PHP : POO

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

Plan

Plan

PHP

Qu'est ce qu'une classe en POO ?

- Ça correspond à un plan, un moule, une usine...
- C'est une description abstraite d'un type d'objets
- Elle représente un ensemble d'objets ayant les mêmes propriétés statiques (attributs) et dynamiques (méthodes)

PHP

Qu'est ce qu'une classe en POO ?

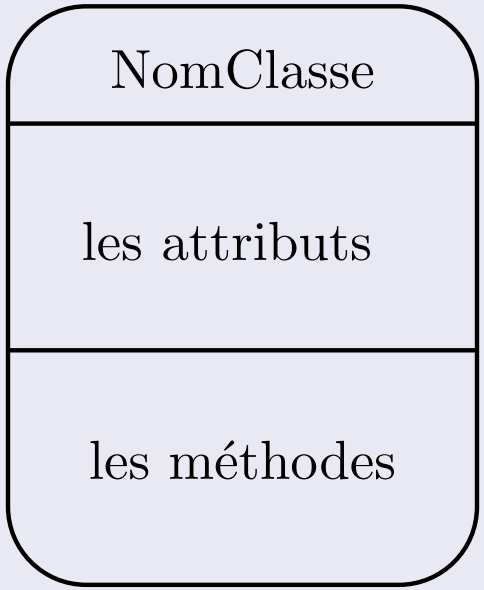
- Ça correspond à un plan, un moule, une usine...
- C'est une description abstraite d'un type d'objets
- Elle représente un ensemble d'objets ayant les mêmes propriétés statiques (attributs) et dynamiques (méthodes)

Qu'est ce que c'est la notion d'instance ?

- Une instance correspond à un objet créé à partir d'une classe (via le constructeur)
- L'instanciation : création d'un objet d'une classe
- instance $\equiv$ objet

PHP

De quoi est composé une classe ?



- Attribut : [visibilité] + nom
- Méthode : [visibilité] + nom + arguments $\equiv$ + valeur de retour : exactement comme les fonctions en procédurale

PHP

Deux normes pour **PHP**

- **PEAR** (PHP Extension and Application Repository) : `https://pear.php.net/manual/en/standards.php`
- **FIG** (PHP Framework Interop Group) : `https://www.php-fig.org/`

PHP

Conventions d'écriture en **PHP**

- Écrire les classe/interface en **Pascal case** (appelé aussi **StudlyCaps**)
- Écrire les attributs, méthodes ou objets en **Camel case** ou **Snake case**
- Une classe par fichier
- Un fichier, contenant une classe, doit porter le nom de la classe

PHP

Particularité du **PHP**

- Le mot-clé `$this` permet de désigner l'objet courant.
- `$this` est obligatoire même si aucune ambiguïté ne se présente.

PHP

Commençons par créer un nouveau projet **PHP** sous **VSC** ?

- Créez un répertoire `cours-poo` dans le répertoire `www`
- Lancez **VSC** et allez dans `File > Open Folder...` et choisissez `cours-poo`
- Dans `cours-poo`, créez deux fichiers `index.php` et `Personne.php`

PHP

Commençons par créer une classe `Personne` avec le contenu suivant (pour les accolades : recommandation PEAR)

```
class Personne
{

}
```

PHP

Considérons le contenu suivant pour `index.php`

```
<?php
include "Personne.php";
?>
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale
      =1.0">
    <title>Programmation objet</title>
</head>

<body>
    <?php

    ?>
</body>

</html>
```

PHP

Structure d'une classe PHP

```php
<?php
    class NomClass{
        //attributs
        visibilité $attribut1;
        visibilité $attribut2;
        ...
        visibilité $attributn;
        //méthodes
    }
?>
```

visibilité : `public`, `protected` ou `private`

PHP

Ajoutons quelques attributs à la classe `Personne`

```
class Personne
{
    public $num;
    public $nom;
    public $prenom;

}
```

PHP

Depuis PHP 7.4, on peut typer les attributs

```php
class Personne
{
    public int $num;
    public string $nom;
    public string $prenom;

}
```

Création d'un objet : instanciation (dans `index.php`)

```
$nomObjet = new NomClasse();
```

PHP

Création d'un objet : instanciation (dans `index.php`)

```
$nomObjet = new NomClasse();
```


Explication

- pour créer un objet on utilise l'opérateur `new` et on fait appel au constructeur
- il existe un constructeur par défaut pour chaque classe en **PHP**
- il est toujours possible de créer son propre constructeur
- la création d'un constructeur écrasera le constructeur par défaut

PHP

Exemple (code à placer dans `index.php`)

```
$personne = new Personne();
```

PHP

Exemple (code à placer dans `index.php`)

```
$personne = new Personne();
```

Exemple (code à placer dans `index.php`)

```
$personne->num = 100;
$personne->nom = "wick";
$personne->prenom = "john";
```

PHP

Exemple (code à placer dans `index.php`)

```
$personne = new Personne();
```

Exemple (code à placer dans `index.php`)

```
$personne->num = 100;
$personne->nom = "wick";
$personne->prenom = "john";
```


Explication

-> : permet d'accéder à un élément de l'objet (méthodes ou attributs)

PHP

Si on essaye d'afficher cet objet dans la page ⇒ erreur fatale : `Object of class Personne could not be converted to string`

```php
echo $personne;
```

PHP

Si on essaye d'afficher cet objet dans la page ⇒ erreur fatale : `Object of class Personne could not be converted to string`

```php
echo $personne;
```

Pour résoudre ce problème, on ajoute une méthode `__toString()`

```php
public function __toString(): string
{
    return $this->num . " " . $this->nom . " " . $this->prenom;
}
```

PHP

Si on essaye d'afficher cet objet dans la page ⇒ erreur fatale : `Object of class Personne could not be converted to string`

```php
echo $personne;
```

Pour résoudre ce problème, on ajoute une méthode `__toString()`

```php
public function __toString(): string
{
    return $this->num . " " . $this->nom . " " . $this->prenom;
}
```


Explication

- `$this` : désigne l'objet courant
- `->` : pointer un élément de l'objet (méthodes ou attributs)

PHP

Ainsi le résultat de l'instruction précédente est :

```
echo $personne;
/* affiche 100 wick john */
```

PHP

Hypothèse

Supposant que l'on n'accepte pas de valeur négative pour l'attribut `num` de `Personne`

PHP

Hypothèse

Supposant que l'on n'accepte pas de valeur négative pour l'attribut `num` de `Personne`

Démarche

1. Bloquer l'accès directe aux attributs (mettre la visibilité à `private`)
2. Définir des méthodes publiques qui contrôlent l'affectation de valeurs aux attributs (les `setter`)

PHP

Hypothèse

Supposant que l'on n'accepte pas de valeur négative pour l'attribut `num` de `Personne`

Démarche

1. Bloquer l'accès directe aux attributs (mettre la visibilité à `private`)
2. Définir des méthodes publiques qui contrôlent l'affectation de valeurs aux attributs (les `setter`)

Convention

- Utiliser la visibilité `private` ou `protected` pour les attributs
- Utiliser la visibilité `public` pour les méthodes

PHP

Mettons la visibilité `private` pour tous les attributs de la classe `Personne`

```php
class Personne
{
    private int $num;
    private string $nom;
    private string $prenom;

    public function __toString(): string
    {
        return $this->num . " " . $this->nom . " " . $this->prenom;
    }
}
```

PHP

Mettons la visibilité `private` pour tous les attributs de la classe `Personne`

```php
class Personne
{
    private int $num;
    private string $nom;
    private string $prenom;

    public function __toString(): string
    {
        return $this->num . " " . $this->nom . " " . $this->prenom;
    }
}
```

En actualisant `index.php`, une erreur fatale sera affichée.

PHP

Explication

Attributs privés $\Rightarrow$ inaccessibles

PHP

Explication

Attributs privés ⇒ inaccessibles

Ajoutons les trois setters suivants

```
public function setNum($num)
{
    $this->num = $num >= 0 ? $num : 0;
}
public function setNom($nom)
{
    $this->nom = $nom;
}
public function setPrenom($prenom)
{
    $this->prenom = $prenom;
}
```

PHP

Modifions le contenu d'`index.php` et utilisons les setters

```php
$personne = new Personne();
$personne->setNum(100);
$personne->setNom("wick");
$personne->setPrenom("john");
echo $personne;
/* affiche 100 wick john  */
```

PHP

Modifions le contenu d'`index.php` et utilisons les setters

```php
$personne = new Personne();
$personne->setNum(100);
$personne->setNom("wick");
$personne->setPrenom("john");
echo $personne;
/* affiche 100 wick john  */
```

Testons aussi avec une valeur négative pour `num`

```php
$personne = new Personne();
$personne->setNum(-100);
$personne->setNom("wick");
$personne->setPrenom("john");
echo $personne;
/* affiche 0 wick john  */
```

PHP

Comment générer les setters avec **VSC** ?

- Installer l'extension **PHP Getters & Setters**
- Sélectionner les attributs
- Faire un clic droit sur un élément de la sélection et choisir `Insert PHP Setter`

PHP

Le setter : contrôle de type de paramètre

- PHP est un langage de programmation faiblement typé
- Par exemple, on peut affecter une chaîne de caractère à une variable qui attendait un nombre
- Solution avec **PHP 7** : typer les méthodes

PHP

Nouveau contenu pour les setters

```
public function setNum(int $num): void
{
    $this->num = $num;
}
public function setNom(string $nom): void
{
    $this->nom = $nom;
}
public function setPrenom(string $prenom): void
{
   b$this->prenom = $prenom;
}
```

PHP

Rien à changer pour le test dans `index.php`

```php
$personne = new Personne();
$personne->setNum(100);
$personne->setNom("wick");
$personne->setPrenom("john");
echo $personne;
/* affiche 100 wick john  */
```

PHP

Hypothèse

Si on voulait afficher les attributs (privés) de la classe `Personne`, un par un, dans la classe `index.php` sans passer par le `toString()`

PHP

Hypothèse

Si on voulait afficher les attributs (privés) de la classe `Personne`, un par un, dans la classe `index.php` sans passer par le `toString()`

Démarche

Définir des méthodes qui retournent les valeurs de nos attributs (les `getter`)

PHP

Pour générer les getters

- Faire clic droit sur la classe `Personne`
- Aller dans `Source > Generate Getters and Setters...`
- Cliquer sur chaque attribut et cocher la case `getNomAttribut()`
- Valider

PHP

Les getters générés

```php
public function getNum()
{
    return $this->num;
}

public function getNom()
{
    return $this->nom;
}

public function getPrenom()
{
    return $this->prenom;
}
```

PHP

Comment générer les getters avec **VSC** ?

- Sélectionner les attributs
- Faire un clic droit sur un élément de la sélection et choisir `Insert PHP Getter`

PHP

Ajoutons le typage aux getters générés

```php
public function getNum(): int
{
    return $this->num;
}

public function getNom(): string
{
    return $this->nom;
}

public function getPrenom(): string
{
    return $this->prenom;
}
```

PHP

Pour tester (`index.php`)

```
$personne = new Personne();
$personne->setNum(100);
$personne->setNom("wick");
$personne->setPrenom("john");
echo $personne->getNom() . " " . $personne->
   getPrenom();
/* affiche wick john */
```

PHP

Pour générer les getters et setters avec **VSC**

- Sélectionner les attributs
- Faire un clic droit sur un élément de la sélection et choisir `Insert PHP Getter & Setter`

PHP

Rappel

- Par défaut, toute classe a un constructeur par défaut sans paramètre.
- Pour simplifier la création d'objets, on peut définir un nouveau constructeur qui prend en paramètre plusieurs attributs de la classe.

PHP

Syntaxe

```
class NomClass{
    //constructeur PHP3
    public function NomClass(){
        ....
    }
}
```

PHP

Syntaxe

```
class NomClass{
    //constructeur PHP3
    public function NomClass(){
        ....
    }
}
```

ou

```
class NomClass {
    //constructeur PHP5
    public function __construct(){
        ....
    }
}
```

PHP

Définissons le constructeur suivant, dans la classe `Personne`, acceptant trois paramètres

```
public function __construct(int $num, string $nom, string
   $prenom)
{
    $this->num = $num;
    $this->nom = $nom;
    $this->prenom = $prenom;
}
```

PHP

Définissons le constructeur suivant, dans la classe `Personne`, acceptant trois paramètres

```
public function __construct(int $num, string $nom, string
   $prenom)
{
    $this->num = $num;
    $this->nom = $nom;
    $this->prenom = $prenom;
}
```

Pour préserver la cohérence, il faut que le constructeur contrôle la valeur de l'attribut `num`


```
public function __construct(int $num, string $nom, string
   $prenom)
{
    $this->num = $num >= 0 ? $num : 0;
    $this->nom = $nom;
    $this->prenom = $prenom;
}
```

PHP

On peut aussi appelé le `setter` dans le constructeur

```
public function __construct(int $num, string $nom,
   string $prenom)
{
     $this->setNum($num);
     $this->nom = $nom;
     $this->prenom = $prenom;
}
```

PHP

Remarque

En actualisant `index.php`, une erreur fatale sera affichée.

PHP

Remarque

En actualisant `index.php`, une erreur fatale sera affichée.

Explication

Le constructeur par défaut a été écrasé (il n'existe plus)

PHP

Remplaçons le contenu précédent d'`index.html` par le suivant

```
$personne = new Personne(100, "wick", "john");
echo $personne;
/* affiche 100 wick john */
```

PHP

Dans PHP 8, le Constructor Promotion a été introduit et qui permet de fusionner la déclaration des attributs et le constructeur

```
class Personne
{

    public function __construct(private int $num, private
       string $nom, private string $prenom)
    {
    }

    // + les autres méthodes
```

PHP

Dans PHP 8, le Constructor Promotion a été introduit et qui permet de fusionner la déclaration des attributs et le constructeur

```
class Personne
{

    public function __construct(private int $num, private
       string $nom, private string $prenom)
    {
    }

    // + les autres méthodes
```


Rien à changer dans `index.html`

```
$personne = new Personne(100, "wick", "john");
echo $personne;
/* affiche 100 wick john */
```

PHP

Le destructeur

- représenté par la méthode `__destruct()`
- peut être appelé lorsque l'objet n'est plus utilisé.
  - explicitement avec `$this->__destruct`
  - implicitement (automatiquement)
    - lorsque l'objet n'est plus utilisé (référencé)
    - à la fin ou à l'arrêt d'exécution du script

PHP

Ajoutons le destructeur suivant à notre classe `Personne`

```
function __destruct()
{
    echo "$this->num est détruit" . "<br>";
}
```

PHP

Testons le contenu précédent d'`index.html` (appel implicite du destructeur)

```php
$personne = new Personne(100, "wick", "john");
echo $personne;
/* affiche
100 wick john
100 est détruit
*/
```

PHP

Testons le contenu précédent d'`index.html` (appel implicite du destructeur)

```
$personne = new Personne(100, "wick", "john");
echo $personne;
/* affiche
100 wick john
100 est détruit
*/
```

Appelons explicitement le destructeur

```
$personne = new Personne(100, "wick", "john");
echo $personne;
/* affiche
100 wick john
100 est détruit
100 est détruit
*/
```

PHP

Exercice (à réaliser dans un nouveau projet nommé `exercice-relation-classes`)

- Reprenez la classe `Personne` du projet `cours-poo`.
- Créez une classe `Adresse` avec trois attributs privés de type `string` : `rue`, `codePostal` et `ville`.
- Implémentez un constructeur acceptant ces trois paramètres, ainsi que les accesseurs (getters) et mutateurs (setters) correspondants.
- Dans la classe `Personne`, ajoutez un attribut privé `adresse` de type `Adresse` et modifiez le constructeur pour accepter quatre paramètres, y compris l'adresse.
- Ajoutez les accesseurs et mutateurs pour ce nouvel attribut, et mettez à jour la méthode `__toString` en conséquence.
- Dans le fichier `index.php`, instanciez un objet `$adresse` de type `Adresse` et un objet `$personne` de type `Personne`, en passant l'objet `$adresse` en paramètre.
- Affichez tous les attributs de l'objet `$personne`.

PHP

Comment procéder pour les `include` ?

- Inclure `Personne.php` dans `index.php` et `Adresse.php` dans `Personne.php`, ou
- Inclure `Personne.php` et `Adresse.php` directement et uniquement dans `index.php`

PHP

Comment procéder pour les `include` ?

- Inclure `Personne.php` dans `index.php` et `Adresse.php` dans `Personne.php`, ou
- Inclure `Personne.php` et `Adresse.php` directement et uniquement dans `index.php`

Remarque

La meilleure pratique serait d'inclure uniquement `Personne.php` dans `index.php` , et d'inclure `Adresse.php` dans `Personne.php` pour les raisons suivantes

- **Principe de modularité et de découplage** : `Personne.php` est autonome et gère elle-même ses propres dépendances, ici la classe `Adresse`.
- **Maintenance et évolutivité** : Si la classe `Personne` dépend d'autres classes ou si vous ajoutez de nouvelles dépendances, `index.php` reste inchangé, même si `Personne.php` ou `Adresse.php` évoluent.
- **Pratique courante en programmation orientée objet** : Dans la majorité des frameworks et projets orientés objet, chaque classe est responsable d'importer ses dépendances, ce qui rend la structure de code plus claire et plus maintenable.

PHP

Récapitulatif

Les instances d'une même classe ont toutes les mêmes attributs mais pas les mêmes valeurs

PHP

Récapitulatif

Les instances d'une même classe ont toutes les mêmes attributs mais pas les mêmes valeurs

Hypothèse

Et si nous voulions qu'un attribut ait une valeur partagée par toutes les instances (par exemple, le nombre d'objets instanciés de la classe `Personne`)

PHP

Récapitulatif

Les instances d'une même classe ont toutes les mêmes attributs mais pas les mêmes valeurs

Hypothèse

Et si nous voulions qu'un attribut ait une valeur partagée par toutes les instances (par exemple, le nombre d'objets instanciés de la classe `Personne`)

Solution : attribut statique ou attribut de classe

Un attribut dont la valeur est partagée par toutes les instances de la classe

PHP

Remarques

- Le mot-clé `self` ⇒ à utiliser dans la classe pour accéder à un élément static
- `::` (Resolution Operator) ⇒ à utiliser avec `self` (comme `$this` et `->`)

PHP

Exemple

- Si on voulait créer un attribut contenant le nombre d'objets créés à partir de la classe `Personne`
- Notre attribut doit être `static`, sinon chaque objet pourrait avoir sa propre valeur pour cet attribut

PHP

Ajoutons un attribut statique `$nbrPersonnes` à la liste d'attributs de la classe `Personne`

```php
private static int $nbrPersonnes = 0;
```

PHP

Ajoutons un attribut statique `$nbrPersonnes` à la liste d'attributs de la classe `Personne`

```
private static int $nbrPersonnes = 0;
```

Incrémentons notre compteur de personnes dans les constructeurs

```
public function __construct(int $num, string $nom, string
    $prenom)
{
    $this->setNum($num);
    $this->nom = $nom;
    $this->prenom = $prenom;
    self::$nbrPersonnes++;
}
```

PHP

Créons un `getter` pour l'attribut static `$nbrPersonnes`

```php
public static function getNbrPersonnes()
{
    return self::$nbrPersonnes;
}
```

PHP

Créons un `getter` pour l'attribut static `$nbrPersonnes`

```
public static function getNbrPersonnes()
{
    return self::$nbrPersonnes;
}
```

Testons cela dans `index.html`

```
echo Personne::getNbrPersonnes() . "<br>";
// affiche 0

$personne = new Personne(100,"wick","john");
echo $personne;
// affiche 100 wick john

echo Personne::getNbrPersonnes() . "<br>";
// affiche 1
```

PHP

Remarques

- Pour référencer un élément `static` à l'intérieur de la classe, on utilise `self`.
- Pour référencer un élément `static` à l'extérieur de la classe, on utilise le nom de la classe.
- Il est possible de remplacer `self` par le nom de la classe à l'intérieur de la classe
- Cependant, il est conseillé d'utiliser `self`.

PHP

Exemple explicatif montrant un cas où `self` et le nom de la classe sont différents (Source : *StackOverflow*)

```
class A {
    static function foo() {
        echo get_called_class();
    }
}
class B extends A {
    static function bar() {
        self::foo();
    }
    static function baz() {
        B::foo();
    }
}
class C extends B {}

C::bar(); // affiche C
C::baz(); // affiche B
```

PHP

`this` VS `self`

- on utilise $ avant `this` mais pas avant l'attribut qui le suit.
- on utilise $ avant l'attribut de la classe qui suit `self` mais pas avant `self`.

PHP

Constante de classe

C’est comme un attribut statique constant

PHP

Constante de classe

C'est comme un attribut statique constant

Syntaxe

```
<?php
    class NomClasse{
        //attributs
        ...
        //constantes
        const NOM_CONSTANTE = 'value';
        //méthodes
        ....
    }
?>
//Pour accéder :
NomClasse::NOM_CONSTANTE
```

PHP

Déclarons une constante `AGE_RETRAITE` dans la classe `Personne`

```
const AGE_RETRAITE = 62;
```

PHP

Déclarons une constante `AGE_RETRAITE` dans la classe `Personne`

```
const AGE_RETRAITE = 62;
```

Pour accéder à cette constante à l'intérieur de la classe (dans `index.php` par exemple

```
echo self::AGE_RETRAITE;
//affiche 62
```

PHP

Déclarons une constante `AGE_RETRAITE` dans la classe `Personne`

```
const AGE_RETRAITE = 62;
```

Pour accéder à cette constante à l'intérieur de la classe (dans `index.php` par exemple

```
echo self::AGE_RETRAITE;
//affiche 62
```

Pour accéder à cette constante à l'extérieur de la classe (dans `index.php` par exemple)

```
echo Personne::AGE_RETRAITE;
//affiche 62
```

PHP

Récapitulatif

- On recommande d'avoir une seule classe par fichier
- Une classe peut avoir comme attribut un objet d'une deuxième classe (définie dans un deuxième fichier)
- Il faut inclure la deuxième classe (avec `include` ou `require`) pour l'utiliser
- Et si dans un fichier on utilise plusieurs classes, faut-il les inclure toutes une par une ?
- Non on peut utiliser la fonction `spl_autoload_register` pour faire l'auto-chargement

PHP

Au tout début du fichier `index.php`, appelons la fonction `spl_autoload_register` qui prend comme paramètre un callback

```php
<?php
spl_autoload_register(function ($class) {
    include $class . '.php';
});
?>
```

PHP

Remarque

- La fonction précédente scanne tous les fichiers : y compris ceux qui ne contiennent pas de classe.
- Une bonne pratique consiste à regrouper les fichiers contenant de classes dans un répertoire (classes) par exemple et de scanner seulement ce dossier.

PHP

Modifions l'appel de `spl_autoload_register` après avoir déplacées toutes les classes dans un répertoire `classes`

```
<?php
spl_autoload_register(function ($class) {
    include "classes/" . $class . '.php';
});
?>
```

PHP

Remarque

On peut aussi créer plusieurs répertoires `models`, `controllers`... et utiliser plusieurs fois la fonction `spl_autoload_register` pour limiter la zone de recherche

PHP

L'héritage, quand ?

- Lorsque deux ou plusieurs classes partagent plusieurs attributs (et méthodes)
- Lorsqu'une `Classe1` est (**une sorte de**) `Classe2`

PHP

L'héritage, quand ?

- Lorsque deux ou plusieurs classes partagent plusieurs attributs (et méthodes)
- Lorsqu'une `Classe1` est (**une sorte de**) `Classe2`

Forme générale

```
class ClasseFille extends ClasseMère
{
    // code
};
```

PHP

Particularité du langage **PHP**

- Une classe ne peut hériter que d'une seule classe
- L'héritage multiple est donc non-autorisé.

PHP

Exemple

- Un enseignant a un numéro, un nom, un prénom et un salaire

PHP

Exemple

- Un enseignant a un numéro, un nom, un prénom et un salaire
- Un étudiant a aussi un numéro, un nom, un prénom et un niveau

PHP

Exemple

- Un enseignant a un numéro, un nom, un prénom et un salaire
- Un étudiant a aussi un numéro, un nom, un prénom et un niveau
- Sémantiquement, enseignant et étudiant sont une sorte de personne

PHP

Exemple

- Un enseignant a un numéro, un nom, un prénom et un salaire
- Un étudiant a aussi un numéro, un nom, un prénom et un niveau
- Sémantiquement, enseignant et étudiant sont une sorte de personne
- En plus, les deux partagent plusieurs attributs tels que `numéro`, `nom` et `prénom`

PHP

Exemple

- Un enseignant a un numéro, un nom, un prénom et un salaire
- Un étudiant a aussi un numéro, un nom, un prénom et un niveau
- Sémantiquement, enseignant et étudiant sont une sorte de personne
- En plus, les deux partagent plusieurs attributs tels que `numéro`, `nom` et `prénom`
- Donc, on peut utiliser la classe `Personne` puisqu'elle contient tous les attributs `numéro`, `nom` et `prénom`

PHP

Exemple

- Un enseignant a un numéro, un nom, un prénom et un salaire
- Un étudiant a aussi un numéro, un nom, un prénom et un niveau
- Sémantiquement, enseignant et étudiant sont une sorte de personne
- En plus, les deux partagent plusieurs attributs tels que `numéro`, `nom` et `prénom`
- Donc, on peut utiliser la classe `Personne` puisqu'elle contient tous les attributs `numéro`, `nom` et `prénom`
- Les classes `Étudiant` et `Enseignant` hériteront donc (`extends`) de la classe `Personne`

PHP

Créons la classe `Enseignant`

```
class Enseignant
{
}
```

PHP

Créons la classe `Enseignant`

```
class Enseignant
{
}
```

Faisons hériter `Enseignant` **de** `Personne` **grâce au mot clé** `extends`

```
class Enseignant extends Personne
{
}
```

PHP

Pour créer un objet de type `Enseignant`

```
$enseignant = new Enseignant(101, "benamar", "karim");
echo $enseignant;
// affiche 101 benamar karim
```

PHP

Pour créer un objet de type `Enseignant`

```php
$enseignant = new Enseignant(101, "benamar", "karim");
echo $enseignant;
// affiche 101 benamar karim
```


Exercice

Faire la même chose pour `Etudiant`.

PHP

Ensuite

- Ajoutons un attribut `$niveau` dans la classe `Etudiant` ainsi que ses getter et setter
- Ajoutons un attribut `$salaire` dans la classe `Enseignant` ainsi que ses getter et setter

PHP

Nouveau contenu de la classe `Etudiant`

```php
class Etudiant extends Personne
{
    private string $niveau;

    public function getNiveau(): string
    {
        return $this->niveau;
    }

    public function setNiveau(string $niveau): void
    {
        $this->niveau = $niveau;
    }
}
```

PHP

Nouveau contenu de la classe `Etudiant`

```php
class Etudiant extends Personne
{
    private string $niveau;

    public function getNiveau(): string
    {
        return $this->niveau;
    }

    public function setNiveau(string $niveau): void
    {
        $this->niveau = $niveau;
    }
}
```

Et comment introduire le `niveau` dans le constructeur ?

Pour introduire le `niveau` dans le constructeur la classe `Etudiant`

```
class Etudiant extends Personne
{
    private string  $niveau;

    public function __construct(int $num, string $nom, string $prenom,
       string $niveau)
    {
        parent::__construct($num, $nom, $prenom);
        $this->niveau = $niveau;
    }
    public function getNiveau(): string
    {
        return $this->niveau;
    }
    public function setNiveau(string $niveau): void
    {
        $this->niveau = $niveau;
    }
}
```

PHP

Explication

- `parent` ⇒ mot-clé pour accéder à la classe mère
- `parent::` ⇒ pointer un élément de la classe mère

On peut simplifier avec le Constructor Promotion

```
class Etudiant extends Personne
{
    public function __construct( int $num, string $nom, string $prenom,
        private string $niveau)
    {
        parent::__construct($num, $nom, $prenom);
    }
    public function getNiveau(): string
    {
        return $this->niveau;
    }
    public function setNiveau(string $niveau): void
    {
        $this->niveau = $niveau;
    }
}
```

PHP

Pour créer un objet de type `Etudiant`

```php
$etudiant = new Etudiant(102, "maggio", "carol", 'master');
echo $etudiant;
// affiche 102 maggio carol
```

PHP

Pour créer un objet de type `Etudiant`

```
$etudiant = new Etudiant(102, "maggio", "carol", 'master');
echo $etudiant;
// affiche 102 maggio carol
```


Question

Le niveau n'a pas été affiché, pourquoi ?

PHP

Pour créer un objet de type `Etudiant`

```
$etudiant = new Etudiant(102, "maggio", "carol", 'master');
echo $etudiant;
// affiche 102 maggio carol
```


Question

Le niveau n'a pas été affiché, pourquoi ?

Réponse

On n'a pas redéfini la méthode `__toString()`, on a utilisé celle de la classe mère.

PHP

À partir de la classe `Enseignant` ou `Etudiant`

- On ne peut avoir accès direct aux attributs de la classe mère
- C'est-à-dire, on ne peut faire `$this->_num` car les attributs ont une visibilité `private`
- Pour modifier la valeur d'un attribut privé de la classe mère, il faut
  - soit utiliser les getters/setters
  - soit mettre la visibilité des attributs de la classe mère à `protected`

PHP

Remarque

Pour connaître la classe d'un objet, on peut utiliser le mot-clé `instanceof`

PHP

Remarque

Pour connaître la classe d'un objet, on peut utiliser le mot-clé `instanceof`

Exemple

```php
var_dump($etudiant instanceof Etudiant);
// affiche true

var_dump($etudiant instanceof Personne);
// affiche true

var_dump($personne instanceof Etudiant);
// affiche false
```

Exercice

1. Créer un objet de type `Etudiant`, un deuxième de type `Enseignant` et un dernier de type `Personne` puis stocker les tous dans un seul tableau.
2. Parcourir le tableau et afficher pour chacun soit le `numero` s'il est personne, soit le `salaire` s'il est enseignant ou soit le `niveau` s'il est étudiant.

Exercice

1. Créer un objet de type `Etudiant`, un deuxième de type `Enseignant` et un dernier de type `Personne` puis stocker les tous dans un seul tableau.
2. Parcourir le tableau et afficher pour chacun soit le `numero` s'il est personne, soit le `salaire` s'il est enseignant ou soit le `niveau` s'il est étudiant.

Pour répondre à la question, vous pouvez utiliser le code suivant

```
$personne = new Personne(106, "wick", "john");
$etudiant = new Etudiant(107, "maggio", "carol", 'master');
$enseignant = new Enseignant(108, "baggio", "roberto", 450);

$personnes = [
    $personne,
    $enseignant,
    $etudiant
];
foreach ($personnes as $perso) {
    // réponse à l'exercice
}
```

PHP

Solution

```
foreach ($personnes as $perso) {
    if ($perso instanceof Enseignant) {
        echo $perso->getSalaire() . '<br>';
    } else if ($perso instanceof Etudiant) {
        echo $perso->getNiveau() . '<br>';
    } else {
        echo $perso->getNum() . '<br>';
    }
}
```

PHP

Overloading

La surcharge n'est pas autorisée en PHP, c'est-à-dire on ne peut pas définir plusieurs méthodes avec des signatures (nombre et type de paramètre) différentes.

PHP

La redéfinition en PHP

On peut redéfinir

- les attributs
- les méthodes

PHP

Exemple de redéfinition d'attribut

```
class Enseignant extends Personne
{

    // attributs de la classe fille
    private int $salaire;
    private string $nom; // existe dans Personne

    // les méthodes
}
```

PHP

Exemple de redéfinition d'attribut

```
class Enseignant extends Personne
{

    // attributs de la classe fille
    private int $salaire;
    private string $nom; // existe dans Personne

    // les méthodes
}
```

Overriding

On peut aussi redéfinir le setter et le getter de l'attribut $nom (ce n'est pas obligatoire).

Dans `Personne`, ajoutons la méthode `afficherDetails()` suivante

```
public function afficherDetails()
{
    echo $this->nom . " " . $this->prenom;
}
```

Dans `Personne`, ajoutons la méthode `afficherDetails()` suivante

```
public function afficherDetails()
{
    echo $this->nom . " " . $this->prenom;
}
```

Dans `Etudiant`, nous pouvons redéfinir la méthode `afficherDetails()`

```
public function afficherDetails()
{
    echo $this->getNom() . " " . $this->getPrenom() . " " . $this->
       niveau;
}
```

Dans `Personne`, ajoutons la méthode `afficherDetails()` suivante

```
public function afficherDetails()
{
    echo $this->nom . " " . $this->prenom;
}
```

Dans `Etudiant`, nous pouvons redéfinir la méthode `afficherDetails()`

```
public function afficherDetails()
{
    echo $this->getNom() . " " . $this->getPrenom() . " " . $this->
       niveau;
}
```

Dans `index.php`, voici comment tester les deux méthodes

```
$personne = new Personne(100, "wick", "john");
$personne->afficherDetails();
// affiche wick john

$etudiant = new Etudiant(101, "maggio", "carol", 'master');
$etudiant->afficherDetails();
// affiche maggio carol master
```

PHP

Dans `Etudiant`, nous pouvons aussi utiliser la méthode `afficherDetails()` de la classe mère

```
public function afficherDetails()
{
    parent::afficherDetails();
    echo " " . $this->niveau;
}
```

PHP

Dans `Etudiant`, nous pouvons aussi utiliser la méthode `afficherDetails()` de la classe mère

```
public function afficherDetails()
{
    parent::afficherDetails();
    echo " " . $this->niveau;
}
```

Dans `index.php`, le test ne change pas

```
$personne = new Personne(100, "wick", "john");
$personne->afficherDetails();
// affiche wick john

$etudiant = new Etudiant(101, "maggio", "carol", 'master');
$etudiant->afficherDetails();
// affiche maggio carol master
```

PHP

Classe abstraite

- C'est une classe qu'on ne peut instancier
- On la déclare avec le mot-clé `abstract`

PHP

Classe abstraite

- C'est une classe qu'on ne peut instancier
- On la déclare avec le mot-clé `abstract`

Syntaxe

```
abstract class NomClasse
{
    // le code
}
```

PHP

Exemple

```
abstract class Personne{
    // attributs
        ...

    // méthodes
        ...
}

$Pers = new Personne();
//ça génère une erreur fatale
```

PHP

Méthode abstraite

- C'est une méthode non implémentée (sans code ⇒ sans accolades). Nous devons juste préciser :
  - sa visibilité (`public`, `protected` ou `private`).
  - sa signature (nom et type de paramètres) [si elle prend de paramètres].
- Une méthode abstraite doit être déclarée dans une classe abstraite
- Une méthode abstraite doit être implémentée par les classes filles de la classe abstraite

PHP

Méthode abstraite

- C'est une méthode non implémentée (sans code ⇒ sans accolades). Nous devons juste préciser :
  - sa visibilité (`public`, `protected` ou `private`).
  - sa signature (nom et type de paramètres) [si elle prend de paramètres].
- Une méthode abstraite doit être déclarée dans une classe abstraite
- Une méthode abstraite doit être implémentée par les classes filles de la classe abstraite

Syntaxe

```
abstract public function nomMethode();
```

PHP

Déclarons une méthode abstraite `afficherCaracteristiques()` **dans** `Personne`

```php
abstract public function afficherCaracteristiques(): void;
```

PHP

Déclarons une méthode abstraite `afficherCaracteristiques()` **dans** `Personne`

```php
abstract public function afficherCaracteristiques(): void;
```


Remarque

La méthode `afficherCaracteristiques()` dans `Personne` est soulignée en rouge car la classe doit être déclarée abstraite.

PHP

Déclarons la méthode dans `Enseignant`

```
public function afficherCaracteristiques(): void
{
    echo "Enseignant : salaire = $this->salaire";
}
```

PHP

Déclarons la méthode dans `Enseignant`

```
public function afficherCaracteristiques(): void
{
    echo "Enseignant : salaire = $this->salaire";
}
```

Et dans `Etudiant`

```
public function afficherCaracteristiques(): void
{
    echo "Etudiant : niveau = $this->niveau";
}
```

PHP

Pour tester

```
$etudiant = new Etudiant(102, "maggio", "carol", 'master');
$etudiant->afficherCaracteristiques();
// affiche Etudiant : niveau = master
```

PHP

Classe finale

C’est une classe qui ne peut avoir de classes filles

PHP

Classe finale

C'est une classe qui ne peut avoir de classes filles

Syntaxe

```
final class NomClasse{
    // le code
}
```

PHP

Pour tester, commentez la méthode abstraite de classe `Personne` et supprimez le mot-clé `abstract` de la déclaration de `Personne`.

PHP

Pour tester, commentez la méthode abstraite de classe `Personne` et supprimez le mot-clé `abstract` de la déclaration de `Personne`.

Exemple

```
final class Personne{
        // le code
}

class Etudiant extends Personne{
        // le code
}

//ça génère une erreur fatale
```

PHP

Pour tester, commentez la méthode abstraite de classe `Personne` et supprimez le mot-clé `abstract` de la déclaration de `Personne`.

Exemple

```
final class Personne{
        // le code
}

class Etudiant extends Personne{
        // le code
}

//ça génère une erreur fatale
```

Les deux classes filles sont affichées en rouge

```
The type Etudiant cannot subclass the final class Personne
```

PHP

Méthode finale

C’est une méthode qu’on ne peut redéfinir

PHP

Méthode finale

C'est une méthode qu'on ne peut redéfinir

Pour tester

Commençons par supprimer le mot-clé `final` dans la classe `Personne`

Exemple

```
class Personne
{
    // code précédent
    final public function decrire(){
        ...
    }
}
class Etudiant extends Personne
{
    // code précédent
    final public function decrire(){
        ...
    }
}
```

Exemple

```
class Personne
{
    // code précédent
    final public function decrire(){
        ...
    }
}
class Etudiant extends Personne
{
    // code précédent
    final public function decrire(){
        ...
    }
}
```

Le code précédent génère une erreur fatale :

```
Cannot override the final method from Personne
```

PHP

Remarques

- Une classe abstraite ne doit pas forcément contenir une méthode abstraite
- Une classe finale ne doit pas forcément contenir une méthode finale
- Une méthode finale ne doit pas forcément être dans une classe finale

PHP

Fonctions utiles pour les classes

- `get_class($obj)` : retourne le nom de la classe de l'objet `$obj`
- `get_class_methods(NomClasse)` : retourne un tableau contenant les méthodes de la classe `NomClasse`
- `class_exists(NomClasse)` : vérifie si la classe `NomClasse` existe
- `get_object_vars($obj)` : retourne un tableau contenant les attributs de l'objet (`$obj`)
- `get_parent_class($obj)` : Retourne le nom de la classe mère d'un objet `$obj`
- autres : `is_a`, `is_subclass_of`, `method_exists`, `property_exists`...

Exercice

1. Créer un objet de type `Etudiant`, un deuxième de type `Enseignant` et un dernier de type `Personne` puis stocker les tous dans un seul tableau.
2. Parcourir le tableau et afficher pour chacun soit le `numero` s'il est personne, soit le `salaire` s'il est enseignant ou soit le `niveau` s'il est étudiant (en utilisant `get_class`).

Exercice

1. Créer un objet de type `Etudiant`, un deuxième de type `Enseignant` et un dernier de type `Personne` puis stocker les tous dans un seul tableau.
2. Parcourir le tableau et afficher pour chacun soit le `numero` s'il est personne, soit le `salaire` s'il est enseignant ou soit le `niveau` s'il est étudiant (en utilisant `get_class`).

Pour répondre à la question, vous pouvez utiliser le code suivant

```
$personne = new Personne(106, "wick", "john");
$etudiant = new Etudiant(107, "maggio", "carol", 'master');
$enseignant = new Enseignant(108, "baggio", "roberto", 450);

$personnes = [
    $personne,
    $enseignant,
    $etudiant
];
foreach ($personnes as $perso) {
    // réponse à l'exercice
}
```

PHP

Solution

```
foreach ($personnes as $perso) {
    $classe = get_class($perso);
    if ($classe == "Enseignant") {
        echo $perso->getSalaire() . '<br>';
    } else if ($classe ==  "Etudiant") {
        echo $perso->getNiveau() . '<br>';
    } else {
        echo $perso->getNum() . '<br>';
    }
}
```

PHP

Refaire l'exercice précédent sans `get_class` ni `instanceof`

```
$personne = new Personne(106, "wick", "john");
$etudiant = new Etudiant(107, "maggio", "carol", '
   master');
$enseignant = new Enseignant(108, "baggio", "roberto
   ", 450);

$personnes = [
    $personne,
    $enseignant,
    $etudiant
];
foreach ($personnes as $perso) {
    // réponse à l'exercice
}
```

PHP

Solution

```
foreach ($personnes as $perso) {
    if (method_exists($perso, "getSalaire")) {
        echo $perso->getSalaire() . '<br>';
    } else if (method_exists($perso, "getNiveau")) {
        echo $perso->getNiveau() . '<br>';
    } else {
        echo $perso->getNum() . '<br>';
    }
}
```

PHP

- L'héritage multiple n'est pas autorisé en **PHP**.
- Donc une classe ne peut étendre qu'une seule classe.
- Mais elle peut implémenter plusieurs interfaces.

PHP

- L'héritage multiple n'est pas autorisé en **PHP**.
- Donc une classe ne peut étendre qu'une seule classe.
- Mais elle peut implémenter plusieurs interfaces.

interface ?

PHP

Une interface

- déclarée avec le mot-clé `interface`
- comme une classe abstraite (impossible de l'instancier)
  - dont toutes les méthodes sont abstraites
  - qui n'a pas d'attribut
- un protocole, un contrat : toute classe qui hérite d'une interface doit implémenter toutes ses méthodes
- pas besoin du mot-clé `abstract` pour les méthodes d'une interface
- pouvant contenir des constantes mais pas des attributs statiques.

Interface

Syntaxe

```
interface NomInterface
{
    ...
}
```

PHP

Créons l'interface `IMiseEnForme`

```
interface IMiseEnForme
{
}
```

PHP

Créons l'interface `IMiseEnForme`

```
interface IMiseEnForme
{
}
```

Définissons la signature de ces deux méthodes dans l'interface `IMiseEnForme`

```
interface IMiseEnForme
{
    public function afficherNomMajuscule(): void;
    public function afficherPrenomMajuscule(): void;
}
```

PHP

Pour hériter d'une interface, on utilise le mot-clé `implements`

```
class Personne implements IMiseEnForme
{
    ...
}
```

PHP

Pour hériter d'une interface, on utilise le mot-clé `implements`

```
class Personne implements IMiseEnForme
{
    ...
}
```


Remarque

La classe `Personne` est soulignée en rouge car elle n'implémente pas les méthodes abstraites de `IMiseEnForme`.

PHP

Déclarons les méthodes abstraites de `IMiseEnForme`

```php
public function afficherNomMajuscule(): void
{}

public function afficherPrenomMajuscule(): void
{}
```

PHP

Déclarons les méthodes abstraites de `IMiseEnForme`

```
public function afficherNomMajuscule(): void
{}

public function afficherPrenomMajuscule(): void
{}
```

Puis ajoutons le code suivant pour les deux dernières méthodes

```
public function afficherNomMajuscule(): void
{
    echo strtoupper($this->nom);
}

public function afficherPrenomMajuscule(): void
{
    echo strtoupper($this->prenom);
}
```

PHP

Pour tester, voici le contenu d'`index.php`

```
$etudiant = new Etudiant(102, "maggio", "carol", '
   master');

$etudiant->afficherNomMajuscule();
// affiche MAGGIO

$etudiant->afficherPrenomMajuscule();
// affiche CAROL
```

PHP

Remarques

- Une interface peut hériter de plusieurs autres interfaces (mais pas d'une classe)
- Pour cela, il faut utiliser le mot-clé `extends` et pas `implements` car une interface n'implémente jamais de méthodes.

PHP

Remarques

- Une interface peut hériter de plusieurs autres interfaces (mais pas d'une classe)
- Pour cela, il faut utiliser le mot-clé `extends` et pas `implements` car une interface n'implémente jamais de méthodes.

Exemple

```
interface I extends I2, I3
    ....
}
```

PHP

Fonctions utiles pour les interfaces

- `get_declared_interfaces()` : retourne un tableau contenant toutes les interfaces déclarées
- `class_implements()` : retourne les interfaces implémentées par une classe ou une interface donnée
- `class_exists()` : vérifie si une classe a été définie
- `interface_exists` : vérifie si une interface existe

PHP

Fonctions utiles pour les interfaces

- `get_declared_interfaces()` : retourne un tableau contenant toutes les interfaces déclarées
- `class_implements()` : retourne les interfaces implémentées par une classe ou une interface donnée
- `class_exists()` : vérifie si une classe a été définie
- `interface_exists` : vérifie si une interface existe


```
if (interface_exists('Imposable')) {
    class Personne implements Imposable
    {
        ....
    }
}
```

PHP

- L'héritage multiple n'est pas autorisé en **PHP**.
- Mais une classe peut implémenter plusieurs interfaces.
- Les interfaces ne prennent pas d'attributs.
- Nous ne pouvons pas implémenter deux interfaces qui ont deux méthodes identiques.

PHP

- L'héritage multiple n'est pas autorisé en **PHP**.
- Mais une classe peut implémenter plusieurs interfaces.
- Les interfaces ne prennent pas d'attributs.
- Nous ne pouvons pas implémenter deux interfaces qui ont deux méthodes identiques.

Trait ?

PHP

Un trait

- Un moyen de réutiliser le code (les méthodes)
- Déclaré avec le mot-clé `trait`

PHP

Un trait

- Un moyen de réutiliser le code (les méthodes)
- Déclaré avec le mot-clé `trait`

Notation

- Une classe (fille) étend (`extends`) une classe (mère)
- Une classe implémente (`implements`) une interface
- Une classe utilise (`use`) un trait

PHP

Syntaxe

```
trait NomTrait
{
    ...
}
```

PHP

Commençons par créer le trait suivant

```
trait France
{
}
```

PHP

Commençons par créer le trait suivant

```
trait France
{
}
```

Ajoutons une méthode à ce trait

```
trait France
{
    public function salutation()
    {
        echo "salut";
    }
}
```

PHP

Utilisons le trait `France` dans la classe `Personne`

```
class Personne implements IMiseEnForme
{

     // attributs, constantes...

    use France;

    // méthodes

}
```

PHP

Utilisons le trait `France` dans la classe `Personne`

```
class Personne implements IMiseEnForme
{

     // attributs, constantes...

    use France;

    // méthodes

}
```


Pour tester le trait dans `index.php`

```
$etudiant = new Etudiant(101, "maggio", "carol", 'master');
$etudiant->salutation();
// affiche salut
```

PHP

Propriétés

- Une classe peut utiliser un ou plusieurs traits.
- Une classe peut utiliser plusieurs traits qui ont une méthode avec le même nom.
- Il suffit de préciser quelle méthode sera utilisée par les objets de la classe.

Définissons un deuxième trait `America`

```
trait America
{
    public function salutation()
    {
        echo "hi";
    }
}
```

Définissons un deuxième trait `America`

```
trait America
{
    public function salutation()
    {
        echo "hi";
    }
}
```

Si la classe `Personne` **utilise les deux traits, elle doit préciser quelle méthode** `salutation` **préfère utiliser**

```
class Personne implements IMiseEnForme
{

     // attributs, constantes...

    use France, America {
        America::salutation insteadof France;
    }

    // méthodes

}
```

PHP

Pour tester le trait dans `index.php`

```
$etudiant = new Etudiant(101, "maggio", "carol", 'master'
   );
$etudiant->salutation();
// affiche hi
```

PHP

Et si la classe `Personne` avait une méthode `salutation`

```
class Personne implements IMiseEnForme
{
    // contenu précédent
    public function salutation(){
        echo "hola";
    }
}
```

PHP

Et si la classe `Personne` avait une méthode `salutation`

```
class Personne implements IMiseEnForme
{
    // contenu précédent
    public function salutation(){
        echo "hola";
    }
}
```


C'est la méthode de la classe `Personne` qui sera exécutée (`index.php`)

```
$etudiant = new Etudiant(101, "maggio", "carol", 'master');
$etudiant->salutation();
// affiche hola
```

Propriété

Une classe qui utilise un trait peut renommer les méthodes de ce dernier.

Propriété

Une classe qui utilise un trait peut renommer les méthodes de ce dernier.

Exemple : si on voulait utiliser les deux méthodes salutations ? on peut dans ce cas renommer les méthodes avec le mot-clé `as`

```
class Personne implements IMiseEnForme
{
    // contenu précédent
    use France, America {
        America::salutation as sayHello;
        France::salutation as direBonjour;
    }
    public function salutation()
    {
        $this->direBonjour();
        echo '<br>';
        $this->sayHello();
    }
}
```

PHP

C'est la méthode de la classe `Personne` qui sera exécutée (`index.php`)

```
$etudiant = new Etudiant(101, "maggio", "carol", 'master');
$etudiant->salutation();
// affiche
// salut
// hi
```

PHP

Propriétés

- Un trait peut utiliser un ou plusieurs autres traits.
- Un trait ou une classe qui utilise un trait peut renommer les méthodes de ce dernier.
- Un trait peut avoir accès aux membres de la classe qui l'utilise

PHP

Propriétés

- Un trait peut utiliser un ou plusieurs autres traits.
- Un trait ou une classe qui utilise un trait peut renommer les méthodes de ce dernier.
- Un trait peut avoir accès aux membres de la classe qui l'utilise

Pour la suite, dans la classe `Personne`, commenter

- la méthode `salutation`
- le `use`

PHP

Créons le trait `Bilingue` qui utilise les deux traits `France` et `America` et qui renomme la méthode salutation

```
trait Bilingue
{
    use France, America {
        America::salutation as sayHello;
        France::salutation as direBonjour;
    }
    public function salutation(string $lang)
    {
        if ($lang === "fr")
            $this->direBonjour();
        else
            $this->sayHello();
    }
}
```

PHP

La classe `Personne` **utilise** `Bilingue`

```
class Personne implements IMiseEnForme
{
    // attributs, constantes

    use Bilingue;

    // méthodes
}
```

PHP

La classe `Personne` **utilise** `Bilingue`

```
class Personne implements IMiseEnForme
{
    // attributs, constantes

    use Bilingue;

    // méthodes
}
```


Pour tester dans `index.php`

```
$etudiant = new Etudiant(101, "maggio", "carol", 'master');
$etudiant->salutation("fr");
// affiche salut

$etudiant->salutation("en");
// affiche hi
```

PHP

Modifions le trait `France` en ajoutant une méthode `salutationPersonnalisee`

```
trait France
{
    public function salutation()
    {
        echo "salut";
    }
    public function salutationPersonnalisee()
    {
        echo "salut $this->nom";
    }
}
```

PHP

Pour tester dans `index.php`

```
$etudiant = new Etudiant(101, "maggio", "carol", '
   master');
$etudiant->salutationPersonnalisee();
// affiche maggio
```

PHP

Fonctions utiles pour les traits

- `class_uses($class)` : retourne un tableau de traits que la classe en paramètre utilise
- `get_declared_traits()` : retourne un tableau de tous les traits déclarés
- `trait_exists(Trait)` : retourne `true` si le trait en paramètre existe, `false` sinon.

PHP

Référence d'objet

```php
$etudiant = new Etudiant(101, "maggio", "carol", 'master');
$etudiant2 = $etudiant;
$etudiant2->setNom("baggio");
echo $etudiant->getNom();
// affiche baggio
```

PHP

Référence d'objet

```php
$etudiant = new Etudiant(101, "maggio", "carol", 'master');
$etudiant2 = $etudiant;
$etudiant2->setNom("baggio");
echo $etudiant->getNom();
// affiche baggio
```


`$etudiant` et `$etudiant2` pointent sur le même objet

PHP

Référence d'objet

```php
$etudiant = new Etudiant(101, "maggio", "carol", 'master');
$etudiant2 = $etudiant;
$etudiant2->setNom("baggio");
echo $etudiant->getNom();
// affiche baggio
```


`$etudiant` et `$etudiant2` pointent sur le même objet

Et si on voulait seulement récupérer une copie de `$etudiant` sans le modifier ?

PHP

Solution

```
$etudiant = new Etudiant(101, "maggio", "carol", '
   master');
$etudiant2 = clone $etudiant;
$etudiant2->setNom("baggio");
echo $etudiant->getNom();
// affiche maggio
```

PHP

Si on veut faire un traitement particulier lorsqu'on clone un objet

```
class Personne
{
    // contenu précédent

    public function __clone(){
        self::$nbrPersonnes++;
    }
}
```

PHP

Si on veut faire un traitement particulier lorsqu'on clone un objet

```
class Personne
{
    // contenu précédent

    public function __clone(){
        self::$nbrPersonnes++;
    }
}
```


Remarque

La méthode `__clone()` appartient à l'objet appelé.

PHP

Pour définir un constructeur qui accepte un nombre variable de paramètre, on utilise les fonctions suivantes

- `func_num_args()` : retourne le nombre d'arguments passés à une fonction
- `func_get_arg(int arg_num)` : retourne l'argument d'indice `arg_num` de la fonction
- `func_get_args()` : retourne un tableau contenant les arguments d'une fonction

PHP

Objectif

```php
<?php
    class Constructeur
    {
        private $a;
        private $b;
        private $c;
        private $d;
    }
    $c1 = new Constructeur(1);
    $c2 = new Constructeur(1 ,2);
    $c3 = new Constructeur(1, 2, 3);
    $c4 = new Constructeur(1, 2, 3, 4);
?>
```

PHP

Objectif

```php
<?php
    class Constructeur
    {
        private $a;
        private $b;
        private $c;
        private $d;
    }
    $c1 = new Constructeur(1);
    $c2 = new Constructeur(1 ,2);
    $c3 = new Constructeur(1, 2, 3);
    $c4 = new Constructeur(1, 2, 3, 4);
?>
```

Impossible d'appeler plusieurs constructeurs avec des signatures différentes.

PHP

Dans la classe `Constructeur`, définissons le constructeur suivant

```
public function __construct()
{
    $nbr = func_num_args();
    $args = func_get_args();
    switch ($nbr) {
        case 4:
            $this->d = $args[3];
        case 3:
            $this->c = $args[2];
        case 2:
            $this->b = $args[1];
        case 1:
            $this->a = $args[0];
    }
}
```

Ajoutons aussi la méthode `__toString`

```
function  __toString(): string {
    return $this->a . $this->b . $this->c .$this->d;
}
```

Ajoutons aussi la méthode `__toString`

```
function  __toString(): string {
    return $this->a . $this->b . $this->c .$this->d;
}
```

Testons cela dans `index.php`

```
$c1 = new Constructeur(1);
echo($c1);
// affiche 1

$c2 = new Constructeur(1 ,2);
echo($c2);
// affiche 12

$c3 = new Constructeur(1, 2, 3);
echo($c3);
// affiche 123

$c4 = new Constructeur(1, 2, 3, 4);
echo($c4);
// affiche 1234
```

PHP

Les méthodes magiques ont la forme `__nomMéthode`

PHP

Les méthodes magiques ont la forme `__nomMéthode`

Quelques exemples qu'on connaît

- `__construct`
- `__destruct`
- `__clone`
- `__tostring`

PHP

Et il existe plusieurs autres...

- `__get`
- `__set`
- `__isset`
- `__unset`
- ...

PHP

Pour définir un seul getter pour tous les attributs de la classe `Constructeur`, on peut définir un seul en utilisant la méthode magique `__get`

```
public function __get($element)
{
    if (isset($element)) {
        return $this->$element;
    }
}
```

PHP

Pour définir un seul getter pour tous les attributs de la classe `Constructeur`, on peut définir un seul en utilisant la méthode magique `__get`

```
public function __get($element)
{
    if (isset($element)) {
        return $this->$element;
    }
}
```


Pour tester dans `index.php`

```
$c4 = new Constructeur(1, 2, 3, 4);
echo($c4->__get("_a"));
// affiche 1
```

PHP

Idem pour `__set`

```
public function __set($element, $val)
{
    if (isset($element)) {
        $this->$element = $val;
    }
}
```

PHP

Idem pour `__set`

```
public function __set($element, $val)
{
    if (isset($element)) {
        $this->$element = $val;
    }
}
```

Pour tester dans `index.php`

```
$c4 = new Constructeur(1, 2, 3, 4);
echo($c4->__get("_a"));
// affiche 1

$c4->set("_a", 5);
echo($c4->__get("_a"));
// affiche 5
```

PHP

`isset` ne peut vérifier l'existence des attributs privé d'une classe

```php
var_dump(isset($c4->a));
// affiche false
```

PHP

`isset` **ne peut vérifier l'existence des attributs privé d'une classe**

```
var_dump(isset($c4->a));
// affiche false
```

Pour pouvoir vérifier l'existence d'un attribut, il faut dans ce cas définir `__isset`

```
public function __isset($nom)
{
    return isset($this->$nom);
}
```

PHP

`isset` ne peut vérifier l'existence des attributs privé d'une classe

```
var_dump(isset($c4->a));
// affiche false
```

Pour pouvoir vérifier l'existence d'un attribut, il faut dans ce cas définir `__isset`

```
public function __isset($nom)
{
    return isset($this->$nom);
}
```

En testant, le résultat est `true`

```
var_dump(isset($c4->a));
// affiche true
```

PHP

Exercice

- Créez une classe `Dynamique`
- Cette classe peut avoir un nombre variable d'attributs privés
- Le constructeur de la classe prend comme paramètre les noms des attributs
- On peut accéder à ces attributs via les méthodes magiques `__get`, `__set`, `__isset` et `__toString`

PHP

Exercice

- Créez une classe `Dynamique`
- Cette classe peut avoir un nombre variable d'attributs privés
- Le constructeur de la classe prend comme paramètre les noms des attributs
- On peut accéder à ces attributs via les méthodes magiques `__get`, `__set`, `__isset` et `__toString`

Exemple d'utilisation de la classe `Dynamique` dans `index.php`

```
$objet = new Dynamique('nom', 'prenom');
$objet->set('nom', 'wick');
$objet->set('prenom', 'john');
echo $objet;
if (isset($objet->nom))
    echo ($objet->get('nom'));
```

Correction

```
class Dynamique
{
    private $attributs = [];
    public function __construct (...$tab)
    {
        foreach ($tab as  $value)
            $this->attributs[$value] = '';
    }
    public function __get($element)
    {
        if (isset($this->attributs[$element])) {
            return $this->attributs[$element];
        }
    }
    public function __set($element, $val)
    {
        if (isset($this->attributs[$element])) {
            $this->attributs[$element] = $val;
        }
    }
    public function __isset($nom)
    {
        return isset($this->attributs[$nom]);
    }
    public function __toString()
    {
        $str = '';
        foreach ($this->attributs as $key => $value)
           $str .= "$key : $value <br>";
        return $str;
    }
}
```

PHP

Constantes magiques

Des constantes prédéfinies ayant la forme `__CONSTANTE__`

PHP

Constantes magiques

Des constantes prédéfinies ayant la forme `__CONSTANTE__`

Exemple

- `__CLASS__`
- `__FUNCTION__`
- `__DIR__`
- `__METHOD__`
- `__TRAIT__`
- `__NAMESPACE__`
- `__LINE__`

PHP

Considérons la classe `Classe` suivante

```
class Classe
{
    public function __construct()
    {
        echo __CLASS__;
    }
    function __destruct()
    {}
}
```

PHP

Considérons la classe `Classe` suivante

```
class Classe
{
    public function __construct()
    {
        echo __CLASS__;
    }
    function __destruct()
    {}
}
```

Pour tester dans `index.php`

```
$classe = new Classe();
// affiche Classe
```

PHP

Ajoutons la constante __DIR__

```
class Classe
{
    public function __construct()
    {
        echo __DIR__;
    }
    function __destruct()
    {}
}
```

PHP

Ajoutons la constante `__DIR__`

```
class Classe
{
    public function __construct()
    {
        echo __DIR__;
    }
    function __destruct()
    {}
}
```

Pour tester dans `index.php`

```
$classe = new Classe();
// affiche C:\wamp64\www\cours-poo\classes
```

PHP

Ajoutons la constante __FILE__

```
class Classe
{
    public function __construct()
    {
        echo __FILE__;
    }
    function __destruct()
    {}
}
```

PHP

Ajoutons la constante `__FILE__`

```
class Classe
{
    public function __construct()
    {
        echo __FILE__;
    }
    function __destruct()
    {}
}
```

Pour tester dans `index.php`

```
$classe = new Classe();
// affiche C:\wamp64\www\cours-poo\classes\Classe.
   php
```

PHP

Ajoutons la constante __FUNCTION__

```
class Classe
{
    public function __construct()
    {
        echo __FUNCTION__;
    }
    function __destruct()
    {}
}
```

PHP

Ajoutons la constante `__FUNCTION__`

```
class Classe
{
    public function __construct()
    {
        echo __FUNCTION__;
    }
    function __destruct()
    {}
}
```

Pour tester dans `index.php`

```
$classe = new Classe();
// affiche __construct
```

PHP

Ajoutons la constante __METHOD__

```
class Classe
{
    public function __construct()
    {
        echo __METHOD__;
    }
    function __destruct()
    {}
}
```

PHP

Ajoutons la constante `__METHOD__`

```
class Classe
{
    public function __construct()
    {
        echo __METHOD__;
    }
    function __destruct()
    {}
}
```

Pour tester dans `index.php`

```
$classe = new Classe();
// affiche Classe::__construct
```

PHP

Ajoutons la constante __LINE__

```
class Classe
{
    public function __construct()
    {
        echo __LINE__;
    }
    function __destruct()
    {}
}
```

PHP

Ajoutons la constante `__LINE__`

```
class Classe
{
    public function __construct()
    {
        echo __LINE__;
    }
    function __destruct()
    {}
}
```

Pour tester dans `index.php`

```
$classe = new Classe();
// affiche 7
```

PHP

Sérialisation ou linéarisation

Transformation d'objet complexe pour pouvoir le transmettre et/ou le stocker.

PHP

Sérialisation ou linéarisation

Transformation d'objet complexe pour pouvoir le transmettre et/ou le stocker.

Fonctions de sérialisation et désérialisation

- `serialize($obj)` : retourne une chaîne de caractères contenant une représentation linéaire d'un objet `$obj` (ou de n'importe quelle autre valeur).
- `unserialize($ch)` : permet d'utiliser la chaîne de caractères `$ch` pour recréer l'objet à partir de sa représentation linéaire.

PHP

Exemple de sérialisation d'objet

```
$etudiant = new Etudiant(101, "maggio", "carol", 'master');
$ch = serialize($etudiant);
file_put_contents("file.txt", $ch);
```

PHP

Exemple de sérialisation d'objet

```
$etudiant = new Etudiant(101, "maggio", "carol", 'master');
$ch = serialize($etudiant);
file_put_contents("file.txt", $ch);
```


Aller voir le contenu de `file.txt`

```
4f3a 383a 2245 7475 6469 616e 7422 3a35
3a7b 733a 3137 3a22 0045 7475 6469 616e
7400 5f6e 6976 6561 7522 3b73 3a36 3a22
6d61 7374 6572 223b 733a 3134 3a22 0045
7475 6469 616e 7400 5f6e 756d 223b 4e3b
733a 3134 3a22 0050 6572 736f 6e6e 6500
5f6e 6f6d 223b 733a 363a 226d 6167 6769
6f22 3b73 3a31 373a 2200 5065 7273 6f6e
6e65 005f 7072 656e 6f6d 223b 733a 353a
2263 6172 6f6c 223b 733a 3134 3a22 0050
6572 736f 6e6e 6500 5f6e 756d 223b 693a
3130 303b 7d
```

PHP

Exemple de désérialisation

```php
$s = file_get_contents("file.txt");
$obj = unserialize($s);
echo $obj->getNom();
// affiche maggio
```

PHP

L'opérateur @

Il permet d'ignorer les messages d'erreurs pouvant être générés par une expression.

PHP

L'opérateur @

Il permet d'ignorer les messages d'erreurs pouvant être générés par une expression.

Par exemple, si on essaye de lire le contenu d'un fichier qui n'existe pas, on aura une erreur

```
$s = @file_get_contents("fichier_qui_n_existe_pas.txt");
```

PHP

L'opérateur @

Il permet d'ignorer les messages d'erreurs pouvant être générés par une expression.

Par exemple, si on essaye de lire le contenu d'un fichier qui n'existe pas, on aura une erreur

```
$s = @file_get_contents("fichier_qui_n_existe_pas.txt");
```

Pour éviter ces messages d'erreur, on utilise l'opérateur @

```
$s = @file_get_contents("fichier_qui_n_existe_pas.txt");
```

PHP

Remarques sur l'opérateur @

- L'opérateur @ ne fonctionne qu'avec les expressions qui retournent une valeur comme
  - variables
  - fonctions
  - include
  - constantes
- On ne peut l'utiliser avec des éléments de langage comme les classes, `if`, `foreach`...