TP 0 301: Principes SOLID

Nom : Sharifatou Malai

Matricule : XXXXXXX

Année académique : 2025 – 2026

Enseignants : Dr Monthe, Mr Dave

ICT4D UY1

Introduction

Les principes SOLID sont un ensemble de bonnes pratiques de conception orientée objet permettant de produire des logiciels maintenables, évolutifs et robustes.

Dans ce travail pratique, nous illustrons chaque principe SOLID à travers un exemple avant refactoring (violant le principe) et un exemple après refactoring (respectant le principe), conformément au support du cours ICT 301.

1. SRP – Single Responsibility Principle

Définition

Une classe ne doit avoir qu'une seule responsabilité.

a) Avant refactoring

La classe Book gère plusieurs responsabilités

Exemple

Une seule classe (Book) gère :

-données

-affichage

-sauvegarde

-logique métier

➡ Violation du SRP

Book

- title

- author

- content

+ printToScreen()

+ saveToDatabase()

+ emprunter()

b)Après refactoring

Les responsabilités ont été séparées en plusieurs classes.

BookSRP -> Données

BookPrinter -> Affichage

BookSaver -> Persistance

BookBusinessLogic -> Logique métier

2. OCP – Open Closed Principle

OCP = Ouvert à l'extension, fermé à la modification

On ne modifie pas le code existant, on ajoute du nouveau code.

Avant refactoring

Utilisation de instanceof

AreaCalculator

|– Rectangle

|– Circle

Après refactoring

Introduction de l'interface Shape

Ajout de nouvelles formes sans modifier le calculateur

Diagramme :

Shape

├─ Rectangle

└─ Circle

AreaCalculator → Shape

STRUCTURE FINALE ATTENDUE POUR OCP

OCP/

├── before/

│ ├── AreaCalculator.java

│ ├── Rectangle.java

│ ├── Circle.java

│ └── Main.java

│

└── after/

├── Shape.java

├── Rectangle.java

├── Circle.java

├── AreaCalculator.java

└── Main.java

3. LSP – Liskov Substitution Principle

Définition

Une classe fille doit pouvoir remplacer sa classe mère sans altérer le comportement.

Avant refactoring

Square hérite de Rectangle

Comportement inattendu

Diagramme :

Rectangle

↑

Square

Après refactoring

Suppression de l'héritage incorrect

Interface commune Shape

Diagramme :

Shape

├─ Rectangle

└─ Square

4. ISP – Interface Segregation Principle

Définition

Une classe ne doit pas être forcée d'implémenter des méthodes inutiles.

Avant refactoring

Interface Worker trop large

Robot obligé d'implémenter eat()

Worker

+ work()

+ eat()

Human, Robot

Après refactoring

Interfaces séparées :

Workable

Feedable

Workable –> Human, Robot

Feedable –> Human

5. DIP – Dependency Inversion Principle

Définition

Les modules de haut niveau doivent dépendre d’abstractions.

Avant refactoring

UserService dépend directement de MySQLDatabase

UserService –> MySQLDatabase

Après refactoring

Interface Database

Possibilité de changer de base sans modifier UserService

Diagramme :

Database

├— MySQLDatabase

└— PostgreSQLDatabase

UserService → Database

Conclusion

L’application des principes SOLID améliore la maintenabilité et la qualité du code.

Ce travail a permis de comprendre l’importance des principes SOLID dans la conception logicielle. Leur application améliore la maintenabilité, la flexibilité et la qualité globale du code. Les exemples étudiés montrent clairement l’intérêt du refactoring dans une démarche d’ingénierie logicielle.

Structure final

TP_0_ICT301_nom_prenom/

├── DIP/

├── ISP/

├── LSP/

├── OCP/

├── SRP/

├── README.md

└── ICT301_SOLID.pdf ← CE PDF