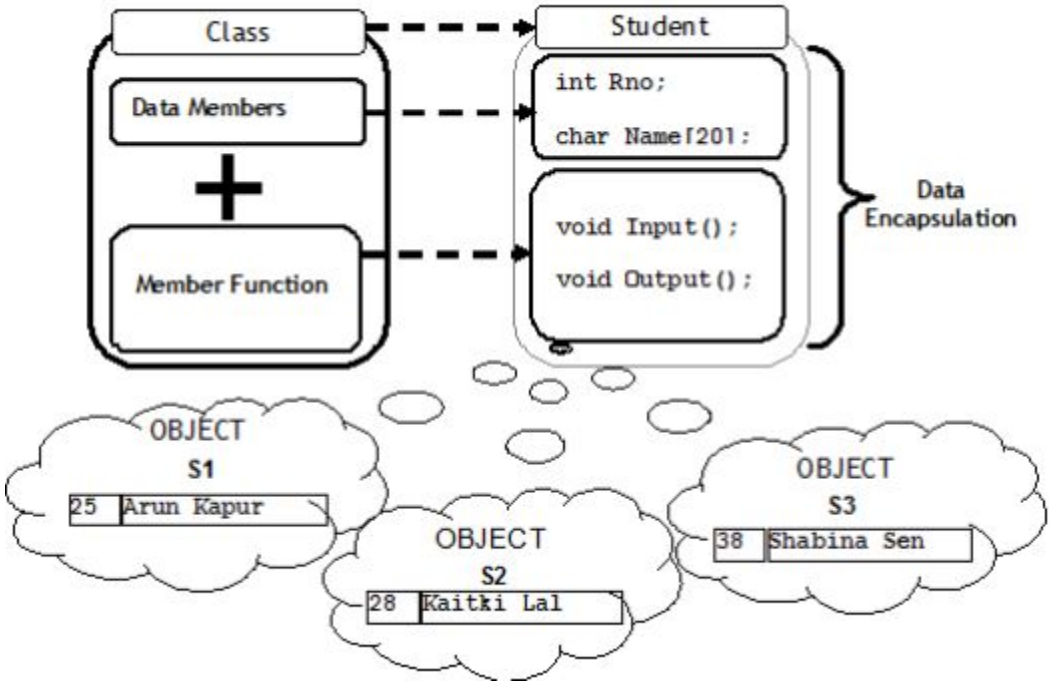
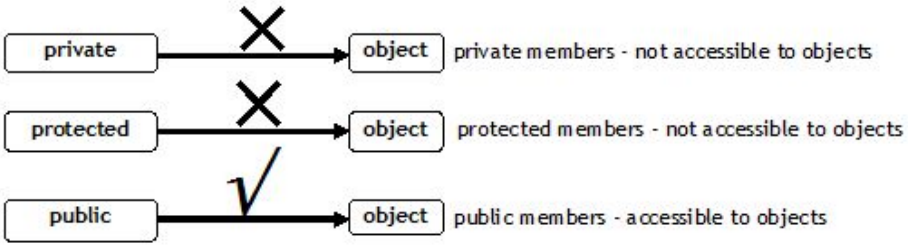
Class & Objects

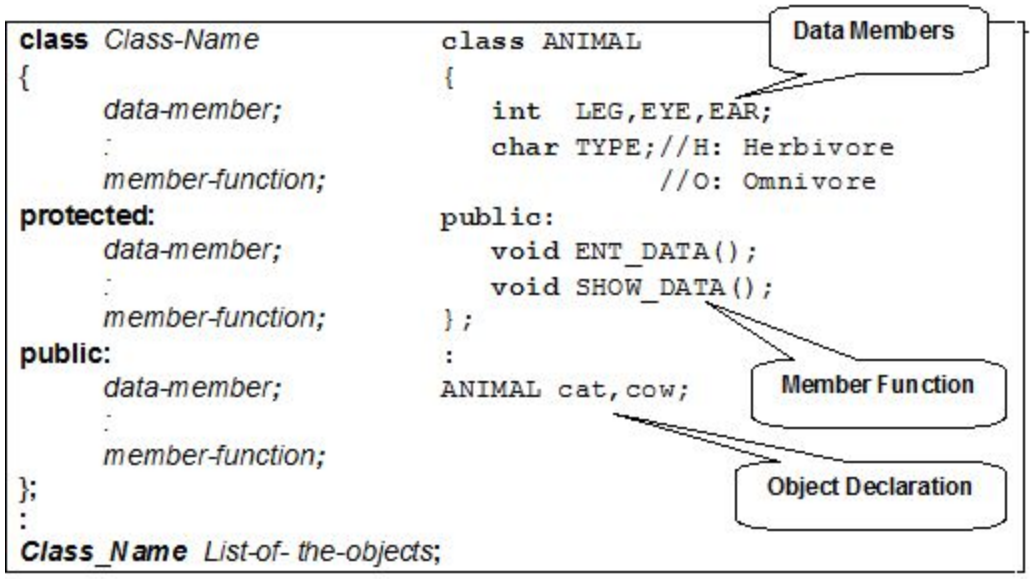
A class is used to encapsulate data and functions in a single unit. Contents of a class are known as **members** of the class; data inside the class is known as **data member** and function included inside the class is known as **member function**. Members of the class are private by default. (i.e. they are not accessible outside the scope of the class). An Object has the similar relationship to a class that a variable has to a data type. An object is said to be an instance of a class, in the same way 'YOU' and 'ME' are the instances of a class 'HUMAN'. Class may contain data as well as functions.



A class with no instances (no objects) is known as an **abstract** class and a class having objects is known as a **concrete** class. Every object has its identity, state and behaviour. ***Identity*** is the property of the object, which distinguishes the object from other objects. Static and dynamic properties associated with the object signify the ***state*** of the object. The operations/functions associated with the object exhibit the ***behaviour*** of the object. A class has three visibility modes for keeping its members:



In C++, a class can be defined with the following syntax:



Members of class can be kept in any of the three visibility modes private, protected and public. Members in ***public visibility mode*** of the class are accessible to the objects of the class whereas members in ***private visibility mode*** are not accessible to the objects of class (are only accessible inside the class to the member functions of the class).

Note: ***Protected members*** will be explained in the inheritance topic.

Following C++ program illustrates the use of class and objects:

```
#include <iostream.h>
const int Max=20;
class  Hospital
{
      int   Pno,Wardno;              //Private data members
      char Name[20];
   public:
      void Register();               //Public member function prototype
                                     //Public memberfunctions definition
      void ShowStatus(){ cout<<Pno<<Name<<Wardno<<endl;}
};                                   //Semicolon is must
void  Hospital::Register()           //memberfunctions definition
{                                    //outside the class
      cout<<"Patient No?";cin>>Pno;
      cout<<"Name      ?";cin.getline(Name,Max);//or gets(Neme);
      cout<<"Ward No   ?";cin>>Wardno;
}                                    //Semicolon not required
```

```
void  main()
{
      Hospital   P1,P2;  //Declaration of the objects P1,P2 of Hospital
      P1.Register();     //Member function call for the object P1
      P2.Register();     //Member function call for the object P2
      P1.ShowStatus();
      P2.ShowStatus();

      //cin>>P1.Wardno;  **NOT ACCESSIBLE as Wardno is a private
      //                   member
      //cin>>P2.Pno;     **NOT ACCESSIBLE as Pno is a private member
}
```

In the above example, one can see that a class Hospital is defined with three private data members Pno, Wardno and Name; and two public member functions Register() and ShowStatus(). Member function can be defined inside the class as well as outside the class; as shown in the example, member function ShowStatus() is defined inside the class whereas member function Register() is defined outside the class by using scope resolution operator (**::**) with the name of the class it belongs to. Both of these functions can be called from the objects of the class in the same way. Generally, single line functions are defined inside the class and multiple line functions are defined outside the class to have clarity in understanding the behaviour of the class. Functions with control structures should be defined outside the class.

The size of object (in bytes) depends upon the data members present in the class it belongs to. In the above example object P1 occupies 24 bytes.

Polymorphism: The process of using an operator or a function in different ways for different set of inputs given is known as polymorphism. Function overloading is an example of polymorphism, where the functions having same name with different set of parameters perform different operations.

Example:

```
void Disp()                    //Function 1
{
  cout<<"Hello"<<endl;
}
void Disp(int N)               //Function 2
{
  for (int I=1;I<=N;I++)
      cout<<I<<endl;
}
void Disp(int N,int M)         //Function 3
{
  for (int I=1;I<=M;I++)
      cout<<N<<"x"<<I<<"="<<N*I<<endl;
}
void main()
{
  int x=5,y=10;
  Disp(x);  //call for Function 2 - Prints numbers from 1 to 10
```

```
  Disp(x,y);//call for Function 3 – Prints table of 5 upto 10 multiples
  Disp();   //call for Function 1 – Prints Hello
}
```

Constructor: It is a special member function of class with the following unique features:

1. It has same name as the name of the class they belong to.
2. It has no return type.
3. It is defined in public visibility mode
4. It is automatically called and executed at the time of creation/declaration of the object
5. Moreover, constructor can be overloaded.

//Example of Constructor

```
class Trial
{
   int a,b;
public:
   Trial()        //Constructor
   {a=0;b=0;}
   void Disp()
   {
     cout<<a<<b<<endl;
   }
   void Raise(){a+=10;b+=10;}
   void Down(){a-=5;b-=5;}
};

void main()
{
   Trial T;       //Automatic call for Constructor
   T.Disp();
   T.Raise();
   T.Disp();T.Down();
   T.Disp();
}
```

//Example of Constructor Overloading

```
class Work
{
  int X,Y;
public:
  Work(){X=10;Y=30;}//Constructor 1
```

```
  Work(int C)        //Constructor 2 (Parameterized Constructor)
  {
    X=C;Y=2*C;
  }
  Work(int tx,int ty) //Constructor 3 (Parameterized Constructor)
  {
    X=tx;Y=ty;
  }
  :
  :
};
void main()
{
  Work W;         //Call of constructor 1
  Work R(30,60); //Call of constructor 3
  Work RR(30);   //Call of constructor 2
  :
  :
}
```

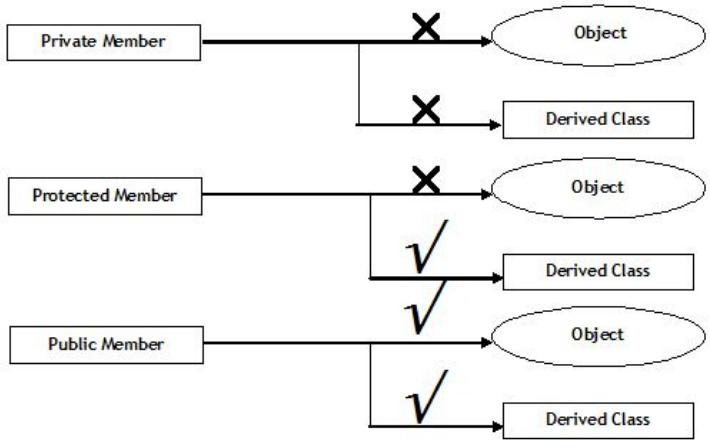
A copy constructor is an overloaded constructor function in which (an) object(s) of the same class is/are passed as a reference parameter(s). It is used when an object's data value is related to or is initialised using another object's data value of the same class. In the following example the values of data members of object **Q** are dependent on the values of data members of object **P**.

```
//Example of Copy Constructor
class Play
{
   int Count, Number;
public:
   Play();      //constructor
   Play(Play &);//copy constructor
   void Disp();
   void Change(int,int);
};
void main()
{
   Play P;           //Call for constructor
   P.Disp();
   P.Change(90,80);
   Play Q(P);        //Copy constructor call
   Q.Disp();
   Play R=Q;         //Copy constructor call [same as Play R(Q);]
```

```
   R.Disp();
}
Play::Play()    //constructor
{
   Count=0;
   Number=0;
}
Play::Play(Play &P) //copy constructor
{
   Count=P.Count+10;
   Number=P.Number+20;
}
void Play::Disp()
{
   cout<<Count;
   cout<<Number<<endl;
}
void Play::Change(int C,int N)
{
   Count=C;
   Number=N;
}
```

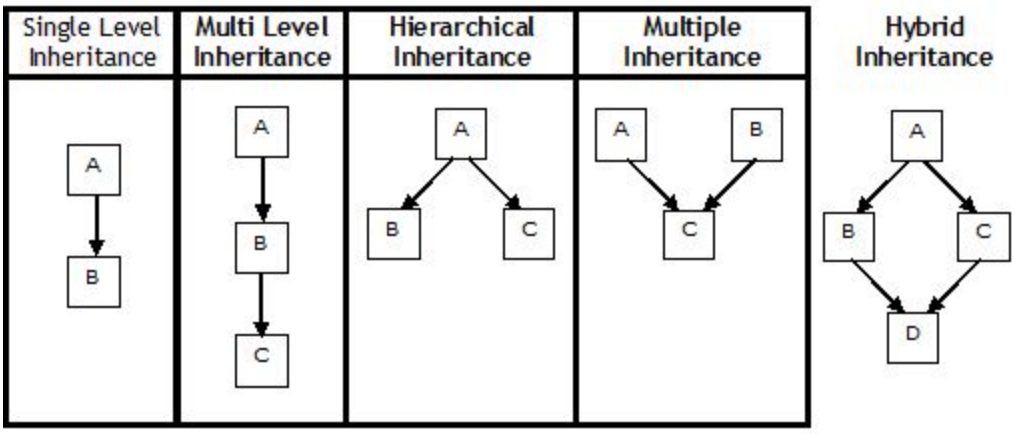
Inheritance: Inheritance is most powerful feature of Object Oriented Programming, after classes themselves. Inheritance is a process of creating new classes (derived classes) from existing classes (base classes). The derived classes not only inherit capabilities of the base class but also can add new features of their own. The process of Inheritance does not affect the base class. The most important aspect of inheritance is that it allows reusability of code, and also a debugged class can be adapted to work in different situations. Reusability of code saves money as well as time and increase program reliability. Inheritance is very useful in original conceptualization and design of a programming problem.

Visibility under various modes



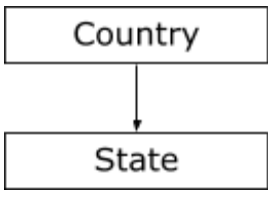
Base Class	Access Specifier	Derived Class
private	private	Not Accessible
	protected	
	public	
protected	private	private
	protected	protected
	public	
public	private	private
	protected	protected
	public	public

Types of Inheritance




```
class <Base Class Name>
{
      <Class Members>;
protected:
      <Class Members>;
public:
      <Class Members>;
};
class <Derived Class Name>:<Access Specifier> <Base Class Name>
{
      <Class Members>;
protected:
      <Class Members>;
public:
      <Class Members>;
};
```

Syntax
(Single Level Inheritance)



Example
(Single Level Inheritance)

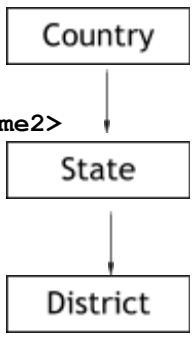
```
class Country
{
     int ArmyFunds;
protected:
     int Infrastructure;
public:
     void Provide();void Collect();
};
class State:public Country
{
     int No_of_Farms;
public:
     void Get();void Put();
};
```

Base Class

Derived Class

Syntax
(Multi Level Inheritance)

```
class <Base Class Name1>
{
     <Class Members>;
protected:
     <Class Members>;
public:
     <Class Members>;
};
class <Derived/Base Class Name2>:<Access Specifier> <Base Class Name1>
{
     <Class Members>;
protected:
     <Class Members>;
public:
     <Class Members>;
};
class <Derived Class Name3>:<Access Specifier> <Base Class Name2>
{
     <Class Members>;
protected:
     <Class Members>;
public:
     <Class Members>;
};
```



Example
(Multi Level Inheritance)

```
class Country
{
      int ArmyFunds;
protected:
      int Infrastructure;
public:
      void Provide();void Collect();
};
class State:public Country
{
      int No_of_Farms;
public:
      void Get();void Put();
};
class District:public State
{
      int No_of_Offices;
public:
      void Input();void Output();
};
```

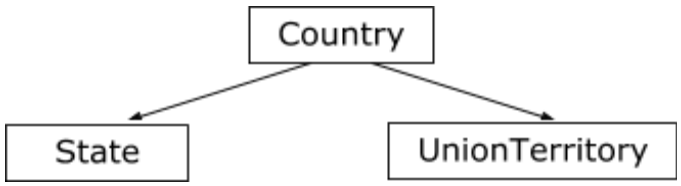
Base Class

Derived/Base Class

Derived Class

Syntax
(Hierarchical Inheritance)

```
class <Base Class Name>
{
      <Class Members>;
protected:
      <Class Members>;
public:
      <Class Members>;
};
class <Derived Class Name1>:<Access Specifier> <Base Class Name>
{
      <Class Members>;
protected:
      <Class Members>;
public:
      <Class Members>;
};
class <Derived Class Name2>:<Access Specifier> <Base Class Name>
{
      <Class Members>;
protected:
      <Class Members>;
public:
      <Class Members>;
};
```



Example
(Hierarchical Inheritance)

```
class Country
{
      int ArmyFunds;
protected:
      int Infrastructure;
public:
      void Provide();void Collect();
};
class State:public Country
{
      int No_of_Farms;
public:
      void Get();void Put();
};
class UnionTerritory:public Country
{
      int No_of_Offices;
public:
      void Input();void Output();
};
```

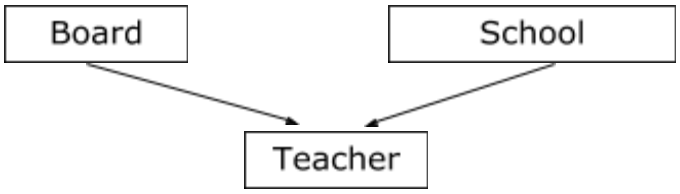
Base Class

Derived Class

Derived Class

Syntax
(Multiple Inheritance)

```
class <Base Class Name1>
{
      <Class Members>;
protected:
      <Class Members>;
public:
      <Class Members>;
};
class <Base Class Name2>
{
      <Class Members>;
protected:
      <Class Members>;
public:
      <Class Members>;
};
class <Derived Class Name>:<Access Specifier> <Base Class Name1>,
                           <Access Specifier> <Base Class Name2>
{
      <Class Members>;
protected:
      <Class Members>;
public:
      <Class Members>;
};
```



Example
(Multiple Inheritance)

```
class Board
{
     long No;
protected:
     float Budget;
public:
     void Register();void Display();
};
class School
{
     long SId; char Name[20];
public:
     void Get();void Put();
};
class Student:public Board,private School
{
     int AdmNo;
public:
     void Enroll();void Display();
};
```

Base Class 1

Base Class 2

Derived Class

Work Sheet

Answer the questions with the help of the portion of program shown below:

```
class U
{
     int X;
protected:
     int Y;
public:
     void Haveit();
     void Giveit();
};
class T:public U
{
     int X1;
     void Calc();
protected:
     int Y1;
```

```
    void Doit();
public:
    int Z1;
    void Showit();
    void Addit();
};
class P:private T
{
    int X2;
protected:
    int Y2;
public:
    int Z2;
    void Display();
    void Change();
};
```

a) Name the type of inheritance illustrated in the above example.
b) Name the member function(s), which can be accessed from the objects of, class P.
c) Name the data member(s), which can be accessed from the member functions of class T.
d) Name the data member(s), which can be accessed by the object of class U.
e) Is the data member Y declared in class U accessible to the member function Change() of class P?
f) Write the definition of member function Display() of class P to display all data members which are accessible from it.
g) Write the name of Base Class and Derived Class of Class T.

Object Oriented Programming & Procedural Programming

Object Oriented Programming	Procedural Programming
Emphasis is on data	Emphasis is on doing things (Functions)
Follows bottom-up approach in program design	Follows top-down approach in program design
Concept of Data Hiding prevents accidental change in the data	Due to presence of global variables, there is a possibility of accidental change in data.
Polymorphism, Inheritance, Data Encapsulation possible	Not Applicable

Data Encapsulation: Wrapping up of data and functions in a single unit is known as data encapsulation. In a **class**, we encapsulate the data and function together in a single unit.

Data Hiding: Keeping the data in private/protected visibility mode in a class to prevent it from accidental modification (change) is known as data hiding.

Destructor: It is a special member function of class with the following unique features:

1. It has same name as the name preceded by a symbol ~ (tilde).
2. It has no return type.
3. It is defined in public visibility mode
4. It is automatically called and executed when scope of an object gets over
5. Moreover, destructor functions can NOT be overloaded.

```
//Example of Destructor
class Item
{
public:
   Item(){cout<<"Item Manufactured..."<<endl;}
   void Working(){cout<<"Item is working"<<endl;}
   ~Item(){cout<<"Item destroyed..."<<endl;}
};
void main()
{
  Item I;
  I.Working();
  for (int C=0;C<2;C++)
  {
    Item J;
    J.Working();
  }// Call of Destructor for J
}// Call of Destructor for I
```

Constructor

Destructor

```
OUTPUT
Item Manufactured
Item is working
Item Manufactured
Item is working
Item Destroyed...
Item Manufactured
Item is working
Item Destroyed...
Item Destroyed...
```