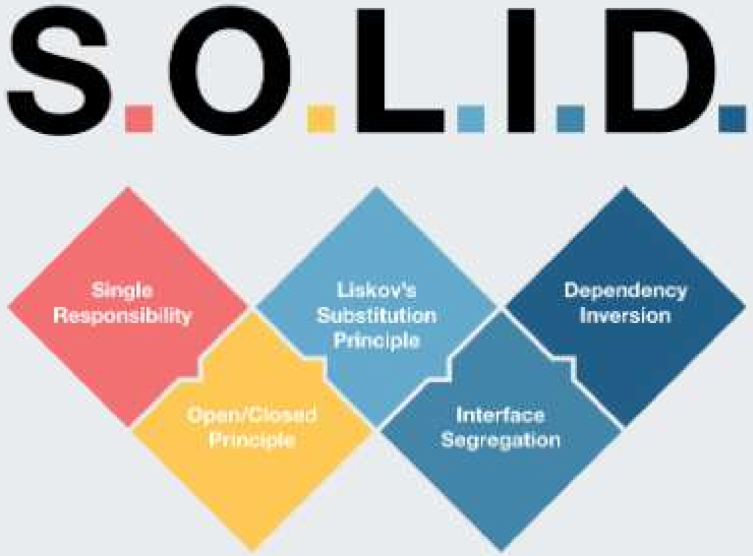
Dependency Inversion Principle



Part 6

Visit GitHub @BCAPATHSHALA

What is Dependency Inversion Principle (DIP)

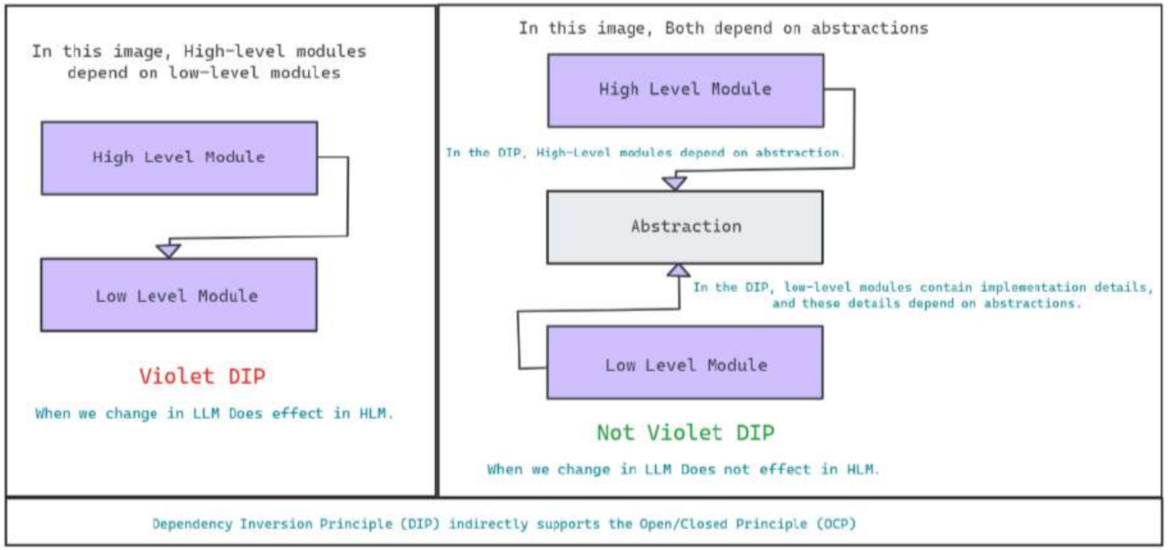
1. Never depend on everything **concrete(Actual Class)**, only depend on **Abstraction.**
2. High level module should not depend on **low level module.** They should depend on Abstraction.
3. Able to change an implementation easily without altering the high level code.
4. By adhering to **DIP,** you can create systems that are resilient to change, as modifications to concrete implementations do not affect **high-level modules**.

In One Statement

*The Dependency Inversion Principle suggests that **high-level modules** should not depend on **low-level modules,** but both should depend on abstractions. Additionally, abstractions should not depend on details; details should depend on abstractions.*

Key Idea

*1. **High-level modules** should not depend on **low-level modules;** both should depend on abstractions.*
*2. **Abstractions** should not depend on **details.** Details should depend on abstractions.*



In this image, High-level modules depend on low-level modules

Violet DIP

When we change in LLM Does effect in HLM.

In this image, Both depend on abstractions

In the DIP, High-level modules depend on abstraction.

In the DIP, low-level modules contain implementation details, and these details depend on abstractions.

Not Violet DIP

When we change in LLM Does not effect in HLM.

Dependency Inversion Principle (DIP) indirectly supports the Open/Closed Principle (OCP)

Real-Time Examples

- *Building a **LEGO tower** – the bricks **(high and low-level modules)** connect through smaller bricks **(abstractions).***

How can Interface Segregation Principle be applied?

Visit GitHub:
@BCAPATHSHALA

Practical Coding Examples in Java #1
Practical Coding Examples in Java #2
Practical Coding Examples in Java #3
*Much more about **Dependency Inversion Principle***