Chapitre 4 : notions élémentaires sur le modèle objet

Notion d'objet

- Beaucoup de langages sont Orientés Objets (OO)
  - C++, Java, JavaScript, Python, PHP, etc
- Un objet est une entité logicielle qui possède :
  - des **attributs**
  - des **méthodes**
- **Attributs** :
  - Son "savoir"
  - L'état de l'objet
  - Comme un ensemble de 'variables'
  - PHP : techniquement similaire à un tableau (Array)
- **Méthodes :**
  - Son "savoir faire"
  - Action sur l'objet pour changer son état
  - Comme un ensemble de 'fonctions' mais sur l'objet

Modèle à base d'objets : "personification"

- Voir les objets comme des 'entités' autonomes d'une même famille (leur classe ou leur prototype)
- Voir les échanges entre objets comme des demandes entre objets qui s'envoient des **messages**
- Messages simples : "donnes moi ton nom"
  - utilise les attributs
  - des "getters", des "setters"
  - possible simplification en accédant directement à l'attribut
- Messages calculés : "bonjour paragraphe, combien as-tu de mots ?"
  - Nécessite un calcul
- Messages complexes : "bonjour document, combien as-tu de mots ?"
  - Nécessite d'envoyer d'autres messages à d'autres objets, par exemple si le document est composé de paragraphes.

Les différents modèles à Objets

- Le "tout objet" :
  - Tout est un objet, même les fonctions (sauf les valeurs)
  - Adapté à la gestion de connaissances ou aux interfaces
  - Partage de propriétés soit :
    - Dans un objet prototype
    - Dans une classe qui est une instance d'une méta-classe
- Le "pas tout objet" :
  - Seul les instances de classes sont des objets
    - Les classes ne sont pas des objets
  - Adapté à la programmation de la partie Modèle d'une application
    - Style : "génie logiciel", orienté production d'une grande quantité d'instances

Caractéristiques des langages OO

- Interprété
  - Pas de compilation, mais un interpréteur, exécution plus lente
- Compilé
  - Phase de compilation, mais exécution plus rapide
- Semi compilé
  - Phase de compilation dans un code intermédiaire
  - Besoin d'un interpréteur, mais exécution plus efficace que l'interprétation simple
  - Phase de compilation explicite, ou implicite au début de la première exécution : "Just In Time" (JIT)
- Statique
  - La structure des objets est non modifiable à l'exécution
  - Adapté à la compilation
- Dynamique
  - Ajout possible d'attributs, de méthodes en cours d'exécution
  - Difficile en cas de compilation

Caractéristiques de quelques langages

	Java	C++	JavaScript	PHP
Tout objet			X	
Seules les instances sont objets	X	X		X
Classe	X	X		X
Prototype			X	
Interpreté			X	
Compilé		X		
Semi-Compilé	X			
Semi-Compilé JIT	X			X
Statique	X	X		
Dynamique			X	X
Coté serveur	X	X	X	X
Coté client (navigateur)	?		X	

Manipulation des objets en PHP

- Les objets appartiennent à une classe
- La classe regroupe le "savoir faire" :
  - Les méthodes
  - Attention : en PHP la classe **ne définit pas tous les attributs**
    - Dynamique : chaque objet d'une même classe peut avoir des attributs différents !
- Objet dans une variable :
  - stocke non pas l'objet, mais un identifiant (i.e. adresse constante) !
- Création d'objets avec constructeur (identique Java, C++):

```
$a = new classe();
```

Possibilité de passer des valeurs à la création

- Accéder aux attributs et déclencher des méthodes
  - Notation flèche comme en C++ (le point déjà utilisé pour la concaténation)

```
$a->attribut = 5;
$a->run();
```

Affectation d'objets

```
class MaClasse {
        public int $attribut; // type uniquement version > 7.4
    }
$a = new MaClasse();
$a->attribut = 10;
$b = $a;

var_dump($a);var_dump($b);
```

- Une variable manipulant un objet : en fait uniquement sa **référence**
- ... donc l'affectation copie l'idf de l'objet (pas de nouvel objet !)

```
object(MaClasse)#1 (1) {
  ["attribut"] => int(10)
}
object(MaClasse)#1 (1) {
  ["attribut"] => int(10)
}
```

Passage d'un objet en paramètre

```
function plus (MaClasse $x) {
        $x->attribut++;
}
class MaClasse {
        public int $attribut;
}

$a = new MaClasse();
$a->attribut = 10;

plus($a);
```

- Identique à l'affectation d'un variable : **copie l'idf** de l'objet
- Conséquence : l'objet passé en paramètre est modifiable!
  - Résultat : l'attribut de $a vaut 11
  - NB: identique à Java et Javascript

Se désigner soi-même

```
class Paragraphe {
        public string $contenu;
        function nombreDeMots(): int {
            return str_word_count($this->contenu,0,'àé!');
        }
    }

    $p = new Paragraphe();
    $p->contenu = "Bonjour à tous les étudiants en AS !";
    print($p->nombreDeMots()."\n");
```

- Utiliser la variable contextuelle : **$this**
- Identique à C++, proche de JavaScript

Constructeur

```
class Voiture {
        private string $marque;
        private string $modele;
        function __construct(string $ma, string $mo) {
                $this->marque = $ma;
                $this->modele = $mo;
         }
}

$v = new Voiture('Renault','Kangoo');
```

- Avec un constructeur préférer mode **private** des attributs
- Attention : pas de surcharge possible de constructeur
- Destructeur possible mais rarement utilisé : __destruct ( )

Dynamicité : modifier ses attributs

```
class Voiture {
        private string $marque;
        private string $modele;
        function __construct(string $ma, string $mo) {
                $this->marque = $ma;
                $this->modele = $mo;
         }
}

$v = new Voiture('Renault','Kangoo');
$v->couleur = 'rouge';
```

- Comme en Java, s'il y a un constructeur préférer des attributs en mode **private**, avec des getters et setters.
- Possibilité d'ajout / suppression d'attributs **public** de manière dynamique, c'est à dire en cours de fonctionnement.

Déclarer les attributs dans une classe

```
class Voiture {
        function __construct(string $ma, string $mo) {
                $this->marque = $ma;
                $this->modele = $mo;
        }
}

$v = new Voiture('Renault','Kangoo');
$v->couleur = 'rouge';
```

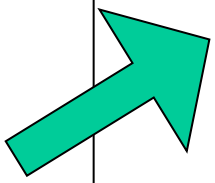
- Déclaration d'attributs+type dans la classe est ***facultative***
  - … mais **fortement conseillée**
- Les attributs déclarés dans la classe sont toujours présents sur les instances.

Accès dynamiques aux attributs

```
$v = new Voiture('Renault','Kangoo');

$attribut = 'couleur'

$v->$attribut = 'rouge';
```



- ***Erreur classique ...***
- Le $ est possible mais réalise une **indirection**
  - Il permet d'indiquer le nom de l'attribut dans une variable.
- Attention : pour la plupart des usages, **PAS DE $ pour l'accès aux attributs**

Attributs : droits et mutateurs

- Comme la pluparts des langages à base de classe, on peut limiter la visibilité des attributs au **niveau de la classe**.
- La visibilité concerne à la fois la lecture et la modification des attributs.
- Elle ne concerne jamais l'objet lui même qui a toujours accès à tous ses propres attributs.
  - **public** : visible à tous les autres objets;
  - **private** : visible uniquement aux objets de la même classe;
  - **protected** : comme private, mais en plus visible aux objets des sous classes;
- Mutateurs : méthodes qui permettent de contrôler l'accès aux attributs :
  - Getters : accès en lecture
  - Setters : accès en écriture

Mutateurs : getter à la main

```
class Voiture {
        private string $marque;
        private string $modele;
        function __construct(string $ma, string $mo) {
                $this->marque = $ma;
                $this->modele = $mo;
         }
        // Donne accès en lecture seule des attributs privés
        function getMarque() {
                return $this->marque;
        }
        function getModele() {
                return $this->marque;
        }
}
```

- Ajouter manuellement des getters : get_xxx(), ou getXxx(), ou simplement xxx()

Mutateurs : getter global

```
class Voiture {
        private string $marque;
        private string $modele;
        function __construct(string $ma, string $mo) {
                $this->marque = $ma;
                $this->modele = $mo;
         }
        // Donne accès en lecture seule a tous les attributs
privés
        function __get(string $attribut) {
                return $this->$attribut;
        }
}
```

- **__get()** : contrôle **globalement** la lecture tous les attributs non public

Mutateurs : contrôle getter à la main

```
class Voiture {
        private string $marque;
        private string $modele;
        function __construct(string $ma, string $mo) {
                $this->marque = $ma;
                $this->modele = $mo;
         }
        // Donne accès en lecture uniquement à 'marque"
        function getMarque() {
                return $this->marque;
        }
}
```

- Contrôle des getters : simplement en supprimant le getter

Mutateurs : contrôle getter global

```
class Voiture {
  private string $marque;
  private string $modele;
  function __construct(string $ma, string $mo) {
    $this->marque = $ma;
    $this->modele = $mo;
  }
  // Donne accès en lecture uniquement à 'marque'
  function __get(string $a) {
    switch($a) {
      case 'marque': return $this->$a;
        break;
      default:
        throw new Exception("Error cannot acces '$a'", 1);
        break;
```

- Contrôle avec switch sur le nom de l'attribut et lever une exception.

Mutateurs : setter à la main

```
class Voiture {
        private string $marque;
        private string $modele;
        function __construct(string $ma, string $mo) {
                $this->marque = $ma;
                $this->modele = $mo;
         }
        // Modification du modèle uniquement
        function setModele(string $value) {
                return $this->modèle = $value;
        }
}
```

- Ajouter manuellement des setters : set_xxx(), ou setXxx(), ou simplement xxx(val)

Mutateurs : contrôle setter global

```
class Voiture {
  private string $marque;
  private string $modele;
  function __construct(string $ma, string $mo) {
    $this->marque = $ma;
    $this->modele = $mo;
  }
  // Donne accès en écriture uniquement pour 'modele'
  function __set(string $a, string $value) {
    switch($a) {
      case 'modele': $this->$a = $value;
        break;
      default:
        throw new Exception("Error cannot read '$a'", 1);
        break;
```

- Contrôle avec switch sur le nom de l'attribut et lever une exception.

Parcourir les attributs publics : **foreach**

```
function __set(string $a, string $value) {
    switch($a) {
      case 'modele':
      case 'couleur':
        $this->$a = $value;
[...]
$v = new Voiture('Renault','Kangoo');
$v->couleur = 'rouge';
foreach ($v as $attrib => $value) {
        echo "$attrib : $value </br>";
}
```

- Parcours les attributs **publics** uniquement

```
couleur : rouge
```

Constructeur avec paramètre tableau

```
class Voiture {
    function __construct(array $attributs) {
      foreach ($attributs as $key => $value) {
        $this->$key = $value;
      }
    }
  };
$v = new Voiture(
        array('marque' => 'Renault','modèle' => 'Kangoo')
        );
    var_dump($v);
```

- Passer en paramètre la liste des attributs et leurs valeurs dans un tableau
- Permet de simuler la **surcharge** avec nombre de paramètres variable

Constructeur avec paramètre tableau

```
class Voiture {
  function __construct(array $attributs=null) {
    if ($attributs == null) return;
    foreach ($attributs as $key => $value) {
      $this->$key = $value;
    }
  }
};

$w = new Voiture();
```

- Rajouter l'attribut par défaut null pour accepter un constructeur vide.
- NB: en PHP 7.1, il y a un nouveau type de passage de paramètre pour indiquer que le paramètre est possiblement null

  ?type $param : function __construct(?array $attributs)

Modèle objet en PHP : à retenir

- **Très très proche du modèle objet de Java**
  - Typage des attributs (depuis PHP 7.4)
  - Typage des paramètres des méthodes
  - Typage du résultat des fonctions et méthodes
  - Droits : public, private, protected
- Objets manipulables automatiquement par référence
  - passage des paramètres par copie de la référence
  - rendu par copie de la référence
  - NB: identique à Java
- Attributs : définit **dynamiquement** (cf. tableau associatif)
  - Getters, setters classique (Java), en plus __get(), et __set()
  - déclaration dans la classe non nécessaire mais conseillée
- Constructeur facultatif
  - mais **pas de surcharge** : un seul constructeur
- Méthodes : **pas de surcharge**