



# Programmation orientée objet en PHP

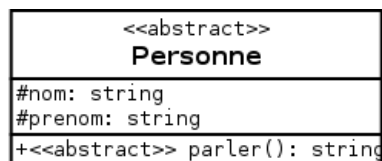
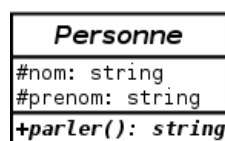
[Accueil](#) ▶ [Mes cours](#) ▶ [Développement logiciel](#) ▶ [POO PHP](#) ▶ [L'héritage](#) ▶ [Les classes abstraites](#)

## Les classes abstraites

Une [classe abstraite](#) est une classe qui ne servira qu'à l'héritage, on ne pourra pas instancier cette classe. Le rôle de la classe abstraite est principalement d'être une super-classe et de créer une classe "socle" à l'héritage. Une classe abstraite peut très bien hériter d'une autre classe, et posséder des attributs et des méthodes public, protected et même private.

En PHP, pour indiquer qu'une classe est abstraite, nous utilisons le mot clé "abstract" dans la déclaration de la classe.

Dans un diagramme de classe, les classes abstraites peuvent être représentées des deux façons :



Les noms des classes et des méthodes abstraites peuvent être écrits en italique ou porter le "stéréotype" abstract ("**<<abstract>>**").

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

Maintenant que nous avons rendu cette classe abstraite, nous ne pouvons plus écrire ceci :

```
1 <?php
2
3 // lanceur.php
4 include './Personne.php';
5 $pers = new Personne('Wayne', 'Bruce');
```

Par contre nous pouvons toujours utiliser cette classe dans le cadre de l'héritage :

```
1 <?php
2
3 // Formateur.php
4 class Formateur extends Personne {
5
6     private $specialite;
7
8     function __construct($nom, $prenom, $specialite) {
9         parent::__construct($nom, $prenom);
10        $this->specialite = $specialite;
11    }
12
13    public function getSpecialite() {
14        return $this->specialite;
15    }
16
17    public function setSpecialite($specialite) {
18        $this->specialite = $specialite;
19    }
20
21    public function enseigner() {
22        return 'j\'enseigne';
23    }
24
25    public function parler() {
26        $msg = parent::parler();
27        return "$msg le PHP est un langage...";
28    }
29
30 }
```

lanceur.php

```
1 <?php
2
3 // lanceur.php
4 include './Personne.php';
5 include './Formateur.php';
6
7 $formateur = new Formateur('Sparrow', 'Jack', 'développement logiciel');
8 echo $formateur->parler();
```

Dans une classe abstraite, il est aussi possible de déclarer des méthodes abstraites. Ces méthodes ne comporteront pas d'implémentation (de code) dans la classe abstraite qui les déclare, mais elles devront forcément être implémentées dans les classes filles non-abstraites.

Personne.php (modification de la méthode parler())

```
1 <?php
2
3 // Personne.php
4 abstract class Personne {
5
6     protected $nom;
```

```

7     protected $prenom;
8
9     function __construct($nom, $prenom) {
10         $this->nom = $nom;
11         $this->prenom = $prenom;
12     }
13
14     public function getNom() {
15         return $this->nom;
16     }
17
18     public function getPrenom() {
19         return $this->prenom;
20     }
21
22     public function setNom($nom) {
23         $this->nom = $nom;
24     }
25
26     public function setPrenom($prenom) {
27         $this->prenom = $prenom;
28     }
29
30     public abstract function parler();
31 }

```

La méthode abstraite ne contient pas d'implémentation et donc pas de bloc {...}. Seuls les noms de la méthode et de ses paramètres sont indiqués, on appelle parfois cela la signature de la méthode.

Dans la classe Formateur nous ne pouvons plus faire appel à la méthode de la super-classe puisque la méthode de la super-classe n'a plus d'implémentation. La classe Formateur doit donc fournir une implémentation à la méthode parler(), ou devenir elle-même une classe abstraite.

Formateur.php (modification de la méthode parler())

```

1  <?php
2
3  // Formateur.php
4  class Formateur extends Personne {
5
6      private $specialite;
7
8      function __construct($nom, $prenom, $specialite) {
9          parent::__construct($nom, $prenom);
10         $this->specialite = $specialite;
11     }
12
13     public function getSpecialite() {
14         return $this->specialite;
15     }
16
17     public function setSpecialite($specialite) {
18         $this->specialite = $specialite;
19     }
20
21     public function enseigner() {
22         return 'j\'enseigne';
23     }
24
25     public function parler() {
26         return 'je parle comme un formateur...';
27     }
28
29 }

```

Le lanceur ne change pas.

Fin

## NAVIGATION



### Accueil

#### ■ Ma page

Pages du site

Mon profil

Cours actuel

#### POO PHP


Participants

Généralités


La programmation orientée objet : premiers pas

L'héritage

 [Introduction](#)

 [T.P. héritage](#)


 [Redéfinition de méthodes](#)

 [T.P. redéfinition](#)

 [Les classes abstraites](#)

 [T.P. véhicule](#)

 [Le mot clé final](#)

 [T.P. employés](#)

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](#)