PHP

Programmation orientée objet et bases de données

La POO

Classe / Objet

Qu'est-ce qu'une classe ?

- Une classe peut être vue comme une abstraction.
- Elle contient des propriétés et des méthodes.
- L'environnement de la classe est cloisonné.

Qu'est-ce qu'un objet ?

- Un objet est une instance de classe
- Avec une classe, on peut créer autant d'objets qu'on veut (sauf si vous décidé du contraire)
- Une classe est une déclaration, alors qu'un objet est contenu dans une variable
- Un objet est une instance de ma classe

```php
class Account
{
    public $id;

    protected $orders = [];

    public static function fromId(integer $id)
    {
      // ...
    }

    public function getPersonages(array $options)
    {
      // ...
    }

    private function log(integer $level, string $message) : void
    {
      // ...
    }
}
```

Exemple de création d'objets

Dans l'exemple ci-joint, nous créons 2 objets Account qui représente chacun le compte d'un utilisateur. C'est deux objet vont avoir l'ensemble des attributs déclaré au niveau de la classe, ainsi que des différente déclaration réalisé. Toutefois, les attributs seront individualisé.

```php
class Account
{
    public $id;
    public $username;
}

// Premier compte
$account1 = new Account();
// Second compte
$account2 = new Account();
// Gestion d'un compte
$account1->id = 123;
$account1->username = 'david';
$account2->id = 124;
$account2->username = 'solene';
```

Déclaration des attributs

Pour déclarer un attribut il faut le précéder par sa **visibilité**. La visibilité d'un attribut indique à partir d'où on peut y avoir accès. Il existe trois types de visibilité:

- **public**: dans ce cas, l'attribut est accessible de partout (de l'intérieur de la classe dont il est membre comme de l'extérieur).
- **private**: dans ce cas, l'attribut est accessible seulement de l'intérieur de la classe dont il est membre.
- **protected**: dans ce cas, l'attribut est accessible seulement de l'intérieur de la classe dont il est membre ainsi que de l'intérieur des classes fille qui héritent de cette classe.

Cependant, c'est la visibilité **private** (ou **protected** qui est considérée comme extension de **private**) qui est recommandée. On parle alors du **principe d'encapsulation.**

Principe d'encapsulation

L'encapsulation est un principe fondamental de la POO. Il vise à masquer les attributs aux utilisateurs du code (les programmeurs qui se serviront de la classe par la suite). En fait, ce qui est important dans une classe ce sont les attributs. Les méthodes ne font qu'agir sur ceux ci. Le fait d'exposer les attributs aux utilisateur peut compromettre le bon fonctionnement de la classe. Il faut donc les masquer et leur limiter l'accès uniquement de l'intérieur de la classe par le biais des méthodes prévues à cet effet.

Pour mieux comprendre comment la manipulation directe des attributs peut compromettre le bon fonctionnement de la classe, il n'y a pas mieux qu'un exemple.

Exemple

```php
class Personage
{
    protected $health;

    public function useObject(Object $object)
    {
        // ...
    }

    public function applyDamage(integer $damage)
    {
        // ...
    }
}
```

Constructeur et destructeur

Le constructeur d'une classe est une méthode publique (dans la plupart des cas). Elle est appelée automatiquement au moment de l'instanciation. Elle sert généralement à initialiser les attributs et fournir à l'objet créé tout ce dont il a besoin pour fonctionner. Cette opération d'initialisation est connue sous le nom d'**hydratation**.

Le destructeur et une méthode publique identifiée par le nom **__destruct()**. Il est appelé automatiquement par le compilateur lorsqu'il n'y a plus aucune référence à l'objet en cours. Autrement dit, le destructeur est appelé quand il n'y a plus aucun appel d'un membre quelconque de l'objet.

```php
class Personage
{
    protected $id;
    protected $name;

    public function __construct(integer $id, string $name)
    {
      $this->id = $id;
      $this->name = $name;
    }

    public function __destruct()
    {
      // ...
    }
}

$personage = new Personage(123, 'david');
```

Constantes de classe

Une constante de classe est un élément statique par défaut. Son rôle est le même que celui d'une constante classique déclarée à l'aide de la fonction define(). Sa valeur est inchangée et elle appartient à la classe dans laquelle est elle évoquée et non à l'objet qui constitue l'instance de classe.

Pour définir une constante on utilise le mot clé **const** suivi du nom de la constante à laquelle on affecte la valeur souhaitée. Par convention l'identifiant de la constante est déclaré en majuscule.

```php
class Personage
{
    const MAX_HEALTH = 100;
    protected $health;
    public function __construct($health) {
      $this->health = $health;
    }
    public function getCurrentHealth() {
      return sprintf(
        'Le personnage à %s/%s de vie.',
        $this->health,
        static::MAX_HEALTH
      );
    }
}

$personage = new Personage(78);
echo $personnage->getCurrentHealth();
// Le personnage à 78/100 de vie
echo 'La valeur maximale de vie est de ', Personnage::MAX_HEALTH;
// La valeur maximale de vie est de 100
```

Attributs et méthode statiques

Les attributs et méthodes statiques appartiennent à la classe et non à l'objet. Par conséquent on ne peut pas y accéder par l'opérateur **->** mais plutôt par l'opérateur de résolution de portée **::** précédé par le nom de la classe dans laquelle ils sont définis.

Il est possible d'utiliser à l'intérieur de la classe:

- static
- self
- parent
- le nom de la classe

```php
class Personage
{
    private static $maxHealth = 100;

    public static function getMaxHealth() {
      return sprintf(
        'La valeur maximale de vie est de %s',
        static::$maxHealth
      );
    }
}

echo Personnage::getMaxHealth();
// La valeur maximale de vie est de 100
```

Les méthodes magiques

Les méthodes magiques sont des méthodes prédéfinies et toutes préfixées par double sous-tirets (___) dans une classe PHP. Elle sont appelées automatiquement suite à un événement spécial qui peut survenir lors de l'exécution.

Il est recommandé de ne jamais préfixer les méthodes personnalisée par double sous-tirets. Cette notation est réservée aux méthodes magiques.

Le plus utilisés:

- __get / __set / __isset / __unset / __toString

```php
class Personage
{
    private $name;

    public function __set($attribute, $value) {
      $this->$attribute = $value;
    }
    public function __get($attribute) {
      return $this->$attribute;
    }
    public function __isset($attribute) {
      return isset($this->$attribute);
    }
    public function __unset($attribute) {
      unset($this->$attribute);
    }
    public function __toString() {
      return 'Le prénom : ' . $this->$attribute;
    }
}
$personage = new Personage();
$personage->name = 'David';
if (isset($personage->name)) {
    // Le prénom : David
    echo $personage->name;
    unset($personage->name);
}
```