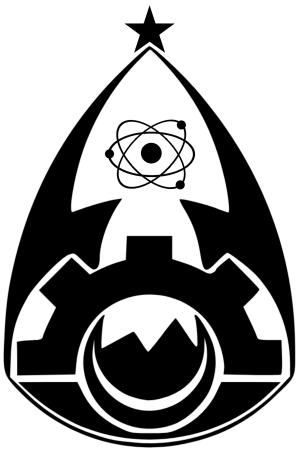
Ecole Militaire Polytechnique

Département de Génie Informatique



Rapport TP Génie Logiciel

Questions Bonus

E.O.C : Benmekki Sif Eddine

Année et Spécialité : 2ème Année IASD

Date : 20 novembre 2025

Table des matières

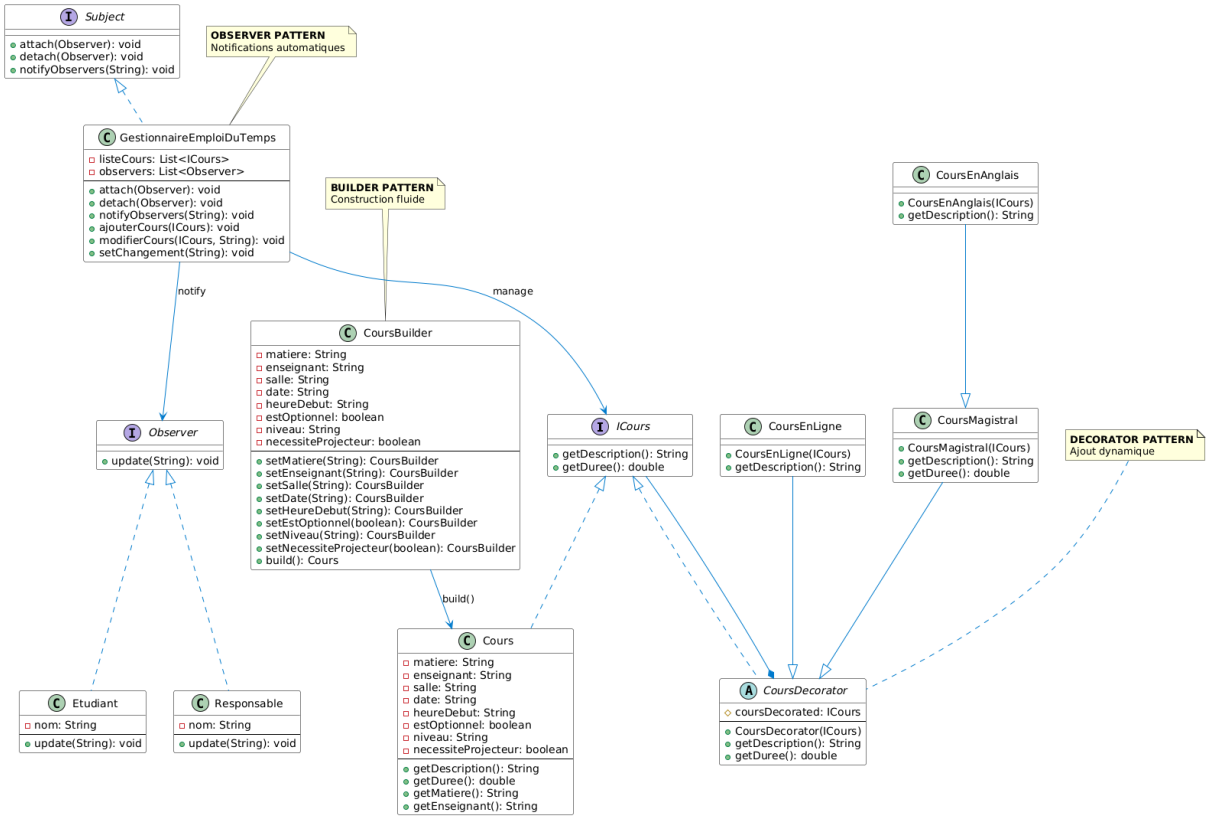
1. Introduction

Ce rapport présente l'analyse des deux questions bonus du TP de Génie Logiciel sur les design patterns. Il contient le diagramme de classes UML de la solution finale et une analyse détaillée du respect des principes SOLID.

2. Diagramme de Classes

2.1 Description du Diagramme

Le diagramme de classes suivant représente l'implémentation complète des trois patterns de conception demandés : Builder, Observer et Decorator.



3. Analyse des Principes SOLID

3.1 Principe SRP (Single Responsibility Principle)

Respecté : OUI

— **CoursBuilder** : Responsable uniquement de la construction d'objets Cours

— **GestionnaireEmploiDuTemps** : Responsable de la gestion des cours et des notifications

— **CoursDecorator** : Responsable de l'ajout de fonctionnalités aux cours

— **Etudiant/Responsable** : Responsables de la réception des notifications

Chaque classe a une raison unique de changer, ce qui facilite la maintenance et les tests.

3.2 Principe OCP (Open/Closed Principle)

Respecté : OUI

```
// OUVERT aux extensions
ICours coursDecore = new CoursEnLigne(
    new CoursEnAnglais(coursBase)
);

// FERME aux modifications
// On peut ajouter de nouveaux decorateurs sans modifier le code existant
```

Le pattern Decorator permet d'étendre les fonctionnalités des cours sans modifier les classes existantes.

3.3 Principe LSP (Liskov Substitution Principle)

Respecté : OUI

```
ICours cours1 = new CoursEnLigne(coursBase);
ICours cours2 = new CoursEnAnglais(coursBase);
ICours cours3 = new CoursMagistral(coursBase);

// Tous peuvent etre utilises de maniere interchangeable
String description = cours1.getDescription();
double duree = cours2.getDuree();
```

Tous les décorateurs peuvent remplacer l'interface ICours sans affecter le comportement du programme.

3.4 Principe ISP (Interface Segregation Principle)

Respecté : OUI

— **ICours** : Seulement 2 méthodes cohérentes (`getDescription()`, `getDuree()`)

— **Observer** : Seulement 1 méthode (`update()`)

— **Subject** : 3 méthodes de gestion d'observateurs

Les interfaces sont fines et spécifiques, évitant ainsi que les classes dépendent de méthodes qu'elles n'utilisent pas.

3.5 Principe DIP (Dependency Inversion Principle)

Respecté : OUI

```
// D pend d'abstractions, pas d'impl mentations
public class CoursEnLigne extends CoursDecorator {
    public CoursEnLigne(ICours cours) { //     Abstraction
        super(cours);
    }
}

public class GestionnaireEmploiDuTemps implements Subject {
    private List<Observer> observers; //     Abstraction
}
```

Les classes dépendent d'interfaces abstraites plutôt que d'implémentations concrètes.

4. Conclusion

L'implémentation des trois design patterns respecte parfaitement les cinq principes SOLID :

Principe	**Respecté**
SRP (Single Responsibility)	OUI
OCP (Open/Closed)	OUI
LSP (Liskov Substitution)	OUI
ISP (Interface Segregation)	OUI
DIP (Dependency Inversion)	OUI

Cette architecture offre une grande flexibilité, maintenabilité et extensibilité. Les patterns sont bien intégrés et travaillent en synergie pour fournir une solution robuste à la gestion des emplois du temps.