

Développement objet

- Définition de **briques logicielles** représentant un concept, une idée ou une entité ainsi que leurs interactions : les objets
- Un objet est une **structure de données** comprenant également les **fonctionnalités** de traitement des données
- L'objet est vu au travers de ses **spécifications**
- Les concepts associés sont :
 - **Encapsulation**
 - **Héritage**
 - **Polymorphisme**

Classe

- Une **classe définit un modèle**, un moule, à partir duquel tous les objets de classe seront créés
- La classe décrit les **données internes** ainsi que les **fonctionnalités** des objets
- La **classe est une vision « inerte »**, une recette de cuisine, visant à décrire la structure et le comportement des objets qui seront créés
- La **construction d'un objet** à partir de la classe génératrice s'appelle **instanciation**
- Les **objets**, entités « vivantes » en mémoire, **sont des instances** de la classe

Instanciation

- La classe est une description « inerte »
- Les objets doivent être instanciés à partir de la classe génératrice pour exister et devenir fonctionnels
- Exemple : la classe Animal
`$bambi = new Animal();
$scrat = new Animal();
$melman = new Animal();`

Encapsulation

- Procédé consistant à rassembler les données et les traitements au sein des objets
- L'implémentation interne des objets est cachée
- Les objets sont vu à travers leurs spécifications
- Les données internes et les fonctionnalités possèdent un niveau de visibilité et peuvent éventuellement être masquées :
 - Public
 - Privé
 - Protégé

Encapsulation

- Les **données internes** des objets sont appelées **attributs**
- Les **fonctionnalités** des objets sont appelées **méthodes**
- Méthodes habituelles :
 - **Constructeur / destructeur**
 - **Accesseurs / modificateurs** (getters / setters)
- Référence à l'objet courant dans la description de la classe : **\$this**

Visibilité

- **Publique** :

Les données internes et les méthodes sont accessibles par tous

- **Protégé** :

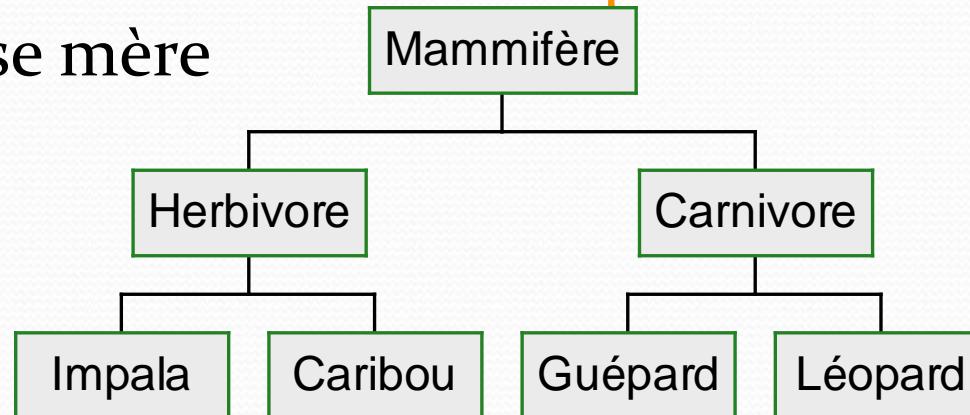
Les données internes et les méthodes sont accessibles aux objets dérivés

- **Privé** :

Les données internes et les méthodes ne sont accessibles qu'aux objets de la classe

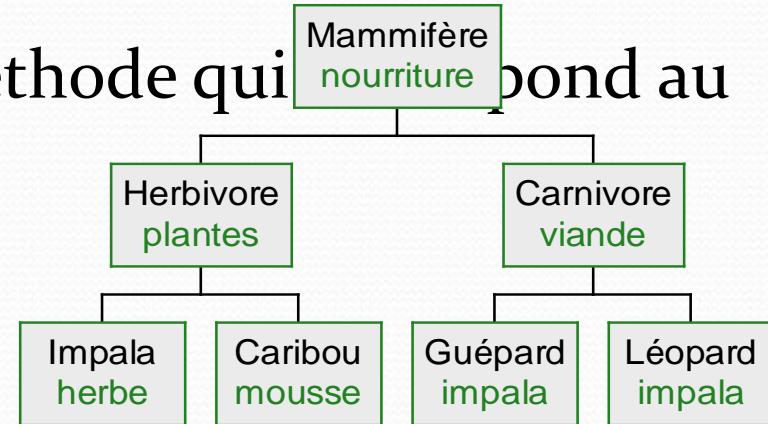
Héritage ou dérivation ou extension

- Création de **nouvelles classes à partir** du modèle **d'une classe existante**
- La nouvelle classe possède **tous les attributs et méthodes de la classe mère**
- La nouvelle classe peut proposer de **nouveaux attributs et de nouvelles méthodes** ou **spécialiser des méthodes de la classe mère**



Polymorphisme

- Choix dynamique de la méthode qui correspond au type réel de l'objet
- Méthode mange()



- function nourriture(Mammifere \$m) {
 return \$m->mange() ;
}
- \$i = new Impala() ; nourriture(\$i) ;
- \$c = new Carnivore() ; nourriture(\$c) ;

herbe
viande

Développement objet en PHP

- PHP 4
 - Classes (tout est public)
 - Constructeur
 - Héritage
 - Méthodes statiques implicites
 - Outils pour la linéarisation (sauvegarde sur disque)
- PHP 5
 - Classes (choix public, protected, private)
 - Constructeur et destructeur
 - Membres et méthodes statiques
 - Classes et méthodes abstraites
 - ...

Définition d'une classe

```
<?php  
class MaClasse {  
    var $madonnee ; // Attribut  
  
    // Constructeur  
    function MaClasse($param) {  
        $this->madonnee = $param ;  
    }  
    function affiche() { // Méthode  
        echo get_class($this)  
        . " contient " . $this->madonnee ;  
    }  
}  
?>
```

Déclaration de classe

<?php

class MaClasse {

var \$madonnee ; // Attribut

Attribut public

// Constructeur

function MaClasse(\$param) {
 \$this->madonnee = \$param ;

Constructeur public

Référence à
l'objet courant

}

function affiche() { // Méthode

echo get_class(\$this)
. " contient " . \$this->madonnee ,

Accès à un attribut

Méthode publique

}

Donne le nom de la
classe de l'objet
courant

Utilisation d'une classe

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

Inclusion de la définition de la classe

```
// Nouvel objet
```

Création d'un objet

```
maclasse contient 12 ;  
// Utilisation d'une méthode
```

Méthode affiche de l'objet \$o

```
$o->affiche() ;
```

Impossible de restreindre l'accès aux attributs

```
$o->madonnee = "coucou" ;
```

```
?>
```

Héritage en PHP 4

```
<?php  
class MaClasseHeritee extends MaClasse {  
    var $autredonnee ;  
    function MaClasseHeritee($param1, $param2) {  
        parent::MaClasse($param1);  
        $this->autredonnee = $param2;  
    }  
    function affiche() {  
        MaClasse::affiche();  
        echo " et ".get_class($this)  
            ." contient ".$this->autredonnee;  
    }  
}
```

Héritage de
MaClasse

Opérateur de
résolution de portée
(Paamayim
Nekudotayim)

Méthode MaClasse
du parent

Méthode affiche de
la classe
MaClasse

Méthodes de classe implicites

```
<?php
class MaClasse {
    var $madonnee ; // Attribut
    // Constructeur
    function MaClasse($param) {
        $this->madonnee = $param ;
    }
    function affiche() { // Méthode
        echo $this->madonnee ;
    }
    function statique() // Méthode
        echo "Statique !!!" ;
    }
?>
```

Méthode publique

Ne fait pas référence à
\$this
→ Méthode statique
(méthode de classe)

Méthodes de classe

```
<?php  
require "maclasse.php" ;  
  
// Création d'un objet  
$o = new MaClasse(12) ;  
  
// Utilisation d'une méthode  
12>affiche() ;  
  
// Utilisation d'une méthode implicite  
Statique !!! tique() ;  
  
?>
```

Inclusion de la définition de la classe

Création d'un objet

Méthode affiche de l'objet \$o

Impossible de restreindre l'accès aux attributs

Développement objet en PHP 5

A partir de maintenant,
développement en **PHP 5**

Définition d'une classe

```
<?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

Utilisation d'une classe

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

Inclusion de la définition de la classe

```
// Nouvel objet  
maclasse contient 12 ;  
// Utilisation d'une méthode
```

Création d'un objet

```
class MaClasse {  
    private $madonnee ;  
    ...  
}
```

Méthode affiche de l'objet \$o

```
function __destruct() {  
    echo "Destruction..." ;  
}
```

L'attribut est privé

```
unset($o) ;
```

Destruction de l'objet \$o

Valeur par défaut des attributs

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

Attribut avec valeur par défaut

madonnee : Défaut

\$o = new MaClasse() ;

madonnee : Nouvelle

\$o->affecte("Nouvelle") ;

\$o->affiche() ;

Nouvel objet

Affichage

Affectation

Affichage

Attributs et méthodes de classe

- 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->méthode()**)
- Dans les méthodes, **\$this** n'est pas disponible

Attributs statiques

```
class MaClasse {  
    private static $n = 0 ;  
    function __construct() {  
        echo ++MaClasse::$n  
            ." instance(s)" ; }  
    function __destruct() {  
        1 instance(s) on" ; self::$n-- ; }  
    2 instance(s)  
destruction  
2 instance(s)}
```

Attribut privé statique :
ne peut être accédé
que par des méthodes
de la classe

Accès à l'attribut
statique

Fatal error: Cannot access private property
MaClasse::\$n in **dummy.php** on line 37

Sy = new MaClasse();
10:11:29 Programmation Web 2011-2012

echo MaClasse::\$n ;

Méthodes statiques

```
class MaClasse {  
    private static $n = 0 ;  
    function __construct() {  
        echo ++MaClasse::$n." instance  
    }  
    function __destruct() {  
        MaClasse::$n-- ; }  
    static function f($i) {  
        echo "Dans f() : ".$i*$i  
    }  
}  
1 instance(s)  
Dans f() : 4  
Dans f() : 9  
$s->f(2) ;  
MaClasse::f(3) ;
```

Méthode publique statique

Appel à partir d'une instance Toléré

Appel sans instance

Constantes de classe

```
class MaClasse {  
    const constante = "Valeur";  
    public function montre() {  
        echo self::constante;  
    }  
}
```

Constante publique de classe

Accès à la constante de classe depuis la classe

```
Valeur  
Valeur  
$c->montre();  
echo MaClasse::constante;
```

Accès à la constante de classe à l'extérieur de la classe

Héritage

```
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();
```

Je suis Simple

Je suis Simple mais aussi Etendue

Classe simple

Une méthode publique

Classe étendue héritant de la classe simple

Surcharge de la méthode

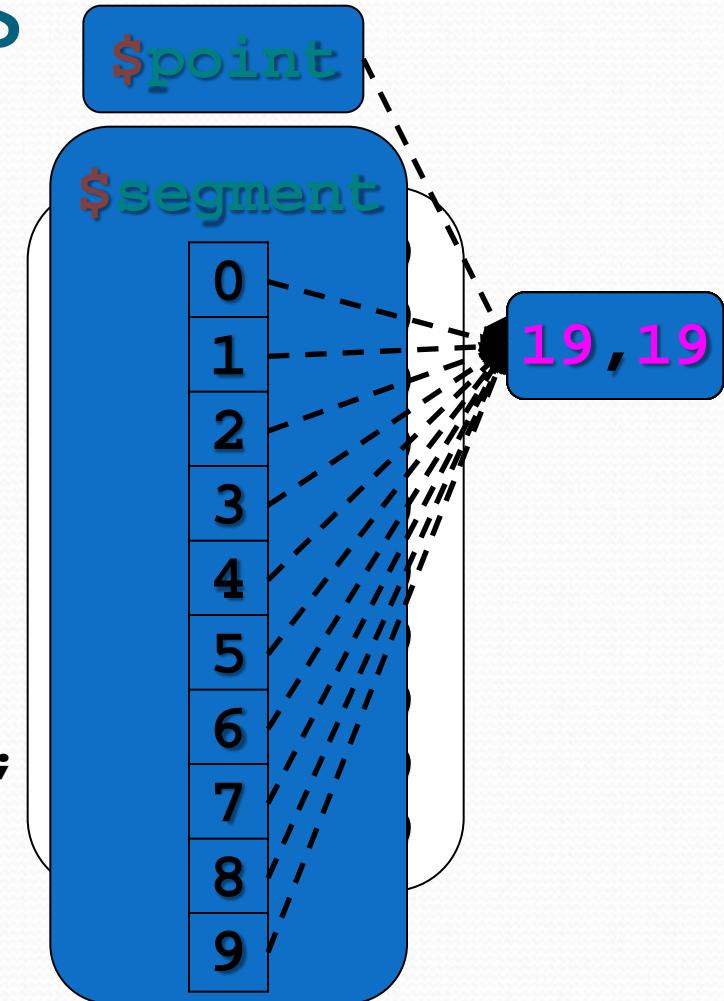
Appel de la méthode du parent

Attribution d'objets

```
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})" ; }  
}
```

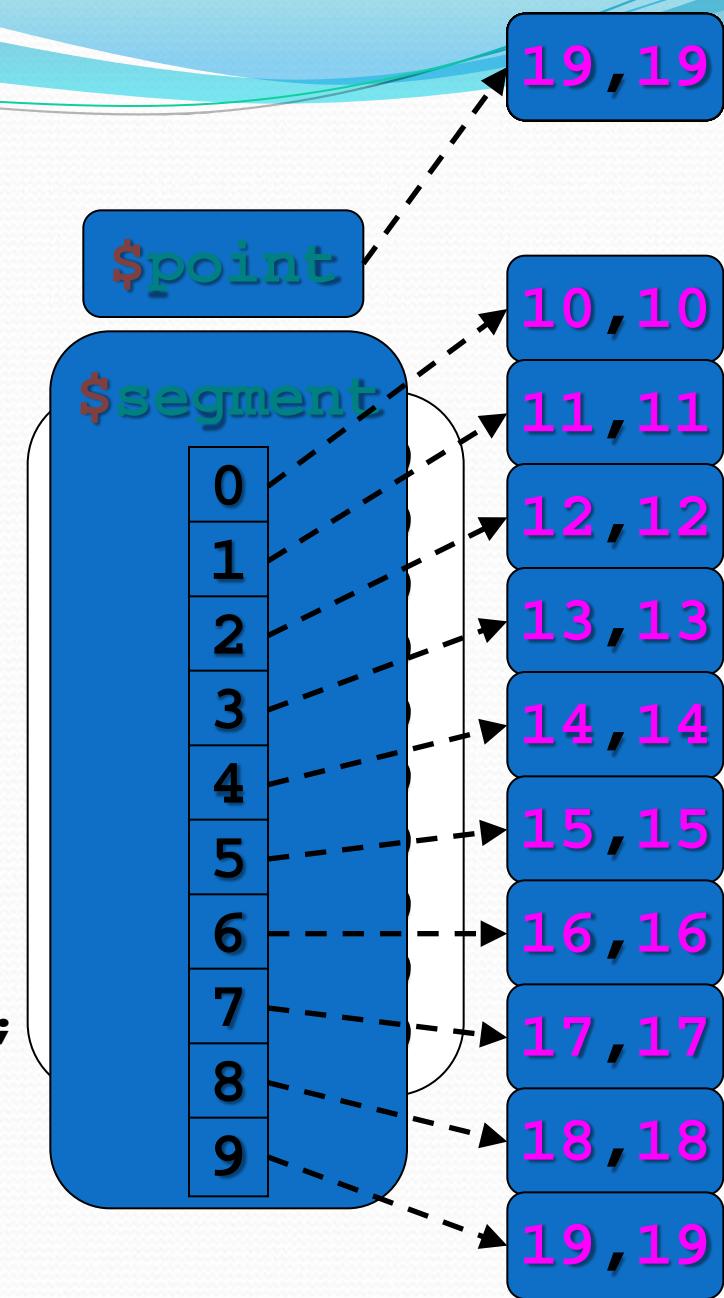
Attribution d'objets

```
$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()}<br>\n" ;
```



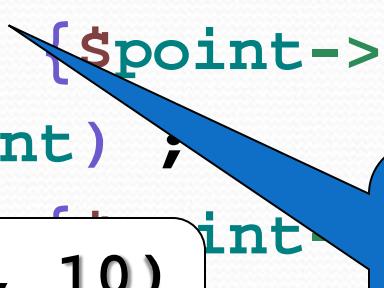
Clonage d'objets

```
$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()}<br>\n" ;
```



Objets comme arguments de fonctions

```
function origine($p) {  
    $p->set(0, 0) ; }  
  
$point = new Point(10, 10) ;  
  
echo "avant: {$point->toString()}<br>\n" ;  
origine($point) ;  
echo "apres: {$point->toString()}" ;  
  
avant: (10, 10)  
apres: (0, 0)
```



Passage de l'objet
Point par référence

Objets dans les chaînes de caractères

- Problème :
 - ambiguïté
 - non évaluable
- Chaîne contenant :
 - un attribut d'un objet dans une chaîne
"a : \$a->attribut"
 - le résultat d'une méthode d'un objet dans une chaîne
"résultat : \$a->calcule()
 - une entrée de tableau associatif
 - **"valeur : \$tab['cle']"**
 - une variable suivie de texte (sans espace)
"\\$\\$a contient \\$aeuros"

Objets dans les chaînes de caractères

- Solution :
 - effectuer des concaténations (pénible)
 - délimiter par { }
- Chaîne contenant :
 - un attribut d'un objet dans une chaîne
"a : {\$a->attribut}"
 - le résultat d'une méthode d'un objet dans une chaîne
"résultat : {\$a->calcule()}"
 - une entrée de tableau associatif
 - **"valeur : {\$tab['cle']}"**
 - une variable suivie de texte (sans espace)
"\\$a contient {\$a} euros"

Gestion des erreurs : exceptions

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

Utilisation des exceptions

```
try {  
    ...  
    // Toujours lancer cette erreur  
    throw new Exception($error);  
}
```

/* Le code après une exception
jamais exécuté. */

Capturer

```
Capture Exception: Toujours lancer cette erreur
```

```
catch (Exception $e) {
```

```
    echo "Capture Exception: "
```

```
    $e->getMessage() . "\n";
```

Bonjour le monde

```
// Poursuite de l'exécution
```

```
echo 'Bonjour le monde';
```

Classe Exception PHP 5

```
<?php
class Exception {
    protected $message = ''; // message de l'exception
    protected $code = 0;      // code de l'exception
    protected $file;         // fichier source de l'exception
    protected $line;         // ligne de la source de l'exception

    function __construct(string $message=NULL, int code=0);

    final function getMessage();           // message de l'exception
    final function getCode();             // code de l'exception
    final function getFile();            // nom du fichier source
    final function getLine();            // ligne du fichier source
    final function getTrace();           // un tableau de backtrace()
    final function getTraceAsString();   // chaîne de trace
    function _toString();                // chaîne pour l'affichage
} ?>
```