Lect-2 : Dive into OOPS

OOPS samjhne ke pehle hum yeh smjhte hai ki humme jaroorat kyu padi?

History of Programming

- Machine Language (010110) -- 0/1
  - Prone to Error
  - Scalable
  - tedious (too long and slow)

  } Problems

- Assembly level Language (MOV A 61H)
  - Not Scalable
  - Prone to Error
  - Tedious / Can skip instruction / jimbling up register

- ~~Parade~~ Procedural Language CC - they introduced if-else, functions, loops, Blocks (switch)
  - cannot solve enterprise level problem
  - Can't write code acc to real world

→ OO Programming (Why it's best)
  → Real World Modeling
    ↳ Cconnecting everything to basic as Objects)
  → Data Security (encapsulation)
  ↳ Scalable and Reusable block of code

Why Real World Modeling ?
- Agar humme real world ki koi problem solve karni hai toh humme real world jaisa dekhna padta hai
i.e. Objects ⇒ Har kuch real world mein objects hai

And objects interacts with each other

What is Object
- For an entity to be object it needs -
  → Characteritics (unique ~~identity~~ property to identy object)
  ↳ Behaviour (methods joh woh perform krta)

Ex : Car

Characteristics	Behaviour (funcs)
→ Engine	→ start()
→ Brand	→ stop()
→ Model	→ gearshift()
↳ Wheels	→ accelerate()
	↳ brake()

But humme single car thodi banani and saare car ke paas same chize present hoti hai just model brand diff hota hai
LL> Isliye hum Class introduce krte

What is Class
↳ Blueprint of Objs

```
Class Car {
   //code
}
```

↳ Object
`Car * myCar = new Car()`

But agar humare pass OOPS nhi hote toh kya kya problems aate hume in procedural language

Car :→ Brand
↳→ Model
↳ IsEngineOn

```
String Brand;
String model;
bool isEngineOn
```

for owner
↓
`string name`

Method
↳ start()
→ stop()
→ gearshift()

```
start(){
}
stop(){
}
```

```
void drive(brand
    mode){
   start()
   gearshift()
   acceleration()
}
```

Ab agar hume owner owns the car batana hai toh ek drive() → method lagega & usmein yeh funcs pass krne padte
But agar multiple car ho toh
↳ fir se redeclare for new car
↳ more repetition
↳ Nt scalable and Nt flexible

But what if there's OOPS

```
class Car {
   string brand;
   string model;
   void start();
   void stop();
}
```

```
class Owner {
   object ← car car
   String name;
   void drive(){
      car.start();
   }
```

Pillars of OOPS

1. Abstraction
2. Encapsulation
3. Inheritance
4. Polymorphism

1. Abstraction : → H.L.L is best example of Abstraction kyuki apko pata nhi internally if / for kaam kaise kr rha
- Hiding internal details and showcasing only essential feature

Ex: Agar apko car chalani hai toh aap engine on/off kr skte ho par yeh jaroori nhi ki woh engine kaise bana hai ; on kaise ho raha hai etc.

Yaane hiding details

(Code is in VS)

* Virtual keyword : declare

Iska mtlb hai ki yahapar bas hum method ~~define~~ kar rahe hai and usko define krne ka kaam child class (jo parent class ko inherit krega) uska task hai...

2. Encapsulation - provides data security
- Wrapping (data & methods) into single capsule/box
↳ characteristics

- Data security kaise ?

Kuch characteristics ko koi bahar se manipulate naa kar paye jaise speed of car koi agar direct 500 krdega toh woh galat hai no isliye humme woh characteristic ka hide krke rkhna padta hai
manipulation

- koi bhi characteristic ko access karo but usse manipulate na karo aur krna hai toh bhi valid checks se

How?

1. Access Modifiers (access outside class ko control)
   - public - allowed outside class
   - private - nt allowed even in inherited/child class
   - protected - allowed in child class

2. Getter and setter
   - get : agar humme characteristic chahiye toh we need access to hum methods banate jo public modifier through declare krte
   - set : sets the value of variable after valid checks

Conclusion :

1. characteristic / variables private modifier ke through declare kro.
2. Methods ko public access modifier ke through