PRINCIPE VIOLES DANS LE TP

ZEGHLACHE Khouloud Assia

20 novembre 2025

Introduction

Ce document analyse les violations des principes de conception dans des patterns de conception Observer, Decorator et Builder qui sont utilisée au Tp .

1 Violations Identifiées

1.1 Violation du Principe d'Inversion des Dépendances (DIP)

1.1.1 Problème

Builder couplé à l'implémentation concrète

```
CoursBuilder --> Cours : build()
//  Dépendance directe vers une implémentation
```

1.1.2 Impact

— Le `CoursBuilder` est fortement couplé à la classe concrète `Cours`
— Impossible de construire d'autres types d'implémentations de `ICours`
— Violation du principe : "Dépendre des abstractions, pas des implémentations"

1.2 Violation du Principe Composition over Inheretance (COI)

1.2.1 Problème

Risque d'incompatibilité des décorateurs

```
class CoursMagistral extends CoursDecorator {
    //  Comportement potentiellement incompatible

}
```

1.2.2 Impact

— Les décorateurs fils herite du classe mére au lieu d'utiliser la composition.

1.3 Violation du Principe de Responsabilité Unique (SRP)

1.3.1 Problème

GestionnaireEmploiDuTemps a trop de responsabilités

```
class GestionnaireEmploiDuTemps {
    -listeCours: List<ICours>
    -observers: List<Observer>
    +ajouterCours(c: ICours)
    +modifierCours(c: ICours, msg: String)
    +setChangement(msg: String)
}
```

1.3.2 Impact

— La classe gère à la fois l'ajout et la modification d'un cours et la notification des observateurs.

2 Conclusion

Malgré les violations techniques identifiées, cette architecture reste excellente car les patterns implémentés apportent des avantages décisifs . Donc une conception qui utilise des patterns standards, même avec quelques écarts théoriques, vaut mieux qu'une architecture parfaitement SOLID mais obscure.