Constantes et propriétés statiques

Les propriétés et les méthodes statiques

Les propriétés et les méthodes statiques sont des méthodes disponibles sur la classe elle-même, et sur les instances de la classe.

Par exemple, pour une propriété statique :

```
<?php
class Math {
  public static float $pi = 3.1415926535898;
}

echo Math::$pi, '<br>'; // 3.1415926535898

$math = new Math();
echo $math::$pi; // 3.1415926535898
```

Notez que nous utilisons l'opérateur de résolution de portée (`::`) et non l'opérateur objet (`->`). pour accéder aux propriétés et aux méthodes statiques.

Voici un exemple avec une méthode :

```
<?php
class Math {

  private static float $pi = 3.1415926535898;

  static function getPi() {
    return self::$pi;
  }

}

echo Math::getPi(), '<br>'; // 3.1415926535898

$math = new Math();
echo $math::getPi(); // 3.1415926535898
```

Notez que nous devons utiliser `self` et non `$this` pour accéder à la classe. En effet, `$this` désigne l'instance de la classe en cours d'utilisation. Or ici, nous n'avons pas d'objet car nous accédons à la classe elle-même, la propriété statique étant sur la classe.

Les constantes de classe

Les constantes de classe sont des constantes déclarées sur une classe. Ce sont des propriétés non modifiables. La visibilité par défaut des constantes est publique.

Par exemple :

```
<?php
class Math {
  const PI = 3.1415926535898;
}

echo Math::PI, '<br>'; // 3.1415926535898

$math = new Math();
echo $math::PI; // 3.1415926535898
```

Utilisation des mots clés `self`, `parent` et `static`

`self` et `static` permettent d'accéder à des constantes, propriétés statiques ou méthodes statiques définies sur une classe.

Si le mot clé est utilisé dans une fonction, **la valeur de `self` sera la classe où la fonction est définie**.

A l'inverse, **`static` permet de définir la classe désignée de manière dynamique**.

Par exemple :

```
<?php
class MaClasse {

  private const A = 42;

  static function func() {
    return self::A;
```

```
  }

  static function func2() {
    return static::A;
  }

}

class ClasseEnfant extends MaClasse {
  const A = 43;
}

echo ClasseEnfant::func(), '<br>'; // 42
echo ClasseEnfant::func2(), '<br>'; // 43
```

Nous avons déjà vu `parent` qui permet de désigner la classe parente.