Classes en PHP

Youssef CHAHIR

Objet et instance

- Un Objet est un Concept.

```
class Voiture {
        private $couleur;
        …
        public function __construct($couleur) {$this->couleur=$couleur;}
        public function getCouleur() {..}
}
```

  - Par défaut, la visibilité est fixée à public.
  - Constructeur est appelé lorsqu'on utilise le mot clé new.

- Une instance est une implémentation de ce concept.

```
$ma_voiture = new Voiture(param)
```

Classes - Concepts

- PHP permet de faire de la programmation orientée objets
- Objet signifie (exemple de la voiture)
  - Attributs
    - caractéristiques de l 'objet (propriétés, constantes)
    - couleur, nombre de portes
  - Méthode
    - actions qui s 'appliquent à l 'objet
    - démarrer, freiner
  - Héritage
    - l 'objet descend d 'un autre, plus général, dont il va hériter des attributs et méthodes
    - limousine descend de voiture, mais aura en plus l 'attribut « contenance du frigo » et la méthode « démarrer la climatisation »
  - Polymorphisme
    - Capacité d 'un ensemble d 'objets à exécuter des méthodes de même nom, mais dont le comportement est propre à chacune des versions

Classes

Quelques fonctions :

get_declared_classes() : retourne un tableau listant toutes les classes définies

class_exists($str) : vérifie qu'une classe dont le nom est passé en argument a été définie

get_class($obj), **get_parent_class** : retournent le nom de la classe de l'objet **$obj**

get_class_methods($str) : retourne les noms des méthodes de la classe **$str** dans un tableau

get_class_vars($str) : retourne les valeurs par défaut des attributs de la classe **$str** dans un tableau associatif

get_object_vars($obj) : retourne un tableau associatif des attributs de l'objet **$obj** les clés sont les noms des attributs et les valeurs, celles des attributs si elles existent

is_subclass_of($obj,$str) : détermine si l'objet **$obj** est une instanciation d'une sous-classe de **$str**, retourne VRAI ou FAUX

method_exists($obj,$str) : vérifie que la méthode **$str** existe pour une classe dont **$obj** est une instance, retourne VRAI ou FAUX

Constante d'un objet

- Le mot clef const permet de définir une constante de classe, c.à.d qui sera partagée par toutes les instances de la classe.
- Attribut ne peut pas être appel avec $this-> mais par self::
- Pas de $ pour les constantes.

```
class R5 {
const MARQUE=”renault”;
public function Afficher_Constante() {    echo  self::MARQUE;  }
…
}
echo R5::MARQUE;
```

Membres statiques

- Le mot clef static permet de définir une constante ou une méthode qui peuvent être appelées sans avoir instancié la classe
- Un attribut statique a la même valeur pour toutes les instances de la classe
- Attribut ou méthode statique : ne peut pas être appelé(e) avec $this->, mais par self::

```
class staticEx {
    private static $compteur = 0;
    public function __construct() { self::$compteur++;}
    public static function Afficher() { }
    public function __destruct() { echo "Destruction d'une instance"; self::$compteur--;}
}
echo staticEx::Afficher() ;
```

Héritage

- Le but de l'héritage est de permettre l'écriture de classe dérivant d'une autre afin de partager les attributs et les méthodes d'une classe à une autre.
- L'héritage permet de représenter une hiérarchie entre différents objets et de définir des notions de spécialisation. Ainsi, on peut simplfier le code et le rendre plus facilement maintenable : ⇒ En factorisant le code
- La classe la plus générale est appelée la classe mère et les classes héritant de celle-ci sont nommées classes filles.
- Pour indiquer qu'une classe hérite d'une autre, il faut utiliser le mot clef extends.
- Les méthodes et membres hérités peuvent être surchargés, à moins que la classe parent ait défini une méthode comme final.
- Pour surcharger, il suffit de redéclarer la méthode avec le même nom que celui défini dans la classe parent. Il est possible d'accéder à une méthode ou un membre surchargé avec l'opérateur parent::

Héritage simple

```
class Voiture {
   private $carburant ;
   public function __construct($carburant) {$this->carburant=$carburant;}
       public function getCarburant {return $this->carburant;}
}
// extension de la classe
class Decapotable extends Voiture {
   private $toit ;
  // Redéfinition de la méthode parent
   public function __construct($carburant) {
                echo "Classe étendue";
                parent::__construct($carburant);
                $this->toit= "fermé" ;
   }
   public function AfficherCarburant {echo "Je fonctionne avec du". $this->getCarburant();}
}
$maDecapotable = new Decapotable(”diesel”) ;
echo ”ma décapotable consomme de ”.$maDecapotable->getCarburant() ;
```

Interface

- Une interface permet d'imposer à un objet d'implémenter un ensemble de fonctions.
- Une interface est le prototype d'une classe. Elle n'implémente pas les méthodes.
- Un véhicule doit pouvoir :
  - Démarrer
  - s'arrêter
- Ainsi pour tous véhicules on va vouloir imposer que ceux-ci implémentent les fonctions demarre() et arrete().

- Pour créer une interface on utilise le mot clef interface et dans le bloc définissant l'interface la liste des fonctions qui verront être implémentées.

```
interface demarrable {
    public function demarre() ;
    public function arrete() ;
}
```

Interface

- Pour forcer un objet à implementer les fonctions d'un interface on doit utiliser le mot clef implements.

```
class Voiture implements demarrable {
private $carburant ;
public function   construct($carburant) {...}
public function getCarburant() {...}
public function demarre() { echo ”La voiture d´emarre” ; }
public function arrete() { echo ”La voiture s'arrête” ; }
}
```

- Un Objet ne peut hériter que d'un seul objet (en PHP) mais peut implémenter plusieurs interfaces. Dans un tel cas on sépare les interfaces par une virgule.

```
class Voiture implements demarrable, lavable { ...}
```

Classe et méthode abstraite

- Il est possible de définir des classes abstraites permettant de représenter des objets qui ne possèdent pas d'identités mais qui représentent des concepts regroupant d'autres concepts.
- Toutes les classes contenant au moins une méthode abstraite doivent également être abstraites.
- ⇒ classe abstraite ≡ classe mère ne pouvant pas être instanciée.
- Le mot clé permettant de définir une classe abstraite est abstract.
- Exemple : Fruit est concept important ne devant pas être instancié. Il regroupe tous les fruits comme les pommes (etc...) qui peuvent être instanciées et manipulées.
- Un autre intérêt majeur d'une classe abstraite est de définir des méthodes que toutes les classes filles devront implémenter. Ces méthodes ne peut être implémenter dans la classe abstraite en indiquant dans celle-ci son prototype et en la déclarant comme abstraite.
- Lors de l'héritage depuis une classe abstraite, toutes les méthodes marquées comme abstraites dans la déclaration de la classe parent doivent être définies par l'enfant .

Classe et méthode abstraite

- Exemple :

```
abstract class Fruit {
   abstract public function mange();
   public function Afficher() {     print $this->mange();   }
}
class Pomme extends Fruit {
   public function mange() {   return "Pomme";   }
}
class Poire extends Fruit {
   public function mange() {     return "Poire";   }
}
$classe1 = new Pomme;  $classe1->Afficher();
$classe2 = new Poire;  $classe2->Afficher();
```

Parent et Self

```
abstract class Fruit {
 protected $couleur ;
  public function __construct() {...}
  public function getCouleur() {...}
}
class Pomme extends Fruit {
   private $pepin ;
   public function construitUnePomme() {self::__construct() ;  //parent::construct() ;}
   public function afficheCouleur() { echo ”Ma pomme est ”.$this->couleur ;}
}
$golden = new Pomme() ;
echo ”La couleur de la pomme est ”.$golden->getCouleur() ;
```

- Le mot clef parent permet de référencer la classe mère d'un objet. Il permet de faire appell de manière explicite à un attribut ou méthode de la classe mère.
- Le mot clef self permet de référencer la classe courrante.
- ! ne pas confondre self et this.

Exercice

- Héritage
- Créer une classe « vehicule » qui va regrouper toutes les méthodes et propriétés communes à tous les types de véhicules
  - Propriétés : couleur, puissance, nb_porte, vitesse, etat_moteur
  - Méthodes : rouler, demarrer, arreter, freiner, accelerer + constructeur
- Créer 2 autres classes, « voiture » et « camion », qui héritent de la classe « vehicule ». Celles-ci vont posséder des méthodes et propriétés qui leur sont propres
- D'autres classes peuvent hériter de la classe voiture:
  - QuatreQuatre, Propulsion, Traction
- Ecrire un programme principal permettant d ’utiliser les classes « voiture » et « camion ».