PHP OBJET

Niveau 1

OBJECTIFS DU MODULE

- Définir les objectifs à atteindre en fin de module :
  - Avoir une approche du développement orienté objet (POO)

PROGRAMME PÉDAGOGIQUE

- Présentation du programme par journée avec les capacités à acquérir :
  - Journée 1 :
    - Comprendre la notion d'Objet.
    - Savoir créer et utiliser une classe.
  - Journée 2 :
    - Savoir gérer son projet en différentes classes.
    - Savoir utiliser l'héritage pour factoriser le code.
  - Journée 3 :
    - Savoir utiliser correctement les classes abstraites et finales.

L'OBJET

La classe

PRÉSENTATION DE L'OBJET

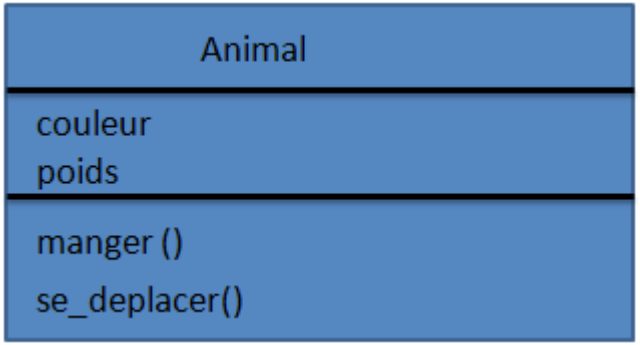
Un Objet a des caractèristiques appelées attributs et des actions appelées méthodes.

Par exemple, l'objet Animal a les propriétés: couleurs et poids. Il a aussi comme méthodes: se_deplacer et manger.
Pour construire ces objets, il faut utiliser une classe. Cette classe est comme un modèle de l'objet à partir de laquelle vous allez créer des copies.

La classe

INTRODUCTION

Une classe sert à fabriquer des objets à partir d'un modèle. Ces objets ont leurs propres attributs et méthodes.
Par exemple, la classe Animal ayant les propriétés couleurs et poids, a comme méthodes manger ou se_deplacer.



La classe

Création d'une classe en PHP:

```
<?php
   class Animal // mot-clé class suivi du nom de la classe.
   {
      // Déclaration des attributs et méthodes.
   }
?>
```

Il est conseillé de mettre une classe par fichier PHP ayant le même nom que la classe.

L'ENCAPSULATION

Les propriétés couleur et poids sont cachés aux autres classes et elles sont donc privées. La classe Animal aura des méthodes pour lire ou écrire dans ces attributs. C'est le principe de l'encapsulation. Cela permet d'avoir un code plus protégé lorsque vous travaillez en équipe.
La classe Animal ayant les propriétés couleurs et poids, aura une méthode pour modifier sa couleur, une méthode pour lire sa couleur, une méthode pour modifier son poids, une méthode pour lire son poids ainsi que d'autres méthodes comme manger ou se_deplacer (voir la section Mettre à jour et lire les attributs de l'instance plus loin dans ce chapitre).

La classe

Visibilité des attributs et des méthodes

Il existe trois types de mot clé pour définir la visibilité d'un attribut ou méthode:
- private: seul le code de votre classe peut voir et accéder à cet attribut ou méthode.
- public: toutes les autres classes peuvent voir et accéder à cet attribut ou méthode.
- protected: seul le code de votre classe et de ses sous-classes peut voir et accéder à cet attribut ou méthode.

Les sous-classes sont abordées lors de l'héritage dans un prochain paragraphe.

Création d'une classe avec ses attributs en PHP:

```
<?php
   class Animal // mot-clé class suivi du nom de la classe.
   {
            // Déclaration des attributs.
                     private $couleur;
                     private $poids;
   }
?>
```

La classe

Il est possible de définir des valeurs par défaut à vos attributs:

```
<?php
   class Animal // mot-clé class suivi du nom de la classe.
   {
            // Déclaration des attributs.
                     private $couleur = "gris";
                     private $poids = 10;
   }
?>
```

Pour ajouter les méthodes à votre classe, les règles de visibilité sont les mêmes que pour celles des attributs:

```
<?php
   class Animal // mot-clé class suivi du nom de la classe.
   {
            // Déclaration des attributs et méthodes.
                     private $couleur = "gris";
                     private $poids = 10;
```

La classe

```
public function manger()
		{
			//méthode pouvant accéder aux propriétés
			//couleur et poids
		}

		public function se_deplacer()
		{
			//méthode pouvant accéder aux propriétés
			//couleur et poids
		}
}
?>
```

Visibilité des attributs et des méthodes

Lorsque vous ajoutez du code dans votre classe, cela s'appelle implémenter la classe. Pour accéder aux attributs de votre classe, il faut utiliser la pseudo-variable $this représentant l'objet dans lequel vous écrivez.
Pour accéder à l'attribut ou la méthode de l'objet, il faut utiliser l'opérateur ->.

La classe

Par exemple, pour implémenter la méthode ajouter_un_kilo() dans la classe Animal:

```
<?php
class Animal // mot-clé class suivi du nom de la classe.
  {
                // Déclaration des attributs et méthodes.
                private $couleur = "gris";
                private $poids = 10;
                public function manger()
                {
                                //méthode pouvant accéder aux propriétés
                                //couleur et poids
                }
                public function se_deplacer()
                {
                                //méthode pouvant accéder aux propriétés
                                //couleur et poids
                }
                public function ajouter_un_kilo()
                {
                                $this->poids = $this->poids + 1;
                }
}?>
```

Lorsque vous allez appeler la méthode ajouter_un_kilo(), cela ajoutera 1 au poids actuel et donc le poids final sera de 11.

La classe

Utilisation de la classe

Dans un premier temps il faut créer un fichier Animal.class.php contenant le code PHP précédent.
Pour utiliser la classe Animal, il faut l'inclure dans la page où vous souhaitez l'appeler.
Créez une page utilisation.php et tapez le code suivant:

```
<?php
include('Animal.class.php');
?>
```

Maintenant que la classe est chargée, il est possible d'instancier la classe, c'est-à-dire de créer un objet ayant comme modèle la classe Animal:

```
<?php
//chargement de la classe
include('Animal.class.php');
//instanciation de la classe Animal
$chien = new Animal();
?>
```

La variable $chien est une instance de la classe Animal avec les attributs couleur, poids et méthodes manger, se_deplacer, ajouter_un_kilo propres à lui-même.

La classe

Mettre à jour et lire les attributs de l'instance

Le principe de l'encapsulation veut que tous les attributs doivent être privés. Il faut donc créer des méthodes publiques permettant de lire ou d'écrire dans ces attributs depuis une autre page PHP.
Ces méthodes sont appelées des accesseurs.

La classe Animal avec les accesseurs est:

```
<?php
class Animal // mot-clé class suivi du nom de la classe.
  {
    // Déclaration des attributs
            private $couleur = "gris";
            private $poids = 10;
            //accesseurs
            public function getCouleur()
            {
                    return $this->couleur; //retourne la couleur
            }
            public function setCouleur($couleur)
            {
                    $this->couleur = $couleur; //écrit dans l'attribut couleur
            }
```

La classe

```
public function getPoids()
		{
			return $this->poids; //retourne le poids
		}
		public function setPoids($poids)
		{
				$this->poids = $poids; //écrit dans l'attribut poids
		}
//méthodes
		public function manger()
		{
				//méthode pouvant accéder aux propriétés
				//couleur et poids
		}

		public function se_deplacer()
		{
				//méthode pouvant accéder aux propriétés
				//couleur et poids
		}
		public function ajouter_un_kilo()
		{
				$this->poids = $this->poids + 1;
		}
} ?>
```

La classe

Les accesseurs sont publiques donc ils permettent de lire ou d'écrire dans les attributs depuis n'importe quelle autre classe ou page PHP.

Exemple avec la page utilisation.php:

```
<?php
//chargement de la classe
include('Animal.class.php');
//instanciation de la classe Animal
$chien = new Animal();
//lire le poids
echo "Le poids du chien est:".$chien->getPoids()." kg<br />";
//ajout d'un kilo au chien
$chien->ajouter_un_kilo();
//lire le poids
echo "Le poids du chien est:".$chien->getPoids()." kg<br />";
//mise à jour du poids du chien
$chien->setPoids(15);
//lire le poids
echo "Le poids du chien est:".$chien->getPoids()." kg<br />" ;
?>
```

Affiche:

Le poids du chien est:10 kg
Le poids du chien est:11 kg
Le poids du chien est:15 kg

La classe

En effet le poids du chien est initialisé à 10. Ensuite la méthode ajouter_un_kilo() ajoute 1 donc son poids devient 11. Enfin la méthode setPoids(15) fixe le poids à 15.

Il est possible de créer autant d'instance que vous voulez.
Par exemple, pour créer un chat blanc de 5kg et un chien noir de 18kg:

```
<?php
//chargement de la classe
include('Animal.class.php');

//instanciation de la classe Animal
$chien = new Animal();

//mise à jour du poids du chien
$chien->setPoids(18);

//lire le poids
echo "Le poids du chien est:".$chien->getPoids()." kg<br />";

//mise à jour de la couleur du chien
$chien->setCouleur("noir");
```

La classe

```
//lire la couleur
echo "La couleur du chien est:".$chien->getCouleur()."<br />";

//instanciation de la classe Animal
$chat = new Animal();

//mise à jour du poids du chat
$chat->setPoids(5);

//lire le poids
echo "Le poids du chat est:".$chat->getPoids()." kg<br />";

//mise à jour de la couleur du chat
$chat->setCouleur("blanc");

//lire la couleur
echo "La couleur du chat est:".$chat->getCouleur()."<br />" ;
?>
```

Affiche:

Le poids du chien est:18 kg
La couleur du chien est:noir
Le poids du chat est:5 kg
La couleur du chat est:blanc

La classe

Passage en paramètre de type objet

$chat et $chien sont des objets de type Animal. Ils peuvent être passés en paramètre d'une méthode à condition que celle-ci accepte ce type d'objet.
Pour tester cet exemple, il faut changer la méthode manger() de la classe Animal. Elle devient manger_animal(Animal $animal_mangé) et prend en paramètre un objet de type Animal.
La page Animal.class.php devient:

```
<?php
class Animal
  {
    // Déclaration des attributs
                private $couleur = "gris";
                private $poids = 10;

                //accesseurs
                public function getCouleur()
                {
                        return $this->couleur; //retourne la couleur
                }
```

La classe

```
public function setCouleur($couleur)
		{
			$this->couleur = $couleur; //écrit dans l'attribut couleur
		}

		public function getPoids()
		{
			return $this->poids; //retourne le poids
		}
		public function setPoids($poids)
		{
			$this->poids = $poids; //écrit dans l'attribut poids
		}
//méthodes
		public function manger_animal(Animal $animal_mange)
		{
			//l'animal mangeant augmente son poids d'autant que
			// celui de l'animal mangé
			$this->poids = $this->poids + $animal_mange->poids;
			//le poids de l'animal mangé et sa couleur son remis à 0
			$animal_mange->poids = 0;
			$animal_mange->couleur = "";
		}
```

La classe

```
public function se_deplacer()
                {
                        //méthode pouvant accéder aux propriétés
                        //couleur et poids
                }

                public function ajouter_un_kilo()
                {
                        $this->poids = $this->poids + 1;
                }
}
?>
```

Pour tester cette méthode, la page utilisation.php devient:

```
<?php

//chargement des classes
include('Animal.class.php');

//instanciation de la classe Animal
$chat = new Animal();
//mise à jour du poids du chat
$chat->setPoids(8);
//lire le poids
```

La classe

```
echo "Le poids du chat est:".$chat->getPoids()." kg<br />";
//mise à jour de la couleur du chat
$chat->setCouleur("noir");
//lire la couleur
echo "La couleur du chat est:".$chat->getCouleur()."<br />";

//instanciation de la classe Animal
$poisson = new Animal();
//mise à jour du poids du poisson
$poisson->setPoids(1);
//lire le poids
echo "Le poids du poisson est:".$poisson->getPoids()." kg<br />";
//mise à jour de la couleur du poisson
$poisson->setCouleur("blanc");
//lire la couleur
echo "La couleur du poisson est:".$poisson->getCouleur()."<br /><br />";
//le chat mange le poisson
$chat->manger_animal($poisson);
//lire le poids
echo "Le nouveau poids du chat est:".$chat->getPoids()." kg<br />";
//lire le poids
echo "Le poids du poisson est:".$poisson->getPoids()." kg<br />";
//lire la couleur
echo "Le couleur du poisson est:".$poisson->getCouleur()."<br /><br />";
?>
```

La classe

Affiche:
Le poids du chat est:8 kg
La couleur du chat est:noir
Le poids du poisson est:1 kg
La couleur du poisson est:blanc
Le nouveau poids du chat est:9 kg
Le poids du poisson est:0 kg
Le couleur du poisson est:

L'objet $chat appelle la méthode manger_animal($poisson) en passant en paramètre l'objet $poisson de type Animal. C'est-à-dire que l'objet $poisson avec ses attributs et ses méthodes sont passés en paramètre. Cela permet de passer plusieurs valeurs en paramètre avec un seul paramètre. La méthode manger_animal(Animal $animal_mangé) accepte uniquement un paramètre de type Animal. Il n'est donc pas possible d'appeler la méthode de cette façon:

```
$chat->manger_animal("Grenouille");
```

Ou de cette façon:

```
$chat->manger_animal(4);
```

Car les types de "Grenouille" (String) et 4 (Integer) ne sont pas de type Animal.

La classe

Le constructeur

Lorsque vous écrivez new Animal(), cela appelle le constructeur par défaut de la classe Animal.
Il est possible de créer vos propres constructeurs et ainsi de pouvoir lui passer en paramètre la valeur des attributs que vous souhaitez affecter à votre objet.
Le constructeur est toujours nommé *__construct* (avec 2 underscores) et ne possède pas de return.

Pour ajouter un constructeur prenant en paramètre le poids et la couleur, la page Animal.class.php devient:

```
<?php
class Animal
   {
// Déclaration des attributs
                private $couleur = "gris";
                private $poids = 10;

                public function __construct ($couleur, $poids=10) // Constructeur demandant 2 paramètres.
                {
                        echo 'Appel du constructeur.<br />';
                        $this->couleur = $couleur; // Initialisation de la couleur.
                        $this->poids = $poids; // Initialisation du poids.
                }
Etc....
} ?>
```

La classe

Le constructeur

```
<?php
class msg{
 private $msg_title; private $msg_content;
private function construct1($content){
                $this->msg_content = $content;
 }
 private function construct2($title, $content){
                $this->msg_title = $title; $this->msg_content = $content;
 }
public function __construct(){
        $numargs = func_num_args();
        $arg_list = func_get_args();
         if ($numargs == 1) {
              construct1($arg_list[0]);
         }
         if ($numargs == 2) {
              construct2($arg_list[0],$arg_list[1]);
        }
        }
}
?>
```

La classe

Appel du constructeur dans la page utilisation.php

```
<?php
//chargement de la classe
include('Animal.class.php');

//instanciation de la classe Animal avec votre constructeur
$chien = new Animal("beige",7);
//lire le poids
echo "Le poids du chien est:".$chien->getPoids()." kg<br />";
//lire la couleur
echo "La couleur du chien est:".$chien->getCouleur()."<br />";
//mise à jour de la couleur du chien
$chien->setCouleur("noir");
//lire la couleur
echo "La couleur du chien est:".$chien->getCouleur()."<br />" ;
?>
```

Affiche:

Appel du constructeur.
Le poids du chien est:7 kg
La couleur du chien est:beige
La couleur du chien est:noir

La classe

Il s'affiche en premier "Appel du constructeur" car l'instruction echo écrite dans le constructeur __construct de votre classe Animal est appelée à chaque fois que vous exécutez new Animal().
Le constructeur prend en paramètre les valeurs de vos attributs, cela évite d'appeler les méthodes setCouleur() et setPoids().

Le destructeur

Le destructeur sert à détruire l'objet pour le libérer de la mémoire. Il est appelé automatiquement à la fin du script PHP ou lorsque l'objet est détruit.
Pour détruire un objet, vous pouvez utiliser la fonction unset(). Cette fonction prend en paramètre l'objet à détruire.
Par exemple, pour détruire l'objet $chien:

```
<?php
//destruction de l'objet
unset($chien);
?>
```

Cela va appeler le destructeur par défaut. Vous pouvez modifier le destructeur par défaut comme le constructeur en ajoutant la fonction __destruct() dans la classe.

La classe

```
<?php
class Animal
{
	// Déclaration des attributs
		private $couleur = "gris";
		private $poids = 10;

		public function __construct ($couleur, $poids) // Constructeur demandant 2 paramètres.
		{
			echo 'Appel du constructeur.<br />';
			$this->couleur = $couleur; // Initialisation de la couleur.
			$this->poids = $poids; // Initialisation du poids.
		}
		public function __destruct()
		{
				echo 'Appel du destructeur.<br />';
		}

Etc...

}
?>
```

La classe

La promotion de la propriété du constructeur
Il permet de réduire la taille de la classe en écrivant une seule fois la propriété.
En PHP7.4

```php
<?php
class BlogPost {
    protected string $Title;
    protected string $Content;
    protected DateTimeImmutable $PostedDate;

    public function __construct(string $Title, string $Content, DateTimeImmutable $PostedDate) {
        $this -> Title = $Title;
        $this -> Content = $Content;
        $this -> PostedDate = $PostedDate;
    }
}
?>
```

En PHP8.0

```php
<?php
class BlogPost {
   public function __construct(
        protected string $Title,
        protected string $Content,
        protected DateTimeImmutable $PostedDate) {}
  }
}
?>
```

La classe

Les constantes de classe

Exemple de déclaration d'une constante normale:

```
define('PI',3.1415926535);
```

Une constante de classe représente une constante mais liée à cette classe.

Lorsque vous créez un animal avec le constructeur __construct ($couleur, $poids), vous tapez:

```
$chien = new Animal("gris",10);
```

Lorsque vous regardez le code, vous ne savez pas immédiatement que 10 représente le poids de l'animal.

Pour cela, vous pouvez utilisez des constantes représentant chacune un certain poids:

```
const POIDS_LEGER = 5;
const POIDS_MOYEN = 10;
const POIDS_LOURD = 15;
```

Les constantes sont toujours en majuscule sans le symbole $ et précédé du mot clé const.

La classe

La classe Animal.class.php devient:

```
<?php
class Animal
  {
    // Déclaration des attributs
                        private striing $couleur = "gris";
                        private int|float $poids = 10;  //possibilité de prendre en charge plusieurs types depuis PHP.8

                        //constantes de classe
                        const POIDS_LEGER = 5;
                        const POIDS_MOYEN = 10;
                        const POIDS_LOURD = 15;
etc...
?>
```

Pour faire appel à cette constante depuis la page utilisation.php, la syntaxe est un peu particulière. Il faut mettre :: entre la classe et sa constante:

```
<?php
//chargement de la classe
include('Animal.class.php');
//instanciation de la classe Animal avec votre constructeur
$poisson1 = new Animal("gris",Animal::POIDS_MOYEN);
$poisson2 = new Animal("rouge",Animal::POIDS_LEGER);
```

La classe

```
//lire le poids
echo "Le poids du poisson1 est:".$poisson1 ->getPoids()." kg<br />";
//lire le poids
echo "Le poids du poisson2 est:".$poisson2->getPoids()." kg<br />" ;
//le poisson1 mange le poisson2
$poisson1->manger_animal($poisson2);
//lire le poids
echo "Le nouveau poids du poisson1 est:".$poisson1->getPoids()." kg<br />";
//lire le nouveau poids
echo "Le nouveau poids du poisson2 est:".$poisson2->getPoids()." kg<br />" ;
?>
```

Affiche:

Appel du constructeur.
Appel du constructeur.
Le poids du poisson1 est:10 kg
Le poids du poisson2 est:5 kg
Le nouveau poids du poisson1 est:15 kg
Le nouveau poids du poisson2 est:0 kg

La classe

Les attributs et méthodes statiques

Méthode statique

La méthode statique est liée à la classe et non à l'objet. Dans l'exemple de la classe Animal, une méthode statique est liée à l'Animal et non aux chiens, aux chats ou aux poissons.
Pour rendre une méthode statique, il faut ajouter le mot clé **static** devant function.
Par exemple, modifier la méthode se_deplacer() pour la rendre statique et afficher "L'animal se déplace.".
La classe Animal.class.php devient:

```
<?php
class Animal
   {       // Déclaration des attributs
                    private $couleur = "gris";
                    private $poids = 10;
//constantes de classe
                    const POIDS_LEGER = 5;
                    const POIDS_MOYEN = 10;
                    const POIDS_LOURD = 15;
public function __construct ($couleur, $poids) // Constructeur demandant 2 paramètres.
                    {
                              echo 'Appel du constructeur.<br />';
                              $this->couleur = $couleur; // Initialisation de la couleur.
                              $this->poids = $poids; // Initialisation du poids.
                    }
```

La classe

```
//accesseurs
		public function getCouleur()
		{
				return $this->couleur; //retourne la couleur
		}
		public function setCouleur($couleur)
		{
				$this->couleur = $couleur; //écrit dans l'attribut couleur
		}

		public function getPoids()
		{
			return $this->poids; //retourne le poids
		}
		public function setPoids($poids)
		{
				$this->poids = $poids; //écrit dans l'attribut poids
		}
//méthodes
		public function manger_animal(Animal $animal_mangé)
		{
				//l'animal mangeant augmente son poids d'autant que
			// celui de l'animal mangé
				$this->poids = $this->poids + $animal_mangé->poids;
```

La classe

```
                        //le poids de l'animal mangé et sa couleur son remis à 0
                        $animal_mangé->poids = 0;
                        $animal_mangé->couleur = "";
                }
public static function se_deplacer()
                        {
                                echo "L'animal se déplace.";
                        }

                        public function ajouter_un_kilo()
                        {
                                $this->poids = $this->poids + 1;
                        }
}
?>
```

Il est impossible de mettre le mot clé $this dans une méthode statique car $this représente l'objet et la méthode statique est liée à la classe.

Pour appeler cette méthode depuis la page utilisation.php, il faut utiliser la même syntaxe que pour les constantes (liées aussi à la classe), c'est-à-dire mettre :: entre la classe et sa méthode statique:

La classe

```php
<?php
//chargement de la classe
include('Animal.class.php');
//appelle de la méthode statique
Animal::se_deplacer();
?>
```

Affiche:
L'animal se déplace.
Il est possible d'appeler une méthode statique à partir d'un objet mais la méthode statique ne peut rien changer sur cet objet:

```php
<?php
//chargement de la classe
include('Animal.class.php');
//instanciation de la classe Animal avec votre constructeur
$chien1 = new Animal("gris",Animal::POIDS_MOYEN);
//appelle de la méthode statique
$chien1->se_deplacer();
?>
```

Affiche:
Appel du constructeur.
L'animal se déplace.

La classe

Attribut statique

Un attribut statique est un attribut propre à la classe et non à l'objet comme pour les méthodes statiques.

Un attribut statique se note en ajoutant le mot clé static devant son nom.

Par exemple, pour ajouter un attribut static représentant un compteur indiquant le nombre de fois où la classe a été instanciée:

La classe Animal.class.php devient:

```
<?php
class Animal
  {
    // Déclaration des attributs
                private $couleur = "gris";
                private $poids = 10;
//constantes de classe
                const POIDS_LEGER = 5;
                const POIDS_MOYEN = 10;
                const POIDS_LOURD = 15;
```

La classe

```
// Déclaration de la variable statique $compteur
                        private static $compteur = 0;

Etc....  ?>
```

Pour changer la valeur de ce compteur, vous ne pouvez pas utiliser $this. En effet, $this représente un objet (chien, chat) et non la classe Animal. Le compteur est de type static donc lié à la classe. Pour faire appel à cet attribut dans la classe, il faut utiliser le mot clé **self** qui représente la classe.

Pour ajouter 1 au compteur à chaque fois que vous allez instancier la classe Animal, il faut modifier le constructeur. Ensuite il faut ajouter une méthode permettant de lire cet attribut privé grâce à une méthode de type public static et de nom getCompteur().

Ajouter dans la classe Animal.class.php :

```
public function __construct ($couleur, $poids) // Constructeur demandant 2 paramètres.
                        {
                        echo 'Appel du constructeur.<br />';
                        $this->couleur = $couleur; // Initialisation de la couleur.
                        $this->poids = $poids; // Initialisation du poids.
                        self::$compteur = self::$compteur + 1;
                        }
// Méthode statique retournant la valeur du compteur
            public static function getCompteur()
            {
                  return self::$compteur;       }
```

La classe

La page utilisation.php:

```
<?php
//chargement de la classe
include('Animal.class.php');
//instanciation de la classe Animal
$chien1 = new Animal("roux",10);
//instanciation de la classe Animal
$chien2 = new Animal("gris",5);
//instanciation de la classe Animal
$chien3 = new Animal("noir",15);
//instanciation de la classe Animal
$chien4 = new Animal("blanc",8);
//appelle de la méthode statique
echo "Nombre d'animaux instanciés:".Animal::getCompteur();
?>
```

Affiche:

Appel du constructeur.
Appel du constructeur.
Appel du constructeur.
Appel du constructeur.
Nombre d'animaux instanciés:4

Exercice 1.1

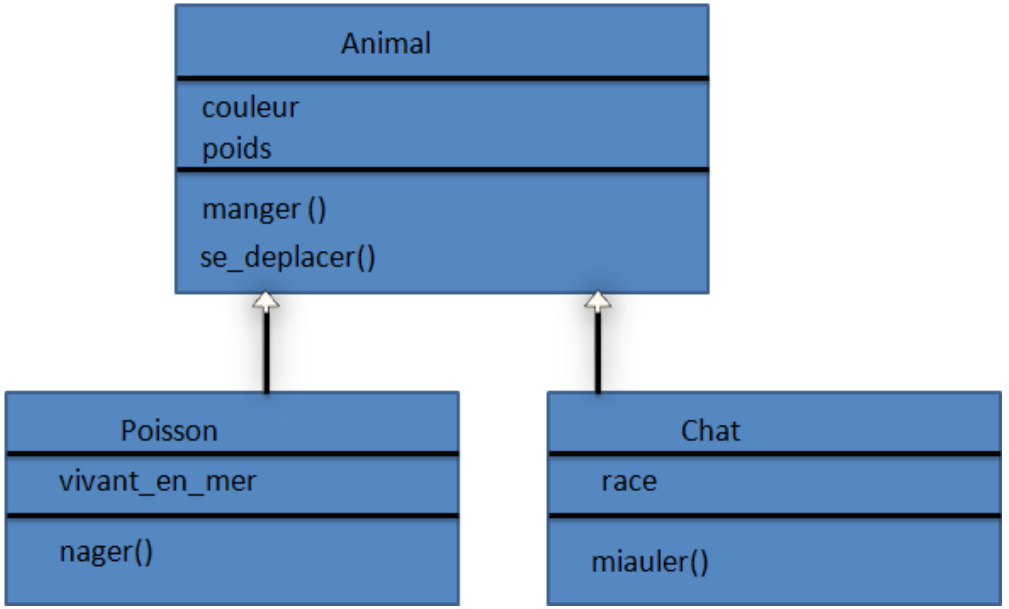
Dans la page utilisation.php, créer deux poissons:
- **poisson1, gris, 10kg**
- **poisson2, rouge, 7kg**

Afficher leur poids puis le poisson1 mange le poisson2.
Réafficher leur poids.
Modifier la méthode manger_animal pour qu'un animal de poids inférieur ne puisse pas manger un animal plus gros.

L'héritage

Introduction

L'héritage est un concept très important en POO. Cela permet de réutiliser le code d'une classe sans le retaper.
Une classe fille hérite d'une classe mère, c'est-à-dire que la classe fille accède alors à tous les attributs et les méthodes publiques de la classe mère.
Par exemple, la classe Mammifère hérite de la classe Animal et la classe Voiture hérite de la classe Véhicule.
Si vous pouvez dire la classe A est une sous-catégorie de la classe B alors vous pouvez certainement faire en sorte que la classe A (Mammifère ou Voiture) hérite de la classe B (Animal ou Véhicule).



L'héritage

Pour créer la classe Poisson qui hérite de la classe Animal, il faut utiliser le mot clé extends entre le nom de la classe fille et le nom de la classe mère.
Créer un fichier Poisson.class.php et taper le code ci-dessous:

```
<?php
class Poisson extends Animal
   {

   }
?>
```

Il faut maintenant lui ajouter un attribut privé correspondant à la variable vivant_en_mer puis les accesseurs get et set et enfin la méthode public nager().

```
<?php
class Poisson extends Animal
   {
                              private $vivant_en_mer; //type du poisson
```

L'héritage

```
 //accesseurs
public function getType()
{
        if ($this->vivant_en_mer) {
                        return "vivant en mer"; //retourne le type
                }
                else if ($this->vivant_en_mer === false) {
                        return "ne vivant pas en mer"; //retourne le type
                }
                else {
                        return "";
                }
}
public function setType($vivant_en_mer)
                        {
                                $this->vivant_en_mer = $vivant_en_mer; //écrit dans l'attribut vivant_en_mer
                        }

                        //méthode
    public function nager()
                        {
                                echo "Je nage <br />";
                        }
                }
?>
```

L'héritage

De la même façon, créer un fichier Chat.class.php:

```
<?php
class Chat extends Animal
   {
                                private $race; //race du chat
//accesseurs
                                public function getRace()
                                {
                                                return $this->race; //retourne la race
                                }
                                public function setRace($race)
                                {
                                                $this->race = $race; //écrit dans l'attribut race
                                }

                                //méthode
                                 public function miauler()
                                {
                                                echo "Miaou <br />";
                                }
}
?>
```

L'héritage

Les classes Chat et Poisson héritant de la classe Animal, ont accès aux attributs publiques de la classe Animal.

La page utilisation.php:

```
<?php
//chargement des classes
include('Animal.class.php');
include('Poisson.class.php');
include('Chat.class.php');

//instanciation de la classe Poisson qui appelle le constructeur de la classe Animal
$poisson = new Poisson("gris",8);
//instanciation de la classe Chat qui appelle le constructeur de la classe Animal
$chat = new Chat("blanc",4);
//lire le poids par l'accesseur de la classe mère
echo "Le poids du poisson est:".$poisson->getPoids()." kg<br />";
//lire le poids par l'accesseur de la classe mère
echo "Le poids du chat est:".$chat->getPoids()." kg<br />";

$poisson->setType(true);
//lire le type par l'accesseur de sa propre classe
echo "Le type du poisson est:".$poisson->getType()."<br />";
//appelle de la méthode de la classe poisson
$poisson->nager();
```

L'héritage

```
$chat->setRace("Angora");
//lire le race par l'accesseur de sa propre classe
echo "La race du chat est:".$chat->getRace()."<br />";
//appelle de la méthode de la classe chat
$chat->miauler();

//appelle de la méthode statique
echo "Nombre d'animaux instanciés:".Animal::getCompteur();

?>
```

Affiche:
Appel du constructeur.
Appel du constructeur.
Le poids du poisson est:8 kg
Le poids du chat est:4 kg
Le type du poisson est:vivant en mer
Je nage
La race du chat est:Angora
Miaou
Nombre d'animaux instanciés:2

L'héritage

Protected

Ce type de visibilité est équivalent à private sauf que les classes filles peuvent voir les attributs protected de la classe mère.
Par exemple, ajouter l'attribut $age de visibilité protected dans la classe mère Animal:

```
<?php
class Animal
   {
      // Déclaration des attributs
                              private $couleur = "gris";
                              private $poids = 10;
                              protected $age = 0;
etc...
```

Cet attribut n'a pas d'accesseur publique donc aucune autre classe que les classes filles héritant d'Animal et la classe Animal ne peuvent modifier ou lire celui-ci.

L'héritage

```
//Dans la classe Poisson
public function afficherAttributs()
    {
            echo "Type:".$this->vivant_en_mer; // ok car privé à cette classe
            echo "<br />";
            echo "Age:".$this->age; // ok, car l'attribut est protégé dans la classe mère.
            echo "<br />";
            echo "Poids:".$this->poids; // erreur car l'attribut est privé
            // dans la classe mère,l'accès est interdit.
            // Il faut passer par ses accesseurs publiques pour modifier
            // ou lire sa valeur
            echo "<br />";
    }
```

Puis pour tester cette méthode, la page utilisation.php devient:

```
<?php

//chargement des classes
include('Animal.class.php');
include('Poisson.class.php');
//instanciation de la classe Poisson qui appelle le constructeur de la classe Animal
$poisson = new Poisson("gris",8);
//lire le poids par l'accesseur de la classe mère
echo "Le poids du poisson est:".$poisson->getPoids()." kg<br />";
```

L'héritage

```
//mise à jour du type du poisson
$poisson->setType(true);
//appelle de la méthode affichant les attributs de la classe poisson et Animal
$poisson->afficherAttributs();

?>
```

Affiche:

Appel du constructeur.
Le poids du poisson est:8 kg
Type:1
Age:0

Notice: Undefined property: Poisson::$poids in **C:\Program Files\EasyPHP-DevServer-13.1VC11\data\localweb\Objet\Poisson.class.php** on line **35**
Poids:

En effet, la classe Poisson n'a pas accès à l'attribut poids de la classe Animal car il est privé.

L'héritage

Substitution

La substitution sert à modifier une méthode déjà existante dans une classe mère afin d'en modifier le comportement. La méthode existe donc dans deux classes différentes et c'est celle de la classe fille ou de la classe mère qui est exécutée suivant le contexte.
Par exemple, pour substituer la méthode manger_animal(Animal $animal_mangé) de la classe Animal afin d'initialiser le type du poisson mangé, il faut ajouter cette même méthode dans la classe Poisson et l'implémenter autrement:
Ajouter dans la classe Poisson.class.php:

```
//méthode substituée
        public function manger_animal(Animal $animal_mangé)
        {
                if(method_exists($animal_mangé, "setRace")) {
                    $animal_mangé->setRace("");
                }
                if (isset($animal_mangé->vivant_en_mer)){
                    $animal_mangé->vivant_en_mer="")
                }
        }
```

L'héritage

Le problème est que cette méthode intialise correctement l'attribut vivant_en_mer du poisson mangé mais n'initialise plus son poids et sa couleur. Vous ne pouvez pas changer son poids et sa couleur ici car ces attributs sont privés dans la classe Animal.
La solution est d'appeler la méthode manger_animal(Animal $animal_mangé) de la classe Animal dans la méthode manger_animal(Animal $animal_mangé) de la classe Poisson:

```
//méthode substituée
        public function manger_animal(Animal $animal_mangé)
        {
                // Appelle la méthode manger_animal() de la classe parente, c'est-à-dire Animal
                parent::manger_animal($animal_mangé);
                if(method_exists($animal_mangé, "setRace")) {
                        $animal_mangé->setRace("");
                    }
                    if (isset($animal_mangé->vivant_en_mer)){
                        $animal_mangé->vivant_en_mer="")
                    }
                    }
```

parent est un mot clé désigant la classe mère, c'est-à-dire la classe Animal.

L'héritage

La page utilisation.php:

```
<?php
//chargement des classes
include('Animal.class.php');
include('Poisson.class.php');
//instanciation de la classe Poisson qui appelle le constructeur de la classe Animal
$poisson = new Poisson("gris",8);
//mise à jour du type du poisson
$poisson->setType(true);
//instanciation de la classe Poisson qui appelle le constructeur de la classe Animal
$autre_poisson = new Poisson("noir",5);
//mise à jour du type du poisson
$autre_poisson->setType(false);
//appelle de la méthode affichant les attributs de la classe Poisson et Animal
$poisson->manger_animal($autre_poisson);
//lire le type par l'accesseur de sa propre classe
echo "Le type du poisson mangé est:".$autre_poisson->getType()."<br />";
//lire le poids par l'accesseur de la classe mère
echo "Le poids du poisson mangé est:".$autre_poisson->getPoids()." kg<br />" ; ?>
```

Affiche:

```
Appel du constructeur.
Appel du constructeur.
Le type du poisson mangé est:
Le poids du poisson mangé est:0 kg
```

L'héritage

Le poids a été initialisé à 0 par la méthode manger_animal(Animal $animal_mangé) de la classe Animal.
Il est possible de connaître la classe d'un objet avec : get_class($object),

Depuis PHP 8.0, il est possible de substituer une méthode avec des arguments variadiques du moment que les types sont compatibles :

```
<?php
class A {
    public function method(int $many, string $parameters, $here) {}
}
class B extends A {
    public function method(...$everything) {}
}
?>
```

De plus, il est aussi possible depuis PHP8 d'ajouter une virgule optionnelle à la fin de la liste d'arguments :

```
function functionWithLongSignature(
    Type1 $parameter1,
    Type2 $parameter2, // ← Cette virgule est permise.
) { }
```

L'héritage

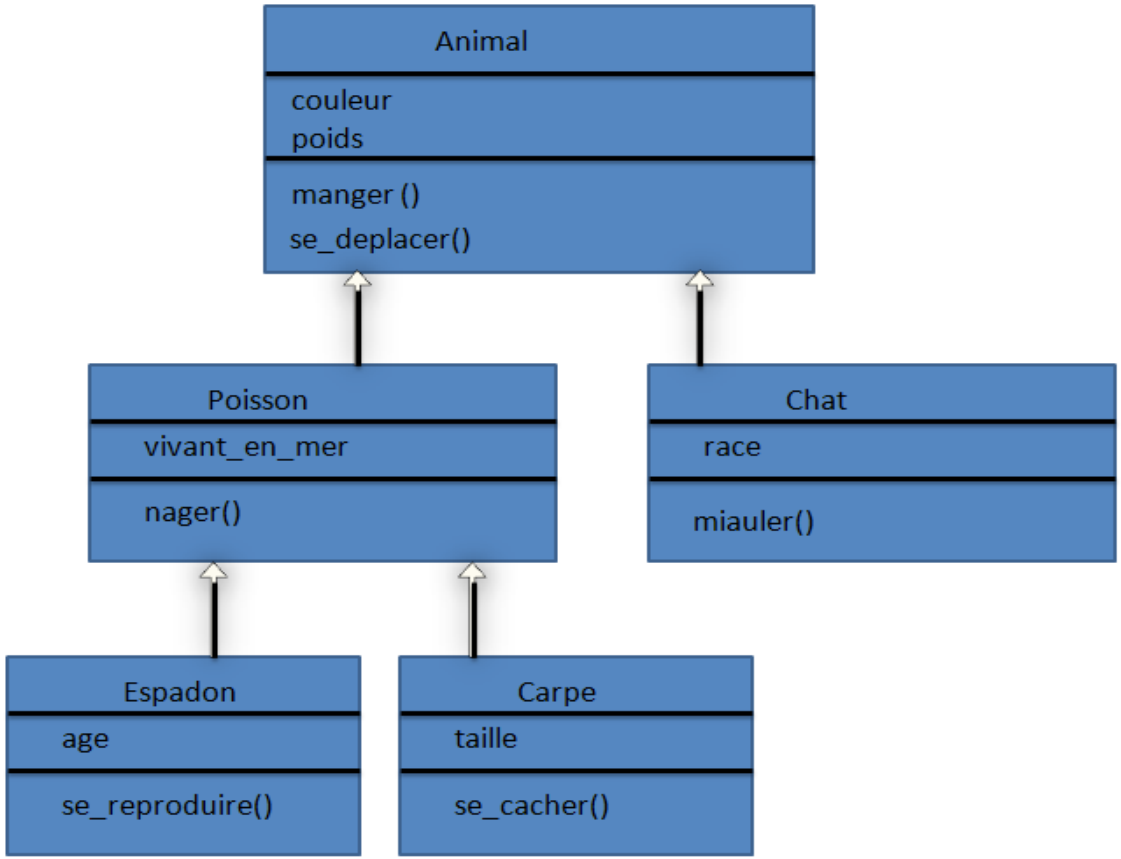
Cet exemple montre que les classes Espadon et Carpe héritent de la classe Poisson qui hérite de la classe Animal.

La classe Espadon accède à tous:
• les attributs et méthodes privés, protégés et publiques d'elle-même

• les attributs et méthodes protégés et publiques de la classe Poisson

• les attributs et méthodes protégés et publiques de la classe Animal

• les attributs et méthodes publiques **statiques** de la classe Chat



Les classes abstraites

Les classes abstraites s'écrivent en ajoutant le mot clé abstract devant le mot clé class. Une classe abstraite ne peut pas être instanciée, c'est-à-dire qu'il n'est pas possible de créer une instance. Ce sont des méthodes dont uniquement la signature est écrite, précédée du mot-clé abstract:
abstract visibilité function nomMethode(attribut type_attribut,...)

Dans l'exemple suivant, la classe Animal est abstraite car vous ne voulez pas créer (instancier) des animaux mais uniquement des poissons ou des chats.
Ajouter aussi une méthode abstraite respire() dans la classe Animal:

```
<?php
abstract class Animal
  {
    // Déclaration des attributs
                                private $couleur = "gris";
                                private $poids = 10;
                                //constantes de classe
                                const POIDS_LEGER = 5;
                                const POIDS_MOYEN = 10;
                                const POIDS_LOURD = 15;

                                // Déclaration de la variable statique $compteur
                                private static $compteur = 0;
```

Les classes abstraites

```
public function __construct($couleur, $poids) // Constructeur demandant 2 paramètres.
                {
                echo 'Appel du constructeur.<br />';
                $this->couleur = $couleur; // Initialisation de la couleur.
                $this->poids = $poids; // Initialisation du poids.
                self::$compteur = self::$compteur + 1;
                }

                //accesseurs
public function getCouleur()
                {
                        return $this->couleur; //retourne la couleur
                }
public function setCouleur($couleur)
                {
                                $this->couleur = $couleur; //écrit dans l'attribut couleur
                }
public function getPoids()
                {
                                 return $this->poids; //retourne la poids
                }
public function setPoids($poids)
                {
                                $this->poids = $poids; //écrit dans l'attribut poids
                }
```

Les classes abstraites

```
//méthodes publiques
                public function manger_animal(Animal $animal_mangé)
                {
                        //l'animal mangeant augmente son poids d'autant que
                        // celui de l'animal mangé
                        $this->poids = $this->poids + $animal_mangé->poids;
                        //le poids de l'animal mangé et sa couleur son remis à 0
                        $animal_mangé->poids = 0;
                        $animal_mangé->couleur = "";
                }
                public static function se_deplacer()
                {
                        echo "L'animal se déplace.";        }
                public function ajouter_un_kilo()
                {
                        $this->poids = $this->poids + 1;    }
// Méthode statique retournant la valeur du compteur
                public static function getCompteur()
                {
                        return self::$compteur;
                }
//code non implémenté car méthode abstraite
                abstract public function respire();
}
?>
```

Les classes abstraites

Vous constatez que la méthode abstraite respire() n'a pas de corps, c'est-à-dire qu'il n'y a pas d'accolades avec l'implémentation de la méthode.

Comme les classe Poisson et Chat héritent de la classe Animal, vous êtes obligé de redéfinir la méthode respire() dans les classes Poisson et Chat.

Il faut ajouter dans la classe Poisson:

```
public function respire()
{
        echo "Le poisson respire.<br />";
}
```

Et dans la classe Chat:

```
public function respire()
{
        echo "Le chat respire.<br />";
}
```

Les classes abstraites

La page utilisation.php devient:

```
<?php

//chargement des classes
include('Animal.class.php');
include('Poisson.class.php');
include('Chat.class.php');

//instanciation de la classe Poisson qui appelle le constructeur de la classe Animal
$poisson = new Poisson("gris",8);
//instanciation de la classe Chat qui appelle le constructeur de la classe Animal
$chat = new Chat("blanc",4);
//appelle de la méthode respire()
$poisson->respire();
$chat->respire();

?>
```

Affiche:
Appel du constructeur.
Appel du constructeur.
Le poisson respire.
Le chat respire.

Les classes finales

Lorsque une classe est finale, vous ne pouvez pas créer de classe fille héritant de cette classe. Cela n'a guère d'intérêt en pratique.
Il faut pour cela ajouter le mot clé final devant le mot clé class.
Par exemple, si vous ne créez pas de classe héritant de la classe Poisson, vous pouvez la mettre final:

```
<?php
final class Poisson extends Animal
  {
                        private $vivant_en_mer; //type du poisson
                        //accesseurs
                        etc.
                        .

                         //méthode
                        public function nager()
                        {
                                    echo "Je nage <br />";
                        }
                        public function respire()
                        {
                                    echo "Le poisson respire.<br />";
                        }
            }
?>
```

Les classes finales

Il est possible aussi de déclarer des méthodes finales. Ces méthodes ne pourront pas alors être substituées. Vous avez vu dans le paragraphe sur la substitution un exemple où la méthode manger_animal(Animal $animal_mangé) a été substituée dans la classe Poisson. Si cette méthode était finale dans la classe Animal:

```
final public function manger_animal(Animal $animal_mangé)
                                {
                                        //l'animal mangeant augmente son poids d'autant que
        // celui de l'animal mangé
                                        $this->poids = $this->poids + $animal_mangé->poids;
                                        //le poids de l'animal mangé et sa couleur son remis à 0
                                        $animal_mangé->poids = 0;
                                        $animal_mangé->couleur = "";
                                }
```

Il est alors impossible de substituer cette méthode dans la classe Poisson.

readonly

Il est possible de rendre une propriété dans une classe en lecture seule.
Vous ne pouvez écrire qu'une seule fois dans cette propriété.

```
class User {
    public readonly int $uid;

    public function fetch(int $uid) {
        $this->uid = $uid;
    }
}

$user = new User();
$user→fetch(42); //ok
$user→fetch(16); //remonte une erreur
```

Type de retour

Si la fonction retourne un type string, il faut ajouter :string après les parenthèses.
Si la fonction retourne un string ou bien null, il faut ajouter :?string après les parenthèses.
Par exemple :

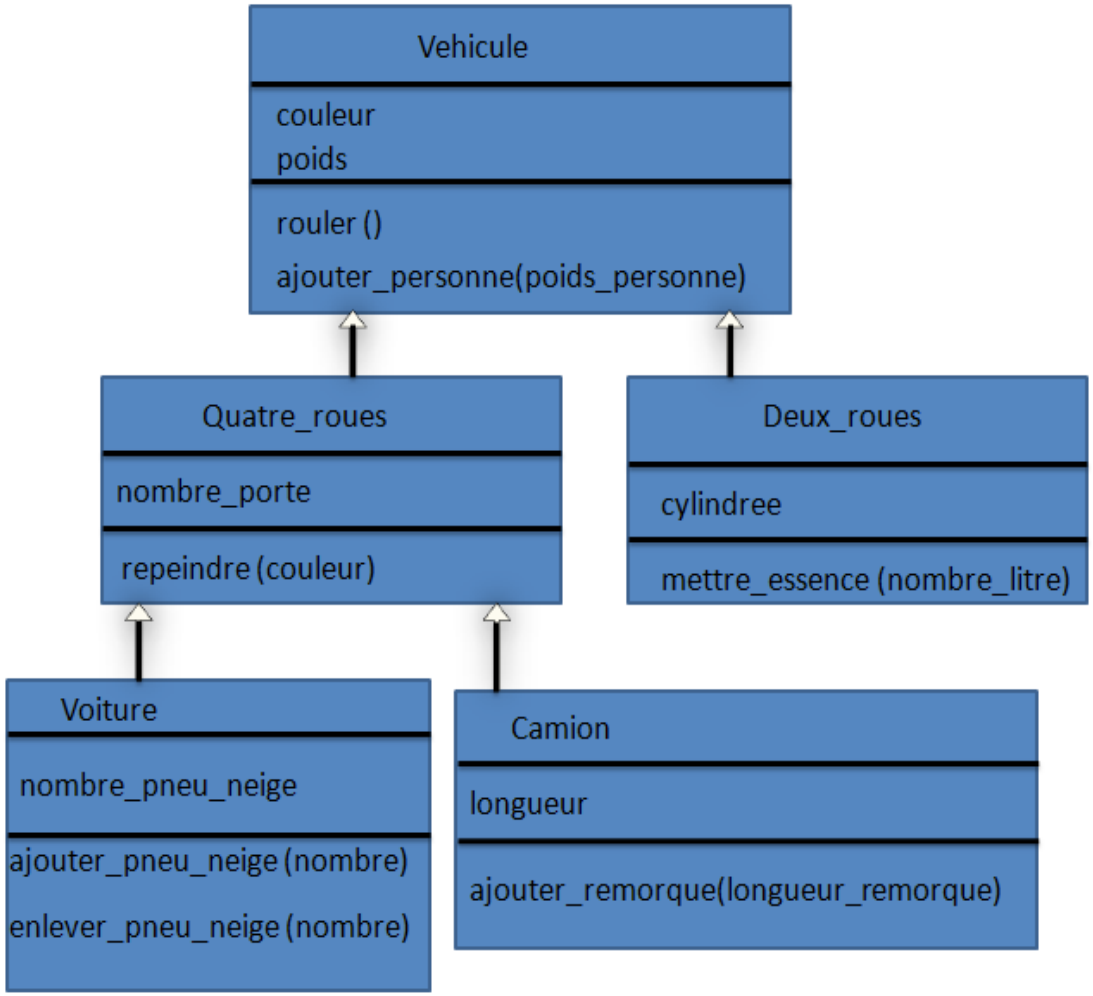
```
function testReturn(): ?string
{
    return null;
}
```

Si la fonction ne retourne rien, il faut ajouter :void après les parenthèses.

Enfin, si la fonction stop le programme avec un exit par exemple, il faut ajouter :never. Ce mot clé existe depuis PHP8.1

Exercice 2.1

Créer les cinq classes du schéma suivant en tenant compte de leur héritage.
Toutes les méthodes sont publiques et les attributs privés.



Exercice 2.2

Créer les accesseurs de tous les attributs. Créer un constructeur dans la classe Vehicule prenant en paramètre la couleur et le poids. Modifier la méthode rouler() pour qu'elle affiche "Le véhicule roule". Modifier la méthode ajouter_personne(poids_personne) pour qu'elle change le poids du véhicule en fonction du poids de la personne passé en paramètre.
Créer une page affichage.php créant un véhicule noir de 1500 kg. Le faire rouler.
Ajouter une personne de 70 kg et afficher le nouveau poids du véhicule.

Exercice 2.3

Implémenter la méthode repeindre(couleur) pour changer la couleur définie dans la classe Véhicule. Implémenter la méthode mettre_essence(nombre_litre) pour changer le poids défini dans la classe Véhicule. Pour cet exercice, un litre correspond à un kilogramme.
Implémenter les méthodes ajouter_pneu_neige(nombre) et enlever_pneu_neige(nombre) modifiant l'attribut nombre_pneu_neige.
Implémenter la méthode ajouter_remorque(longueur_remorque) modifiant l'attribut longueur.
Dans la page affichage.php, créer une voiture verte de 1400 kg. Ajouter deux personnes de 65 kg chacune.
Afficher sa couleur et son nouveau poids.
Repeindre la voiture en rouge et ajouter deux pneus neige.
Afficher sa couleur et son nombre de pneus neige.

Exercice 2.3 suite

Créer un objet Deux_roues noir de 120 kg. Ajouter une personne de 80 kg. Mettre 20 litres d'essence.
Afficher la couleur et le poids du deux-roues.
Créer un camion bleu de 10000 kg d'une longueur de 10 mètres avec 2 portes. Lui ajouter une remorque de 5 mètres et une personne de 80 kg.
Afficher sa couleur, son poids, sa longueur et son nombre de portes.

Exercice 2.4

Rendre la classe Vehicule et sa méthode ajouter_personne(poids_personne) abstraites.
Définir la méthode ajouter_personne(poids_personne) dans la classe Deux_roues pour que cette méthode ajoute le poids de la personne plus 2 kg correspondant au casque du deux-roues.
Définir la méthode ajouter_personne(poids_personne) dans la classe Quatre_roues pour effectuer la même chose qu'elle faisait dans la classe Vehicule.
Créer une méthode publique statique dans la classe Vehicule s'appelant afficher_attribut.
Cette méthode prend un objet en paramètre et affiche la valeur de tous ses attributs (si ils existent), c'est-à-dire la couleur, le poids, le nombre de portes, la cylindrée, la longueur et le nombre de pneus neige.
(method_exists($objet,'getCouleur'))
Dans la page affichage.php créer un deux-roues rouge de 150 kg.

Exercice 2.4 suite

Ajouter une personne de 70 kg et afficher son poids total.
Changer la couleur du deux-roues en vert. Lui affecter une cylindrée de 1000.
Afficher toutes les valeurs des attributs du deux-roues à l'aide de la fonction afficher_attribut.
Créer un camion blanc de 6000 kg.
Lui ajouter une personne de 84 kg. Le repeindre en bleu. Lui affecter 2 portes.
Afficher toutes les valeurs des attributs du camion à l'aide la fonction afficher_attribut.

Exercice 2.5

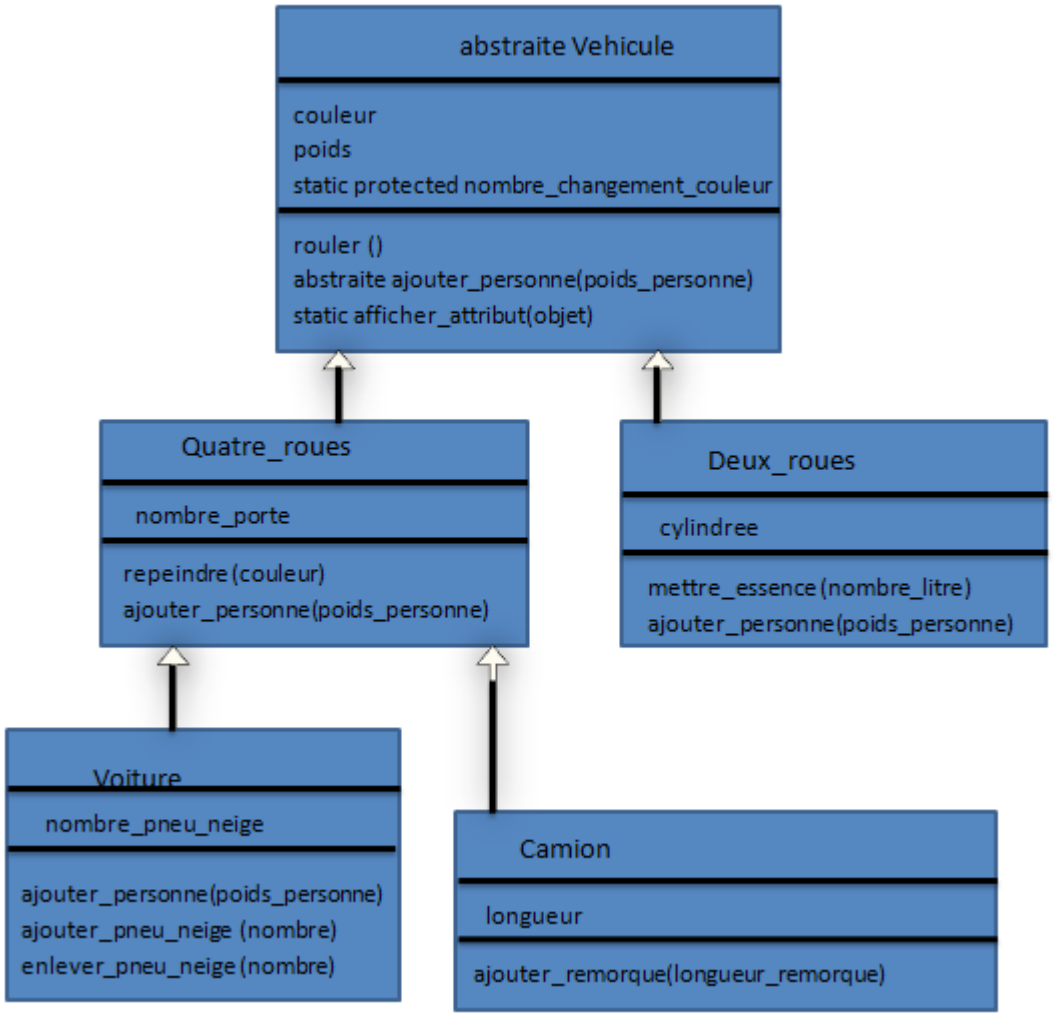
Ajouter un constructeur dans la classe Quatre_roues prenant en paramètre la couleur, le poids et le nombre de portes.
Substituer la méthode publique ajouter_personne(poids_personne) dans la classe Voiture. Cette méthode exécute la méthode ajouter_personne(poids_personne) de la classe Quatre_roues et affiche "Attention, veuillez mettre 4 pneus neige." si le poids total du véhicule est supérieur ou égal à 1500 kg et si il y a 2 pneus neige ou moins.
Ajouter une constante SAUT_DE_LIGNE ='
' dans la classe Vehicule et modifier la méthode afficher_attribut($objet) pour remplacer les '
'.
Ajouter un attribut protected static dans cette classe s'appelant nombre_changement_couleur et l'initialiser à 0. Cet attribut représente le nombre de fois que la couleur va changer quelque soit l'objet dont vous changez la couleur. Ce changement de couleur s'effectue dans la méthode setCouleur().

Exercice 2.5 suite

Changer l'accesseur setPoids() de la classe Vehicule pour que le poids total du véhicule fasse au maximum 2100 kg.
Dans la page affichage.php créer une voiture verte, 2100 kg avec 4 portes.
Ajouter 2 pneus neige puis ajouter une personne de 80 kg.
Changer la couleur de la voiture en bleu.
Enlever 4 pneus neige.
Repeindre la voiture en noire
Afficher tous les attributs de la voiture et le nombre de fois où la couleur a été changée avec la méthode afficher_attribut($objet).

Exercice 2.5 suite

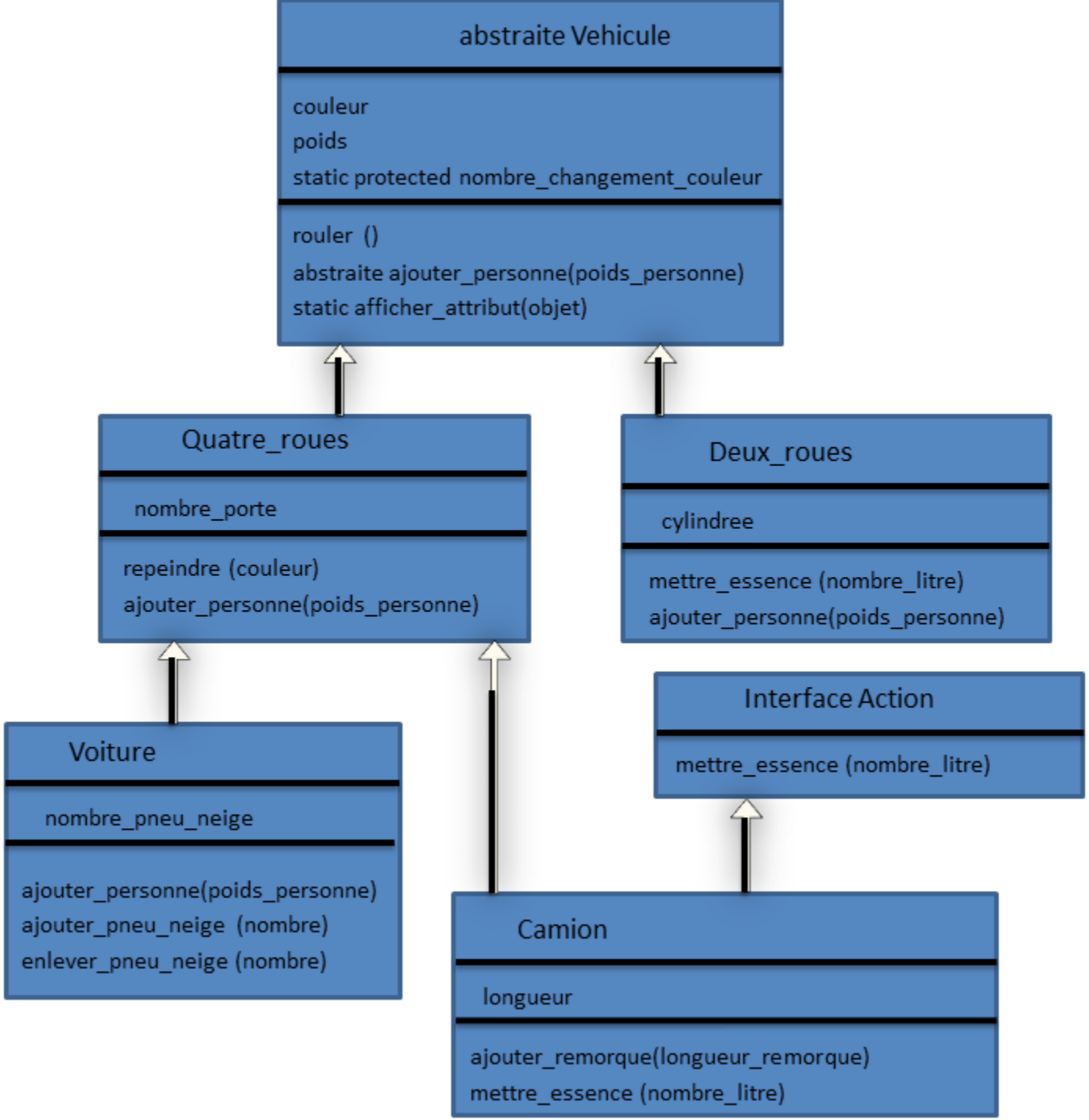
Le nouveau modèle est:



Les accesseurs ne sont pas représentés.

Exercice 2.6

Créer une interface Action contenant la signature de méthode mettre_essence(int $nombre_litre) :void
Modifier la classe Camion pour qu'elle implémente l'interface Action et donc redéfinisse la méhode mettre_essence(int $nombre_litre) :void comme dans la classe Deux_roues.
Dans la page affichage.php créer un camion bleu, 10000 kg avec 2 portes.
Fixer la longueur à 10 m.
Mettre 100 litres d'essence.
Repeindre le camion en vert.
Afficher tous les attributs de la voiture et le nombre de fois où la couleur a été changée avec la méthode afficher_attribut($objet).
Le nouveau modèle est :


abstraite Vehicule
couleur
poids
static protected nombre_changement_couleur
rouler ()
abstraite ajouter_personne(poids_personne)
static afficher_attribut(objet)
Quatre_roues
nombre_porte
repeindre (couleur)
ajouter_personne(poids_personne)
Deux_roues
cylindree
mettre_essence (nombre_litre)
ajouter_personne(poids_personne)
Interface Action
mettre_essence (nombre_litre)
Voiture
nombre_pneu_neige
ajouter_personne(poids_personne)
ajouter_pneu_neige (nombre)
enlever_pneu_neige (nombre)
Camion
longueur
ajouter_remorque(longueur_remorque)
mettre_essence (nombre_litre)

Namespaces

Un namespace est déclaré avec le mot-clé namespace suivit de son nom en début de fichier.
Par exemple :
Espace_nom.php

```
<?php
// Définition de l'espace de noms.
namespace Biliotheque;
// Définition d'une constante.
const PI = 3.1416;
// Définition d'une fonction.
function maFonction() {
echo "Bonjour";
}
// Définition d'une classe.
class maClasse {
/*
...
*/
}
?>
```

Namespaces

Utilisation_espace_nom.php

```
<?php
include('espace_nom.php');
Biliotheque\maFonction(); //Appel du namespace Biliotheque à la racine
?>
```

Affiche :

Bonjour

La constante __NAMESPACE__ retourne le nom de l'espace de noms courant.
Il est possible de créer des sous-espaces de nom en écrivant :
namespace Espace1\SousEspace1 ;
Les chemins pour trouver une fonction, une classe ou une constante dans un espace de nom sont relatifs si vous commencez par le namespace ou absolus si vous commencez par \.

Par exemple :
Espace_nom.php

```
<?php
// Définition de l'espace de noms.
```

Namespaces

```
namespace Biliotheque;
// Définition d'une constante.
const PI = 3.1416;
// Définition d'une fonction.
function maFonction() {
                    echo "Bonjour <br />";
}

// Définition d'une classe.
class Animal
   {
      // Déclaration des attributs
      private $couleur = "gris";
                    //accesseurs
       public function getCouleur()
       {
          return $this->couleur; //retourne la couleur
       }
       public function setCouleur($couleur)
       {
          $this->couleur = $couleur; //écrit dans l'attribut couleur
       }
}
?>
```

Namespaces

Utilisation_espace_nom.php

```
<?php
namespace Projet;
include('espace_nom.php');
// Affichage de l'espace de noms courant.
echo 'Espace de noms courant = ', __NAMESPACE__,'<br />';
\Bilioteque\maFonction(); //Appel du namespace Biliotheque à la racine
echo \Bilioteque\PI."<br />";
$chien = new \Biliotheque\Animal();
$chien->setCouleur("noir");
echo "La couleur du chien est:".$chien->getCouleur();
?>
```

Affiche :

```
Espace de noms courant = Projet
Bonjour
3.1416
La couleur du chien est:noir
```

Namespaces

Enfin, vous pouvez créer un alias sur un espace de nom ou sur un objet contenu dans cet espace nom.
Il suffit d'utiliser l'opérateur use [namespace] as nouveau_nom
Par exemple : use \Biliotheque as biblio;

Avec les alias, la page Utilisation_espace_nom.php devient

```
<?php
namespace Projet;
include('espace_nom.php'); // Affichage de l'espace de noms courant.
echo 'Espace de noms courant = ', __NAMESPACE__,'<br />';
\Bilioteheque\maFonction(); //Appel du namespace Biliotheque à la racine
use \Biliotheque as biblio; // alias d'un namespace
echo biblio\PI."<br />";
use \Biliotheque\Animal as ani; // alias d'une classe
$chien = new ani(); //Appel de l'alias de la classe Animal$chien->setCouleur("noir");
echo "La couleur du chien est:".$chien->getCouleur();
?>
```

Affiche :

Espace de noms courant = Projet
Bonjour
3.1416
La couleur du chien est:noir

Exo

A partir du blog écrit en objet, ajouter les fonctions de suppression et de modification.