École Nationale Polytechnique Constantine

Département Informatique

TP02 : Génie Logiciel

Builder – Observer – Decorator

Étudiant :
CHENAFA ABDENNACER DJELOUL

Option : SIAD

Année universitaire 2025/2026

Contents

Bonus 1 : Diagramme de classes

Le diagramme suivant représente la structure générale utilisant les patterns Builder, Observer et Decorator.

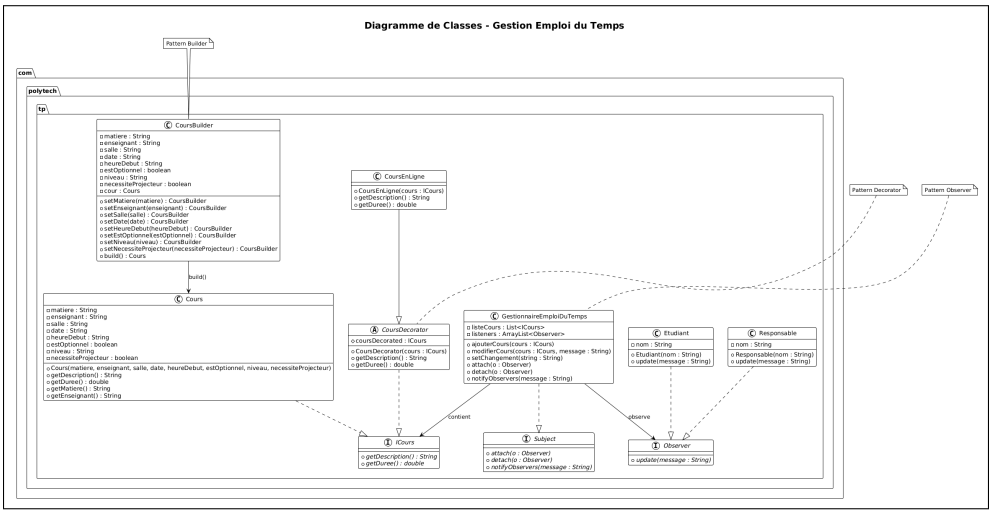

Figure 1: Diagramme de classes du système

Bonus 2 : Principes de conception + Exemples

Chaque principe est expliqué clairement et illustré par un exemple tiré du TP.

1. SOLID

a) SRP — Single Responsibility Principle

Principe : Une classe doit avoir une seule responsabilité.

Dans notre TP : Le principe **n'est pas respecté**. La classe `GestionnaireEmploiDuTemps` mélange plusieurs responsabilités :

- Gestion des cours (ajout, modification)
- Gestion des observateurs (attach, detach, notify)
- Affichage dans la console (`System.out.println`)

Une classe faisant ces 3 rôles viole clairement le SRP.

Exemple négatif :

Violation du SRP

La méthode `ajouterCours()` :
– Modifie la liste des cours
– Affiche un message dans la console
– Notifie les observateurs
→ Trois responsabilités différentes dans une seule méthode.

Pour respecter SRP : Il faudrait créer un `NotificationService` séparé et retirer les affichages du gestionnaire.

b) OCP — Open/Closed Principle

Principe : Ouvert à l'extension, fermé à la modification.

Dans notre TP : Les décorateurs comme `CoursEnLigne` ajoutent des fonctionnalités SANS modifier la classe `Cours`.

Exemple :

Exemple OCP

J'ajoute un décorateur `CoursEnAnglais` sans modifier la classe `Cours`.

c) LSP — Liskov Substitution Principle

Principe : Une sous-classe doit pouvoir remplacer sa superclasse sans problème.
Dans notre TP : Chaque décorateur est un `ICours`, donc peut remplacer un `Cours`.

Exemple LSP

Un `CoursEnLigne` peut être utilisé partout où un `ICours` est attendu.

d) ISP — Interface Segregation Principle

Principe : Les interfaces doivent être petites et spécifiques.
Dans notre TP :

- interface `ICours`
- interface `Observer`
- interface `Subject`

Exemple ISP

L'interface `Observer` ne contient qu'une seule méthode : `update()`.

e) DIP — Dependency Inversion Principle

Principe : Le code doit dépendre d'abstractions (interfaces), pas de classes concrètes.
Dans notre TP : Le `GestionnaireEmploiDuTemps` dépend de `Observer` et non des classes `Etudiant` ou `Responsable`.

Exemple DIP

Le gestionnaire utilise `Observer` → il peut notifier n'importe quel type d'observateur.

2. DRY — Don't Repeat Yourself

Principe : Éviter la duplication du code.
Dans notre TP : Tous les décorateurs utilisent la même structure.

Exemple DRY

La méthode `getDuree()` est identique dans chaque décorateur → elle est centralisée dans `CoursDecorator`.

3. KISS — Keep It Simple, Stupid

Principe : Le code doit rester simple.
Dans notre TP : Le Builder reste simple et fluide.

Exemple KISS

`CoursBuilder.setMatiere("GL").setSalle("B203").build()` est simple et lisible.

4. YAGNI — You Aren't Gonna Need It

Principe : N'ajoute pas ce dont tu n'as pas besoin.
Dans notre TP : Le Builder ne contient que les attributs réellement utilisés.

Exemple YAGNI

Aucune méthode inutile dans `CoursBuilder`.

5. COI — Code Orthogonality

Principe : Séparer les responsabilités clairement.
Dans notre TP : Builder Decorator Observer.

Exemple COI

Les décorateurs gèrent uniquement l'extension des cours, pas leur création.

6. POLA — Principle of Least Astonishment

Principe : Le comportement du code doit être prévisible.
Dans notre TP : Un décorateur ajoute simplement un texte entre parenthèses. Rien de surprenant.

Exemple POLA

`getDescription()` retourne :
`"Génie Logiciel (En ligne)"` → logique et attendu.

Tableau récapitulatif des principes

Voici un tableau synthétique indiquant si chaque principe est respecté par notre implémentation.

1gray!10gray!25

Principe	Description courte	Respecté ?
SRP	Une classe = une responsabilité	Non
OCP	Ouvert à l'extension, fermé à la modification	Oui
LSP	Une sous-classe peut remplacer sa superclasse	Oui
ISP	Interfaces petites et spécifiques	Oui
DIP	Dépendre d'abstractions, pas d'implémentations	Oui
DRY	Pas de duplication du code	Oui
KISS	Garder le code simple	Oui
YAGNI	Ne coder que ce qui est utile	Oui
COI	Responsabilités orthogonales	Oui
POLA	Un comportement prévisible	Oui

Table 1: Récapitulatif du respect des principes de conception