

PHP POO PDO

Clevory Training



Programmation Orientée Objet avec PHP

Formateur : Dr Aymen SELLAOUTI

Maitre Assistant

Formateur et Consultant en Technologies du Web

aymen.sellaouti@gmail.com

Plan



-  • Présentation du formateur
-  • Plan de la formation
-  • Objectifs de la formation
-  • Public concerné
-  • Connaissances requises
-  • Références bibliographiques

Aymen SELLAOUTI

- Docteur en informatique
- Enseignant universitaire
- Créeur de contenu (PHP, Symfony, Angular, NestJs) et Fondateur de la chaîne Youtube Techwall (+20000 abonnés)



<https://www.youtube.com/c/TechWall>

- Formateur en Fullstack Javascript (Angular, Nest JS)
- Consultant PHP Symfony et Fullstack Javascript

Plan de formation



- Introduction
- Objectifs de la POO



- Concepts de base de la POO
- Paradigmes de la POO



- Classes et Objets en PHP
- Héritage et Polymorphisme



- Interface
- PDO

A l'issue de la formation, le participant sera en mesure de :

- Comprendre le paradigme de la programmation orientée Objet (POO)
- Raisonner selon le paradigme objet
- Maitriser la conception Orienté Objet
- Assimiler les relations entre les classes
- Maitriser l'encapsulation
- Pratiquer L'héritage
- Pratiquer le polymorphisme



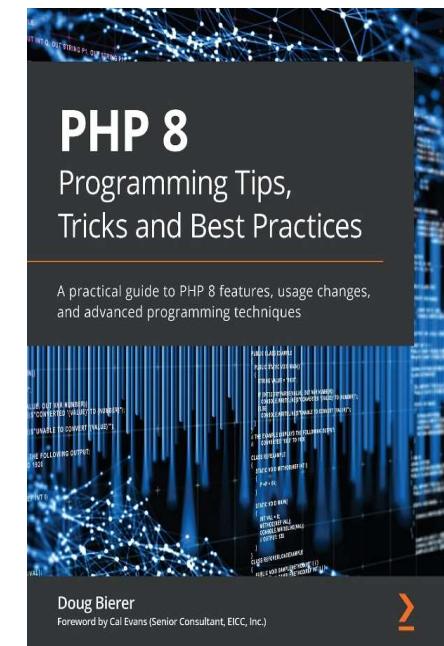
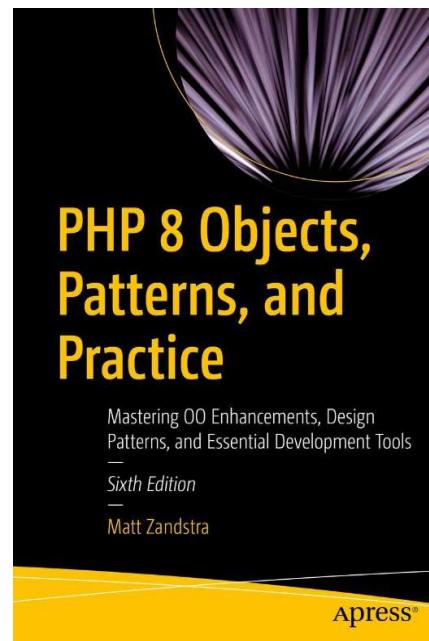
Développeurs Web



Les Bases de PHP

L'algorithmique

Conception UML



Introduction

La poo?

- Un paradigme de programmation qui consiste à la définition d'une collection d'objets qui communiquent entre eux par envoi de messages.
- Une approche naturelle de conception et de développement de logiciels.
- Représenter les éléments du monde réel par des entités informatiques dénommées « objets ».

Introduction

La poo?

Raisonnement Orienté Objet?

Programmation Orientée Objet	Programmation Procédurale
La structuration des programmes autour des données en association avec leurs traitements spécifiques	Il y a une dissociation entre les données et les fonctions: des difficultés lorsque l'on désire changer les structures de données.

Objectifs de la POO

1. Modélisation Simplifiée



2. Robustesse



3. Extension simple



4. Réutilisabilité



Objectifs de la POO

Modélisation Simplifiée

- La POO est adapté à la conception de grosses applications car elle offre une modélisation plus naturelle
- **Modélisation simplifiée** vu que le monde réel n'est pas représenté par des structures de données et des fonctions isolées (cas de la programmation classique) mais par des objets correspondants aux éléments traités du champ de l'étude.

Objectifs de la POO

Modélisation Simplifiée

Exemple: Gestion d'une bibliothèque

1. Approche procédurale :

« Que doit faire mon programme? »

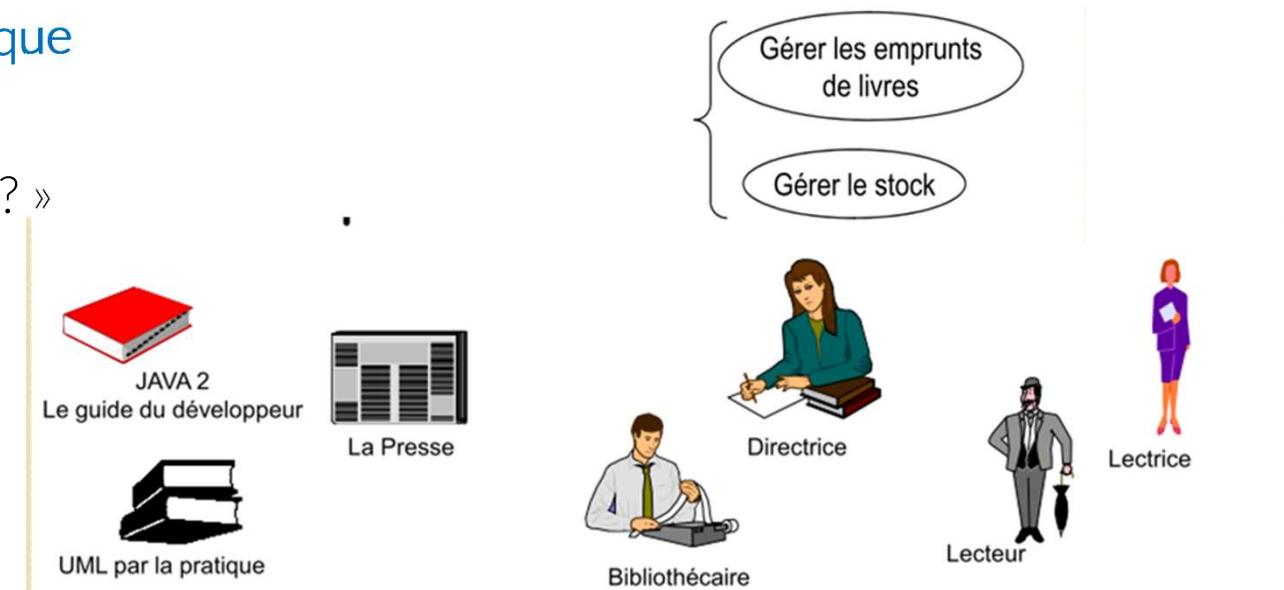


Figure: Gestion d'une bibliothèque tirée du web

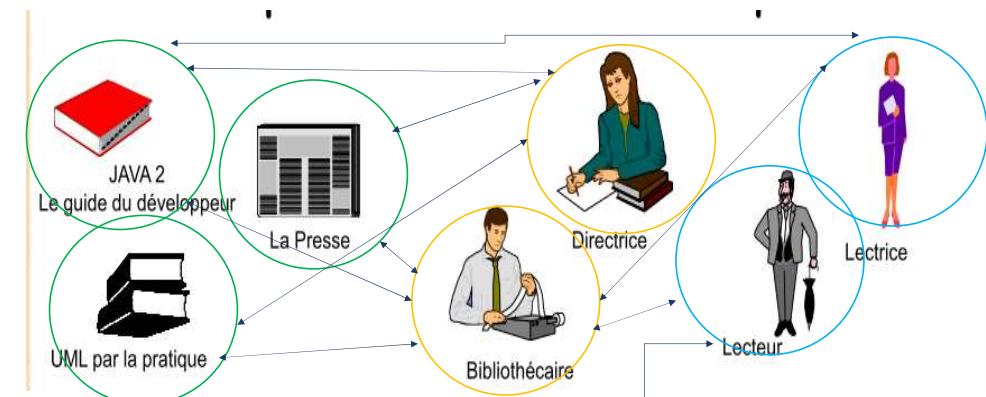
Objectifs de la POO

Modélisation Simplifiée

Exemple: Gestion d'une bibliothèque

2. Approche Orientée Objet :

« De quoi doit être composé mon programme? »



Objectifs de la POO

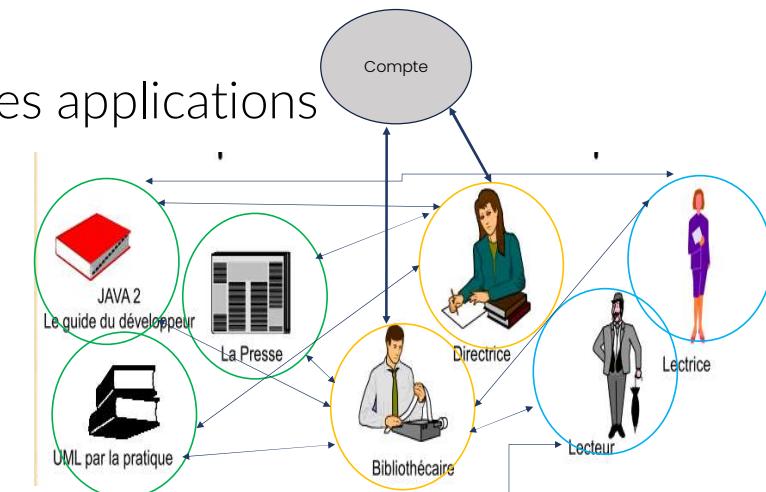
Robustesse

- **Maintenance facile** : Déetecter et cibler les bugs est plus facile dans un programme orienté objet avec la gestion des exceptions
- **Typage** : Le typage dans les classes offrent une certaine robustesse aux applications

Objectifs de la POO

Extension simple

- Un programme orienté objet est facilement **extensible**. Il suffit d'ajouter les classes d'objets dans le programme et d'établir les liens nécessaires entre les objets.
- Si on souhaite surveiller le compte bancaire des employés, on peut étendre l'application par l'objet Compte et établir les connexions adéquates avec les autres objets.
- Capacité de réutilisation totale ou partielle dans de nouvelles applications



Concepts de base de la POO

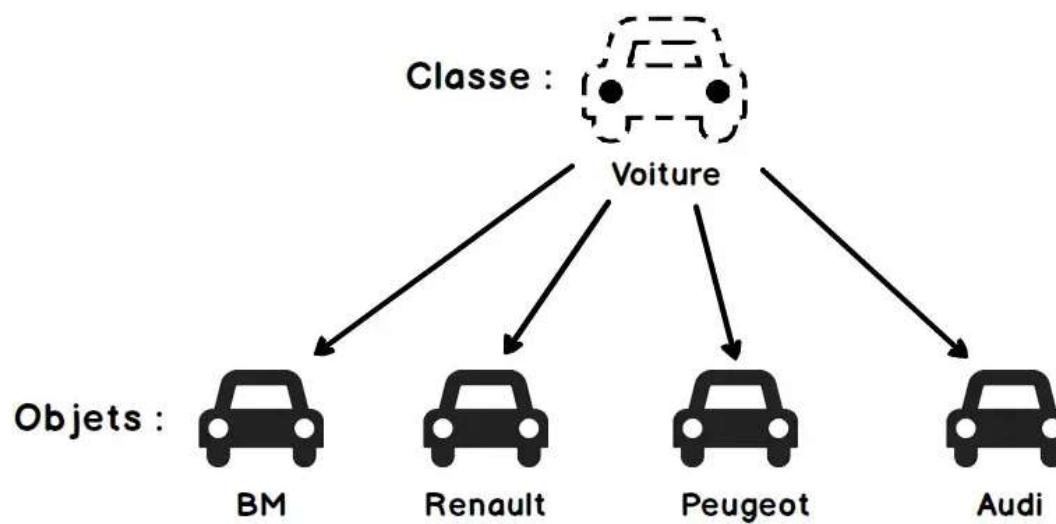
Classe

- Une classe représente un modèle de construction d'objets.
- Il s'agit d'une description abstraite en terme de données et de comportements d'une famille d'objets.
- Une classe d'objets est constituée d'une **partie statique** et d'une **partie dynamique**

Concepts de base de la POO

Classe

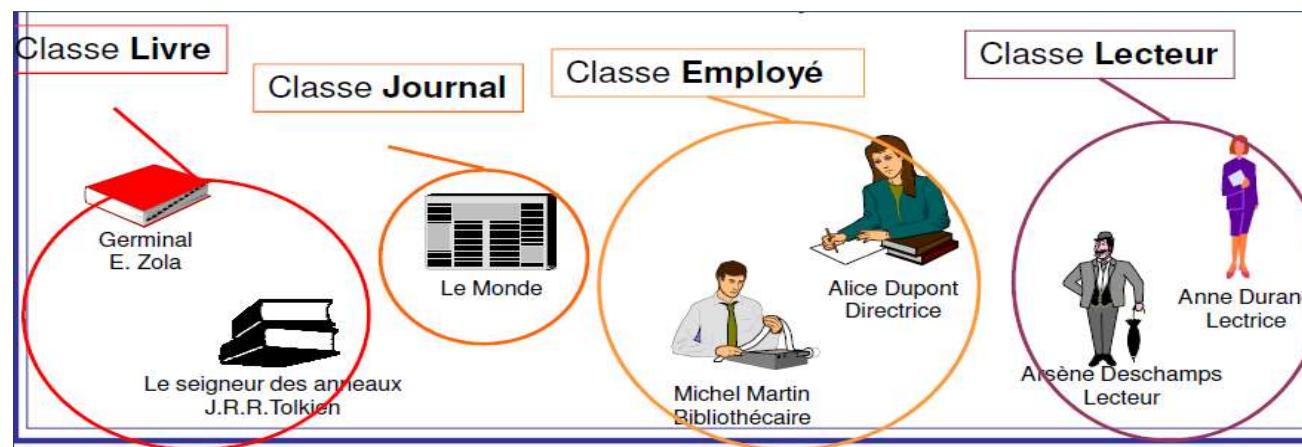
- Une classe peut être vue comme le « plan de construction » d'objets



Concepts de base de la POO

Classe

- Des objets similaires peuvent être informatiquement modélisés par une même abstraction: Une classe
 - Même **structure de données** et mêmes **traitements**
 - Valeurs **differentes** pour chaque objet



Concepts de base de la POO

Classe

- Une classe est constituée de deux parties:
- *Partie statique : LES ATTRIBUTS* : Les attributs représentent la description des données propres à chaque classe d'objets. Ceux-ci peuvent être des objets d'autres classes ou des références sur d'autres objets. Pour qualifier les attributs d'une classe, on raisonnera en termes de **propriétés** et **d'ETAT**
- *Partie dynamique : LES METHODES* : Les méthodes représentent l'ensemble des actions, procédures, fonctions ou opérations que l'on peut associer à une classe. L'ensemble des méthodes de la classe définit le "**COMPORTEMENT**"

Concepts de base de la POO

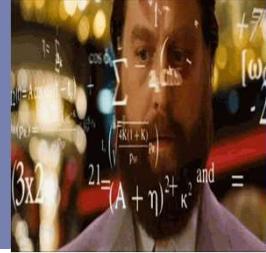
Classe



CLASSE VOITURE	
ATTRIBUTS	METHODES
Marque	Démarrer
Modèle	Accélérer
Immatriculation	Freiner
Vitesse	Stopper
Niveau de carburant	Vidanger
Puissance	Etc
Etc ...	

Concepts de base de la POO

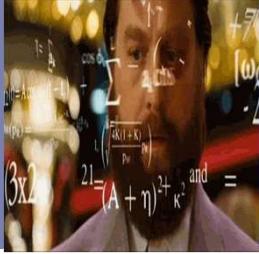
Classe



- Essayer d'extraire les classes, leurs propriétés et proposer leurs méthodes.
 - Nous souhaitons calculer la distance euclidienne entre deux points représentés sur un repère à 2D orthonormé. Pour rappel, la distance entre deux points se calcule à base des coordonnées (abscisses et ordonnées) des points considérés.
 - Un appareil de chauffage électrique possède une résistance et une tension d'alimentation. Les autres caractéristiques qui concernent son état sont : l'intensité du courant le traversant et la puissance qu'il consomme.
 - L'intensité se calcule en divisant la tension par la résistance (Loi d'Ohm $U = R I$).
 - La puissance s'obtient en multipliant la tension par l'intensité ($P = U I$)
 - Dans un parc automobile, nous souhaitons connaître le nombre de véhicules de chaque marque ainsi que leurs kilométrages. Le système utilisé enregistre pour chaque véhicule sa marque, son numéro d'immatriculation et les informations lues sur son compteur. Ce dernier est composé du kilométrage du compteur partiel ainsi que celui du compteur total.
 - La recherche d'un jeu vidéo dans une vidéothèque se fait selon sa date de sortie composée du jour, du mois et de l'année. Une fois un jeu vidéo sélectionné, le système donne accès à d'autres informations sur le jeu qui sont : son titre, sa catégorie, son type de console et sa note.

Concepts de base de la POO

Classe



- Un agenda est un ensemble de RDVs ayant une heure début (heures, minutes, secondes), heure fin (heures, minutes, secondes) et un objet.
- Nous considérons qu'un film est défini par l'ensemble de ses acteurs, sa date de sortie et sa catégorie. Pour chaque acteur, nous retenons son nom, son prénom, sa date de naissance ainsi que sa nationalité.

Concepts de base de la POO

Objet

- Une **instance de classe** est un élément construit selon le modèle de sa classe: On nomme cet élément "**OBJET**".
- Une classe est la définition d'un type.
- Un objet est une déclaration de variable.
- Après avoir crée une classe, on peut créer autant d'objets basés sur cette classe.
- Le processus de création d'un objet à partir d'une classe est appelé **instanciation** d'un objet ou création d'une **occurrence** d'une classe.
 - Les **attributs** (i.e. des variables) de la classe ont des valeurs.
 - Les **méthodes** de la classe fonctionnent sur l'objet.

Concepts de base de la POO

Objet

- Objet = Etat + Comportement + Identité
- Un objet représente une entité du monde réel qui se caractérise par une identité, des états et par un comportement
- L'identité d'un objet permet distinguer les objets les uns par rapport aux autres
- Son état correspond aux valeurs de tous les attributs à une instance donnée
- Le comportement d'un objet se définit par l'ensemble des opérations qu'il peut exécuter en réaction aux messages envoyés (Un message = demande d'exécution d'une opération) par les autres objets

Concepts de base de la POO

Objet

CLASSE VOITURE	
ATTRIBUTS	METHODES
Marque	Démarrer
Modèle	Accélérer
Immatriculation	Freiner
Vitesse	Stopper
Niveau de carburant	Vidanger
Puissance	Etc
Etc ...	

EXEMPLES D'INSTANCES DE LA CLASSE VOITURE



PEUGEOT
3008
3412 Tunis 200
Automatique
300 Km /h
Essence
5 CV, ...



CITROEN
C5
300 Tunis 149
Mécanique
300 Km /h
Diesel
8 CV, ...



Paradigmes de la POO

1. Abstraction
2. Encapsulation
3. Polymorphisme
4. Héritage
5. Composition, Agrégation

Paradigmes de la POO

Abstraction

- L'**abstraction** est le processus qui consiste à représenter des **objets qui appartiennent au monde réel dans le monde du programme** que l'on écrit. Il consiste essentiellement à extraire des **variables pertinentes**, attachées aux objets que l'on souhaite manipuler, et à **les placer dans un modèle informatique convenable**.
- Le processus d'abstraction consiste à **identifier pour un ensemble d'éléments** des caractéristiques et des **mécanismes communs** à tous les éléments
- **Description génériques** pour l'ensemble des éléments: **Se focaliser sur l'essentiel et cacher les détails**

Paradigmes de la POO

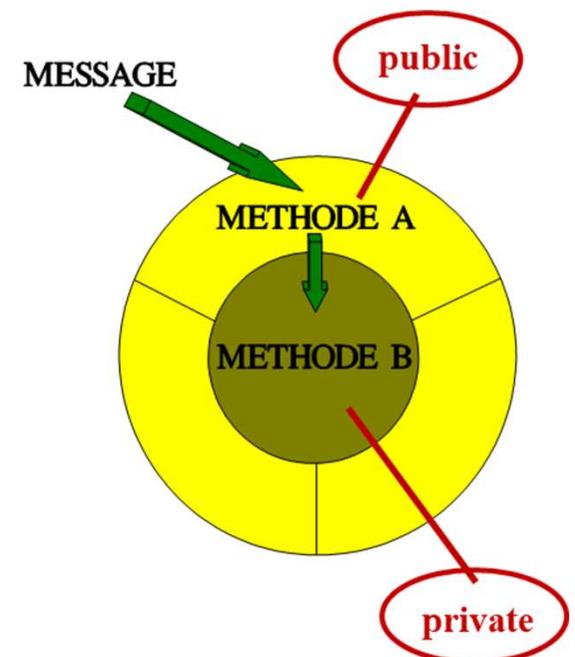
Encapsulation

- L'encapsulation est le fait qu'un objet **renferme ses propres attributs et ses méthodes**
 - Elle consiste à rendre les membres d'un objet **plus ou moins visibles** pour les autres objets.
 - La **visibilité** dépend des membres : certains membres peuvent être visibles et d'autres non.
 - La **visibilité dépend de l'observateur** : les membres de l'objet encapsulé peuvent être visibles pour certains objets mais pas pour d'autres.
- ➡ L'encapsulation a pour objectif **d'améliorer la robustesse et l'évolutivité des programmes.**

Paradigmes de la POO

Encapsulation

- Message: Débiter (somme, compte, code confidentiel).
- Objet: compte bancaire.
- Méthode A: debiterCarteCredit
 - Visible depuis l'interface
- Méthode B: Algorithme de validation du code confidentiel.
 - Non visible par l'utilisateur mais appelée par la méthode A



Concepts de base de la POO

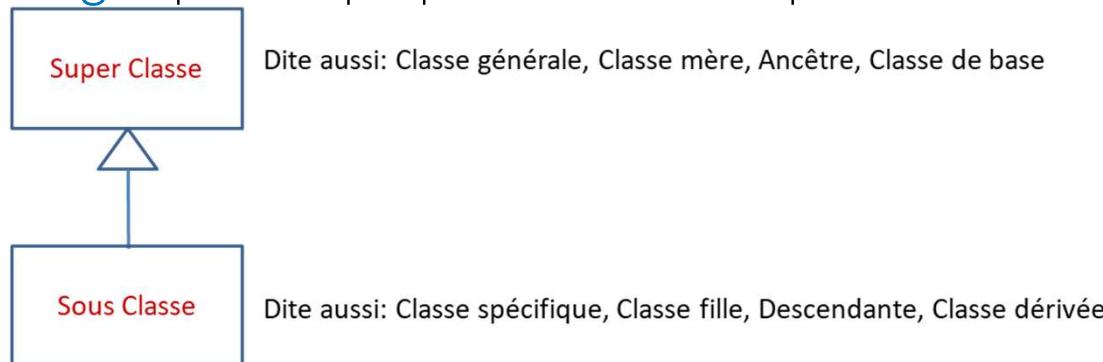
Encapsulation

- Donner des cas du quotidien qui reflètent cette notion d'encapsulation

Paradigmes de la POO

Héritage

- L'héritage est un mécanisme destiné à **exprimer les similitudes entre classes**.
- Il met en œuvre les principes de généralisation et de spécialisation en **partageant explicitement les attributs et méthodes communs** au moyen d'une hiérarchie de classes
- Une classe **fille hérite de sa classe mère ses caractéristiques** (attributs et méthodes) mais elle **se distingue par ses propres caractéristiques**.



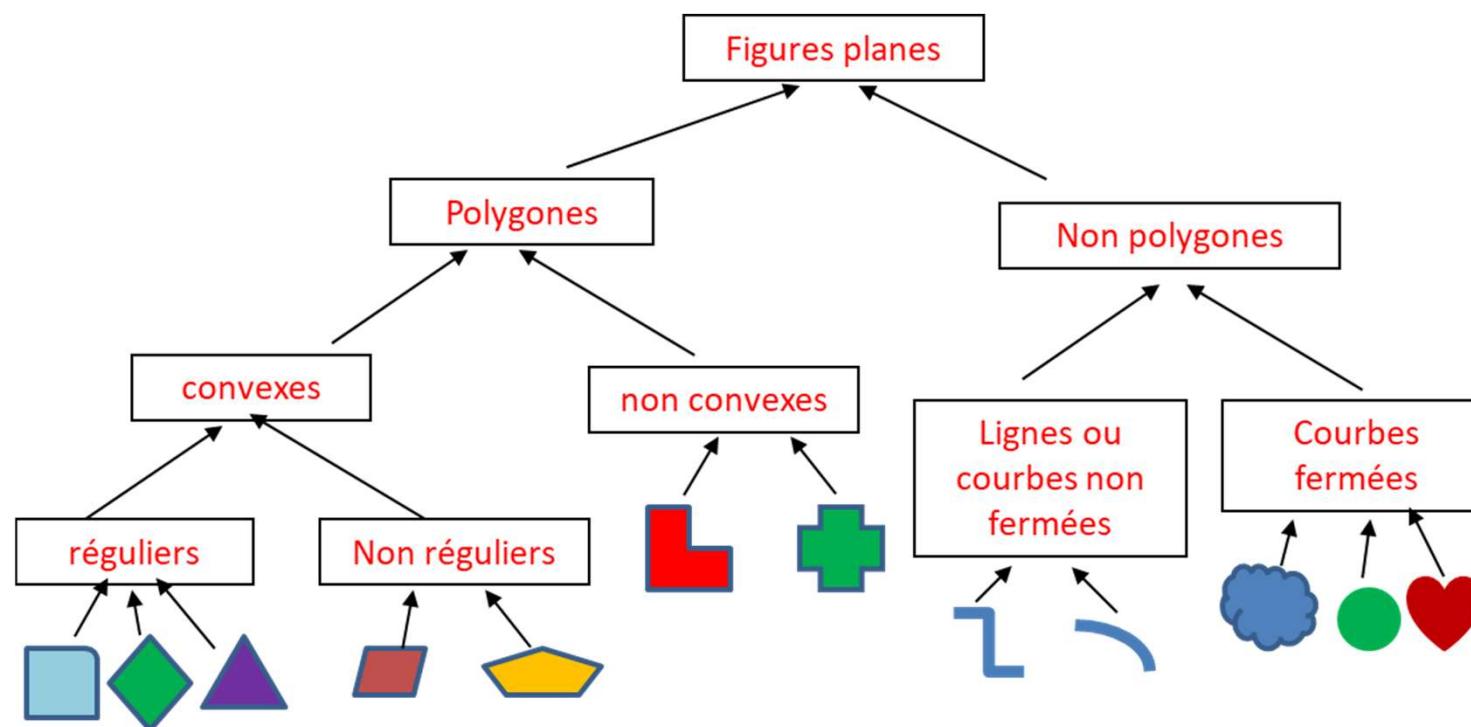
Concepts de base de la POO

Encapsulation

- Donner des cas qui reflètent cette notion d'héritage

Paradigmes de la POO

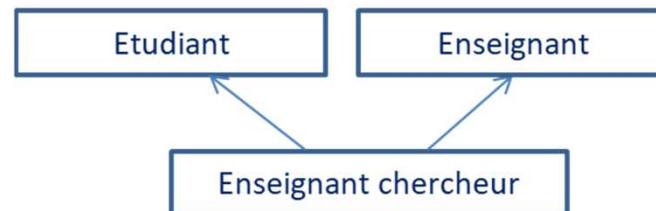
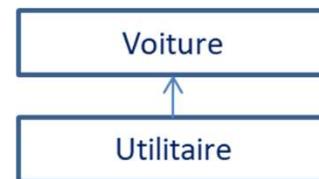
Héritage



Paradigmes de la POO

Héritage

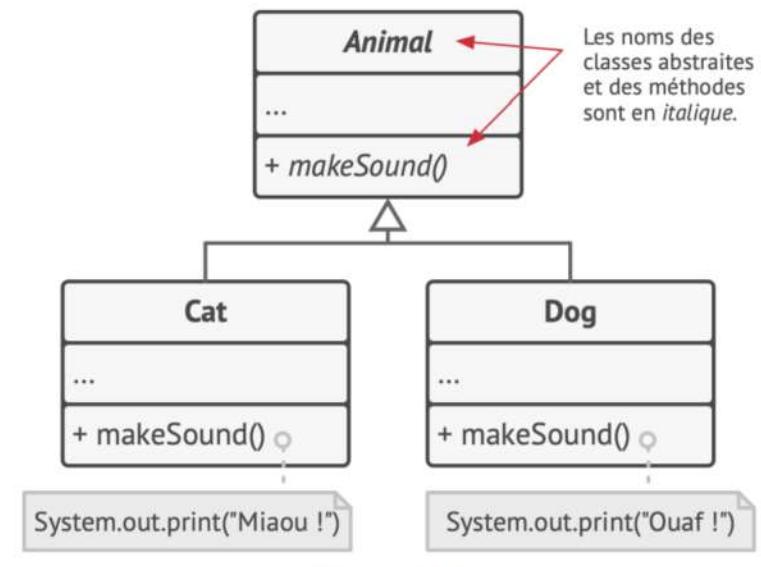
- On parle **d'héritage simple** lorsqu'une **classe fille** ne possède qu'**une classe mère**.
- On parle **d'héritage multiple** lorsqu'une **classe fille** possède **plusieurs classes mères**.



Paradigmes de la POO

Polymorphisme

- Le **polymorphisme** définit des **exécutions différentes pour une méthode commune** à une hiérarchie d'objets.
- Un code qui n'utilisera pas le polymorphisme pourrait utiliser une instruction à choix multiple suivant la classe des objets rencontrés.



Ce sont des commentaires UML. En général, ils expliquent les détails de l'implémentation des classes ou méthodes données.

Concepts de base de la POO

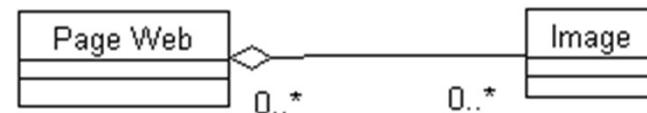
polymorphisme

- Donner des cas qui reflètent cette notion de polymorphisme

Paradigmes de la POO

Agrégation

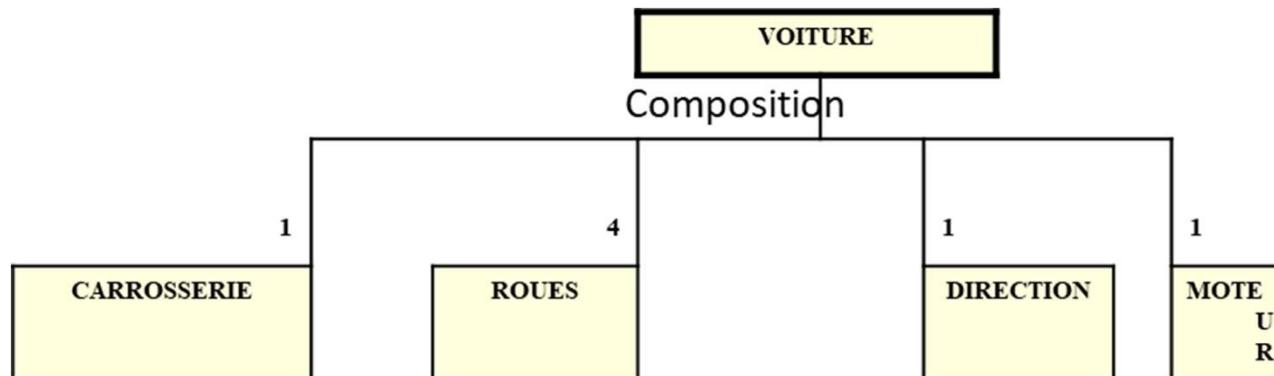
- C'est une relation particulière qui attribut à l'une des classes le rôle d'agrégat et à l'autre classe le rôle d'agrégré. L'agrégation peut être assimilée à une **appartenance faible**.
- Une page peut contenir des images mais celles-ci peuvent appartenir à d'autres pages.
- la **destruction d'une page n'entraîne pas celle de l'image** mais seulement la suppression du lien.



Paradigmes de la POO

Composition

- La composition consiste à réunir des objets pour en former un autre.
- Dans le cas d'une composition, on s'attachera à préciser le cardinal de cette relation.



Paradigmes de la POO

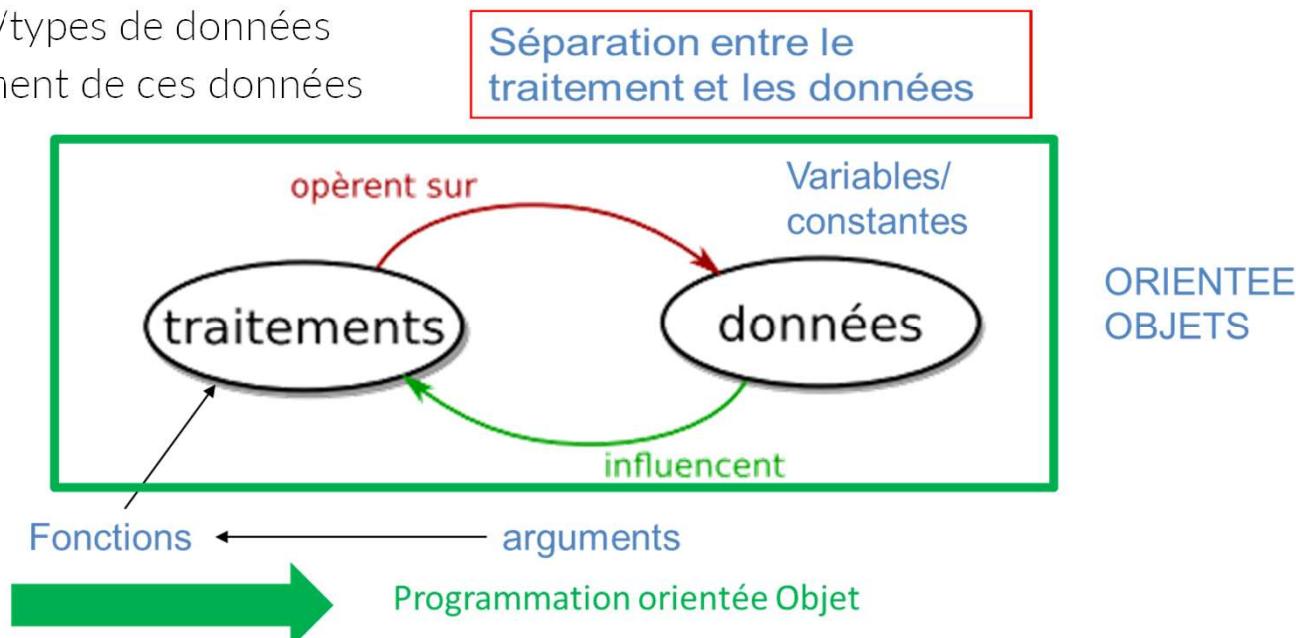
Composition Vs Agrégation

Agrégation	Composition
En agrégation, il existe une relation dans laquelle un enfant peut exister indépendamment du parent.	En composition, l'enfant ne peut pas exister indépendamment du parent.
"a un"	"partie de"
Association faible	Association forte
Représenté par un diamant creux à côté de la classe d'assemblage.	Représenté par un diamant plein à côté de la classe d'assemblage.
La suppression de l'assemblage n'affecte pas ses pièces.	Si l'objet de classe propriétaire est supprimé, cela peut affecter considérablement l'objet de classe contenant.

Paradigmes de la POO

La programmation impérative/procédurale (rappel)

- Dans les scripts que vous avez écrits jusqu'à maintenant, nous avons vu les notions
 - de variables/types de données
 - et de traitement de ces données



Classes et Objets en PHP

Définition d'une classe et d'un objet

- Une classe en orientée objet représente un regroupement d'attributs (variables) et de méthodes (fonctions).
- Un objet est une instance de classe.



Classe



Objet

Classes et Objets en PHP

La classe

- Une classe permet de **modéliser** des objets du monde Réel.
- Une classe permet de présenter des objets qui sont **conceptuellement similaires**.
- Les **objets** sont des **instances de classe**.
- Le mot **class** doit être écrit entièrement en minuscule.

Classes et Objets en PHP

La classe

- Syntaxe:

```
class <nom_de_classe>
[extends <nom_de_superclasse>] [implements <interface_1>,<interface_2,...>]
{
    <Attributs_et_Méthodes>
}
```

Classes et Objets en PHP

La classe

```
<?php
class Point
{
    public $x; //abscisse du point
    public $y; //ordonnée du point
    public function translater()
    {
        $this->x = $this->x + 4;
        $this->y = $this->y + 4;
    }
    public function afficher()
    {
        echo ("x= {$this->x} et y= {$this->y}");
    }
}
```

Attributs

méthodes

Classes et Objets en PHP

La classe

```
class NomClasse
{
    // Contenu de la classe
}
```

Déclaration d'un attribut

```
class NomClasse
{
    VisibilitéDeLaVariable $nomVariable ;
    // Contenu de la classe
}
Exemple :
class Gateau
{
    private $parfum;
}
```

Classes et Objets en PHP

La classe

Déclaration d'une méthode

```
class NomClasse
{
    VisibilitéDeLaVariable $nomVariable ;
    VisibilitéDeLaFonction function nomDeLaFonction ;
}
```

Exemple :

```
class Gateau
{
    private $_parfum;
    public function appliquerGlacage(){}
    // Contenu de la classe
}
```

Classes et Objets en PHP

La classe

- A partir de la version **7,4** vous pouvez **typer vos propriétés**
- Les propriétés typées doivent être initialisées avant d'y accéder, sinon, **une erreur sera déclenchée.**

```
class NomClasse
{
    VisibilitéDeLaVariable type $nomVariable ;
    VisibilitéDeLaFonction function nomDeLaFonction ;
```

Exemple :

```
class Gateau
{
    private string $_parfum;
    public function appliquerGlacage(){}
    // Contenu de la classe
}
```

Classes et Objets en PHP

La classe

- Vous pouvez aussi **typer les paramètres de vos méthodes ainsi que leur valeur de retour**. Ceci vous permettra d'avoir un **code plus robuste**

```
class NomClasse
{
    VisibilitéDeLaVariable type $nomVariable ;
    VisibilitéDeLaFonction function nomDeLaFonction(): typeValeurDeRetour ;
}
```

Exemple :

```
class Gateau
{
    private string $_parfum;
    public function appliquerGlacage(string $parfaum): void{}
    // Contenu de la classe
}
```

Classes et Objets en PHP

Visibilité des attributs et des méthodes

- La visibilité permet de définir de quelle manière un attribut ou une méthode d'un objet est accessible.
- Les 3 niveaux de visibilité en PHP sont :
 - **public** (comportement par défaut) : Accessible partout
 - **private** : Accessible au sein de la classe elle même
 - **Protected** : Accessible au sein de la classe elle-même, ainsi qu'aux classes qui en héritent, et à ses classes parentes.
- Les objets de mêmes types ont accès aux membres privés et protégés les uns des autres.

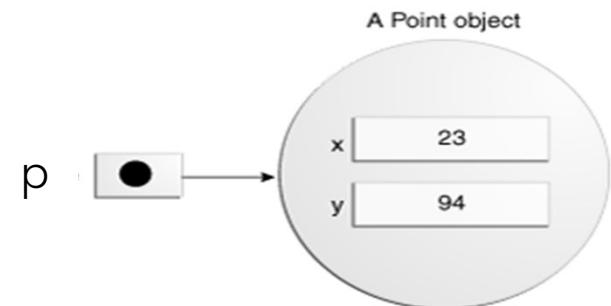
Classes et Objets en PHP

Créer un objet

- Assez de théorie maintenant qu'on a compris les concepts de bases commençons la création de notre premier objet.
- La création d'un objet à partir d'une classe est **l'instanciation**.
- L'instanciation se fait à l'aide de `new()`;
 - Exemple `$monObjet = new MaClasse();`

L'opérateur `new` instancie une classe par:

- **Allocation de mémoire** pour le nouvel objet
- **Retour d' une référence** à cette mémoire.



L'opérateur `new` invoque le constructeur de l'objet

- Pour **accéder à un propriété** de votre objet vous devez utiliser l'opérateur '`->`'
 - Exemple `$monObjet->maPropriete;`
 - Point `p = new Point(23, 94);`

Classes et Objets en PHP

Créer un objet

- Pour utiliser une classe dans un fichier et pouvoir l'instancier, il faut toujours l'appeler en utilisant le mot clé require ou include.
- Il faut indiquer à require le chemin à partir du fichier qui utilisera la classe vers la classe.
- Si les deux fichiers se trouvent dans le même dossier, il suffit d'écrire le nom du fichier.
 - require 'cheminVersMaClasse';
- Ceci permet de dire à PHP que je vais utiliser cette classe la

Classes et Objets en PHP

Comparaison des objets

- Avec PHP lorsque vous comparez deux objets avec « `==` », vous vérifiez qu'ils sont des **instances de la même classes** qui ont les **mêmes valeurs pour tous les attributs**.
- Si vous les comparez avec « `===` », vous vérifiez qu'ils ont la **même référence** et donc le même objet.

Classes et Objets en PHP

La référence `this`

- Le mot clé `this` représente une référence sur l'objet courant (celui qui est entrain d'exécuter la méthode contenant le `this`).
- La référence `this` peut être utile :
 - Lorsqu'une variable locale ou paramètre d'une méthode porte le même nom qu'un attribut de la classe.
 - Lisibilité du code

Classes et Objets en PHP

La référence this

```
class Person
{
    private string $name = "";
    private int $age = 0;

    public function estMajeur()
    {
        return $this->age >= 18;
    }
}
```

Classes et Objets en PHP

Constructeur

- A chaque instantiation d'un nouvel objet avec `new`, une méthode magique appelé `__construct()` est **automatiquement appelée**. C'est le **constructeur** de l'objet.
- Comme son nom l'indique, elle permet de **construire un objet**.
- Généralement elle permet **d'irriguer l'objet** en **initialisant ces attributs**.
- Elle peut aussi préparer l'objet autrement.
- Elle **peut prendre des paramètres** mais **ne retourne aucune valeur**.

Classes et Objets en PHP

Destructeur

- A la **fin du cycle de vie d'un objet**, une autre méthode est appelé **implicitement**, c'est la méthode **`__destruct()`** qui **ne prend pas quant à elle de paramètres** et qui permet généralement d'avoir une destruction propre de l'objet.
- Il servira par exemple à fermer un fichier ou une connexion à une base de donnée.

Classes et Objets en PHP

Constructeur par défaut

- Si aucun constructeur n'est défini, PHP offre pour chaque classe **un constructeur par défaut**, implicite et sans arguments qui instancie par défaut les attributs.
- Si vos paramètres sont **typés** vous devez les initialiser avant d'y accéder.
- Sinon vous aurez des variables vides non typées
- Si on définit un constructeur, le constructeur par défaut est écrasé

Classes et Objets en PHP

Constructeur et Destructeur

```
private $name;  
private $age;  
public function __construct( $name, $age ) {  
    $this->name = $name;  
    $this->age = $age;  
}  
  
public function __destruct() {  
    echo 'Au revoir';  
}
```

Classes et Objets en PHP

Clone

- Afin de **cloner** un objet on utilise le mot clé **clone**.
 - \$monNouvelObjet = clone \$monObjet.
- Ceci permettra d'avoir **deux objets différents** avec les mêmes propriétés.
Tester cette fonctionnalité.

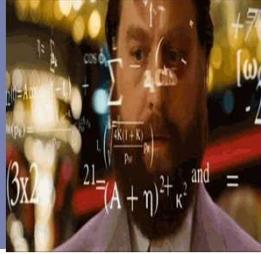
- Afin de parcourir l'ensemble des **propriétés visibles** d'un objet, nous pouvons utiliser le **foreach**.
 - Exemple :

```
foreach($this as $cle => $valeur) {  
    print "$cle => $valeur\n";  
}
```

Cet exemple permet à **chaque itération** d'avoir dans la variable \$cle le nom de l'attribut et dans la variable \$valeur sa valeur.

Classes et Objets en PHP

Classe et Objet



- Créer une classe Voiture sans constructeur qui a les attributs privés suivants :
 - une marque
 - une vitesse
 - une couleur
 - un nombre de chevaux.
- Créer un script pour tester votre classe et qui va :
 - Instancier un objet voiture.
 - Affecter lui une vitesse, une couleur. Tester votre code.
 - Que se passe t-il ?

Classes et Objets en PHP

Getters et Setters

- En affectant directement une valeur à la vitesse

➤ `$voiture->vitesse = 1000;`

- vous obtenez le message suivant :

Fatal error: Uncaught Error: Cannot access private property Voiture::\$strength in C:\xampp\htdocs\gmc\mainClass.php:12 Stack trace: #0 {main} thrown in C:\xampp\htdocs\gmc\mainClass.php on line 12

- Ce message est normal vu qu'on a mis notre attribut en **private**.
- La solution est d'utiliser les **getters** et les **setters** ou accesseurs et mutateurs.
- Les **getters** permettent de **récupérer** les valeur des attributs dont nous voulons partager la valeur. Par convention le nom d'un getter commence toujours par **get** suivi du **nom de l'attribut**. Dans notre cas, c'est **getStrength**.
- Les setters permettent quant à eux de modifier les valeurs d'un attribut tout en respectant l'encapsulation. En effet votre setter permettra de modifier la valeur en gardant vos règles métiers intactes. Pour le nom c'est **set** suivi du **nom de l'attribut**. Dans notre cas, c'est **setStrength**.

Classes et Objets en PHP

Classe et Objet

- Ajouter des getters et setters et manipuler l'objet

Classes et Objets en PHP

Classe et Objet

- Ajouter des règles dans vos setters ainsi que les propriétés nécessaires afin de Controller le fonctionnement de vos objets. Par exemple, une voiture ne peux pas avoir une vitesse de 1000km/h

Classes et Objets en PHP

Classe et Objet

- Ajouter un constructeur permettant de construire une voiture
- Tester la création.

Classes et Objets en PHP

Créer un objet

Remarque : new retourne l'identifiant de l'objet créé, donc \$monObjet ne contient pas réellement l'objet mais son identifiant.

➤ Que fait alors \$monNouvelObjet=\$monObjet ?

On aura deux variable qui font référence au même objet et non deux objets différents.

Créer une nouvelle variable et affecter à cette variable la voiture que vous avez créée.

Modifier la vitesse de cette variable puis tester si votre première voiture a gardé son ancienne vitesse ou non.

Classes et Objets en PHP

Promotion de propriétés de constructeur

- A partir de PHP8 un nouveau concept a été ajouté afin d'éviter le code redondant du constructeur, c'est la [Promotion de propriétés de constructeur](#).

```
<?php  
class Person  
{  
    private $name;  
    private $age;  
    public function __construct(  
        $name,  
        $age)  
    {  
        $this->name = $name;  
        $this->age = $age;  
    }  
}
```



```
class Person  
{  
    public function __construct(  
        private $name,  
        private $age  
    ) {}  
}
```

Classes et Objets en PHP

Classe et Objet

- Modifier votre code et tester la création.

Classes et Objets en PHP

Valeurs par défaut

- Vous pouvez aussi ajouter des valeurs par défaut

```
<?php  
class Person  
{  
    private $name;  
    private $age;  
    public function __construct(  
        $name = "",  
        $age = 0)  
    {  
        $this->name = $name;  
        $this->age = $age;  
    }  
}
```



```
class Person  
{  
    public function __construct(  
        private $name,  
        private $age  
    ) {}  
}
```

Classes et Objets en PHP

Classe et Objet

- Modifier votre code et tester la création.

Classes et Objets en PHP

Attributs nommés

- A partir de PHP8 vous pouvez utiliser les attributs nommés comme pour les fonctions.

```
<?php
class Person
{
    private $name;
    private $age;
    public function __construct(
        $name = '',
        $age = 0)
    {
        $this->name = $name;
        $this->age = $age;
    }
}
```

```
$person = new Person(name: sellaouti")
```

Classes et Objets en PHP

Classe et Objet

- Modifier votre code et tester la création.

Classes et Objets en PHP

Documenter vos classes : PHPDoc

- Lorsque vous travailler en équipe, il faut que votre code soit **lisible et compréhensible**.
- Pour cela, **il faut toujours commenter votre code**.
- Cependant, dans beaucoup de cas, ce que vos collègues ont besoin de savoir, **c'est ce que font vos classes** et **pas comment vous les avez implémentées**.
- Afin de gérer ca, vous pouvez utiliser **PHPDoc**

Classes et Objets en PHP

Documenter vos classes : PHPDoc

- PHPDoc, une convention de documentation pour PHP qui permet de documenter efficacement votre code.
- La documentation de code est importante pour **que les autres développeurs (ou vous-même à l'avenir)** comprennent **comment utiliser** vos classes, méthodes et fonctions.
- PHPDoc est un standard largement accepté pour documenter du code PHP de manière structurée.

Classes et Objets en PHP

Commentaires de documentation

- Les commentaires de documentation PHPDoc commencent par `/**` et se terminent par `*/`.
- Vous pouvez les placer au-dessus
 - d'une **classe**,
 - d'une **méthode**
 - ou d'une **fonction** pour documenter leur utilisation.

```
/**  
 * Cette classe représente un utilisateur.  
 */  
class Utilisateur {  
  
}
```

Classes et Objets en PHP

Balises PHPDoc courantes

- **@param**: Utilisée pour documenter les paramètres d'une méthode ou d'une fonction.
- Vous spécifiez le **nom du paramètre, son type et une description**.

```
/**  
 *  * Additionne deux entiers.  
 *  
 * @param int $a Le premier entier.  
 * @param int $b Le deuxième entier.  
 */  
function addition(int $a, int $b): int {  
    return $a + $b;  
}
```

Classes et Objets en PHP

Balises PHPDoc courantes

- **@return:** Utilisée pour documenter la valeur renvoyée par une méthode ou une fonction.

```
/**  
 *  * Additionne deux entiers.  
 *  
 * @param int $a Le premier entier.  
 * @param int $b Le deuxième entier.  
 * @return int  
 */  
function addition(int $a, int $b): int {  
    return $a + $b;  
}
```

Classes et Objets en PHP

Balises PHPDoc courantes

- **@var**: Utilisée pour documenter le **type** d'une **propriété** d'une **classe**.

```
/**  
 * @var string le nom de l'utilisateur  
 */  
private $name;
```

Classes et Objets en PHP

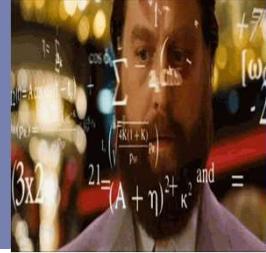
Balises PHPDoc courantes

- **@throws**: Utilisée pour documenter les exceptions levées par une méthode ou une fonction.

```
/**  
 * Additionne deux entiers.  
 *  
 * @param int $a Le premier entier.  
 * @param int $b Le deuxième entier.  
 * @return int La somme des deux nombres.  
 * @throws InvalidArgumentException Si l'un des paramètres n'est pas un entier.  
 */  
  
function addition($a, $b) {  
    if (!is_int($a) || !is_int($b)) {  
        throw new InvalidArgumentException("Les paramètres doivent être des entiers.");  
    }  
    return $a + $b;  
}
```

Concepts de base

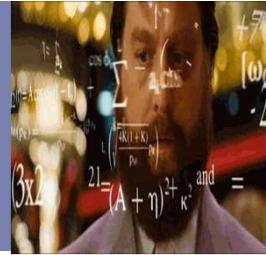
Exercice



- Créer la classe Rectangle qui contient :
 - deux attributs privés entiers width et height,
 - un constructeur qui initialise la longueur et la largeur du rectangle
 - une méthode « perimeter() »,
 - une méthode « surface() ».
- une méthode isBiggerOrEqual qui permet de voir si le rectangle est plus grand ou égale qu'un autre rectangle.
- une méthode canContain qui permet de dire si on peut mettre dans ce rectangle un autre rectangle, même en effectuant une rotation (il peut l'inclure complètement).
- Pensez à documenter votre code
- Tester votre classe

Concepts de base

Exercice



Soit un ensemble d'étudiants.

- 1) Définir une classe Etudiant qui se caractérise par un nom et un ensemble de notes.
- 2) Ajouter un constructeur qui initialise les différents attributs de la classe,
- 3) Ecrire une méthode qui affiche les notes de chaque étudiant,
- 4) Ecrire une méthode qui calcule la moyenne,
- 5) Ecrire une méthode qui affiche si l'étudiant est « admis » ou « non admis ».
- 6) Ecrire une page php permettant de calculer la moyenne et d'afficher le résultat de plusieurs étudiants tester l'exemple pour les deux étudiants suivants :
 - Aymen, 11, 13, 18, 7, 10, 13, 2, 5, 1
 - Skander, 15, 9, 8, 16

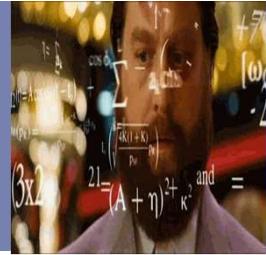
Faites en sorte que lorsque la note est <10 le background soit rouge

Faites en sorte que lorsque la note est >10 le background soit vert

Faites en sorte que lorsque la note est =10 le background soit orangé

Concepts de base

Exercice



Aymen

11

13

18

7

10

13

2

5

1

Votre moyenne est 8.888888888889

Skander

15

9

8

16

Votre moyenne est 12

Classes et Objets en PHP

Classes offertes par PHP

- Vous pouvez aussi créer des objets à partir de classes offertes par PHP
- La classe DateTime est un parfait exemple, elle vous permet de créer des objet de date et de les manipuler via les méthodes qu'elle offre

```
$today = new DateTime();  
  
echo $today->format('D, d M Y H:i:s');
```

<https://www.php.net/manual/en/class.datetime.php>

Classes et Objets en PHP

stdClass

- PHP nous offre aussi une classe générique vide avec des propriétés dynamiques.
- Les objets de cette classe peuvent être instanciés avec l'opérateur new ou créés en utilisant la conversion en objet.
- Plusieurs fonctions PHP créent également des instances de cette classe, par exemple json_decode().

```
5 $object = json_decode('{
6   "userId": 1,
7   "id": 1,
8   "title": "delectus aut autem",
9   "completed": false
10 }');
11
12 foreach ($object as $key => $value) {
13   echo "$key => $value" . PHP_EOL;
14 }
15 }
```

PROBLEMS OUTPUT TERMINAL PORTS SEARCH TERMINAL

```
aymen@DESKTOP-OS8QNV6 MINGW64 /e/php/projects/test
$ php P0O/testVoiture.php
userId => 1
id => 1
title => delectus aut autem
completed =>
```

<https://www.php.net/manual/fr/class.stdclass.php>

Concepts de bases

Introduction

- La POO nous permet de mieux organiser des programmes complexes grâce à quatre notions de bases :
- Abstraction
- Encapsulation
- Héritage
- Polymorphisme

Concepts de bases

Abstraction

- Pour être véritablement intéressant, une classe doit permettre un certain degré d'abstraction.
- Le processus d'abstraction consiste à identifier pour un ensemble d'éléments :
 - des caractéristiques communes à tous les éléments
 - des mécanismes communs à tous les éléments
- Description **générique** de l'ensemble considéré : Se focaliser sur l'essentiel, cacher les détails.
- Pour une même classe, on peut avoir différents attributs et méthodes selon le contexte dans lequel on est.

Concepts de bases

Abstraction

- Exemple : Rectangles
- la notion d'«objet rectangle» n'est intéressante que si l'on peut lui associer des propriétés et/ou mécanismes généraux (valables pour l'ensemble des rectangles)
- Les notions de largeur et hauteur sont des propriétés générales des rectangles (**attributs**),
- Le mécanisme permettant de calculer la surface d'un rectangle (**surface = largeur × hauteur**) est commun à tous les rectangles (**méthodes**)

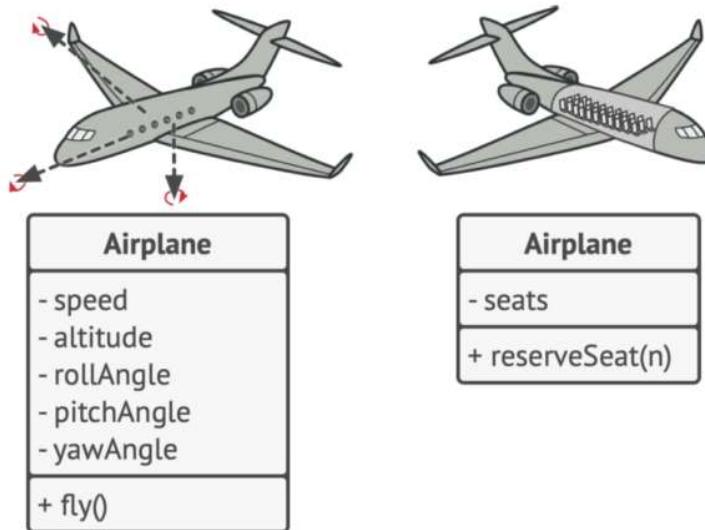
Concepts de bases

Abstraction

- Imaginez que vous ayez à implémenter deux applications.
 - Un simulateur de vol
 - Une agence de vente de billets d'avion
- Quelle abstraction avons-nous besoin d'avoir pour les deux scénarios et pour le même objet du monde réel

Concepts de bases

Abstraction



Différentes modélisations du même objet du monde réel.

L'abstraction est la modélisation d'un objet ou phénomène du monde réel, **limité à un contexte spécifique** dont on **relève tous les détails utiles** avec une grande précision, et dont on **ignore le reste**.

Plongée au Coeur des patrons de conception

Concepts de bases

Abstraction

- Prenons un exemple de la vie réelle et étudions le ensemble : Une voiture ? Posons nous ces questions :
- **Quels sont les :**
 - caractéristiques communes à toutes les voitures
 - fonctionnalités communes à tous les éléments

Concepts de bases

Encapsulation

- L'encapsulation est un principe universel et visible au quotidien
- Regarder vos téléphones, votre télécommande, votre voiture ...
- Est-ce que pour conduire votre voiture, vous avez besoin de savoir que pour faire tourner le moteur il faut :
 - Connecter des cables
 - Faire tourner les cylindres
 - Amorcer le cycle d'alimentation
 - ...

Concepts de bases

Relation entre encapsulation et abstraction

L'encapsulation permet de définir **deux niveaux** de perception des objets :

➤ **niveau externe** : partie « visible » (par les programmeurs-utilisateurs) :

➤ **l'interface** : prototypes de quelques méthodes bien choisies

➤ **niveau interne** : détails d'implémentation

➤ **corps** :

➤ méthodes et attributs accessibles uniquement depuis l'intérieur de l'objet (ou d'objets similaires)

➤ définition de toutes les méthodes de l'objet

Concepts de bases

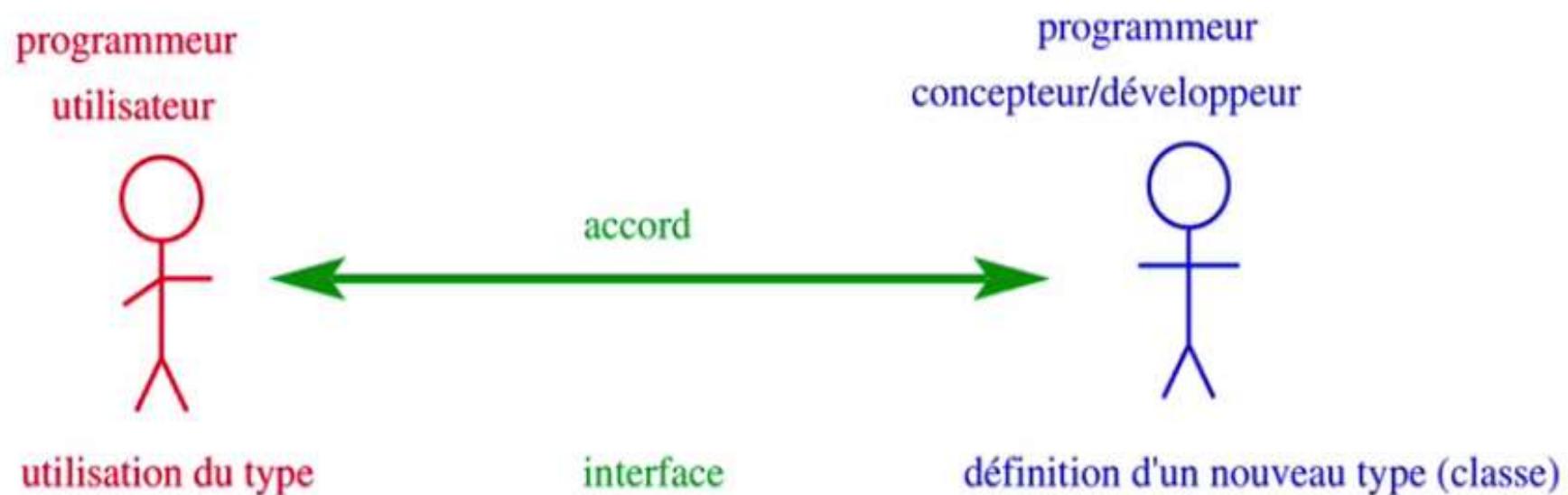
Encapsulation

Donc pour notre cas, l'interface d'une voiture vis-à-vis du conducteur :

- Volant, accélérateur, pédale de frein, etc.
- Tout ce qu'il faut savoir pour la conduire (**mais pas la réparer ! ni comprendre comment ça marche**)
- L'interface ne change pas, même si l'on change de moteur... et même si on change de voiture (dans une certaine mesure) :
- **abstraction** de la notion de voiture (en tant qu'« objet à conduire »)

Concepts de bases

Encapsulation et Interface



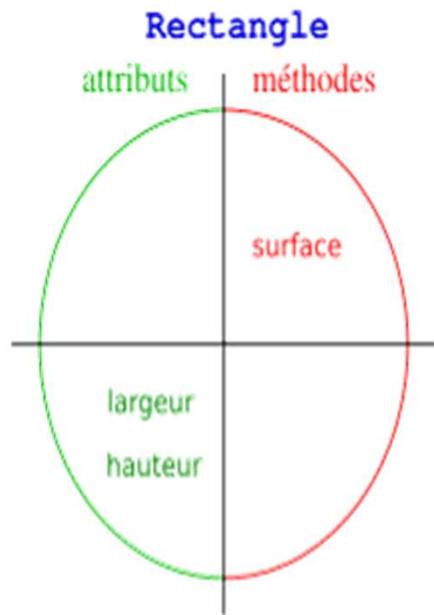
Concepts de bases

Récapitulatif

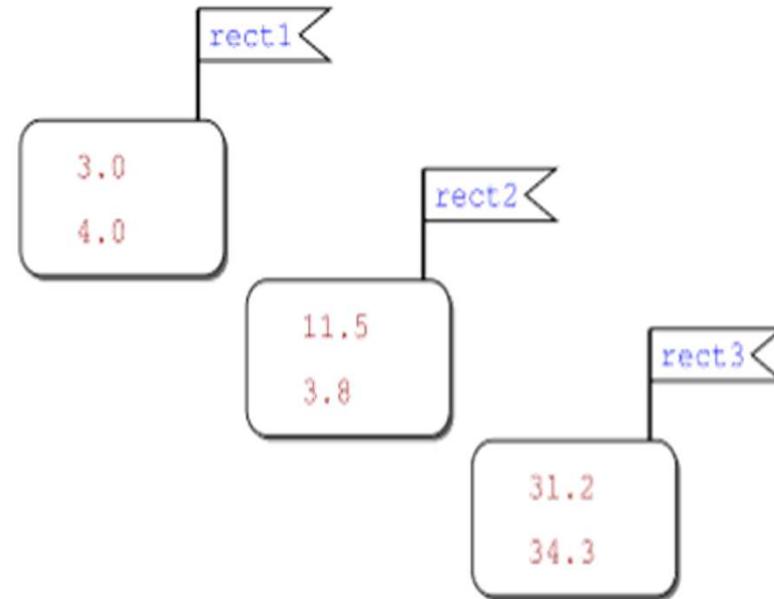
- L'intérêt de séparer les niveaux interne et externe est de donner un **cadre plus rigoureux** à l'utilisation des objets utilisés dans un programme
- Les objets ne peuvent être utilisés qu'au travers de leurs interfaces (les méthodes : niveau externe) et donc les éventuelles **modifications** de la structure interne restent **invisibles** à l'extérieur
- Une voiture ne peut pas accélérer de 1000 km d'un coup
- Règle : les attributs d'un objet ne doivent pas être accessibles depuis l'extérieur, mais uniquement par des méthodes.

Concepts de bases

Récapitulatif



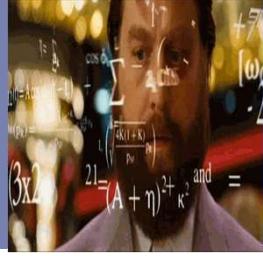
classe
type (abstraction)
existence conceptuelle
(écriture du programme)



objets/instances
variables en mémoire
existence concrète
(exécution du programme)

Concepts de base

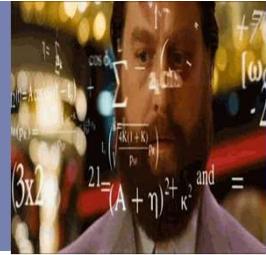
Exercice



- Ajouter les méthodes suivantes à la classe Voiture
- tableauDeBord qui affiche les informations de la voiture
- Accélérer qui permet d'augmenter la vitesse
- Freiner qui permet de freiner la vitesse
- Ajouter les propriétés nécessaires en cas de besoin sachant qu'une voiture ne peut pas accélérer à l'infini.
- Quels autres méthodes peut-on ajouter ?

Concepts de base

Exercice



Nous voulons centraliser la gestion des chaînes de caractères dans une classe.

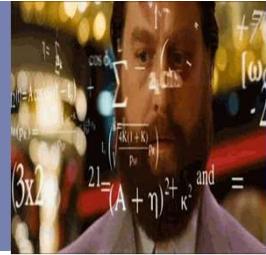
Créer la classe Chaine qui permet d'offrir les fonctionnalités suivantes:

1. Retourne votre chaîne
2. Retourner la taille de la chaîne (length)
3. Récupérer le ième caractère de la chaîne (charAt)
4. Vérifie qu'une sous chaîne est incluse dans votre chaîne à partir d'une position p et retourne la position de sa première occurrence (pensez à utiliser `stripos`) `findPosition(Chaine $chaine, int $start)`.
5. Affiche et retourne toutes les positions d'une sous chaîne dans votre chaîne.
6. Retourne une partie de votre chaîne en prenant l'indice de début et de fin et qui sont optionnels.
7. Une fonction qui supprime les espaces selon un paramètre d'option passé en argument (si 0 (valeur par défaut) enlever les espaces à droite et à gauche, si 1 les espaces à gauche sinon les espaces à droite).
8. Retourner la chaîne en minuscule
9. Retourner la chaîne en majuscule
10. Comparer avec une autre chaîne en considérant la casse.
11. Comparer avec une autre chaîne sans prendre en considération la casse.
12. Décomposer votre chaîne en un tableau en précisant le séparateur
13. Modifier votre chaîne en fusionnant les paramètres d'un tableau et un séparateur
14. Tester votre classe

<https://www.php.net/manual/fr/ref.strings.php>

Concepts de base

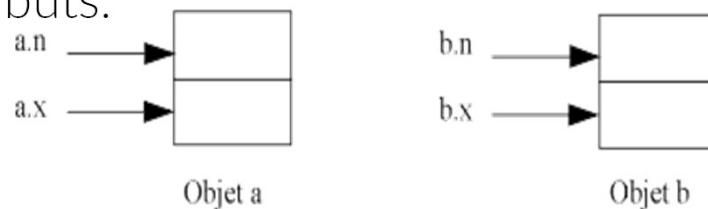
Exercice



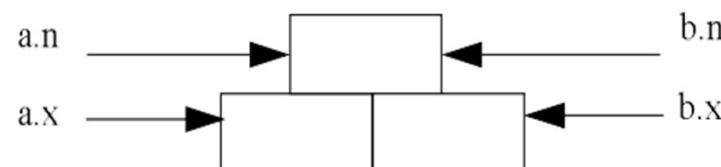
- Nous voulons modéliser une classe pour gérer nos sessions au lieu de passer par la variable `$_SESSION` directement.
- Réfléchissez aux attributs et méthodes que nous pouvons utiliser.
- Implémenter la classe
- Créer une page qui en utilisant les sessions affiche un des messages suivants :
 - Si c'est la première visite, on affiche à l'utilisateur : Bienvenu à notre plateforme.
 - Si c'est la néme visite, afficher le message suivant : Merci pour votre fidélité, c'est votre « n » éme visite.
- Ajouter un bouton dans la page permettant de réinitialiser la session.

Attributs et méthodes statiques

- Les attributs et méthodes **statiques** sont des **attributs et méthodes de classe**. Elles sont **accessibles via la classe sans** avoir besoin d'instancier un objet.
- Un attribut static est donc un attribut commun à la classe et tous ces instances. Pour les attributs standard si nous déclarons deux objets, chacun aura ses propres attributs.



- Par contre si on déclare un attribut n static, il sera alloué une seule fois. Un attribut statique est un attribut partagé par tout le monde. **UNE SEULE COPIE**.



Attributs et méthodes statiques

- Les attributs et méthodes statiques sont déclarées avec le mot clé **static**.
- Une méthode de classe ne peut utiliser que les attributs et les méthodes statiques et ne peut pas faire référence à l'objet courant (this)
- Il n'est pas nécessaire d'instancier une classe pour accéder à un des membres statiques. On peut y accéder sans création d'instance de la classe.
- Pour accéder à un attribut ou méthode statique on écrit le nom de la classe suivi de **l'opérateur de résolution de portée « :: »**.
- Dans une méthode statique, il est Interdit d'utiliser **this** !!!!!!!!!!!!! qui représente la référence de l'objet.
- « L'équivalent » de this pour la classe est **self**.

Attributs et méthodes statiques

Les cas d'utilisations des attributs et méthodes statiques sont :

➤ Gestion de configurations globales

Supposons que vous développez une application qui nécessite la gestion de configurations globales, telles que des paramètres d'API, des clés d'authentification, etc. Vous pourriez utiliser des attributs statiques pour stocker ces configurations afin qu'elles soient partagées par toutes les instances de vos classes.

➤ Comptage d'instances

Supposons que vous souhaitez suivre le nombre total d'instances créées pour une classe particulière, peut-être à des fins de statistiques ou de surveillance.

➤ Fonctions utilitaires

Supposons que vous développez une classe liée aux opérations mathématiques, et vous avez une fonction utilitaire liée à cette classe qui n'a pas besoin d'accéder à des données d'instance.

Attributs et méthodes statiques

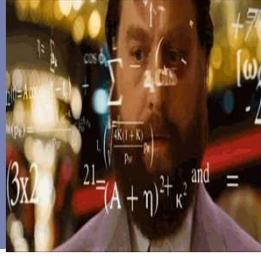
```
class MaStatic
{
    private $x;
    private static $static1=0;
    /**
     * MaStatic constructor.
     */
    public function __construct()
    {
        self::$static1++;
    }
    public static function nbInstance()
    {
        echo self::$static1;
    }
}
```

```
$ma1= new MaStatic();
$ma2= new MaStatic();
$ma3= new MaStatic();

MaStatic::nbInstance();
```

Attributs et méthodes statiques

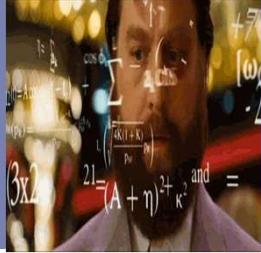
Exercice



- Créer une classe MathematiqueOperations qui offre les méthodes suivantes:
- Somme d'un nombre illimité d'arguments
- Produit d'un nombre illimité d'arguments
- Soustraction de deux réels
- Division de deux réels
- Tester votre classe

Attributs et méthodes statiques

Exercice



- Reprenez la classe Session
- Faites en sorte d'ajouter une propriété permettant de connaitre à tout moment le nombre de session active.
- Afficher le dans la page
- Que remarquer vous ? Qui peut expliquer ce qui se passe ?

Les constantes de classe

- Les **constantes de classes** sont des attributs constants et qui appartiennent à la classe et non à l'objet.
- Définit avec le mot clé **const**.
- Accessible via **l'opérateur de résolution de portée « :: »**.
- Elles sont allouées une fois par classe et non pour chaque instance
- Permettent d'éviter des **données muettes**.
- Leur visibilité par défaut est publique
- A partir de PHP 7.1.0, les modificateurs de visibilités sont autorisés pour les constantes
- Par convention, le nommage d'une constante se fait avec des majuscules.
- Ne peuvent contenir que des valeurs primitives (pas d'objet)
- Exemple **const PARIS=«Aéroport Charles de Gaulle»;**

Readonly

- À partir de PHP 8.1.0, une propriété peut être déclarée avec le modificateur **readonly**, qui empêche la modification de la propriété après l'initialisation.
- Une propriété en lecture seule **ne peut être initialisée qu'une seule fois**. Tout autre affectation ou modification de la propriété résultera en une exception
- Créer une **propriété en lecture seule** avec une **valeur par défaut** n'est pas autorisé. Ceci revient à créer une constante

```
class Person
{
    public function __construct(
        private $name,
        private $age,
        private readonly string $eyeColor
    ) {
    }
}
```

Attributs et méthodes statiques

Exercice

- Donner des exemples d'utilisation de readonly

Conventions de nommage

- Nom de la **classe** DOIT être déclaré comme **StudyCaps**. Le nom de la classe doit être **descriptif**. Il est conseillé **d'éviter les abréviations** tel que Pers pour parler d'une personne.
- Pour les **noms composés** utiliser le séparateur « _ » et les **majuscules** pour passer d'un nom à un autre ou uniquement les majuscules selon la norme que vous suivez.
- Les **méthodes et les attributs** doivent être nommées en utilisant le **camelCase**, \$parfumGlacage, \$getParfum(), ceci s'applique pour la **visibilité non privée**.
- Ceci n'est pas toujours utilisé, pour certaines conventions les attributs avec une **visibilité privée** doivent avoir un nom commençant par « _ ». Exemple : **\$_parfum**.
- Les **constantes** doivent avoir un nom qui est en totalité en **Majuscule** avec « _ » en séparateur, par exemple **MA_CONSTANTE**.

Classes et Objets en PHP

Les fonctions de Classe et Objet

- `__autoload` – Tente de charger une classe indéfinie
- `class_alias` – Crée un alias de classe
- `class_exists` – Vérifie si une classe a été définie
- `enum_exists` – Vérifie si l'énumération est définie
- `get_called_class` – Le nom de la classe en "Late Static Binding"
- `get_class_methods` – Retourne les noms des méthodes d'une classe
- `get_class_vars` – Retourne les valeurs par défaut des propriétés d'une classe
- `get_class` – Retourne le nom de la classe d'un objet
- `get_declared_classes` – Liste toutes les classes définies dans PHP
- `get_declared_interfaces` – Retourne un tableau avec toutes les interfaces déclarées
- `get_declared_traits` – Retourne un tableau contenant tous les traits déclarés

<https://www.php.net/manual/fr/ref.classobj.php>

Classes et Objets en PHP

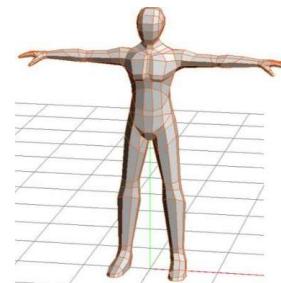
Les fonctions de Classe et Objet

- `get_mangled_object_vars` – Retourne un tableau de propriétés de l'objet manipulé
- `get_object_vars` – Retourne les propriétés d'un objet
- `get_parent_class` – Retourne le nom de la classe parente d'un objet
- `interface_exists` – Vérifie si une interface a été définie
- `is_a` – Vérifie si l'objet est d'un certain type ou sous-type.
- `is_subclass_of` – Détermine si un objet est une sous-classe d'une classe donnée ou l'implémente
- `method_exists` – Vérifie si la méthode existe pour une classe
- `property_exists` – Vérifie si un objet ou une classe possède une propriété
- `trait_exists` – Vérifie si un trait existe
- `instanceof` – Vérifie si un objet est une instance d'une classe : \$obj `instanceOf` MaClasse

<https://www.php.net/manual/fr/ref.classobj.php>

Héritage

Commençons par un exemple concret : Les personnages de jeux vidéo



Héritage

Class Guerrier

string nom
int energie
int duree_vie

Arme arme

Rencontrer(Personnage)

Class Voleur

string nom
int energie
int duree_vie

Rencontrer(Personnage)

Voler(Personnage)

Class Magicien

string nom
int energie
int duree_vie

Baguette baguette

Rencontrer(Personnage)

Class Sorcier

string nom
int energie
int duree_vie

Baguette baguette

Baton baton

Rencontrer(Personnage)

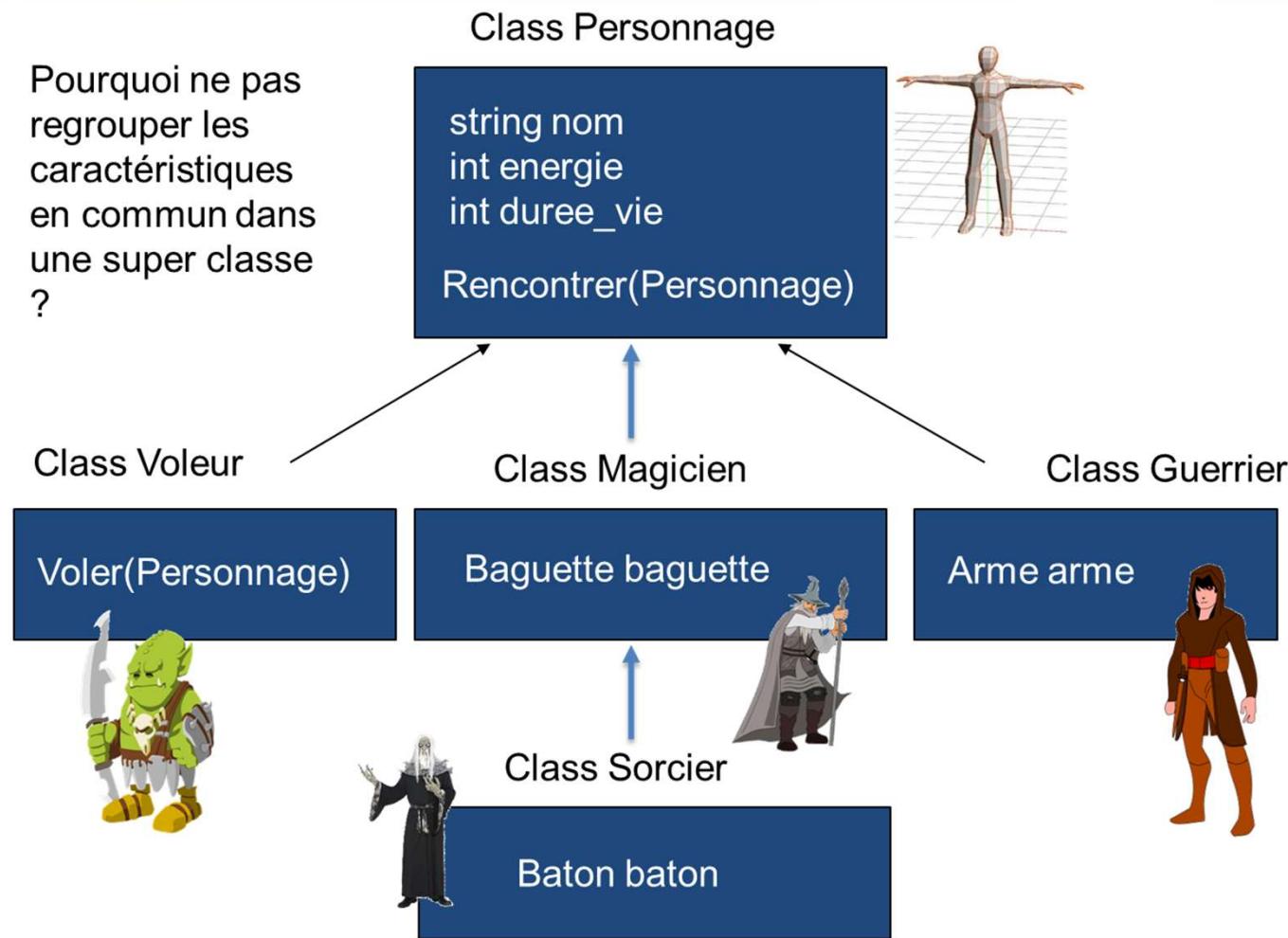
PROBLEMES ?

- ✓ Duplication de codes
- ✓ Problèmes de maintenance : Supposons qu'on veuille changer le nom ou le type d'un attribut, il faudra le faire pour chacune des classes !!!!

Solution : Héritage

Héritage

Pourquoi ne pas regrouper les caractéristiques en commun dans une super classe ?



Héritage

- Le concept d'héritage consiste à réutiliser ce qui a été défini pour une classe (la super-classe) par une autre classe (la classe dérivée) tout en le spécialisant.
- Une sous classe **n'accède pas** aux membres **privés** de la super-classe.
- Pour que les variables et les méthodes de la classe mère soient accessibles uniquement aux classes dérivées, on utilise le modificateur **protected**

Héritage

- En PHP pour dire qu'une classe **A hérite d'une classe B** on utilise le mot clé **extends**.
- La classe fille héritera de toutes les méthodes et les attributs de la classe mère mais elle ne pourra accéder qu'aux attributs **public** et **protected**.
- Une classe fille **peut redéfinir les méthodes de la classe mère**.
- Pour accéder à une méthode redéfinie de la classe mère on utilise l'objet **parent** et **l'opérateur de portée :: parent::**

Héritage

- Quand vous redéfinissez une méthode de la classe mère, vous serez amené à faire la réflexion suivante :
 - Dois je tout refaire ?
 - Dois je me baser sur l'existant et ajouter ma touche personnelle ?
- Si on veut se baser sur l'existant, on utilise la référence parent et l'opérateur de portée :: parent:: et on appelle la méthode de la classe fille.

```
class Voiture extends Vehicule
{
    public function tableauDeBord()
    {
        echo "J'affiche les infos hérités de ma classe mère Vehicule :)";
        parent::tableauDeBord();
        echo "Je définis mes propres infos :)";
    }

    public function acceler()
    {
        echo "Je m'émancipe complètement et je redéfini toute la logique";
        $this->vitesse += 20;
    }
}
```

Héritage

Héritage et constructeur

- Lorsque une classe A hérite d'une classe B, la classe A est construite en se basant sur la classe B.
- Il est donc de la responsabilité de la classe A de fournir ce qu'il faut pour la construction de la classe B.
- Pour se faire, dans le constructeur de la classe A appelé le constructeur de la classe B avec le mot clé parent et passé y les paramètres nécessaires.

Héritage

Héritage et constructeur

The screenshot shows three code editor panes illustrating PHP inheritance and constructor usage.

Left Pane: A file named `Person.php` containing the following code:

```
<?php
class Person
{
    public function __construct(
        private $name,
        private $age)
    {
    }
}
```

Middle Pane: A file named `Humain.php` containing the following code:

```
1 <?php
2 include_once 'Person.php';
3 class Humain extends Person
4 {
5 }
```

Right Pane: A file named `humain.php` containing the following code, with a red box highlighting the error message and the problematic line:

```
1 <?php
2
3 include 'humain.php';
4
5 $humain = new Humain();
```

Red Box Content (Error Message):

```
Missing argument 1..2 for
__construct() PHP(PHP0423)
```

Bottom Right Area: A code completion dropdown or suggestion list showing:

- `class Humain extends Person`
- `function Person::__construct(mixed $name, mixed $age)`
- `<?php`
- `class Humain extends Person`
- Humain** (bolded)
- `<?php`
- `class Humain extends Person { }`

Héritage

Héritage et constructeur

```
class Humain extends Person
{
    public function __construct($name, $age)
    {
        parent::__construct($name, $age);
    }
}
```

Héritage

Transitivité de l'héritage

- Dans une hiérarchie de classes, la sous-classe hérite de la super-classe :
 - tous les attributs/méthodes (sauf constructeurs et destructeur)
 - le type : on peut affecter un objet de type sous-classe à une variable de type super-classe :

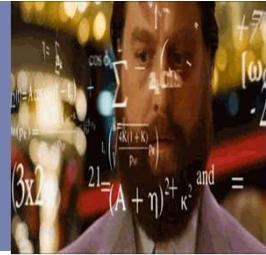


```
Personnage p;  
Guerrier g;  
// ...  
p = g;
```

- L'héritage est transitif : un Sorcier est un Magicien qui est un Personnage

Héritage

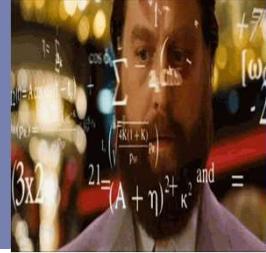
Exercice



- Soient les classes suivantes :
- Une classe **Personne** qui comporte trois champs privés : nom, prénom et date de naissance. Cette classe comporte un **constructeur** pour permettre d'initialiser les données. Elle comporte également une méthode **Afficher** pour afficher les données de chaque personne.
- Une classe **Employé** qui **dérive** de la classe **Personne**, avec en plus un champ **salaire**, un **constructeur** et la redéfinition de la méthode **Afficher**.
- Une classe **Chef** qui **dérive** de la classe **Employé**, avec en plus un champ **service**, un **constructeur** et la redéfinition de la méthode **Afficher**.
- Une classe **Directeur** qui **dérive** de la classe **Chef**, avec en plus un champ **societe**, un **constructeur** et la redéfinition de la méthode **Afficher**.
- Implémenter les 4 classes précédentes.

Héritage

Exercice (1)



- Comptes bancaires Au sein d'une banque, chaque client pourra avoir un compte bancaire qui peut être un compte épargne ou un compte courant.
- La classe compte bancaire comporte les données membres protégées suivantes : id du compte qui ne doit pas changer et qui est une chaîne unique, le solde et les opérations suivantes :
 - déposer de l'argent dans le compte,
 - retirer de l'argent à partir d'un compte
 - virement
 - afficher les caractéristiques d'un compte.
- Nous pouvons avoir deux types de comptes :
 - CompteEpargne
 - CompteCourant
- Au sein d'un **compte épargne**, la valeur minimale du solde est de 5 et contient un taux d'intérêt annuel et une fonction permettant de calculer l'intérêt annuel.
- La banque ajoute un intérêt dès la création du compte à la demande du client
- Un intérêt ne peut être réclamé qu'une année après la dernière réclamation.
- Pour un **compte courant**, son solde est limité à un seuil min égal par défaut à -500.
- Pour un compte courant, une commission de 1% est facturée pour chaque dépôt d'un montant.
- Définir les trois classes et tester le fonctionnement.
- Redéfinissez les fonctions nécessaires

Héritage

Exercice (2)

Définissez la classe **Client** qui est composée :

Attributs: *ID* : est l'identifiant unique du client (Il ne change pas)

nom : le nom du client

prenom : le prénom du client.

dateDeNaissance : la date de naissance du client

compte : chaque client a le droit à un seul compte bancaire.

Méthodes: *deposer (montant)* : dépose le montant dans le compte bancaire

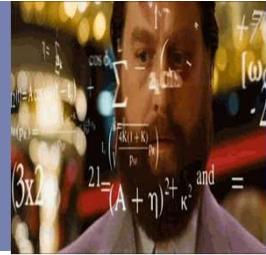
retirer (montant) : retire le montant du compte bancaire.

Ajouter l'ensemble des méthodes indiquées, en modifiant au besoin la classe **CompteBancaire**. Ajouter le constructeur de la classe **Client**.

Ajouter à la classe une méthode qui affiche toutes les informations d'un client.

Héritage

Exercice (1)



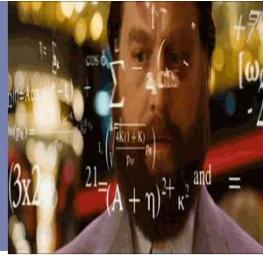
Nous allons utiliser l'exemple des Pokémons pour illustrer l'héritage.

Créer une classe Pokemon qui contient :

- un attribut name qui représente le nom du Pokéémon.
- un attribut url qui représente l'url de l'image du Pokéémon.
- un attribut hp (pour Health Points) qui représente les points de vie du Pokéémon.
- un attribut qui s'appelle attackPokemon qui représente un objet de la classe AttackPokemon qui est constitué de 4 attributs :
 1. attackMinimal qui représente le nombre de points minimal infligé par le pokemon lorsqu'il attaque
 2. attackMaximal qui représente le nombre de points maximal infligé par le pokemon lorsqu'il attaque
 3. specialAttack qui représente le coefficient de l'attaque spéciale du pockémon. Par exemple, si la valeur est de deux donc la force de son attaque sera multiplié par 2
 4. probabilitySpecialAttack qui représente la probabilité sur 100 d'avoir une attaque spéciale. Si la valeur est de 50, le pokemon aura une chance sur deux lors de l'attaque d'un autre pokemon d'effectuer une attaque spéciale

Héritage

Exercice (2)



- un constructeur pour instancier des Pokémons.
- les getters et les setters.
- une méthode isDead() qui retourne un boolean pour indiquer si un Pokémon est mort ($hp \leq 0$) ou non.
- une méthode attack(Pokemon p) qui permet au Pokémon appelant d'attaquer un Pokémon

L'attaque déduit en cas normal atk points de la vie hp du Pokémon attaqué p. En cas d'attaque spéciale, elle déduit specialAttack *attackPoint.

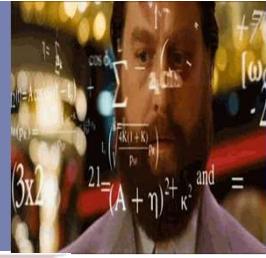
- une méthode whoAmI qui affiche les informations du Pokémon.

Créer un script qui affiche le scénario de combat entre deux Pokémon. Un combat consiste à une succession de coup entre les deux Pokémon (chacun un coup), jusqu'à la mort d'un des Pokémons.

Le pokemon vainqueur est celui qui a le plus grand niveau de vie qui a gagné

Héritage

Exercice (2)



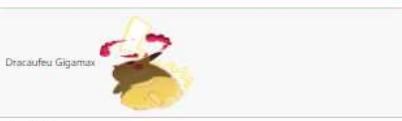
Les combattants



Points : 200
Min Attack Points : 10
Max Attack Points : 100
Special Attack : 2
Probability Special Attack : 20

Round 1

48 44



Points : 200
Min Attack Points : 30
Max Attack Points : 80
Special Attack : 4
Probability Special Attack : 20

Round 1

48 44



Points : 156
Min Attack Points : 10
Max Attack Points : 100
Special Attack : 2
Probability Special Attack : 20

Round 2

70 52



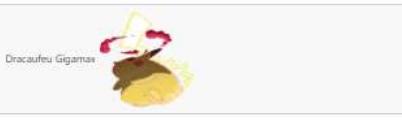
Points : 152
Min Attack Points : 30
Max Attack Points : 80
Special Attack : 4
Probability Special Attack : 20

Round 2

70 52



Points : 104
Min Attack Points : 10
Max Attack Points : 100
Special Attack : 2
Probability Special Attack : 20

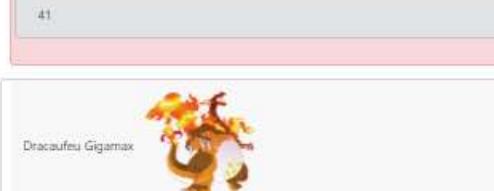


Points : 82
Min Attack Points : 30
Max Attack Points : 80
Special Attack : 4
Probability Special Attack : 20

Round 3

41 36

Round 3



Points : 68
Min Attack Points : 10
Max Attack Points : 100
Special Attack : 2
Probability Special Attack : 20

Round 4

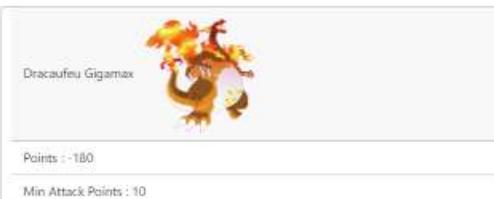
48 248



Points : 41
Min Attack Points : 30
Max Attack Points : 80
Special Attack : 4
Probability Special Attack : 20



Points : -7
Min Attack Points : 30
Max Attack Points : 80
Special Attack : 4
Probability Special Attack : 20



Points : -180
Min Attack Points : 10
Max Attack Points : 100
Special Attack : 2
Probability Special Attack : 20



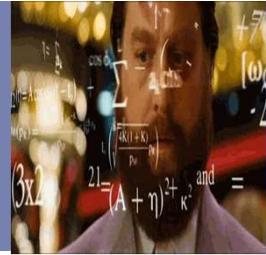
Héritage

Classe Abstraite

- Afin de forcer la classe à n'être qu'un modèle on la transforme en classe **abstraite**. Une classe abstraite est donc une classe **non instanciable**. Elle ne sert qu'à être une classe mère. On ne peut pas créer une voiture sans marque dans notre exemple.
- Une classe abstraite peut contenir des méthodes « normales » et des **méthodes abstraites**.
- Une **méthode abstraite** est une méthode non définie dans une **classe abstraite**. Elle sert à obliger toute classe héritant de la classe mère de redéfinir cette fonction.

Héritage et Classe abstraite

Exercice (1)



Définir une classe Abstraite Vehicule qui a pour attributs des informations valables pour tout type de véhicule :

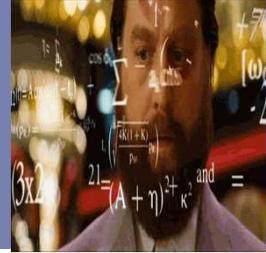
- sa marque ;
- sa date d'achat ;
- son prix d'achat ;

2) Définissez un **constructeur** prenant en paramètre les trois attributs correspondant à la marque, la date d'achat et le prix d'achat. Faites le nécessaire pour avoir une méthode **calculPrixVente**. Le prix courant sera calculé plus tard puisqu'il est spécifique et qu'il doit être disponible pour chaque classe Fille.

3) Définissez une méthode publique void **tableauDeBord** qui affiche l'état de l'instance, c'est-à-dire la valeur de ses attributs.

Héritage

Exercice (2)



4) Définissez deux classes Voiture et Avion, héritant de la classe Vehicule et ayant les attributs supplémentaires suivants :

pour la classe Voiture :

- sa cylindrée ;
- son nombre de portes ;
- sa puissance ;
- son kilométrage.

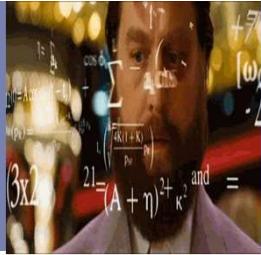
pour la classe Avion :

- son type (hélices ou réaction) ;
- son nombre d'heures de vol.

5) Définissez, pour chacune de ces classes, un **constructeur** permettant l'initialisation explicite de l'ensemble des attributs, ainsi qu'une méthode **tableauDeBord** affichant la valeur des attributs.

Héritage

Exercice (3)



6) Implémentez la méthode `calculPrixVente` cette méthode dans les deux sous-classes `Voiture` et `Avion` de sorte à calculer le prix courant en fonction de certains critères :

Pour une `voiture`, le prix courant est égal au prix d'achat, moins :

- 5% pour chaque tranche de 10000km parcourus (on arrondit à la tranche la plus proche)
- 10% s'il s'agit d'un véhicule de marque "Renault" ou "Fiat" (ou d'autres marques de votre choix)

et `plus` 20% s'il s'agit d'un véhicule de marque "Ferrari" ou "Porsche" (idem).

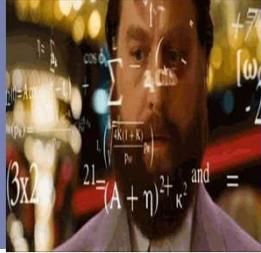
• Pour un `avion`, le prix courant est égal au prix d'achat, moins :

- 10 % pour chaque tranche de 1000 heures de vol s'il s'agit d'un avion à réaction.
- 10 % pour chaque tranche de 100 heures de vol s'il s'agit d'un avion à hélices.

Le prix doit rester positif (donc s'il est négatif, on le met à 0).

Héritage

Exercice (4)



9) Dans une script, créer deux tableaux d'objets : *garage* (contenant 3 voitures) et *hangar* (contenant 2 avions).

Calculer les prix actuels et afficher les résultats.

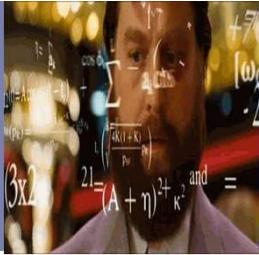
Héritage

Méthodes Finales

- Le mot-clé `final` empêche les classes enfants de redéfinir une méthode ou constante en prefixant la définition avec `final`.
- Si la classe elle-même est définie comme finale, elle ne pourra pas être étendue.
- Les propriétés ne peuvent être déclarées comme finales : seules les méthodes, classes, constantes (à partir de PHP 8.1.0) peuvent être défini comme finale.
- À partir de PHP 8.0.0, les méthodes privées ne peuvent pas être déclaré finale à l'exception du constructeur.

Héritage

Exercice



- Implémentez une méthode `quatreFeu` qui est identique pour toutes les voiture et qui affiche juste un message 4 clignotons.
- Faite en sorte que cette méthode **ne puisse pas être redéfinie**.
- Créez une classe `VoitureElectrique` qui hérite de la classe Voiture
- Essayez de redéfinir la méthode `quatreFeu`

Interface

- Une **interface** vous permet de spécifier quelles méthodes les classes implémentant cette interface doivent implémenter.
- C'est un **contrat** permettant de préciser exactement le modèle à créer.
- Une interface est un **ensemble de prototype** de méthodes publiques.
- Une interface est déclarée comme une classe. Cependant, à la place du mot clé **class** vous devez mettre le mot clé **interface**.
- Dans une Interface **aucune méthode n'est spécifiée**.
- Afin de spécifier qu'une classe doit implémenter une interface on ajoute devant le nom de la classe le mot clé **implements** suivi du **nom de l'interface**.
- Une classe implémentant une interface **doit obligatoirement respecter la totalité du contrat**. **Toutes les méthodes** de l'interface doivent être **implémentées**.

Interface

➤ Les interfaces servent généralement pour deux buts:

1. Permettre aux développeurs de créer des objets de classes différentes mais qui se partagent un ensemble de comportement communs définis par l'interface. Ceci va faire en sorte qu'elles soient interchangeables.
Donc ceci peut permettre à une fonction ou méthode d'accepter et opérer sur un paramètre qui conforme à une interface, sans se soucier de quoi d'autre l'objet peut faire ou comment c'est implémenté. Ceci nous permet d'avoir un code générique qui gère des objets sans se soucier de leur Type.
2. Définir un contrat permettant une abstraction totale de l'implémentation. Ceci facilite le découplage et la modification de l'implémentation sans pour autant briser les relations.

<https://www.php.net/manual/fr/language.oop5.interfaces.php>

Interface

```
interface IVehicule
{
    public function demarrer();
    public function arreter();
}
```



```
class Voiture implements IVehicule
{
    protected $matricule;
    protected $nbChevaux;
    protected $couleur;
    protected $vitesse;

    public function demarrer()
    {
        // TODO: Implement
        // demarrer() method.
    }

    public function arreter()
    {
        // TODO: Implement arreter()
        // method.
    }
}
```

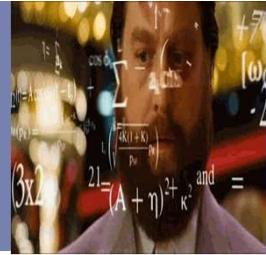
Interface

- Vous pouvez **implémenter plusieurs interfaces** en les **séparant** par une **virgule**
- Les interfaces **peuvent être étendues** comme des classes, en utilisant l'opérateur `extends`
- Les **interfaces peuvent contenir des constantes.**
- Les constantes d'interfaces fonctionnent exactement comme les **constantes de classe.**
- Antérieur à PHP **8.1.0**, elles **ne peuvent pas être redéfinies** par une classe/interface qui les hérite.

<https://www.php.net/manual/fr/language.oop5.interfaces.php>

Interface

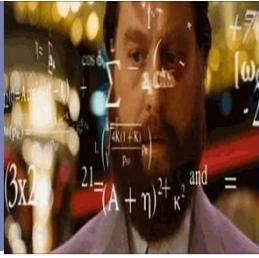
Exercice (1)



- Créer une interface ICrud. Cette Interface permet de gérer les fonctionnalités suivantes (Pour simplifier, vous pouvez vous contenter des deux premières) :
 - `add(element)` qui ajoute un élément à la liste des éléments
 - `findAll()` qui retourne un tableau d'éléments
 - `findOneByCriteria(index, value)` qui retourne un élément par son identifiant
 - `remove(element)` qui supprime un élément de la liste
- Ajouter une interface IdataTable qui permet d'avoir le contrat suivant :
 - Une méthode `getHead` qui retourne les clés des éléments à afficher
 - Une méthode `getData` qui retourne un tableau d'objet avec les keys des éléments à afficher
- Ajouter une Classe Annuaire qui contient un tableau de Personne (id, name, phoneNumber).
- Faite en sorte que la Classe Annuaire implémente l'interface `Icrud et IdataTable`
- Ajouter une page datatables.php qui affiche dynamiquement les classes implémentant le Idataitable
- Remarque pour récupérer une propriété dynamiquement, utiliser la syntaxe suivante
`$element->{$propertyName}` avec `$propertyName` le nom de la propriété à lire
- Remarque pour la datatable consulter le lien suivant :
https://datatables.net/examples/basic_init/zero_configuration.html

Interface

Exercice (2)



localhost:8000/datatables.php

Show 10 entries Search:

name	age	num
aymen	41	23584455
skander	5	2358

Showing 1 to 2 of 2 entries

Previous 1 Next

Polymorphisme

- Supposons qu'on veuille écrire un code pour un personnage qui va rencontrer des magiciens, des guerrier,... (supposons que l'on ait un tableau de personnages qu'il va rencontrer)
- Que faire sachant que la façon dont un Personnage en rencontre un autre peut prendre plusieurs formes : le saluer (Magicien), le frapper (Guerrier), le voler (Voleur)... ??!!!!



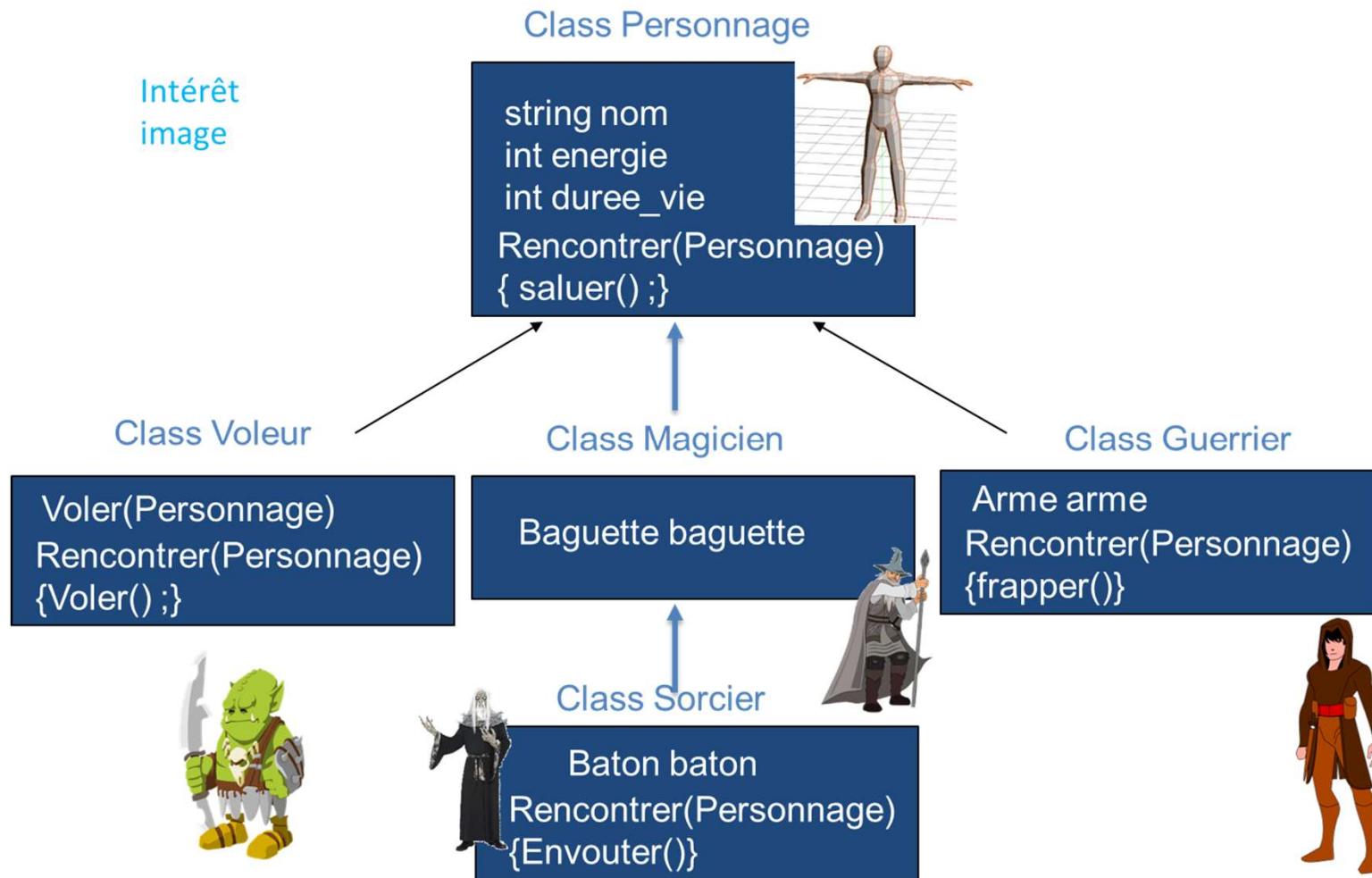
Une fonction qui va chercher le type du personnage et appeler sa fonction !!!!!!!!

Polymorphisme

- Grâce à l'héritage ou les interfaces, le même code pourra être appliqué à un Magicien, un Guerrier , ... qui sont des Personnage.
- Si grâce à l'héritage on peut avoir des fonctions différentes selon le personnage pourquoi ne pas avoir un mécanisme qui permet d'avoir une fonction **générique appliquée à tous les personnages et qui s'adaptera au type du personnage en ayant un comportement différent, propre à chacun**

POLYMORPHISME

Polymorphisme



Polymorphisme

Etudions le pseudo code d'un joueur qui rencontrera des personnes

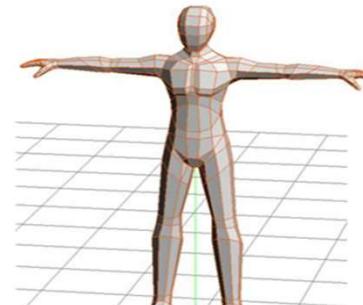
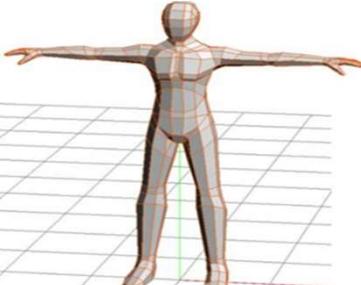
RecontrerPersos()

```
{  
Personnage $joueur;  
$tableauPersonnages = array();  
$ tableauPersonnages[] = new Voleur();  
$ tableauPersonnages[] = new Magicien();
```

Si le personnage posséde sa propre fonction rencontrer il l'utilise sinon il utilisera la fonction de la super classe

```
foreach ($ tableauPersonnages as $personnage) {  
    $joueur.recontrer($personnage);  
}
```

Dans la première itération de la boucle c'est un voleur qui est appelé, dans la seconde c'est un magicien



**Pas de fonction
rencontrer donc :
Saluer**



Polymorphisme

- En POO, le polymorphisme est le fait que les instances d'une sous-classe, lesquelles sont substituables aux instances des classes de leur ascendance (en argument d'une méthode, lors d'affectations), gardent leurs propriétés propres.
- Le choix des méthodes à invoquer se fait lors de l'exécution du programme en fonction de la nature réelle des instances concernées.
- La mise en œuvre se fait au travers de :
 - l'héritage (hiérarchies de classes) ;
 - Les Interfaces

Polymorphisme

Exemple

```
interface Forme
{
    public function calculerSurface();
}

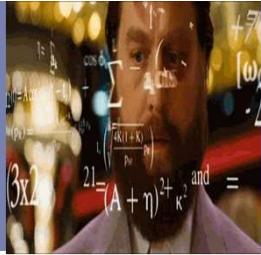
class Cercle implements Forme
{
    public function __construct(private $rayon)
    {
        $this->rayon = $rayon;
    }
    public function calculerSurface()
    {
        return pi() * pow($this->rayon, 2);
    }
}
```

```
class Rectangle implements Forme
{
    public function __construct(private $longueur, private $largeur)
    {
        $this->longueur = $longueur;
        $this->largeur = $largeur;
    }
    public function calculerSurface()
    {
        return $this->longueur * $this->largeur;
    }
}
```

```
function afficherSurface(Forme $forme)
{
    echo 'Surface : ' . $forme->calculerSurface() . PHP_EOL;
}
```

Héritage

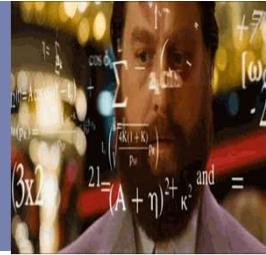
Exercice



- Pour votre classe Person, comment faire pour réunir dans un même tableau 5 employés, 2 chefs et 1 directeur ?
- Implémenter le tableau et afficher les informations de tout les éléments.

Polymorphisme

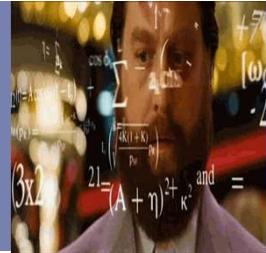
Exercice



1. Créer une classe **Produit**. Cette classe contient la désignation du produit, son prixHt et sa référence.
2. Créer ensuite la classe **Electromenager** qui hérite de la classe Produit et qui a comme attributs **consommation** qui représente la consommation en courant ainsi que **TaxeConsommation** qui varie selon la valeur de la consommation sur 3 paliers. Si la consommation est inférieur à 100 on a 0% taxe, si la consommation est entre 100 et 300, c'est 10% de taxe, sinon c'est 35% de taxe.
3. Créer finalement la Classe **Textile** qui a comme attributs couleur et TVA.
4. En utilisons le **polymorphisme**, nous voulons maintenant et selon les achats dans le panier d'un utilisateur lui afficher le prix Total.
5. Créer la classe **panier** qui permettra d'ajouter des **éléments** au panier de l'utilisateur, de les **supprimer** et d'afficher le Prix total du panier avec une méthode **calculTotalPrix**.
6. La fonction **calculTotalPrix** devra être générique dans le sens ou même si on ajoute de nouveaux catégories de Produits cette fonction ne changera pas.
7. Ajouter une classe **ProduitSolde** qui contient un attribut **soldé** contenant le pourcentage à déduire du prix de base et qui est non taxé.
8. Faites le nécessaire pour pouvoir ajouter des produits soldés à votre panier et vérifier que tout fonctionne comme il faut sans modifier votre classe Panier.

Polymorphisme

Exercice



- Reprenons l'exercice des Pokémons. En plus des Pokémons normaux (décrits à travers la classe Pokemon) on identifie trois types de Pokémons.
- Les Pokémons de type Feu, les Pokémons de type Eau et les Pokémons de type Plante:
- les Pokémons de type Feu sont super efficaces contre les Pokémons de type Plante et leur infligent deux fois plus de dégâts ($2 * \text{atk}$). Par contre, ils sont très peu efficaces contre les Pokémons de type Eau ou de type Feu et ne leur infligent que la moitié des dégâts ($0.5 * \text{atk}$). Ils infligent des dégâts normaux aux Pokémons de type Normal.
- les Pokémons de type Eau sont super efficaces contre les Pokémons de type Feu et leur infligent deux fois plus de dégâts ($2 * \text{atk}$). Par contre, ils sont très peu efficaces contre les Pokémons de type Eau ou de type Plante et ne leur infligent que la moitié des dégâts ($0.5 * \text{atk}$). Ils infligent des dégâts normaux aux Pokémons de type Normal.
- Les Pokémons de type Plante sont super efficaces contre les Pokémons de type Eau et leur infligent deux fois plus de dégâts ($2 * \text{atk}$). Par contre, ils sont très peu efficaces contre les Pokémons de type Plante ou de type Feu et ne leur infligent que la moitié des dégâts ($0.5 * \text{atk}$). Ils infligent des dégâts normaux aux Pokémons de type Normal.
- Créez trois classes PokemonFeu, PokemonEau et PokemonPlante qui héritent de la classe Pokemon.
- Créer des combats entre Pokémons de différents types.

Auto Chargement

Autoload

- Il faut inclure une Classe afin de l'utiliser un script PHP en utilisant le mot clé **require**.
 - Exemple : require “ConnexionBD.php”
- Afin d'éviter de **charger manuellement** chaque classe à utiliser dans un script, il vaut mieux utiliser le concept d'**autoload**.
- PHP possède une **pile d'autoloads** qui permet de lister les fonctions à utiliser pour gérer le **chargement automatique** des **classes instanciées et non déclarées**.
- La solution est donc d'ajouter une fonction qui permet de charger ces classes.
- Afin d'ajouter une fonction à la Pile de fonctions d'auto chargement on utiliser la fonction **spl_autoload_register()** qui prend en paramètre une chaîne de caractère représentant le nom de la fonction.

Auto Chargement

Autoload

```
function loadClass($maClasse)
{
    require $maClasse . '.php';
}
```

Les méthodes magiques

- Les **méthodes magiques** sont des méthodes spéciales qui sont **appelées implicitement** suite à un évènement particulier.
- L'idée des méthodes magiques est **d'intercepter un événement**, de **faire un traitement**.
- Le nom des méthodes magiques **commence par __**.
- Nous avons déjà rencontré des méthodes magiques. **Qui sont elles ?**

<http://php.net/manual/en/language.oop5.magic.php>

Les méthodes magiques

__get

- Lorsque vous essayez d'accéder à un attribut inexistant ou private d'un objet, la méthode magique __get() est appelé. Elle prend en paramètre le nom de l'attribut appelé.
- Cette fonction retourne une valeur de retour.
- Ajouter cette fonction à une classe et tester la.

```
class MyExampleClass
{
    public function __get ($name)
    {
    }

}
```

Les méthodes magiques

__set

- De même, lorsque vous accéder à un attribut pour le modifier et qu'il n'existe pas ou que vous n'avez pas le droit de le modifier, la méthode magique __set() est invoquée. Cette fonction reçoit deux paramètres qui sont le nom de l'attribut et la valeur à affecter.
- Ajouter cette fonction à une classe et tester la.

```
class MyExampleClass
{
    public function __set($name, $value)
    {
    }
}
```

Les méthodes magiques

__get

```
public function __set($name, $value)
{
    echo "you try to change {$name} attribute,
with {$value} we are sorry this attribute is not
accessible";
}
```

Les méthodes magiques

__call

- Quand vous essayez d'accéder à une méthode inexistante ou privée pour vous, `__call()` est automatiquement appelée. Cette fonction reçoit deux paramètres contenant le nom de la méthode et en second paramètre un tableau contenant les arguments passés à la méthode.
- Ajouter cette fonction à une classe et tester la.

```
class MyExampleClass
{
    public function __call($name, $arguments)
    {
    }
}
```

Les méthodes magiques

__toString

- La méthode __toString() est quant à elle **appelée lorsque vous essayé d'afficher l'objet en tant que string**. Par exemple si vous utilisez echo pour afficher votre objet.
- Essayez d'afficher un objet avec echo. Que se passe-t-il ?
- Implémentez la méthode __toString() afin qu'elle affiche les attributs de votre objet. Retestez encore votre echo.

```
class MyExampleClass
{
    public function __toString()
    {

    }
}
```

Les méthodes magiques

__invoke

- Afin d'utilisez votre objet en tant que méthode, vous pouvez utiliser la fonction magique `__invoke()` qui prend en paramètres autant d'arguments passé lors de l'appel.
- Commencez par instancier un objet \$myObject.
- Utilisez cet objet en tant que fonction comme ceci : \$myObject(1,2,3);

```
class MyExampleClass
{
    public function __invoke(...$arguments)
    {
        }
}
```

- Implémentez la méthode `__invoke()` (faite en sorte qu'elle affiche juste un message et la liste des paramètres reçues) et réessayer \$myObject(1,2,3);

PHP Accès à la BD

Extensions gestion BD

- Pour la gestion des Bases de données, PHP offre plusieurs extensions :
- L'extension **mysql_** : Ensemble de fonction commençons par mysql_ et permettant de communiquer avec mysql. Elle sont devenue **obsolète**.
- L'extension **mysqli_** : ce sont des fonctions améliorées d'accès à MySQL.
- L'extension **PDO** : Outil complet qui permettant une abstraction du type de la base de données traitée permettant ainsi de se connecter aussi bien à MySQL que PostgreSQL ou Oracle.

PHP Accès à la BD

Extensions gestion BD

Tableau comparatif

Src : <http://www.php.net/manual/fr/mysqli.overview.php>

	Extension MySQL	Extension mysqli	PDO (avec le pilote PDO MySQL Driver et MySQL Native Driver)
Version d'introduction en PHP	Avant 3.0	5.0	5.0
Inclus en PHP 5.x	Oui, mais désactivé	Oui	Oui
Statut de développement MySQL	Maintenance uniquement	Développement actif	Développement actif depuis PHP 5.3
Recommandée pour les nouveaux projets MySQL	Non	Oui	Oui
L'API supporte les jeux de caractères	Non	Oui	Oui
L'API supporte les commandes préparées côté client	Non	Oui	Oui
L'API supporte les procédures stockées	Non	Oui	Oui
L'API supporte les commandes multiples	Non	Oui	La plupart
L'API supporte toutes les fonctionnalités MySQL 4.1 et plus récent	Non	Oui	La plupart



Nous nous focalisons donc sur **PDO**

PHP Accès à la BD

PDO

- PDO : PHP Data Objects
- Extension fournissant des services d'accès aux bases de données.
- Fournie avec plusieurs drivers (MySQL, sqlite, PostgreSQL)
- Disponible par défaut à partir des serveurs PHP 5.1.0

PHP Accès à la BD

Connexion à une BD

➤ Pour se connecter à une base de données on instancie un objet PDO de la façon suivante :

```
$maDb_connexion = new PDO('mysql:host=localhost;dbName=nomDeLaBase',  
'userName', 'motDePasse');
```

On crée donc une nouvelle instance de PDO qu'on récupère dans la variable `$maDb_connexion`. Le constructeur nécessite trois paramètres :

- . le driver utilisé, l'adresse du serveur et le nom de la base noté ainsi `driver:host=serveur; dbName=nomDeLaBase'`
- . le nom d'utilisateur à utiliser pour la connexion au serveur
- . le mot de passe du dit utilisateur
- Ces informations diffèrent un peu selon le pilote.

PHP Accès à la BD

Connexion à une BD

- Afin de gérer les erreurs liées à la Connexion à la BD, il faut capturer les erreurs de type **PDOException**.

```
try {  
    $db_connexion = new PDO('mysql:host=localhost;dbname=user1', 'user1',  
    'motdepass');  
}  
catch (PDOException $e) {  
    print "Erreur : " . $e->getMessage();  
    die();  
}
```

PHP Accès à la BD

Pattern Design singleton

Pattern Design singleton

- But : Singleton garantit **qu'une classe n'a qu'une seule instance** et fournit un point d'accès global à cette instance.
- Principe
 - Empêcher les développeur d'utiliser le ou les constructeurs de la classe : déclarer privé tous les constructeurs de la classe.
 - Problème : la classe n'est plus instanciable que par elle-même.
 - Solution : Construire un pseudo constructeur à travers une méthode static. Par convention il sera appelé `getInstance`.
 - On crée un attribut statique stockant **l'unique instance** de la classe.
 - Dans `getInstance` on teste si cet attribut est nul
 - Si `null` on instancie un objet et on le retourne
 - Sinon on retourne l'instance existante

PHP Accès à la BD

Pattern Design singleton

```
<?php
class ConnexionBD
{
    private static $_dbname = "bdphp5";
    private static $_user = "root";
    private static $_pwd = "";
    private static $_host = "localhost";
    private static $_bdd = null;
    private function __construct()
    {
        try {
            self::$_bdd = new PDO("mysql:host=" . self::$_host . ";dbname=" . self::$_dbname . ";charset=utf8", self::$_user, self::$_pwd,
array(PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES UTF8'));
        } catch (PDOException $e) {
            die('Erreur : ' . $e->getMessage());
        }
    }
    public static function getInstance()
    {
        if (!self::$_bdd) {
            new ConnexionBD();
        }
        return (self::$_bdd);
    }
}
```

PHP Accès à la BD

Pattern Design singleton

- Afin d'afficher les détails des erreurs liées à la BD, il faut activer le suivi de ces erreurs lors de la connexion à la BD.

```
try {  
    $db_connexion = new PDO('mysql:host=localhost;dbname=user1', 'user1',  
    'motdepass', array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION)  
);  
}  
} catch (PDOException $e) {  
    print "Erreur : " . $e->getMessage();  
    die();  
}
```

PHP Accès à la BD

Requête simple

➤ Afin d'interroger une BD via PDO, nous utilisons la méthode `query` qui prend en paramètre la requête à exécuter.

➤ Exemple :

```
$req=<< select * From maTable >>;
```

```
$reponse = $_bdd->query($req);
```

➤ La variable \$reponse contiendra un objet contenant la réponse de MySQL qui n'est pas directement exploitable.

➤ Pour exploiter ces données nous utilisons la méthode `fetch` qui retourne une ligne ou `fetchAll` qui retourne un tableau contenant toutes les lignes du jeu d'enregistrements

➤ L'un des paramètres des méthodes `fetch` et `fetchAll` est l'attribut `fetch_style` qui permet de spécifier le type de la valeur de retour de `fetch` et `fetchAll`

PHP Accès à la BD

Fetch style

- Contrôle comment la prochaine ligne sera retournée à l'appelant. Cette valeur doit être une des constantes `PDO::FETCH_*`.
- `PDO::FETCH_BOTH` (défaut): retourne un tableau indexé par les noms de colonnes et aussi par les numéros de colonnes, commençant à l'index 0, comme retournés dans le jeu de résultats
- `PDO::FETCH_ASSOC` : retourne un tableau indexé par le nom de la colonne comme retourné dans le jeu de résultats
- `PDO::FETCH_NUM` : retourne un tableau indexé par le numéro de la colonne comme elle est retourné dans votre jeu de résultat, commençant à 0
- `PDO::FETCH_OBJ` : retourne un objet anonyme avec les noms de propriétés qui correspondent aux noms des colonnes retournés dans le jeu de résultats
- ...

PHP Accès à la BD

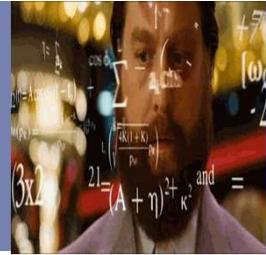
Fetch style

➤ Exemple de parcours en utilisant des objets :

```
$games = $rep->fetchAll(PDO::FETCH_OBJ);  
  
foreach($games as $game) :  
    echo $game->nom." - ".$game->commentaires."<br>";  
endforeach;
```

PDO

Exercice



1. Créer une base de données
2. Créer la table « student »
3. Chaque étudiant devra avoir:
 1. Un id qui est sa clé primaire
 2. Un name
 3. Une date de naissance
4. Remplissez manuellement la table avec quelques étudiants
5. Créer une page qui affiche la liste des étudiants

id	name	birthday
1	Aymen	1982-02-07
2	Skander	2018-07-11

PDO

Exercice

- Devant Chaque étudiant, ajouter un lien détails, en cliquant sur ce lien, rediriger l'utilisateur vers une page détailEtudiant.php affichant les détails de l'étudiant.

id	name	birthday	
1	Aymen	1982-02-07	
2	Skander	2018-07-11	



Avmen >

GI

1982-02-07

PHP Accès à la BD

Requête paramétrable

- Afin de paramétriser une requête nous pouvons utiliser deux méthodes:
- Paramétrage manuelle en concaténons les paramètres dans la requête. **ENORME FAILLE DE SECURITE SQL INJECTION**
- les requêtes préparées
- Deux types de requêtes préparées :
- En utilisant les marqueurs « ? »
- En utilisant les marqueurs nominatifs

PHP Accès à la BD

Requête préparée

- Pour simplifier les choses une requête préparé est une requête dont les paramètres sont insérés dans la fonction lors de l'exécution.
- Elle est effectuée en 2 étapes :
 - Préparer la requête à l'aide de la méthode `prepare`
 - Transmettre les paramètres dans un tableau et exécuter la requête préparée à l'aide de la méthode `execute`
 - Les paramètres sont indiqués dans `l'ordre d'apparition` dans la requête préparée
 - Le contenu des variables est automatiquement sécurisé pour prévenir les risques d'injection SQL.

Exemple :`prepare`

```
$req = $bdd->prepare ('SELECT * FROM personne WHERE nom = ? AND age <= ? ORDER BY cin');  
$req->execute(array($nom, $age));
```

PHP Accès à la BD

Requête préparée

- Pour simplifier les choses une requête préparé est une requête dont les paramètres sont insérés dans la fonction lors de l'exécution.
- Elle est effectuée en 2 étapes :
 - Préparer la requête à l'aide de la méthode `prepare`
 - Transmettre les paramètres dans un tableau et exécuter la requête préparée à l'aide de la méthode `execute`
 - Les paramètres sont indiqués dans **l'ordre d'apparition** dans la requête préparée
 - Le contenu des variables est automatiquement sécurisé pour prévenir les risques d'injection SQL.

Exemple :

```
$req = $bdd->prepare('SELECT * FROM personne WHERE nom = ? AND age <= ? ORDER BY cin');  
$nom=<< test >>;$age=<< 10 >>;  
$req->execute(array($nom, $age));
```

PHP Accès à la BD

Requête préparée

- Requête paramétrable avec des [marqueurs nominatifs](#)
- Afin de rendre la requête préparée plus lisible, ion peut remplacer les ? Par des [marqueurs nommés](#)
- Un marqueur nommé est un [nom précédé par « : »](#)

➤ Exemple :

```
$req = $bdd->prepare('SELECT * FROM personne WHERE nom = :nom AND age <= :age ORDER BY cin');  
$req->execute(array('nom'=>$nom, 'age'=>$age));
```

L'ordre des paramètres [n'a plus d'importance](#) vu que nous utilisons des [tableaux associatifs](#).

PDO

Exercice

1. Modifier le code de la page détail afin de récupérer les détails d'un étudiant en utilisant les prepared statements

id	name	birthday	
1	Aymen	1982-02-07	
2	Skander	2018-07-11	



Avmen >

Gl

1982-02-07

PHP Accès à la BD

Requête préparée

- Afin de récupérer le nombre d'enregistrement retourné par la requête on utilise la méthode `rowCount`.

Exemple :

```
$req="select * maTable";
$rep = $bdd->query($req);
echo "le nombre d'enregistrements est :".$rep->rowCount();
```

- Afin de récupérer l'id du dernier enregistrement, on utilise la méthode `lastInsertId`.

Remarque : Ca ne marche qu'après un INSERT.

Exemple :

```
echo "le dernier id est :".$bdd->lastInsertId()."<br>";
```

PHP Accès à la BD

Ajout d'enregistrement BD

- Afin d'ajouter, modifier et supprimer un enregistrement dans la Base de données PDO nous offre la méthode `execute`.
- Cette méthode prend en paramètre la requête à exécuter.
- On peut utiliser la méthode `prepare` afin de préparer la requête à exécuter.
 - `$req= $bdd->prepare(« La requête à préparer »)`
- Une fois la requête préparée, on utilise la méthode `execute` en lui passant un tableau associatif contenant la liste des paramètres.

```
$req= $bdd->prepare ("insert into matable (`champ1`,  
`champ2` , `champn`) VALUES (:val1,:val2,:valn)");  
$req->execute (array (  
    'val1'=>'val1',  
    'val2'=>'val2',  
    'valn'=>5  
)) ;
```

PHP Accès à la BD

Modification d'enregistrement BD

➤ Même fonctionnement que l'ajout.

➤ Requête update.

```
$req= $bdd->prepare(<< update matable set champ1=:val1,  
champ2= :val2, champ3= :champ3 where champ_condition=  
:cnd" );  
  
$req->execute(array(  
    'val1'=>'newval1',  
    'val2'=>'newval2',  
    'valn'=>7,  
    'cnd'=>'valCnd',  
) ) ;
```

PHP Accès à la BD

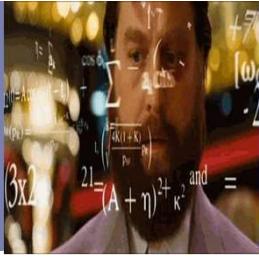
Suppression d'enregistrement BD

- Même fonctionnement que l'ajout et la mise à jour.
- Requête delete.

```
$req= $bdd->prepare(<< delete from matable where  
champ_condition= :cnd" );  
  
$req->execute(array(  
    'cnd'=>'valCnd',  
) );
```

PDO

Exercice



1. Nous voulons implémenter une petite application de gestion des étudiants.
2. La plateforme ne peut être accessible que pour un utilisateur authentifié.
3. Un utilisateur possède les propriétés suivantes :
 1. id
 2. username
 3. email
 4. role
4. Un administrateur peut consulter et gérer (CRUD) les étudiants et les sections
5. Il peut faire la même chose pour les sections
6. Un étudiant a
 1. Id
 2. Name
 3. birtday
 4. Image
 5. section

PDO

Exercice

7. Une section possède un

- id
- Designation
- description

Pour chaque table, penser à créer une classe qui encapsule ses fonctionnalités.

Students Management System [Home](#) [Liste des étudiants](#) [Liste des sections](#) [Logout](#)

Hello, PHP LOVERS! Welcome to your
adminstration Platform

PDO

Exercice

La liste est paginé en utilisant une datatable

Vous pouvez exporter la liste en excel, csv et PDF

Vous pouvez filtrer la liste des étudiants par nom (en PHP) ou via le filtre de la datatable

localhost:8000/studentList.php

Students Management System Home Liste des étudiants Liste des sections Logout

Liste des étudiants

Veuillez renseigner votre

Search:

ID	Image	Name	Birthday	Section	Actions
1		Aymen	1982-02-07	GI	
2		Skander	2018-07-11	GI	

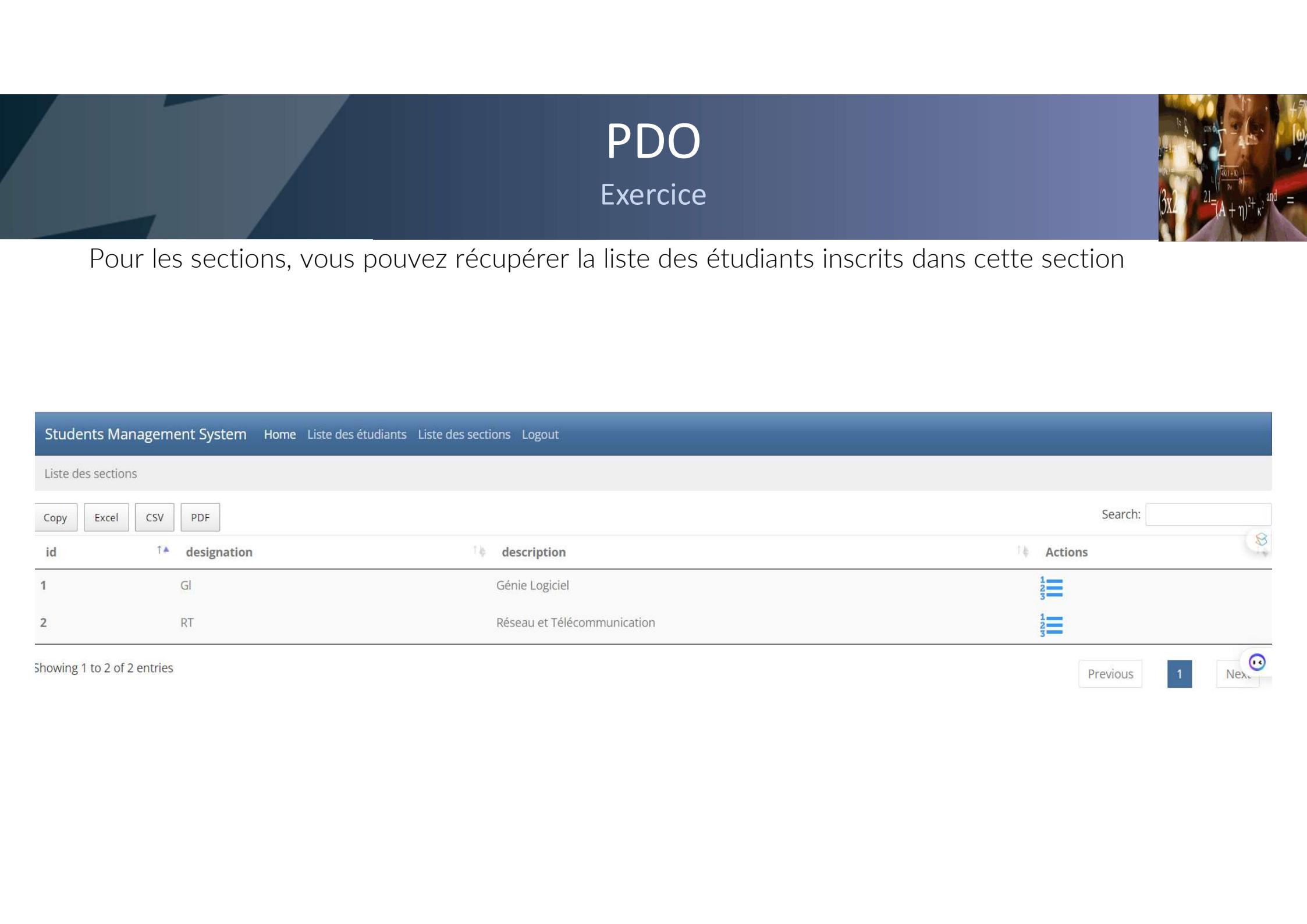
Showing 1 to 2 of 2 entries

Previous Next

PDO

Exercice

Pour les sections, vous pouvez récupérer la liste des étudiants inscrits dans cette section



Students Management System [Home](#) [Liste des étudiants](#) [Liste des sections](#) [Logout](#)

Liste des sections

[Copy](#) [Excel](#) [CSV](#) [PDF](#) [Search:](#)

id	designation	description	Actions
1	GL	Génie Logiciel	
2	RT	Réseau et Télécommunication	

Showing 1 to 2 of 2 entries

[Previous](#) [1](#) [Next](#)

PDO

Exercice

Pour un utilisateur avec un rôle user, il peut accéder uniquement en mode lecture.

Liste des étudiants						
	Copy	Excel	CSV	PDF	Search:	
id	image	name	birthday	section	Actions	
1		Aymen	1982-02-07	GI		
2		Skander	2018-07-11	GI		

Showing 1 to 2 of 2 entries

[Previous](#) [1](#) [Next](#)

PDO

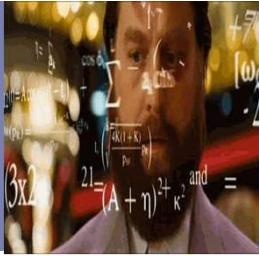
Exercice

1. Créer une classe générique Repository qui offre les fonctionnalités de base d'une base de données suivantes :

1. findAll qui retourne la liste de tous les enregistrements
2. findById qui cherche l'enregistrement d'un id
3. create qui ajoute un enregistrement
4. delete qui supprime l'enregistrement d'un id

Utiliser ce qu'on a vu en POO

Tester votre code avec la table section et user.





MERCI.

