La Programmation Orientée Objet (POO)

Plan

- Introduction
  - La POO c’est quoi ?
  - Un peu d’histoire
- Quelques notions de bases importantes
  - C’est quoi un objet
  - C’est quoi une classe
  - Retour sur UML
  - THIS ? c’est quoi ?
- Déclaration et instanciation

Plan

Les principaux concepts

- Le constructeur
- L’encapsulation
- L’association
- Héritage

Pour aller plus loin

- L’abstraction
- UN peu de statique

La POO c’est quoi ?

La qualité d’un logiciel :

1. **L’exactitude** : aptitude d’un programme à fournir le résultat voulu et à répondre ainsi aux spécifications
2. La **robustesse** : aptitude à bien réagir lorsque l’on s’écarte des conditions normales d’utilisation
3. **L’extensibilité** : facilité avec laquelle un programme pourra être adapté pour répondre à l’évolution des spécifications
4. La **réutilisabilité** : possibilité d’utiliser certaines parties du logiciel pour résoudre un autre problème
5. La **portabilité** : facilité avec laquelle on peut exploité un même logiciel dans différentes implémentations
6. **L’efficience** : temps d’exécution, taille mémoire

L’approche fonctionnelle

C’est coder en privilégiant l’utilisation de fonction tant pour la hiérarchisation que pour la structure du code

Le code est exécuté lignes après lignes

En suivant un algorithme linéaire

Co[illegible]

```php
if(!isset($_SESSION['user'])){
    header("Location:login.php");
} else {
    $secure = new Securite($_SESSION['user']['email'], $_SESSION['user']['password'], $_SESSION['user']['name'], false);
}
```

L’approche Objet

On considère ce que le système doit faire, mais aussi ce qu’il est !

Tout est objet !

Un site devient un ensemble d’objets

Qui interagissent entre eux

```php
<?php
  class Securite
  {
    var $name;
    var $email;
    var $password;

    function __construct($e, $p, $n = null, $crypt = true)
    {
      $this->setName($n);
      $this->setEmail($e);
      if($crypt)
        $this->setPassword($p);
      else {
        $this->setPassword2($p);
      }
    }

    public function getName () { return $this->name;}
    public function setName($newName){$this->name = $newName; }
    public function getEmail () { return $this->email;}
    public function setEmail($newEmail){$this->email = $newEmail; }
    public function getPassword () { return $this->password;}
    public function setPassword($newPassword){$this->password = sha1($newPassword); }
    public function setPassword2($newPassword){$this->password = $newPassword; }

    public function is_connected($pdo)
    {
      $sql = 'SELECT * FROM `users` WHERE `email` = \''.$this->getEmail().'\' AND `password` = \''.$this->getPassword().'\' LIMIT 1;';
      $req = $pdo->query($sql);
      $result = $req->fetchAll();
      $_SESSION['sql'] = $sql;
      if(sizeof($result) == 1)
      {
        return $this->connection($result);
      }
      else
      {
        return false;
      }
    }

    public function connection($user){
      $this->setName(utf8_encode($user[0]['name']));
      $_SESSION['user']['name'] = $this->getName();
```

Comparons les approches

En programmation procédurale : décomposition de tâches en sous-tâches

En programmation objet : on identifie les acteurs (les entités comportementales) du problème puis on détermine la façon dont ils doivent interagir pour que le problème soit résolu.

Un peu d’histoire

Les concepts de la POO naissent au cours des années 1970 avec Simula puis Smalltalk.

A partir des années 1980, les principes de la POO sont appliqués dans de nombreux langages.

Avènement de la POO dans les années 1990

Notion de base : l’objet

Selon Wikipedia :

Un objet est un conteneur symbolique et autonome qui contient des informations et des mécanismes concernant un sujet, manipulés dans un programme. Le sujet est souvent quelque chose de tangible appartenant au monde réel. C'est le concept central de la programmation orientée objet (POO).

Notion de base : l'objet

Objet = Etat + Comportement [+ Identité]

- Etat : valeurs des attributs (données) d'un objet
- Comportement : opérations possibles sur un objet déclenchées par des stimulations externes (appels de méthodes ou messages envoyées par d'autres objets)
- Identité : chaque objet à une existence propre (il occupe une place en mémoire qui lui est propre). Les objets sont différenciés par leurs noms

Notion de base : l’objet

Un personnage est un objet avec

Des caractéristiques (des attributs) :

Un nom

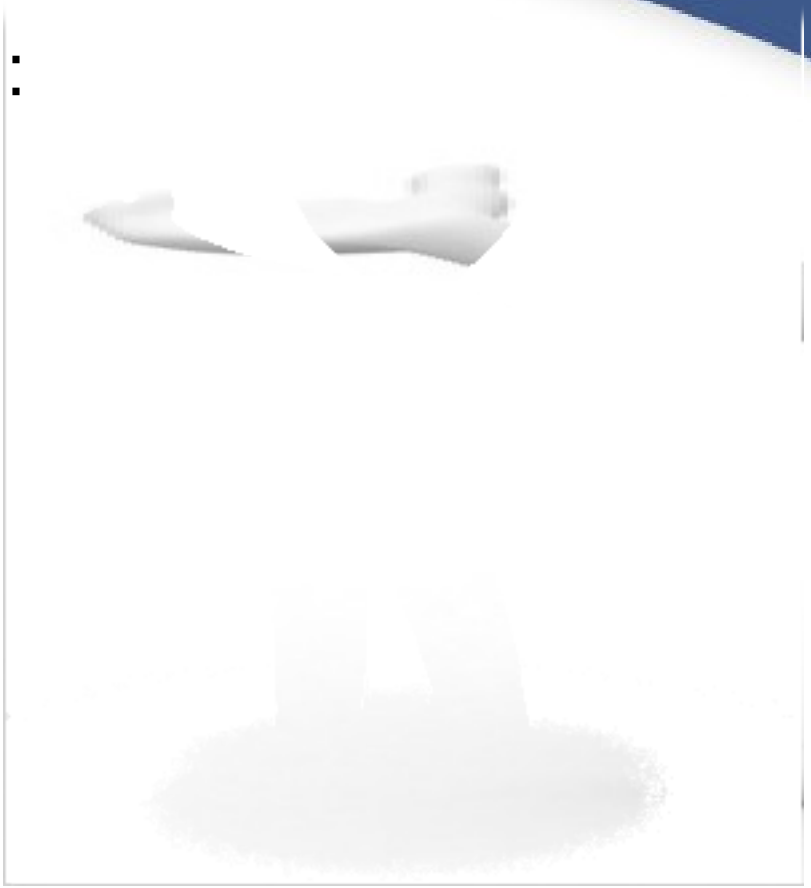
Un prénom

Une taille

Un poids

Une localisation

Etc.



Notion de base : l’objet

Un personnage est un objet avec

Des actions (des fonctions):

Se déplacer

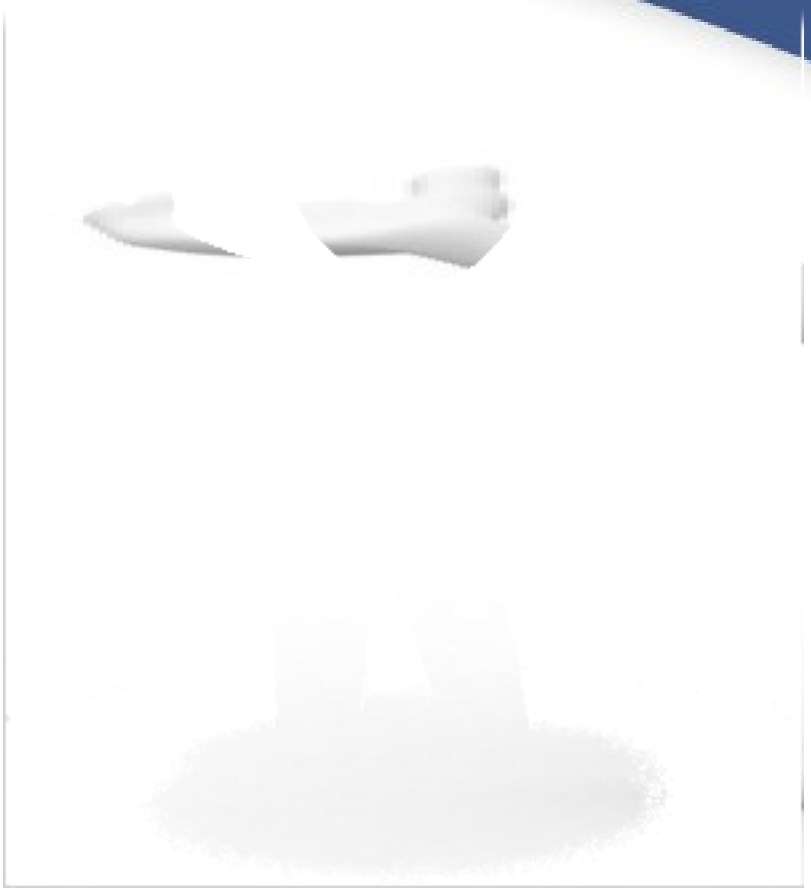
Parler

Apprendre

Créer

Supprimer

Etc.



Le Tower Défense

Le jeu du tower défense est constitué de plusieurs éléments :

- Des systèmes de défenses
- Des projectiles
- Un terrain
- Voyez vous autre chose comme objet ?

Notion de base : la classe

Selon Wikipedia :

En programmation orientée objet, la déclaration d'une classe regroupe des membres, méthodes et propriétés (attributs) communs à un ensemble d'objets.

La classe déclare, d'une part, des attributs représentant l'état des objets et, d'autre part, des méthodes représentant leur comportement.

Une classe représente donc une catégorie d'objets. Elle apparaît aussi comme un moule ou une usine à partir de laquelle il est possible de créer des objets ; c'est en quelque sorte une « boîte à outils » qui permet de fabriquer un objet. On parle alors d'un objet en tant qu'instance d'une classe (création d'un objet ayant les propriétés de la classe).

Le Tower Defense

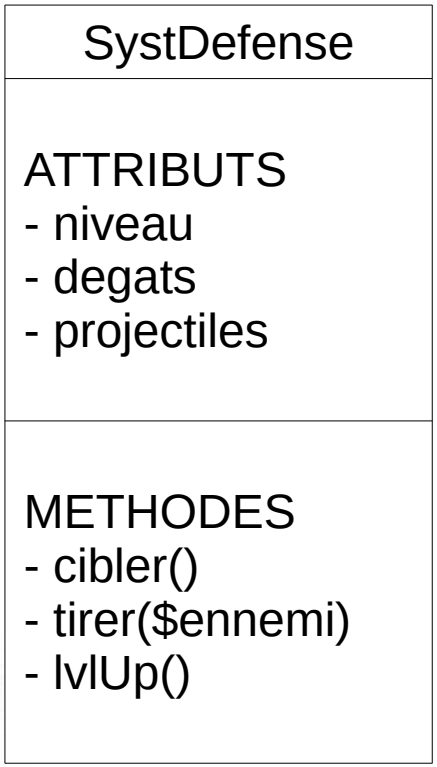
Un objet se définit à travers une classe :

La classe SystDefense :

```php
<?php

/**
 *
 */
class SystDefense
{
    //Attributs
    private $niveau;
    private $degats;
    private $projectiles

    //Méthodes

    // Le constructeur
    function __construct()
    {
        # code...
    }

    public function cibler(){
        # code...
    }

    public function tirer($ennemi){
        # code...
    }

    public function lvlUp(){
        # code...
    }
}

```

Retour sur UML
Le Tower Defense

On revient sur le diagramme de classe SystDefense :



THIS ? C’est quoi ?

This est un mot clé spécifique à l’orienté objet

En php il est précédé d’un $

Dans tous les cas il représente l’objet « courant » (celui dans lequel se situe la ligne de code).

Il n’a pas besoin d’être instancié et s’utilise comme une variable désignant l’objet courant

Déclaration et instanciation

```php
<?php

/**
 *
 */
class SystDefense
{
    //Attributs
    private $niveau;
    private $degats;
    private $projectiles

    //Méthodes

    // Le constructeur
    function __construct()
    {
        # code...
    }

    public function cibler(){
        # code...
    }

    public function tirer($ennemi){
        # code...
    }

    public function lvlUp(){
        # code...
    }
}

$tour = new SystDefense;
$tour->tirer($ennemi);
$tour->lvlUp();
```

Le mot clé class en php permet de déclarer une classe

Ensuite on créé :

- les attributs
- le constructeur
- les méthodes

Pour instancier un objet on utilise le mot clé new

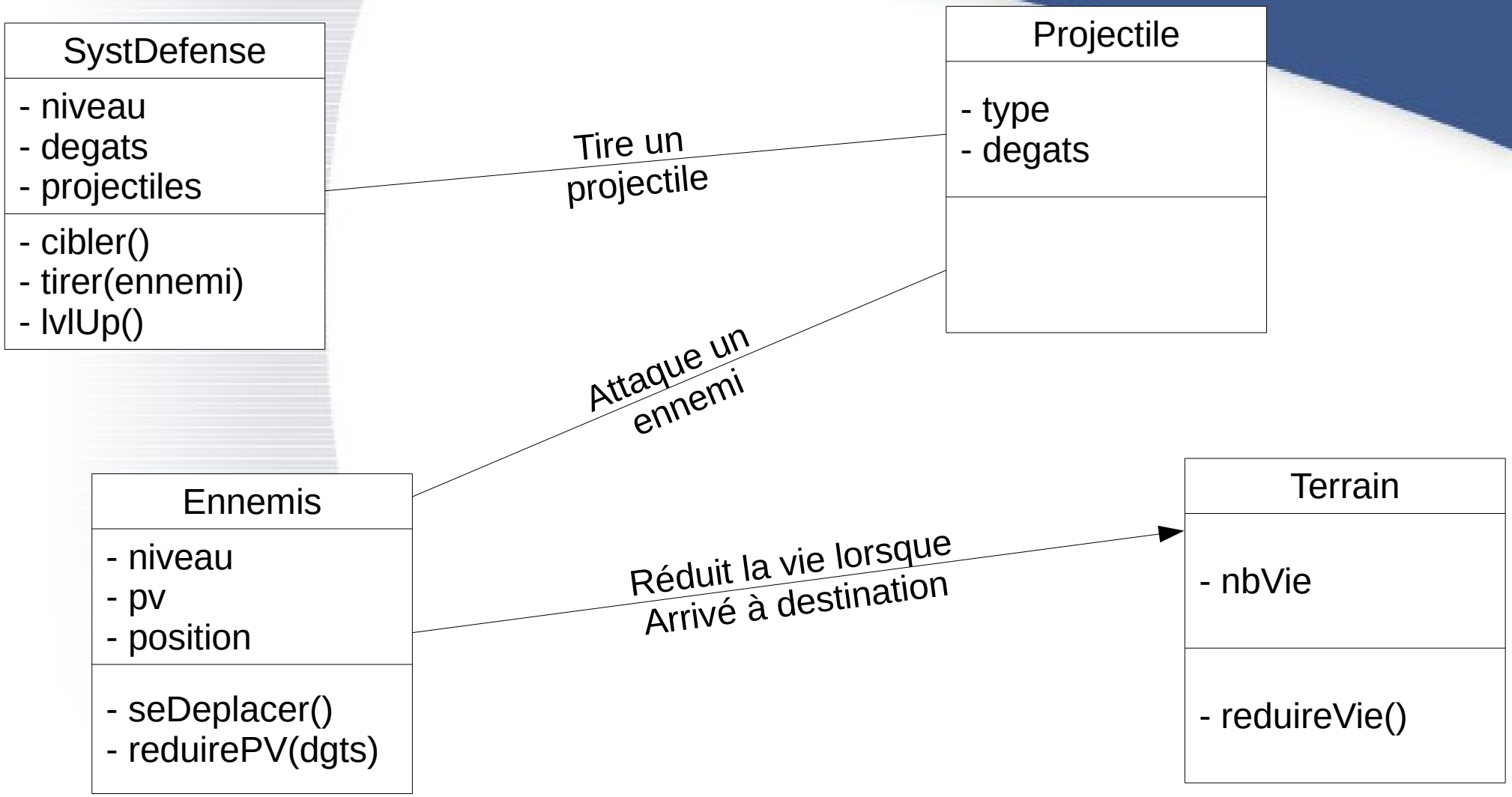
A vous de jouer

Créez un schéma d'interaction et de classe entre les différents objets suivant :

- SystDefense
- Projectile
- Ennemis

Pour les plus avancés :
créez les classes correspondantes !

A vous de jouer
Tower Defense



A vous de jouer

Créez les classes parents ainsi

que les classes filles correspondantes :

- SystDefense
  - Tour missile
  - Tour flamme
- Ennemis
  - Boss

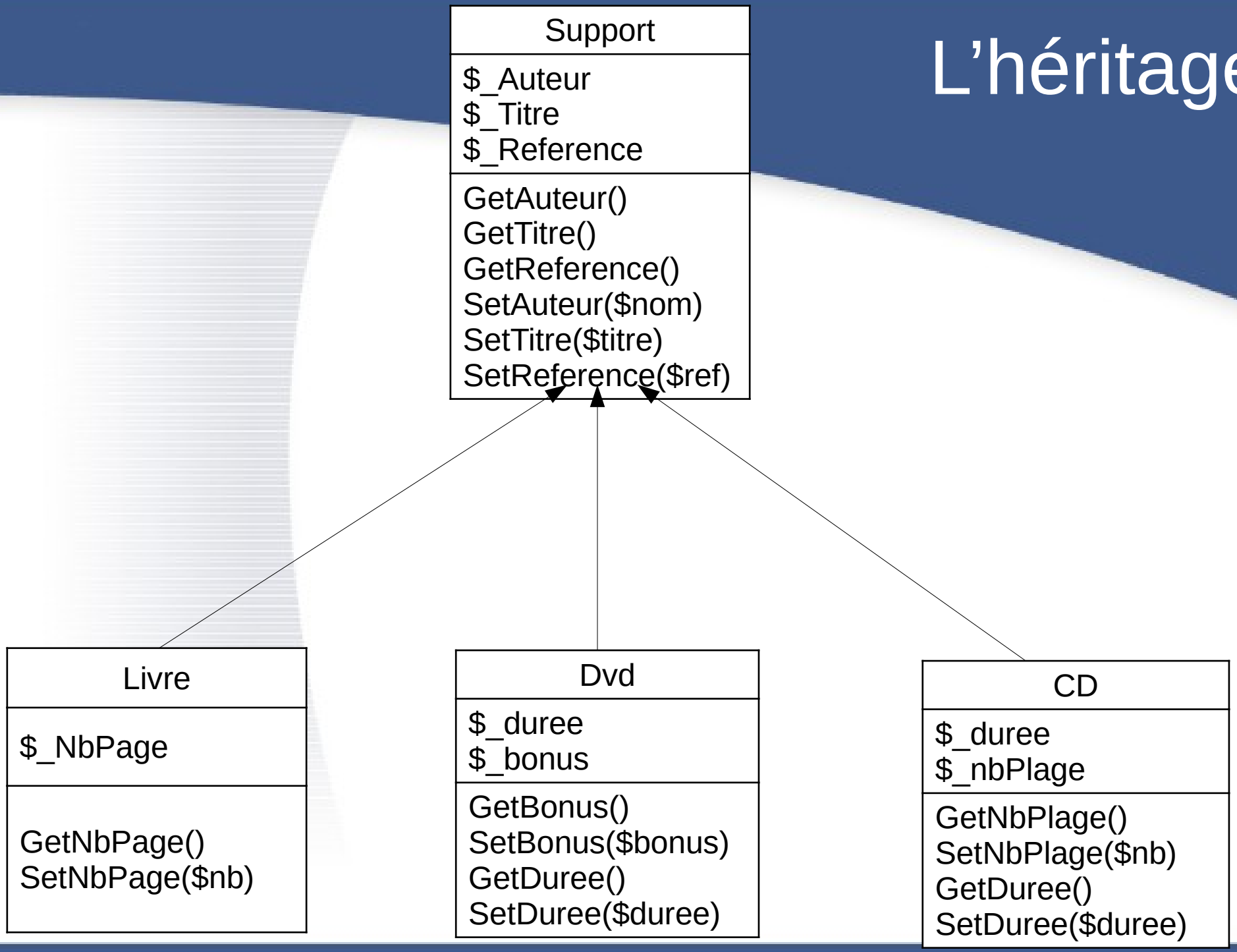
Inutile de coder le contenu des fonctions mais mettre en commentaire se que fait la fonction, et ce qu'elle renvoie si elle renvoie quelque chose

L’héritage

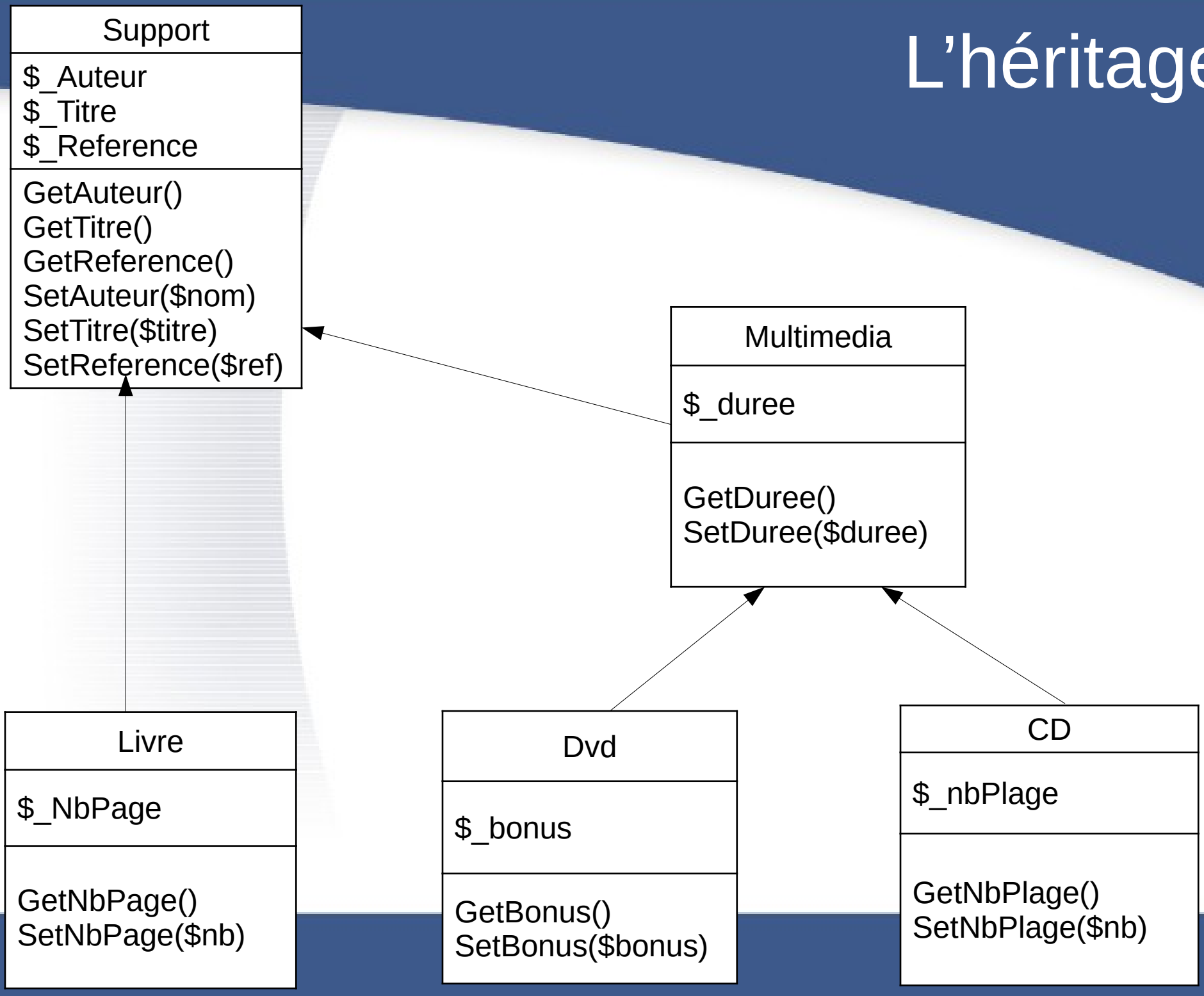
Selon Wikipedia :

En programmation orientée objet, l’héritage est un mécanisme qui permet, lors de la déclaration d’une nouvelle classe, d'y inclure les caractéristiques d’une autre classe.

L’héritage


Support
$_Auteur
$_Titre
$_Reference
GetAuteur()
GetTitre()
GetReference()
SetAuteur($nom)
SetTitre($titre)
SetReference($ref)
Livre
$_NbPage
GetNbPage()
SetNbPage($nb)
Dvd
$_duree
$_bonus
GetBonus()
SetBonus($bonus)
GetDuree()
SetDuree($duree)
CD
$_duree
$_nbPlage
GetNbPlage()
SetNbPlage($nb)
GetDuree()
SetDuree($duree)

L’héritage


Support
$_Auteur
$_Titre
$_Reference
GetAuteur()
GetTitre()
GetReference()
SetAuteur($nom)
SetTitre($titre)
SetReference($ref)
Multimedia
$_duree
GetDuree()
SetDuree($duree)
Livre
$_NbPage
GetNbPage()
SetNbPage($nb)
Dvd
$_bonus
GetBonus()
SetBonus($bonus)
CD
$_nbPlage
GetNbPlage()
SetNbPlage($nb)

Un peu d'exercice

Dans une entreprise un employé est décrit comme ci-dessous. Un technicien est quant à lui caractérisé par un grade et une prime calculée en fonction du grade

Employé
Nom Age Salaire N°Secu
Augmentation(…) ; toString() ; afficher() ; calculeSalaire() ;

Technicien
Grade
Prime() ; toString() ; calculeSalaire() ;

Si grade= C alors Prime= 100
Si grade= B alors Prime= 200
Si grade= A alors Prime= 300

Écrire un programme qui saisie un employé puis un technicien et affiche leurs informations avant et après augmentation de leurs salaires.

Polymorphisme

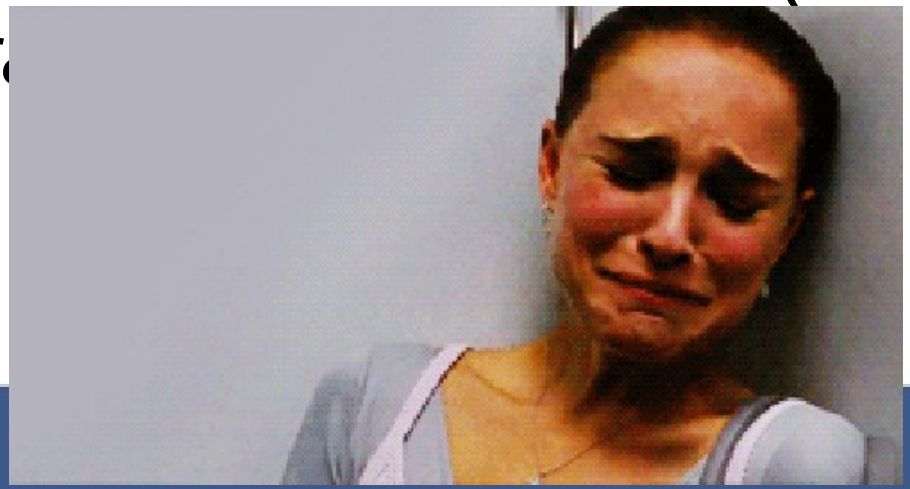
Dans le cadre d’un typage stricte comme en java:

Support[] tableau = new Support[4] ;
tableau[0] = new Livre(…) ;
tableau[1] = new Multimedia(…) ;
tableau[2] = new Dvd(…) ;
tableau[3] = new Cd(…) ;

Parlons statique

Selon Wikipedia :

Le typage statique est une technique utilisée dans certains langages de programmation impératifs pour associer à un symbole dénotant une variable le type de la valeur dénotée par la variable ; et dans certains langages de programmation fonctionnels pour associer à une fonction (un calcul) le type de son par[illegible] valeur calculée.



Côté static

Que ce soit un attribut ou une fonction, lorsqu'un élément est statique il se réfère à la classe et pas pas a son instance de classe, ou a son objet.

Exemple :

La classe Integer possède un attribut MAX_LENGTH qui détermine la taille maximum d'un nombre entier.

Ce n'est pas une propriété de l'objet 12 ou 67856 mais bien une caractéristique de la classe : un entier possède au maximum max_length chiffres et ce peu importe sa valeur.

C'est une valeur commune à tous les objets !

Exemple

Géométrie
PII surfaceCercle
calculerSurfaceCercle($rayon) calculerPerimetreCercle($rayon)

```php
class Geometrie
{
    const PII = 3.14; // --- Une constante
    public static $surfaceCercle; // --- Une propriété statique

    // --- Une méthode statique
    public static function calculerSurfaceCercle($rayon)
    {
        self::$surfaceCercle = $rayon * $rayon * self::PII;
    }

    // --- Une méthode statique
    public static function calculerPerimetreCercle($rayon)
    {
        return $rayon * 2 * self::PII;
    }
}


echo "<br />PII (Constante) : ", Geometrie::PII;

echo "<br />Perimetre (Methode statique) : ", Geometrie::calculerPerimetreCercle(10);

$rayon = rand ( 1 , 10 );

Geometrie::calculerSurfaceCercle($rayon);

echo "<br />Surface (Attribut statique) pour un rayon de ".$rayon.": ", Geometrie::$surfaceCercle;
```

A vous de jouer

Vous allez créer une classe voiture contenant

- Un attribut statique : nbInstance
- Une méthode statique : getNbInstance

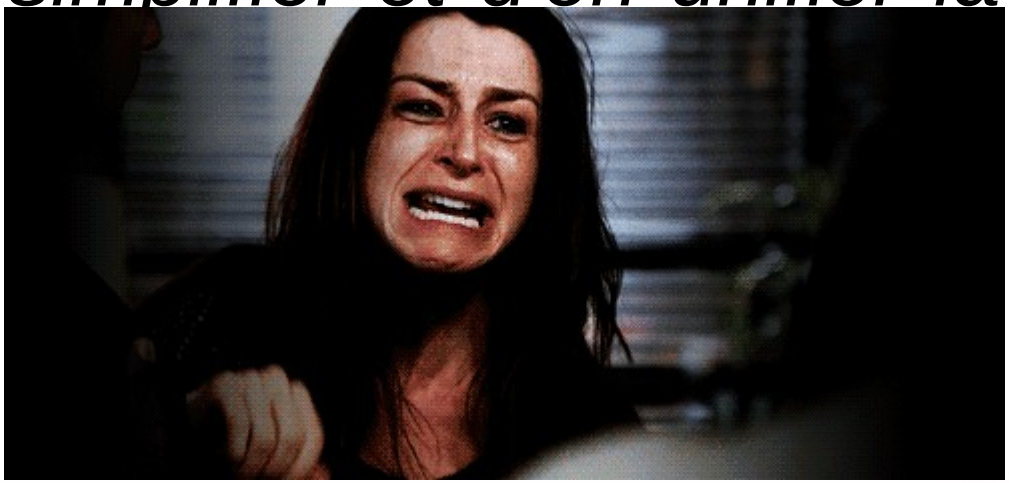
La classe devra avoir au moins 2 attributs non-statiques (ex : marque et modèle) ainsi que leurs accesseurs.
Vous devrez également créer 2 méthodes non-statiques autres que les accesseurs (à vous d'être imaginatif)

Lorsque la fonction getNbInstance() sera appelée, elle devra afficher le nombre d'objets préalablement créé !

Un peu d’abstraction

Selon Wikipedia :

En informatique, le concept d'abstraction identifie et regroupe des caractéristiques et traitements communs applicables à des entités ou concepts variés ; une représentation abstraite commune de tels objets permet d'en simplifier et d'en unifier la manipulation.

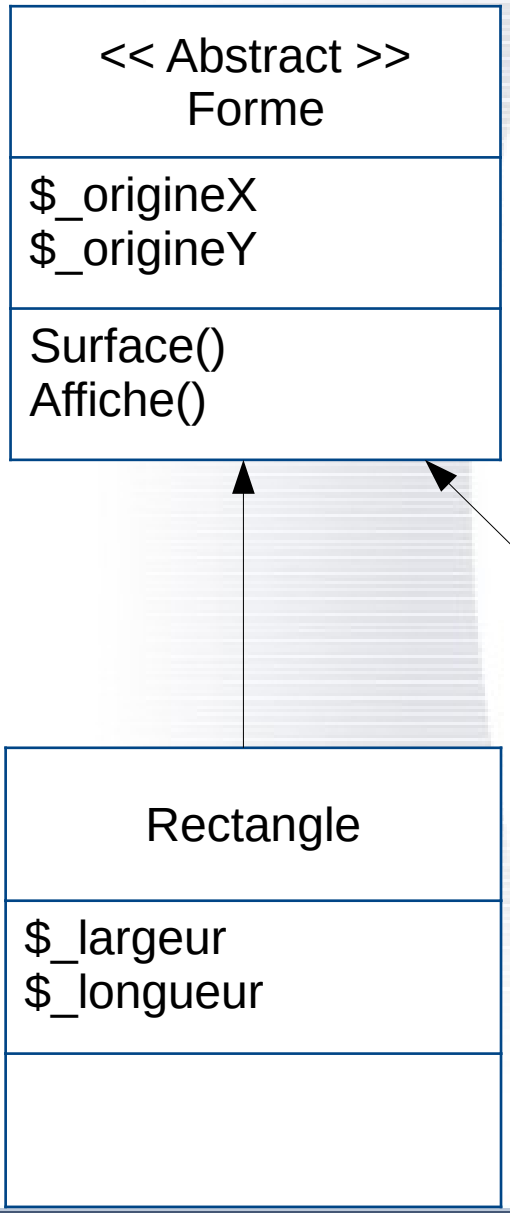


Et en clair

Il s’agit d’une classe presque comme les autres mais ne représentant rien de concret. D’un point de vue technique une classe abstraite ne s’instancie pas.

Elles ont pour objectif de servir de classe mère avec pour but l’uniformisation des traitements de toutes les classes qui en hérite.

Abstract




```php
abstract class Forme {
    protected $_origineX;
    protected $_origineY;

    public abstract function Surface();
    public abstract function Affiche();

    public function __get($attr){
        return $this->$attr;
    }

    public function __set($val, $attr){
        $this->$attr = $val;
    }
}
```

```php
class Rectangle extends Forme
{
    private $_largeur, $_longueur;

    function __construct($x, $y, $larg, $long)
    {
        # code...
        $this->_origineX = $x;
        $this->_origineY = $y;
        $this->_largeur = $larg;
        $this->_longueur = $long;
    }

    public function Surface(){
        return $this->_largeur * $this->_longueur;
    }
    public function Affiche(){
        echo "Le rectangle fait ".$this->_largeur." * ".$this->_longueur." cm ";
        echo "(soit une surface de ".$this->Surface()." cm²) et est positionné ";
        echo "aux coordonées ".$this->_origineX.":".$this->_origineY."<hr />";
    }
}
```

```php
*/
class Cercle extends Forme
{
    private $_rayon;

    function __construct($x, $y, $rayon)
    {
        # code...
        $this->_origineX = $x;
        $this->_origineY = $y;
        $this->_rayon = $rayon;
    }

    public function Surface(){
        return round(M_PI * pow($this->_rayon,2),2);
    }
    public function Affiche(){
        echo "Le cercle a un rayon de ".$this->_rayon." cm (soit une surface de ";
        echo $this->Surface()."cm²) et a pour centre : ";
        echo $this->_origineX.":".$this->_origineY."<hr />";
    }
}
```

Un peu d’interface

Selon Wikipedia :

Une interface définit la frontière de communication entre deux entités, comme des éléments de logiciel, des composants de matériel informatique, ou un utilisateur. Elle se réfère généralement à une image abstraite qu'une entité fournit d'elle-même à l'extérieur. Cela permet de distinguer les méthodes de communication avec l'extérieur et les opérations internes, et autorise à modifier les opérations internes sans affecter la façon dont les entités externes interagissent avec elle, en même temps qu'elle en fournit des abstractions multiples. On appelle aussi interfaces des dispositifs fournissant un moyen de traduction entre des entités qui n'utilisent pas le même langage, comme entre un être humain et un ordinateur. Étant donné que ces interfaces réalisent des traductions et des adaptations, elles entraînent des coûts de développement supplémentaires par rapport à des communications directes.

Interfaçons

Une interface décrit un ensemble de signatures de méthodes, sans implémentation, qui doivent être implémentées dans toutes les classes qui implémentent l'interface.

Et voici quelques interfaces intégrées directement dans php :

SeekableIterator :

- **current**: renvoie l'élément courant ;
- **key**: retourne la clé de l'élément courant ;
- **next**: déplace le pointeur sur l'élément suivant ;
- **rewind**: remet le pointeur sur le premier élément ;
- **valid**: vérifie si la position courante est valide ;
- **Seek** : permet de placer le curseur interne à une position précise

ArrayAccess

- **offsetExists**: méthode qui vérifiera l'existence de la clé entre crochets lorsque l'objet est passé à la fonction *isset* ou *empty* (cette valeur entre crochet est passée à la méthode en paramètre)
- **offsetGet**: méthode appelée lorsqu'on fait un simple $obj['clé']. La valeur 'clé' est donc passée à la méthode *offsetGet*;
- **offsetSet**: méthode appelée lorsqu'on assigne une valeur à une entrée. Cette méthode reçoit donc deux arguments, la valeur de la clé et la valeur qu'on veut lui assigner.
- **offsetUnset**: méthode appelée lorsqu'on appelle la fonction *unset* sur l'objet avec une valeur entre crochets. Cette méthode reçoit un argument, la valeur qui est mise entre les crochets.

Interfaçons

L’utilité du concept d’interface réside dans le regroupement de plusieurs classes, tel que chacune implémente un ensemble commun de méthodes, sous un même type. Une interface possède les caractéristiques suivantes :

- elle contient des signatures de méthodes ;
- elle ne peut pas contenir de variables ;
- une interface peut hériter d’une autre interface (avec le mot-clé *extends*) ;
- une classe (abstraite ou non) peut implémenter plusieurs interfaces. La liste des interfaces implémentées doit alors figurer après le mot-clé *implements* placé dans la déclaration de classe, en séparant chaque interface par une virgule.

Interfaçons

Fatal error: Class MaClass contains 1 abstract method and must therefore be declared abstract or implement the remaining methods (MonInterface::fonction) in **/var/www /html/PHP/POO/interface.php** on line **10**

```php
<?php
interface MonInterface
{
  public function fonction($arg);
}

class MaClass implements MonInterface
{
    public function fonction($arg){
        return $arg;
    }
}
```

A moi de rire
Évaluation

Vous aller créer une boutique en ligne simplifiée.

Le contexte :

- Votre client : une clinique vétérinaire « chez Pupuce »
- Le besoin : la clinique a besoin d'une visibilité sur internet pour vendre ses produits pour animaux. Il peux y avoir de la nourriture, des jouets, des médicaments.
- Le personnel de la clinique veux pouvoir TOUT modifier dans la boutique sauf le design
- Aucun CMS n'est autorisé
- Travail individuel
- Diagramme de classe fournis
- Les suggestions d'améliorations sont les bienvenues

Diagramme de Classe