



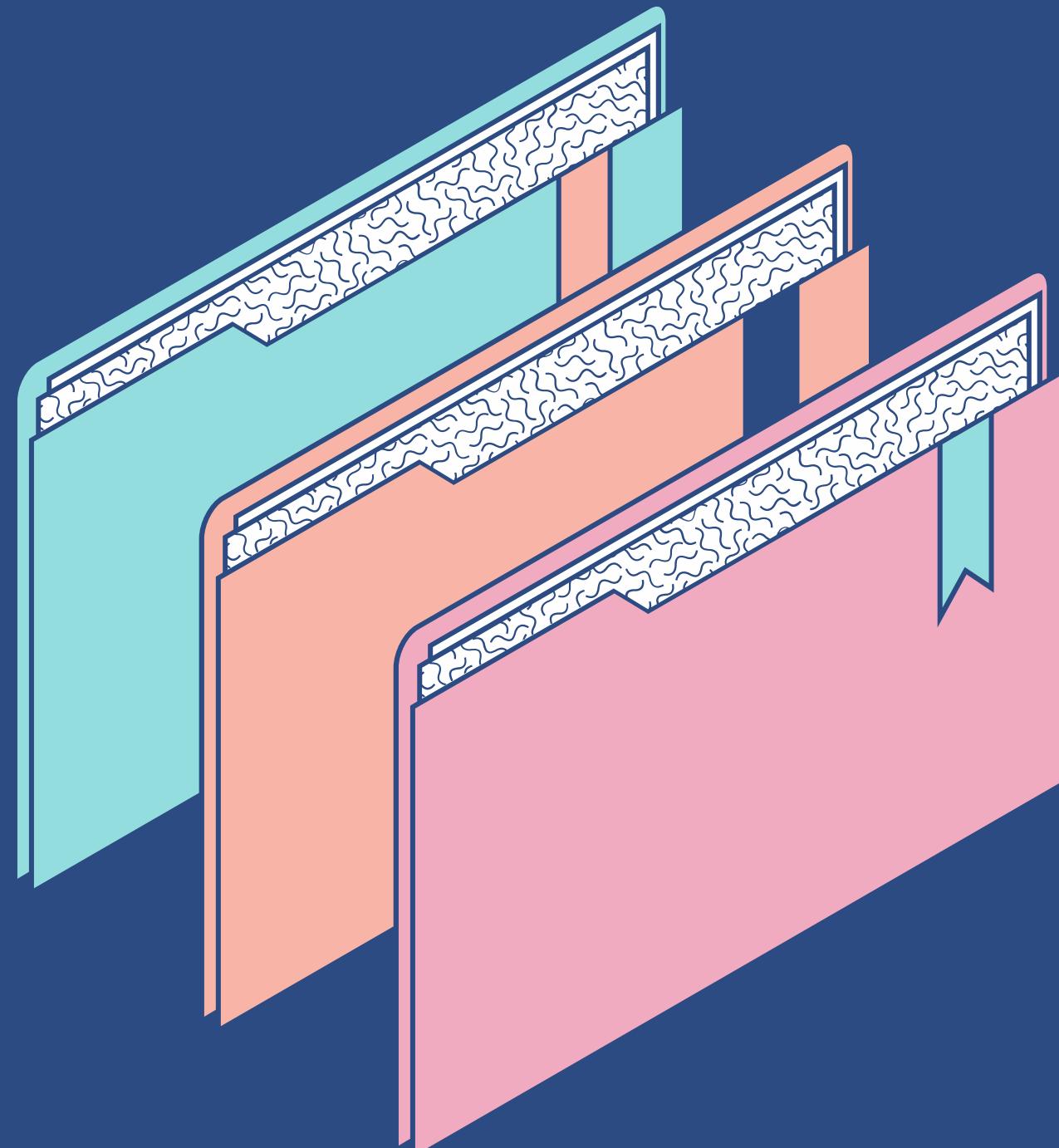
PHP

```
1 <?php
2 require_once 'class/Message.php';
3 require_once 'class/GuestBook.php';
4 $errors = null;
5 $success = false;
6 $guestbook = new GuestBook(__DIR__ . DIRECTORY_SEPARATOR . 'data' . DIRECTORY_SEPARATOR . 'messages');
7 if (isset($_POST['username'], $_POST['message'])) {
8     $message = new Message($_POST['username'], $_POST['message']);
9     if ($message->isValid()) {
10         $guestbook->addMessage($message);
11         $success = true;
12         $_POST = [];
13     } else {
14         $errors = $message->getErrors();
15     }
16 }
17 $messages = $guestbook->getMessages();
18 $title = 'Livre d\'or';
19 require 'elements/Header.php';
20 ?>
21
22 <div class="container">
23     <h1>Livre d'or</h1>
24
25     <?php if (!empty($errors)): ?>
26     <div class="alert alert-danger">
27         Formulaire invalide
28     </div>
29     <?php endif ?>
30
31     <?php if ($success): ?>
32     <div class="alert alert-success">
33         Merci pour votre message !
34     </div>
```

Apprendre PHP Introduction

SOMMAIRE

1. Classes et Objets
2. Portée des objets
3. Gestion BDD
4. Héritage



Classes et objets POO

```
<?php
    class Utilisateur{
        protected $user_name;
        protected $user_pass;
        public const ABONNEMENT = 15;

        public function __construct($n, $p){
            $this->user_name = $n;
            $this->user_pass = $p;
        }

        public function __destruct(){
            //Du code à exécuter
        }

        public function getNom(){
            echo $this->user_name;
        }
    }
?>
```

Présentation de la POO

La POO nous permet de modéliser dans notre code des éléments de la vie réelle : Un véhicule, un animal, un bâtiment ...

Une classe fonctionne comme une recette de cuisine, qui va nous permettre de créer des objets :

Nous allons créer une classe (recette), qui contiendra des attributs (ingrédients), et méthodes, permettant de réaliser notre objet (plat)

Exemple de la classe voiture

Chaque classe sera créée dans un fichier spécifique nomClasse.php

```
<?php  
    // remarque : un nom de classe commence  
    // toujours par une Majuscule  
    class Vehicule{  
    }  
?>
```



Instanciation d'objets

Pour créer un nouvel objet, nous utiliserons la syntaxe suivante :

```
<?php  
    //import du fichier class.php qui contient la classe Vehicule  
    require './Voiture.php';  
    //création d'un nouveau véhicule depuis la classe Vehicule  
    $voiture = new Vehicule();  
?>
```

Déclaration d'attributs

La déclaration d'attributs s'effectue directement au sein de la classe

```
<?php  
    class Vehicule  
    {  
        //Attributs :  
        public $nomVehicule ;  
        public $nbrRoue;  
        public $vitesse ;  
    }  
?>
```

Affecter une valeur à un attribut

```
<?php  
    //appel du fichier class.php qui contient la classe Vehicule  
    //require est équivalent à include  
    require './voiture.php';  
    //création d'un nouveau véhicule depuis la classe Vehicule  
    $voiture = new Vehicule();  
    //ajout de valeur aux attributs de la classe Vehicule  
    $voiture->nomVehicule = "Audi A3";  
    $voiture->nbrRoue = 4;  
    $voiture->vitesse = 250;  
?>
```

NB : Cette syntaxe est valide uniquement pour les attributs **public**.

Création de méthodes

```
<?php
    class Vehicule{
        /**
         *-----|
         * | | | | | Attributs : |
         *-----|
        public $nomVehicule ;
        public $nbrRoue;
        public $vitesse ;
        /**
         *-----|
         * | | | | | Fonctions : |
         *-----|
        //fonction démarrer le véhicule
        public function demarrer(){
            echo "<p>Démarrage de la $this->nomVehicule Vrooom !!!!</p>";
        }
    }
?>
```



Appel de méthodes

```
<?php  
    //appel du fichier class.php qui contient la classe Vehicule  
    //require est équivalent à include  
    require './class.php';  
    //création d'un nouveau véhicule depuis la classe Vehicule  
    $voiture = new Vehicule();  
    //ajout de valeur aux attributs de la classe Vehicule  
    $voiture->nomVehicule = "Audi A3";  
    $voiture->nbrRoue = 4;  
    //utilisation de la méthode démarrer  
    $voiture->demarrer();  
?  
?
```



Exercice d'application n°1 POO

Créer un fichier test_objet.php

Créer une nouvelle classe Maison maison.php, qui contient les attributs : nom, longueur, largeur, nbEtages

Instancier une nouvelle maison dans le fichier test_objet.php

Créer une méthode surface, qui calcule et affiche la superficie de la maison

Appeler la méthode surface et affichez le résultat (Attention à bien prendre en compte les étages)



Correction d'application n°1 POO

maison.php

```
<?php

class Maison{
    /**
     *----- Attributs :
    -----
    public $nom ;
    public $longueur ;
    public $largeur;
    public $nbEtages ;
    /**
     *----- Fonctions :
    -----
    //fonction afficher la surface
    public function surface()
    {
        echo "<p>la surface vaut ".$this->longueur * $this->largeur * $this->nbEtages." m2</p>";
    }
}

?>
```





Correction d'application n°1 POO

test_objet.php

```
<?php
    //appel du fichier maison.php qui contient la classe Maison
    //require est équivalent à include
    require './maison.php';
    //création d'un nouveau véhicule depuis la classe Vehicule
    $maison = new Maison();
    //ajout de valeur aux attributs de la classe Vehicule
    $maison->nom = "myHouse";
    $maison->longueur = 15;
    $maison->largeur = 7;
    $maison->nbEtages = 2;
    //utilisation de la méthode démarrer
    $maison->surface();

?>
```



Portée des objets

```
<?php
    class Utilisateur{
        protected $user_name;
        protected $user_pass;
        public const ABONNEMENT = 15;

        public function __construct($n, $p){
            $this->user_name = $n;
            $this->user_pass = $p;
        }

        public function __destruct(){
            //Du code à exécuter
        }

        public function getNom(){
            echo $this->user_name;
        }
    }
?>
```

Présentation des portées : public

En POO, chaque attribut aura une portée en fonction du paramètre devant celle ci :

- Public : L'attribut ou la méthode sera accessible n'importe où dans notre projet.
Les méthodes seront généralement publics pour être accessible depuis n'importe quel endroit du projet.
Il est fortement déconseillé d'accéder aux attributs de notre classe par une portée public. En pratique,
nous créerons des méthodes getters et setters pour lire et écrire les valeurs des attributs



Présentation des portées : private

En POO, chaque attribut aura une portée en fonction du paramètre devant celle ci :

- Private : L'attribut ou méthode sera accessible uniquement au sein de la classe.
C'est la valeur par défaut pour les attributs



Présentation des portées : protected

En POO, chaque attribut aura une portée en fonction du paramètre devant celle ci :

- Protected : L'attribut ou méthode sera accessible depuis n'importe où au sein du même dossier. Cette portée sera marginale pour le moment, mais très utile lorsque vous aborderez les notions d'héritage, et permet de sécuriser nos attributs et méthodes à l'extérieur du dossier

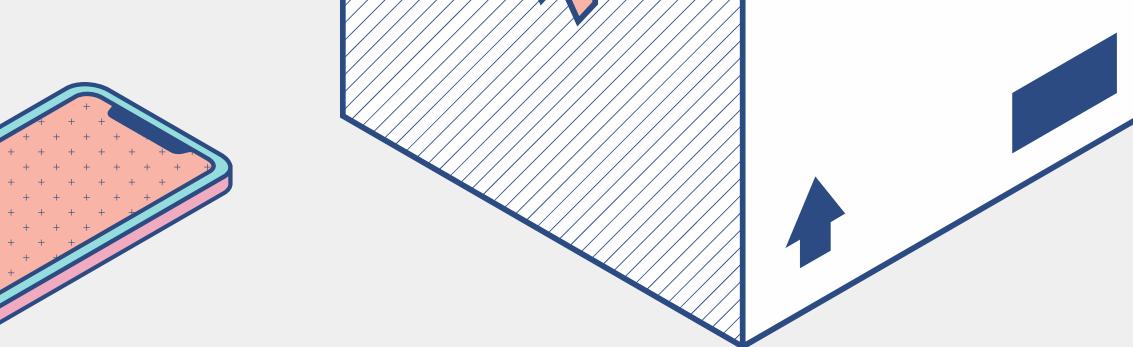
Getters / Setters

Afin de sécuriser les attributs de nos classes, nous allons les passer en private.

Pour lire et écrire du contenu au sein des attributs, nous allons donc devoir créer des méthodes spécifiques (Getters et Setters)

```
<?php
class Vehicule{
    /*-----
     |   |   |   |   |   Attributs :
     -----*/
    public $nomVehicule ;
    public $nbrRoue;
    public $vitesse ;

    /*-----
     |   |   |   |   |   Getters / Setters :
     -----*/
    //Getter nomVehicule récupère le nom du véhicule
    public function getNomVehicule()
    {
        return $this->nomVehicule;
    }
    //setter nomVehicule remplace le nom du véhicule
    public function setNomVehicule($new_nom_vehicule)
    {
        $this->nomVehicule = $new_nom_vehicule;
    }
    /*-----
     |   |   |   |   |   Fonctions :
     -----*/
    //fonction démarrer le véhicule
    public function demarrer(){
        $demarrage = '<p>Démarrage de la '.$this->getNomVehicule(). 'Vrooom !!!!</p>';
        return $demarrage;
    }
}
?>
```



Getters / Setters

Exercice d'application n°1 Getters/Setters

Créer un nouveau fichier `vehicule_private.php`

Recréer la classe `Vehicule` et passer tous les attributs en `private`

Ajouter les getters et setters

Editer les méthodes pour qu'elles s'adaptent aux nouveaux paramètres

Correction Exercice d'application n°1 Getters

```
src/vehicule.php
<?php
    class Vehicule{
        /**
         * Attributs :
         */
        private $nomVehicule ;
        private $nbrRoue;
        private $vitesse ;

        /**
         * Getters / Setters :
         */
        //Getter nomVehicule récupère le nom du véhicule
        public function getNomVehicule()
        {
            return $this->nomVehicule;
        }
        public function getNombreRoue()
        {
            return $this->nbrRoue;
        }
        public function getVitesse()
        {
            return $this->vitesse;
        }
    }
}
```

Correction Exercice d'application n°1 Setters

```
//setter nomVehicule remplace le nom du véhicule
public function setNomVehicule($new_nom_vehicule)
{
    $this->nomVehicule = $new_nom_vehicule;
}
public function setNombreRoue($new_nombre_roue)
{
    $this->nbrRoue = $new_nombre_roue;
}
public function setVitesse($new_vitesse)
{
    $this->vitesse = $new_vitesse;
}
/*
----- Fonctions :
-----*/
//fonction démarrer le véhicule
public function demarrer(){
    $demarrage = '<p>Démarrage de la '.$this->getNomVehicule(). 'Vrooom !!!!</p>';
    return $demarrage;
}
```



Correction Exercice d'application n°1 createVoiture

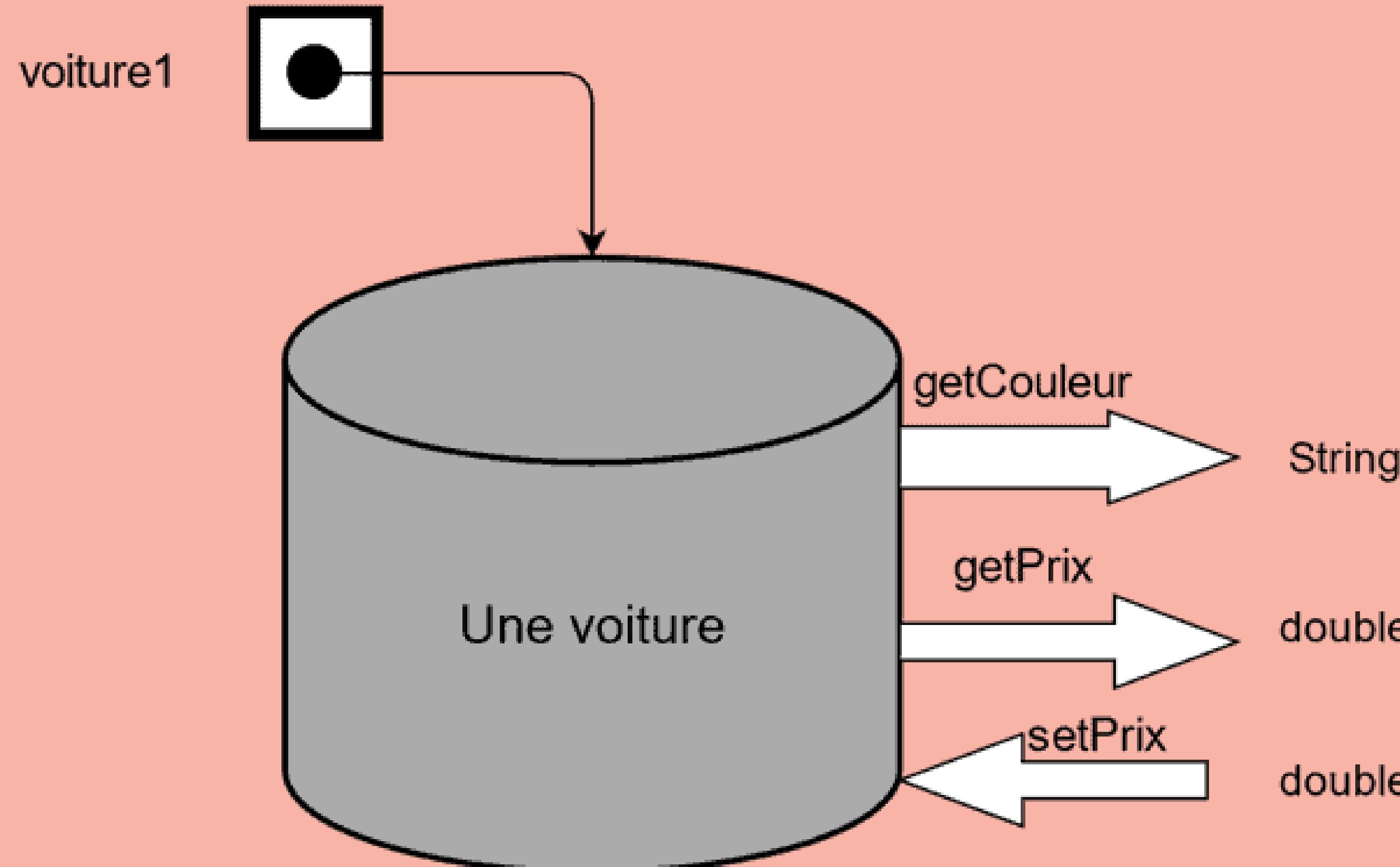
```
<?php  
    //appel du fichier class.php qui contient la classe Vehicule  
    //require est équivalent à include  
    require './voiture.php';  
    //création d'un nouveau véhicule depuis la classe Vehicule  
    $voiture = new Vehicule();  
    //ajout de valeur aux attributs de la classe Vehicule  
    $voiture->setNomVehicule("Audi A3");  
    $voiture->setNombreRoue(4);  
    $voiture->setVitesse(200);  
    //utilisation de la méthode démarrer  
    echo $voiture->demarrer();  
?  
?>
```



TP1 POO

Gestion BDD

25



Classes manager

En POO, une classe doit toujours correspondre à un, et un seul rôle !

L'instanciation d'une classe telle que Personnage a pour rôle de représenter une ligne de notre BDD.

Pour gérer ce Personnage dans notre BDD, nous allons devoir créer une nouvelle classe :

la classe Manager : PersonnagesManager

Attention à ne jamais utiliser une même classe pour représenter et gérer votre objet

Classes manager

Pour fonctionner, votre classe manager aura besoin en attribut de votre connexion à votre BDD

```
<?php  
    class PersonnagesManager  
    {  
        private $bdd;  
    }  
?>
```

Classes manager

Votre manager sera en charge du CRUD de vos données, soit :

- L'enregistrement de données (Create)
- La récupération (Read)
- La modification (Update)
- La suppression (Delete)

Classes manager

```
<?php
    class BaseManager
    {
        private $_table;
        private $_object;
        protected $_bdd;

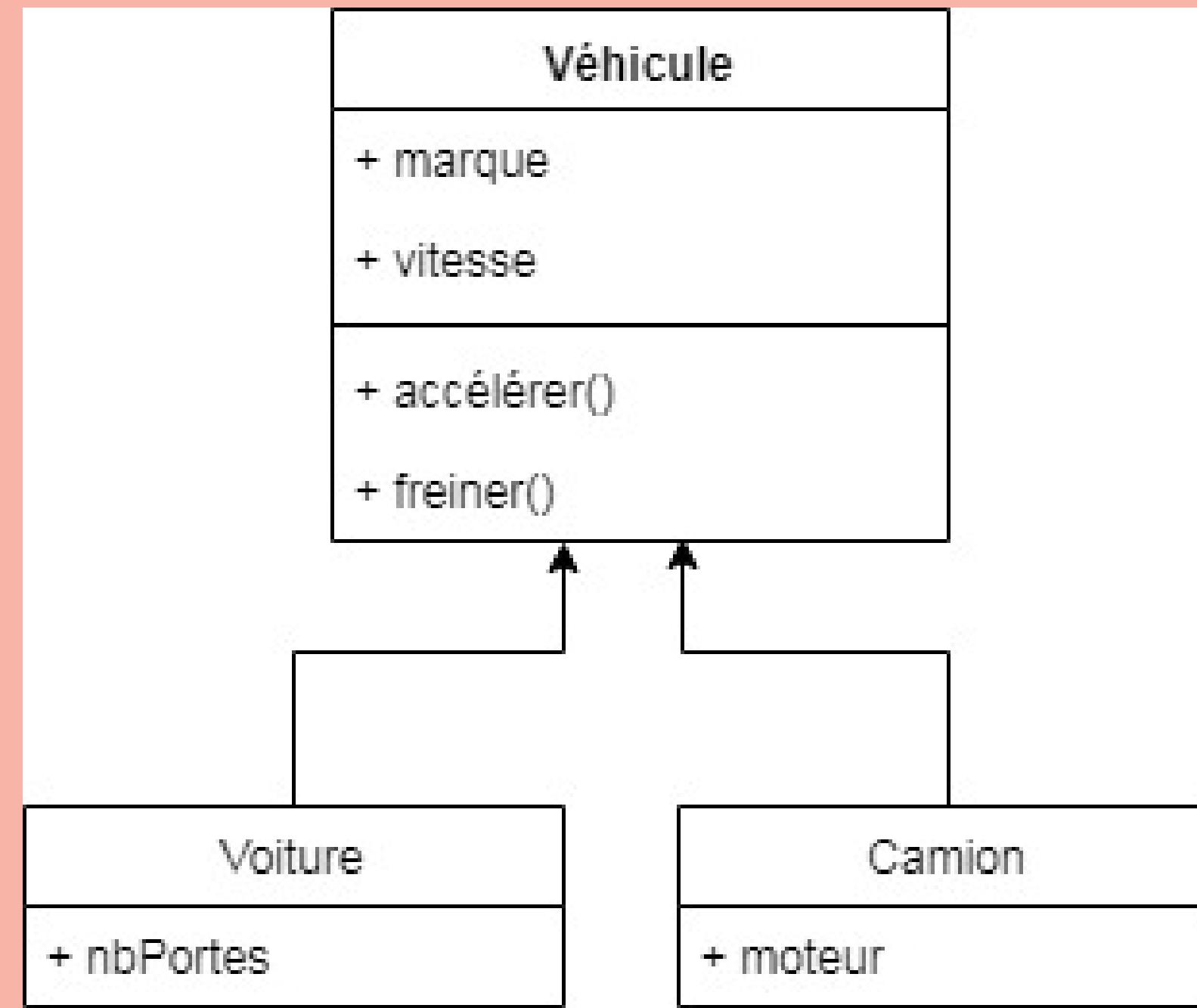
        public function __construct($table,$object,$datasource)
        {
            $this->_table = $table;
            $this->_object = $object;
            $this->_bdd = BDD::getInstance($datasource);
        }

        public function getById($id)
        {
            $req = $_bdd->prepare("SELECT * FROM " . $this->_table . " WHERE id=?");
            $req->execute(array($id));
            $req->setFetchMode(PDO::FETCH_CLASS|PDO::FETCH_PROPS_LATE,$this->_obj);
            return $req->fetch();
        }

        public function getAll()
        {
            $req = $_bdd->prepare("SELECT * FROM " . $this->_table);
            $req->execute();
            $req->setFetchMode(PDO::FETCH_CLASS|PDO::FETCH_PROPS_LATE,$this->_obj);
            return $req->fetchAll();
        }
    }
}
```

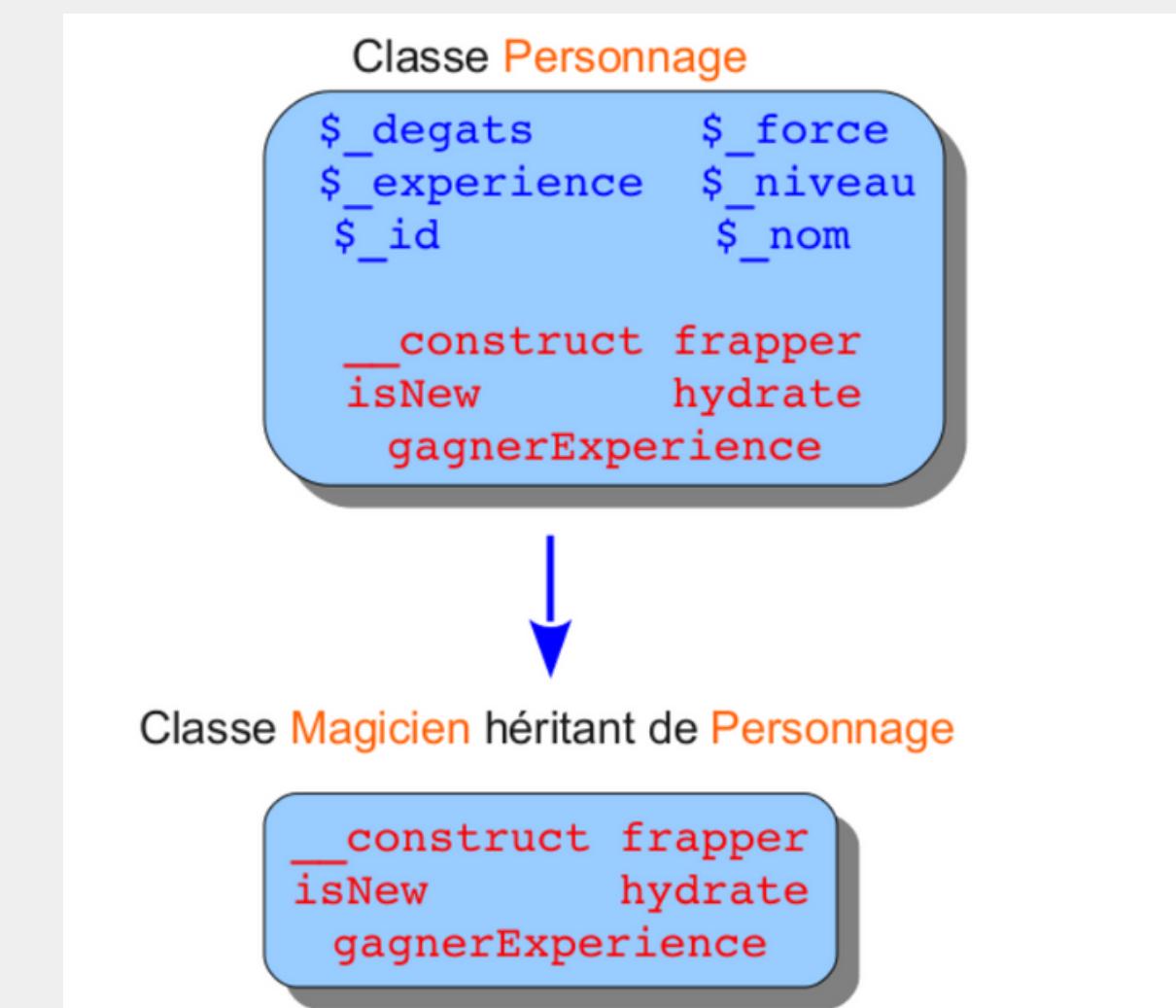
TP2 & 3 POO

Héritage



Notion d'héritage

Lorsqu'une classe B hérite d'une classe A, B hérite de tous les attributs et méthodes PUBLIQUES ou PROTECTED de la classe A.



Ici, seul les méthodes sont héritées, car les attributs sont privés

Héritage simple

On définit que B hérite de A à l'aide du mot clé `extends` dans la déclaration de la classe

```
<?php
class Personnage // Création d'une classe simple.
{
}

class Magicien extends Personnage // Notre classe Magicien hérite des attributs et méthodes de Personnage.
{}
```

Héritage simple

Une classe mère peut avoir une infinité de filles, mais une classe fille n'aura qu'une classe mère

```
class Personnage // Création d'une classe simple.  
{  
}  
  
// Toutes les classes suivantes hériteront de Personnage.  
  
class Magicien extends Personnage  
{  
}  
  
class Guerrier extends Personnage  
{  
}  
  
class Brute extends Personnage  
{  
}
```



Héritage simple

En plus des attributs et méthodes héritées, chaque classe pourra créer des attributs et méthodes qui lui sont propres, afin de la spécialiser.

```
class Personnage // Création d'une classe simple.  
{  
}  
  
// Toutes les classes suivantes hériteront de Personnage.  
  
class Magicien extends Personnage  
{  
    private $_magie; // Indique la puissance du magicien sur 100, sa capacité à produire de la magie.  
  
    public function lancerUnSort($perso)  
    {  
        $perso->recevoirDegats($this->_magie); // On va dire que la magie du magicien représente sa force.  
    }  
}
```



Surcharge de méthodes

Une classe fille peut compléter ou réécrire certaines méthodes héritées, afin de spécifier et modifier leurs comportements

```
class Personnage // Création d'une classe simple.
{
    public function gagnerExperience()
    {

    }

}

// Toutes les classes suivantes hériteront de Personnage.

class Magicien extends Personnage
{
    private $_magie; // Indique la puissance du magicien sur 100, sa capacité à produire de la magie.

    public function lancerUnSort($perso)
    {
        $perso->recevoirDegats($this->_magie); // On va dire que la magie du magicien représente sa force.
    }

    public function gagnerExperience()
    {
        // On appelle la méthode gagnerExperience() de la classe parente
        parent::gagnerExperience();

        if ($this->_magie < 100)
        {
            $this->_magie += 10;
        }
    }
}
```

Attention : la visibilité d'une méthode surchargée doit être identique à la visibilité de la méthode mère

Héritage infini

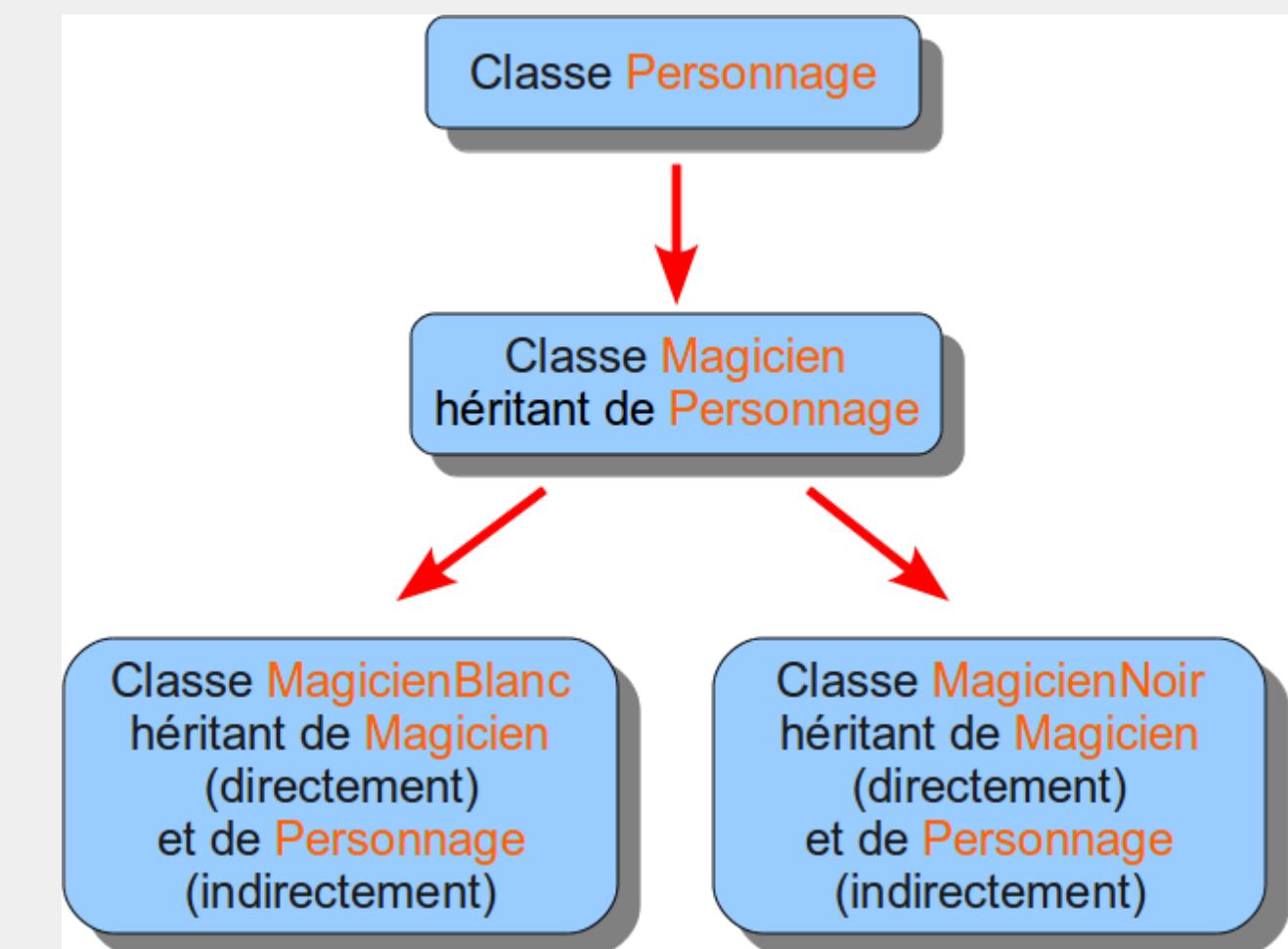
Toute classe peut être à la fois une classe fille, et une classe mère

```
class Personnage
{
}

class Magicien extends Personnage
{
}

class MagicienBlanc extends Magicien
{
}

class MagicienNoir extends Magicien
{
}
```



TP4 POO

