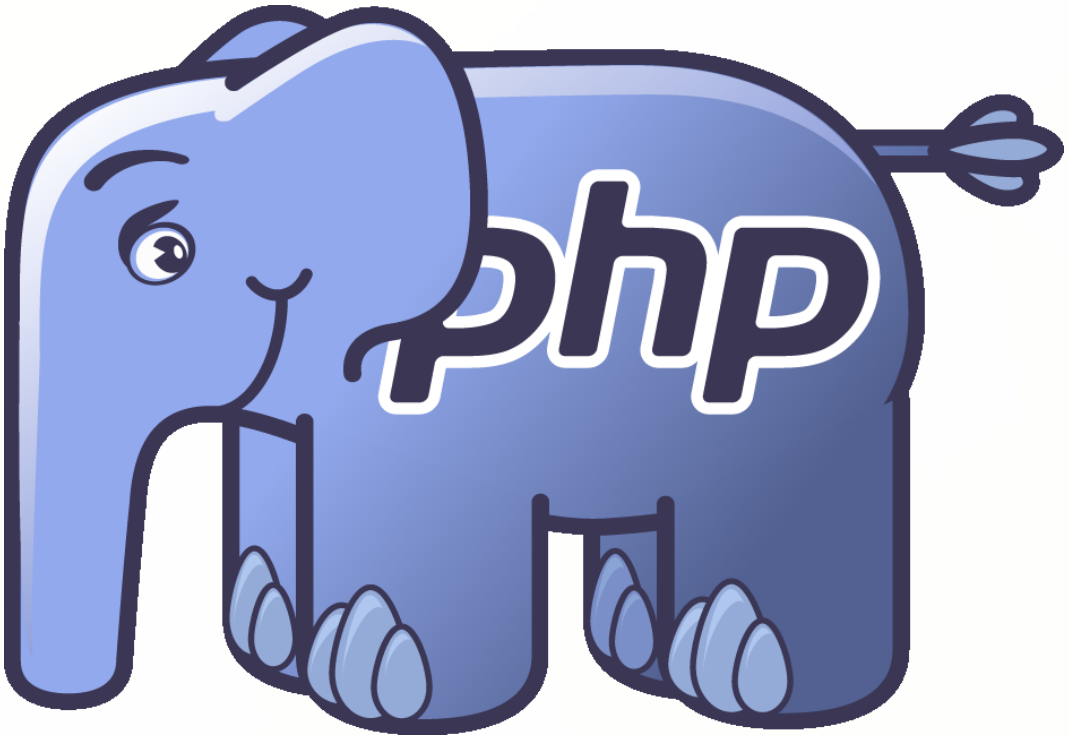
PHP Orienté Objet



RAPPELS DE DÉVELOPPEMENT OBJET

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 la 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 vus 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 (ou propriétés voire champs)
- 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

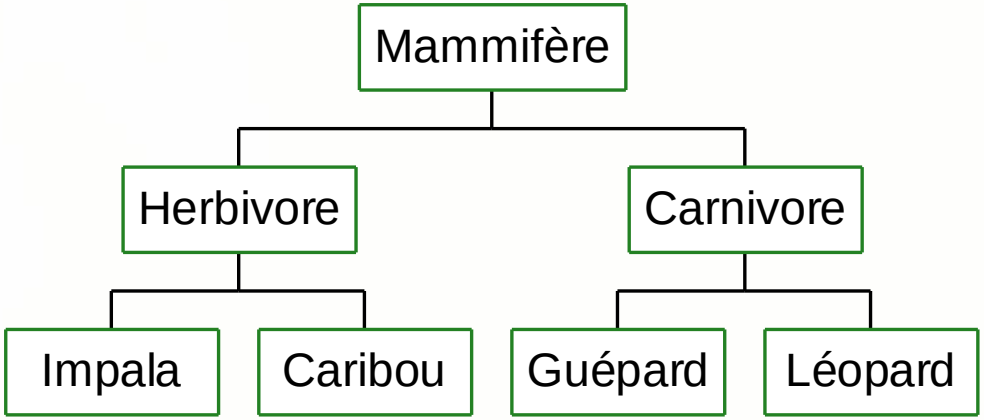
- Publique :
  Les données internes et les méthodes sont accessibles par tous

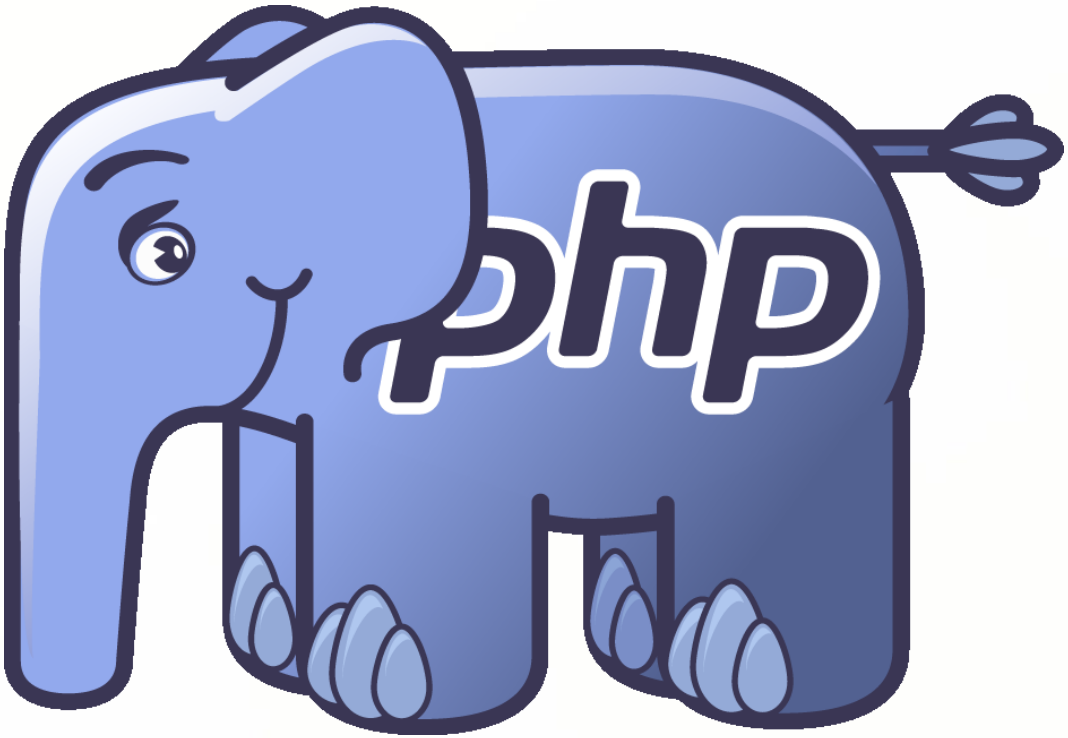
- Protégé :
  Les données internes et les méthodes sont accessibles aux objets dérivés

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

Héritage ou dérivation ou extension

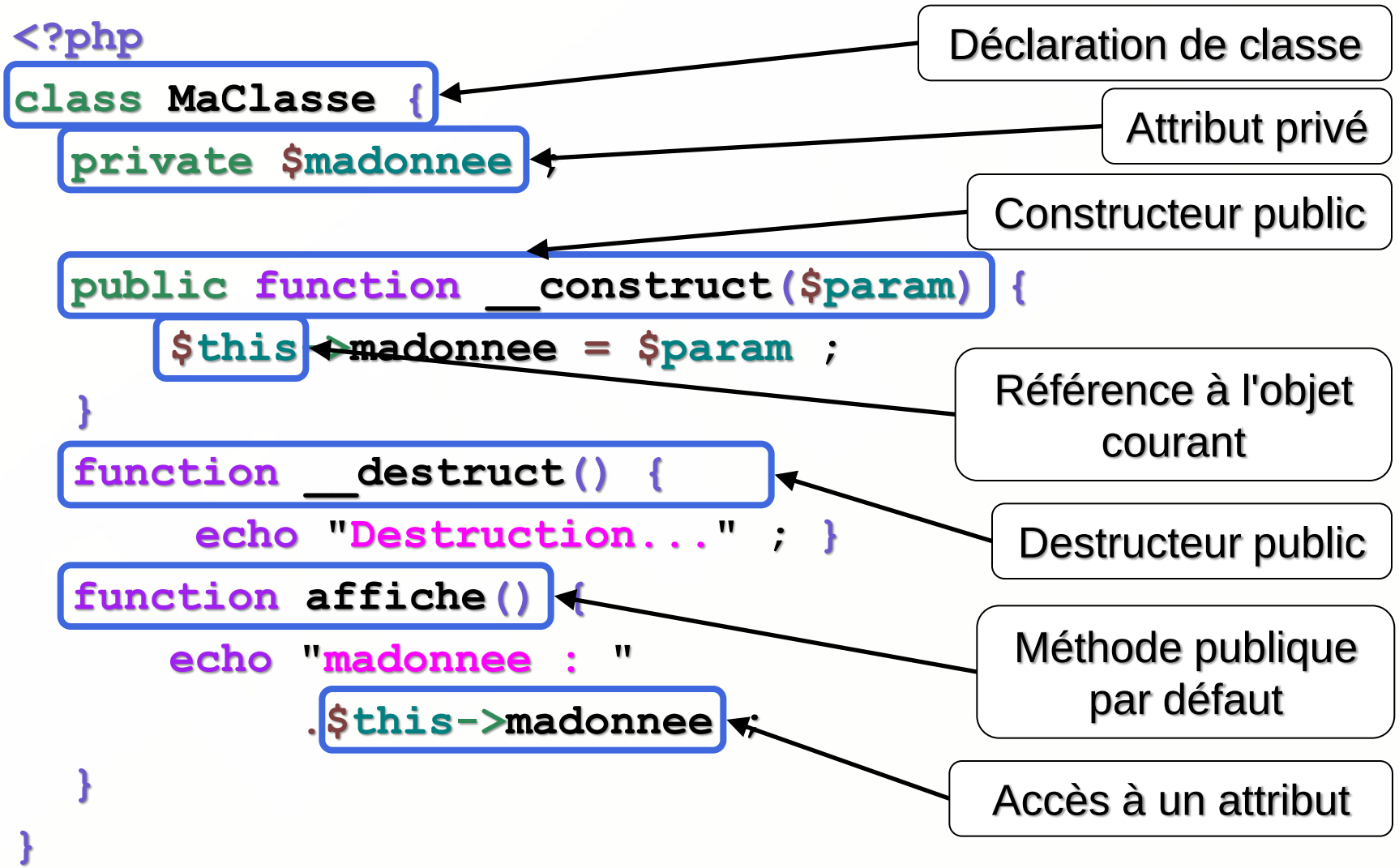
- Création de nouvelles classes à partir du modèle d'une classe existante
- La nouvelle classe possède tous les attributs et méthodes de la classe mère
- La nouvelle classe peut proposer de nouveaux attributs et de nouvelles méthodes ou spécialiser des méthodes de la classe mère





CLASSE, ATTRIBUTS ET MÉTHODES D'INSTANCE

Définition d'une classe



Utilisation d'une classe

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

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

$o->madonnee = "coucou" ;

unset($o) ;
```

Inclusion de la définition de la classe

Création d'un objet

Méthode affiche de l'objet $o

L'attribut est privé

Destruction de l'objet $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() ;
```

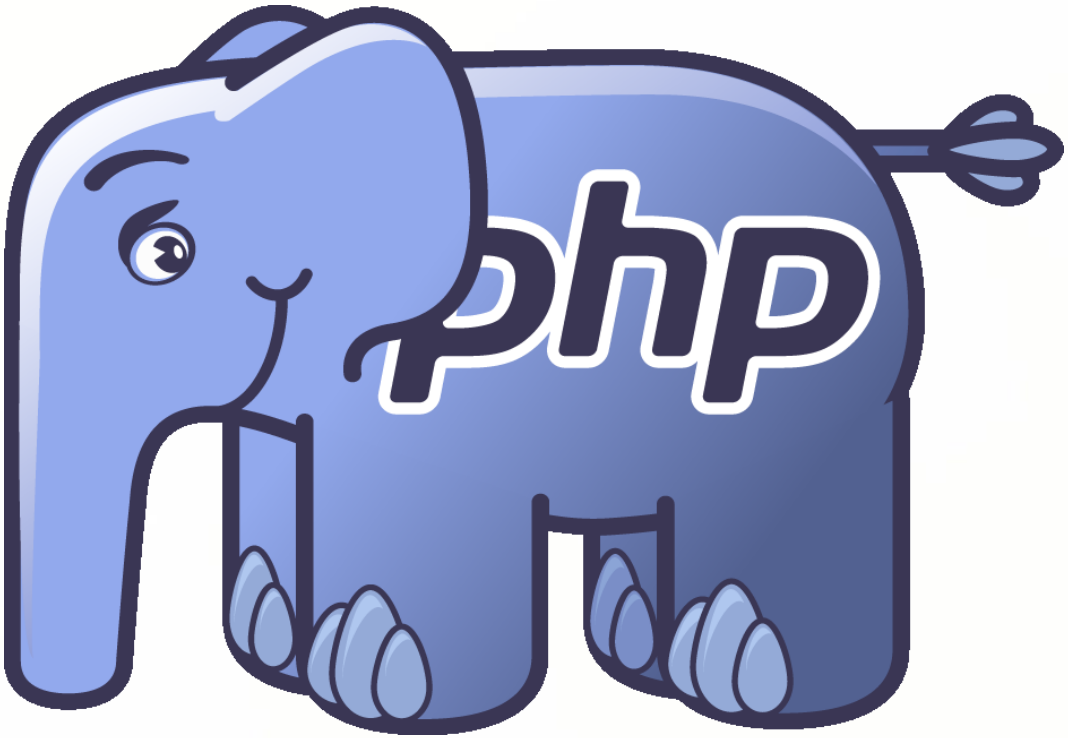
Attribut avec valeur par défaut

Nouvel objet

Affichage

Affectation

Affichage



ATTRIBUTS ET MÉTHODES DE CLASSE

Attributs et méthodes de classe

- Mot clé `static`
- Attributs et méthodes utilisables sans instance de la classe (=attributs et méthode de classe)
- 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 statiques

```php
class MaClasse {
   private static $n = 0 ;
    function __construct() {
        echo ++self::$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 ;
```

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

Accès à l'attribut statique

```
1 instance(s)
2 instance(s)
destruction
2 instance(s)
3 instance(s)
```

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

Méthodes statiques

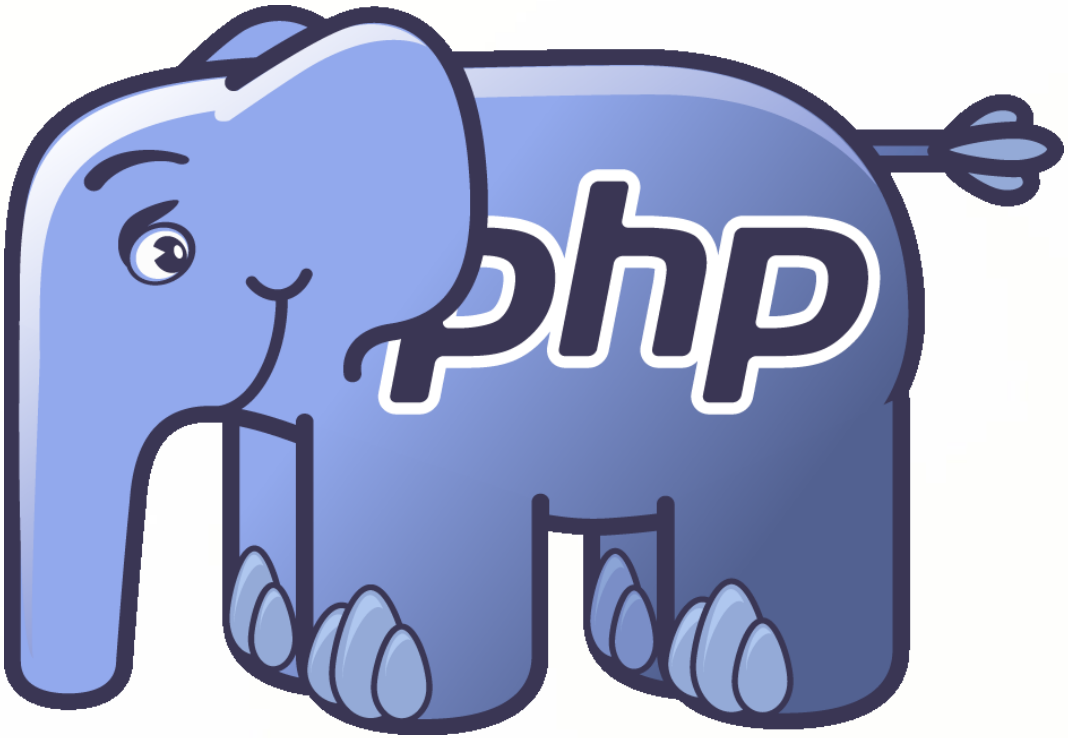
```
class MaClasse {
   private static $n = 0 ;
   function __construct() {
       echo ++self::$n." instance(s)\n" ; }
   function __destruct() {
        self::$n-- ; }
   public static function f($i) {
       echo "Dans f() : ".($i*$i) ; }
}

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

Méthode publique statique

Appel à partir d'une instance
Toléré

```
1 instance(s)
Dans f() : 4
Dans f() : 9
```

Appel sans instance



CONSTANTES

Constantes de classe

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

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

Constante publique de classe

Accès à la constante de classe depuis la classe

```
Valeur
Valeur
```

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

Php OO Exercice:

Construire une classe pagination où l'utilisation se fait avec le code suivant:

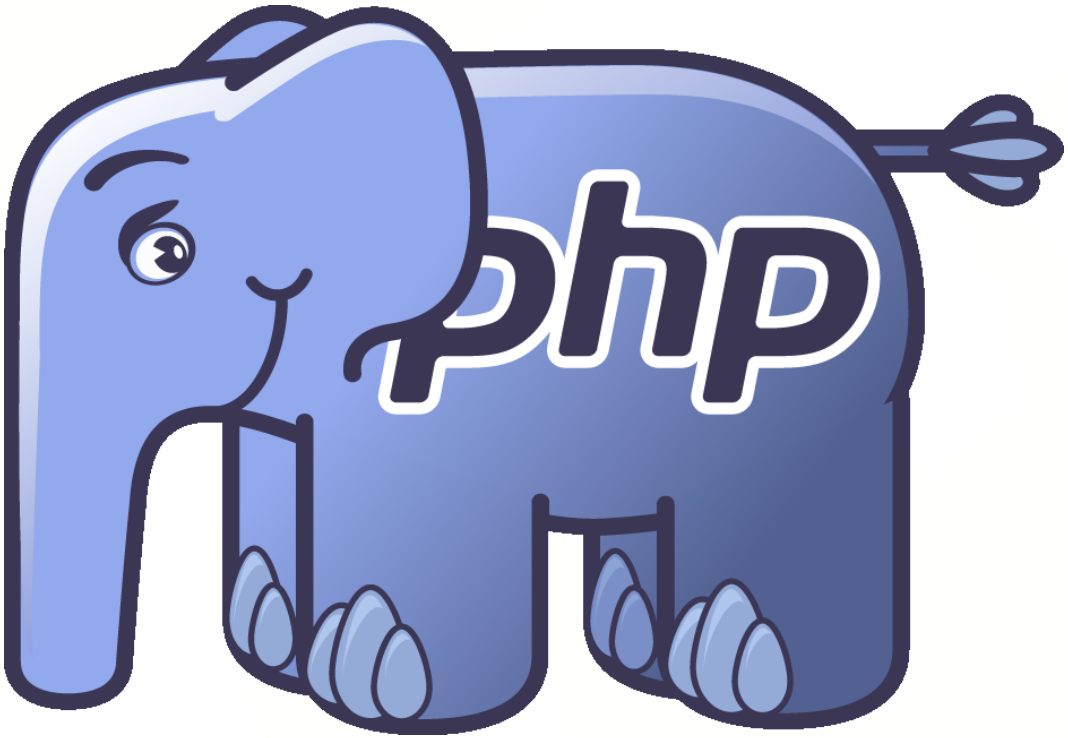
```
$pagination=new pagination();
$pagination->setMaxPages(100);
$pagination->setLinksCount(8);
$pagination->setSelectedLinkClass('SelectedPageLink');
$pagination->setLinkClass('PageLink');
$pagination->setQuickButtonClass('QuickButtons');
$pagination->setQuickBottons();
$pagination->show();
```

La liste des méthodes est:

- *setTotalPages: permet de designer le nombre total des pages.*
- *setLinksNumber: permet paramétrer le nombre de liens à afficher dans la pagination.*
- *setSelectedLinkColor: permet de designer la couleur du lien de la page en cours.*
- *setLinkColor: permet de designer la couleur des liens.*
- *setQuickButtonColor: permet de designer la couleur des boutons de navigation.*
- *addQuickBottons: permet d'ajouter les boutons de navigation.*

Les paramètres doivent avoir des valeurs par défaut tel que le code suivant doit permettre l'affichage de la pagination avec ces valeurs:

```
$pagination=new pagination();
$pagination->show();
```



HÉRITAGE

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() ;
```

Classe simple

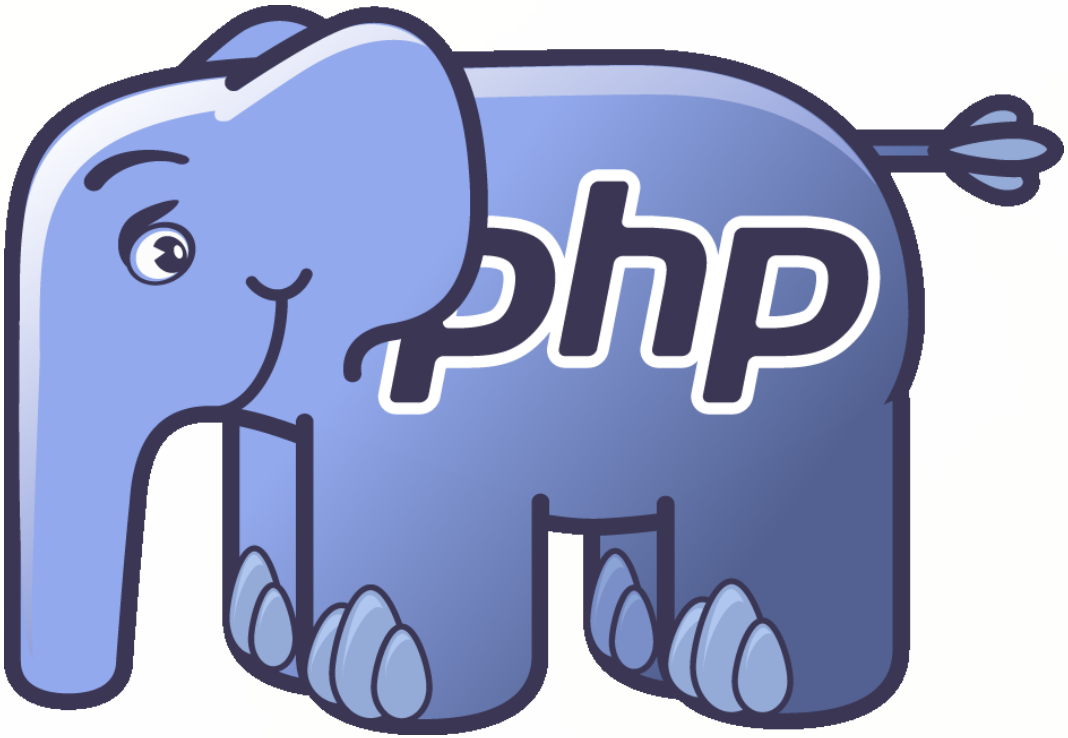
Une méthode publique

Classe étendue héritant de la classe simple

Surcharge de la méthode

Appel de la méthode du parent

```
Je suis Simple
Je suis Simple mais aussi Etendue
```

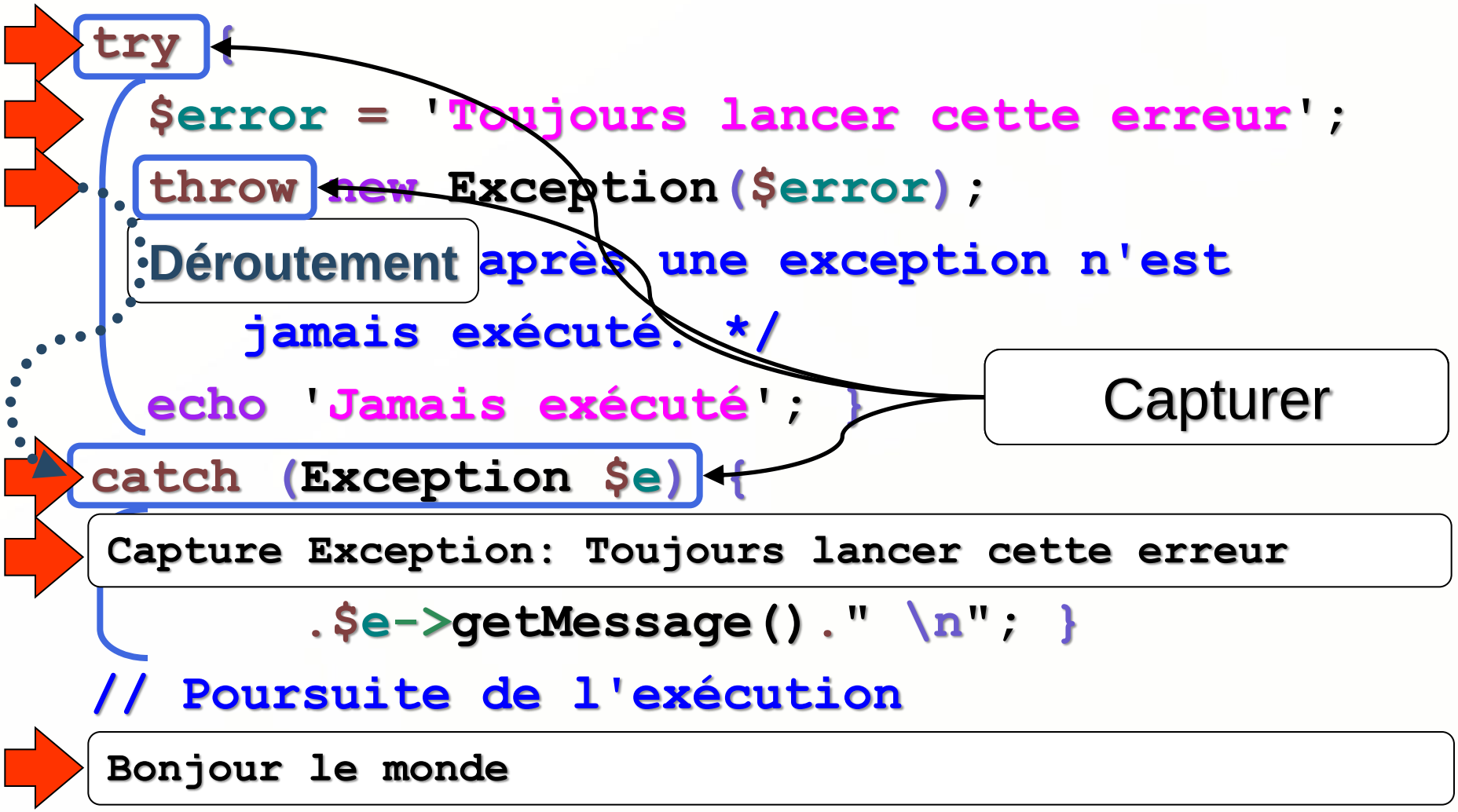


EXCEPTIONS

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



Classe **Exception** PHP 5

```php
<?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
} ?>
```