

TP ICT-301 : Principes SOLID en Java

Diagrammes UML des exemples avant et après refactoring

NKONGMENOCK LOIC PARFAIT

Matricule : 23U2241

Introduction

Ce document met en exergue la représentation UML de chaque modèle des exemples des cinq principes SOLID vus en cours.

Pour chaque principe (SRP, OCP, LSP, ISP, DIP), nous présentons :

- Le diagramme de classes "Avant Refactoring" : illustrant la violation du principe avec le code initial implémenté.
- Le diagramme de classes "Après Refactoring" : montrant comment le respect du principe est obtenu grâce à une meilleure conception.

Certains principes incluent également un exemple supplémentaire issu directement des supports de cours pour compléter la démonstration.

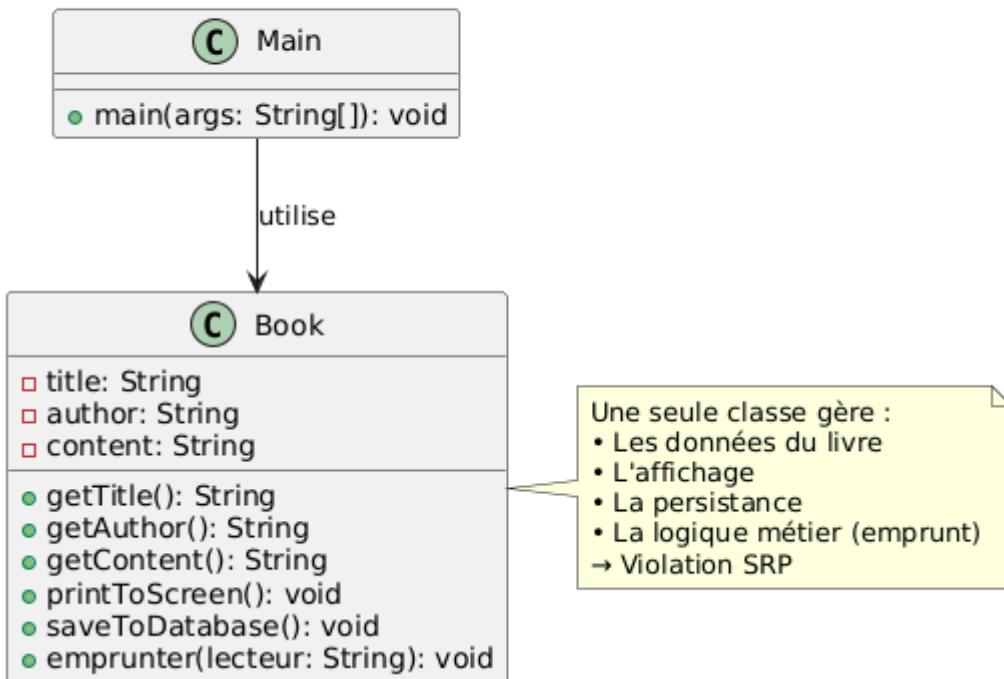
Ces diagrammes ont été réalisés à partir du code Java développé dans le cadre de ce TP, en respectant fidèlement les implementations avant et après application de chaque principe SOLID.

Objectif : visualiser clairement les problèmes de conception initiaux et les solutions apportées par les bonnes pratiques SOLID.

1. PRINCIPE SRP(Single Responsibility Principle)

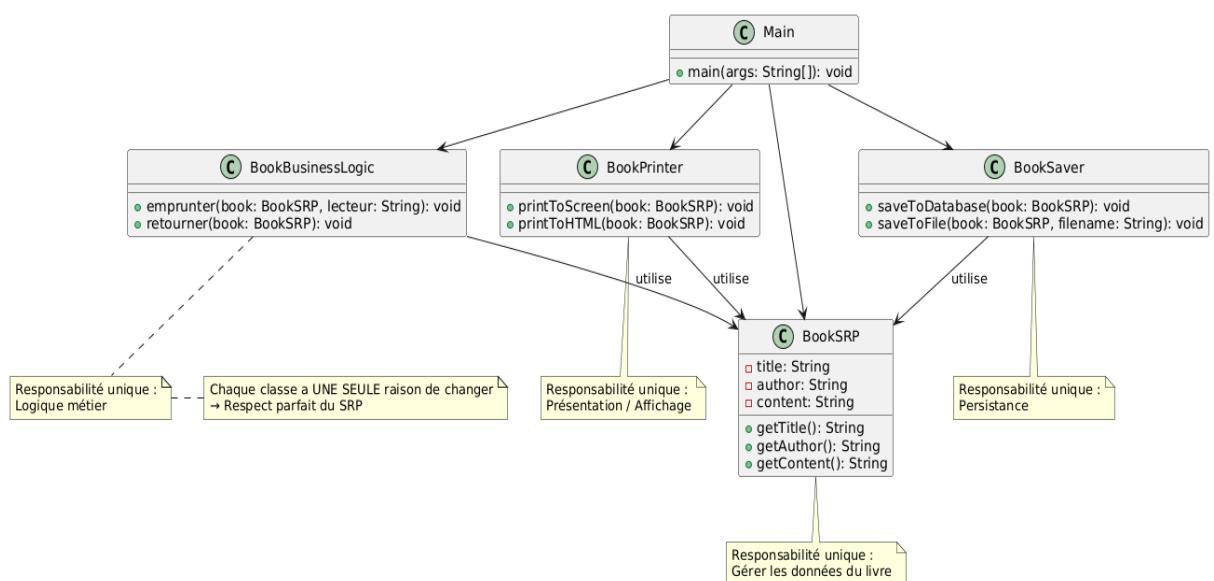
- Violation SRP - Avant Refactoring

Violation SRP - Avant Refactoring (La classe Book fait trop de choses)



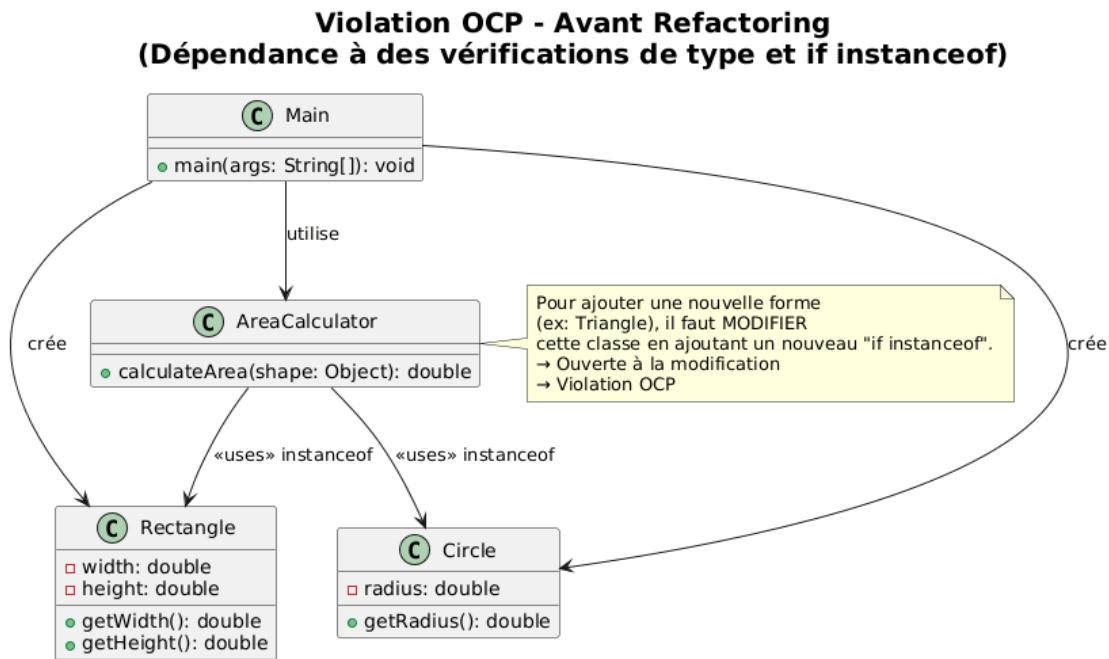
- Respect SRP - Après Refactoring

Respect SRP - Après Refactoring (Chaque classe a une seule responsabilité)



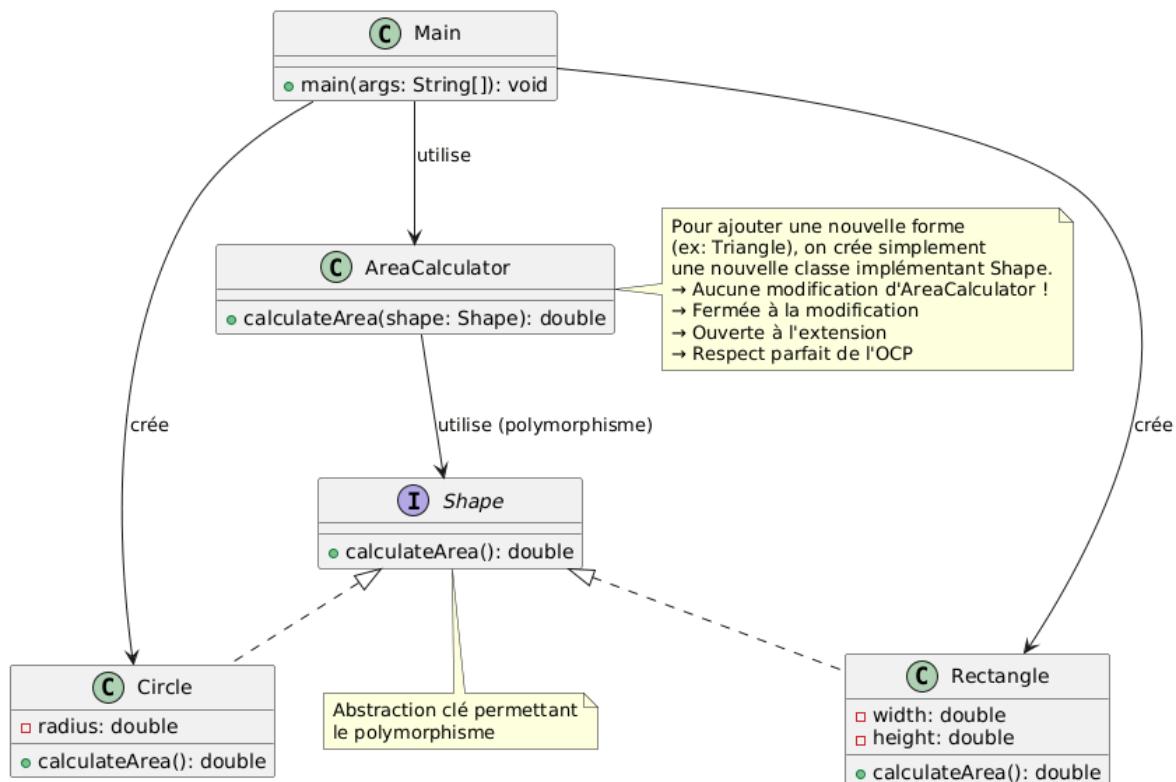
2 PRINCIPE OCP(Open/Closed Principle)

- Violation OCP - Avant Refactoring



- Respect COP - Après Refactoring

Respect OCP - Après Refactoring (Ouvert à l'extension, fermé à la modification via polymorphisme)

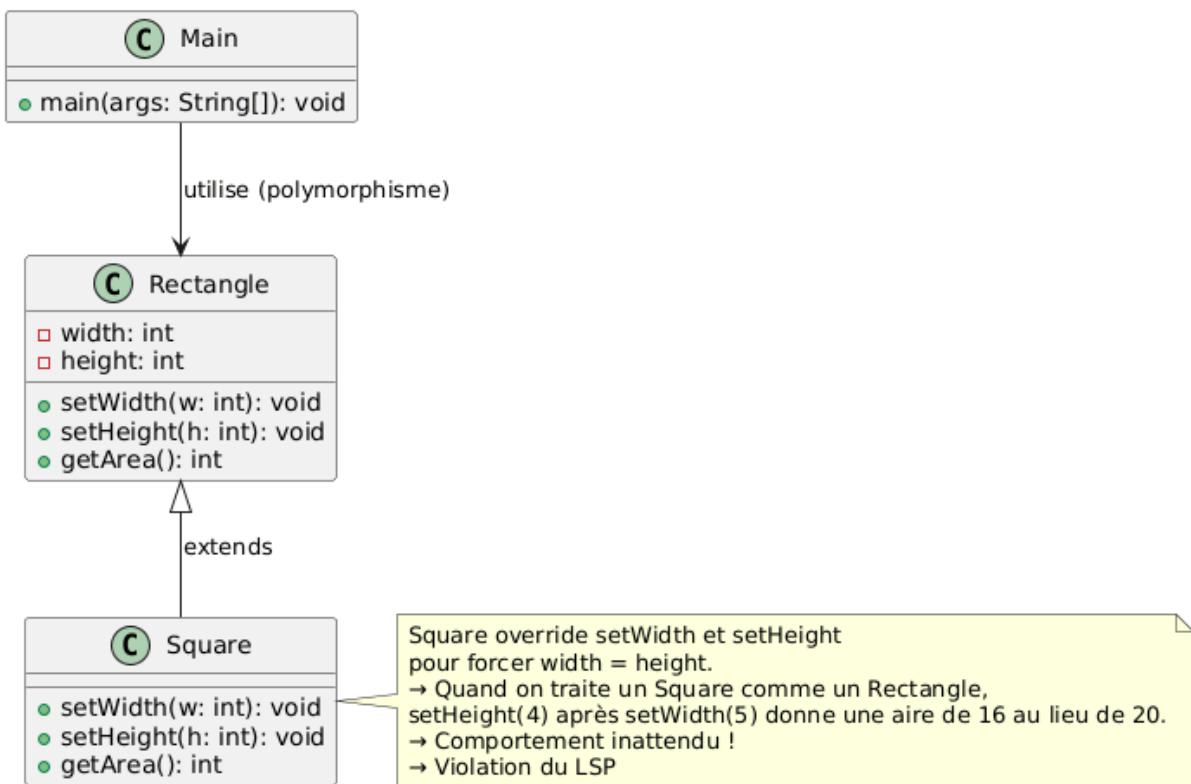


3 PRINCIPE LSP(Liskov Substitution Principle)

3.1- Exemple 1

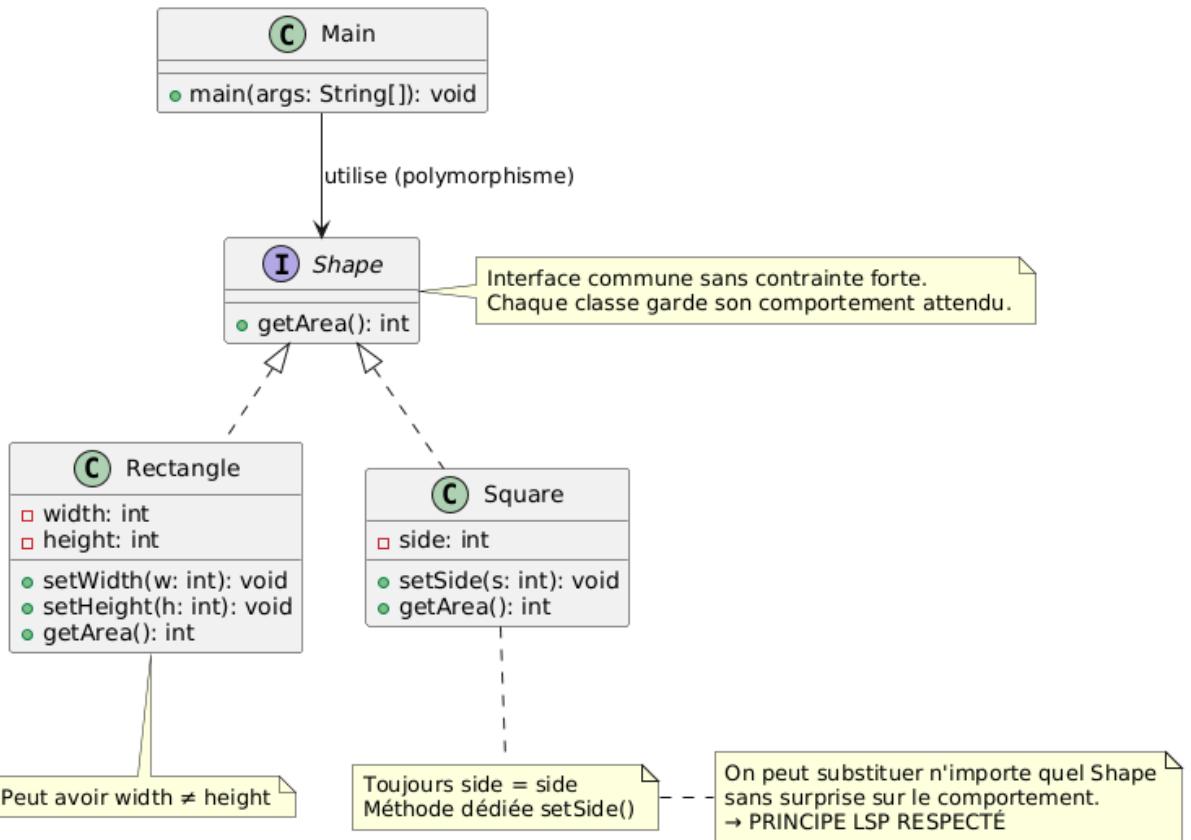
- Violation LSP - Avant Refactoring

Violation LSP - Avant Refactoring (Square hérite de Rectangle → comportement inattendu)



- Respect LSP - Après Refactoring

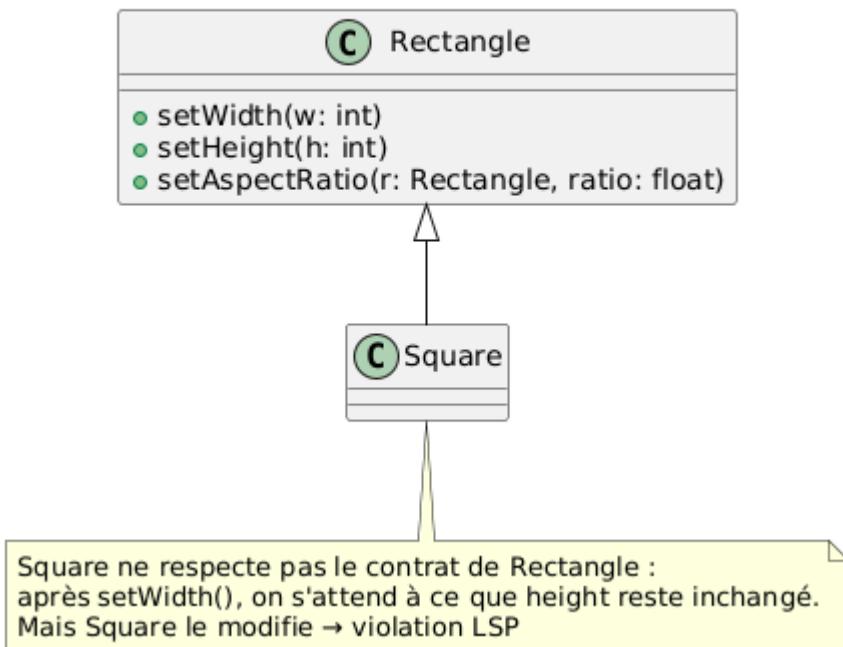
Respect LSP - Après Refactoring (Pas d'héritage forcé → substitution sûre)



3.2 – Exemple 2

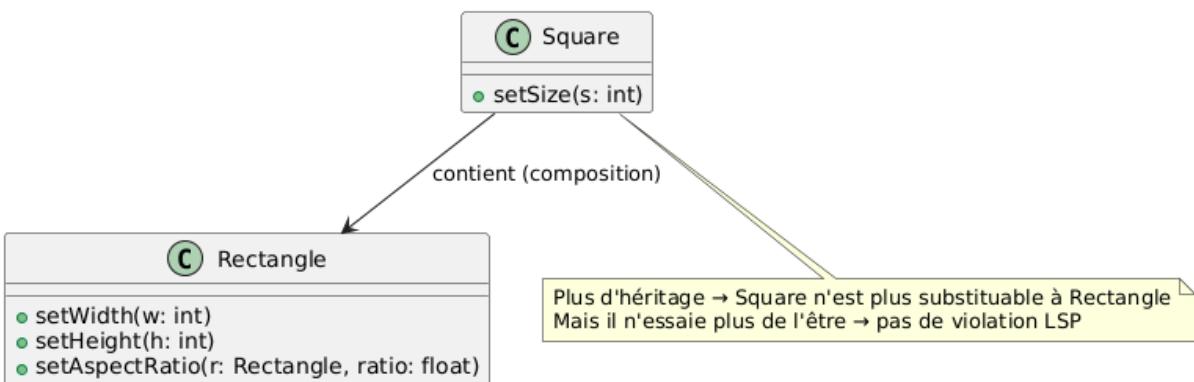
- Violation LSP - Avant Refactoring

LSP Exemple 1 - Violation (du cours) (Carré hérite de Rectangle)



- Respect SRP - Après Refactoring

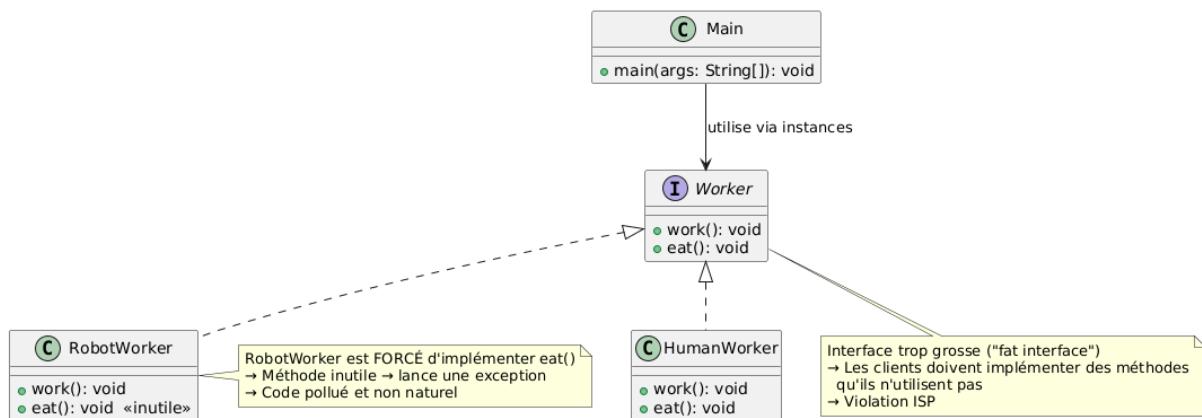
LSP Exemple 1 - Refactoring (du cours) (Carré utilise Rectangle par composition)



4 PRINCIPE ISP(Interface Segregation Principle)

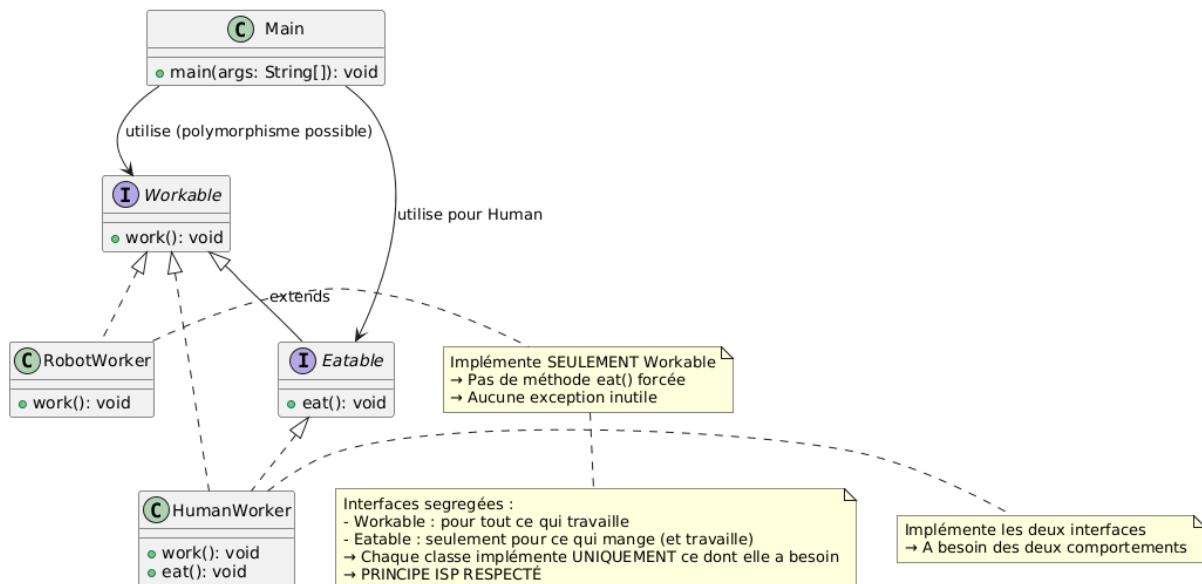
- Violation ISP - Avant Refactoring

Violation ISP - Avant Refactoring (Interface "fat" Worker → méthodes inutiles forcées)



- Respect ISP - Après Refactoring

Respect ISP - Après Refactoring (Interfaces petites et ciblées)

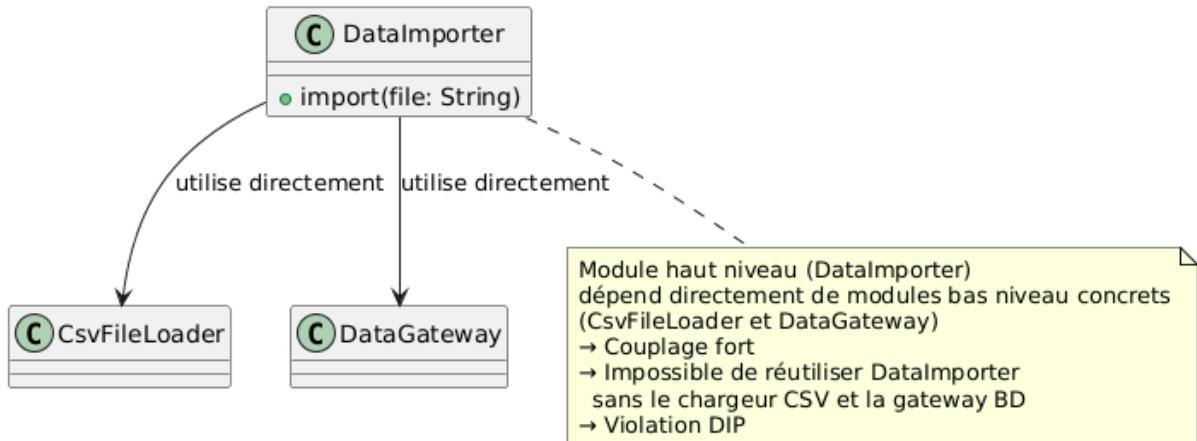


5. PRINCIPE DIP(Dependency Inversion Principle).

5.1 – exemple 1

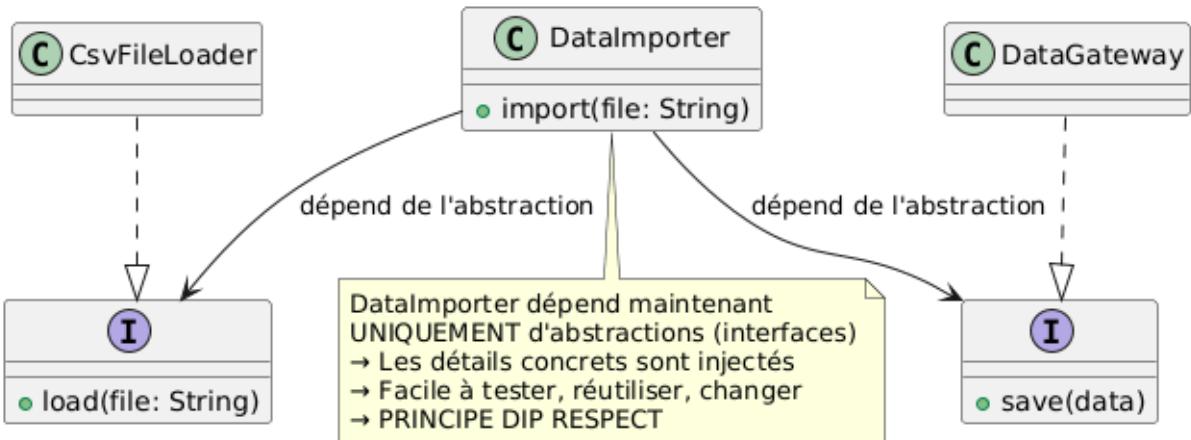
- Violation DIP - Avant Refactoring

Violation DIP - Exemple 1 (du cours) (DataImporter dépend de détails bas niveau concrets)



- Respect DIP - Après Refactoring

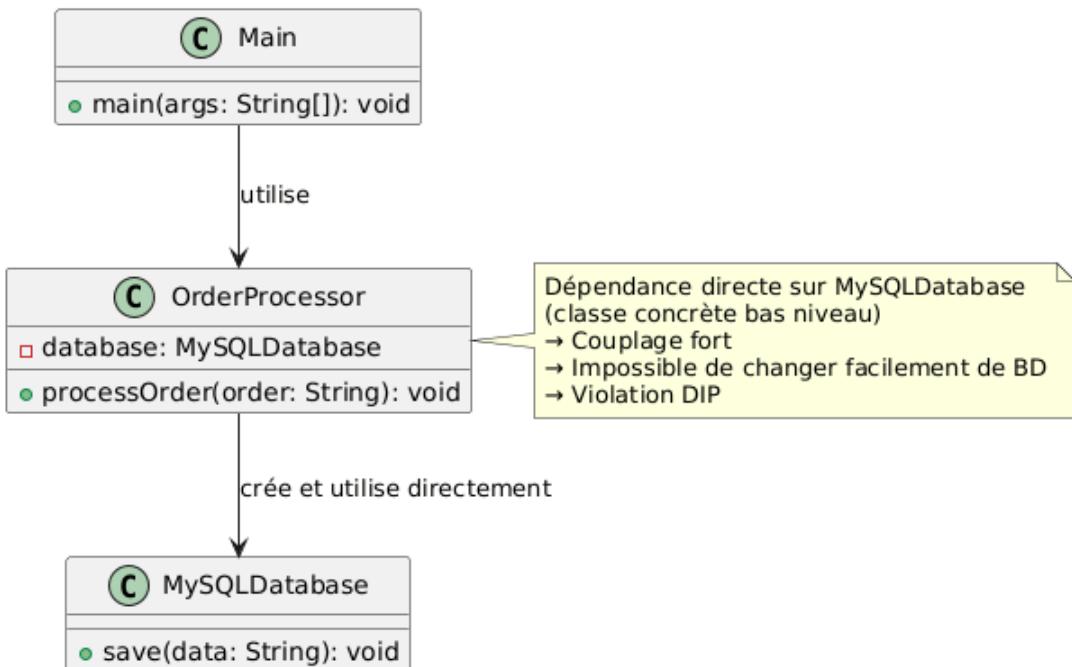
Respect DIP - Exemple 1 Refactoring (Inversion via abstractions)



5.2- Exemple 2

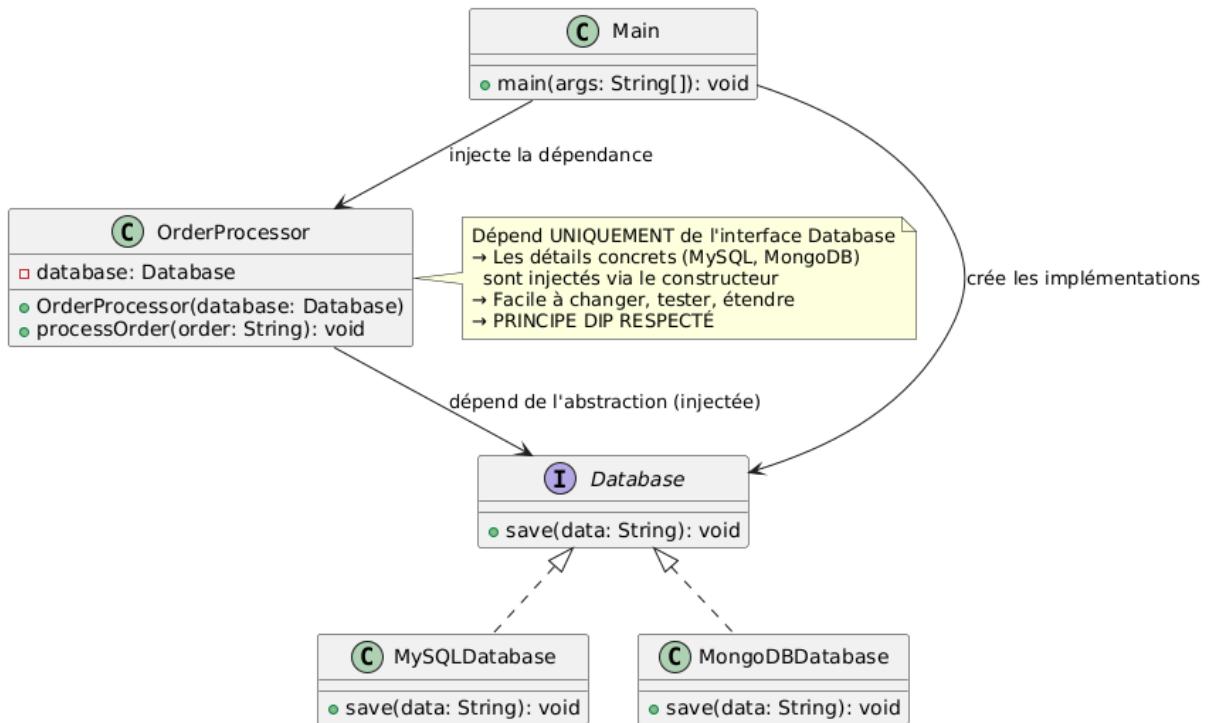
- Violation DIP - Avant Refactoring

Violation DIP - Exemple 2 (notre implémentation avant) (OrderProcessor dépend d'une implémentation concrète)



- Respect DIP - Après Refactoring

Respect DIP - Exemple 2 (notre implémentation après) (Inversion de dépendance via abstraction)



Conclusion

Les principes SOLID constituent les fondations d'une conception logicielle robuste, maintenable et extensible.

Ce TP a permis de mettre en pratique chaque principe à travers des exemples concrets en Java, en passant d'une implémentation problématique à une version refactored respectant les bonnes pratiques.