SOLID object oriented principles

Tuesday, 18 April 2023 10:52 AM

The Liskov Substitution Principle is the 3rd of Robert C. Martin's famous SOLID desig

- **S**ingle Responsibility Principle
- **O**pen/Closed Principle
- **L**iskov Substitution Principle -> https://stackify.com/solid-design-liskov-substitutio
  - It extends the Open/Closed principle and
  - enables you to **replace objects of a parent class with objects of a subcla application**. This requires all subclasses to behave in the same way as the
  - To achieve that, your subclasses need to follow these rules:
    - Don't implement any stricter validation rules on input parameters t parent class.
    - Apply at the least the same rules to all output parameters as applie
  - If some functions are not common for all subclasses to implement, don't instead add it in the subclasses directly.

- **I**nterface Segregation Principle
  - "Clients should not **be forced to depend u**pon interfaces that they do not
  - By following this principle, you prevent **bloated interfaces** that define me responsibilities.
  - Define light weight interfaces if the implementation cannot be common f classes
    - Coffee Machine Interface -> BasicCoffeeInterface
      - -> EspressoInterface
        - Let the client implements sub interfaces if they want deviated
  - As explained in the Single Responsibility Principle, you should avoid classe multiple responsibilities because they change often and make your softw

- **D**ependency Inversion
  - DI implements IOC(Inversion of Control principle)
  - https://www.tutorialsteacher.com/ioc/dependency-inversion-principle
  - Thus, we have implemented DIP in our example where **module** (CustomerBusinessLogic) and **low-level modu** (CustomerDataAccess) are **dependent on an abstract**

n principles:

n-principle/

ss without breaking the
e parent class.

han implemented by the

d by the parent class.
keep them in interface,

t use.”
ethods for multiple

or all implementation

implementation.
es and interfaces with
are hard to maintain.

a **high-level**
ule
tion

(CustomerDataAccess) are **dependent on an abstrac**
(ICustomerDataAccess). Also, the abstraction (ICustom
does not depend on details (CustomerDataAccess), b
depend on the abstraction.