OBJECT ORIENTED PROGRAMMING

●**<u>Class</u>** is a user-defined data type.Class does not occupy any memory space at runtime till the time an object is instantiated(made).
Class members consist of Data Member(Variable) and Member functions(functions/methods).

●**<u>Object</u>** : Note : When an object is created using new keyword, then space is allocated for the variable in a heap, and the starting address is stored in the stack memory.
When an object is created without a new keyword, then space is not allocated in the heap memory, and the object contains the null value in the stack.

●**<u>Overloading</u>** : To have more than 1 function of same name we need to have different types or different numbers of parameters/arguments passed to that function.

Functions which cannot be overloaded :
•Function that differ only in return type.
•function that differ only by static or const keyword.
•int fun(int i=1),int fun(int i).
•int fun(int *ptr);int fun(int ptr[]).
•functions that differ only by passing criteria in which one is passed using a pointer while another is not.

●**<u>Local class</u>** : Class defined inside a function (can be used only locally inside that function).
●**<u>Nested class</u>**: Class inside another class.

● **<u>Friend function :</u>** Functions declared in a class as friend can access all the private & protected members of that class.
Friend function is not a member function of the class in which it is declared.
● **<u>Friend class :</u>** Class declared in a class as friend can access all the private & protected members of that class.
Friend Class is not a member of the class in which it is declared.

● **<u>Inside Class function</u>** : Functions which are either defined inside class or if they are inline functions.
<u>Inline functions</u> : Used for small functions.Eg: inline void classname::solve(){cout<<0;}
● **<u>Outside Class function</u>** : Functions which are defined outside class. Eg: void classname::solve(){cout<<0;}

●**<u>Constructor and Destuctor:</u>**

Constructor	**Destructor**
Automatically called when a new object is constructed.	Automatically called when an object is destructed due to ending of program or due to use of delete keyword.
It is defined in class/struct as member function without any return type.	It is defined in class/struct as member function with ~ sign without any return type & without any parameters.
It must be in public section with same name of that class. Note: It can be made private by using friend function.	It must be in public section with same name of that class. Note: It can be made private by using friend function
There can be many constructor of one class i,e, there can be overloading in constructor(**constructor overloading).**	There is only one destructor of a class i.e. there cant be overloading in destructor.
Types of constructor:- <u>Default constructor</u> :1.Constructor without parameter/arguments. 2. Constructor generated by compiler if there is no user defined constructor i.e. classname(){} <u>Parameterized constructor</u>: Constructor with parameters. <u>Copy constructor</u> : Constructor in which another object is	Types of destructor:- <u>Default destructor</u> : 1.Destructor without parameters 2.Destructor generated by compiler if there is no user defined destructor i.e. ~classname(){} <u>Virtual destructor</u>: Destructor with or without definition which can be redefined in subclass. <u>Pure Virtual destructor</u> : Destructor without definition which

passed as parameter i.e. constructor which uses another object to initialize an object. Virtual Copy constructor : Constructor used for creating objects without knowing the exact data type of object. Note:C++ has no virtual constructor.	can be redefined in subclass. Note: Destructor can't be parameterized,copy or virtual copy because all of them have some parameters & we know that destructors can't have any parameters.

4 Pillars of oops: Inheritance,encapsulation,abstraction,polymorphism

● **INHERITANCE**: Property of a class to inherit things from other class
In such a way, you can reuse, extend or modify the attributes and behaviors which are defined in other classes

Sub/Child/Derived/Extended class: The class which inherits the property of another class
Super/Parent/Base: The class whose features are inherited by subclass.

Advantage of inheritance :To have code reusability
Disadvantage: If data members of parent class are unused in subclass it leads to memory wastage.

Virtual function : Function already declared & defined in base class which can be redefined in subclass.
virtual void solve(){cout<<23;}
Pure virtual function : Function just declared not defined in base class which will be defined in subclass.
virtual void solve()=0;
Note: virtual void solve(){} is virtual function not a pure virtual because it is not =0.

● Access Specifiers/Visibility mode: There are 3 types of access specifiers:
Public: Class members declared as public can be accessed from anywhere outside class.
Protected(generally used in inheritance) : 1)Class members declared as protected can be accessed through sub/derived class.
2)Class Members declared as protected can be accessed through members of the same class.
Private: Members declared as private can be accessed only through members of the same class.

Type of Inheritance on basis of visibility modes(access specifiers)of Parent and Sub classes-:

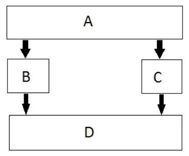
Visibility mode of Super/parent Class	Public(visibility mode of subclass)	Protected(visibility mode of subclass)	Private(visibility mode of subclass)
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Private	Private	Private

Note: Private is not directly accessible.

Types of Inheritance :
1. Single inheritance : When one class inherits another class .eg:Super- Sub.
2. Multi Level inheritance :When inheritance is done through multiple levels/ more than 2 classes indirectly.eg:Super-x-Sub
3. Hierarchical inheritance :When more than 1 sub class directly inherits super class. Sub-Super-Sub
4. Multiple inheritance : When subclass inherits more than 1 super class directly. Super-Sub-Super
eg: maruti and tesla partnered to make mates car which inherits property of both tesla and maruti.
Note:Java doesn't support multiple inheritance because of **Diamond Problem:**The problem with multiple inheritance is that

two or more Superclasses may have different methods and the sub class copies both , and then the subclass can't choose which one to pick to compile and gives a compile time error in java.

5. Hybrid/Virtual inheritance : A combination of hierarchical and multilevel inheritance & multiple inheritance.



Object slicing: When we assign an object of derived class equal to base class object, then derived class objects are sliced off.

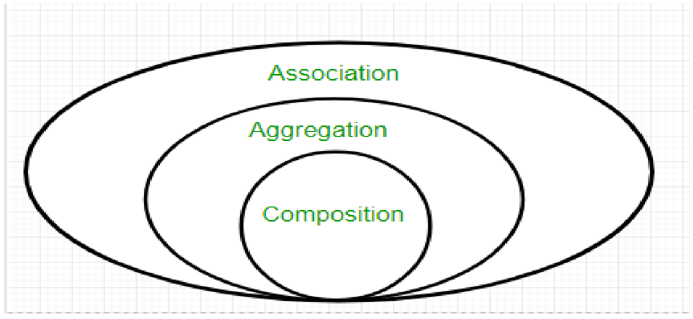
```
Eg: int main(){
        derivedclass d;
        baseclass b = d;  //everything stored in base class is sliced off.

   }
```

Virtual class : Base class defined as virtual while inheritance using virtual keyword. Eg: class derived: virtual public base{}
Used for avoiding diamond problem in inheritance.

Abstract class : A class specially designed to be used as base. Must have at least 1 pure virtual function.

Simulating final class : Class which cant be inherited.Made using friend keyword and other things.

Association is a 2 directional relationship between 2 things .
Aggregation is 1 directional relationship in which 1 object contains another object.Also called has-a relationship.
Composition is an aggregation in which object which is contained in another object can't exist without it.



● POLYMORPHISM :

Data binding: 1.Static(function is resolved early during compile time) 2.Dynamic(function is resolved late during run time)

Types of Polymorphism

CompileTime Polymorphism(Static/Early binding)	**Runtime Polymorphism(Dynamic/Late binding)**
Function/Method overloading and operator overloading causes compile-time polymorphism. Functions can be in same class or different class ,i,e **Inheritance can be used but not compulsory for overloading**.	**Function/Method overriding**(using virtual functions and pointers) causes runtime polymorphism . Here functions have same name,same number and type of parameters(signature) but have different definitions and are in different classes i,e, **Inheritance is compulsory for overriding**.

This type of functions are called **overloaded functions**.	Here child class definition overrides parent class definition. (i,e,Definition according to child class is executed) This type of functions are called **overridden functions**.
Compile Time polymorphism provides fast execution because functions execute at compile time,i.e call resolved by compiler or datatype(class) of object is determined at compiling time.	Runtime polymorphism provides slow execution because functions execute at run time,i,e call is not resolved by compiler or datatype(class) of object is determined at runtime.

Note: Operator overloading is defining operator like +,-,&,| to do some other task irrespective of their real task when called for a particular class.
Some Operators which cannot be overloaded : ?:(conditional operator), sizeof, "." Member access or dot operator, "::" Scope resolution operator,etc..

● ENCAPSULATION :

Encapsulation is the process of combining data and functions into a single unit called class. In Encapsulation, the data is not accessed directly; it is accessed through the functions present inside the class. In simpler words, attributes of the class are kept private and public getter and setter methods are provided to manipulate these attributes. Thus, encapsulation makes the concept of data hiding possible.
Data hiding: a language feature to restrict access to members of an object, reducing the negative effect due to dependencies. e.g. "protected", "private" feature in C++.

● ABSTRACTION :In abstraction we abstract relevant things and reduce unnecessary details.

Difference between struct & class:
1) By default struct is public and class is private.
2) By default a derived struct is public while a derived class is private. Eg: struct derivedd : (public by default) parentt
Else everything is same in class and struct like:constructor,overloading,overriding,everything ….