



POO - PHP - Partie 2 : Héritage et encapsulation

⊖ Cours	PHP
⊖ Type	Guide
📎 Supports	https://openclassrooms.com/fr/courses/1665806-programmez-en-orienté-objet-en-php
⊖ Difficulté	Moyen

L'héritage est un design pattern permettant de structurer le code en réduisant les répétitions et de donner du sens aux objets.

Extends.

Ce mot clé est la base de la définition d'une classe fille. Voici comment cela se présente :

```
class Voiture {  
    public CONST VITESSE_MAX;  
}  
  
class VoitureDeCourse extends Voiture {  
    public function get_vitesse(){  
        echo parent::VITESSE_MAX;  
    }  
}
```

Dans cet exemple, on crée une classe mère nommée `Voiture` et une classe fille `VoitureDeCourse` qui prend les mêmes propriétés que la classe mère. La différence étant qu'on peut attribuer de nouvelles propriétés à la classe fille que le parent ne récupéra pas.

On peut accéder aux propriétés de la classe parent grâce au mot clé `parent::`

Si on écrit à nouveau une propriété déjà existante dans la classe parent, et que l'on redéfinit celle-ci, on parle de **surcharge**.

Lors d'une surcharge :

- vous ne pouvez **pas enlever** des arguments ;
- vous pouvez **ajouter** un argument **uniquement** s'il est **optionnel** ;
- Vous pouvez **changer** le type d'un **argument uniquement** s'il est **compatible** avec le type d'origine ([voir un exemple](#)) ;
- vous pouvez **changer** le type de **retour** de la méthode **uniquement** s'il est **compatible** avec le type d'origine.

Vous venez de voir la possibilité d'étendre l'accès d'une classe mère à des classes filles. L'inverse est tout autant possible. Les méthodes et propriétés `private` sont inaccessible depuis les classes filles. Si l'on souhaite configurer quelque chose comme `private`, mais que l'on préfère que les classes filles en héritent quand même, il existe une troisième possibilité : `protected`.

Abstraction

Une classe définie comme `abstract` ne peut instancier d'objet seule. Celle-ci devra être `extends` auparavant.

L'abstraction sert à définir un modèle de classe pour d'autres classes.

```
<?php
abstract class MaClasseAbstraite {
}
```

Le mot clé `abstract` s'utilise aussi sur des méthodes de classe pour instaurer un modèle (on parle de déclarer une signature)

```
<?php
abstract class MaClasseAbstraite {
    abstract public function getSomething(): string {
        //Rien ici, on attribue juste le modèle pour imposer aux enfants de cette classe de posséder cette fonction.
    }
}
```

i Il est possible de cumuler plusieurs classes abstraites qui héritent d'autres classes abstraites.

Interdiction d'héritage

Après avoir déclaré les classes filles qui héritent d'un modèle abstrait, on peut utiliser le mot clé `final` pour interdire les prochains enfants d'hériter de certaines propriétés.

```
final class MaClasse extends ClasseMere {
    // ICI la classe ne peut plus être utilisée pour y étendre un nouveau enfant.
}
```

Gérer le comportement du parent

Les constantes et les références `self` d'une classe enfant peuvent être réaffectées en modifiant leur appel dans le constructeur.

Ainsi, une méthode qui était définie dans la partie parent comme `self::` peut être modifiée à l'avenir dans les classes enfant que l'on ne connaît pas encore .

Le code ressemblerait donc à ceci :

```
<?php

declare(strict_types=1);

abstract class User
{
    public const STATUS_ACTIVE = 'active';
    public const STATUS_INACTIVE = 'inactive';

    public function __construct(public string $email, public string $status = self::STATUS_ACTIVE)
    {
    }

    // le mot clé self a été modifié pour le mot clé static dans la méthode :
    public function setStatus(string $status): void
    {
        assert(
            in_array($status, [static::STATUS_ACTIVE, static::STATUS_INACTIVE]),
            sprintf('Le status %s n\'est pas valide. Les status possibles sont : %s', $status, implode(
                ', ',
                [static::STATUS_ACTIVE, static::STATUS_INACTIVE]))
        );

        $this->status = $status;
    }

    public function getStatus(): string
    {
        return $this->status;
    }

    abstract public function getUsername(): string;
}

final class Admin extends User
{
    public const STATUS_ACTIVE = 'is_active'; // Modification de la const par rapport au parent
    public const STATUS_INACTIVE = 'is_inactive';

    // Ajout d'un tableau de roles pour affiner les droits des administrateurs :
    public function __construct(string $email, string $status = self::STATUS_ACTIVE, public array $roles = [])
    {
        parent::__construct($email, $status);
    }

    public function getUsername(): string
    {
        return $this->email;
    }

    // ...
}

$admin = new Admin('michel@petrucciani.com');
var_dump($admin);
$admin->setStatus(Admin::STATUS_INACTIVE);
```

On utilise donc le mot clé `static::` dans la classe parent pour redéfinir l'appel `self` au sein de la classe enfant sans remonter à la classe parent.