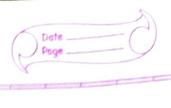
| | Gession-2 |
|----------|--|
| | 2 Solid Design Principles |
| | What are SOLID Principles? SOUD principles Of OOPS (Object Oriented Programming) |
| | SOLID principles of OOPS (Object Osiented Programming -g System). The five concepts make up our Solid principles. |
| | S -> Single Responsibility Principle |
| | O > Open Closed Principle |
| | L > Liskov Substitution Principle |
| | I > Interface Segmented Principle |
| | D -> Dependency Inversion Principle |
| | Why we need GOLID Principle. Advantages of GOLID Principles |
| | Help us to write better code by: . Avoiding duplicate values |
| | · Easy to maintain · Easy to Understand |
| <u> </u> | · Flexible Software . Reduce Complexity |
| | |
| , (-) | |



(1) S - Single Responsibility Principle (one reason to change).

· A class should have only I susponsibility principle / I reason to Change.

→ Simplifies Debugging

Advantages:-

> Testing and

-> Enhances maintainability

(9) Have you ever seen the worst single responsibility principle vaiolation? Ves when I merge my entity layer.

"Entity layer with my service layer.

(As all the lawer and a service layer.

(As all the layers are segregated accordingly)

UI layer Represents accepts the is very first layer of Client-server

Bervice Layer >> service layer Business layer where we write business logics.



Why Single Responsibility Principle
Important?

(a) In Real world, Requirement changes
and so does our code implementation
to maintain the changing requirements.

(b) Testing is Easier > With a Dingle responsibility, the class will have

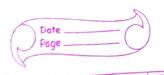
(d) Prevents frequent changes to the same class.

What will happen if there will be frequent changes to some class?

The more we changes to the same class the existing implemented functionality of that class may shift here and there boz of which some code may make and can face problem during production.

serious regal course of James

that consuct stars a



(2) -> O > Open for Extension but closed for modification

Benefital

Promotes flexibility and minimizes

the risk of breaking existing code.

Wer for Extension you should be able to add new functionality to the dose to add new functionality to modification for shouldn't modify existing code in the class to add new function modification.

The term open for extension! means

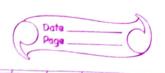
that we can extend and include extra functionalities features in our code without affecting altering our existing implementation.

The term "Closed for modification"

means that after we add the extra functionality, we should not modify the existing implementation.

How to implement Open Closed principles

By simply extends the class and overvide some functions.



JUX (3) [-> Liskov Substitution Principles]

If class B is subtype of class A, then we should be able to replace Object of A with B without breaking the behaviour of the program.

Subclass should extend the capability of parent class not narrow it down.

Advantages

Ensures Compatibility and promotes reusable and reliable code.

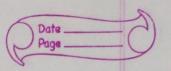
Doorgly implementing it can prove real world objects wrong like: Jenney it's not toue but can be easily implemented through code.

To prevent this better use

Liskov substitution poinciples.

Interview XXXXX Is squaree a subclass of Reclargle? No making square a Subclass of rectangle in object-arriented programming is considered a bad idea in practice because it violates the Liskov Substitution Principle (LSP) Which is a key principle of object - oriented design.

| | (4) I - Interjace Principle Segregation |
|----------|---|
| | Segregation |
| | |
| | > Interfaces should be such that client |
| | should not implement unnecessary |
| | functions they do not need. |
| | |
| → | ISP States that we should split our interfaces into smaller and more specific ones. |
| | Advandages Procés Specific ones. |
| | |
| | Reduces code bloat and enhances |
| | modularity. |
| | · T |
| | to prevent Client from unnecessarily getting |
| | - To prevent Client from unnecessarily getting Stuck in implementing unwanted methods. |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| - 11 | |



5) D. Dependency Inversion Principle)

- (a) It states that "Depend on abstractions, not on concretions".
- (b) We Should design our software in such a may that Various modules can be separated from each other using an abstract layer to bind them together

DRAFE BY Employee Solony DESC LIMIT