

PHP : la programmation objet (PHP-OO)
Partie 1

Benoît Roche
@rocheb83



LES FONDAMENTAUX

Benoît Roche / @rocheb83

2



PHP OO : PRÉSENTATION

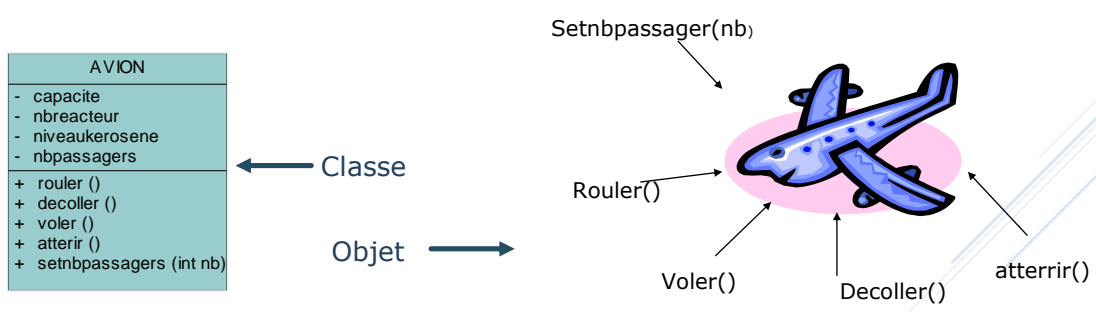
- Une classe est un type complexe, pouvant contenir :
- **Des Propriétés**
et/ou
 - **Des méthodes**
et/ou
 - **Des événements**

- On pourra trouver des classes :
- **Métier**
 - **Techniques**
 - **widgets**



PHP OO : PRÉSENTATION

Un objet est une variable dont le type est une classe.
On dit aussi que l'objet est une instance d'une classe





PHP OO : PRÉSENTATION

Convention de nommage

Upper Camel Case	une classe	ClsExemple
	une interface	IntExemple
	Un espace de noms	NsApplication
lowerCamelCase	Une variable	maVariable
	Une fonction	maFonction
	Un attribut	unAttributPrive
	Une méthode	uneMethode
MAJUSCULE	Une constante	MACONSTANTE



PHP OO : PRÉSENTATION

Syntaxe de création d'une classe :

```
<?php

class NomClasse { // mot-clé class suivi du nom de la classe.

    private $attributPrive;

    public function fonctionPublique() {

    }

    private function fonctionPrivee() {

    }

}
```



Une classe = 1 fichier
Dans un dossier spécifique
Avec la même extension (.php)



PHP OO : PRÉSENTATION

Instanciation d'une classe :

- Pour créer une instance d'une classe, le mot-clé *new* doit être utilisé.
- Un objet sera alors créé à moins qu'il ait un constructeur défini qui lance une exception en cas d'erreur

```
$objet= new NomClasse();
```



Utiliser le try... catch lors des instanciations

[Code](#)

Benoît Roche / @rocheb83

7



PHP OO : PRÉSENTATION

Visibilité des membres :

Pour chaque attribut ou méthode, on aura à choisir 3 niveaux de visibilité :

- ✓ **public** : n'importe qui a accès à la méthode ou à l'attribut demandé.
- ✓ **private** : seule la classe ayant défini l'élément peut y accéder.
- ✓ **protected** : seule la classe ainsi que ses sous classes éventuelles (classes héritées, on verra ce que c'est plus loin).

Benoît Roche / @rocheb83

8



PHP OO : PRÉSENTATION

Accès à un membre de la classe :

➤ Accès à un membre public depuis l'extérieur de la classe :

✓ **Attribut** : `UneClasse->unAttribut` ... **Fortement déconseillé**



✓ **Méthode** : `UneClasse->uneMethode(liste_des_paramètres)`

➤ Accès à un membre depuis l'intérieur de la classe (**opérateur \$this**):

✓ **Attribut** : `$this->unAttribut`

✓ **Méthode** : `$this->uneMethode(liste_des_paramètres)`

\$this et une référence à l'objet courant.

Benoît Roche / @rocheb83

9



PHP OO : PRÉSENTATION

Accesseurs et mutateurs (getters et setters) :

✓ Un **accesseur** est une méthode publique qui permet d'obtenir la valeur d'un attribut privée ou protégée.

✓ Un **mutateur** est une méthode publique qui permet d'affecter une valeur à un attribut privé ou protégé.

Benoît Roche / @rocheb83

10



PHP OO : PRÉSENTATION

Accesseurs et mutateurs (getters et setters) :

Quelques recommandations pour les accesseurs et mutateurs :

Préfixer les noms des attributs privés ou protégés avec un underscore afin de les distinguer plus rapidement à la relecture du code.

Utiliser autant que possible les conventions de nommage pour les accesseurs et les mutateurs :

- **getNomAttribut()**
- **setNomAttribut()**

Benoît Roche / @rocheb83

11



PHP OO : PRÉSENTATION

Accesseurs et mutateurs (getters et setters) :

```
class employe // Présence du mot-clé class suivi du nom de la classe.
{
    private $_numemp;
    private $_nomemp;
    private $_prenomemp;
    protected $_salaire;
    public function setSalaire($unSalaire)
    {
        if($unSalaire>1000)
        {
            $this->_salaire=$unSalaire;
        }
        else
        {
            throw new Exception("Le salaire ne peut etre supérieur a 1000");
        }
    }
    public function getSalaire()
    {
        return $this->_salaire;
    }
}
```



Les principales IDE permettent de générer automatiquement les getters et/ou setters

Benoît Roche / @rocheb83

12



PHP OO : PRÉSENTATION

Constructeur de classe :

Le constructeur est une méthode appelée automatiquement au moment de l'instanciation d'un objet. objet.

On peut utiliser le constructeur pour initialiser les attributs d'un objet. Le constructeur n'est pas toutefois obligatoire

Il existe un constructeur par défaut

Pour définir un constructeur, on implémente **LA méthode magique**
`__construct`

[Démonstration](#) [code](#)



La surcharge du constructeur ne se fait pas en php.

Benoît Roche / @rocheb83

13



PHP OO : PRÉSENTATION

Destructeur de classe :

Le **destructeur** est une méthode appelée automatiquement, soit :

- ✓ à la fin d'exécution du script
- ✓ À l'appel de la fonction delete
- ✓ À l'appel de la fonction unset()

Pour définir un destructeur, on implémente **LA méthode magique**
`__destruct`

Benoît Roche / @rocheb83

[Démonstration](#) [code](#)

14



PHP OO : PRÉSENTATION

Quelques fonctions PHP intéressantes :

La fonction **get_class(unObjet)**
Retourne le nom de la classe passée en paramètre

La fonction **is_a(unObjet, UnNomDeClasse)**
Retourne vrai si l'objet unObjet appartient à la classe UnNomDeClasse ou à une classe parente

La fonction **class_exists(unNomDeClasse [, unBooleen])**
Retourne vrai si la classe unNomDeClasse existe. Si le deuxième paramètre facultatif vaut true, il y a appel à la méthode magique __autoload()



PHP OO : PRÉSENTATION

Quelques fonctions PHP intéressantes :

```
class NomClasse { // mot-clé class suivi du nom de la classe.

    private $attributPrive;

    public function fonctionPublique() {
    }

    private function fonctionPrivee() {
    }

    public function test() {
        echo (method_exists($this, "fonctionPrivee")) ? ("<br>I
    }
}
```



```
$objet = new NomClasse();
echo get_class($objet);
echo (is_a($objet, "NomClasse")) ? ("<br>c'est un objet de la classe NomC
echo (class_exists("UneClasse")) ? ("<br>La classe UneClasse existe") :
```

NomClasse
c'est un objet de la classe NomClasse
La classe UneClasse n'existe pas



PHP OO : PRÉSENTATION

Quelques fonctions PHP intéressantes :

- La fonction **method_exists(unObjet, uneMethode)**
Retourne vrai si la méthode uneMethode existe pour l'objet unObjet
Ne veut pas dire que la méthode soit invoquable !!! 
- La fonction **is_callable(unTableau)**
unTableau est un array(\$objet, \$nomFonction)
Retourne vrai si la méthode \$nomFonction peut être invoquée depuis l'objet \$objet
- La fonction **property_exists(unNomDeClasse , unePropriete)** 
Retourne vrai si la propriété existe dans la classe unNomDeClasse
Ne veut pas dire que la méthode soit invoquable !!!



PHP OO : PRÉSENTATION

Quelques fonctions PHP intéressantes :

```
echo (method_exists($objet, "fonctionPublique")) ? ("<br>La méthode publique existe pour l'objet") : ("<br>La méthode publique n'existe pas pour l'objet");
echo (method_exists($objet, "fonctionPrivee")) ? ("<br>La méthode privée existe pour l'objet") : ("<br>La méthode privée n'existe pas pour l'objet");
$argument = array($objet, "fonctionPublique");
echo (is_callable($argument)) ? ("<br>La méthode publique existe pour l'objet") : ("<br>La méthode publique n'existe pas pour l'objet");
$argument = array($objet, "fonctionPrivee");
echo (is_callable($argument)) ? ("<br>La méthode privée existe pour l'objet") : ("<br>La méthode privée n'existe pas pour l'objet");
echo (property_exists("NomClasse", "attributPrive")) ? ("<br>L'attribut privé existe") : ("<br>L'attribut privé n'existe pas");
?>
```

La méthode publique existe pour l'objet
La méthode privée existe pour l'objet
La méthode privée existe pour l'objet
La méthode publique existe pour l'objet
La méthode privée n'existe pas pour l'objet
L'attribut privé existe



PHP OO : PRÉSENTATION

Constructeur/destructeur de classe :

```
public function __construct($num, $nom, $prenom) { // debut du programme
    try {
        $_numemp = $num;
        $_nomemp = $nom;
        $_prenomemp = $prenom;
        $salaire=10000;
        echo "dans le constructeur <br />";
    } catch (Exception $e) {
        throw Exception();
    }
}

public function __destruct() {
    try {
        echo "dans le destructeur <br />";
    } catch (Exception $e) {
        throw Exception();
    }
}
```

↓

dans le constructeur
dans le constructeur
dans le destructeur
Fin du script
dans le destructeur

Benoît Roche / @rocheb83

19



PHP OO : EXERCICE

Surcharge du constructeur:

La surcharge du constructeur ne se fait pas en php.

✓ **PHP nous donne toutefois l'occasion de simuler la surchargeen déclarant un constructeur sans paramètre et en utilisant dans le constructeur les fonctions liées au passage de paramètres en PHP**



Exercice:

Surchargez le constructeur de la classe Personnage. Vous pourrez lui passer aucun paramètre, 3 paramètres (numéro, nom et prénom) ou 4 paramètres (numéro, nom et prénom et salaire), les numéros et salaires doivent être numériques. Tester aussi nom et prénom avec une expression régulière. Ils commencent obligatoirement par une majuscule suivie de caractères alphabétiques. Dans le cas où il n'y a que 3 paramètres, le salaire est mis à 0.

si il y a 1 ou 2 paramètres, une exception est déclenchée. Le message affiché sera ERREUR

Le nombre d arguments passés au constructeur est invalide

Testez avec toutes les combinaisons possibles :

nombre de paramètres et initiale majuscule.

Benoît Roche / @rocheb83

20



PHP OO : PRÉSENTATION

L'opérateur de résolution de portée :

L'opérateur de résolution de portée dont le symbole est le "double deux points" (::), permet d'accéder :

- ✓ aux membres **statiques**
- ✓ aux membres **constants**
- ✓ aux éléments redéfinis par la classe, avec l'opérateur **parent**.

En dehors de la classe, on fait précéder cet opérateur du nom de la classe
Dans la classe on utilise le mot clé **self**

Exemples :

Hors de classe	Dans la classe
uneClasse::uneConstante	self::UNECONSTANTE
uneClasse::uneVarStatique	self::uneMethode()
uneClasse::une fonctionStatique()	parent::uneMethode()



PHP OO : PRÉSENTATION

Les membres statiques (static):

Ce sont des membres qui appartiennent à la classe et non à un objet de la classe.

Un membre static peut être déclaré public ou privé

On peut déclarer static :

- ✓ Des données membres (propriétés)
- ✓ des méthodes membres
- ✓ Une donnée déclarée comme statique ne peut donc être accédée avec l'objet instancié d'une classe
- ✓ une méthode statique peut être accédée avec l'objet instancié d'une classe **À éviter bien entendu !!!**



PHP OO : PRÉSENTATION

Les membres statiques (static):

```
<?php
class clsTestStatique {
    private static $statVariablePrivee = 'Je suis une variable statique privee';
    public static $statVariablePublique = 'Je suis une variable statique publique';
    public static function fonctionStatique() {
        echo "fonction statique qui affiche la variable statique : " . self::$statVariablePrivee;
    }
}
$unObjet = new clsTestStatique();
echo "acces au membre fonction statique : " . clsTestStatique::fonctionStatique() . "<br />";
echo "acces au membre donnee statique : " . clsTestStatique::$statVariablePublique . "<br />";
echo "acces au membre fonction statique : " . $unObjet->fonctionStatique() . "<br />";
echo "acces au membre donnee statique : " . $unObjet->statVariablePublique . "<br />"; //
erreur
?>
```

[Démonstration](#) [code](#)

Erreur !!!

À éviter !!!



PHP OO : PRÉSENTATION

Les constantes de classe :

On peut déclarer une constante de classe dans une classe ou une interface.

Depuis PHP 5.3.0, il est possible de référencer une classe en utilisant une variable. La valeur de la variable ne peut être un mot clé (e.g. self, parent et static).

```
class classeTest {
    const CONSTANTE = 'Je suis une constante de classe ';
}

echo $unObjet::CONSTANTE;
echo classeTest::CONSTANTE;

$vn=new classeTest();
echo $vn::CONSTANTE;
echo $unObjet->CONSTANTE;
```

[Démonstration](#) [code](#)

Erreur !!!



PHP OO : PRÉSENTATION

Egalité des objets:



Rappel :

```
$obj= new MaClasse()
```

\$obj contient une référence qui pointe sur la zone contenant les valeurs des attributs de l'objet

Avec l'opérateur ==

Deux objets seront égaux si

- ✓ Ils appartiennent à la même classe
- ✓ Ils possèdent les mêmes attributs
- ✓ Les attributs ont les mêmes valeurs

[Démonstration](#) [code](#)



PHP OO : PRÉSENTATION

Egalité des objets:

Avec l'opérateur ==

Deux objets seront égaux si

- ✓ Ils appartiennent à la même classe
- ✓ Ils possèdent les mêmes attributs
- ✓ Les attributs ont les mêmes valeurs

```
class MaClasse {  
    public $id;  
    public $nom;  
}
```

```
$id = 100;  
$nom = 'Dupont';  
$unObjet = new MaClasse();
```

```
$unObjet->id=$id;  
$unObjet->nom=$nom;
```

```
$unAutreObjet = new MaClasse();  
$unAutreObjet->id=$id;  
$unAutreObjet->nom=$nom;
```

```
echo ($unObjet==$unAutreObjet)?('Objets égaux'):( 'Objets différents');
```

Objets égaux



Les objets ont les mêmes attributs et valeurs mais pas la même référence mais sont considérés comme égaux



PHP OO : PRÉSENTATION

Egalité des objets:



Rappel :

```
$obj= new MaClasse()
```

\$obj contient une référence qui pointe sur la zone contenant les valeurs des attributs de l'objet

Avec l'opérateur ===

Deux objets seront égaux si

- ✓ ils ont la même référence !!!

[Démonstration](#) [code](#)



PHP OO : PRÉSENTATION

Egalité des objets:

Avec l'opérateur ===

Deux objets seront égaux si

- ✓ Ils ont la même référence

```
$id = 100;
$nom = 'Dupont';
$unObjet = new MaClasse();
```

```
$unObjet->id=$id;
$unObjet->nom=$nom;
```

```
$unAutreObjet = new MaClasse();
$unAutreObjet->id=$id;
$unAutreObjet->nom=$nom;
```

```
$leMemeObjet=$unObjet;
```

```
echo ($unObjet=== $unAutreObjet)?('Objets égaux'):( 'Objets différents') . '<br>';
echo ($unObjet=== $leMemeObjet)?('Objets égaux'):( 'Objets différents') . '<br>';
```

```
class MaClasse {
    public $id;
    public $nom;
}
```

Objets différents
Objets égaux



Pour que les objets soient égaux, il faut qu'ils aient la même référence



PHP OO : PRÉSENTATION

TP #1

Convertisseur de base 10 en base n

[Voir sujet](#) (fichier tp_piles.pdf)

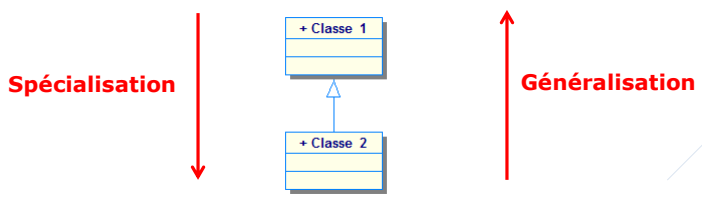


L'HÉRITAGE

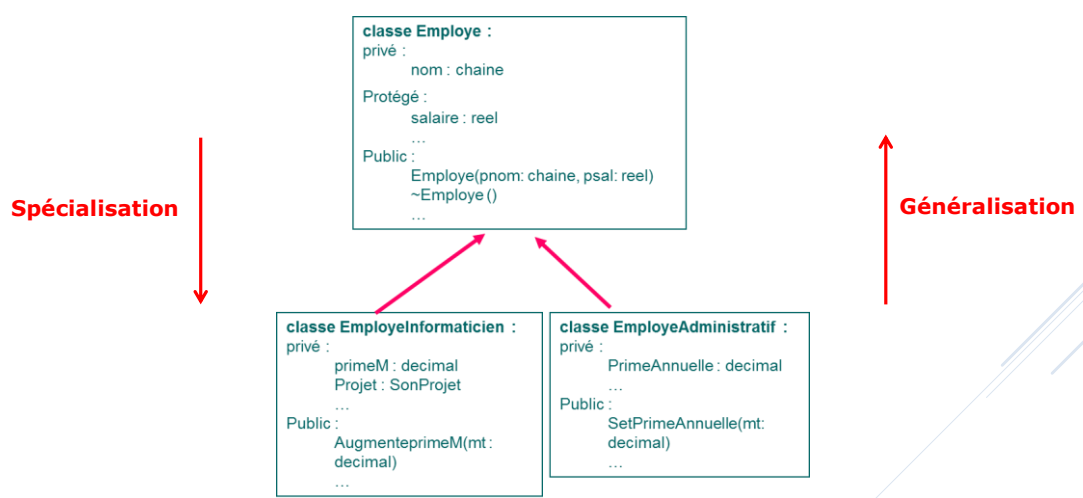


PHP OO : HÉRITAGE

- L'héritage fait partie des principes fondamentaux de la programmation orientée objet.
- ***Il permet de spécialiser une classe à partir d'une classe plus générique. On parle de classe (ou super-classe) et de sous-classe***
- Ainsi, la sous-classe hérite des propriétés et des méthodes de la classe générique.
- Le langage PHP ne supporte pas l'héritage multiple, ***c'est-à-dire qu'une classe ne peut hériter que d'une seule et unique classe.***



PHP OO : HÉRITAGE





PHP OO : HÉRITAGE

Déclaration d'une classe héritée :

- On utilisera le mot clé **extends** qui signifiera qu'une classe fille étend une classe mère

```
class Employe { ...27 lines }

class PersonnelInformaticien extends Employe {
    private $_primeM;

    public function __construct($num, $nom, $prenom, $salaire, $prime) { ...3 lines }

    public function augmenterPrimeM($montant) { ...3 lines }
}
```

Benoît Roche / @rocheb83

33



PHP OO : HÉRITAGE

Déclaration d'une classe héritée :

- Un membre de la classe fille peut accéder à :
 - ✓ Un membre protégé de la classe mère,
 - ✓ Un membre public de la classe mère
- Les classes doivent être connues avant d'être utilisées
- On utilisera en principe un mécanisme d'autochargement de classes (voir diapo auto chargement des classes)
- Les classes mères doivent être définies avant l'écriture d'un héritage.
- Cette règle générale s'applique dans le cas d'héritage ou d'implémentation d'interfaces

Benoît Roche / @rocheb83

34



PHP OO : HÉRITAGE

Accès aux membres `protected` et `public` de la classe mère depuis une classe héritée :

Il suffit d'utiliser l'opérateur `$this` :

`$this->propriétéClasseMère;`
`$this->méthodeClasseMère();`

```
class Personnage {
    protected $_nomemp;
    protected $_salaire;
    protected function coucouDeClasseMèreProtegee() {
        return "coucou";
    }
}

class personnelInformaticien extends Personnage {
    private $_primeM;

    public function __construct($nom, $salaire, $prime) {
        $this->_primeM = $prime;
        parent::__construct($nom, $salaire);
    }

    public function getGainInformaticien() {
        return $this->_primeM + $this->_salaire;
    }

    public function afficheCoucou() {
        return $this->coucouDeClasseMèreProtegee();
    }
}
```

Benoît Roche / @rocheb83

35



PHP OO : HÉRITAGE

L'héritage : Les opérateurs ***parent*** et ***self***

- L'opérateur de résolution de portée `parent` permet, à l'intérieur d'une classe, de faire référence à des membres de la classe mère (statique ou non)
- L'opérateur de résolution de portée `self` permet, à l'intérieur d'une classe, de faire référence à des membres de la classe courante (statique ou non).
- On se servira notamment de l'opérateur `parent::` pour appeler le constructeur de la classe mère

```
class personnelInformaticien extends personnel {
    private $_prime;

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

Benoît Roche / @rocheb83

36



PHP OO : HÉRITAGE

L'héritage : Les opérateurs *parent* et *self*

```
class employe {
    protected function affiche() {
        return "Je suis dans la classe Employe <br>";
    }
}

class personnelInformaticien extends employe {
    protected function affiche() {
        return "Je suis dans la classe EmployeInformaticien <br>";
    }

    public function afficheParent() {
        return parent::affiche();
    }

    public function afficheSelf() {
        return self::affiche();
    }
}
```

\$unInformaticien = new personnelInformaticien();
echo "Parent : " . \$unInformaticien->afficheParent();
echo "self : " . \$unInformaticien->afficheSelf();

Parent : Je suis dans la classe Employe
self : Je suis dans la classe EmployeInformaticien



PHP OO : HÉRITAGE

L'héritage : Quelques fonctions :

- **get_parent_class(unObjet) ou get_parent_class(uneClasse)**
Retourne le nom de la classe parente pour un objet ou pour une classe
- **is_subclass_of(unObjet, uneClasse [unBooleen=true])**
Vérifie si l'objet(ou la class) passé en paramètre a comme parent la classe passée en paramètre. Si le troisième paramètre est à false, l'autoloader ne chargera pas la classe)
- **get_declared_classes()**
Retourne un tableau contenant la liste des noms des classes déclarées dans le script courant



Le nombre de classes chargée peut être très important en fonction des extensions chargées (voir php.ini)



PHP OO : HÉRITAGE

L'héritage : Quelques fonctions :

+

class Mere { ...5 lines }

+

Class Fille extends Mere { ...5 lines }

+

Class Fils extends Mere { ...5 lines }

```
$uneFille = new Fille();
echo get_parent_class($uneFille);           echo "<br>";
echo get_parent_class("Fille");             echo "<br>";
echo "is_subclass_of(\$uneFille, 'MERE') : " . is_subclass_of($uneFille, "MERE");
echo "is_subclass_of(\$uneFille, 'FILS') : " . is_subclass_of($uneFille, "FILS");
print_r(get_declared_classes());
```



On voit ici le grand nombre de classes chargées....

[Démonstration Code](#)

```
Mere
Mere
is_subclass_of($uneFille, 'MERE')1
is_subclass_of($uneFille, 'FILS')
Array ( [0] => stdClass [1] => Exception [2] => ErrorException [3] => C
BadFunctionCallException [10] => BadMethodCallException [11] => D
RuntimeException [16] => OutOfBoundsException [17] => OverflowEx
```



LES CLASSES ABSTRAITES



PHP OO : HÉRITAGE

L'héritage : classes abstraites

Elles ne peuvent pas être instanciées

Elles doivent être héritées

Elles assurent une fonction de sécurité et d'intégrité

Elles peuvent contenir des membres :

- Privés,
- Publics,
- Protégés

....

Mais aussi des fonction abstraites



PHP OO : HÉRITAGE

L'héritage : classes abstraites

Elles ne peuvent pas être instanciées.

Elles doivent donc être héritées

Elles assurent une fonction de sécurité et d'intégrité

Elles peuvent contenir des membres :

- Privés,
- Publics,
- Protégés

On les déclare avec le mot clé **abstract**

```
abstract class employe { ...17 lines }  
  
class personnelInformaticien extends employe {  
    static private $_nbInformaticien;  
}
```



PHP OO : HÉRITAGE

L'héritage : Méthodes abstraites

- Une méthode abstraite est une méthode qui ne possède que son prototype.
- Une méthode abstraite déclarée dans une classe parent doit obligatoirement être implémentée dans les classes fille
- De plus, le degré de visibilité dans la sous classe ne doit pas être restreint par rapport au degré de visibilité dans la sur classe.
- Les méthodes abstraites assurent une fonction de sécurité et d'intégrité
- Une méthode abstraite ne peut être définie **QUE** dans une classe abstraite



Benoît Roche / @rocheb83

43



PHP OO : HÉRITAGE

L'héritage : Méthodes abstraites

```
abstract class employe {

    protected $_nomemp;
    private $_prenomemp;
    static protected $_nbPersonnage;

    // déclaration méthode abstraite
    abstract public function afficheNomPrenomAbstrait();

    class personnelInformaticien extends employe {

        static private $_nbInformaticien;

        public function afficheNomPrenomAbstrait() {
            return "avec abstraction, mon nom est : " . $this->_prenom;
        }
    }
}
```

[Code](#) [Démonstration](#)

Benoît Roche / @rocheb83

44



LES CLASSES ET MÉTHODES FINALES

Benoît Roche / @rocheb83

45



PHP OO : HÉRITAGE

L'héritage : classes Final

- Une classe déclarée **final** ne pourra plus être dérivée
- Une méthode déclarée **final** ne pourra plus être redéfinie dans une sous classe
- Une propriété ne peut être déclarée final !!!!



L'intérêt (relatif) de déclarer une classe ou une méthode final est de faire en sorte qu'un développeur ne pourra pas profiter d'une classe dérivée pour faire autre chose que ce qui a été initialement prévu. Ainsi la logique de développement sera préservée.

Pour une méthode, il s'agit d'interdire la redéfinition.

Benoît Roche / @rocheb83

46



PHP OO : HÉRITAGE

L'héritage : classes Final

```
abstract class employe {
    protected $nomemp;
    protected $_prenomemp;

    final public function afficheNomPrenom() {...3 lines }
}

final class personnelInformaticien extends employe {

    public function __construct($nom, $prenom) {...4 lines }

    // cette méthode déclenchera une erreur
    public function afficheNomPrenom() {
        return "impossible : " . $this->_prenomemp . " " . $this->nomemp;
    }

}

// cette définition de classe va déclencher une erreur
class personnelInformaticienDeveloppeur extends personnelInformaticien {...5 lines }
..
```



L'AUTOCHARGEMENT SIMPLE DES CLASSES



PHP OO : AUTOCHARGEMENT SIMPLE

Principe de l'autochargement

Le principe est simple Ne pas se soucier dans un script de gérer le chargement des classes nécessaires au script à travers l'utilisation des fonctions include/require

Comment ? A l'aide d'une fonction magique **__autoload**

Principe : lors de l'appel à une classe, si la classe n'est pas chargée, la fonction magique **__autoload** va être exécutée.

Impératif : Un fichier par classe
Nom des fichiers standardisé



Benoît Roche / @rocheb83

49



PHP OO : AUTOCHARGEMENT SIMPLE

Principe de l'autochargement

Ainsi :

La fonction **__autoload** :

- intercepte un appel à une classe non encore définie
- essaye de charger cette classe. Par exemple à travers un include de ce type :

include (\$uneClasse. '.php');

- admet un paramètre : le nom de la classe à charger !

Benoît Roche / @rocheb83

50



PHP OO : AUTOCHARGEMENT SIMPLE

Principe de l'autochargement

La fonction __autoload (inclue dans le script)

```
function __autoload($nomClasse) {  
    echo "chargement de la classe " . $nomClasse . "<br />";  
    include (CLASSES_PATH. $nomClasse . '.php');  
}
```

- Les classes sont stockées dans le sous-dossier classes sous la racine du site (constate CLASSES_PATH définie préalablement)
- Ici la classe Employe sera stockée dans le fichier Employe.php



PHP OO : AUTOCHARGEMENT SIMPLE

Principe de l'autochargement

La fonction __autoload (inclue dans le script)

```
$unInformaticien = new personnelInformaticien("Java", "Jean", 1000);  
echo $unInformaticien->afficheNomPrenom() . "<br />";  
$unDeveloppeur = new personnelInformaticienDeveloppeur("Petithommepeilu", "Robert", 300, "Php");
```

chargement de la classe personnelInformaticien
chargement de la classe personnel
Informaticien Jean Java
chargement de la classe personnelInformaticienDeveloppeur



On voit que les classes ont été chargées dans l'ordre du besoin



PHP OO : AUTOCHARGEMENT SIMPLE

Principe de l'autochargement

Depuis PHP 5.3 on peut gérer l'autochargement des classes avec une gestion des exceptions :

```
function __autoload($nomClasse) {
    $fichier = CLASSES_PATH . $nomClasse . '.php';
    echo "classe a charger : " . $nomClasse . " <br />et nom fichier : " . $fichier . "<br />";
    if (file_exists($fichier)) {
        include ($fichier);
    } else {
        throw new Exception("Impossible de charger la classe " . $nomClasse);
    }
}

try {
    $unInformaticien = new informaticien();
    echo "classe chargée";
} catch (Exception $e) {
    echo $e->getMessage();
}
```

classe a charger : informaticien
et nom fichier : \classes\informaticien.php
Impossible de charger la classe informaticien

[Code](#) [fichiersClasses](#) [Démonstration](#)



PHP OO : PRÉSENTATION

TP #2
Employe Héritage



Fin du cours, merci

Questions/Réponses

Benoît Roche
@rocheb83