PHP 5 : PROGRAMMATION OBJET

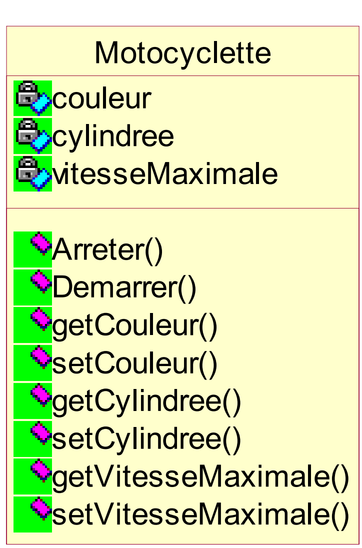
1. Déclaration d’une classe
2. Déclaration d’un objet
3. Encapsulation et visibilité
4. Membres statiques
5. Constructeur
6. Destructeur
7. Héritage
8. Surcharge et surdéfinition
9. L’opérateur : :
10. Mots réservés
11. Interface
12. Classe abstraite
13. Appel statique (LSB)
14. Les exceptions
15. Méthodes magiques
16. Remarques

Visiter le site www.zend.com ou www.php.net

© BTS IRIS 2011 [tv]

1 . Déclaration d'une classe

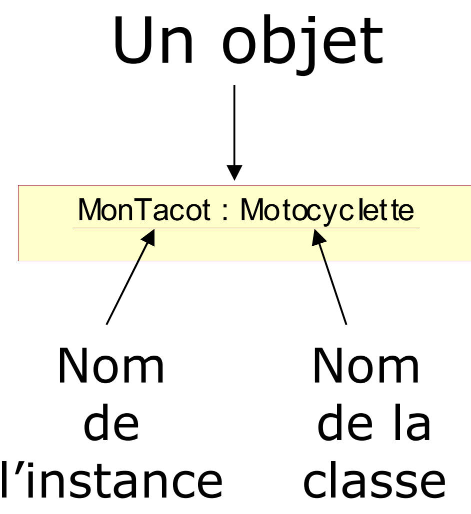
✓ En PHP5, la définition, la spécification et la réalisation d'une classe sont déclarées dans le même bloc :




```
class Motocyclette
{
   //attributs
   private $couleur;
   private $cylindree;
   private $vitesseMaximale;

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

   public function setCouleur($couleur)
   {
      $this->couleur = $couleur;
   }

   //etc …
}
```

2 . Déclaration d’un objet

✓ En PHP5, la déclaration d’un objet revient à instancier une classe avec **new** :




```
//Déclaration d’une instance :
$MonTacot = new Motocyclette();

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

//Vérification :
if($MonTacot instanceof Motocyclette)
   echo "L'objet \$MonTacot
          est du type Motocyclette";
```

3 . Encapsulation

✓ Les membres d'une classe PHP5 peuvent être **publics, protégés ou privés :**

```
class Demo
{
  //Attributs
  private $aPrivate = "Attribut privé";
  protected $aProtected = "Attribut protégé";
  public $aPublic = "Attribut public";

  //Méthodes
  private function getPrivate() {
    echo "get : ",$this->aPrivate,"<br />";
  }
  protected function getProtected() {
    echo "get : ",$this->aProtected,"<br />";
  }
  public function getPublic() {
    echo "get : ",$this->aPublic,"<br />";
    $this->getProtected();
    $this->getPrivate();
  }
}
```

```
$objet = new Demo();

$objet->getPublic();
$objet->getProtected(); //Erreur
$objet->getPrivate(); //Erreur
```

4 . Membres static

✓ Les membres d'une classe PHP5 peuvent être **statiques :**

```
class Bourse
{
  //Propriété statique
  public static $lieu = "Paris";

  //Méthodes statiques
  public static function getheure()
  {
    $heure = @date("h : m : s");
    return $heure;
  }
  public static function afficheInfo()
  {
    $texte = Bourse::$lieu.", il est ".self::getheure();
    return $texte;
  }

  public function setLieu($lieu)
  {
    Bourse::$lieu = $lieu;
  }
}
```

```
echo Bourse::$lieu,"<br />";
echo Bourse::getheure(),"<br />";
echo Bourse::afficheInfo(),"<hr />";
```

5 . Constructeur

✓ Il faut ici distinguer les différentes versions :

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

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

❖ PHP5 (Zend Engine 2.x) : une fonction membre spécifique `__construct()`

```
class Motocyclette {
  //attribut
  private $couleur;

  //Constructeur
  function __construct($couleur="")   {
      echo "Constructeur<br />";
      $this->couleur = $couleur;
  }
  //etc …
}

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

6 . Destructeur

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

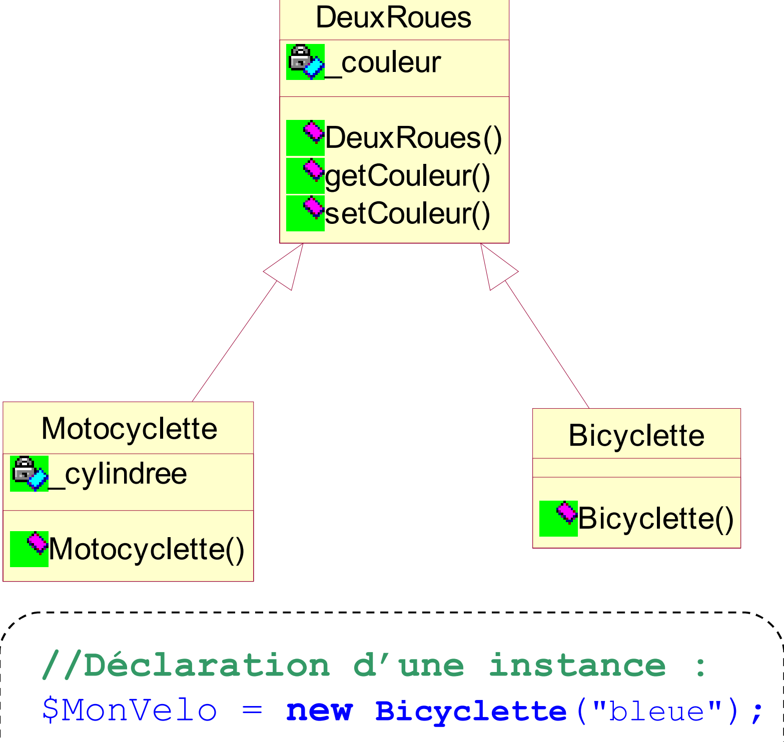
✓ **PHP5 (Zend Engine 2.x) introduit la notion de destructeur** avec la fonction membre spécifique `__destruct()`

```
function __destruct() {
    echo "Destructeur<br />";
}
```

Destruction : on utilisera la fonction **`unset($MonTacot)`**.

7 . Héritage

✓ En PHP5, une classe peut être déclarée comme étant une **sous-classe** d'une autre classe en spécifiant le mot clé **extends** :




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

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

```
class DeuxRoues {
   //attribut
   private $_couleur;

   //Constructeur
   function __construct($couleur= "") {
      $this->_couleur = $couleur;
   }
   public function getCouleur() {
      return $this->_couleur;
   }
   //etc ...
}

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

8 . Surcharge et surdéfinition

❖ Surcharge (*overloading*)

PHP ne permet pas la surcharge de fonction (ou de méthodes) 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**.

```
class DeuxRoues
{
          //Attribut
          protected $_couleur ;

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

```
class Bicyclette extends DeuxRoues
{


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

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

$MonVelo->getCouleur();
```

Appel de cette méthode

9 . Résolution de portée : 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 **`self`** :

```
class MaClasse {
    //Attribut
    public $attribut = "valeur";
    //Méthodes
    public function Methode1() {
        echo "Je suis une méthode publique !<br />";
        $this->Methode2();
        //Methode2(); Erreur : essaye d'appeler une fonction et non la méthode
        MaClasse::Methode2();
        self::Methode2();
    }
    private function Methode2() {
        echo "Je suis une méthode privée !<br />";
        echo "\$attribut = $this->attribut<br />";
        //echo "MaClasse::attribut = ".MaClasse::$attribut."<br />";
            //Erreur : devrait être statique
    }
}
```

```
//Exemple :
$objet = new MaClasse;

$objet->Methode1();
```

10 . Mots réservés

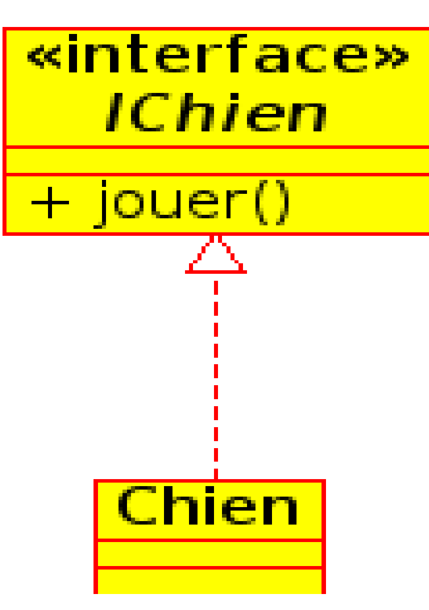
- ✓class : Déclaration de classe ;
- ✓const : Déclaration de constante de classe ;
- ✓function : Déclaration d'une méthode ;
- ✓public/protected/private : Accès (par défaut "public" si aucun accès n'est explicitement défini) ;
- ✓new : Création d'objet ;
- ✓self : Résolution de portée (la classe elle-même) ;
- ✓parent : Résolution de portée (la classe "parent") ;
- ✓static : Membres statiques ou Résolution de portée (appel statique) disponible depuis PHP 5.3 et 6.0 ;
- ✓extends : Héritage de classe ;
- ✓implements : Implémentation d'une interface (dont il faut redéclarer toutes les méthodes).
- ✓abstract : Classe abstraite

11 . Interface (1/2)

- ✓Une interface permet de créer un modèle que les classes qui l'implémentent doivent respecter.
- ✓Une interface regroupe toutes les méthodes qu'une classe doit implémenter.
- ✓Les méthodes déclarées dans une interface ne peuvent être de type private.
- ✓Les interfaces permettent également de définir des constantes, il suffit de les y déclarer à l'aide du mot clé "const"
- ✓Une des caractéristiques principales des interfaces est d'obliger les classes qui les implémentent à créer toutes leurs méthodes. Si ce n'est pas le cas, une erreur sera générée.
- ✓Une classe peut implémenter plusieurs interfaces.

11 . Interface (2/2)

✓ L'implémentation se fait de la même manière que l'héritage, sauf que l'on utilise le mot clé "implements" :




```php
<?php
interface Ichien {
   public function jouer();
}
class Chien implements Ichien {
   private $nom;

   public function __construct($nom) {
      $this->nom = $nom;
   }
   // on DOIT définir la méthode jouer()
   public function jouer() {
      echo 'Ouah!';
   }
}

$monChien = new Chien("medor");
$monChien->jouer();
?>
```

12 . Classe abstraite

✓Une classe abstraite est une classe qui ne peut pas être instanciée.

✓Elle est utilisée pour être héritée.

✓Une classe abstraite a au moins une opération abstraite. Mais elle peut aussi avoir des opérations concrètes.

✓Remarque : une classe ne peut hériter que d'une seule classe, ce qui n'est pas le cas avec les interfaces.

valeur
nom : string # valeur : float
getinfo() # affiche() # *__construct()*

action
- bourse : string

```
abstract class valeur {
  protected $nom;
  protected $prix;
  abstract protected function __construct() ;
  abstract protected function getinfo();
  protected function affiche() {
     $info = "Le prix de $this->nom est de $this->prix";
     return $info;
  }
}

class action extends valeur {
  private $bourse;
  ...
}

$action1 = new action("Alcotel", 9.76);
...
```

13 . Late Static Bindings

✓C'est une innovation de PHP 5.3 et 6.0 que l'on appelle les "Late Static Bindings" (LSB).

✓Les LSB permettent de résoudre les appels statiques plus tard dans la chronologie des évènements.

✓Les Late Static Bindings permettent d'utiliser un nouveau mot clef "static::" pour résoudre correctement la portée statique :

```
class Chien {
   protected function aboyer() {
      return 'Je suis un chien';
   }
   public function identifier() {
      return static::aboyer(); //appel statique
   }
}
class Labrador extends Chien {
   protected function aboyer() {
      return 'Je suis un labrador';
   }
}

$medor = new Chien();
$sultan = new Labrador();
echo $medor->identifier().'<br/>'; // Je suis un chien
echo $sultan->identifier().'<br/>'; // Je suis un labrador
```

14 . Les exceptions

✓Une exception est un traitement qui sera appelé lorsqu'un cas particulier est détecté pendant le déroulement du programme.

✓Remarque : une erreur est le résultat d'un fonctionnement anormal alors qu'une exception est un fonctionnement normal, mais exceptionnel.

✓Les exceptions fonctionnent de la même façon dans tous les langages. Dans un bloc défini (try = essayer), l'environnement d'exécution envoie un signal (throw = lancer) au gestionnaire d'exception qui diffusera l'information aux différents blocs de traitement (catch = attraper). On quitte le bloc try (sans exécuter le code restant du bloc try) et le bloc catch correspondant exécutera un code adapté.

```
function verifier($var) {
    if (empty($var)) {
        throw new Exception ('La variable est vide !');
    }
    else {
        return $var;
    }
}
```

```
try {
    $var1 = "abc";
    $var2 = "";
    verifier($var1);
    echo $var1."<br />";
    verifier($var2);
    echo $var2."<br />"; // code non exécuté
}
catch(Exception $myException) {
    echo $myException->getMessage();
}
```

15 . Méthodes magiques

- __construct() : Constructeur de la classe ;
- __destruct() : Destructeur de la classe ;
- __set() : Déclenchée lors de l'accès en écriture à une propriété de l'objet ;
- __get() : Déclenchée lors de l'accès en lecture à une propriété de l'objet ;
- __call() : Déclenchée lors de l'appel d'une méthode inexistante de la classe (appel non statique) ;
- __callstatic() : Déclenchée lors de l'appel d'une méthode inexistante de la classe (appel statique)
- __isset() : Déclenchée si on applique isset() à une propriété de l'objet ;
- __unset() : Déclenchée si on applique unset() à une propriété de l'objet ;
- __sleep() : Exécutée si la fonction serialize() est appliquée à l'objet ;
- __wakeup() : Exécutée si la fonction unserialize() est appliquée à l'objet ;
- __toString() : Appelée lorsque l'on essaie d'afficher directement l'objet echo $object; ;
- __set_state() : Méthode statique lancée lorsque l'on applique la fonction var_export() à l'objet ;
- __clone() : Appelé lorsque l'on essaie de cloner l'objet ;
- __autoload() : Cette fonction n'est pas une méthode, elle est déclarée dans le scope global et permet d'automatiser les "include/require" de classes PHP.

16 . Remarques

On structurera son application en utilisant des fichiers séparés. Pour les fichiers contenant des classes, utiliser des noms sous la forme **xxx.class.php**

```
<?php
include("poo_autoload.inc.php");

//Déclaration d'une instance :
//chargement automatique de Motocyclette.class.php"
$MonTacot = new Motocyclette();

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

```
<?php
function __autoload($class) {
    $file = $class.'.class.php';
    if(file_exists($file)) {
        require_once $file;
    }
    //echo $file.'<br/>';
}
// etc ...
?>
```

```
<?php
class Motocyclette
{
    // etc ...
}
?>
```