

La programmation Orientée Objet : L'encapsulation et la visibilité

Table des matières

I. Contexte	3
II. L'encapsulation en PHP	3
A. Introduction.....	3
B. Encapsulation des propriétés et méthodes d'une classe.....	4
C. Utilisation des accesseurs (getters) et mutateurs (setters)	4
D. Exemples de mise en pratique de l'encapsulation en PHP	5
E. Exercice : Quiz	7
III. La visibilité en PHP	8
A. Introduction.....	8
B. Niveaux de visibilité : public, protected et private.....	9
C. Rôle de la visibilité dans l'encapsulation	9
D. Bonnes pratiques d'utilisation de la visibilité en PHP.....	10
E. Exercice : Quiz	11
IV. Essentiel	12
V. Auto-évaluation	12
A. Exercice	12
B. Test.....	13
Solutions des exercices	13

I. Contexte

Durée : 60 minutes

Pré-Requis : connaître les notions de base de la POO en PHP

Environnement de travail : un navigateur web, easyPHP et un éditeur de texte (type notepad++ ou Sublim) installés et configurés sur la machine de l'apprenant

Contexte

L'encapsulation et la visibilité sont deux principes fondamentaux de la Programmation Orientée Objet (POO) qui visent à améliorer la qualité et la maintenabilité du code. L'encapsulation consiste à regrouper les données et les méthodes associées dans une classe, et à limiter leur accès depuis l'extérieur. Cette technique permet de masquer la complexité interne d'un objet et de ne rendre disponibles que les fonctionnalités essentielles à son utilisation.

Concrètement, cela signifie que les données sont stockées dans des variables privées ou protégées, qui ne peuvent être accessibles qu'à travers des méthodes publiques spécialement conçues pour interagir avec ces données. Les méthodes publiques sont la porte d'entrée de l'objet et permettent aux utilisateurs de l'objet d'interagir avec lui, tout en garantissant la cohérence et l'intégrité des données stockées.

L'encapsulation et la visibilité sont étroitement liées, car l'utilisation de niveaux de visibilité appropriés permet de garantir que les données sont manipulées de manière sûre et cohérente dans une classe. Ensemble, l'encapsulation et la visibilité permettent de créer des classes plus sûres, plus modulaires et plus faciles à maintenir.

II. L'encapsulation en PHP

A. Introduction

L'encapsulation des propriétés et des méthodes d'une classe en PHP est un mécanisme qui permet de cacher les détails d'implémentation d'une classe à l'extérieur de celle-ci. L'objectif est d'assurer la cohérence et la sécurité des données stockées dans la classe, en limitant l'accès direct à ces données à travers des méthodes spéciales appelées « *accesseurs* » et « *mutateurs* ».

C'est un concept qui permet de limiter l'accès à certains éléments d'une classe, tels que ses attributs et méthodes. L'idée est de ne rendre accessibles que les éléments strictement nécessaires pour que la classe puisse être utilisée, tout en cachant les détails d'implémentation et les données internes. Cela permet de mieux contrôler l'utilisation de la classe, de garantir la cohérence des données et de prévenir les erreurs de manipulation.

Les propriétés encapsulées sont stockées dans des variables privées ou protégées, et ne sont pas directement accessibles en dehors de la classe. Les méthodes encapsulées sont également définies avec des niveaux de visibilité appropriés, qui limitent leur accès en fonction des besoins de la classe.

Les accesseurs sont des méthodes publiques qui permettent d'accéder aux propriétés encapsulées depuis l'extérieur de la classe. Ils sont généralement nommés « *getters* » et retournent la valeur de la propriété encapsulée. Les mutateurs sont des méthodes publiques qui permettent de modifier la valeur d'une propriété encapsulée. Ils sont généralement nommés « *setters* » et prennent une valeur en entrée pour modifier la propriété encapsulée.

L'encapsulation permet de masquer les détails d'implémentation de la classe, ce qui facilite la maintenance et la réutilisation du code. Les utilisateurs de la classe peuvent accéder aux propriétés encapsulées à travers des méthodes spéciales, sans avoir à connaître les détails d'implémentation de la classe. Cela garantit également que les données sont manipulées de manière cohérente et sûre, car les accesseurs et mutateurs peuvent inclure des contrôles de sécurité et de validation pour éviter les erreurs de manipulation de données.

B. Encapsulation des propriétés et méthodes d'une classe

Le principe d'encapsulation : avantages et inconvénients

L'encapsulation est un concept fondamental de la programmation orientée objet en PHP et de sa bonne mise en œuvre offre de nombreux avantages. L'un des plus importants est la garantie de l'intégrité de la structure d'une classe, en obligeant l'utilisateur à passer par des méthodes spécifiques pour modifier une donnée. En effet, l'accès direct aux attributs peut être risqué, car il peut compromettre la cohérence et la stabilité de l'ensemble du code. Par conséquent, l'encapsulation est considérée comme l'un des piliers de la POO avec l'héritage. Il reste un concept clé à maîtriser pour tout développeur PHP qui souhaite écrire des codes modulaires et évolutifs.

Le principe d'encapsulation est particulièrement important lors de la création d'interfaces modulables, telles que des sites web ou des modules pour des interfaces modulables telles que WordPress ou PrestaShop. En effet, la programmation orientée objet est souvent utilisée dans ce type de structures, car elle offre une grande modularité et facilite la maintenance grâce à la création de classes distinctes.

Cependant, il est essentiel de réfléchir soigneusement à qui peut avoir accès à chaque élément de chaque classe, pour garantir l'intégrité de la structure et éviter les conflits potentiels entre les propriétés et les méthodes de différentes classes. Par conséquent, la définition des niveaux de visibilité est cruciale pour contrôler l'accès aux éléments d'une classe, et garantir que seuls les éléments nécessaires sont accessibles de l'extérieur de la classe. Cela permet de protéger les données de la classe contre des modifications non autorisées et de prévenir les erreurs potentielles.

C. Utilisation des accesseurs (getters) et mutateurs (setters)

En POO, il est généralement préférable de ne pas accéder directement aux attributs d'un objet, mais de passer par des méthodes pour y accéder. Les méthodes qui permettent de lire les valeurs des attributs sont appelées accesseurs (ou getters), tandis que celles qui permettent de modifier ces valeurs sont appelées mutateurs (ou setters). Ces méthodes permettent de garantir l'encapsulation, en évitant l'accès direct aux propriétés de la classe.

Il est courant de suivre une convention de nommage qui consiste à nommer les méthodes en utilisant le nom de l'attribut correspondant, en ajoutant "get" pour les accesseurs et "set" pour les mutateurs. Cela permet une meilleure organisation du code et facilite la compréhension de celui-ci. L'utilisation d'accesseurs et de mutateurs peut également renforcer l'encapsulation, en permettant de contrôler l'accès aux attributs et en limitant les erreurs de manipulation des données.

Exemple

Supposons que nous avons une classe `Personne` avec une propriété privée `nom`. Nous pouvons utiliser un accesseur pour récupérer la valeur de la propriété `nom`, de cette façon :

```
1 class Personne {
2     private string $nom;
3
4     public function getNom() : string{
5         return $this->nom;
6     }
7 }
```

Dans cet exemple, la méthode `getNom()` permet d'obtenir la valeur de la propriété `$nom`. Nous pouvons également utiliser un mutateur pour modifier la valeur de cette propriété, de cette façon :

```
1 class Personne {
2     private string $nom;
3
4     public function setNom(string $nom) {
5         $this->nom = $nom;
6     }
7 }
```

Ici, la méthode `setNom()` permet de modifier la valeur de la propriété `$nom` en lui passant une nouvelle valeur.

L'utilisation d'accesseurs et de mutateurs peut avoir plusieurs avantages, notamment en offrant une meilleure encapsulation et en facilitant la validation et la manipulation des données. En outre, cela peut aider à éviter les erreurs de manipulation de données, en fournissant un contrôle supplémentaire sur la façon dont les données sont manipulées.

D. Exemples de mise en pratique de l'encapsulation en PHP

Exemple	Déclaration d'une classe
	<pre>1 class Utilisateur { 2 private string \$nom; 3 private string \$prenom; 4 private string \$email; 5 6 public function __construct(String \$nom,string \$prenom,string \$email) { 7 \$this->nom = \$nom; 8 \$this->prenom = \$prenom; 9 \$this->email = \$email; 10 } 11 12 public function getNom() :string { 13 return \$this->nom; 14 } 15 16 public function setNom(string \$nom) { 17 \$this->nom = \$nom; 18 } 19 20 public function getPrenom() :string { 21 return \$this->prenom; 22 } 23 24 public function setPrenom(string \$prenom) { 25 \$this->prenom = \$prenom; 26 } 27 28 public function getEmail():string { 29 return \$this->email; 30 } 31 32 public function setEmail(string \$email) { 33 \$this->email = \$email; 34 } 35 }</pre>

Dans cet exemple, la classe `Utilisateur` a des propriétés privées (nom, prénom, email) qui ne peuvent pas être accédées directement. Les accesseurs (getters) et mutateurs (setters) sont utilisés pour lire et modifier les valeurs des propriétés respectives.

Exemple Utilisation de la portée "protected"

```

1 class Animal {
2     protected string $nom;
3
4     public function setNom(string $nom) {
5         $this->nom = $nom;
6     }
7 }
8
9 class Chat extends Animal {
10     public function getNom() :string {
11         return $this->nom;
12     }
13 }
14
15 $monChat = new Chat();
16 $monChat->setNom("Minou");
17 echo $monChat->getNom(); // affiche "Minou"

```

Dans cet exemple, la propriété `nom` de la classe `Animal` est définie avec une portée "protected". Cela signifie que la propriété n'est pas accessible depuis l'extérieur de la classe, mais est accessible par les sous-classes (ici, la classe `Chat`). Les accesseurs et mutateurs ne sont pas nécessaires, car la propriété est accessible par la sous-classe.

Fondamental

En PHP, les méthodes magiques `__get()` et `__set()` permettent d'accéder et de modifier des propriétés privées et protégées d'une classe depuis l'extérieur de celle-ci.

La méthode `__get()` est appelée automatiquement lorsque l'on tente d'accéder à une propriété inexistante ou inaccessible depuis l'extérieur de la classe. Elle prend en paramètre le nom de la propriété demandée et doit retourner sa valeur. Cela permet d'accéder à une propriété protégée ou privée d'une classe sans enfreindre l'encapsulation.

Exemple Utilisation de `__get()`

```

1 class MyClass {
2     private $myProperty = 'value';
3
4     public function __get($property) {
5         if (property_exists($this, $property)) {
6             return $this->$property;
7         }
8     }
9 }
10
11 $obj = new MyClass();
12 echo $obj->myProperty; // Affiche 'value'

```

Fondamental

La méthode `__set()` est appelée automatiquement lorsque l'on tente de modifier une propriété inexistante ou inaccessible depuis l'extérieur de la classe. Elle prend en paramètre le nom de la propriété et sa nouvelle valeur. Cela permet de modifier une propriété protégée ou privée d'une classe sans enfreindre l'encapsulation.

Exemple Utilisation de `__set()`

```
1 class MyClass {
2     private $myProperty = 'value';
3
4     public function __set($property, $value) {
5         if (property_exists($this, $property)) {
6             $this->$property = $value;
7         }
8     }
9 }
10
11 $obj = new MyClass();
12 $obj->myProperty = 'new value';
13 echo $obj->myProperty; // Affiche 'new value'
```

Complément

Cependant, l'utilisation des méthodes magiques `__get()` et `__set()` peut avoir des impacts sur la performance de l'application, car elles sont appelées automatiquement à chaque tentative d'accès ou de modification de propriétés de l'objet. Il est donc important de bien les utiliser et de ne pas en abuser.

Exemple Utilisation de la méthode "__get()"

```
1 class Produit {
2     private $nom;
3     private $prix;
4
5     public function __get($propriete) {
6         if (property_exists($this, $propriete)) {
7             return $this->$propriete;
8         }
9     }
10
11     public function __set($propriete, $valeur) {
12         if (property_exists($this, $propriete)) {
13             $this->$propriete = $valeur;
14         }
15     }
16 }
17
18 $monProduit = new Produit();
19 $monProduit->nom = "Ordinateur";
20 $monProduit->prix = 1200;
21 echo $monProduit->nom; // affiche "Ordinateur"
22 echo $monProduit->prix; // affiche "1200"
```

Dans cet exemple, la méthode magique `__get()` est utilisée pour accéder aux propriétés privées `\$nom` et `\$prix` de la classe `Produit`. Cette méthode est appelée automatiquement lorsque l'on tente d'accéder à une propriété qui n'est pas accessible directement. La méthode magique `__set()` est également utilisée pour définir la valeur de la propriété. Cela permet d'encapsuler les propriétés tout en offrant un accès facile aux valeurs.

E. Exercice : Quiz

[solution n°1 p.15]

Question 1

Qu'est-ce que l'encapsulation en PHP ?

- ☐ Un moyen de cacher des données à l'utilisateur
- ☐ Une méthode pour rendre toutes les propriétés publiques
- ☐ Un concept qui n'existe pas en PHP

Question 2

Quel est l'intérêt de l'encapsulation en PHP ?

- ☐ Assurer l'intégrité de la structure d'une classe
- ☐ Faciliter la compréhension du code par l'utilisateur
- ☐ Permettre à l'utilisateur de modifier les données à sa guise

Question 3

L'encapsulation est un principe de la programmation orientée objet qui permet de masquer les détails d'implémentation d'une classe à l'utilisateur.

- ☐ Vrai
- ☐ Faux

Question 4

Que fait la méthode get dans l'encapsulation en PHP ?

- ☐ Elle permet de récupérer la valeur d'un attribut de la classe
- ☐ Elle permet de définir une propriété privée
- ☐ Elle permet de rendre une propriété publique

Question 5

Que fait la méthode set dans l'encapsulation en PHP ?

- ☐ Elle permet de récupérer la valeur d'une propriété privée
- ☐ Elle permet de définir un attribut de la classe
- ☐ Elle permet de rendre une propriété publique

III. La visibilité en PHP

A. Introduction

Définition

La visibilité permet de définir les niveaux d'accès aux propriétés et aux méthodes d'une classe. Il existe trois niveaux de visibilité en PHP : public, protected et private. La visibilité « *public* » permet à la propriété ou à la méthode d'être accessible depuis n'importe où dans le code. La visibilité protected permet à la propriété ou à la méthode d'être accessible dans la classe elle-même et dans les classes dérivées (classes enfants ou héritières). La visibilité private limite l'accès à la propriété ou à la méthode à la classe elle-même.

B. Niveaux de visibilité : public, protected et private

En PHP, il existe trois niveaux de visibilité pour les propriétés et les méthodes d'une classe :

1. **Public** : le niveau de visibilité public est le plus permissif, car les propriétés et les méthodes qui sont définies comme publiques sont accessibles depuis n'importe où dans le code, à la fois à l'intérieur et à l'extérieur de la classe. Cela signifie que n'importe quel objet peut accéder et modifier les propriétés publiques d'une classe, ou appeler ses méthodes publiques. Il est donc important de faire attention à ce que l'on expose publiquement, en ne définissant que les éléments qui sont nécessaires pour l'utilisation de la classe par les autres parties du code. En effet, l'utilisation excessive de propriétés et méthodes publiques peut rendre la classe plus difficile à maintenir et plus sujette aux erreurs et aux conflits de noms.
2. **Protected** : le niveau de visibilité protégé est intermédiaire entre public et privé. Les propriétés et les méthodes protégées sont accessibles uniquement à l'intérieur de la classe et de ses classes dérivées (ou héritées). Les classes qui héritent d'une classe avec des propriétés et des méthodes protégées peuvent y accéder comme si elles étaient publiques, mais les autres classes ne peuvent pas y accéder. Cela permet de restreindre l'accès aux propriétés et aux méthodes à des parties du code spécifiques, tout en permettant aux classes dérivées d'accéder à ces éléments pour étendre ou modifier leur fonctionnalité. Les niveaux de visibilité dits protégé sont donc souvent utilisés pour les méthodes et les propriétés qui sont nécessaires aux classes dérivées, mais qui ne devraient pas être accessibles à l'extérieur de la classe.
3. **Private** : le niveau de visibilité private est le plus restrictif et permet de limiter l'accès aux propriétés et méthodes de la classe à son propre contexte, c'est-à-dire qu'elles ne peuvent pas être appelées ou utilisées à l'extérieur de la classe, y compris par les classes dérivées. Ce niveau de visibilité permet de protéger les données et les comportements internes de la classe en les rendant inaccessibles à d'autres parties du code, sauf par le biais de méthodes publiques spécialement conçues pour y accéder. Cela permet de maintenir une encapsulation stricte et d'éviter les effets de bord indésirables qui pourraient résulter de l'interaction de parties du code non autorisées avec les propriétés et méthodes privées de la classe.

Les niveaux de visibilité en PHP permettent de contrôler l'accès aux propriétés et aux méthodes d'une classe, en limitant leur accessibilité aux parties de code qui en ont réellement besoin. En utilisant les niveaux de visibilité appropriés, on peut renforcer l'encapsulation des données et protéger la structure interne de la classe.

L'utilisation de niveaux de visibilité appropriés aide également à mieux organiser son code en limitant les interactions non désirées entre les différentes parties d'un programme. En effet, en définissant les propriétés et méthodes comme privées ou protégées, on peut s'assurer qu'elles ne sont accessibles que par les méthodes de la même classe ou par ses classes dérivées. Cela permet de réduire les risques d'erreur de programmation et de faciliter la maintenance du code en isolant les différentes parties de la classe. De plus, l'utilisation de niveaux de visibilité permet de rendre le code plus sécurisé. En limitant l'accès à certaines parties du code, on peut s'assurer que les données sensibles ne sont pas exposées à des parties non autorisées du programme.

C. Rôle de la visibilité dans l'encapsulation

Il est essentiel de comprendre que la portée des propriétés et des méthodes n'affecte pas leur fonctionnalité. En d'autres termes, la visibilité ne détermine pas si une propriété ou une méthode peut être utilisée ou non. Elle contrôle simplement l'accès à la propriété ou à la méthode à partir de l'extérieur de la classe. Ainsi, les propriétés et les méthodes peuvent toujours être appelées et utilisées par d'autres parties du code, tant que l'accès est autorisé. La portée des propriétés et des méthodes est utilisée principalement pour renforcer l'encapsulation des données et pour mieux organiser son code.

Exemple

Voici un exemple de déclaration de propriétés et méthodes avec différents niveaux de visibilité :

```
1 class MaClasse {
2     public $publicProp = 'Propriété publique';
3     protected $protectedProp = 'Propriété protégée';
4     private $privateProp = 'Propriété privée';
5
6     public function publicMethod() {
7         echo 'Méthode publique';
8     }
9
10    protected function protectedMethod() {
11        echo 'Méthode protégée';
12    }
13
14    private function privateMethod() {
15        echo 'Méthode privée';
16    }
17 }
```

L'utilisation des bonnes pratiques de visibilité en PHP est cruciale pour garantir l'intégrité de la structure d'une classe et éviter les erreurs de manipulation des données. En effet, en définissant des niveaux de visibilité appropriés pour les propriétés et les méthodes d'une classe, on peut s'assurer que les données sont manipulées de manière cohérente et sécurisée.

En général, il est recommandé de déclarer les propriétés comme privées et de fournir des méthodes publiques pour accéder et modifier les valeurs de ces propriétés, afin de garantir l'encapsulation. Les propriétés protégées sont généralement utilisées pour les éléments que les sous-classes doivent pouvoir accéder, mais qui ne devraient pas être accessibles à l'extérieur de la classe ou de ses sous-classes.

L'utilisation de niveaux de visibilité appropriés peut aider à améliorer la sécurité et la robustesse des programmes en empêchant l'accès non autorisé aux éléments de la classe, en limitant les effets de bord, en empêchant la modification de l'état de l'objet de manière inattendue, et en facilitant la maintenance et l'évolution du code.

D. Bonnes pratiques d'utilisation de la visibilité en PHP

L'utilisation appropriée des niveaux de visibilité est une pratique importante en programmation orientée objet, car elle permet de garantir l'intégrité et la sécurité de la structure de notre code. Voici quelques bonnes pratiques à suivre pour une utilisation efficace des niveaux de visibilité en PHP :

1. Utiliser le niveau de visibilité le plus restrictif possible : il est généralement conseillé de définir la visibilité des propriétés et des méthodes comme privée (private) par défaut et de les exposer uniquement si nécessaire. En effet, en limitant l'accès aux éléments de notre classe, on évite des erreurs potentielles et on garantit l'intégrité de notre code.
2. Éviter d'utiliser le niveau de visibilité public (public) de manière excessive : L'utilisation excessive du niveau public peut entraîner des risques d'erreurs et de conflits, ainsi que des problèmes de sécurité. Il est préférable d'utiliser des accesseurs (getters) et des mutateurs (setters) pour accéder aux propriétés de la classe.
3. Utiliser le niveau de visibilité protégé (protected) avec parcimonie : le niveau de visibilité protégé permet aux classes filles (héritage) d'accéder aux propriétés et aux méthodes de la classe parente. Cependant, l'utilisation excessive de ce niveau de visibilité peut entraîner une forte dépendance entre les classes et une difficulté à maintenir le code à long terme. Il est donc important de limiter son utilisation aux cas où cela est vraiment nécessaire.

4. Éviter de modifier les propriétés directement à l'extérieur de la classe : il est recommandé d'utiliser des accesseurs et des mutateurs pour lire et modifier les propriétés de la classe, plutôt que de les modifier directement à l'extérieur de la classe. Cela permet de mieux contrôler l'accès aux propriétés et de garantir leur intégrité.
5. Documenter les niveaux de visibilité dans le code : il est important de documenter clairement les niveaux de visibilité utilisés dans le code, afin que les développeurs qui travaillent sur le projet puissent comprendre les règles d'accès et d'utilisation de la classe. Les commentaires peuvent être utilisés pour expliquer la raison de l'utilisation de chaque niveau de visibilité.

En suivant ces bonnes pratiques, on peut garantir la sécurité, la fiabilité et la maintenabilité de notre code en utilisant correctement les niveaux de visibilité en PHP.

E. Exercice : Quiz

[solution n°2 p.16]

Question 1

Les propriétés et les méthodes déclarées comme `private` ne peuvent pas être accessibles en dehors de la classe.

- ☐ Vrai
- ☐ Faux

Question 2

En PHP, il est préférable de rendre toutes les propriétés et méthodes d'une classe publiques pour faciliter leur accès depuis l'extérieur de la classe.

- ☐ Vrai
- ☐ Faux

Question 3

Que signifie le niveau de visibilité `protected` en PHP ?

- ☐ Les propriétés et les méthodes marquées comme `protected` ne peuvent être accédées qu'à partir de la classe elle-même
- ☐ Les propriétés et les méthodes marquées comme `protected` peuvent être accédées à partir de n'importe où dans le code
- ☐ Les propriétés et les méthodes marquées comme `protected` ne peuvent être accédées qu'à partir de la classe elle-même et de ses sous-classes

Question 4

Quel est le niveau de visibilité par défaut en PHP ?

- ☐ `public`
- ☐ `protected`
- ☐ `private`

Question 5

Peut-on modifier le niveau de visibilité d'une propriété ou méthode en PHP ?

- ☐ Non, il est figé une fois défini
- ☐ Oui, il est possible de le modifier avec la méthode `setVisibility()`
- ☐ Oui, il est possible de le modifier en changeant simplement le mot-clé correspondant

IV. Essentiel

L'encapsulation et la visibilité sont des concepts fondamentaux de la programmation orientée objet en PHP et jouent un rôle crucial dans la création de programmes modulaires, maintenables et évolutifs.

En utilisant l'encapsulation, nous pouvons cacher les détails de l'implémentation de notre code et offrir une interface bien définie pour les utilisateurs. Cela facilite la compréhension du code et permet aux développeurs de travailler sur des parties distinctes du programme sans perturber le fonctionnement d'autres parties. De plus, l'encapsulation permet de garantir l'intégrité de la structure d'une classe, en forçant l'utilisateur à passer par un chemin prédéfini pour modifier une donnée.

La visibilité des propriétés et des méthodes en PHP joue également un rôle important dans la conception d'un code robuste et maintenable. En définissant les niveaux de visibilité des propriétés et des méthodes, nous pouvons contrôler l'accès aux données et aux fonctionnalités de notre programme.

Le niveau de visibilité `public` est généralement utilisé pour les méthodes et les propriétés qui doivent être accessibles à partir de l'extérieur de la classe. Le niveau de visibilité `protected` est utilisé pour les méthodes et les propriétés qui ne doivent être accessibles qu'à l'intérieur de la classe et des classes dérivées. Enfin, le niveau de visibilité `private` est utilisé pour les méthodes et les propriétés qui ne doivent être accessibles qu'à l'intérieur de la classe.

Il est important de suivre les bonnes pratiques d'utilisation de la visibilité en PHP, comme ne pas rendre publiques des propriétés ou des méthodes qui ne devraient pas l'être, utiliser les getters et setters pour accéder et modifier des propriétés protégées ou privées, et éviter de modifier les propriétés protégées ou privées directement à partir d'une classe dérivée.

En conclusion, l'encapsulation et la visibilité sont des concepts essentiels en programmation orientée objet en PHP. Ils permettent de créer des programmes modulaires, maintenables et évolutifs en offrant une interface bien définie pour les utilisateurs et en contrôlant l'accès aux données et aux fonctionnalités. Les bonnes pratiques d'utilisation de la visibilité doivent être suivies pour garantir l'intégrité du code et faciliter la collaboration entre les développeurs.

V. Auto-évaluation

A. Exercice

Vous êtes développeur web au sein d'une équipe de 10 personnes et devez réaliser une application de e-commerce pour l'entreprise BigBroZer en vue de permettre le fichage de personnes. Supposons que nous ayons une classe "Client" qui stocke des informations sur les clients d'une entreprise. La classe contient les propriétés "nom", "adresse", "email" et "numéro de téléphone". Nous voulons utiliser l'encapsulation pour protéger ces propriétés et permettre l'accès à ces propriétés uniquement via des méthodes publiques.

Question 1

[solution n°3 p.17]

Écrivez la classe "Client" en respectant les principes de visibilité et d'encapsulation, puis écrivez le code permettant d'instancier un objet de la classe "Client" avec les informations de votre choix et d'afficher ces informations, il vous est impossible d'utiliser uniquement les méthodes magiques.

Question 2

[solution n°4 p.18]

Pourquoi est-il important de limiter l'accès aux propriétés d'une classe ?

B. Test**Exercice 1 : Quiz**

[solution n°5 p.18]

Question 1

La visibilité "public" permet d'accéder à une propriété ou une méthode depuis n'importe quelle partie du code PHP.

- ☐ Vrai
- ☐ Faux

Question 2

La visibilité "protected" permet d'accéder à une propriété ou une méthode depuis une classe parente ou une classe enfant.

- ☐ Vrai
- ☐ Faux

Question 3

La visibilité "private" permet d'accéder à une propriété ou une méthode depuis n'importe quelle partie du code PHP.

- ☐ Vrai
- ☐ Faux

Question 4

L'encapsulation permet de protéger les propriétés et les méthodes d'une classe.

- ☐ Vrai
- ☐ Faux

Question 5


Qu'est-ce que sont les méthodes magiques en PHP ?

- ☐ Les méthodes magiques en PHP sont des méthodes spéciales qui permettent de manipuler les propriétés et les méthodes d'une classe
- ☐ Les méthodes magiques en PHP sont des méthodes permettant d'accéder à des fonctions spécifiques du système d'exploitation de l'hôte PHP
- ☐ Les méthodes magiques en PHP sont des méthodes qui sont cachées et qui ne sont pas accessibles depuis l'extérieur de la classe

Solutions des exercices


Exercice p. 7 Solution n°1**Question 1**

Qu'est-ce que l'encapsulation en PHP ?

- ☒ Un moyen de cacher des données à l'utilisateur
- ☐ Une méthode pour rendre toutes les propriétés publiques
- ☐ Un concept qui n'existe pas en PHP
-  L'encapsulation en PHP est un moyen de cacher des données à l'utilisateur en utilisant les niveaux de visibilité public, protected et private.


Question 2

Quel est l'intérêt de l'encapsulation en PHP ?

- ☒ Assurer l'intégrité de la structure d'une classe
- ☐ Faciliter la compréhension du code par l'utilisateur
- ☐ Permettre à l'utilisateur de modifier les données à sa guise
-  L'encapsulation en PHP permet d'assurer l'intégrité de la structure d'une classe en forçant l'utilisateur à passer par un chemin prédéfini pour modifier une donnée.


Question 3

L'encapsulation est un principe de la programmation orientée objet qui permet de masquer les détails d'implémentation d'une classe à l'utilisateur.

- ☒ Vrai
- ☐ Faux
-  L'encapsulation est un principe clé de la programmation orientée objet qui permet de cacher les détails d'implémentation d'une classe de l'utilisateur. En encapsulant les propriétés et les méthodes, on peut contrôler l'accès à ces éléments depuis l'extérieur de la classe et garantir l'intégrité de la structure de la classe.


Question 4

Que fait la méthode get dans l'encapsulation en PHP ?

- ☒ Elle permet de récupérer la valeur d'un attribut de la classe
- ☐ Elle permet de définir une propriété privée
- ☐ Elle permet de rendre une propriété publique
-  En PHP, l'encapsulation permet de protéger les propriétés d'une classe en les définissant comme privées. Cela signifie que ces propriétés ne peuvent être accédées que depuis l'intérieur de la classe. La méthode get permet de récupérer la valeur de ces propriétés privées en fournissant un accès contrôlé à l'extérieur de la classe.

Question 5

Que fait la méthode set dans l'encapsulation en PHP ?


- ☐ Elle permet de récupérer la valeur d'une propriété privée
- ☒ Elle permet de définir un attribut de la classe
- ☐ Elle permet de rendre une propriété publique
-  La méthode set est utilisée pour définir ou modifier la valeur d'un attribut d'une classe en fournissant un accès contrôlé à l'extérieur de la classe. En utilisant la méthode set, nous pouvons définir les règles de validation et de sécurité pour la modification de la valeur de la propriété privée. Cela garantit que la propriété est modifiée de manière appropriée et sécurisée.

Exercice p. 11 Solution n°2

Question 1

Les propriétés et les méthodes déclarées comme private ne peuvent pas être accessibles en dehors de la classe.


- ☒ Vrai
- ☐ Faux

 Les propriétés et les méthodes déclarées comme private ne peuvent pas être accessibles en dehors de la classe. Cela signifie qu'elles ne peuvent être appelées ou modifiées que depuis l'intérieur de la classe elle-même.

Question 2

En PHP, il est préférable de rendre toutes les propriétés et méthodes d'une classe publiques pour faciliter leur accès depuis l'extérieur de la classe.


- ☐ Vrai
- ☒ Faux

 Il n'est pas recommandé de rendre toutes les propriétés et méthodes d'une classe publiques, car cela peut entraîner des problèmes d'intégrité de la structure de la classe et de sécurité. Il est important de définir des niveaux de visibilité appropriés pour chaque élément de la classe en fonction de ses besoins.

Question 3

Que signifie le niveau de visibilité protected en PHP ?

- ☐ Les propriétés et les méthodes marquées comme protected ne peuvent être accédées qu'à partir de la classe elle-même
- ☐ Les propriétés et les méthodes marquées comme protected peuvent être accédées à partir de n'importe où dans le code
- ☒ Les propriétés et les méthodes marquées comme protected ne peuvent être accédées qu'à partir de la classe elle-même et de ses sous-classes

 Le niveau de visibilité protected signifie que les propriétés et les méthodes marquées comme protected ne peuvent être accédées qu'à partir de la classe elle-même et de ses sous-classes. Cela signifie que si vous déclarez une propriété ou une méthode comme protected dans une classe, elle ne pourra pas être accédée directement à partir de l'extérieur de la classe.

Question 4

Quel est le niveau de visibilité par défaut en PHP ?

☒ public

☐ protected

☐ private

Q Le niveau de visibilité par défaut en PHP est public. Cela signifie que si vous ne spécifiez pas explicitement le niveau de visibilité d'une propriété ou d'une méthode, elle sera publique par défaut. Les propriétés et les méthodes publiques peuvent être accessibles à partir de n'importe où dans le code, à la fois à l'intérieur et à l'extérieur de la classe.

Question 5

Peut-on modifier le niveau de visibilité d'une propriété ou méthode en PHP ?

☐ Non, il est figé une fois défini

☐ Oui, il est possible de le modifier avec la méthode setVisibility()

☒ Oui, il est possible de le modifier en changeant simplement le mot-clé correspondant

Q Il est possible de modifier le niveau de visibilité d'une propriété ou méthode en PHP en changeant simplement le mot-clé correspondant (public, protected ou private). Cependant, cela doit être fait avec précaution, car cela peut avoir des effets sur la sécurité et la stabilité du code. Il est donc recommandé de définir les niveaux de visibilité de manière appropriée dès le départ et de ne pas les modifier sauf si cela est absolument nécessaire et fait avec une grande prudence et une compréhension complète des conséquences potentielles.

p. 12 Solution n°3

Voici à quoi ressemblerait la classe Client :

```
1 class Client {
2     private String $nom;
3     private String $adresse;
4     private String $email;
5     private String $telephone;
6
7     public function getNom() : string {
8         return $this->nom;
9     }
10
11    public function setNom(string $nom) {
12        $this->nom = $nom;
13    }
14
15    public function getAdresse() : string {
16        return $this->adresse;
17    }
18
19    public function setAdresse(string $adresse) {
20        $this->adresse = $adresse;
21    }
22
23    public function getEmail() : string {
24        return $this->email;
25    }
26
27    public function setEmail(string $email) {
28        $this->email = $email;
29    }
}
```

```

30
31     public function getTelephone():string {
32         return $this->telephone;
33     }
34
35     public function setTelephone(string $telephone) {
36         $this->telephone = $telephone;
37     }
38 //MÉTHODES MAGIQUES
39     public function __get($propriete) {
40         if (property_exists($this, $propriete)) {
41             return $this->$propriete;
42         }
43     }
44
45     public function __set($propriete, $valeur) {
46         if (property_exists($this, $propriete)) {
47             $this->$propriete = $valeur;
48         }
49     }
50 }

```

Avec cette classe, nous pouvons créer des objets "Client" et accéder à leurs propriétés via les méthodes publiques `get` et `set`. Les propriétés sont protégées et ne peuvent être modifiées que par les méthodes `set`.

```

1 $client = new Client();
2 $client->setNom("John Doe");
3 $client->setAdresse("123 rue de la Paix");
4 $client->setEmail("johndoe@example.com");
5 $client->setTelephone("01 23 45 67 89");
6
7 echo $client->getNom(); // affiche "John Doe"
8 echo $client->getAdresse(); // affiche "123 rue de la Paix"
9 echo $client->getEmail(); // affiche "johndoe@example.com"
10 echo $client->getTelephone(); // affiche "01 23 45 67 89"

```

p. 12 Solution n°4

L'encapsulation est un concept fondamental de la programmation orientée objet qui permet de limiter l'accès aux propriétés et méthodes d'une classe. En PHP, cela peut être réalisé en utilisant les niveaux de visibilité tels que public, protected et private. En limitant l'accès aux propriétés, on peut mieux contrôler leur utilisation et leur modification, ce qui peut éviter des erreurs et des bugs difficiles à résoudre. De plus, cela permet de mieux organiser le code et de rendre le système plus robuste et maintenable. En effet, si l'accès aux propriétés n'est pas contrôlé, n'importe quelle partie du code peut les modifier, ce qui peut entraîner des effets de bord non désirés. En limitant leur accessibilité aux parties de code qui en ont réellement besoin, on peut renforcer l'encapsulation des données et mieux organiser le code. Cela peut également faciliter la compréhension du code par d'autres développeurs et rendre le code plus facile à maintenir à long terme.


Exercice p. 13 Solution n°5

Question 1

La visibilité "public" permet d'accéder à une propriété ou une méthode depuis n'importe quelle partie du code PHP.

☒ Vrai

☐ Faux


 La visibilité "public" permet d'accéder à une propriété ou une méthode depuis n'importe quelle partie du code PHP.

Question 2

La visibilité "protected" permet d'accéder à une propriété ou une méthode depuis une classe parente ou une classe enfant.

☒ Vrai

☐ Faux


 La visibilité "protected" permet d'accéder à une propriété ou une méthode depuis une classe parente ou une classe enfant.

Question 3

La visibilité "private" permet d'accéder à une propriété ou une méthode depuis n'importe quelle partie du code PHP.

☐ Vrai

☒ Faux


 La visibilité "private" permet d'accéder à une propriété ou une méthode uniquement à l'intérieur de la classe dans laquelle elle est définie.

Question 4

L'encapsulation permet de protéger les propriétés et les méthodes d'une classe.

☒ Vrai

☐ Faux

 L'encapsulation permet de protéger les propriétés et les méthodes d'une classe en définissant des niveaux de visibilité.

Question 5

Qu'est-ce que sont les méthodes magiques en PHP ?

☒ Les méthodes magiques en PHP sont des méthodes spéciales qui permettent de manipuler les propriétés et les méthodes d'une classe

☐ Les méthodes magiques en PHP sont des méthodes permettant d'accéder à des fonctions spécifiques du système d'exploitation de l'hôte PHP

☐ Les méthodes magiques en PHP sont des méthodes qui sont cachées et qui ne sont pas accessibles depuis l'extérieur de la classe

Q Les méthodes magiques en PHP sont des méthodes spéciales qui permettent de manipuler les propriétés et les méthodes d'une classe. Elles commencent toutes par un double underscore (__) et sont automatiquement appelées par PHP dans certaines situations, comme la création d'un nouvel objet ou l'appel d'une méthode inexistante. Cette réponse est vraie, les méthodes magiques sont en effet des méthodes spéciales qui permettent de manipuler les propriétés et les méthodes d'une classe.