Welcome

Object Oriented Programming (UTA018)

(Session: 2026-2027 ODD)

Instructor: Dr. Suresh Raikwar

Department of Computer Science and Engineering,

Thapar Institute of Engineering and Technology, Patiala, Punjab, India

Important Instruction(s):

- This material is intended for guidance and conceptual understanding. It does not guarantee that examination questions will be drawn directly from it.
- Merely studying this material is not sufficient for achieving good performance.
- Sessional Tests, MST, and EST will assess your understanding of Object-Oriented Programming concepts through **logical reasoning, analytical thinking, and problem-solving**.
- Students are strongly encouraged to attend regular classes and practice programming problems from multiple sources to develop a deeper conceptual understanding and improve problem-solving skills.

Inheritance Part 1

Inheritance

- **Inherit Definition** -

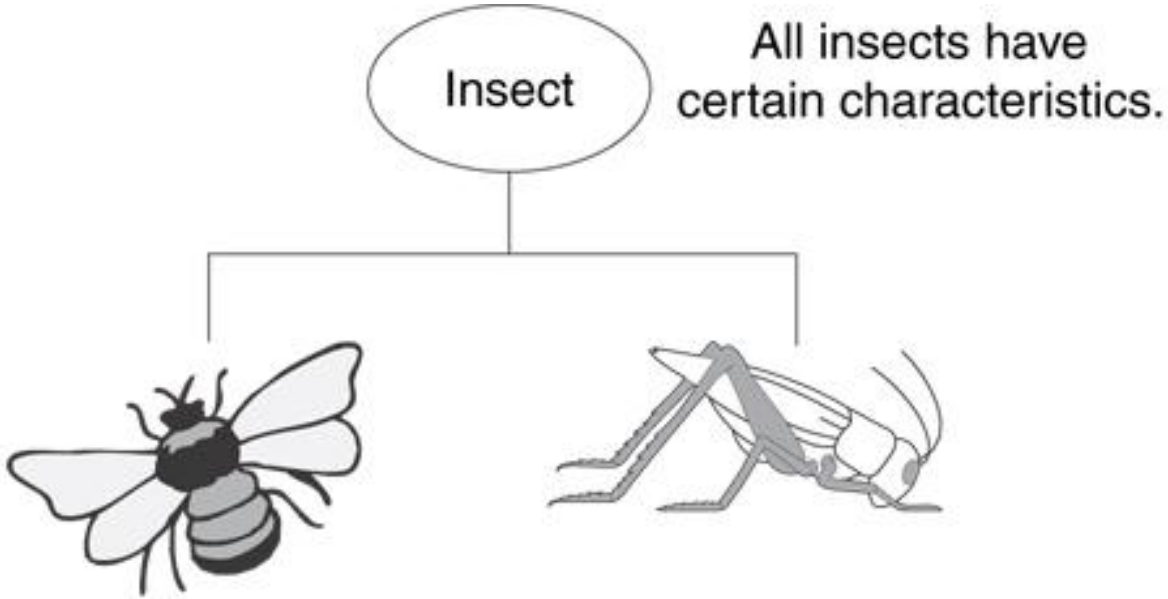
  Derive quality and characteristics from parents or ancestors.
- **Example:**

  “He/She had inherited the wisdom of his/her parents"
- Inheritance in Object Oriented Programming can be described as a process of creating new classes from existing classes.

Inheritance (Cont…)

- New classes inherit some of the properties and behaviour of the existing classes.
- The existing class that is "parent" of a new class is called a **base class**.
- New class that inherits properties of the base class is called a **derived class(“child class”)**.
- Inheritance is a technique of code reuse.

Example: Insect Taxonomy



In addition to the common insect characteristics, the bumble bee has its own unique characteristics such as the ability to sting.

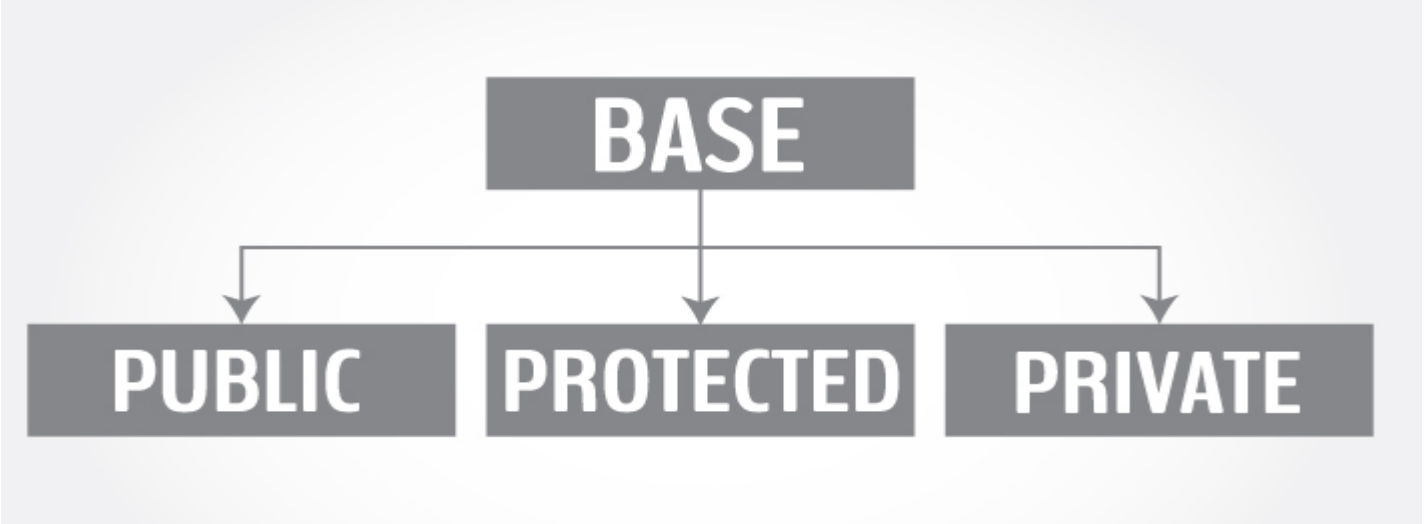
In addition to the common insect characteristics, the grasshopper has its own unique characteristics such as the ability to jump.

The "is a" Relationship

- Inheritance establishes an "is a" relationship between classes.
  - A car is a vehicle
  - A flower is a plant
  - A football player is an athlete

Base Class access control

- Derived class can be declared from a base class with different access control, i.e., public inheritance, protected inheritance or private inheritance.



Protected Members and Class Access

- Protected member access specification: like private, but accessible by objects of derived class
- Class access specification: determines how **private, protected, and public** members of base class are inherited by the derived class

Class Access Specifiers

public – object of derived class can be treated as object of base class (not vice-versa)

protected – more restrictive than public, but allows derived classes to know details of parents

private – prevents objects of derived class from being treated as objects of base class.

Syntax

1. **#include <iostream>**
2. **using namespace std;**
3. **class base**
4. **{ …. … …. };**
5. **class derived : access_specifier base**
6. **{ …. … …. };**

- Either ***public, protected*** **or** ***private*** keyword is used in place of *access_specifier* term used in the syntax code code.

```
1.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 }
```

Observations

- **base class** 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 class.
- **Protected_Derived** inherits variables *x* and *y*. Both variables become protected. *z* is not inherited.
- **Private_Derived** inherits variables *x* and *y*. Both variables become private. *z* is not inherited

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

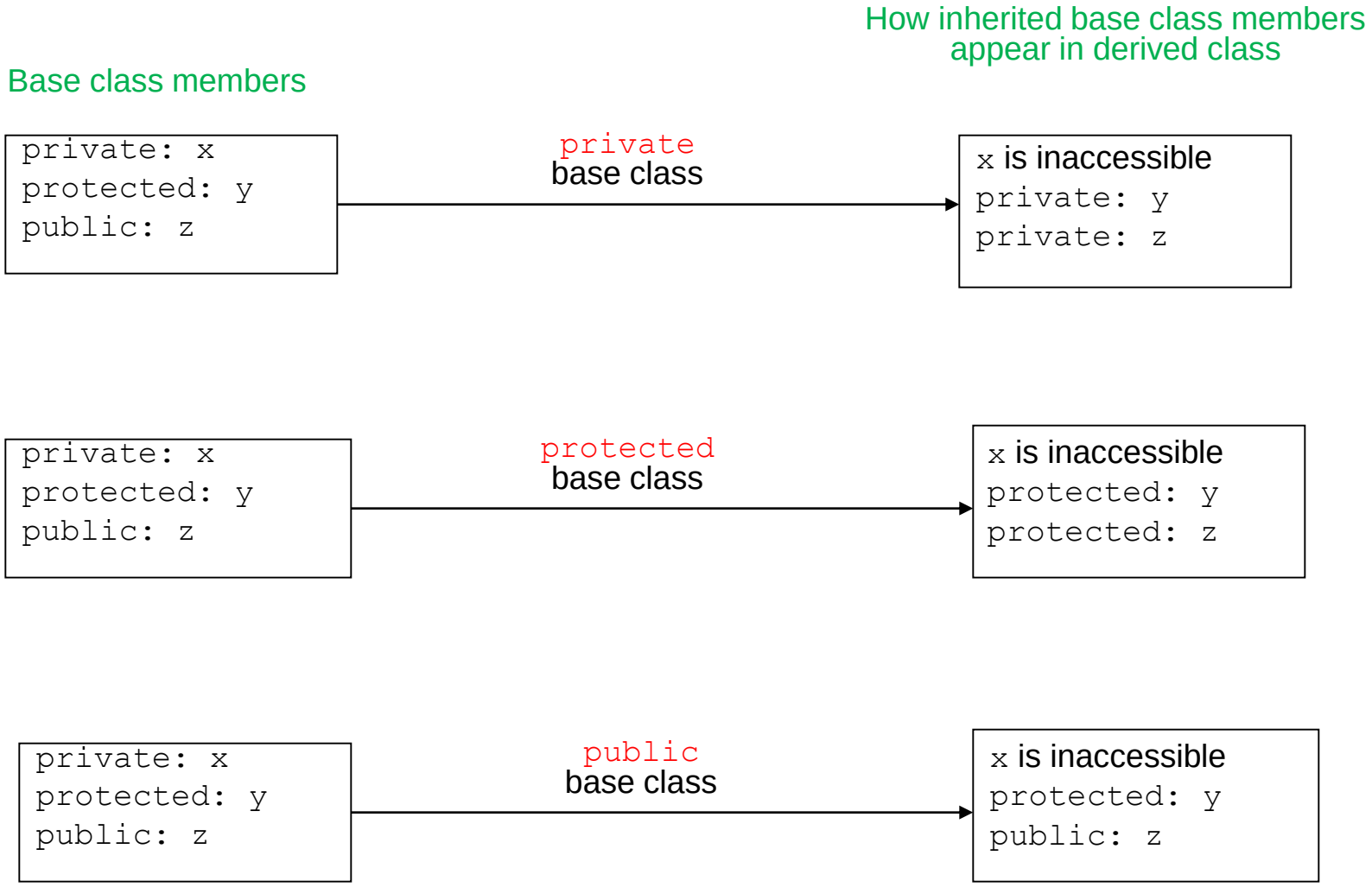
Accessibility in Protected Inheritance

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

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

Inheritance vs. Access



Inheritance vs. Access

class Grade

```
private members:
   char letter;
   float score;
   void calcGrade();
public members:
   void setScore(float);
   float getScore();
   char getLetter();
```

class Test : public Grade

```
private members:
   int numQuestions;
   float pointsEach;
   int numMissed;
public members:
   Test(int, int);
```

When `Test` class inherits from `Grade` class using `public` class access, it looks like this:

```
private members:
   int numQuestions:
   float pointsEach;
   int numMissed;
public members:
   Test(int, int);
   void setScore(float);
   float getScore();
   char getLetter();
```

Inheritance vs. Access

class Grade

```
private members:
   char letter;
   float score;
   void calcGrade();
public members:
   void setScore(float);
   float getScore();
   char getLetter();
```

class Test : protected Grade

```
private members:
   int numQuestions;
   float pointsEach;
   int numMissed;
public members:
   Test(int, int);
```

When `Test` class inherits from `Grade` class using `protected` class access, it looks like this:

→

```
private members:
   int numQuestions:
   float pointsEach;
   int numMissed;
public members:
   Test(int, int);
protected members:
   void setScore(float);
   float getScore();
   float getLetter();
```

Inheritance vs. Access

class Grade

```
private members:
   char letter;
   float score;
   void calcGrade();
public members:
   void setScore(float);
   float getScore();
   char getLetter();
```

class Test : private Grade

```
private members:
   int numQuestions;
   float pointsEach;
   int numMissed;
public members:
   Test(int, int);
```

When `Test` class inherits from `Grade` class using `private` class access, it looks like this:

```
private members:
   int numQuestions:
   float pointsEach;
   int numMissed;
   void setScore(float);
   float getScore();
   float getLetter();
public members:
   Test(int, int);
```

What Does a Child Have?

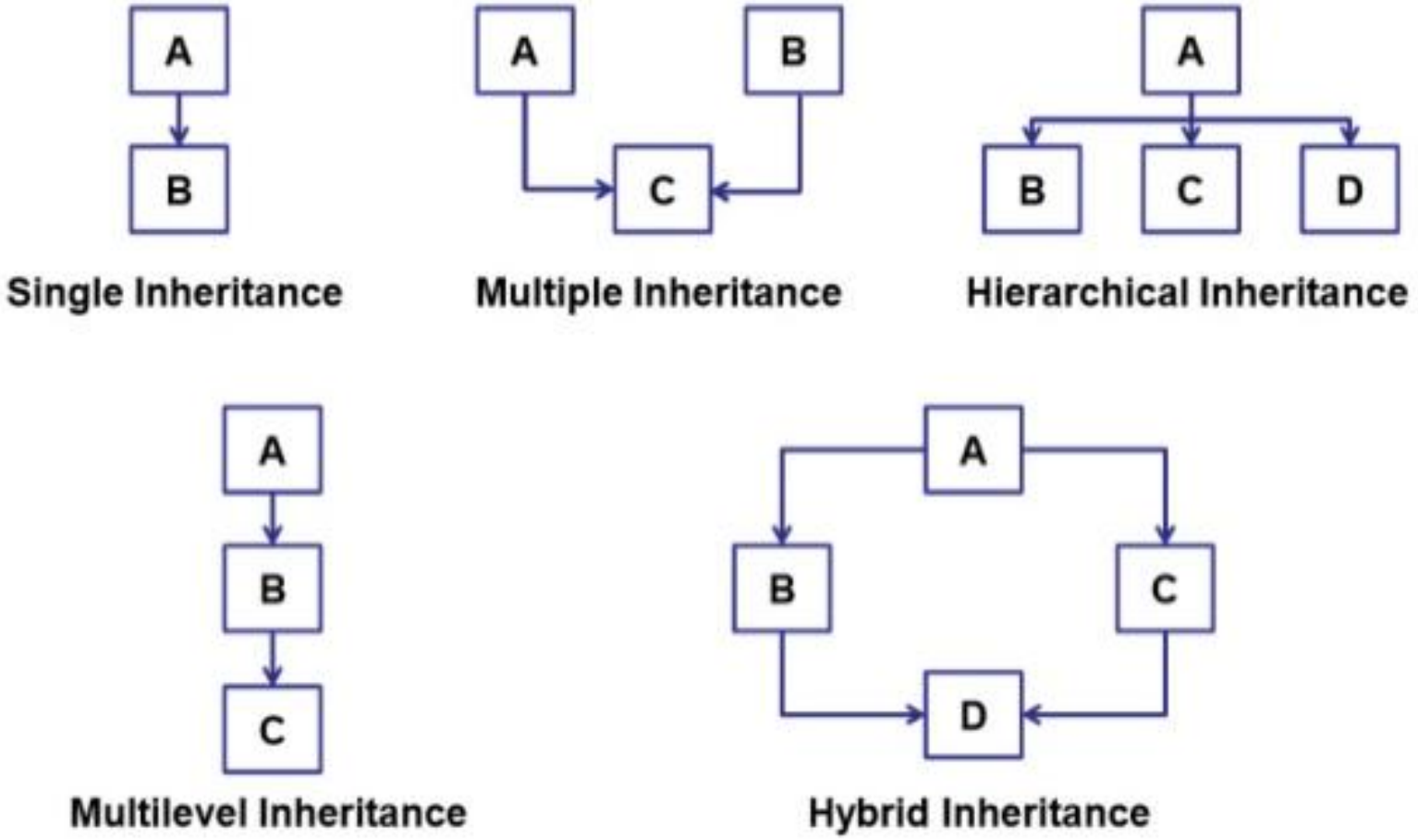
An object of the derived class has:

- all members defined in child class
- all members declared in parent class

An object of the derived class can use:

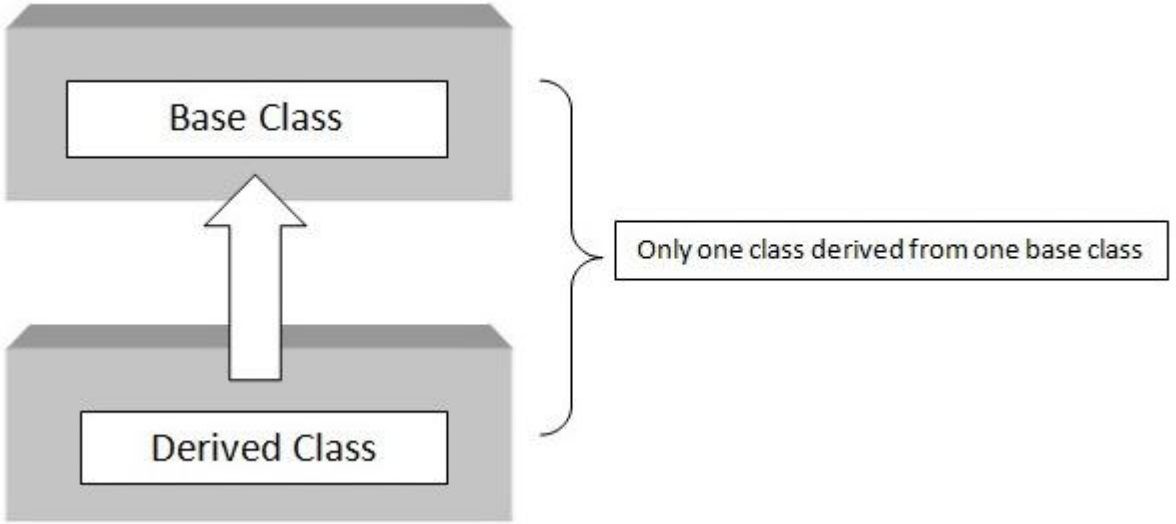
- all public members defined in child class
- all public members defined in parent class

Types of Inheritance



Single Inheritance

- **Single Inheritance:** It is the inheritance hierarchy wherein one derived class inherits from one base class.



Single Inheritance Syntax

```
class A   // base class
{
    ..........
};
class B : acess_specifier A   // derived class
{
    ...........
} ;
```

Single Inheritance Example

```
#include <iostream.h>
using namespace std;
class Shape {
protected:
        int width;
        int height;
public:
    void setWidth(int w) {
          width = w;    }
    void setHeight(int h) {
           height = h;  } } ;
class Rectangle: public Shape {
 public:
        int getArea() {
        return (width * height); } }
 int main {
 Rectangle Rect;
 Rect.setWidth(5);
 Rect.setHeight(7);
 cout << "Total area: " << Rect.getArea() << endl;    //   Print the area of the object.
 return 0;
        }
```

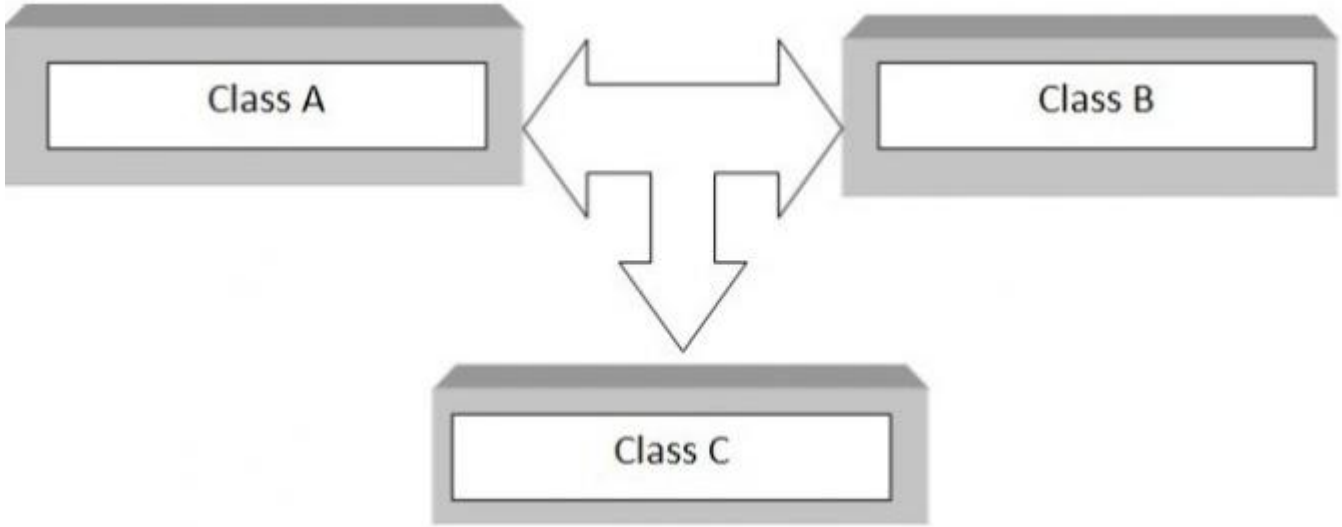
```
#include <iostream.h>
using namespace std;
class base {
int i, j;
public:
void set(int a, int b){
        i=a; j=b; }
void show() {
        cout << i << " " << j << "\n"; }
};
class derived : public base {
int k;
public:
derived(int x) {
        k=x; }
void showk() {
        cout << k << "\n"; }
};
```

```
int main()
{
derived ob(3);
ob.set(1, 2); // access member of base
ob.show(); // access member of base
ob.showk(); // uses member of derived class
return 0;
}
```

```
// This program won't compile.
#include <iostream.h>
using namespace std;
class base {
int i, j;
public:
void set(int a, int b) { i=a; j=b; }
void show() { cout << i << " " << j <<
    "\n";}
};
// Public elements of base are private
    in derived.
class derived : private base {
int k;
public:
derived(int x) { k=x; }
void showk()
{ cout << k << "\n"; }
};
int main()
{
derived ob(3);
ob.set(1, 2);
// error, can't access set()
ob.show();
// error, can't access show()
return 0;
}
```

Multiple Inheritance

- **Multiple Inheritance:** It is the inheritance hierarchy wherein one derived class inherits from multiple base class(es).



Syntax

```
class A
{
  ..........
};
class B
{
  ...........
 } ;
class C : acess_specifier A,access_specifier A // derived class from A and B
{
  ...........
} ;
```

Inheriting Multiple Base Classes

```
#include <iostream>
using namespace std;
class base1 {
protected:
int x;
public:
void showx() { cout << x << "\n"; }
};
class base2 {
protected:
int y;
Public:
void showy() {cout << y << "\n";}
};
```

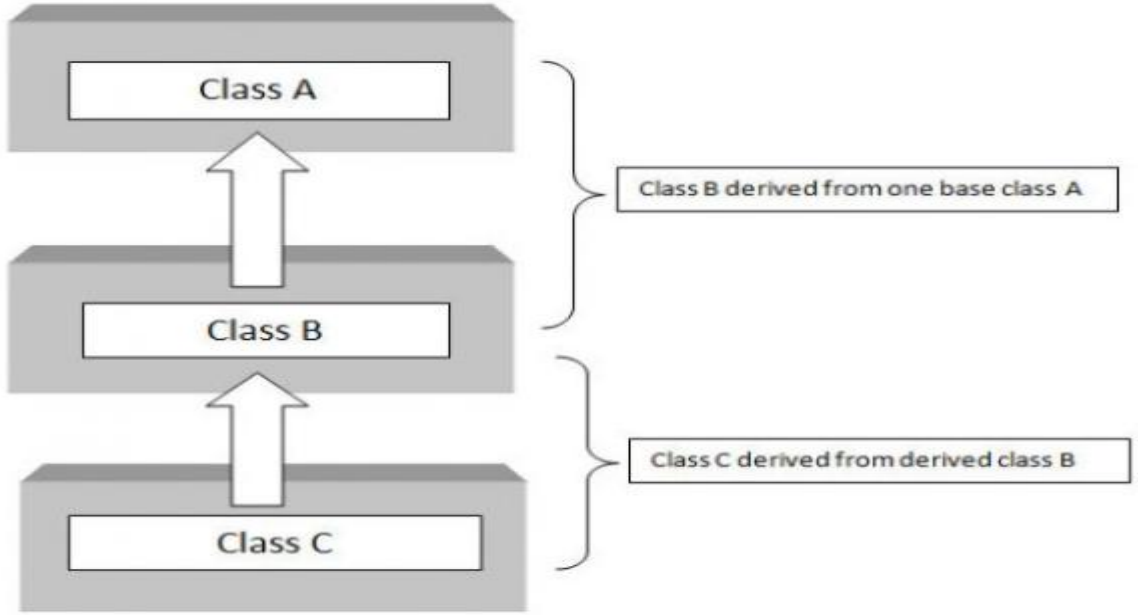
```
// Inherit multiple base classes.
class derived: public base1, public base2 {
public:
void set(int i, int j) { x=i; y=j; }
};
int main()
{
derived ob;
ob.set(10, 20);
// provided by derived
ob.showx(); // from base1
ob.showy(); // from base2
return 0;
}
```

```
//EXAMPLE
#include <iostream.h>
using namespace std;
// Base class Shape
class Shape {
  public:
 void setWidth(int w) {
                    width = w; }
 void setHeight(int h) {
                    height = h; }
  protected:
    int width;
    int height;
};
// Base class PaintCost
class PaintCost {
  public:
  int getCost(int area) {
      return area * 70;     }};
```

```
// Derived class
class Rectangle: public Shape, public PaintCost {
  public:
    int getArea() {
      return (width * height);
    } };
 int main(void) {
  Rectangle Rect;
  int area;
  Rect.setWidth(5);
  Rect.setHeight(7);
  area = Rect.getArea();
// Print the total cost of painting
  cout << "Total paint cost: $" << Rect.getCost(area)
     << endl;
 return 0; }
```

Multilevel Inheritance

- **Multilevel Inheritance:** It is the inheritance hierarchy wherein subclass acts as a base class for other classes.



Syntax

```
class A // base class
{
     ...........
};
class B : acess_specifier A // derived class
{
     ...........
 } ;
 class C : access_specifier B // derived from derived class B
 {
     ...........
 } ;
```

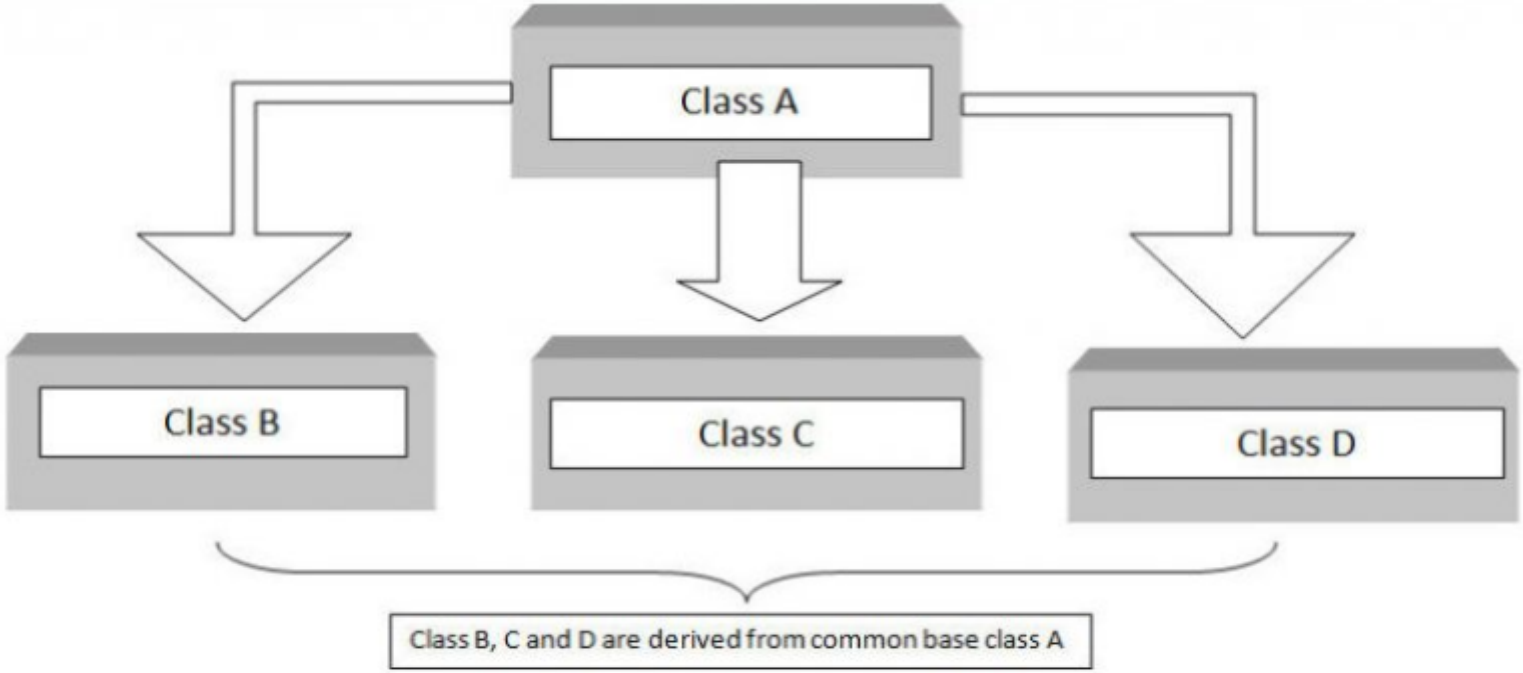
Multi-level Inheritance

```
#include <iostream>
using namespace std;
//Base class
class base {
public:
void display1() {
 cout << "\nBase class content."; }
 };
//derived class
class derived : public base
{
public:
void display2()
{
cout << "1st derived class content."; } };
```

```
//derived2 class
class derived2 : public derived
{ public:
 void display3(){
 cout << "\n2nd Derived class content.";
 } };
 int main()
 {
derived2 D;
D.display3();
D.display2();
D.display1();
return(0);
}
```

Hierarchical Inheritance

- **Hierarchical Inheritance:** It is the inheritance hierarchy wherein multiple subclasses inherit from one base class.



Hierarchical Inheritance

```
class A // base class
{
    ..............
};
class B : access_specifier A // derived class from A
{
    ...........
 } ;
 class C : access_specifier A // derived class from A
 {
    ...........
 } ;
 class D : access_specifier A // derived class from A
 {
     ...........
 } ;
```

Hierarchical Inheritance

```
#include <iostream>
#include <string.h>
using namespace std;
//Base Class
class member {
char gender[10];
 int age;
public:
 void get()
{ cout << "Age: "; cin >> age;
 cout << "Gender: "; cin >> gender; }
```

```
void disp() {
 cout << "Age: " << age << endl; cout
    << "Gender: " << gender << endl; }
};
//derived from member
class stud : public member
 { char level[20];
public:
void getdata() {     member::get();
    cout << "Class: ";
    cin >> level;    }
```

Continue...

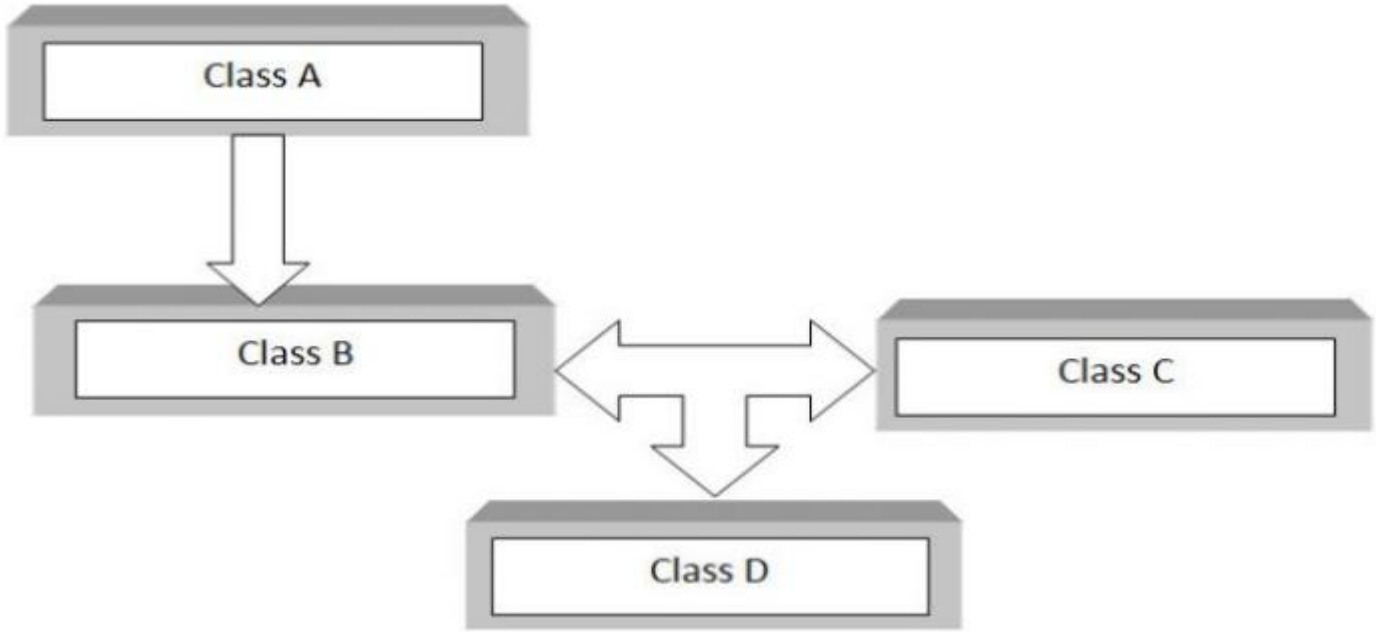
```
void disp2()
 {   member::disp();
 cout << "Level: " << level;
 } };
//staff class derived from member
class staff : public member
{ float salary;
public:
 void getdata() { member::get();
cout << "Salary: Rs.";
 cin >> salary; }
```

```
void disp3() { member::disp();
cout << "Salary: Rs." << salary << endl;
 } };

int main() {
//member M;
staff S;
stud s;
s.getdata();
s.disp();
S.getdata();
S.disp();
return(0); }
```

Hybrid Inheritance

- **Hybrid Inheritance:** The inheritance hierarchy that reflects any legal combination of other four types of inheritance.



Syntax

```
class A
{
      .........
};
class B : public A
{
      ..........
} ;
class C
{
      ..........
};
 class D : public B, public C
{
      ..........
};
```

```
#include <iostream>
using namespace std;
class A {
        public: int x;
             };
class B : public A {
        public: B() //constructor to initialize
    x in base class A
                          { x = 10; }
                          };
class C {
public: int y; C() //constructor to initialize
    y { y = 4; }
             };
class D : public B, public C
  //D is derived from class B and class C
{ public:
    void sum()
    { cout << "Sum= " << x + y; }
 };
 int main()
{ D obj1; //object of derived class D
obj1.sum();
 return 0;
}
```

```
Sum= 14
```

```cpp
#include<iostream>
using namespace std;
class Base{
    private:
        int a;
    protected:
        int b;
    public:
        int c;
    void setVal(int x,int y,int z){
        a=x; b=y; c=z;
    }
    void display(){
cout<<"Base:\n"<<"a:"<<a<<",b:"<<b<<",c:"<<c<<endl;
    }
};

class publicDerived: public Base{
    public:
        void display(){
            cout<<"public:\n"<<",b:"<<b<<",c:"<<c<<endl;
        }
};
class protectedDrived:protected Base{
    public:
            void display(){
            cout<<"protected:\n"<<",b:"<<b<<",c:"<<c<<endl;
            }
};
class privateDrived:private Base{
    public:
            void display(){
            cout<<"private:\n"<<"b:"<<b<<"c:"<<c<<endl;
            }
};

int main(){
    Base base;
    base.setVal(1,2,3);
    base.display();
    publicDerived obj1;
    obj1.display();
    protectedDrived obj2;
    obj2.display();
    privateDrived obj3;
    obj3.display();
    return 0;

}
```

Inheritance Part 2 and Poly-morphism

Content

- **Constructor and Destructor in Inheritance**
- **Inheritance and Static Functions**
- **Virtual class**
- **Abstract Class**
- **Pure Virtual Functions**
- **Polymorphism**
- **Overloading and Overriding**

Constructor and Destructor in Inheritance

Constructors and Destructors in Base and Derived Classes

- Derived classes can have their own constructors and destructors.
- When an object of a derived class is created, the base class's constructor is executed first, followed by the derived class's constructor.
- When an object of a derived class is destroyed, its destructor is called first, then that of the base class.

```
//EXAMPLE
#include<iostream>
Using namespace std;
//base class
class base {
public:
base()  { cout << "Constructing base\n"; }
~base()  { cout << "Destructing base\n"; }
};
//derived class
class derived: public base {
public:
derived() { cout << "Constructing derived\n"; }
~derived()
{ cout << "Destructing derived\n"; }
};
int main()
{
derived ob;
// do nothing but construct and destruct ob
return 0;
}
```

Program Output
Constructing base
Constructing derived
Destructing derived
Destructing base

Constructors and Destructors with Multiple Base Classes

- **Constructors are called in order of derivation**, left to right, as specified in derived's inheritance list.
- **Destructors are called in reverse order,** right to left.

```
//MULTI-LEVEL
#include <iostream>
using namespace std;
class base {
public:
base()
    { cout << "Constructing base\n"; }
~base()
    { cout << "Destructing base\n"; }
};
class derived1 : public base {
public:
derived1()
    { cout << "Constructing derived1\n"; }
~derived1()
    { cout << "Destructing derived1\n"; }
};
```

```
class derived2: public derived1 {
public:
derived2()
    { cout << "Constructing derived2\n"; }
~derived2()
    { cout << "Destructing derived2\n"; }
};
int main()
{
derived2 ob;
// construct and destruct ob
return 0;
}
```

Program Output:
Constructing base
Constructing derived1
Constructing derived2
Destructing derived2
Destructing derived1
Destructing base

```
//MULTIPLE BASE CASES
#include <iostream>
using namespace std;
class base1 {
public:
base1()
    { cout << "Constructing base1\n"; }
~base1()
    { cout << "Destructing base1\n"; }
};
class base2 {
public:
base2()
    { cout << "Constructing base2\n"; }
~base2()
    { cout << "Destructing base2\n"; }
};
```

```
class derived: public base1, public base2 {
public:
derived()
    { cout << "Constructing derived\n"; }
~derived()
    { cout << "Destructing derived\n"; }
};
int main()
{
derived ob;
// construct and destruct ob
return 0;
}
```

Program Output:
Constructing base1
Constructing base2
Constructing derived
Destructing derived
Destructing base2
Destructing base1

Important Note:

Besides being unable to directly access private members of a parent class, several parent class members are never inherited:

- » constructors
- » destructors
- » friend functions
- » overloaded new operators
- » overloaded = operators

If a derived class requires any of the preceding items, the item must be explicitly defined within the derived class definition

Question:1

If a derived class uses the public access specifier for inheritance, then which of the following statements are true:

A. Base class members that are public remain public in the derived class.

B. Base class members that are protected become public in the derived class.

C. Base class members that are private are inaccessible in the derived class.

D. None of the Above

Question:2

If a derived class uses the protected access specifier for inheritance, then which of the following statements are true:

A. Base class members that are public remain public in the derived class.

B. Base class members that are protected remain protected in the derived class.

C. Base class members that are private are inaccessible in the derived class.

D. None of the above

Question:3

If a derived class uses the private access specifier for inheritance, then which of the following statements are true:

A. Base class members that are public become private in the derived class.

B. Base class members that are protected become private in the derived class.

C. Base class members that are private are inaccessible in the derived class.

D. None of the above

Question:4

Class A serves as a base class to class B, and B is a parent to class C. When you instantiate a class B object, the first constructor called belongs to class _____ .

a. A

b. B

c. C

d. none of the above

Question:5

Class A contains a nonstatic void public function named functionA() that requires an integer argument. Class B derives from class A, and also contains a nonstatic void public function named functionA() that requires an integer argument. An object of class B can use _____ .

a. the class A version of the function

b. the class B version of the function

c. both of the above

d. none of the above

Practice Question

Create a `RestaurantMeal` class that holds the name and price of a food item served by a restaurant. Its constructor requires arguments for each field. Create a `HotelService` class that holds the name of the service, the service fee, and the room number to which the service was supplied. Its constructor also requires arguments for each field. Create a `RoomServiceMeal` class that inherits from both `RestaurantMeal` and `HotelService`. Whenever you create a `RoomServiceMeal` object, the constructor assigns the string "room service" to the name of the service field, and $4.00 is assigned to the service fee inherited from `HotelService`. Include a `RoomServiceMeal` function that displays all of the fields in a `RoomServiceMeal` by calling display functions from the two parent classes. Additionally, the display function should display the total of the meals plus the room service fee. In a `main()` function, instantiate a `RoomServiceMeal` object that inherits from both classes. For example, a "steak dinner" costing $19.99 is a "room service" provided to room 1202 for a $4.00 fee. Save the file as **RoomService.cpp**.

Passing Arguments to Base Class Constructor

derived-constructor(arg-list) : base1(arg-list), base2(arg-list),// ... baseN(arg-list)

{

// body of derived constructor

}

Order of Constructor Call

- Base class constructors are always called in the derived class constructors.
- Whenever you create derived class object, first the base class default constructor is executed and then the derived class's constructor finishes execution.

```
//EXAMPLE
#include <iostream>
using namespace std;
class base {
protected:
int i;
public:
base(int x)
{ i=x;
cout << "Constructing base\n"; }
~base() { cout << "Destructing base\n"; }
};
class derived: public base {
int j;
public:
// derived uses x; y is passed along to base.
derived(int x, int y): base(y)
{ j=x;
cout << "Constructing derived\n"; }
~derived()
{ cout << "Destructing derived\n"; }
void show()
{ cout << i << " " << j << "\n"; }
};
int main()
{
derived ob(3, 4);
ob.show(); // displays 4 3
return 0;
}
```

```
//MULTIPLE BASE CASES
#include <iostream>
using namespace std;
class base1 {
protected: int i;
public:
base1(int x)
{ i=x; cout << "Constructing base1\n"; }
~base1()  { cout << "Destructing base1\n"; }};
class base2 {
protected: int k;
public:
base2(int x) { k=x; cout << "Constructing base2\n"; }
~base2() { cout << "Destructing base1\n"; } };
```

```
class derived: public base1, public base2 {
int j;
public:
derived(int x, int y, int z): base1(y), base2(z)
{ j=x; cout << "Constructing derived\n"; }
~derived() { cout << "Destructing derived\n"; }
void show() { cout << i << " " << j << " " << k << "\n"; }
};
int main()
{ derived ob(3, 4, 5);
ob.show(); // displays 4 3 5
return 0; }
```

Here's what actually happens when derived is instantiated:

- Memory for derived is set aside (enough for both the Base and Derived portions)
- The appropriate Derived constructor is called
- **The Base object is constructed first using the appropriate Base constructor**. If no base constructor is specified, the default constructor will be used.
- The initialization list initializes variables
- The body of the constructor executes
- Control is returned to the caller

Points to note

- It is important to understand that **arguments to a base-class constructor are passed via arguments** to the derived class' constructor.

- Even **if a derived class‘ constructor does not use any arguments**, it will **still need to declare one if the base class requires it**.

- In this situation, the arguments passed to the derived class are simply passed along to the base.

Why is Base class Constructor called inside Derived class?

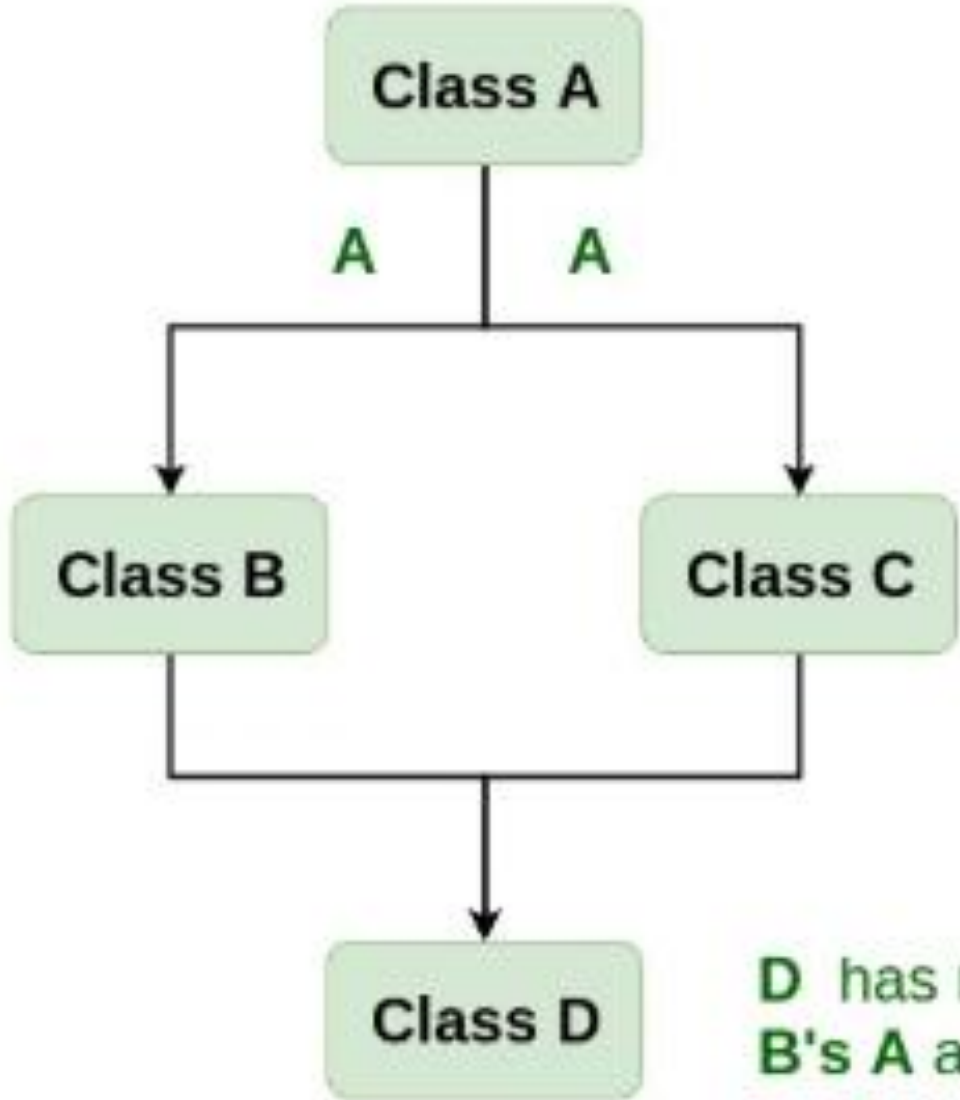
- Constructors have **a special job of initializing the object properly.**
- A Derived class constructor has access only to **its own class members**, but a Derived class object also **have inherited property of Base class**, and **only base class constructor can properly initialize base class members**.
- Hence all the constructors are called, else object wouldn't be constructed properly.

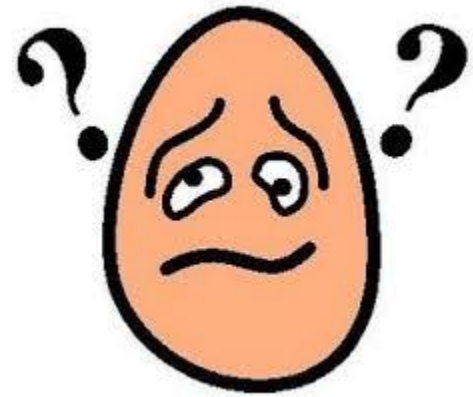
```
// This program contains an error and will
   not compile.
#include <iostream>
using namespace std;
class base {
public: int i; };
class derived1 : public base {
public: int j; };
class derived2 : public base {
public: int k; };
class derived3 : public derived1, public
   derived2 {
public: int sum; };
```

```
int main() {
derived3 ob;
ob.i = 10; // this is ambiguous, which
   i???
ob.j = 20;
ob.k = 30;
// i ambiguous here, too
ob.sum = ob.i + ob.j + ob.k;
// also ambiguous, which i?
cout << ob.i << " ";
cout << ob.j << " " << ob.k ;
cout << ob.sum;
return 0;
}
```

Discussion

- which **'i'** is being referred to, the one in derived1 or the one in derived2?
- Because there are **two copies** of base present in object ob, there are **two ob.is**! ->, *the statement is inherently ambiguous*.
- There are two ways to remedy the preceding program.
- The first is to apply the scope resolution operator to i and manually select one **i**. Example (on next slide).


Class A
A
A
Class B
Class C
Class D
D has now 2 A:
B's A and C's A



```
// This program uses explicit scope
   resolution to select i.
#include <iostream>
using namespace std;
class base { public: int i; };
// derived1 inherits base.
class derived1 : public base {
public: int j; };
// derived2 inherits base.
class derived2 : public base {
public: int k; };
class derived3 : public derived1, public
   derived2 {
public: int sum; };
```

```
int main()
{
derived3 ob;
ob.derived1::i = 10; // scope resolved, use
     derived1's i
ob.j = 20;
ob.k = 30;
// scope resolved
ob.sum = ob.derived1::i + ob.j + ob.k;
// also resolved here
cout << ob.derived1::i << " ";
cout << ob.j << " " << ob.k << " ";
cout << ob.sum;
return 0; }
```

Discussion

- As you can see, because **the :: was applied**, the program has manually **selected derived1's version of base**.
- What if only **one copy of base is actually required**? Is there some way to **prevent two copies from being included in derived3**?
- This solution is achieved using **virtual base classes**.

- When two or more objects are derived from a common base class, you can prevent multiple copies of the base class from being present in an object derived from those objects by **declaring the base class as virtual when it is inherited**.
- You accomplish this by preceding the base class' name with the keyword **virtual** when it is inherited.
- For example, here is another version of the example program in which derived3 contains only one copy of base:

Example Program Next Slide

```
// This program uses virtual base classes.
#include <iostream>
using namespace std;
class base { public: int i; };

// derived1 inherits base as virtual.

class derived1 : virtual public base {
public: int j; };

// derived2 inherits base as virtual.

class derived2 : virtual public base {
public: int k; };

class derived3 : public derived1, public derived2
{ public: int sum; };
```

```
int main()
{
derived3 ob;
ob.i = 10; // now unambiguous
ob.j = 20;
ob.k = 30;
// unambiguous
ob.sum = ob.i + ob.j + ob.k;
// unambiguous
cout << ob.i << " ";
cout << ob.j << " " << ob.k << " ";
cout << ob.sum;
return 0;
}
```

Discussion

- As you can see, the keyword **virtual** precedes the rest of the inherited class‘specification.
- Now that **both derived1 and derived2** have inherited base as **virtual**, any multiple inheritance involving them **will cause only one copy of base to be present**.
- Therefore, in derived3, there is **only one copy of base and ob.i = 10** is *perfectly valid and unambiguous*.

```
class A {
public: void show() {cout<<"In A"<<endl;}
};

class B : public A {};
class C : public A {};
class D : public B, public C {};

int main() {D d; d.show(); }
```

Output

[Error] request for member 'show' is ambiguous

Need ***virtual*** keyword to fix this error

```
class A {
public: void show() {cout<<"In A"<<endl;}
};

class B : public virtual A {};
class C : public virtual A {};
class D : public B, public C {};

int main() {D d; d.show(); }
```


दृश्य एवं संकेत सूचना प्रसंसकरण समूह
Visual Signal and Information Processing Group
TIET


ti
THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Thank You!