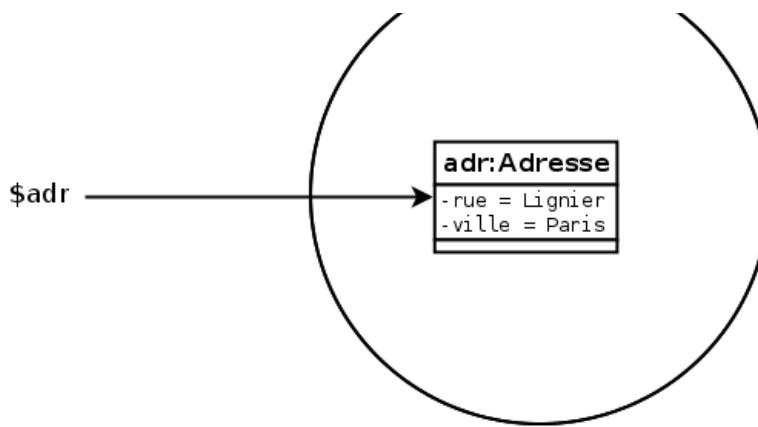
Quand nous instancions une classe, nous créons un objet dans une zone particulière de la mémoire que nous appellerons le "tas". Les objets vivent donc sur le tas. Les variables de notre script seront, elles, créées dans un autre endroit de la mémoire que nous appellerons la "pile". Les termes de "tas" et de "pile" ont été choisis pour simplifier le débat...

Dans notre lanceur, l'instruction

```
$adr = new Adresse('Lignier', 'Paris');
```

crée une variable dans la pile qui sera une référence au nouvel objet de type Adresse sur le tas.

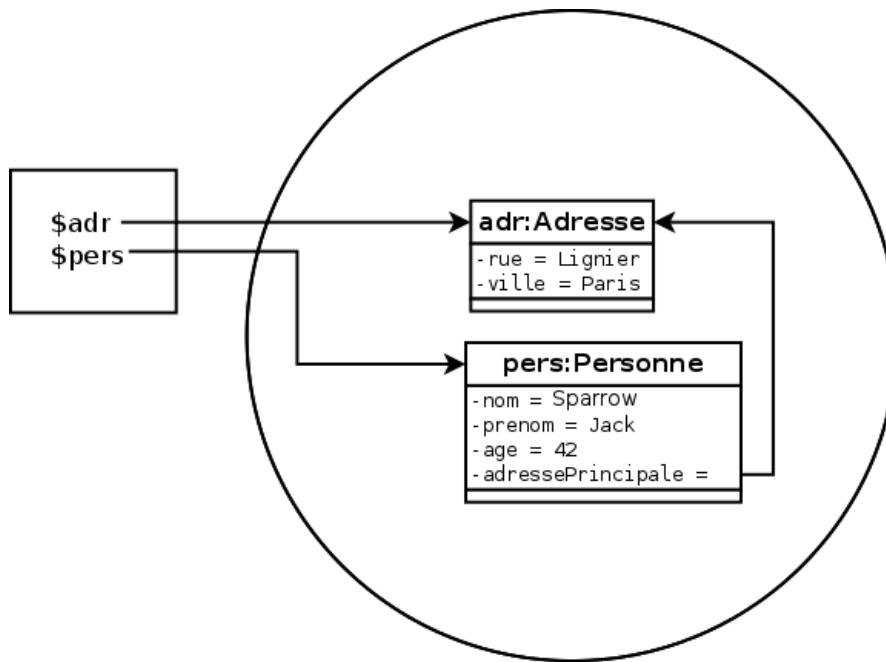




L'instruction

```
$pers = new Personne('Sparrow', 'Jack', 42, $adr);
```

va créer une référence pers à un objet de type Personne, cet objet possédera en attribut d'instance l'objet référencé par adr. L'attribut adressePrincipale de l'objet pers sera donc une autre référence à l'objet de type Adresse créé sur le tas.

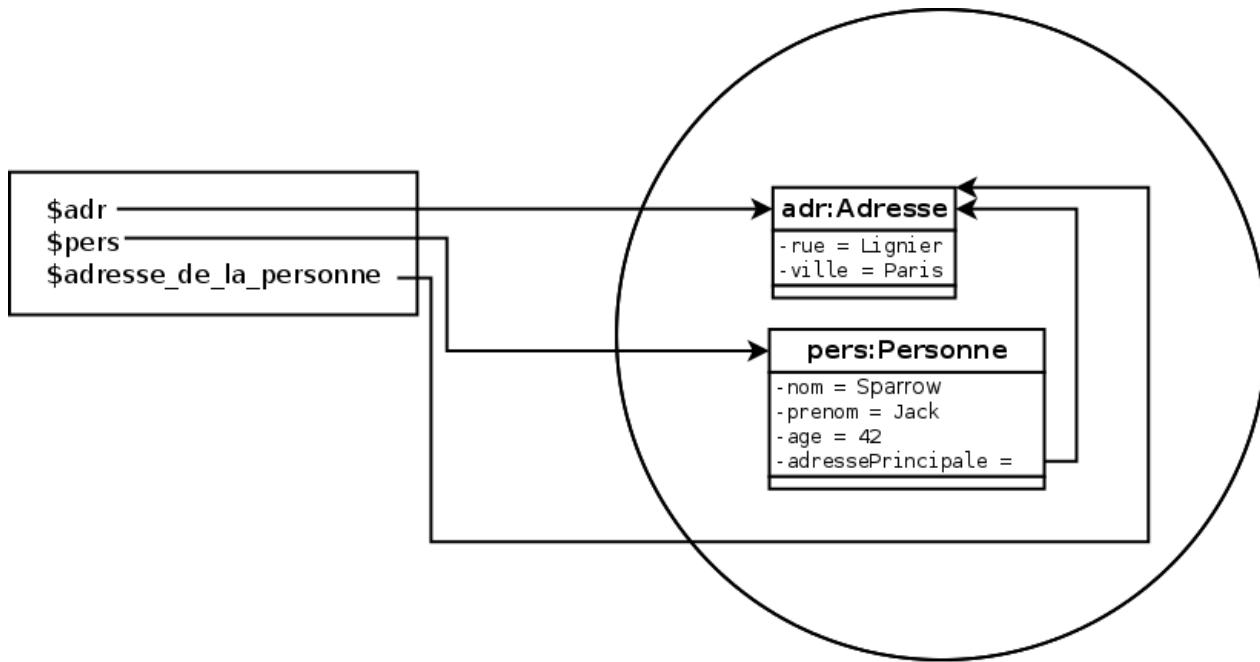


La variable \$adr et l'attribut d'instance adressePrincipale font tous les deux références au même objet. Ce sont en fait des "flèches" qui indiquent un objet sur le tas, nous pouvons donc accéder à cet objet en utilisant l'une ou l'autre de ces références.

L'instruction

```
$adresse_de_la_personne = $pers->getAdressePrincipale();
```

crée une variable adresse_de_la_personne et lui assigne la valeur renvoyée par le getter, c'est-à-dire une référence à l'objet Adresse de pers.



La variable `adresse_de_la_personne` constitue donc une troisième référence à l'objet `adr` de type `Adresse`.

Pour en finir (provisoirement) avec les références, je vous rappelle que dans une fonction les objets en argument sont passés par référence. Cela signifie que dans une fonction, ou méthode, le traitement s'effectue sur le même objet que celui passé en argument et non sur une copie.

Voici un exemple : nous allons créer dans notre lanceur une fonction `modifierAge()` qui prendra en paramètre un objet de type `Personne` et un entier `age` et qui assignera à l'attribut `d'instance age` de `Personne` la valeur `age`. Puis nous afficherons la valeur de l'`age` de la `Personne` :

```

1 <?php
2
3 // lanceur.php
4 include './Adresse.php';
5 include './Personne.php';
6
7 // création d'une instance d'adresse
8 $adr = new Adresse('Lignier', 'Paris');
9
10 // création d'une instance de Personne avec son adresse
11 $pers = new Personne('Sparrow', 'Jack', 42, $adr);
12
13 // récupération de l'adresse de pers via le getter
14 $adresse_de_la_personne = $pers->getAdressePrincipale();
15
16 // récupération et affichage de la rue via le getter
17 echo $adresse_de_la_personne->getRue();
18
19 // ou en une seule ligne
20 echo $pers->getAdressePrincipale()->getRue();
21
22 function modifierAge($personne, $age) {
23     $personne->setAge($age);
24 }
25
26 modifierAge($pers, 789);
27 echo $pers->getAge();

```

Fin

NAVIGATION

[Accueil](#)

■ [Ma page](#)



[Pages du site](#)[Mon profil](#)[Cours actuel](#)

POO PHP

[Participants](#)[Généralités](#)[La programmation orientée objet : premiers pas](#) [Introduction](#) [Classes et objets](#) [T.P. première classe](#) [Les classes en PHP](#) [T.P. Personne](#) [Le constructeur et this](#) [T.P. Personne v2](#) [Accesseurs et mutateurs](#) [Les méthodes magiques](#) [T.P. Personne v3](#) [Les constantes de classe](#) [Le modificateur static](#) [T.P. Personne v4](#) [Une Personne et une Adresse : la relation has-a](#) [Histoire de références](#) [T.P. formation](#) [T.P. formation v2](#) [Une Personne et plusieurs Adresse\(s\)](#) [T.P. formation v3](#) [Bricolage et dépendance](#)[L'héritage](#)[Les interfaces](#)[Le typage](#)[Les namespaces](#)[Les exceptions](#)[Les bases de données avec PDO](#)[Les tests avec PHPUnit](#)[Petite application version 2](#)[Petitie application version 3](#)[Mes cours](#)

ADMINISTRATION

[Administration du cours](#)[Réglages de mon profil](#)Connecté sous le nom « Arnaud Lemaïs » ([Déconnexion](#))

POO PHP

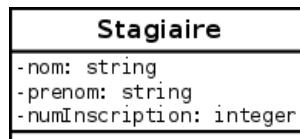


Programmation orientée objet en PHP

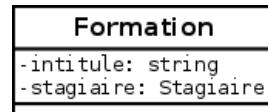
[Accueil](#) ▶ [Mes cours](#) ▶ [Développement logiciel](#) ▶ [POO PHP](#) ▶ La programmation orientée objet : premiers pas ▶ [T.P. formation](#)

T.P. formation

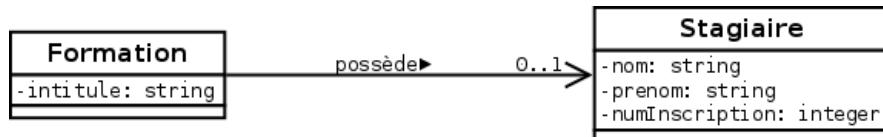
Dans un fichier Stagiaire.php, créez une classe Stagiaire qui possédera un nom, un prénom et un numéro d'inscription (numInscription) en attribut. Générez un constructeur personnalisé qui prendra en argument un nom, un prénom et un numéro d'inscription. Générez les getters et setters.



Dans un fichier Formation.php, créez une classe Formation qui possédera en attribut d'instance un intitulé et un stagiaire. Générez un constructeur personnalisé qui prendra en argument un intitulé et un objet de type Stagiaire. Générez les getters et setters. Cette formation sera très personnalisée puisque une formation ne pourra posséder qu'un seul stagiaire !



Nous aurions pu aussi représenter ces classes de cette façon :



Dans un fichier lanceur.php, créez un objet Stagiaire, un objet Formation qui possédera ce stagiaire et afficher à partir de la référence à l'objet Formation le nom du stagiaire.

Vous remettrez une archive zip contenant ces fichiers. Le nom de l'archive sera de la forme "formation_nom_prenom.zip".

État du travail remis

Numéro de tentative	Ceci est la tentative 1.
Statut des travaux remis	Aucune tentative
Statut de l'évaluation	Pas évalué
Dernière modification	vendredi 27 mars 2015, 15:07

[Ajouter un travail](#)

[Modifier votre travail remis](#)

NAVIGATION



[Accueil](#)

■ [Ma page](#)

[Pages du site](#)[Mon profil](#)[Cours actuel](#)**POO PHP**[Participants](#)[Généralités](#)[La programmation orientée objet : premiers pas](#) [Introduction](#) [Classes et objets](#) [T.P. première classe](#) [Les classes en PHP](#) [T.P. Personne](#) [Le constructeur et this](#) [T.P. Personne v2](#) [Accesseurs et mutateurs](#) [Les méthodes magiques](#) [T.P. Personne v3](#) [Les constantes de classe](#) [Le modificateur static](#) [T.P. Personne v4](#) [Une Personne et une Adresse : la relation has-a](#) [Histoire de références](#) [T.P. formation](#) [T.P. formation v2](#) [Une Personne et plusieurs Adresse\(s\)](#) [T.P. formation v3](#) [Bricolage et dépendance](#)[L'héritage](#)[Les interfaces](#)[Le typage](#)[Les namespaces](#)[Les exceptions](#)[Les bases de données avec PDO](#)[Les tests avec PHPUnit](#)[Petite application version 2](#)[Petitie application version 3](#)[Mes cours](#)**ADMINISTRATION**[Administration du cours](#)[Réglages de mon profil](#)Connecté sous le nom « [Arnaud Lemaïs](#) » ([Déconnexion](#))

POO PHP



Programmation orientée objet en PHP

Accueil ► Mes cours ► Développement logiciel ► POO PHP ► La programmation orientée objet : premiers pas ► T.P. formation v2

T.P. formation v2

Ajoutez à la classe Stagiaire une méthode `__toString()` qui renverra une chaîne de caractères contenant le nom, le prénom et le numéro d'inscription du stagiaire.

Ajoutez à la classe Formation une méthode `__toString()` qui affichera l'intitulé de la formation puis le nom, le prénom et le numéro d'inscription du stagiaire en utilisant la méthode `__toString()` de Stagiaire.

Modifiez votre lanceur pour qu'il affiche l'objet Formation.

Vous remettrez une archive zip contenant ces fichiers. Le nom de l'archive sera de la forme "formation_2_nom_prenom.zip".

État du travail remis

Numéro de tentative	Ceci est la tentative 1.
Statut des travaux remis	Aucune tentative
Statut de l'évaluation	Pas évalué
Dernière modification	vendredi 27 mars 2015, 15:07

[Ajouter un travail](#)

[Modifier votre travail remis](#)

NAVIGATION



[Accueil](#)

■ [Ma page](#)

Pages du site

Mon profil

Cours actuel

[POO PHP](#)

Participants

Généralités

La programmation orientée objet : premiers pas

[Introduction](#)

[Classes et objets](#)

[T.P. première classe](#)

[Les classes en PHP](#)

[T.P. Personne](#)

[Le constructeur et this](#)

[T.P. Personne v2](#)

[Accesseurs et mutateurs](#)

[Les méthodes magiques](#)

[T.P. Personne v3](#)

[Les constantes de classe](#)

[Le modificateur static](#)

-  **T.P. Personne v4**
-  Une Personne et une Adresse : la relation has-a
-  Histoire de références
-  T.P. formation
-  **T.P. formation v2**
-  Une Personne et plusieurs Adresse(s)
-  T.P. formation v3
-  Bricolage et dépendance
- L'héritage
- Les interfaces
- Le typage
- Les namespaces
- Les exceptions
- Les bases de données avec PDO
- Les tests avec PHPUnit
- Petite application version 2
- Petitie application version 3

[Mes cours](#)

ADMINISTRATION



[Administration du cours](#)

[Réglages de mon profil](#)

Connecté sous le nom « [Arnaud Lemaïs](#) » ([Déconnexion](#))
POO PHP

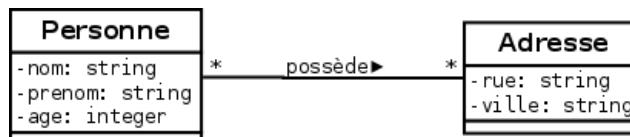


Programmation orientée objet en PHP

Accueil ► Mes cours ► Développement logiciel ► POO PHP ► La programmation orientée objet : premiers pas ► Une Personne et plusieurs Adresse(s)

Une Personne et plusieurs Adresse(s)

Comment pourrions-nous donner à une classe Personne plusieurs Adresse(s) ? Si une Personne possède plusieurs Adresse(s) cela signifie que la classe Personne possédera plusieurs attributs d'instance Adresse. Ce que nous pourrions représenter par le schéma :



L'association (le lien) entre les deux classes signifie qu'une Personne peut posséder plusieurs Adresse(s) et qu'une Adresse peut appartenir à plusieurs Personne(s). Le caractère "*" signifie "plusieurs".

Il nous suffit donc de créer autant d'attributs d'instance que d'adresses : si la Personne possède 4 adresses alors nous aurons 4 attributs d'instance (adresse1, adresse2, etc...). Mais si le nombre d'Adresse(s) par Personne est indéterminé alors il nous faudra choisir une autre solution, celle du tableau. Notre classe Personne possédera un attribut d'instance de type array qui stockera l'ensemble de ses Adresse(s). Cet attribut d'instance sera nommé adresses (avec un "s" à la fin pour bien montrer qu'il y en a plusieurs).

```

1 <?php
2
3 // Personne.php
4 class Personne {
5
6     private $nom;
7     private $prenom;
8     private $age;
9     private $adresses;
10
11    function __construct($nom, $prenom, $age, $adresses = []) {
12        $this->nom = $nom;
13        $this->prenom = $prenom;
14        $this->age = $age;
15        $this->adresses = $adresses;
16    }
17
18    public function getNom() {
19        return $this->nom;
20    }
21
22    public function getPrenom() {
23        return $this->prenom;
24    }
25
26    public function getAge() {
27        return $this->age;
28    }
29
30    public function getAdresses() {
31        return $this->adresses;
32    }
33
34    public function setNom($nom) {
35        $this->nom = $nom;
  
```

```

36     }
37
38     public function setPrenom($prenom) {
39         $this->prenom = $prenom;
40     }
41
42     public function setAge($age) {
43         $this->age = $age;
44     }
45
46     public function setAdresses($adresses) {
47         $this->adresses = $adresses;
48     }
49
50 }
```

Le constructeur de Personne possède un paramètre facultatif qui correspond au tableau de Personne. Nous pourrons, lors de l'instanciation de l'objet, passer en argument au constructeur un tableau d'Adresse(s), dans ce cas une copie de ce tableau deviendra attribut d'instance. Mais nous pourrons aussi laisser ce paramètre vide, dans ce cas l'attribut d'instance sera un tableau vide que nous pourrons remplir plus tard avec une méthode addAdresse() par exemple :

```

1 <?php
2
3 // Personne.php
4 class Personne {
5
6     private $nom;
7     private $prenom;
8     private $age;
9     private $adresses;
10
11    function __construct($nom, $prenom, $age, $adresses = []) {
12        $this->nom = $nom;
13        $this->prenom = $prenom;
14        $this->age = $age;
15        $this->adresses = $adresses;
16    }
17
18    public function getNom() {
19        return $this->nom;
20    }
21
22    public function getPrenom() {
23        return $this->prenom;
24    }
25
26    public function getAge() {
27        return $this->age;
28    }
29
30    public function getAdresses() {
31        return $this->adresses;
32    }
33
34    public function setNom($nom) {
35        $this->nom = $nom;
36    }
37
38    public function setPrenom($prenom) {
39        $this->prenom = $prenom;
40    }
41
42    public function setAge($age) {
43        $this->age = $age;
```

```

44     }
45
46     public function setAdresses($adresses) {
47         $this->adresses = $adresses;
48     }
49
50     public function addAdresse($adresse) {
51         $this->adresses[] = $adresse;
52     }
53
54 }
```

Et voici notre lanceur :

```

1 <?php
2
3 // lanceur.php
4 include './Adresse.php';
5 include './Personne.php';
6
7 $adr1 = new Adresse('Lignier', 'Paris');
8 $adr2 = new Adresse('Turbigo', 'Paris');
9 $sparrow = new Personne('Sparrow', 'Jack', 42, [$adr1, $adr2]);
10 $wayne = new Personne('Wayne', 'Bruce', 40);
11 print_r($sparrow->getAdresses());
12 $wayne->addAdresse($adr2);
13 $wayne->addAdresse($adr1);
14 print_r($wayne->getAdresses());
```

Nous pouvons ajouter à notre classe Adresse une méthode `__toString()` qui renverra une chaîne de caractères contenant la rue et la ville :

```

1 <?php
2
3 // Adresse.php
4 class Adresse {
5
6     private $rue;
7     private $ville;
8
9     function __construct($rue, $ville) {
10         $this->rue = $rue;
11         $this->ville = $ville;
12     }
13
14     public function getRue() {
15         return $this->rue;
16     }
17
18     public function getVille() {
19         return $this->ville;
20     }
21
22     public function setRue($rue) {
23         $this->rue = $rue;
24     }
25
26     public function setVille($ville) {
27         $this->ville = $ville;
28     }
29
30     public function __toString() {
31         return "rue : $this->rue, ville : $this->ville";
32     }
33 }
```

34

}

Nous ajoutons à Personne une méthode `displayAdresses()` qui renverra une chaîne de caractères avec toutes les Adresses et qui réutilisera la méthode `__toString()` d'Adresse. Et nous créons une méthode `__toString()` qui renverra une chaîne contenant le nom, le prénom, l'age et les adresses de la Personne.

```
1 <?php
2
3 // Personne.php
4 class Personne {
5
6     private $nom;
7     private $prenom;
8     private $age;
9     private $adresses;
10
11    function __construct($nom, $prenom, $age, $adresses = []) {
12        $this->nom = $nom;
13        $this->prenom = $prenom;
14        $this->age = $age;
15        $this->adresses = $adresses;
16    }
17
18    public function getNom() {
19        return $this->nom;
20    }
21
22    public function getPrenom() {
23        return $this->prenom;
24    }
25
26    public function getAge() {
27        return $this->age;
28    }
29
30    public function getAdresses() {
31        return $this->adresses;
32    }
33
34    public function setNom($nom) {
35        $this->nom = $nom;
36    }
37
38    public function setPrenom($prenom) {
39        $this->prenom = $prenom;
40    }
41
42    public function setAge($age) {
43        $this->age = $age;
44    }
45
46    public function setAdresses($adresses) {
47        $this->adresses = $adresses;
48    }
49
50    public function addAdresse($adresse) {
51        $this->adresses[] = $adresse;
52    }
53
54    public function displayAdresses() {
55        $result = "";
56        foreach ($this->adresses as $adresse) {
57            $result .= $adresse . " ";
58        }
59        return $result;
60    }
61
```

```

61
62     public function __toString() {
63         return "nom : $this->nom prénom : $this->prenom age : "
64             . "$this->age adresse(s) : {$this->displayAdresses()}";
65     }
66 }
67 }
```

Notre lanceur devient :

```

1 <?php
2
3 // lanceur.php
4 include './Adresse.php';
5 include './Personne.php';
6
7 $adr1 = new Adresse('Lignier', 'Paris');
8 $adr2 = new Adresse('Turbigo', 'Paris');
9 $sparrow = new Personne('Sparrow', 'Jack', 42, [$adr1, $adr2]);
10 $wayne = new Personne('Wayne', 'Bruce', 40);
11 echo $sparrow;
12 $wayne->addAdresse($adr2);
13 $wayne->addAdresse($adr1);
14 echo $wayne;
```

Fin

NAVIGATION



Accueil

- Ma page

Pages du site

Mon profil

Cours actuel

POO PHP

Participants

Généralités

La programmation orientée objet : premiers pas

[Introduction](#)

[Classes et objets](#)

[T.P. première classe](#)

[Les classes en PHP](#)

[T.P. Personne](#)

[Le constructeur et this](#)

[T.P. Personne v2](#)

[Accesseurs et mutateurs](#)

[Les méthodes magiques](#)

[T.P. Personne v3](#)

[Les constantes de classe](#)

[Le modificateur static](#)

[T.P. Personne v4](#)

[Une Personne et une Adresse : la relation has-a](#)

[Histoire de références](#)

[T.P. formation](#)

[T.P. formation v2](#)

[Une Personne et plusieurs Adresse\(s\)](#)

[T.P. formation v3](#)

[Bricolage et dépendance](#)

L'héritage

Les interfaces

Le typage

Les namespaces
Les exceptions
Les bases de données avec PDO
Les tests avec PHPUnit
Petite application version 2
Petitie application version 3

[Mes cours](#)

ADMINISTRATION



[Administration du cours](#)

[RégLAGES de mon profil](#)

Connecté sous le nom « [Arnaud Lemais](#) » ([Déconnexion](#))
[POO PHP](#)

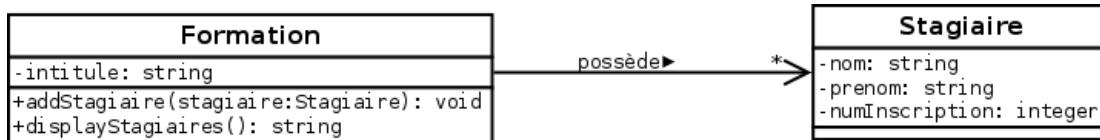


Programmation orientée objet en PHP

Accueil > Mes cours > Développement logiciel > POO PHP > La programmation orientée objet : premiers pas > T.P. formation v3

T.P. formation v3

Nous allons modifier notre classe Formation pour qu'elle puisse posséder plusieurs stagiaires.



Pour cela, dans la classe Formation, supprimez l'attribut stagiaire et créez un attribut stagiaires de type array. Modifiez le constructeur pour qu'il accepte un paramètre facultatif de type array. Ajoutez une méthode `addStagiaire()` qui prendra en argument un objet stagiaire et qui ajoutera ce stagiaire à l'attribut d'instance `stagiaires`. Implémentez une méthode `displayStagiaires()` qui renverra une chaîne de caractères avec les noms, les prénoms et les numéros d'inscription de chaque stagiaire. Modifiez la méthode `__toString()` pour qu'elle renvoie une chaîne de caractères avec l'intitulé de la formation et la liste de tous les stagiaires (utilisez la méthode `displayStagiaires()`).

Dans un fichier lanceur.php, créez des objets Stagiaire, un objet Formation, ajoutez les stagiaires à la formation et affichez l'objet Formation.

Vous remettrez une archive zip contenant ces fichiers. Le nom de l'archive sera de la forme "formation_3_nom_prenom.zip".

État du travail remis

Numéro de tentative	Ceci est la tentative 1.
Statut des travaux remis	Aucune tentative
Statut de l'évaluation	Pas évalué
Dernière modification	vendredi 27 mars 2015, 15:07

[Ajouter un travail](#)

[Modifier votre travail remis](#)

NAVIGATION



[Accueil](#)

[■ Ma page](#)

Pages du site

Mon profil

Cours actuel

[POO PHP](#)

Participants

Généralités

La programmation orientée objet : premiers pas

[Introduction](#)

[Classes et objets](#)

- [T.P. première classe](#)
- [Les classes en PHP](#)
- [T.P. Personne](#)
- [Le constructeur et this](#)
- [T.P. Personne v2](#)
- [Accesseurs et mutateurs](#)
- [Les méthodes magiques](#)
- [T.P. Personne v3](#)
- [Les constantes de classe](#)
- [Le modificateur static](#)
- [T.P. Personne v4](#)
- [Une Personne et une Adresse : la relation has-a](#)
- [Histoire de références](#)
- [T.P. formation](#)
- [T.P. formation v2](#)
- [Une Personne et plusieurs Adresse\(s\)](#)
- [**T.P. formation v3**](#)
- [Bricolage et dépendance](#)
- L'héritage
- Les interfaces
- Le typage
- Les namespaces
- Les exceptions
- Les bases de données avec PDO
- Les tests avec PHPUnit
- Petite application version 2
- Petitie application version 3

[Mes cours](#)



ADMINISTRATION

[Administration du cours](#)

[RégLAGES de mon profil](#)

Connecté sous le nom « Arnaud Lemais » ([Déconnexion](#))
POO PHP



Programmation orientée objet en PHP

Accueil > Mes cours > Développement logiciel > POO PHP > La programmation orientée objet : premiers pas > Bricolage et dépendance

Bricolage et dépendance

Bricolage et dépendance

Un bricoleur souhaite fixer au mur un nouveau tableau. Pour cela il va avoir besoin d'un marteau (je laisse de côté le clou...), et d'un tableau. La classe Bricoleur possédera donc une méthode fixerTableau() dans laquelle elle utilisera un objet de type Marteau et un objet de type Tableau.

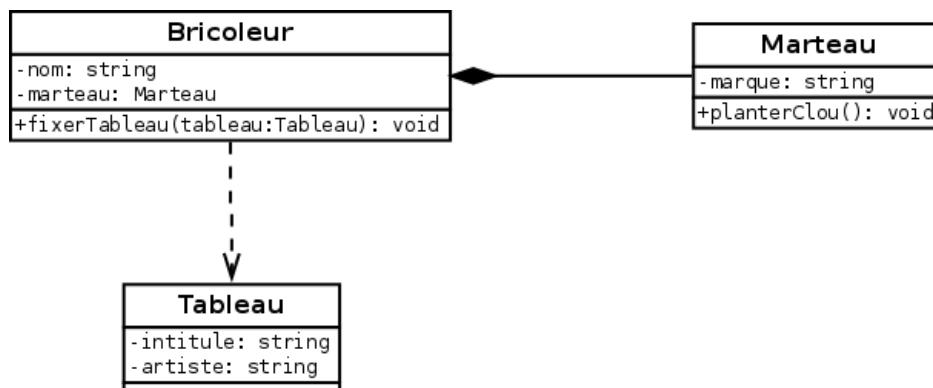
En POO, lorsque qu'une classe a besoin d'une autre classe pour effectuer un certain travail (on parle de dépendance), alors la première classe possède très souvent la deuxième en attribut d'instance. Par exemple, si la classe A est dépendante de la classe B, alors A possède un objet de type B en attribut d'instance.

Mais il est aussi possible de régler ce problème de dépendance d'une autre façon : en passant un objet en argument d'une méthode. Par exemple, dans une classe Truc, une méthode faireTruc() a besoin d'un objet de type Machin pour effectuer son traitement. Mais cette instance de Machin ne sera utilisée que par la méthode faireTruc() et nulle part ailleurs dans la classe . La méthode faireTruc() prendra donc en argument un objet de type Machin.

Si mon bricoleur fixe un tableau au mur, il n'aura vraisemblablement plus besoin de ce tableau une fois fixé. Par contre il aura encore besoin de son marteau pour fixer le tableau suivant. Un même objet marteau pourra servir plusieurs fois, il est donc intéressant de le garder en attribut d'instance. Par contre, à chaque fois que mon bricoleur fixera un tableau, l'objet tableau sera différent, il est ici inutile de garder en attribut d'instance les tableaux fixés, car le bricoleur ne se réservera pas des tableaux une fois ces derniers au mur.

Nous aurons donc une classe Marteau possédant un attribut d'instance marque de type string, et une méthode planterClou() qui affichera une chaîne de caractères "Clou planté !". Une classe Bricoleur possédant un attribut nom de type string, un attribut marteau de type Marteau, et une méthode fixerTableau() qui prendra en argument un objet de type Tableau, et qui affichera une chaîne de caractères "Le tableau '*intitulé du tableau*' est fixé !". Et enfin une classe Tableau possédant deux attributs d'instance de type string : intitule et auteur.

Ce qui nous donne ceci en UML :



En UML, l'association représente une composition. Ici cela signifie que l'objet Bricoleur est "composé" d'un objet Marteau. Nous pourrions aussi dire que l'objet Bricoleur possède un objet Marteau. Il est important de noter que cette relation est une relation forte et exclusive : un marteau ne pourra appartenir qu'à un seul Bricoleur.

L'association représente une dépendance, cela signifie ici que l'objet Bricoleur va avoir besoin, à un moment donné, d'un objet de type Tableau. Pour nous ce sera dans la méthode fixerTableau().

En PHP, cela donnerait :

```
1 <?php
2
3 // Tableau.php
4
5 class Tableau {
6
7     private $intitule;
8     private $artiste;
9
10    function __construct($intitule, $artiste) {
11        $this->intitule = $intitule;
12        $this->artiste = $artiste;
13    }
14
15    public function getIntitule() {
16        return $this->intitule;
17    }
18
19    public function getArtiste() {
20        return $this->artiste;
21    }
22
23    public function setIntitule($intitule) {
24        $this->intitule = $intitule;
25    }
26
27    public function setArtiste($artiste) {
28        $this->artiste = $artiste;
29    }
30
31 }
```

```
1 <?php
2
3 // Marteau.php
4
5 class Marteau {
6
7     private $marque;
8
9     function __construct($marque) {
10        $this->marque = $marque;
11    }
12
13    public function planterClou() {
14        echo 'Clou planté !';
15    }
16
17    public function getMarque() {
18        return $this->marque;
19    }
20
21    public function setMarque($marque) {
22        $this->marque = $marque;
23    }
24
25 }
```

```
1 <?php
2
3 // Bricoleur.php
4
5 class Bricoleur {
6
7     private $nom;
```

```

8     private $marteau;
9
10    function __construct($nom, $marteau) {
11        $this->nom = $nom;
12        $this->marteau = $marteau;
13    }
14
15    public function fixerTableau($tableau) {
16        $this->marteau->planterClou();
17        echo " Tableau '{$tableau->getIntitule()}' est fixé !";
18    }
19
20    public function getNom() {
21        return $this->nom;
22    }
23
24    public function getMarteau() {
25        return $this->marteau;
26    }
27
28    public function setNom($nom) {
29        $this->nom = $nom;
30    }
31
32    public function setMarteau($marteau) {
33        $this->marteau = $marteau;
34    }
35
36}

```

```

1 <?php
2
3 // lanceur.php
4
5 include './Tableau.php';
6 include './Marteau.php';
7 include './Bricoleur.php';
8
9 $memoire = new Tableau("La mémoire", "Magritte");
10 $marteau = new Marteau('Truc');
11 $bricolo = new Bricoleur("Toto", $marteau);
12 $bricolo->fixerTableau($memoire);

```

Lorsque l'on donne à un objet un autre objet dont il a besoin, on dit que l'on satisfait une dépendance. Dans le lanceur, lorsque j'appelle le constructeur de Bricoleur et que je lui passe en argument un objet de type Marteau, j'effectue ce que l'on nomme une "injection de dépendance". En somme, je donne à l'instance de Bricoleur l'instance de Marteau dont elle a besoin, et je la lui donne via le constructeur.

Ici c'est donc le constructeur qui me permet d'injecter la dépendance, mais il y a bien d'autres solutions...

Fin

NAVIGATION



Accueil

- Ma page

Pages du site

Mon profil

Cours actuel

POO PHP

Participants

Généralités

La programmation orientée objet : premiers pas

Introduction

- [!\[\]\(e71c4d7962cad8b9fc722e0bffccd835_img.jpg\) Classes et objets](#)
- [!\[\]\(fa7352425c1444866f6f947dacc0b460_img.jpg\) T.P. première classe](#)
- [!\[\]\(c3b1b4ad2170592f7113a986889c9cc7_img.jpg\) Les classes en PHP](#)
- [!\[\]\(a31670dfbb8af1101a4926c6c51f86bc_img.jpg\) T.P. Personne](#)
- [!\[\]\(839197710b9835cb80e6469ebe05a3dd_img.jpg\) Le constructeur et this](#)
- [!\[\]\(8a784664d6c10da9fbb4b9a193e0f823_img.jpg\) T.P. Personne v2](#)
- [!\[\]\(f99541cec2368d08ae93f13e55bc5f6c_img.jpg\) Accesseurs et mutateurs](#)
- [!\[\]\(15f0fe7ef8c29154e957efce17dbf7e9_img.jpg\) Les méthodes magiques](#)
- [!\[\]\(3a70f276b4294782c74a04fe87830877_img.jpg\) T.P. Personne v3](#)
- [!\[\]\(1b171644f98f32118e046fa6b283dfa1_img.jpg\) Les constantes de classe](#)
- [!\[\]\(924ed0e2a615aae2c6380b28af6b5e44_img.jpg\) Le modificateur static](#)
- [!\[\]\(c1fe12bb62869bb241bddab8c7145b8d_img.jpg\) T.P. Personne v4](#)
- [!\[\]\(51ed0680a9da061f0a40260345871d3d_img.jpg\) Une Personne et une Adresse : la relation has-a](#)
- [!\[\]\(e4c29f5ed42bef50c53ae30bd3c898c4_img.jpg\) Histoire de références](#)
- [!\[\]\(13bdc1c2ab01d5207a834db314023c1c_img.jpg\) T.P. formation](#)
- [!\[\]\(671e37de37e53ec85ad428f7ddfcb5de_img.jpg\) T.P. formation v2](#)
- [!\[\]\(c532383f19e044eb5fa1953e0932119a_img.jpg\) Une Personne et plusieurs Adresse\(s\)](#)
- [!\[\]\(fd36db79d3da2253bdd21aa488a78769_img.jpg\) T.P. formation v3](#)
- [!\[\]\(20a38cab68ee35990bdfbf1ca405b99a_img.jpg\) **Bricolage et dépendance**](#)
- L'héritage
- Les interfaces
- Le typage
- Les namespaces
- Les exceptions
- Les bases de données avec PDO
- Les tests avec PHPUnit
- Petite application version 2
- Petitie application version 3

Mes cours

ADMINISTRATION



[Administration du cours](#)

[Réglages de mon profil](#)

Connecté sous le nom « [Arnaud Lemais](#) » ([Déconnexion](#))
[POO PHP](#)