Rapport-TP-GL-4

Messaoud Fares Yacine

November 2025

Conclusion sur les principes de conception logicielle

Dans ce projet, nous avons mis en œuvre trois design patterns (**Builder**, **Observer**, **Decorator**) pour améliorer la gestion des cours. L'analyse de notre code par rapport aux principes de conception logicielle montre les points suivants:

Principe de Responsabilité Unique (SRP)

Chaque classe a une responsabilité bien définie:

- `Cours` contient uniquement les informations d'un cours.
- `CoursBuilder` s'occupe uniquement de la construction fluide des cours.
- `GestionnaireEmploiDuTemps` gère uniquement la notification des changements.
- `Etudiant` et `Responsable` reçoivent uniquement les notifications.
- Les décorateurs (`CoursEnLigne`, etc.) ajoutent une responsabilité spécifique sans modifier la classe de base.

SRP est globalement respecté.

Principe Ouvert/Fermé (OCP)

Les classes sont ouvertes à l'extension mais fermées à la modification:

- Les décorateurs permettent d'ajouter dynamiquement de nouvelles caractéristiques aux cours.
- Le Builder permet de créer de nouvelles instances de cours sans modifier `Cours`.

OCP est respecté.

Principe de Substitution de Liskov (LSP)

Tous les objets `ICours` peuvent être remplacés par leurs décorateurs sans altérer le comportement attendu. **LSP est respecté.**

Principe de Ségrégation des Interfaces (ISP)

Les interfaces (`Observer`, `Subject`, `ICours`) sont fines et spécifiques. Aucun client n'est contraint d'implémenter des méthodes inutiles. **ISP est respecté.**

Principe d'Inversion des Dépendances (DIP)

Les classes dépendent d'abstractions plutôt que de classes concrètes:

- `GestionnaireEmploiDuTemps` dépend de l'interface `Observer`.
- Les décorateurs dépendent de l'interface `ICours`.

DIP est respecté.

Conclusion générale

Le projet respecte globalement les principes de conception logicielle: chaque classe est claire et cohérente, les responsabilités sont séparées, et le code est extensible sans être modifié. L'usage combiné des patterns **Builder**, **Observer** et **Decorator** assure un code robuste, flexible et facile à maintenir, tout en respectant les bonnes pratiques de conception.