Plan

POO avec PHP

Notions O.O.

- Classe (méthodes et attributs)
- Objet
- Instanciation
- Encapsulation et niveaux de visibilité:
  - Public
  - Privé
  - Protégé
- Les différentes méthodes:
  - Constructeurs / destructeurs
  - Les getters et setters
- Surcharge de méthodes
- Auto-référencement : $this
- Héritage
- Polymorphisme

POO avec PHP

Définition d'une classe

```php
<?php
class MaClasse {
   private $madonnee ;

   public function __construct($param) {
        $this->madonnee = $param ;
   }
   function __destruct() {
        echo "Destruction..." ; }
   function affiche() {
        echo "madonnee : "
            .$this->madonnee ;
   }
}
```

Déclaration de classe

Attribut privé

Constructeur public

Référence à l'objet courant

Destructeur public

Méthode publique par défaut

Accès à un attribut

POO avec PHP

Instanciation et utilisation des classes

```php
<?php
require "maclasse.class.php" ;

// Nouvel objet
$o = new MaClasse(12) ;
// Utilisation d'une méthode
$o->affiche() ;

$o->madonnee = "coucou" ;


unset($o) ;
```

Valeur par défaut des attributs

```php
<?php
class MaClasse {
  private $madonnee = "Défaut" ;
  function affecte($val) {
      $this->madonnee = $val ; }
  function affiche() {
      echo "madonnee : ".$this->madonnee ; }
}
$o = new MaClasse() ;
$o->affiche() ;
$o->affecte("Nouvelle") ;
$o->affiche() ;
```


Attributs et méthodes statiques

- Mot clé static
- Attributs et méthodes utilisables sans instance de la classe (=attributs et méthode de classe)
- Attributs NE peuvent PAS être accédés depuis une instance

~~($objet->attribut)~~

- Attributs partagés par toutes les instances de la classe
- Méthodes peuvent être accédés depuis une instance

($objet->methode())

- Dans les méthodes, $this n'est pas disponible

Attributs et méthodes statiques

```php
class MaClasse {
  private static $n = 0 ;
  function __construct() {
      echo ++MaClasse::$n
          ." instance(s)" ; }
  function __destruct() {
      echo "destruction" ; self::$n-- ; }
}
$s = new MaClasse() ;
$t = new MaClasse() ;
unset($t) ;
$u = new MaClasse() ;
$v = new MaClasse() ;
echo MaClasse::$n ;
```


Attributs et méthodes statiques

```php
class MaClasse {
  private static $n = 0 ;
  function __construct() {
      echo ++MaClasse::$n." instance(s)\n" ; }
  function __destruct() {
      MaClasse::$n-- ; }
  static function f($i) {
      echo "Dans f() : ".($i*$i) ; }
}

$s = new MaClasse() ;
$s->f(2) ;
MaClasse::f(3) ;
```

L'héritage

```php
class Simple {
  function affiche() {
      echo "Je suis Simple" ; }
}
class Etendue extends Simple {
  function affiche() {
      parent::affiche() ;
      echo " mais aussi Etendue" ;
  }
}
$s = new Simple() ;
$e = new Etendue() ;
$s->affiche() ;
$e->affiche() ;
```


Utilisation des objets

```php
class Point {
  private $_x ;
  private $_y ;
  public function __construct($x=0, $y=0) {
      $this->_x = $x ;
      $this->_y = $y ; }
  public function set($x, $y) {
      $this->_x = $x ;
      $this->_y = $y ; }
  public function toString() {
      return "({$this->_x}, {$this->_y})" ; }
}
```


Utilisation des objets

```php
$segment = array() ;
$point = new Point() ;
for ($i=10; $i<20; $i++)
{
  $point->set($i, $i) ;
  $segment[] = $point ;
}

foreach ($segment as $k => $p)
  echo "$k: {$p->toString()}\n" ;
```

```
0: (19, 19)
1: (19, 19)
2: (19, 19)
3: (19, 19)
4: (19, 19)
5: (19, 19)
6: (19, 19)
7: (19, 19)
8: (19, 19)
9: (19, 19)
```


Utilisation des objets

```php
$segment = array() ;
$point = new Point() ;
for ($i=10; $i<20; $i++)
{
  $point->set($i, $i) ;
  $segment[] = clone $point ;
}

foreach ($segment as $k => $p)
  echo "$k: {$p->toString()}\n" ;
```

```
0: (10, 10)
1: (11, 11)
2: (12, 12)
3: (13, 13)
4: (14, 14)
5: (15, 15)
6: (16, 16)
7: (17, 17)
8: (18, 18)
9: (19, 19)
```

Objets comme arguments de fonctions

```
function origine($p) {
  $p->set(0, 0) ; }

$point = new Point(10, 10) ;

echo "avant: {$point->toString()}\n" ;
origine($point) ;
echo "apres: {$point->toString()}\n" ;
```


Gestion des exceptions

- Gestion des exception identiques à Java
- Exception peut être :
  - throw : Jetée
  - try : Essayée
  - catch :
- Exception jetée : code après throw non exécuté
- Capture : 1 ou plusieurs blocs (selon type)
- Exception non capturée : erreur fatale

Gestion des exceptions

```
$a = 5;
try {
$resultat = $a / 0;
echo $resultat ;
}
catch (DivisionByZeroError $exception)
{
echo "Exception division par 0";

}
```


Gestion des exceptions

- Possibilité de définir une exception

```
<?php
class myException extends Exception{
function __toString()  {
       return "<table border=\"1\"> <tr>
      <td><strong>Exception ".$this->getCode()."
</strong>: ".$this->getMessage()."<br />"."
dans".$this->getFile()." ligne".$this->getLine()."
</td>       </tr>       </table><br />";
 }
}

?>
```

Gestion des exceptions

- Utilisation de l'exception

```
try
{
  throw new myException("Exception fatale", 42);
}
catch (myException $m)
{
   echo $m;
}
```

Affichage de l'exception

```
<table border="1">
      <tr>
      <td><strong>Exception 42
      </strong>: Exception fatale<br />
      dans le fichier calcul.php ligne 21
      </td>
      </tr>
      </table><br />
```


Plan