OBJECT-ORIENTED PROGRAMMING (OOP)

Week – 07
Lecturer:Sobia Iftikhar

Week Six– Class One

Object Relationship

- Object oriented programming generally support 4 types of relationships that are:

Inheritance
- Is-a relationship

Composition
- Part-of relationship

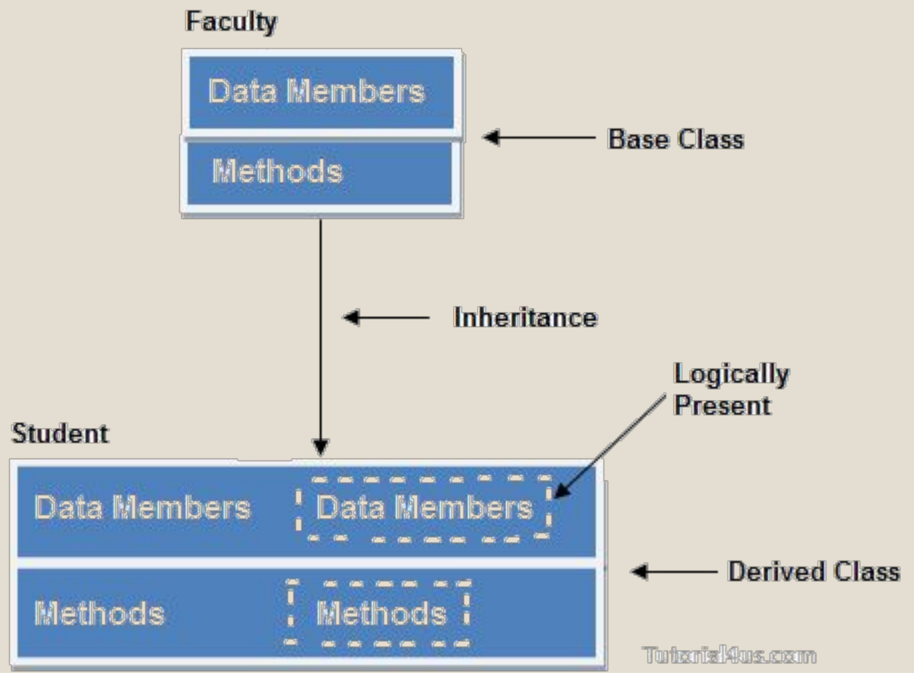
Aggregation and Association
- has-a relationship

Inheritance : Is-a Relationship

- Sometimes, one class is an extension of another class.
- inheritance is an **is-a relationship**. We use inheritance only if an **is-a relationship** is present between the two classes.
- It is just like saying that “A is type of B”.
  - For example
  - is “Apple is a fruit”, “Ferrari is a car”.
  - A car is a vehicle.
  - Orange is a fruit.
  - A surgeon is a doctor.
  - A dog is an animal.

Inheritance

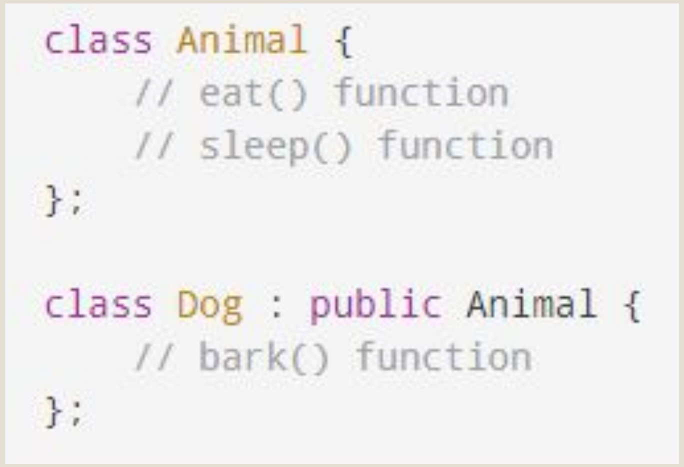
- The technique of deriving a new class from an old one is called inheritance
- Capability of a class to derive properties and characteristics from another class.
- The extended (or child) class contains all the features of its base (or parent) class, and may additionally have some unique features of its own.



Inheritance cont.

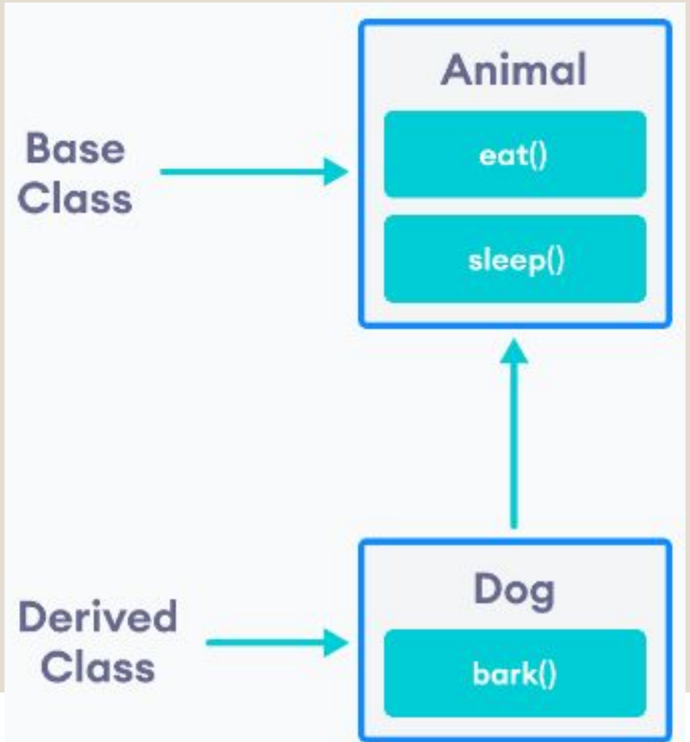
- **Sub Class:** The class that inherits properties from another class is called Sub class or Derived Class.
  Super Class: The class whose properties are inherited by sub class is called Base Class or Super class.

For example,



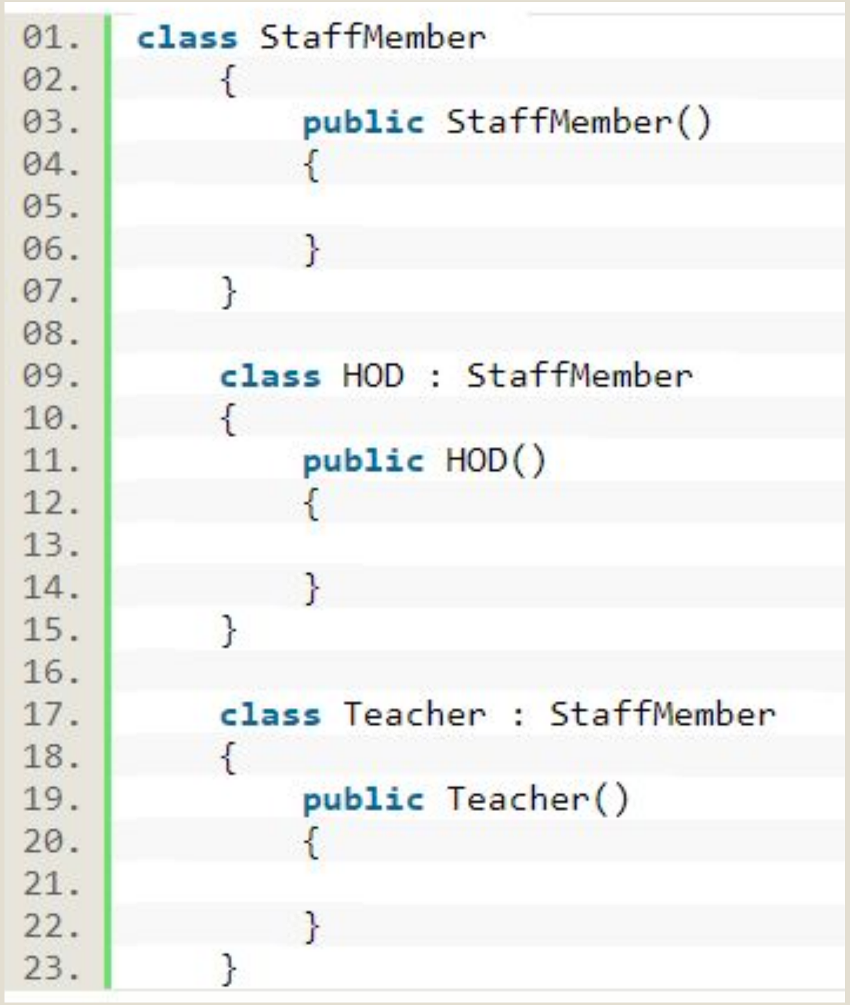
```
class Animal {
    // eat() function
    // sleep() function
};

class Dog : public Animal {
    // bark() function
};
```



Inheritance - Real World Scenario

- HOD is a staff member of college.
- All teachers are staff member of college.



```
class StaffMember
    {
        public StaffMember()
        {

        }
    }

    class HOD : StaffMember
    {
        public HOD()
        {

        }
    }

    class Teacher : StaffMember
    {
        public Teacher()
        {

        }
    }
```

Example

```
#include <bits/stdc++.h>
using namespace std;
  //Base class
class Parent
{    public:
      int id_p;
};
 // Sub class inheriting from Base Class(Parent)
class Child : public Parent
{
     public:
      int id_c; };
```

```
//main function
int main()
   {

       Child obj1;

       // An object of class child has all data members
       // and member functions of class parent
       obj1.id_c = 7;
       obj1.id_p = 91;
       cout << "Child id is " <<  obj1.id_c << endl;
       cout << "Parent id is " <<  obj1.id_p << endl;

       return 0;
   }
```

Example

```
// C++ program to demonstrate
inheritance
#include <iostream>
using namespace std;
// base class
class Animal {
 public:
   void eat()
  { cout << "I can eat!" << endl; }
void sleep()
  { cout << "I can sleep!" << endl; }
};
```

```
// derived class
class Dog : public Animal {
 public:
       void bark()
       { cout << "I can bark! Woof
woof!!" << endl; }
};
```

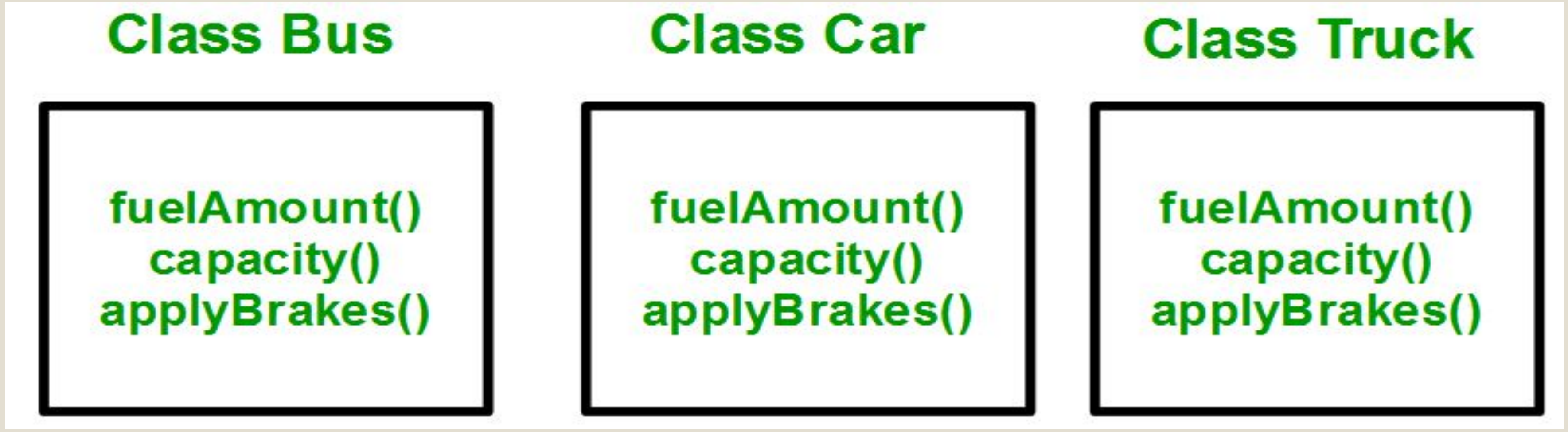
```
int main() {
// Create object of the Dog class Dog
dog1;
// Calling members of the base class
dog1.eat();
dog1.sleep();
// Calling member of the derived class
dog1.bark();
return 0;
}
```

Output

```
I can eat!
I can sleep!
I can bark! Woof woof!!
```

Why and when to use inheritance?

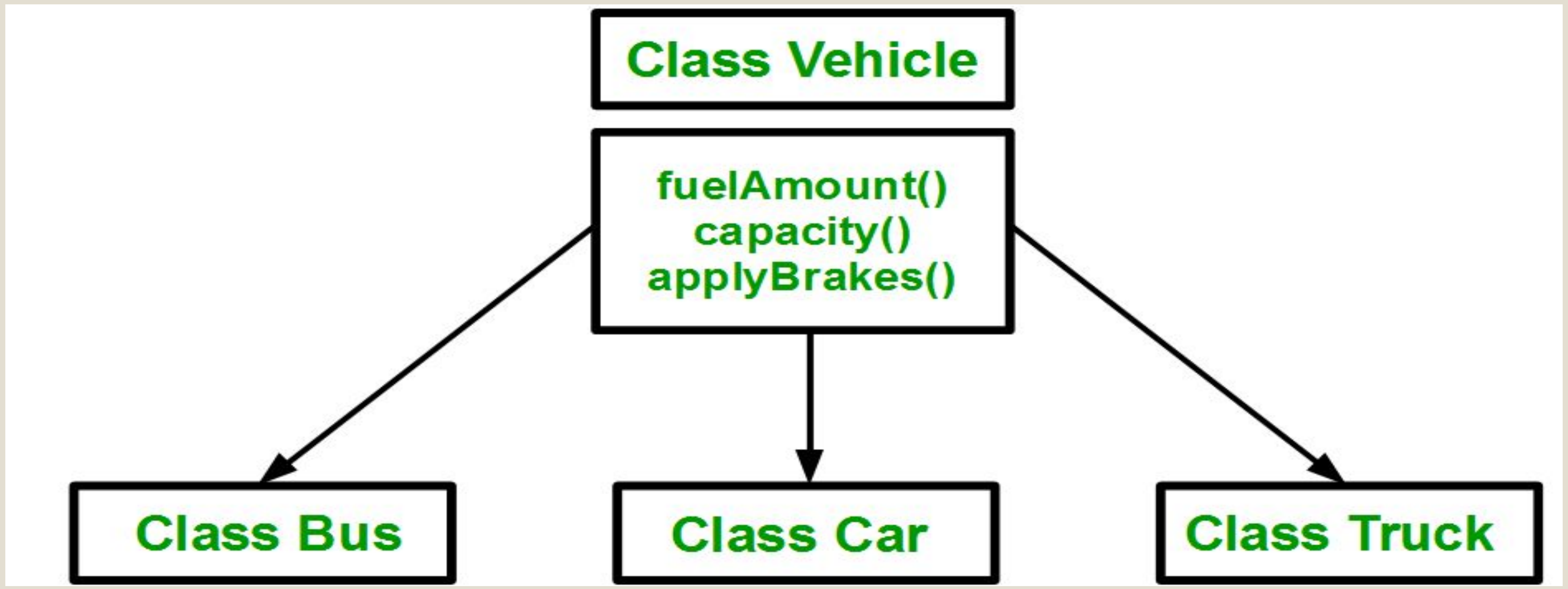
- Consider a group of vehicles. You need to create classes for Bus, Car and Truck. The methods fuelAmount(), capacity(), applyBrakes() will be same for all of the three classes.



Why and when to use inheritance?

- You can clearly see that above process results in duplication of same code 3 times. This increases the chances of error and data redundancy. To avoid this type of situation, inheritance is used.
- If we create a class Vehicle and write these three functions in it and inherit the rest of the classes from the vehicle class, then we can simply avoid the duplication of data and increase re-usability.

Why and when to use inheritance?



Implementing inheritance in C++

- Syntax:

```
class subclass_name : access_mode base_class_name
{
//body of subclass
};
```

Modes of Inheritance

- **Public mode**: If we derive a sub class from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in derived class.
- **Protected mode**: If we derive a sub class from a Protected base class. Then both public member and protected members of the base class will become protected in derived class.
- **Private mode**: If we derive a sub class from a Private base class. Then both public member and protected members of the base class will become Private in derived class.

Mode of Inheritance

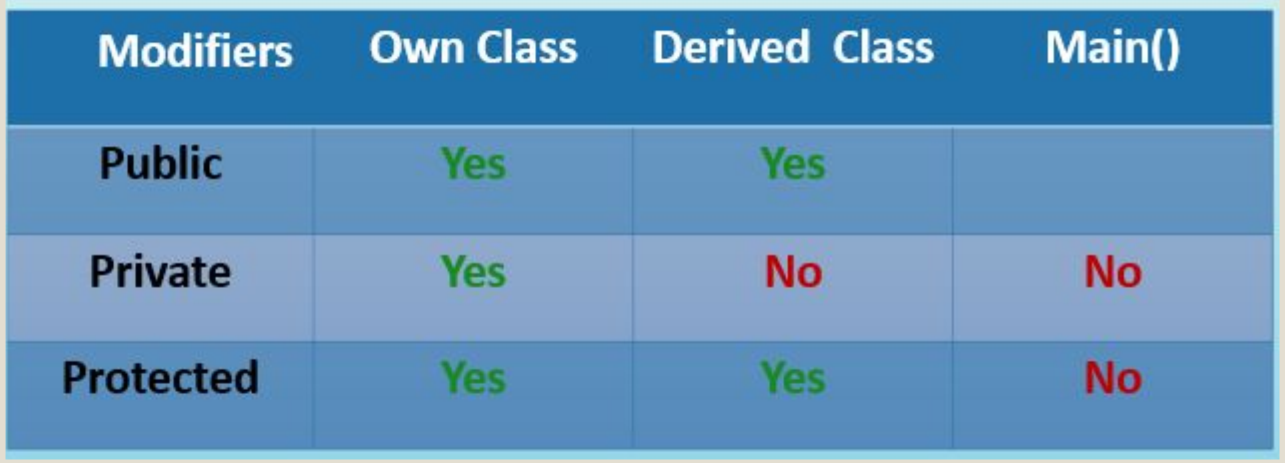
```
class A
{
public:
    int x;
protected:
    int y;
private:
    int z;
};
```

```
class B : public A
{
    // x is public
    // y is protected
    // z is not
accessible from B
};
```

```
class C : protected
A
{
    // x is protected
    // y is protected
    // z is not
accessible from C
};
```

```
class D : private
A    // 'private' is
default for classes
{
    // x is private
    // y is private
    // z is not
accessible from D
};
```

Mode of Inheritance



Modifiers	Own Class	Derived Class	Main()
Public	Yes	Yes	
Private	Yes	No	No
Protected	Yes	Yes	No

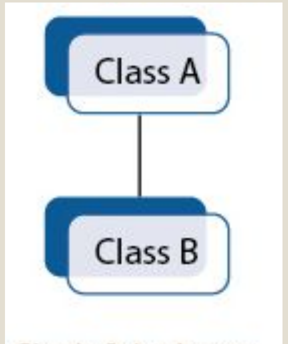
Week Six– Class Two

Types of Inheritance in C++

1. Single Inheritance
2. Multiple Inheritance
3. Multilevel Inheritance
4. Hierarchical Inheritance
5. Hybrid (Virtual) Inheritance

Single Inheritance

- In single inheritance, a class is allowed to inherit from only one class. i.e. one sub class is inherited by one base class only.



Single Inheritance

Single Inheritance (Continue..)

Syntax:

class subclass_name : access_mode base_class

{

//body of subclass

};

```
#include <iostream>
using namespace std;

// base class
class Vehicle {
public:
        Vehicle()
        {
        cout << "This is a Vehicle" << endl;
        }
};
```

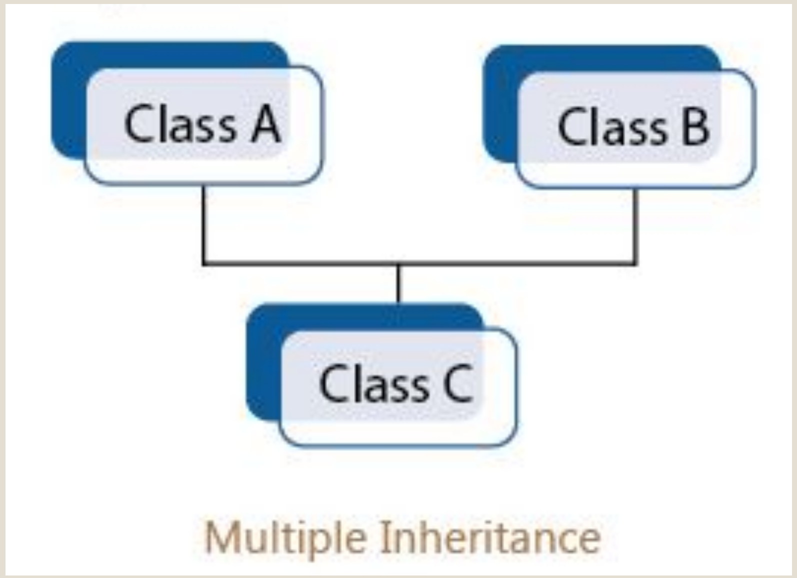
```
// sub class derived from two base classes
class Car: public Vehicle{

};

// main function
int main()
{
        // creating object of sub class will
        // invoke the constructor of base classes
        Car obj;
        return 0;
}
```

Multiple Inheritance

- Multiple Inheritance is a feature of C++ where a class can inherit from more than one classes. i.e one **sub class** is inherited from more than one **base classes**.



Multiple Inheritance (Continue..)

◦ Syntax:

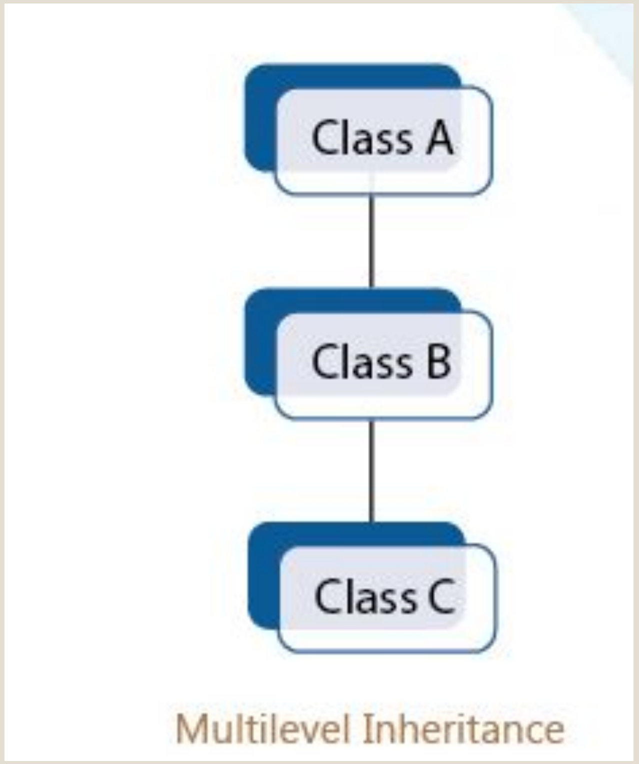
class subclass_name : access_mode base_class1, access_mode base_class2, ...{//body of subclass};

```
class stud {
  protected:
    int roll, m1, m2;
  public:
      void get()
      { cout << "Enter the Roll No.: ";
      cin >> roll;
      cout << "Enter the two highest marks: ";
      cin >> m1 >> m2; } };
class extracurriculam {
    protected:
      int xm;
    public:
      void getsm()
     { cout << "\nEnter the mark for Extra Curriculam
Activities: "; cin >> xm; } }
```

```
class output : public stud, public extracurriculam {    int tot, avg;
public:
      void display() {
           tot = (m1 + m2 + xm);
           avg = tot / 3;
           cout << "\n\n\tRoll No : " << roll <<  "\n\tTotal : " << tot;
           cout << "\n\tAverage : " << avg; } };
int main() {
     output O;
     O.get();
     O.getsm();
     O.display(); }
```

Multilevel Inheritance

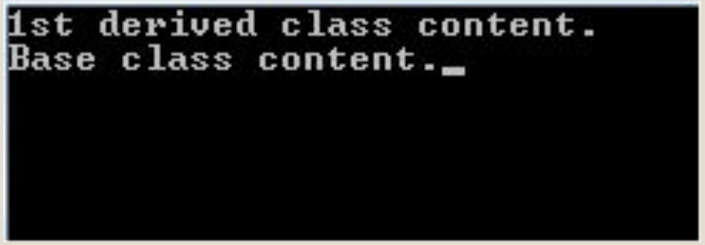
- In this type of inheritance, a derived class is created from another derived class.



Multilevel Inheritance (Continue..)

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

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

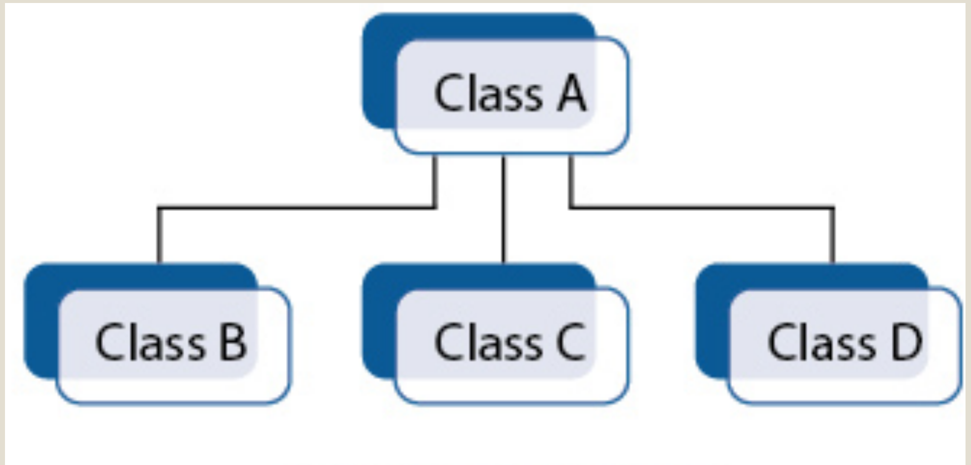


```
1st derived class content.
Base class content._
```

Week Six– Class Three

Hierarchical Inheritance

In this type of inheritance, more than one sub class is inherited from a single base class. i.e. more than one derived class is created from a single base class.



Hierarchical Inheritance

Hierarchical Inheritance (Continue..)

```
include <iostream>
#include <string.h>
using namespace std;
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;
    }};
```

```
class stud : public member {
    char level[20];

public:
    void getdata()
    {
        member::get();
        cout << "Class: "; cin >> level;
    }
    void disp2()
    {
        member::disp();
        cout << "Level: " << level << endl;
    }
};
```

```
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;
    cout << "Student" << endl;
    cout << "Enter data" << endl;
    s.getdata();
    cout << endl
        << "Displaying data" << endl;
    s.disp();
    cout << endl
        << "Staff Data" << endl;
    cout << "Enter data" << endl;
    S.getdata();
    cout << endl
        << "Displaying data" << endl;
    S.disp();
}
```

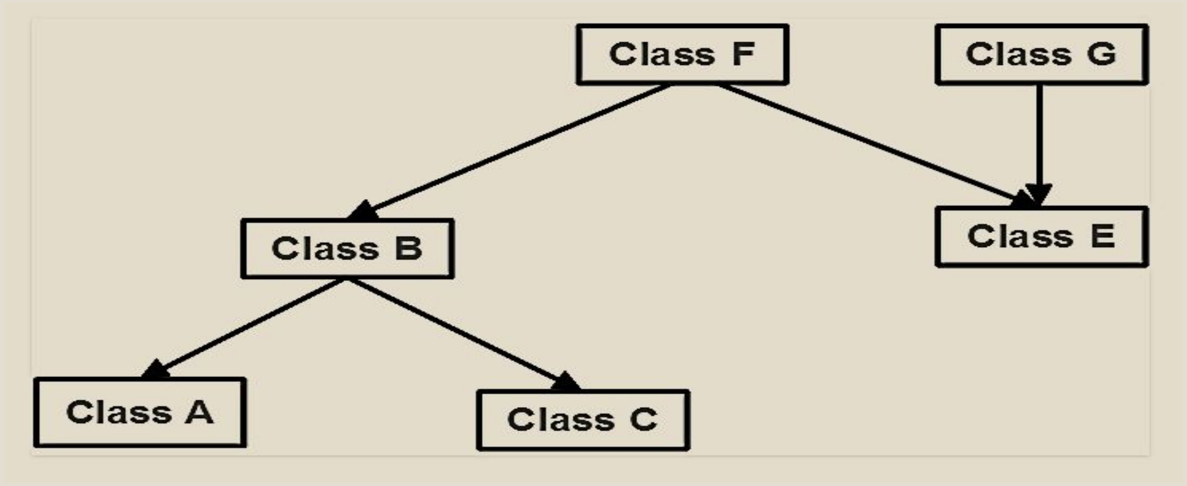
Output

```
Student
Enter data
Age: 12
Gender: Female
Class: 10
Displaying data
Age: 12
Gender: Female
```

```
Staff Data
Enter data
Age: 12
Gender: male
Salary: Rs.100000
Displaying data
Age: 12
Gender: male
```

Hybrid (Virtual) Inheritance

- Hybrid Inheritance is implemented by combining more than one type of inheritance. For example: Combining Hierarchical inheritance and Multiple Inheritance.



Hybrid Inheritance (Continue..)

```
// base class
class Vehicle
{
 public:
   Vehicle()
   {
    cout << "This is a Vehicle" << endl;
   }
};
//base class
class Fare
{
   public:
   Fare()
   {
      cout<<"Fare of Vehicle\n";
   } };
```

```
// first sub class
class Car: public Vehicle
{

};
// second sub class
class Bus: public Vehicle, public Fare
{

};
```

```
// main function
int main()
{
    // creating object of sub class will
    // invoke the constructor of base class
    Bus obj2;
    return 0;
}
```

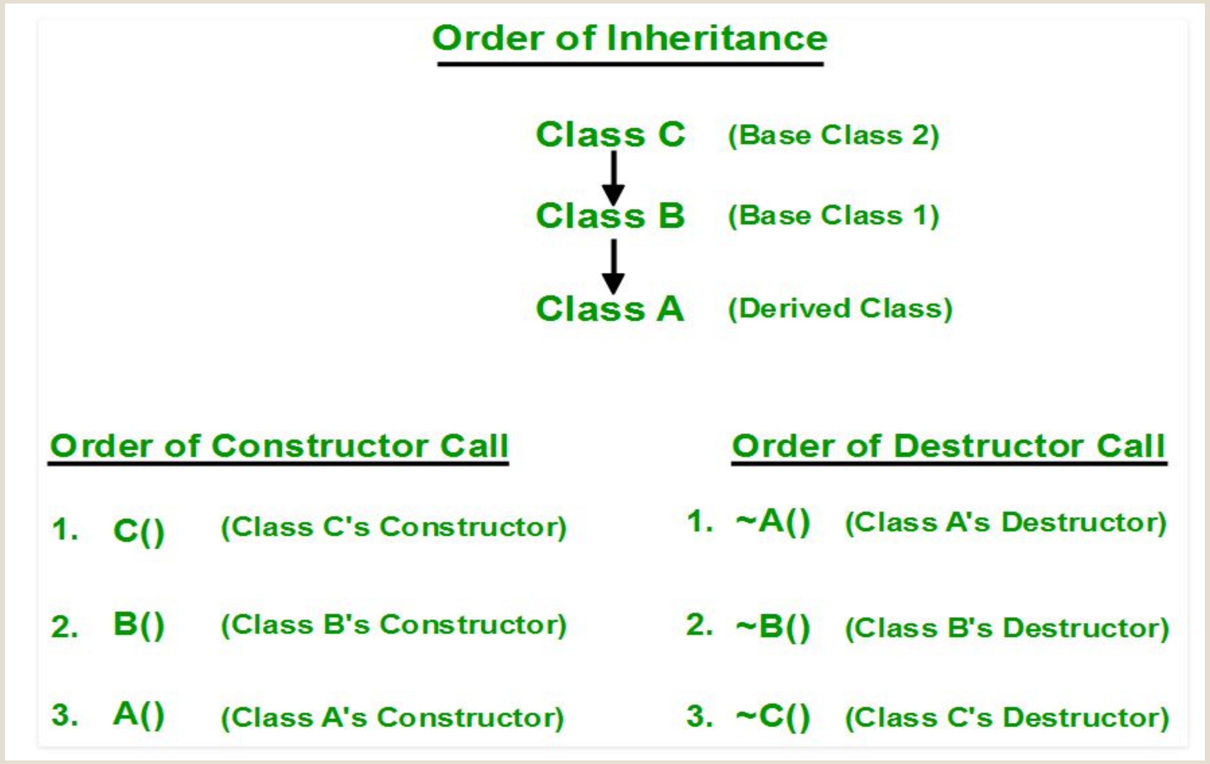
Output

```
This is a Vehicle
Fare of Vehicle
```

Order of constructor and Destructor call

- Whenever we create an object of a class, the default constructor of that class is invoked automatically to initialize the members of the class.
  If we inherit a class from another class and create an object of the derived class, it is clear that the default constructor of the derived class will be invoked but before that the default constructor of all of the base classes will be invoke, i.e the order of invokation is that the base class's default constructor will be invoked first and then the derived class's default constructor will be invoked.

Order of constructor and Destructor call


Order of Inheritance
Class C (Base Class 2)
Class B (Base Class 1)
Class A (Derived Class)
Order of Constructor Call
1. C() (Class C's Constructor)
2. B() (Class B's Constructor)
3. A() (Class A's Constructor)
Order of Destructor Call
1. ~A() (Class A's Destructor)
2. ~B() (Class B's Destructor)
3. ~C() (Class C's Destructor)

Example

```
#include <iostream>
using namespace std;

// base class
class Parent
{
   public:

   // base class constructor
   Parent()
   {
      cout << "Inside base class" << endl;
   }
};
```

```
// sub class
class Child : public Parent
{
   public:

   //sub class constructor
   Child()
   {
      cout << "Inside sub class" << endl;
   }
};
```

```
// main function
int main() {

   // creating object of sub class
   Child obj;

   return 0;
}
```

Output

```
Inside base class
Inside sub class
```

Concept: Base class Default Constructor in Derived class Constructors!

Example

```
class Base
{
    int x;
    public:
    // default constructor
    Base()
    {
        cout << "Base default constructor\n";
    }
};
```

```
class Derived : public Base
{
    int y;
    public:
    // default constructor
    Derived()
    {
        cout << "Derived default
constructor\n";
    }
    // parameterized constructor
    Derived(int i)
    {
        cout << "Derived parameterized
constructor\n";
    }
};
```

```
int main()
{
    Base b;
    Derived d1;
    Derived d2(10);
}
```

Output

```
Base default constructor
Base default constructor
Derived default constructor
Base default constructor
Derived parameterized constructor
```

Concept: Calling parameterized constructor of base class in derived class constructor!

- To call the parameterized constructor of base class when derived class's parameterized constructor is called, you have to explicitly specify the base class's parameterized constructor in derived class

Concept: Calling parameterized constructor of base class in derived class constructor!

Example

```
class Base
{
    int x;
    public:
    // parameterized constructor
    Base(int i)
    {
        x = i;
        cout << "Base Parameterized
Constructor\n";
    }
};
```

```
class Derived : public Base
{
    int y;
    public:
    // parameterized constructor
    Derived(int j):Base(j)
    {
        y = j;
        cout << "Derived Parameterized
Constructor\n";
    }
};
```

```
int main()
{
    Derived d(10) ;
}
```

Output

```
Base Parameterized Constructor
Derived Parameterized Constructor
```

Important Points

- Whenever the derived class’s default constructor is called, the base class’s default constructor is called automatically.
- To call the parameterized constructor of base class inside the parameterized constructor of sub class, we have to mention it explicitly.