SOLID Principles

PV260 Software Quality

Stanislav Chren

18. 3. 2021

SOLID Principles

- Problems the SOLID principles help to address [1]
  - **Rigidity**
    making small changes ripples throughout the entire system
  - **Fragility**
    changes to one module causes other unrelated modules to misbehave
  - **Immobility**
    a module's internal components cannot be extracted and reused in new environments
  - **Viscosity**
    building and testing are difficult to perform and take a long time to execute
- Only recommendations and best practices, not hard rules

[1] Taken from http://zeroturnaround.com/rebellabs/object-oriented-design-principles-and-the-5-ways-of-creating-solid-applications/

SINGLE RESPONSIBILITY PRINCIPLE

Just Because You Can, Doesn't Mean You Should

Single Responsibility Principle

SOLID

- A class should have exactly one responsibility
- Responsibility is the purpose of the class
- Responsibility is a reason to change
- Seems simple, very hard to get right

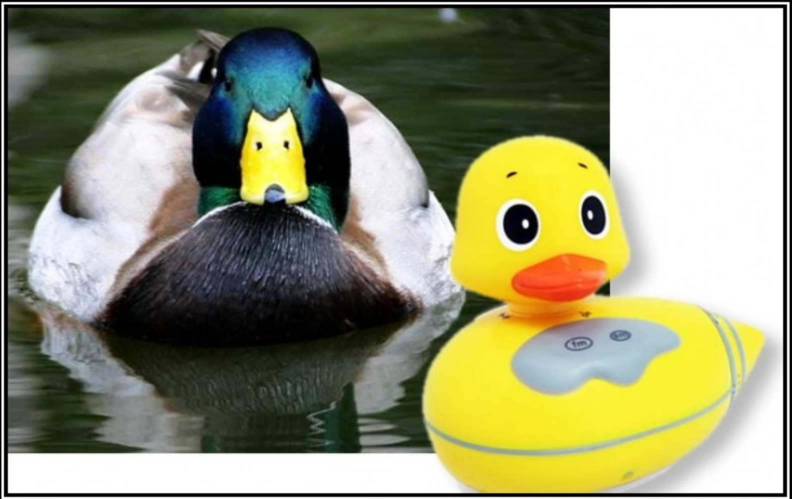
OPEN CLOSED PRINCIPLE

Open Chest Surgery Is Not Needed When Putting On A Coat

Open Closed Principle

SOLID

- Behavior of a class should be extendable without modifying the class itself
- Modules should be open for extension, closed for modification
- Changing existing code could break other part of the system
- Adhering to the principle yields reusability and maintainability



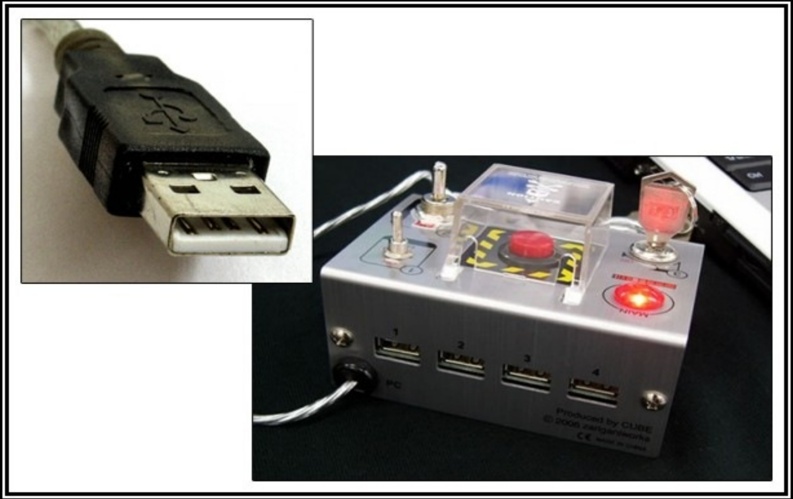
LISKOV SUBSTITUTION PRINCIPLE

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

Liskov Substitution Principle

SOLID

- Class should be substitutable for any of its subclasses
- The contract of the supertype must be satisfied, implementation details are irrelevant
- The principle is broken if client has to check which implementation is actually used



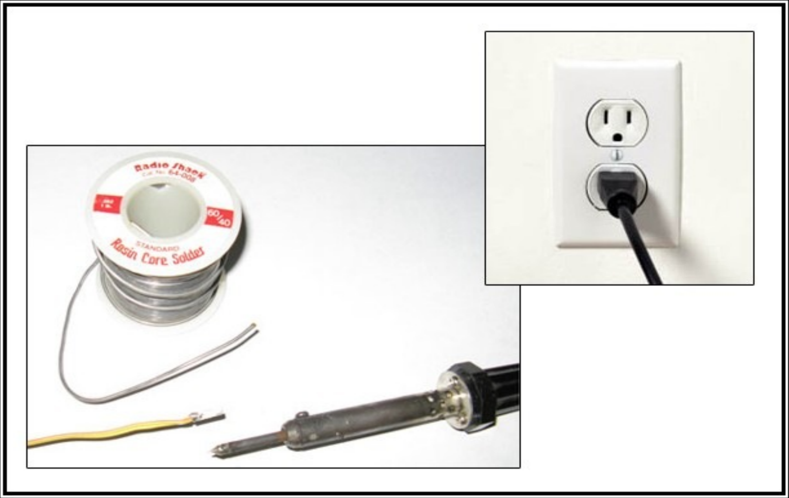
INTERFACE SEGREGATION PRINCIPLE

You Want Me To Plug This In, Where?

Interface Segregation Principle

SOLID

- Many client-specific interfaces are better than one general-purpose interface
- Clients should not be forced to depend on interfaces they do not use
- Adhering to the principle results in high cohesion and low coupling
- The principle is broken if usually only a small subset of the interface is used



DEPENDENCY INVERSION PRINCIPLE

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

Dependency Inversion Principle

SOLID

- High level modules should not depend upon low level modules, both should depend upon abstractions
- Abstractions should not depend upon details, details should depend upon abstractions
- Violating the principle leads to hard to change and fragile software

Further Reading

- http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod
- http://zeroturnaround.com/rebellabs/object-oriented-design-principles-and-the-5-ways-of-creating-solid-applications/
- http://code.tutsplus.com/series/the-solid-principles–cms-634