République Algérienne Démocratique et Populaire

Ministère de la Défense Nationale

École Militaire Polytechnique

Département Génie Informatique

TP2 : Gestion de l'Emploi du Temps

Àuteurs :
EOC. Mehalaine Souha Djihane

Chargé de module :
Cdt. Mazari Abdessameud Oussama

Année Universitaire
2025/2026

Diagramme de classe et respect des principes de conception

1. Diagramme de classes

Le diagramme de classes ci-dessous illustre la structure finale de ma solution, intégrant les patterns Builder, Decorator et Observer :

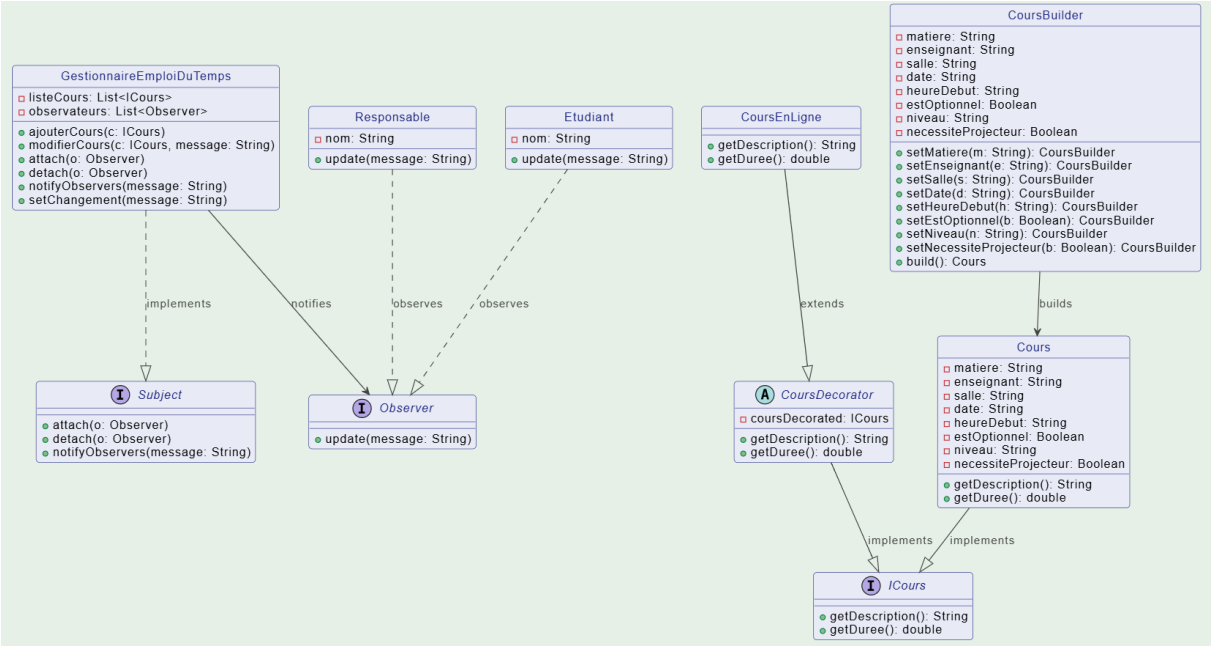

FIGURE 1 – Diagramme de classe

2. Respect des principes de conception logicielle

L'implémentation actuelle respecte globalement les principes SOLID :

— **Single Responsibility Principle (SRP) :** Chaque classe a une responsabilité unique. Par exemple, `Cours` représente un cours, `GestionnaireEmploiDuTemps` gère les cours et les notifications, et `Etudiant`/`Responsable` reçoivent les notifications.

— **Open/Closed Principle (OCP) :** Les cours sont extensibles via les décorateurs (`CoursDecorator`, `CoursEnLigne`) sans modifier la classe de base.

— **Liskov Substitution Principle (LSP) :** Les objets `Cours` et `CoursDecorator` peuvent être utilisés de manière interchangeable partout où `ICours` est attendu.

— **Interface Segregation Principle (ISP) :** Les interfaces `ICours`, `Observer` et `Subject` sont cohérentes et n'imposent pas de méthodes inutiles aux classes implémentant ces interfaces.

— **Dependency Inversion Principle (DIP) :** `GestionnaireEmploiDuTemps` dépend des abstractions (`Observer`) et non des implémentations concrètes (`Etudiant`

ou `Responsable`).

Points d'amélioration

— La méthode `setChangement` pourrait violer le SRP si elle accumule trop de logique métier au lieu de se limiter à la notification.
— Le `CoursBuilder` ne valide pas les données fournies (par exemple, la date ou l'heure du cours), ce qui pourrait entraîner des incohérences.

L'architecture du projet est conforme aux bonnes pratiques de conception, en particulier aux principes SOLID, tout en exploitant efficacement les patterns Builder, Decorator et Observer.