



# Introduction à la Programmation Orientée Objet

La programmation orientée objet (POO) est un **paradigme de programmation** qui permet de modéliser des systèmes en s'inspirant du monde réel. Elle repose sur la notion d'objets : des entités qui possèdent des **caractéristiques** (données) et des **comportements** (méthodes).

La POO propose une approche plus modulaire que l'impératif, orientée vers l'**organisation logique** du code.

## Pourquoi utiliser la POO ?

- Facilite la **réutilisation** de code à travers des classes et de l'héritage ;
- Permet une meilleure **lisibilité** grâce à une structure plus naturelle ;
- Offre une **évolutivité** accrue via l'extension contrôlée des fonctionnalités ;
- Assure une meilleure **maintenance**, en réduisant les dépendances excessives.

Elle permet également de mettre en œuvre des **bonnes pratiques de conception logicielle**, essentielles pour créer du code propre, clair, évolutif et maintenable.

## Des principes qui guident la conception

Le paradigme de la programmation orientée objet n'est pas seulement une technique : c'est aussi une **philosophie**, guidée par des principes de conception bien établis.

### Les principes SOLID

Ces 5 règles fondamentales favorisent un code modulaire et flexible :

- **S – Single Responsibility Principle** : Une classe ne doit avoir **qu'un seul rôle**. Par exemple, une classe qui gère les utilisateurs ne devrait pas aussi écrire dans des fichiers.
- **O – Open/Closed Principle** : Le code doit pouvoir être **étendu** sans avoir à être **modifié**. Ainsi, on ajoute de nouvelles fonctionnalités sans casser l'existant.
- **L – Liskov Substitution Principle** : Une **classe fille doit pouvoir remplacer sa classe mère** sans altérer le comportement du programme. Cela garantit que les héritages restent cohérents.
- **I – Interface Segregation Principle** : Il vaut mieux **plusieurs petites interfaces** qu'une grosse avec des méthodes inutiles. Une classe ne devrait jamais être forcée d'implémenter ce qu'elle n'utilise pas.
- **D – Dependency Inversion Principle** : On **dépend des abstractions**, pas des détails. Cela réduit l'impact des changements en isolant les composants.

## D'autres principes essentiels

- **KISS (Keep It Simple, Stupid)** : La **simplicité est une force**. Un code simple est plus facile à comprendre, à maintenir et à faire évoluer. Il faut éviter les solutions inutiles ou trop complexes.
- **DRY (Don't Repeat Yourself)** : Ne pas répéter du **code** identique. On privilégie la réutilisation par des fonctions, classes ou modules communs, ce qui limite les erreurs et facilite les modifications.
- **YAGNI (You Aren't Gonna Need It)** : N'ajoutez que ce qui est nécessaire. Il est inutile d'anticiper des fonctionnalités hypothétiques : cela ajoute de la complexité inutile et rend le code plus fragile.
- **La Loi de Demeter** : Une classe ne devrait parler qu'à ses proches collaborateurs, et non à toute la chaîne d'objets. Cela diminue le couplage entre les composants et renforce la robustesse du système.

## Ce que vous allez apprendre dans ce cours

Dans les prochains chapitres, vous découvrirez les fondements de la POO :

- **Classes et objets** : la base de tout système orienté objet ;
- **Attributs et méthodes** : les données internes et le comportement ;
- **Encapsulation** : comment protéger les données sensibles ;
- **Héritage et polymorphisme** : comment spécialiser ou généraliser des comportements ;
- **Gestion des exceptions** : anticiper et gérer les erreurs avec élégance ;

Ces notions seront accompagnées de cas concrets, d'exercices et d'illustrations pratiques. Elles vous permettront de concevoir des programmes robustes, lisibles et facilement maintenables, en appliquant les principes de qualité logicielle évoqués ici.