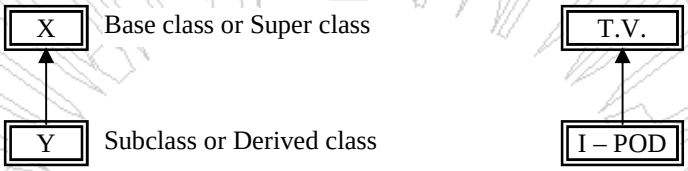
INHERITANCE ⇨ Inheritance is the capability of one class to inherit the properties from another class. Inheritance generates a model that is closer to the real world.

NEED FOR INHERITANCE ⇨

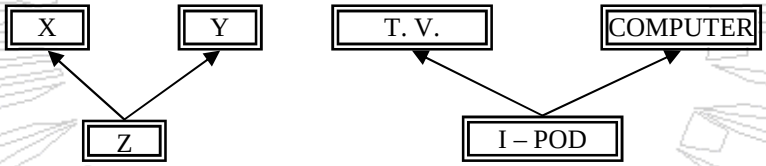
① Closeness with the real world model
② Reusability of codes
③ Faster development of a program time
④ Easier to maintain a program
⑤ Easy to extend a program
⑥ Transitive nature of inheritance

DIFFERENT FORMS OF INHERITANCE ⇨

① **SINGLE INHERITANCE** ⇨ When a subclass inherits only from one base class, it is called single inheritance.

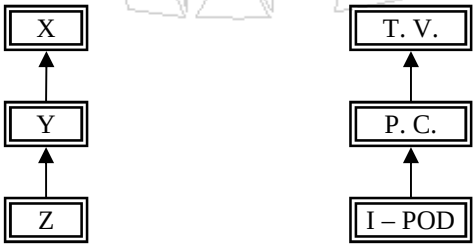


② **MULTIPLE INHERITANCE** ⇨ When a subclass inherits from two or more than two base classed, it is known as multiple inheritance.

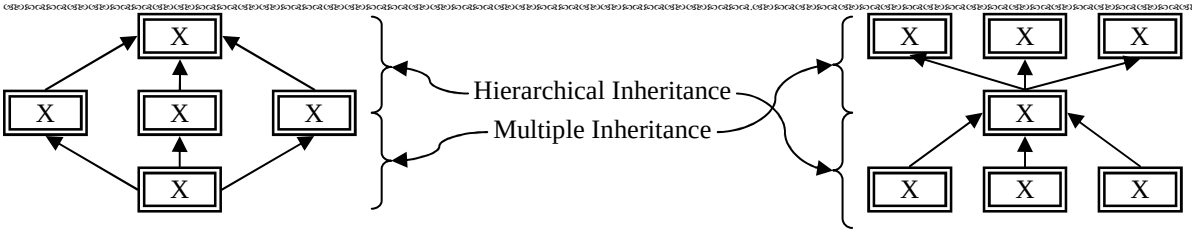


③ **HIERACHICAL INHERITANCE** ⇨ When two or more than two subclasses inherit from one base class, it is called hierarchical inheritance.

④ **MULTILEVEL INHERITANCE** ⇨ When a subclass inherits from one class that itself inherits from another class, it is called multilevel inheritance. Multilevel inheritance represents the transitive nature of inheritance.



⑤ **HYBRID INHERITANCE** ⇨ When two or more than two forms of inheritance are mixed, it is called hybrid inheritance.

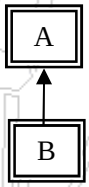


SYNTAX FOR SINGLE INHERITANCE ⇨

```
class name_of_subclass : visibility_mode  baseclass_name
{ ……
  ……  //members of subclass
} ;
```

The visibility mode controls the visibility and visibility of baseclass. The visibility mode may be either private, public or protected.

Ex:
```
class B : public A
{ ……
  …… //members of subclass
} ;
```

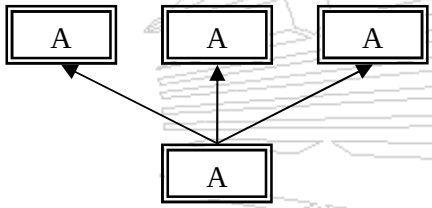


Note:✍ Actually a derived class inherits all the members of a base class; however, the derived class has access-rights only to the non-private (protected and public) members of base class.

SYNTAX FOR MULTPLE INHERITANCE ⇨ A

```
class subclass_name : vis_mode  I-baseclass_name, vis_mode II-baseclass_name, ……
{ ……
  …… //members of derived class
} ;
```

Ex:
```
class D : public A, bublic B, private C
{ ……
  …… //members of derived class.
```



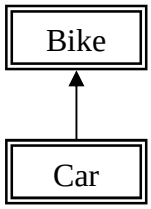
Note-1:✍ We must derive in public visibility mode, when we want that the derived class may inherit all the attributes/properties of the base class, plus some extra attributes/properties.
Note-2:✍ We must derive in private visibility mode when we want that the derived class may use some attributes of base class and these inherited attributes may not be inherited further.
Note-3:✍ We must derive in protected mode, when we want that the attributes of base class may be hidden from the outside world.

WORKING OF CONSTRUCTOR AND DESTRUCTOR IN INHERITANCE ⇨ When an object of a derived class is created, first the base class constructor is invoked/executed.

When an object of a derived class expires, first the derived class destructors are invoked/ executed, then the base class destructor id invoked/executed.

We must keep in mind that the constructor or destructor of base class is not inherited by the derived class. In such a situation, the constructor or destructor of base class is used by the derived class in explicit manner as is shown in the following example:

```
class Bike{      char engine[25] ;
                 int capacity ;
           public:
                 int wheel_no ;
                 int convert_engine( ) ;
         } ;
class Car : public Bike
{     int axel_no ;
      int chesis_no ;
   public :
      char gear_type[25] ;
      int pickup( 0 ;
} ;
int main ( )
{  car Maruti ;
   ……
   ……
}
```



When the object Maruti is created, firstly the base class constructor Bike : : Bike( ) is invoked and then the derived class constructor Car : : Car( ) is invoked. At the end of main( ) function, when the object Mruti expires, firstly the derived class destructor Car : : ~Car( ) is invoked and then the base class destructor Bike : : ~Bike( ) is invoked.

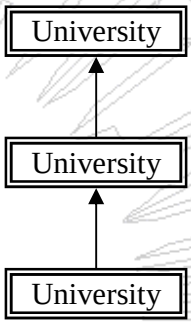
SOME FACTS ABOUT INHERITANCE ⇨

Fact-① A derived class inherits all the members of the base class, but the derived class can directly access only to the non-private members of the base class.

Fact-② The size of a class (in bytes) as well as its object is sum of sizes of all its data members. The size of a derived class is sum of base class-size (including friendly member of class) and sum of its own data members. If we declare a class without any data member then the class will have size of 1 byte.

EBSQ: Consider the following and answer the question given below.

```
#include<iostream.h>
class University{     int NOC ; // No. of colleges
                  protected :
                     char Uname[25] ; //University name
                     char State[25] ;
                     void EnterData( ) ;
                     void DisplayData( ) ;
                  } ;
class College : public University
          {     int NOD ; //No. of Departments
                char Cname[25] ; //College name
              protected :
                   void Affiliation( ) ;
              public :
                   College( ) {     } ;
                   void Enroll( int, int ) ;
                   void Show( ) ;
          } ;
class Department : public College
          {     char Dname[25] ; //Department name
                int NOFM ; //No. of Faculty Mewmbers
              public :
                   Department ( ) {      }
                   void Display( ) ;
                   void Input( ) ;
          } ;
```



(i) Which class's constructor will be called first at he time of creation of an object of class Department ?

(ii) How many bytes does an object belonging to class Department require?

(iii) Name member functions which are accessed from the object(s) of class Department?

(iv) Name the data member(s) which are accessible from the object(s) of class College.

Sol : (i) The constructor of class will be called first.

(ii) Size of class Department is calculated as given below:

Object of University = 2 + 25 + 25 = 52
Object of College = 2 + 25 = 27
Object of Department = 25 + 2 = 27
Hence, Total size = 52 + 27 + 27 = 106 byte

Data type	Size (in bytes)
char	1
int	2
float	4
long	4
long float	8

(iii) EnterData( ) , DisplayData( ), Enroll( ), Show( ) Input( ), Affiliation( ).

(iv) Uname, Dname, NOFM.

Fact-③ The private members of the class are visible in the derived class but they are not directly

accessible.

```
Ex:   int Test ;        //global variable
      class I_UnitTest{   int test ; //privat members hides global test
                          public :
                              void getit( )
                              { cin>>test ; }
      class II_UnitTest : public I_UnitTest
      {  public :
            void check( )
            {
               test++ ;      //not allowed, even with global test because global test is hidden here also
                                 even though visible here but not directly accessible.
            }
      } ;
```

Fact-④ We cannot deny access to certain members of a base-class when inheriting publicly.

```
Ex:   class Animal{   public :
                          int a ;
                      private :
                          Animal : : x ; //invalid ; not allowed
                  } ;
```

But we can allow access to some of the class members when deriving privately.

```
Ex:   class Animal {  public :
                          int x, y, z ;
                      } ;
      class Human : private Animal
      {  public :
            Animal : : x ; // allowed access to x of class Animal
            int a ;
      } ;
```

ABSTRACT CLASS ⇨ Abstract class is the base class from which other classes are derived with the condition that no object of this base type exists.

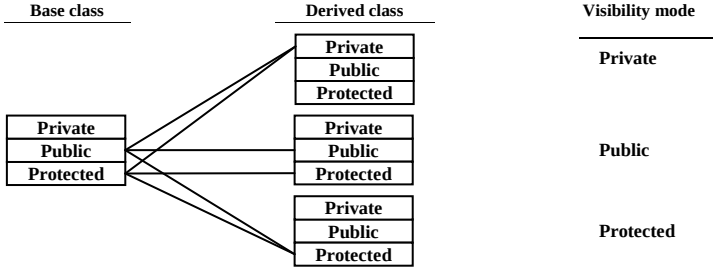
MAKING A PRIVATE MEMBER INHERITABLE ⇨ We know that the derived class can directly access only to the public and protected members of the base class. If we need to have access privilege to the private members of the base class by the derived class, then this can be done in following two methods:

(i) By making the visibility mode of private members as public,
(ii) By making the visibility mode of the private members as protected.

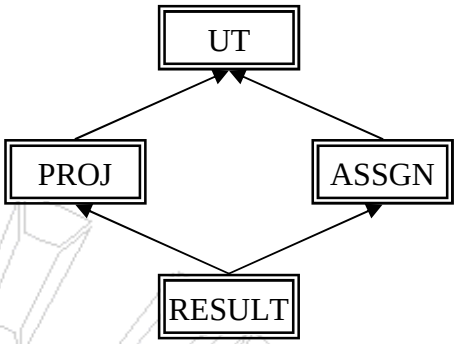
The first method has a serious drawback that private members become accessible to all other functions of the program, hence breaking the benefits of data-hiding.

The second method retains the data hiding of the private members and at the same time makes it inheritable. Hence private members become accessible to the member functions and friends of the base class as well as derived class.

RELATIONSHIP OF INHERITANCE AND VISIBILITY MODE ⇨ This is shown in the following figure. It also shows the mechanism of access-control to the members of a base class.



VIRTUAL BASE CLASS ⇨ Virtual base class means inheritance of subclass from the base class as virtual, so that copies of data members of a base class might be checked and ambiguity might be prevented while inheriting by the derived class. Generally, ambiguity enters when hybrid inheritance is used, as is given in the following example:




```
Ex:   #include<iostream.h>
      class UT{ public :
                    Int UTM ;  //unit test marks
                } ;
      class PROJ : public UT  // PROJ inherits form UT
            {  public :
                    int PRM ; // Project marks
            } ;
      class ASSGN : public UT  //ASSGN also inherits form UT
            {  public :
                    int ASM ; // Assignment marks
            } ;
      class RESULT : public PROJ , public ASSGN // RESULT inherits from both UT and ASGN, so
                                                   there are two copies of UT
            {  public :
                    int TOTAL ;
            } ;
      void main( )
      {  RESULT Arun ; //object Arun of RESULT type created
         Arun•UTM = 35 ; //This is ambiguous, which UTM. Since UTM is in PROJ as well as in
                              ASSGN, and RESULT inherits from both.
         Arun•PRM = 35 ;
         Arun•ASM = 35 ;
         Arun•TOTAL = Arun•UTM + Arun•PRM + Arun•ASM  ;//UTM is ambiguous still
         cout << "\nTotal marks of Arun is =" << Arun•TOTAL ;
      }
```

Now, let us see, how this ambiguity can be done away (terminated, got rid of) with the help of virtual base class Write the same (above) program with following changes:

```
class PROJ : virtual public UT// PROJ inherits UT virtually
class ASSGN : virtual public UT
class RESLUT : public PROJ, public ASSGN //This time here in only one copy of UT.
```

MULTILEVEL INHERITANCE ⇨ The situation of inheritance where, a derived class acts as a base class is called multilevel inheritance. The chain of classes forming multilevel is called inheritance-hierarchy or inheritance path. There can be any number of classes in inheritance hierarchy.

NESTING OF CLASSES ⇨ When a class contains object of another class types as its member, it is referred to as nesting of classes or containership or containment or composition or aggregation. We must not confuse between nesting of classes and nested class.

```
Ex:   class A { …… } ;
      class B { …… } ;
      class C {   A Ob1 ;
                  B Ob2 ;
            } ;
```

Nesting of Classes

```
Class A { ……
          ……
          class B { …… } ;
        } ;
```

Nested Class

RELATIONSHIP BETWEEN CLASSES ⇨ When a class inherits from another class, the derived class has "IS – A" relationship with its base class.

When a class contains objects of another class type, then containing/enclosing class has "HAS-A" relationship with contained/enclosed class.

The class having HAS-A relationship with other class has the ownership of contained object. Ownership means the responsibility ofr creation and destruction of an object.

A class that indirectly contains another object via pointer or reference, is said to have "HOLDS-A" relationship with class whose object is its indirect member.

THE END