



PHP

PRÉSENTATION GÉNÉRALE



Aurélien Delorme
OCT 2021



Les interfaces

Les méthodes magiques

La méthode magique get

La méthode magique set

La méthode magique isset

La méthode magique unset

Principe de sérialisation / désérialisation

La méthode magique sleep

La méthode magique wakeup



Les interfaces

```
<?php
interface CrudInterface {
    public function add($element);
    public function remove($id);
    public function edit($element);
    public function getOne($id);
    public function getAll();
}
```

```
<?php
require "CrudInterface.php";
require "OrdinateurCrud.php";

$crudOrdi = new OrdinateurCrud();
```

Liste les **méthodes** que **doit implémenter une classe**.

Ne contient **que la déclaration des méthodes**

Classe totalement abstraite. On ne peut pas l'instancier.

Ne peut pas avoir le même nom qu'une classe

Une interface ne peut pas lister de méthodes abstraites ou finales.

Elle doivent être ajoutés dans le programme via le même procédé qu'une classe

Une interface doit être requise avant la classe qui l'utilise.

Ici, OrdinateurCrud implémente CrudInterface. L'interface CrudInterface est donc importée avant OrdinateurCrud

Ne fonctionne pas !

```
<?php
class OrdinateurCrud implements CrudInterface {
    // ...
}
```

(!) Fatal error: Class OrdinateurCrud contains 5 abstract methods and must therefore be declared abstract or implement the remaining methods (CrudInterface::add, CrudInterface::remove, CrudInterface::edit, ...) in /var/www/html/php-backpack/cours_poo/OrdinateurCrud.php on line 2			
Call Stack			
#	Time	Memory	Function
1	0.0022	415192	{main}()
2	0.0073	417696	require('/var/www/html/php-backpack/cours_poo/OrdinateurCrud.php')

Les interfaces

```
<?php
interface CrudInterface {
    public function add($element);
    public function remove($id);
    public function edit($element);
    public function getOne($id);
    public function getAll();
}
```

```
<?php
class OrdinateurCrud implements CrudInterface {
    public function add($element){...}
    public function remove($id){...}
    public function edit($element){...}
    public function getOne($id){...}
    public function getAll(){...}
    public function findAll(){...}
    public function findOne($id){...}
    public function delete($id){...}
}
```



Pour répondre au contrat proposé par notre interface CrudInterface, nous devons impérativement avoir les méthodes suivante dans notre classe :

- add prenant un paramètre
- remove prenant 1 paramètre
- Edit prenant un paramètre
- get one prenant un paramètre
- getAll ne prenant pas de paramètre

Les interfaces

Une classe pourra implémenter **plusieurs interface**.

Une interface pourra hériter **d'une autre interface**. Une classe qui implémentera **l'interface enfant** devra aussi contenir les méthodes **décrites dans l'interface parente**.

Implémenter plusieurs interfaces

```
<?php
interface DbInterface {
    public function save();
    public function rollBack();
}
```

```
<?php
class OrdinateurCrud implements CrudInterface, DbInterface {
    public function add($element){...}

    public function remove($id){...}

    public function edit($element){...}

    public function getOne($id){...}

    public function getAll(){...}

    public function findAll(){...}

    public function findOne($id){...}

    public function delete($id){...}

    public function save(){...}

    public function rollBack(){...}
}
```

Héritage d'interfaces

```
<?php
interface CrudInterface extends DbInterface {
    public function add($element);
    public function remove($id);
    public function edit($element);
    public function getOne($id);
    public function getAll();
}
```



Erreurs et exceptions

Lancer une erreur avec PHP



```
<?php
throw new Exception( message: "Ici je décide qu'il y a une erreur !");
```

[!] Fatal error: Uncaught Exception: Ici je décide qu'il y a une erreur ! in /var/www/html/php-backpack/cours_poo/index.php on line 3				
[!] Exception: Ici je décide qu'il y a une erreur ! in /var/www/html/php-backpack/cours_poo/index.php on line 3				
Call Stack				
#	Time	Memory	Function	Location
1	0.0022		414328 {main}()	.../index.php:0

Utilisation de la classe Exception PHP

```
public function __construct($message = "", $code = 0, Throwable $previous = null) { }

/** Gets the Exception message ... */
#[Pure]
final public function getMessage() { }

/** Gets the Exception code ... */
#[Pure]
final public function getCode() { }

/** Gets the file in which the exception occurred ... */
#[Pure]
final public function getFile() { }

/** Gets the line in which the exception occurred ... */
#[Pure]
final public function getLine() { }

/** Gets the stack trace ... */
#[Pure]
final public function getTrace() { }

/** Returns previous Exception ... */
#[Pure]
final public function getPrevious() { }

/** Gets the stack trace as a string ... */
#[Pure]
final public function getTraceAsString() { }
```

- Sert à gérer les erreurs
- Objet qui contient un message d'erreur
- Contient un code
- Contient le fichier où a été lancée l'exception
- Contient la ligne du déclenchement de l'exception
- Contient la « stack trace » (le chemin menant à l'erreur fichier par fichier)

Erreurs et exceptions

try / catch

```
<?php

function inferieurA100($chiffre)
{
    if ($chiffre > 100)
    {
        throw new Exception( message: 'Le nombre doit être < 100');
    }

    return $chiffre;
}

// Nous allons essayer d'effectuer les instructions situées dans ce bloc.
try
{
    echo inferieurA100( chiffre: 12), '<br>';
    echo inferieurA100( chiffre: 120), '<br>';
}

// Si il y a une erreur, on passe dans notre bloc catch
catch (Exception $e){
    // J'affiche une instruction qui affiche l'erreur puis je continue ...
    var_dump($e);
    // Si j'avais lancé une exception, le script de serait arrêté.
    // throw $e;
}

// Le script continue
echo 'Fin du script';
```

Un bloc try doit obligatoirement avoir un bloc catch (code à exécuter en cas d'erreur) ou finally (code qui sera exécuté en cas d'erreur ou non)

Exemple : Test de connexion à une base de donnée
(Utilisation de PDO)

```
<?php

class DbManager{

    protected $bdd;
    private $host = 'mysql';
    private $dbName = 'forum';
    private $username = 'root';
    private $password = 'tiger';

    public function __construct()
    {
        try {
            $this->bdd = new PDO( 'dsn: 'mysql:host='.$this->host.';
                                dbname='.$this->dbName , $this->username, $this->password);
            $this->bdd->setAttribute( attribute: PDO::ATTR_ERRMODE, value: PDO::ERRMODE_EXCEPTION);
        }
        catch(PDOException $e) {
            throw $e;
        }
    }
}
```

Dès que l'on instanciera un nouvel objet DbManager, notre application essaiera de se connecter à notre base de donnée.

Si il y a une erreur, une exception de type PDOException est lancée (La classe PDOException hérite de la classe Exception).



Les méthodes magiques

Les méthodes magiques sont des méthodes qui sont **déclenchées en fonction d'évènement**.
Les méthodes magiques se reconnaissent car elle commencent par 2 underscores « __ »

Exemple :

__construct : Appelé quand on instance un nouvel objet (utilisation du mot clé new)

__destruct : Appelé quand notre objet est supprimé. Il est aussi appelé implicitement à la fin d'un script

Nous étudierons 6 méthodes magiques supplémentaires.

__get : Permet de détecter l'accession à un attribut inaccessible.

__set : Permet de détecter l'affectation d'une valeur à un attribut inaccessible.

__isset : Permet de détecter l'appel de la fonction PHP « isset » ou « empty » sur un attribut inaccessible.

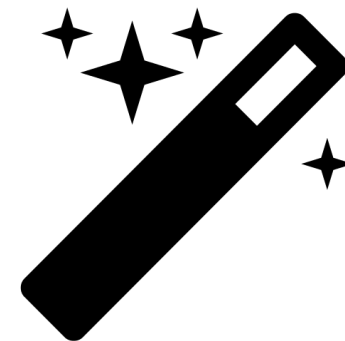
__unset : Permet de détecter l'appel de la fonction PHP « unset » sur un attribut inaccessible

__sleep : Appelée quand on utilise la fonction php « serialize » sur un objet

__wakeUp : Appelée quand on utilise la fonction « unserialize » sur une chaîne de caractère

Il en existe d'autres

<https://www.php.net/manual/fr/language.oop5.magic.php>



La méthode magique __get

Cart.php

```
<?php
class Cart {
    public function __get($name){
        var_dump($name);
    }
}
```

index.php

```
<?php
require "Cart.php";

$cart = new Cart();
$cart->attributQuiNexistePas;
```

Permet de détecter l'**accession à un attribut inaccessible** (private, protected, inexistant).

La méthode magique **__get(\$name)** prend un argument (**le nom de l'attribut auquel on a tenté d'accéder**)

Ici la classe Cart n'a pas d'attribut « attributQuiNexistePas » donc je passe dans ma méthode magique.

\$name aura donc pour valeur « attributQuiNexistePas »


```
/var/www/html/php-backpack/cours_poo/Cart.php:4:string 'attributQuiNexistePas' (length=21)
```



La méthode magique __set

 Cart.php

```
<?php
class Cart {
    public function __set($name, $value){
        var_dump( value: "Pourquoi essaies-tu de mettre de mettre la valeur de ".$value.
            " dans ".$name." ?");
    }
}
```

 index.php

```
<?php
require "Cart.php";

$cart = new Cart();
$cart->attributQuiNexistePas = 10;
```

Permet de détecter **l'affectation d'une valeur à un attribut inaccessible** (private, protected, inexistant).

La méthode magique __set(\$name, \$value) prend 2 arguments (**le nom de l'attribut auquel on a tenté d'accéder**)

Ici la classe Cart n'a pas d'attribut « attributQuiNexistePas » et je tente de lui assigner une valeur donc je passe dans ma méthode magique.

\$name aura donc pour valeur
« attributQuiNexistePas »
\$value aura donc pour valeur : 10

```
/var/www/html/php-backpack/cours_poo/Cart.php:5:string 'Pourquoi essaies-tu de mettre de mettre la valeur de 10 dans attributQuiNexistePas ?' (length=84)
```

La méthode magique __isset

 Cart.php

```
<?php
class Cart {
    public function __isset($name){
        var_dump( value: "Non, l'attribut ". $name." n'existe pas !");
    }
}
```

 index.php

```
<?php
require "Cart.php";

$cart = new Cart();
isset($cart->unAttributQuiNexistePas);

?>
```

Permet de détecter **l'appel de la fonction PHP isset sur un attribut inaccessible** (private, protected, inexistant).

La méthode magique __isset(\$name) prend 1 arguments(**le nom de l'attribut auquel sur lequel on a appelé la fonction isset**).

Ici la classe Cart n'a pas d'attribut « attributQuiNexistePas » et je tente d'appeler la fonction php isset sur cet attribut.

\$name aura donc pour valeur
« attributQuiNexistePas »


```
/var/www/html/php-backpack/cours_poo/Cart.php:4:string 'Non, l'attribut unAttributQuiNexistePas n'existe pas !' (length=54)
```



La méthode magique __unset

 Cart.php

```
<?php
class Cart {
    public function __unset($name){
        var_dump( value: "Tu supprimes, l'attribut ". $name." qui n'existe pas !");
    }
}
```

 index.php

```
<?php
require "Cart.php";

$cart = new Cart();
unset($cart->unAttributQuiNexistePas);
```

Permet de détecter **l'appel de la fonction PHP unset sur un attribut inaccessible** (private, protected, inexistant).

La méthode magique __unset(\$name) prend 1 arguments(**le nom de l'attribut auquel sur lequel on a appelé la fonction unset**).

Ici la classe Cart n'a pas d'attribut « attributQuiNexistePas » et je tente d'appeler la fonction php unset sur cet attribut.

\$name aura donc pour valeur « attributQuiNexistePas »

```
/var/www/html/php-backpack/cours_poo/Cart.php:4:string 'Non, l'attribut unAttributQuiNexistePas n'existe pas !' (length=54)
```



Pratique : Création d'un objet SessionManager

```
<?php
class Session
{
    private $attributs = [];

    public function __get($nom){
        return $this->attributs[$nom];
    }

    public function __isset($nom)
    {
        return isset($this->attributs[$nom]);
    }

    public function __unset($nom)
    {
        unset($this->attributs[$nom]);
    }

    public function __set($poids, $valeur){
        $this->attributs[$poids] = $valeur;
    }
}
?>
```

Ici, notre classe Session aura un tableau d'attributs (tableau vide à l'initialisation).

La méthode magique `__get` nous permettra d'obtenir une valeur dans ce tableau.

La méthode magique `__isset` nous permet de vérifier si un élément existe dans ce tableau.

La méthode magique `__unset` nous permet de supprimer un élément dans ce tableau.

La méthode magique `__set` nous permet d'ajouter un élément dans ce tableau.

Nous venons d'implémenter une gestion dynamique d'attributs !

Principe de sérialisation

```
<?php
require 'Session.php';
$session = new Session();
$session->firstname = 'Aurelien';
$session->lastname = 'DeLorme';

echo(serialize($session));
?>
```

Transforme un objet en chaîne de caractère

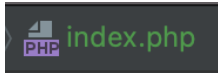
Permet de le stocker sous format textuel

Utilisation de la fonction PHP serialize

O:7:"Session":1:{s:18:"Sessionattributs";a:2:{s:9:"firstname";s:8:"Aurelien";s:8:"lastname";s:7:"Delorme";}}



Principe de désérialisation



```
<?php
require 'Session.php';
$session = new Session();
$session->firstname = 'Aurelien';
$session->lastname = 'Delorme';

$chaine = serialize($session);
var_dump($chaine);

$object = unserialize($chaine);
var_dump($object);
?>
```

- 1) Transforme notre objet en chaîne de caractère
- 2) Retransforme cette chaîne de caractère en objet

Transforme une chaîne de caractère en objet

Permet de récupérer un objet stocké en chaîne de caractère avant

Utilisation de la fonction PHP unserialize

```
/var/www/html/php-backpack/cours_poo/index.php:8:string 'O:7:"Session":1:{s:18:"@Session@attributs";a:2:{s:9:"firstname";s:8:"Aurelien";s:8:"lastname";s:7:"Delorme";}}' (length=110)
/var/www/html/php-backpack/cours_poo/index.php:11:
object(Session)[2]
  private 'attributs' =>
    array (size=2)
      'firstname' => string 'Aurelien' (length=8)
      'lastname' => string 'Delorme' (length=7)
```



La méthode magique sleep

 Session.php

```
<?php
class Session
{
    private $attributs = [];
    private $test = true;
    private $autreAttribut = 'hello';

    public function __sleep(){
        return ['attributs', 'autreAttribut'];
    }
}
```

 index.php

```
<?php
require 'Session.php';
$session = new Session();
$session->firstname = 'Aurelien';
$session->lastname = 'Delorme';

$chaine = serialize($session);
echo($chaine);

?>
```


La méthode magique sleep permet de nettoyer les objets avant leur serialisation.

Elle ne prend aucun paramètre

Si elle n'existe pas, tous les attributs seront sauvés

O:7:"Session":2:{s:18:"Sessionattributs";a:2:
{s:9:"firstname";s:8:"Aurelien";s:8:"lastname";s:7:"Delorme";};s:22:"SessionautreAttribut";s:5:"hello";}


La méthode magique wakeup

 index.php

```
<?php
require 'Session.php';
$session = new Session();
$session->firstname = 'Aurelien';
$session->lastname = 'Delorme';

$chaine = serialize($session);
var_dump($chaine);
$object = unserialize($chaine);
var_dump($object);

?>
```

 Session.php

```
<?php
class Session
{
    private $attributs = [];
    private $test = true;
    private $autreAttribut = 'hello';

    public function __wakeup(){
        echo('Je me reconstruit depuis une chaine de caractère !');
    }
}
```

La méthode magique `__wakeup` est déclenchée quand on appelle la fonction PHP `unserialize` sur un objet. Permet d'effectuer des actions lorsqu'un objet est recréé depuis une chaîne de caractère



Pratique : Création d'un objet SessionManager

```
<?php
class Session
{
    private $attributs = [];
    private $sleepingTime = null;

    public function __construct(){...}

    public function __sleep(){...}

    public function __wakeup(){...}

    public function __destruct(){...}

    public function __get($nom){...}

    public function __isset($nom){...}

    public function __unset($nom){...}

    public function __set($key, $valeur){...}
}
?>
```

Le but de cet objet sera de gérer une session HTTP.

On pourra :

- Ajouter un attribut en session
- Récupérer un attribut en session
- Vérifier si un attribut est en session

Bonus :

- Un calcul du délai depuis lequel la session HTTP n'a pas été demandé



Pratique : Création d'un objet SessionManager

```
<?php
class Session
{
    private $attributs = [];
    private $sleepingTime = null;

    public function __construct(){
        if(array_key_exists( key: 'serialized', $_SESSION)){
            $session = unserialize($_SESSION['serialized']);
            $this->attributs = $session->attributs;
            $this->sleepingTime = $session->sleepingTime;
        }
    }
}
```

```
public function __get($nom){
    return $this->attributs[$nom];
}

public function __set($key, $valeur){
    $this->attributs[$key] = $valeur;
}
```

```
public function __destruct(){
    $_SESSION['serialized'] = serialize($this);
}
```

```
public function __sleep()
{
    $this->sleepingTime = time();
    return ['attributs', 'sleepingTime'];
}
```

```
public function __wakeup(){
    $this->sleepingTime = time() - $this->sleepingTime;
}
```

```
public function __isset($nom)
{
    return isset($this->attributs[$nom]);
}
```

```
public function __unset($nom)
{
    unset($this->attributs[$nom]);
}
```



Echanges / Questions

