SOLID Principels

O - Open closed

- Open for extension
  - software module, or class can be extended to make it behave in new and different ways.
- Close for modifications
  - software module, say a class can be extended to make it behave in new and different ways.
  - source code of such a module remains unchanged.
    - Java is abstraction. You can create abstractions (Java interfaces and abstract classes) that are fixed and yet represent an unbounded group of possible behaviors through concrete subclasses.

S - Single responsability - SRP

- A component/module should have only one reason to change -> end up doing one thing
  - This means that a division of concerns is performed in the program, and the methods for every concern should be completely encapsulated by a single class.
    - this approach contributes to the high cohesion – since methods related to the same concern (same part of the functionality) will be members of the same class, and robustness – since this reduces the possibility of error. F

Liskov Substitution

- if you have objects in your program we should be able to replace them with instances of subtypes
  WITHOUT ALTERING THE CORECTNESS OF THE PROGRAM

Interface Segregation

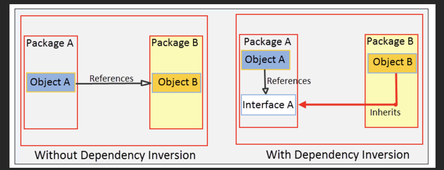
- Clients should not be forced to depend on methods that they do not use". The term "Clients" refers to the implementing classes of an interface.
  - interface should not be bloated with methods that implementing classes don't require
    - Advocates segregating a "fat interface" into smaller and highly cohesive interfaces, known as "role interfaces". Each "role interface" declares one or more methods for specific behavior.

implementation classes now implem...

OOP terms / traits of soft

- robustness
- cohesion
  - high
    - robustness
    - reliability
    - reusability
  - low
- coupling

Dependency Inversion



- Software
  - one class should depend on abstractions not on concrete classes
    - ElectricPowerSwitch — interface Switch
    - LightBulb — interface Switchable

```
public class ElectricPowerSwitch implements Switch {
    public Switchable client;
    public boolean on;
    public ElectricPowerSwitch(Switchable client) {
        this.client = client;
        this.on = false;
    }
```

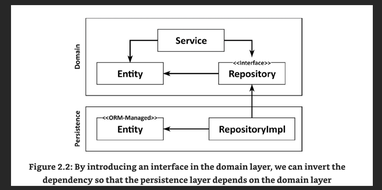


Figure 2.2: By introducing an interface in the domain layer, we can invert the dependency so that the persistence layer depends on the domain layer

- Architecture
  - we can turn around (invert) the direction of any dependency
    - Entities — Domain Layer — domain code will change state of entities
    - Interface for repository — Domain Layer
    - Implementation for repository — Persistence Layer