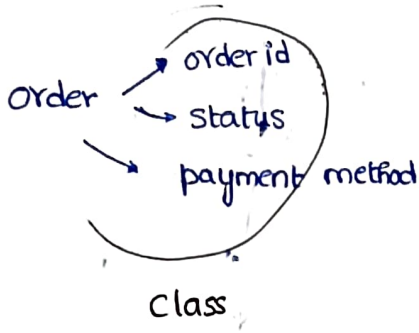
LLD - Day 01 :

- Modular, Scalable, Extensible Code.
- Classes are blueprint. Objects are actual the instance of class.



Object

orderid : 1
Status : delievering
payment method : UPI

OOPs

- Way of Designing the Modular, Scalable, Extensible Code.
- (i) Polymorphism (ii) Inheritance
  (iii) Encapsulation (iv) Abstraction

Encapsulation : [private - public - protected]
we will have acess control here.

* It is a practise of bundling data (variables) and the methods (functions) that operate on that data into a single unit (class)

Abstraction : The process of hiding the implementation details of a system & showing only the essential features to the user.

Inheritance : It is the mechanism by which one class (called the child class/subclass) can acquire the properties & behaviors of another class (called the parent class /superclass)

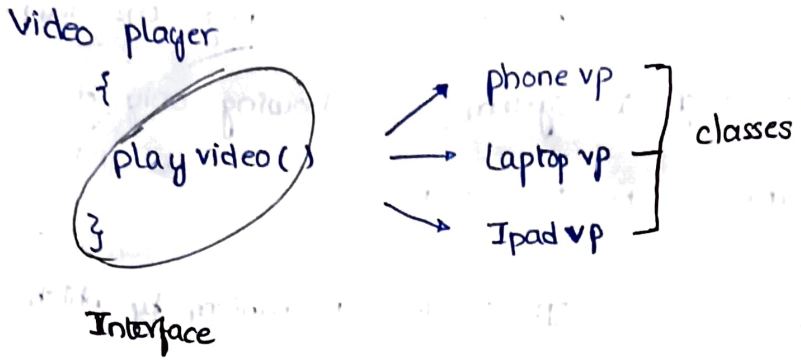
Polymorphism : (many forms)

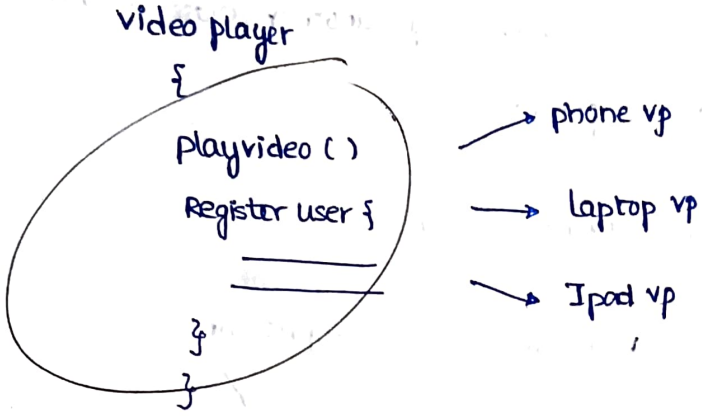
- Run Time & Compile Time
  - Function Overloading
  - Operator overloading

Same function, method or operator to behave differently depending on the object /context.

Q. Code Run Time & Compile Time Polymorphism.

Abstract Class	Interface
abstract methods (without implementation) + Concrete methods (with implementation)	only contains abstract methods and constants.





Abstract class

Register user ( ) { } - common rules for all

- Extend Abstract Class
- Implement Interface

can't extend

final class : we can't make sub class of it.

final method : we can't override it.

final variable : we can't the value.

P.T.O]

SOLID Rincipals :

S : Single Responsibity Principle

O : Open / Closed Principle

→ one class only one responsibility.

→ open for extension, closed for modification.

(adds new features without changing existing code)

~~class Area {~~

L : Liskov Substitution Principle

→ Objects of a superclass should be replaceble with objects of a subclass without breaking the program. (proper hierachy)

I : Interface Segregation Principle

→ Better to Create multiple small, specific interfaces rather than one big "fat" interface

P.T.O]

D : Dependency Inversion Principle

Too have loose coupling, not tightly coupled code. High Level Module must not depend on LL Module

Q. Code For SOLID Principles