Design Principles

Intro

Problem

How to write code that:

- easy to maintain
- easy to read
- very flexible

Solution

Use the experience of other software engineers:

Design Principles

Common Design Principles

DRY (Don’t Repeat Yourself)

This principle states that each small pieces of knowledge (code) may only occur exactly once in the entire system. This helps us to write scalable, maintainable and reusable code.

Example – Asp.Net MVC framework works on this principle.

KISS (Keep it simple, Stupid!)

This principle states that try to keep each small piece of software simple and unnecessary complexity should be avoided. This helps us to write easy maintainable code.

YAGNI (You aren’t gonna need it)

This principle states that always implement things when you actually need them never implements things before you need them.

Principles of Object-Oriented Design(OOD)

Problem

- Poor dependency managment

Principles of class design: SOLID

SRP	**Single Responsibility Principle**	A class should have one, and only one, reason to change.
OCP	**Open Closed Principle**	You should be able to extend a classes behavior, without modifying it.
LSP	**Liskov Substitution Principle**	Derived classes must be substitutable for their base classes.
ISP	**Interface Segregation Principle**	Make fine grained interfaces that are client specific.
DIP	**Dependency Inversion Principle**	Depend on abstractions, not on concretions.

Principles are about package cohesion

The first three package principles are about **package cohesion**, they tell us what to put inside packages:

REP	**Release Reuse Equivalency Principle**	The granule of reuse is the granule of release.
CCP	**Common Closure Principle**	Classes that change together are packaged together.
CRP	**Common Reuse Principle**	Classes that are used together are packaged together.

Principles are about couplings between packages

The last three principles are about the **couplings between packages**, and talk about metrics that evaluate the package structure of a system.

ADP	**Acyclic Dependencies Principle**	The dependency graph of packages must have no cycles.
SDP	**Stable Dependencies Principle**	Depend in the direction of stability.
SAP	**Stable Abstractions Principle**	Abstractness increases with stability.

SOLID

SOLID

- **SOLID -** принцип дизайна классов, которые будет легко поддерживать и расширять в течение долгого времени.
- **S.O.L.I.D** is an acronym for the **first five object-oriented design(OOD)** principles by Robert C. Martin, popularly known as Uncle Bob.
  - **SRP** (single responsibility principle)
  - **OCP** (open–closed principle
  - **LSP** (Liskov substitution principle)
  - **ISP** (interface segregation principle)
  - **DIP** (dependency inversion principle)

SRP

Просто потому, что ты можешь, не значит, что ты должен.

SRP (Принцип единственной ответственности)

- У компонента в системе должна быть одна ответственность.
- Это не означает, что компонент должен делать только что-то одно.

OCP

При надевании пальто не требуется операция на открытой груди.

OCP (Принцип открытости/закрытости)

- Программные компоненты должны быть открыты для расширения, но закрыты для изменения
- Необходимо стремиться к тому, чтобы менять поведение компонента, не меняя кода базового класса.
- При этом дизайн системы должен быть простым и устойчивым к изменениям за счет абстракции с инкапсуляцией и наследованием.
- Необходимо поддерживать баланс между гибкостью изменения и скоростью разработки

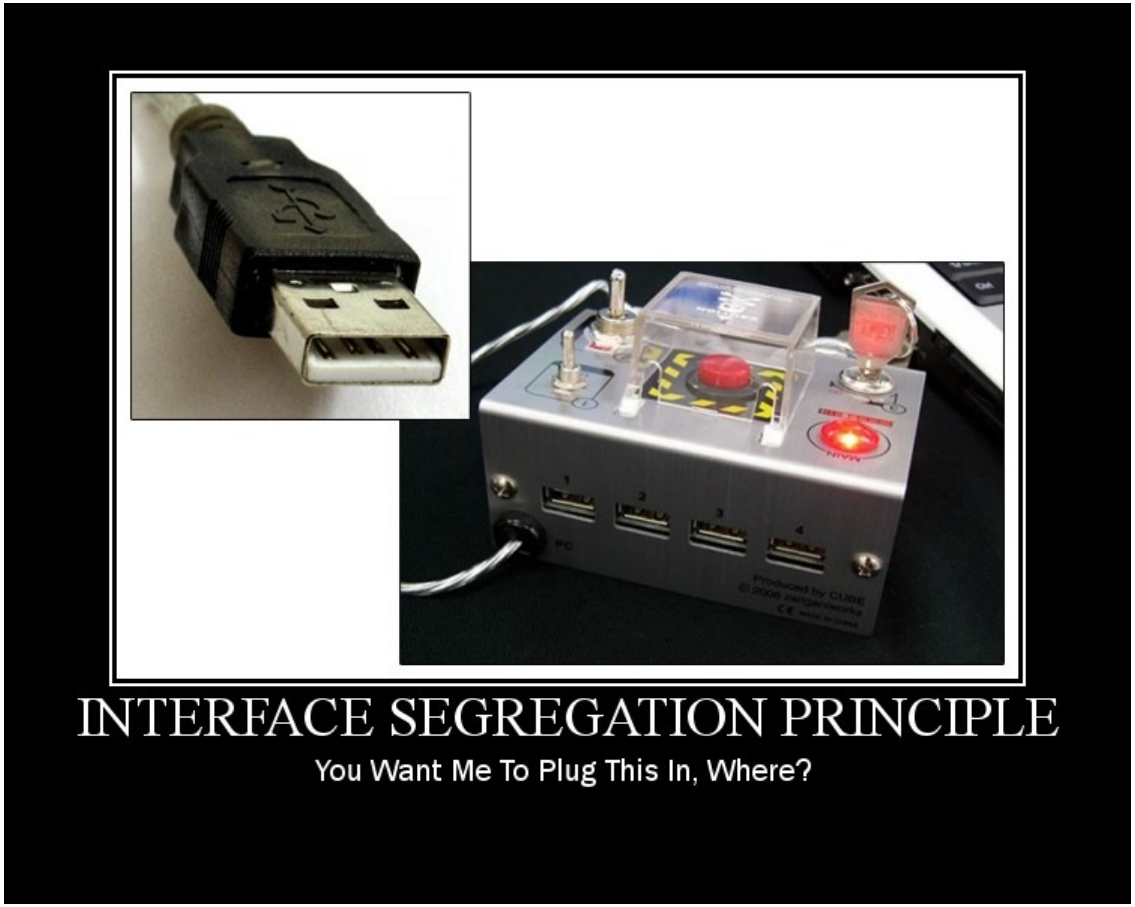
LSP

Если оно похоже на утку, крякает как утка, но нужны батарейки - вероятно, у вас неправильная абстракци.

LSP (Принцип подстановки Лисков)

- Функции, использующие базовый тип, должны иметь возможность использовать подтипы базового типа не зная об этом
- ИЛИ: тип `A` будет подтипом `B` тогда и только тогда, когда в любом месте программы `B` можно заменить на `A`.

ISP

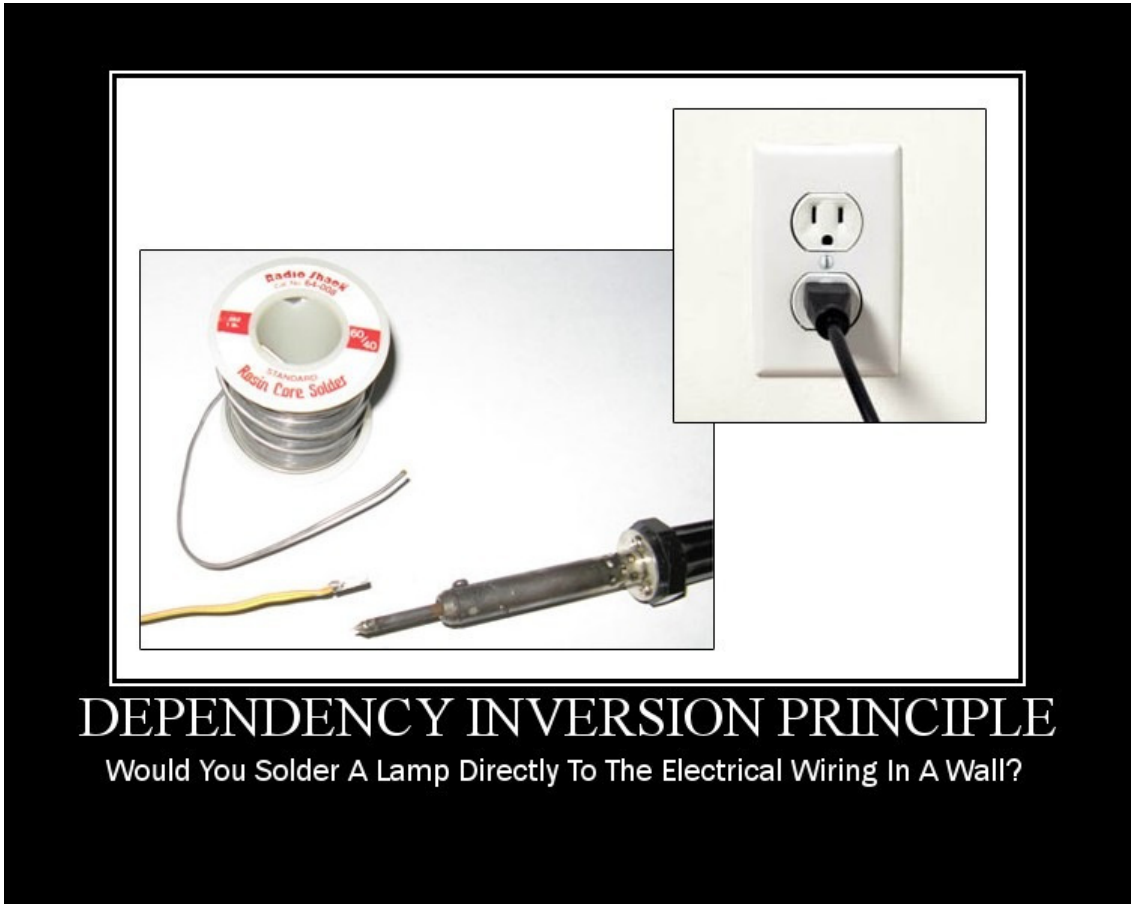


Вы хотите, чтобы я подключил это? Только куда и как?

ISP (Принцип разделения интерфейса)

- Клиенты не должны зависеть от методов, которые они не используют
- ИЛИ: если какой-то метод интерфейса не используется клиентом, то изменения этого метода не должны приводить к необходимости внесения изменений в клиентский код.

DIP



Вы бы припаяли лампу прямо к розетке в стене?

DIP (Принцип инверсии зависимостей)

- Класс не должен зависеть от другого класса, они оба должны зависеть от абстракции (интерфейса)
- применим в большинстве случаев, но не везде.