

MISE EN PRATIQUE DE LA POO EN PHP 5/6

Auteur : Sébastien Inion

studi

AU PROGRAMME DE CE COURS...

Dans cette cinquième partie, nous verrons :

- ✱ Ce qu'est une **interface** et leurs rôles,
- ✱ La notion d'**héritage** entre interface
- ✱ Les interfaces prédéfinies en PHP

LES INTERFACES

- ▶ En POO un interface est une structure qui définit **un contrat** que les classes doivent suivre.
- ▶ Elle ne contient pas d'implémentation seulement **la signature de méthodes**.
- ▶ Les interfaces garantissent donc que les classes implémentent ces interfaces (méthodes) un peu comme le faisait déjà les classes mères mais sans cette notion d'héritage (est-un !)

LES INTERFACES

- ▶ Elles ne peuvent contenir que des déclarations de méthodes **pas de variables d'instance**.
- ▶ Les méthodes sont implicitement **abstraites et publiques**.
- ▶ Une classe peut implémenter **plusieurs interfaces**.

AVANTAGES DES INTERFACES

- ▶ **Standardisation** : en effet cela garantit que certaines méthodes existent dans les classes qui implémentent l'interface.
- ▶ **Polymorphisme** : cela permet de traiter des objets de différentes classe de manière uniforme si elle implémentent la même interface.

MISE EN APPLICATION...

ON UTILISE UNE INTERFACE PAYABLE

- ▶ Elle peut s'appliquer sur tout objet qui peut être vendu
 - ▶ Pour faire simple elle a trois méthodes
 - ▶ getPriceHT()
 - ▶ getPriceTTC()
 - ▶ getTVA()
- ▶ Nous appliquerons cette interface sur deux classes : Boisson et Vêtement

ON UTILISE UNE INTERFACE PAYABLE

```
<?php
// PayableInterface.php
interface Payable {
    public function getPriceHT(): float;
    public function getTVA(): float;
    public function getPriceTTC(): float;
}
?>
<?php
```

```
<?php

abstract class Article implements Payable
{
    protected $priceHT;
    protected $type;

    public function getPriceHT(): float {
        return $this->priceHT;
    }

    public function getPriceTTC(): float {
        return $this->priceHT + $this->getTVA();
    }

    public function getType(): string {
        return $this->type;
    }

    abstract public function getTVA(): float;
}

?>
```

<https://replit.com/@Sebastien11/cours5POO>

LES INTERFACES DÉJÀ PRÉSENTES...

Les interfaces prédéfinies

1. **Iterator**: Cette interface permet à un objet d'être itérable via une boucle foreach. Les classes qui implémentent Iterator doivent fournir des méthodes pour itérer sur leurs éléments.
2. **Countable**: Les classes qui implémentent Countable doivent fournir une méthode `count()` qui retourne le nombre d'éléments dans l'objet.
3. **ArrayAccess**: Cette interface fournit des méthodes pour accéder à un objet comme s'il était un tableau.
4. **Serializable**: Les classes qui implémentent Serializable doivent fournir des méthodes pour sérialiser et désérialiser l'objet.
5. **Traversable**: Une interface de marqueur pour indiquer qu'un objet peut être traversé avec foreach. Toutes les classes qui implémentent Iterator ou IteratorAggregate implémentent également cette interface implicitement.
6. **JsonSerializable**: Les classes qui implémentent JsonSerializable doivent fournir une méthode `jsonSerialize()` qui retourne les données de l'objet dans un format pouvant être sérialisé en JSON.
7. **Throwable**: Cette interface est implémentée par toutes les exceptions de PHP 7 et représente toutes les exceptions et erreurs.

Les interfaces prédéfinies

- Voyons plus en détail l'interface **Iterator**.
- Cette interface a 5 méthodes :
 - **current** // *renvoie l'élément courant*
 - **key** // *renvoie la clef de l'élément courant*
 - **next** // *passe à l'élément suivant*
 - **rewind** // *remet le pointeur sur le premier élément*
 - **valid** // *vérifie si la position courante est valide.*

<https://www.php.net/manual/en/class.iterator.php>

Les interfaces prédéfinies

```
class TestInterfaceIterator implements Iterator {
    private $array = array("Elise", "Bernard", "Sophie", "Fred");
    private $position = 0;

    #[\ReturnTypeWillChange]
    public function rewind(){
        $this->position = 0;
    }

    #[\ReturnTypeWillChange]
    public function current(){
        return $this->array[$this->position];
    }

    #[\ReturnTypeWillChange]
    public function key() {
        return $this->position;
    }

    #[\ReturnTypeWillChange]
    public function next(){
        ++$this->position;
    }

    #[\ReturnTypeWillChange]
    public function valid(){
        return isset($this->array[$this->position]);
    }
}
```



**MERCI POUR
VOTRE ATTENTION**