Principes SOLID

ICT 301

Mahamat Ali

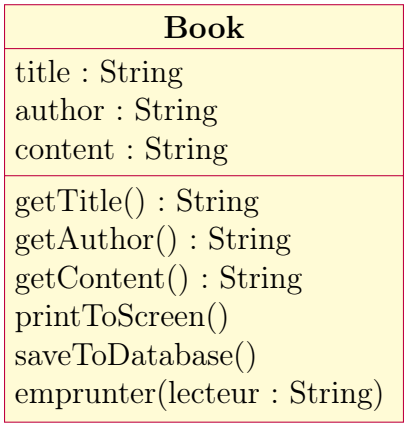
19 décembre 2025

Table des matières

1 Single Responsibility Principle (SRP)

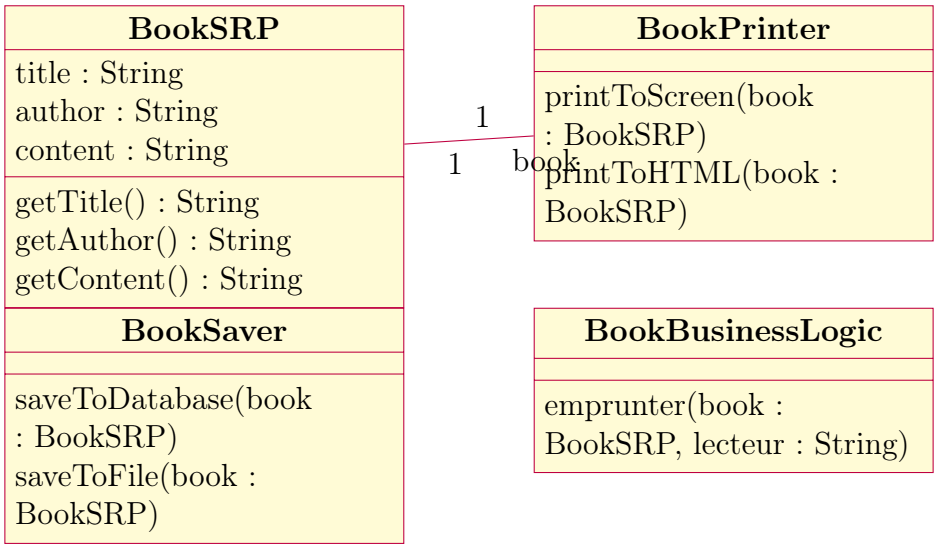
1.1 Avant refactoring

Le principe de responsabilité unique (SRP) stipule qu'une classe ne doit avoir qu'une seule raison de changer. Avant le refactoring, la classe suivante viole ce principe car elle gère plusieurs responsabilités à la fois.



1.2 Après refactoring

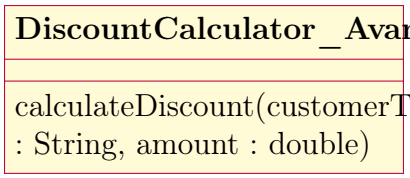
Après refactoring, chaque responsabilité est séparée dans une classe distincte, ce qui permet de respecter pleinement le principe SRP.



2 Open Closed Principle (OCP)

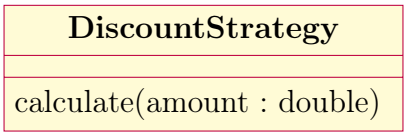
2.1 Avant refactoring

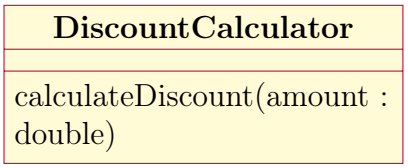
Le principe Open/Closed stipule qu'une classe doit être ouverte à l'extension mais fermée à la modification. Avant le refactoring, la classe suivante viole ce principe car chaque nouveau type de client nécessite une modification du code.

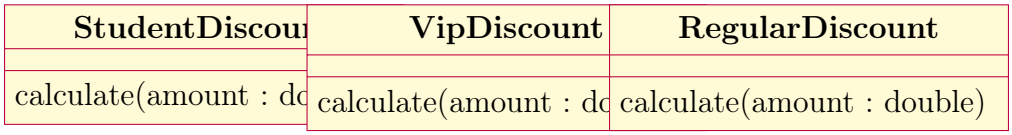


2.2 Après refactoring

Après refactoring, le comportement est rendu extensible grâce au polymorphisme. L'ajout d'un nouveau type de réduction se fait sans modifier les classes existantes.



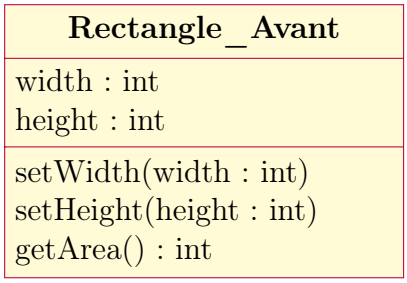


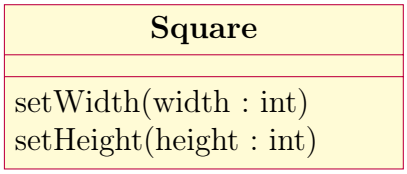


3 Liskov Substitution Principle (LSP)

3.1 Avant refactoring

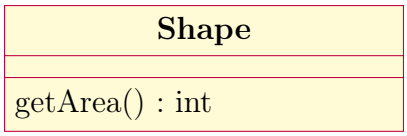
Le principe de substitution de Liskov stipule qu'une classe dérivée doit pouvoir être utilisée à la place de sa classe de base sans altérer le comportement du programme. Dans cet exemple, la classe *Square* hérite de *Rectangle* et modifie son comportement attendu, ce qui viole le principe LSP.

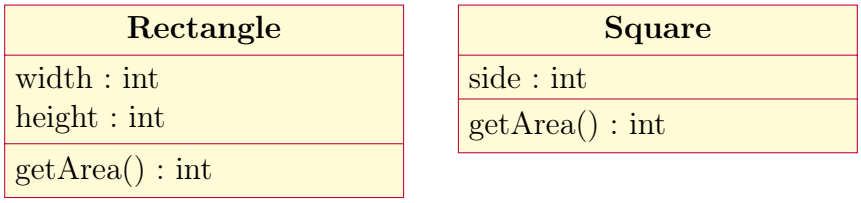




3.2 Après refactoring

Après refactoring, les formes géométriques implémentent une abstraction commune sans relation d'héritage incorrecte, ce qui garantit le respect du principe LSP.

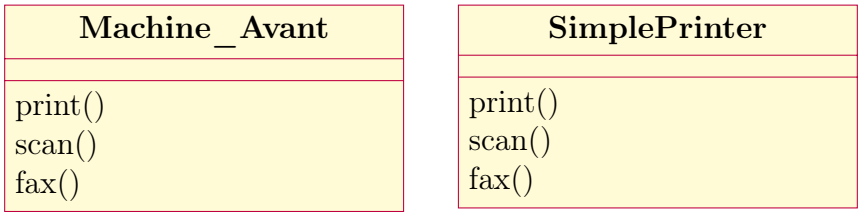




4 Interface Segregation Principle (ISP)

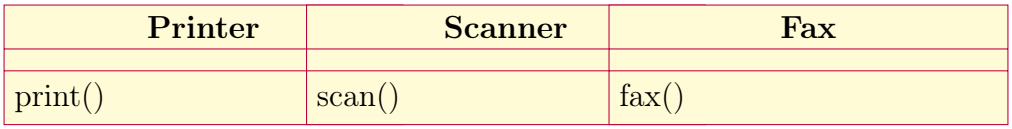
4.1 Avant refactoring

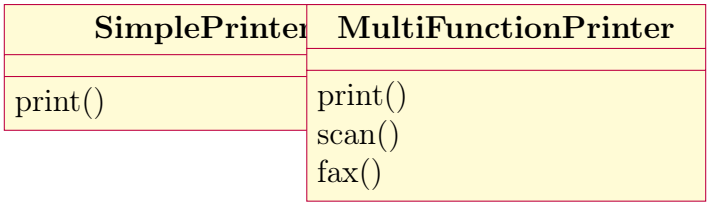
Le principe de ségrégation des interfaces stipule qu'une classe ne doit pas être forcée d'implémenter des méthodes qu'elle n'utilise pas. Dans cet exemple, une interface trop large oblige certaines classes à implémenter des méthodes inutiles.



4.2 Après refactoring

Après refactoring, les interfaces sont séparées selon leurs responsabilités, permettant à chaque classe d'implémenter uniquement les fonctionnalités nécessaires.





5 Dependency Inversion Principle (DIP)

5.1 Avant refactoring

Le principe d'inversion des dépendances stipule que les modules de haut niveau ne doivent pas dépendre des modules de bas niveau, mais d'abstractions. Dans cet exemple, le module de haut niveau dépend directement d'une classe concrète.

5.2 Après refactoring

Après refactoring, le module de haut niveau dépend d'une abstraction, ce qui permet de respecter le principe DIP.