Secteur Tertiaire Informatique

Filière « Etude et développement »

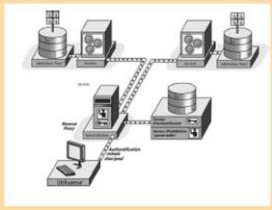
Séquence « Développer des pages Web en lien avec une base de données »

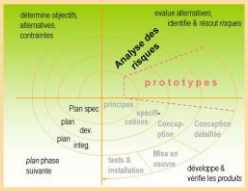
Le langage PHP orienté objet

Apprentissage

Mise en pratique

Evaluation





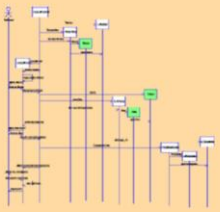


TABLE DES MATIERES

Objectifs

Ce document est une présentation des principes de base de la programmation orientée objet appliquée au langage PHP.

Il précise les concepts et la syntaxe PHP correspondante.

Pour une information plus complète sur les instructions et opérateurs cités, se reporter à la documentation de référence PHP en français sur http://php.net/manual/fr/language.oop5.php

Pré requis

Etre familier avec la programmation procédurale en PHP.

Outils de développement

Aucun.

Méthodologie

Mode d'emploi

Symboles utilisés :

Renvoie à des supports de cours, des livres ou à la documentation en ligne constructeur.

Propose des exercices ou des mises en situation pratiques.

Point important qui mérite d'être souligné !

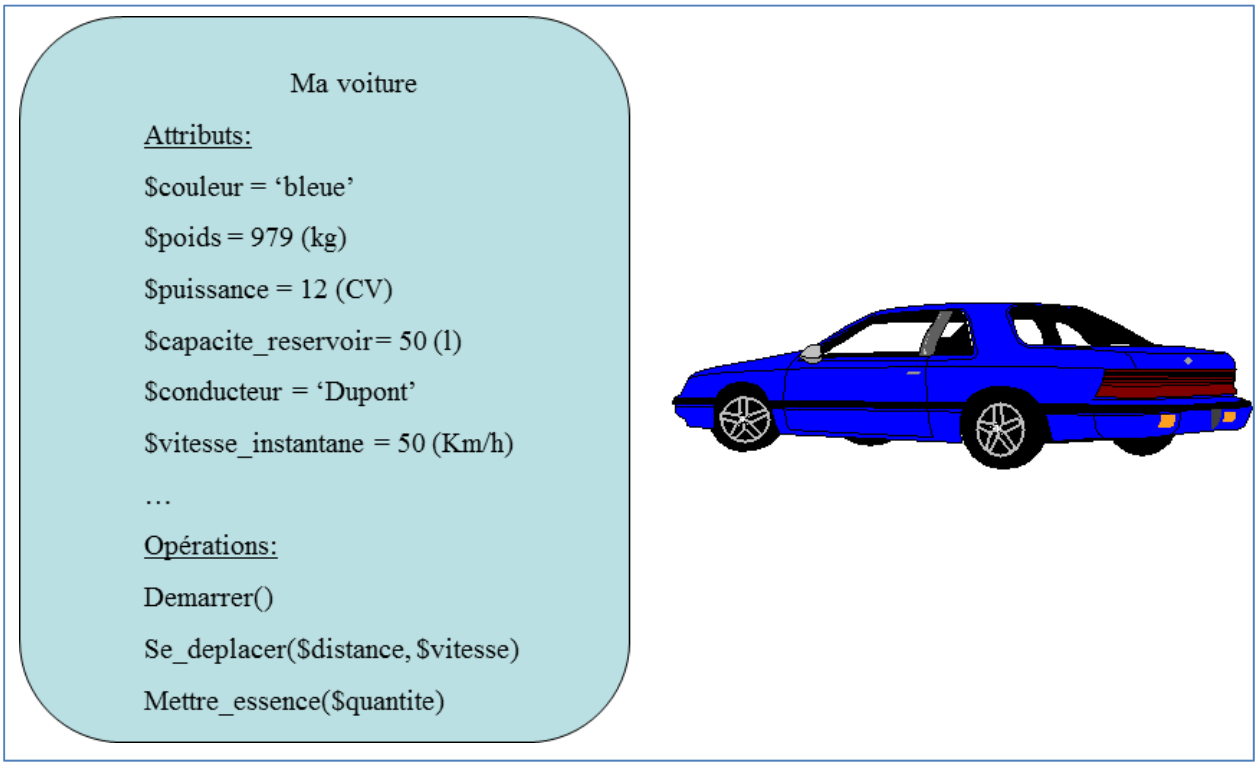
Ressources

Lectures conseillées

1. PROCEDURAL OU ORIENTE OBJET ?

A l'origine, les langages de programmation étaient de type 'procédural' en ce sens qu'un programme décrivait avant tout les opérations à réaliser et les ordonnait selon une procédure bien définie permettent de produire le résultat voulu en fonction des données en entrée. Il en résulte une certaine confusion dans les données dès lors que le programme devient un tant soit peu complexe.

Pour mieux coller 'au monde réel', l'orientation 'objet' est basée sur la définition et la manipulation d'ensembles cohérents et indissociables de données et de traitements sur ces données. Ainsi le développeur qui doit gérer des stagiaires en apprentissage peut définir des entités stagiaires, formateurs, cours, contrôles et autre examen qui, chacun, sont dotés de données appelées *attributs* ou *propriétés* (le nom du stagiaire, le nom du formateur…) et capables d'assurer des traitements élémentaires (appelés *opérations* ou *méthodes*) qui les concernent (un stagiaire peut recevoir une note ou fournir sa moyenne, un formateur peut assurer un cours…). C'est le principe de base **d'encapsulation**.



A partir de ces entités, le développeur pourra facilement en appeler/invoquer les services dans un programme principal qui se limitera à manipuler des objets.

Le principe de base de la programmation orientée objet est donc de constituer des **ensembles indissociables de données et traitements sur ces données** qui représentent le monde réel à gérer dans les applications.

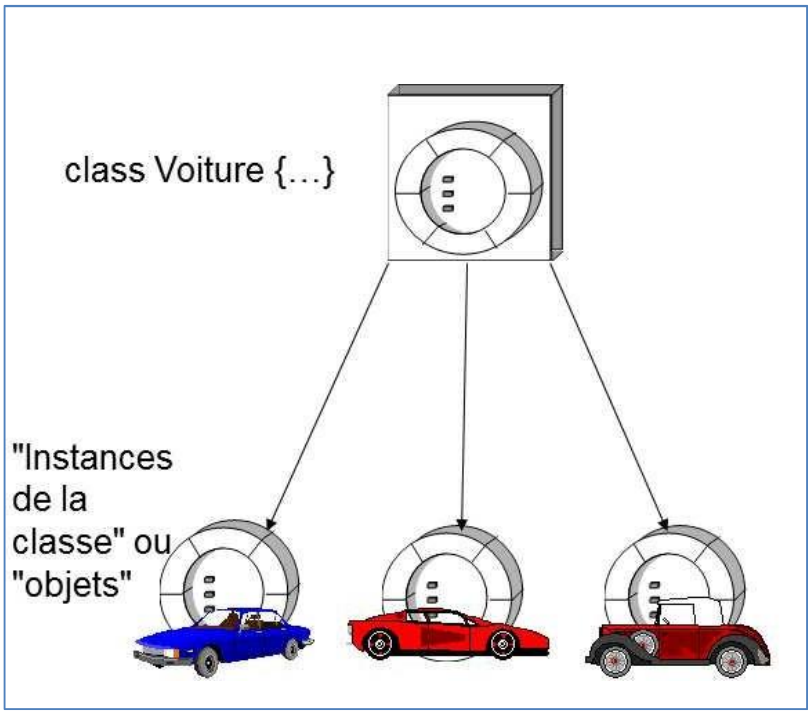
Un intérêt (qui devient une règle) est que certaines données et certains traitements d'un objet peuvent être privés, donc non-accessibles directement de l'extérieur.

Un objet doit être **responsable de lui-même**, **se protéger** contre des usages maladroits ou malveillants (on ne me change pas mon nom ni la couleur de mes yeux comme ça !) et **offrir des services**.

2. CLASSE OU OBJET ?

Une entité constituée de données et traitements sur ces données est décrite par une structure de programme spéciale, une classe. **Une classe décrit tous les membres de données et de traitements** qui constituent les caractéristiques de la classe.

Pour pouvoir manipuler un objet, le développeur doit lui donner vie ; c'est ce qu'on appelle **l'instanciation** des objets. Une classe est donc un programme qui sert de 'moule à fabriquer des objets' de même nature.



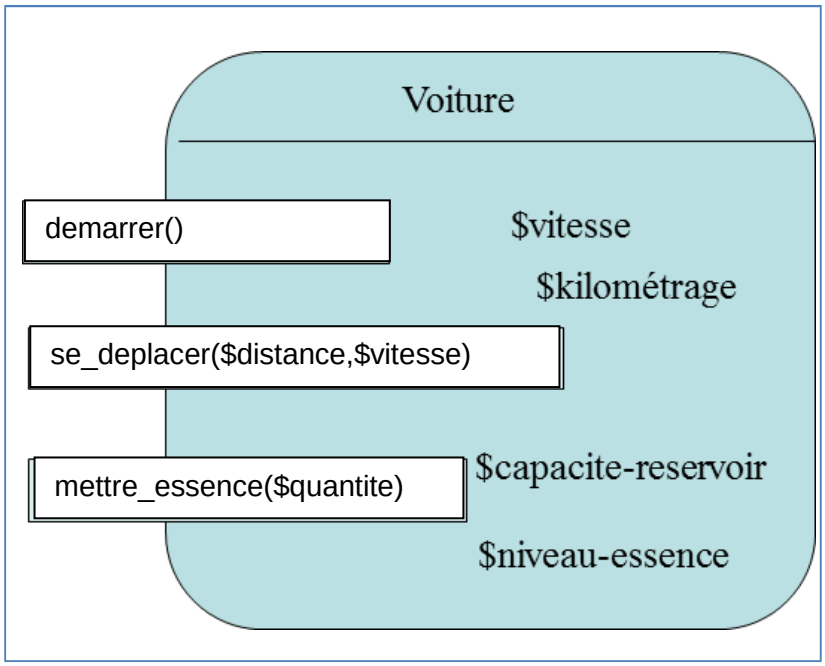
Il peut bien entendu exister de multiples objets, instances de la même classe (tout comme il existe plusieurs stagiaires, chacun étant doté de son propre nom, son prénom et tout le reste).

Chaque objet instancié est identifié par une donnée particulière attribuée par le système et que l'on appelle sa **référence**. Chaque objet connaît sa référence sous un nom symbolique et peut accéder à tous ses membres.

Le développeur écrit tout d'abord les programmes de classes puis il les utilise dans les programmes d'application en instanciant les objets dont il a besoin.

3. EXPOSITION (Ecapsulation)

Un objet n'expose pas tous ses membres à l'extérieur ; certains restent privés, d'autres sont exposés publiquement pour simple consultation, d'autres encore pour mise à jour. Les membres exposés constituent l'interface de l'objet avec le monde extérieur.



Les membres exposés sont en général matérialisés par des fonctions (on parle de '***méthodes'*** en programmation orientée objet) ; ils correspondent aux commandes par lesquelles le programme utilisateur de ces objets pourra les manipuler, tout comme les différentes commandes d'une voiture permettent au conducteur d'utiliser son véhicule sans en connaître les détails techniques, en consultant certaines informations mais sans pouvoir tout 'bidouiller' à son gré (et pourtant, qu'il serait doux de pouvoir modifier directement le niveau de carburant sans passer par la pompe à essence !).

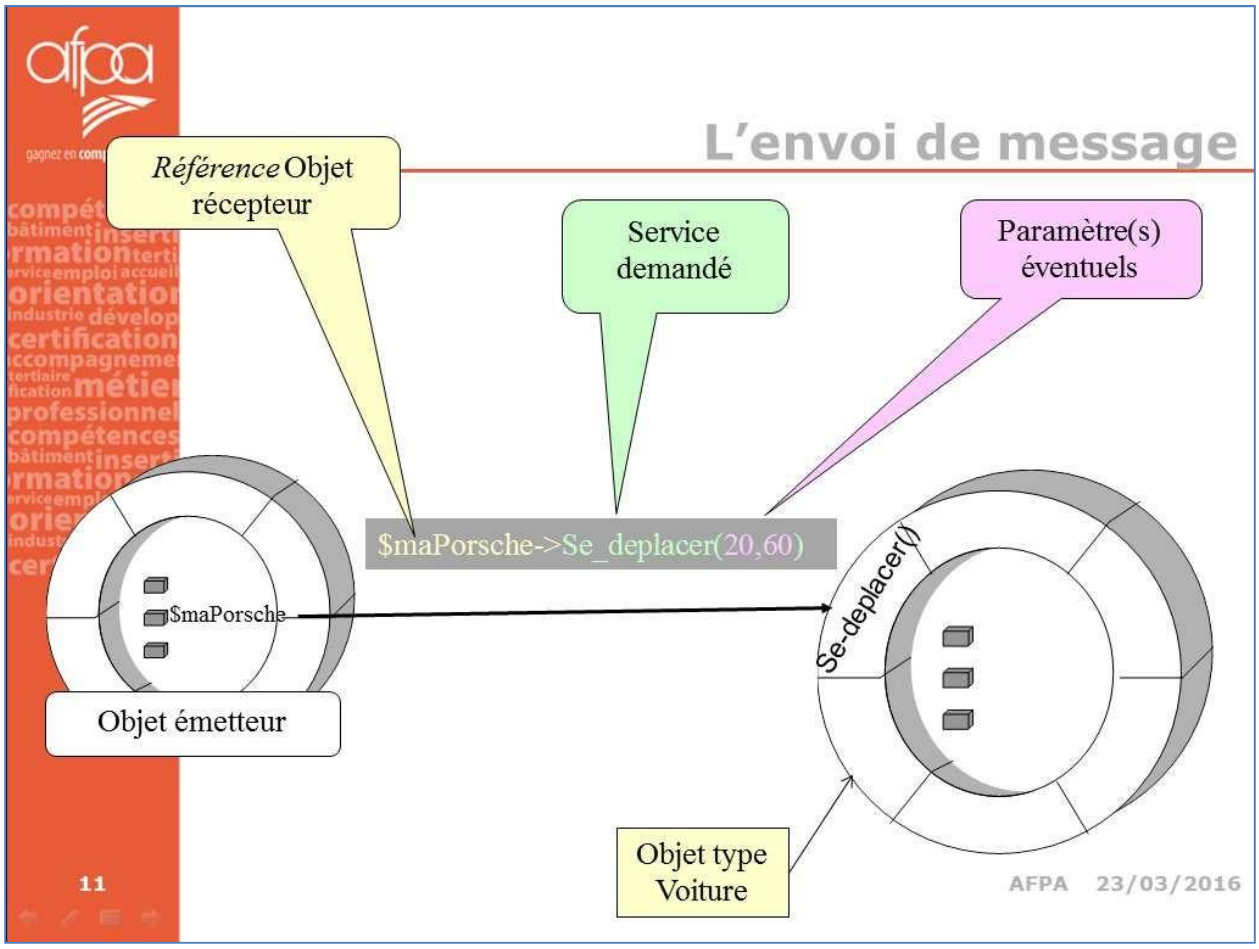
Les membres d'une classe sont donc dotés d'un **niveau d'exposition**, à la base **privé** ou **public**.

Un objet connaît et accède bien entendu à tous ses membres, qu'ils soient exposés ou non à l'extérieur.

Un objet connaît sa propre référence attribuée par le système lors de sa naissance, pardon, son instanciation ; en PHP il la connaît sous le nom symbolique `$this`.

4. MESSAGES ET APPEL DE METHODES

Un objet (ou un programme procédural classique) utilise un autre objet en faisant appel à l'une ou l'autre de ses méthodes exposées. Il devra préciser la **référence de l'objet** de destination, **la méthode appelée** et éventuellement **la valeur de certains paramètres** :



Les méthodes exposées par une classe sont principalement :

- Les constructeurs : fonctions appelées lors de l'instanciation et qui permettent éventuellement d'initialiser certaines données ;
- Les sélecteurs ou getters : fonctions qui permettent de consulter la valeur des données (exemple quel est la vitesse courante ?) ;
- Les modifieurs ou setters : fonctions qui permettent de modifier la valeur des données d'un objet (par exemple repeindre la carrosserie) ;
- Les services divers ou spécifiques au domaine modélisé (par exemple, démarrer ou se déplacer).

5. DECRIRE UNE CLASSE EN PHP

5.1 STRUCTURE DE CLASSE

Un script PHP décrit une classe grâce à la structure de bloc `class … {…}.`

Chaque classe doit porter un nom unique et tous ses membres sont déclarés à l'intérieur du bloc (entre les accolades).

Exemple :

```
class Voiture {…} ;
```

5.2 MEMBRES DE DONNEES

Les membres de données sont de simples variables exposées ou non et sont déclarés avec leur indicateur d'exposition `private` ou `public` (mas pas leur type puisque les données ne sont pas pré-typées en PHP).

Exemple :

```
class Voiture {
      private $couleur ;
      private $poids ;
      …
}
```

5.3 MEMBRES DE TRAITEMENT

Les membres de traitement (méthodes) sont déclarés comme des `function` dotées d'un indicateur d'exposition et d'un nom unique dans la classe suivi d'une paire de parenthèses contenant les noms des paramètres éventuels reçus, et enfin d'un bloc d'instructions définissant les actions à exécuter quand cette méthode est invoquée par un script PHP. Ces traitements peuvent bien entendu comporter des contrôles.

Exemple :

```
class Voiture {
      private $couleur ;
      …
      public function mettre_essence($quantite){
       if($quantite > ($this->capacite_reservoir - $this->niveau_essence))
            {echo 'tu vas mouiller tes chaussures !';}
       else
            {$this->niveau_essence += $quantite;}
      }
}
```

5.4 CONSTRUCTEURS

Parmi les méthodes, le constructeur est la `function` appelée automatiquement lors de l'instanciation d'un objet par l'instruction PHP `new`.

Le constructeur a un nom réservé `contruct()` et peut recevoir des paramètres afin d'initialiser certaines données.

Exemple :

```
class Voiture {
      private $couleur ;
      private $poids ;
      …
      public function mettre-essence($quantite){…} ;
      …
      public function_construct($uneCouleur, $unePuissance){
            $this->couleur = uneCouleur;
            $this->puissance = $unePuissance;
      };

}
```

5.5 GETTERS

Les getters sont décrits par des `function` exposée en `public`, chacun ayant pour but de retourner (par l'instruction `return`) la valeur d'une donnée privée.

Exemple :

```
class Voiture {
      private $couleur ;
      private $poids ;
      …
      public function mettre_essence($quantite){…} ;
      …
      public function __construct($une_couleur, $une_puissance){…} ;
      …
      public function get_couleur(){
            return $this->couleur;
      }
}
```

5.6 SETTERS

Les setters sont décrits par des `function` exposée en `public`, chacun ayant pour but d'affecter la valeur d'une donnée privée.

Exemple :

```
class Voiture {
      private $couleur ;
      private $poids ;
      …
      public function mettre_essence($quantite){…} ;
      …
      public function __construct($une_couleur, $une_puissance){…} ;
      …
      public function get_couleur(){…} ;
      …
      public function set_couleur($une_couleur){
            $this->couleur = $une_couleur;
      };
}
```

5.7 LIMITES DE L'ORIENTE OBJET EN PHP

L'orientation objet en PHP reste récente et présente des lacunes par rapport aux langages purement orientés objet comme Java ou C#. Il manque essentiellement au PHP orienté objet :

- **Le typage** des paramètres et valeurs de retour des méthodes ;
- **La surcharge de méthodes** (technique qui autorise des variantes en nombre et types de paramètres) ;

Ces lacunes sont parfois reprochées au PHP car elles limitent les possibilités et altèrent les bonnes pratiques de conception orientée objet ; toutefois, de bien belles applications orientées objet existent et donnent satisfaction.

Il faut encore souligner qu'une programmation orientée objet nécessite plus de code à interpréter et donc consomme plus de ressources côté serveur Web, ce qui peut se poser problème lors de la montée en charge.

5.8 PHP ET LES CONCEPTS OBJET AVANCES

L'orientation objet du langage PHP va au-delà de ces principes de base. PHP supporte encore les concepts de :

- **Héritage**
- **Association**
- Classe abstraite
- Interface
- Traits
- Membres statiques…

Dans le cadre de cette initiation, on se limitera à la découverte de l'héritage et de l'association en PHP.

5.8.1 L'héritage

Une classe existante peut à son tour servir de 'moule' à fabriquer de **nouvelles classes** plus ***spécialisées*** présentant des similarités et des caractéristiques spécifiques :

- Une classe Animal peut se spécialiser en classes Mammifère et Poisson
- Une classe Mammifère peut encore se spécialiser en classes Canin et Félin
- Et ainsi de suite avec des classes Chat, Tigre, Chien…

La notion d'héritage correspond à la relation « ***est un*** » entre classes : un Chat ***est un*** Canin ; un Canin ***est un*** Mammifère.

Attention tout de même que si un chat, dérivé de Canin, est un Mammifère, tous les Mammifères ne sont pas des chats ! Cette relation entre classes est bien à sens unique.

On parle de ***spécialisation*** ou ***dérivation*** en observant les classes qui héritent d'une classe.

On parle de ***généralisation*** en observant les classes 'parentes' depuis une classe spécialisée.

L'intérêt de définir une classe dérivée est que le **contenu d'une classe dérivée se limite aux membres spécifiques** à cette classe car elle hérite automatiquement des membres de ses classes parentes.

Ainsi, dans notre exemple la classe Chat ne contient peut-être que la méthode Miauler() mais elle hérite de la méthode Allaiter() de la classe Mammifère et de la méthode SeReproduire() de la classe Animal.

En PHP une classe peut dériver d'une seule autre classe à la fois.

En PHP l'héritage est défini par le mot-clé **`extends`**.

Exemples :

```
class Chat extends Canin {
      public miauler(…) {…};
};

class Mammifere extends Animal{
      public allaiter(…) {…} ;
}
```

Lors de l'héritage, les membres déclarés privés ne sont pas transmis aux classes dérivées ; ainsi, si le magot de l'ancêtre a été déclaré `private`, ses descendants ne pourront pas en profiter !

Pour qu'un membre puisse être hérité par les classes dérivées, il doit être déclaré dès l'origine en exposition `public` ou `protected`.

Ainsi dans l'exemple précédent de la classe `Voiture`, si on conçoit qu'une voiture peut être spécialisée en `Berline`, `Monospace`, `Sport` ou `Fourgon` par exemple, les membres cités de la classe `Voiture` doivent être déclarés en exposition `protected`.

Exemple :

```
class Voiture {
      protected $couleur ;
      protected $poids ;
      …
      public function mettre_essence($quantite){…} ;
      …
      public function __construct($une_couleur, $une_puissance){…} ;
      …
      public function get_couleur(){…} ;
      …
      public function set_couleur($une_couleur){…}
      …
      };
}
```

Attention dans ce cas qu'une nouvelle classe `Voiture_electrique` ne peut être une spécialisation de la classe `Voiture` car une voiture électrique n'a pas de réservoir de carburant ! Dans ce cas de figure, il serait nécessaire de définir une classe `Vehicule`, ancêtre des classes `Voiture` et `Voiture_electrique` afin d'y déclarer tous les membres communs. Tout cela doit être murement réfléchi AVANT programmation ; c'est l'objet de la modélisation UML par exemple.

6. UTILISER UN OBJET EN PHP

Pour raisonner orienté objet, il faut être au moins à deux ! Le programme principal instancie puis manipule et utilise les objets qu'il a ainsi créés. D'autres programmes définissent la structure des classes et le fonctionnement de ces objets.

En PHP, le programme principal est en général de type procédural car il correspond à une demande de pages exprimée par un navigateur.

Il est de bonne pratique de stocker les différentes parties de code dans des scripts séparés que le programme principal fusionne selon les besoins avec la fonction `require()`.

En PHP, on instancie un objet par l'instruction `new` qui effectue 3 choses :

- Demander au système de réserver un espace mémoire pour y stocker toutes les données de l'objet ;
- Retourner la référence fournie par le système et lui permettant de retrouver cet espace mémoire ;
- Invoquer et exécuter la méthode constructeur quand elle existe ;

Exemples :

```
// instancier un objet Voiture vierge et retourner sa référence
$ma_porsche = new Voiture() ;

// instancier un objet Chat et initialiser ses propriétés grâce au constructeur paramétré
// puis retourner sa référence
$mon_chat = new Chat('Minou', 'Angora','tigré',3') ;
```

Enfin, après instanciation, un script désigne un objet par la variable stockant sa référence et accède à une propriété ou une méthode par l'opérateur `->`.

Exemple :

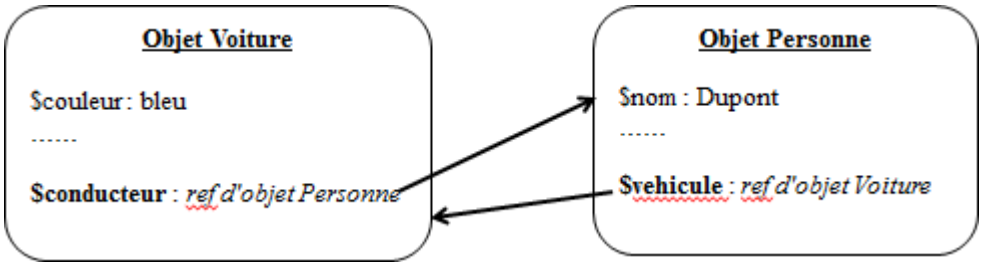
```
$mon_chat->miauler() ;
```

7. LA RELATION D'ASSOCIATION ENTRE OBJETS

Au niveau des objets, certaines propriétés représentent des liens entre objets. Ainsi un objet `Voiture` est en relation avec un objet `Personne` en ce sens que la voiture appartient à une personne et qu'une personne conduit ou possède une voiture.

Pour représenter ce type de relation d'association entre objets, on définit dans une classe un attribut qui stockera la référence de l'objet associé ; cette référence permettra d'accéder à tous les membres de l'objet associé.

Exemple :



Dans l'objet de type `Voiture`, `$conducteur` contient la référence de l'objet `Personne`, ce qui permet de retrouver son nom mais aussi toutes les autres propriétés de la personne (`$age`, `$type_permis_conduire` …) et même d'invoquer ses méthodes (comme `Supprimer_point($nb_points)` en cas d'infraction).

Comme le langage PHP n'est pas typé, il reste difficile de connaître le type de donnée attendu pour une propriété d'une classe manipulée par une autre classe et une bonne documentation du code est indispensable (ici, la variable `$conducteur` pourrait a priori aussi bien stocker le seul nom de la personne, type `string`, que la référence à la classe `Personne`).

CREDITS

ŒUVRE COLLECTIVE DE l'AFPA
Sous le pilotage de la DIIP et du centre d'ingénierie sectoriel Tertiaire-Services

Equipe de conception (IF, formateur, mediatiseur)
B. Hézard - Formateur
Ch. Perrachon – Ingénieure de formation

Date de mise à jour : 31/03/16