**Constante et attributs statics**

Définir un attribut statique dans une classe PHP présente plusieurs avantages :  
  
 **1. Partage de données entre instances de la classe**  
  
Un attribut statique est partagé entre toutes les instances de la classe. Cela permet de stocker des informations communes à toutes les instances sans les dupliquer.  
  
/\*   
php  
class Compteur {  
 public static $compteur = 0;  
  
 public function \_\_construct() {  
 self::$compteur++;  
 }  
}  
  
**$instance1 = new Compteur(); // $compteur = 1  
$instance2 = new Compteur(); //$compteur = 2**  
  
echo Compteur::$compteur; **// Affiche 2**  
\*/  
  
**2. Mémorisation de l'état global**  
  
Les attributs statiques peuvent être utilisés pour mémoriser l'état global d'une application ou d'un module.  
  
/\*

php  
class Configuration {  
 public static $settings = [];  
  
 public static function set($key, $value) {  
 self::$settings[$key] = $value;  
 }  
  
 public static function get($key) {  
 return self::$settings[$key] ?? **null**; // ?? null **signifie : si ce qu’ il y a dans la clé $key est** **nul**, **cela affiche** **null**  
 }  
}  
  
Configuration::set('database\_host', 'localhost');  
echo Configuration::get('database\_host'); // **Affiche 'localhost'**  
\*/

**3. Accès rapide et centralisé aux données**  
  
Les attributs statiques permettent d'accéder rapidement et de manière centralisée aux données sans avoir besoin de créer une instance de la classe.  
  
/\*

php  
class Utilisateur {  
 public static $nombreUtilisateurs = 0;  
  
 public function \_\_construct() {  
 self::$nombreUtilisateurs++;  
 }  
}  
  
new Utilisateur();   
new Utilisateur();  
  
echo Utilisateur::$nombreUtilisateurs; **// Affiche 2 (car 2 instances ont été créées et à chaque instance créée $nombreUtilisateurs est incrémenté de 1**  
\*/  
  
 **4. Singleton Pattern**  
  
Les attributs statiques sont souvent utilisés dans le pattern Singleton pour s'assurer qu'une classe n'a qu'une seule instance.  
  
/\*

php  
class Singleton {  
 private static $instance = null;  
  
 private function \_\_construct() {  
 // Constructeur privé pour empêcher l'instanciation directe, on ne peut instancier que à l’interieur de la classe  
 }  
  
 public static function getInstance() {  
 if (self::$instance === null) {  
 self::$instance = new self(); ***//si l instance est null on créé une instance et on la renvoie sinon on renvoie l’instance déjà créée.***  
 }  
 return self::$instance;  
 }  
}  
  
$instance1 = Singleton::getInstance(); *//on créé l’instance et on la renvoie*  
$instance2 = Singleton::getInstance(); //*on renvoie l instance*  
  
var\_dump($instance1 === $instance2); **// Affiche bool(true) car dans les deux cas c’est la même instance (principe du singleton)**  
\*/  
  
### 5. Caching  
  
Les **attributs statiques** peuvent être utilisés pour stocker des données en cache afin d'améliorer les performances.  
  
```php  
class Cache {  
 private static $data = [];  
  
 public static function set($key, $value) {  
 self::$data[$key] = $value;  
 }  
  
 public static function get($key) {  
 return self::$data[$key] ?? null;  
 }  
}  
  
Cache::set('username', 'john\_doe');  
echo Cache::get('username'); // Affiche 'john\_doe'  
```  
  
 **Conclusion**  
  
Les attributs statiques sont utiles pour **partager des données et des états communs à toutes les instances d'une classe**, accéder rapidement aux informations sans avoir besoin de créer une instance, et pour implémenter des patterns comme le Singleton. Ils offrent une manière efficace de **gérer des informations partagées** à travers les instances et les appels de méthode statiques.