DIAGRAMMES UML DES PRINCIPES SOLID

SOMMAIRE

DIAGRAMMES UML DES PRINCIPES SOLID

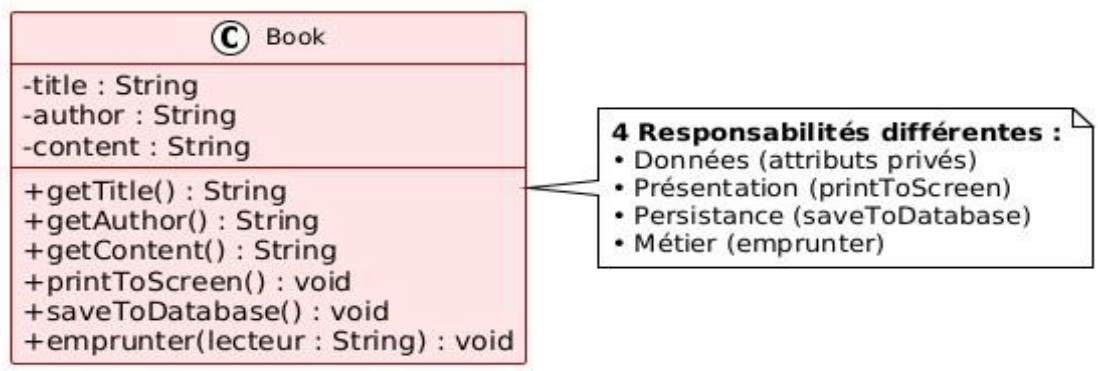
I. SRP – Single Responsability Principle

A. Avant Refactoring – Violation du SRP

Problème

La classe Book a 4 responsabilités différentes, violant le principe SRP :

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



B. Après Refactoring – Respect du SRP

Solution

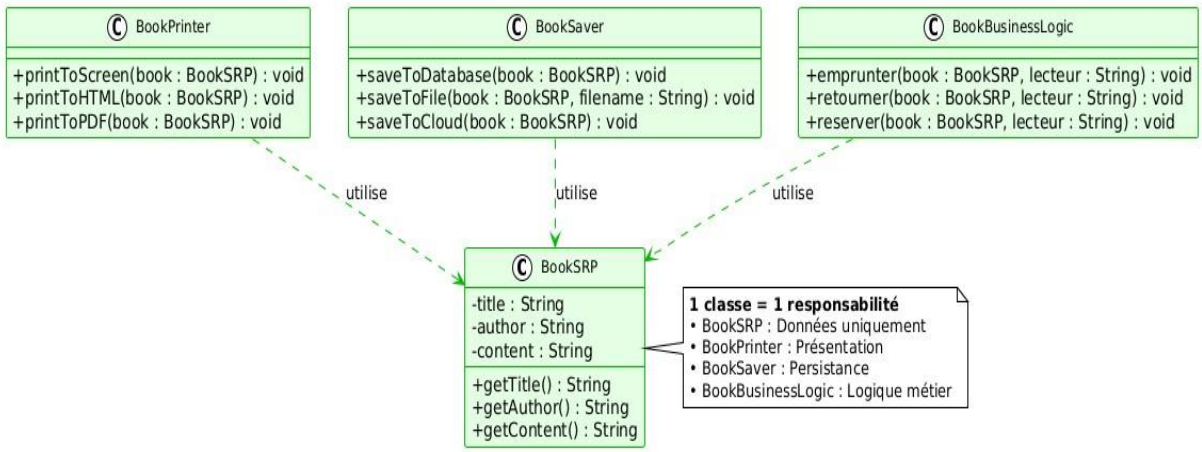
Séparation en 4 classes avec une responsabilité unique chacune :

— BookSRP : Données uniquement

— BookPrinter : Présentation (écran, HTML, PDF)

— BookSaver : Persistance (BD, fichier, cloud)

— BookBusinessLogic : Operations métier (emprunter, retourner, réserver)

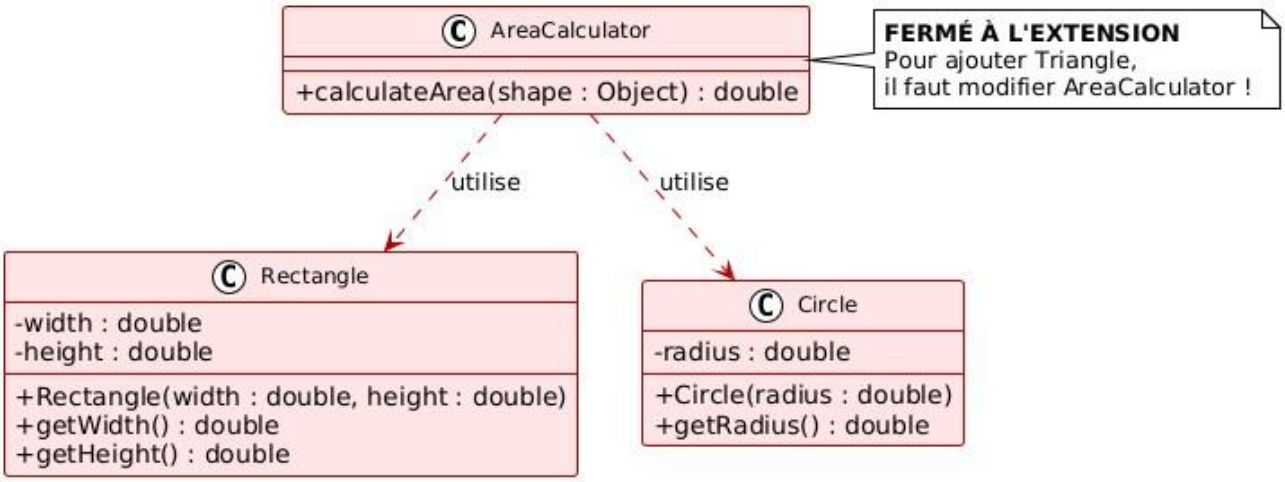


II. OCP – Open/Closed Principle

A. Avant Refactoring- Violation du OCP

Problème

Pour ajouter une nouvelle forme (ex : Triangle), il faut **modifier** la classe AreaCalculator (pas fermé à la modification).

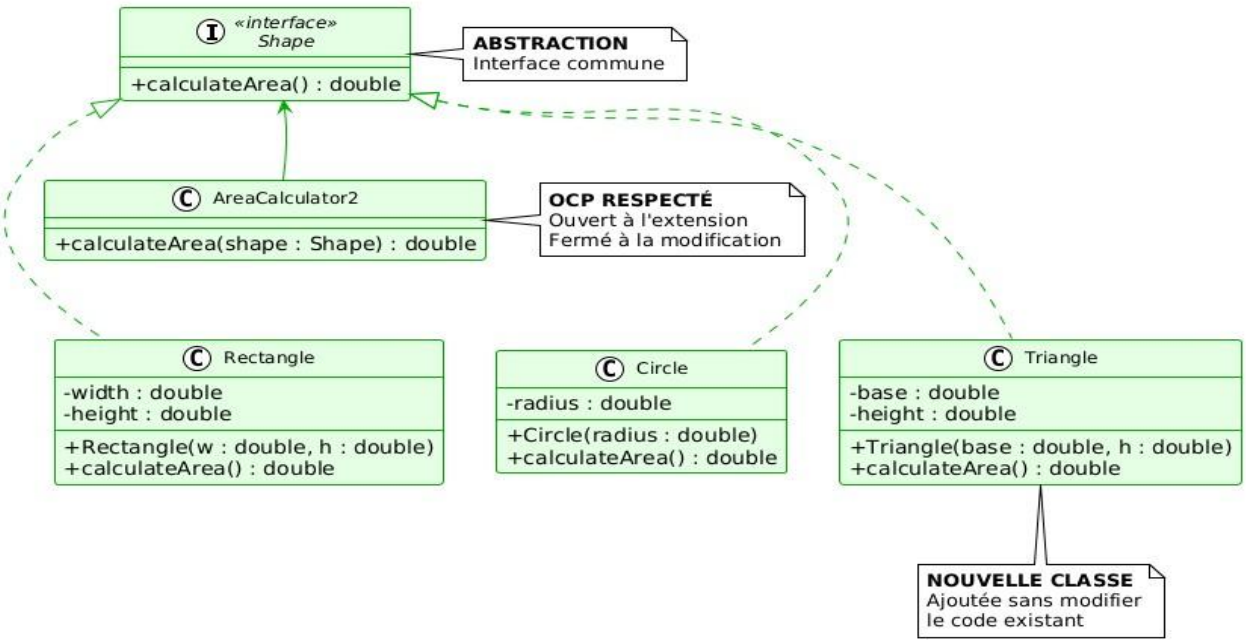


B. Après Refactoring – Respect du OCP

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.

DIAGRAMMES UML DES PRINCIPES SOLID

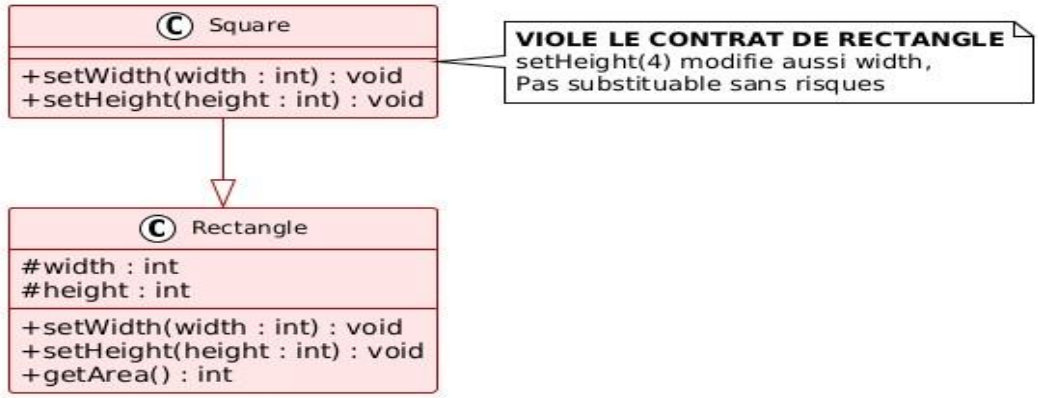


III. LSP- Liskov Substitution Principle

A. Avant Refactoring- Violation du LSP

Problème

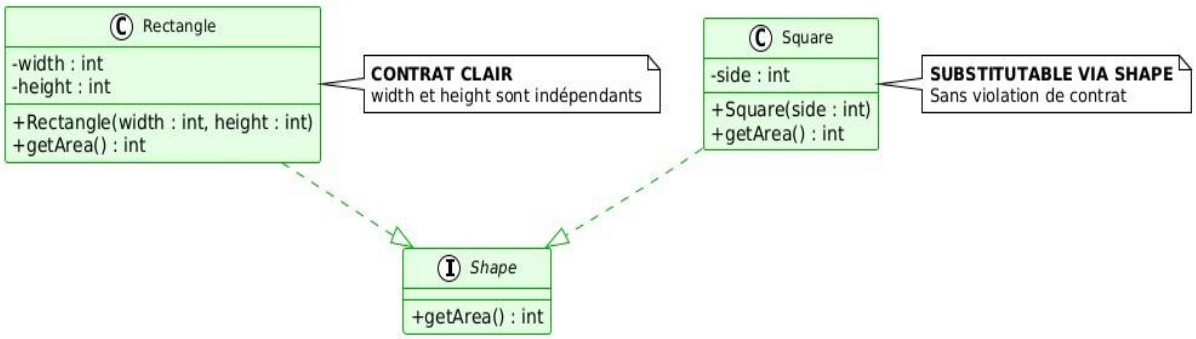
Square hérite de Rectangle mais viole le contrat de Rectangle. Pas substituable sans risques.



B. Après Refactoring- Respect du LSP

Solution

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

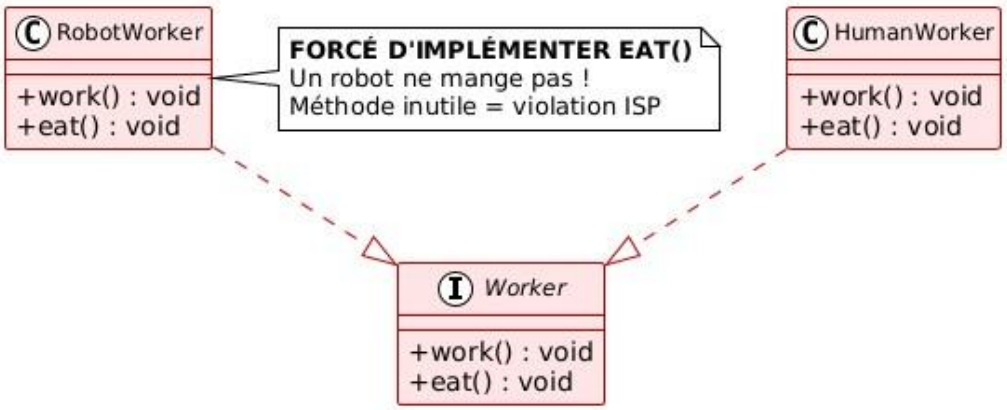


IV. ISP- Interface Segregation Principle

A. Avant Refactoring- Violation du ISP

Solution

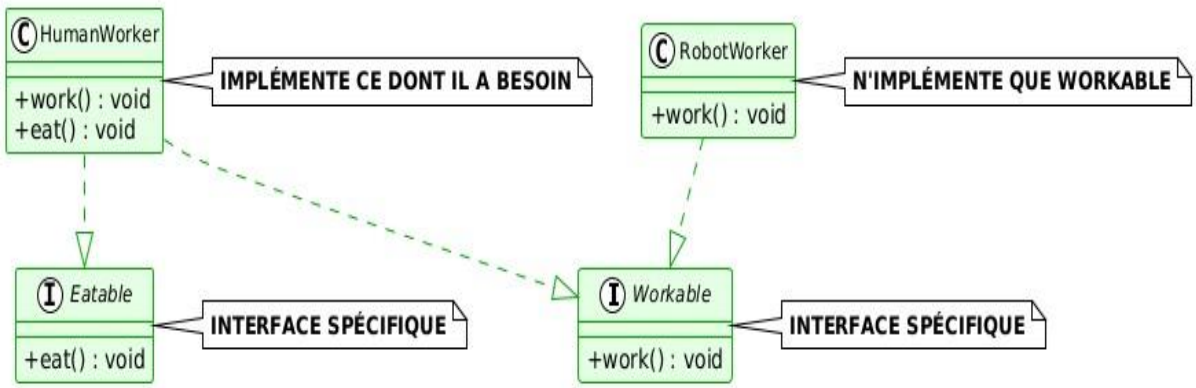
Interface Worker trop large : force RobotWorker à implémenter eat(), une méthode inutile pour lui.



B. Après Refactoring- Respect du ISP

Solution

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

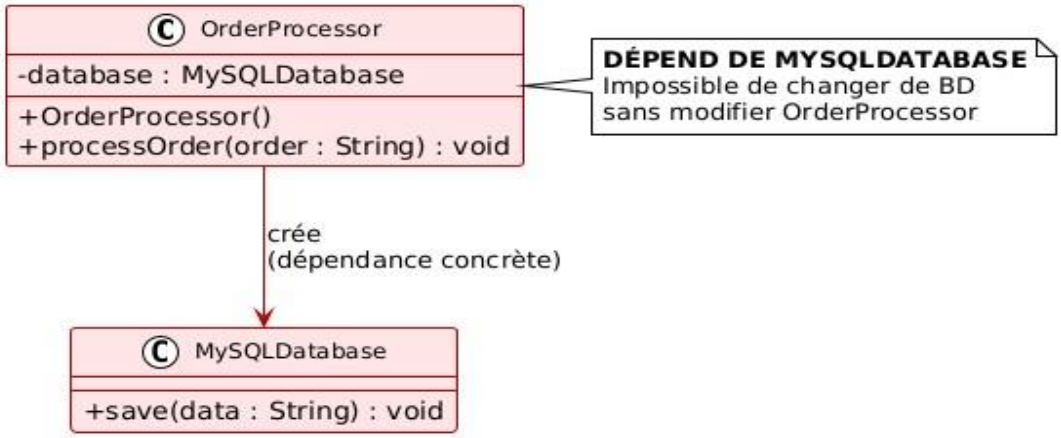


V. DIP- Dependency Inversion Principle

A. Avant Refactoring- Violation du DIP

Problème

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



B. Après Refactoring- Respect du DIP

Solution

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