OOP: Helps us to provide a way to write and polish our code in a better way.

OOD: We need to think beyond this so that we can have better design SOLID helps us with this.

S - Single Responsibility principle

O - Open and close principle

L - Liskov sunstitution principle

I - Interface segregation principle

D – Dependency inversion principle

SRP single responsibility principle:

As name suggest class should have a single responsibility.

There should be only one reason to changes the class. Assigning more behaviour to the class lead to changing the class very now and then.

Eg: I have a person table in which I have to perform CRUD operations this has to be designed separately so that not all operations are done as apart of one class.

Single focused funtionality

Eg: Journals with check list

Maintaining the journals

Maintaining the persistence of the Journals.

Open close principle:

Eg:1. Product with name, size ,color

ISpecification, Ifilter

Filter product based on size and Then color, next is based on both

2. Invoice validation

Add a validator class and perform validation for each

Liskov substitution principle:

A parent cannot replace child seamlessly (always its not true)

Eg: Invoice, Complex invoice

Make 2 separate implementation if the parent child is using some common behaviour and child implements few additional things.

Base can be replaces to a child class.

Eg : Shape rectangle and square

Implement virtual properties.

Interface segregation principle:

Interface shouldn’t force the class to implement the method which They actually don’t need.

Instead of having one big interface with lot of functionality have simple interface with single to 2 responsibility

Eg: printer interface with print, scan, fax operation

Invoice printer with and without header

Dependency inversion principle

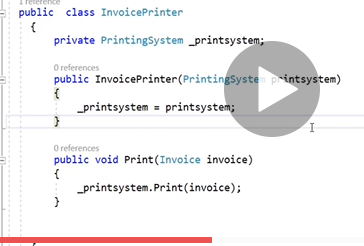
High level modules are not dependent on low level modules

High level modules are nothing but the class that consuming some existing implementation

Low level modules are nothing but the class that provides the implementation

Eg: Person relationship class where the children’s are access through abstraction

Objects should be loosely coupled and injection should happen at the run time.



Eg: invoice printing

With printing system1 and printing system2 Above code is tightly coupled because invoice print is using printing system 1 directly

Loosely coupled 2 classes dependent on each other through abstraction

