E-COD
Design digital
CCI BORDEAUX GIRONDE

Programmation Orientée Objet

Stéphane Verdy

Version du 11/01/2020

php
POO

Plan

1. Introduction à la POO
2. Les éléments de base
3. Les éléments avancés
4. ...

Règles en vigueur lors des cours

Le respect : du prof (et oui je pense à moi) et des autres, cela passe par :

- Respecter les horaires, faire son possible pour être à l'heure (à l'impossible nul n'est tenu bien sur).
- Respecter la parole des autres
- Respecter les opinions des autres

La politesse : Pas de place pour des comportements insultants ou vulgaires

La tolérance :

- On n'a pas tous les mêmes forces mais on a tous besoin d'apprendre des autres
- Toutes les questions que vous avez doivent être posées… ne repartez pas avec !

Introduction à la POO(*)

1. Historique
2. Il était une fois le procédural
3. Naissance de la POO
4. La POO… mais pourquoi ?
   a. Avantages de la POO
   b. Inconvénients
5. UML

(*) : Dans ce cours nous utiliserons POO pour Programmation Orientée Objet

Historique

- La notion d'objet a été introduite avec le langage de programmation ***Simula***, créé à Oslo entre 1962 et 1967 dans le but de faciliter la programmation de logiciels de simulation. Avec ce langage de programmation, les caractéristiques et les comportements des objets à simuler sont décrits dans le code source.
- Le langage de programmation orientée objet ***Smalltalk*** a été créé par le centre de recherche Xerox en 1972.
- La programmation orientée objet est devenue populaire en 1983 avec la sortie du langage de programmation **C++**, un langage orienté objet, dont l'utilisation ressemble volontairement au populaire langage C. Puis en 1995 naissance de **Java** et **Javascript**.

Il était une fois le procédural

Commençons ce cours par une question :

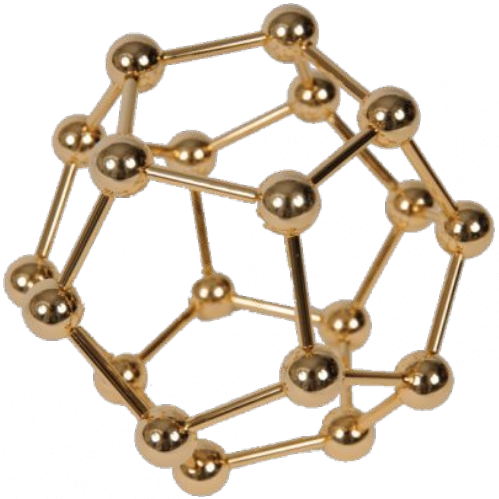
Qu’elle type de programmation utilisez vous ?

La réponse est généralement :

Une programmation de type « procédurale », qui consiste à séparer le traitement des données des données elles-mêmes.

Imaginons par exemple que vous avez un système de news sur votre site. D’un côté, vous avez les données (les news, une liste d’erreurs, une connexion à la BDD, etc.), et de l’autre côté, vous avez une suite d’instructions qui viennent accéder/modifier ces données.

Naissance de la POO



L’idée principale de la POO est que l’unité de base de la structure de mon programme c’est l’objet. En d’autres termes, tout est objet.

Un objet c’est l’association dans un même éléments syntaxique des traitements (le comportement) et des données (valeur des propriétés de l’objet).

La force de la programmation objet, c’est qu’elle s’appuie sur un modèle calqué sur la réalité physique du monde. Les objets se comportent comme des entités indépendantes, autosuffisantes qui collaborent par échange de messages.

La POO... pourquoi ?

Le problème qui se pose quand nous avons besoin d'élaborer des programmes complexes, c'est que nous sommes très vite confrontés aux problèmes suivants :

- comment comprendre et réutiliser les programmes faits par d'autres ?
- comment "cloner" rapidement des programmes déjà faits pour des applications légèrement différentes ?
- comment programmer simplement des actions simples sur des éléments variés et complexes ?
- comment ajouter des fonctions sans tout réécrire et tout retester ?

Avantages de la POO (1/2)

L'orienté objet remplace le procédural dans les grands programmes car il présente de multiples avantages:

- Facilité de compréhension (Permet de regrouper toutes informations sur un objet dans le code : si mon programme gère des voitures, l'objet du même nom contiendra la marque de celle-ci, sa vitesse ainsi que sa couleur, etc...)
- L'encapsulation (la programmation orientée objet permet la protection de l'information contenue dans un objet. Les données ne sont pas accessibles directement par l'utilisateur, celui-ci devant passer par les méthodes publiques.).

Avantages de la POO (2/2)

- La modularité du code (On peut généralement récupérer 80 % du code d'un projet pour le réutiliser sur un projet similaire contrairement à la programmation procédurale. De plus les objets permettent d'éviter la création de code redondant, permettant donc un gain de temps et donc

Inconvénients de la POO

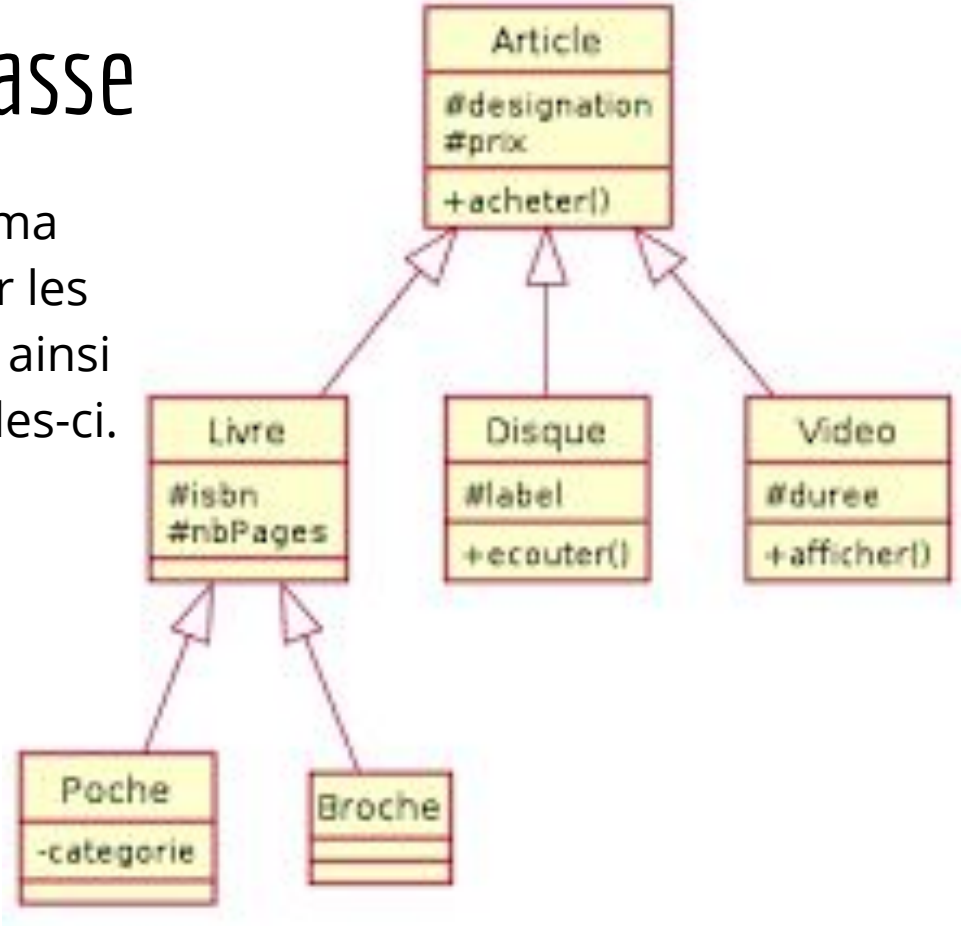
L’approche objet est moins intuitive que l'approche fonctionnelle. Malgré les apparences, il est plus naturel pour l'esprit humain de décomposer un problème informatique sous forme d'une hiérarchie de fonctions et de données, qu'en termes d'objets et d'interaction entre ces objets.

UML pour venir au secour des concepteurs

UML (en anglais Unified Modeling Language ou « langage de modélisation unifié ») est un langage de modélisation graphique à base de pictogrammes. Il est apparu dans le monde du génie logiciel, dans le cadre de la « conception orientée objet ». Couramment utilisé dans les projets logiciels, il peut être appliqué à toutes sortes de systèmes ne se limitant pas au domaine informatique.

UML - Diagramme de Classe

Le diagramme de classes est un schéma utilisé en génie logiciel pour présenter les classes et les interfaces des systèmes ainsi que les différentes relations entre celles-ci.



UML - Diagramme de Séquence

Les diagrammes de séquences sont la représentation graphique des interactions entre les acteurs et le système selon un ordre chronologique.

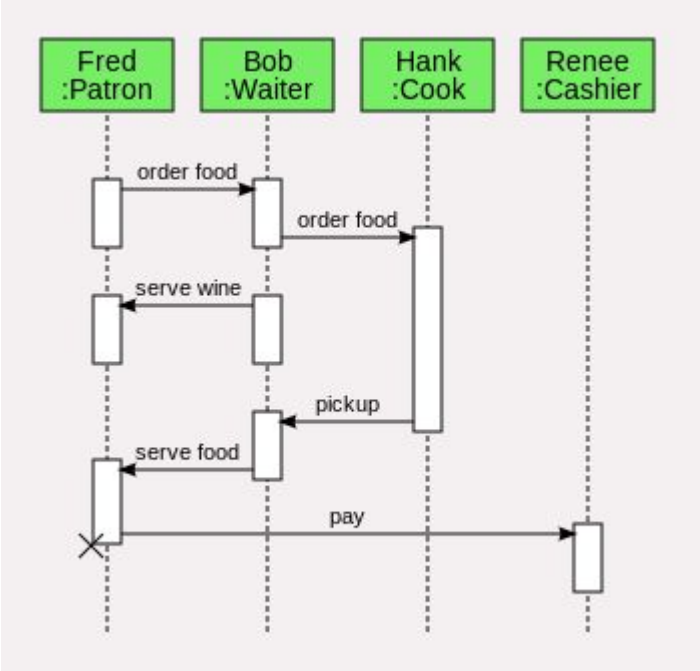
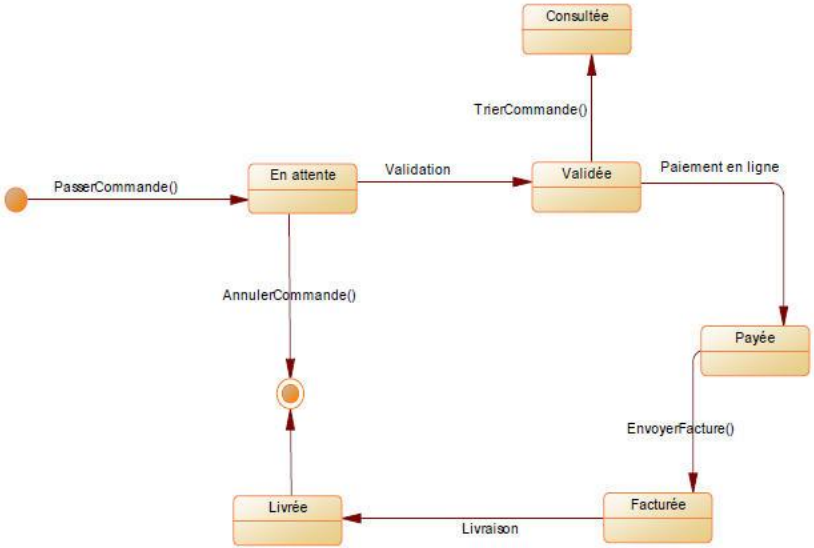


Diagramme États/Transitions

Les diagrammes d'états-transitions d'UML décrivent le comportement interne d'un objet à l'aide d'un automate à états finis. Ils présentent les séquences possibles d'états et d'actions qu'une instance de classe peut traiter au cours de son cycle de vie en réaction à des événements discrets (de type signaux, invocations de méthode).



Les éléments de base

1. Définition d’un Objet
2. Définition d’une Classe
3. Définition d’une Instance
4. Définition d’attribut
5. Définition de méthode
6. Le principe d’encapsulation
7. TP : Créons notre première classe
    a. La Classe Voiture
    b. Jouons aux petites voitures

Définition d’un objet

Un objet est un ***conteneur symbolique***, qui possède sa propre existence et englobe des caractéristiques, états et comportements et qui, par métaphore, représente quelque chose de tangible du monde réel manipulé par informatique.

En programmation orientée objet, un objet est créé à partir d'un modèle appelé ***classe***, duquel il hérite les comportements et les caractéristiques. Les comportements et les caractéristiques sont typiquement basés sur celles propres aux choses qui ont inspiré l'objet : une personne, un dossier, un produit, une lampe, une chaise, un bureau.

Définition d’une classe

Les classes sont présentes pour « fabriquer » des objets. En programmation orientée objet, un objet est créé sur le modèle de la classe à laquelle il appartient.

Exemple simple: les gâteaux et leur moule. Le moule, il est unique. Il peut produire une quantité infinie de gâteaux. Dans ces cas-là, les gâteaux sont des instances d’objets et le moule est la classe.

Définition d'instance

Une instance, c'est tout simplement le résultat d'une instanciation. Une instanciation, c'est le fait d'instancier une classe. Instancier une classe, c'est se servir d'une classe afin qu'elle crée un objet. En gros, une instance est un objet.

En php comme dans la plupart des langage objet c'est le mots clef "new" qui est utilisé pour instancier un nouvel objet. « new » peut être traduit comme « fabrique-moi un nouvel objet émanant de cette classe ».

Par analogie à l'exemple des cannelés, le moule étant la "classe" chacun des cannelés est une "instance" de cannelé.

Définition d’attribut

On a vu qu’un objet est composé de propriété et de comportement.

Les “attributs” d’un objet sont ces propriétés, on peut retrouver ici la notion de variables informatique. Il servent à mémoriser l’état de l’objet, c’est à dire l’ensemble des valeurs de ces propriété à un instant donnée.

Il est à noter que dans la documentation de référence du langage Php les “attributs” sont nommés “propriétés”.

Visibilité des attributs

- **Public :** Permet l'accès universel à la propriété, aussi bien dans la classe que dans tout le script, y compris pour les classes dérivées, comme vous l'avez vu jusqu'à présent

- **Protected :** La propriété n'est accessible que dans la classe qui l'a créée et dans ses classes dérivées

- **Private :** C'est l'option la plus stricte : l'accès à la propriété n'est possible que dans la classe et nulle part ailleurs

Définition de méthode

Les méthodes des objets sont des sorte de fonctions informatique qui sont définies uniquement dans les classes des objets et qui permette de donner du “comportement” au objets instanciés.

Par exemple : si j’ai créer une classe “Compte Bancaire” je retrouverais sans doute les méthode “débiter” et “créditer” pour pouvoir agir sur l’attribut “solde” de mon compte.

Visibilité d’une méthode

- Public : La méthode est utilisable par tous les objets et instances de la classe et de ses classes dérivées

- Protected : La méthode est utilisable dans sa classe et dans ses classes dérivées, mais par aucun objet

- Private : La méthode n’est utilisable que dans la classe qui la contient, donc ni dans les classes dérivées, ni par aucun objet

Tout appel d’une méthode en dehors de son champ de visibilité provoque une erreur fatale

Le principe d'encapsulation

L'un des gros avantages de la POO est qu'on peut masquer le code à l'utilisateur (l'utilisateur est ici celui qui se servira de la classe, pas celui qui chargera la page depuis son navigateur). Le concepteur de la classe a englobé dans celle-ci un code qui peut être assez complexe et il est donc inutile, voire dangereux, de laisser l'utilisateur manipuler ces objets sans aucune restriction. Ainsi, il est important d'interdire à l'utilisateur de modifier directement les attributs d'un objet.

Un des premiers réflexe à avoir est de limiter au maximum les attributs de visibilité "public" et de forcer l'utilisateur de notre classe à passer par l'appel de setter et/ou de getter pour agir sur les données de ces objets.

Une méthode particulière : le constructeur

Parmis les méthodes que vous trouverez il en existe des spéciales comme notamment le constructeur. C'est la méthode utilisée par le programme pour construire une instance de votre objet.

Il en existe une par défaut mais vous pouvez en créer une ou plusieurs en fonction de vos besoins.

Il existe le pendant à cette méthode, c'est le destructeur. Moins utilisé il permet de faire des traitements à la suppression de l'instance de l'objet. Par exemple on pourrait clore les connection à un pool de connexion à une base de donnée.

TP : Notre première classe...Voiture

Une enveloppe vide...

Nous allons donc créer un fichier VoitureVide.php à la racine de notre site web comme ceci :

Une page pour afficher notre voiture

Créer un fichier exemple1.php tel que :

exemple1.php

```php
<?php
    require_once 'VoitureVide.php';
    $voiture1 = new Voiture();
?>
<html>
    <body>
        <h1>Ma belle voiture : </h1>
        <pre>
            <?php var_dump($voiture1); ?>
        </pre>
    </body>
</html>
```

Affichage de la page exemple1.php

Aller un peux plus loin...

Avoir une Voiture qui n’a aucune propriétés n’est pas très utile..

Ajouter à notre voiture les propriétés suivantes :

- Une couleur
- Un nombre de porte
- Une puissance
- Un type de carburant

Voiture ++

Nous avons donc modifié la classe Voiture de tel sorte :

```
class Voiture {
    private $_puissance = 0;
    private $_carburant = "Gasoil";
    private $_nbPorte = 4;
    private $_couleur = "bleu";
}
```

Ce qui donne en affichage

```
class Voiture#1 (4) {
  private $_puissance =>
  int(0)
  private $_carburant =>
  string(6) "Gasoil"
  private $_nbPorte =>
  int(4)
  private $_couleur =>
  string(4) "bleu"
}
```

Challenge suivant

Pourriez-vous créer trois voitures avec des couleurs différentes, des puissance différentes, des nombres de portes différents et des carburants différents ?

Solution 1

Il faut rajouter une constructeur qui prend en paramètres les différentes valeurs pour les attributs de la voiture.

```php
public function __construct( $puissance, $carburant, $nbPorte, $couleur) {
    $this->_puissance = $puissance;
    $this->_carburant = $carburant;
    $this->_nbPorte = $nbPorte;
    $this->_couleur = $couleur;
}
```

Solution 2

Ajouter les méthodes permettant de changer la valeur des attributs de la voiture une fois instancié. On appel ces méthodes des “setteurs”.

```php
public function setPuissance( $puissance ) { $this->_puissance = $puissance; }
public function setCarburant( $carburant ) { $this->_carburant = $carburant; }
public function setNbPorte( $nbPorte ) { $this->_nbPorte = $nbPorte; }
public function setCouleur( $couleur ) { $this->_couleur = $couleur; }
```

Illustration de la visibilité des attributs

Voici un exemple pour illustrer la visibilité des attributs :

Fichier : VisibiliteAttributs.php

Illustration de la visibilité des méthodes

Voici un exemple pour illustrer la visibilité des méthodes ::

Fichier : VisibiliteMethodes.php

Les éléments avancés

1. Les éléments statiques
    a. Les constantes
    b. Les attributs
    c. Les méthodes
    d. L’opérateur de résolution de portée
    e. les petits mots “self” et “$this”
2. L’héritage
3. Les classes abstraites
4. Les classes finales
5. Les interfaces

Les éléments “statiques”

Les éléments dit “statiques” sont des éléments rattachés non pas à des instances d’objets mais directement aux classes. Là ou des éléments “standards” nécessitent l’existence d’une instance d’objet qui les supporte, les éléments “statiques” existent simplement par leur définition dans une classe d’objet.

En tant qu'éléments statiques nous trouvons les constantes de classe, les attributs et les méthodes statiques.

Les constantes de classes

Une constante est une sorte d’attribut appartenant à la classe, dont la valeur ne change jamais.

Elle est donc valorisé dès sa déclaration.

```
const GASOIL = "Gasoil";
const ELECTRIQUE = "Electrique";
const E10 = "E10";
```

Les attributs statiques

Se sont des attributs qui sont définis dans la classe, ils sont donc partagés par toutes les instances des objets de cette classe. Ils sont même accessible hors toute instance.

En PHP on les définit par le mot clef “statique” comme suit :

```
private static $_nbVoiture = 0;
```

Les méthodes statiques

Les méthodes “statiques” sont à l’image des attributs “statiques” des méthodes qui sont portées par la classe. Elles existent en dehors de toute instances d’objet et peuvent accéder seulement aux constantes et attributs statiques de la classe.

En PHP on les déclares comme suit :

```php
public static function getNombreVoiture() {
    return self::$_nbVoiture;
}
```

L’opérateur de résolution de portée

Problème comment référencer (accéder) aux constantes, attributs et méthodes statiques hors de tout instance de la classe … ??

On ne peut en effet pas écrire NomClasse->$_NomAttributStatique, ou NomClasse->MethodeStatique()… l’opérateur “->” ne fonctionne qu’avec des instance d’objet !

Pour palier à ça un nouvel opérateur doit être utilisé, c’est le “::” (Opérateur de résolution de portée). on peut donc écrire :

```
echo "<li>", Voiture::GASOIL, "</li>";
echo "<li>", Voiture::ELECTRIQUE, "</li>";
echo "<li>", Voiture::E10, "</li>";
```

les petits mots “self” et “$this”

Comme vous l’avez vu dans les slides précédents nous avons souvent besoin des deux mots clefs “self” et “$this”...

- le mots clef “self”
  - Représente la classe courante au sein de cette dernière
  - Est utilisé en combinaisons de l’opérateur “::”

- le mots clef “$this”
  - Représente l’instance courante au sein du code de la classe
  - Est utilisé en combinaisons de l’opérateur “->”

Challenge suivant

Pourriez-vous ajouter à notre classe Voiture :

1. Les constantes définissant les carburants
2. Un moyen de savoir combien de voiture ont été instanciées

L’héritage (1/4)

L’héritage est un concept puissant de la programmation orientée objet, permettant entre autres la réutilisation (décomposition du système en composants) et l'adaptabilité des objets grâce au polymorphisme. Elle se nomme ainsi car le principe est en quelque sorte le même que celui d’un arbre généalogique. Ce principe est basé sur des classes dont les « filles » héritent des caractéristiques de leur(s) « mère(s) ».

Chaque classe possède des caractéristiques (attributs et méthodes) qui lui sont propres. Lorsqu'une classe fille hérite d'une classe mère, elle peut alors utiliser ses caractéristiques.

L’héritage (2/4)

Quand nous parlons d'héritage, l’exemple type est le suivant :

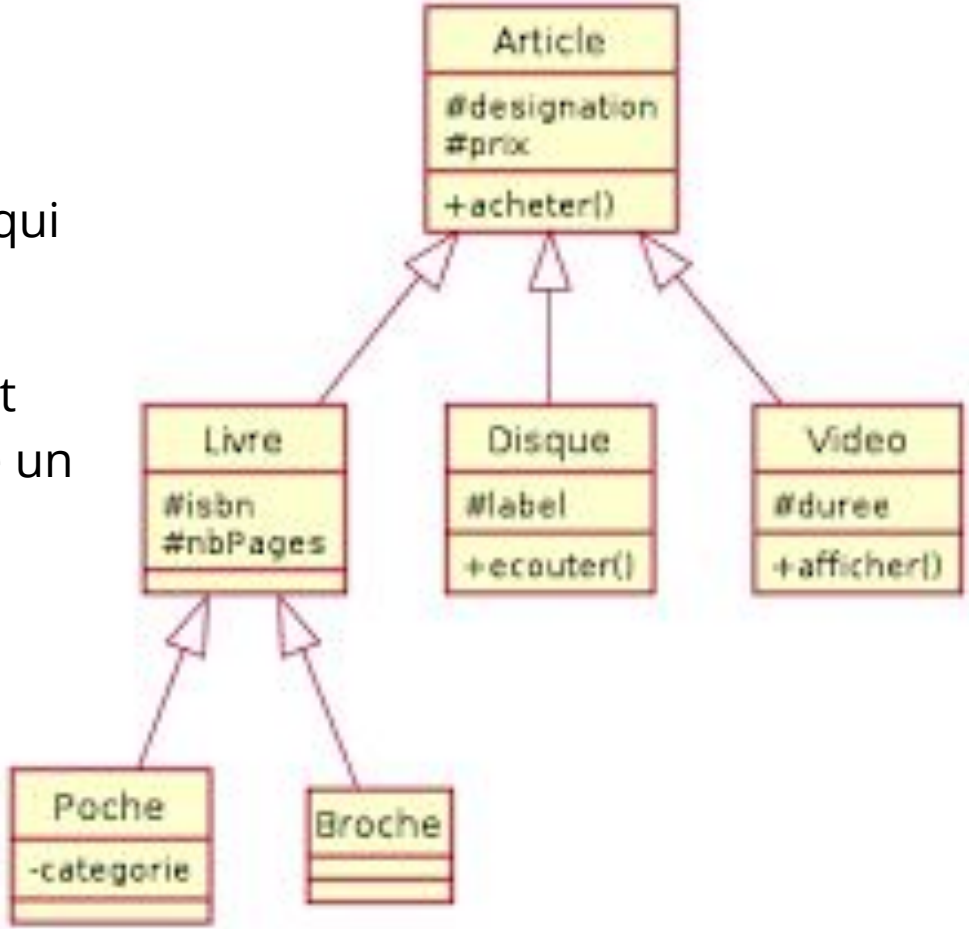
Lorsqu'une classe B hérite d'une classe A. La classe A est donc considérée comme la classe mère et la classe B est considérée comme la classe fille.

La classe B hérite de tous les attributs et méthodes de la classe A. Si nous déclarons des méthodes dans la classe A, et que nous créons une instance de la classe B, alors nous pourrons appeler n'importe quelle méthode déclarée dans la classe A, du moment qu'elle est publique ou protégée.

L’héritage (3/4)

Ici un Poche est un Livre qui est lui même un Article.

Donc un objet Poche peut tout à fait être vu comme un “Article”, donc on peux connaitre son prix par exemple.



L’héritage (4/4)

En PHP on écrit :

```
class Voiture extends Vehicule {
```

Ici par exemple on défini la classe “Voiture” qui hérite de la classe “Vehicule”. En PHP l’héritage multiple n’est pas possible, on ne peut donc n’avoir qu’une seule classe mère.

Surcharge des méthodes

Si on définit dans une classe fille une méthode déjà présente dans la classe mère cette dernière est remplacée corps et bien. On appel ce mécanisme la “surcharge” des méthodes.

On peut compléter ce mécanisme en utilisant “parent::” pour dans la méthode fille exécuter quand même le code de la méthod de la classe mère. Ce qui vient à compléter le comportement de la mère par des actions spécifique à la classe fille !

Challenge suivant

Réorganisez la classe Voiture pour qu’elle soit la classe fille de Vehicule et définir la classe Camion qui elle aussi est une fille de Vehicule.

Les classes abstraites (1/2)

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 la manipulation.

Une classe abstraite n’est pas « instanciable » et ne pourra donc pas « fabriquer » d’objets.

Une classe abstraite définie partiellement un comportement et laisse aux classes filles le soin de compléter (spécialiser) les partie restantes pour en faire une classe complète et instanciable.

Les classes abstraites (2/2)

Imaginons que nous ayons une classe Animal et des classes : Chien, Chat, Loup, Eléphant, Chameau. Tous sont des animaux et héritent naturellement de la classe « Animal ».

Cependant aucun d'entre eux n'est « juste animal », un chien est un chien, et un chat est un chat. Par conséquent la classe Animal ne sera jamais « instanciée », la classe Animale sera abstraite.

Elle servira de classe modèle pour celles qui en hériteront. En effet chaque animal dort, chaque animal mange… Ces méthodes « dormir() » ou encore « manger() » seront relatives à la classe Animal.

En revanche chaque animal a un cri différent : le chien aboie, le chat miaule. Ces méthodes cri() seront dans les classes filles.

Les classes finales

Le concept des classes et méthodes finales est exactement l’inverse du concept d’abstraction. Si une classe est finale, vous ne pourrez pas créer de classe fille héritant de cette classe.

Pour ma part, je ne rends jamais mes classes finales (de même que, à quelques exceptions près, je mets toujours mes attributs en protected) pour me laisser la liberté d’hériter de n’importe quelle classe. Pour déclarer une classe finale, vous devez placer le mot-clé final juste avant le mot-clé class, comme abstract.

Les interfaces

Les interfaces sont des classes particulières qui autorisent d'autres classes à hériter d'un certain comportement, et qui ***définissent des méthodes à implémenter***. Cela vous permet de créer du code qui spécifie quelles méthodes une classe peut implémenter, sans avoir à définir comment ces méthodes seront gérées.

Les interfaces ***servent à créer des comportements génériques*** ; si plusieurs classes doivent obéir à un comportement particulier, nous créons une interface décrivant ce comportement, nous la faisons implémenter par les classes qui en ont besoin. Ces classes devront ainsi obéir strictement aux méthodes de l'interface (nombre, type et ordre des paramètres, type des exceptions), sans quoi la compilation ne se fera pas.

Différence entre extends et implements

Extends : Est utiliser pour définir un héritage entre une classe fille et sa classe mère.

```
class Voiture extends Vehicule {
```

Implements : Est utiliser pour définir que la classe respecte une ou plusieurs interfaces.

```
class Voiture implements Svgable, Sortable {
```

Challenge suivant

La on va frapper fort … Faite en sorte que les Camions et les Voitures aient une représentation graphique en SVG. Pour ce faire on définiera une interface rendant nos objets Svgable et on transformera la classe Vehicule en classe abstraite Implémentant l'interface Svgable…

C'est pas du challenge ça !!