OOPs

A Programming Technique where every thing revolve around objects.

Object → Entity → State
→ Behaviour

In OOPs we can relate to Real life.

Class → Khud ka dataType

↓

userdefined DataType

Wednesday

4

int num
string str
bool abcd

→ This is pre built Data type

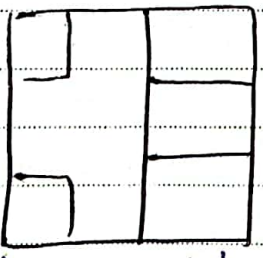
Ex of Class-

Ramesh
- int
- bool
- string.

Animal
- name
- intage
- int weight

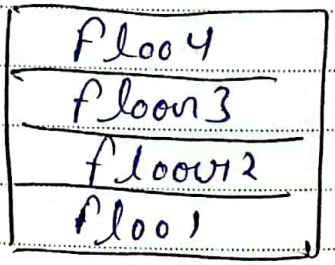
Ex

Class



Blueprint / Design of Ghr

Actual or Real ka Ghr



object

August

Sun	Mon	Tue	Wed	Thu	Fri	Sat
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

Thursday 5

Object is an instance of Class

Note - Generally empty class ka size 1 hota hai.
Because 1 means hum us class ko ek minimum size de ki ye bta ske us class ka existence hai ki nhi

Class ka track rkhne ke lie hume minimum possible memory (1) allot kr dia hai class ko

Syntax -

```
class Animal {

};
```

Friday 6

H.W

Padding & Greedy Alignment

W9-L1

OPP

- Object Oriented Programming
- programming technique in which everything revolves around objects
  - creation
  - interaction
  - access
  - changes
- 2 things
  - property / state → variables
  - behaviour / method → functions
- Why ?
  - readable
  - reusable
  - easily mantain
  - easily to understand

Class

- to make user defined / custom datatype
- has structure definition

class - design / blueprint / idea

object - actual entity

- instance of class → has existence

Static Object Creation

```
animal ramesh ;
```

Accessing state and method of object (not class)

```
objectName. state ;
objectName. method( parameters);
ex- ramesh. age ;
    ramesh . sleep();
```

Access Modifiers

→ defines access scope of state / method

3 types

public

can access that state / behaviour inside as well as outside the class

private (by default)

can access that state / behaviour only inside the class, not outside the class

protected

same as private but can be access inside class child

```
class animal {
    public :
    string name;
    string age;
    void sleep (){
        cout << name;
    }
    private :
    void eat (){
    }
};
```

→ after marking, all below state / behaviour will be of that type

Saturday

Access Modifier in C++

Public → Use inside the class or use outside the class.

Private → Use only inside the class

Note → By default class ke saare objects and behaviour Private Hote hai.

1) If we write Private → so Private ke niche jo bhi state or behaviour hai all as Private

8 Sunday

2) If we write Public → so Public ke niche jo bhi state or behaviour hai all as Public.

But if i want to access the Private member from outside

So we need to use getter & setter Function

↓ to get the value.

↓ to set the value

```
#include <iostream>
#using namespace std;

class Animal {
  Public:
    # int age;
      string name;

  // Behaviour or function
  void eat()
  {
   cout << "The dog eat " << endl;
  }
  void sleep()
    {
      cout << " The dog sleeps " << endl;
    }
};

int main() {

// Statically object Create

  Animal Dog;   Object name → Dog
  Dog.age = 20
  Dog.name = "fluppy"

cout << "The age of Animal << Dog.age << endl;
cout << "The name of Animal << Dog.name << endl;
```

Tuesday 10

11 Wednesday

If we call any Behaviour, so simply call function

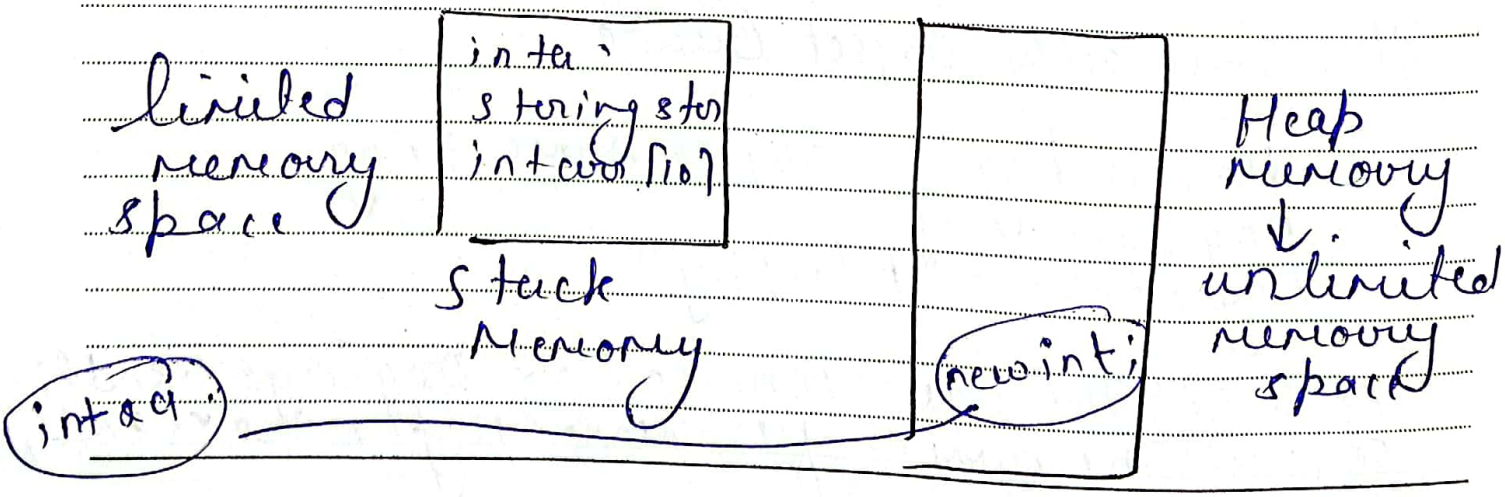
Dog.eat(); → Call function with the
Dog.sleep(); → help of object.

// Note - Jb bhi hume kisi Private number ko access krna ho from class use always
Getter and Setter function.

12 Thursday

Dynamic Memory

When we are doing Programming we have only two types of memory space



new keyword - new keyword se hum dynamic memory ko allot karte hai.

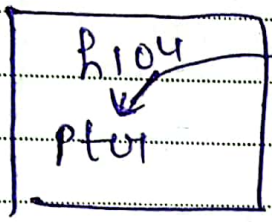
```
int *a = new int();
```

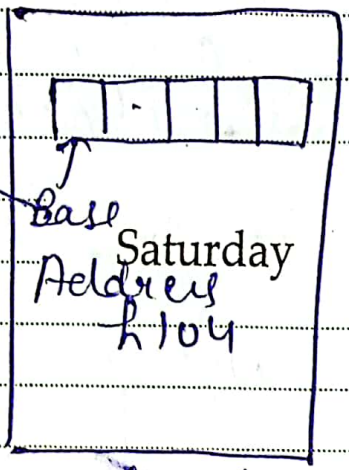
Catch

```
int *ptr = new int[5]
```

left wala part or pointer stack memory use karta hai

But Right wala new keyword heap memory use karta hai.





Saturday 14

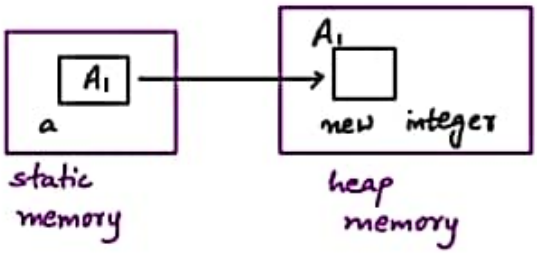
Dynamic Memory Allocation

2 types of memory

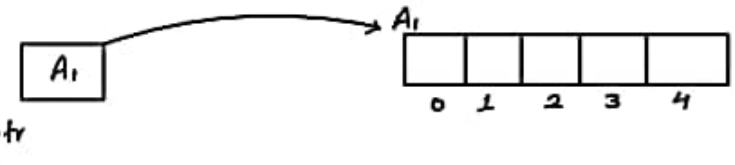
- stack
  - small
  - static memory allocation
- heap
  - large
  - dynamic memory allocation
  - new keyword
    - allocate memory in heap
    - and returns it memory location address

```
int a = 5;
```


```
int * a = new int
```




```
int * ptr = new int [5];
```



- space allocated in heap memory, cannot be cleaned automatically even after finishing of func.
- there is no garbage collector in C++

Allocation	De allocation
`int * a = new int;`	`delete a;`
`int * arr = new int [5];`	`delete arr [];`

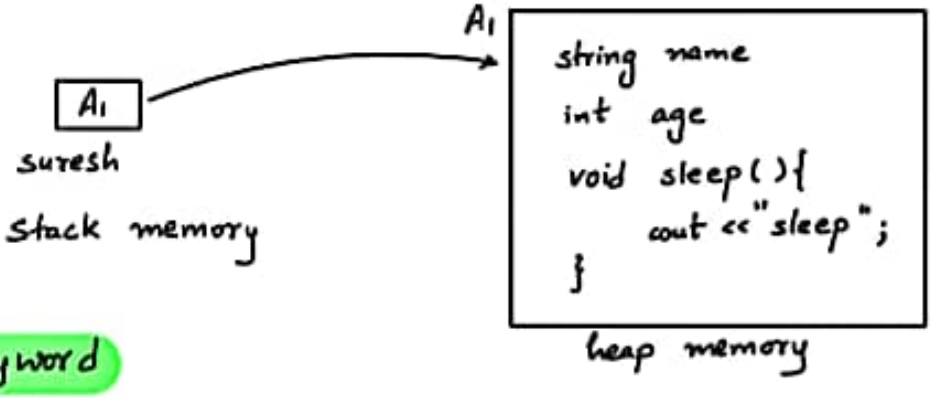
Object Creation using dynamic memory allocation

```
animal * suresh = new animal;
```

`suresh.name;` ⟶ ERROR as suresh is a pointer, * suresh is object

```
(*suresh).name;
(*suresh).sleep();
```
} 1st way of access

```
suresh -> name;
suresh -> sleep();
```
} 2nd way of access



this keyword

↳ pointer to current object

```
class animal {
    int age;
    public:
    void setAge ( int age){
        this -> age = age;
        // (*this).age = age;
    }
};
```
} both are correct

15 Sunday

					Fri	Sat
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

Difference b/w Static & Dynamic Allocation

Static Allocation

1) memory automatically release Block ka kaam khatam hone ke bad stack automatically release

Dynamic Allocation

memory automatically delete or release nhi hote hume krna pdta hai

"delete" keyword se

16 Monday

2) int arr[50]

50 x 4 = 200 byte stack me

int* arr = new int[50]

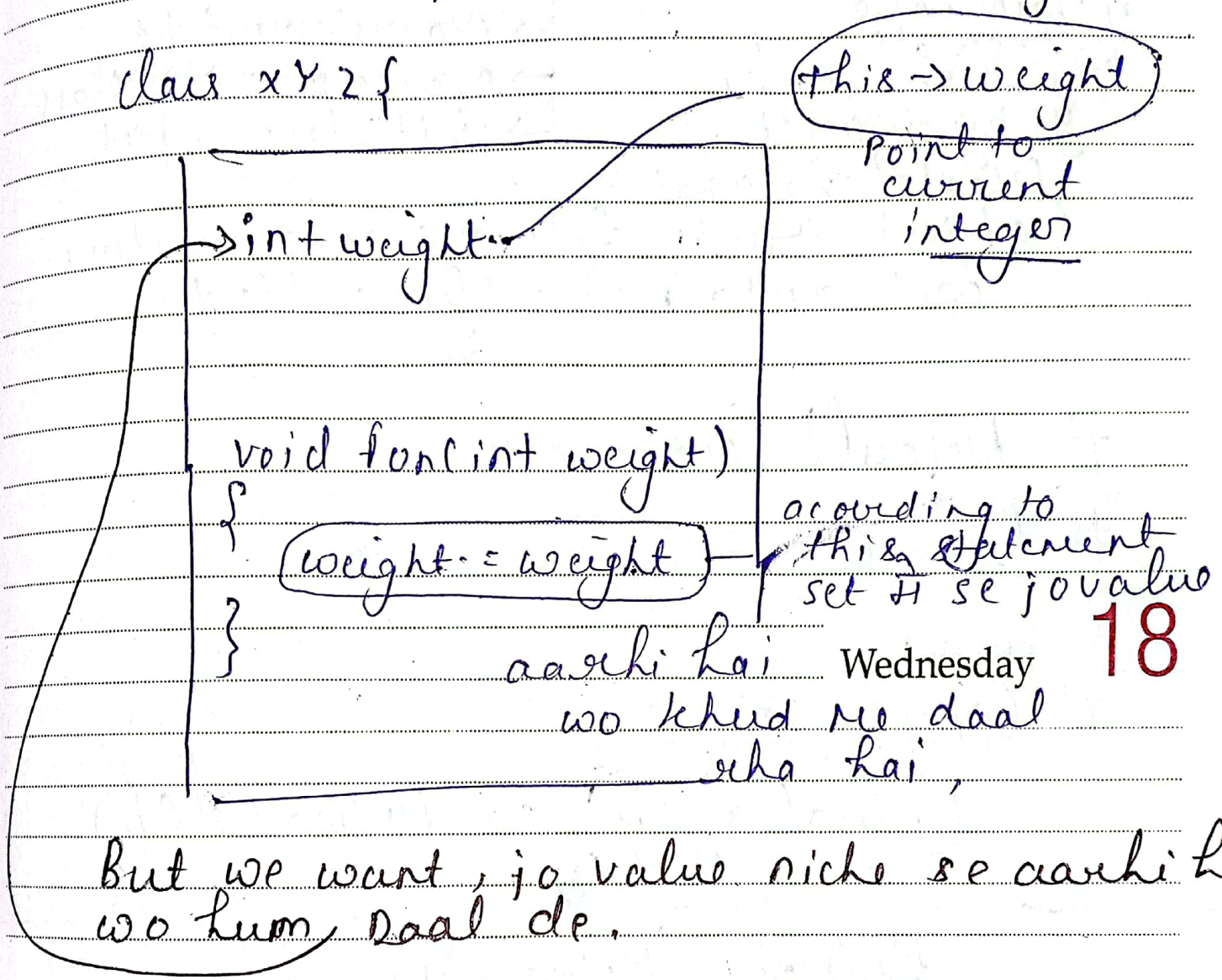
Pointer take 8 byte → stored in stack

50x4 = 200 byte

"new" keyword → for memory Allocation So Total size 208

"delete" → for memory Deallocation

this → "this keyword"
This is a pointer to current object.



But we want, jo value niche se aa rhi hai wo hum, daal de.

```
void fun(int weight)
{this->weight = weight;
}
```

19 Thursday

2018 July

Sun Mon Tue Wed Thu Fri Sat
1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30

Object Creation

used for object initialisation

- Constructor
  - → No return type
  - → Name same as class
  - → initialise object.

step 1
if hum khud se create nhi krenge then ye by default create hota hai

Object create hote hi constructor will automatically called.

Default Constructor

Parameterised Constructor

20 Friday

Copy Constructor

Animal a = b or Animal d(b)

Animal (Animal Obj)

copy Constructor

{

}

Pass by value means copy gyi hai

again call copy constructor

again & again

Infinite loop

So isliya copy Constructor me hume hamesa Call by Reference use krna chaiye.

Copy pss krenge, to wo bar bar copy Constructor ko call krega, It goes on infinite loop.

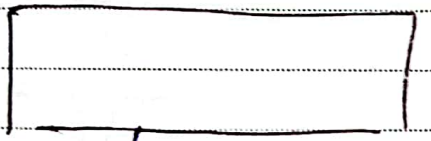
Deep Copy / Shallow Copy (Pending)

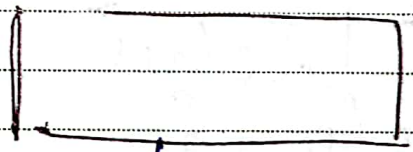
Destructor

Static Object Creation

Dynamically Object Creation

Sunday 22





Destructor automatically Called

Destructor called manually.

Destructor & Constructor both have no Return type

No input parameter for Destructor

Difference

Destructor denoted by ~

Object Creation

→ **Constructor** Call (in both static and dynamic allocation)

- → initialize the object
- → function with no return type
- → its name is same as class
- → created by default

```
class animal {
    int age;
    int weight;

    // default constructor
    animal () {
        cout << "Constructor Called";
        this -> age = 10;
        this -> weight = 20;
    }

    // parameterised constructor
    animal (int age) {
        this -> age = age;
        cout << "Parameterised constructor called";
    }
```

now built-in constructor does not called, it will be called

→ accessing members and methods using this keyword is GOOD PRACTICE

```
    // parameterised constructor 2
    animal (int age, int weight) {
        this -> age = age;
        this -> weight = weight;
        cout << "Parameterised constructor 2 called";
    }

    // copy constructor (wrong way)
    animal (animal obj) {
        this -> age = obj.age;
        this -> weight = obj.weight;
        cout << "Copy Constructor Called";
    }

    // copy constructor (right way)
    animal (animal &obj) {
        this -> age = obj.age;
        this -> weight = obj.weight;
        cout << "Copy Constructor Called";
    }
};
```

```
animal a;
animal b = new animal;
animal c = a;
animal d(a);
animal e = *b;
animal f(*b);
```

↳ copy constructor call
↳ pass by value
⇓
infinite loop

Destructor

- free memory
- for static object creation, destructor gets called automatically where object's scope ends
- for dynamic object creation, you have to do it manually using delete objName;
- no return type
- no input parameter

```
class animal {
    int age;
    int weight;

    // destructor
    ~ animal (){
        cout << "destructor called";
    }
}
```

W9-L2

4 Pillers of OOPs

- Encapsulation
- Inheritence
- Polymorphism
- Abstraction (Pitaji of all OOPs pillers)

Encapsulation

- Wrap data members and member functions in one parent entity (class)
- Objective → Data Hiding
  - security
  - privacy
  - can make readonly
  - can decide what to hide and what to show
  - work with parent entity is simple

Objective of class → Encapsulation

Perfect / Full / 100% Encapsulation

- when all data members are marked private and access them using geller and setter

Monday

4 Pillars of OOPs

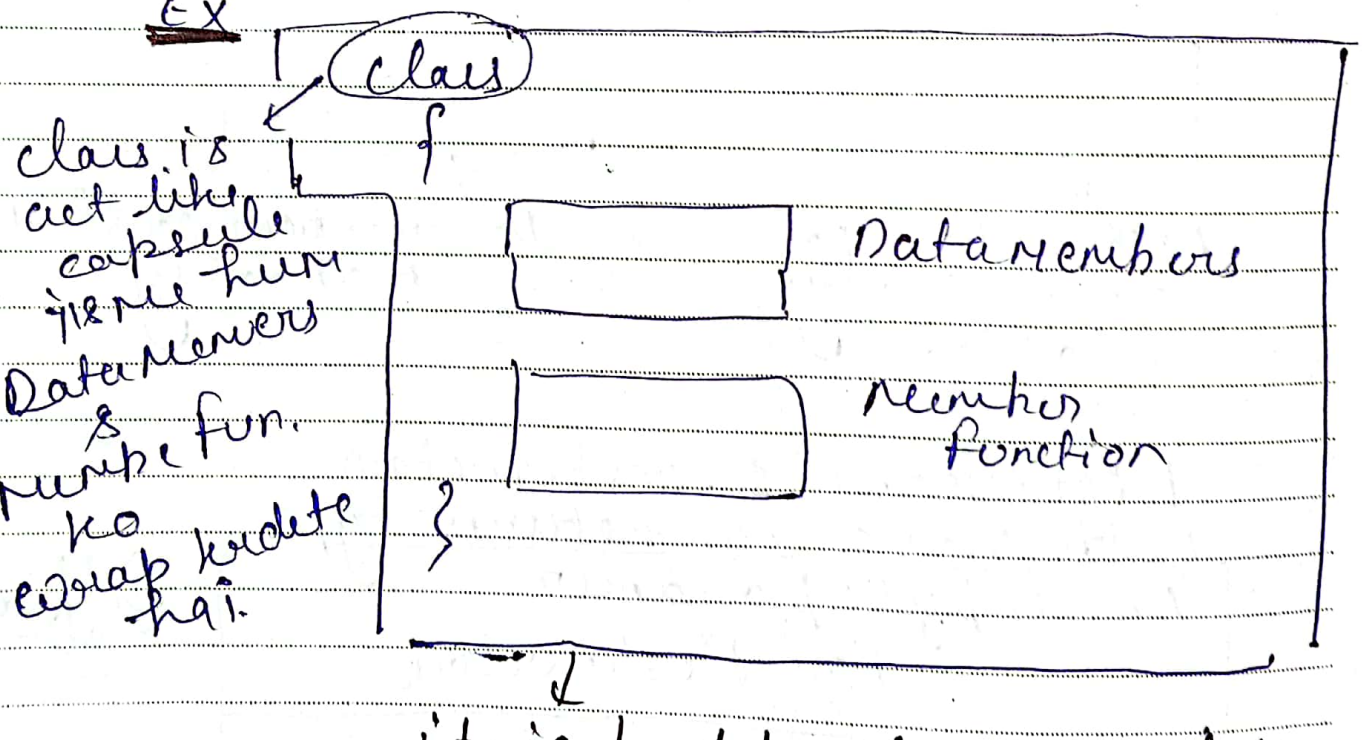
- Encapsulation
- Polymorphism
- Inheritence
- Abstraction

1) Encapsulation - Ek capsulaes ke ander hum apne data member and member function ko wrap ko deta hai.

→ Isme Data Hiding Hota hai

24 Tuesday

Ex



↓

it is by default Encapsulation hi hai.

Wednesday 25

This class concept more on less encapsulation par hi based hai

Perfect Encapsulation - When Humne saare saare data members ko ~~saare saare member functions~~ mark as private.

Ab is private member or function ko Access krenge via getters & setters.

Encapsulation → Data hiding using Access Modifier

Thursday 26

Why we use Encapsulation

1) Security → jo hum essential data user ko dikhana chahte hai, wo dikha denge baaki hide kr skte hai

2) Can make Read Only
3) Reusability.

Inheritence

- inherit some properties (data members and data functions) from parent / super / base class to sub / child / derived class
- Objective → Reusability
- is a relationship
- syntax

childName is a parentName

```
class childName : mode1 parentName 1,
                  ..., mode N parentName N {
    //additional states and methods
}
```

```
class Animal {
    int age;
public:
    int weight;

protected:
    void eat() {
        cout << "eating";
    }
};
```

```
class Dog : public Animal {
};
```

Dog is a Animal

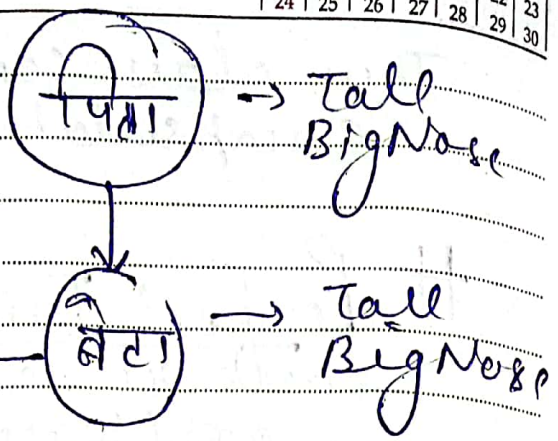
```
class cat : public Animal,
    protected Dog {

}
```

27 Friday

Sun	Mon	Tue	Wed	Thu	Fri	Sat
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

Inheritance -



Thus Beta inherit some property from its parent (Pita)

Base Class | super class | Parent Class

super class से property inherit की जाती है via child class

sub Class | child Class | Derived Class

28 Saturday # Syntax of Implement (Inheritance)

child class : _____ Parents

↓

mode of Inheritance

- Public
- Private
- Protected.

August

Sun	Mon	Tue	Wed	Thu	Fri	Sat
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

Sunday 29

Code

Parent class

```
class Animal {
Public:
    int age;
    int weight;
    void eat()
    {
        cout << "I am Eating" << endl;
    }
};
```

child class

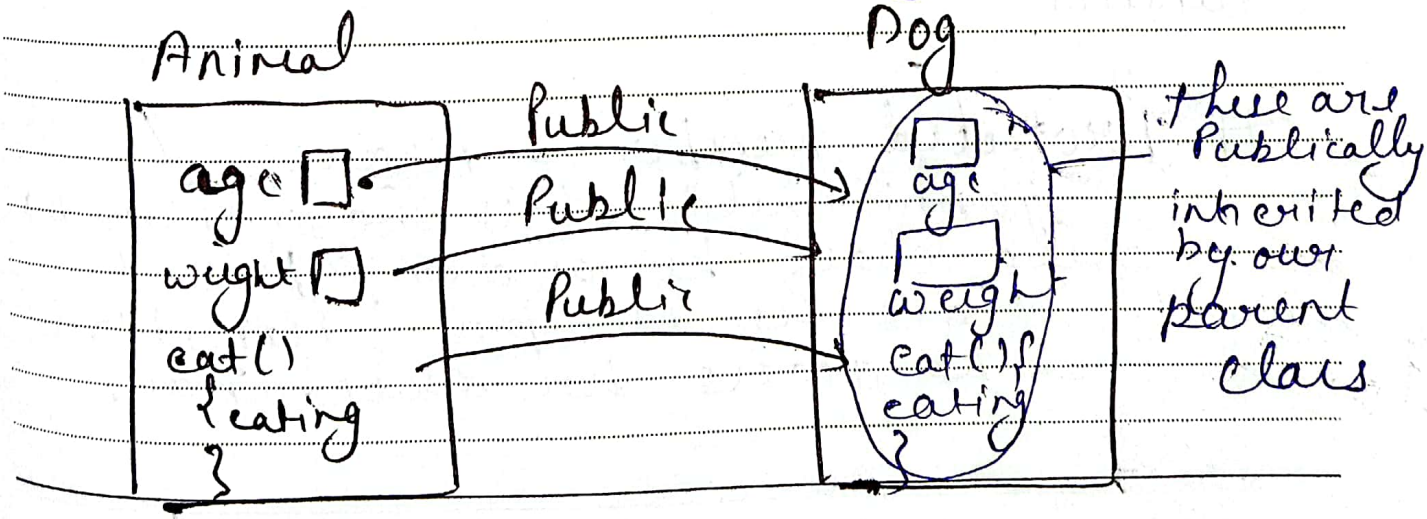
```
class Dog : Public Animal{
};
```

```
int main()
{
    Dog d1;
    d1.eat();
    return 0
}
```

Monday 30

Class Dog inherit properties from Animal class

Output - I am Eating



31 Tuesday

Important Chart

jb child class banati hai modifier declare karte hai wo yaha hai

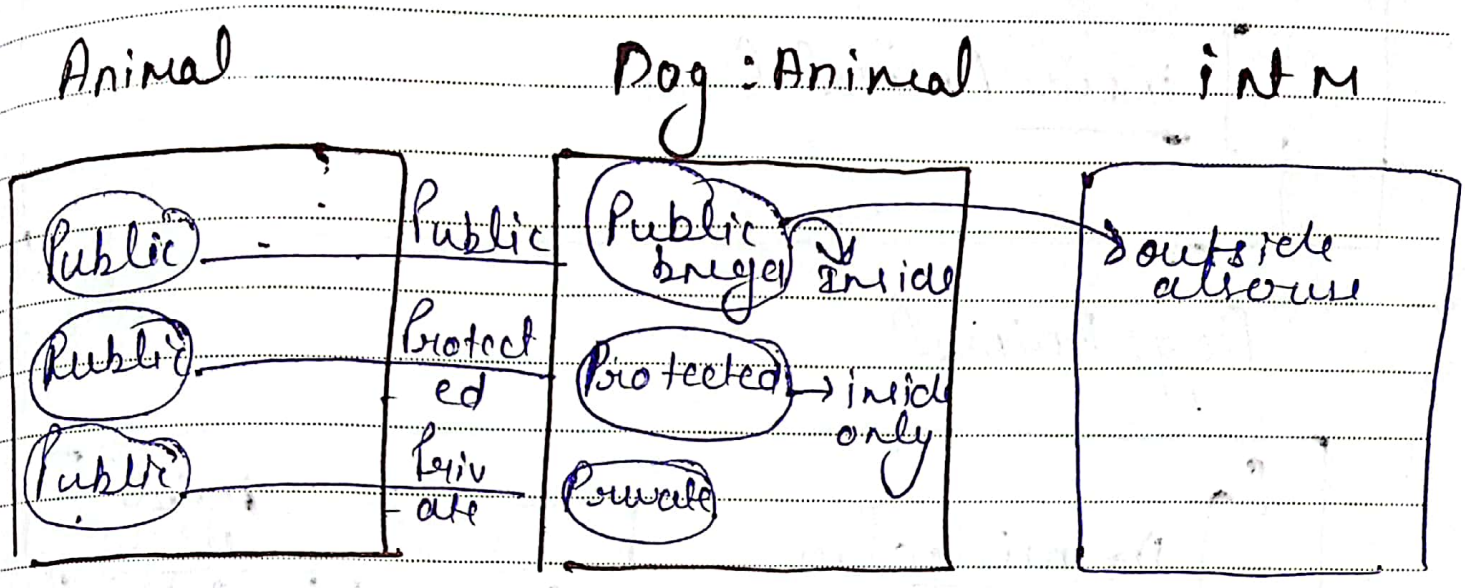
	Baseclass ka Access Modifier	Mode of Inheritance		
		Public	Protected	Private
1)	Public	Public	Protected	Private
2)	Protected	Protected	Protected	Private
3)	Private	N/A	N/A	N/A

↓ These Access modifier uses in Parent class to define member and member function

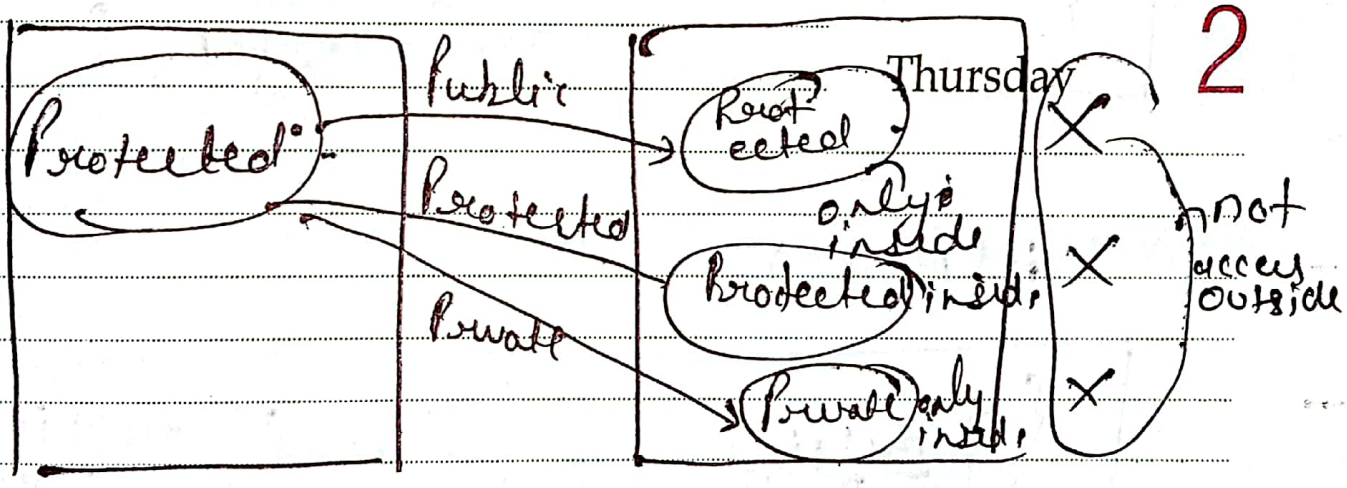
↓ yeh modifier tells, ki jo new child class hai uska modifier ka kya hoga.

Protected member → means jo bhi child class hoga hum bs child class ke ander Parent class ke member / function ko access kr skte hai.

1st Row

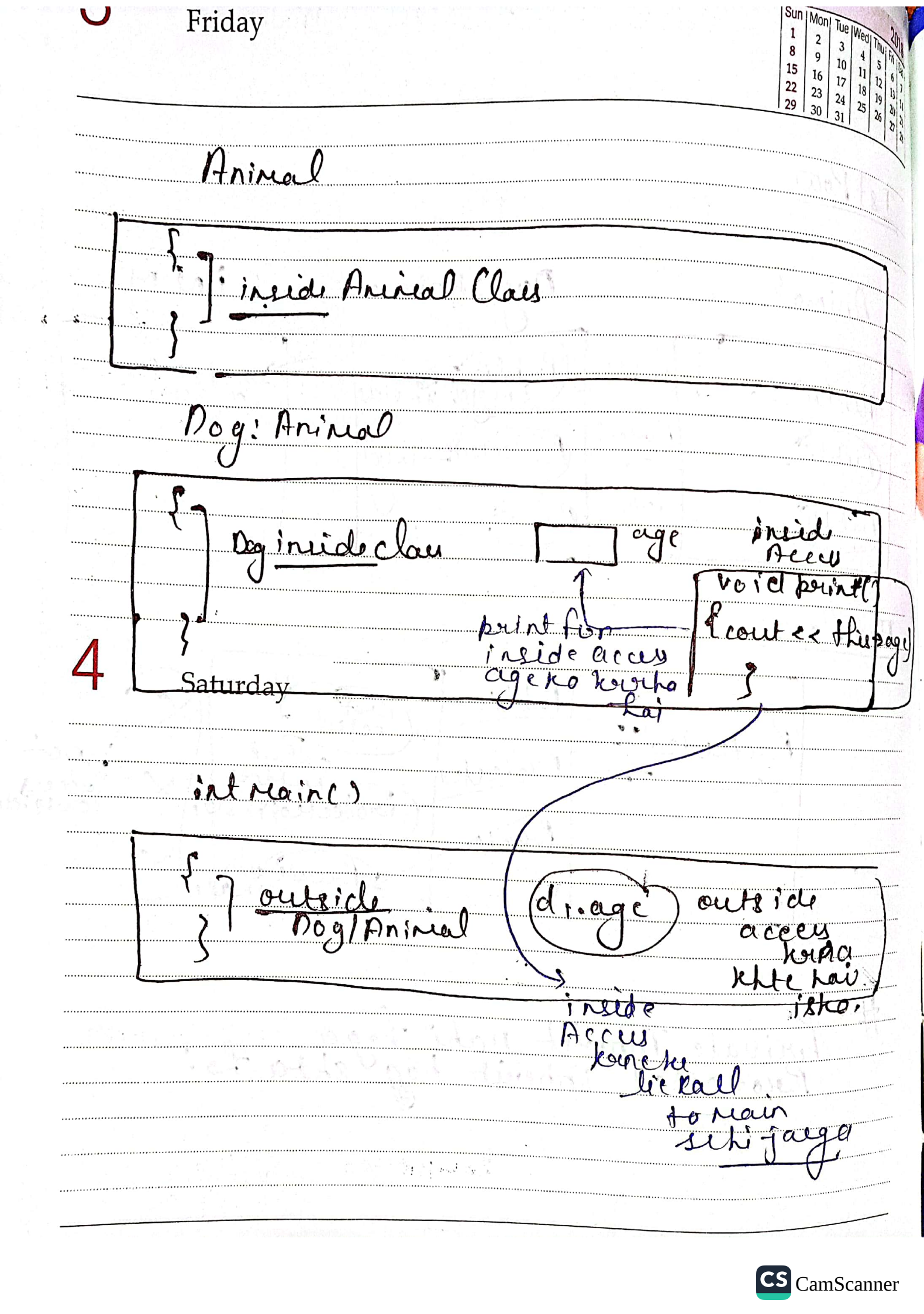


IInd Row



Private Inherit nahi hoga
Protected Inherit hoga skta hai

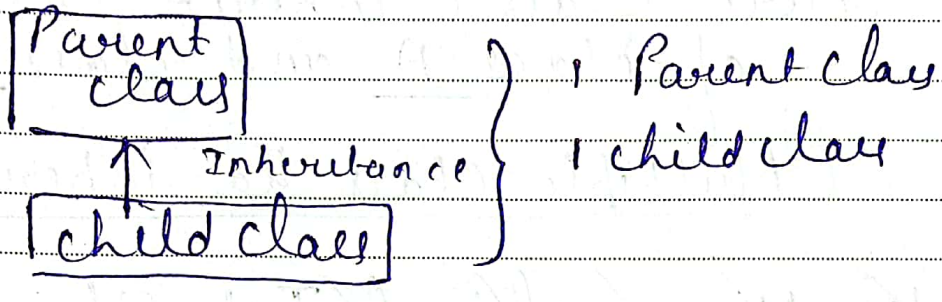
Friday


Animal
{ } inside Animal Class
Dog: Animal
{ } Dog inside class
age
inside Acces
void print()
{ cout << this->age }
print for inside access age ko kyuha kai
4
Saturday
int main()
{ } outside Dog/Animal
d.age
outside access kyuha khte hai isko.
inside Access karneke liekall to main sehi jaega

Types of Inheritance

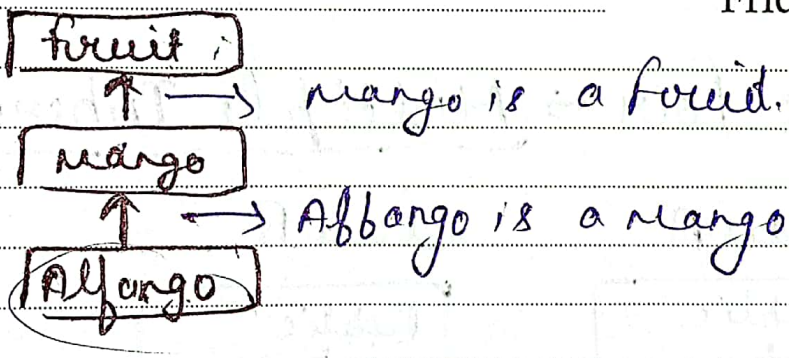
1) Single 2) Multi-level 3) Multiple
4) Hierarchical 5) Hybrid

Single



Multilevel Inheritance

Friday 10



Why we use Inheritance

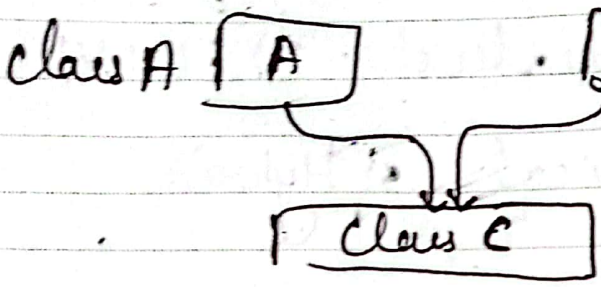
↳ Reusability of code

11 2018 August
Saturday

July 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Multiple Inheritance



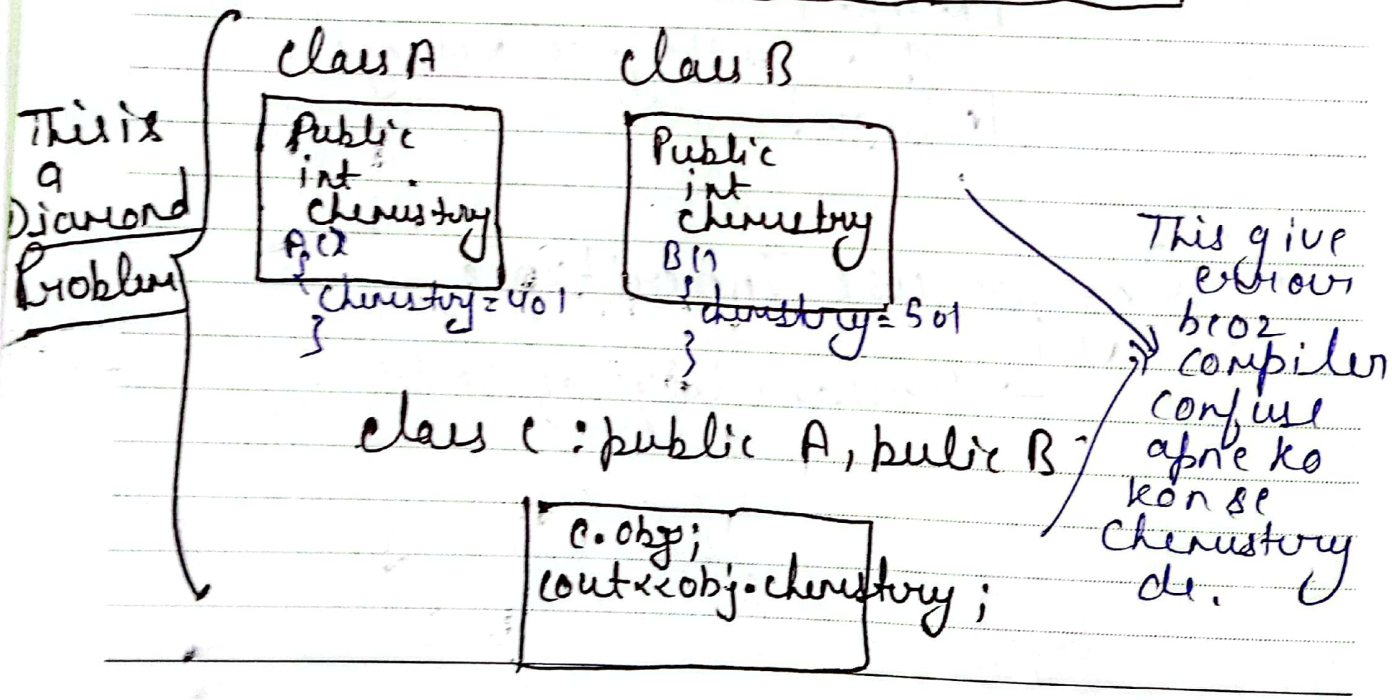
Here Class C inherit the properties of class A and class B

(Multiple class ka inheritance)

12 Sunday

Hierarchical/Inheritance

Note four → Multiple Inheritance



9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

To solve this problem
we use scope Resolution Operation

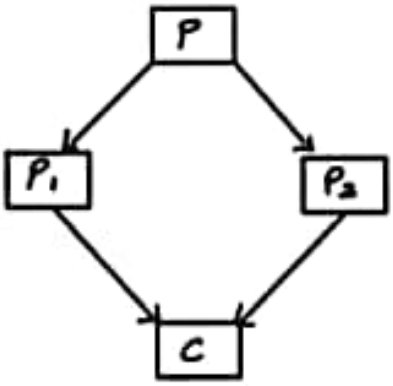
```
cout << obj.A::chemistry; → 401
cout << obj.B::chemistry; → 501
```

Diamond Problem / Inheritence Ambiguity Problem

→ If multiple parents have same property then how to access them

→ objectName . ParentClass Name :: property ;

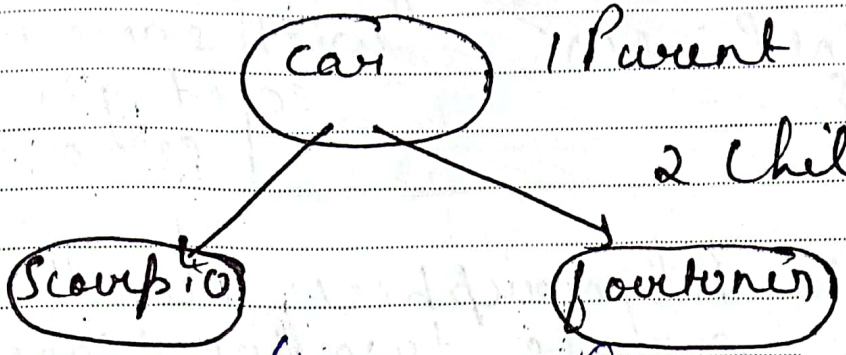
scope resolution operator



→ If parent and child have same property, then parent property inside child class will get overwrite by child property

→ Multiple inheritence is possible in C++, not in Java

Hierarchical Inheritance



Scorpio is a car

Fortuner is a car

Tuesday 14

Hierarchical Inheritance

Mixture of previous four.

Imp- Diamond problem / Inheritance Ambiguity problem.

15 Wednesday

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Polymorphism

Poly → many
morph → form
ism → existed in

Polymorphism means existed in many forms.

```
int Print()
{
}
int Print()
{
}
```

If we create 2 function with same name, so it give an error.

16 Thursday

So in Polymorphism, we can create two functions with same name without expecting error.

Types of Polymorphism

- Compile Time Polymorphism
  - function Overloading
  - Operator Overloading
- Runtime Polymorphism

Sun	Mon	Tue	Wed	Thu	Fri	Sat
30						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

Function Overloading

↳ one functions existing in Multiple forms / signature

→ Multiple signature →

↳ difference in either no. of parameter or type of parameters, but not differ in return type.

5.12 → double
5.12f → float


```
Class Math {
  Public:
    int sum (int a, int b) {
        return a+b;
    }
    int sum (int a, double b) {
        return a+b;
    }
    int sum (int a, int b, int c) {
        return a+b+c;
    }
};
```

Operator Overloading

↳ to make new implementation of an operator.

means (+) → sum krskte hai
→ concatnation krskte hai
↓
but i'm goona substract via (+)
(ulta kaam)

20 Monday

a+b → This is Binary operator

a as a current Object

(a wale object ke liye +wal operator ki definition ko class me dekha jayega)

यहाँ पर b as an input parameter gya hai

2) Runtime Polymorphism

↳ also known as Dynamic Binding

Polymorphism after execution

Function Overriding / Method Overriding

java

compile time → Overloading
Runtime → overriding

class Dog : Public Human

class - Human

```
void speak()
{
cout<< speaking
}
```

```
void speak()
{
speaking
}
void speaking
{ bark
}
```

Dog Bark

override

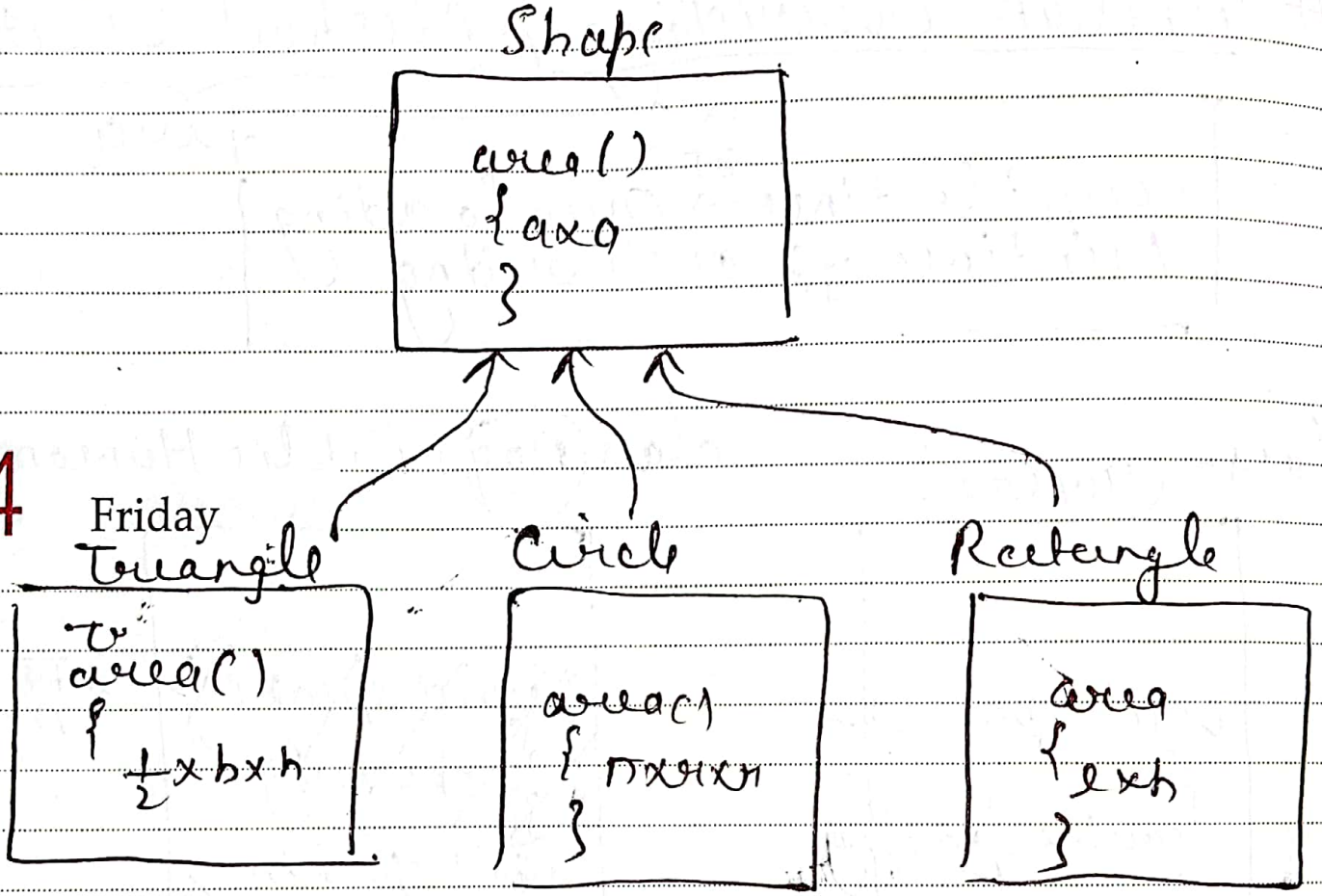
Here hum human ki property to inherit krna chahte hai, but working apne hisab se change krna chahte hai.

dog d1
d1.speaking()

23 Thursday

Parent class ke kisi function ko khud se child class ✗ dubara define karte ho
so
That is Function Overriding

Ex



24 Friday

sbne Triangle, Circle, Rectangle ne shape ko inherit kia hai bs jaha custom behaviour define krna chahte ho wo krdia,

1) Reusability of code

Saturday 25

upCasting

Parent *P1 = new child()

where child inherit property of Parent

P1.speak()
# Print Human ka member function

Down Casting

child *C1 = new Parent();

where child inherit property of parent

P1.speak() → sometime it give result, but it is compiler dependent

2) Above same ke use karke hum down casing ka Result bhi la sakte hai, bs

bs parent ka jo number hai, uske aage virtual likh do

Sunday 26

But Phir bhi agr down casing Casting krna hai to hum C++ style Cast krsakte hai

```
child* c1 = (child*) new Parent();
c1.speak();
```

Q.

4 Patterns.

1) Parent* P1 = new Parent();

2) Parent* P2 = new child();

3) Dog* d1 = new Dog();

4) Dog* d2 = (Dog*) new Dog();

28 Tuesday

Up Casting → Function of Parent class always called

Down Casing
↳ compiler dependent (Error or not)

Note - Through Up Casting we can print down casting result if we make Virtual (member function of parent)

Pointer Type

Parent * P1 = new Parent(); — object type

Parent * P2 = new Child(); — object type

* If pointer Type होता to, always parent class ko member function

* If mai ye chahta hu, ki object type pr depend kre, to we use virtual keyword

* Default behaviour of compiler jo bhi pointer ka type hoga

uske according hum uske DM / MF ko access krte hai

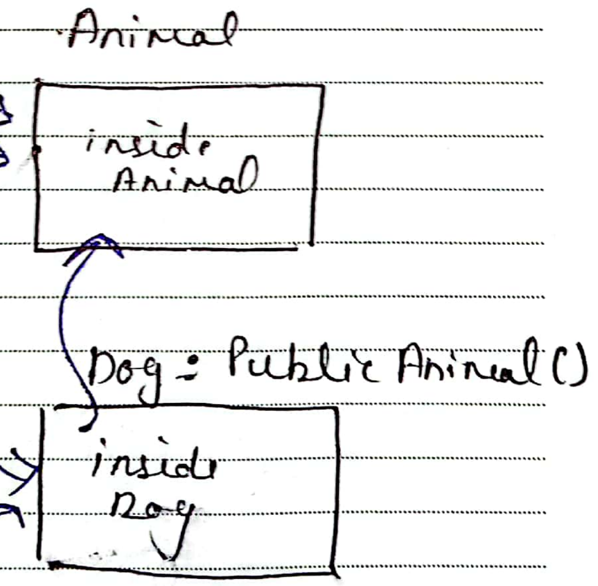
---

(1)
Animal * a = new Animal() — object Animal type ka

Dog * b = (Dog*) new Animal — obj Animal type ka

(2)
Dog * c = new Dog() — Dog object type ka Dog

Animal * d = new Dog — object Dog type ka



four 1st → print Only (Inside Animal)

four 2nd - print → (Inside Animal)
Inside Dog)

Abstraction → objective
Implementation hiding

Encapsulation is a subset of Abstraction.

Encapsulation - Data Hiding
Abstraction - Implementation Hiding

Abstraction

Encapsulation

essential info show
Ex - key → car-start
Button't know how start

Abstraction can be achieve by Access modifier
Public | Protected
Private

wrap in class (DM/MF)

Perfect Encapsulation achieved by Private accessmodifier

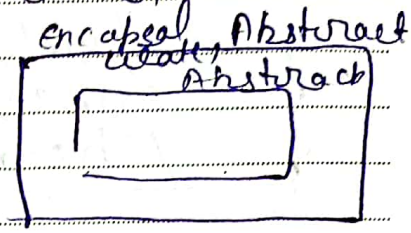
15
2018 March
Thursday

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28			

Some statement that clear-biw Encapsulation / Abstraction

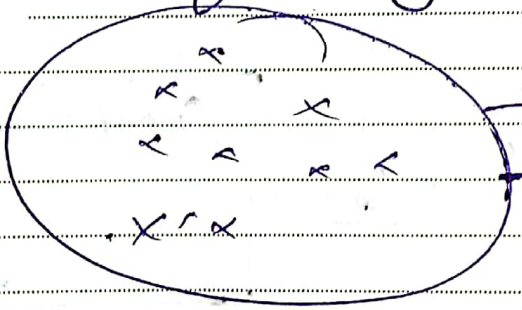
1) a container containing multiple things
↓
Abstraction & Encapsulation → Gift Box

2) a container containing container
↳ Abstraction & Encapsulation



16
Friday

3) A group of many thing



→ Abstraction & encapsulation

Ex- Abstraction → generalization → Implementation chupaee, upri level kabtuna not in detail.

Encapsulation ← wrapping or Data hiding in Parent Entity

Ex - Ek beg me hooks daal dia this is Encapsulation
But us beg me kon si hook hai wo nhi pta, this is Abstraction

Q asked in Interview

1) Difference between Encapsulation & Abstraction

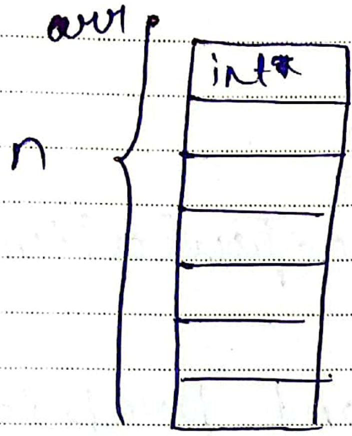
2) Inheritance ka Diamond problem. (use scope resolution) wala

3) Polymorphism
- compile time
  - function overloading
    - same function name create multiple time without expecting error
  - operator overloading (+ ki use kr ke substract)
- Runtime
  - overriding function or function overriding
    - create our own custom definition

Dynamic 2D Array

doubl Pointer ↑ (points to `**`) — array of int* ↗ (points to `int*[n]`)

```
int ** arr = new int*[n]
```



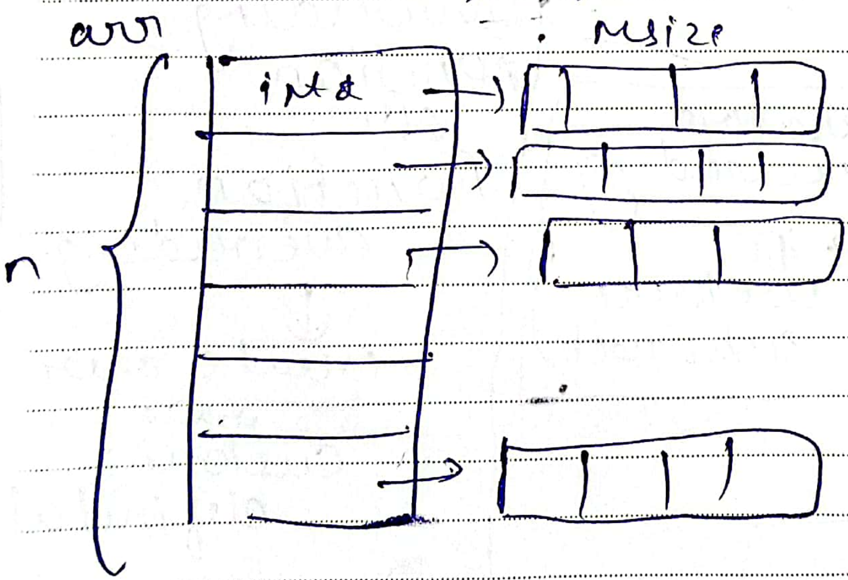
ek Array bnaya size n hai usme int* type ka data hai aur array ka name arr hai

left me double pointer bcoz left me Pointer hai, means ek pointer ko store karne ke liye left me ** ka


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

arr[i] → Pointer

new int[M] → integer array of size M



Wednesday 21

Code

```
int** arr = new int*[row]
for(int i=0; i<row; i++)
    arr[i] = new int[col]
    }
```

↓

2D Array → arr[row][col]

for delete

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

Thursday 22

+

```
delete[] arr
```

23

2018 March

Friday

February 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28			

Encapsulation → Binding

Abstraction → Hiding

1) What is implicit return type of constructor

A class object in which it defined.

2) Data hiding, Data Binding, Message Passing are all inside OOPs concept