

Diagrammes UML des exemples avant et après utilisation des principes solides

ZIEM DAVID DE JESUIS

Matricule : 23V2456

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

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

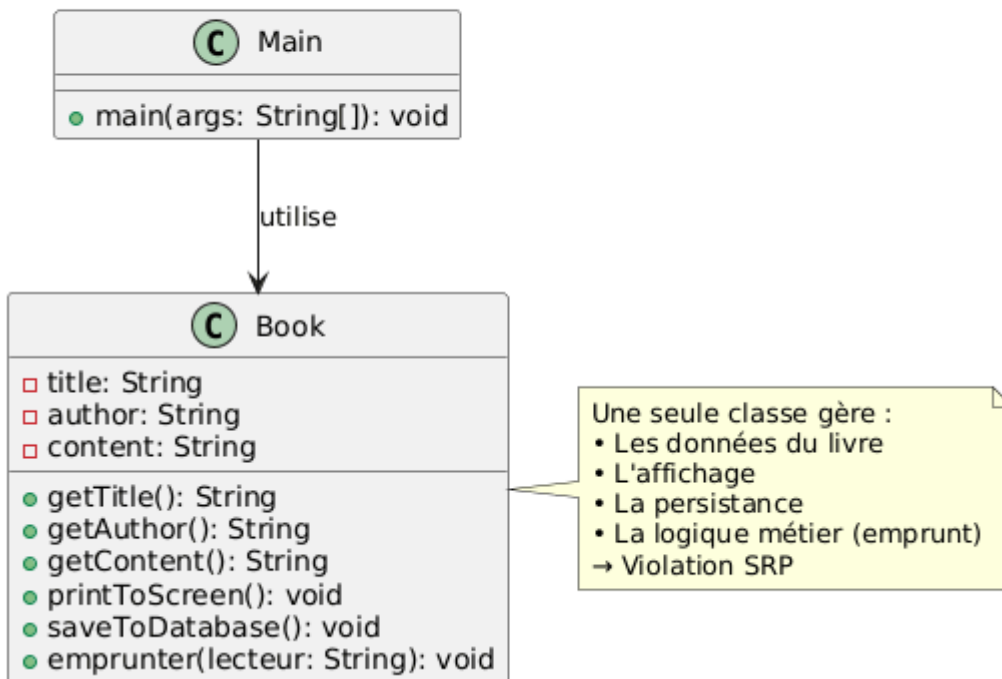
Certains principes incluent également un exemple supplémentaire issu directement des supports de cours pour compléter la démonstration.

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

1. PRINCIPE SRP(Single Responsibility Principle)

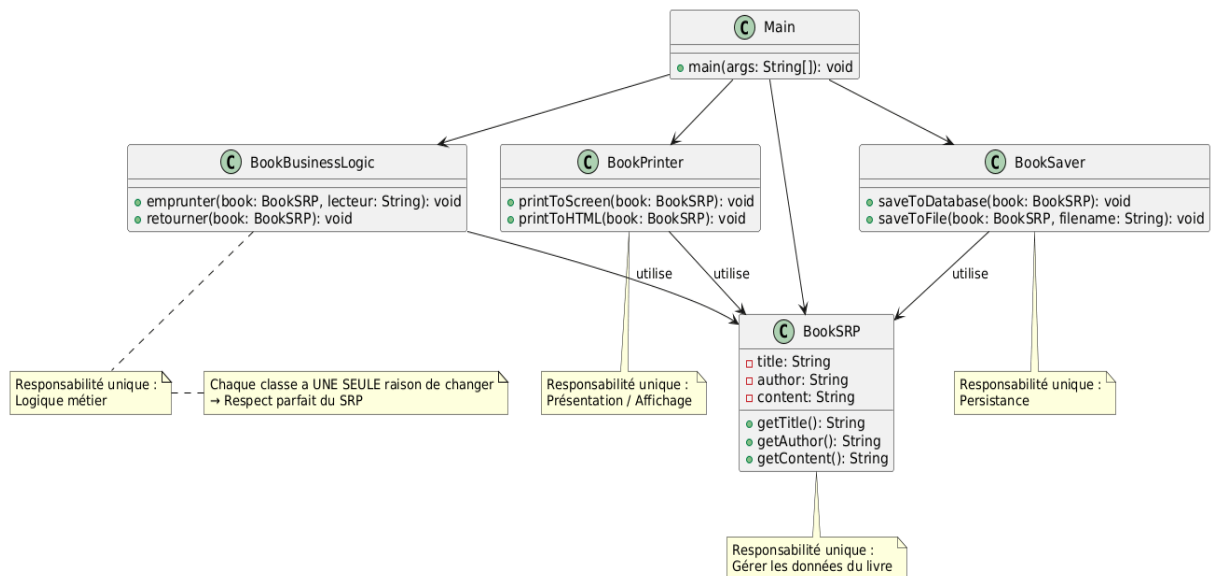
- Violation SRP - Avant Refactoring

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

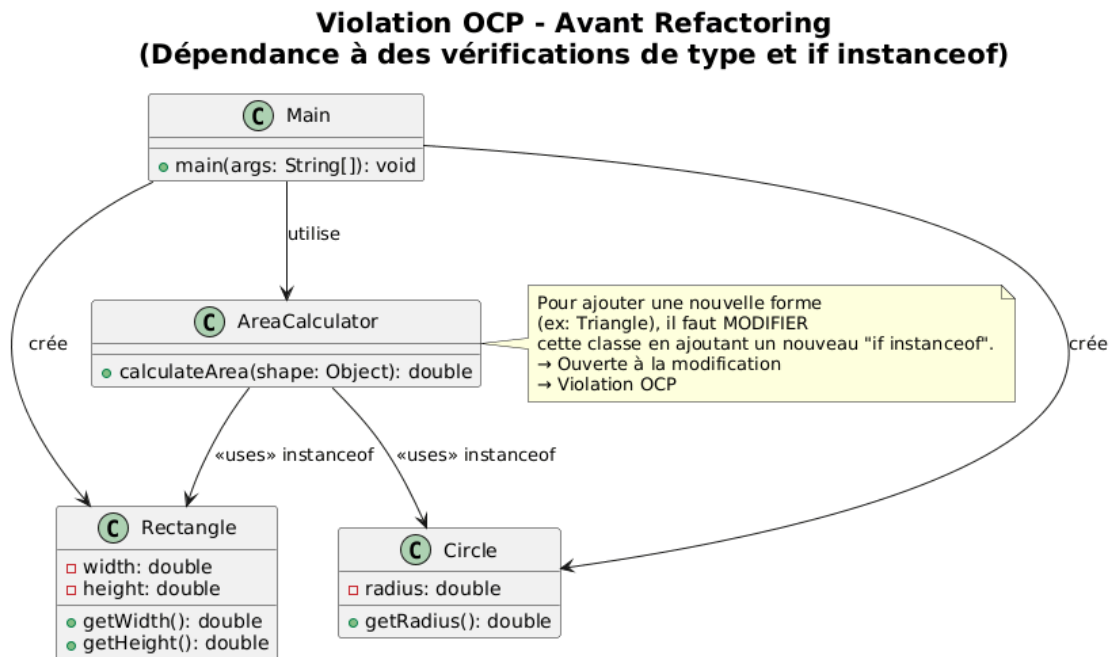


- Respect SRP - Après Refactoring

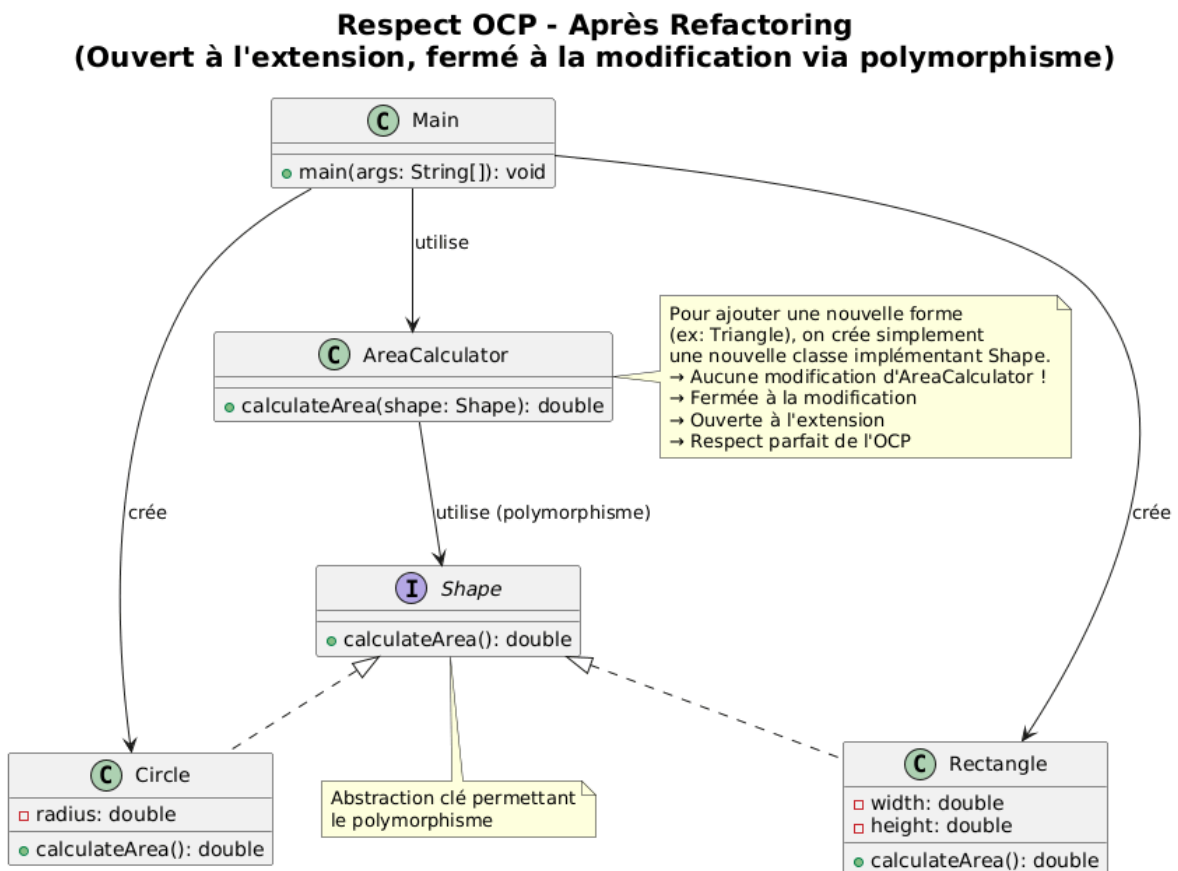
Respect SRP - Après Refactoring (Chaque classe a une seule responsabilité)



- Violation OCP - Avant Refactoring



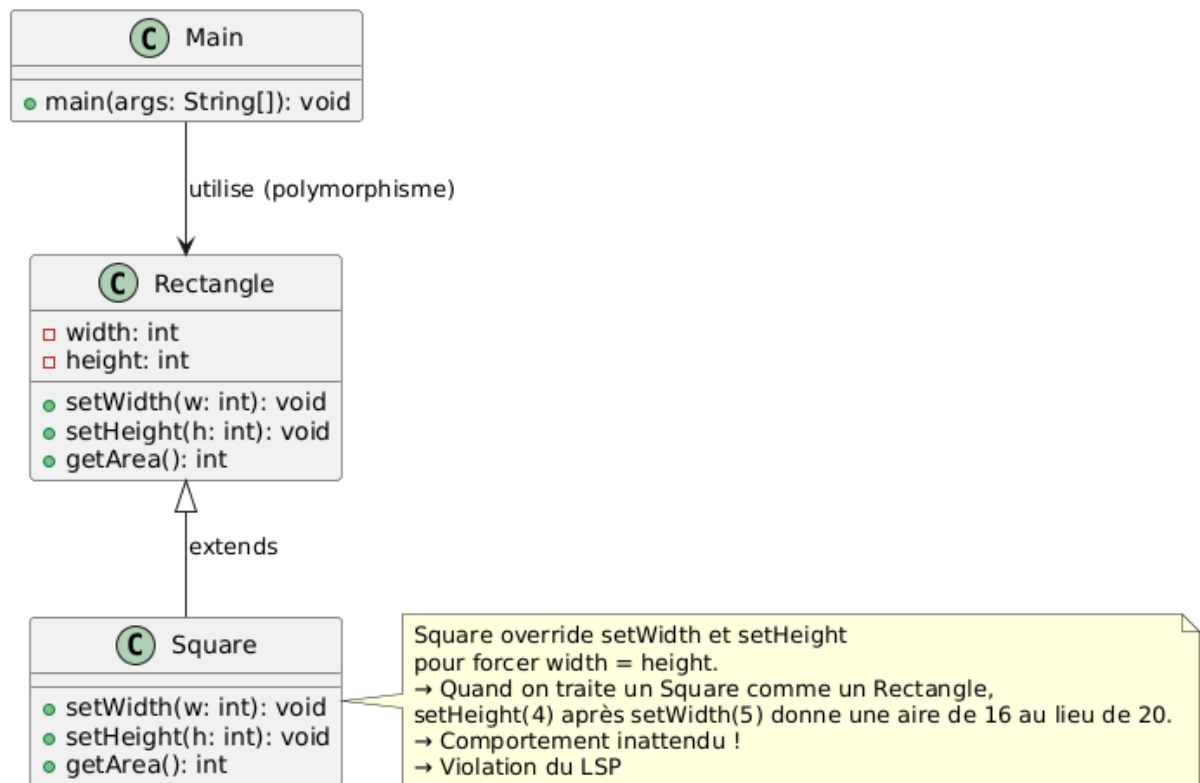
- Respect OCP - Après Refactoring



3.1- Exemple 1

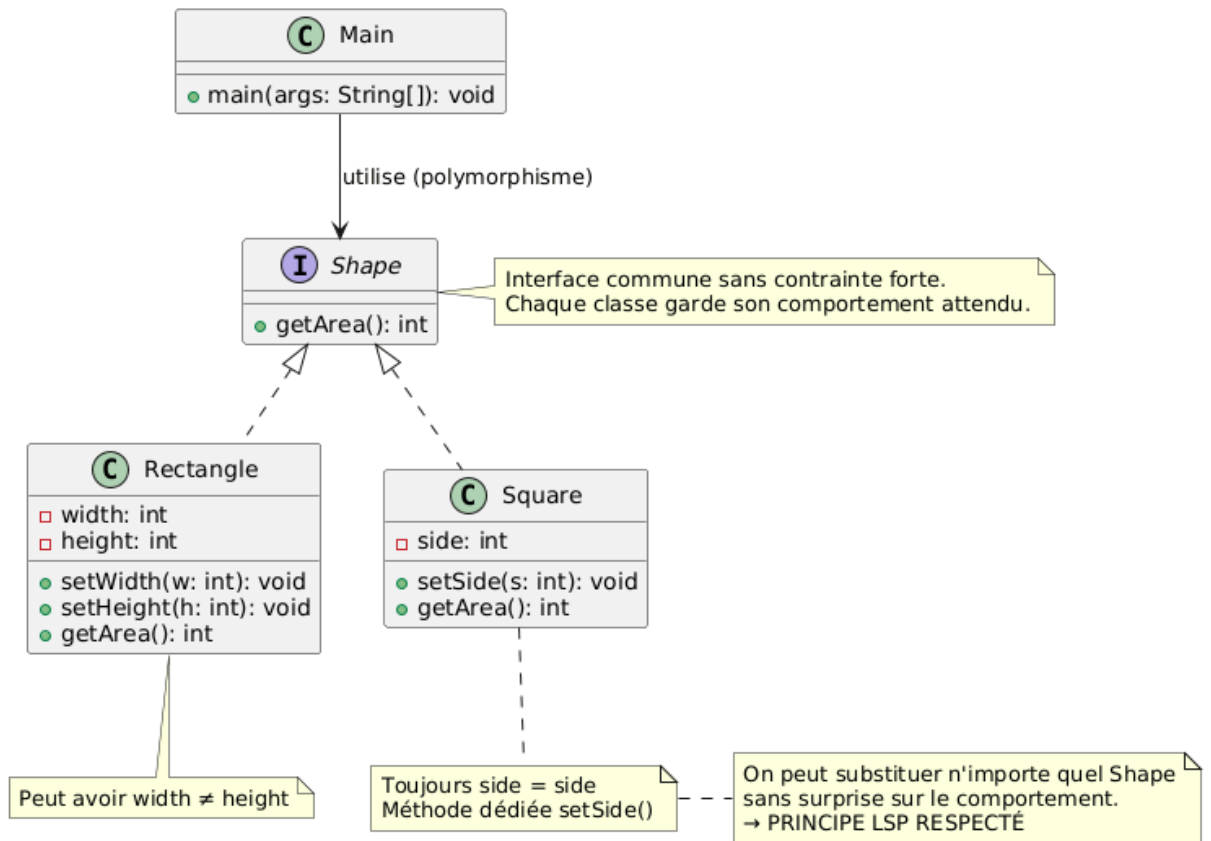
- Violation LSP - Avant Refactoring

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



- Respect LSP - Après Refactoring

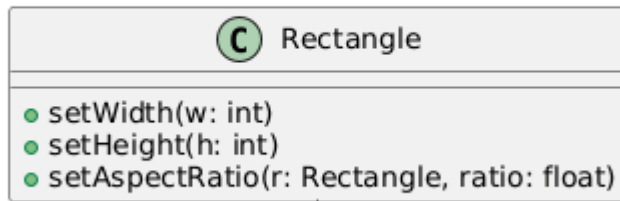
Respect LSP - Après Refactoring (Pas d'héritage forcé → substitution sûre)



3.2 – Exemple 2

- Violation LSP - Avant Refactoring

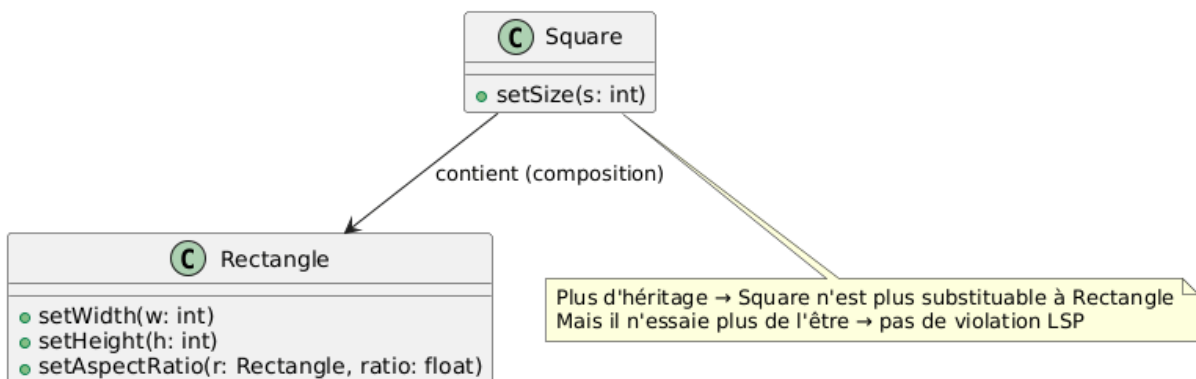
LSP Exemple 1 - Violation (du cours) (Carré hérite de Rectangle)



Square ne respecte pas le contrat de Rectangle :
après setWidth(), on s'attend à ce que height reste inchangé.
Mais Square le modifie → violation LSP

- Respect SRP - Après Refactoring

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

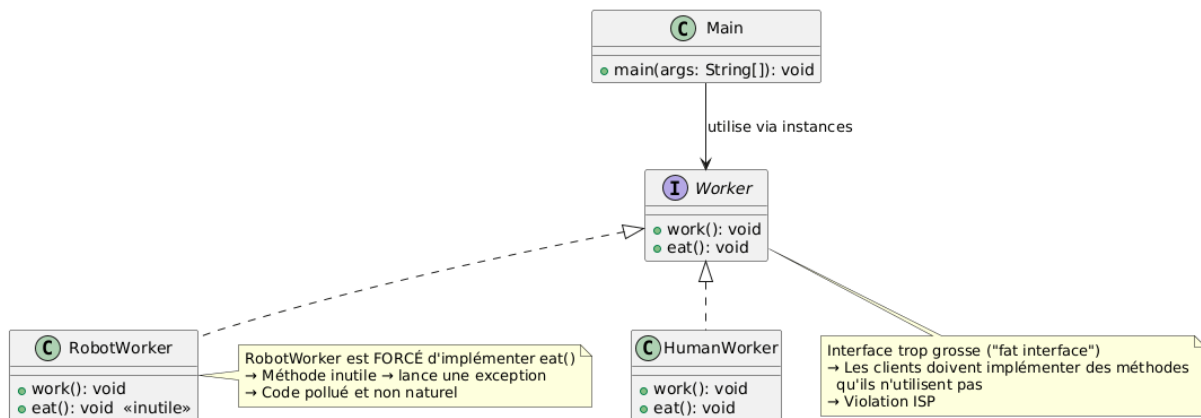


Plus d'héritage → Square n'est plus substituable à Rectangle
Mais il n'essaie plus de l'être → pas de violation LSP

4 PRINCIPE ISP (Interface Segregation Principle)

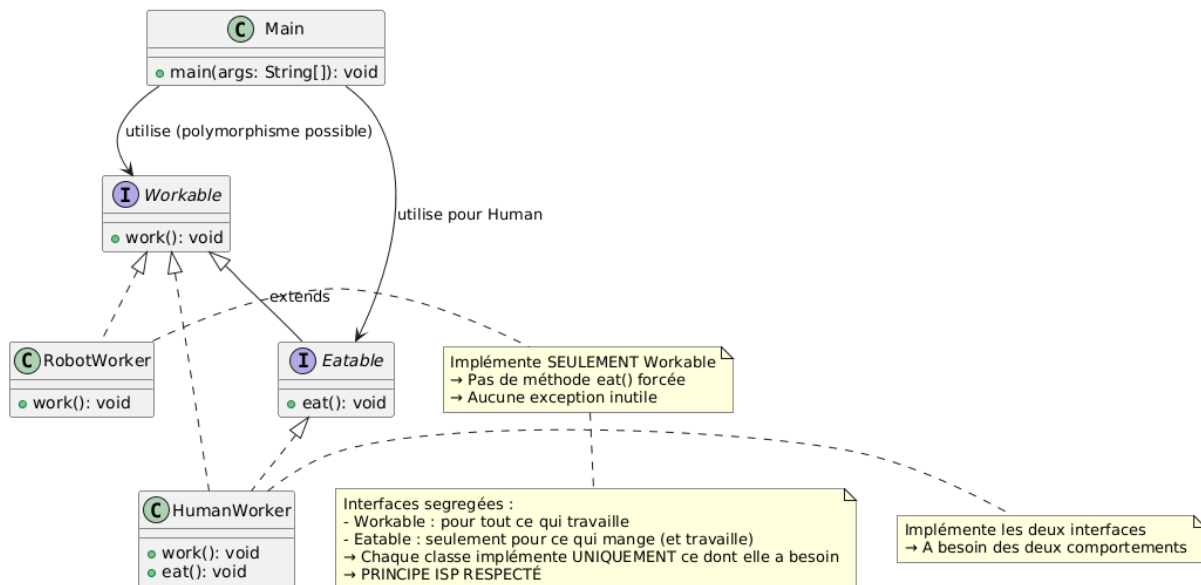
- Violation ISP - Avant Refactoring

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



- Respect ISP - Après Refactoring

Respect ISP - Après Refactoring (Interfaces petites et ciblées)

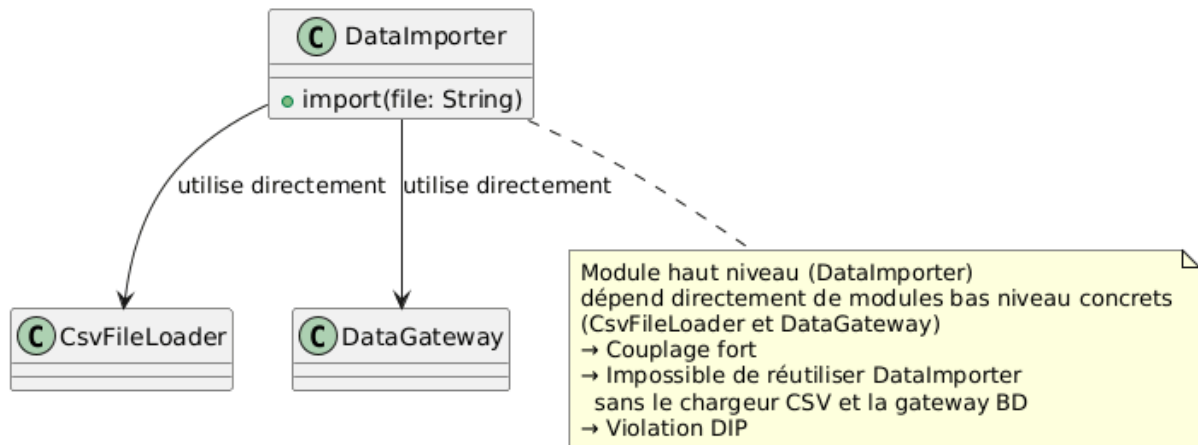


5. PRINCIPE DIP(Dependency Inversion Principle).

5.1 – exemple 1

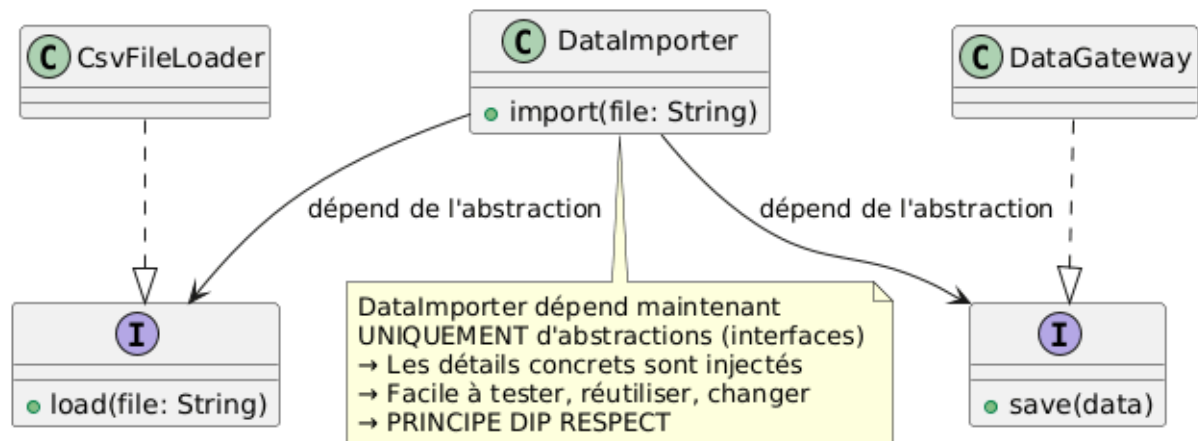
- Violation DIP - Avant Refactoring

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



- Respect DIP - Après Refactoring

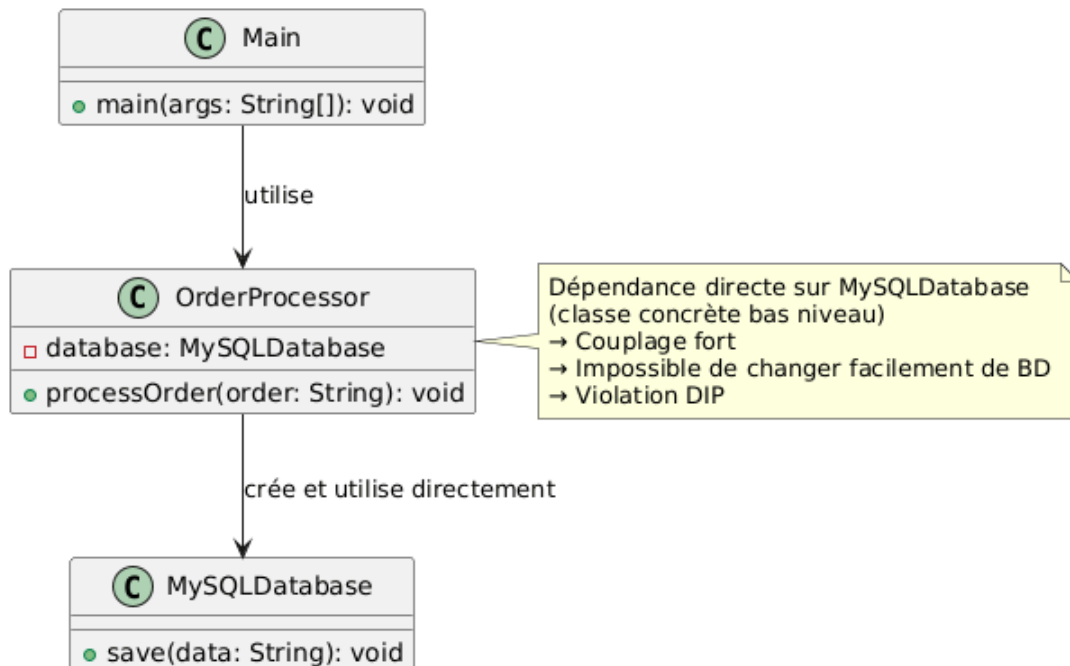
Respect DIP - Exemple 1 Refactoring (Inversion via abstractions)



5.2- Exemple 2

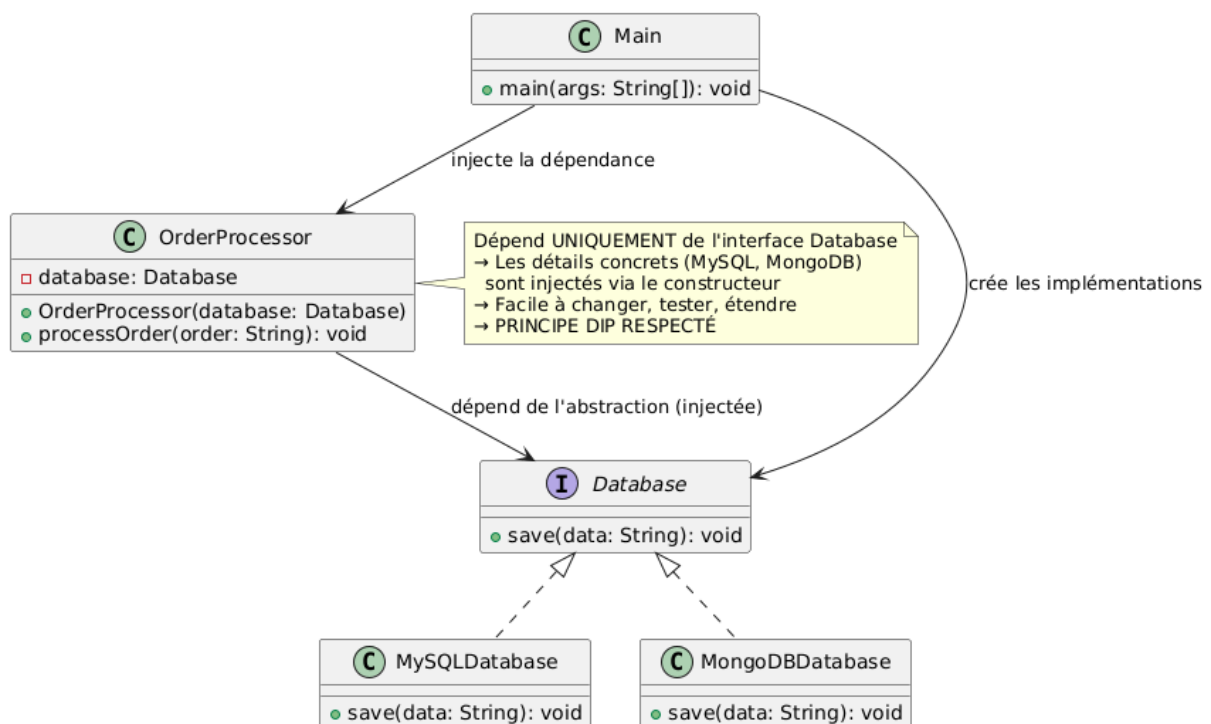
- Violation DIP - Avant Refactoring

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)



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