Mastering Object Oriented Design Principles

S.O.L.I.D.

Part 1

Single Responsibility
Liskov's Substitution Principle
Dependency Inversion
Open/Closed Principle
Interface Segregation

Visit GitHub @BCAPATHSHALA

1. What is A Good Coder / Quality Code?

- *A good coder produces quality code that is not only functional but also maintainable, scalable, and robust.*
- *Quality code adheres to coding standards, is well-documented, and follows best practices.*
- *It should be easy to understand, modify, and extend without introducing bugs or unexpected behaviour.*

A good coder produces quality code that possesses the following characteristics:

1. ***Readable:*** *Code is easy to understand and maintain.*
2. ***Rememberable Code:*** *Comments are used judiciously to explain complex logic or algorithms.*
3. ***Extensible:*** *Code can be easily extended without major modifications.*
4. ***Flexible:*** *Code is adaptable to changes in requirements or environment.*
5. ***Stable:*** *Exception handling is implemented effectively to ensure stability.*
6. ***Modular:*** *Code is organized into logical modules, promoting reusability and maintainability.*
7. ***Secure:*** *Security measures are implemented to protect against vulnerabilities and threats.*
8. ***Correct:*** *Code functions as intended and produces expected outcomes.*

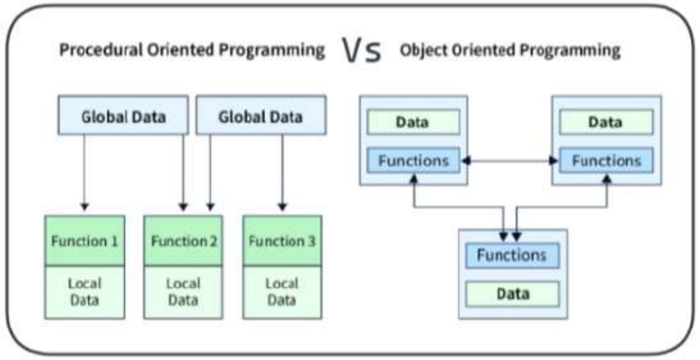
2. Functional Vs Object Oriented Programming

*Functional Programming **(FP)** and Object-Oriented Programming **(OOP)** are two paradigms used in software development.*

Functional Programming: (TREE) *Focuses on the evaluation of expressions, avoiding changing-state and mutable data. Functions are treated as first-class citizens, enabling higher-order functions and immutability.*

Object-Oriented Programming: (LEGO) *Organizes software design around objects, which encapsulate data and behaviour. Objects interact with each other through methods and messaging.*

Both paradigms have their strengths and weaknesses, and the choice between them depends on the specific requirements of the project.



3. What is SOLID Principles?

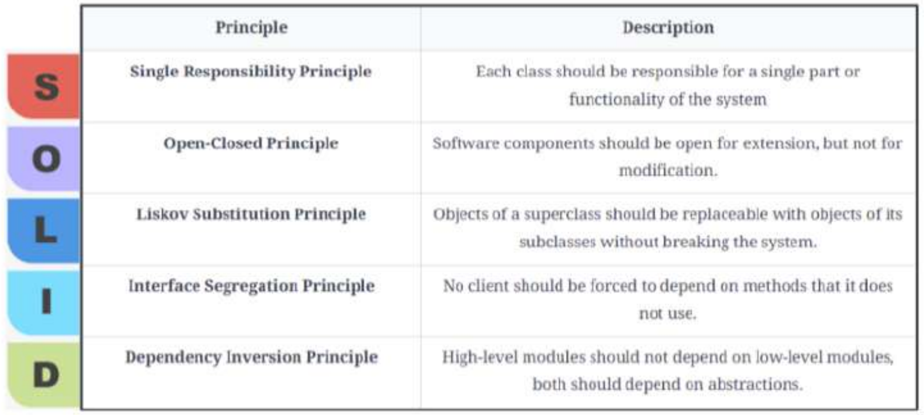
*The **SOLID** principles are a set of five fundamental design principles that were introduced by **Robert C. Martin (Uncle Bob)**, named by Michael Feathers.*

Why SOLID Principles? (Good Coder)

- *To make code more maintainable, easier to understand, easy to use.*
- *To make it easier to quickly extend the system with new functionality without breaking the existing ones.*
- *To make the code easier to read and understand, thus spend less time figuring out what it does and more time actually developing the solution. (Time Saving)*

The SOLID principles are:

1. *Single Responsibility Principle **(SRP)***
2. *Open-Closed Principle **(OCP)***
3. *LISKOV Substitution Principle **(LSP)***
4. *Interface Segregation Principle **(ISP)***
5. *Dependency Inversion Principle **(DIP)***



	Principle	Description
S	**Single Responsibility Principle**	Each class should be responsible for a single part or functionality of the system
O	**Open-Closed Principle**	Software components should be open for extension, but not for modification.
L	**Liskov Substitution Principle**	Objects of a superclass should be replaceable with objects of its subclasses without breaking the system.
I	**Interface Segregation Principle**	No client should be forced to depend on methods that it does not use.
D	**Dependency Inversion Principle**	High-level modules should not depend on low-level modules, both should depend on abstractions.