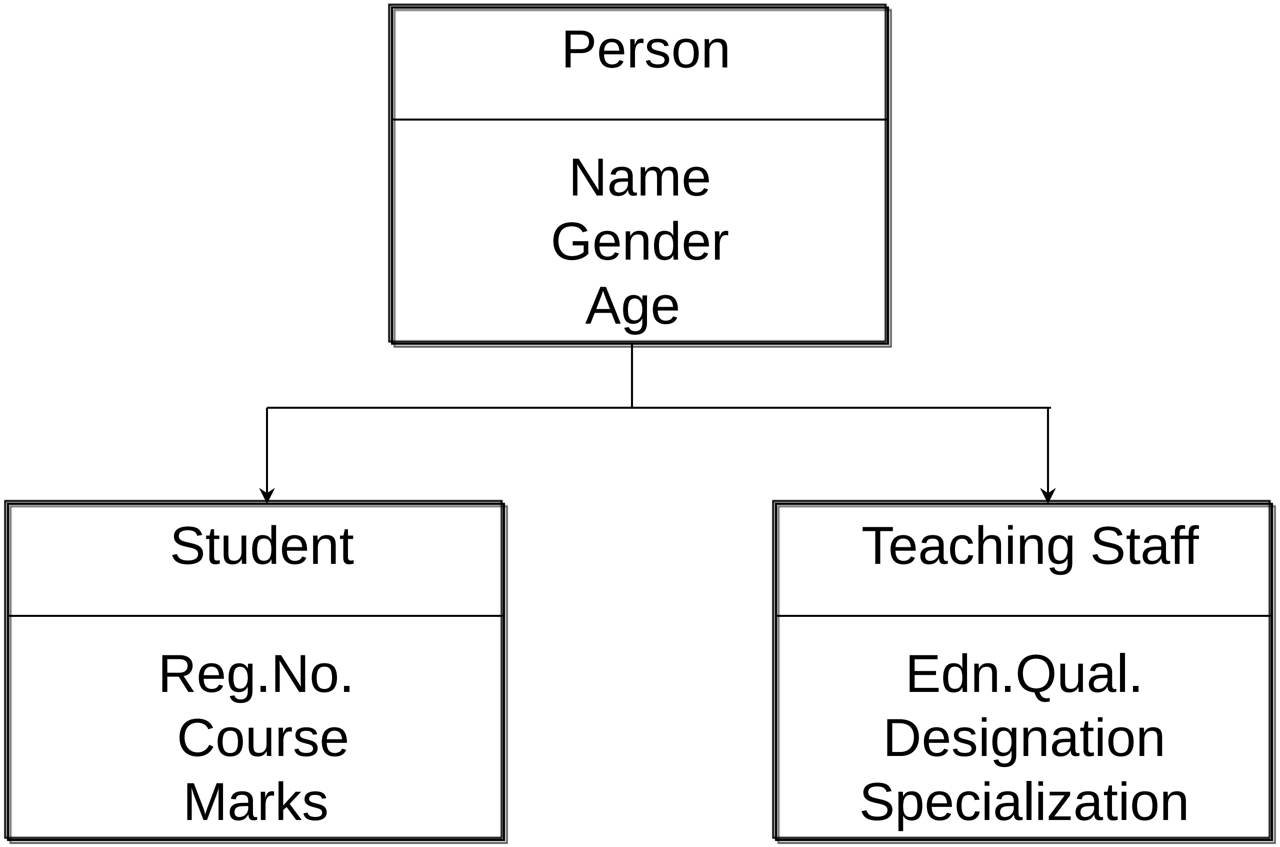
Inheritance

- By using the concepts of inheritance, it is possible to create a new class from an existing one and add new features to it.
- Inheritance provides a mechanism for class level Reusability.
- Semantically, inheritance denotes an *"is-a"* relationship.

Inheritance

- Inheritance is the relationship between a class and one or more refined version of it.
- The class being refined is called the superclass or base class and each refined version is called a subclass or derived class.
- Attributes and operations common to a group of subclasses are attached to the superclass and shared by each subclass.
- Each subclass is said to inherit the features of its superclass.

Inheritance



Person

Name
Gender
Age

Student

Reg.No.
Course
Marks

Teaching Staff

Edn.Qual.
Designation
Specialization

"Person" is a generalization of "Student".
"Student" is a specialization of "Person".

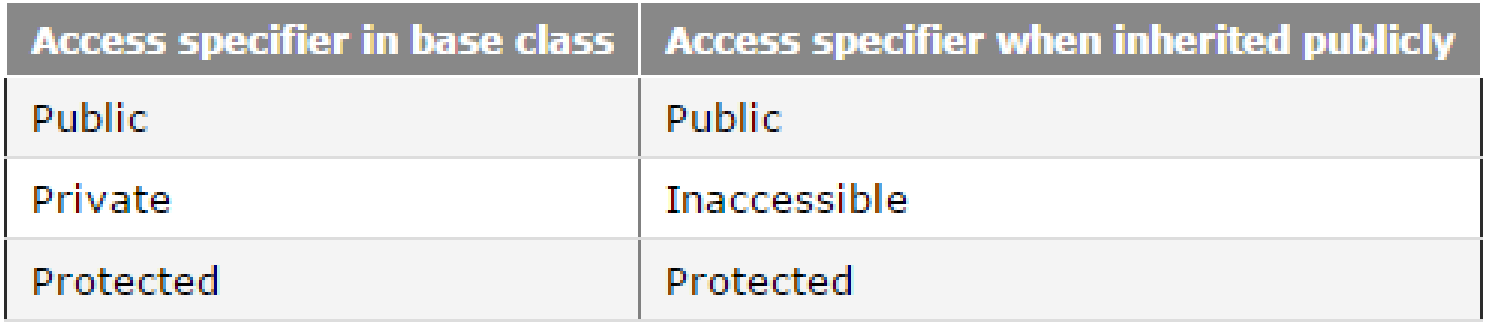
Defining Derived Class

- The general form of deriving a subclass from a base class is as follows

```
Class derived-class-name : [visibility-mode] base-class-name
{
            ………………….  //
            ………………….// members of the derived class
};
```

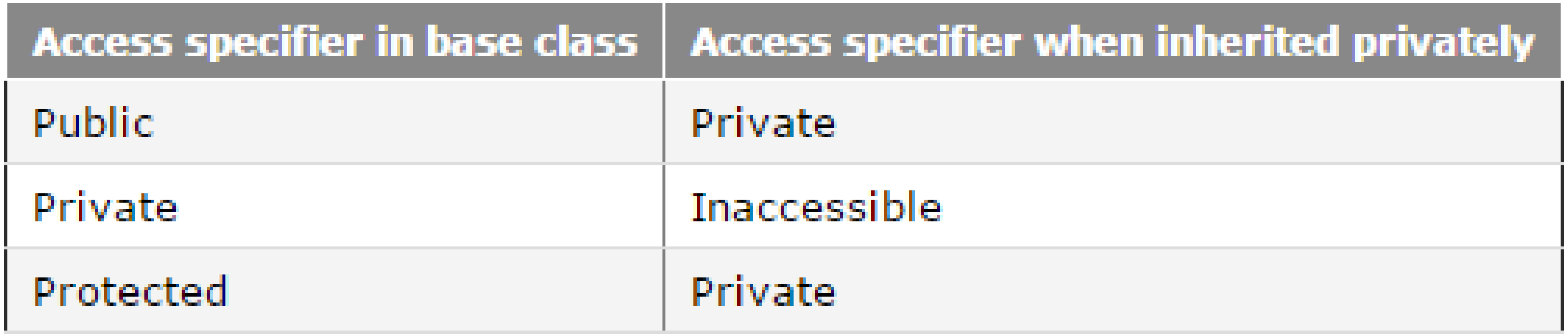
- The visibility-mode is optional.
- It may be either private or public, by default it is private.
- This visibility mode specifies how the features of base class are visible to the derived class.

Public Inheritance



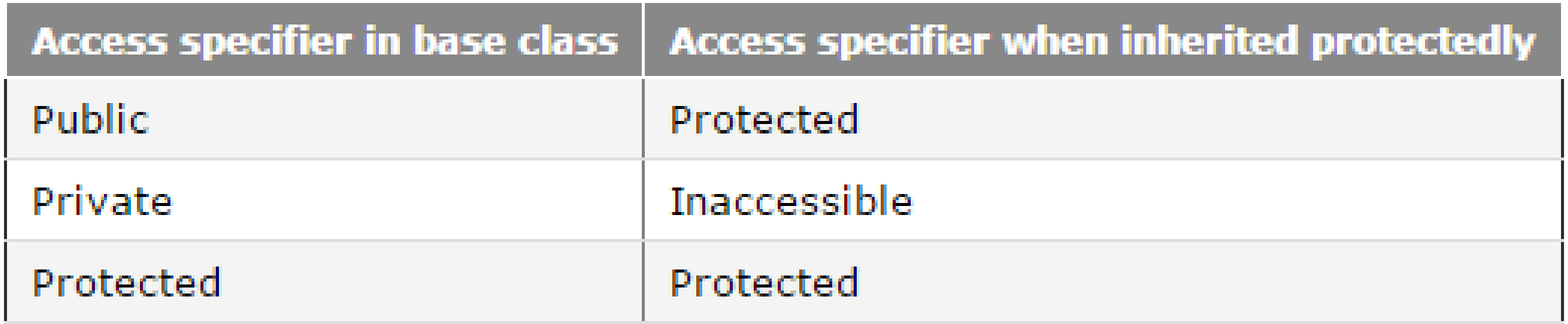
Access specifier in base class	Access specifier when inherited publicly
Public	Public
Private	Inaccessible
Protected	Protected

Private Inheritance



Access specifier in base class	Access specifier when inherited privately
Public	Private
Private	Inaccessible
Protected	Private

Protected Inheritance

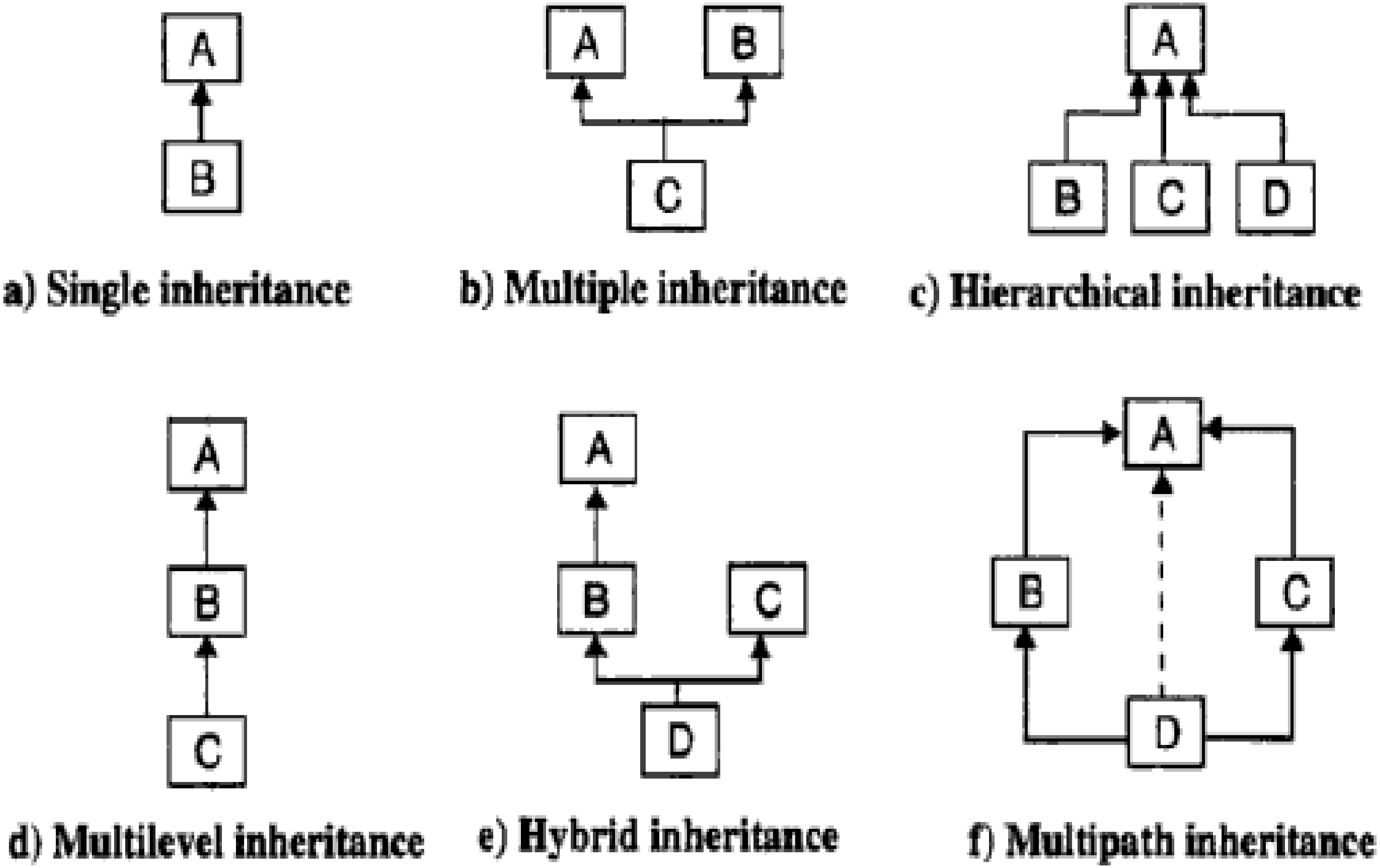


Access specifier in base class	Access specifier when inherited protectedly
Public	Protected
Private	Inaccessible
Protected	Protected

Inheritance

Types of Inheritance

- Inheritance are of following types
    - Simple or Single Inheritance
    - Multi level or Varied Inheritance
    - Multiple Inheritance
    - Hierarchical Inheritance
    - Hybrid Inheritance
    - Virtual Inheritance



a) Single inheritance

b) Multiple inheritance

c) Hierarchical inheritance

d) Multilevel inheritance

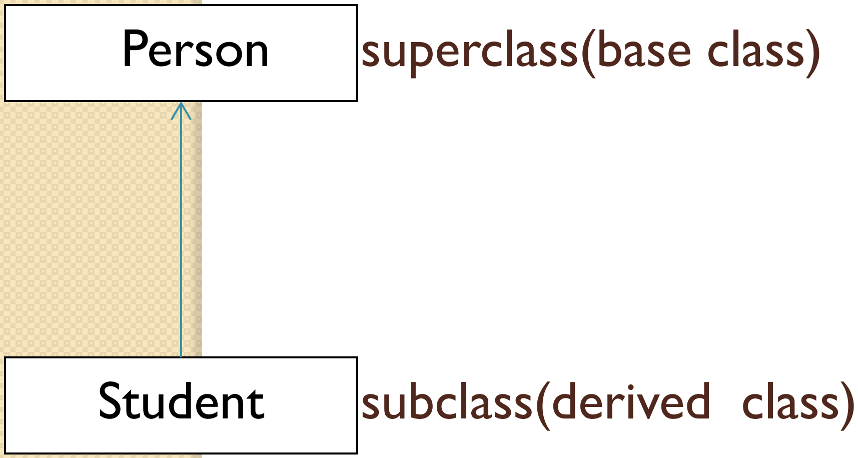
e) Hybrid inheritance

f) Multipath inheritance

Figure 14.6: Forms of Inheritance

Simple or Single Inheritance

- This is a process in which a sub class is derived from only one superclass.
- a Class Student is derived from a Class Person



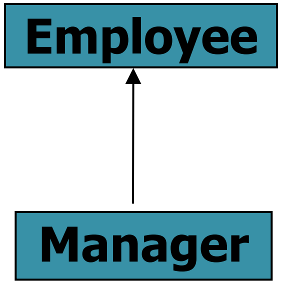
Person superclass(base class)

Student subclass(derived class)

```
class  Person

            {  .....    };

            class Student :  public  Person

            {
                ...........
            };
```

visibility mode

Single Inheritance



```
const int size= 50;
class Employee {
char ename[size];
unsigned long enumber;
public:
void getdata()     {
cout<<" Enter name :";
cin>>ename;
cout<<" Enter Employee no :";
cin>>enumber;     }
void putdata() {
cout<<"Name:"<<ename;
cout<<"EmployeeNumber:“
<<enumber; }
};
```

```
class Manager : public Employee
{
char branch[size];
unsigned long div_code;
public:
void getdata()   {
Employee::getdata();
cout<<"Enter Branch Name :";
cin>>branch;
cout<<"Enter Division Code";
cin>>div_code;   }
void putdata()   {
Employee::putdata();
cout<<Branch Name :"<<branch;
cout<<"DivisionCode<<div_code;}
};
```

```
void main() {
Employee e1; Manager m1;
cout<<"Enter employee  data:";
e1.getdata();
cout<<"Enter Manager  data:";
m1.getdata();
cout<<"\n Employee data:";
e1.putdata();
cout<<"\n Manager data: \n";
m1.putdata();
}
```

```
Enter employee data:
Enter name :Raja
Enter Employee no :1000
Enter Manager data:
Enter name :Vikas
Enter Employee no :2000
Enter Branch Name :Bangalore
Enter Division Code :1256
Employee data:
Name:  Raja
Employee Number:1000
Manager data:
Name: Vikas
Employee Number:2000
Branch Name :Bangalore
Division Code :1256
```

```
class counter
{
 protected:
  unsigned int count;
 public:
  counter() {count=0;}
  counter(int c) {count=c;}
  int show() {return count;}
  counter operator++()
  {
   count++;
   return counter(count);
  }
};
```

```
class counterD: public counter
{
 public:
  counter operator--()
  {
   count--;
   return counter(count);
  }
};

int main()
{
 counterD c1;
 cout<<"\nc1=" << c1.show();
 ++c1;++c1;++c1;
 cout<<"\nc1=" << c1.show();
 --c1;
 --c1;
 cout<<"\nc1="<<c1.show();
 getch();
 return(0);
}
```

Single inheritance: public

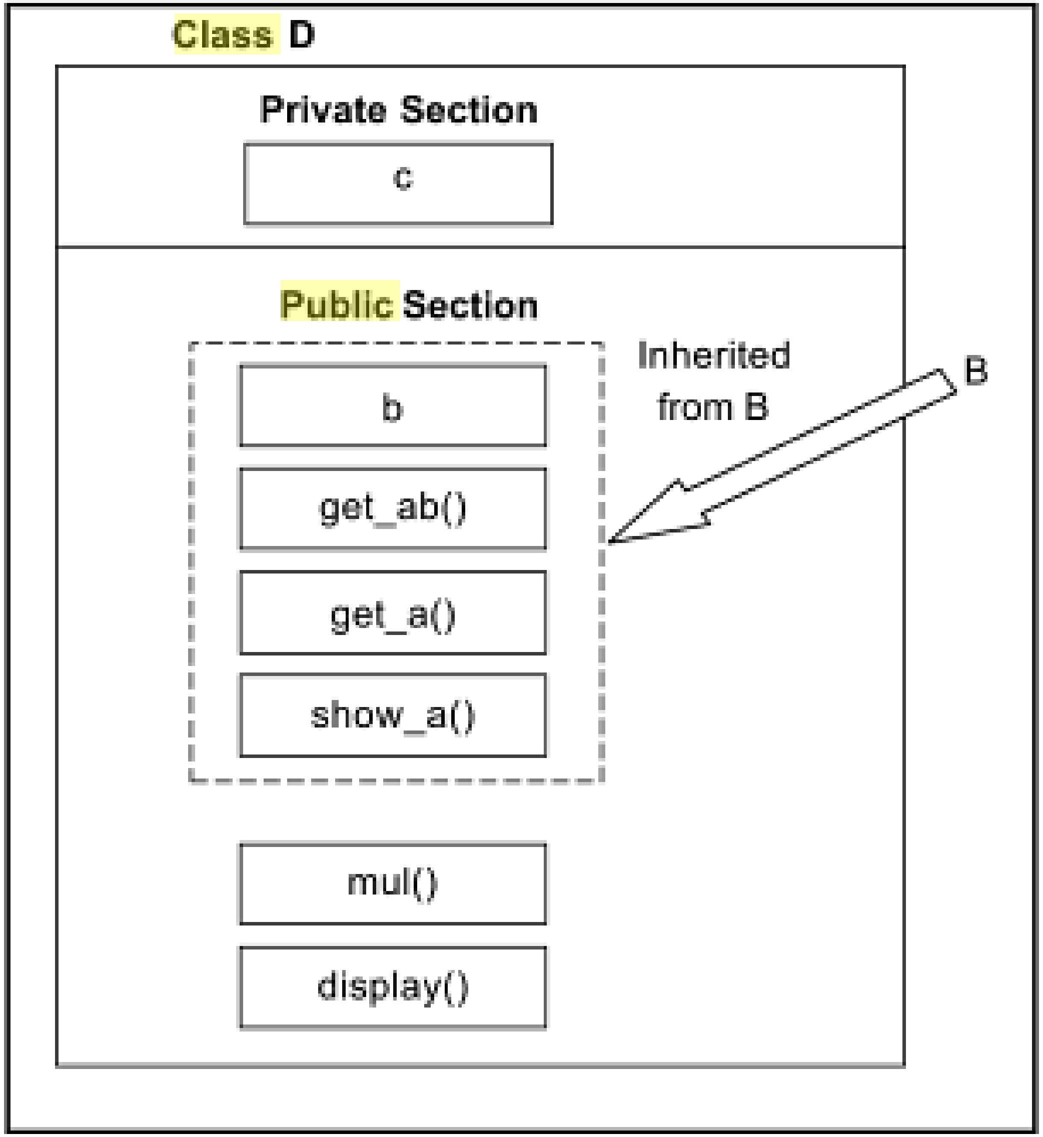
```
#include<iostream.h>
using namespace std;
class B
{
int a;
public:
int b;
void set_ab();
int get_a(void);
void show_a(void);
};
```

```
class D:public B
{
int c;
public:
void mul(void);
void display(void);
};
void B::set_ab(void)
{
a=5;
b=10;
}
```

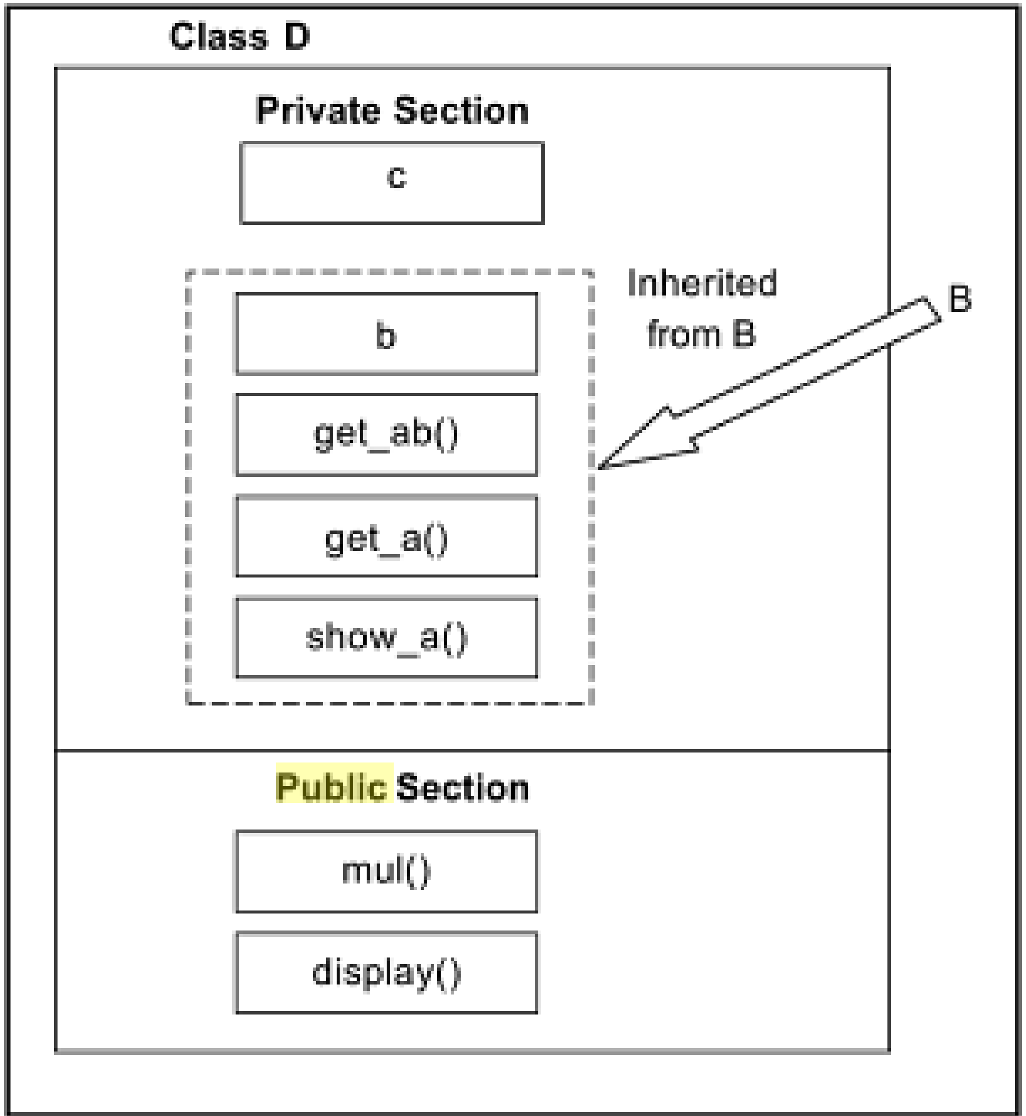
```
int B::get_a()
{
return a;
}
void B::show_a()
{
cout<<"a="<<a<<"\n";
}
void D::mul()
{
c=b*get_a();
}
void D::display()
{
cout<<"a="<<get_a()<<"\n";
cout<<"b="<<b<<"\n";
cout<<"c="<<c<<"\n";
}
```

```
int main()
{
D d;
d.set_ab();
d.mul();
d.show_a();
d.display();
d.b=20;
d.mul();
d.display();
return 0;
}
```

Adding more members to a class(by public derivation)



Adding more members to a class(by private derivation)



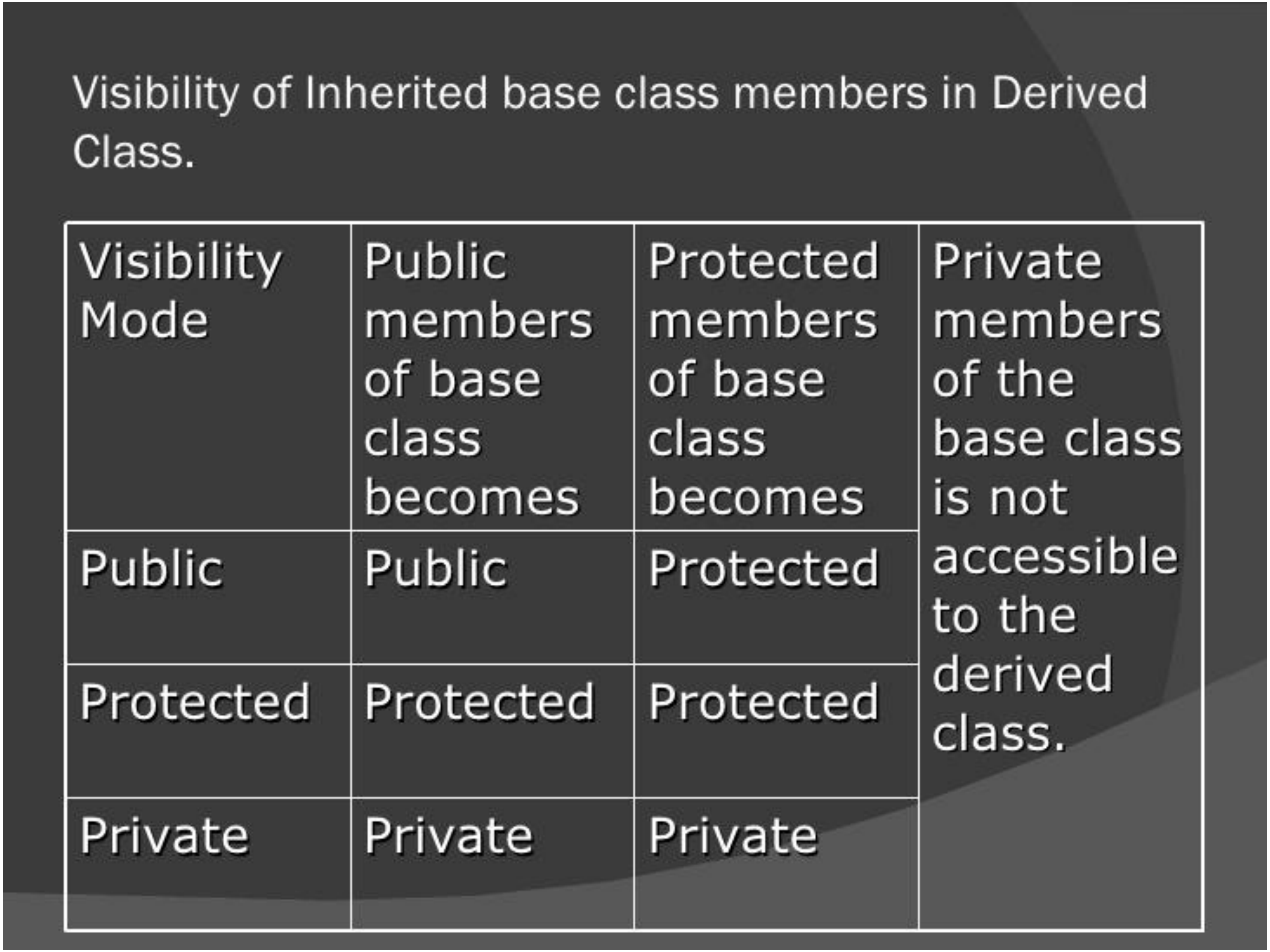
Single inheritance: private

```
#include<iostream.h>
using namespace std;
class B
{
int a;
public:
int b;
void get_ab();
int get_a(void);
void show_a(void);
};
```

```
class D:private B
{
int c;
public:
void mul(void);
void display(void);
};
void B::get_ab(void)
{
cout<<"entera and b value:";
cin>>a>>b;
}
```

```
int B::get_a()
{
return a;
}
void B::show_a()
{
cout<<"a="<<a<<"\n";
}
void D::mul()
{
get_ab();
c=b*get_a();
}
void D::display()
{
```

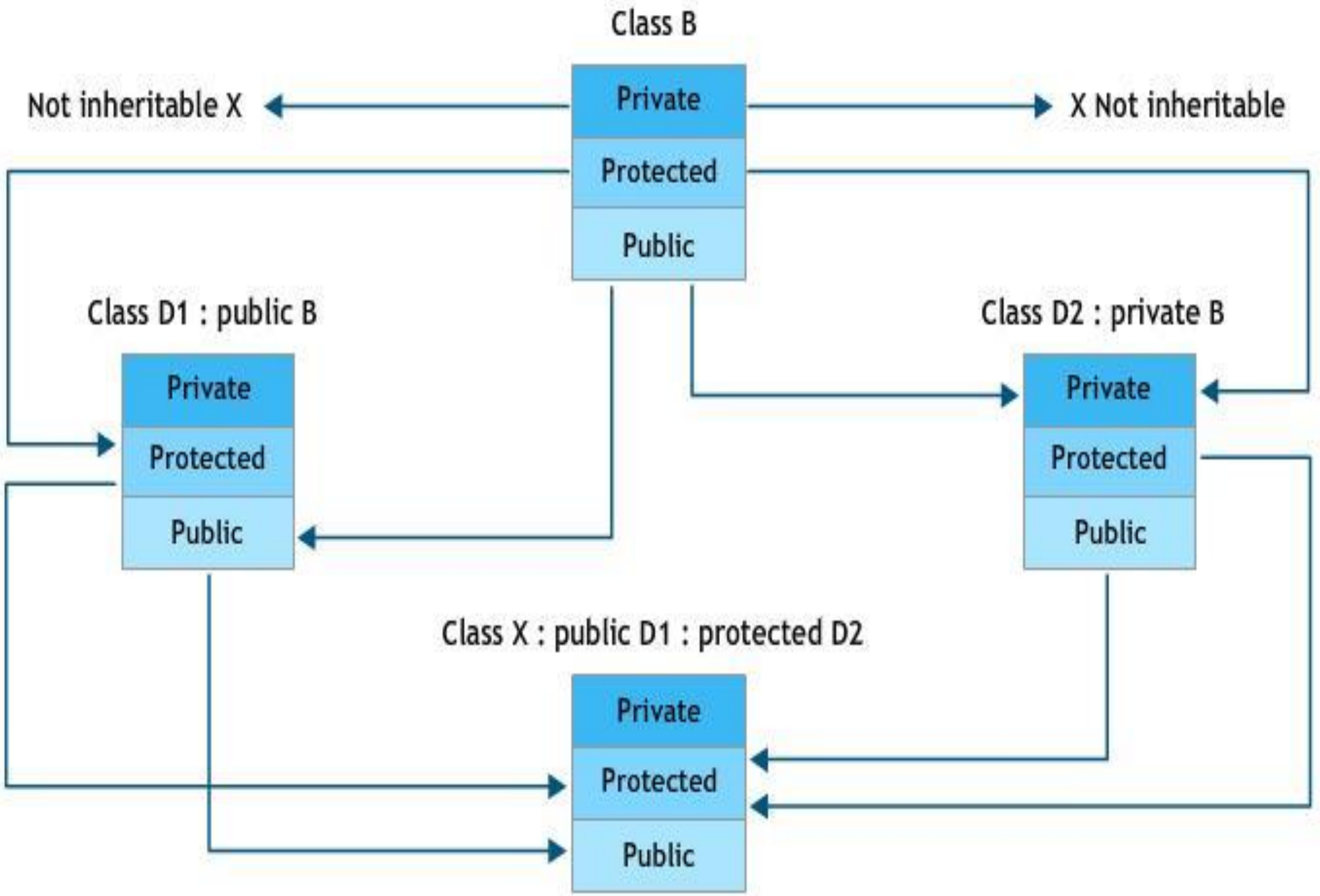
```
show_a();

cout<<"b="<<b<<"\n";
cout<<"c="<<c<<"\n";
}
int main()
{
D d;
//d.get_ab();    won’t work
d.mul();
//d.show_a();    won’t work
d.display();
//d.b=20;        won’t work
d.mul();
d.display();
return 0;
}
```



Visibility of Inherited base class members in Derived Class.

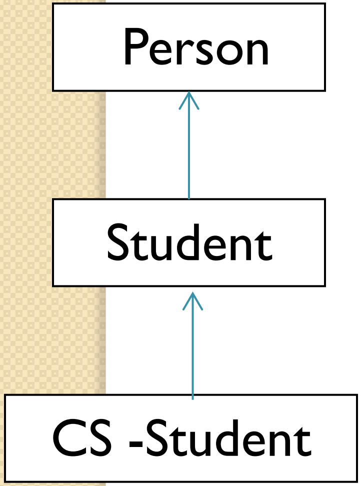
Visibility Mode	Public members of base class becomes	Protected members of base class becomes	Private members of the base class is not accessible to the derived class.
Public	Public	Protected	
Protected	Protected	Protected	
Private	Private	Private	

Effect of inheritance on the visibility of members



Multilevel Inheritance

- The method of deriving a class from another derived class is known as Multilevel Inheritance.
- A derived class CS-Student is derived from another derived class Student. EXAMPLE



```
Class Person
            { ……};
Class Student : public Person
            { ……};
Class CS -Student : public Student
            {  …….};
```

Multilevel inheritance

```
#include <iostream.h>
#include<conio.h>
using namespace std;
class student
 {
   protected:
     int rollno;
   public:
     void get_num(int a)
       { rollno = a; }
     void put_num()
       {
                    cout << "Roll
   Number Is:\n"<< rollno <<
   "\n"; }
 };
```

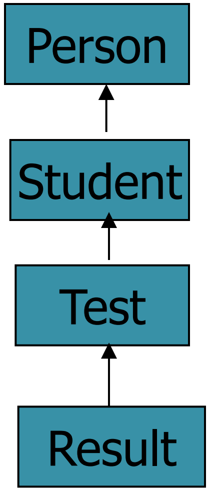
```
class marks : public student
 {
   protected:
     int sub1;
     int sub2;
   public:
     void get_marks(int x,int y)
       {
          sub1 = x;
          sub2 = y;
       }
     void put_marks(void)
       {
         cout << "Subject 1:" <<
   sub1 << "\n";
         cout << "Subject 2:" <<
   sub2 << "\n";
       }   };
```

```
class res : public marks
  {
    protected:
      float tot;
    public:
      void disp(void)
        {
          tot = sub1+sub2;
          put_num();
          put_marks();
          cout << "Total:"<< tot;
        }
  };
```

```
main()
  {
    res std1;
    std1.get_num(5);

   std1.get_marks(10,20);
    std1.disp();
    getch();
 }
```

Multilevel Inheritance

```
class person {
protected:
char name[20],sex; int age;
public:
void readdata() {
cin>>name>>sex>>age;
}
void showdata() {
cout<<" name is "<<name;
cout<<" sex is "<<sex;
cout<<" age is "<<age;
} };
```



Person

Student

Test

Result

```
class student:protected person
{
protected: int rollno;
char branch[20];
public:
void readdata() {
person::readdata();
cin>>rollno>>branch; }
void showdata() {
person::showdata();
cout<<" rollno:"<<rollno;
cout<<" branch :"<<branch;
} };
```

```
class test:protected student {
protected:
int mark1,mark2,mark3;
public:
void readdata() {
student::readdata();
cin>>mark1>>mark2>>mark3;
}
void showdata() {
student::showdata();
cout<<"mark1 is :"<<mark1;
cout<<"mark2 is :"<<mark2;
cout<<"mark3 is :"<<mark3;
}};
```

```
class result:protected test
{
protected: int total;
public:
void processmark() {
test::readdata();
total=mark1+mark2+mark3;
test::showdata();
cout<<"\n total is "<<total;
}
};
```

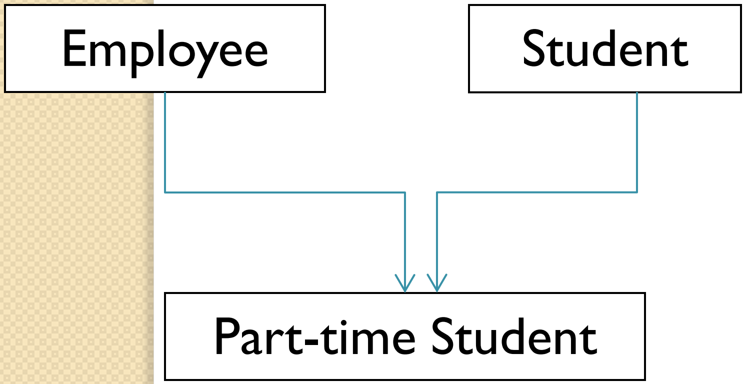
```
void main() {
Result  r1;
   r1.processmark();
}
```

```
enter name :raja
enter sex :m
enter age :21
enter rollno :25
enter branch :mca
enter sub1 mark :50
enter sub2 mark :60
enter sub3 mark :70
```

```
name is raja
sex is m
age is 21
rollno:25
branch :mca
mark1 is :50
mark2 is :60
mark3 is :70
total is 180
```

Multiple Inheritance

- A class is inheriting features from more than one super class
- Class Part-time Student is derived from two base classes, Employee and Student
- example



```
Class Employee
            {……..};
Class Student
            {……..};
Class Part-time Student : public Employee,
                          public Student
            {…….};
```

Multiple Inheritance

```
using namespace std;
class Area
{
  public:
    float area_calc(float l,float b)
    {
        return l*b;
    }
};
```

```
class Perimeter
{
    public:
    float peri_calc(float l,float b)
    {
        return 2*(l+b);
    }
};
```

```
/* Rectangle class is derived from classes
     Area and Perimeter. */
class Rectangle : private Area, private
     Perimeter
{
    private:
        float length, breadth;
    public:

       void get_data( )
       {
           cout<<"Enter length: ";
           cin>>length;
           cout<<"Enter breadth: ";
           cin>>breadth;
       }

       float area_calc()
       {
       /* Calls area_calc() of class Area and
      returns it. */

           return
      Area::area_calc(length,breadth);
       }
```

```
float peri_calc()
      {
      /* Calls peri_calc() function of class
    Perimeter and returns it. */

          return
    Perimeter::peri_calc(length,breadth);
      }};
 main()
{
   Rectangle r;
   r.get_data();
   cout<<"Area = "<<r.area_calc();
   cout<<"\nPerimeter = "<<r.peri_calc();
   //return 0;
   getch();
}
```

Multiple Inheritance

```
class liquid
{
    float specific_gravity;
    public:
    void input()
    {
    cout<<"Specific gravity: ";
    cin>>specific_gravity;
    }
    void output()
    {
    cout<<"Specific gravity:“
<<specific_gravity<<endl;
        }
};
```

```
class fuel
{
     float rate;
     public:
     void input()
    {
     cout<<"Rate(per liter): $";
     cin>>rate;
    }
    void output()
    {
     cout<<"Rate(per liter): $“
<<rate<<endl;
         }
};
```

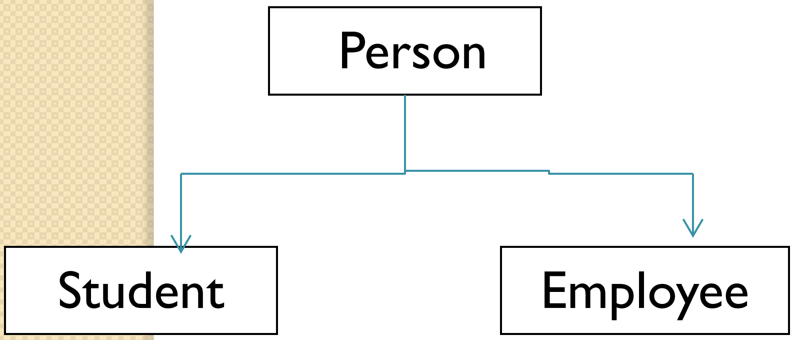
```
class petrol: public liquid, public fuel
{
    public:
        void input()
        {
            liquid::input();
            fuel::input();
        }
        void output()
        {
            liquid::output();
            fuel::output();
        }
};
```

```
int main()
{
    petrol p;
    cout<<"Enter data"<<endl;
    p.input();
    cout<<endl<<"Displaying
data"<<endl;
    p.output();
    getch();
    return 0;
}
```

```
Enter data
Specific gravity: 0.7
Rate(per liter): $0.99

Displaying data
Specific gravity: 0.7
Rate(per liter): $0.99
```

Hierarchical Inheritance

- Many sub classes are derived from a single base class
- The two derived classes namely Student and Employee are derived from a base class Person. Example



```
Class Person
            {……};
Class Student : public  Person
            {……};
Class Employee : public  Person
            {……};
```

```
class Shape
{
protected:
    float width, height;
public:
void set_data(float a,float b)
{
    width = a;
    height = b;
}
};
```

```
class Triangle: public Shape
{
public:
    float area ()
    {
        return (width *
height / 2);
    }
};
```

```
class Rectangle:public Shape
{
public:
    float area ()
    {
        return (width *
height);
    } };
```

```
int main ()
{
   Rectangle rect;
   Triangle tri;
   rect.set_data (5,3);
   tri.set_data (2,5);
   cout << rect.area() << endl;
   cout << tri.area() << endl;
   return 0;
}
```

output :

```
15
5
```

```
class Side
 {
  protected:
    int l;
  public:
    void set_values (int x)
      { l=x;}
 };
```

```
class Square: public Side
  {
    public:
      int sq()
      {
          return (l * l);
      }
  };
```

```
class Cube:public Side
  {
   public:
     int cub()
      {
              return (l * l * l);
       }   };
```

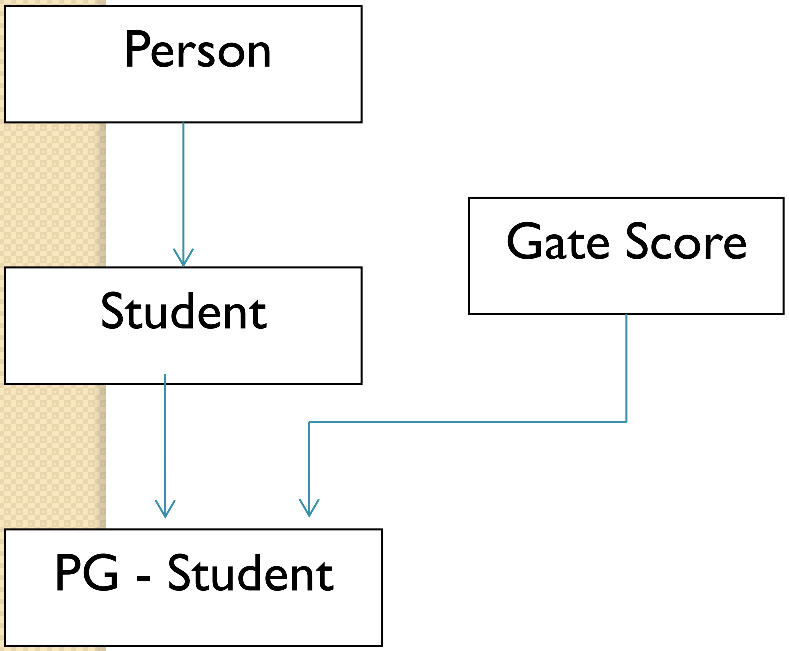
```
main ()
 {
   Square s;
   s.set_values (10);
   cout << "The square value is::" << s.sq();
   Cube c;
   c.set_values (20);
   cout << "The cube value is::" << c.cub();
  }
```

```
class student
{
 public:
  int rno , m1 , m2 ;
  void get()
  {
   rno = 15, m1 = 10, m2 = 10;
  }
};
class sports
{
 public:
  int sm;
  void getsm()
  {
    sm = 10;
  }
};
```

```
class statement:public
student,public sports
{
 int tot,avg;
 public:
  void display()
  {
   tot = (m1 + m2 + sm);
   avg = tot / 3;
   cout << tot;
   cout << avg;
  }
 };
    int main()
    {
        statement obj;
        obj.get();
        obj.getsm();
        obj.display();
    }
```

Hybrid Inheritance

- In this type, more than one type of inheritance are used to derive a new sub class
- Multiple and multilevel type of inheritances are used to derive a class PG-Student Example



```
Class Person
            { ……};
Class Student : public Person
            { ……};
Class Gate Score
            {…….};
Class PG - Student : public Student
                             public Gate Score
            {……..};
```

```
#include <iostream.h>
#include<conio.h>
using namespace std;
class mm
 {
   protected:
     int rollno;
   public:
     void get_num(int a)
      { rollno = a; }
     void put_num()
      { cout << "Roll
  Number Is:"<< rollno
  << "\n"; }
 };
```

```
class marks : public mm
 {
  protected:
    int sub1;
    int sub2;
  public:
    void get_marks(int x,int y)
     {
       sub1 = x;
       sub2 = y;
     }
    void put_marks(void)
     {
        cout << "Subject 1:" <<
   sub1 << "\n";
        cout << "Subject 2:" <<
   sub2 << "\n";
     } };
```

```
class extra
  {
    protected:
      float e;
    public:
    void get_extra(float s)
      {e=s;}
    void put_extra(void)
      { cout << "Extra Score::" << e <<
     "\n";}
  };
class res : public marks, public extra{
   protected:
     float tot;
 public:
     void disp(void)
       {  tot = sub1+sub2+e;
         put_num();
         put_marks();
         put_extra();
         cout << "Total:"<< tot;
       } };
```

```
main()
 {
   res std1;
   std1.get_num(10);
   std1.get_marks(10,20);
   std1.get_extra(33.12);
   std1.disp();
   getch();
//   return 0;
 }
```

Ambiguity Resolution in Inheritance – Single Level

```
class a {
int x;
public:
void get_x(int x1) {x=x1;}
void show() {
cout<<“\n x="<<x;}
};
class b:public a {
int y;
public :
void get_y(int y1) {y=y1;}
void show() {
cout<<“\n y="<<y;}
};
```

```
void main() {
b b1;
b1.get_x(100);
b1.get_y(200);
b1.show();
b1.show();
}
```

```
y=200
y=200
```

```
class a {
int x;
public:
void get_x(int x1) {x=x1;}
void show() {
cout<<“\n x="<<x;}
};
class b:public a {
int y;
public :
void get_y(int y1) {y=y1;}
void show() {
a::show();
cout<<“\n y="<<y;}
};
```

```
void main() {
b b1;
b1.get_x(100);
b1.get_y(200);
b1.show();
}
```

```
x=100
y=200
```

```
class a {
int x;
public:
void get_x(int x1) {x=x1;}
void show() {
cout<<“\n x="<<x;}
};
class b:public a {
int y;
public :
void get_y(int y1) {y=y1;}
void show() {
cout<<“\n y="<<y;}
};
```

```
void main() {
b b1;
b1.get_x(100);
b1.get_y(200);
b1.a::show();
b1.b::show();
}
```

```
x=100
y=200
```