INHERITANCE

(DERIVATIONS)

* INHERITANCE

Introduction:

- Classes can be adapted by other programmers to suit their requirements.
- This is done by creating new classes, reusing the properties of the existing ones.
- The mechanism of deriving a new class from the old class is called **Inheritance (or) Derivation.**

INHERITANCE

The old class is referred to as the **Base class** (Super class)

&

The new class is called the **Derived class** (Sub Class)

*NEED FOR INHERITANCE

* Represents real world relationships well
* Provides reusability of the code.
* Supports transitive nature

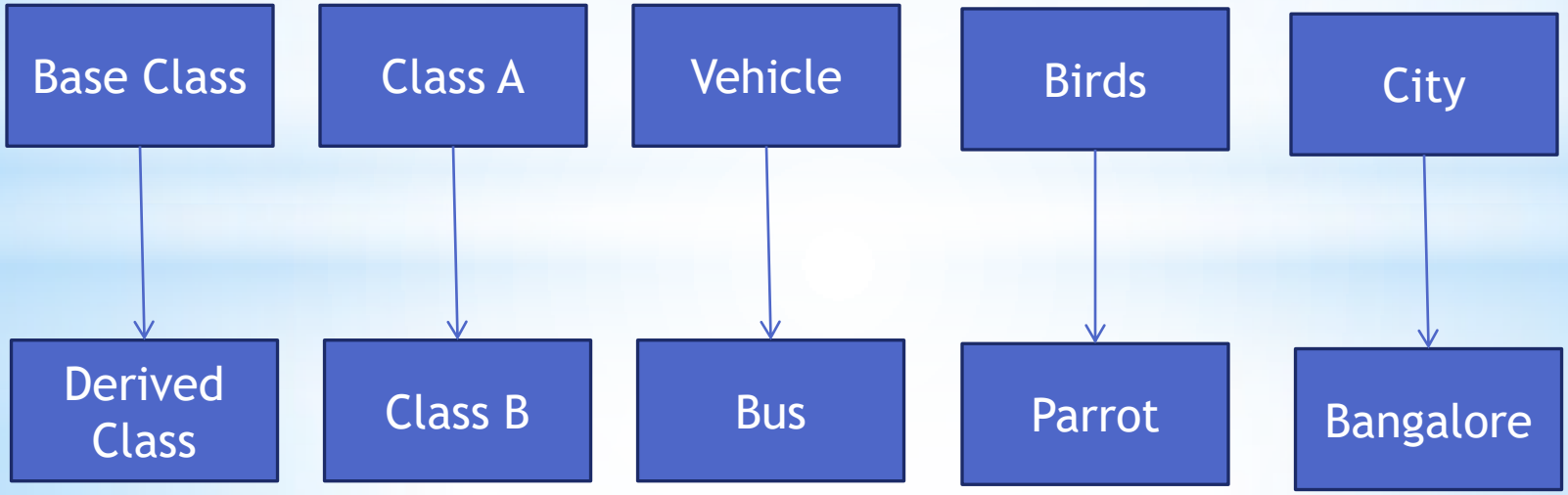
* Types of Inheritance

1. Single Inheritance
2. Multiple Inheritance
3. Hierarchical Inheritance
4. Multi-Level Inheritance
5. Hybrid Inheritance

* SINGLE INHERITANCE

A derived class with only one base class is called “Single Inheritance”

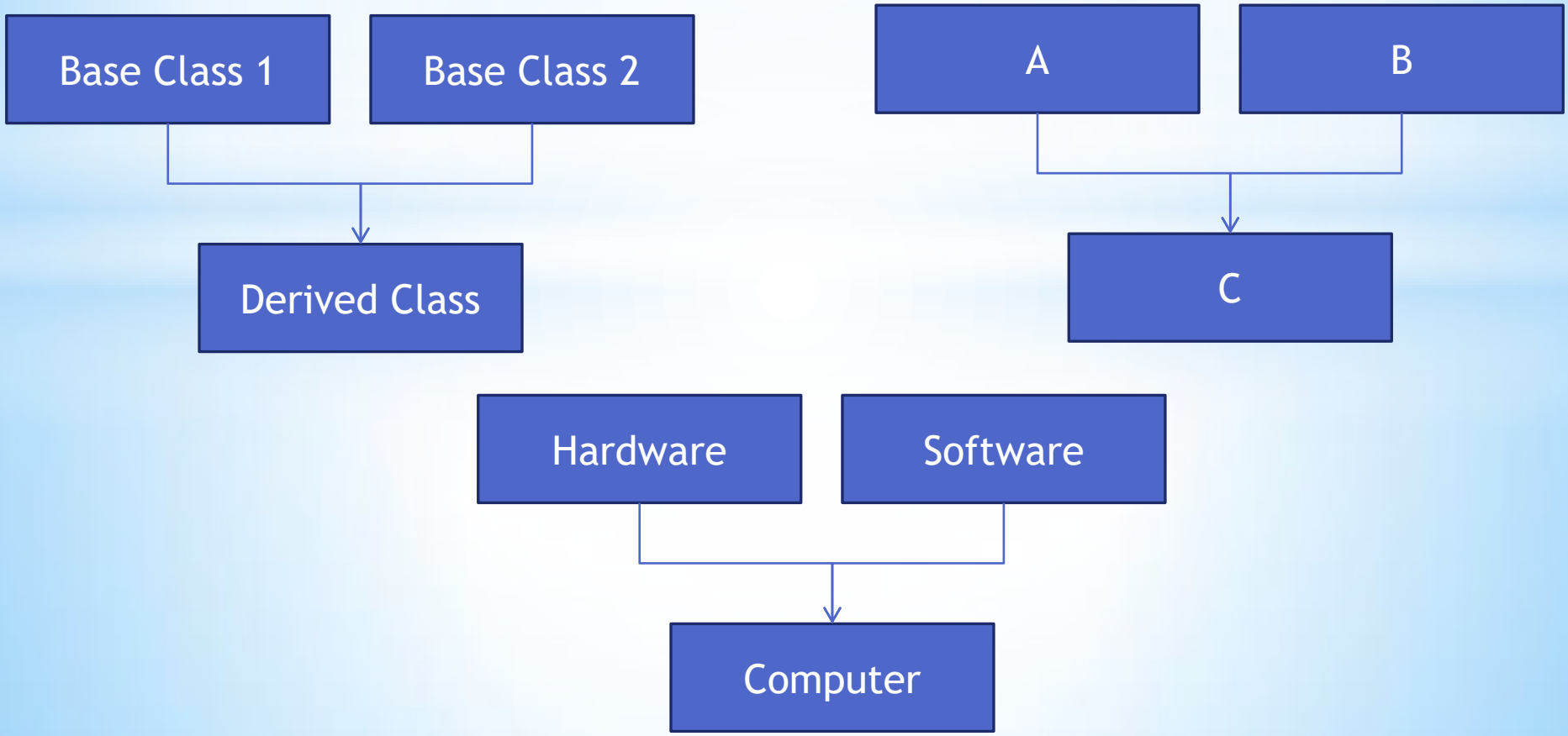
Eg.



Multiple Inheritance

A derived class with several base class is called “Multiple Inheritance”

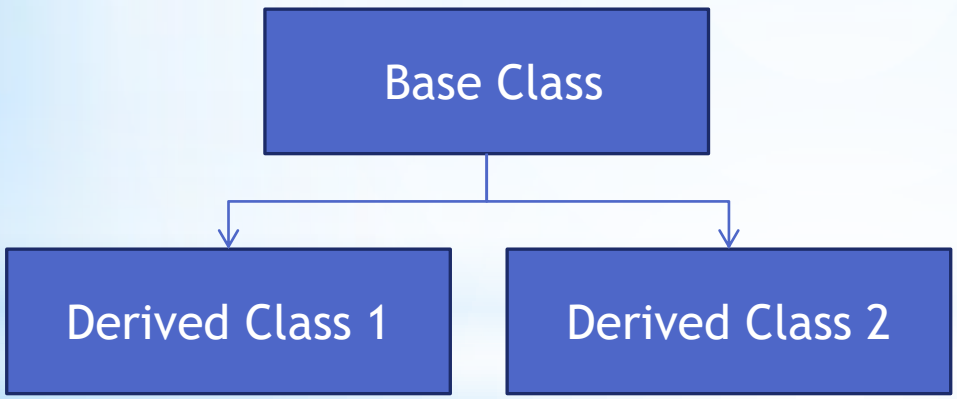
Eg.

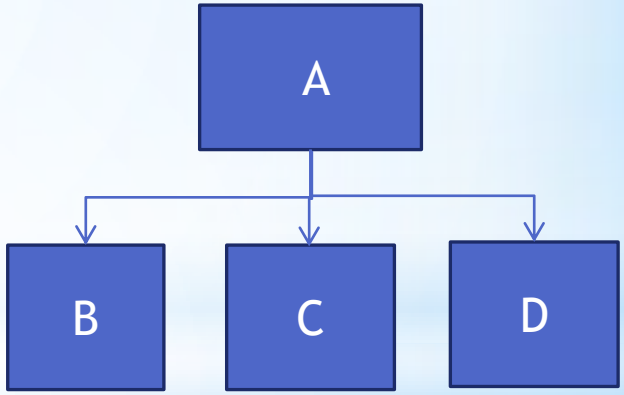


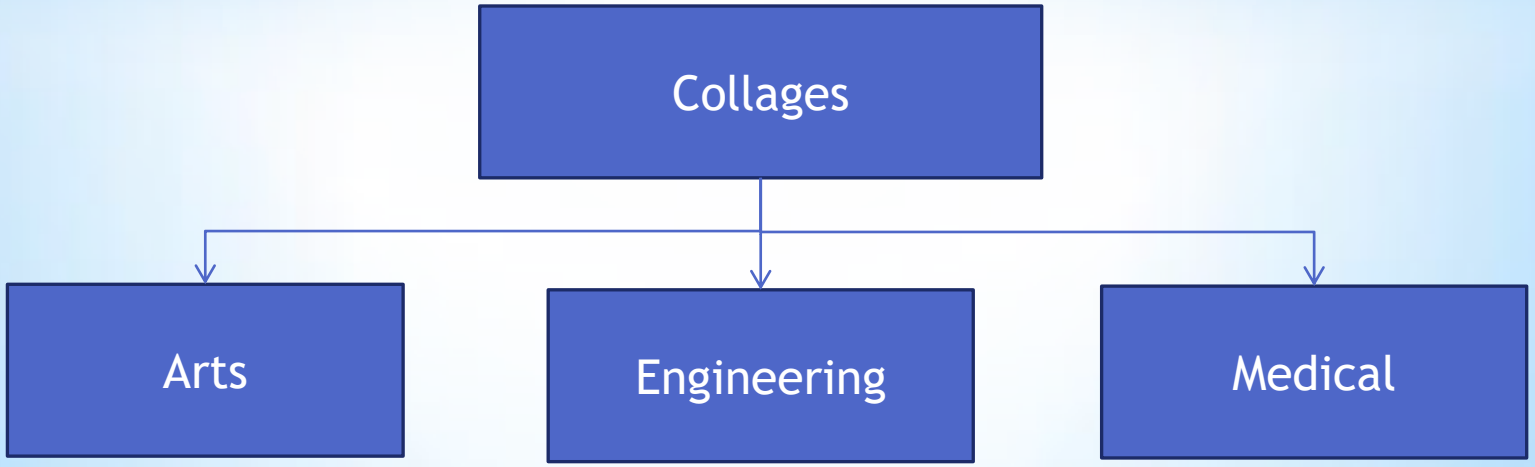
Hierarchical inheritance

A Base class may be inherited by more than one class is known as “Hierarchical Inheritance.”

Eg.



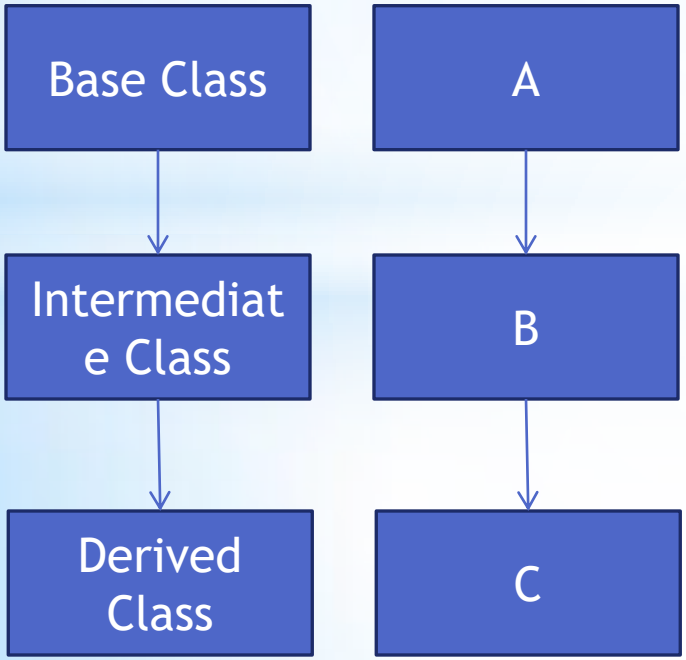




Multi-level inheritance

Deriving a class from another derived class is known as "Multilevel Inheritance"

Eg.



A is Base class of B
B is Derived class of A which is the Base class of C
C is Derived Class of B

The class A serves as base class for the derived class B, Which in turn serves as base class for the derived class C

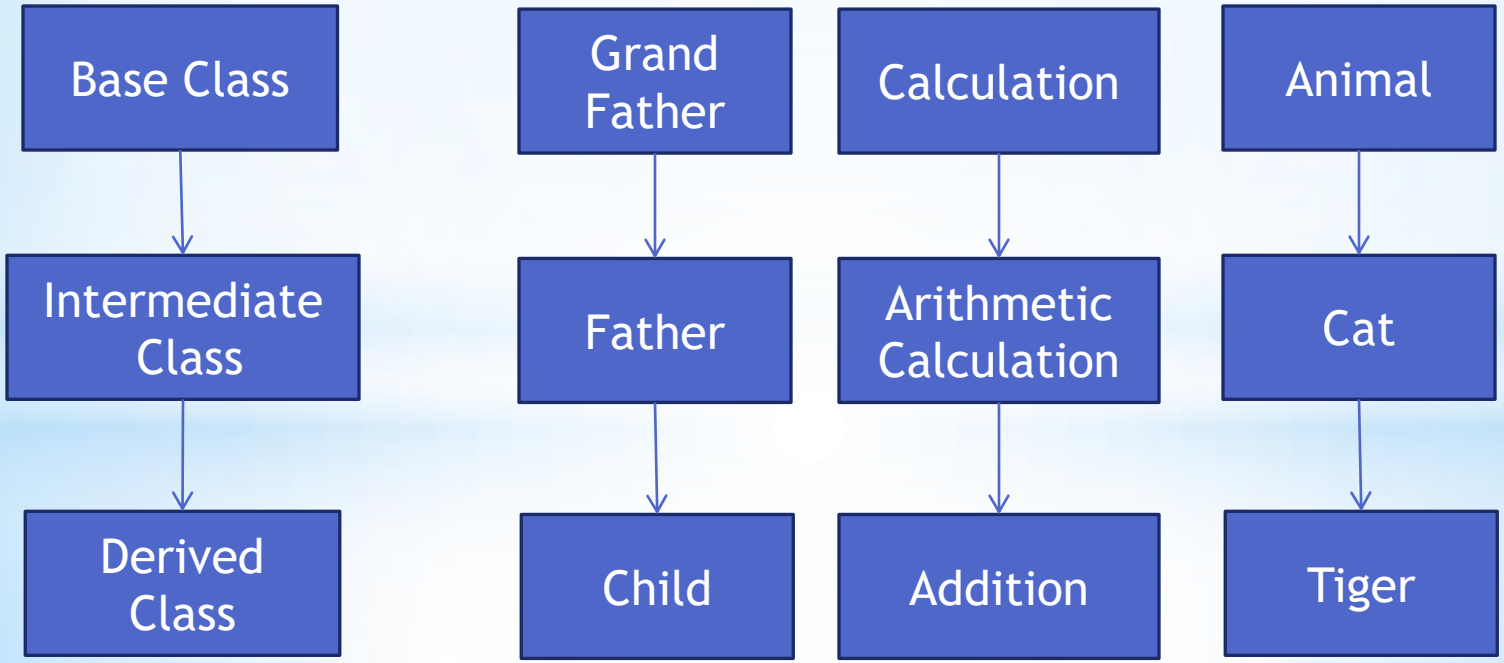
The class B is known as intermediate base class since it provides a link for the inheritance between A and C.

The Chain ABC is known as inheritance path.

This process can be extended to any number of levels.

* Multi-level inheritance

Some more examples.

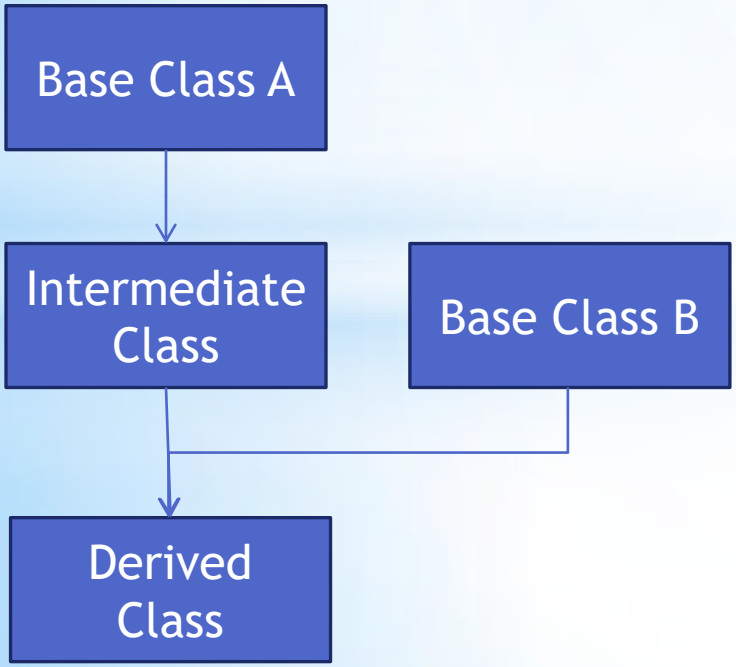


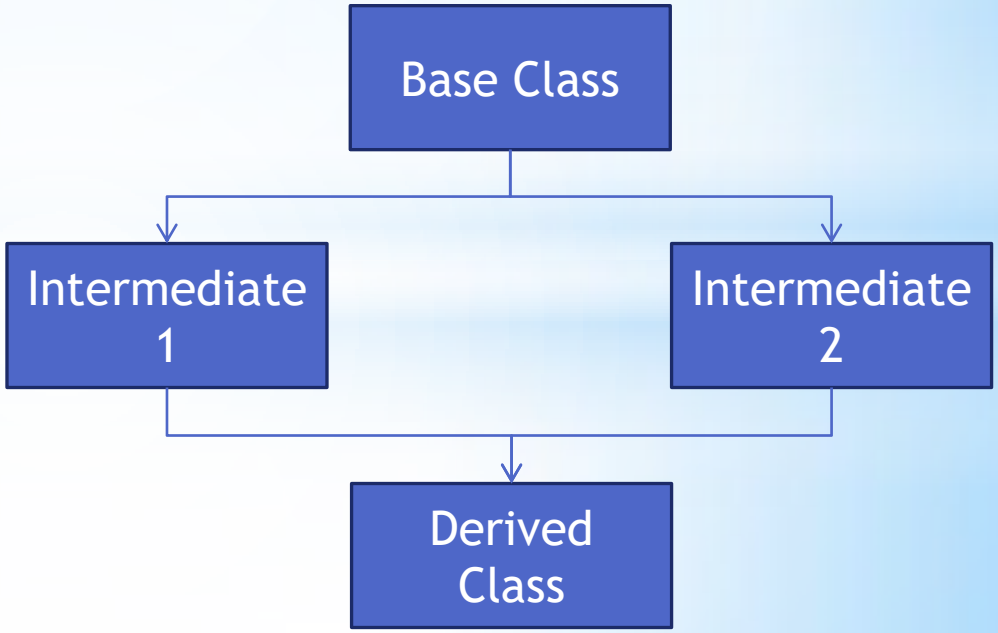
Note : The direction of arrow indicates the direction of Inheritance

* Hybrid inheritance

Deriving a class from more than one derived class is known as “Hybrid Inheritance”.

Eg.





Note : The direction of arrow indicates the direction of Inheritance

* How To Define Derived Classes?

A derived class can be defined by specifying relationship with the base class in addition to its own details.

Syntax:

```
class   DerivedClassName  :  VisiblityMode    BaseClassName
{
……….
……….
……….
};
```

Note:

The Colon indicates that the DerivedClassName is derived from the BaseClassName

The visiblityMode is optional and it may be either **"public"** or **"Private"** or "**protected**"

The default visibility mode is **private**.

A class having 3 visibility modes

Example:

```
class sample
{
private :                    // the keyword "private" is optional / default mode
        ..............       // it is visible to member functions within its class.
        ..............       // it cannot be inherited
        ..............

protected :
        ..............       // it is inheritable
        ..............       // it is visible to the member functions of its
        ..............       //                          own & derived classes

public :
        ..............       // it is inheritable
        ..............       // it is visible to all functions in the program.
        ..............
} ;
```

Note : The default visibility mode is **private**.

Three modes of inheritance

Example:

```
class XYZ : public  ABC
{
   Members of XYZ
}
```

Or

```
class XYZ : protected  ABC
{
   Members of XYZ
}
```

Or

```
class XYZ : private  ABC
{
   Members of XYZ
}
```

Or

```
class XYZ :  ABC
{
   Members of XYZ
}
```

Note :

- The default visibility mode is **private**.
- While **applying inheritance** we usually create objects using the **derived class.**

Note :

* **Private members** are cannot be inherited and they are not accessible to the objects.

* **protected members** are inheritable but they are not accessible to the objects.

* **public members** are inheritable and they are accessible to the objects.

When a base class is publicly inherited by a derived class

* “private members” are cannot be inherited. So the private members of a base class will never become the members of its derived class.
* “public members” of the base class become “public members” of the derived class and therefore they are accessible to the objects of the derived class.
* “protected members” of the base class become “protected members” in the derived class too. And therefore it is accessible by the member functions of the derived class. (It is available for further inheritance.) but
* “Protected members” are not accessible to the objects of the derived class.

Note :

*In public derivation,

the public members of the base class become public member of derived class and the protected members of base class become protected members of the derived class.

When a base class is privately inherited by a derived class

* “private members” are cannot be inherited. So the private members of a base class will never become the members of its derived class.
* “public members” of the base class become “private members” of the derived class and therefore the public members of the base class can only be accessed by the member function of the derived class.
* A public members of derived class can be accessed by its own objects using dot operator but No members of the base class is accessible to the object of the derived class.
* “protected members” of the base class become “private member” of the derived class. (It is not available for further inheritance)
* The default visibility mode is “private”

Note :

*In private derivation,

both the public and protected members of the base class become

private members of the derived class.

When a base class is protectively inherited by a derived class

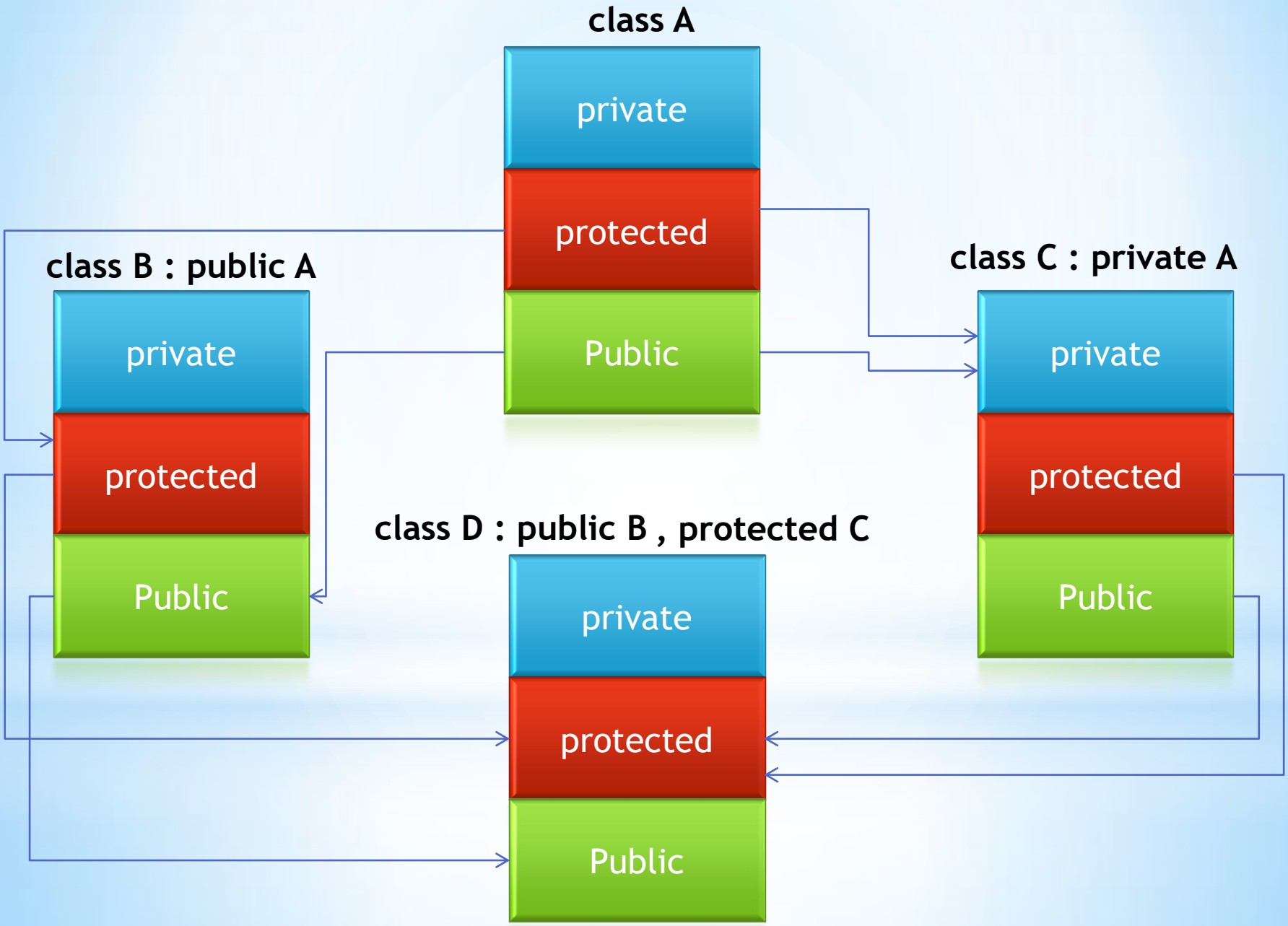
* “private members” are cannot be inherited. So the private members of a base class will never become the members of its derived class.
* “public members” of the base class become “protected members” of the derived class and therefore the public members of the base class can only be accessed by the member function of the derived class.
* And they are not accessible to the objects of the derived class. (
* “protected members” of the base class become “protected member” of the derived class. (It is available for further inheritance)
* However no members of the base class is accessible to the object of the derived class.

Note :

* **In protected derivation,**

both the public and protected members of the base class become protected members of the derived class.

The following figure shows the two levels of derivation.



Note :

* **In private derivation,**

both the public and protected members of the base class become private members of the derived class.

* **In public derivation,**

the public members of the base class become public member of derived class and the protected members of base class become protected members of the derived class.

* **In protected derivation,**

both the public and protected members of the base class become protected members of the derived class.

Visibility of inherited members :

Base class Visibility	Derived Class Visibility		
	private	**protected**	**public**
private	Not inherited	Not inherited	Not inherited
protected	private	protected	Protected
public	private	protected	public

The various functions that can have access to the private and protected member functions of a class

- **Member function of the class**
- **Member function of the derived class**
- **Friend function of a derived class**

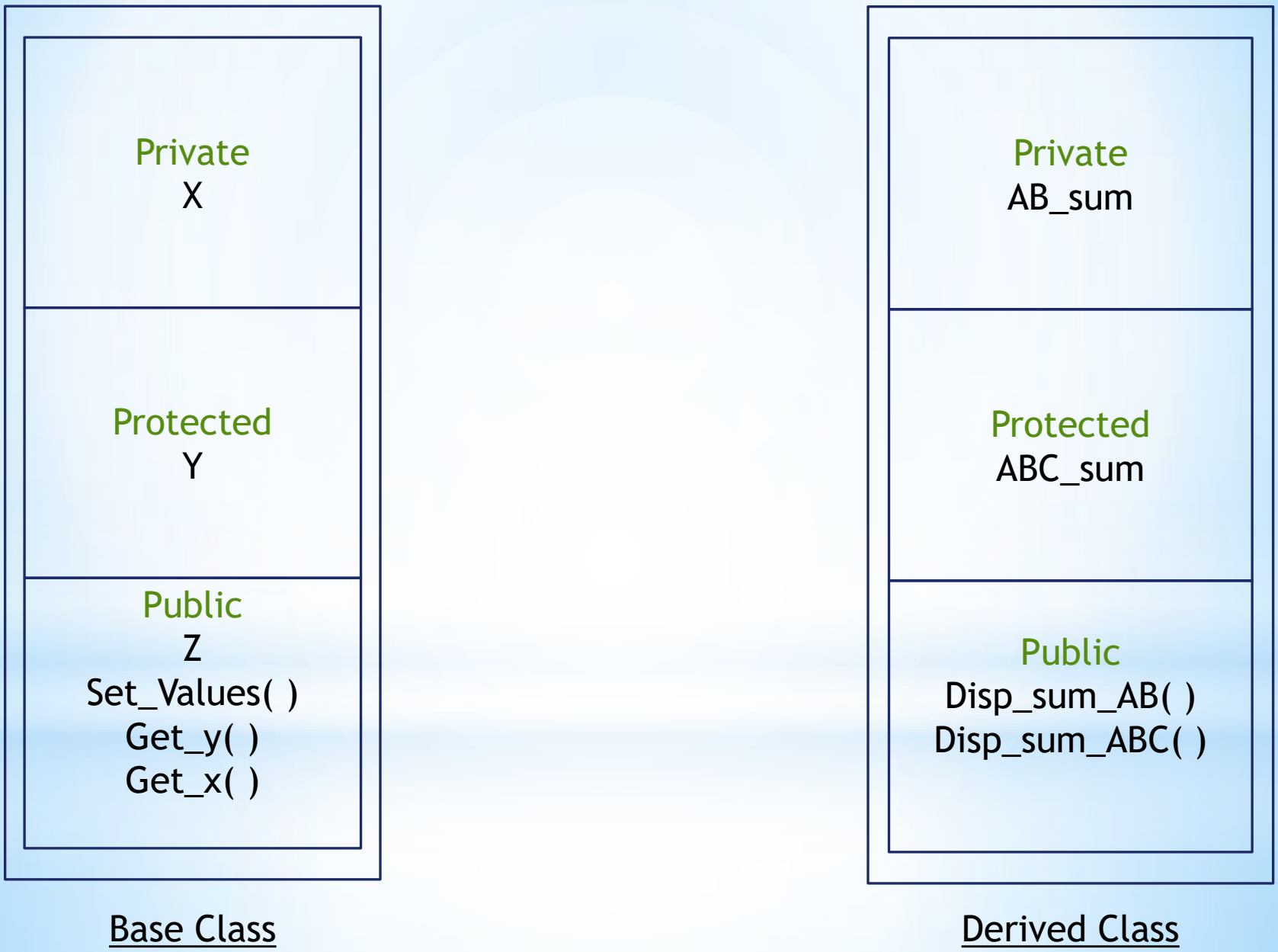
Note1:
Friend function & member functions can have direct access to both the private and protected data.

Note2 :
Member function of derived class can directly access only the protected data. They can access the private data through the member functions of the base class.

Write a C++ program using the above classes, and display the sum of x+y & x+y+x from main program.

Note :

1. define a necessary member functions in both the classes.
2. You should create object only for derived class(Addition).


Number
Private
X
Protected
Y
Public
Z
Set_Values( )
Get_y( )
Get_x( )
Base Class
Addition
Private
AB_sum
Protected
ABC_sum
Public
Disp_sum_AB( )
Disp_sum_ABC( )
Derived Class

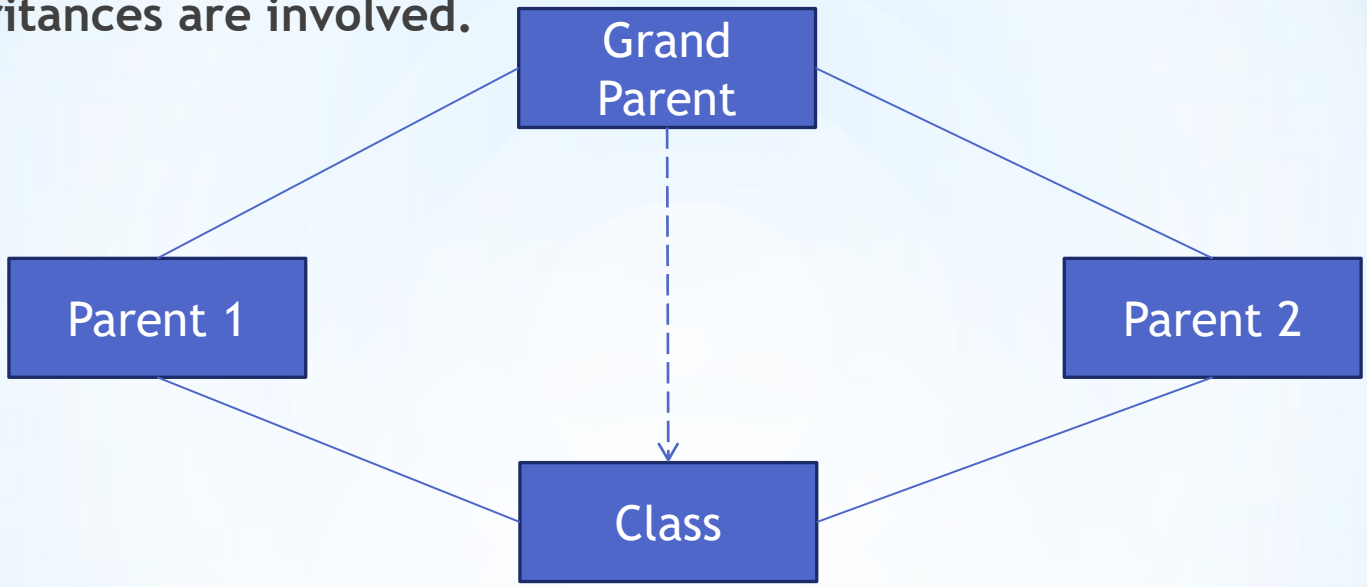
* INHERITANCE

Abstract Class

- An abstract class is not used to create object. It is designed only to act as a base class.
- It is to be inherited by other classes.

* Virtual Base Class

* In the following diagram the multi-level, multiple & hierarchical inheritances are involved.



the child has two direct base classes Parent-1 & Parent 2 which themselves have a common base class grand parent.

the child inherits the traits of “grand Parent” is also inherits directly as shown by dotted line. The “grandparent” is also referred to as indirect base class.

Inheritance of the child class as shown in fig might poses some problems. All the public & protected members of “Grand parent” are inherited into “child” class twice. This introduces ambiguity and should be avoided.

The duplication of inherited members due to these multiple paths can be avoided by making the common base class or ancestor class as **virtual base class**.

* Virtual Base Class

When a class is made a virtual base class, then only one copy of that class is inherited.

CONSTRUCTOR IN INHERITANCE

- as long as no base class constructor takes any arguments, the derived class need not have a constructor function.
- If any base class contains a constructor with more than one arguments, then it is mandatory for the derived class have a constructor and pass the arguments to the base class constructor.
- While applying inheritance we usually create objects using the derived class.
- The derived class to pass arguments to the base constructor.
- When the derived & base class have constructor, the base class constructor is executed first and then the derived class constructor executed.
- the derived class takes the responsibility of supplying initial values to its base classes, we supply the initial values that are required by all the classes together, when a derived class object is declared.
- The constructor of the derived class receives the entire list of values as its arguments and passes then on to the base class constructors in the order in which they are declared in the derived class.
- The base class constructors are called and executed before the statements in the body of the derived constructor.

Syntax of derived constructor:

Multiple Inheritance :

```
derived (arg_list1, arg_list2,....., arg_listN, arg_listD) : base1(arg_list1), base2(arg_list2) ,...., baseN(arg_listN)
{
Body of the derived constructor
}
```

```
derived_constructor  ( arg_list1, arg_list2,.....arg_listN, arg_listD ):

base1(arg_list1),

base2(arg_list2),
.....
.....
baseN(arg_listN),

{
Body of the derived constructor
}
```

The header line of the derived_cinstructor function contains two parts seperated by a colon (:)

The **first part** provides the declaration of the arguments that are passed to the derived constructor and the **second part** list the function calls to the base constructor

base1 (arglist 1) , base2 (arglist 2)… are function calls to base constructor base1(), base2() … and arglist1 , arglist2 … represent the actual parameters that are passed to the base constructor.

arglist 1…….arglist N are the argument declaration for base constructors base 1…….base N.

arglist D provides the parameters that are necessary to initialize the members of the derived class.

Example :

```
D(int a1, int a2, float b1, float b2, int d1): A(a1,a2),  B(b1,b2)
{
 d=d1;
}
```

A(a1, a2) calls the base constructor A( ), B(b1,b2) calls another constructor B( ).

The constructor D supplies the values for the 4 arguments. Additionally it has its own argument. Totally it was 5 arguments. D( ) may be called as followes

D Obj_d (5, 12, 2.5 ,7.5 ,15)

These values are assigned like

5 → a1 12→ a2 2.5→ b1 7.5→ b2 15→d1

Execution of base class constructors :

Method of inheritance	Order of execution

class B : public A
{
 	
};
 	
	A( ) : base constructor
	B( ) : derived constructor

class C : public A, public B
{
 	
};
 	
	A( ) : base constructor1
	B( ) : base constructor2
	C( ) : derived constructor

class C: public A, virtual public B
{
 	
};
 	
	B( ) : virtual base constructor
	A( ) : Ordinary base constructor
	C( ) : derived constructor

Overriding in Inheritance

What is Overriding ?

When we used same member function name in both base and derived classes, the derived class object overrides the function definition of base class.

So only the derived class function definition will get executed.

In multiple inheritance : when we inherit more then one base class that have the same member function name, Ambiguity error may be arise.

How to access base class function when derived class also having the same member function name?

* Initialization List in Constructor

*C++ supports another method of initializing the class objects. This method uses what is known as initialization list in the constructor.

Syntax

```
Constructor (arg_list) :  initialization-section
{
  assignment section
}
```

The body of the constructor is used to assign initial values to its members.

The initialization –section used to provide initial values to the base constructor and also to derived constructor separated by commas.

Example

```
class ABC
{
int a;
int b;
public :
ABC( int i, int j ) : a(i) , b(2*j)
{
  cout<<endl<<“a= ”<<a;
  cout<<endl<<“b= “<<b;
}
};
void main()
{
  ABC obj(2,3);
}
```

This will initialize a to 2 & b to 6

The constructor may also be written as :

ABC(int i , int j): b(i) , a(i+j)

{ ---------- }

Here a will be initialized as 5 and b as 2

Note :

The data members are initialized in the order of declaration.

* ABC(int i, int j); a(i), b(a+j){ }. Here a is initialized to 2 & b to 6, a has been declared first, it is initialized first and then its value is used to initialize b.

* **The following will not work:**
* ABC(int i, int j); b(i), a(b*j) { }

because the value of b is not available to a which is to be initialized first.

* **The following statements are also valid:**
* ABC(int i, int j); b(i), a(b=j)

* ABC(int i, int j)

```
{
   a=i;
   b=j;
}
```

Containership or containment or Aggregation

* When a class contains objects of other class as its member, it is referred to as containership

* **When a class inherits from another class, it is known as IS-A relationship.**

  When a class contains objects of other class type as its member known HAS-A relationship.