Développement PHP
Partie 7 : Programmation Orientée Objet

DENIS LOEUILLET – IFA - 2017

Partie 7 : Programmation Orientée Objet
Les bases de la POO

1. Introduction à la POO
2. Utiliser la classe
3. L'opérateur de résolution de portée
4. Manipulation de données stockées
5. L'héritage
6. Les méthodes magiques

Partie 7 : Programmation Orientée Objet
Les bases de la POO

2. Utiliser la classe

- Une classe, c'est bien mais, au même titre qu'un plan de construction pour une maison :
  - si on ne sait pas comment se servir de notre plan pour construire notre maison, cela ne sert à rien !
  - Nous verrons donc comment se servir de :
    - ✓ notre classe,
    - ✓ notre modèle de base,
    - ✓ afin de créer des objets et pouvoir s'en servir.
- Ce chapitre sera aussi l'occasion d'approfondir un peu le développement de nos classes :
  - actuellement, la classe que l'on a créée contient des méthodes vides, chose un peu inutile

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe

2. Utiliser la classe : Créer et manipuler un objet

- Créer un objet
  - ➢ Nous allons voir comment créer un objet :
    - ✓ c'est-à-dire que nous allons utiliser notre classe afin qu'elle nous fournisse un objet.
    - ✓ Pour créer un nouvel objet, vous devez faire précéder le nom de la classe à instancier du mot-clé « new » :
      - ❖ $perso sera un objet de type Personnage.
      - ❖ On dit que l'on instancie la classe Personnage,
      - ❖ que l'on crée une instance de la classe Personnage.

```php
<?php
$perso = new Personnage;
```

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe

2. Utiliser la classe : Créer et manipuler un objet

- Appeler les méthodes de l'objet
  - Pour appeler une méthode d'un objet :
    - ✓ il va falloir utiliser un opérateur :
      - ❖ il s'agit de l'opérateur « -> » (une flèche composée d'un tiret suivi d'un chevron fermant).
      - ❖ Celui-ci s'utilise de la manière suivante :
        - ○ À gauche de cet opérateur, on place l'objet que l'on veut utiliser.
        - ○ Dans l'exemple pris juste au-dessus, cet objet aurait été $perso.
        - ○ À droite de l'opérateur, on spécifie le nom de la méthode que l'on veut invoquer.

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe – créer et manipuler un objet

- La ligne 18 signifie donc « va chercher l'objet $perso, et invoque la méthode parler() sur cet objet ».
- Notez donc bien quelque part l'utilisation de cet opérateur : de manière générale, il sert à accéder à un élément de la classe.
- Ici, on s'en est servi pour atteindre une méthode, mais nous verrons plus tard qu'il nous permettra aussi d'atteindre un attribut.

```php
<?php
// Nous créons une classe « Personnage ».
class Personnage
{
  private $_force;
  private $_localisation;
  private $_experience;
  private $_degats;

  // Nous déclarons une méthode dont le seul but est d'afficher un texte.
  public function parler()
  {
    echo 'Je suis un personnage !';
  }
}

$perso = new Personnage;
$perso->parler();
```

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe

2. Utiliser la classe : Créer et manipuler un objet

- Accéder à un élément depuis la classe
  - On vient de créer un objet,
  - et vous êtes maintenant capables d'appeler une méthode.
  - Cependant, le contenu de cette dernière était assez simpliste :
    - ✓ elle ne faisait qu'afficher un message.
    - ✓ Ici, nous allons voir comment une méthode peut accéder aux attributs de l'objet.
  - Lorsque vous avez un objet :
    - ✓ vous savez que vous pouvez invoquer des méthodes grâce à l'opérateur «->».
    - ✓ c'est également grâce à cet opérateur qu'on accéde aux attributs de la classe.
    - ✓ Cependant, rappelez-vous : nos attributs sont privés.
      - ❖ Par conséquent, ce code lèvera une erreur fatale :

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe – créer et manipuler un objet

- Ici, on essaye d'accéder à un attribut privé hors de la classe.
- Ceci est interdit, donc PHP lève une erreur.
- Dans notre exemple (qui essaye en vain d'augmenter de 1 l'expérience du personnage), il faudra demander à la classe d'augmenter l'expérience.
- Pour cela, nous allons écrire une méthode gagnerExperience():

```php
<?php
class Personnage
{
  private $_force;
  private $_experience;
  private $_degats;
}

$perso = new Personnage;
$perso->_experience = $perso->_experience + 1; // Une erreur fatale est levée suite à cette instruction.
```

```php
<?php
class Personnage
{
  private $_experience;

  public function gagnerExperience()
  {
    // Cette méthode doit ajouter 1 à l'expérience du personnage.
  }
}

$perso = new Personnage;
$perso->gagnerExperience();
```

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe – créer et manipuler un objet

- Accéder à un élément depuis la classe
  - comment accéder à l'attribut $_experience dans notre méthode ?
    - ✓ C'est là qu'intervient la pseudo-variable $this.
  - Dans notre script, $perso représente l'objet.
  - Il est donc facile d'appeler une méthode à partir de cette variable.
  - Mais dans notre méthode, nous n'avons pas cette variable pour accéder à notre attribut $_experience pour le modifier !
  - En fait, un paramètre représentant l'objet est passé implicitement à chaque méthode de la classe.

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe – créer et manipuler un objet

- Accéder à un élément depuis la classe
  - ligne 8 :
    - ✓ la variable $this dont je vous ai parlé représente l'objet que nous sommes en train d'utiliser.
    - ✓ Ainsi, dans ce script, les variables $this et $perso représentent le même objet.
    - ✓ L'instruction surlignée veut donc dire : « Affiche-moi cette valeur : dans l'objet utilisé (donc $perso), donne-moi la valeur de l'attribut $_experience. »
    - ✓ Ainsi, je vais vous demander d'écrire la méthode gagnerExperience().
      - ❖ Cette méthode devra ajouter 1 à l'attribut $_experience.

```php
<?php
class Personnage
{
  private $_experience = 50;

  public function afficherExperience()
  {
    echo $this->_experience;
  }
}

$perso = new Personnage;
$perso->afficherExperience();
```

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe – créer et manipuler un objet

```php
<?php
class Personnage
{
  private $_experience = 50;

  public function afficherExperience()
  {
    echo $this->_experience;
  }

  public function gagnerExperience()
  {
    // On ajoute 1 à notre attribut $_experience.
    $this->_experience = $this->_experience + 1;
  }
}

$perso = new Personnage;
$perso->gagnerExperience();   // On gagne de l'expérience.
$perso->afficherExperience(); // On affiche la nouvelle valeur de l'attribut.
```

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe – créer et manipuler un objet

- Implémenter d'autres méthodes
  - Nous avons créé :
    - ✓ notre classe Personnage
    - ✓ et une méthode, gagnerExperience().
  - Nous allons implémenter une autre méthode :
    - ✓ la méthode frapper(), qui devra infliger des dégâts à un personnage.

```php
<?php
class Personnage
{
  private $_degats = 0; // Les dégâts du personnage.
  private $_experience = 0; // L'expérience du personnage.
  private $_force = 20; // La force du personnage (plus elle est grande, plus l'attaque est puissante).

  public function gagnerExperience()
  {
    // On ajoute 1 à notre attribut $_experience.
    $this->_experience = $this->_experience + 1;
  }
}
```

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe – créer et manipuler un objet

- Implémenter d'autres méthodes
  - Commençons par écrire notre méthode frapper().
    - ✓ Cette méthode doit accepter un argument : le personnage à frapper.
    - ✓ La méthode aura juste pour rôle d'augmenter les dégâts du personnage passés en paramètre.
    - ✓ Pour vous aider à visualiser le contenu de la méthode, imaginez votre code manipulant des objets. Il doit ressembler à ceci :

```php
<?php
// On crée deux personnages
$perso1 = new Personnage;
$perso2 = new Personnage;

// Ensuite, on veut que le personnage n°1 frappe le personnage n°2.
$perso1->frapper($perso2);
```

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe – créer et manipuler un objet

- Implémenter d'autres méthodes
  - Pour résumer :
    - ✓ la méthode frapper() demande un argument : le personnage à frapper ;
    - ✓ cette méthode augmente les dégâts du personnage à frapper en fonction de la force du personnage qui frappe.

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe – créer et manipuler un objet

```php
<?php
class Personnage
{
  private $_degats; // Les dégâts du personnage.
  private $_experience; // L'expérience du personnage.
  private $_force; // La force du personnage (plus elle est grande, plus l'attaque est puissante).

  public function frapper($persoAFrapper)
  {
    $persoAFrapper->_degats += $this->_force;
  }

  public function gagnerExperience()
  {
    // On ajoute 1 à notre attribut $_experience.
    $this->_experience = $this->_experience + 1;
  }
}
```

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe – créer et manipuler un objet

- Implémenter d'autres méthodes
  - Le contenu de cette méthode frapper() comporte une instruction composée de deux parties :
    1. La première consiste à dire à PHP que l'on veut assigner une nouvelle valeur à l'attribut $_degats du personnage à frapper.
    2. La seconde partie consiste à donner à PHP la valeur que l'on veut assigner. Ici, nous voyons que cette valeur est atteinte par $this->_force.
  - Question :
    - ✓ la variable $this fait-elle référence au personnage qui frappe ou au personnage frappé ?
      - ❖ Il faut savoir sur quel objet est appelée la méthode.
      - ❖ En effet, souvenez-vous que $this est une variable représentant l'objet à partir duquel on a appelé la méthode.
      - ❖ Dans notre cas, on a appelé la méthode frapper() à partir du personnage qui frappe, donc $this représente le personnage qui frappe.

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe – créer et manipuler un objet

- Implémenter d'autres méthodes
  - L'instruction contenue dans la méthode signifie donc : « Ajoute la valeur de la force du personnage qui frappe à l'attribut $_degats du personnage frappé. »
  - Maintenant, nous pouvons créer une sorte de petite mise en scène qui fait interagir nos personnages.
    - ✓ Par exemple, nous pouvons créer un script qui fait combattre les personnages.
    - ✓ Le personnage 1 frapperait le personnage 2 puis gagnerait de l'expérience,
    - ✓ puis le personnage 2 frapperait le personnage 1 et gagnerait de l'expérience. Procédez étape par étape :
      - ❖ créez deux personnages ;
      - ❖ faites frapper le personnage 1 ;
      - ❖ faites gagner de l'expérience au personnage 1 ;
      - ❖ faites frapper le personnage 2 ;
      - ❖ faites gagner de l'expérience au personnage 2.
  - Ce script n'est qu'une suite d'appels de méthodes.
    - ✓ Chaque puce (sauf la première) correspond à l'appel d'une méthode, ce que vous savez faire.

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe – créer et manipuler un objet

```php
<?php
$perso1 = new Personnage; // Un premier personnage
$perso2 = new Personnage; // Un second personnage

$perso1->frapper($perso2); // $perso1 frappe $perso2
$perso1->gagnerExperience(); // $perso1 gagne de l'expérience

$perso2->frapper($perso1); // $perso2 frappe $perso1
$perso2->gagnerExperience(); // $perso2 gagne de l'expérience
```

- Implémenter d'autres méthodes
  - Problème :
    - ✓ À la base, les deux personnages ont :
      - ❖ le même niveau de dégâts,
      - ❖ la même expérience
      - ❖ et la même force,

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe – créer et manipuler un objet

- Implémenter d'autres méthodes
  - Problème :
    - ✓ À la base, les deux personnages ont :
      - ❖ le même niveau de dégâts,
      - ❖ la même expérience
      - ❖ et la même force,
    - ✓ ils seront à la fin toujours égaux.
    - ✓ Pour pallier ce problème, il faudrait pouvoir assigner des valeurs spécifiques aux deux personnages, afin que le combat puisse les différencier.
    - ✓ Or, vous ne pouvez pas accéder aux attributs en-dehors de la classe !
    - ✓ Pour savoir comment résoudre ce problème, je vais vous apprendre deux nouveaux mots :
      - ❖ accesseur
      - ❖ et mutateur

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe – créer et manipuler un objet

- Petite parenthèse : exiger des objets en paramètre
  - ➢ Exemple de la méthode frapper() :
    - ✓ Celle-ci accepte un argument : un personnage à frapper.
    - ✓ Cependant, qu'est-ce qui vous garantit qu'on passe effectivement un personnage à frapper ?
      - ❖ On pourrait très bien passer un argument complètement différent, comme un nombre par exemple :

```php
<?php
$perso = new Personnage;
$perso->frapper(42);
```

- Que se passe-t-il?
  - Une erreur est générée car, à l'intérieur de la méthode frapper(), nous essayons d'appeler une méthode sur le paramètre qui n'est pas un objet.
  - C'est comme si on avait fait ça :

```php
<?php
$persoAFrapper = 42;
$persoAFrapper->_degats += 50; // Le nombre 50 est arbitraire, il est censé représenter une force.
```

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe – créer et manipuler un objet

- Petite parenthèse : exiger des objets en paramètre
  - Exemple de la méthode frapper() :
    - Ce qui n'a aucun sens.
    - Il faut donc s'assurer que le paramètre passé est bien un personnage, sinon PHP arrête tout et n'exécute pas la méthode.
    - Pour cela, il suffit d'ajouter un seul mot : le nom de la classe dont le paramètre doit être un objet.
    - Dans notre cas, si le paramètre doit être un objet de type Personnage, alors il faudra ajouter le mot-clé Personnage, juste avant le nom du paramètre, comme ceci :
    - Grâce à ça, vous êtes sûrs que la méthode frapper() ne sera exécutée que si le paramètre passé est de type Personnage,
    - sinon PHP interrompt tout le script.
    - Vous pouvez donc appeler les méthodes de l'objet sans crainte qu'un autre type de variable soit passé en paramètre.

```php
<?php
class Personnage
{
  // …

  public function frapper(Personnage $persoAFrapper)
  {
    // …
  }
}
```

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe

2. Utiliser la classe : Les accesseurs et mutateurs

- Le principe d'encapsulation nous empêche d'accéder directement aux attributs de notre objet puisqu'ils sont privés :
  - seule la classe peut les lire et les modifier.
  - Par conséquent, si vous voulez récupérer un attribut, il va falloir le demander à la classe, de même si vous voulez les modifier.

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe – accesseurs et mutateurs

- Accéder à un attribut : l'accesseur
  - Comment faire pour récupérer la valeur d'un attribut ?
    - ✓ nous allons implémenter des méthodes dont le seul rôle sera de nous donner l'attribut qu'on leur demande !
    - ✓ Ces méthodes ont un nom bien spécial : ce sont des accesseurs (ou getters).
    - ✓ Par convention, ces méthodes portent le même nom que l'attribut dont elles renvoient la valeur.
      - ❖ Par exemple, voici la liste des accesseurs de notre classe Personnage : accesseurs.php

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe – accesseurs et mutateurs

- Modifier la valeur d'un attribut : les mutateurs
  - Comment modifier un attribut ?
    - ✓ Il va falloir demander à la classe de le faire pour nous :
      - ❖ Le principe d'encapsulation est là pour vous empêcher d'assigner un mauvais type de valeur à un attribut :
        - si vous demandez à votre classe de le faire, ce risque est supprimé car la classe « contrôle » la valeur des attributs.
        - Ce sera par le biais de méthodes que l'on demandera à notre classe de modifier tel attribut.
    - ✓ La classe doit impérativement contrôler la valeur afin d'assurer son intégrité car :
      - ❖ si elle ne le fait pas, on pourra passer n'importe quelle valeur à la classe et le principe d'encapsulation n'est plus respecté !
      - ❖ Ces méthodes ont aussi un nom spécial : il s'agit de mutateurs (ou setters).
      - ❖ Ces méthodes sont de la forme setNomDeLAttribut().
      - ❖ Voici la liste des mutateurs (ajoutée à la liste des accesseurs) de notre classe Personnage: accesseurs_mutateurs.php

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe – accesseurs et mutateurs

- Retour sur notre script de combat
  - Exercice : afficher_attributs_personnage.php
    - ✓ afficher, à la fin du script :
      - ❖ la force,
      - ❖ l'expérience
      - ❖ et le niveau de dégâts de chaque personnage.
    - ✓ Comme nous l'avions dit, les valeurs finales des deux personnages sont identiques.
      - ❖ Pour pallier ce problème, nous allons modifier, juste après la création des personnages :
        - la valeur de la force
        - et de l'expérience des deux personnages.
      - ❖ Vous pouvez par exemple favoriser un personnage en lui donnant une plus grande force et une plus grande expérience par rapport au deuxième. (les valeurs n'ont aucune importance ce n'est qu'un exemple) : modifier_attributs_personnage.php

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe – accesseurs et mutateurs

- Retour sur notre script de combat
  - ➢ Exercice : afficher_attributs_personnage.php
    - ✓ Ce script va afficher :

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe

2. Utiliser la classe : le constructeur

- Imaginez un objet pour lequel vous avez besoin d'initialiser les attributs dès sa création, sans connaitre leurs valeurs à l'avance.
- Par exemple, vous souhaiteriez pouvoir spécifier la force et les dégâts d'un personnage dès sa création.
- Actuellement, la seule possibilité qui s'offre à vous est de modifier ces attributs manuellement, une fois l'objet créé.
- Il existe en PHP une méthode, appelée le constructeur, qui remplira ce rôle.
- Ce constructeur n'est autre qu'une méthode écrite dans votre classe.

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe – le constructeur

- Constructeur de la classe Personnage : constructeur.php
  - Le constructeur ne peut pas avoir n'importe quel nom (sinon, comment PHP sait quel est le constructeur ?).
  - Il a le nom suivant : __construct, avec deux underscores au début.
  - Comme son nom l'indique, le constructeur sert à construire l'objet.
    - ✓ si des attributs doivent être initialisés ou qu'une connexion à la BDD doit être faite :
      - ❖ C'est dans le constructeur que ça va se faire.
    - ✓ Le constructeur est exécuté dès la création de l'objet :
      - ❖ aucune valeur ne doit être retournée, même si ça ne génèrera aucune erreur.
    - ✓ une classe fonctionne très bien sans constructeur, il n'est en rien obligatoire !
      - ❖ Si vous n'en spécifiez pas, cela revient au même que si vous en aviez écrit un vide (sans instruction à l'intérieur).

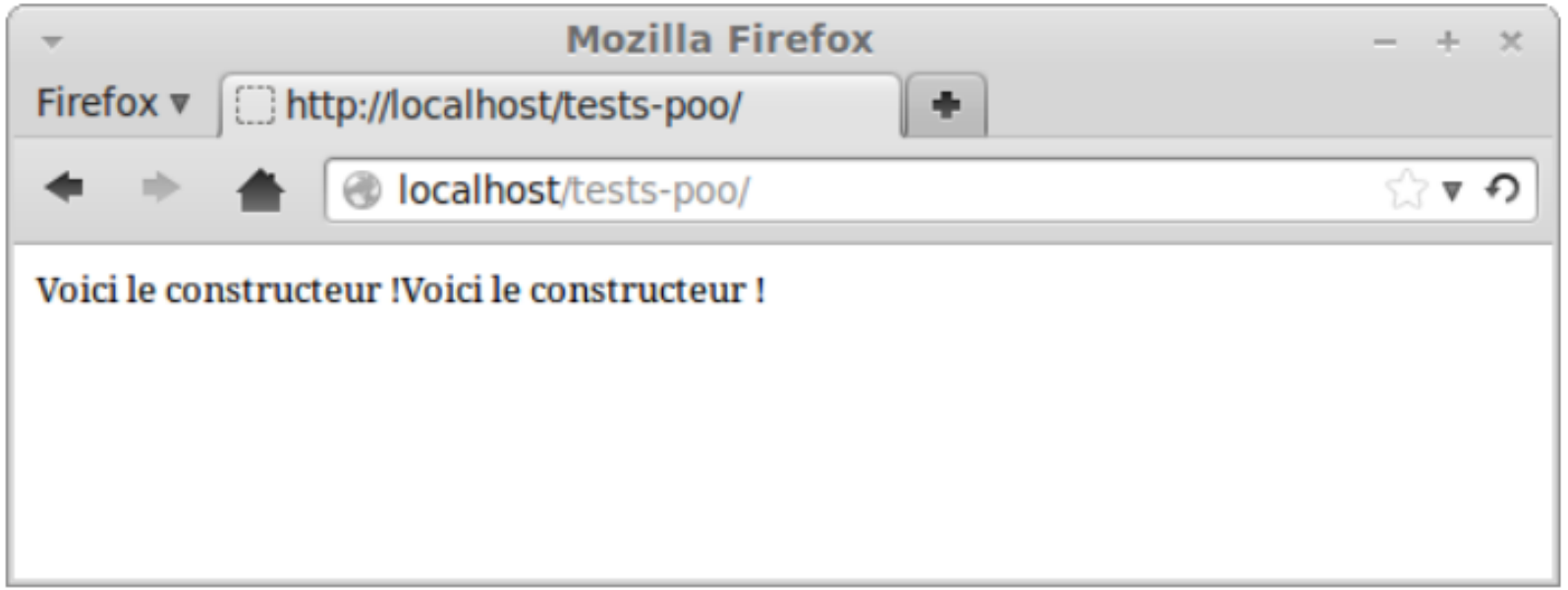
Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe – le constructeur

- Constructeur de la classe Personnage : constructeur.php
  - le constructeur demande :
    - ✓ la force et les dégâts initiaux du personnage que l'on vient de créer.
  - Il faudra donc lui spécifier ceci en paramètre :

```php
<?php
$perso1 = new Personnage(60, 0); // 60 de force, 0 dégât
$perso2 = new Personnage(100, 10); // 100 de force, 10 dégâts
```

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe – le constructeur

- Constructeur de la classe Personnage : constructeur.php
  - Résultat affiché par le script



Résultat affiché par le script

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe – le constructeur

- Constructeur de la classe Personnage : constructeur.php
  - Notez quelque chose d'important dans la classe :
    - ✓ dans le constructeur, les valeurs sont initialisées en appelant les mutateurs correspondant.
    - ✓ En effet, si on assignait directement ces valeurs avec les arguments, le principe d'encapsulation ne serait plus respecté et n'importe quel type de valeur pourrait être assigné !
  - Remarque :
    - ✓ Ne mettez jamais la méthode __construct avec le type de visibilité private car :
      - ❖ elle ne pourra jamais être appelée,
      - ❖ vous ne pourrez donc pas instancier votre classe !
      - ❖ Cependant, sachez qu'il existe certains cas particuliers qui nécessitent le constructeur en privé

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe

2. Utiliser la classe : l'auto-chargement de classes
   - Pour une question d'organisation, il vaut mieux créer un fichier par classe.
   - Vous appelez votre fichier comme bon vous semble et placez votre classe dedans.
   - Vous pouvez nommer vos fichiers « MaClasse.php ».
     - Ainsi, si on veut pouvoir utiliser la classe MaClasse, il n'y aura qu'à inclure ce fichier :

```php
<?php
require 'MaClasse.php'; // J'inclus la classe.

$objet = new MaClasse; // Puis, seulement après, je me sers de ma classe.
```

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe – auto-chargement de classes

- Imaginons que vous ayez plusieurs dizaines de classes :
  - Pas très pratique de les inclure une par une !
  - Vous vous retrouverez avec des dizaines d'inclusions, certaines pouvant même être inutile si vous ne vous servez pas de toutes vos classes.
  - Et c'est là qu'intervient l'auto-chargement des classes.
  - Vous pouvez créer dans votre fichier principal (c'est-à-dire celui où vous créerez une instance de votre classe) :
    - ✓ une ou plusieurs fonction(s) qui tenteront de charger le fichier déclarant la classe.
    - ✓ Dans la plupart des cas, une seule fonction suffit.
    - ✓ Ces fonctions doivent accepter un paramètre, c'est le nom de la classe qu'on doit tenter de charger.
    - ✓ Par exemple, voici une fonction qui aura pour rôle de charger les classes :

```php
<?php
function chargerClasse($classe)
{
  require $classe . '.php'; // On inclut la classe correspondante au paramètre passé.
}
```

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe – auto-chargement de classes

- Essayons maintenant de créer un objet pour voir si il sera chargé automatiquement :
  - Exemple de la classe Personnage : script autoload.php
    - ✓ On prends en compte le fait qu'un fichier Personnage.php existe

```php
<?php
function chargerClasse($classe)
{
  require $classe . '.php'; // On inclut la classe correspondante au paramètre passé.
}

$perso = new Personnage(); // Instanciation de la classe Personnage qui n'est pas déclarée dans ce fichier.
```

      - ❖ Que se passe-t-il ?
        - Erreur fatale ! La classe n'a pas été trouvée, elle n'a donc pas été chargée…
        - PHP ne sait pas qu'il doit appeler cette fonction lorsqu'on essaye d'instancier une classe non déclarée.
        - On va donc utiliser la fonction spl_autoload_register en spécifiant en premier paramètre le nom de la fonction à charger :

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe – auto-chargement de classes

- Essayons maintenant de créer un objet pour voir si il sera chargé automatiquement :
  - Exemple de la classe Personnage
    - ✓ On prends en compte le fait qu'un fichier Personnage.php existe

```php
<?php
function chargerClasse($classe)
{
  require $classe . '.php'; // On inclut la classe correspondante au paramètre passé.
}

spl_autoload_register('chargerClasse'); // On enregistre la fonction en autoload pour qu'elle soit appelée dès qu'on instanciera une classe non déclarée.

$perso = new Personnage;
```

      - ❖ Que se passe-t-il ?
        - aucune erreur ne s'affiche ! Notre auto-chargement a donc bien fonctionné.

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe – auto-chargement de classes

- Essayons maintenant de créer un objet pour voir si il sera chargé automatiquement :
  - Exemple de la classe Personnage
    - ✓ On prends en compte le fait qu'un fichier Personnage.php existe
      - ❖ Que s'est-il passé ?
        - En PHP, il y a ce qu'on appelle une « pile d'autoloads ».
        - Cette pile contient une liste de fonctions.
        - Chacune d'entre elles sera appelée automatiquement par PHP lorsque l'on essaye d'instancier une classe non déclarée.
        - Nous avons donc ici ajouté notre fonction à la pile d'autoloads afin qu'elle soit appelée à chaque fois qu'on essaye d'instancier une classe non déclarée.

Partie 7 : Programmation Orientée Objet
Les bases de la POO : utiliser la classe

- Résumé :
  - Un objet se crée grâce à l'opérateur new.
  - L'accès à un attribut ou à une méthode d'un objet se fait grâce à l'opérateur «->».
  - Pour lire ou modifier un attribut, on utilise des accesseurs et des mutateurs.
  - Le constructeur d'une classe a pour rôle principal d'initialiser l'objet en cours de création, c'est-à-dire d'initialiser la valeur des attributs (soit en assignant directement des valeurs spécifiques, soit en appelant diverses méthodes).
  - Les classes peuvent être chargées dynamiquement (c'est-à-dire sans avoir explicitement inclus le fichier la déclarant) grâce à l'auto-chargement de classe (utilisation de spl_autoload_register).