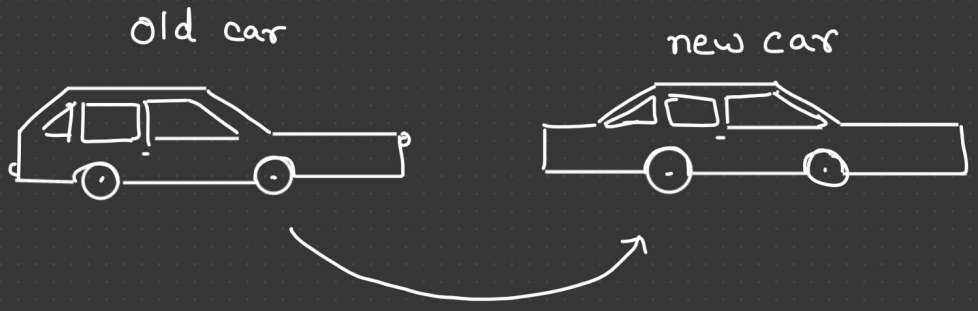
Inheritance

One of the main goal of OOP's is to provide reusable code.

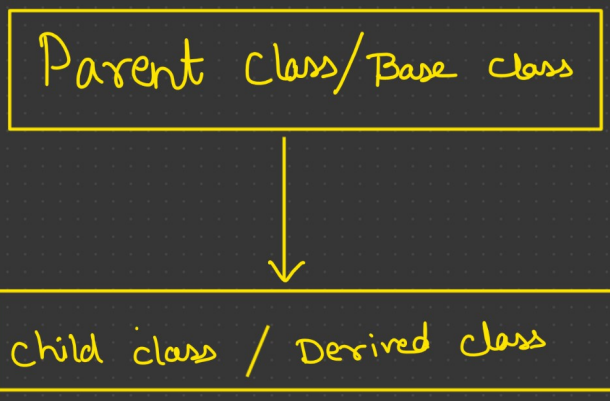


+ engine ↑
+ design ↑
+ new features ↑

+ Old functionality

```
Class Calculator
{

    add()
    {

    }
    sub()
    {

    }

};
```

```
Class NewCalculator
{

    add() ✓
    sub() ✓
    mult() +
    divide() +

};
```



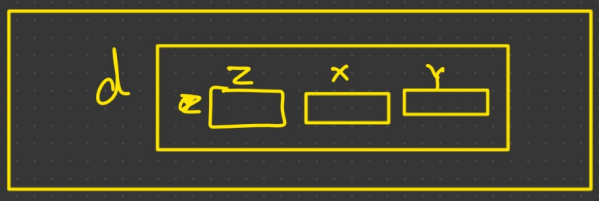

```
Class A
{
  ____
  ____
  ____
  ____
};
```

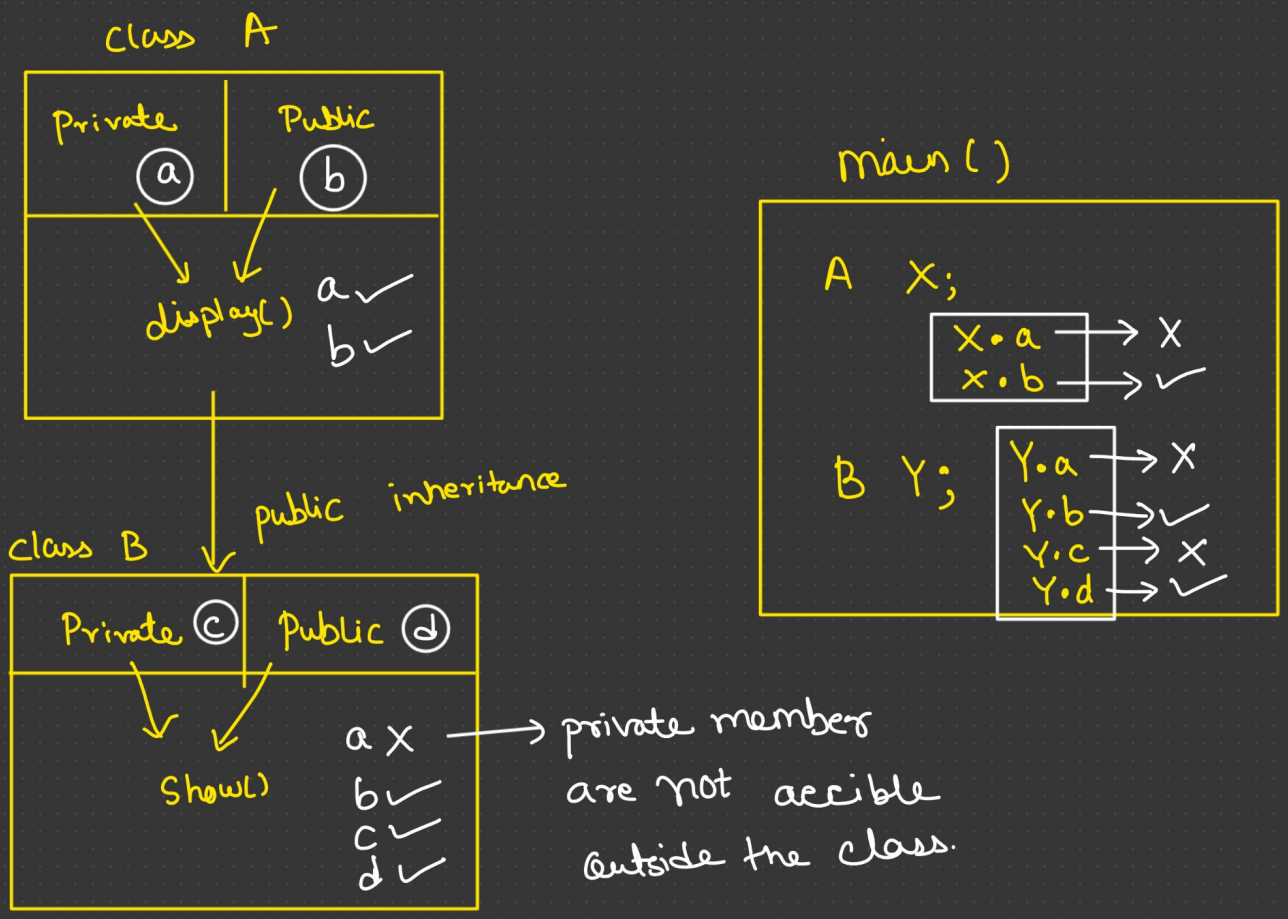
```
Class B : public A
{
  ____
  ____
  ____
  ____
  +
  New
};
```

```
Class ABC
{
  public :
    int x, y;
};
```

```
Class DEF : Public ABC
{
  public :
    int z;
};
```

```
int main()
{
  DEF d;

};
```




Class A
Private
Public
a
b
display()
a ✓
b ✓
public inheritance
Class B
Private c
Public d
Show()
a X
b ✓
c ✓
d ✓
private member are not accible outside the class.
main()
A X;
X.a → X
X.b → ✓
B Y;
Y.a → X
Y.b → ✓
Y.c → X
Y.d → ✓

```
class Parent
{
  public :
    int x, y;

    void display()
    {
      cout << x << y;
    }

};
```

```
class Child : public Parent
{
  pubic :
      int z;

      void display()
      {
        cout << x << y << z;
      }

};
```

```
int main()
{
   Parent a;
   Child  b;

   Parent *p = &a;
   p->display();
```

① ⟶ parent class display() called

```
   Child  *c = &b;
   c -> display();
```

② ⟶ Child class display() called.

```
   parent *p1 = &b;
```

③ → parent pointer ⟶ Child object

```
   Child *c1 = &a;
```

④ → child pointer ⟶ parent object

```
Parent *p1 = &y;  //1. parent class pointer can point to child class object
p1->display();    //2. Function of pointer type will be called

Child *c1 = &x;  // It is an error Child class pointer cannot point to parent class object
 c1->display();  // error
```

Example :-

```
Class Base
{
   int x;
   void fun( Base *p)
   {
      =
    }
};
```

```
int main()        → old code
{
  Base a, b;
  a.fun( &b);
  return 0;
}
```

```
Class Derived1 : public Base
{
   int y;
}
```

```
int main()        → new code
{
   Derived1 c, d;
   c.fun(&d);
   return 0;
}
```

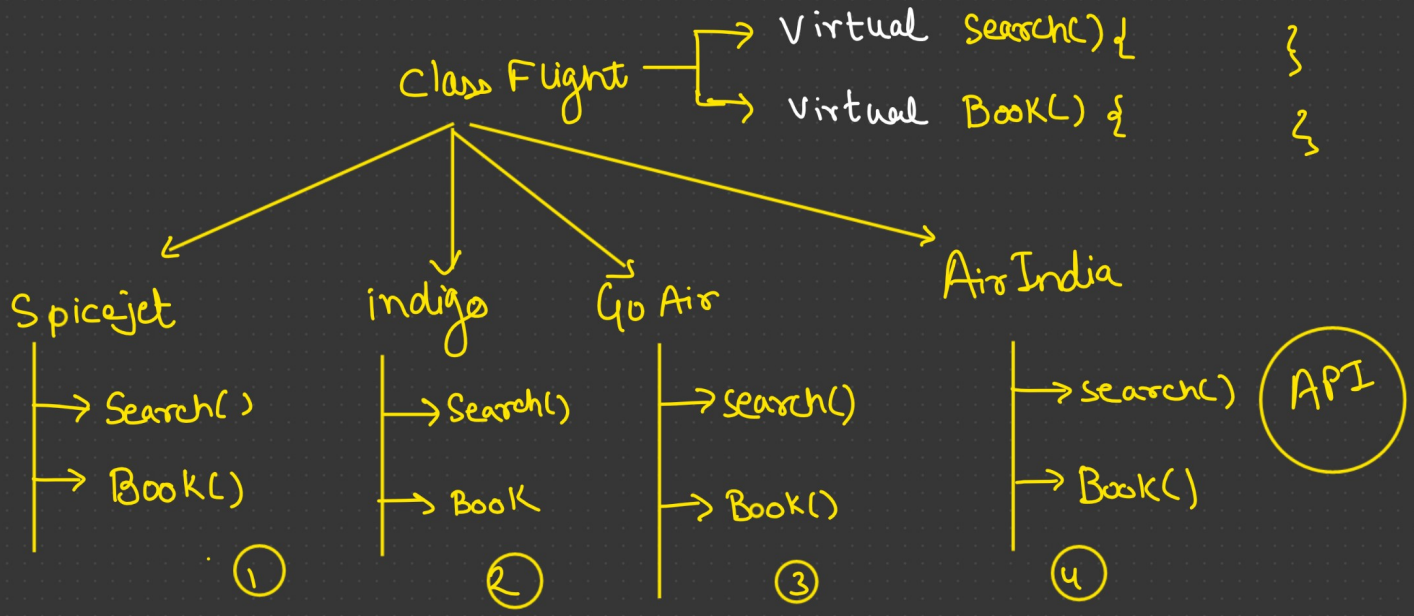
Base *p = &d; ✓

```
class Derived2 : public Base
{
    void fun( Base * p)
    {
        == //modified code
    }
};
```

```
int main()
{
    Derived2 b, c
    Base * a = &b;
    a->fun(&c);

    return 0;
}
```

Base class fn will be called. But I want to call derived2 class function.

Virtual Function



```
int main()
{ int x;
  cout<<"Select flight";
  cin>>x;
  Flight *f = Choice(x);
  f -> Search();
  f -> Book();

  return 0;
}
```

compile

Runtime

```
Flight * Choice(int x)
  {
    if(x==1)
      return new Spicejet();
    else if(x==2)
      return new Indigo();
    else if(x==3)
      return new Goair();
    else if(x==4)
      return new AirIndia();
  }
```

- How the virtual function work?
- What is Static Binding / Compile time Binding / Early Binding?
- What is Dynamic / Runtime / Late Binding?

Static & Dynamic Binding

Compile time
[Before execution of the code]

Runtime
[At the time of execution of code]

Object + function

```
int x;
float y;
Student s;
```

} Compile time → Early Binding → static

```
new Student();
malloc();
calloc();
```

} Runtime → late → Dynamic

Virtual Keyword indicate to the compiler that instead of taking decision at compile time, please take the decision at Runtime.

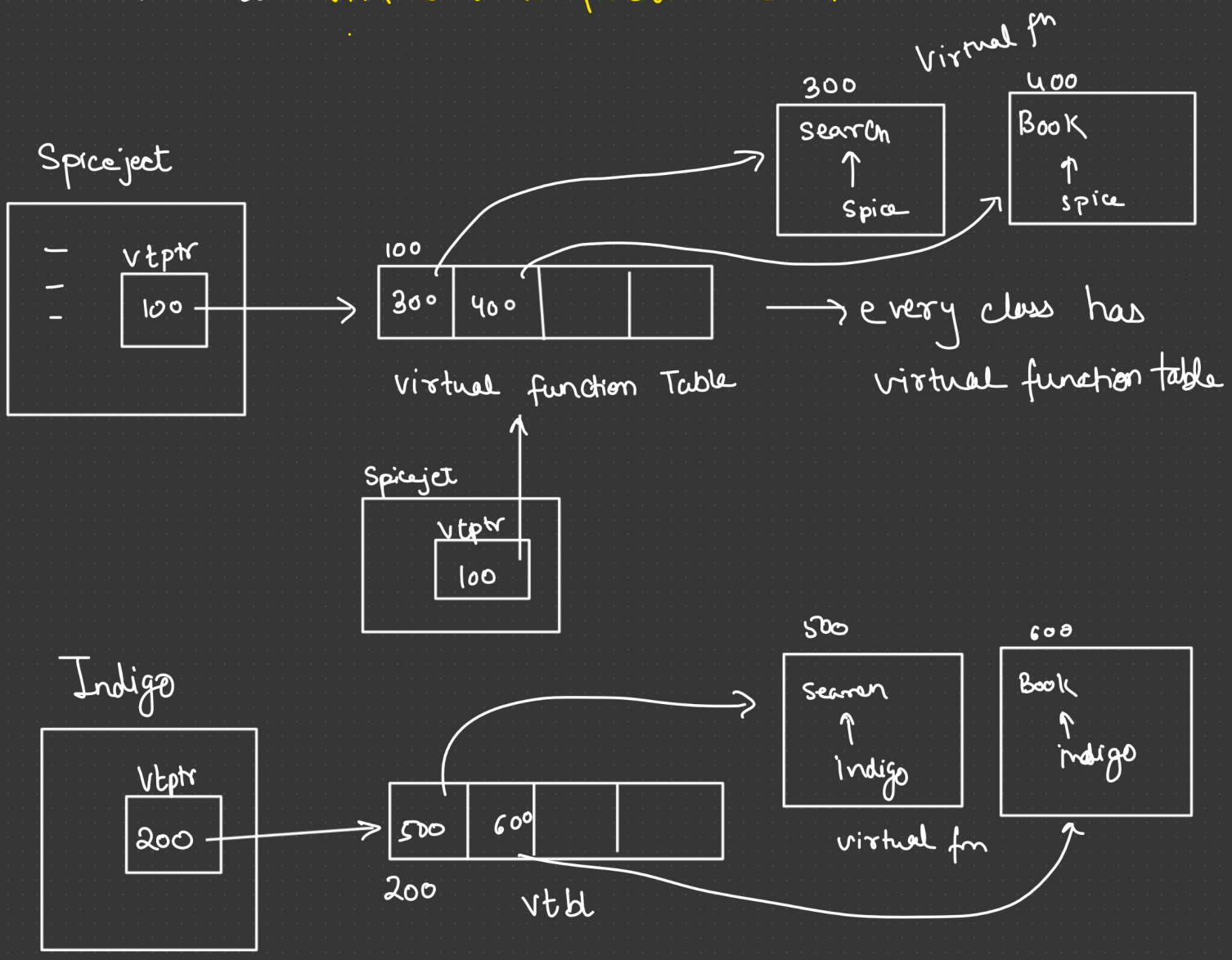
Why two Kinds of Binding ?

- efficiency → By default → Compile time
  Because it is fast.
- Conceptual Model → May be you don't want to redefine the function in derived classes.

How Virtual functions works ?

- Every object has a hidden member that holds the address of an array of functions addresses. Such an array is usually termed as Virtual function Table (Vtbl).

- And the pointer that holds the address of Vtbl is known as virtual table pointer (Vtptr).



Constructor & Destructor

* Constructor can't be virtual — Because constructor does not inherited by derived class. And if we create an object of derived class, it will first call base class constructor. ∴ we don't need to make it virtual.

* Destructor should be virtual —

```
Flight *f = new Spicejet;
delete f;
```

→ ~Flight() or ~Spicejet()

virtual ~Flight();

if flight class destructor is called then child class (spicejet) object will not released.

* But we know that, first child class destructor should be called & there after parent class destructor need to be called.

∴ it should be virtual so that correct destructor is called.

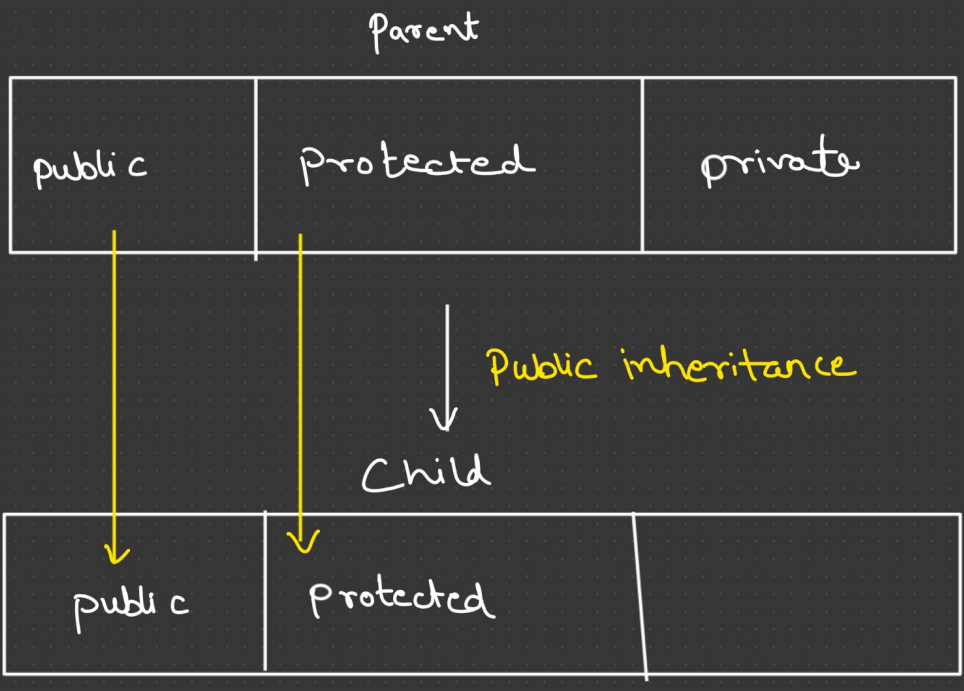
Access Control : Protected

```
class A
{
   private :
            int x;

   protected :
            int y;

   public  :  int z;

            void display()
            {
             cout << x <<y<< z;

              }

};
```

```
int main()
{
   A a;

   a.x = 5; //private not accessible

   a.y = 10; // protected = private for outside the class

   ∴ y is also not accessible.

   a.z = 20; // valid

   a.display(); //valid

   return 0;
}
```

```
class B : public A
{
    // x not accessible, because x is private.
    // But y is accessible, it means protected is accessible
       in child class.
    ∴ For child class protected = public.

    void display()
    {
        cout << y << z;
    }
};
```

```
int main()
{
    B b;
    b.y = 5; // error
             ∵ y is protected
    b.z = 10; //valid.
    return 0;
}
```



Security can only be increased.

public ⟶ protected ⟶ private

protected ⟶ private

private

---

max (public, public) = public

max (public, protected) = protected

max (public, private) = private

↑

for Public inheritance.

max (protected , public) = protected

max (protected , protected) = protected

max (protected , private) = private

For protected inheritance

max (private , public) = private

max (private , protected) = private

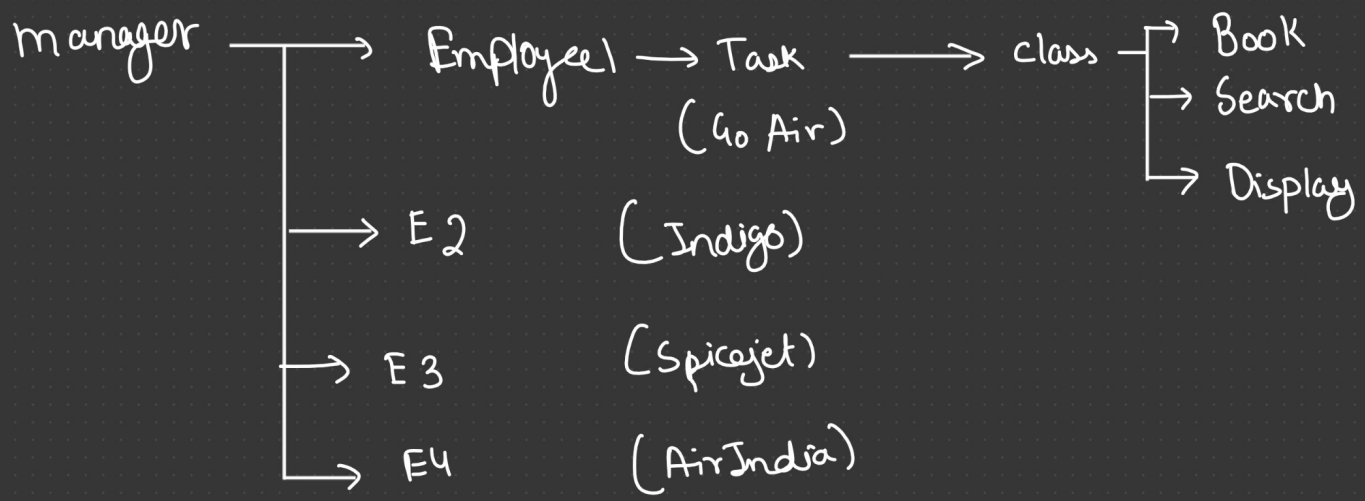
max (private , private) = private.

for private inheritance

	public	protected	private ← parent
public	✓	✓	✗
protected	✓	✓	✗
private	✓	✓	✗

child

Abstract class / Pure virtual function



E1 → Book() , E2 → book , E3 → Booking , E4 → Book()

main()
→ Book
→ GoAir ✓
→ AirIndia ✓

Class Flight → Abstract class
{
public :

E1 , E2 , E3 , E4 inherite my class

Virtual Void Book() = 0;
Virtual Void Search() = 0; → Pure Virtual function

Void display()
{
cout<<"welcome";
}
};

* A function which do not have defination & virtual keyword is used in it with =0; is known as pure function.

* If a class contains atleast one pure virtual function then it is known as Abstract class.

What is the use of pure virtual fn / Abstract class ?

* We can't create an object of Abstract class. Because it is an incomplete class.

* If we inherit an Abstract class in another class then we need to define all the pure virtual fn in our class, Otherwise our class will also becomes Abstract class.

Inheritance & Dynamic Memory Allocation

① new → delete

new ↓ Constructor

delete ↓ distructor

② Child Constructor → parent constructor

Calls

execute

③ Child destructor execute → parent destructor execute.

④ Every class contains →

- default constructor
- copy constructor (Shallow)
- Assignment operator (shallow)
- Destructor.

Scenario 1 :-

```
Class Parent
{
    char *s;
    int age;

public :
    Parent()
    {
        s = new char [10];
    }
    // Copy constructor need to define (Deep Copy)
    // Assignment operator (Deep Copy)
    // Destructor delete [] s; (It should be virtual)
};

class Child1 : public Parent
{
    ⟶ But this class does not doing any DMA.
       than No need to worry.
};
```

⟶ if I am using DMA than, I need to define following fns.

Scenario 2:-

```
Class  child2 : public parent
{
    char *p;
    public :
        child2()
        {
            p= new char[20];
        }

        ~child2()
        {
            delete [] p;
        }

        child2( const child2 & x) : parent(x)
        {
            p = new  char[ strlen( X.s) +1];
            strcpy (p, x.s);
        }
```

It is necessary to call parent cc from child cc

string of parent class

```
Child2 & Operator = ( const Child2 & x )
{
    if ( this == &x)
        return *this;
    parent :: Operator = (x);
    delete [] p;
    p = new char[ strlen(x.s)+1];
    strcpy( p, x.s);
    retur *this;
}
```

← we need to call parent class Assignment operator from child class Assignment operator. So that Deep copy of parent object can be done.

Summary :- when both parent & child class has DMA.

The derived-class destructor, copy constructor & assignment operator all must use their base-class counterparts to handle the base class component.

1) For destructor it is done automatically.

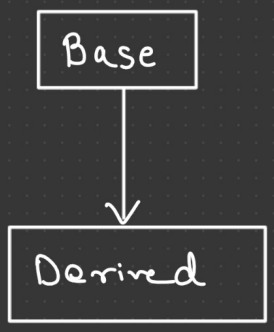
2) For constructor, it is accomplished by invoking the base class copy-constructor in member intialization list, or else the default constructor is invoked

automatically.

3) For assignment operator, it is accomplished by using the scope resolution operator in an explicit call of base-class assignment operator.

Types of Inheritance

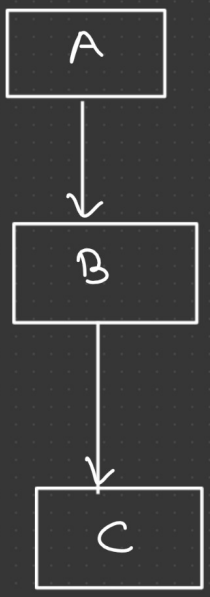
1) Single Inheritance :-




```
Class Base
{

};

class Derived : public Base
{

};
```

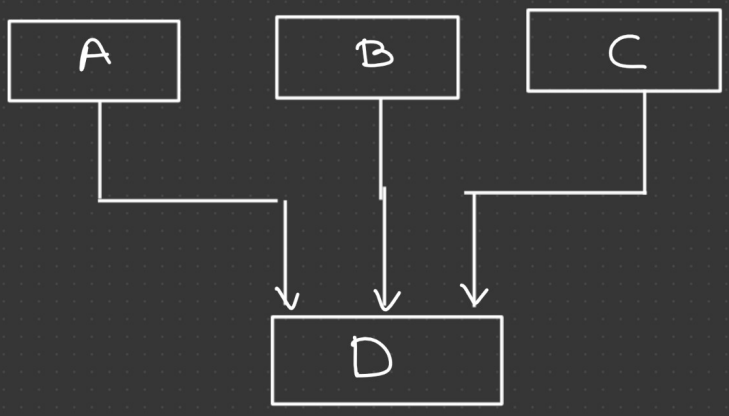
Visibility Mode

2) Multilevel Inheritance :-




```
class A
{
  ---
  ---
};

Class B : public A
{

};
Class C : public B
{

};
```

3) Multiple Inheritance :-



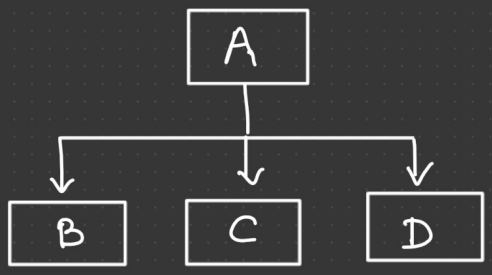

```
Class A
{
    int x;
};
```

```
Class B
{
    int x;
};
```

```
Class C
{
    int x;
};
```

```
Class D : public A, public B, public C
{
    cout<< x;      <-- error
    cout<< A::x;   <-- correct

    B::x;
    C::x;
};
```

4) Hierarchical Inheritance

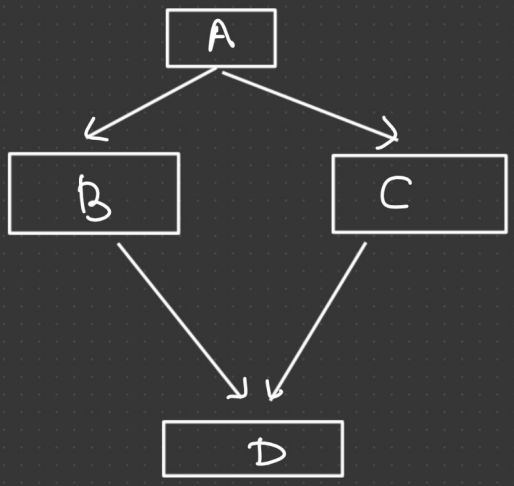



```
class A
{

};
```

```
class B : public A
{

};
```

```
class C : public A
{

};
```

5) Hybrid Inheritance :— Hir + Multiple

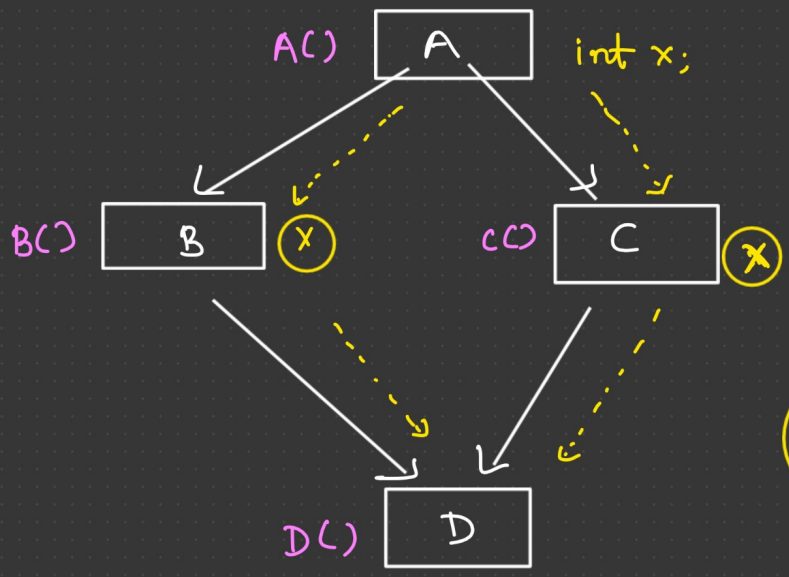


```
class Shape
{
   public :
        virtual void area() = 0;

};

class Rectangle : public Shape
{
   public :
        void area()
        {
          Cout<<"Rectangle";
        }

};

class Triangle : public Shape
{
    public :
          void area()
          {
            Cout<<"Triangle";
          }

};
```

```
Class Object : public Rectangle , public Triangle
{
    public:
        void area()
        {
            Rectangle::area();
            Triangle::area();
        }
};

int main()
{
    Object O;
    O.area();

    return 0;
}
```

Diamond Problem

The Diamond problem is an ambiguity that arises in multiple inheritance when 2 parent classes inherit from the same grandparent class, & both parent classes are inherited by a single child class.



B::x
C::x
→ Not possible

:: x is present only in class A.

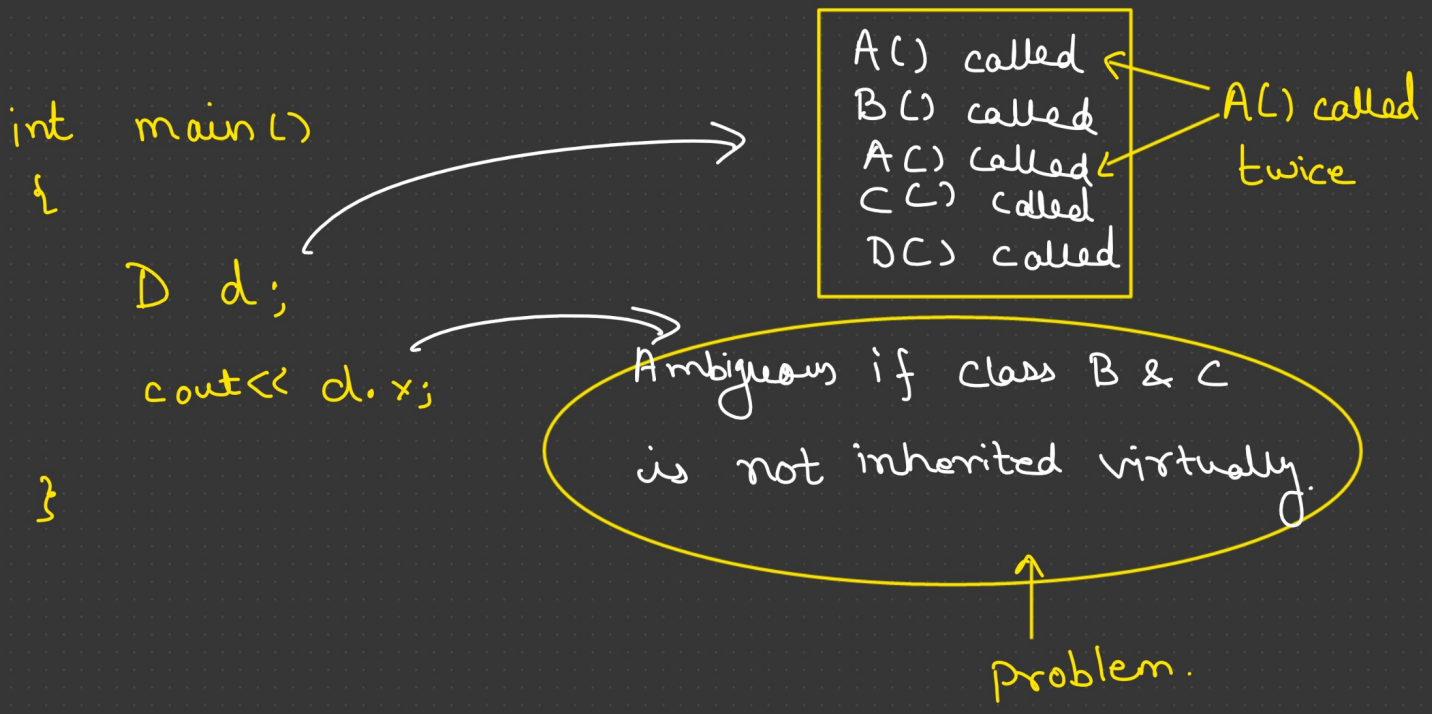
```
class A
{
    public :
        int x;

        A()
        {
         cout<<" A() called ";
        }
};
```

virtual (solution) ← virtual

```
Class B : public A
{
   public :

       B()
       {
        cout<<" B() called";
       }
};
```

```
class C : public A
{
  public :

       C()
       {
        cout<<"C() called";
       }
};
```

```
Class D : public B, public C
{
    public :
          D()
          {
           cout<<" D() called";
          }
};
```



Solution :-

We need to inherit B & C virtually. So that grandparent object not created twice at the time of creation of Class D object.