

Chapitre 3

Utiliser les fonctions PHP

1. Préambule

L'objectif de ce chapitre est de présenter les fonctions les plus utiles dans le cadre du développement d'un site web.

PHP propose de nombreuses fonctions ; la description de chaque fonction est accessible en ligne sur le site www.php.net.

● Version 8

Depuis la **version 8**, il est possible de passer des paramètres à une fonction en utilisant le nom du paramètre au lieu de sa position. Cette fonctionnalité est présentée dans le chapitre Écrire des fonctions et des classes PHP, mais elle peut être utilisée pour les fonctions natives du langage PHP, et donc pour les fonctions présentées dans ce chapitre. Par contre, dans ce chapitre, les noms réels des paramètres des fonctions ne sont pas présentés (ils sont traduits) ; pour les connaître, consultez la documentation en ligne des fonctions.

Depuis la **version 8.1**, passer la valeur `NULL` à un paramètre qui n'est pas explicitement optionnel est déprécié et génère donc une alerte de niveau `E_DEPRECATED`.

Exemple

```
<?php
$x = null;
$n = strlen($x);
?>
```

Résultat

Deprecated: strlen(): Passing null to parameter #1 (\$string) of type string is deprecated in `/app/scripts/index.php` on line 3

2. Manipuler les constantes, les variables et les types de données

2.1 Constantes

PHP propose un certain nombre de fonctions utiles sur les constantes :

Nom	Rôle
defined	Indique si une constante est définie ou non.
constant	Retourne la valeur d'une constante.

defined

La fonction `defined` permet de savoir si une constante est définie ou non.

Syntaxe

booléen `defined(chaine nom)`

`nom` Nom de la constante.

La fonction `defined` retourne `TRUE` si la constante est définie et `FALSE` dans le cas contraire.

Exemple

```
<?php
// Tester si la constante CONSTANCE est définie.
$ok = defined('CONSTANCE');
if ($ok) {
    echo 'CONSTANCE est définie.<br />';
} else {
    echo 'CONSTANCE n'est pas définie.<br />';
};
// Définir la constante CONSTANCE
define('CONSTANCE','valeur de CONSTANCE');
// Tester si la constante CONSTANCE est définie.
$ok = defined('CONSTANCE');
if ($ok) {
    echo 'CONSTANCE est définie.<br />';
} else {
    echo 'CONSTANCE n'est pas définie.<br />';
};
?>
```

Résultat

CONSTANTE n'est pas définie.
CONSTANTE est définie.

constant

La fonction `constant` retourne la valeur d'une constante dont le nom est passé en paramètre.

Syntaxe

mixte `constant(chaine nom)`

Avec :

`nom` Nom de la constante.

Cette fonction est pratique pour récupérer la valeur d'une constante dont le nom n'est pas connu a priori.

Exemple

```
<?php
// définir le nom de la constante dans une variable
$nomConstante = 'AUTRE CONSTANTE';
// définir la valeur de la constante
define($nomConstante, 'valeur de AUTRE CONSTANTE');
// afficher la valeur de la constante
echo $nomConstante, ' = ', constant($nomConstante);
?>
```

Résultat

AUTRE CONSTANTE = valeur de AUTRE CONSTANTE

D'autres fonctions permettent de connaître le type d'une constante (cf. section Manipuler les constantes, les variables et les types de données - Types de données).

2.2 Variables

PHP propose un certain nombre de fonctions utiles sur les variables :

Nom	Rôle
<code>empty</code>	Indique si une variable est vide ou non.
<code>isset</code>	Indique si une ou plusieurs variables sont définies ou non.
<code>unset</code>	Supprime une ou plusieurs variables.
<code>var_dump</code>	Affiche des informations sur une ou plusieurs variables (type et valeur).

empty

La fonction `empty` permet de tester si une variable est vide ou non.

Syntaxe

```
booléen empty(mixte variable)
```

variable Variable à tester.

`empty` retourne `TRUE` si la variable est vide et `FALSE` dans le cas contraire.

Une variable est considérée comme vide si elle n'a pas été affectée ou si elle contient une chaîne vide (`""`), une chaîne égale à 0 (`"0"`), un 0, `NULL`, `FALSE` ou un tableau vide.

La fonction `empty` peut aussi être utilisée pour tester si une expression est vide ou non.

Exemple

```
<?php
// Test d'une variable non initialisée.
$est_vide = empty($variable);
echo '$variable non initialisé<br />';
if ($est_vide) {
    echo '=> $variable est vide.<br />';
} else {
    echo '=> $variable n\'est pas vide.<br />';
}
// Test d'une variable contenant une chaîne vide.
$variable = '';
$est_vide = empty($variable);
echo '$variable = \'' . $variable . '<br />';
if ($est_vide) {
    echo '=> $variable est vide.<br />';
} else {
    echo '=> $variable n\'est pas vide.<br />';
}
// Test d'une variable contenant une chaîne égale à 0.
$variable = '0';
$est_vide = empty($variable);
echo '$variable = \'' . $variable . '<br />';
if ($est_vide) {
    echo '=> $variable est vide.<br />';
} else {
    echo '=> $variable n\'est pas vide.<br />';
}
// Test d'une variable contenant 0.
$variable = 0;
$est_vide = empty($variable);
echo '$variable = ' . $variable . '<br />';
```

Chapitre 3

```
if ($est_vide) {
    echo '=> $variable est vide.<br />';
} else {
    echo '=> $variable n\'est pas vide.<br />';
}
// Test d'une variable contenant une chaîne non vide.
$variable = 'x';
$est_vide = empty($variable);
echo '$variable = \''.$variable.'\''<br />';
if ($est_vide) {
    echo '=> $variable est vide.<br />';
} else {
    echo '=> $variable n\'est pas vide.<br />';
}
?>
```

Résultat

```
$variable non initialisé
=> $variable est vide.
$variable = ''
=> $variable est vide.
$variable = '0'
=> $variable est vide.
$variable = 0
=> $variable est vide.
$variable = 'x'
=> $variable n'est pas vide.
```

isset

La fonction `isset` permet de tester si une ou plusieurs variables sont définies ou non.

Syntaxe

```
booléen isset(mixte variable[, ...])
```

`variable` Variable à tester (éventuellement plusieurs, séparées par une virgule).

`isset` retourne `TRUE` si la variable est définie et `FALSE` dans le cas contraire.

Si plusieurs paramètres sont fournis, la fonction retourne `TRUE` uniquement si toutes les variables sont définies.

Une variable est considérée comme non définie si elle n'a pas été affectée ou si elle contient `NULL`. À la différence de la fonction `empty`, une variable qui contient une chaîne vide (`""`), une chaîne égale à 0 (`"0"`), un 0, un `FALSE` ou un tableau vide, n'est pas considérée comme non définie.

Exemple

```
<?php
// Test d'une variable non initialisée.
$est_définie = isset($variable);
echo '$variable non initialisé<br />';
if ($est_définie) {
    echo '=> $variable est définie.<br />';
} else {
    echo '=> $variable n\'est pas définie.<br />';
}
// Test d'une variable contenant une chaîne vide.
$variable = '';
$est_définie = isset($variable);
echo '$variable = \''<br />';
if ($est_définie) {
    echo '=> $variable est définie.<br />';
} else {
    echo '=> $variable n\'est pas définie.<br />';
}
// Test d'une variable contenant une chaîne égale à 0.
$variable = '0';
$est_définie = isset($variable);
echo '$variable = \''<br />';
if ($est_définie) {
    echo '=> $variable est définie.<br />';
} else {
    echo '=> $variable n\'est pas définie.<br />';
}
// Test d'une variable contenant 0.
$variable = 0;
$est_définie = isset($variable);
echo '$variable = \''<br />';
if ($est_définie) {
    echo '=> $variable est définie.<br />';
} else {
    echo '=> $variable n\'est pas définie.<br />';
}
// Test d'une variable contenant une chaîne non vide.
$variable = 'x';
$est_définie = isset($variable);
echo '$variable = \''<br />';
if ($est_définie) {
    echo '=> $variable est définie.<br />';
} else {
    echo '=> $variable n\'est pas définie.<br />';
}
?>
```

Chapitre 3

Résultat

```
$variable non initialisé  
=> $variable n'est pas définie.  
$variable = ''  
=> $variable est définie.  
$variable = '0'  
=> $variable est définie.  
$variable = 0  
=> $variable est définie.  
$variable = 'x'  
=> $variable est définie.
```

unset

La fonction `unset` permet de supprimer une ou plusieurs variables.

Syntaxe

```
unset(mixte variable[, ...])
```

`variable` Variable à supprimer (éventuellement plusieurs, séparées par une virgule).

Après suppression, la variable se trouve dans le même état que si elle n'avait jamais été affectée. L'utilisation de la fonction `isset` sur une variable supprimée retourne `FALSE` notamment.

Exemple

```
<?php  
// Définir une variable.  
$variable = 1;  
// Afficher la variable et tester si elle est définie.  
$est_définie = isset($variable);  
echo '$variable = ', $variable, '<br />';  
if ($est_définie) {  
    echo '=> $variable est définie.<br />';  
} else {  
    echo '=> $variable n\'est pas définie.<br />';  
}  
// Supprimer la variable.  
unset($variable);  
// Afficher la variable et tester si elle est définie.  
$est_définie = isset($variable);  
echo '$variable = ', $variable??'', '<br />';  
if ($est_définie) {  
    echo '=> $variable est définie.<br />';  
} else {  
    echo '=> $variable n\'est pas définie.<br />';  
}  
?>
```

Résultat

```
$variable = 1
=> $variable est définie.
$variable =
=> $variable n'est pas définie.
```

■ Remarque

Affecter un 0 ou une chaîne vide à une variable ne la supprime pas.

var_dump

La fonction `var_dump` affiche des informations sur une ou plusieurs variables (type et contenu).

Syntaxe

```
var_dump(mixte variable[, ...])
```

`variable` Variable à afficher (éventuellement plusieurs, séparées par une virgule).

La fonction `var_dump` est surtout intéressante lors des phases de mise au point.

Exemple

```
<?php
// afficher les informations sur une variable non initialisée
$variable = NULL;
var_dump($variable);
// initialiser la variable avec un nombre entier
$variable = 10;
// afficher les informations sur la variable
echo '<br />';
var_dump($variable);
// modifier la valeur (et le type) de la variable
$variable = 3.14; // nombre décimal
// afficher les informations sur la variable
echo '<br />';
var_dump($variable);
// modifier la valeur (et le type) de la variable
$variable = 'abc'; // chaîne de caractères
// afficher les informations sur la variable
echo '<br />';
var_dump($variable);
?>
```

Chapitre 4

Le design pattern Abstract Factory

1. Description

Le but du design pattern `Abstract Factory` est la création d'objets regroupés en familles sans avoir à connaître leurs classes concrètes.

2. Exemple

Le système de vente de véhicules gère des véhicules fonctionnant à l'essence et des véhicules fonctionnant à l'électricité. Cette gestion est confiée à l'objet `Catalogue`, à qui incombe la responsabilité de créer de tels objets.

Pour chaque produit, nous disposons d'une classe abstraite, d'une sous-classe concrète décrivant la version du produit fonctionnant à l'essence et d'une sous-classe concrète décrivant la version du produit fonctionnant à l'électricité. Par exemple, à la figure 4.1, pour l'objet `scooter`, il existe une classe abstraite `Scoter` et deux sous-classes concrètes : `ScoterElectricite` et `ScoterEssence`.

L'objet `Catalogue` peut utiliser ces sous-classes concrètes pour instancier les produits. Cependant, si par la suite de nouvelles familles de véhicules doivent être prises en compte (diesel ou hybride essence-électricité), les modifications à apporter à l'objet `Catalogue` peuvent s'avérer assez fastidieuses.

Le design pattern `Abstract Factory` résout ce problème en introduisant une interface `FabriqueVehiculeInterface` qui contient la signature des méthodes à utiliser pour créer chaque produit. Le type de retour de ces méthodes est constitué par l'une des classes abstraites de produit. Ainsi, l'objet `Catalogue` n'a pas besoin de connaître les sous-classes concrètes et reste parfaitement indépendant des familles de produits. En révélant l'interface de nos fabriques et non leur implémentation, nous découplons le code client des produits concrets : notre objet client `Catalogue` demande des produits à la fabrique qu'on lui passe en paramètre lors de sa construction sans avoir la moindre idée de qui elle est ni de ce qui se passe en coulisses pour qu'il obtienne le bon produit.

Une classe implémentant `FabriqueVehiculeInterface` est créée pour chaque famille de produits, à savoir les classes `FabriqueVehiculeElectricite` et `FabriqueVehiculeEssence`. Une telle classe a la responsabilité d'implémenter les opérations de création du véhicule appropriée pour la famille à laquelle elle est associée.

L'objet client `Catalogue` prend alors pour paramètre un objet se conformant à l'interface `FabriqueVehiculeInterface`, c'est-à-dire soit une instance de `FabriqueVehiculeElectricite`, soit une instance de `FabriqueVehiculeEssence`. Avec une telle instance, le catalogue peut créer et manipuler des véhicules sans devoir connaître les familles de véhicules et les classes concrètes correspondantes.

L'ensemble des classes du design pattern Abstract Factory pour cet exemple est détaillé à la figure 4.1.

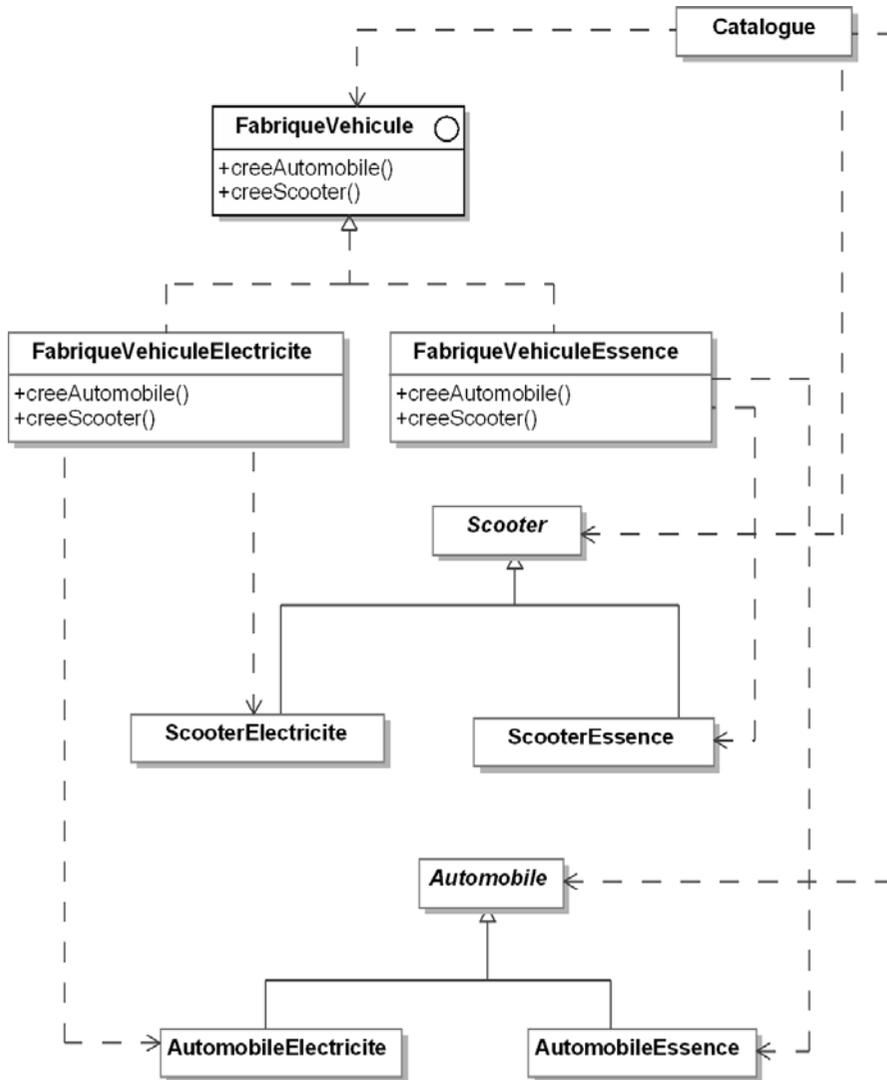


Figure 4.1 - Le design pattern Abstract Factory appliqué à des familles de véhicules

3. Structure

3.1 Diagramme de classes

La figure 4.2 détaille la structure générique du design pattern Abstract Factory.

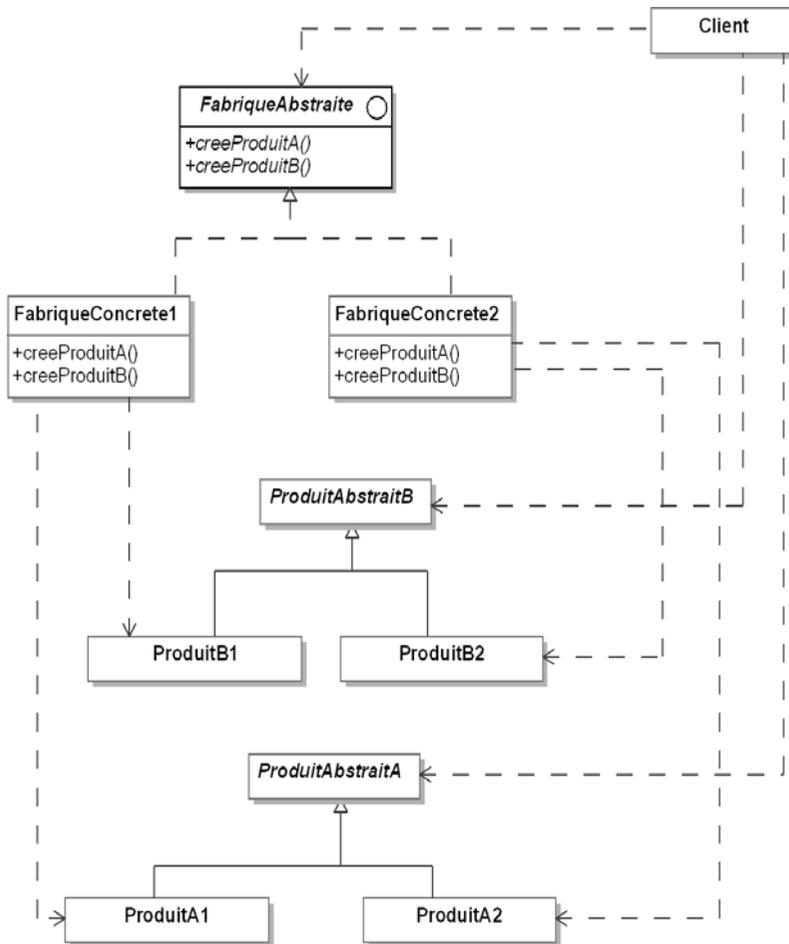


Figure 4.2 - Structure du design pattern Abstract Factory

3.2 Participants

Les participants au design pattern Abstract Factory sont les suivants :

- FabriqueAbstraite (FabriqueVehiculeInterface) est une interface spécifiant les signatures des méthodes créant les différents produits.
- FabriqueConcretel, FabriqueConcrete2 (FabriqueVehicule-
Electricite, FabriqueVehiculeEssence) sont les classes concrètes implémentant les méthodes créant les produits pour chaque famille de produits. Connaissant la famille et le produit, elles sont capables de créer une instance du produit pour cette famille.
- ProduitAbstraitA et ProduitAbstraitB (AbstractScooter et AbstractAutomobile) sont les classes abstraites des produits indépendamment de leur famille. Les familles sont introduites dans leurs sous-classes concrètes.
- Client (Catalogue) est la classe qui utilise l'interface FabriqueAbstraite.

3.3 Collaborations

La classe Catalogue utilise une instance de l'une des fabriques concrètes pour créer ses produits au travers de l'interface exposée par FabriqueAbstraiteInterface.

■ Remarque

Il est recommandé de ne créer qu'une seule instance des fabriques concrètes, celle-ci pouvant être partagée par plusieurs clients. Nous verrons plus tard un design pattern capable de garantir qu'une seule instance d'une classe est disponible à l'exécution : Singleton.

4. Domaines d'utilisation

Le design pattern Abstract Factory est utilisé dans les domaines suivants :

- Un système utilisant des produits a besoin d'être indépendant de la façon dont ces produits sont créés et regroupés.
- Un système est paramétré par plusieurs familles de produits qui peuvent évoluer.

5. Exemple en PHP

Voici maintenant un exemple d'utilisation du design pattern écrit en PHP. Le code PHP correspondant à la classe abstraite AbstractAutomobile et ses sous-classes est donné à la suite. Il est très simple, il décrit les quatre propriétés des automobiles ainsi que la méthode afficheCaracteristiques qui permet de les afficher.

```
<?php
declare(strict_types=1);

namespace ENI\DesignPatterns\AbstractFactory;

abstract class AbstractAutomobile
{
    protected string $marque;

    protected string $couleur;

    protected int $puissance;

    protected float $espace;

    public function __construct(string $marque, string $couleur,
int $puissance, float $espace)
    {
        $this->marque = $marque;
        $this->couleur = $couleur;
        $this->puissance = $puissance;
    }
}
```

```
        $this->espace = $espace;
    }

    abstract public function afficheCaracteristiques(): void;
}

<?php
declare(strict_types=1);

namespace ENI\DesignPatterns\AbstractFactory;

class AutomobileElectricite extends AbstractAutomobile
{
    public function afficheCaracteristiques(): void
    {
        echo "Automobile électrique - marque: $this->marque"
            . ", couleur: $this->couleur"
            . ", puissance: $this->puissance"
            . ", espace: $this->espace" . PHP_EOL;
    }
}

<?php
declare(strict_types=1);

namespace ENI\DesignPatterns\AbstractFactory;

class AutomobileEssence extends AbstractAutomobile
{
    public function afficheCaracteristiques(): void
    {
        echo "Automobile à essence - marque: $this->marque"
            . ", couleur: $this->couleur"
            . ", puissance: $this->puissance"
            . ", espace: $this->espace" . PHP_EOL;
    }
}
}
```

Le code PHP correspondant à la classe abstraite `AbstractScooter` et ses sous-classes est donné à la suite. Il est similaire à celui des automobiles, à