Bonus TP 2 Génie Logiciel : Diagramme de Classe et Analyse des Principes de Conception

Rezak Abderaouf RSD

Novembre 2025

1 Diagramme de Classe UML

Le diagramme de classe suivant décrit le résultat final du TP, intégrant les patterns Builder, Observer et Decorator.

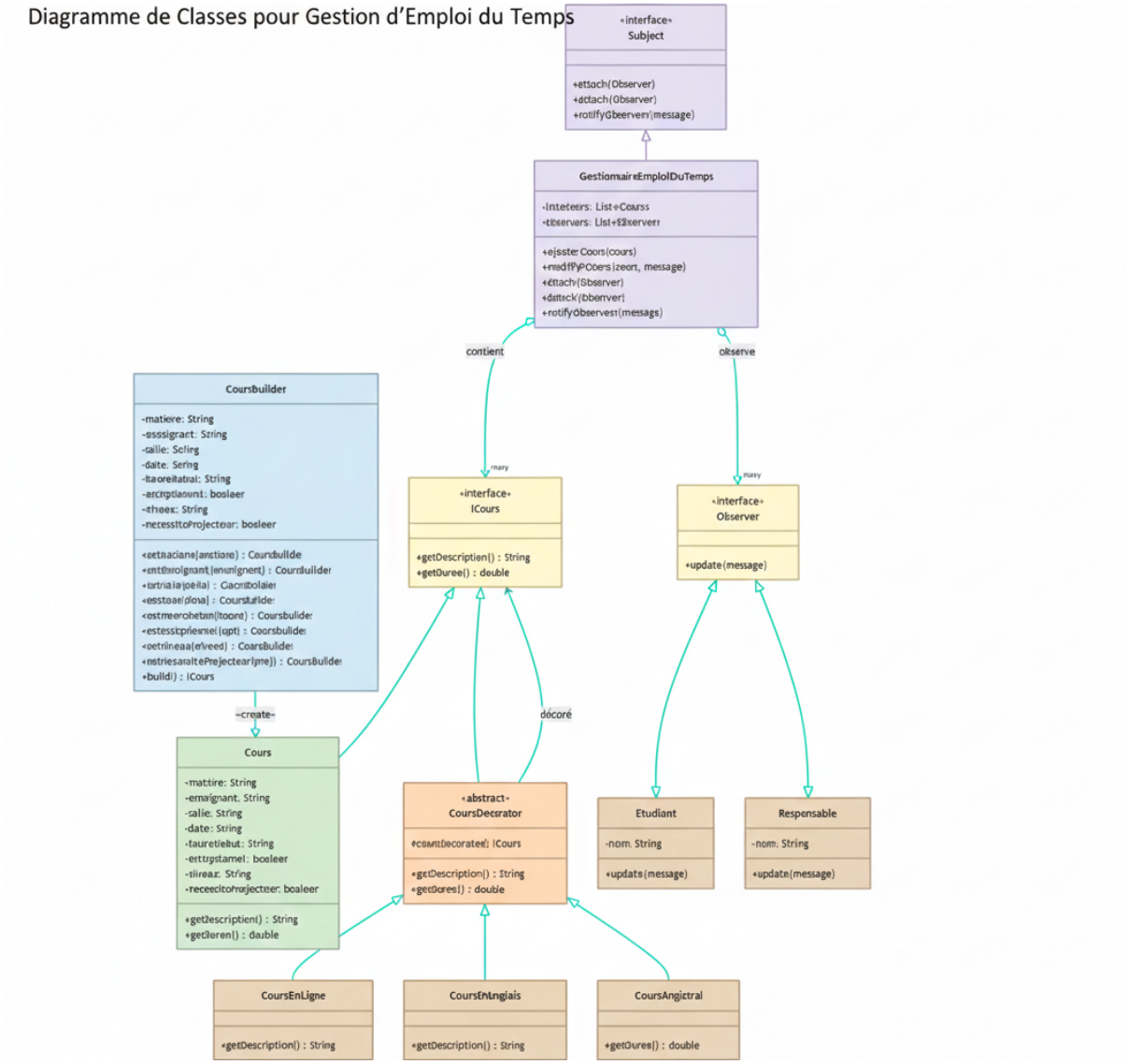

FIGURE 1 – Diagramme de classe - Gestion de l'Emploi du Temps

2 Analyse : Respect des Principes de Conception Logicielle

Le code implémenté dans ce TP respecte globalement les principes de conception logicielle, en particulier les principes SOLID, grâce à l'utilisation appropriée des design patterns. Voici une analyse détaillée :

2.1 Principes SOLID

- **Single Responsibility Principle (SRP)** : Oui, respecté. Chaque classe a une responsabilité unique. Par exemple, `Cours` gère uniquement les données et la description d'un cours, `CoursBuilder` se concentre sur la construction fluide de l'objet, `GestionnaireEmploiDuTemps` gère les notifications, et les décorateurs comme `CoursEnLigne` ajoutent des fonctionnalités spécifiques sans surcharger la classe de base.
- **Open/Closed Principle (OCP)** : Oui, respecté. Le système est ouvert à l'extension (par exemple, on peut ajouter de nouveaux décorateurs pour d'autres spécificités comme "En anglais" sans modifier `Cours` ou `ICours`) mais fermé à la modification des classes existantes.
- **Liskov Substitution Principle (LSP)** : Oui, respecté. Les sous-classes et décorateurs peuvent être substitués à leurs parents sans altérer le comportement attendu. Par exemple, un `CoursEnLigne` peut être utilisé partout où un `ICours` est attendu.
- **Interface Segregation Principle (ISP)** : Oui, respecté. Les interfaces sont petites et ciblées (`ICours` ne contient que deux méthodes essentielles, `Subject` et `Observer` sont minimalistes), évitant les interfaces "gros" qui forceraient des implémentations inutiles.
- **Dependency Inversion Principle (DIP)** : Oui, respecté. Le code dépend des abstractions (interfaces comme `ICours`, `Subject`, `Observer`) plutôt que des implémentations concrètes, facilitant les tests et les extensions.

2.2 Autres Principes Respectés

- **DRY (Don't Repeat Yourself)** : Les méthodes de délégation dans `CoursDecorator` évitent la duplication de code.
- **KISS (Keep It Simple Stupid)** : Les patterns sont utilisés de manière simple, sans surcomplexité.
- **YAGNI (You Ain't Gonna Need It)** : Seul ce qui est nécessaire pour les exigences du TP est implémenté.

2.3 Points Potentiels d'Amélioration

- Le constructeur public de `Cours` avec 8 arguments pourrait violer légèrement le SRP et le KISS, car il est encombrant et propice aux erreurs (ordre des paramètres). Il serait mieux de le rendre privé ou déprécié en faveur du builder, mais il est conservé pour compatibilité avec les tests (`TpTests.java`).
- Les méthodes `update` dans `Etudiant` et `Responsable` utilisent `System.out.println`, ce qui couple le code à la console (violation mineure du DIP). Une meilleure approche serait d'utiliser un logger ou de retourner une chaîne pour plus de flexibilité.

Globalement, le code est bien conçu et respecte les principes essentiels, rendant le système maintenable et extensible.