SYNTAXE PHP OBJET : « COMPARAISON AVEC LE LANGAGE C# »

Sommaire

➢ 1 - D'UN POINT DE VUE « UTILISATION DE LA POO »

1.1 - CREATION D'OBJETS (INSTANCES)

Exemple : création d'une instance de Personne dont on connait le prénom et le nom.

C#	`Personne unePersonne = new Personne("Gaston", "LAGAFFE");`
PHP	$unePersonne = new Personne("Gaston", "LAGAFFE");

1.2 - ACCES A UN MEMBRE PUBLIC

Exemple : accès au getteur getNom() de la classe Personne

C#	`string leNom = unePersonne.getNom();`
PHP	$leNom = $unePersonne->getNom();

1.3 - ACCES A UN MEMBRE STATIQUE (PUBLIC)

Exemple : accès au compteur d'instances de la classe Personne

C#	`int nbInstances = Personne.getNbPersonnes();`
PHP	$nbInstances = Personne::getNbPersonnes();

1.4 - GESTION DES EXCEPTIONS

C#	`try`
	`{//Traitement à essayer`
	`}`
	`catch (Exception e)`
	`{//Traitement si exception levée`
	`}`

PHP (identique à C#)	try {
	  //Traitement à essayer
	} catch (Exception $e) {
	  //Traitement si exception levée
	}

1.5 - PARCOURS DE COLLECTIONS (TABLEAUX D'OBJETS EN PHP)

Exemple : affichage des noms de personnes d'une collection « lesPersonnes » (tableau d'objets en PHP)

C#	`foreach (Personne unePersonne in lesPersonnes)`
	`{`
	`  Console.WriteLine(unePersonne.getNom());`
	`}`

PHP	foreach ($lesPersonnes as $unePersonne) //on inverse : d'abord la liste ensuite l'item
	{
	  echo $unePersonne->getNom();
	}

Pour plus d'infos sur toutes les fonctions disponibles concernant les tableaux (array) : http://php.net/manual/fr/ref.array.php

➢ 2 - D'UN POINT DE VUE « CONCEPTION POO »

2.1 - STRUCTURE D'UNE CLASSE

Exemple : classe Personne

C#

```
public class Personne
{
}
```

PHP

```
class Personne //pas de préfixe public
{
}
```

2.2 - DEFINITION ET INITIALISATION DE CHAMPS PRIVES

Exemple : champs nom et prénom

C#

```
private string _nom = "inconnu";
private string _prenom = "inconnu";
```

PHP

```
private $_prenom = "inconnu";
private $_nom = "inconnu";
```

2.3 - DEFINITION ET INITIALISATION DE CHAMPS STATIQUES PRIVES

Exemple d'école : compteur d'instances

C#

```
private static int _nbPersonnes = 0;
```

PHP

```
private static $_nbPersonnes = 0;
```

2.4 - DEFINITION ET INITIALISATION DE METHODES PUBLIQUES

Exemple : getteur/setteur pour le champ _nom

C#

```
public string getNom()
{
    return this._nom;
}

public void setNom(string nom)
{
   this._nom = nom;
}
```

PHP

```
public function getNom() {
      return $this->_nom;
}

public function setNom($nom){  //Remarque : pas de void en PHP, que des fonctions (même sans retour)
      $this->_nom = $nom;
}
```

2.5 - METHODES MAGIQUES __get() et __set() : EQUIVALENCE DES PROPRIETES C#

C#

```
public string nom
{    get { return _nom; }
     private set { _nom = value; }
}

public string prenom
{    get { return _prenom; }
     set { _prenom = value; }
}
```

PHP

```
public function __get($propriete) {
      switch ($propriete) {
         case "prenom" : return $this->_prenom; break;
         case "nom" : return $this->_nom; break;
        }
   }
public function __set($propriete, $value) {
      switch ($propriete) {
         case "prenom" : $this->_prenom = $value; break;
         case "nom" : $this->_nom = $value; break;
        }
   }
```

Rmq : cette pratique évite donc de définir toutes les méthodes publiques de type getteurs et setteurs

Inconvénients des méthodes magiques __get() et __set() : *Bien que ces deux méthodes magiques soient très pratiques à utiliser, elles posent tout de même deux désagréments non négligeables lorsque l'on développe en environnement professionnel :*
- l'utilisation de __get() et __set() ***empêche tout d'abord la génération automatique de documentation de code*** *au moyen des APIs (PHPDocumentor par exemple) utilisant les objets d'introspection (Reflection)*
- D'autre part, cela empêche également les IDE tels qu'Eclipse ou NetBeans d'introspecter le code de la classe et ainsi proposer ***l'auto-complétion du code.***
Ceci dit, ces méthodes magiques étant très pratiques, il est très courant de les utiliser…

2.6 - DEFINITION DU CONSTRUCTEUR (METHODE MAGIQUE PHP __construct()) [getteurs et setteurs magiques implémentés]

C#

```
public Personne(string prenom, string nom)
{
    this.prenom = prenom;
    this.nom = nom;
    Personne._nbPersonnes++; //incrémentation du nombre d'instances
}
```

PHP

```
public function Personne($prenom, $nom) {
      $this->prenom = $prenom;
      $this->nom = $nom;
      Personne::$_nbPersonnes++; // OU self::$_nbPersonnes++;  //accès au membre statique
}
```

OU

```
public function __construct($prenom, $nom) { //Utilisation de la méthode MAGIQUE __construct
      $this->prenom = $prenom;
      $this->nom = $nom;
      Personne::$_nbPersonnes++; // OU self::$_nbPersonnes++;  //accès au membre statique
}
```

Intérêt d'utiliser la méthode magique **__construct()** en PHP : le nom du constructeur ne dépend plus du nom de la classe.
Cela s'avère plus pratique dans le cas d'un héritage, pour appeler le constructeur parent :
parent::__construct() au lieu de parent::nomDeLaClasseParenteAConnaitre()

2.7 - SURCHARGE DE CONSTRUCTEUR OU AUTRE METHODE

Dans une classe PHP, 2 méthodes ne peuvent porter le même nom.
Il existe par contre une astuce qui consiste à placer des paramètres optionnels. Le même principe existe également en C#

C#

```
public Personne(string nom, string prenom = null)
{
    this.nom = nom;
    if (prenom!=null)
       this.prenom = prenom;
}
```

PHP

```
public function __construct($nom, $prenom = null) {
    $this->nom = $nom;
    if ($prenom != null)
       $this->prenom = $prenom;
  }
```

Exemples d'appel avec le constructeur défini ci-dessus :

C#

```
Personne unePersonne = new Personne("Dupont");
Personne uneAutrePersonne = new Personne("Lagaffe","Gaston");
```

PHP

```
$unePersonne = new Personne("Dupont");
$uneAutrePersonne = new Personne("Lagaffe","Gaston");
```

Attention par contre à l'ordre des paramètres avec cette technique dès lors qu'ils sont nombreux, pour éviter l'appel suivant :

```
$unePersonne = new Personne("Lagaffe",null,null,null,null,"valeur");
```

S'il y a beaucoup de paramètres, on peut aussi choisir de passer un tableau de paramètres.

Cela signifie que l'entête suivante permet de définir tous les cas (aucun ou une infinité de paramètres) :

```
public function nomMethodeOuConstruct($tabParams = null) { // traitements…}
```

2.8 - METHODE MAGIQUE __tostring()

C#

```
public override string ToString()
{
     return this.prenom + " " + this.nom;
}
```

PHP

```
public function __tostring() {
    return $this->prenom . " " . $this->nom;
  }
```

Intérêt : lorsque l'on veut afficher le contenu d'un objet (ex : echo $unePersonne ;), le texte sera celui du retour de __tostring() ;

2.9 - DEFINITION DE CONSTANTE

C#

```
private const int AGE_MAJORITE = 18;
```

PHP

```
const AGE_MAJORITE = 18; //pas de possibilité d'accès "private" (privilégier static dans ce cas)
```

2.10 - DECLENCHEMENT D'EXCEPTION

C#

```
throw new Exception("Erreur...");
```

PHP (identique à C#)

```
throw new Exception("Erreur…");
```

➢ 3 - MISE EN ŒUVRE DE L'HERITAGE

3.1 - STRUCTURE D'UNE CLASSE DERIVEE

C#

```
public class PersonneSecure : Personne
{
      private string _md5pw;
      //etc...
}
```

PHP

```
class PersonneSecure extends Personne {
   private $_md5pw;
}
```

3.2 - REDEFINITION DU CONSTRUCTEUR

C#

```
public PersonneSecure(string nom, string prenom, string md5pw):base(nom,prenom)
{
      this.md5pw = md5pw;
}
```

PHP

```
public function __construct($nom, $prenom, $md5pw) {
      parent:: __construct($nom, $prenom);
      $this->md5pw=$md5pw;
   }
```

3.3 - REDEFINITION D'AUTRES METHODES

C#

```
public override string ToString()
{
      return base.ToString() + this.md5pw;
}
```

PHP

```
public function __toString() {
      return parent:: __toString() . " " . $this->md5pw;
   }
```

➢ 4 - AUTRES BONNES PRATIQUES

4.1 - REGIONS DE CODE (code déroulable)

C#

```
#region Champs
    //définition des champs
#endregion
```

PHP

```
//  <editor-fold defaultstate="collapsed" desc="région Champs">
      //définition des champs
// </editor-fold>
```

4.2 - DOCUMENTATION DU CODE

C#
(Génération par « /// »)

```
/// <summary>
/// Initialise une nouvelle instance de Personne
///dont on connait des caractéristiques.
/// </summary>
/// <param name="nom">Nom de la personne.</param>
/// <param name="prenom">Prénom de la personne.</param>
public Personne(string nom, string prenom)
{
    this.nom = nom;
    this.prenom = prenom;
}
```

PHP
(Génération par « / »**
puis touche Entrée)

```
/**
    * Initialise une nouvelle instance de Personne
    * dont on connait des caractéristiques.
    * @param string $nom Nom de la personne.</
    * @param string $prenom Prénom de la personne.</
    *
    */
   public function Personne($nom, $prenom) {
      $this->nom = $nom;
      $this->prenom = $prenom;
   }
```

Intérêt : doc accessible lors de l'autocomplétion (équivalent des spécifications externes de l'IntelliSense de Visual Studio)

4.3 - RANGEMENT DANS DES NAMESPACES

Déclaration :

C#

```
namespace MondeVivant.Humains
{
    public class Personne
    { //etc..
    }
```

PHP

```
namespace MondeVivant\Humains {
   class Personne { // etc…
   }
}
```

OU

```
namespace MondeVivant\Humains ;
class Personne { // etc…
 }
```

Utilisation :

C#

```
using MondeVivant.Humains; //accès à toutes les classes du namespace
```

PHP

```
use MondeVivant\Humains\Personne; //chemin jusqu'à la classe (pas d'accès à l'ensemble des classes)

//pour utiliser une classe du namespace « global » (exemple avec la classe Exception) :
use \Exception;
```