**EARLY BINDING**

Le early binding (ou liaison précoce) en PHP fait référence à la capacité d'utiliser une classe avant sa déclaration dans le même fichier, sous certaines conditions. On se trouve dans un contexte de early binding lorsque les conditions suivantes sont remplies :

1. La classe est utilisée (instanciée, accès à ses propriétés/méthodes statiques) avant sa déclaration dans le même fichier PHP.
2. La classe n'implémente pas d'interfaces et n'utilise pas de traits.
3. Toutes les dépendances de la classe (classes parentes, classes utilisées dans les signatures de méthodes, etc.) sont disponibles au moment de l'utilisation de la classe.

Si ces conditions sont remplies, le moteur PHP tentera de déclarer la classe à ce point d'utilisation précoce plutôt qu'à son point de déclaration d'origine dans le fichier. **Cela permet d'utiliser la classe avant même qu'elle ne soit déclarée**.

Cependant, si l'une de ces conditions n'est pas remplie (par exemple, si la classe implémente une interface ou utilise un trait), *le moteur PHP ne fera pas de early binding et déclarera la classe à son point de déclaration d'origine*.

Le early binding est une fonctionnalité assez complexe de PHP, avec des règles et des limitations spécifiques, notamment lorsque l'opcache est impliqué. Son comportement peut différer légèrement avec ou sans opcache activé.

**LATE STATIC BINDING**

Le late static binding (résolution statique à la volée) est une fonctionnalité qui permet de résoudre la portée des appels statiques (propriétés et méthodes) au moment de l'exécution plutôt qu'au moment de la définition de la classe.  
  
Avec le late static binding, lorsque vous utilisez le mot-clé `static` à l'intérieur d'une méthode statique ou non-statique, PHP résoudra la portée de l'appel en fonction de la classe réelle utilisée pendant l'exécution, plutôt que de la classe dans laquelle la méthode est définie.  
  
Voici comment cela fonctionne :  
  
1. Lorsque vous utilisez `**self::`, PHP résout la portée en fonction de la classe dans laquelle la propriété/méthode est définie**.  
  
2. Lorsque vous utilisez `**static::`, PHP résout la portée en fonction de la classe réelle depuis laquelle la propriété/méthode est appelée au moment de l'exécution**.  
  
Cela permet d'implémenter le polymorphisme pour les méthodes et propriétés statiques dans les classes enfants. Prenons un exemple :  
  
```php  
class Mere  
{  
 **protected static $data = "Données de la classe Mere";**  
   
 public static function getData()  
 {  
 return static::$data; // Résolution statique à la volée  
 }  
}  
  
class Fille extends Mere  
{  
  **protected static $data = "Données de la classe Fille";**  
}  
  
echo Mere::getData(); // Affiche "Données de la classe Mere"  
echo Fille::getData(); // Affiche "Données de la classe Fille"  
```

**la classe fille va se servir de la méthode de la classe parent mais dans son propre contexte de classe et non celui du parent. C’est pour cela que c’est SA variable $data qui apparait**

**Grâce au static::$data à l'intérieur de getData(), PHP résout la portée de $data en fonction de la classe réelle depuis laquelle la méthode est appelée au moment de l'exécution.**

**IL PEUT ETRE UTILISER DANS UN ENVIRONNENT STATIC COMME DANS  
 L’EXEMPLE OU OBJET**

Dans cet exemple, `static::$data` dans la méthode `getData()` fait référence à la propriété `$data` de la classe réelle depuis laquelle la méthode est appelée (`Mere` ou `Fille`), plutôt que de toujours faire référence à la propriété de la classe `Mere` où la méthode est définie.  
  
Le late static binding permet donc une meilleure réutilisation du code et une meilleure implémentation du polymorphisme dans les hiérarchies de classes en PHP. Cependant, il est important de l'utiliser avec précaution, car **il peut rendre le code plus difficile à comprendre et à maintenir si mal utilisé**.