Author: Diego JIN

SOLID principle

Single Responsibility principle

1) A component has **limit responsibilities** to reduce changes' frequencies & scope.
2) For a component, Achieve **high cohesion** and **loose coupling**.

- **High cohesion**: All functions in component is highly related.

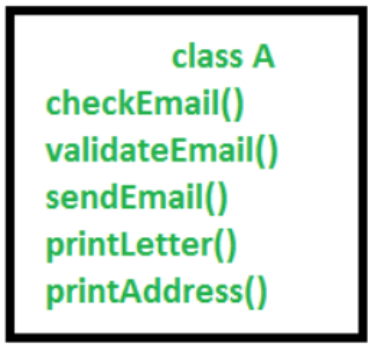


Fig: Low cohesion

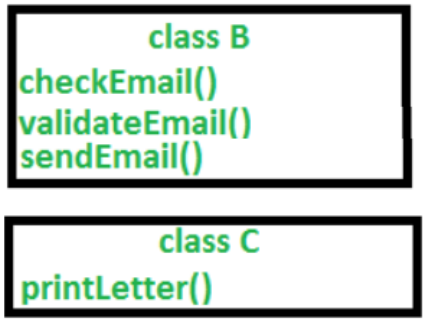
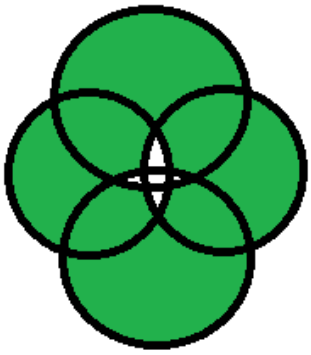
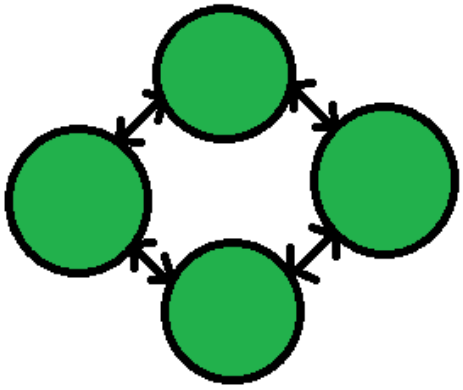


Fig: High cohesion

- **Loose coupling**: **No overlap in responsibility** of different components.



Tight coupling:
1. More Interdependency
2. More coordination
3. More information flow



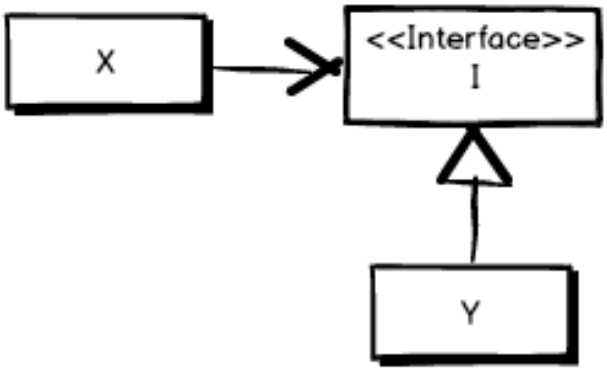
Loose coupling:
1. Less Interdependency
2. Less coordination
3. Less information flow

Author: Diego JIN

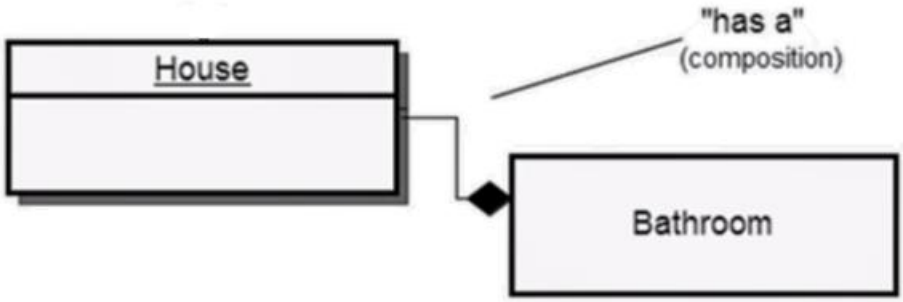
Open/Closed principle

1) A component should be able to extend new features **without modification on API.**
2) **Minimize changes on other code that uses this component.**
3) **Solution: Inheritance** and **composition**

Inheritance: Using a common interface as API, and different classes implement this interface.



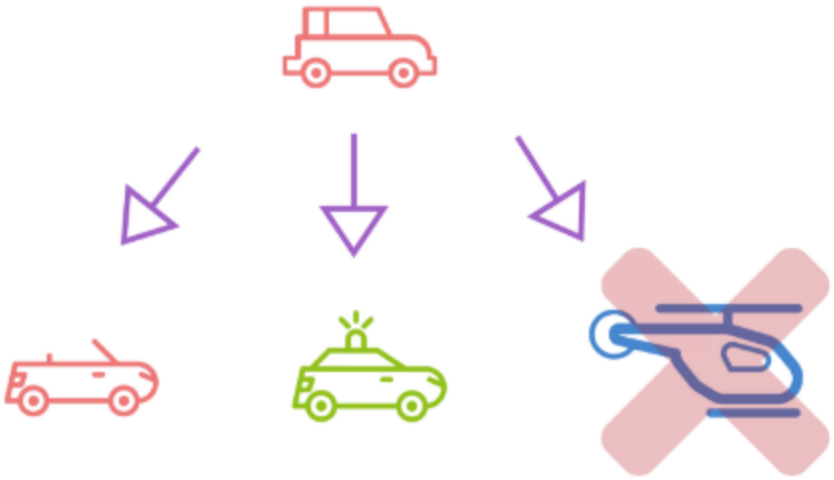
Composition: A component has an internal used component and add new feature by modifying the internal component.



Child-parent principle

A parent can be replaced by their children subtype's instance **correctly**.

- Subtype can be used in anywhere of their parent type used without any modification.
- In practice, using **unit test** to guarantee the correctness.

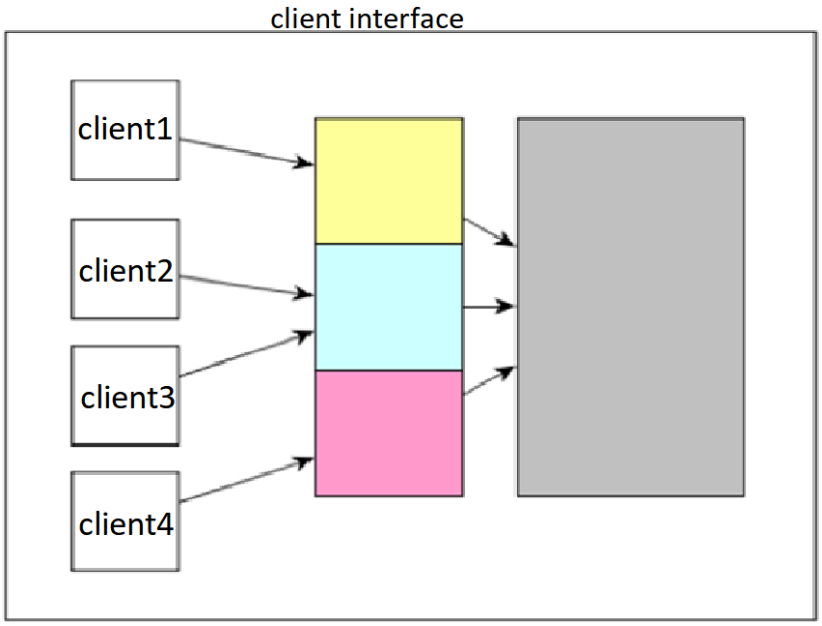


Author: Diego JIN

Interface segregation principle

The client should depend on an abstraction interface, which only includes methods used by client.

- Minimize impact of changes by hiding unused method from client.
- Separate a large interface into several small pieces.



Dependency inversion principle

- Entities must depend on abstractions, not on detail implementation.
- High-level module must only depend on abstraction interface of low-level module.
- Avoid **changes on details** breaking the code.