Object Oriented Programming

Q what is OOP?

Ans: Procedural programming is about writing procedures or functions that perform operations on the data, while object-oriented programming is about creating objects that contain both data and functions.

Opps has several advantages over procedural programming:

1) OOP is faster and easier to execute.
2) OOP provides a clear structure for the programs.
3) OOP helps to keep the C++ code DRY "Don't Repeat Yourself", and makes that code easier to maintain, modify and debug.
4) OOP makes it possible to create full reusable applications with less code and shorter development time.

Q Merit of OOP

Ans ① Programs written in OOP language are easy to understand.

② since everything is treated as objects, so we can model a real world concept using OOP's.

③ OOP provides feature of reusability we can reuse the classes that are already created without writting them again & again. (To classes pehle se bani hue hain, unke code ko reuse kr skte hai using inheritance).

④ Since the parallel development of classes is possible in OOP concept, it results in the quick development of the complete programe. (ek class kisi aur ko dedi implement k liye dusri kisi aur ko) aise development ki speed badh jati hai

⑤ Program written in OOP languages are easy to test, manage & maintain.
(Kyunki har cheej obj k forum hoti hai to errors find karna easy hota hai)

⑥ It is secured development technique for programe designing.
Since data is hidden & can't be accessed by external func.
(secured development because of data hiding)

⑦ Object oriented systems can be easily upgraded from small to large system.
(ek class jo pehle se bani hui hai usmein new feature add karne ke liye ek aur class bna skte hai, using inheritance)
add bhi kr skte hai

⑧ It is easy to partitions the work in project based on objects.

⑨ we can build programs from the standard working modules rather than having to start writing the code from beginning. This leads to saving of development time. Because of this we can develop more projects in less time.

3Q Demerits of OOPs

Sol:- (1) The programmer should have a proper planning before designing a program using OOP approach.

(2) Since everything is treated as objects in OOP, the programmers need proper skill such as design skills, programming skills, thing in terms of objects etc.

(3) The size of programs developed with OOP is larger than the procedural approach.
(Kyuki har bari class banani pdegi uske under function banane pdenge but procedural mein direct function banadete hai).

(4) Since object oriented programs are larger in size, that results slower execution of programe.
(Kyuki compile time and execution time zyada lagega)

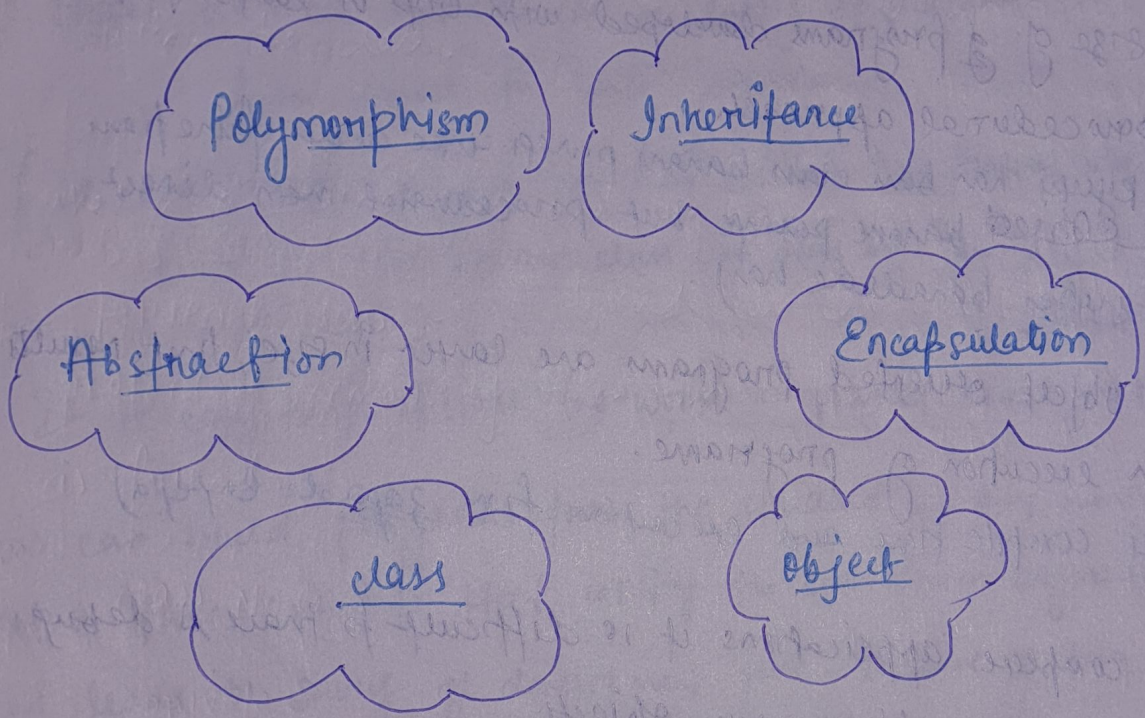
(5) For complexe applications it is difficult to trace & debug message passing b/w many objects.
(Error checking difficult hoga complexe applications mein.)

4Q what is the aim of OOPs?

Solve: The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

~~OOPs is characteristics~~

characteristics of an OOP.



5Q. what is class?

Solve: The building block of C++ that leads to Object-Oriented Programming is a class. It is a user-defined data-type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

- A class is a user-defined data-type which has data members and member function.
- Data members are the data variable and member functions are the functions used to manipulate these variables and together these data members and member functions define the properties and behaviour of the objects in a class.

6Q. what are objects?

Soln: Objects are instance of a class these are defined as users define data types. Object take up space in memory and have an associated address. we declare objects of a class with exactly the same sort of declaration that are declare variables of basic types.

Access modifiers

- used to implement important feature of OOP known as Data Hiding.
- Access modifiers / Access Specifiers are used to set the accessibility of the class members. That is, it sets some restrictions on the class members not to get directly accessed by the outside function.

There are three access modifiers:

Public → the property or method (class members) can be accessed from anywhere.

Protected → The property or method (class members) can be accessed within the class and by the classes derived from that class.

private → The class members can ONLY by accessed within the class.

7Q. what is a friend class and function in C++.?

Solut Friend class :- A friend class can access private and protected members of other class in which it is declared as friend.

Eg:-

```
class Node {
    private:
      int Key;
      Node * next;

    friend class LinkedList ; // Now class LinkedList can
                              access private members of Node
};
```

Friend function :- Like friend class, a friend function can be given special grant to access private and protected members.

A friend function can be :

a) A function of another class.

b) A global function.

Eg:

```
class Node {
    private :
        int key;
        Node *next;

    friend int LinkedList::search();
};
```

8Q. what is Gretters and Setters ?

Solu:- Gretters and Setters allow you to effectively protect your data. This is a technique used greatly when creating classes. For each variable, a get() method will return its value and a set() method will set the value.

Inheritance

- Capability of a class to derive properties and characteristics from another class.
- Sub class : The class that inherits properties from another class is called Sub class on derive class.
- Super class : The class whose properties are inherited by sub-class is called Base class on Super class.

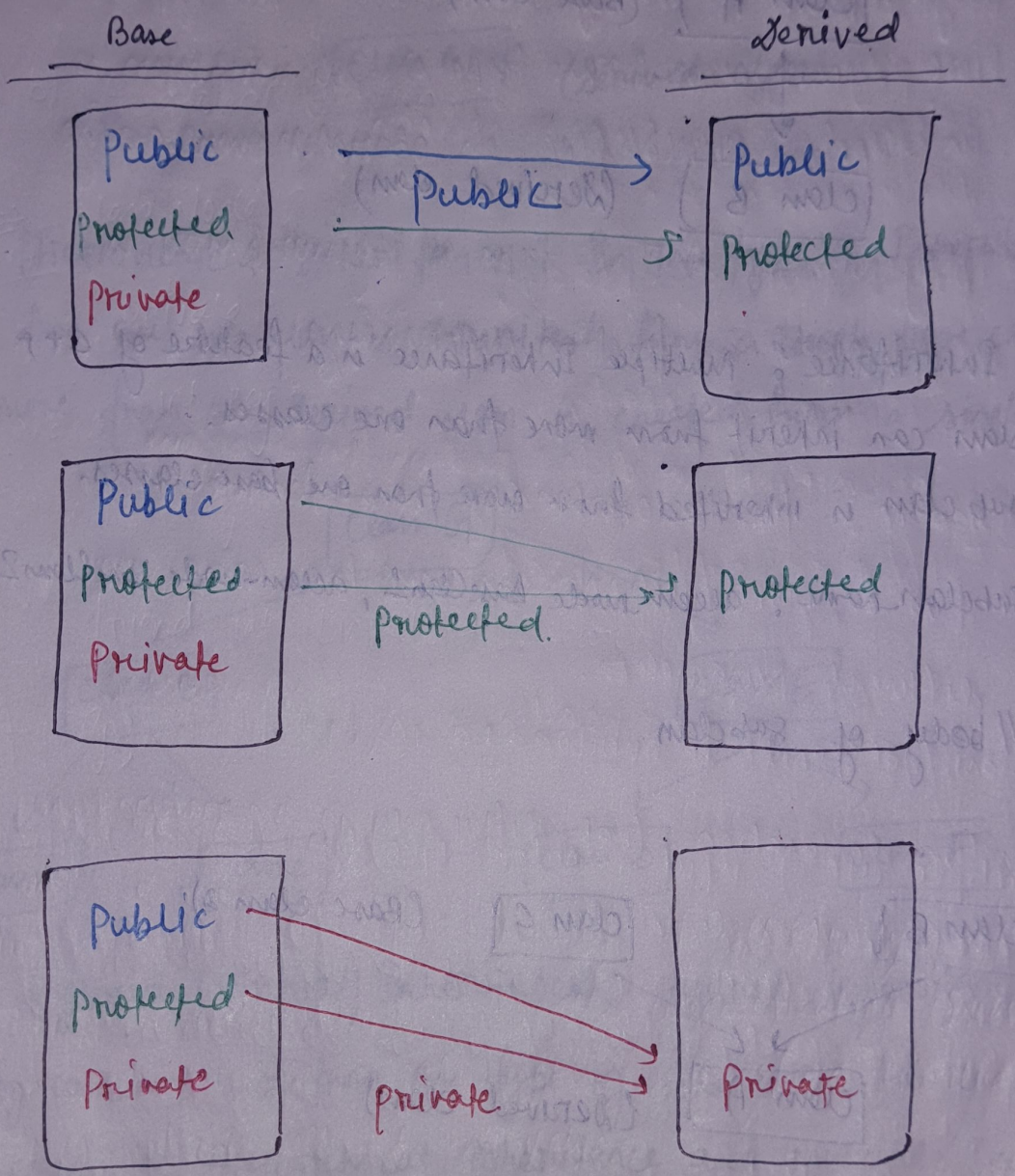
Syntax :-

```
class subclass_name : access_mode baseclass_name
{
    // body of subclass
};
```

Mode of Inheritance :

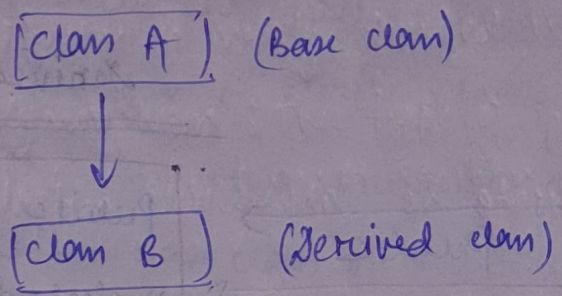
- Public mode :- If we derive a sub class from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in derived class. private members of the base case will never get inherited in sub class.
- Protected mode :- If we derive a sub class from a Protected base class. Then both public member and protected members of the base class will become protected in ~~no~~ derived class. private members of the base class will never get inherited in sub class.

- Private mode :- If we derive a sub class from private base class. Then both public member and protected members of the base class will become private in derived class. Private members of the base class will never get inherited in sub class.



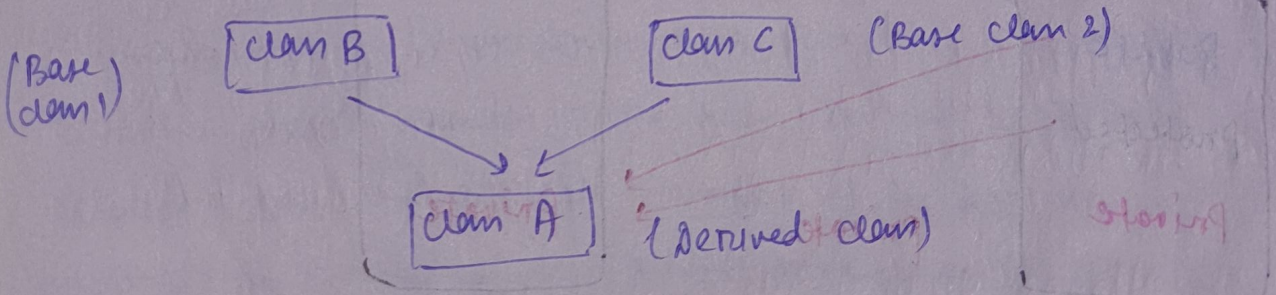
Types of Inheritance

1. Single Inheritance : A class is allowed to inherit from only one class. i.e one sub class is inherited by one base class only.

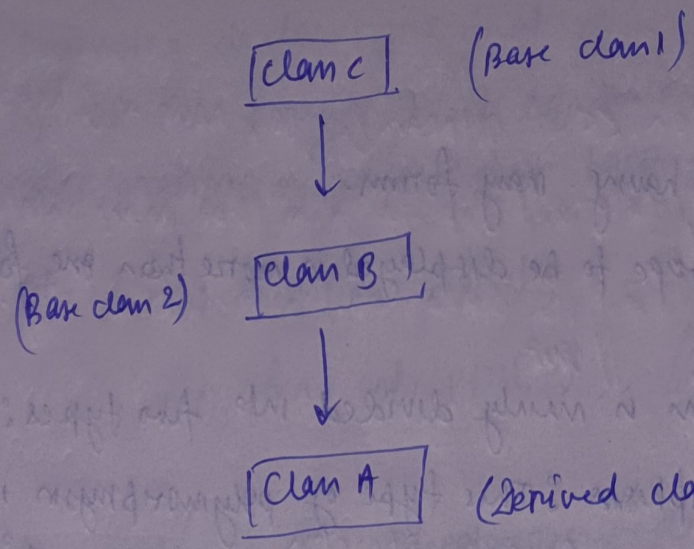


2. Multiple Inheritance : Multiple Inheritance is a feature of C++ where a class can inherit from more than one classes.
i.e one sub class is inherited from more than one base classes.

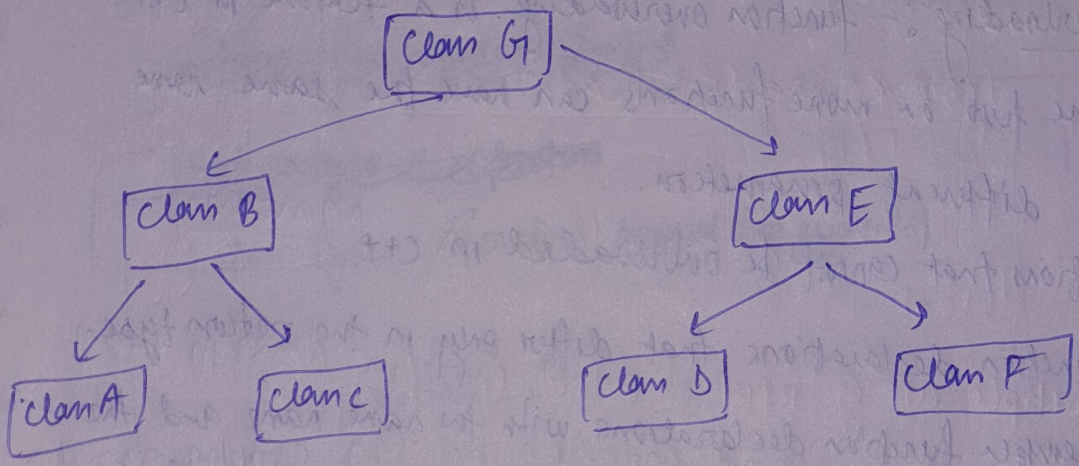
```
class subclass_name : access_mode baseClass1, access-mode baseClass2
{
  // body of subclass.
};
```



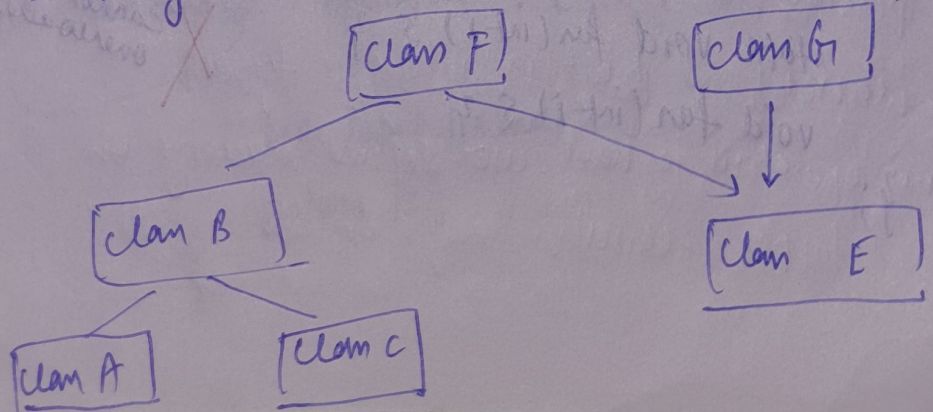
3. Multilevel Inheritance : In this type of Inheritance, a derived class is created from another derived class.



4. Hierarchical Inheritance: In this type of Inheritance, more than one sub class is inherited from a single base class. i.e more than one derived class is created from a single base class.



5. Hybrid (Virtual) Inheritance: Hybrid inheritance is implemented by combining more than one type of inheritance. For eg. combining Hierarchical inheritance and Multiple Inheritance.



Polymorphism

- Polymorphism means having many forms.
- The ability of a message to be displayed in more than one forms.

→ In C++ polymorphism is mainly divided into two types:-

1. **Compile time polymorphism**: This type of polymorphism is achieved by **function overloading** or **operator overloading**.
2. **Runtime polymorphism**: This type of polymorphism is achieved by **function overriding**.

function overloading:- function overloading is a feature in C++ where two or more functions can have the same name but different parameters.

→ functions that cannot be overloaded in C++

- Function declarations that differ only in the return type.
- Member function declarations with the same name and the same parameter-type-list cannot be overloaded if any of them is a static member function declaration.

eg:-

```
class Test {
    static void fun(int i) { }
    void fun(int i) { }
};
```

✗ cannot be overloaded.

- parameter declarations that differ only in a pointer * versus an array [] are equivalent. That is, the array declaration is adjusted to become a pointer declaration.

eg
```
int fun(int *ptr);
int fun(int ptr[]); // redeclaration of fun(int *ptr)
```
✗ cannot be overloaded. !!

operator overloading :- Operator overloading is a compile-time polymorphism in which the operator is overloaded or redefined to provide the special meaning to the user-defined data-type.

→ Syntax

~~return type operator symbol~~

~~return type operator~~

```
return type operator symbol (arguments)
{
    // operations ...
}
```

SYNTAX

cascading of operator (next page)

cin >> head;

eg)
```
istream& operator >> (istream& is, node *& head)
{
    take Input (head);
    return is; // cin
}
```

eg:

```
ostream& operator<<(ostream& os, node* &head){
    print(head);
    return os;  // cout
}
```

this is called cascading of operator

```
cout << head1 << head2;
```

⇓

```
cout << head2;
```

otherwise, if we didn't return this, it become

```
cout << head1 << head2;
```

⇓

```
void << head2;   // ✗ No meaning
```

The multiple use of input or output operators ("<<" or ">>") in one statement is called cascading of I/O operator. → input/output

benefits of cascading operator overloading

→ It makes it possible for templates to work equally well with classes and built-in/intrinsic types.

→ It allows C++ operators to have user-defined meaning on user-defined types (classes).

function overriding :- It occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be overridden.

Eg:

```
class Parent {
    public:
    void print() {
        cout << "The parent was called" << endl;
    }
};
class child : public Parent {
    public:
    //definition of a member function already present in parent
    void print() {
        cout << "The child was called" << endl;
    }
};
int main() {
    parent obj1;
    child obj2;
    obj1.print();
    obj2.print();
    return 0;
}
```

Output :

```
The parent was called
The child was called
```

Encapsulation

→ Binding together the data and the functions that manipulates them.
→ Encapsulation also lead to data abstraction or hiding.

```
class Encapsulation {

    private:
      // data hidden from outside world
      int x;

    public:
      void set (int a) {
            x = a;
      }
      int get () {
          return x;
      }

};
```

→ The variable x is made private. This variable can be accessed and manipulated only using the functions get() and set() which are present inside the class. Thus we can say that here, the variable x and the functions get() and set() are binded together which is nothing but encapsulation.

Abstraction

- Abstraction means displaying only essential information and hiding the detail.
- Data abstraction refers to providing only essential information abt the data to the outside world, hiding the background details or implementation.

Example

```
class implementAbstraction {

    private:
      int a, b;
    public:
      void set (int x, int y);
          a = x;
          b = y;
      }
      void display() {
          cout<< "a = " <<a<<endl;
          cout << "b = " <<b <<endl;
      }
};
```

→ You can see in the above program we are not allowed to access the variables a and b directly, however one can call the function set() to set the values in a and b and the function display() to display the values of a and b.

Since, it helps to increase security of an application or program as only important details are provided to the user.

Extra (leftover)

Virtual function :- A member function in the base class which is declared using virtual keyword is called virtual func. They can be redefined in the derived class.

Eg To demonstrate the concept of virtual function an example program is shown below.

```
#include<iostream.h>
#using namespace std;
class BaseClass {
    public:
    int var_base = 1;
    virtual void display() {
        cout<<"1 Display Base class"<<endl;
    }
};
class DerivedClass : public BaseClass {
    public:
    int var_derived = 2;
    void display() {
        cout<<"2 Display Base class"<<endl;
    } cout<<"2 Display derived class"<<endl;
};
```

```
int main()
{
    Base clan * base_clan_pointer;
    Base clan   obj_base;
    Derived clan  obj_derived;

    base_clan_pointer = & obj_derived;
    base_clan_pointer -> display();
    return 0;
}
```

Output:

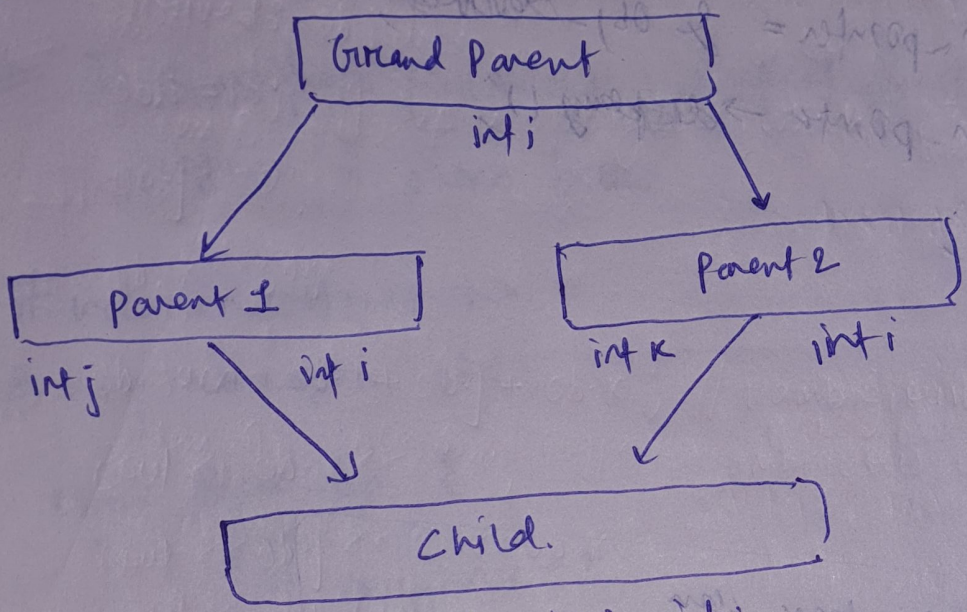
```
2 displaying base clan
2. displaying derived clan.
```

If we don't use virtual function, then the pointer would have pointed to our base clan's display function even if it is pointing to derived clan (This is its default behaviour) (see code with Harry if confused)

Q. what is multipath Inheritance?

Sol: multipath inheritance is a hybrid inheritance.
It is combination of hierarchical inheritance and multiple inheritance.

Eg



(Ambiguity in multiple inheritance)

Q. what Ambiguity problem in multipath Inheritance?

Sol: Suppose GrandParent has a data member int i. parent 1 has a data member int j. Another Parent 2 has a data member int k. Child class which is inherited from parent1 and Parent2.

child class have data member:

int j (one copy of data member Parent 1)

int k (one copy of data member Parent 2)

int i (two copies of data member Grandparent)

↳ This is ambiguity problem. In child class have two copies of base class. There are two duplicate copies of int i of base class. one copy through Parent 1 and another copy from Parent 2.

This problem is also called as Diamond problem.

⟹ Demonstration of ambiguity in multipath inheritance.

// This program contains an error and will not compile.

```
#include<iostream>
class base {
  public:
   int i;
};
class derived1 : public base
  { public:
     int j;
};
class derived2 : public base
  { public:
     int k;
};
```

```
class derived3 : public derived1, public derived2
{
  public:
    int sum;
};
void main()
{
 derived3 ob;
 ob.i = 10;  // this is ambiguous, which i ??
 ob.j = 20;
 ob.k = 30;
 ob.sum = ob.i + ob.j + ob.k;  // i ambigious here, too
 cout << ob.i << " ";  // also ambiguous, which i ?
 cout << ob.j << " " << ob.k << " ";
 cout << ob.sum;
}
```

Solution:- There are two ways to remedy the preceding program. The first is to apply the scope resolution operator to i and manually select one i. The second is to use virtual base class.

(1) way Remove ambiguity using scope resolution operator.

In above program, the ambiguous statements.

```
ob.i = 10;
ob.sum = ob.i + ob.j + ob.k;
cout << ob.i << " ";
```

} ⇒ replaced by

```
ob.derived1::i = 10;
ob.sum = ob.derived1::i + ob.j + ob.k;
cout << ob.derived1::i << " ";
```

(2) ~~Indwg~~ Remove using virtual base class :-

Virtual Base Case :- The virtual base class is a concept used in multiple inheritance to prevent ambiguity between multiple instances.

eg.

```
#include <iostream>
class base {
  public:
  int i;
};
class derived1 : virtual public base
{
  public:
  int j;
};
class derived2 : virtual public base
{
  public:
  int k;
};
```

↓ same as in previous case (NO compile error).

(any problem see CWH (web wh--))

Construction and Destruction Execution in Inheritance

When an object of a derived class is created, if the base class contains a constructor, it will be called first, followed by the derived class' constructor. When an derived object is destroyed, its destructor is called first, followed by the base class destructor, if it exists.

eg1 Mode of inheritance

```
class B : public A
{

};
```

order of execution

A(); base constructor
B(); derived constructor

eg2

```
class A : public B, public C
{

};
```

B(); base (first)
C(); base (second)
A(); derived

eg3

```
class A : public B, virtual public C
{

};
```

C(); virtual base
B(); ordinary base
A(); derived

eg4 order of inheritance

class C (Base class 1)
↓
class B (Base class 2)
↓
class A (Derived class)

Order of constructor call

1) C()
2) B()
3) A()

order of destructor call

1) ~A()
2) ~B()
3) ~C().

How to call the parameterized constructor of base class in derived class constructor?

Solu: To call the parameterized constructor of base class when derived class's parameterized constructor is called, you have to explicitly specify the base class's parameterized constructor in derived class as shown in below program:

```
// base class
class Parent {
  public:
    // base class's parameterized const.
    Parent(int i) {
      int x = i;
      cout << "Inside base class's parameterized constructor" << endl;
    }
};
// sub class
class Child : public Parent {
  public:
  // sub class parameterized constructor
  Child(int j) : Parent(j) {
      cout << "Inside sub class's parameterized const" << endl;
  }
};
```

Structure Vs class

Structure

eg

```
struct number {
    float x;
};
```

→ initial variables declaration.

① way.

```
struct point {
    int x, y;
} p1;  // the variable p1 is declared with point.
```

② nd way.

```
struct point {
    int x, y;
};
int main ()
{
    struct Point P1;  // the variable P1 is declared like a non variable
}
```

→ Limitation of Structure

- No data Hiding : C structure do not permit data hiding. Structure members can be accessed by any function, anywhere in the scope of the structure.
- functions inside Structure : C structures do not permit functions inside structure.
- Static members : C structures cannot have static members inside their body.
- Access Modifiers : C programming language do not support access modifiers, so they cannot be used in C structures.
- Construction creation in Structure : Structures in C cannot have constructor inside structures.

Static data member → also called class variable

→ Static data members are class members that are declared using static keywords. A static member has certain special characteristics. These are :

- Only one copy of that member is created for entire class and is shared by all the objects of that class, no matter how many objects are created.
- It doesn't need any object to access it.

we need to initialize static data member before using it.

Eg

```
Hero {

    int value = 10;
    int health = 70;
    static int timetocomplete;

};
```

→ Now to initialize the static member

```
datatype className :: fieldname = value;
```

↳ scope resolution operator

Eg

```
int Hero :: timetocomplete = 5;
```

⇒ we don't need object to use it.

Eg

```
int main() {

    Hero a;
    cout << Hero :: timetocomplete << endl;
                ↳ aise bhi likh sakte hai
                cout << a.timetocomplete << endl;
}
```

Static member function

→ A member function that is defined using the static keyword.
→ These functions are special functions used to access the static data members or other static member functions.
→ It can be accessed by class name and not by object's name.
ie. class-name :: function-name;
→ A static member function doesn't have this pointer.
→ There cannot be static and non-static versions of same func.
→ A static member function may not be virtual / const.

```
class Hero {
    int value = 10;
    static int timeToComplete;
    static int random() {
        return timeToComplete;
    }
};

int Hero::timeToComplete = 5;

int main() {
    cout << Hero::random() << endl;
}
```

(dynamic memory allocation ⇕ new and delete)

Dynamic Constructor

→ when allocation of memory is done dynamically using dynamic memory allocator new in a constructor, it is known as dynamic constructor.

```
// class geeks {
    const char *p;
public:
    // default constructor
    geeks()
    {
        p = new char[6];
        p = "geeks";
    }

};
```

Object based programming

→ It doesn't support all the features of OOPs like polymorphism and Inheritance

→ It does have built in object

→ eg → JavaScript, VB etc.

Object oriented Programming

→ It supports all the features of OOPs.

→ It doesn't have built in object.

→ eg of C++, C#, Java.

Pure object oriented language is fully object oriented language which supports & have features which treats everything inside program as objects.
It doesn't support primitive data type (like int, char, float, bool, etc).

eg Smalltalk

A private constructor is a special instance constructor.
It is generally used in classes that contain static member only. If a class has one or more private constructors and no public constructor, other class also cannot create instances of this class.