CONFIDENTIAL & RESTRICTED

SOLID Principles Of Object-Oriented Programming

Source

The SOLID principles are five principles of Object-Oriented class design. They are a set of rules and best practices to follow while designing a class structure.

- **S**ingle Responsibility Principle
- **O**pen-Closed Principle
- **L**iskov Substitution Principle
- **I**nterface Segregation Principle
- **D**ependency Inversion Principle

Single Responsibility Principle

It states that a class should do one thing and therefore it should have only a single reason to change. Only one potential change in the software’s specification should be able to affect the specification of the class. For example, if a class is a data container, like a Book class, it should change only when we change the data model.

CONFIDENTIAL & RESTRICTED

SINGLE RESPONSIBILITY PRINCIPLE
Just Because You Can, Doesn't Mean You Should

Why to follow SRP?

- Many different teams can work on the same project and edit the same class for numerous reasons, this could lead to incompatible modules
- It makes version controlling a lot easier
- Fewer merge conflicts will appear and conflicts that do exist will be easier to resolve

Common Pitfalls

-Do not mix business logic with persistence logic (saving to database, making an API call or other stuff related to persistence)

Open-Closed Principle

It states that classes should be open for extension and closed to modification.

In simple words, we should be able to add new functionality without changing the existing code for the class. We should avoid touching the tested and reliable production code whenever possible.

This principle is usually achieved with the help of interfaces and abstract classes.

Liskov Substitution Principle

It states that subclasses should be substitutable for their base classes. In simple words, given a class B which is a subclass of class A, we should be able to pass an object of class B to any method that expects an object of class A and the method should not give any weird output in that case.

LISKOV SUBSTITUTION PRINCIPLE
If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction

This is the expected behavior because a child class should inherit everything that the superclass has. Liskov’s principle is easy to understand bu hard to detect in code.

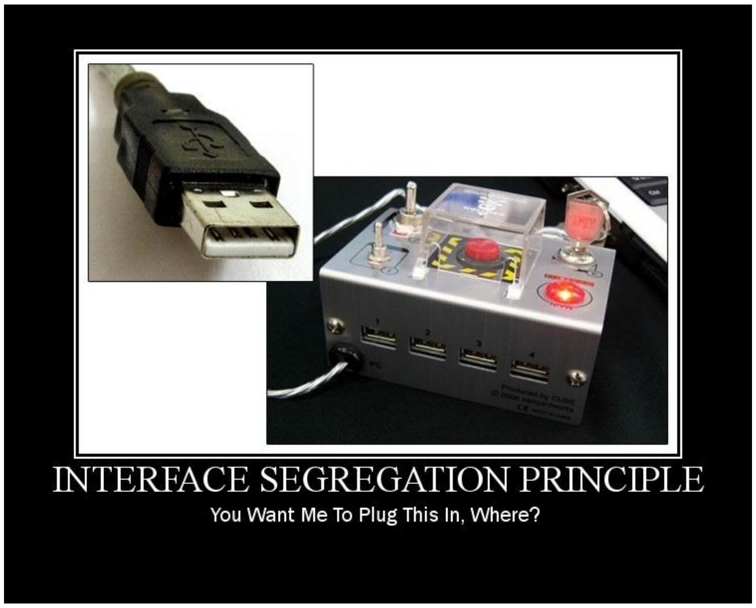
Read this.

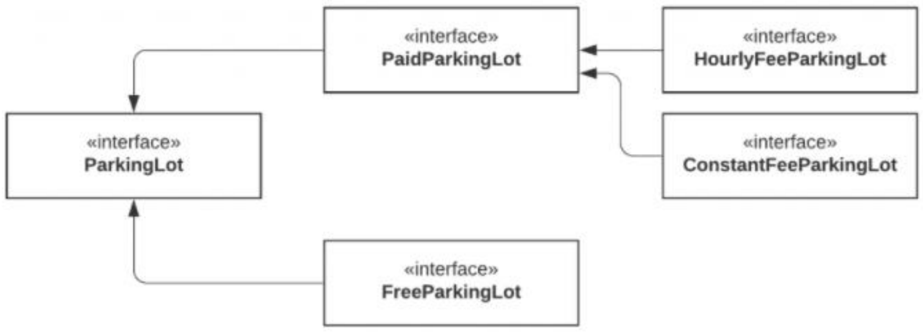
Interface Segregation Principle

Segregation means keeping things separated, and the Interface Segregation Principle is about separating the interfaces.

It states that many client-specific interfaces are better than one general-purpose interface. Clients should not be forced to implement a function they do not need.

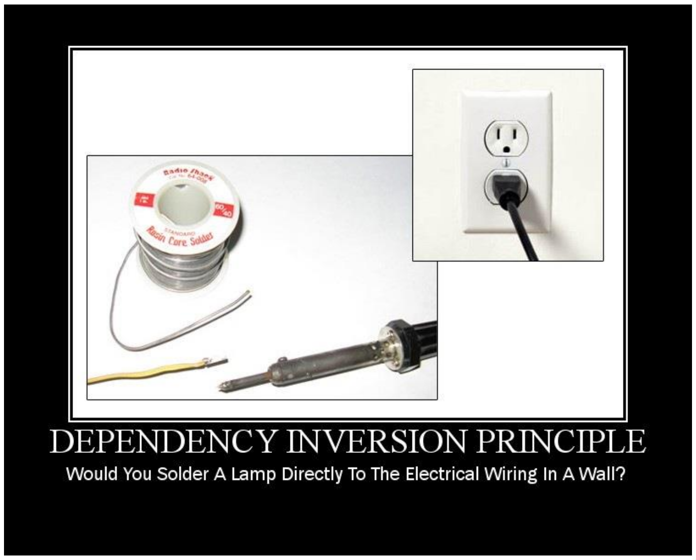
CONFIDENTIAL & RESTRICTED


INTERFACE SEGREGATION PRINCIPLE
You Want Me To Plug This In, Where?


«interface» PaidParkingLot
«interface» HourlyFeeParkingLot
«interface» ParkingLot
«interface» ConstantFeeParkingLot
«interface» FreeParkingLot

Dependency Inversion Principle

It states that our classes should depend upon interfaces or abstract classes instead of concrete classes and functions.


DEPENDENCY INVERSION PRINCIPLE
Would You Solder A Lamp Directly To The Electrical Wiring In A Wall?

CONFIDENTIAL & RESTRICTED

“If the OCP states the goal of OO architecture, the DIP states the primary mechanism” – Uncle Bob

We want our classes to be open to extension, so reorganize dependencies to depend on interfaces instead of concrete classes.