

PHP Orientée Objet

Apprendre la POO en php

PHP Orientée Objet

Apprendre la POO en php

Quelques Prérequis avant de commencer

PHP Orientée Objet

Apprendre la POO en php

Quelques Prérequis avant de commencer

- Connaître PHP (les boucles, variables, conditions et fonctions)

PHP Orientée Objet

Apprendre la POO en php

Quelques Prérequis avant de commencer

- Connaître PHP (les boucles, variables, conditions et fonctions)
- Connaître MySql (les requêtes SQL, la base de données)

PHP Orientée Objet

Apprendre la POO en php

Quelques Prérequis avant de commencer

- Connaître PHP (les boucles, variables, conditions et fonctions)
- Connaître MySql (les requêtes SQL, la base de données)
- Avoir un éditeur de code (comme VSCode)

PHP Orientée Objet

Apprendre la POO en php

Quelques liens utiles

- Le lien du code source de la formation sur [github](#)
- Un [tutoriel complet sur PHP](#)
- La [documentation officiel de PHP](#)
- Une [extension VSCode](#) pour compléter le code PHP
- Une [extension VSCode](#) pour formater le code PHP automatiquement

PHP Orientée Objet

Apprendre la POO en php

Un objet c'est quoi ?

PHP Orientée Objet

Un objet c'est quoi ?

Les objets en PHP sont un nouveaux **type de variables**

Vous connaissez surement les types suivant :

PHP Orientée Objet

Un objet c'est quoi ?

Les objets en PHP sont un nouveaux **type de variables**

Vous connaissez surement les types suivant :

- string
- int
- float
- Array

PHP Orientée Objet

Concrètement, ça ressemble à
quoi ?

PHP Orientée Objet

Concrètement, ça ressemble à quoi ?

Voici un exemple d'utilisation de Date en PHP :

```
1 <?php
2
3 // Spécifie la timezone à utiliser
4 date_default_timezone_set('Europe/Paris');
5
6 // Voici une date sous forme de chaîne de caractère
7 $date = "2021-05-05";
8 // Nous ajoutons 3 mois à cette date
9 $newDate = date('Y-m-d', strtotime($date . ' +3 months'));
10 // Nous affichons la nouvelle date en format français
11 echo date('d/m/Y', strtotime($newDate));
```

PHP Orientée Objet

Concrètement, ça ressemble à quoi ?

Nous pouvons améliorer l'exemple grâce à des fonctions :

```
1 <?php
2
3 // Voici une date sous forme de chaîne de caractère
4 $date = "2021-05-05";
5
6 // Nous ajoutons 3 mois à cette date
7 $newDate = addMonths($date, 3)
8
9 // Nous affichons la nouvelle date en format français
10 echo formatDate($newDate, 'd/m/Y');
```

PHP Orientée Objet

Concrètement, ça ressemble à quoi ?

Ou bien nous pouvons utiliser un objet :

```
1 <?php
2 // Création de l'objet "MaDate"
3 $date = new MyDate('2021-05-05');
4
5 // On ajoute 3 mois
6 $date->addMonths(3);
7
8 // On formate la date en français
9 echo $date->format('d/m/Y');
```

PHP Orientée Objet

Ça sert à quoi ?

PHP Orientée Objet

Ça sert à quoi ?

Les objets possèdent de nombreux avantages :

PHP Orientée Objet

Ça sert à quoi ?

Les objets possèdent de nombreux avantages :

- Un code plus claire

PHP Orientée Objet

Ça sert à quoi ?

Les objets possèdent de nombreux avantages :

- Un code plus claire
- Une meilleur organisation

PHP Orientée Objet

Ça sert à quoi ?

Les objets possèdent de nombreux avantages :

- Un code plus claire
- Une meilleur organisation
- Moins de répétition de code

PHP Orientée Objet

Ça sert à quoi ?

Les objets possèdent de nombreux avantages :

- Un code plus claire
- Une meilleur organisation
- Moins de répétition de code
- Plus facile à utiliser

PHP Orientée Objet

Ça marche comment ?

PHP Orientée Objet

Ça marche comment ?

Afin d'utiliser des objets nous devons tout d'abord créer :

une instance

PHP Orientée Objet

Ça marche comment ?

Afin d'utiliser des objets nous devons tout d'abord créer :

une instance

Une instance se crée avec le mot clef :

new

PHP Orientée Objet

Ça marche comment ?

Éxemple, création **d'instance** de **MyDate** :



```
1 <?php  
2  
3 $firstDate = new MaDate('2021-05-10');  
4 $secondDate = new MaDate('2021-05-12');
```

PHP Orientée Objet

Ça marche comment ?

Éxemple, création **d'instance** de **MyDate** :



```
1 <?php  
2  
3 $firstDate = new MaDate('2021-05-10');  
4 $secondDate = new MaDate('2021-05-12');
```

Création d'une première instance de
MyDate le 10 mai 2021

PHP Orientée Objet

Ça marche comment ?

Éxemple, création **d'instance** de **MyDate** :



```
1 <?php  
2  
3 $firstDate = new MaDate('2021-05-10');  
4 $secondDate = new MaDate('2021-05-12');
```

Création d'une secondes instance de
MyDate le 12 mai 2021

PHP Orientée Objet

Ça marche comment ?

Chaque instances est indépendante !



```
1 <?php
2
3 $firstDate = new MaDate('2021-05-10');
4 $secondDate = new MaDate('2021-05-12');
5
6 echo $firstDate->format('d/m/Y');
7 echo $secondDate->format('d/m/Y');
```

Affiche : "10/05/2021"

PHP Orientée Objet

Ça marche comment ?

Chaque instances est indépendante !



```
1 <?php  
2  
3 $firstDate = new MaDate('2021-05-10');  
4 $secondDate = new MaDate('2021-05-12');  
5  
6 echo $firstDate->format('d/m/Y');  
7 echo $secondDate->format('d/m/Y');
```

Affiche : "12/05/2021"

PHP Orientée Objet

Ça marche comment ?

Afin de pouvoir **créer une instance** nous utilisons un **schéma de construction** :

La Class

PHP Orientée Objet

Ça marche comment ?

Dans notre exemple: **MyDate**

```
1 <php  
2  
3 $date = new MyDate('2021-05-20');
```

Variable qui contiendra
l'instance

Nom de la **Class**

Création de l'**instance**

PHP Orientée Objet

Ça marche comment ?

Dans notre exemple: **MyDate**

```
● ● ●  
1 <php  
2  
3 $date = new MyDate('2021-05-20');
```

On dit ici que :

\$date est une instance de la class « MyDate »

PHP Orientée Objet

Ça marche comment ?

Un objet PHP contient des **variables interne** que l'on appelle

Des Propriétés

PHP Orientée Objet

Ça marche comment ?

Voici un exemple de **propriétés** simple
avec la class **MyDate**

```
1 <php
2
3 $date = new MyDate('2021-05-20');
4
5 // Notre objet date contient des propriétés (comprenez variable)
6 // pour chaque membre de la date :
7 $date->day // Contient 20
8 $date->month // Contient 05
9 $date->year // Contient 2021
```

PHP Orientée Objet

Ça marche comment ?

Tout comme les variables nous pouvons changer leurs valeurs :

```
1 <php
2
3 $date = new MyDate('2021-05-20');
4
5 // On change le jour de notre date
6 $date->day = 25;
7
8 echo $date->format('d/m/Y'); // affiche 25/05/2021
```

PHP Orientée Objet

Ça marche comment ?

Les objets en PHP contiennent aussi des fonctions que l'on appelle :

Les Méthodes

PHP Orientée Objet

Ça marche comment ?

Voici un exemple de méthode sur notre objet MyDate



```
1 <php
2
3 $date = new MyDate('2021-05-20');
4
5 $date->addMonths(3);
6
7 echo $date->format('d/m/Y');
```

PHP Orientée Objet

Ça marche comment ?

Voici un exemple de méthode sur notre objet MyDate

```
1 <php
2
3 $date = new MyDate('2021-05-20');
4
5 $date->addMonths(3);
6
7 echo $date->format('d/m/Y');
```

On utilise la **méthode "addMonth"** sur l'instance de **MyDate**

PHP Orientée Objet

Ça marche comment ?

Voici un exemple de méthode sur notre objet MyDate



```
1 <php
2
3 $date = new MyDate('2021-05-20');
4
5 $date->addMonths(3);
6
7 echo $date->format('d/m/Y');
```

On utilise la **méthode "format"** sur l'instance de **MyDate**

PHP Orientée Objet

Ça marche comment ?

Pour résumer tout ça ! Un objet c'est :

- Un schéma de construction : **La Class**
- Une **instance** que l'on créé avec "**new**"
- Des **propriétés** (les variables de notre objet)
- Des **méthodes** (les fonctions de notre objet)

PHP Orientée Objet

Écrire notre première class

PHP Orientée Objet

Écrire notre première class

Afin de bien comprendre comment fonctionne les objets nous allons maintenant créer notre première class

Un Personnage

PHP Orientée Objet

Écrire notre première class

Créons un fichier PHP dans
'src/Personnage.php' et plaçons le code
suivant :



```
1 <?php
2
3 class Personnage
4 {
5     public int $vie = 100;
6
7     public int $attaque = 20;
8 }
```

PHP Orientée Objet

Écrire notre première class

Maintenant importons et utilisons notre première class dans `public/index.php` :



```
1 <?php
2
3 require_once '../src/Personnage.php';
4
5 $merlin = new Personnage();
6
7 var_dump($merlin);
```

PHP Orientée Objet

Écrire notre première class

Nous pouvons maintenant afficher le résultat dans notre navigateur :

```
/usr/src/myapp/public/index.php:7:  
object(Personnage)[1]  
    public int 'vie' => int 100  
    public int 'attaque' => int 20
```

PHP Orientée Objet

Écrire notre première class

Rétournons dans `src/Personnage.php`
afin de comprendre

```
1 <?php
2
3 class Personnage
4 {
5     public int $vie = 100;
6
7     public int $attaque = 20;
8 }
```

Ici nous créons un schéma de construction
: une **class Personnage**

PHP Orientée Objet

Écrire notre première class

Rétournons dans `src/Personnage.php`
afin de comprendre

```
1 <?php
2
3 class Personnage
4 {
5     public int $vie = 100;
6
7     public int $attaque = 20;
8 }
```

ATTENTION !

Les class **commencent toutes par une Majuscule et correspondent au nom de notre fichier PHP !**

PHP Orientée Objet

Écrire notre première class

Rétournons dans `src/Personnage.php`
afin de comprendre

```
1 <?php
2
3 class Personnage
4 {
5     public int $vie = 100;
6
7     public int $attaque = 20;
8 }
```

Si notre class s'appel **Personnage** alors
notre fichier PHP doit s'appeler
Personnage.php

PHP Orientée Objet

Écrire notre première class

Rétournons dans `src/Personnage.php`
afin de comprendre

```
1 <?php
2
3 class Personnage
4 {
5     public int $vie = 100;
6
7     public int $attaque = 20;
8 }
```

Ici nous déclarons 2 propriétés : "\$vie" et
"\$attack"

PHP Orientée Objet

Écrire notre première class

Un propriété possède 4 sections :

```
1 <?php  
2  
3 class Personnage  
4 {  
5     public int $vie = 100;  
6     public int $attaque = 20;  
7 }  
8 }
```

La porté de notre propriété (nous en reparlerons un peu plus tard :))

Valeur de la propriété

Nom de la propriété

Type de la propriété, ici "int"

PHP Orientée Objet

Écrire notre première class

Maintenant regardons comment utiliser notre personnage :

```
1 <?php
2
3 require_once '../src/Personnage.php';
4
5 $merlin = new Personnage();
6
7 var_dump($merlin);
```

Nous importons le fichier PHP de notre class

PHP Orientée Objet

Écrire notre première class

Maintenant regardons comment utiliser notre personnage :

```
1 <?php
2
3 require_once '../src/Personnage.php';
4
5 $merlin = new Personnage();
6
7 var_dump($merlin);
```

Création d'une première **instance** de
Personnage : **\$merlin**

PHP Orientée Objet

Écrire notre première class

Maintenant regardons comment utiliser notre personnage :

```
1 <?php
2
3 require_once '../src/Personnage.php';
4
5 $merlin = new Personnage();
6
7 var_dump($merlin);
```

Nous déboguons notre objet

PHP Orientée Objet

Écrire notre première class

Nous pouvons maintenant jouer avec notre objet PHP :

```
1 <?php
2
3 require_once '../src/Personnage.php';
4
5 $merlin = new Personnage();
6 $merlin->vie = 50;
7 $merlin->attaque = 40;
8
9 $arthur = new Personnage();
10 $arthur->vie = 100;
11 $arthur->attaque = 30;
12
13 var_dump($merlin);
14 var_dump($arthur);
```

PHP Orientée Objet

Écrire notre première class

Voici ce que notre navigateur nous affiche

```
/usr/src/myapp/public/index.php:13:  
object(Personnage)[1]  
    public int 'vie' => int 50  
    public int 'attaque' => int 40  
  
/usr/src/myapp/public/index.php:14:  
object(Personnage)[2]  
    public int 'vie' => int 100  
    public int 'attaque' => int 30
```

PHP Orientée Objet

Écrire notre première class

Il est possible dans une **méthode** de faire référence à l'**instance en cours** et d'accéder au propriétés de notre objet en utilisant

\$this

PHP Orientée Objet

Écrire notre première class

Exemple : Une méthode qui régénère notre **Personnage**

```
● ● ●  
1 <?php  
2  
3 class Personnage  
4 {  
5     public int $vie = 100;  
6  
7     public int $attaque = 20;  
8  
9     public function regenerer(): void  
10    {  
11        $this->vie = 100;  
12    }  
13 }
```

PHP Orientée Objet

Écrire notre première class

Voici comment écrire une méthode !

```
1 <?php
2
3 class Personnage
4 {
5     public int $vie = 100;
6
7     public int $attaque = 20;
8
9     public function regenerer(): void
10    {
11        $this->vie = 100;
12    }
13 }
```

Type de retour de notre méthode. Ici **void** signifie **rien du tout** car notre méthode n'as pas d'instruction "**return**"

PHP Orientée Objet

Écrire notre première class

Voici comment écrire une méthode !

```
1 <?php
2
3 class Personnage
4 {
5     public int $vie = 100;
6
7     public int $attaque = 20;
8
9     public function regenerer(): void
10    {
11        $this->vie = 100;
12    }
13 }
```

Nom de la méthode

PHP Orientée Objet

Écrire notre première class

Voici comment écrire une méthode !

```
1 <?php
2
3 class Personnage
4 {
5     public int $vie = 100;
6
7     public int $attaque = 20;
8
9     public function regenerer(): void
10    {
11        $this->vie = 100;
12    }
13}
```

Porté de la fonction

(nous verrons cela un
peu plus tard :))

PHP Orientée Objet

Écrire notre première class

Voici comment écrire une méthode !

```
● ● ●  
1 <?php  
2  
3 class Personnage  
4 {  
5     public int $vie = 100;  
6  
7     public int $attaque = 20;  
8  
9     public function regenerer(): void  
10    {  
11        $this->vie = 100;  
12    }  
13 }
```

Nous pouvons faire **référence à l'instance** en cour
et **accéder au propriétés** de notre objet en utilisant
\$this

PHP Orientée Objet

Écrire notre première class

Utilisons notre méthode "régénérer"



```
1 <?php
2
3 require_once '../src/Personnage.php';
4
5 $merlin = new Personnage();
6 $merlin->vie = 50;
7 $merlin->attaque = 40;
8 $merlin->regenerer();
9
10 $arthur = new Personnage();
11 $arthur->vie = 100;
12 $arthur->attaque = 30;
13
14 var_dump($merlin);
15 var_dump($arthur);
```

PHP Orientée Objet

Écrire notre première class

Tout comme les fonctions, les méthodes peuvent recevoir des paramètres :

```
● ● ●  
1 <?php  
2  
3 class Personnage  
4 {  
5     public int $vie = 100;  
6  
7     public int $attaque = 20;  
8  
9     public function regenerer(int $vie = 100): void  
10    {  
11        $this->vie = $this->vie + $vie;  
12    }  
13 }
```

PHP Orientée Objet

Écrire notre première class

La valeur par défaut de notre paramètre



```
1 <?php
2
3 class Personnage
4 {
5     public int $vie = 100;
6
7     public int $attaque = 20;
8
9     public function regenerer(int $vie = 100): void
10    {
11        $this->vie = $this->vie + $vie;
12    }
13 }
```



PHP Orientée Objet

Écrire notre première class

```
1 <?php
2
3 class Personnage
4 {
5     public int $vie = 100;
6
7     public int $attaque = 20;
8
9     public function regenerer(int $vie = 100): void
10    {
11        $this->vie = $this->vie + $vie;
12    }
13 }
```

Le nom de notre paramètre

PHP Orientée Objet

Écrire notre première class

```
1 <?php
2
3 class Personnage
4 {
5     public int $vie = 100;
6
7     public int $attaque = 20;
8
9     public function regenerer(int $vie = 100): void
10    {
11        $this->vie = $this->vie + $vie;
12    }
13 }
```

Le **type** de notre **paramètre**

PHP Orientée Objet

Écrire notre première class

```
1 <?php
2
3 class Personnage
4 {
5     public int $vie = 100;
6
7     public int $attaque = 20;
8
9     public function regenerer(int $vie = 100): void
10    {
11        $this->vie = $this->vie + $vie;
12    }
13 }
```



On **assigne** la vie de notre objet à la **valeur envoyé**
en paramètre

PHP Orientée Objet

Écrire notre première class

Utilisons la méthode régénérer avec un paramètre !

```
1 <?php
2
3 require_once '../src/Personnage.php';
4
5 $merlin = new Personnage();
6 $merlin->vie = 50;
7 $merlin->attaque = 40;
8 $merlin->regenerer(20);
9
10 $arthur = new Personnage();
11 $arthur->vie = 100;
12 $arthur->attaque = 30;
13
14 var_dump($merlin);
15 var_dump($arthur);
```

PHP Orientée Objet

Écrire notre première class

Exercice

Écrire une méthode "afficher" qui affiche la vie et l'attaque du personnage et utiliser la méthode dans `public/index.php`

PHP Orientée Objet

Écrire notre première class

Solution



```
1 <?php
2
3 class Personnage
4 {
5     public int $vie = 100;
6
7     public int $attaque = 20;
8
9     public function regenerer(int $vie = 100): void
10    {
11        $this->vie = $this->vie + $vie;
12    }
13
14    public function afficher(): void
15    {
16        echo 'Ce personnage a '
17        .
18        .
19        .
20        .
21    }
22 }
```

PHP Orientée Objet

Écrire notre première class

Solution



```
1 <?php
2
3 require_once '../src/Personnage.php';
4
5 $merlin = new Personnage();
6 $merlin->vie = 50;
7 $merlin->attaque = 40;
8 $merlin->regenerer(20);
9
10 $arthur = new Personnage();
11 $arthur->vie = 100;
12 $arthur->attaque = 30;
13
14 $merlin->afficher();
15
16 echo '<br/>';
17
18 $arthur->afficher();
```

PHP Orientée Objet

Écrire notre première class

Exercice

Créer une fonction "attaquer" qui reçoit un Personnage et enlève la vie du personnage en fonction de son attaque !

Utiliser cette méthode dans
`public/index.php`

PHP Orientée Objet

Écrire notre première class

Solution

```
● ● ●  
1 <?php  
2  
3 class Personnage  
4 {  
5     public int $vie = 100;  
6  
7     public int $attaque = 20;  
8  
9     public function attaquer(Personnage $cible): void  
10    {  
11        $cible->vie = $cible->vie - $this->attaque;  
12    }  
13  
14    public function regenerer(int $vie = 100): void  
15    { ...  
16    }  
17  
18    public function afficher(): void  
19    { ...  
20    }  
21 }
```

PHP Orientée Objet

Écrire notre première class

Solution



```
1 <?php
2
3 require_once '../src/Personnage.php';
4
5 $merlin = new Personnage();
6 $merlin->vie = 50;
7 $merlin->attaque = 40;
8
9 $arthur = new Personnage();
10 $arthur->vie = 100;
11 $arthur->attaque = 30;
12
13 $arthur->attaquer($merlin);
14
15 $merlin->afficher();
16
17 echo '<br/>';
18
19 $arthur->afficher();
```

PHP Orientée Objet

Écrire notre première class

Nos objets possèdent une méthode spécial qui nous permet de construire notre instance :

Le constructeur

PHP Orientée Objet

Écrire notre première class

Nous pouvons facilement définir un constructeur sur notre Personnage afin que l'on puisse donner **un nom** à notre **Personnage**

PHP Orientée Objet

Écrire notre première class

Exemple



```
1 <?php
2
3 class Personnage
4 {
5     ...
6
7     public string $nom;
8
9     public function __construct(string $nom)
10    {
11        $this->nom = $nom;
12    }
13
14    ...
15 }
```

PHP Orientée Objet

Écrire notre première class

Exemple



```
1 <?php
2
3 class Personnage
4 {
5     ...
6
7     public string $nom;
8
9     public function __construct(string $nom)
10    {
11        $this->nom = $nom;
12    }
13
14    ...
15 }
```



Nous définissons une nouvelle propriété "\$nom"

PHP Orientée Objet

Écrire notre première class

Exemple



```
1 <?php
2
3 class Personnage
4 {
5     ...
6
7     public string $nom;
8
9     public function __construct(string $nom)
10    {
11         $this->nom = $nom;
12    }
13
14    ...
15 }
```



Nous créons la méthode **constructeur**

PHP Orientée Objet

Écrire notre première class

Exemple



```
1 <?php
2
3 class Personnage
4 {
5     ...
6
7     public string $nom;
8
9     public function __construct(string $nom)
10    {
11         $this->nom = $nom;
12    }
13
14    ...
15 }
```



Nous ajoutons un **paramètre** de type **string** \$nom

PHP Orientée Objet

Écrire notre première class

Exemple



```
1 <?php
2
3 class Personnage
4 {
5     ...
6
7     public string $nom;
8
9     public function __construct(string $nom)
10    {
11         $this->nom = $nom;
12    }
13
14    ...
15 }
```



Nous **assignons la propriété** nom de notre objet à
la valeur envoyé en paramètre

PHP Orientée Objet

Écrire notre première class

Utilisons maintenant notre constructeur
dans `public/index.php` :

```
● ● ●  
1 <?php  
2  
3 require_once '../src/Personnage.php';  
4  
5 $merlin = new Personnage('Merlin');  
6 $merlin->vie = 50;  
7 $merlin->attaque = 40;  
8  
9 $arthur = new Personnage('Arthur');  
10 $arthur->vie = 100;  
11 $arthur->attaque = 30;  
12  
13 $arthur->attaquer($merlin);  
14  
15 $merlin->afficher();  
16  
17 echo '<br/>';  
18  
19 $arthur->afficher();
```

PHP Orientée Objet

Écrire notre première class

Exercice

Retoucher la méthode "afficher" pour utiliser la propriété nom de notre objet

PHP Orientée Objet

Écrire notre première class

Solution

```
● ● ●  
1 <?php  
2  
3 class Personnage  
4 {  
5     ...  
6  
7     public function afficher(): void  
8     {  
9         echo $this->nom  
10        . ' a '  
11        . $this->vie  
12        . ' points de vie et '  
13        . $this->attaque  
14        . ' en attaque.';  
15    }  
16 }
```

PHP Orientée Objet

La portée

PHP Orientée Objet

La portée

Dans un objet nous avons vu que chaque propriétés ou méthodes possèdent une portée c'est ce que nous appelons les

Access Modifiers

(Modificateurs d'accès en français)

PHP Orientée Objet

La portée

Il en existe 3 !

PHP Orientée Objet

La portée

public

La propriété ou la méthode est accessible
depuis n'importe où

PHP Orientée Objet

La portée

private

La propriété ou la méthode est accessible
uniquement depuis sa class

PHP Orientée Objet

La portée

protected

La propriété ou la méthode est accessible
**uniquement depuis sa class ou ses
classes hérité**

(Nous verrons celui ci un peu plus tard)

PHP Orientée Objet

La portée

Exemple, définissons notre propriété "nom" en **private**

```
1 <?php
2
3 class Personnage
4 {
5     ...
6
7     private string $nom;
8
9     ...
10
11    public function afficher(): void
12    {
13        echo $this->nom
14            . ' a '
15            . $this->vie
16            . ' points de vie et '
17            . $this->attaque
18            . ' en attaque.';
19    }
20 }
```

PHP Orientée Objet

La portée

Exemple, définissons notre propriété "nom" en **private**

```
1 <?php
2
3 class Personnage
4 {
5     ...
6
7     private string $nom;
8     ...
9
10
11    public function afficher(): void
12    {
13        echo $this->nom
14        . ' a '
15        . $this->vie
16        . ' points de vie et '
17        . $this->attaque
18        . ' en attaque.';
```

Nous définissons cette propriété comme "privée"

PHP Orientée Objet

La portée

Exemple, définissons notre propriété "nom" en **private**

Ici, nous pouvons utiliser notre propriété "name" car
nous sommes à l'intérieur de la classe !

```
3 class Personnage
4 {
5     ...
6
7     private string $nom;
8
9     ...
10
11    public function afficher(): void
12    {
13        echo $this->nom
14        . ' a '
15        . $this->vie
16        . ' points de vie et '
17        . $this->attaque
18        . ' en attaque.';
19    }
20 }
```

PHP Orientée Objet

La portée

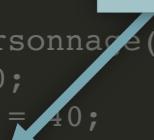
Exemple, définissons notre propriété "nom" en **private**



```
1 <?php  
2  
3 require_once '../src/P  
4  
5 $merlin = new Personnage('Merlin');  
6 $merlin->vie = 50;  
7 $merlin->attaque = 40;  
8  
9 echo $merlin->nom;  
10  
11 $arthur = new Personnage('Arthur');  
12 $arthur->vie = 100;  
13 $arthur->attaque = 30;  
14  
15 $arthur->attaquer($merlin);  
16  
17 $merlin->afficher();  
18  
19 echo '<br/>';  
20  
21 $arthur->afficher();
```

Mais maintenant nous **générons une erreur !!**

Il est **impossible** d'accéder à une propriété "private"
depuis l'**extérieur de notre classe !**



PHP Orientée Objet

La portée

En POO il existe une règle qui définit la porté de nos membres :

L'encapsulation

PHP Orientée Objet

La portée

Ces règles stipule :

1. Toutes les propriétés doivent être "private" ou "protected"
1. Seul certaines méthodes sont "public"

PHP Orientée Objet

La portée

L'encapsulation permet de

1. Conserver des objets "stable" (évite les bug et les mauvaises utilisations)
2. Fournir un « manuel d'emploi »
3. Rendre notre code « ouvert à l'extension » mais « fermé à la modification » (une fois qu'un objet est codé nous ne reviendrons plus dessus :))

PHP Orientée Objet

La portée

Afin d'accéder à nos propriétés depuis l'extérieur de notre class nous pouvons mettre en place des

Accesseurs

PHP Orientée Objet

La portée

Afin d'accéder à nos propriétés depuis l'extérieur de notre class nous pouvons mettre en place des

Accesseurs

Ils portent aussi le nom de

Getters and Setters

PHP Orientée Objet

La portée

Exemple avec la propriété "nom"

```
1 <?php
2
3 class Personnage
4 {
5
6     private string $nom;
7
8     ...
9
10    public function getNom(): string
11    {
12        return $this->nom;
13    }
14
15    public function setNom(string $nom): void
16    {
17        $this->nom = $nom;
18    }
19 }
```

PHP Orientée Objet

La portée



```
1 <?php
2
3 class Personnage
4 {
5
6     private string $nom;
7
8     ...
9
10    public function getNom(): string
11    {
12        return $this->nom;
13    }
14
15    public function setNom(string $nom): void
16    {
17        $this->nom = $nom;
18    }
19 }
```

La propriété "nom" reste private !

PHP Orientée Objet

La portée



```
1 <?php
2
3 class Personnage
4 {
5
6     private string $nom;
7
8     ...
9
10    public function getNom(): string
11    {
12        return $this->nom;
13    }
14
15    public function setNom(string $nom): void
16    {
17        $this->nom = $nom;
18    }
19 }
```

Nous créons un "Getter" (`getNom`) qui retourne la valeur de notre propriété nom

PHP Orientée Objet

La portée



```
1 <?php
2
3 class Personnage
4 {
5
6     private string $nom;
7
8     ...
9
10    public function getNom(): string
11    {
12        return $this->nom;
13    }
14
15    public function setNom(string $nom): void
16    {
17        $this->nom = $nom;
18    }
19 }
```

Nous créons un "Setter" qui change la valeur de notre propriété "nom"

PHP Orientée Objet

La portée

Il est difficile de voir l'intérêt de l'encapsulation sans avoir beaucoup pratiqué avant.

PHP Orientée Objet

La portée

Il est difficile de voir l'intérêt de l'encapsulation sans avoir beaucoup pratiqué avant.

Même si ce System paraît plus compliqué il nous assure que nos objets soit manipulé correctement !

PHP Orientée Objet

La portée

Il est difficile de voir l'intérêt de l'encapsulation sans avoir beaucoup pratiqué avant.

Même si ce System paraît plus compliqué il nous assure que nos objets soit manipulé correctement !

C'est aussi grâce à l'encapsulation que notre code devient « réutilisable » et extensible !

PHP Orientée Objet

La portée

Il est difficile de voir l'intérêt de l'encapsulation sans avoir beaucoup pratiqué avant.

Même si ce System paraît plus compliqué il nous assure que nos objets soit manipulé correctement !

C'est aussi grâce à l'encapsulation que notre code devient « réutilisable » et extensible !

Nous aurons l'occasion de voir les bienfaits de l'encapsulation un peu plus tard :)

PHP Orientée Objet

La portée

Pour résumer !

- Toutes les propriétés d'un objet doivent être "private" ou "protected"
- Seulement certaines méthodes de notre objets peuvent être "public"
- Ces règles correspondent à "l'encapsulation". Elle est utilisé dans tout les langages de programmation orientée objet !

PHP Orientée Objet

La portée

Éxercice

Rendre toutes les propriétés de personnage "private" !

PHP Orientée Objet

La portée

Solution



```
1 <?php
2
3 class Personnage
4 {
5     private int $vie = 100;
6
7     private int $attaque = 20;
8
9     private string $nom;
10
11     ...
12 }
```

PHP Orientée Objet

La portée

Nous n'avons pas besoin d'accesseur pour le moment ! :)

PHP Orientée Objet

L'héritage

PHP Orientée Objet

L'héritage

L'héritage est un concept qui nous permet d'étendre le fonctionnement d'un objet sans avoir à retoucher son code !

PHP Orientée Objet

L'héritage

L'héritage est un concept qui nous permet d'étendre le fonctionnement d'un objet sans avoir à retoucher son code !

Il permet à une **class** d'être **l'enfant** d'une autre **class**

PHP Orientée Objet

L'héritage

L'héritage est un concept qui nous permet d'étendre le fonctionnement d'un objet sans avoir à retoucher son code !

Il permet à une **class** d'être **l'enfant** d'une autre **class**

Ainsi la **classe enfant** possède **les méthodes et propriétés** de la **classe parente** !

PHP Orientée Objet

L'héritage

Exemple : Une class Magicien qui possède
toutes les propriétés et méthodes de
Personnage mais avec moins de vie et plus
d'attaque !

PHP Orientée Objet

L'héritage

Exemple : src/Magician.php

```
● ● ●  
1 <?php  
2  
3 class Magician extends Personnage  
4 {  
5 }
```

PHP Orientée Objet

L'héritage

Exemple : public/index.php

```
1 <?php
2
3 require_once '../src/Personnage.php';
4 require_once '../src/Magician.php';
5
6 $merlin = new Magician('Merlin');
7 $arthur = new Personnage('Arthur');
8
9 $arthur->attaquer($merlin);
10 $merlin->afficher();
11
12 echo '<br/>';
13
14 $arthur->afficher();
```

PHP Orientée Objet

L'héritage



```
1 <?php
2
3 require_once '../src/Personnage.php';
4 require_once '../src/Magician.php';
5
6 $merlin = new Magician('Merlin');
7 $arthur = new Personnage('Arthur');
8
9 $arthur->attaquer($merlin);
10 $merlin->afficher();
11
12 echo '<br/>';
13
14 $arthur->afficher();
```

On inclue notre class Magician

PHP Orientée Objet

L'héritage



```
1 <?php
2
3 require_once '../src/Personnage.php';
4 require_once '../src/Magician.php';
5
6 $merlin = new Magician('Merlin');
7 $arthur = new Personnage('Arthur');
8
9 $arthur->attaquer($merlin);
10 $merlin->afficher();
11
12 echo '<br/>';
13
14 $arthur->afficher();
```

On crée un magicien !

PHP Orientée Objet

L'héritage



```
1 <?php
2
3 require_once '../src/Personnage.php';
4 require_once '../src/Magician.php';
5
6 $merlin = new Magician('Merlin');
7 $arthur = new Personnage('Arthur');
8
9 $arthur->attaquer($merlin);
10 $merlin->afficher();
11
12 echo '<br/>';
13
14 $arthur->afficher();
```

Notre magicien s'utilise pareil que notre personnage car il hérite de "Personnage" !

PHP Orientée Objet

L'héritage

Dans la class magicien nous pouvons dès lors spécifié un nouveau comportement !

Nous pouvons aussi redéfinir des comportements !

PHP Orientée Objet

L'héritage

Exemple : Réécrire la méthode "afficher"
dans Magician :

```
1 <?php
2
3 class Magician extends Personnage
4 {
5     public function afficher(): void
6     {
7         echo 'Le magicien ';
8
9         parent::afficher();
10    }
11 }
```

PHP Orientée Objet

L'héritage

```
1 <?php
2
3 class Magician extends Personnage
4 {
5     public function afficher(): void
6     {
7         echo 'Le magicien ';
8
9         parent::afficher();
10    }
11 }
```

Nous redéfinissons la méthode 'afficher'

PHP Orientée Objet

L'héritage

```
1 <?php
2
3 class Magician extends Personnage
4 {
5     public function afficher(): void
6     {
7         echo 'Le magicien ';
8
9         parent::afficher();
10    }
11 }
```

Nous rajoutons le mot "Le magicien "

PHP Orientée Objet

L'héritage

```
1 <?php
2
3 class Magician extends Personnage
4 {
5     public function afficher(): void
6     {
7         echo 'Le magicien ';
8
9         parent::afficher();
10    }
11 }
```

Nous appelons la méthode de class parent "afficher" (ici, appel la méthode "afficher" de la classe Personnage !)

PHP Orientée Objet

L'héritage

Exercice

Redéfinir la méthode "attaquer" afin de faire 20 dégat de plus !!

PHP Orientée Objet

L'héritage

Solution

```
1 <?php
2
3 class Magician extends Personnage
4 {
5     public function afficher(): void
6     {
7         echo 'Le magicien ';
8
9         parent::afficher();
10    }
11
12    public function attaquer(Personnage $cible): void
13    {
14        parent::attaquer($cible);
15
16        $cible->vie -= 20;
17    }
18 }
```

PHP Orientée Objet

L'héritage

Solution

```
1 <?php
2
3 class Magician extends Personnage
4 {
5     public function afficher(): void
6     {
7         echo 'Le magicien ';
8
9         parent::afficher();
10    }
11
12    public function attaquer(Personnage $cible): void
13    {
14        parent::attaquer($cible);
15
16        $cible->vie -= 20;
17    }
18 }
```

PHP Orientée Objet

Gros Exercice !

Créer une class « Form » qui permet d'afficher des input et des labels !

Cette classe devrait pouvoir s'utiliser de tel manière :



```
1 <?php
2
3 $loginForm = new Form();
4
5 ?>
6 <form method="POST">
7     <?php $loginForm->label('Email :', 'email'); ?>
8     <?php $loginForm->input('email', 'email'); ?>
9     <?php $loginForm->label('Mot de passe :', 'password'); ?>
10    <?php $loginForm->input('password', 'password'); ?>
11    <?php $loginForm->submitButton('Envoyer'); ?>
12 </form>
```

PHP Orientée Objet

Gros Exercice !

Retoucher la class Form pour contenir un séparateur de block !

```
1 <?php
2
3 $loginForm = new Form('div');
4
5 ?>
6 <form method="POST">
7     <?php $loginForm->startBlock(); ?>
8     <?php $loginForm->label('Email : ', 'email'; ?>
9     <?php $loginForm->input('email', 'email'); ?>
10    <?php $loginForm->endBlock(); ?>
11
12    <?php $loginForm->startBlock(); ?>
13    <?php $loginForm->label('Mot de passe : ', 'password'); ?>
14    <?php $loginForm->input('password', 'password'); ?>
15    <?php $loginForm->endBlock(); ?>
16
17    <?php $loginForm->startBlock(); ?>
18    <?php $loginForm->submitButton('Envoyer'); ?>
19    <?php $loginForm->endBlock(); ?>
20 </form>
```

PHP Orientée Objet

Gros Exercice !

Nous pouvons maintenant créer une méthode "widget" qui contiendra les différents appels !



```
1 <?php
2
3 $loginForm = new Form('div');
4
5 ?>
6 <form method="POST">
7     <?php $loginForm->widget('Email : ', 'email', 'email'); ?>
8
9     <?php $loginForm->widget('Mot de passe : ', 'password', 'password'); ?>
10
11    <?php $loginForm->startBlock(); ?>
12    <?php $loginForm->submitButton('Envoyer'); ?>
13    <?php $loginForm->endBlock(); ?>
14 </form>
```

PHP Orientée Objet

Gros Exercice !

Nous pouvons faire la même chose avec notre "submitButton" et faire une méthode "button" :



```
1 <?php
2
3 $loginForm = new Form('div');
4
5 ?>
6 <form method="POST">
7     <?php $loginForm->widget('Email :', 'email', 'email'); ?>
8
9     <?php $loginForm->widget('Mot de passe :', 'password', 'password'); ?>
10
11    <?php $loginForm->button('Envoyer'); ?>
12 </form>
```

PHP Orientée Objet

Gros Exercice !

Nous pouvons même générer la balise form !



```
1 <?php
2
3 $loginForm = new Form('div');
4
5 ?>
6 <?php $loginForm->start('POST'); ?>
7     <?php $loginForm->widget('Email :', 'email', 'email'); ?>
8
9     <?php $loginForm->widget('Mot de passe :', 'password', 'password'); ?>
10
11    <?php $loginForm->button('Envoyer'); ?>
12 <?php $loginForm->end(); ?>
```

PHP Orientée Objet

Gros Exercice !

Nous pouvons aussi retoucher le constructeur pour envoyer de la configuration !



```
1 <?php
2
3 $loginForm = new Form( 'POST' , 'div' );
4
5 ?>
6 <?php $loginForm->start(); ?>
7     <?php $loginForm->widget('Email :', 'email', 'email'); ?>
8
9     <?php $loginForm->widget('Mot de passe :', 'password', 'password'); ?>
10
11    <?php $loginForm->button('Envoyer'); ?>
12 <?php $loginForm->end(); ?>
```

PHP Orientée Objet

Gros Exercice !

Nous pouvons même contracter tout notre code avec une seule méthode !



```
1 <?php
2
3 $loginForm = new Form( 'POST' , 'div' );
4
5 ?>
6 <?php $loginForm->display([
7     'widgets' => [
8         [ 'Email :', 'email', 'email' ],
9         [ 'Mot de passe :', 'password', 'password' ]
10    ],
11    'button' => 'Envoyer'
12 ]); ?>
```

PHP Orientée Objet

Méthodes et Propriété static

PHP Orientée Objet

Méthodes et Propriété static

Il est possible de créer des méthodes et des propriétés dite **static**

PHP Orientée Objet

Méthodes et Propriété static

Il est possible de créer des méthodes et des propriétés dite **static**

Ces dernières sont **directement rattaché à une class et non à son instance !**

PHP Orientée Objet

Méthodes et Propriété static

Il est possible de créer des méthodes et des propriétés dite **static**

Ces dernières sont **directement rattaché à une class et non à son instance !**

Ce sont comme de simple "fonction" mais rangé dans une class

PHP Orientée Objet

Méthodes et Propriété static

Un exemple de class static serait une class Text avec la possibilité de sauter une ligne !

```
● ● ●  
1 <?php  
2  
3 class Text  
4 {  
5     static public function lineBreak(): void  
6     {  
7         echo "<br />";  
8     }  
9 }
```

PHP Orientée Objet

Méthodes et Propriété static

Afin d'utiliser cette méthode on utilise le nom de class suivie de l'opérateur static !

```
1 <?php
2
3 require_once '../src/Personnage.php';
4 require_once '../src/Magician.php';
5 require_once '../src/Text.php';
6
7 $merlin = new Magician('Merlin', 100, 40);
8 $arthur = new Personnage('Arthur', 100, 10);
9
10 $merlin->afficher();
11
12 Text::lineBreak();
13
14 $arthur->afficher();
```

PHP Orientée Objet

Méthodes et Propriété static

Inclusion de la class

```
1 <?php
2
3 require_once '../src/Personnage.php';
4 require_once '../src/Magician.php';
5 require_once '../src/Text.php';
6
7 $merlin = new Magician('Merlin', 100, 40);
8 $arthur = new Personnage('Arthur', 100, 10);
9
10 $merlin->afficher();
11
12 Text::lineBreak();
13
14 $arthur->afficher();
```

PHP Orientée Objet

Méthodes et Propriété static

Appel de la méthode static "lineBreak" sur notre Text

```
● ● ●  
1 <?php  
2  
3 require_once '../src/Personnage.php';  
4 require_once '../src/Magician.php';  
5 require_once '../src/Text.php';  
6  
7 $merlin = new Magician('Merlin', 100, 40);  
8 $arthur = new Personnage('Arthur', 100, 10);  
9  
10 $merlin->afficher();  
11  
12 Text::lineBreak();  
13  
14 $arthur->afficher();
```

PHP Orientée Objet

Autoload

PHP Orientée Objet

Autoload

Il est possible grâce à la fonction
"spl_autoload_register" de **"require"** des
classes automatiquement !

PHP Orientée Objet

Autoload

A chaque instantiation (`new`) cette
fonction sera appelé !



```
1 <?php
2
3 spl_autoload_register(function ($class) {
4     require_once __DIR__ . '/../src/' . $class . '.php';
5 });
6
7 $merlin = new Magician('Merlin', 100, 40);
8 $arthur = new Personnage('Arthur', 100, 10);
```

PHP Orientée Objet

Autoload

Enregistrement d'une fonction d'autoload.

Nous recevons le nom de la class en première argument !



```
1 <?php
2
3 spl_autoload_register(function ($class) {
4     require_once __DIR__ . '/../src/' . $class . '.php';
5 });
6
7 $merlin = new Magician('Merlin', 100, 40);
8 $arthur = new Personnage('Arthur', 100, 10);
```

PHP Orientée Objet

Autoload

Nous incluons notre fichier de class ! (ici
DIR fais référence au chemin du
répertoire du fichier en cours)



```
1 <?php
2
3 spl_autoload_register(function ($class) {
4     require_once __DIR__ . '/../src/' . $class . '.php';
5 });
6
7 $merlin = new Magician('Merlin', 100, 40);
8 $arthur = new Personnage('Arthur', 100, 10);
```

PHP Orientée Objet

Autoload

Lors de l'instanciation, PHP sera capable
de chargé les fichiers de class
automatiquement !



```
1 <?php
2
3 spl_autoload_register(function ($class) {
4     require_once __DIR__ . '/../src/' . $class . '.php';
5 });
6
7 $merlin = new Magician('Merlin', 100, 40);
8 $arthur = new Personnage('Arthur', 100, 10);
```

PHP Orientée Objet

Les namespace

PHP Orientée Objet

Les namespace

Dans un vrai projet PHP, nous utilsons des
milliers de class !

PHP Orientée Objet

Les namespace

Dans un vrai projet PHP, nous utilsons des
milliers de class !

Afin d'éviter que 2 class soit nommé pareil
(le conflit de nommage)

PHP Orientée Objet

Les namespace

Dans un vrai projet PHP, nous utilsons des **milliers** de class !

Afin d'éviter que 2 class soit nommé pareil
(le conflit de nommage)

PHP met à disposition des **namespace** !

PHP Orientée Objet

Les namespace

Dans un vrai projet PHP, nous utilsons des **milliers** de class !

Afin d'éviter que 2 class soit nommé pareil
(le conflit de nommage)

PHP met à disposition des **namespace** !

Ces derniers permette de ranger nos class
dans des espaces de nommage prédéfinie
!

PHP Orientée Objet

Les namespace

Dans un vrai projet PHP, nous utilsons des **milliers** de class !

Afin d'éviter que 2 class soit nommé pareil
(le conflit de nommage)

PHP met à disposition des **namespace** !

Ces derniers permette de ranger nos class
dans des espaces de nommage prédéfinie
!

PHP Orientée Objet

Les namespace

Exercice : Déplaçons notre class Text dans un répertoire Util : `src/Util/Text.php` et ajoutons un namespace

```
● ● ●  
1 <?php  
2  
3 namespace Util;  
4  
5 class Text  
6 {  
7     static public function lineBreak(): void  
8     {  
9         echo "<br />";  
10    }  
11 }
```

PHP Orientée Objet

Les namespace

Indique à PHP que cette class se trouve dans l'espace de nom "Util"

```
1 <?php
2
3 namespace Util;
4
5 class Text
6 {
7     static public function lineBreak(): void
8     {
9         echo "<br />";
10    }
11 }
```

PHP Orientée Objet

Les namespace

Les namespace doivent correspondre au répertoire de notre application !

```
1 <?php
2
3 namespace Util;
4
5 class Text
6 {
7     static public function lineBreak(): void
8     {
9         echo "<br />";
10    }
11 }
```

PHP Orientée Objet

Les namespace

Utilisons notre class Text en spécifiant le namespace devant le nom de notre class !

```
1 <?php
2
3 spl_autoload_register(function ($class) {
4     ...
5 });
6
7 $merlin = new Magician('Merlin', 100, 40);
8 $arthur = new Personnage('Arthur', 100, 10);
9
10 $merlin->afficher();
11
12 Util\Text::lineBreak();
13
14 $arthur->afficher();
15
16 Util\Text::lineBreak();
```

PHP Orientée Objet

Les namespace

Retouchons maintenant notre autoload pour prendre en compte nos namespace



```
1 <?php
2
3 spl_autoload_register(function ($class) {
4     $path = str_replace('\\', '/', $class);
5     require_once __DIR__ . '/../src/' . $path . '.php';
6 });
```

PHP Orientée Objet

Les namespace

Nous remplaçons les \ par des / afin que le chemin de notre class soit correct

```
1 <?php
2
3 spl_autoload_register(function ($class) {
4     $path = str_replace('\\', '/', $class);
5     require_once __DIR__ . '/../src/' . $path . '.php';
6 });

● ● ●
```

PHP Orientée Objet

Les namespace

Nous incluons le fichier de notre class

```
1 <?php
2
3 spl_autoload_register(function ($class) {
4     $path = str_replace('\\', '/', $class);
5     require_once __DIR__ . '/../src/' . $path . '.php';
6 });
```

PHP Orientée Objet

Les namespace

Nous pouvons aussi "use" un namespace
(ce qui nous évite d'écrire le namespace
complete de la class à chaque fois)

```
1 use Util\Text;
2
3 $merlin = new Magician('Merlin', 100, 40);
4 $arthur = new Personnage('Arthur', 100, 10);
5
6 $merlin->afficher();
7
8 Text::lineBreak();
9
10 $arthur->afficher();
11
12 Text::lineBreak();
```

PHP Orientée Objet

Les namespace

En php il existe des suites de règles à suivre afin de produire un code « standard »

Ces normes sont les normes **PSR**

L'une de ces normes stipule que **TOUT notre code** doit être situé dans le namespace "**App**" !

PHP Orientée Objet

TP : Crédit d'un blog !