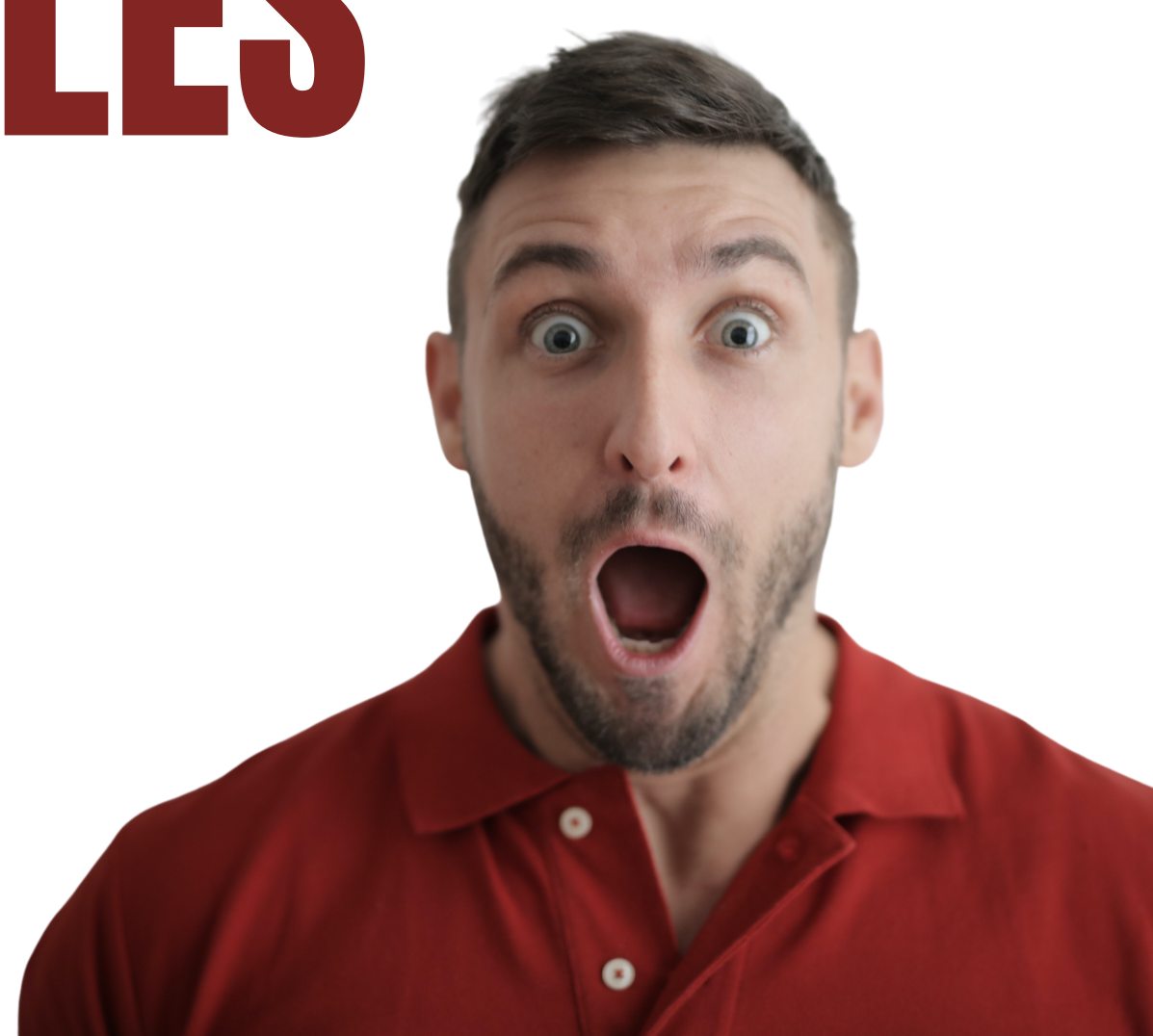
SOLID DESIGN PRINCIPLES



NamasteDev.com

SOLID

is an acronym for five design principles intended to make software designs more understandable, flexible, and maintainable

SOLID stands for:

S - Single Responsibility Principle
O - Open Closed Principle
L - Liskov Substitution Principle
I - Interface Segregation Principle
D - Dependency Inversion Principle

NamasteDev.com

Single Responsibility Principle

" A class should have one and only one reason to change, meaning that a class should have only one job."

EXAMPLE

Consider a module that calculates data and prints it.

In the future, this module may need to change for two reasons either data parameters could change or the way of printing results could be changed.

Hence this will be a bad practice, as there could be more than one reason for changing a module.

Open Closed Principle

"Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification."
This means that a class should be extendable without changing the source code.

EXAMPLE

- You have written code for module A, your friend wants to add another functionality to your module.
- He will be allowed to extend the existing class A but he is not supposed to modify the class directly.
- This separates the existing code from the modified code and provides stability.

Liskov Substitution Principle

"If S is a subtype of T, then objects of type T may be replaced with objects of type S”.
This means that every parent or base class can be substituted by its child classes.

EXAMPLE

- You have a class of Rectangles that has unequal width and length.
- Now, you want to extend the class to Square, which has an equal size of width and length.
- If you swap child’s object to parent’s, you will modify parent’s properties and hence violate Liskov Principle

Interface Segregation Principle

It states that no client should be forced to depend on methods or interfaces it does not use

EXAMPLE

- You search for purchasing a laptop online then the website should show you a catalog of only laptops of different brands and should not include other devices such as phone or cameras which you don’t want to purchase.
- The common or general catalog for everyone can be divided into multiple catalogs instead of just one.

Dependency Inversion Principle

High-level modules should not depend on low-level modules. Both should depend on abstractions (e.g., interfaces). Abstractions should not depend on details. Details should depend on abstractions.

EXAMPLE

- This principle is a specific form of decoupling software modules
- High-level modules should be independent of the low-level module implementation details.
- Main motive of this principle is decoupling the dependencies so if class A changes, class B doesn't need to care or know about the changes.

NamasteDev.com

Like !

Dont forget to Save the Post

FOLLOW

@NamasteDevOfficial

on

NamasteDev.com