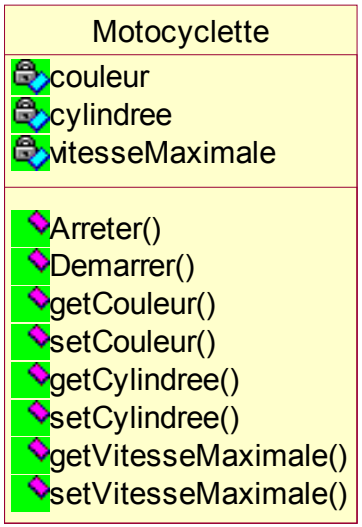
PHP 7° PARTIE :
PROGRAMMATION OBJET

1. Déclaration d'une classe
2. Déclaration d'un objet
3. Encapsulation
4. Constructeur
5. Destructeur
6. Héritage
7. Surcharge et surdéfinition
8. L'opérateur : :
9. Limites de PHP en POO
10. Travaux pratiques

1 . Déclaration d'une classe

✓ En PHP, la définition, la spécification et la réalisation d'une classe sont déclarées dans le même bloc :



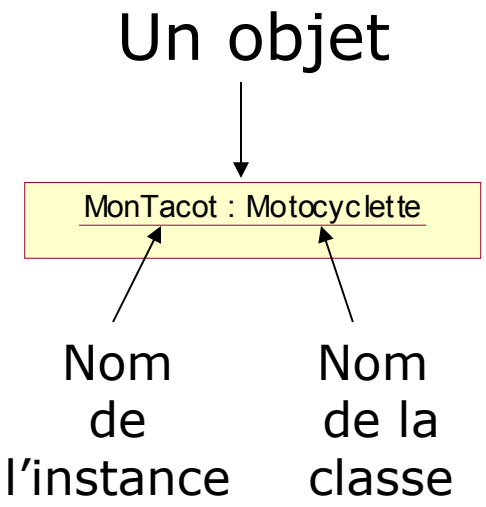

```php
class Motocyclette
{
        //attributs
        var $couleur ;
        var $cylindree ;
        var $vitesseMaximale ;

        //méthodes
        function getCouleur()
        {
return $this->couleur;
        }

        function setCouleur($couleur)
        {
$this->couleur = $couleur;
        }
         //etc …
}
```

2 . Déclaration d'un objet

✓ En PHP, la déclaration d'un objet revient à instancier une classe avec **new** :




```php
//Déclaration d'une instance :
$MonTacot = new Motocylette;

//Utilisation :
$MonTacot->setCouleur("rouge");
```

3 . Encapsulation

✓ Pour les *parser* Zend Engine 1.x (PHP 4.x), tous les membres sont **publics :**

```
//possible mais déconseillé :
$MonTacot->couleur = "vert";
```

✓ Les versions Zend Engine 2.x introduisent la notion des **attributs privés** :

```
class CNomClasse
{
        var $a ;  //attribut public
        private $b ; //attribut privé
}
```

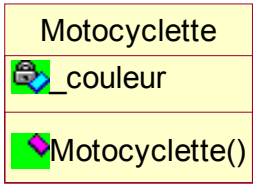
4 . Constructeur

✓ Il faut ici distinguer les différentes versions :

❖ PHP 3 : une fonction portant le même nom que la classe

❖ **PHP 4.x et Zend Engine 1.x : une fonction membre portant le même nom que sa classe**

❖ Zend Engine 2.x : une fonction membre spécifique `__construct()`




```php
class Motocyclette
{
        //attribut
        var $_couleur ;

        //Constructeur
        function Motocyclette($couleur= "")
        {
$this->_couleur = $couleur;
        }
        //etc …
}

//Déclaration d'une instance :
$MonTacot1 = new Motocylette();
//ou :
$MonTacot2 = new Motocylette("rouge");
```

5 . Destructeur

✓ **Il n'y a pas de destructeurs en PHP 4.x et Zend Engine 1.x.**

Destruction : en PHP 4.x, on utilisera la fonction **`unset($MonTacot)`**.

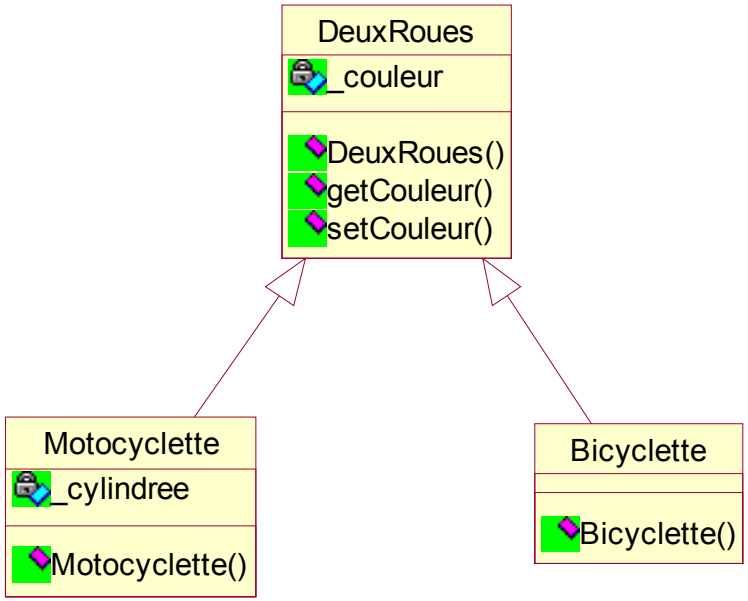
➢Solution PHP 4.x : étant donné que le destructeur est appelé lorsqu'un objet est détruit, on peut alors **simuler** un destructeur avec **`register_shutdown_function(string fct)`**, qui permet d'enregistrer la fonction `fct` pour son exécution à la fin du script.

✓ **Zend Engine 2.x introduit la notion de destructeur** avec la fonction membre spécifique `__destruct()` a

Destruction : on utilisera **`delete($objet)`**.

6 . Héritage

✓ En PHP, une classe peut être déclarée comme étant une **sous-classe** d'une autre classe en spécifiant le mot clé **`extends`** :




```php
//Déclaration d'une instance :
$MonVelo = new Bicylette("bleue");

echo $MonVelo->getCouleur();
```

```php
class DeuxRoues
{
        //attribut
        var $_couleur ;

        //Constructeur
        function DeuxRoues($couleur= "")
        {
$this->_couleur = $couleur;
        }
        //etc …
}

class Bicyclette extends DeuxRoues
{
        //Constructeur
        function Bicyclette($couleur= "")
        {
//Appel du constructeur parent
parent::DeuxRoues($couleur);
        }
}
```

7 . Surcharge et surdéfinition

❖ Surcharge (*overloading*)

PHP ne permet pas la surcharge de fonction et donc on ne peut attribuer le même nom à plusieurs fonctions. Par contre la redéfinition est possible.

❖ Redéfinition (*overriding*)

PHP permet la redéfinition, c'est-à-dire la possibilité de redéclarer les mêmes attributs et opérations d'une super classe au sein d'une **sous classe**.

```php
class DeuxRoues
{
        //Attribut
        var $_couleur ;

        //Méthode
        function getCouleur()
        {
return $this->_couleur;
        }
}
```

```php
class Bicyclette extends DeuxRoues
{
        //redéfinition de l'attribut
        var $_couleur ;

        //redéfinition de la méthode
        function getCouleur()
        {
echo "Nouvelle méthode !";
return $this->_couleur;
        }
}
```

retourne cet attribut

```php
//Déclaration d'une instance :
$MonVelo = new Bicylette;

$MonVelo->getCouleur();
```

Appel de cette méthode

8 . L'opérateur ::

Il est possible de faire référence aux méthodes d'une classe de base.

Pour cela, on utilise l'opérateur `::` en précisant soit le **nom de la classe** de base soit le mot clé **`parent`** :

```
class MaClasse
{
          //Attribut
          var $attribut ;

          //Méthode
          function Methode()
          {
echo "Je suis une méthode !";
          }
}
```

```
//Exemple :
MaClasse::Methode();

//Attention :
//Étant donné qu'il n'y a pas de
//déclaration d'un objet
//les attributs
```

9 . Les limites de PHP en POO

Mini FAQ :

1 . Peut-on utiliser `include` et `require` dans la définition d'une classe ?

Non.

2 . PHP fait-il la distinction entre fonctions et méthodes ?

Oui, PHP distingue bien les méthodes d'un objet des fonctions indépendantes.

3 . Est-on obligé de déclarer un constructeur ?

Non. Si une classe n'a pas de constructeur, le constructeur de la classe de base est appelé s'il existe.

4 . Que ne peut-on pas encore faire avec les classes en PHP ?

Tout dépend de la version utilisée : d'importants changements sont prévus dans les futures versions de PHP (visiter le site www.zend.com ou www.php.net).

Actuellement, les limites pour PHP 4.x sur un moteur Zend Engine 1.x sont :

- Pas de méthodes, ni d'attributs privées : tout est public;
- Pas de destructeur;
- Pas d'appel automatique au constructeur de la classe père;
- Pas d'héritage multiple.

10 . Travaux pratiques

1. Tester les exemples du cours.

2. Reprendre l'application de gestion de revues du TP n°4 et l'écrire en POO. Pour rappel, l'application doit permettre les fonctions principales suivantes : ajouter des revues, modifier des revues existantes, supprimer des revues existantes et afficher les revues disponibles. On structurera son application en utilisant des fichiers séparés. Pour les fichiers contenant des classes, utiliser les noms xxx.class.php