Algorithmique et développement web S2
– 5. Classes –

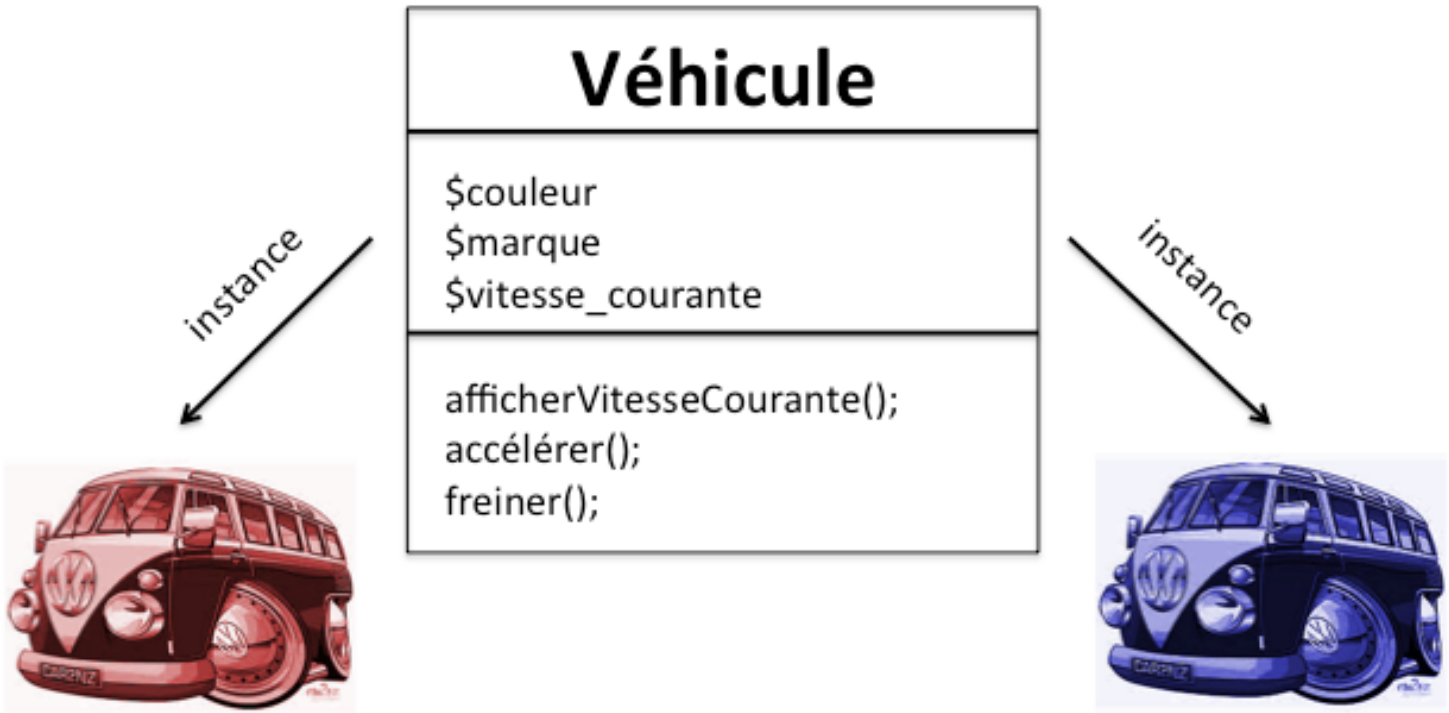
Christophe BLANC

– IUT MMI –
IUT d'Allier
Université Clermont Auvergne
www.christophe-blanc.fr

2016-2017

- ✔ Un objet a un état et un comportement
- ✔ Il modélise des entités réelles...
  - Voitures, personnes, comptes bancaires
- ✔ Il dispose d'attributs :
  - couleur, poids, équilibre, etc...
- ✔ et de méthodes :
  - ralentir, parler, créditer, etc...

- ✔ Une structure qui dispose d'attributs et de méthodes
- ✔ Définition d'un groupe d'objets ayant :
  - même caractéristiques
  - même comportement
  - exemple : toutes les voitures ont 4 roues, et peuvent freiner
- ✔ Un nouveau tyep de données
  - sur lequel on peut faire des instances



Chaque véhicule a les mêmes attributs mais pas les mêmes valeurs.
Chaque véhicule a le même comportement.

PHP propose des fonctionnalités classiques de programmation orientée objet :

- ✔ définition d'une classe ;
- ✔ utilisation de méthodes constructeur et destructeur ;
- ✔ notions d'attribut ou de méthode public, privé, protégé ;
- ✔ notions de classe ou méthode abstraite, d'interface ;

Une classe est un type composite regroupant des variables (appelées attributs de la classe) et des fonctions (appelées méthodes de la classe). En soi, une classe ne contient pas de données ; c'est juste un modèle, une définition.
A partir de la classe, il est possible de définir (instancier) des objets qui ont la structure de la classe et qui, eux, contiennent des données.

Le mot clé *class* permet d'introduire la définition d'une classe.

Syntaxe

```
class nom_class
{
        //definition des attributs
        public | private | protected $attribut = expression_litterale;
        ...
        //definition des methodes
        public | private | protected function methode()
        {
        ...
        }
        ...
}
```

- ✔ nom_classe : nom de la classe (doit respecter les règles de nommage)
- ✔ $attribut : nom d'une variable correspondant à un attribut de la classe.
- ✔ expression_litterale : valeur initiale de l'attribut.
- ✔ méthode : définition d'une fonction correspondant à une des méthodes de la classe.

La visibilité des attributs et des méthodes est définie par une des mots clés suivants :

- ✔ public : l'attribut ou la méthode sont publics et l'on peut y accéder de l'extérieur de la classe.
- ✔ private : l'attribut ou la méthode sont privés et l'on ne peut y accéder que de l'intérieur de la classe.
- ✔ protected : l'attribut ou la méthode sont protégés et l'on ne peut y accéder que de l'intérieur de la classe ou des classes dérivées de la classe (héritage).

Par défaut, une méthode est publique. Par contre, la visibilité de l'attribut doit être spécifiée.

Classes
Définir une classe : exemple PHP

```php
class Vehicule
{
        public $couleur, $marque;
        public $vitesse_courante = 0;

        public function afficherVitesseCourante(){
                echo 'La vitesse est '. $this->vitesse_courante;
        }

        public function accelerer($inc){
                $this->vitesse_courante += $inc;
        }

        public function freiner($dec){
                if($this->vitesse_courante - $dec >=0)
                {
                        $this->vitesse_courante -= $dec;
                }
                else
                {
                        $this->vitesse_courante = 0;
                }
        }
}
```

- ✔ Appelée aussi attribut
- ✔ Définit l'état de l'objet
- ✔ Sa valeur est spécifique pour une unique instance
  - dans l'exemple précédent :
    - $couleur, $marque, $vitesse_courante sont des attributs
- ✔ accès à ces attributs :
  - $nomDeInstance->nomDeAttribut

Instancier une classe signifie créer un objet basé sur la définition de la classe. Dans une certaine mesure, cela revient à définir une variable ayant comme "type", la classe. L'instanciation s'effectue grâce à l'opérateur *new*.

```
$monVehicule = new Vehicule();
$monVehicule->marque = "Porsche";
echo "La marque est ". $monVehicule->marque;

$sonVehicule = new Vehicule();
$sonVehicule->marque = "Ford";
echo "La marque est ". $sonVehicule->marque;
```

- ✔ bloc de code définissant un comportement de l'instance
- ✔ déclarée dans la classe :
  - dans l'exemple précédent :
    - accelerer(), freiner(), afficherVitesseCourante() sont des méthodes
- ✔ pour appeler ces méthodes
  - $nomDeInstance->maMethode()

```
$monVehicule = new Vehicule();
$monVehicule->accelerer(60);
$monVehicule->freiner(10);
$monVehicule->afficherVitesseCourante();

$sonVehicule = new Vehicule();
$sonVehicule->accelerer(30);
$sonVehicule->afficherVitesseCourante();
```

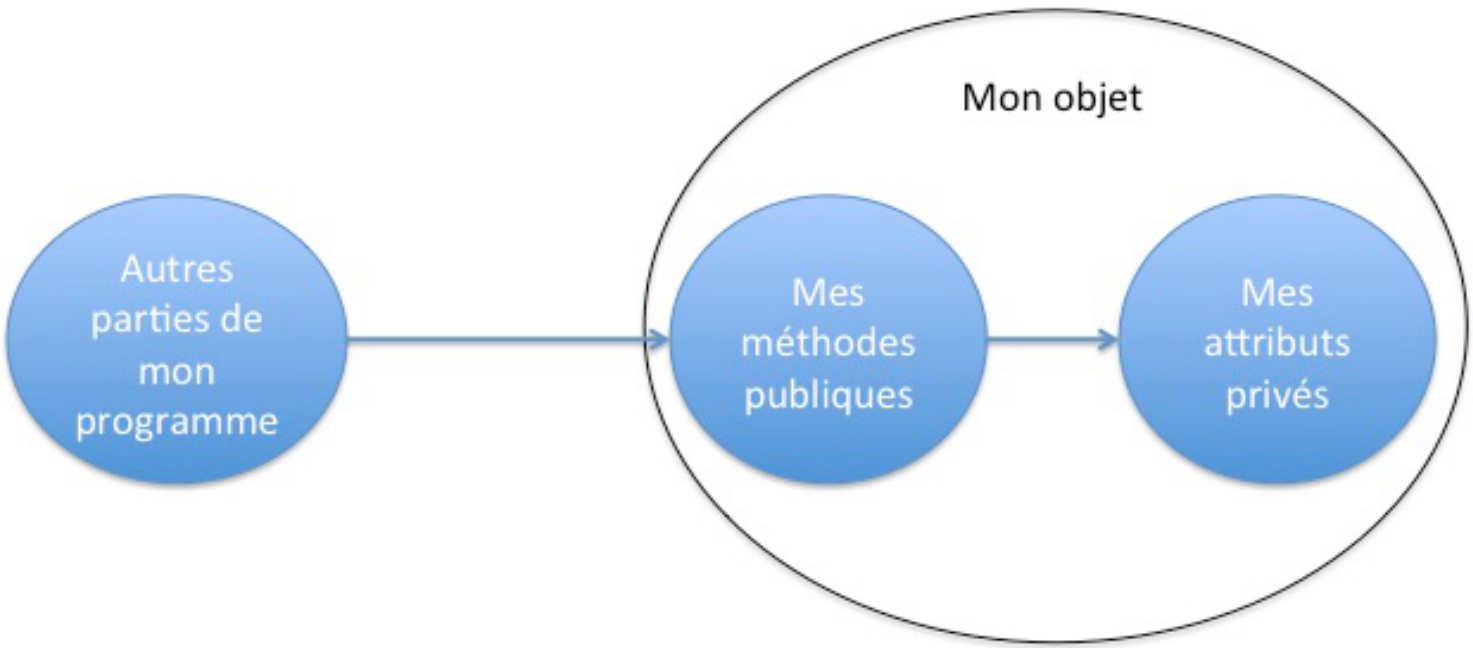
- ✔ Méthode spéciale
  - exécuté quand une nouvelle instance est créé.
  - appelé avec le mot clef *new*.
- ✔ 2 façons de le définir :
  - pour toutes les versions PHP : __construct()
  - seulement après PHP5 : le nom de la classe

```
class Vehicule
{
        public $couleur, $marque,$vitesse_courante;

        public function __construct($marque = null, $couleur = null,
        $vitesse_courante = 0)
        {
                $this->marque = $marque;
                $this->couleur = $couleur;
                $this->vitesse_courante =$vitesse_courante;
        }
}

$vehicule = new Vehicle("Porsche");
echo $vehicule->marque;
```

- ✔ Considéré un objet comme une sorte de "boite noire".
- ✔ La seule façon de modifier l'état d'un objet ?
  - utiliser les méthodes d'instance
- ✔ Les attributs sont déclarés privés (*private*) ou protégés (*protected*)
- ✔ Pour les modifier en dehors de la classe
  - utiliser les méthodes publiques


Mon objet
Autres parties de mon programme
Mes méthodes publiques
Mes attributs privés

- ✔ PHP a 3 modes de contrôle d'accès :
  - publique : *public*
  - privé : *private*
  - protégé : *protected*

- ✔ accessible partout dans votre application
- ✔ si la visibilité n'est pas spécifiée alors l'accès est publique

```php
class MaClasse
{
        public $maVariablePublic1 = "var1";
        $maVariablePublic2 = "var2";
}
$monInstance = new maClasse();
echo $monInstance->maVariablePublic1; // var1
echo $monInstance->maVariablePublic2; // var2
```

✔ accessible seulement à l'intérieur de la classe.

```
class MaClasse
{
        private $maVariable = 0;
        public function incVar()
        {
                echo "La valeur de la variable est : ".
                ++$maVariable;
        }
}
$monInstance = new maClasse();
echo $monInstance->incVar(); // 1
echo $monInstance->maVariable; // fatal error
```

- ✔ accessible à l'intérieur de la classe courante et à l'intérieur des classes qui l'étendent.

```
class MaClasse
{
        protected $maVariable = 0;
}
class MaSousClasse extends MaClasse
{
        public function incVar()
        {
                echo "La variable est : ",++$maVariable;
        }
}
$monInstance = new maSousClasse();
echo $monInstance->incVar(); // 1
```

- ✔ méthodes spéciales afin d'accéder aux attributs privés
  - Getters : récupérer la valeur des attributs. Nom usuel : getXXX() ;
  - Setters : modifier la valeur des attributs. Nom usuel : setXXX() ;

```php
class Vehicule
{
        private $couleur, $marque;

        public function getCouleur()
        {
                return $this->couleur;
        }
        public function setCouleur($couleur)
        {
                $this->couleur = $couleur;
        }

        public function getMarque()
        {
                return $this->marque;
        }
        public function setMarque($marque)
        {
                $this->marque = $marque;
        }

}
```

Si un jour on décide que chaque marque doit être en majuscule.

```
class Vehicule
{
        public $couleur, $marque;

        ...

        public function getMarque()
        {
                return $this->marque;
        }
        public function setMarque($marque)
        {
                $this->marque = strtoupper($marque);
        }

}
```

- ✔ Imaginez que vous souhaitez créer deux classes :
  - Bicyclette : représente une bicyclette
  - Bateau : représente un bateau
- ✔ Ces classes sont des véhicules et partagent des attributs et comportements communs :
  - vitesse courante, marque, couleur,...
  - accélérer, freiner,...
- ✔ fastidieux de dupliquer le code dans chaque classe
- ✔ l'héritage résout ce problème
- ✔ définir une classe avec les attributs et méthodes communes
  - faire hériter Bicyclette et Bateau de cette classe

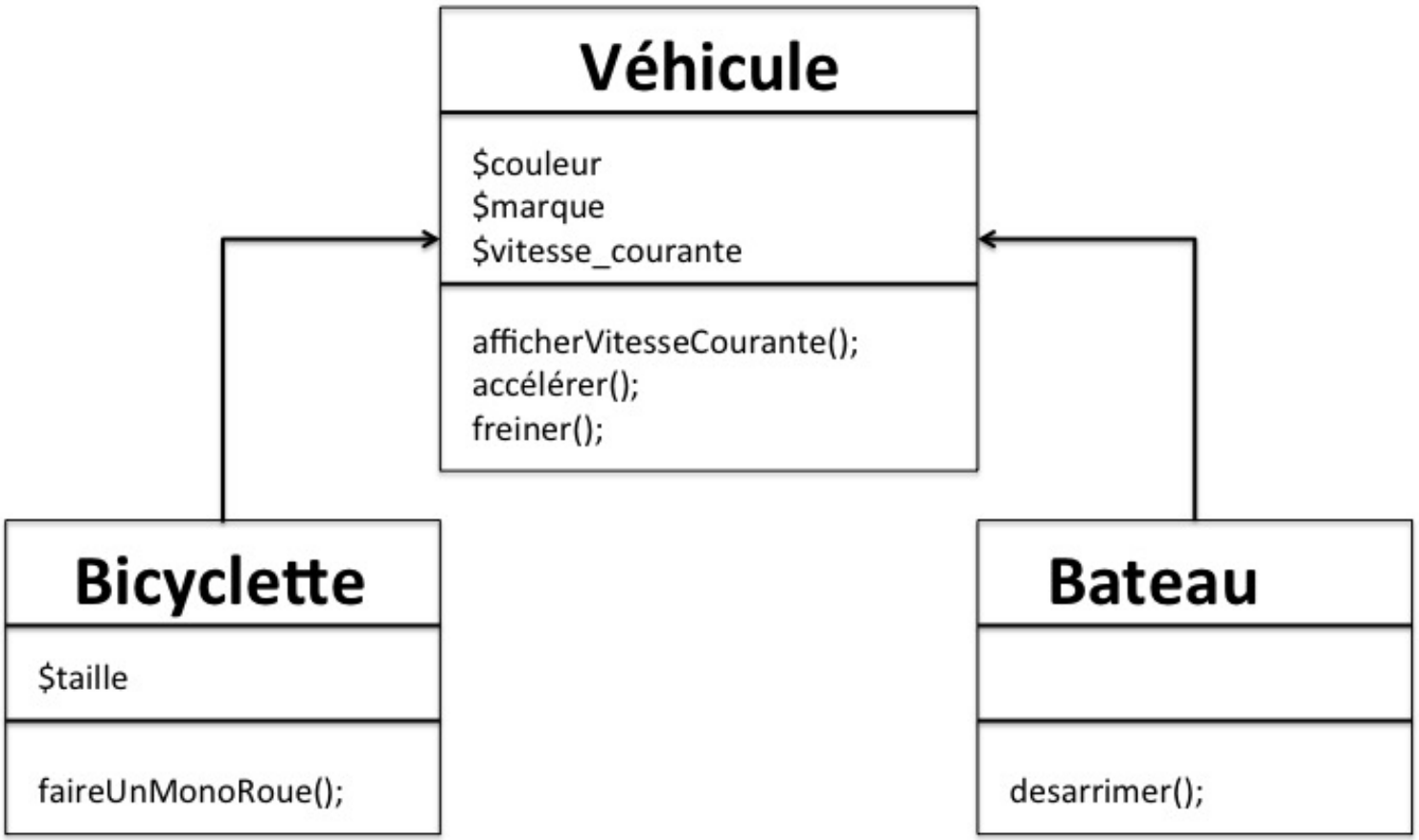
- ✔ Pour définir qu'une classe hérite d'une autre :

```
class Bicyclette extends Vehicule{
        ...
}
```

- ✔ en POO on considère :
  - Bicyclette est une sous classe ou classe fille
  - Vehicule est une classe mère

- ✔ Tous les membres publiques et protégés de la classe mère sont accessibles à l'intérieur des classes filles

Une classe fille ne peut hériter que d'une et une seule classe.


Véhicule
$couleur
$marque
$vitesse_courante
afficherVitesseCourante();
accélérer();
freiner();
Bicyclette
$taille
faireUnMonoRoue();
Bateau
desarrimer();

```php
class Vehicule
{
        public $couleur, $marque;
        public $vitesse_courante = 0;

        public function afficherVitesseCourante(){
                echo 'La vitesse est '. $this->vitesse_courante;
        }

        public function accelerer($inc){
                $this->vitesse_courante += $inc;
        }

        public function freiner($dec){
                if($this->vitesse_courante - $dec >=0)
                {
                        $this->vitesse_courante -= $dec;
                }
                else
                {
                        $this->vitesse_courante = 0;
                }
        }
}
```

```php
class Bicyclette extends Vehicule
{
        private $taille;

        function __construct($taille)
        {
                $this->taille = $taille;
        }

        public function faireUnMonoRoue()
        {
                echo "Yeah, avec une seule roue !";
        }
}
$maBicyclette = new Bicyclette("XL");
$maBicyclette->accelerer(20);
$maBicyclette->afficherVitesseCourante();
```

- ✔ L'héritage c'est bien mais...
  - comment faire si on veut qu'une classe fille exécute une fonction de manière différente à la classe mère.
- ✔ Prenons par la méthode accélérer :
  - Le bateau doit entraîner ses propulseurs
  - La bicyclette doit faire tourner les pédales
  - les deux doivent augmenter leur vitesse

- ✔ Version bateau pour la méthode accélérer

```
class Bateau extends Vehicule
{
        ...

        public function accelerer($inc)
        {
                $this->entrainerPropulseurs();
                parent::accelerer($inc);
        }
        public function entrainerPropulseurs()
        {
                echo "Entrainement des propulseurs";
        }
}
```

- ✔ Version bicyclette pour la méthode accélérer

```
class Bicyclette extends Vehicule
{
        ...

        public function accelerer($inc)
        {
                $this->tournerPedales();
                parent::accelerer($inc);
        }
        public function tournerPedales()
        {
                echo "Pedales tournees";
        }
}
```

```php
$maBicyclette = new Bicyclette("XL");
$maBicyclette->accelerer(10);      //Affiche "Pedales tournees"

$monBateau = new Bateau();
$monBateau->accelerer(30);                 //Affiche "Entrainement des propulseurs"

echo $maBicyclette->afficherVitesseCourante(); //La vitesse est 10
echo $monBateau->afficherVitesseCourante(); //La vitesse est 30
```

- ✔ Comme vous le voyez, si une classe fille redéfinit une méthode d'une méthode existante de la classe mère, elle la surpasse.
- ✔ Appeler la méthode parent comme ceci
  - `parent::nomDeLaMethode()`

- ✔ Souvenez vous de la hiérarchie pour la classe Vehicule :
  - 2 classes filles : Bateau et Bicyclette
  - si nous avons besoin d'un autre type de véhicule, nous avons juste besoin de créer une autre classe fille
- ✔ donc, il n'est plus nécessaire de laisser la classe Vehicule instanciable
  - pour ceci, on peut la définir abstraite : mot clé *abstract*

- ✔ La classe abstraite est faite pour être héritée : elle ne peut pas être instancié.
- ✔ elle peut être composée de méthodes abstraites
  - les méthodes sont définies sans implémentation
  - elles sont implémentées dans les classes filles

```
abstract class MaClasseAbstraite(){
        ...
}
```

```php
abstract class MaClasseAbstraite
{
        public abstract function hello();
}

class MaSousClasse1 extends MaClasseAbstraite
{
        public function hello()
        {
                echo "Hello de MaSousclasse1";
        }
}

class MaSousClasse2 extends MaClasseAbstraite
{
        public function hello()
        {
                echo "Hello de MaSousclasse2";
        }
}//Tout est ok
```

```php
abstract class MaClasseAbstraite
{
        public abstract function hello();
}

class MaSousClasse extends MaClasseAbstraite
{
        public function helloWorld()
        {
                echo "Hello de MaSousclasse";
        }
}

//Fatal error
```

```php
abstract class MaClasseAbstraite
{
        public function hello()
        {
                echo "Hello World!";
        }
}

$monInstance = new MaClasseAbstraite();

//Fatal error
```

- ✔ Toutes les méthodes sont abstraites
- ✔ Comme pour une classe abstraite, elle ne peut pas être instanciée
- ✔ Déclarée comme ceci :

```
interface MonInterface
{
        ...
}
```

- ✔ Pour implémenter l'interface :

```
class MaClasse implements MonInterface
{
        ...
}
```

Classes
Héritage : exemple d'interface

```php
interface Animal
{
        function manger();
        function bruit();
        function dormir();
        function chasser();
}
class Chat implements Animal
{
        public function manger()
        {
                echo "Miam";
        }
        public function bruit()
        {
                echo "Miaou";
        }
        public function dormir()
        {
                echo "zzZZZZzz";
        }
        public function chasser()
        {
                echo "...";
        }
}
```