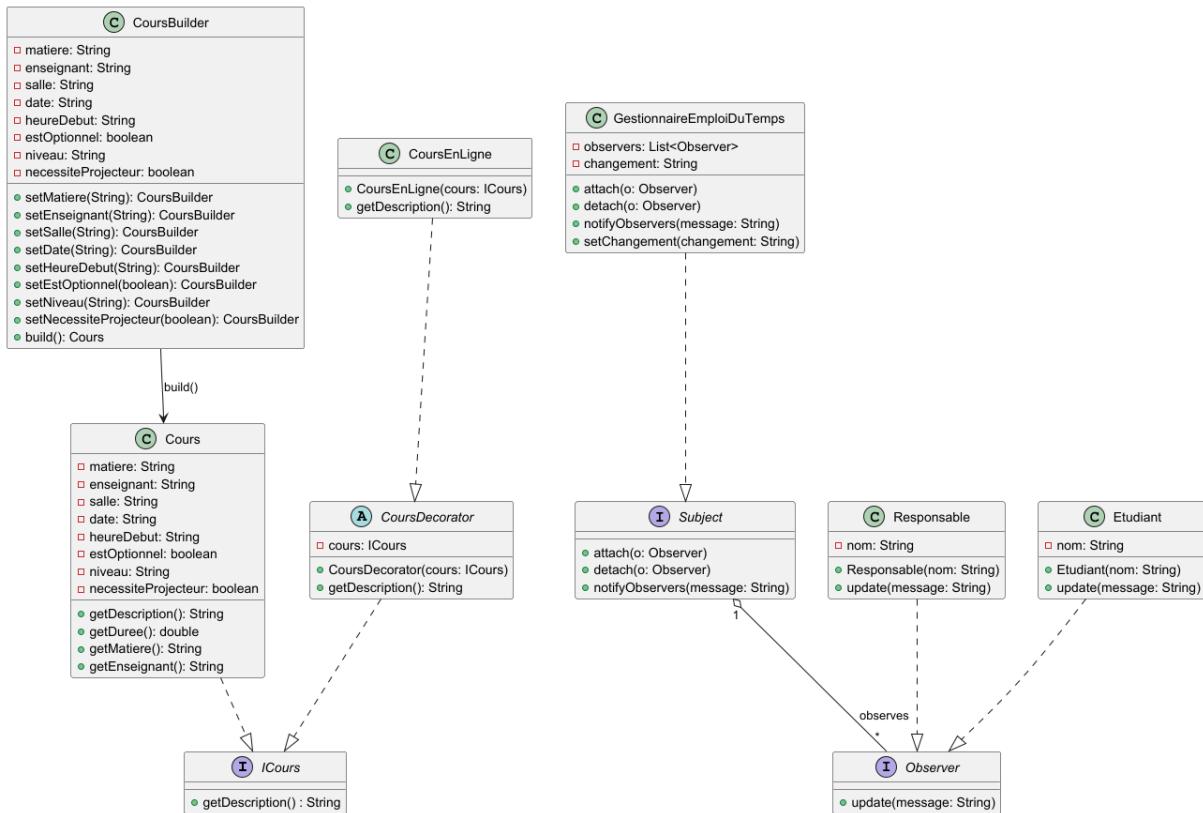


diagramme de classe :



Respect des principes de conception logicielle

Le code développé pour ce projet respecte les principaux principes de conception logicielle, ainsi que quelques bonnes pratiques reconnues :

1. SOLID:

- **SRP (Single Responsibility Principle)** : chaque classe a une responsabilité unique, par exemple CoursBuilder se charge uniquement de construire des objets Cours, GestionnaireEmploiDuTemps gère les notifications.
 - **OCP (Open/Closed Principle)** : le code est ouvert à l'extension mais fermé à la modification, illustré par l'utilisation de CoursDecorator et CoursEnLigne.
 - **LSP (Liskov Substitution Principle)** : les sous-classes respectent le comportement des superclasses, comme CoursEnLigne qui étend CoursDecorator sans altérer le fonctionnement attendu.
 - **ISP (Interface Segregation Principle)** : les interfaces sont spécifiques et concises (ICours, Subject, Observer).
 - **DIP (Dependency Inversion Principle)** : les classes dépendent d'abstractions plutôt que de classes concrètes, facilitant la modularité et le test.

2. Autres bonnes pratiques :

- **DRY (Don't Repeat Yourself)** : le code évite les duplications, centralisant la construction et la décoration des cours.
 - **KISS (Keep It Simple, Stupid)** : le code est simple, lisible et facile à comprendre.

- **YAGNI (You Aren't Gonna Need It)** : seules les fonctionnalités réellement nécessaires ont été implémentées, sans ajouter de complexité superflue.

Conclusion : le design choisi est clair, modulaire et extensible, respectant les principes de conception logicielle et les bonnes pratiques de programmation.