__/__/____

OOPs

video 1

def :- programming techniques in which et revolve and objects.

wht →

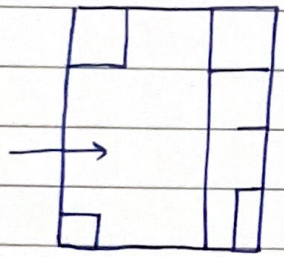
obj :- entity / chuez → state / properties
→ Behaviours / method

why → ① real life app relatable ② readable ③ reusable

- class :- custom datatype / user defined data type
  eg :- Ramesh → int
  → float
  → string

→ obj and class diff wid eg :-

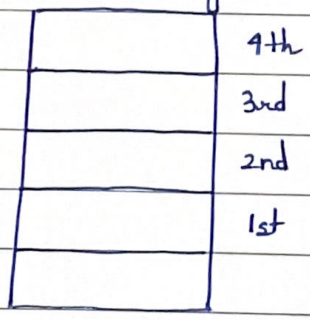
architect



Blueprint / design

class

actual real ghar



obj :- instance of class

__/__/__

→ class syntax :-

class Animal {
};

= size is 1 coz we need to give min possible memory to it- if 0 den class ka existance nahi hota.

→ Access modifier :- access ka scope define karte

- public - can be accessed inside n outside d class
- private - only accessed inside class, by default all mem pvt
- protected - behaves like pvt can only be accessed inside chi class

• for accessing class we need to use dot operator with obj name eg: ramesh.

→ class exp :-

class ____ { (name)

properties
int a;
string str

} data member

} stack memory

behaviour
void eat {
}

} → member func

};

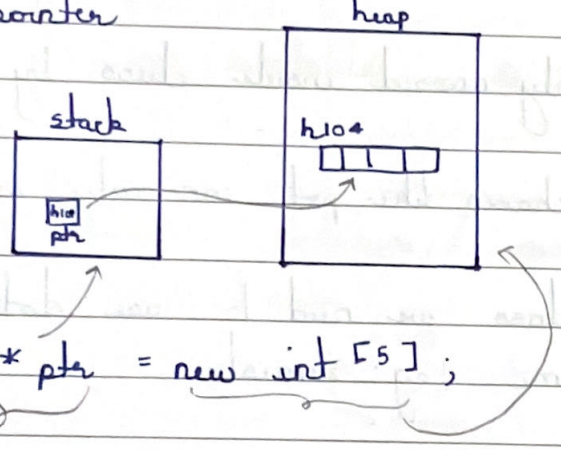
__/__/____

- when i want to use or access pvt member outside class we use get and set.
  eg: int weight and its func

→ DYNAMIC MEMORY :- heap

eg: int * a = new int ;

add is stored by pointer → returns add (v get 4 byte space in heap memory)



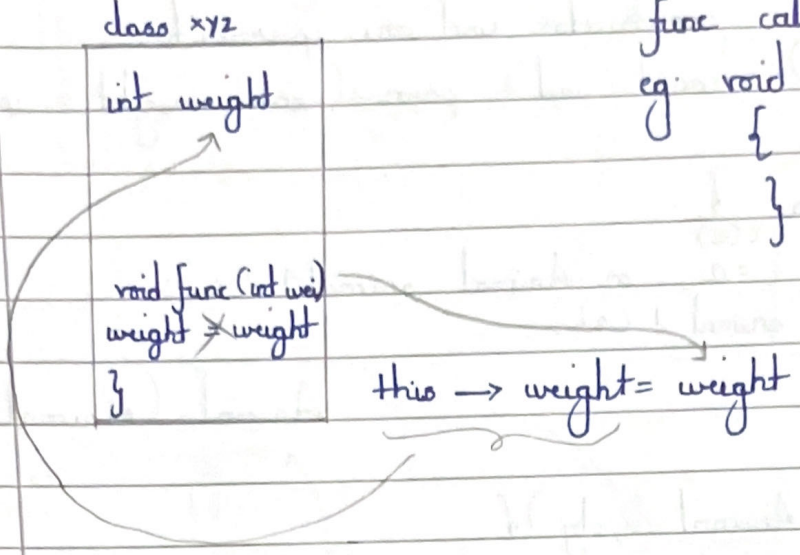
int * ptr = new int [5] ;

★ - stack ka andar space v take cleans automatically but not in heap we need to do manually.
alloc :- new keyword
dealloc :- delete keyword

→ this :- pointer to current obj

↳ jis obj ka liya setweight func call kiya hai

eg: void setweight (int weight)
{
}



→ Object creation :-

Constructor :- whenever v want to create obj first dis is called. if v don't create it by default hojata

- no return type
- name same as class
- initialize obj

1) default constructor

eg: Animal ( ) {
cout << " called " << endl;
}

this -> weight = 0;
or
this -> age = 0;

2) parameterized constructor

eg: Animal (int age) {
this -> age;
cout << "para";
}

we need to pass para in int main for dis

- jitna main mein pass karte value woh waise constructor search karta
  eg: (10) - constructor wid one parameter
  (10,1) - const wid 2 para (int weight, int age)

a) copy constructor

```
int main {
Animal c(a) = a; or Animal animal1 (c);
}   or animal d (*b);

Animal (Animal &obj){
this->age = obj.age
this->weight = obj.weight }
```

```
Animal (Animal obj){
}
```

copy not called

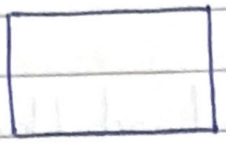
pass by value → obj copy

animal a=b

- By default constructor does shallow copy to avoid dis v want to do deep copy on our own

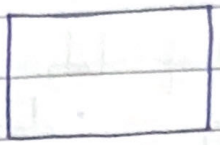
→ **Destructor** :- for memory free (~)

- no return type
- no i/p



static
↓
auto

eg: ~Animal () {
} = auto
↙
jahan obj create kiya hai n its scope ends there



dynamic
↓
manually

eg: delete b ;

video 2

4 pillars of OOPs

Encapsulation | Inheri | Poly | abstraction (dad of all)

→ **Encapsulation :-** DM | MF

- wrapping up of data
- data hiding · class by default act like this

perfect encap – mark all data mub as ~~pvt~~ private

adv :- 1) security 2) make class read only 3) reusable

→ **Inheritance :-** is a r/s (C·C → P·C)

- Base / Super / Parent class

↓ P·C properties inherited by C·C

sub / child / derived class

- **syntax** - class child : ____ parent class name {

↑ mode of inheri

→ public

→ pvt

→ protected

};

eg - class Dog : public Animal {

};

→

Base class ka access modifier	Mode of Inheritance		
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	NA	NA	NA

- from dis table ✓ & trying to find child class mein kya inherit hoga ke baad laga hoga by difault.
- pvt ko inherit nahi kar sakte, protected ko yess

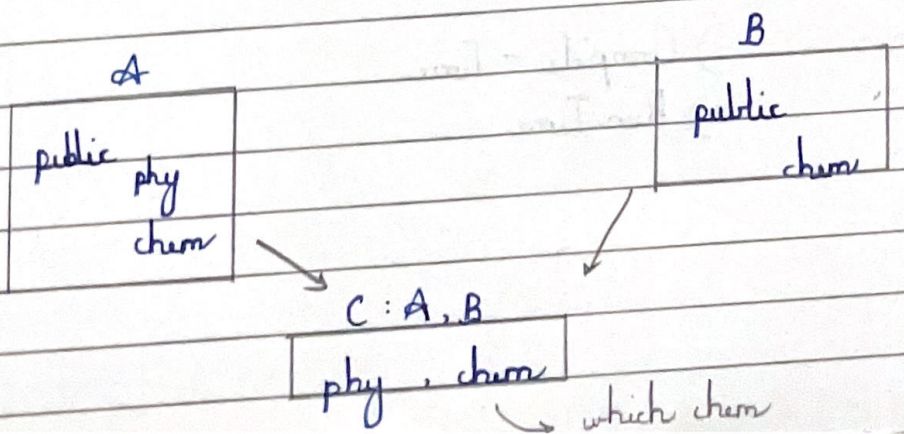
→ Types of Inheritance

① Single :- [parent] → [child] eg. car → scorpio

② Multilevel :- [parent] → [child] → [grandchild]
eg. fruit → mango → alphanso

③ Multiple :- [Mom] [Dad] → [Child]
eg. Tiger, Lion → Liger
syn :- class C : public A, public B

④ diamond prob :-

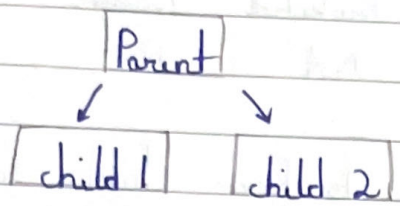


g. multi inheri possi in C++ but nd java

g. Diamond prob solv g. chart

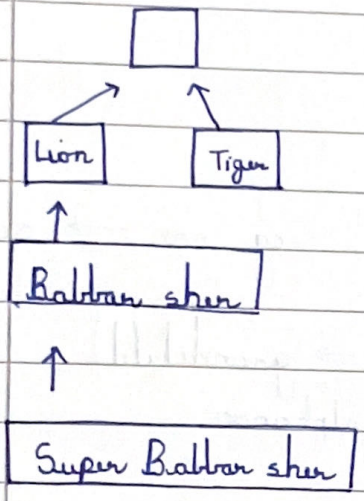
use scope resolution in dis case = obj. A :: chemistry

④ Hierarchical :- multiple ulta



eg. Car → Scorpio, fortuner

⑤ Hybrid :- mixture of all types of inheritance



→ Polymorphism :- existing in many forms

many + form

many of form ture

1) (same function name se kaise banaye ? func)

2) + se - ka work

- Types :- 1) Compile - Time
2) Run Time

① Compile Time :- → func overloading
→ operator overloading

eg:
```
class Maths {
    public:

    int sum (int a, int b) {
        return a+b;
    }
```

function overloading

```
    int sum (int a, int b, int c){
        return a+b+c;
    }

    int sum (int a, float b ){
        return a+b+100;
    }
```

→ same func name but v need to pass diff no: parameters or diff type parameter

syntax :- return_type operator +/-/etc ( ) {

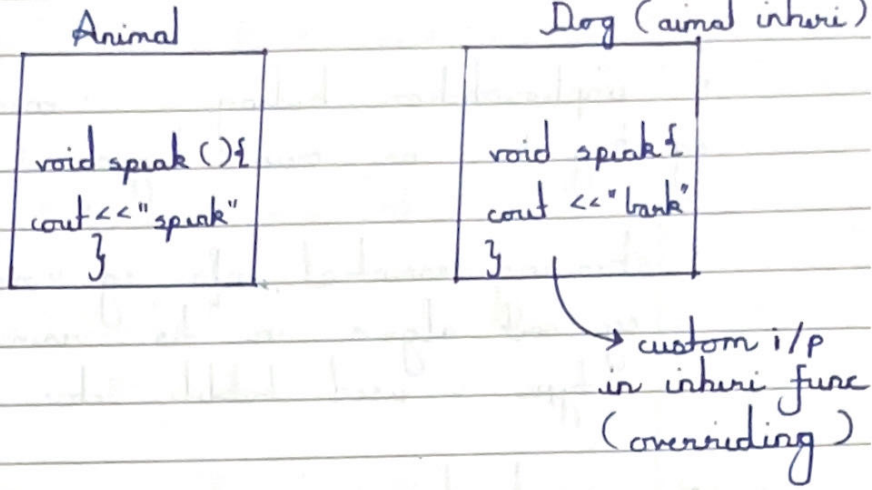
}

operator overloading

Video 3

② Runtime Poly :-

overridding :-



upcasting :- parent class pointer on child class obj
eg: Animal *a = new Dog ();
a -> speak ();

downcasting :- opp, eg: Dog *b = (Dog *) new Animal();

note :- when v do upcasting or downcasting without virtual keyword humesha pointer a method call hoga.

If v apply virtual keyword den jiskka obj banaya hoga uska func call hoga

→ In constructor cases :-

if we call child class method in all parent class method will print first den child class because it depends on parent class.
accpt :- Animal *a = new Animal () (dis is indipendent of everything)

overridding

q.) C.T vs R.T q.) encap vs abstraction

q.) diamond prob intro q.) abs eg

__/__/__

→ Abstraction :-

- implementation hiding • encap subset of this
  eg. key ko car mein lagana (not telling how engine etc works)
- showing essential info eg. container of diff things
  eg. sort algo = in dis user doesn't know which type v used bubble etc.
- generalization

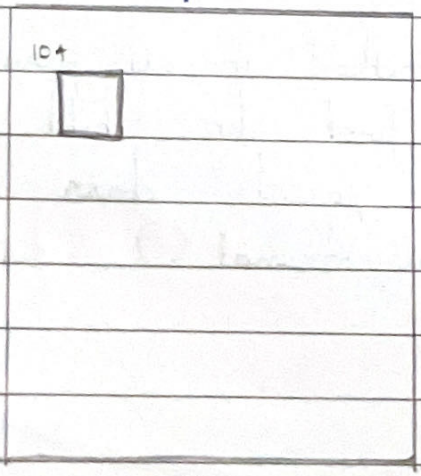
→ Dynamic Memory alloc :-

program start :-

→ stack memory - local variable, func parameter
  - by default small
  - can chg dis in OS, setting
  - func call ke vari etc.

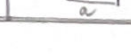
→ heap memory - Big by default

Heap



stack

1) int a = 5;

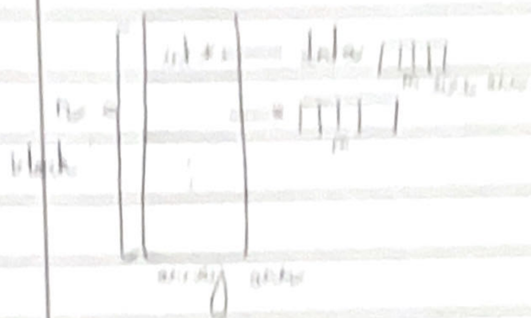
2) 

2) stack → int * a = new int; → heap

/ /

Y bad pract :- int arr[n];
n= 10 lakh n ko stack dia much space na
so v do int * arr= new int [n];

- 2D array :- int * * arr= new int * [n]
↓ do pointer ko store kro in RHS array
↓ pointer pass kro store



```
for (int i=0; i<n; i++)
{
    arr [i] = new int [m];
}
```

arr [i] → pointer; new int [m] → integer arr of size m

or

```
vector < vector <int> > arr (n, vector <int> (m, 0))
```

n ↑ row; m ↑ col; 0 ↑ initial

dealloc :-

```
1)  for (int i=0; i<n; i++)
    {
        delete [] arr [i];
    }

2)  delete [] arr;
```

[ ] [ ] [ ] → del kiya pehle

① const keyword
② shallow vs deep copy
③ Local & global variable
④ static keyword
⑤ abstraction
⑥ Inline Func
⑦ Friend keyword
⑧ constructor ext
⑨ virtual const vs virtual
⑩ memory layout