PHP

21 Héritage (Objet) :

L'un des intérêts de la **POO** est de rendre notre code modulable, réutilisable (nous l'avons vu avec les **classes** et le modèle **MVC**) et de nous permettre de mettre à jour et d'étendre notre code facilement.

L'héritage va nous permettre **d'étendre** une **classe** (*récupérer tout ce qui est en **public** ou **protected** dans la **classe parente***) par une autre **classe**. Les classes **étendues** pourront bénéficier d'**attributs**, **constructeurs**, **méthodes propres**,

Pour étendre une classe nous utiliserons le mot clé **extends**.

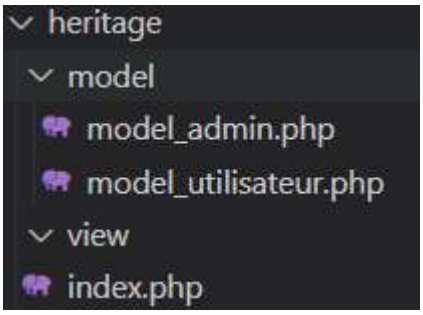
21.1 Exemple :

```php
< ?php
class Admin extends Utilisateur{
}
?>
```

Notre classe **Admin** va étendre la classe **Utilisateur**, elle va pouvoir utiliser toutes les **méthodes** et **attributs** de la classe **Utilisateur** qui ne sont pas en **private**.

NB : Attention Pour qu'une classe puisse **hériter** d'une autre celle-ci doit être **existante** dans le projet et être **include**, une classe ne peut hériter que d'une seule classe à la fois.

Structure des classes :



PHP

Classe Utilisateur :

```php
<?php
    class Utilisateur{
        //attributs
        private $name;
        private $firstName;
        //constructeur
        public function __construct($name, $first){
            $this->name = $name;
            $this->firstName = $first;
        }
        //getter and setter
        public function getName():string{
            return $this->name;
        }
        public function getFirstName():string{
            return $this->firstName;
        }
        public function setName($name):void{
            $this->name = $name;
        }
        public function setFirstName($first):void{
            $this->firstName = $first;
        }
        //Méthodes
        public function showUser():void{
            echo 'Nom : '.$this->getName().'
            Prénom : '.$this->getFirstName().'';
        }
    }
?>
```

PHP

21.2 Appel des classes et méthodes :

```php
<?php
    //imports
    include './model/model_utilisateur.php';
    include './model/model_admin.php';

    //instances des objets :
    $util = new Utilisateur("Dupond", "Marc");
    //admin utilise le constructeur d'Utilisateur
    $admin = new Admin("Durand", "Marie");

    echo '<p>Utilisateur : '.$util->getName().'</p>';
    //admin utilise le getter public d'utilisateur
    echo '<p>Admin : '.$admin->getName().'</p>';
?>
```

NB : notre classe **Admin** utilise le **constructeur** ainsi que **les getters setters** de la classe **Utilisateur**.

21.2 Redéfinition d'une méthode dans la classe enfant :

Si nous essayons de redéfinir dans la classe **Admin** le **getter getName** comme ci-dessous et que nous l'appelons dans la page **index** (code précédent), nous aurons une erreur :

Car le **paramètre name** est en **Private** (dans la classe **Utilisateur**) et donc **inaccessible** en dehors de la classe **Utilisateur**.

```php
<?php
    class Admin extends utilisateur{
        //getter and setter
        //redéfinition dans la classe
        public function getName():string{
            return $this->name;
        }
    }
?>
```

PHP

22 Etendu des classes Héritage (Objet) :

Pour pouvoir étendre des classes et redéfinir leurs attributs, méthodes dans la classe étendu nous allons devoir passer les **attributs** et ou **méthodes** à redéfinir en **protected** dans la classe parente (classe **Utilisateur**).

22.1 Correction classe Utilisateur :

```php
<?php
    class Utilisateur{
        //attributs
        protected $name;
        protected $firstName;
?>
```

Le code de la section précédente (**getNom** dans la classe **Admin**) est alors **utilisable**. C'est le **getter getNom** de la classe **Admin** qui est appelé (il peut résoudre le **$this->name** car l'attribut est en **protected**).

22.2 Surcharge de méthode :

Redéfinition (surcharge) de la méthode getNom dans la classe Admin :

```php
//redéfinition dans la classe
public function getName():string{
      //retourne le nom en Majuscule
      return strtoupper($this->name);
}
```

NB : On modifie dans la classe **Admin** (surcharge) le fonctionnement de la méthode **getName** qui est déjà définie dans la classe **Utilisateur**.

22.3 Méthodes de la classe Admin qui utilisent des attributs de la classe Utilisateur :

```php
<?php
    class Admin extends Utilisateur{
        //attributs
        protected $activate;
        //getter and setter
        //redéfinition dans la classe
        public function getName():string{
            //retourne le nom en Majuscule
            return strtoupper($this->name);
        }
        //méthodes
        //activer un compte Utilisateur (objet)
        public function setActiveUser($obj):void{
```

PHP

```
            //ajouter le nom dans la liste activate(tableau)
            $this->activate[] = $obj;
        }
        //affichage de la liste des comptes activés
        public function getActiveUser():void{
            echo 'Liste des comptes Utilisateurs activés :';
            //boucle affichage des comptes(objets)
            foreach($this->activate as $value){
                echo '<p>'.$value->name.' '.$value->firstName.',</p>';
            }
        }
    }
?>
```

22.4 Appel dans index.php des méthodes de la classe Admin (setActivateUser et getActivateUser) :

```
<?php
    //imports
    include './model/model_utilisateur.php';
    include './model/model_admin.php';
    //instances des objets :
    $util = new Utilisateur("Dupond", "Marc");
    $util2 = new Utilisateur("Albert", "Patricia");
    //admin utilise le constructeur d'Utilisateur
    $admin = new Admin("Durand", "Marie");
    echo '<p>Utilisateur : '.$util->getName().'</p>';
    echo '<p>Utilisateur : '.$util2->getName().'</p>';
    //admin utilise le getter public d'utilisateur
    echo '<p>Admin : '.$admin->getName().'</p>';
    //appel de la méthode setActiveUser
    $admin->setActiveUser($util);
    $admin->setActiveUser($util2);
    //affichage de la liste des utilisateurs activés
    $admin->getActiveUser();
?>
```

Repository github héritage :

https://github.com/mithridatem/extends.git

PHP

23 Opérateur de résolution de portées Héritage (Objet) :

Pour pouvoir accéder à la définition de base d'un **attribut** ou d'une **méthode surchargée**, il existe en **PHP** un opérateur :

::

Il se nomme **opérateur de portée**. Il rend disponible 3 nouveaux mots clés.

parent, **self** et **static** (**static** sera vu dans un prochain chapitre sur les **attributs** et **méthodes static** d'une **classe**).

23.1 Définition :

Parent sert à faire appel à la méthode depuis sa définition dans la classe **parente** (dans le cas où la méthode existe dans les 2 classes **enfant** et **parent**).

Self sert à faire appel à la méthode depuis sa définition dans la classe **parente** (dans le cas où la méthode existe dans les 2 classes **enfant** et **parent**).

Static sert à faire appel à une méthode qui ne nécessite pas d'instance d'un objet. Nous verrons en détail son fonctionnement dans un prochain chapitre.