# Inheritance

* It is the process of acquiring features of an existing class into a new class.

* The class that inherits properties is called the sub class or derived class or child class.

* And the class that provides properties is called the super class or base class or parent class.

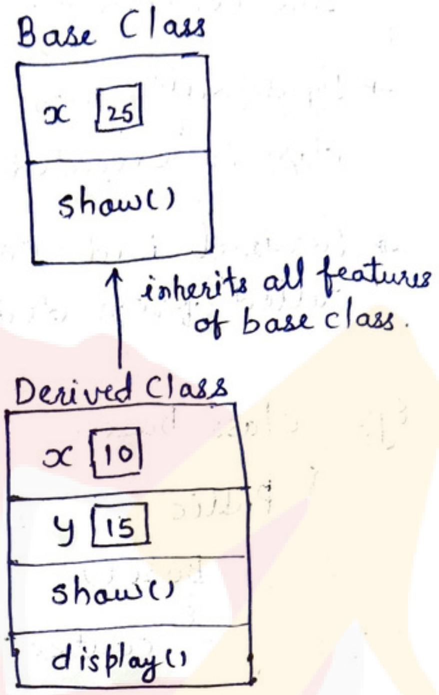
Eg:-

```
class Rectangle
{
    int length;
    int breadth;
    - - - -
    - - - -
    - - -
};

class cuboid : Rectangle     ( Inherits the property
                                of Rectangle class)
{
    int height;
    - - -
    - - - -
    - - -
};
```

Everything that is there in rectangle to be borrowed in cuboid class also.

Example -

```
class Base
{
   Public :
      int x;
      void show ()
      {
         cout << x;
      }
};

class Derived : public Base
{
   Public :
      int y;
      void display ()
      {
         cout << x << y;
      }
};

int main ()
{
   Base b;
   b.x = 25;
cout<<b.show (); → 25

   Derived d;
   d.x = 10;
   d.y = 15;
   ~~d.sho~~ cout<< d.show (); → 10
            cout<< d.display (); → 10, 15
```



## * Constructor in Inheritance -

→ Constructor of base class is executed first then the constructor of ~~base~~ derived class is executed.

→ By default, non-parameterised constructor of base class is executed.

→ Parameterised constructor of base class must be called from derived class constructor.

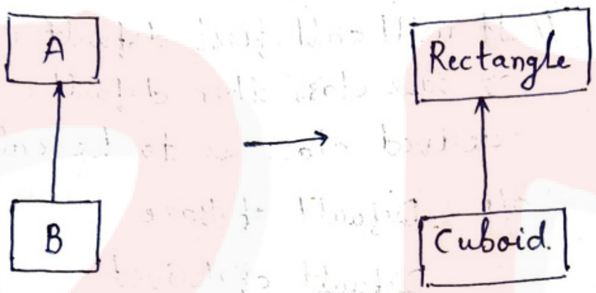
Eg:-
```
class Base
{ Public :
      Base ()
      {
        cout<<" Default of Base "<< endl;
      }
      Base (int x)
      {
        cout<< "Parameterised of Base" << x << endl;
      }
};
class Derived : Public Base
{ Public:
      Derived ()
      {
        cout << " Default of derived "<< endl;
      }
      Derived (int a)
      {
        cout << "Parameterised of derived" << a << endl;
      }
Derived ( int x, int y ) : Base (x)      → By using this syntax
                                          we are able to call
                                          parameterised const. of
                                          Base class.
{
    cout<<"Parameterised of derived"<< y << endl;
}
};

int main()
{
   Derived d;      // It will call first default constructor
                      of Base class then default const. of
                      derived class is to be called.

                   o/p - Default of Base
                         Default of derived

   Derived d(10);  // It will also execute first default
                      const. of Base then parameterised
                      const. of derived class is to be
                      executed.

                   o/p - Default of Base
                         Parameterized of derived 10

   Derived d (30, 40);  // It will call the first
                           parameterised of Base

                   o/p - Parameterised of Base 30
                         Parameterised of derived 40
}
```

(Above `x`: 30; above `y`: 40; above `Base (x)`: 30.)

## * Types of Inheritance -

i) Single Inheritance -

When a class is inherited from an existing class.

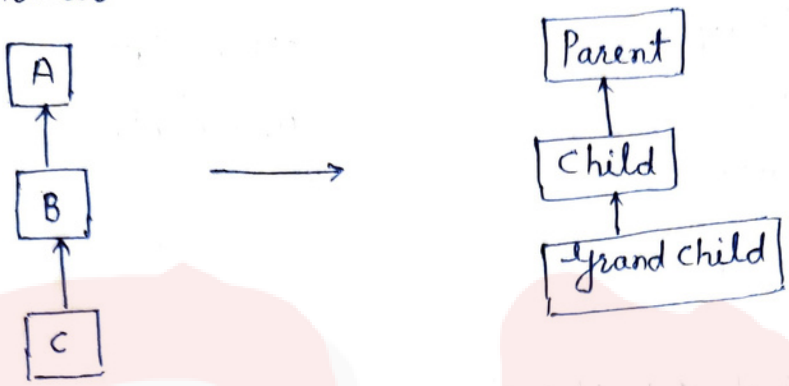


Eg:-
```
class Rectangle
{
  ~~Private:~~ Public:
     int length;
     int breadth;
  Public:
     int area()
     {
        return length * breadth;
     }
};
class Cuboid : Public Rectangle
{  Public:
      int height;
    int volume() { return length * breadth * height; }
};
```

```
int main()
{
  Cuboid c;
   c.length = 4;
   c.breadth = 5;
   c.height = 3;
  cout<<"Volume = "<< c.volume();
  cout<<"Area :"<< c.area();
```

## ii) Multilevel Inheritance -

When each class are inherited from one another.



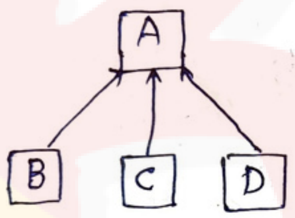
Eg:-
```
class Parent
{
 Public:
    void show()
    {
     cout<< "Parent class"<< endl;
    }
 };
class Child : Public Parent
{
 Public:
    void get()
    {
     cout<<"Child class"<< endl;
    }
};
class Grandchild : Public child
{
 Public:
    void display()
    {
     cout<<"grand Child class"<< endl;
    }
};
```

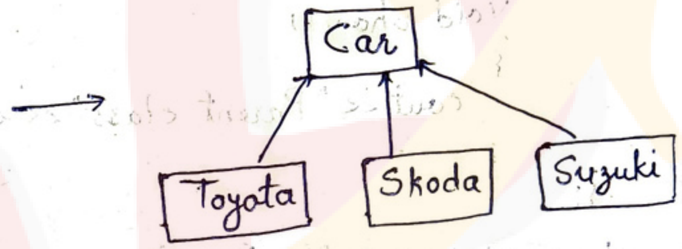
```
int main()
{
    Grandchild g;
    g.display();
    g.get();
    g.show();
}
```

o/p - Grand child class
Child class
Parent class

iii) Hierarchical Inheritance-

When more than classes are inherited from a single class.





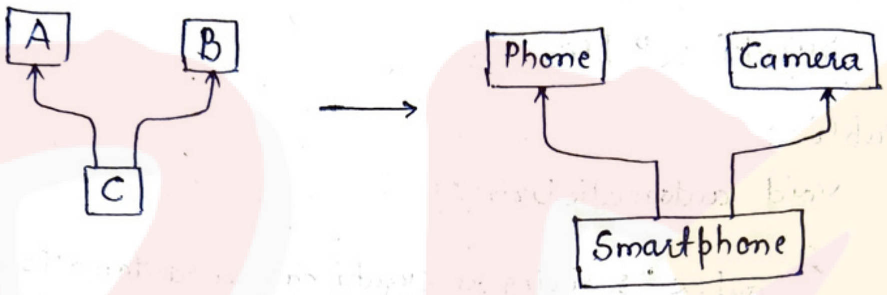
Eg:-
```
class Car
{
 Public:
    void drive()
    {  cout << "Driving a car" << endl;
    }
};

class Toyota : Public Car
{
 Public:
    void hybridCar()
    {  cout<< Driving a car in hybrid mode"<< endl;
};  }
```

```
class Skoda : Public Car
{
  Public:
     void sportDrive()
     {
       cout<<"Driving a Skoda car in sport mode";
     }
};

class Suzuki : Public Car
{
  Public:
     void automaticDrive()
     {
       cout<<"Driving a suzuki car in automatic mode";
     }
};
int main()
{
   Car c;                      o/p
   c.drive()         →   Driving a car

  Toyota t;
   t.drive()         →  Driving a car
   t.hybridCar(); → Driving a Toyota car in hybrid mode

  Skodo sk;
   sk.drive;         →  Driving a car
   sk.sportDrive(); → Driving a skoda car in sport mode

  Suzuki sz;
   sz.drive;              → Driving a car
   sz.automaticDrive(); → Driving a suzuki car in automatic
                                                       mode
}
```

## iv) Multiple Inheritance -

When a class are inherited from more than one class.

→ It means, for one class, there can be more than one base classes.

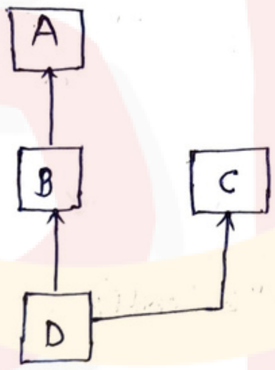


```
class Phone
{
 Public:
    void property1()
    {
       cout<<"Phone's property"<< endl;
    }
};
class Camera
{
 Public:
    void Property2()
    {
      cout<< "Camera's property "<< endl;
    }
};

class SmartPhone : Public Camera, Public Phone
{
 Public:
    void display()
    {
      cout<< " SmartPhone" << endl;
    }
};
```

```
int main()
{
    SmartPhone sp;
     sp.property1();      → Phone's property
     sp.property2();      → Camera's property
     sp.display();        → Smartphone
}
```

## v) Hybrid Inheritance -

When more than one inheritance are mixed each other then it is known as hybrid inheritance.



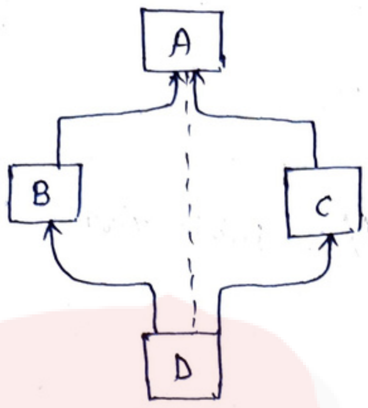
$$\begin{tikzcd} & A \\ & B \arrow[u] & C \\ & D \arrow[u] \arrow[ur] \end{tikzcd}$$

( combination of Multilevel and Multiple Inheritance)

Eg:-1)
```
class A
{
 Public :
     void display1()
      {
        cout<< "class A" << endl;
      }
};

Class B : public A
{
 Public:
    void display2()
    {
      cout<<" Class B" << endl;
    }
};

class c
{
 Public:
    void display3()
     {
      cout << "Class C " << endl;
     }
};
Class D: Public c, Public B
{
 Public:
    void display4() { cout<<" Class D" <<  endl; }
};
int main()
{
   D  obj ;
   obj. display1();
   obj .display2();
   obj. display3();
   obj. display4();
}
```

o/p -
```
Class A
class B
class C
class D
```

## * Hybrid Inheritance using Virtual function -



( combination of hierarchical and multiple inheritance)

→ When we use this type (above) of inheritance, then the features of the class 'A' will be available in class 'D' via class 'B' and class 'C'. It is called as multipath inheritance.

→ It means that class 'D' is getting the features of class 'A' via 'B' as well as 'C'. Due to this reason, 'D' will have two copies.

→ And it creates the ambiguity.

→ To remove this ambiguity, we use the concept of virtual Base class.

→ To avoid duplicacy, we make parent class as virtual.

Eg:-
```
class A
{
  Public :
      display1()
      {
         cout<< " Class A " << endl;
      }
};

Class B : Virtual Pubic A        // or Public virtual A
{
  Pubic :
      void display2()
      {
         cout<< " Class B "<< endl;
      }
};

class C : virtual Public A
{
  Public :
      void display3()
      {
         cout<< "Class C " << endl;
      }
};

class D : public C, Public B
{
  Public :
      void display4()
      {
         cout << "Class D" << endl;
      }
};
int main()
{
    D obj;
    obj. display1();
    obj. display2();
    obj. display3();
    obj. display4();
```