**INTERFACE**

Une interface en PHP est un type spécial de classe qui définit un contrat pour les classes qui l'implémentent. Elle **spécifie une liste de méthodes** **que ces classes doivent obligatoirement implémenter, sans en fournir le code**.   
  
Les interfaces servent principalement à :  
  
1. Assurer une certaine ***structure commune*** entre différentes classes, en forçant l'implémentation de méthodes spécifiques. Cela **facilite le polymorphisme\*1** *(permet à des classes différentes d être traitées de façon similaire)* **et le couplage faible** entre les classes *(peu ou pas de dépendance entre les classe)*.   
  
2. Définir des **contrats de programmation** que les classes doivent respecter, améliorant ainsi la maintenabilité et la modularité du code.   
  
3. Permettre l'**héritage multiple**, puisqu'*une classe peut implémenter plusieurs interfaces* à la fois, contrairement à l'héritage classique limité à une seule classe parente.   
  
4. **Séparer les interfaces des implémentations,** ce qui permet de modifier les implémentations sans affecter le code utilisant ces interfaces.  
  
Les interfaces définissent un contrat de programmation que les classes doivent respecter, favorisant ainsi la réutilisation, la modularité et la maintenabilité du code orienté objet en PHP.

## Principe des interfaces

Une interface définit un ensemble de méthodes sans en fournir l'implémentation.   
Les classes qui implémentent cette interface doivent alors **fournir leur propre implémentation** **concrète de ces méthodes**. Voici un exemple simple :

php

// Définition de l'interface FormeGeometrique

interface FormeGeometrique {

public function calculerAire();

public function calculerPerimetre();

}

// Classe Cercle implémentant l'interface

class Cercle implements FormeGeometrique {

private $rayon;

public function calculerAire() {

return M\_PI \* $this->rayon \*\* 2;

}

public function calculerPerimetre() {

return 2 \* M\_PI \* $this->rayon;

}

}

// Classe Rectangle implémentant l'interface

class Rectangle implements FormeGeometrique {

private $longueur;

private $largeur;

public function calculerAire() {

return $this->longueur \* $this->largeur;

}

public function calculerPerimetre() {

return 2 \* ($this->longueur + $this->largeur);

}

}

## Avantages du polymorphisme avec les interfaces

1. **Code générique** : On peut écrire du code qui fonctionne avec n'importe quelle classe implémentant une interface donnée, sans connaître les détails d'implémentation.

* **Couplage faible** : Les classes dépendent de l'interface et non des implémentations concrètes, réduisant les dépendances.
* **Extensibilité** : On peut facilement ajouter de nouvelles implémentations sans modifier le code existant, tant que la nouvelle classe respecte l'interface.
* **Testabilité** : Il est plus facile de tester les différents composants de manière isolée en utilisant des implémentations de test pour les interfaces

**LES DIFFERENCES AVEC CLASSE ABSTRACT**

une interface est différente d'une classe abstraite , même si elles partagent certaines similitudes :

* **Une interface ne peut contenir que des méthodes abstraites** (sans implémentation), tandis qu'une classe abstraite peut contenir à la fois des méthodes abstraites et des méthodes implémentées.
* **Toutes les méthodes d'une interface doivent être publiques**, alors que les méthodes d'une classe abstraite peuvent avoir différents niveaux de visibilité (public, protected, private).
* **Une classe peut implémenter plusieurs interfaces,** mais ne peut hériter que d'une seule classe abstraite.
* **Les interfaces ne peuvent pas avoir de propriétés**, contrairement aux classes abstraites qui le peuvent.
* **Une interface ne peut pas être instanciée**, elle définit uniquement un contrat que les classes qui l'implémentent doivent respecter. Une classe abstraite ne peut pas non plus être instanciée directement, mais **elle peut contenir une implémentation partielle**.
* Les interfaces favorisent la **composition et le couplage faible** entre les classes, tandis que l'héritage de classes abstraites **promeut une hiérarchie de classes plus rigide**.

**Les interfaces définissent un contrat purement abstrait que les classes doivent implémenter, tandis que les classes abstraites fournissent une implémentation partielle pouvant être étendue. Les interfaces encouragent la modularité et la réutilisabilité du code, alors que les classes abstraites facilitent l'héritage et le partage de fonctionnalités communes.**

**Concretement, une class peut implémenter plusieurs interface mais elle doit respecter « le contrat » avec chacune d elles (on transmet le même comportement).**

**\*1**

**PRINCIPE DE POLYMORPHISME :**

**Voici un exemple simple de polymorphisme avec l'héritage en PHP :**

php

class Animal {

public function sound() {

echo "L'animal fait un son...";

}

}

class Dog extends Animal {

public function sound() {

echo "Le chien aboie !";

}

}

class Cat extends Animal {

public function sound() {

echo "Le chat miaule !";

}

}

$animal = new Animal();

$chien = new Dog();

$chat = new Cat();

$animal->sound(); // Affiche "L'animal fait un son..."

$chien->sound(); // Affiche "Le chien aboie !"

**$chat->sound(); // Affiche "Le chat miaule !"**