

CS-112

Object Oriented Programming (3+1)

Prerequisites:

Programming Fundamentals

SYED MUHAMMAD RAFI

LECTURER, DEPARTMENT OF SOFTWARE ENGINEERING
FACULTY OF ENGINEERING SCIENCE AND TECHNOLOGY,
ZIAUDDIN UNIVERSITY

Inheritance

Lec-8

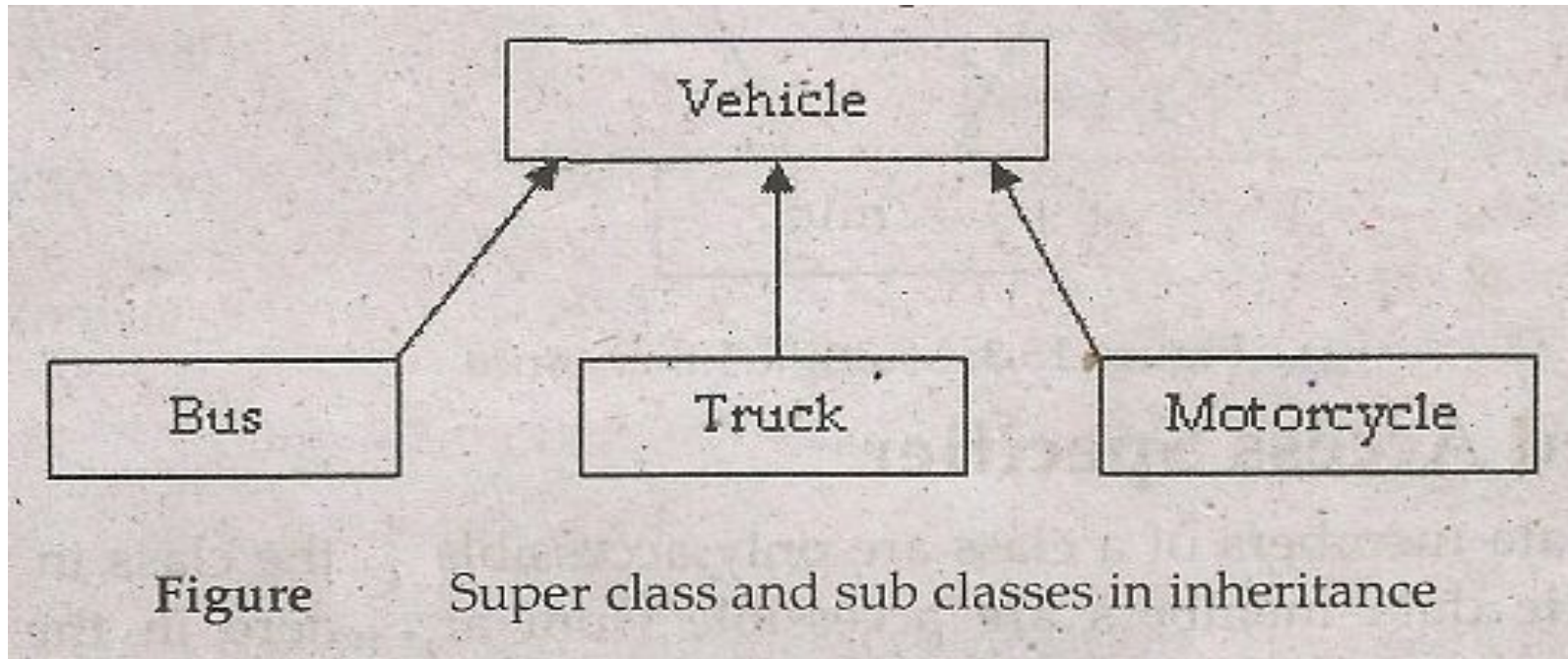
Inheritance

- The programming technique that is used to reuse an existing class to build a new class is known as **inheritance**.
- The new class inherits all the behaviors of the original class.
- The existing class that is reused to create a new class is known as **super class, base class or parent class**.
- The new class that inherit the properties and functions of an existing class is known as **sub class, derived class or child class**.
- The inheritance relation between the classes of a program is called a class **hierarchy**.

- Inheritance is one of the most powerful feature of object oriented programming.
- The basic principle of inheritance is that each sub class shares common properties with the class from which it is derived.
- The child class inherits all the properties of parents class and can add its own capabilities.

Example

- Suppose we have a class **vehicle**.
- The subclass of this class may share similar properties such as **wheels** and **motors** etc.
- Additionally a sub class may have its own particular characteristics. For example a sub class **Bus** may have seats for people but another sub class **Truck** may have space to carry goods.
- A class of animals can be divided into sub classes like mammals, amphibians, insects, reptiles.
- A class of vehicle can be divided into cars, trucks, buses and motorcycles.
- A class of shapes can be divided into the subclass lines, ellipse, boxes .

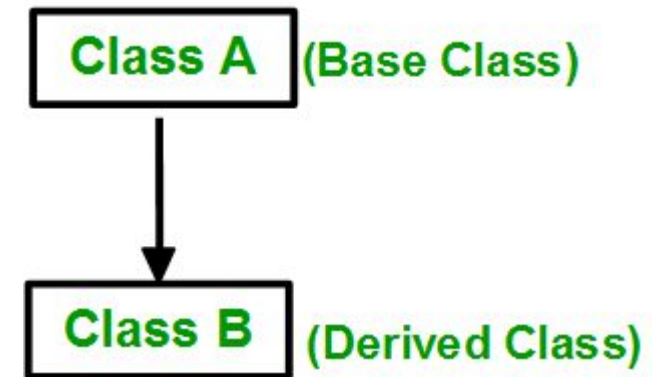
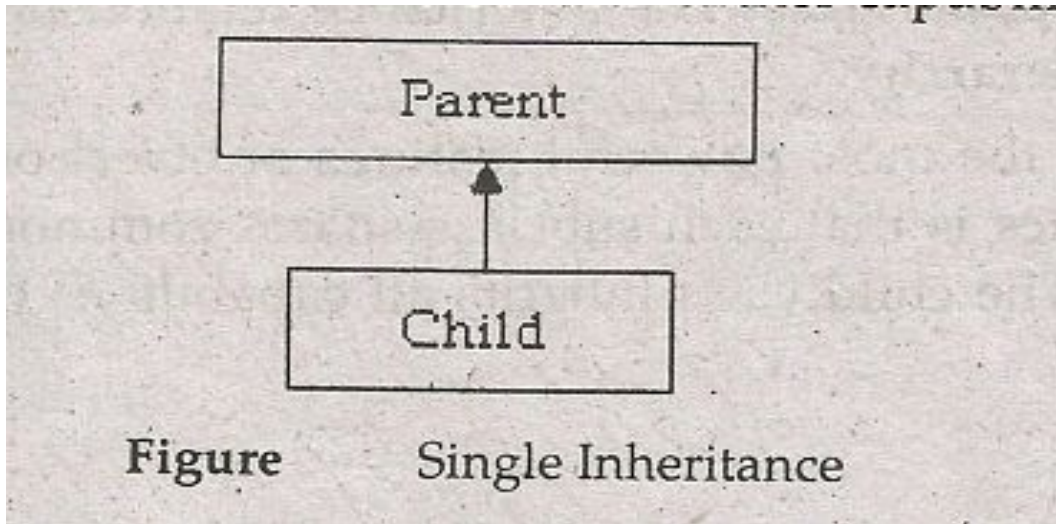


- The above figure shows that **vehicle** is a parent class and **Bus**, **Truck** and **Motorcycle** are three sub classes.

Categories of Inheritance

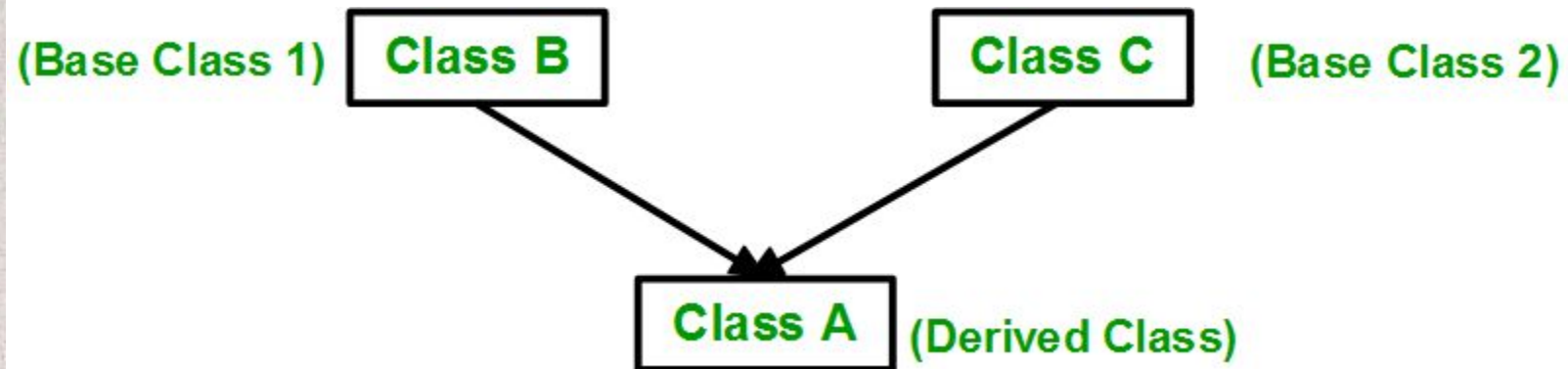
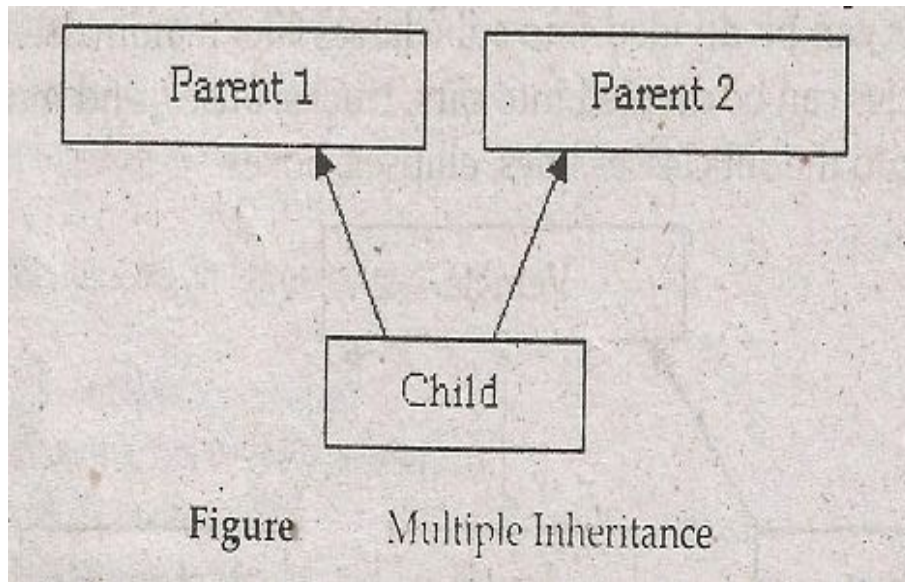
1. Single inheritance

- The type of inheritance in which a child is derived from a single parent class is known as **single inheritance**.
- The child class in this inheritance inherits all data members and member functions of the parent class.



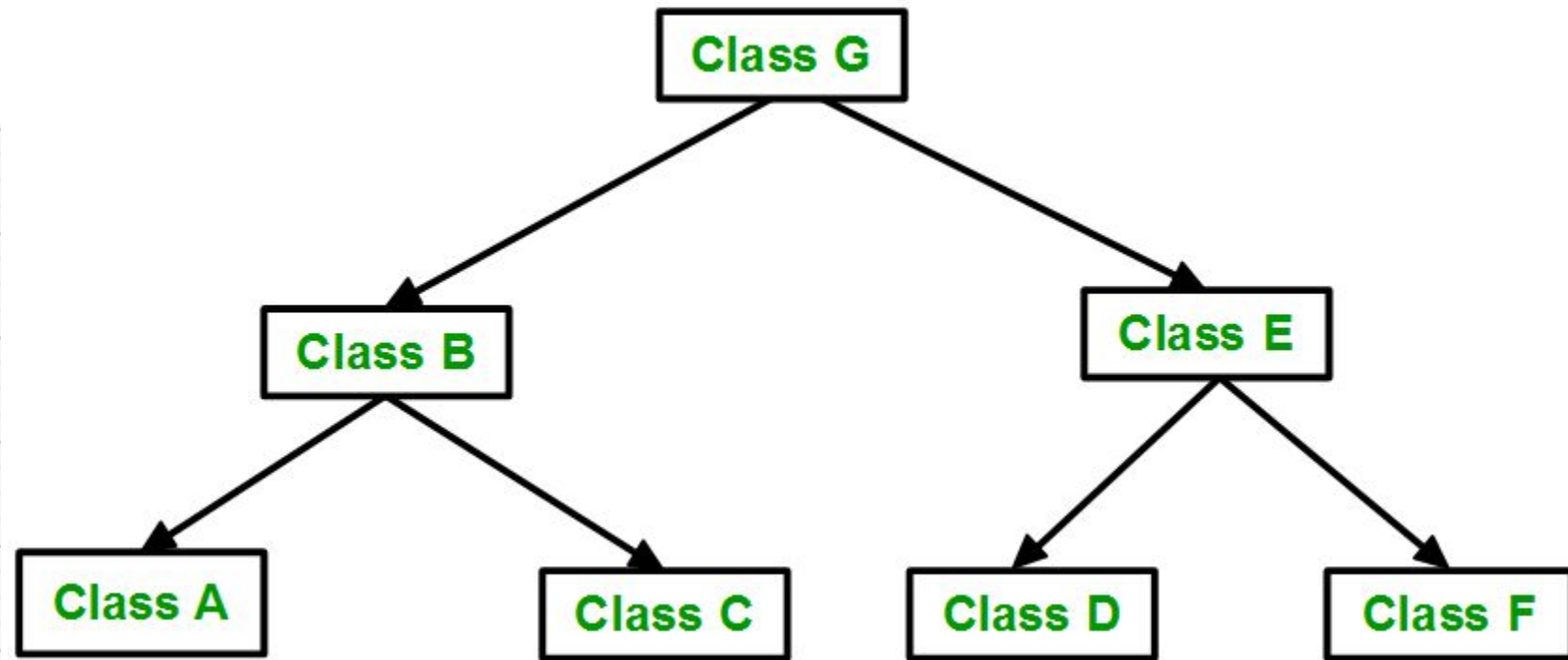
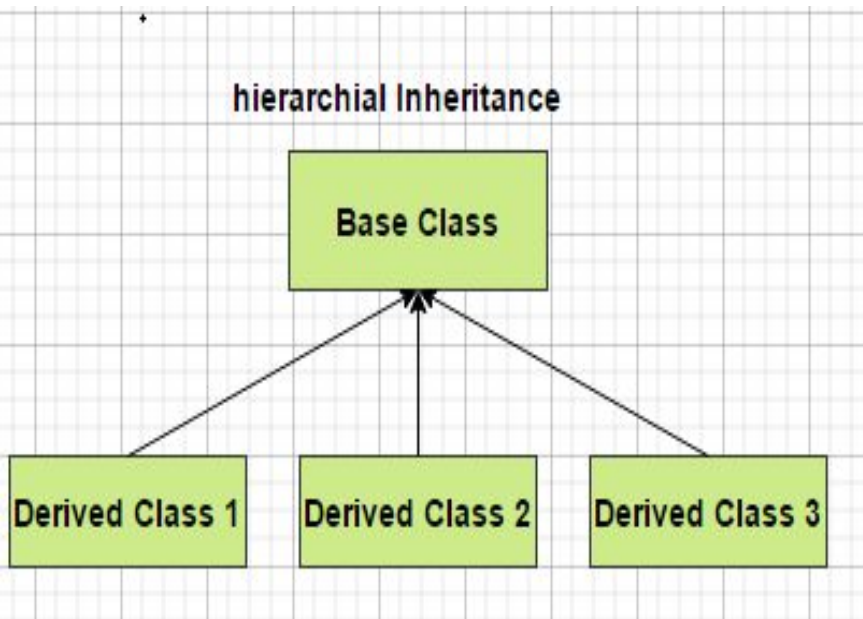
2. Multiple inheritance

- A type of inheritance in which a child class is derived from multiple parent class is known as **multiple inheritance**.
- The child class in this inheritance inherits all data members and member functions of all parent classes.
- It can add further capabilities of its own.



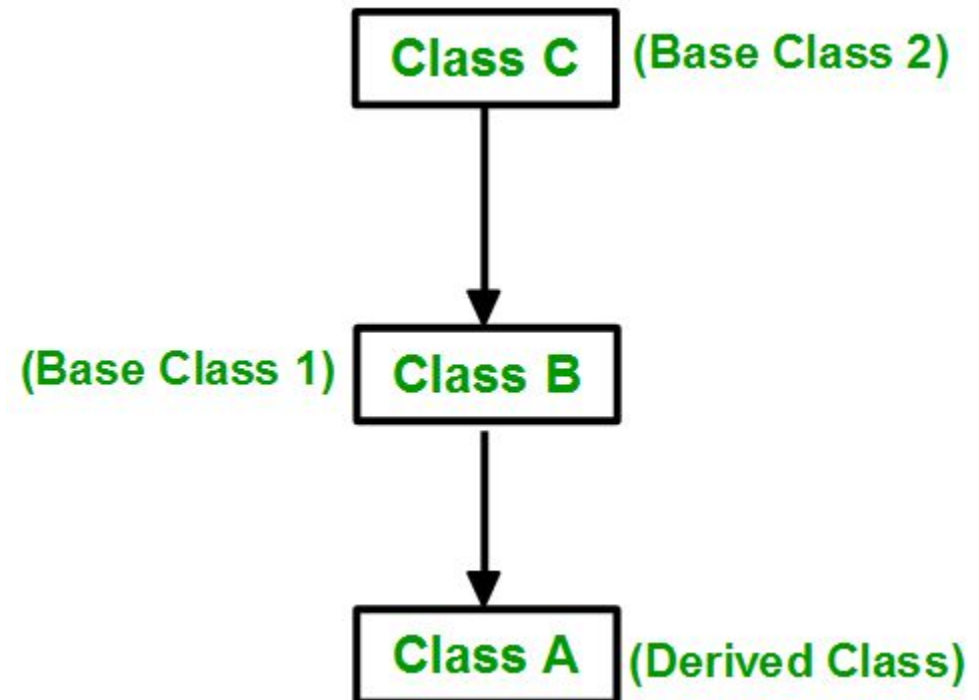
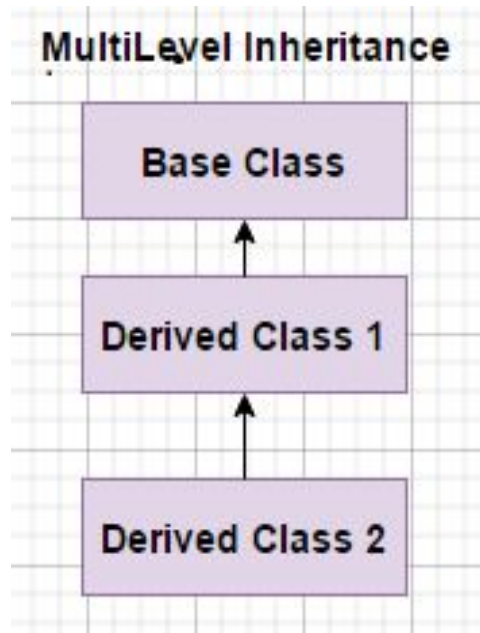
3. Hierarchical Inheritance

- In this type of inheritance, more than one sub class/child class is inherited from a single base class/ parent class.
- More than one derived class is created from a single base class.



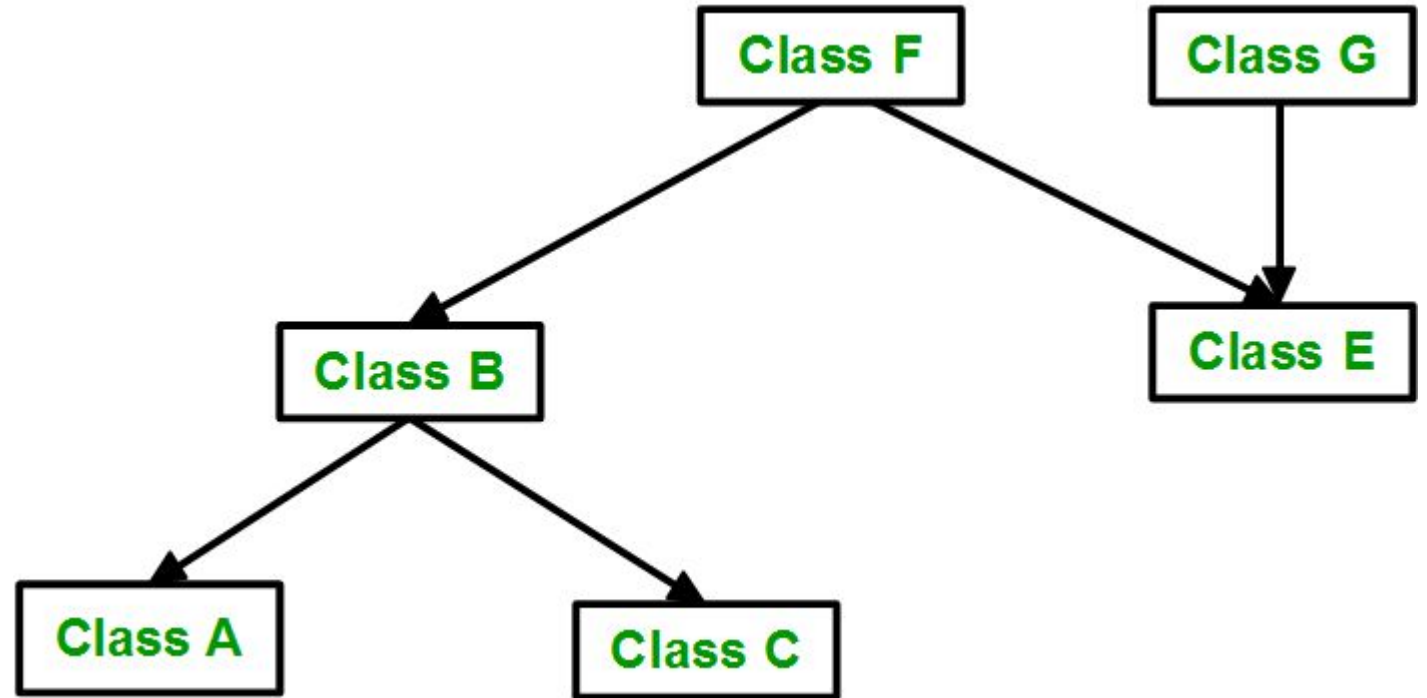
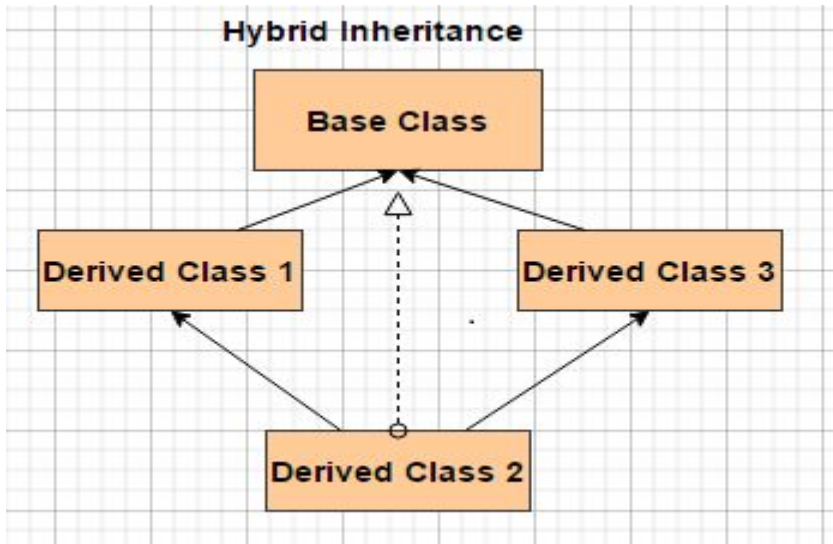
4. Multilevel Inheritance

- In this type of inheritance the derived class inherits from a class, which in turn inherits from some other class.
- The Super class for one, is sub class for the other.



5. Hybrid (Virtual) Inheritance

- Hybrid Inheritance is implemented by combining more than one type of inheritance.
- Hybrid Inheritance is combination of Hierarchical and Multilevel Inheritance.



Protected Access specifier

- The **private** data member of a class are only accessible in the class in which they are declared.
- The **public** data members are accessible from any where in the program.
- The **protected** access specifier is different from **private** and **public** access specifiers. It is specially used in inheritance.
- It allows **protected** data members to accessed from all derived classes.
- It means that child class can access all protected data members of parent class.

The different between private, public, protected access specifiers is as follows.

Access Specifier	Accessible from own class	Accessible from derived class	Accessible from objects outside class
public	Yes	Yes	Yes
protected	Yes	Yes	No
private	Yes	No	No

Table Difference between different access specifiers

Specifying a Derived Class

- The syntax of specifying a derived class is as follows

```
class sub_class :specifier parent_class
```

```
{
```

Body of the class

```
};
```

Where,

Class	It is keyword that is used to declare class
sub_class	It is name of derived class
:	It creates relationship between derived class and super class.
specifier	It indicates the type of inheritance.it can be private, protected protected, public
parent_class	It indicates the name of parent class that is being inherited.

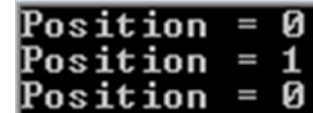
Example

```
#include <iostream>
using namespace std;
class Move
{
protected:
    int position;
public:
    Move()
    {
        position=0;
    }
    void forward()
    {
        position++;
    }
}
```

```
void show()
{
    cout<<"Position = "<<
position<<endl;
}
};

class Move2: public Move
{
public:
    void backward()
    {
        position--;
    }
};
```

```
int main()
{
    Move2 m;
    m.show();
    m.forward();
    m.show();
    m.backward();
    m.show();
}
```



```
Position = 0
Position = 1
Position = 0
```


Move

protected:

int position;

public:

forward();

show();

Move2

public:

backward();

Output:

Position = 0

Position = 1

Position = 0

How above program works

- The above program declares two classes **Move** and **Move 2**.
- The class **Move** is parent class with data member **position** that is declared **protected** so that it may be accessed in derived classes also.
- The class **Move 2** derives **Move** and also declares a member function **backward()** that decrements the value of **position**.
- The program declares an object **m** of type **Move 2** that is derived class.
- The object contains one data member **position** and three member functions **show()**, **forward()** and **backward()**.
- The data member and two member functions are derived from parent class and only one member function **backward()** is declared in the derived class.

Accessing Members of Parent Class

- An important issue in the inheritance is the accessibility of base class members by the objects of derived class.
- It is known as **accessibility**.
- The objects of derived class can access certain members of parent class.

1. Accessing constructor of Parent Class

- The objects of derived class can access certain constructors of parent class.
- It will use an appropriate constructor from parent class if no constructor is declared in the derived class.
- The program in previous example declares an object **m** of derived class. There is no constructor in the derived class. The compiler automatically use the constructor of parent class.

- The objects of derived class can automatically access the constructor of parent class and parent has a constructor with no parameter.
- The derived class can also access constructors of parent class with parameter by passing values to them.

Syntax

The syntax of accessing the constructor of parent class in derived class is as follows:

```
Child_con : parent_con(parameters)
{
    Body of constructor
}
```

Where,

Child_con It is the name of constructor of derived class.

parent_con It is the name of constructor of parent class.

parameter It is the list of parameter passed to constructor of parent class.

EXAMPLE

```
#include <iostream>
using namespace std;
class parent
{
protected:
    int n;
public:
    parent()
    {
        n=0;
    }
    parent(int p)
    {
        n=p;
    }
}
```

```
void show()
{
    cout<<"n="<<n<<endl;
}

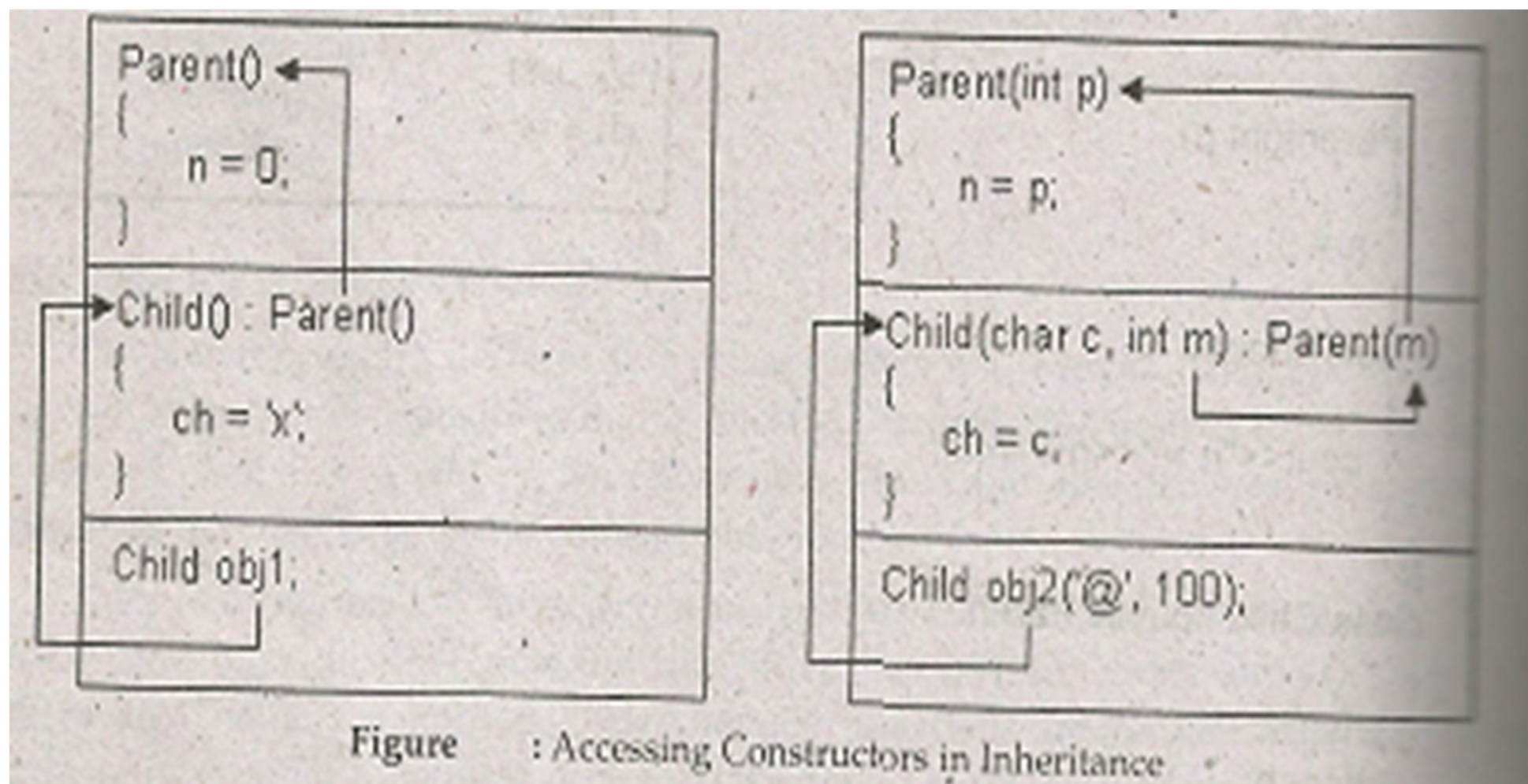
class child : public parent
{
private:
    char ch;
public:
    child() : parent()
    {
        ch='x';
    }
}
```

```
child(char c, int m):parent(m)
{
    ch=c;
}

void display()
{
    cout<<"ch="<<ch<<endl;
}

int main()
{
    child obj1,obj2('@',100);
    cout<<"obj 1 is as follows:\n";
    obj1.show();
    obj1.display();
    cout<<"obj 2 is as follows:\n";
    obj2.show();
    obj2.display();
}
```

```
obj 1 is as follows:  
n=0  
ch=x  
obj 2 is as follows:  
n=100  
ch=@
```



How Above Program Works

- The above program declares two classes **Parent** and **Child**.
- Both classes declare two constructor each.
- One constructor use no parameter and the second constructor takes parameter for data members.
- Each constructor in child class also calls the corresponding constructor of the parent class.
- The program declares two objects of child class.
- The **obj1** one uses the constructor with no parameters and the **obj2** uses the constructor with parameters.
- The **obj2** takes two parameters for the data members.
- The first parameter **@** is used for data member **ch** declared in child class. The second parameter **100** is for data members **n** declared in parent class.

2. Accessing Member Functions of Parent Class

- The objects of derived class can access all member functions of parent class that are declared as **protected** or **public**.

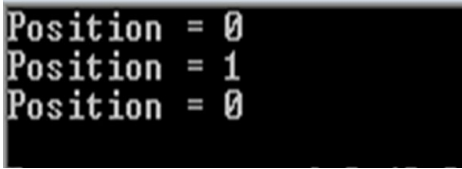
Example

```
#include <iostream>
using namespace std;
class Move
{
protected:
    int position;
public:
    Move()
    {
        position=0;
    }
    void forward()
    {
        position++;
    }
}
```

```
void show()
{
    cout<<"Position = "<<
position<<endl;
};

class Move2: public Move
{
public:
    void backward()
    {
        position--;
    }
};
```

```
int main()
{
    Move2 m;
    m.show();
    m.forward();
    m.show();
    m.backward();
    m.show();
}
```



```
Position = 0
Position = 1
Position = 0
```


Explanation

- The object m in the above program contains three member functions **show()**, **forward()** and **backward()**.
- The first two member functions are declared in parent class and only **backward()** is declared in derive class.

❑ Write a class person that has the attributes of id, name and address, it has a constructor to initialize, a member function to input and a member function to display data members. Create another class Student that inherits Person class. It has additional attributes of roll number and marks. It has a member function to input and display its data members.

```

#include<iostream>
using namespace std;
class Person
{
protected:
    int id;
    char name[50],address[100];
public:
    Person()
    {
        id=0;
        name[0]='\0';
        address[0]='\0';
    }

    void GetInfo()
    {
        cout<<"Enter you id: ";
        cin>>id;
        cout<<"Enter your name: ";
        cin>>name;
        cout<<"Enter your address: ";
        cin>>address;
    }

    void ShowInfo()
    {
        cout<<"\n Your personal
information is as follows:\n";
        cout<<"id="<<id<<endl;
        cout<<"Name="<<name<<endl;

        cout<<"Address="<<address<<endl;
    }
};

class Student : public Person
{
private:
    int rno, marks;
public:
    Student()
    {
        Person:Person();
        rno=marks=0;
    }

    void GetEdu()
    {
        cout<<"Enter your roll no:";
        cin>>rno;
        cout<<"Enter your marks:";
        cin>>marks;
    }

    void ShowEdu()
    {
        cout<<"\n Your educational informati
as follows:\n";
        cout<<"Roll No="<<rno<<endl;
        cout<<"Marks="<<marks<<endl;
    }
};

int main()
{
    Student s;
    s.GetInfo();
    s.GetEdu();
    s.ShowInfo();
    s.ShowEdu();
}

```

```

Enter you id: 4
Enter your name: ALI
Enter your address: KARACHI
Enter your roll no: 2
Enter your marks: 91

Your personal information is as follows:
id=4
Name=ALI
Address=KARACHI

Your educational information is as follows:
Roll No=2
Marks=91

```

How Above Program Works

- The above program declares two classes.
- The **Person** class is a parent class and **Students** class is a derived class.
- The **Person** class is designed to store and process the information of any person.
- The **Student** class is designed to store and process the information of a student.
- A student is also a person, so the **Student** class can inherit **Person** class.
- The program declare an object **s** of **Student** class.
- The object has five data members and four member functions.
- When the object calls **GetInfo()** member function, the compiler looks in derived class. The derived class contains no such function, so that compiler automatically uses the function from parent class.

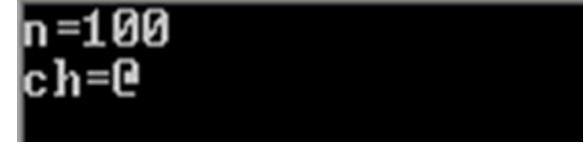
Function Overriding

- The process of declaring member function in derived class with same name and same signature as in parent class is known as **function overriding**.
- Function overriding allows the user to use same names for calling the member functions of different classes.
- The member function is overridden in the derived class, the object of derived class cannot access the function of parent class.
- However, the function of parent class can be accessed by using resolution operator

EXAMPLE

```
#include <iostream>
using namespace std;
class Parent
{
protected:
    int n;
public:
    Parent(int p)
    {
        n=p;
    }
    void show()
    {
        cout<<"n="<<n<<endl;
    }
};
class Child : public Parent
{
private:
    char ch;
```

```
public:
    Child(char c,int m) : Parent(m)
    {
        ch=c;
    }
    void show()
    {
        Parent::show();
        cout<<"ch="<<ch<<endl;
    }
};
int main()
{
    Child obj('@',100);
    obj.show();
}
```



```
n=100
ch=@
```

How Above Program Works

- The above program declares two classes.
- Both classes declare a member function **show()** to display the value of data member.
- The derived class overrides **show()** function.
- The object of derived class cannot access this function directly .
- The object calls the function declared in derived class.
- The object calls the function of parent class with the following statement :

Parent:: show();

□ Write a program that declares two classes. The Parent class is called **Simple** that has two members **a** and **b** to store two numbers. It also has four member functions:

- The **add()** function adds two number and displays the result .
- The **sub()** function subtracts two numbers and displays the result .
- The **mul()** function multiplies two numbers and displays the result.
- The **div()** function divides two numbers and displays the result.

The child class is called **Complex** that overrides all four functions. Each functions in the child class checks the values of data members. It calls the corresponding member function in the parent class if the values are greater than 0. Otherwise it displays error message.


```

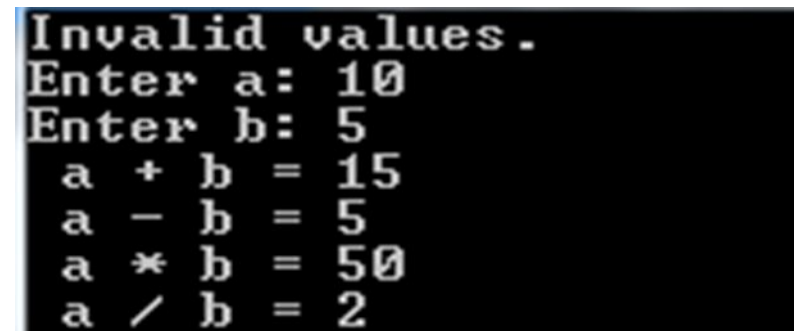
#include <iostream>
using namespace std;
class Simple
{
protected:
    int a,b;
public:
    Simple()
    {
        a=b=0;
    }
    void in()
    {
        cout<<"Enter a: ";
        cin>>a;
        cout<<"Enter b: ";
        cin>>b;
    }
};

void add()
{
    cout<<" a + b = "<<a+b<<endl;
}
void sub()
{
    cout<<" a - b = "<<a-b<<endl;
}
void mul()
{
    cout<<" a * b = "<<a*b<<endl;
}
void div()
{
    cout<<" a / b = "<<a/b<<endl;
}
};

class Complex : public Simple
{
public:
    void add()
    {
        if(a<=0 || b<=0)
            cout<<"Invalid values.
"<<endl;
        else
            Simple::add();
    }
    void mul()
    {
        if(a<=0 || b<=0)
            cout<<"Invalid values.
"<<endl;
        else
            Simple::mul();
    }
};

void div()
{
    if(a<=0 || b<=0)
        cout<<"Invalid values.
"<<endl;
    else
        Simple::div();
}
int main()
{
    Complex obj;
    obj.in();
    obj.add();
    obj.sub();
    obj.mul();
    obj.div();
}

```



```

Invalid values.
Enter a: 10
Enter b: 5
a + b = 15
a - b = 5
a * b = 50
a / b = 2

```

Types of Inheritance

- A parent class can be inherited using public, protected or private type of inheritance.
- The type of inheritance defines the access status of parent class member in derived class.
- Different types of inheritance are as follows.

1. Public Inheritance

- In public inheritance ,the access status of parent class member in derived class remains the same.
- The **public** members of parent class become **public** members of derived class.
- The **protected** members of parent class become **protected** of derived class.
- The **private** members of parent class become **private** members of derived class.

Syntax

The syntax of specifying a derived class is as follows

```
class child_class : public parent_class
{
    Body of the class
};
```

Where,

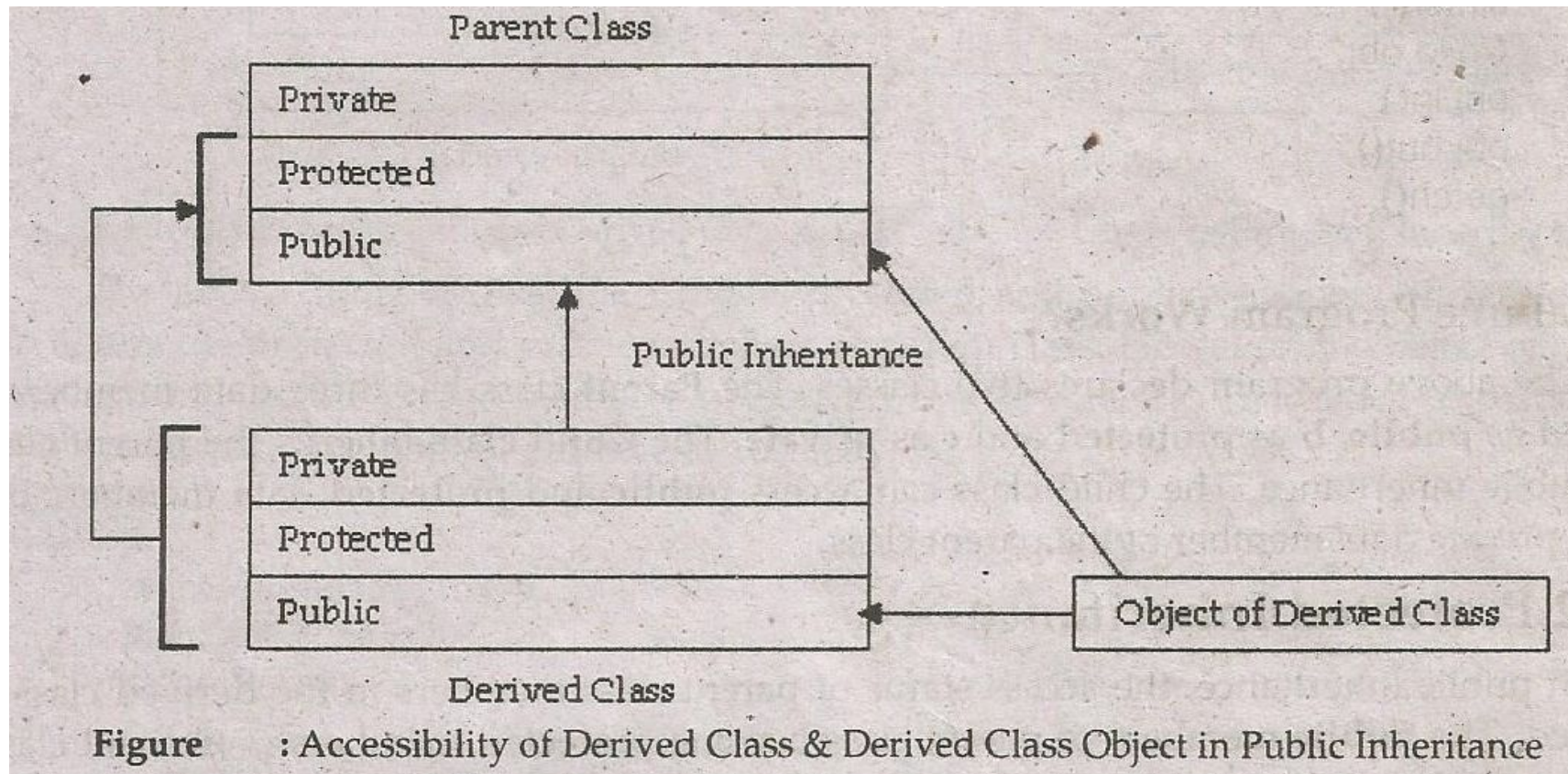
Class	It is keyword that is used to declare class.
child_class	It is name of derived class.
:	It creates relationship between derived class and super class.
Public	It is a keyword that is used to define public inheritance.
parent_class	It indicates the name of parent class that that inherited.

The accessibility of derived class in public inheritance is as follows.

- The derived class can access **public** members of parent class.
- The derived class can access **private** members of parent class.
- The derived class can access **protected** members of parent class.

The accessibility of an object of derived class is as follows.

- The object of derived class can access **public** members of parent class.
- The object of derived class cannot access **private** members of parent class.
- The objects of derived class cannot access **protected** members of parent class.



- The above figure shows that private protected and public members of a derived class can access the public and protected member of parent class.
- However, the object of derived class can only access public members of both classes directly.

Write a program that declares two classes and defines a relationship between them using public inheritance.

```
#include <iostream>    class Child : public Parent    void out()
using namespace std;  {
                        {
class Parent          public:      cout<<"a= "<<a<<endl;
{                      void in()   cout<<"b= "<<b<<endl;
public:                {           }
    int a;             {           };
protected:            cout<<"Enter a: ";
    int b;             cin>>a;
private:               cout<<"Enter b: ";
    int c;             cin>>b;
};                     }

                        }

int main()
{
    Child obj;
    obj.in();
    obj.out();
}
```

```
Enter a: 3
Enter b: 7
a= 3
b= 7
```

2. Protected Inheritance

- In public inheritance the access status of parent class members in the derived class is restricted.
- The **public** members of parent class becomes **protected** members of derived class.
- The **protected** members of parent class becomes **protected** members of derived class.
- The **private** members of parent class become **private** member of derived class.

Syntax

The syntax of specifying a derived class is as follows

```
class child_class : protected parent_class
{
    Body of the class
};
```

Where,

Class	It is keyword that is used to declare class.
child_class	It is name of derived class.
:	It creates relationship between derived class and super class.
Protected	It is a keyword that is used to define protected inheritance.
parent_class	It indicates the name of parent class that that inherited.

The accessibility of derived class in protected inheritance is as follows.

- The derived class can access **public** members of parent class.
- The derived class can not access **private** members of parent class.
- The derived class can access **protected** members of parent class.

The accessibility of an object of derived class is as follows.

- The object of derived class cannot access **public** members of parent class.
- The object of derived class cannot access **private** members of parent class.
- The objects of derived class cannot access **protected** members of parent class.

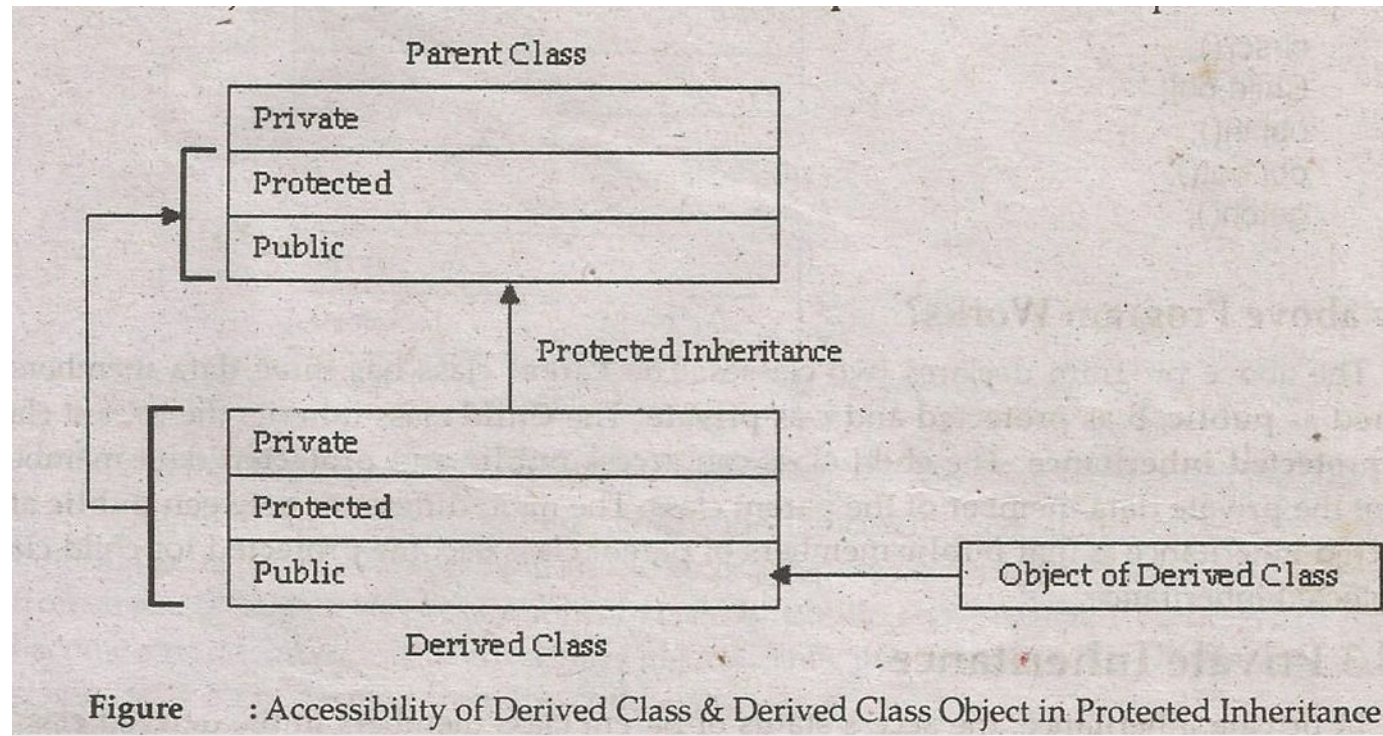


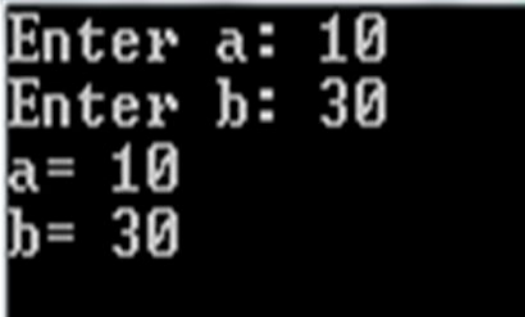
Figure : Accessibility of Derived Class & Derived Class Object in Protected Inheritance

- The above figure shows that private, protected and public members of derived class can access the protected and public members of parent class.
- However, the object of derived class can only access public members of derived classes directly .

Write a program that declares two classes and defines a relationship between them using protected inheritance.

```
#include <iostream>    class Child : protected Parent    void out()
using namespace std;  {
                        {
                            cout<<"a= "<<a<<endl;
                            cout<<"b= "<<b<<endl;
                        }
class Parent          public:
{
    public:            void in()
                        {
                            cout<<"Enter a: ";
                            cin>>a;
                            cout<<"Enter b: ";
                            cin>>b;
                        }
    int a;
    protected:
        int b;
    private:
        int c;
};

int main()
{
    Child obj;
    obj.in();
    obj.out();
}
```

A screenshot of a terminal window showing the output of the program. The text is white on a black background. It shows the prompts 'Enter a:' and 'Enter b:' followed by the user input '10' and '30' respectively. Then it shows the program's output 'a= 10' and 'b= 30'.

```
Enter a: 10
Enter b: 30
a= 10
b= 30
```

3. Private Inheritance

- In private inheritance, the access status of parent class in the derived class is restricted.
- The **private**, **protected** and **public** members of parent class all become the **private** member of derived class.

Syntax

The syntax of specifying a derived class is as follows

```
class child_class : private parent_class
{
    Body of the class
};
```

Where,

Class	It is keyword that is used to declare class.
child_class	It is name of derived class.
:	It creates relationship between derived class and super class.
Private	It is a keyword that is used to define private inheritance.
parent_class	It indicates the name of parent class that that inherited.

The accessibility of derived class in private inheritance is as follows.

- The derived class can access **public** members of parent class.
- The derived class can access **private** members of parent class.
- The derived class cannot access **protected** members of parent class.

The accessibility of an object of derived class is as follows.

- The object of derived class cannot access **public** members of parent class.
- The object of derived class cannot access **private** members of parent class.
- The objects of derived class cannot access **protected** members of parent class.

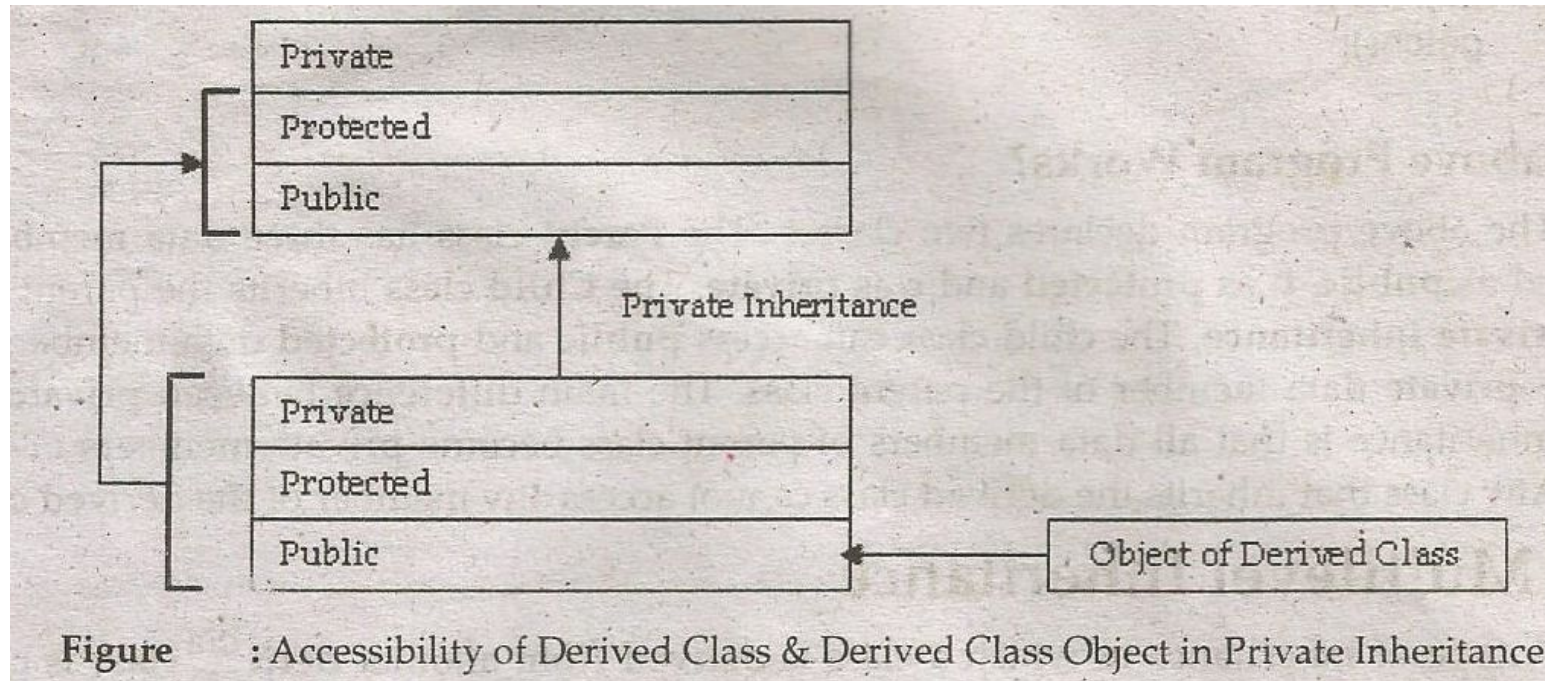


Figure : Accessibility of Derived Class & Derived Class Object in Private Inheritance

- The above figure shows that private, protected and public members of derived class can access the protected and public members of parent class.
- However, the object of derived class can only access public members of derived classes directly .

Write a program that declares two classes and defines a relationship between them using private inheritance.

```
#include <iostream>    class Child : private Parent    void out()
using namespace std;  {
                        {
class Parent          public:      cout<<"a= "<<a<<endl;
{                      void in()   cout<<"b= "<<b<<endl;
public:                {           }
    int a;             {           };
protected:            cout<<"Enter a: ";
    int b;             cin>>a;
private:               cout<<"Enter b: ";
    int c;             cin>>b;
};                     }

                        }

int main()
{
    Child obj;
    obj.in();
    obj.out();
}
```

```
Enter a: 45
Enter b: 78
a= 45
b= 78
```

Accessibility of variables in Public, Private and protected Inheritance

1)Accessibility in Protected Inheritance

Accessibility	private variables	protected variables	public variables
Accessible from own class?	yes	yes	yes
Accessible from derived class?	no	yes	yes (inherited as protected variables)
Accessible from 2nd derived class?	no	yes	yes

2)Accessibility in Public Inheritance

Accessibility	private variables	protected variables	public variables
Accessible from own class?	yes	yes	yes
Accessible from derived class?	no	yes	yes
Accessible from 2nd derived class?	no	yes	yes

3)Accessibility in Private Inheritance

Accessibility	private variables	protected variables	public variables
Accessible from own class?	yes	yes	yes
Accessible from derived class?	no	yes (inherited as private variables)	yes (inherited as private variables)
Accessible from 2nd derived class?	no	no	no

Example of public, protected and private inheritance in C++

Example of public, protected and private inheritance in C++

class base

{

public:

int x;

protected:

int y;

private:

int z;

};

class publicDerived: public base

{

// x is public

// y is protected

// z is not accessible from publicDerived

};

class protectedDerived: protected base

{

// x is protected

// y is protected

// z is not accessible from protectedDerived

};

class privateDerived: private base

{

// x is private

// y is private

// z is not accessible from privateDerived

}

Explanation

In the above example, we observe the following things:

base has three member variables: x, y and z which are public, protected and private member respectively.

Public Derived inherits variables x and y as public and protected. z is not inherited as it is a private member variable of base.

Protected Derived inherits variables x and y. Both variables become protected. z is not inherited.

If we derive a class derived From Protected Derived from protected Derived, variables x and y are also inherited to the derived class.

Private Derived inherits variables x and y. Both variables become private. z is not inherited

If we derive a class derived From PrivateDerived from private Derived, variables x and y are not inherited because they are private variables of private Derived.



End of lecture