

# Classes et objets

Programmation Orientée Object (POO) en PHP

# Créer et manipuler les objets

Classes et objets

# Rappels - Créer un objet

- Pour créer un **objet**, il suffit d'**instancier** la **classe** sur laquelle on souhaite se baser à l'aide du mot-clé *new* à placer depuis le nom de la **classe**

```
$gandalf = new Personnage();
```

- Ainsi, la variable *\$gandalf* est un **objet** de type *Personnage* - On dit que l'on **instancie** la **classe** *Personnage*, que l'on crée une **instance** de cette même **classe**

# Rappels - Appeler les méthodes d'un objet

- Pour appeler une **méthode** d'**objet**, il suffit d'utiliser l'**opérateur PHP** -> - Il s'agit d'une flèche composée d'un tiret et d'un chevron fermant
- Pour l'utiliser correctement, on place l'**objet** depuis lequel on veut faire l'appel à gauche de l'opérateur et la **méthode** en question à droite (`$monObjet->maMethode()`)

```
1  <?php
2
3  class Personnage {
4      // Attributs
5      private $nom;
6      private $pointsDeVie;
7      private $pointsDeMagie;
8
9      // Méthodes
10     public function __construct($nom, $PV) {
11         $this->nom = $nom;
12         $this->pointsDeVie = $PV;
13         $this->pointDeMagie = 60;
14     }
15
16     public function attaque() {
17         echo "Le personnage attaque !";
18     }
19 }
20
21 $gandalf = new Personnage();
22 $gandalf->attaque();
23
24 ?>
```

# Explications

- Dans cet exemple, la ligne 22 récupère l'**objet** issue de la classe **Personnage** et stocké dans la variable `$gandalf` afin d'appeler la méthode `attaque()` déclarée dans sa **classe**
- En réalité l'opérateur `->` permet également d'accéder aux **attributs** d'une **classe**

# Accéder aux attributs d'un objet (1/2)

- L'opérateur `->` permet, effectivement d'accéder aux **attributs** d'une **classe** mais s'ils sont déclarés **privés**, une erreur se produira si nous tentons d'y accéder depuis l'extérieur de la **classe**

```
3  class Personnage {
4      // Attributs
5      private $nom;
6      private $pointsDeVie;
7      private $pointsDeMagie;
8
9      public function attaque() {
10         echo "Le personnage attaque !";
11     }
12 }
13
14 $gandalf = new Personnage();
15 $gandalf->nom = "Gandalf"; // Cette instruction génère une erreur
```

# Accéder aux attributs d'un objet (2/3)

- Nous tentons, ici, d'accéder à un **attribut privé** depuis l'extérieur de la **classe** - Nous avons vu précédemment que ce genre de pratique n'est pas permise - PHP lève une **erreur**
- Pour continuer de respecter le **principe d'encapsulation**, nous allons plutôt déclarer une **méthode** au sein de la **classe** réalisant les traitements souhaités - Ici, nous souhaitons renommer un personnage



# Accéder aux attributs d'un objet (3/3)

```
3  class Personnage {
4      // Attributs
5      private $nom;
6      private $pointsDeVie;
7      private $pointsDeMagie;
8
9      // Méthodes
10     public function renommer($nouveauNom) {
11         // Cette méthode doit renommer le personnage
12     }
13 }
14
15 $gandalf = new Personnage();
16 $gandalf->renommer("Gandalf");
```

# La pseudo-variable *\$this* (1/2)

- Désormais, nous souhaitons accéder à l'**attribut privé** *\$nom* depuis la **méthode**
- Dans l'exemple précédent, c'est la variable *\$gandalf* qui stocke l'**objet** de type *Personnage* à modifier - Nous savons appeler une **méthode** depuis cet **objet** mais pas encore comment le récupérer
- Pour ce faire, nous avons besoin d'utiliser la pseudo-variable *\$this*
- En réalité, l'**objet courant** *\$gandalf* est passé implicitement dans la **méthode** et stocké dans la **pseudo-variable** *\$this*

# La pseudo-variable *\$this* (2/2)

```
3  class Personnage {
4      // Attributs
5      private $nom;
6      private $pointsDeVie;
7      private $pointsDeMagie;
8
9      // Méthodes
10     public function renommer($nouveauNom) {
11         $this->nom = $nouveauNom;
12     }
13 }
14
15 $gandalf = new Personnage();
16 $gandalf->renommer("Gandalf");
```

# Explications

- Cet exemple comporte une seule nouveauté : la ligne 11
- Cette simple ligne récupère l'**objet courant** *\$this*, soit l'**objet** depuis lequel la méthode est appelée - Ici, il s'agit de *\$gandalf*
- Elle récupère le nom de l'**objet courant** *\$this->nom* - Pour le moment, l'**objet** n'a pas de nom
- Enfin, elle affecte la valeur stockée dans la variable *\$nouveauNom* au nom de cet **objet** - Désormais l'**objet** *\$gandalf* a le nom « *Gandalf* »

# Les accesseurs et mutateurs

Classes et objets

# Accéder à un attribut via un accesseur (getter)

- Nous avons vu que le **principe d'encapsulation** exige que seule la **classe** elle-même est capable de lire ou modifier un **attribut**
- Ainsi, pour accéder à un **attribut**, nous allons simplement ajouter une **méthode** à la **classe** dont le seul rôle consiste à retourner la valeur de l'**attribut** souhaité
- Ce type de **méthode** s'appelle **accesseur** (ou **getter**)
- Par convention, nous nommons ces **méthodes** avec le même nom que l'**attribut** à retourner précédé du mot-clé **get** - Dans le cas de l'**attribut** `$nom`, nous aurons l'**accesseur** `getNom()`
- Ajoutons tous les **getters** nécessaires à notre **classe** *Personnage*

```

3  class Personnage {
4      // Attributs
5      private $nom;
6      private $pointsDeVie;
7      private $pointsDeMagie;
8
9      // Méthodes
10     public function renommer($nouveauNom) {
11         |     $this->nom = $nouveauNom;
12     }
13
14     public function getNom() {
15         |     return $this->nom;
16     }
17
18     public function getPointsDeVie() {
19         |     return $this->pointsDeVie;
20     }
21
22     public function getPointsDeMagie() {
23         |     return $this->pointsDeMagie;
24     }
25 }

```



# Modifier un attribut via un mutateur (setter)

- Ainsi, pour modifier un **attribut**, nous allons ajouter une **méthode** à la **classe** dont le seul rôle consiste de mettre à jour la valeur de l'**attribut** souhaité avec une valeur passée en argument
- Ce type de **méthode** s'appelle **mutateur** (ou **setter**)
- Par convention, nous nommons ces **méthodes** avec le même nom que l'**attribut** à retourner précédé du mot-clé **set** - Dans le cas de l'**attribut** *\$nom*, nous aurons le **mutateur** *setNom()* - En réalité, la **méthode** *renommer()* est un **setter** de l'**attribut** *\$nom*
- Ajoutons tous les **setters** nécessaires à notre **classe** *Personnage*
- Ces **méthodes** sont souvent utilisées pour réaliser des vérifications avant modification de l'**attribut** ciblé



```

3  class Personnage {
4      // Attributs
5      private $nom;
6      private $pointsDeVie;
7      private $pointsDeMagie;
8
9      // Méthodes
10     public function getNom() {
11         return $this->nom;
12     }
13
14     public function setNom($nom) {
15         $this->nom = $nom;
16     }
17
18     public function getPointsDeVie() {
19         return $this->pointsDeVie;
20     }
21
22     public function setPointsDeVie($pointsDeVie) {
23         if($pointsDeVie > 0 && $pointsDeVie < 200) {
24             $this->pointsDeVie = $pointsDeVie;
25         }
26     }
27
28     public function getPointsDeMagie() {
29         return $this->pointsDeMagie;
30     }
31
32     public function setPointsDeMagie($pointsDeMagie) {
33         if($pointsDeMagie > 0 && $pointsDeMagie < 100) {
34             $this->pointsDeMagie = $pointsDeMagie;
35         }
36     }
37 }

```

# Le constructeur

Classes et objets

# Le constructeur en quelques mots...

- La **méthode publique constructeur** (`__construct()`) d'une **classe** est appelée implicitement à la **création** d'un **objet** issu de celle-ci
- Cela peut être très utile pour réaliser certains traitements à l'**instanciation de la classe** comme pour lui donner des **valeurs initiales**, par exemple
- Un **constructeur** peut prendre 0, 1 ou plusieurs **arguments**
- On utilise généralement ces **paramètres** pour initialiser l'**objet** avec certaines **valeurs**
- Le constructeur doit également respecter le **principe d'encapsulation** pour garder une cohérence dans les valeurs passées en **argument**

```

3  class Personnage {
4      // Attributs
5      private $nom;
6      private $pointsDeVie;
7      private $pointsDeMagie;
8
9      // Méthodes
10     public function __construct($nom, $PV) {
11         $this->setNom($nom);
12         $this->setPointsDeVie($PV);
13         $this->setPointsDeMagie(80);
14     }
15
16     public function setNom($nom) {
17         $this->nom = $nom;
18     }
19
20     public function setPointsDeVie($pointsDeVie) {
21         if($pointsDeVie > 0 && $pointsDeVie < 200) {
22             $this->pointsDeVie = $pointsDeVie;
23         }
24     }
25
26     public function setPointsDeMagie($pointsDeMagie) {
27         if($pointsDeMagie > 0 && $pointsDeMagie < 100) {
28             $this->pointsDeMagie = $pointsDeMagie;
29         }
30     }
31 }
32
33 $gandalf = new Personnage("Gandalf", 120);

```

# L'auto-chargement de classes

Classes et objets

# Charger une classe depuis un autre fichier

- Par convention et soucis d'organisation, un **fichier PHP** ne devra contenir qu'une seule **classe** nommée de la même manière que le fichier dans lequel elle est écrite - Ainsi, le fichier *Personnage.php* contient uniquement la **classe** *Personnage*
- De cette manière, pour utiliser cette **classe** depuis un autre fichier, je n'ai qu'à l'inclure à l'aide de la **fonction PHP** *require()*, par exemple

```
1  <?php
2  require "Personnage.php";
3
4  $aragorn = new Personnage("Aragorn", 150);
5  echo $aragorn->getNom();
6
7  ?>
```

# Fonction de chargement

- Ce fonctionnement ne pose pas de problème pour une unique **classe** - En revanche, si nous avons des dizaines de **classes**, ça peut être plus compliqué à gérer
- Heureusement, on va pouvoir se baser sur l'**auto-chargement des classes** pour ne pas avoir à écrire une multitude d'inclusions de fichier à la main
- Pour ce faire, il suffit de déclarer une **fonction de chargement** dans le script principal du projet - Cette **fonction** accepte un **paramètre** unique correspondant au nom de la **classe** à charger

```
function chargerClasse($classe) {  
    require $classe . '.php';  
}
```



# L'auto-chargement de classes

- Une fois déclarée, il suffit de passer le nom de cette **fonction** en **argument** de la **fonction PHP** `spl_autoload_register()` pour l'enregistrer en **autoload** et ainsi l'appeler dès l'**instanciation d'une classe** pas encore déclarée

```
1  <?php
2
3  function chargerClasse($classe) {
4      |   require $classe . '.php';
5  }
6
7  spl_autoload_register('chargerClasse');
8
9  $aragorn = new Personnage("Aragorn", 150);
10 echo $aragorn->getNom();
11
12 ?>
```



# Explications

- **PHP** comporte une **pile d'autoloads** contenant une liste de **fonctions** appelées automatiquement lorsque l'on tente d'**instancier une classe** non déclarée
- Ces **fonctions** sont appelées des **chargeurs automatiques** et doivent permettre d'inclure la (ou les) **classe(s) à instancier**
- La **fonction PHP** `spl_autoload_register()` permet d'ajouter ces **chargeurs** dans la **pile d'autoloads**
- Notez qu'elles sont appelées dans l'ordre dans lequel elles sont ajoutées

# TP - Un maître et son chien

Classes et objets

# TP - La classe Chien

- Reprenez la classe *Chien* pour permettre à ses **méthodes** de **renommer** et **faire grandir** un chien correctement - Attention, un chien ne peut QUE grandir !
- Que représentent ces 2 **méthodes** ? Écrivez une réponse courte en **commentaire** dans la **classe**
- Prévoyez une **méthode** permettant d'afficher les informations d'un **chien** (Exemple : « *Pluto mesure 60 cm* »)

# TP - La classe maître

- Dans le même dossier que le fichier **Chien.php**, écrivez un fichier « **Maitre.php** » contenant une classe **Maître**
- Un **maître** est caractérisé par un **nom**, un **prénom**, un **âge** et un **chien**
- Un **maître** n'a **pas de chien au départ** mais peut en **adopter**
- Prévoyez une **méthode** permettant d'afficher les informations d'un **maître** (Exemple : « **Chris Chevalier a 28 ans et un chien : Pluto mesure 60 cm** »)

# TP - Un maître et son chien

- Les **constructeurs** des 2 classes doivent permettre d'initialiser leurs **attributs** avec des valeurs passées en **paramètre**
- Prévoyez des **getters** et **setters** pour chacun des **attributs** des 2 **classes**
- Tester et affichez le résultat de vos **méthodes** dans un fichier *Main.php* représentant le programme principal

# TP - Un maître et son chien

- Envoyez vos fichiers *Chien.php*, *Maitre.php* et *Main.php* à l'adresse email [chevalier@chris-freelance.com](mailto:chevalier@chris-freelance.com)

# Des questions ?