**LE ‘TRAIT’**

TRAIT ET INTERFACE QUELLES DIFFERENCES :

le trait est utilisé quand il n y a pas de relation ‘extends’ besoin d une méthode indépendante de l héritage.

Le trait est différent de l interface qui elle, oblige les classes à avoir une méthode (quoi) dont elles doivent définir ce qu elle fait (le comment).   
Dans le cas de l’interface, ce sont des méthodes au principe commun, mais qui varient selon la classe.

Ce sont des variantes de méthode (elles se ressemblent mais ne sont pas totalement identiques).

Dans le cas du trait, c’est une méthode identique qui est utilisé dans des classes différentes.  
L’avantage, c’est d avoir une méthode commune à plusieurs classes, qui est écrite une seul fois à un seul endroit et totalement indépendante de l’héritage .  
  
  
// Définition d'un trait

trait **LoggerTrait**

{

public function **log**($message)

{

echo $message . "\n";

}

}

// Utilisation du trait dans une classe

class Utilisateur //ceci est un trait

{

**use** **LoggerTrait**;

public function \_\_construct($name)

{

$this->**log**("Nouvel utilisateur créé : " . $name);

}

}

// Utilisation du trait dans une autre classe

class Commande

{

**use** **LoggerTrait**;//c’est un use qui est différent du use du namespace qui dit ou se trouve la class

public function \_\_construct($produit)

{

$this->**log**("Nouvelle commande pour : " . $produit);

}

}

$user = new Utilisateur("John Doe");

$order = new Commande("Livre");

Le trait LoggerTrait définit une méthode log().   
Les classes Utilisateur et Commande utilisent ce trait pour bénéficier de la méthode log() sans avoir à la redéfinir.   
Les traits permettent **une meilleure modularité et une réutilisation de code plus efficace** dans les applications PHP. Ils offrent une **alternative intéressante à l'héritage** classique et facilitent la composition de fonctionnalités dans les classes.