Page No.
Date

(*) Access Modifiers :

They are used to implement an important aspect of object oriented programming known as data hiding.

(•) Public :
- (*) accessible every where

(•) Private :
- (*) accessible only by members or friend of the class

(•) Protected :
- (*) used with Inheritance

Note : we can't access the private data members by using the object with the dot operator as it will give compile error.

We need to use the getter and setter methods and different friend member of the class which were declared in public.

(*) Include guards in C++.

While programming in C++, we often use a class multiple times and hence it requires to create a header file and just include it in main program. Now, Sometimes it happens that a certain header file directly or indirectly get included multiple times, then the class declared in the header file gets re-declared which gives an error.

To overcome this error we use include guard in our header file of class.

- Include guard ensures that compiler will process this file once, no matter how many times it is included.

Include guard are just lines of preprocessor directives that guarantees file will only include once.

- #ifndef : if not defined, determines if provided macros doesn't exist
- #define : Define the macros
- #endif : closes off #ifndef directive.

Page No.
Date

Syntax. =>

```
#ifndef _ANIMAL_ (Any name can be given but it should be unique)
#define _ANIMAL_ (same word as used earlier)
    class Animal {
        // code
    };
#endif
```

Note. #include <filename> // Standard library header

#include "filename" // user defined header

Q. learn about the #pragma Directive ?

Q. learn about the constructor list ?

Page No.
Date

(#) Encapsulation :-

(•) Wrapping up of data and information in a single unit.

(•) Binding together the data and the function that manipulate them.

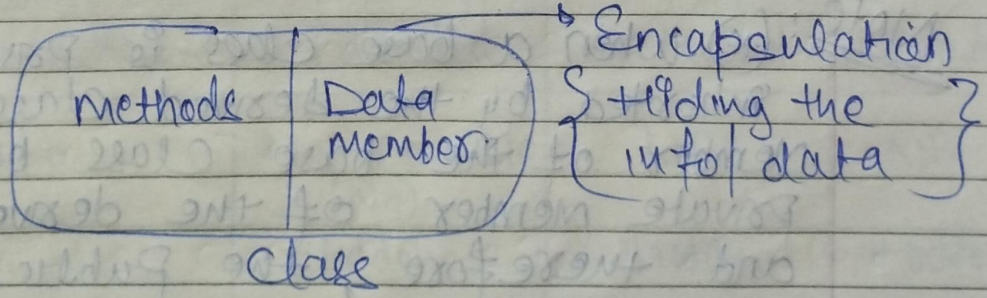
(*) fully Encapsulated class :-

All the data members are declared in private

(*) Two important features :-

(1) Data member and variables are declared private so to provide security.

(2) Member function should be declared public so that anyone can't change and work according to that function.



→ Static binding ?
→ dynamic binding ?

Page No.
Date

Inheritance :-

The capability of a class to derive properties and characteristic from another class.

The new created class is called derived class, sub class, child class.

The class from which it is derived is called base class, super class, Parent class.

Syntax :

```
Class <derived_class_name> : <access_specifier>
                                <base_class_name>
{
    // body
}
```

Eg:- Class Dog : Public Animal {

Note : (*) When a base class is privately inherited by the derived class, public member of the base class become private member of the derived class and therefore, the public member

Note : Composition vs Inheritance ?

Page No.
Date

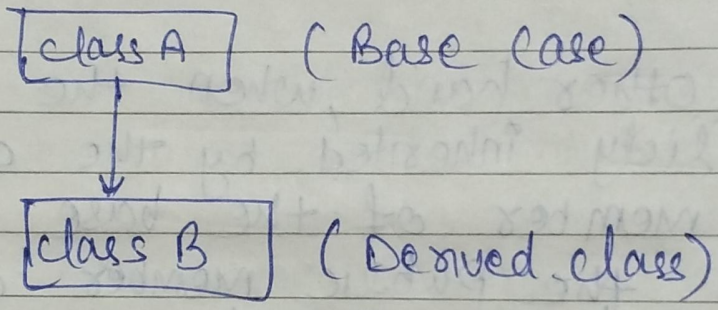
of the base class can only be accessed by the member function of derived class. They are inaccessible to the object of the derived class.

(*) On the other hand, when the base class is publicly inherited by the derived class public member of the base class also becomes the public member of the derived class, therefore, the public member of the base class are accessible by the objects of the derived class as well as by the member function of the derived class.

Base class member access specifier	Type of inheritance		
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not accessible (NA) Hidden	NA Hidden	NA Hidden

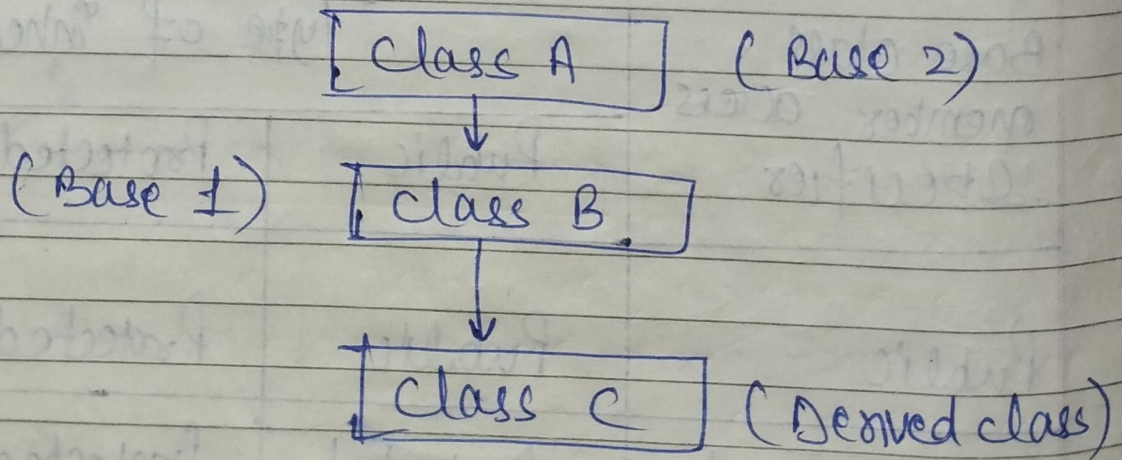
(*) Single Inheritance :-

A class is allowed to inherit from only one class



(*) Multilevel Inheritance :-

A derived class is created from another derived class.

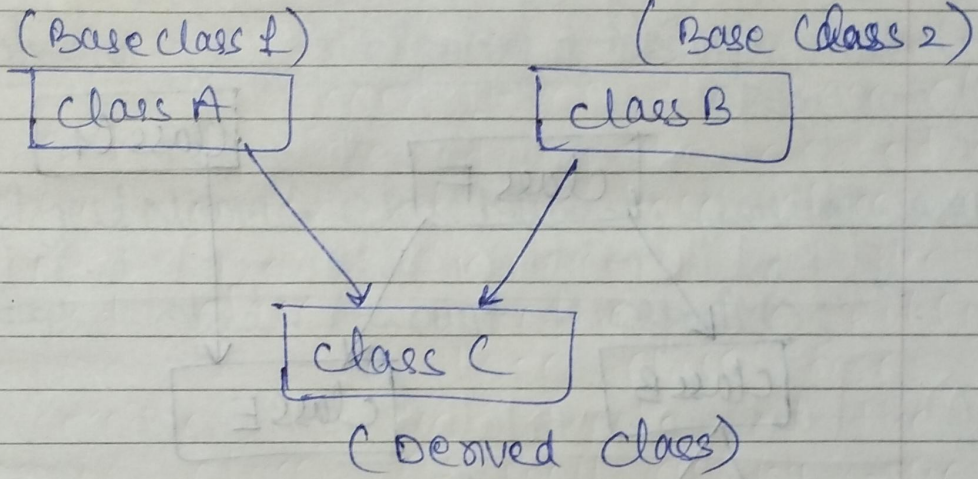


Note : Arrow should be toward the base class as it shows the Hierarchy.

Page No.
Date

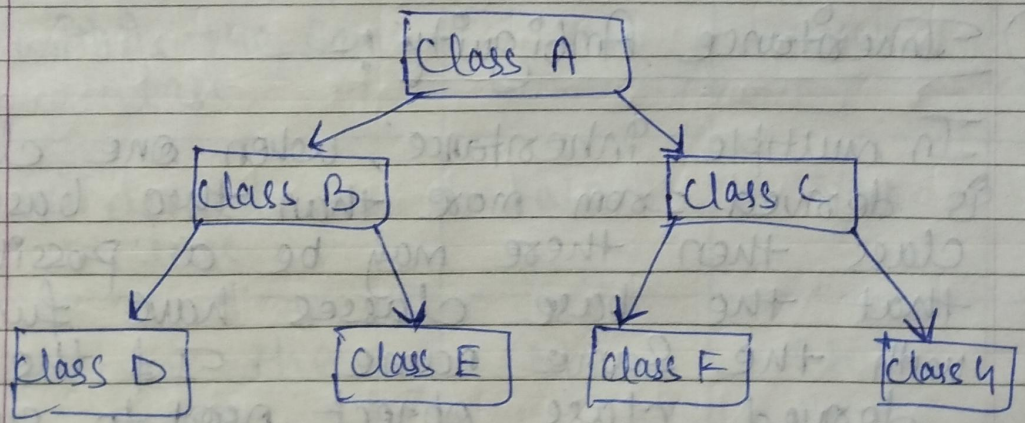
(*) Multiple Inheritance :-

where a class can inherit from more than one class



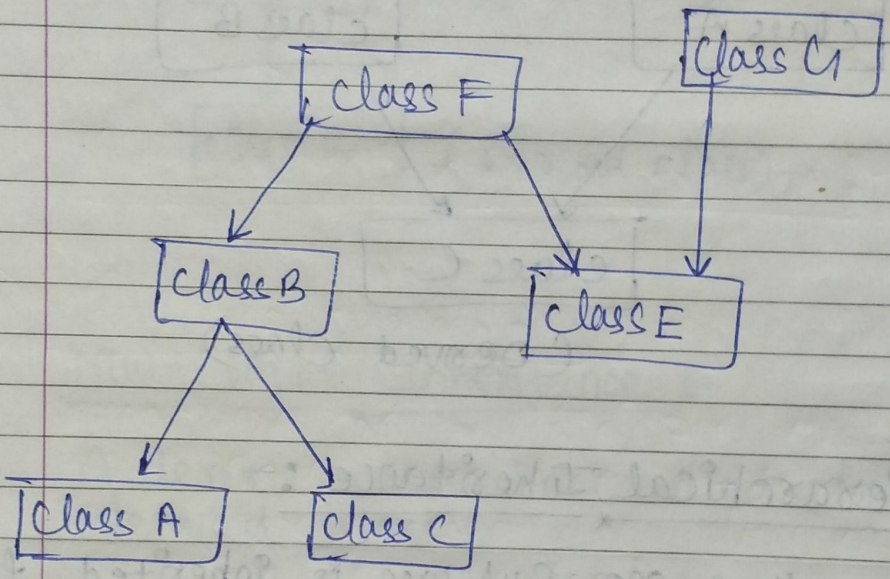
(*) Hierarchical Inheritance :-

More than one Subclass is inherited from a single base class



(*) Hybrid (Virtual) Inheritance :-

Implemented by combining more than one type of inheritance.



(#) Inheritance Ambiguity :-

In multiple inheritance, when one class is derived from more than two base class then there may be a possibility that the base classes have fun" with the same name. If the derived class object need to access one of similarly named function of the base classes then it result

In ambiguity because the compiler get confused about which base's class member function should be called.

Solution to Ambiguity → Scope Resolution operator (::)

objectName . className :: functionName();

Eg:- obj1 . Account :: Deposit();

(*) Access modifiers visibility

visibility	Private	Protected	Public
within the same class	yes	yes	yes
In derived class	No	yes	yes
outside the class	No	No	yes.

It is achieved by using the virtual functions.

Page No.
Date

Polymorphism :-

Polymorphism means having many forms.

The ability of a message to be displayed in more than one form.

(1) Compile-Time Polymorphism :- (Static) → early binding

(A) function overloading :

When there are multiple functions with the same name but different parameter, then the function are said to be overloaded

function can be overloaded by just changing the number of arguments or changing the type of argument

Eg:-
```
void func( double x){
=
}

void func( int x, int y){
==
}
```

→ Learn more on Udemy C++.

Page No.
Date

(B) Operator overloading :-

The ability to provide the operator with a special meaning for a data type, this ability is known as operator overloading.

```
Eg:- class B {
     Public:
        int a;
        int b;

     Public:
       int add() {
            return a+b;
          }

       void operator + (B &obj) {

          int value1 = this->a;
          int value2 = obj.a;
          cout<<" output " << value2 - value1;

       }
};
     int main() {
       B obj1, obj2;

       obj1.a = 4;
       obj2.a = 7;
       obj1 + obj2;
     }
```

// 7 - 4
// = 3.

→ Better than static Polymorphism.
→ late Binding.

Page No.
Date

(2) Run-Time Polymorphism // (dynamic)

(A) Method (function) overridding:-

It occurs when a derived class has a definition for one of the member function of the base class.

Eg:-
```
class Parent {
    Public:
        void GeeksforGeeks(){
            // Statements 1
        }
};

class child : public Parent {
    Public:
        void GeeksforGeeks(){
            // Statements 2
        }
};
```

Note: (8) Pure virtual function.

Page No.
Date

```
int main() {
    Child obj1;
    obj1.GeeksforGeeks();
    return 0;
}
```

Output: Statements 2

because we are calling the GeeksforGeeks of the child class by using its object.

If it doesn't have the member function, then it will look in the Parent class for GeeksforGeeks.

// for better understanding of the dynamic Polymorphism refers to the udemy c++ Polymorphism section

Q Virtual function? Q Base class reference?

Q Override specifier?

Q final specifier?

Abstract class ?

Page No.
Date

Abstraction :-

→ It means displaying only essential information and hiding the details.

Hiding the implementation details.

(*) Types :-

(1) Data Abstraction
(2) Control Abstraction.

Advantages :

(•) Only you can make changes to your data or function

(•) It makes Application secure

(•) Increases the reusability of code

(•) Avoid duplication of the code.

(*) Constructor list :- // Initialization list.

(•) Previous way:

```
Player (std::string name_val){
    name = name_val;
    health = 0;
    age = 0;
} // assignment not Initialization
```

(•) Better way :

```
Player (std::string name_val)
    : name{name_val}, health{0}, age{0} {
}
// this way we are directly initialising
and their is no assignment.
// Also, the constructor body is empty and
we can write code over their.
// this is more efficient and usable.
```

→ only works in Initialization list.

Page No.
Date

(*) Constructor Delegation in C++:

Sometimes it is useful for a constructor to be able to call another constructor of the same class

Eg:-

```
                    // constructor list
Player :: player (Std :: String name_val , int hea
                  int xp_val)

    : name {name_val} , health {health_val},
{
        // body

}

player :: player ()
    : player { "None", 0, 0}
{
    // body.
}

player :: player (Std :: String name_val)
    : player { name_val, 0, 0}
{
        // body.
}
```

constructor calling another constructor to reduce the again & again initialization list syntax.

Page No.
Date

(*) Default constructor parameter :-

// constructor with default parameter value

Eg:-

```
Player (std::string name_value = "NONE",
        int health_val = 0,
        int xp_val = 0);
```

th_val,

Q Move constructor?

xp{xp_val} Q Copy elision? RVO (Return value optimization)

Q L-value & R-value reference parameter?

Q using const with classes and object?

Q friend of a class? // friend function & class in C++;