PHP

PHP OBJET - HERITAGE

HERITAGE

- **Traduction de la relation de spécialisation**

  une personne → un étudiant
  une commande → une commande fournisseur
  un animal → un chien

- **Principe**

  nouvelle classe = ancienne classe + nouvelles informations

- **Vocabulaire**

Ancienne classe		Nouvelle classe
classe de base	–	*classe dérivée*
classe parent/mère	–	*classe fille*
super classe	–	*sous classe*

PRELIMINAIRE

- **Classe Personne**

```
<?php

class Personne
{
		private $_nom;
		private $_prenom;
		private $_ddn;

		// constructeur
		public function __construct ($p_nom, $p_prenom, $p_ddn) { … }

		// get - set
		public function getNom  () {… }

		// autres methodes
		public function sePresenter () { … }
		public function calculerAge ();
?>
```

EXTENDS

- **Déclaration de la sous-classe**
  préalable : inclure le fichier de la classe parent

- **Syntaxe**

class *ClasseFille* **extends** *ClasseParent*;

```
<?php
 require_once («/classes/Personne.class.php);
class Etudiant extends Personne
{
        // les informations à ajouter à la personne pour obtenir un étudiant
        private $_classe;
        private $_notes;

        public function __construct (…) { … }
        public function ajouterNote (…) {…}
        public function calculerMoyenne () { … }
}
?>
```

PARENT::

- **Accès aux informations de la classe de base**
  informations publiques ou protégées uniquement

- **Syntaxe**

$this->attribut
$this->methode ($arg); *// méthode non redéfinie*
parent::methode($arg); *// méthode redéfinie*

```
<?php
class Etudiant extends Personne {
        ...
        public function sePresenter () {
                parent::sePresenter ();
                echo “et je suis en ”. this->_classe;    }
?>
```

CONSTRUCTEUR

- **Appel au constructeur parent**
  à faire (manuel)

- **Syntaxe**

parent::__constructor ($arg);

```
<?php
require_once («/classes/Personne.class.php »);
class Etudiant extends Personne
{
        private $_classe;        // class où est inscrit l'étudiant;

        public function __construct ($p_nom, $p_prenom, $p_ddn, $p_classe) {
                parent::__construct ($p_nom, $p_prenom, $p_ddn);
                $this->_classe = $p_classe;
        }
}
$moi = new Etudiant ('Maldonado', 'Michel', '02/06/1966', 'B2');
?>
```

PROTECTED

- **Accès depuis la sous-classe aux informations de la classe de base**
  - get / set
  - statut protected

```
<?php
class Personne {
        protected $_nom;
        protected function calculerAge () {… }
}
class Etudiant extends Personne {
        …
        public function sePresenter () {
                echo “je m’appelle ” . $this->_nom . “, je suis en ”. this->_classe;
                echo “ et j’ai “ . $this->calculerAge() . “ ans”
        }
?>

$moi = new Etudiant («maldonado», «michel», «02/06/1966», «B2»);
echo $moi->_nom;                    // erreur
echo $moi->calculerAge ();          // erreur
```

POLYMORPHISME

- **Redéfinition d'une méthode de la classe parent**
  même nom de méthode, paramètres identiques ou non

```
<?php
class Personne {
        …
        public function sePresenter () { echo « je m'appelle $this->_prenom $this->_nom »; }
}
class Etudiant extends Personne
{
        …
        public function function sePresenter () {
                parent::sePresenter ();
                echo « et je suis en $this->_classe »;
        }
}
$moi = new Personne('Maldonado', 'Michel', '02/06/1966');
$toi = new Etudiant ('Dupont', 'Jean', '17/10/1995', 'B2');

$moi->sePresenter ();    // je m'appelle Michel Maldoando
$toi->sePresenter ();    // je m'appelle Jean Dupont et je suis en B2
?>
```

FINAL

- **Méthode non redéfinissable**

  final public function méthode () {…

- **Classe non dérivable**

  final class NomClasse {…

HERITAGE SIMPLE/MULTIPLE

- **Héritage multiple**
  Plusieurs classes parents
  C++, Python, …

- **Exemple**
  Classe Avion – Classe TransportCollectif
  ➔ *Classe AirBusA380 = Avion + TransportCollectif*

- **Héritage simple**
  Une seule classe parent
  Java, .NET (C#), PHP

- **Autre mécanisme**
  cf interface

ABSTRACT

- **Classes abstraites**
  non instanciables

  ~~$obj = new ClasseAbstraite();~~ *// erreur !*

- **Doit contenir au moins une méthode abstraite**
  préfixée par abstract
  pas de corps

  public abstract function méthode ();

- **Les sous-classes doivent implémenter toutes les méthodes abstraites**
  à défaut, abstraites elles-aussi

ABSTRACT

- **Exemple :**

```php
<?php
abstract class Figure {
        private $_couleur;
        public function __construct ($p_couleur) {… }

        public function getCouleur () {  return $this->couleur; }
        public abstract function dessiner ();
        public abstract function calculerSurface ();
}
?>
```

ABSTRACT

```
<?php
class Cercle extends Figure {
        private $_centre;
        private $_rayon;
        public function __construct (…) {…}
        public function dessiner () { echo "je dessine un cercle de rayon $this->_rayon…"    ); }
        public function calculerSurface () { return pi() * $this->_rayon * $this->_rayon; }
}
class Carre extends Figure {
        private $_pointSupGauche;
        private $_cote;
        public function __construct (…) {…}
        public function dessiner () { echo "je dessine un carré de coté $this->_cote…"      ); }
        public function calculerSurface () { return $this->_cote * $this->_cote; }
}

$fig1 = new Cercle (new Point (1, 1), 1, "rouge");
$fig2  =  new Carre (new Point (0,0), 1, "bleu");
echo "surface de la premiere figure : " . $fig1->calculerSurface ();
echo "surface de la premiere figure : " . $fig2->calculerSurface ();
?>
```

INTERFACE

- **Contrat à respecter**
  liste de méthodes que les sous-classes devront implémenter.

- **Pas d'attributs**

- **Méthodes**
  toujours publiques
  pas de corps (comme classe abstraite)
  interface INomInterface { … }

- **Un classe peut implémenter plusieurs interfaces**
  class nomClasse implements INomInterface {…

INTERFACE

```
<?php
Interface IFigure {
        function dessiner ();
        function calculerSurface ();
}
class Cercle implements IFigure {
        private $_pointSupGauche;
        private $_cote;
        private $_couleur;
        public function __construct (…) {…}
        public function dessiner () { echo "je dessine un carré de coté $this->_cote…"        ); }
        public function calculerSurface () { return $this->_cote * $this->_cote; }
}
class Carre implements Ifigure { …

$fig1 = new Cercle (new Point (1, 1), 1, "rouge");
$fig2  =  new Carre (new Point (0,0), 1, "bleu");
echo "surface de la premiere figure : " . $fig1->calculerSurface ();
echo "surface de la premiere figure : " . $fig2->calculerSurface ();
?>
```

INTERFACE vs CLASSE ABSTRAITE

Interface	Classe Abstraite
Aucune données mais getter/setter possibles	Attributs possibles
Pas de constructeur	Constructeur possible
Methodes publiques uniquement	Méthodes publiques, privées ou protégées
Prototypes uniquement (pas de code)	Corps de méthodes possibles
Plusieurs **implements**	Un seul **extends**
Toutes les méthodes à implémenter	Zéro ou plusieurs méthodes non implémentées (classe restant abstraite

- **Interface**
  objets indépendants – petits ensembles de fonctionnalités

- **Classe abstraite**
  versonnable (héritage) – objets hiérarchiques – objets plus gros – code par défaut

CONCLUSION