Classes abstraites, interfaces et traits

Les classes abstraites

Les classes abstraites sont des classes qui ne peuvent pas être instanciées.

Les classes abstraites peuvent inclure des méthodes abstraites qui permettent de déclarer une signature de fonction sans mettre en place l'implémentation.

Les classes héritant de la classe abstraite doivent définir toutes les méthodes abstraites et les implémenter.

Les classes abstraites **servent à partager des fonctionnalités communes entre plusieurs classes enfants et / ou à définir un ensemble de règles d'implémentation**.

Exemple :

```
<?php
abstract class User {
  public function __construct(public string $nom) {}

  abstract public function greeting() : string;
}

class Admin extends User {
  public function greeting() : string {
    return "Je suis un admin et je m'appelle $this->nom !";
  }
}

class Moderator extends User {
  public function greeting() : string {
    return "Je suis un modérateur et je m'appelle $this->nom !";
  }
}

class Registered extends User {
  public function greeting() : string {
    return "Je suis un utilisateur et je m'appelle $this->nom !";
  }
}
```

```
$admin = new Admin("Jean"); // Je suis un admin et je m'appelle Jean !
echo $admin->greeting();
echo "<br>";

$modo = new Moderator("Paul"); // Je suis un modérateur et je m'appelle
Paul !
echo $modo->greeting();
echo "<br>";

$user = new Registered("Bob"); // Je suis un utilisateur et je m'appelle
Bob !
echo $user->greeting();
```

Les interfaces

Les interfaces permettent de définir quelles méthodes une classe doit implémenter, sans avoir à définir leur implémentation.

Toutes les méthodes de l'interface doivent être implémentées dans une classe qui implémente l'interface.

Une interface est un contrat pour une classe. Elle permet d'obliger à ce qu'une ou plusieurs classes respectent un modèle.

Contrairement à une classe abstraite, une interface n'implémente aucune logique.

Par exemple :

```
<?php
interface Database
{
  function createOrder();
  function getOrder();
  function updateOrder();
  function removeOrder();
  function listOrders();
}

class SQLDatabase implements Database
{
  function createOrder()
```

```
    {
      // Implémentation
    }
    function getOrder()
    {
      // Implémentation
    }
    function updateOrder()
    {
      // Implémentation
    }
    function removeOrder()
    {
      // Implémentation
    }
    function listOrders()
    {
      // Implémentation
    }
}
```

Les traits

Les traits permettent de réutiliser du code dans plusieurs classes.

Un trait est un ensemble de fonctionnalités qui peut être facilement partagé entre plusieurs classes.

Pour utiliser un trait dans une classe il suffit de faire `use nomDuTrait` .

Il est possible d'utiliser plusieurs traits dans la même classe.

Vous pouvez voir un trait comme un moyen d'éviter la duplication. Dès que vous avez du code dupliqué entre différentes classes, et que ces classes n'ont pas de relation d'héritage, c'est là qu'il faut utiliser un trait !

```
<?php
trait Trait1
{
    function faireUnCafé()
    {
    }
```

```
  function faireLaVaisselle()
  {
  }
}

trait Trait2
{
  public $compteur2 = 0;

  function compter()
  {
    $this->compteur2++;
    echo $this->compteur2, '<br>';
  }
}

class User
{
  use Trait1, Trait2;
}

class Admin
{
  use Trait2;
}

$user = new User();
$admin = new Admin();

$user->compter(); // 1
$user->compter(); // 2
$admin->compter(); // 1
```