

ICT301 : Architecture Logicielle et Conception

Diagramme UML des principes SOLID

MONDJO NZUKAM VANELLE SANDRA

Matricule : 25I2283

ICT-4D L3

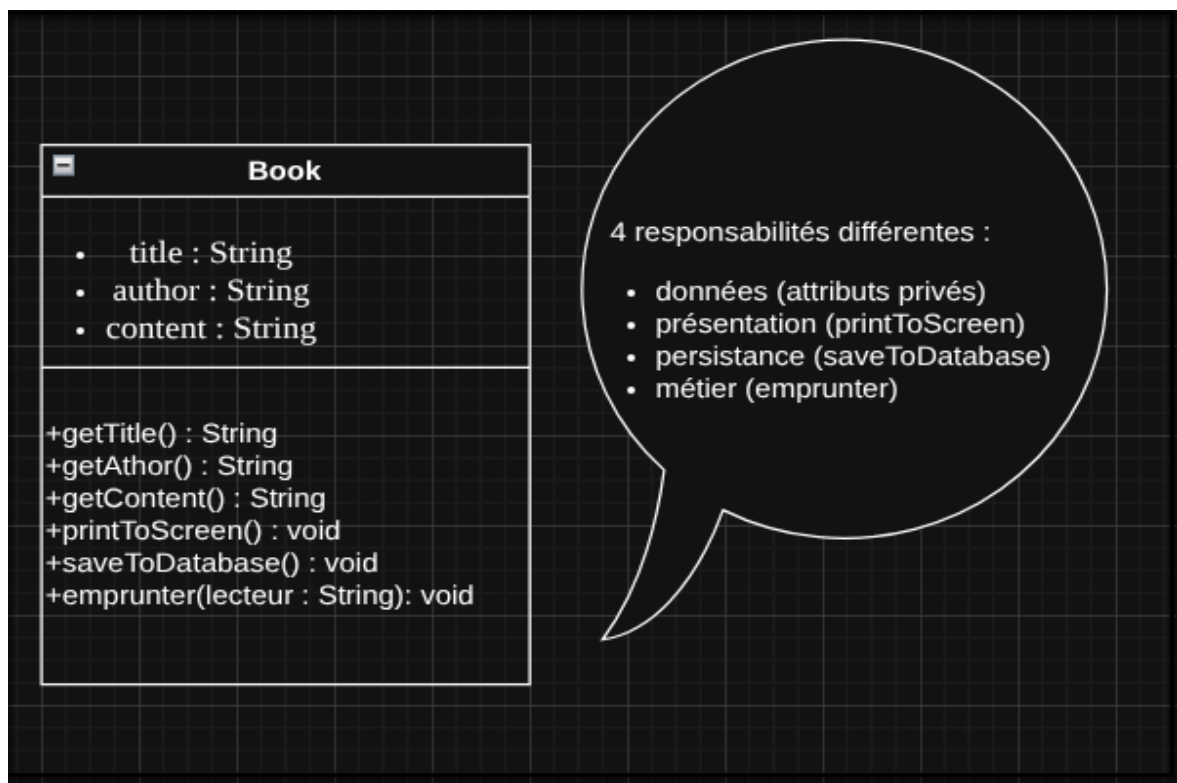
1- SRP : Single Responsibility Principle

1.1- Avant refactoring

* Problèmes

La classe Book à 4 responsabilités différentes, ce qui viole le principe SRP :

- **gestion des données** : propriétés du livre
- **Presentatiion** : printToScreen ()
- **Persistance** : saveToDatabase ()
- **Logique métier** : emprunter ()

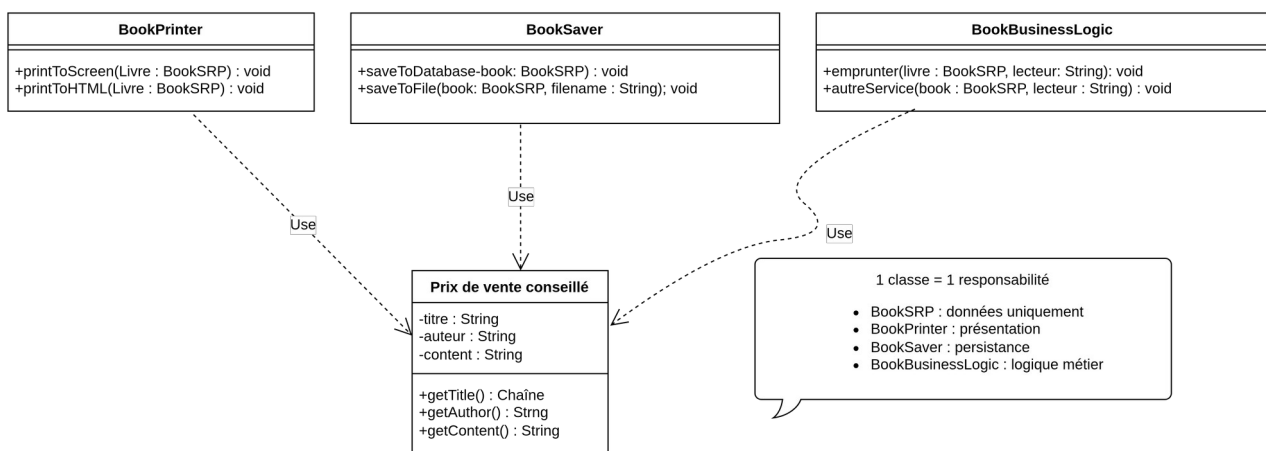


1.2- Après Refactoring

Solution

Séparation en 4 classes avec une responsabilité unique chacune :

- BookSRP : données uniquement
- BookPrinter : présentation (écran, HTML,)
- BookSaver : persistance (BD, fichier,)
- BookBusinessLogic : opérations métier (emprunter, autres services)

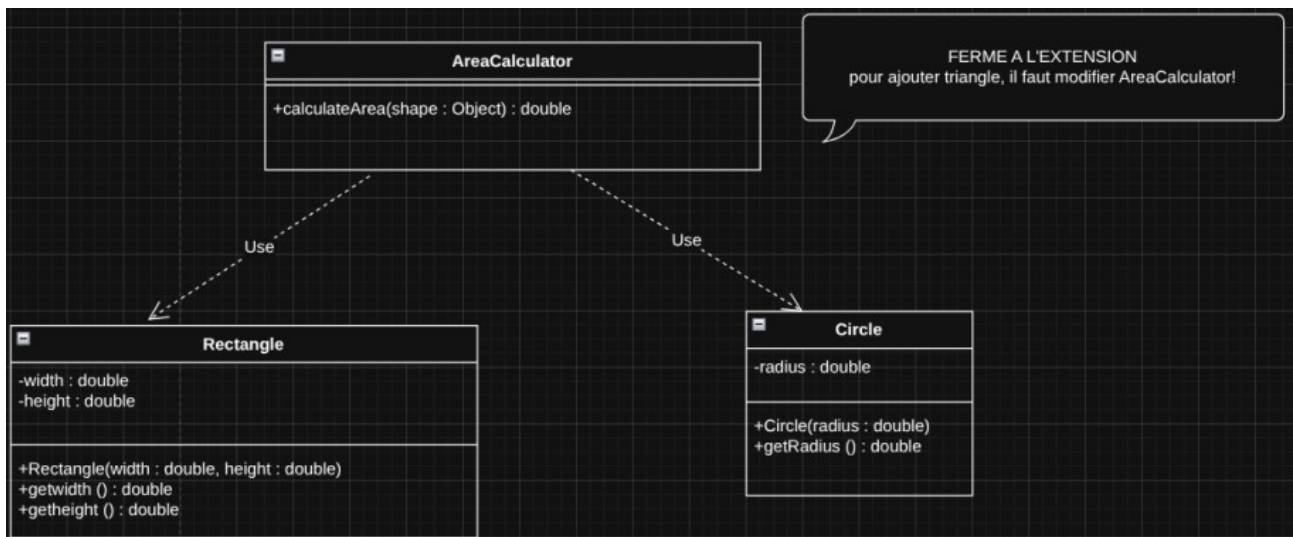


2- OCP : Open/Closed Principle

2.1- Avant refactoring

Problème

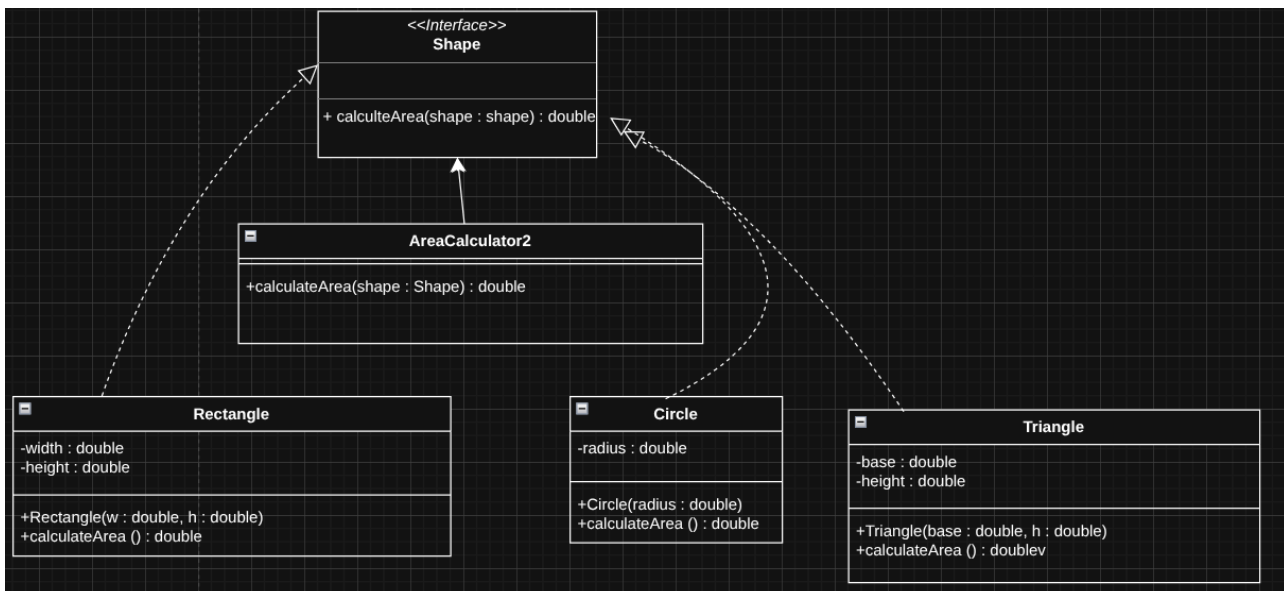
Pour ajouter une nouvelle forme il faut modifier la classe AreaCalculator



2.2- Après Refactoring

Solution

Utilisation du polymorphisme : nouvelles formes ajoutées sans modifier le code existant. Les classes sont ouvertes à l'extension (héritage) mais fermées à la modification.

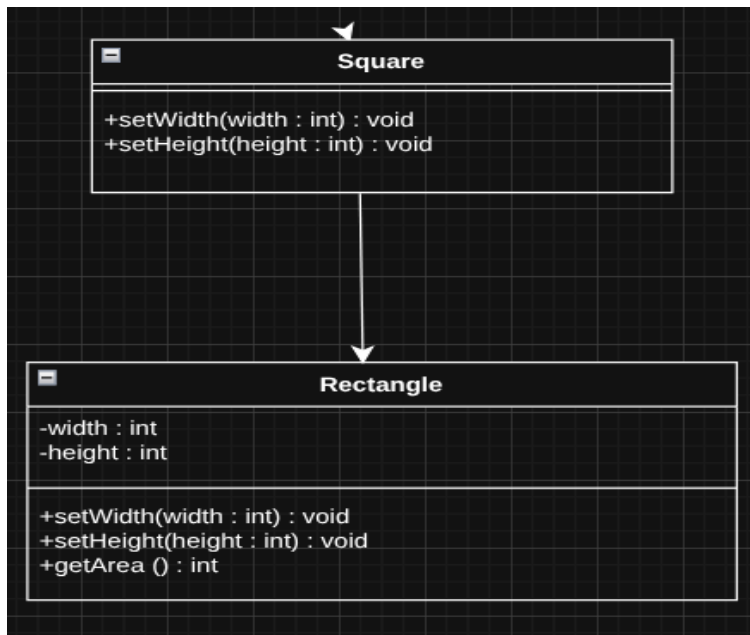


3- LSP : Liskov Substitution Principle

3.1- Avant Refactoring

Problème

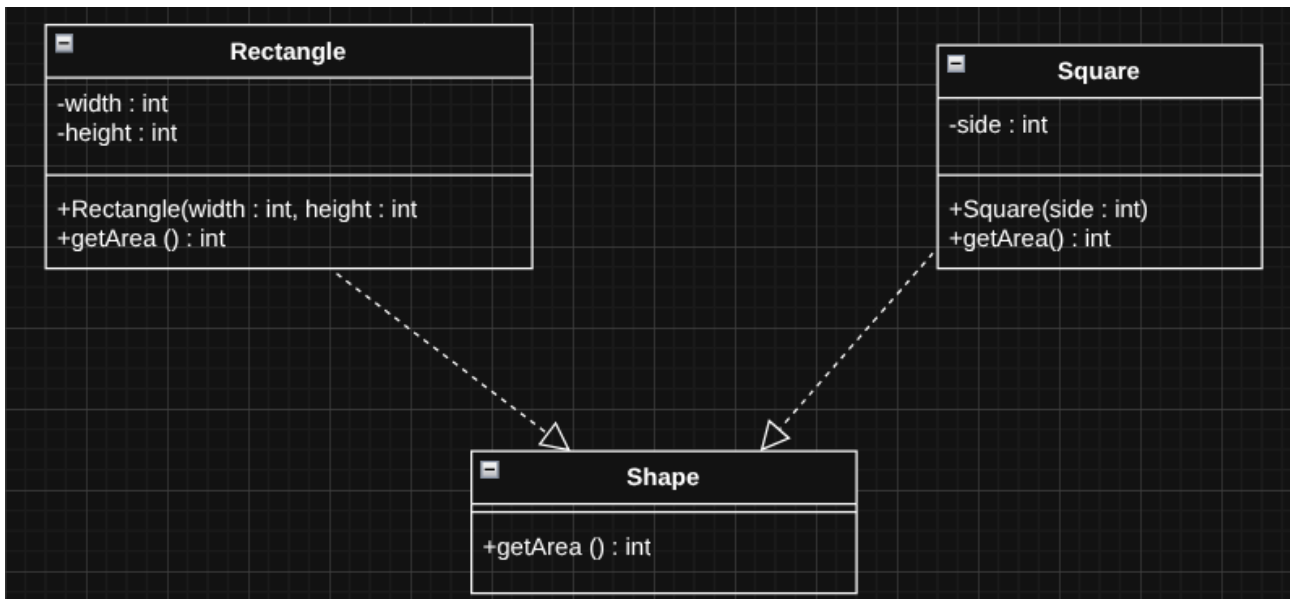
Square hérite de Rectangle mais viole le contrat de Rectangle. Pas substituable sans risques.



3.2- Après Refactoring

Solution

Rectangle et Square implémentent séparément l'interface Shape. Les objets sont substituables via l'interface sans violer de contrat.

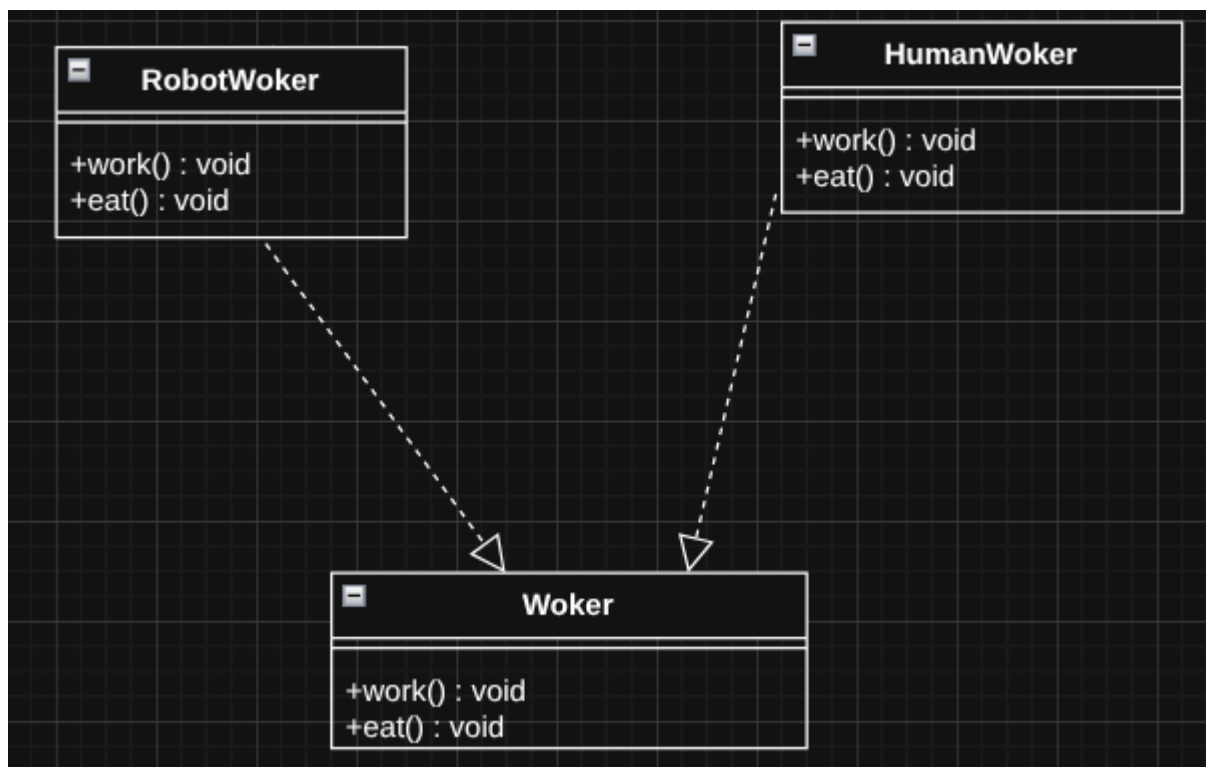


4- ISP : Interface Segregation Principle

4.1 – Avant Refactoring

Probleme

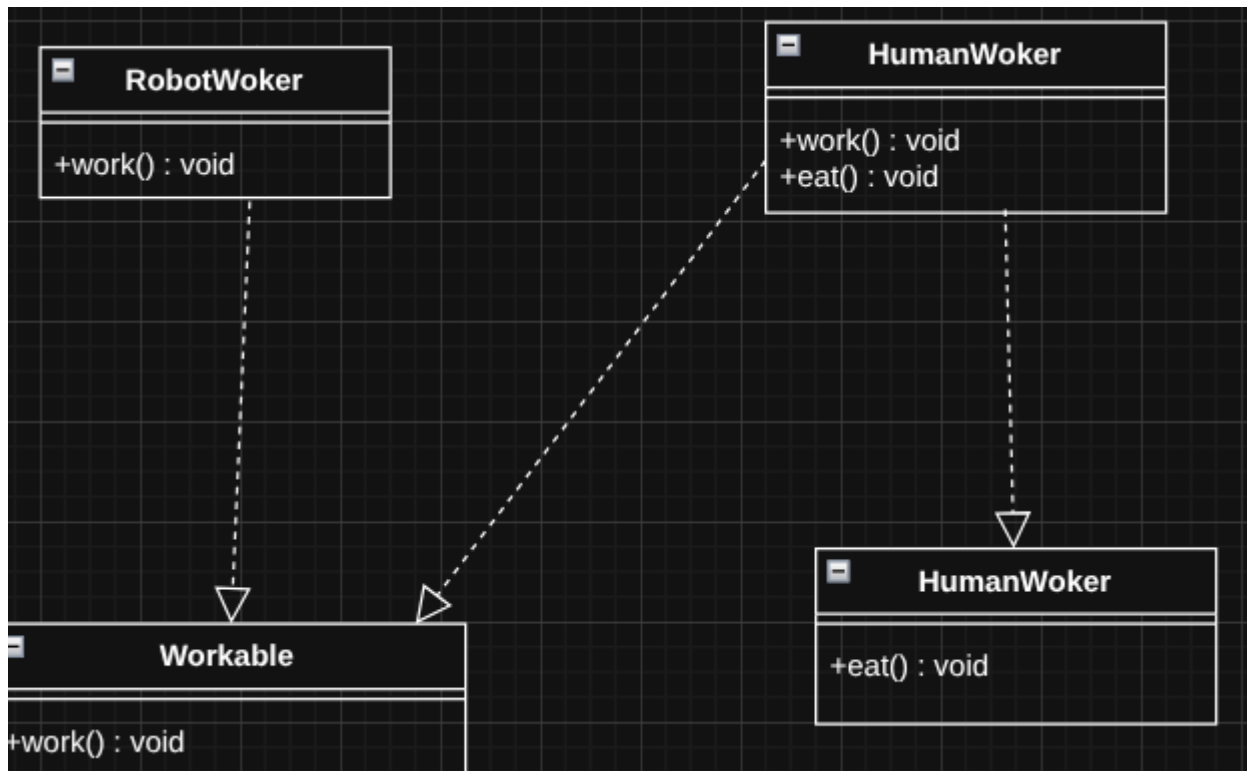
Interface Woker trp large : force RobotWoker à implémenter eat() , une methode inutile pour lui.



4.2- Après Refactoring

Solution

Interfaces séparées et ciblées : chaque classe implémente uniquement le interfaces dont elle a besoin.
Pas de dépendances inutiles à des méthodes non pertinentes.

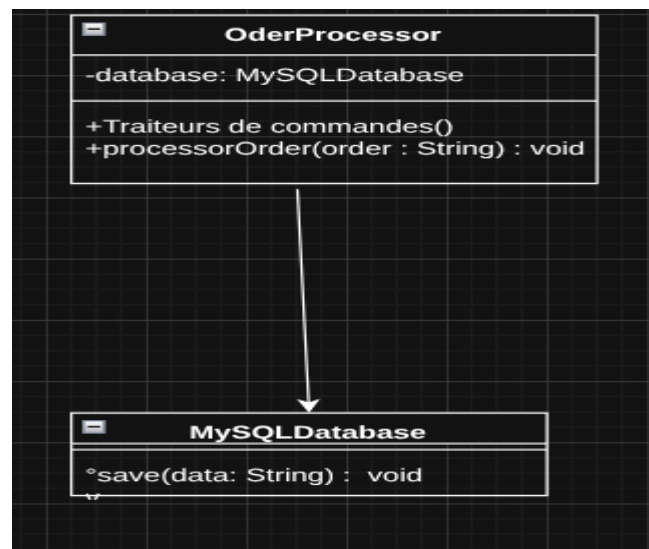


5. DIP : Dependency Inversion Principle

5.1 Avant Refactoring

Problème

OderProcessor (module haut niveau) dépend directement de MySQLDatabase (module bas niveau, implémentation concrète). Cette dépendance rend le code rigide.



5.2 – Après Refactoring

Solution

Inversion de dépendance : les deux modules (haut et bas niveau) dépendent de l'abstraction Database.
Injection de dépendance via le constructeur.

