



# 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.

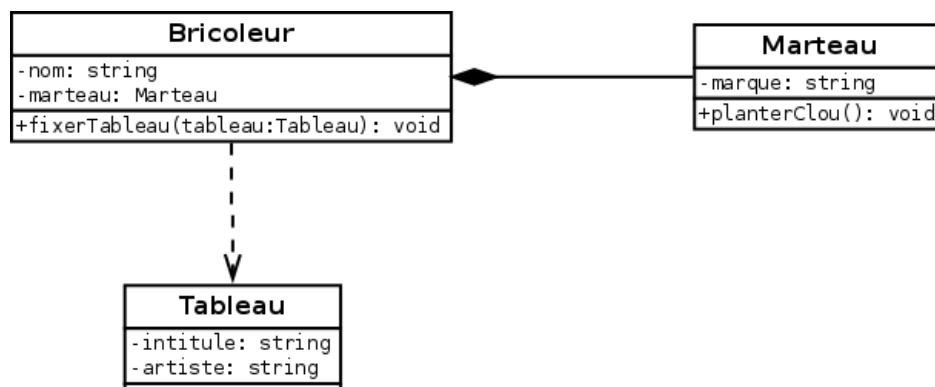
En POO, lorsque qu'une classe a besoin d'une autre classe pour effectuer un certain travail (on parle de dépendance), alors la première classe possède très souvent la deuxième en attribut d'instance. Par exemple, si la classe A est dépendante de la classe B, alors A possède un objet de type B en attribut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

-  [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](#)