compléments sur les objets

Affectation

- L'affectation duplique la valeur d'une variable pour les types simples et les tableaux

```
$a =  $b ;
```

$a et $b ont la même valeur mais référencent 2 contenus différents

```
<?php
$v1 = 5;
$v2 = $v1;
$v1 +=1 ;
echo '$v1 : '.$v1. '<br>';
echo '$v2 : '.$v2. '<br>';
```

```
$v1 : 6
$v2 : 5
```

Manipulation de Références

- On peut affecter une *référence* : `&$v` désigne la référence à $v et non pas sa valeur

```
$a = & $b ;
```

$a et $b référencent le même contenu

```
<?php
$v1 = 5;
$v2 =  & $v1;
$v1 +=1 ;
echo '$v1 : '.$v1. '<br>';
echo '$v2 : '.$v2. '<br>';
```

```
$v1 : 6
$v2 : 6
```

affectation d'objets

- Lorsqu'elle concerne des objets, l'affectation copie la référence

```
<?php
$o1 = new Rectangle();
$o2 = $o1;

// $o1 et $o2 référencent le même objet

$o2->largeur = 123;
print $o1->largeur ;  // 123
```

variables dynamiques

- La valeur d'une variable peut être utilisée comme un nom de variable

```php
<?php
$v1=1; $v2=2; $v3=3; $v4=4;

 $var='v2';
 echo "valeur  :  ${$var}";
 $$var = 5;
 echo "<br> valeur :  ${$var}  - $v2";
```

```
valeur: 2

valeur: 5 - 5
```

Ça marche aussi pour les attributs et les méthodes/fonctions

```php
<?php
 …

 $o = new Rectangle( 22, 33) ;

 $s = $o->calculerSurface() ;
 $o->afficher() ;

 $t = $o->longueur ;

 $att = 'largeur' ;
 $methode = 'modifierLongueur' ;

 echo $o->$att ;
 $o->$methode( 12 ) ;
```

encapsulation et méthodes d'accès

- utiliser la dynamicité pour écrire des getter/setter :

```
class Rectangle {
 private $la, $lo ;

 public function __construct( $la, $lo ) {
  $this->la = $la ;
  $this->lo = $lo ;
 }

 public function getAttr( $at) {
   if ( property_exists ($this, $at) ) {
         return $this->$at ;
  } else throw new Exception ("$at : invalid property");
 }

 public function setAttr( $at, $val ) {
   if ( property_exists ($this, $at) ) {
         $this->$at = $val ;
         return $this->$at ;
  } else throw new Exception ("$at : invalid property");
 }
}
```

```
$r= new Rectangle (5,5) ;

$la = $r->getAttr('la') ;



$r->setAttr('lo', 13) ;

$lo = $r->getAttr('lo') ;
```

- un getter générique pour tous les attributs
- un setter générique pour tous les attributs

- pas de getter/setter, mais des méthodes ***magiques***
  - les programmeurs php n'aiment pas les getter/setter

```
class Rectangle {
 private $la, $lo ;

 public function __construct( $la, $lo ) {
  $this->la = $la ;  $this->lo = $lo ;
 }

 public function __get( $at ) {
  if ( property_exists ($this, $at) ) {
    return $this->$at ; }
  else throw new Exception ("$at : invalid property") ;
 }
 public function __set( $at, $val ) {
   if ( property_exists ($this, $at) ) {
      $this->$at = $val ;
      return $this->$at ;
   } else throw new Exception ("$at : invalid property");
 }
}
```

```
$r= new Rectangle (5,5) ;

$la = $r->la ;

$r->lo = 13 ;

echo $r->lo ;
```

- l'encapsulation est réalisée en redéfinissant les accesseurs
- à l'utilisation : pas de getter/setter

- les méthodes magiques permettent des choses intéressantes :
  - accéder à des propriétés non visibles, ou n'existant pas
  - appeler des méthodes sous forme d'accès à une propriété

```
class Rectangle {
   private $largeur, $longueur ;

   public function __get( $attname ) {
   if ( property_exists ($this, $attname) )
              return $this->$attname ;

   if ($attname === 'width') return $this->largeur ;

   if ($attname === 'perimetre')
            return ($this->largeur+$this->longueur)*2 ;

  if ($attname == 'surface')
            return $this->computeSurface() ;

  throw new Exception ("$attname : invalid property") ;
}
```

```php
$r1 = new Rectangle ( 12, 24 ) ;

// appel automatique de la méthode __get
// car la propriété n'est pas accessible

print $r1->width;
print $r1->perimetre;
print $r1->surface ;
```

interface

```
interface iRectangle  {

   public function largeur( $l = null) ;
   public function longueur( $l = null) ;
   public function surface() ;
```

```
class Rectangle implements iRectangle {
 private $largeur, $longueur ;

   public function __construct( $la, $lo ) {
      $this->largeur = $la ;  $this->longueur = $lo ;

   public function __get( $at ) {…
    }
  public function __set( $at, $val ) { ...
    }

   public function largeur( $l = null ) {
        if (is_null $l) return $this->largeur ;
        $this->largeur = $l ;
    }
}
```

héritage

- héritage simple avec redéfinition de méthodes
- sous-classe : définie avec *extends*

```
class Employe extends Personne {

   protected $categorie ;

public function __construct( $n, $p, $a, $cat) {

   parent::__construct($n,$p,$a) ;
   $this->categorie = $cat ;
}

public function affiche() {
   parent::affiche() ;
   print "employé de catégorie : " . $this->categorie ;
}
```

- abstract :

```
abstract class Figure {
  …
 abstract public function move() ;
}
```

Exceptions (et erreurs en PHP 7)

- Nouvelle hiérarchie d'exceptions en PHP 7 :
  - Throwable
    - Error *(erreurs de PHP 5)*
      - ArithmeticError
        - » DivisionByZeroError
      - AssertionError
      - ParseError
      - TypeError
        - » ArgumentCountError
    - Exception
      - *MyException*
      - ...

Si l'exception n'est pas gérée, elle devient une erreur fatale traditionnelle.

Il est possible de définir un autre gestionnaire d'exception avec `set_exception_handler()`

mon_message

```
class MyException extends Exception {}

try {
   throw new MyException("mon_message");
} catch (MyException $e) {
   echo $e->getMessage();
}
```

Exercices

- TD n°3