Solid Principles:- 18/6/25

It helps developers write cleaner, more understandable code.

-> By following these principles it will become less prone to bugs.

S -> Single Responsibility Principle
O -> Open & Close Principle
L -> Leskov's Substitution Principle
I -> Interface segregation Principle
D -> Dependency Inversion Principle.

i) ~~Solid~~ Single Responsibility:

It states that every class must perform single functionality.

-> If we write multiple functionality in single class then it leads to mashup the code.

ii) Open for Extension & closed for Modification:

It states that the module should be open for extension but closed for modification.

which means a class should be extends to other but not able to Modify the class.

iii) Liskov's Substitution Principle

-> LSP states that an object of child class must be able to replace an object of parent class without breaking the application.

Eg:- Employee -> parent
  └ method 1 -> calculate salary()
  └ method-2 -> calculateBonus()

Child1 -> PermanentEmployee extends parent
  └ use all methods

Child2 -> contractualEmployee extends parent
  └ This class can't use Bonus & override to method and throw error.

But according to this principle every methods should be used in child class but in Bonus class we can't use all methods this Breaks this Principle.

iv) Interface segregation Principle

It states that the class should not be forced to implement interface that it does not use.

-> solution -> keep multiple smaller interfaces than larger interfaces

nothing but "use multiple inheritance concept"

v) Dependency inversion Principle:

-> It states that high level class must not be depend upon a low level class

Live class:-

Principles -> when building some system is the software we should following some principle.

-> so, it will be easy for other to develop & extension.

Design Principle

↳ SDLC -> Design, develop, test, integrate, .

Solid

↳ single Responsibility -> student & Professor example.

↳ main idea is if a class is there it should only have only one functionality

↳ Open/close

Bank -> car & electric car example.

Class -> skeleton, consits of variable & methods

abstract -> Abstract

interface -> ~~extends~~ implements.

0.10 of

= 0.