OOPs Placement Preparation

Class 1

OOPs → Object Oriented Programming

* Defination :- It is a programming technique where everything revolves around the object.

What is object → object is an entity, which have two things a) State or properties b) behaviour or Method [It is a collection of Data]

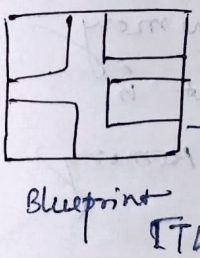
* Instance of Class

* why :→ when we write Code by object orient, then we gets many benifits
  → we can relate with real life example.
  → Readability is good, reusable,
  → easy to maintain, easy to understand

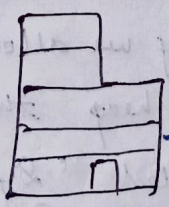
* Class :-
To create user define data type we use 'class'.
It contains the data members and member functions that operate on the data member.

[Actual Entities are 'Object', Blueprint or design are 'Class']

[Assume that we are creating a building then the Architech will draw the blueprint on the paper and the 'Karigar' will make the real house]



[This is Class]



→ [This is object]

Class creation

Syntax : class Animal {

};

- Size of empty class is : 1 ; [1 byte]

Object Creation

Object creation can be two type,

→ ① Static memory allocation

→ ② Dynamic Memory Allocation

* If we want to ~~accen~~ access a property of an object, then I have to use 'dot' operator.

ramesh.age; (see code in vscode)

* Accen Modifier (Accen Modifier define the scope of accen)

→ Public (public means, we can accen it inside the class and also outside the class) (data member)

→ Private → (we can only accen it inside the class.)

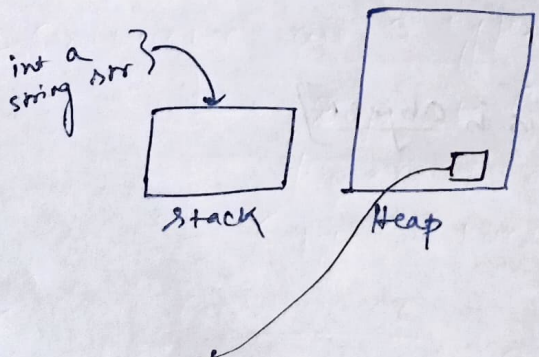
→ Protected

(By default 'Private' hota hai)

⇒ When we need to accen the private member, outside the class, then we use getter & setters.

- These are basically functions, 'getter' helps to fetch property & setter helps to set property.

Dynamic Memory Allocation



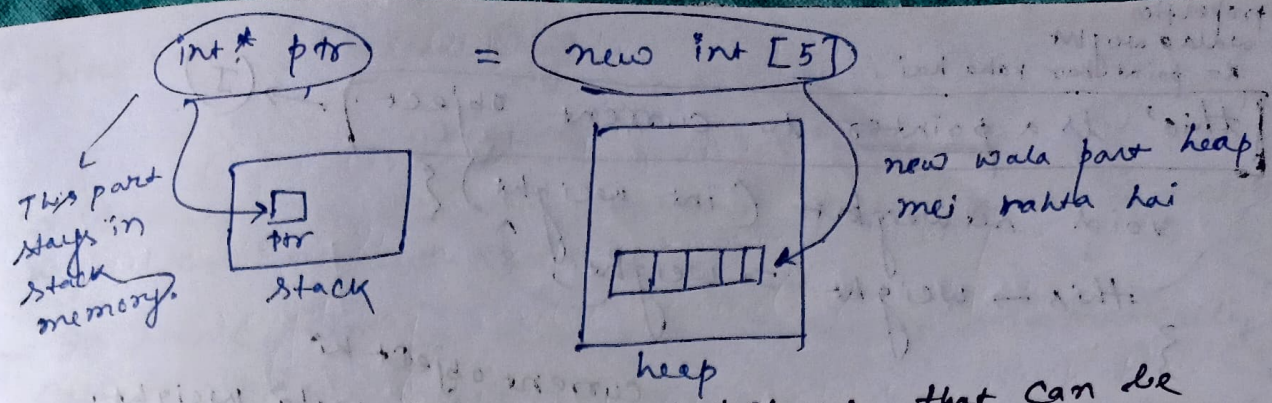
[If we allocate memory by heap then that is called 'Dynamic Memory Allocation']

new int; → This returns address.

using "New" keyword we allocate memory here (heap).

int* a = new int;

[As pointer stores address in C++, thats why here we are using pointer 'a', and the int is inside the heap memory.



* The space which we take in 'stack', that can be cleaned automatically, but the space taken in heap, can not clean automatically. That should be cleaned manually.

On heap → (allocation) using 'new' keyword
→ (deallocation) using 'delete' keyword.

```
int * a = new int;      alloc →  int * ptr = new int[len]
delete a;               Deallo → delete [] ptr;
```

Syntax

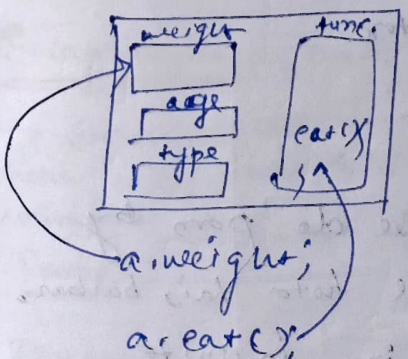
```
Man * Suresh = new Man;
(*Suresh).age = 15;
(*Suresh).type = "man";
```

[we are writing (*Suresh) because (*Suresh) it define actual object]

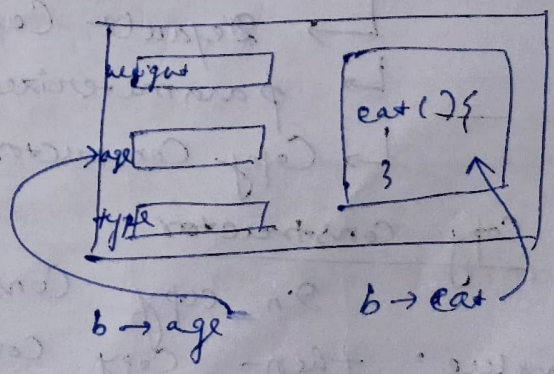
alternate way

```
Suresh -> age = 15;
Suresh -> type = "man";
```

Animal a;



Animal* b = new Animal;



properties wala weight ko point kar raha hai.

'this' is a pointer to current object. → (I)

```
void setWeight (int weight) {
    this → weight = weight;
}
```

this → weight → refers to the current object ki properties wala weight.

weight → refers to the ~~nearest~~ nearest weight. (input parameter wala)

Object Creation

- When we create a object at first 'Constructor' Call hota hai.

★ Constructor helps to initilize the object. 59:47
- ↳ No return type
- ↳ Name is same as class.
- ↳ initilize object.

★ By Default constructor stay kare, ;, in static & dynamic in every way at first constructure call hota hai.
When we create a constructor then a by default wala constructor overright ho jata hai.

★ Constructor are 3 types.
- ↳ Default Constructor
- ↳ parameterized Constructor.
- ↳ Copy Constructor.

★ Copy Constructor

In copy Constructor, if we do 'pass by value' then Copy Constructor call hota hai, barbar. That's why here we got stucked in infinite loop. Thats why we should have to 'pass by reference'

* Deep Copy / Shallow Copy.

Destructor

→ Destructor is used for memory free. Memory distraction wala kam Destructor karta hai.

→ In static Memory Allocation Destructor automatically call hota hai.

→ In Dynamic " " we have to delete manually delete.

⇒ Destructor is has no return type.

→ No input parameter.

→ '~' sign is used in Destructor.

→ When scope end the destructor called.

Brief :-

1) Class → Empty
→ Non empty

2) Object → Static
→ Dynamic

3) Access Modifier → Private
→ Public
→ P. - -

4) getter & setter

5) object in Memory

6) Obj Creation → default constructor
→ Parameterized Constructors
→ Copy constructor

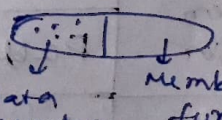
7) Destructor → Static (default call)
→ Dynamic (Manual call)

Class → 2

1) 4 Pillars of OOPs. → Encapsulation
→ Inheritance
→ Polymorphism
→ Abstraction.

Encapsulation (Data hiding)

→ Encapsulation is a such type of concept where we wrap data and hide data, sensetive data is hidden from the user.



Example :- Class creation.

Perfect Encapsulation

↳ When every data members are marked as private. (we can access those by using getter & setter)

Profit / Why

(1) Increase security & privacy. (3) Reusability.
(2) We can create read only class.

(*) Here we wrap the data member & member function into the class.
(*) Here we want to show only the essential data.
(*) & we can achive it by access modifier.

Inheritance :=

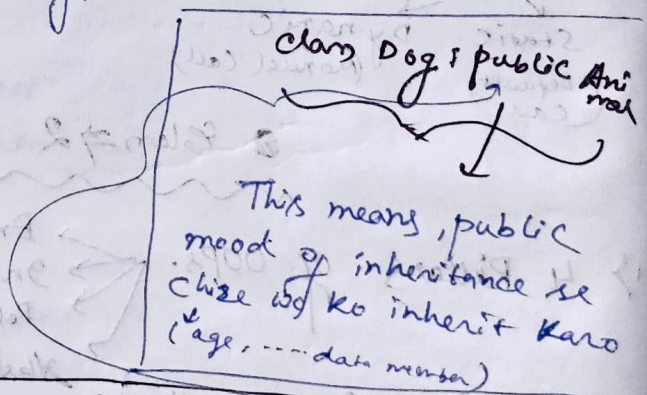
Properties are inherited by child class from the Super class. (we can inherit all the data members & member functions from the Super class to the sub class / child class)

Base class / Parent class / Super class
↕
Sub class / Child class

Syntax

Class child : ____ Parent
↑
mode of inheritance

Mode of inheritance can be 3 type
- ↳ Public,
- ↳ Private
- ↳ Protected.



(II)

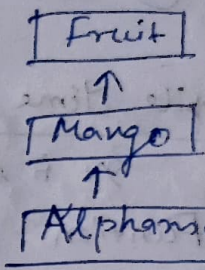
Base Class ka Access Modifier	Mode of Inheritance		
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	NA	NA	NA

⊛ We can not inherit private member at any ~~cost~~ mood.

Type of Inheritance

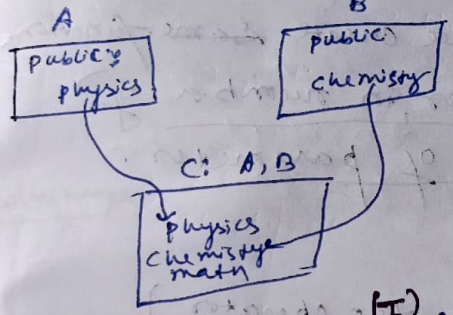
1> Single 2> Multi-level 3> Multiple 4> Hierarchical 5> Hybrid.

- **Sigle** Normal Inheritance.
- **Multilevel** →



Alphanso is a Mango, Mango is a Fruit.

- **Multiple (I)**

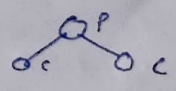


[Multiple Inheritance is not possible in Java]

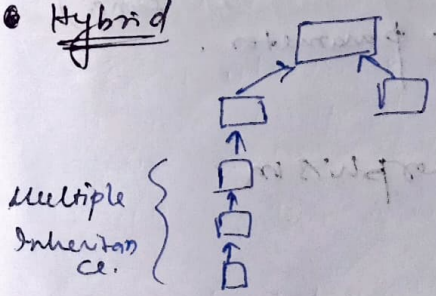
★ **Diamond Problem (I)** (To solve diamond problem we use scope resolution property) → obj.A::Chem

⇒ If we take properties with same name in Multiple Inheritance then Diamond problem happens. (that mean same data members will present in multiple classes)

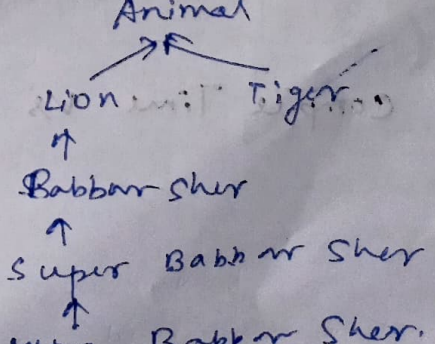
- **Heirarchicel** (1 parent, 2 many child)



- **Hybrid**



Example :-



Polymorphism (V.V.I)

Poly morphism
many form

existing in many forms

[It is the property of some code to behave differently for different context.] are

Types of Polymorphism

1. Compile Time polymorphism
2. Run Time polymorphism.

Compile Time Polymorphism

It is based on two concepts, those are

1. Function Overloading
2. Operator Overloading.

Function Overloading means → where we create same function in same, but we change either in number of parameter or we chang in type of parameter.

Don't change here in return type !!

Operator Overloading (we can overload the operator)

Syntax: return-type operator (keyword) + (function)

a + b mean → a.add(b).

(a + b) → add
a → current obj
b → i/p parameter

a → current object
+ → function call (Member func.)
b → input parameter.

(V.I) Compile Time vs Runtime polymorphism

OOPs Class - 3

Polymorphism ⇒ Run time

Run Time Polymorphism := (After execution)

When we start the execution of the code, then in the run time what polymorphism happened that is called Run time Polymorphism.

Important concept → Function (C++) / Method Overriding (Java)

★ Function Overriding

Function Overriding is a feature of object-oriented programming that happend in Run time Polymorphism, which allows a derived class to redifine a function of its base class with the help of virtual function. (virtual will apply in parent class)

Advantage
- Reusability
- Custom Behaviour

★ when virtual is not defined and we do 'upcasting', [Animal *a = new Dog(); a → speak();] , then always parent ka function call hota hai. (pointer parent type, actual object child type)

If we want, to decide function call in run time, then we have to set 'virtual' in parent's function.

Remember this 4 pattern :-

(I)
- Parent *a = new Parent()
- Parent *a = new Child()
- ~~Parent *a~~
- Child *a = new Child()
- Child *a = (Child*) new Parent()

VIRTUAL

(I) When we do upcasting & Downcasting, and without virtual keyword, then always pointer type ka method call hoga & if apply virtual then then object ka method call.

(A) If we want output depends on type of object

1) Animal* a = new Animal();
→ Animal ka constructor call hoga.

(Animal (parent), Dog (child))

2) Dog* a = new Dog();
→ Both Animal and Dog's constructor will call.
Becz, Dog depends on Animal.

3) Dog a;
→ Same as 2.

4) Animal* a = new Dog(); (animal type ka pointer, Dog type ka objcet)
→ same as 2.

5) Dog* a = (Dog*) new Animal();
→ Animal ka constructor call.

Abstraction [Implementation hiding]

* Encapsulation is a subset of Abstraction.
* In ~~abstract~~ abstraction we show essential info.

Example [I gave you the key of the car, but I have not told you how engine work etc. etc.)

(A group of many things)

Ex:- sort* a = new quicksort();

If we give it to end user then he will not know the implementation (algo), so its a abstraction.

[Abstraction means showing only the necessary information and ~~hiding~~ hiding the other irrelevant information from the user. Here implementation hiding happened.] (gfg)

Dynamic Memory Allocation

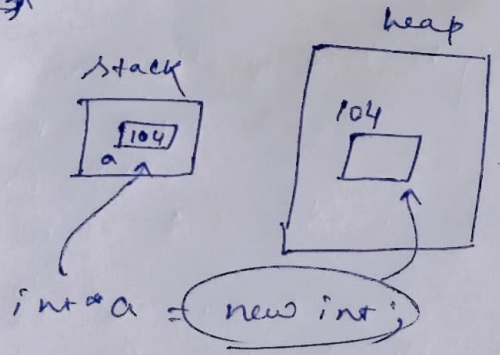
⇒ We can ~~alloc~~ allocate a program's memory in two ways → 1) ~~Static~~ Stack Memory
2) Heap Memory

- Stack Memory is generally used to store local variable + function parameter.
- By default Stack Memory is short.

(But we can change stack memory, by compiler setting during + OS configuration) → Not important

For alloting something in heap we use 'new' keyword. when we write "new int", → by writing this, ek integer block ka address share kia jata hai, so by writing this a address is returned.

and addresses are always stored in pointer thats why int* a = new int;

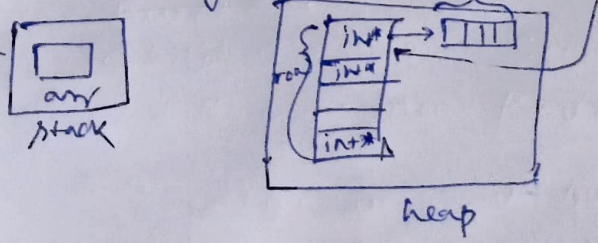


(If we want memory in Run time, then use Heap Memory)

2D Array

int **arr = new int*[n]

↳ This means we have created an array 'arr', size is n, where int* type ka data hai



int
int *a = new int;

char
~~char~~ char *ch = new char;

float
~~float~~ float *ft = new float;

Array (1D)
int * array = new int[n];

delete arr[]

If we want to deallocate in 2D array, then we have to do -

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

delete arr[]

If we want to deallocate in 2D array, then we have to do —

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