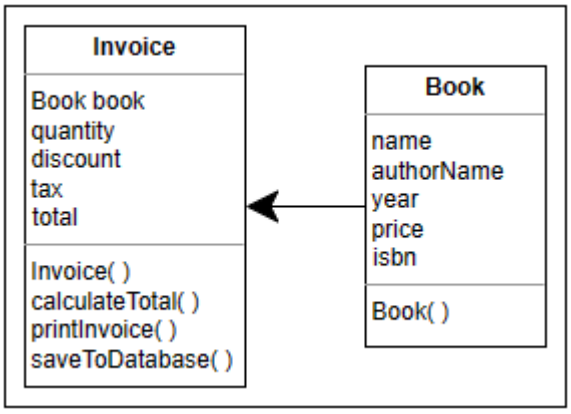
SOLID design principles

Tuesday, January 16, 2024 7:28 PM

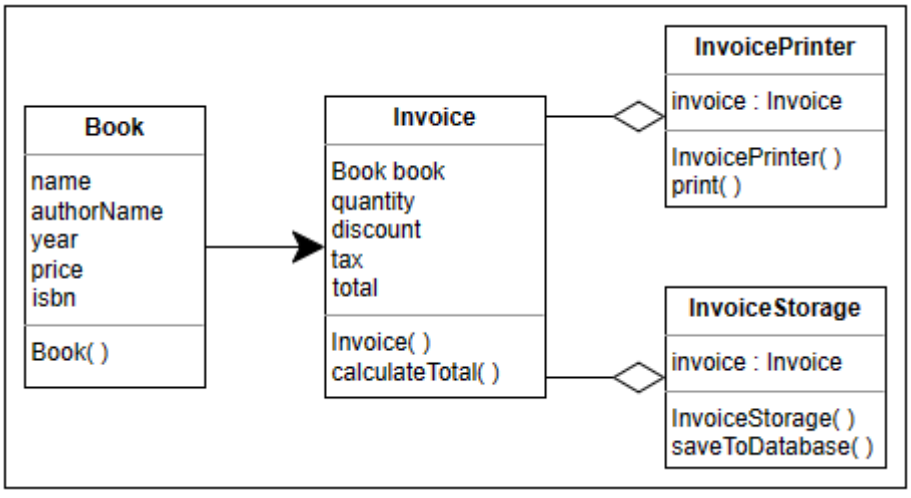
SOLID:

- S: Simple responsibility principle
  - Each class should be responsible for a single part of the system
- O: Open/closed principle
  - Software should be open for extension but close for modification
- L: Liskov substitution principle
  - Objects of superclass should be replaceable with objects of subclass without breaking the system
- I: Interface Segregation Principle
  - Make fine grained interfaces that are client specific
- D: Dependency inversion principle
  - High level modules are not dependent on the low-level modules.

Opposite example of **Single Responsibility Principle**: the invoice class has printing and saving functionality.



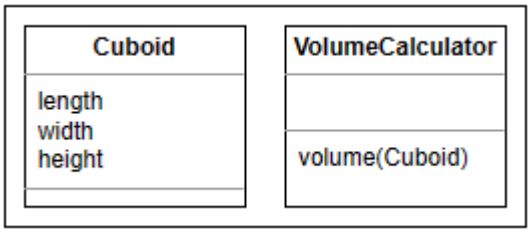
Adapted example that complies with the **Single Responsibility Principle:**

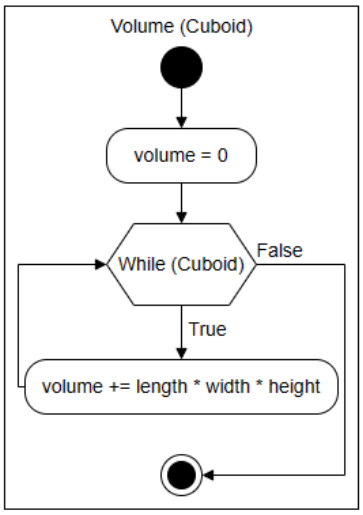


Open Closed Principle

Tuesday, January 16, 2024 7:28 PM

Opposite example of **Open Closed Principle:**

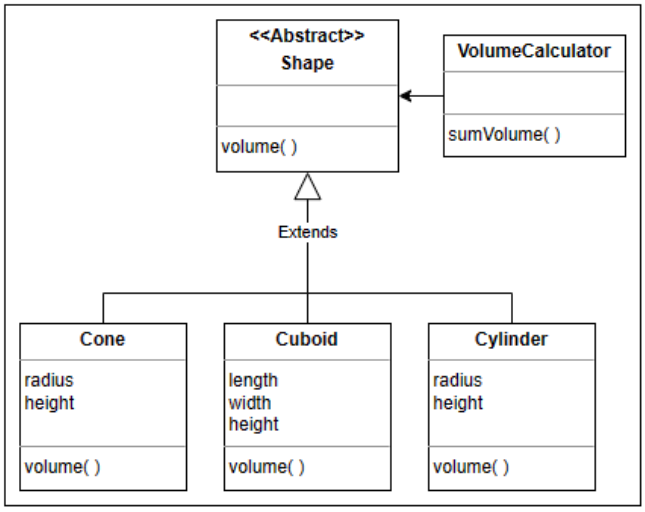




This design seems fine, but what if there are more and more boxes with different shapes. The while loop will become increasingly complicated.

We could solve this issue by using inheritance:

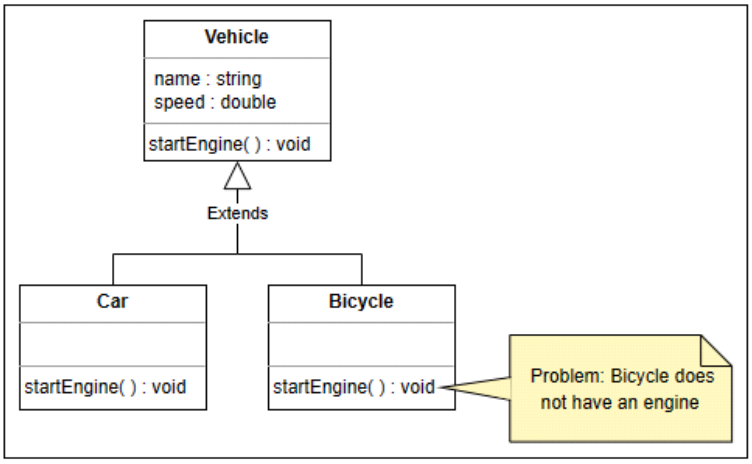
Define a superclass "shape" and use inheritance whenever there are boxes with different shapes.



Liskov Substitution Principle

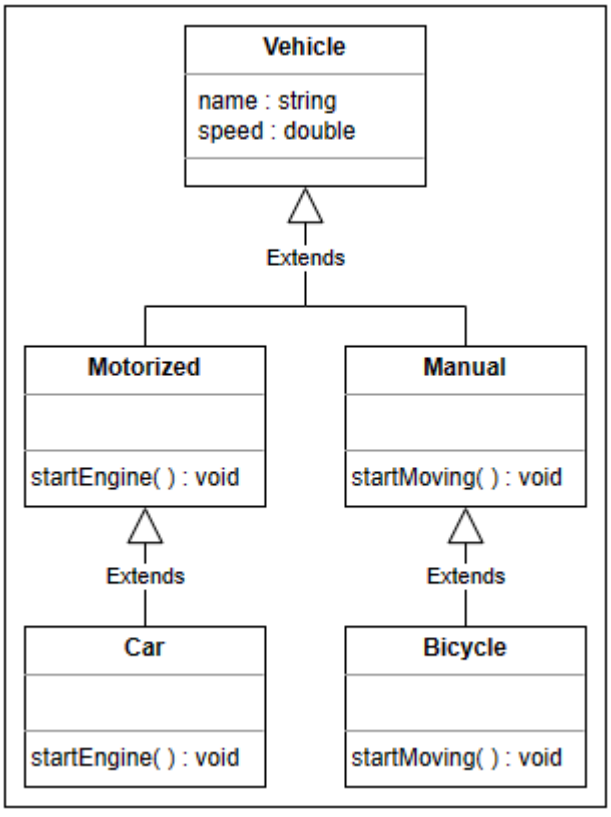
Tuesday, January 16, 2024 7:28 PM

An opposite example of **Liskov Substitution Principle:**



The derived subclass "Bicycle" does not behave the same way as its superclass

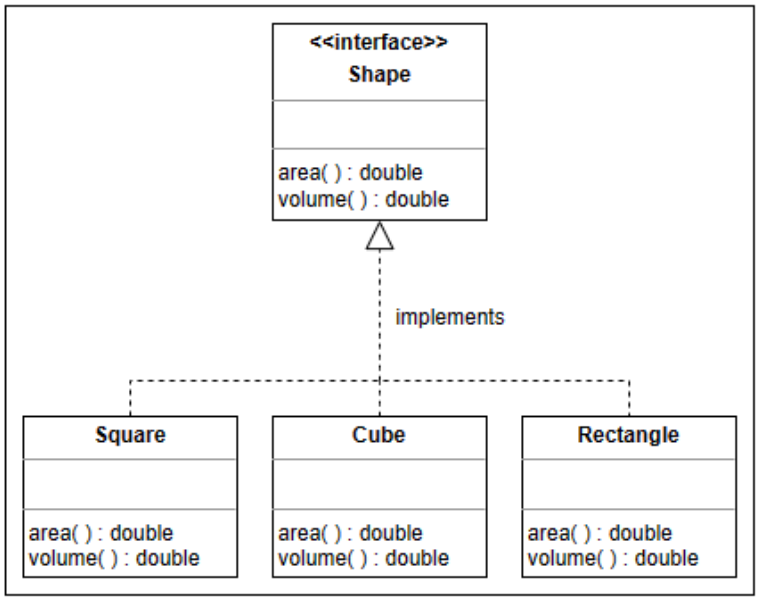
This issue can be solved by defining two additional subclasses:



Interface Segregation Principle

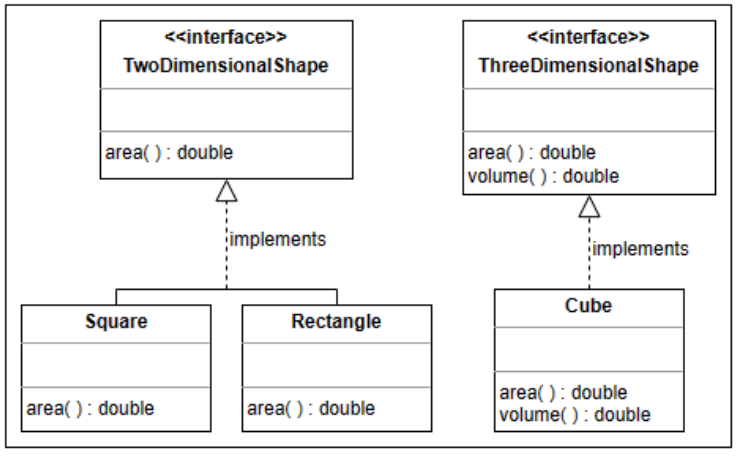
Tuesday, January 16, 2024 7:28 PM

Opposite example of **Interface Segregation Principle**:



The 2D shape does not need the method of "volume" because it does not have this attribute.

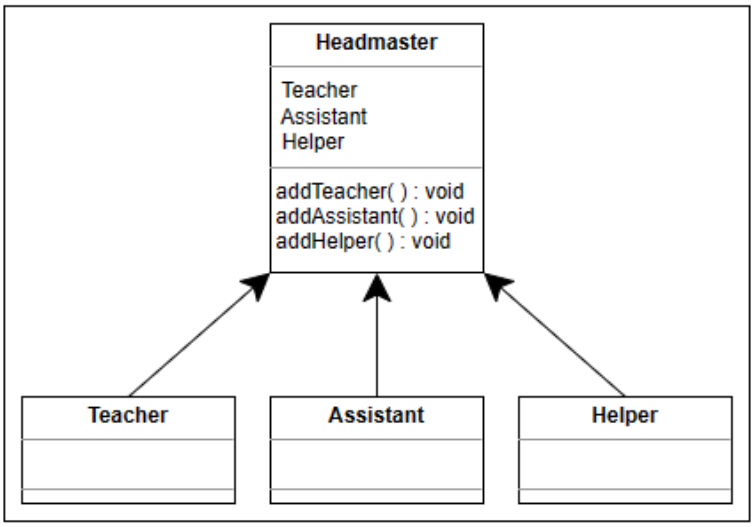
This issue is solved by adding additional superclass:



Dependency Inversion Principle

Tuesday, January 16, 2024 7:28 PM

Opposite example:



If we need to add one additional occupation (e.g., secretary), then we need to revise the entire system.

To solve this issue, we can introduce an intermediate layer called "faculty":