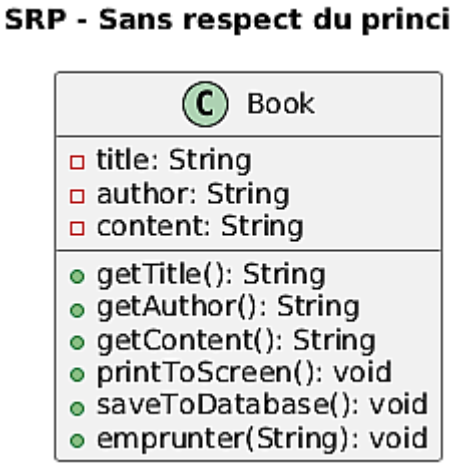
INF4067: UML et Design Patters

Rapport Fiche de TD-TP1 Principes SOLID

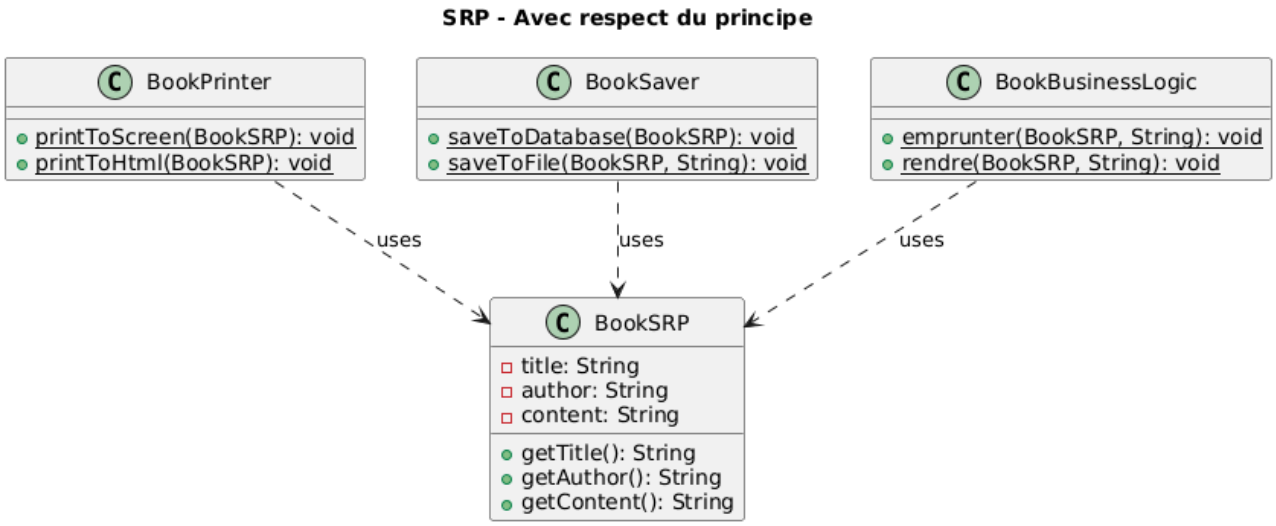
Nom : FOTSO BOPDA
Prénom : ACHILLE JORDAN
Matricule : 22T2961

I. SRP : Single Responsability Principle

Le diagramme UML correspondant au code qui ne respecte pas le SRP. On peut voir que la classe **Book** a plusieurs responsabilités (gestion des données, affichage, persistance, métier).

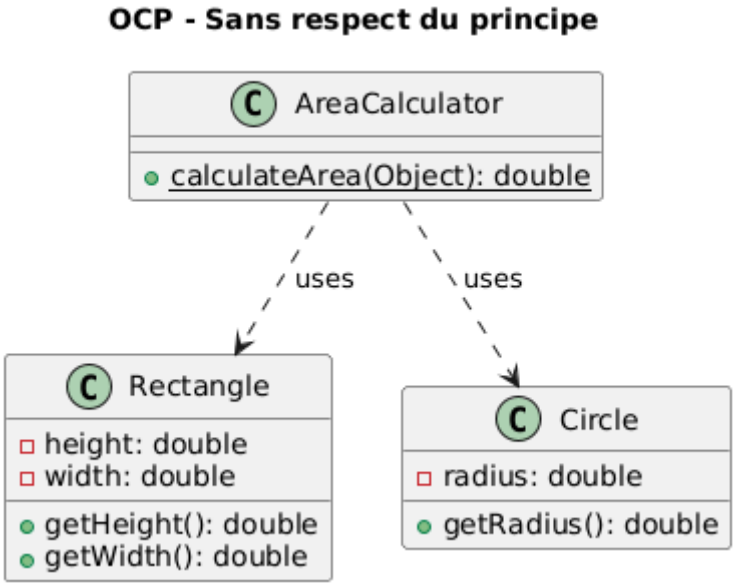


Le diagramme UML pour la version respectant le SRP. Chaque classe a maintenant une seule responsabilité, et les classes de service (**BookPrinter**, **BookSaver**, **BookBusinessLogic**) dépendent de la classe de données BookSRP.

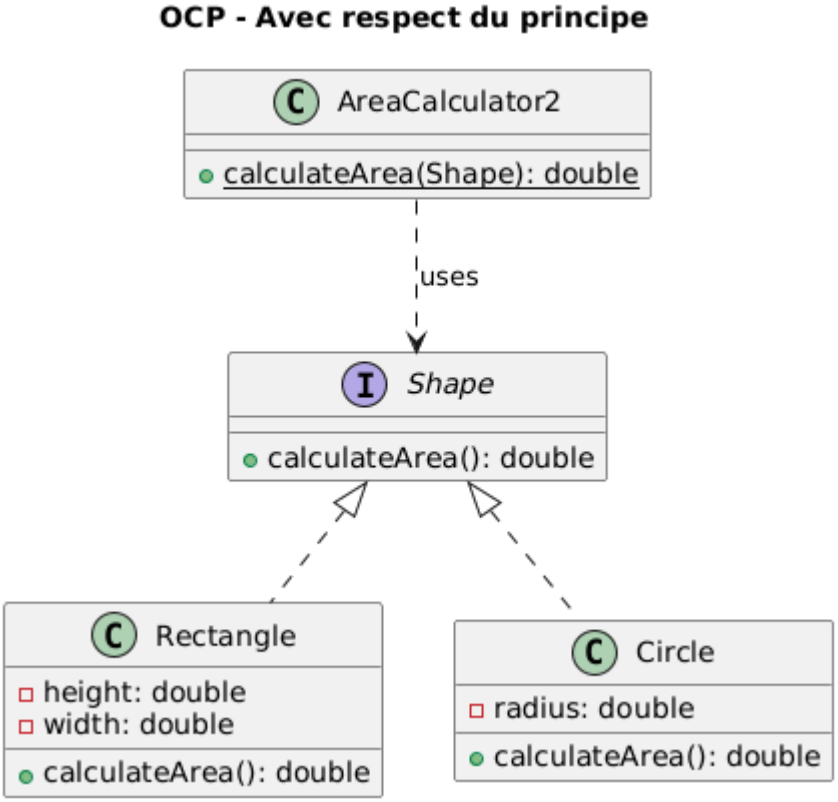


II. OCP : Open-Close Principle

Le diagramme pour la version ne respectant pas l'OCP. La classe AreaCalculator doit être modifiée à chaque fois qu'une nouvelle forme (**Shape**) est ajoutée.



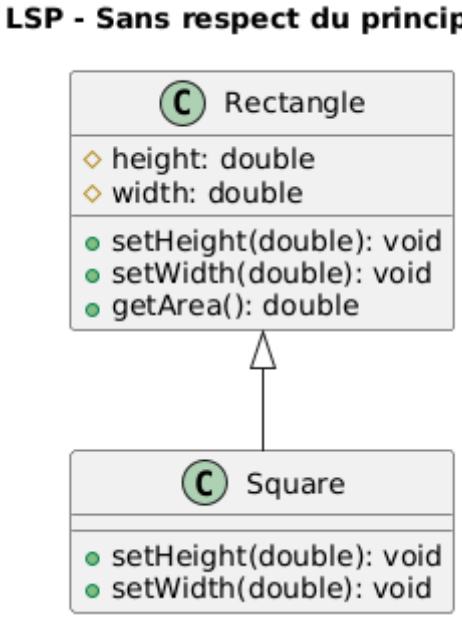
Le diagramme pour la version respectant l'OCP. Grâce à l'interface **Shape**, la classe **AreaCalculator2** n'a plus besoin d'être modifiée. Pour ajouter une nouvelle forme, il suffit de créer une nouvelle classe qui implémente **Shape**.



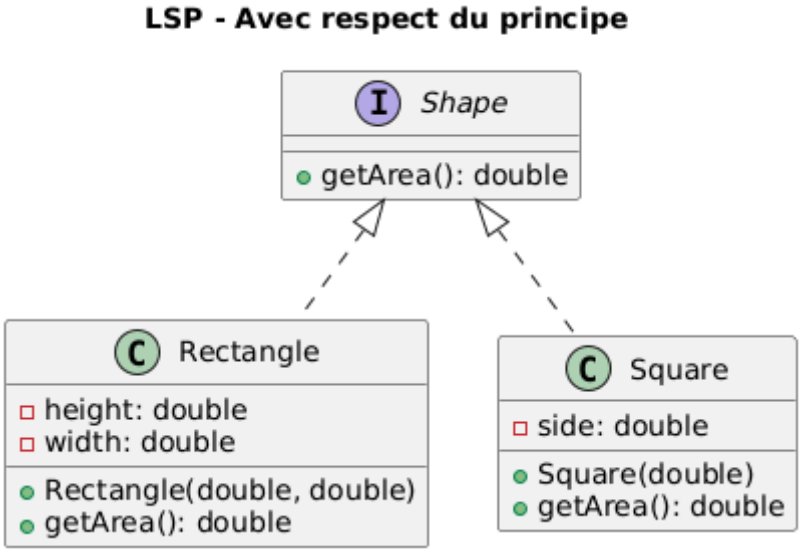
III. Liskov Substitution Principle

Le diagramme pour la version ne respectant pas le LSP. La classe **Square** hérite de **Rectangle**, mais modifie le comportement des méthodes **setHeight** et **setWidth** d'une manière qui

viole le contrat de la classe de base (changer la largeur modifie la hauteur et vice-versa), ce qui peut mener à des résultats inattendus.

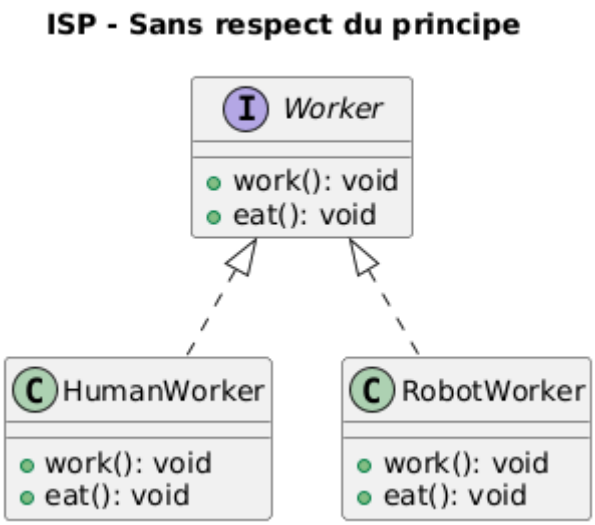


Le diagramme pour la version respectant le LSP. Au lieu d'une relation d'héritage incorrecte, **Rectangle** et **Square** implémentent tous deux une interface commune **Shape**. Cela supprime la relation "est un" trompeuse et garantit que les objets de type **Square** et **Rectangle** peuvent être utilisés de manière interchangeable là où un **Shape** est attendu, sans comportement inattendu.

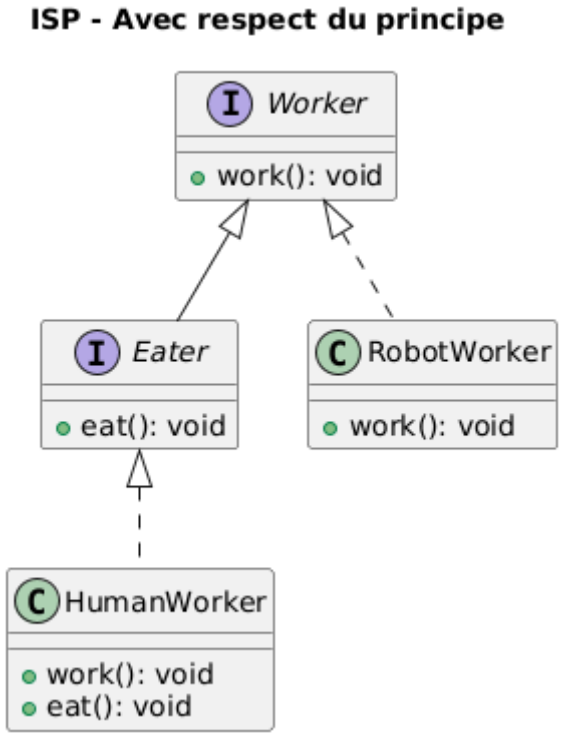


IV. Interface Segregation Principle

Le diagramme pour la version ne respectant pas l'ISP. L'interface **Worker** est "grosse" : elle force toutes les classes qui l'implémentent (comme **RobotWorker**) à fournir une implémentation pour la méthode **eat**(), même si cela n'a pas de sens pour elles.

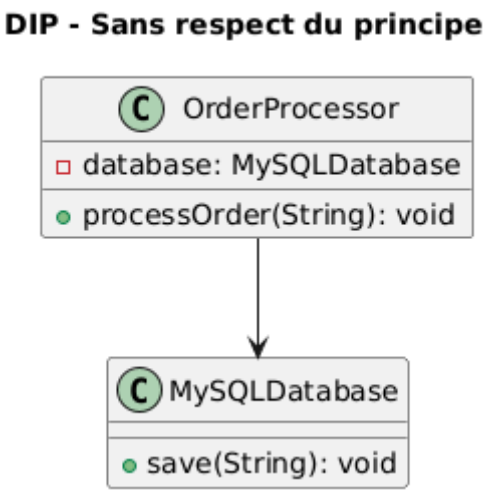


Le diagramme pour la version respectant l'ISP. L'interface **Worker** a été divisée en interfaces plus petites et plus spécifiques (**Worker**, **Eater**). Les classes implémentent uniquement les interfaces dont elles ont besoin. **RobotWorker** n'est plus obligé d'implémenter une méthode **eat**() inutile.



V. Dependency Inversion Principle

Le diagramme pour la version ne respectant pas le DIP. Le module de haut niveau **OrderProcessor** dépend directement du module de bas niveau **MySQLDatabase**. Ce couplage fort rend le code difficile à maintenir et à tester.



Le diagramme final pour la version respectant le DIP. **OrderProcessor** ne dépend plus d'une implémentation concrète, mais d'une abstraction (**Database** interface). Les modules de bas niveau (**MySQLDatabase**, **MongoBDDatabase**) dépendent également de cette même abstraction. Cela inverse la dépendance : le code de haut niveau ne dépend plus des détails du code de bas niveau.