
Fiche Mémo : Le Singleton avec PDO en PHP

Le **design pattern Singleton** permet de s'assurer qu'une classe n'aura qu'une seule instance dans toute l'application. C'est particulièrement utile pour une classe de connexion à la base de données afin d'éviter des connexions multiples qui consommeraient des ressources inutilement.

Cette fiche montre comment implémenter un singleton pour gérer **une seule instance de connexion PDO** en PHP.

Principe du Singleton

1. La classe `Database` n'est **instanciée qu'une seule fois**.
2. Elle gère et contrôle l'accès à une unique instance de **PDO**.
3. L'accès à l'instance se fait via une méthode statique.

Étapes pour Créer un Singleton avec PDO

1. Déclarer une Propriété Statique pour l'Instance Unique

```php
private static ?Database $instance = null;
private ?PDO $connection = null;
```

- `$instance` est une propriété statique pour stocker l'unique instance de `Database`.
- `$connection` contient l'instance de PDO, mais elle reste privée à la classe `Database`.

2. Rendre le Constructeur Privé

Le constructeur est défini comme privé pour empêcher toute création d'instances en dehors de la classe.

```php
private function __construct() {
    try {
        $this->connection = new
        PDO('mysql:host=localhost;dbname=test', 'root',
        'password');
        $this->connection->setAttribute(PDO::ATTR_ERRMODE,
        PDO::ERRMODE_EXCEPTION);
    } catch (PDOException $e) {
        die("Erreur de connexion : " . $e->getMessage());
    }
}
```

3. Créer une Méthode Statique `getInstance()`

Cette méthode statique initialise `Database` si elle ne l'est pas déjà, garantissant ainsi qu'il n'y a qu'une seule instance pour toute l'application.

```
public static function getInstance(): Database {
    if (self::$instance === null) {
        self::$instance = new Database();
    }
    return self::$instance;
}
```

4. Accéder à la Connexion PDO

Une méthode publique `getConnection()` permet d'accéder à l'instance PDO, utilisée pour exécuter des requêtes.

```
public function getConnection(): PDO {
    return $this->connection;
}
```

5. Empêcher le Clonage

On rend la méthode `__clone` privée pour éviter de dupliquer l'instance.

```
private function __clone() {}
```

Exemple Complet : Classe de Connexion à la Base de Données (Singleton)

Voici la classe `Database` complète, qui encapsule une connexion PDO en utilisant le pattern Singleton :

```
class Database {
    private static ?Database $instance =
        null; // Instance unique de Database
    private ?PDO $connection = null;           // Instance PDO

    // Constructeur privé pour initialiser PDO une seule fois
    private function __construct() {
        try {
            $this->connection = new
        PDO('mysql:host=localhost;dbname=test', 'root',
        'password');
            $this->connection->setAttribute(PDO::ATTR_ERRMODE,
        PDO::ERRMODE_EXCEPTION);
```

```
        } catch (PDOException $e) {
            die("Erreur de connexion : " . $e->getMessage());
        }
    }

    // Méthode statique pour obtenir l'instance unique de
        Database
    public static function getInstance(): Database {
        if (self::$instance === null) {
            self::$instance = new Database();
        }
        return self::$instance;
    }

    // Méthode pour récupérer la connexion PDO
    public function getConnection(): PDO {
        return $this->connection;
    }

    // Empêche le clonage de l'instance
    private function __clone() {}
}
```

Utilisation du Singleton pour la Connexion

Pour utiliser `Database` et accéder à la connexion PDO, vous pouvez appeler la méthode statique `getInstance()` :

```
// Récupère l'instance unique de Database
$dbInstance = Database::getInstance();
$connection = $dbInstance->getConnection(); // Accède à
        l'instance PDO

// Vérifie que la connexion est unique
$dbInstance2 = Database::getInstance();
var_dump($dbInstance === $dbInstance2); // Affiche : bool(true)
```

Pourquoi ne pas faire Hériter Database de PDO ?

1. **Séparation des responsabilités** : `Database` contrôle l'accès et la gestion de l'instance PDO, tandis que PDO gère les requêtes SQL.
2. **Simplicité de modification** : si les besoins en connexion évoluent (ajout de configurations, changement de moteur de base de données), il suffit de modifier `Database` sans toucher aux méthodes internes de PDO.

En suivant ce modèle, vous obtiendrez une **instance unique de PDO** centralisée par une classe singleton `Database`, sans complexité d’héritage.