DÉVELOPPEMENT BACKEND

PHP orienté objet : concevoir des applications structurées

Maîtriser la programmation orientée objet en PHP 8. Classes, héritage, interfaces, traits, design patterns et architecture MVC : écrivez un code PHP professionnel, maintenable et réutilisable.

DURÉE	TARIF HT	NIVEAU	LANGUE	GROUPE	FORMAT
3j (21h)	~~1200.00 €~~ **720.00 €**	**Intermédiaire**	**Français**	**3-12**	**Formation**

1 PUBLIC VISÉ

- Développeurs PHP connaissant les bases du langage (variables, fonctions, tableaux, PDO) souhaitant structurer leur code de manière professionnelle.
- Personnes ayant suivi la formation PHP moderne : bases du développement côté serveur ou disposant d'une expérience équivalente.
- Développeurs autodidactes voulant abandonner le code procédural et adopter une architecture orientée objet maintenable.

2 PRÉREQUIS

- Bonne maîtrise des bases de PHP : variables, fonctions, tableaux, formulaires, PDO.
- La formation PHP moderne : bases du développement côté serveur est requise ou une expérience équivalente.
- Savoir utiliser le terminal et VS Code.

3 OBJECTIFS PÉDAGOGIQUES

- Comprendre les principes fondamentaux de la programmation orientée objet et leur application en PHP 8
- Concevoir et instancier des classes avec propriétés, constructeur et méthodes
- Maîtriser l'héritage, le polymorphisme et les classes abstraites
- Utiliser les interfaces et les traits pour composer des comportements réutilisables
- Appliquer les principes SOLID pour écrire un code maintenable et extensible
- Mettre en place une architecture MVC sans framework
- Utiliser Composer et gérer les dépendances d'un projet PHP
- Appliquer les design patterns les plus courants en PHP

4 PROGRAMME DÉTAILLÉ

Jour 1 (7h) – Les fondamentaux de la POO en PHP 8

Module 1 – Classes et objets (3h)

Pourquoi la POO ? (30min)

- Les limites du code procédural : duplication, couplage, maintenabilité
- Les quatre piliers de la POO : encapsulation, abstraction, héritage, polymorphisme
- La POO en PHP dans l'écosystème : Symfony, Laravel, tous écrits en POO

Déclarer et instancier une classe (1h30)

- La syntaxe d'une classe : propriétés, constructeur, méthodes
- Instancier un objet avec new et accéder à ses membres avec $this
- Visibilité : public, protected, private — pourquoi encapsuler
- Les propriétés et méthodes statiques : accès sans instanciation
- Les constantes de classe avec const
- Les nouveautés PHP 8 : constructor property promotion, readonly properties
- Cas pratique : modéliser une classe Produit avec prix, stock et méthodes métier

Getters, setters et magic methods (1h)

- Pourquoi ne pas exposer les propriétés directement
- Écrire des getters et setters avec validation interne
- Les magic methods : __construct, __toString, __get, __set, __isset
- Cas pratique : classe Utilisateur avec validation de l'email dans le setter

Module 2 – Héritage et polymorphisme (2h)

Héritage (1h)

- Étendre une classe avec extends : hériter propriétés et méthodes
- Appeler le constructeur parent avec parent::__construct()
- Surcharger une méthode dans une classe enfant
- Le mot-clé final : interdire l'héritage ou la surcharge
- Cas pratique : hiérarchie de classes pour des types de contenu (Article, Vidéo, Podcast)

Polymorphisme (1h)

- Même interface, comportements différents selon la classe concrète
- Le typage des paramètres avec les classes : injecter un objet dans une méthode
- instanceof : vérifier le type d'un objet à l'exécution
- Cas pratique : système de paiement avec comportement différent selon le moyen de paiement

Module 3 – Classes abstraites et interfaces (2h)

Classes abstraites (1h)

- Déclarer une classe abstraite avec abstract
- Méthodes abstraites : définir un contrat sans implémentation
- Quand utiliser une classe abstraite : factoriser du code commun tout en forçant une implémentation
- Cas pratique : classe abstraite Notification avec des sous-classes Email, SMS, Push

Interfaces (1h)

Déclarer une interface avec interface

- Implémenter une interface avec implements
- Implémenter plusieurs interfaces : la solution au manque d'héritage multiple
- Interface vs classe abstraite : quand choisir l'une ou l'autre
- Les interfaces natives PHP : Countable, Iterator, Stringable
- Cas pratique : interface Exportable implémentée par plusieurs entités

Jour 2 (7h) – Concepts avancés et architecture

Module 4 – Traits et namespaces (2h)

Traits (1h)

- Le problème que les traits résolvent : réutiliser du code sans héritage
- Déclarer et utiliser un trait avec use
- Résoudre les conflits entre traits : insteadof et as
- Traits avec propriétés abstraites : forcer l'implémentation
- Cas pratique : traits Timestampable, SoftDeletable, Sluggable pour les entités

Namespaces et autoloading (1h)

- Pourquoi les namespaces : éviter les conflits de noms entre classes
- Déclarer un namespace et importer une classe avec use
- La convention PSR-4 : correspondance dossiers et namespaces
- Configurer l'autoloading PSR-4 avec Composer
- Cas pratique : organiser un projet en namespaces AppModel, AppService, AppController

Module 5 – Principes SOLID (2h)

Les cinq principes SOLID (2h)

- S — Single Responsibility : une classe, une responsabilité
- O — Open/Closed : ouvert à l'extension, fermé à la modification
- L — Liskov Substitution : une sous-classe doit pouvoir remplacer sa classe parente
- I — Interface Segregation : des interfaces spécifiques plutôt qu'une interface générale
- D — Dependency Inversion : dépendre des abstractions, pas des implémentations
- Cas pratique : identifier les violations SOLID dans du code existant et les corriger

Module 6 – Design patterns en PHP (3h)

Patterns de création (1h)

- Singleton : garantir une instance unique — le cas de la connexion PDO
- Factory : déléguer la création d'objets selon un contexte
- Builder : construire des objets complexes étape par étape
- Cas pratique : factory de connexions à différentes bases de données

Patterns de comportement (1h)

- Observer : réagir à des événements sans couplage fort
- Strategy : choisir un algorithme à l'exécution
- Repository : abstraire l'accès aux données derrière une interface
- Cas pratique : système d'événements découplé avec Observer

Patterns de structure (1h)

Decorator : enrichir un objet sans modifier sa classe

- Adapter : faire communiquer deux interfaces incompatibles
- Dependency Injection : injecter les dépendances plutôt que les instancier
- Cas pratique : système de cache transparent avec Decorator

Jour 3 (7h) – Architecture MVC et projet de synthèse

Module 7 – Architecture MVC sans framework (3h)

Comprendre le pattern MVC (1h)

- Model, View, Controller : rôle et responsabilité de chaque couche
- Le flux d'une requête dans une architecture MVC
- Pourquoi MVC : séparation des responsabilités, testabilité, maintenabilité
- MVC en PHP avant les frameworks : comprendre ce que Symfony et Laravel automatisent

Implémenter un mini-framework MVC (2h)

- Le point d'entrée unique : index.php et le front controller
- Le routeur : associer une URL à un contrôleur et une action
- Les contrôleurs : recevoir la requête et orchestrer la réponse
- Les modèles : encapsuler l'accès aux données avec PDO
- Les vues : séparer le HTML de la logique PHP
- L'injection de dépendances : passer le modèle au contrôleur
- Cas pratique : application CRUD complète en architecture MVC

Module 8 – Projet de synthèse (4h)

Réalisation d'une application PHP orientée objet complète

- Cahier des charges : application de gestion de bibliothèque ou de boutique en ligne
- Fonctionnalités : architecture MVC avec routeur, entités modélisées en classes, repository pattern pour l'accès aux données PDO, authentification avec sessions, traits pour les comportements partagés, au moins deux design patterns appliqués
- Étape 1 : conception des classes et de la hiérarchie
- Étape 2 : mise en place de l'architecture MVC et du routeur
- Étape 3 : implémentation des fonctionnalités métier
- Étape 4 : refactoring avec les principes SOLID
- Revue de code collective : architecture, respect des principes, lisibilité
- Retour formateur individualisé sur le projet rendu

5 COMPÉTENCES VISÉES

- Concevoir une hiérarchie de classes adaptée à un problème métier réel
- Écrire du code PHP orienté objet respectant les principes SOLID
- Structurer une application PHP selon le pattern MVC sans framework
- Utiliser Composer pour gérer l'autoloading et les dépendances
- Identifier et appliquer les design patterns adaptés selon le contexte
- Refactoriser du code procédural existant vers une architecture orientée objet

6 MODALITÉS PÉDAGOGIQUES

Formation délivrée en présentiel ou distanciel (visioconférence). Le formateur alterne entre méthode démonstrative (live coding commenté et progressif), méthode interrogative (analyse critique de code existant et identification des violations de principes) et méthode active (exercices de conception et de refactoring). L'accent est mis sur la compréhension des mécanismes de la POO et des raisons derrière chaque principe plutôt que sur la mémorisation de la syntaxe.

7 MOYENS PÉDAGOGIQUES ET TECHNIQUES

- Support de cours numérique mis à disposition des apprenants
- Dépôt GitHub de démonstration avec exercices et corrections par module
- Environnement de développement : VS Code + XAMPP ou Laragon + Composer
- Pour le distanciel : visioconférence (Zoom ou équivalent), partage d'écran, chat en direct
- Accès à la plateforme pédagogique LaPolaris (supports, ressources, émargement)

8 MODALITÉS D'ÉVALUATION

- En cours de formation : exercices de conception de classes et de refactoring corrigés à chaque module
- En fin de formation : réalisation d'une application PHP orientée objet complète en architecture MVC
- Questionnaire d'auto-évaluation des acquis en fin de parcours

9 CRITÈRES D'ÉVALUATION

- Conception correcte d'une hiérarchie de classes adaptée au problème métier
- Respect des principes SOLID identifiable dans le code produit
- Application pertinente d'au moins deux design patterns dans le projet final
- Architecture MVC fonctionnelle avec séparation claire des responsabilités
- Qualité et lisibilité du code : nommage, organisation, absence de duplication

10 MODALITÉS DE VALIDATION

Attestation de fin de formation délivrée à l'issue du parcours, conditionnée à une assiduité d'au moins 80 % et à la réalisation du projet de synthèse. L'attestation précise les objectifs atteints et les compétences acquises.

11 SUIVI ET ACCOMPAGNEMENT

- Feuilles d'émargement signées par demi-journée (présentiel) ou émargement numérique (distanciel)
- Traçabilité des activités pédagogiques réalisées
- Attestation d'assiduité délivrée en fin de formation
- Suivi individuel via les exercices de refactoring et le projet de synthèse

12 CONDITIONS D'ACCÈS

Formation accessible sur inscription directe, sans prérequis administratif particulier. Le financement peut être pris en charge par l'employeur dans le cadre d'un plan de développement des compétences, ou en autofinancement avec possibilité de paiement en plusieurs fois.

13 DÉLAIS D'ACCÈS

Inscription possible jusqu'à **5 jours ouvrés** avant le début de la session. Pour toute demande urgente, nous contacter directement.

S PROCHAINES SESSIONS

27/05/2026 → 29/05/2026
À distance
12 places restantes

01/06/2026 → 03/06/2026
À distance
12 places restantes

03/06/2026 → 05/06/2026
À distance
12 places restantes

08/06/2026 → 10/06/2026
À distance
12 places restantes

10/06/2026 → 12/06/2026
À distance
12 places restantes

15/06/2026 → 17/06/2026
À distance
12 places restantes

ACCESSIBILITÉ · HANDICAP

Nos formations sont accessibles aux personnes en situation de handicap. Pour toute situation nécessitant un aménagement (matériel, temporel ou pédagogique), nous vous invitons à nous contacter avant l'inscription afin d'étudier les adaptations possibles. Référent handicap : contact@lapolaris.fr

LaPolaris
TÉL. +33762584798
EMAIL contact@lapolaris.fr
WEB lapolaris.fr