Course Title:

Software Development Fundamentals-2

DEPARTMENT

OF

COMPUTER SCIENCE AND ENGINEERING
&
INFORMATION TECHNOLOGY

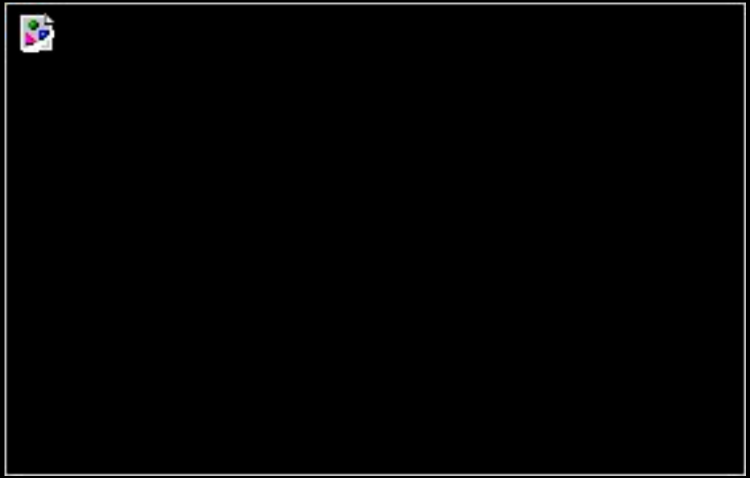
Basic concept of Object Oriented Concept Attributes of Object Oriented Programming

- Class
- Object
- Encapsulation and Data hiding
- Data abstraction
- Inheritance
- Polymorphism
- Dynamic Binding
- Message Passing

Inheritance

- Inheritance is the process by which objects of one class acquire the properties of object of another class.
- The capability of a class to derive properties and characteristics from another class is called **Inheritance**. Inheritance is one of the most important feature of Object Oriented Programming.

Super Class-----

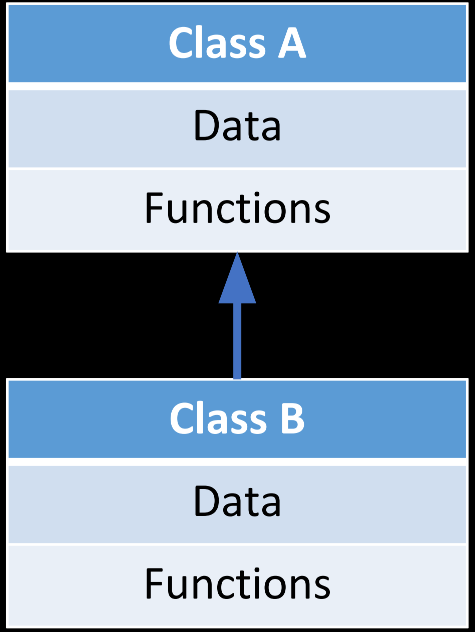


Sub Class------

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.

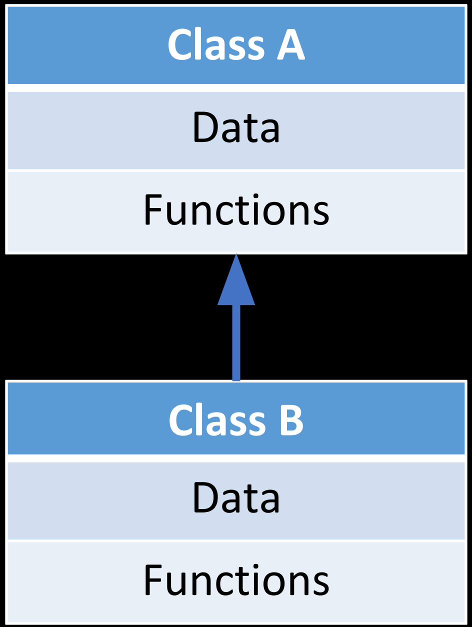
Inheritance

- **Inheritance** is a way to include functionalities of a parent class to a child class.
- It doesn't allow inheritance of private members from parent class.
- It enhances code reusability.



Inheritance

- **Polymorphism** enables to define many forms of a function and it is invoked based on the object calling it.
- The functionality of a function differs according to the object that calls it.
- Inheritance helps to implement polymorphism.



Class A

Data

Functions

Class B

Data

Functions

Advantages of inheritance

- When a class inherits from another class, there are three benefits:
- (1) You can *reuse* the methods and data of the existing class

  (2) You can *extend* the existing class by adding new data and new methods

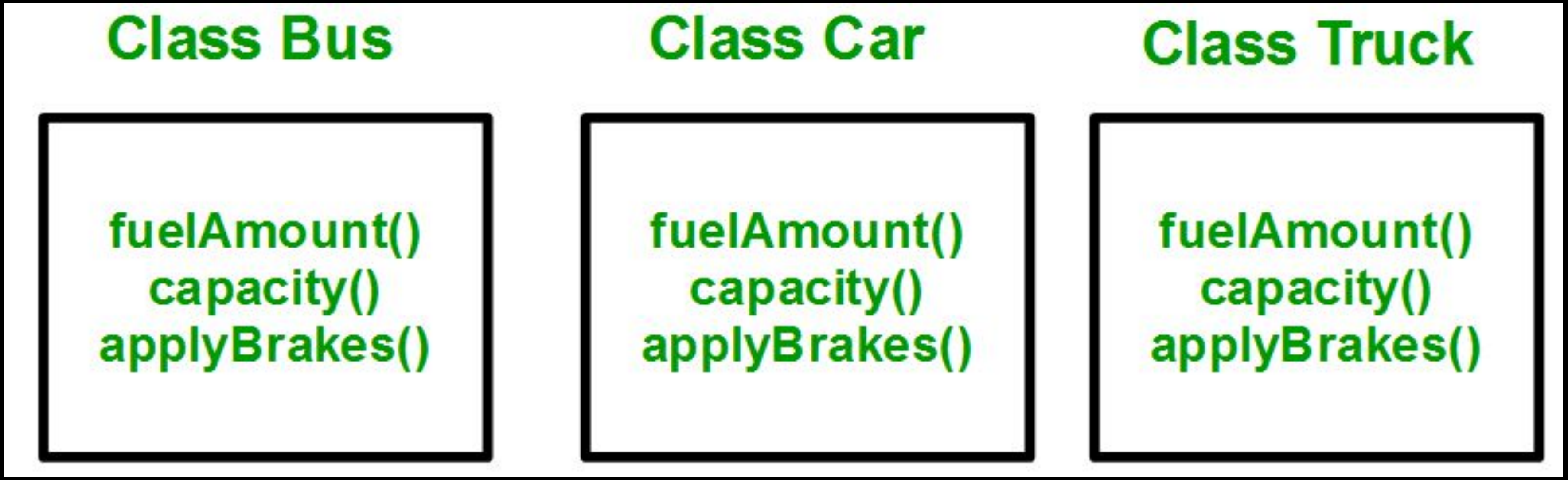
  (3) You can *modify* the existing class by overloading its methods with your own implementations

Inheritance

- Why and when to use inheritance?
- Modes of Inheritance
- Types of Inheritance

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.



Class Bus

fuelAmount()
capacity()
applyBrakes()

Class Car

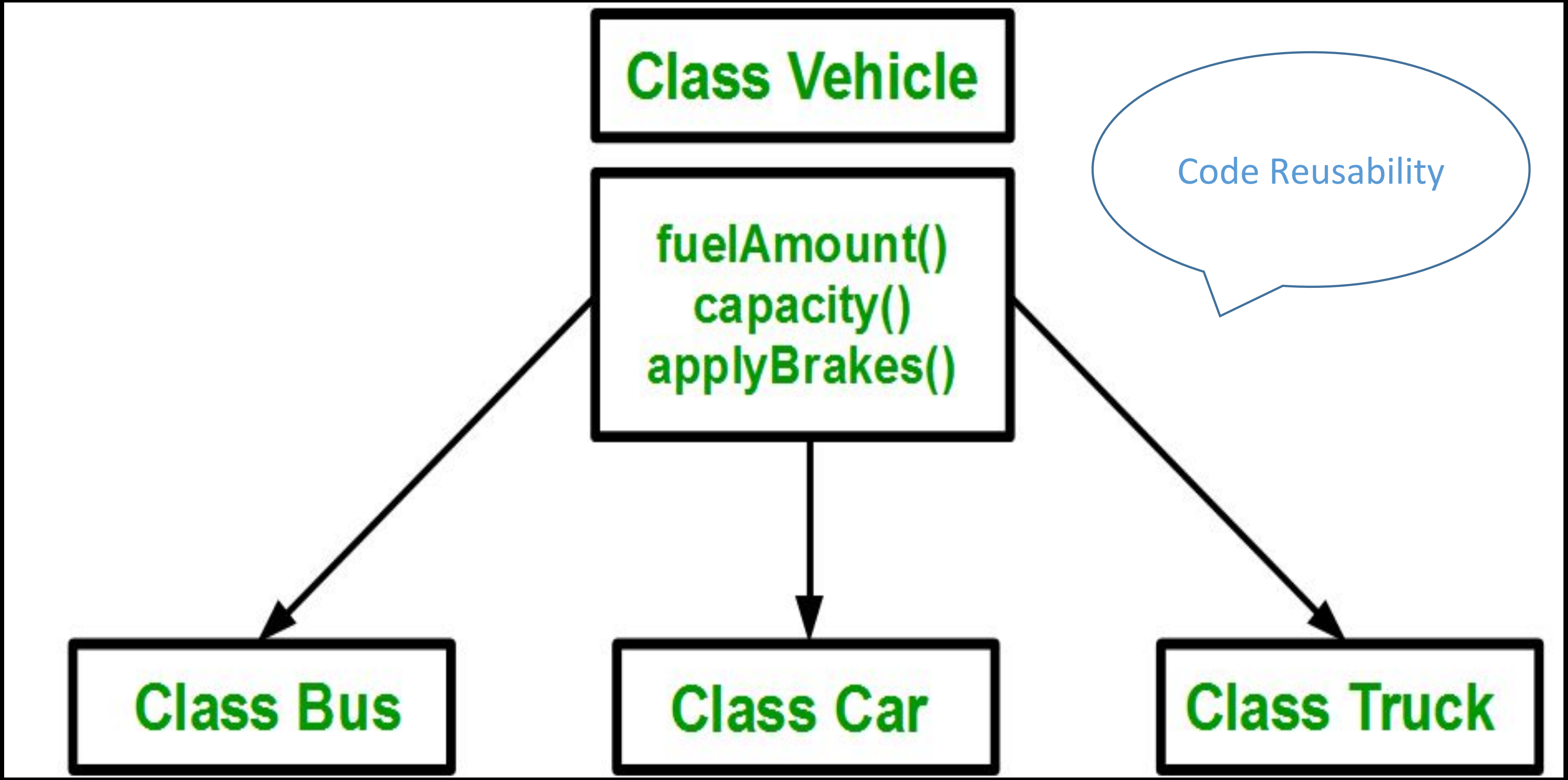
fuelAmount()
capacity()
applyBrakes()

Class Truck

fuelAmount()
capacity()
applyBrakes()

Without Inheritance

Why and when to use inheritance?



Code Reusability

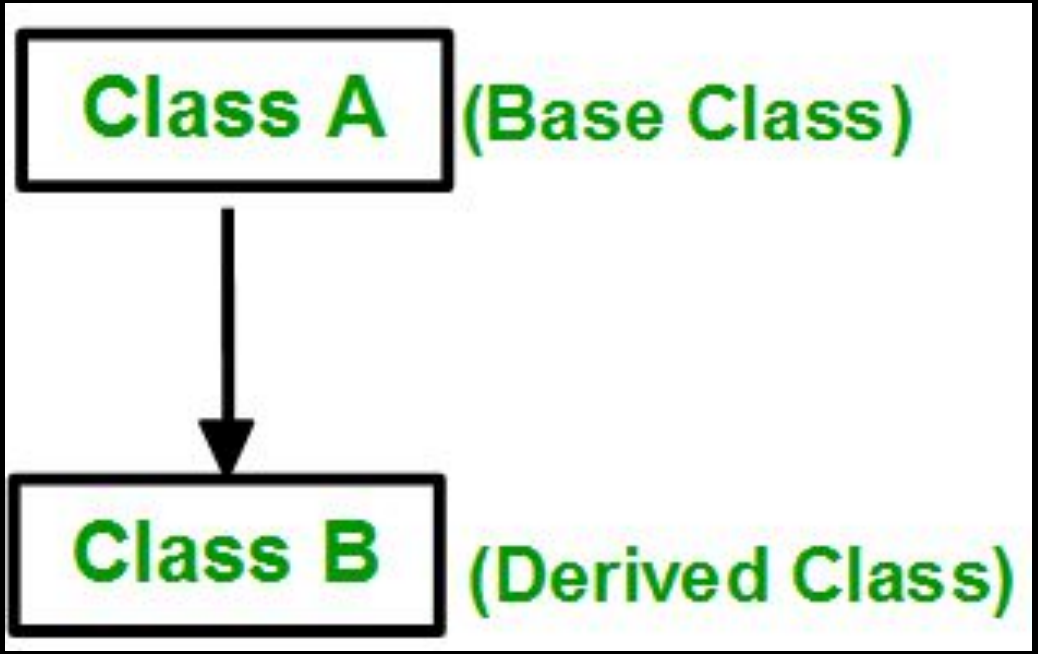
With Inheritance

Modes of Inheritance

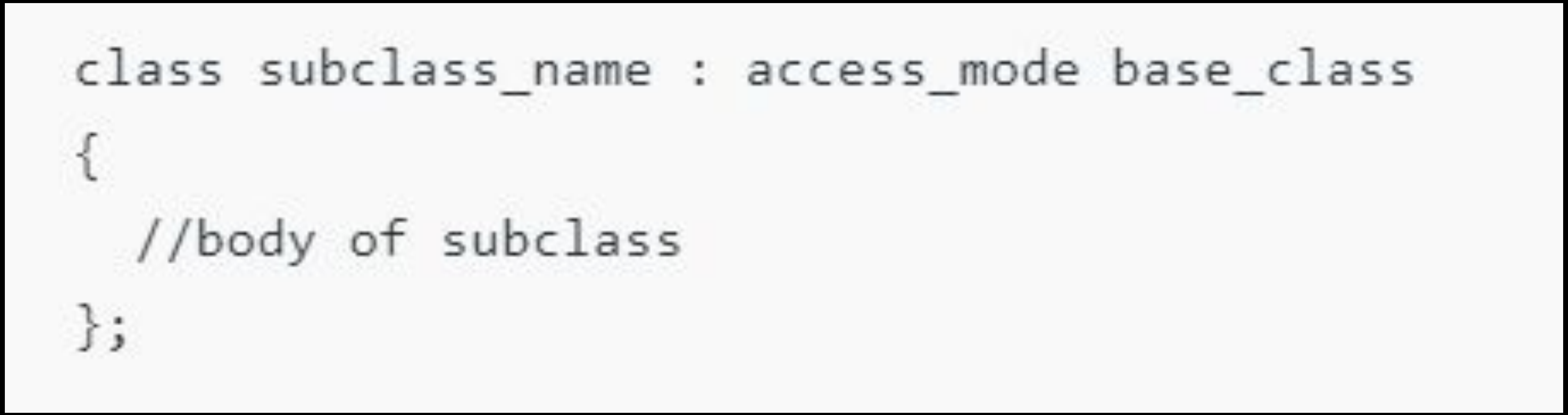
- **Public mode**: If we derive a sub class from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in derived class. Private members of the base class will never get inherited in sub class.
- **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 members of the base class will never get inherited in sub 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. Private members of the base class will never get inherited in sub class.

Types of 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.



Types of Inheritance: **Single Inheritance**



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

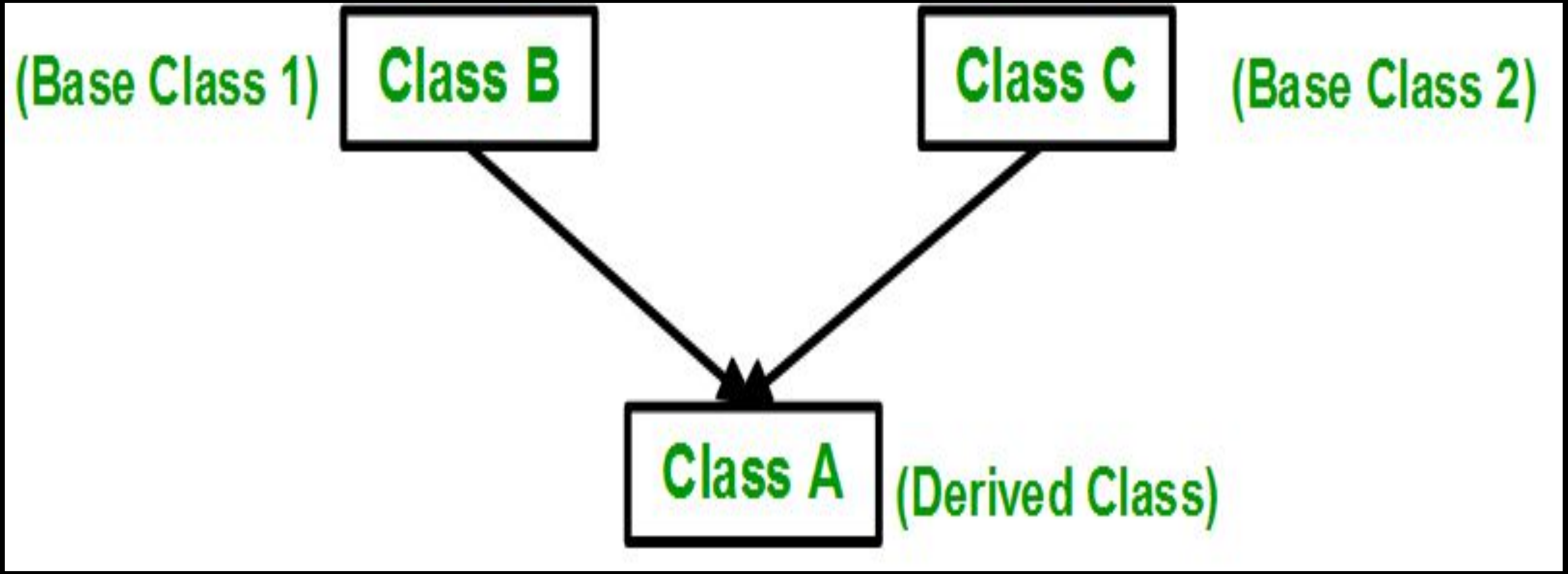
Types of Inheritance: **Single Inheritance**

```
// C++ program to explain
// Single inheritance
#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;
}
```

Types of Inheritance- **Multiple Inheritance**

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 Example Program

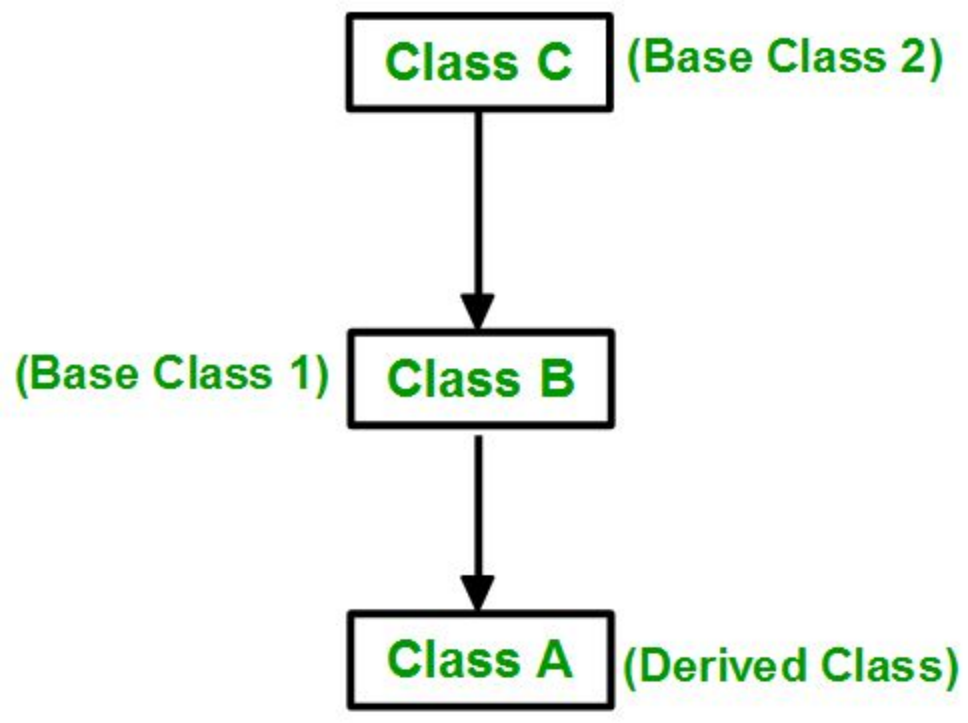
```
#include <iostream>
using namespace std;
class Mammal {
        public:
              Mammal()
              {
                   cout << "Mammals can give direct birth." << endl;
              }
};
class WingedAnimal {
         public:
WingedAnimal()
   {
     cout << "Winged animal can flap." << endl;
   }
};
class Bat: public Mammal, public WingedAnimal { };
int main()
{
   Bat b1;
   return 0;
}
```

Output

```
Mammals can give direct birth.
Winged animal can flap.
```

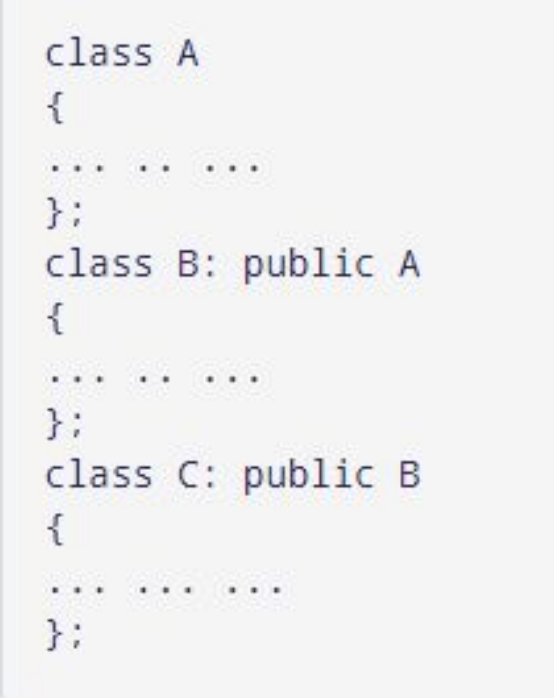
Types of Inheritance- **Multilevel Inheritance**

Multilevel Inheritance: In this type of inheritance, a derived class is created from another derived class.



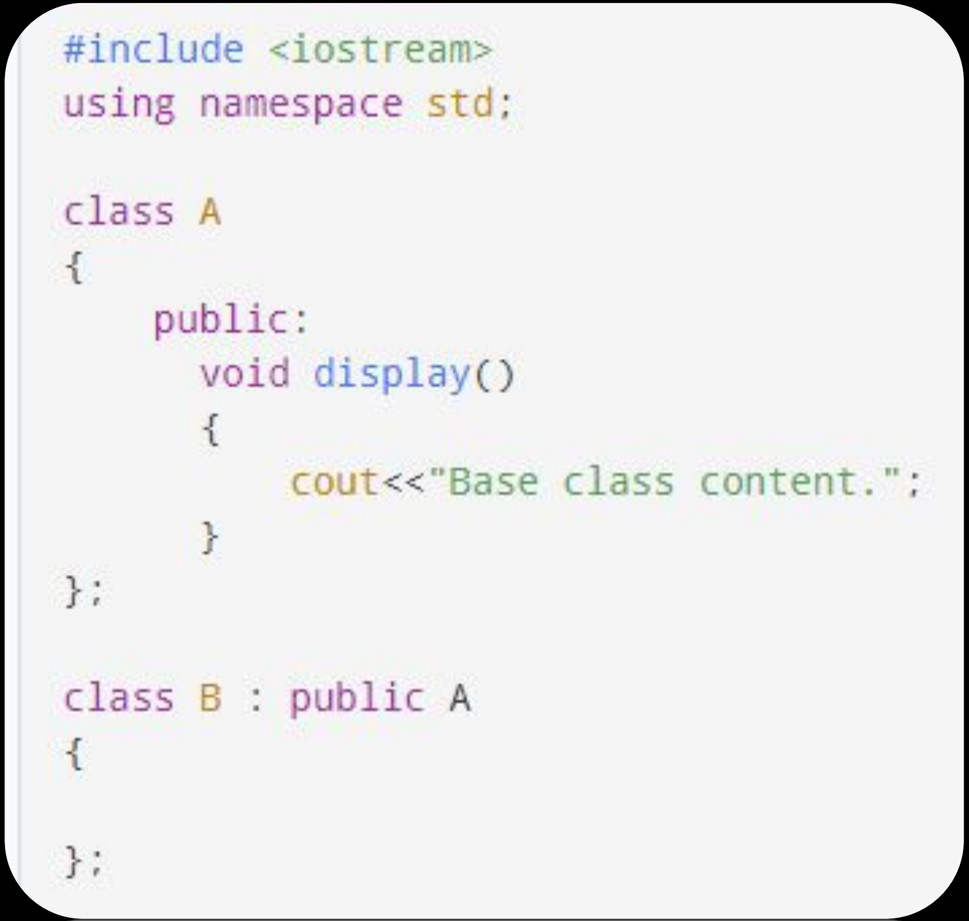
Types of Inheritance: **Multilevel Inheritance**

- In C++ programming, not only you can derive a class from the base class but you can also derive a class from the derived class. This form of inheritance is known as multilevel inheritance.



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

Multilevel Inheritance: Example

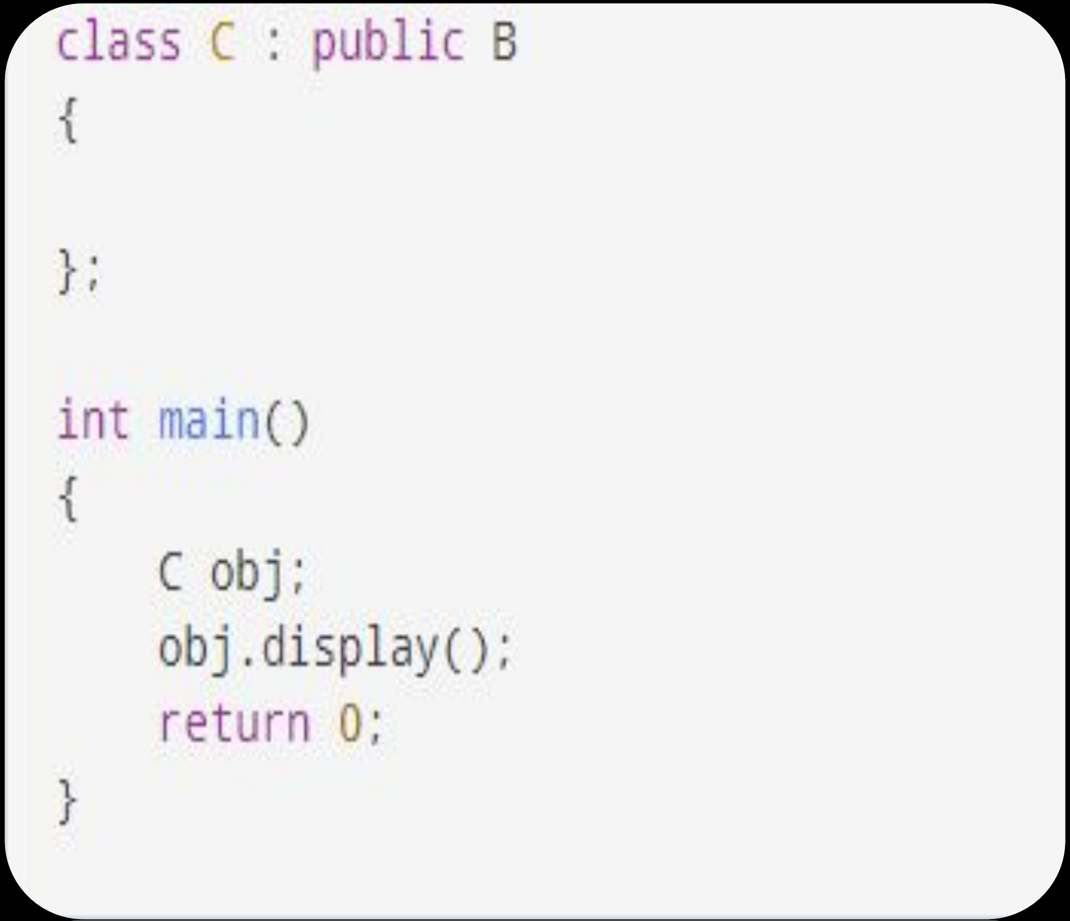


```
#include <iostream>
using namespace std;

class A
{
    public:
      void display()
      {
          cout<<"Base class content.";
      }
};

class B : public A
{

};
```



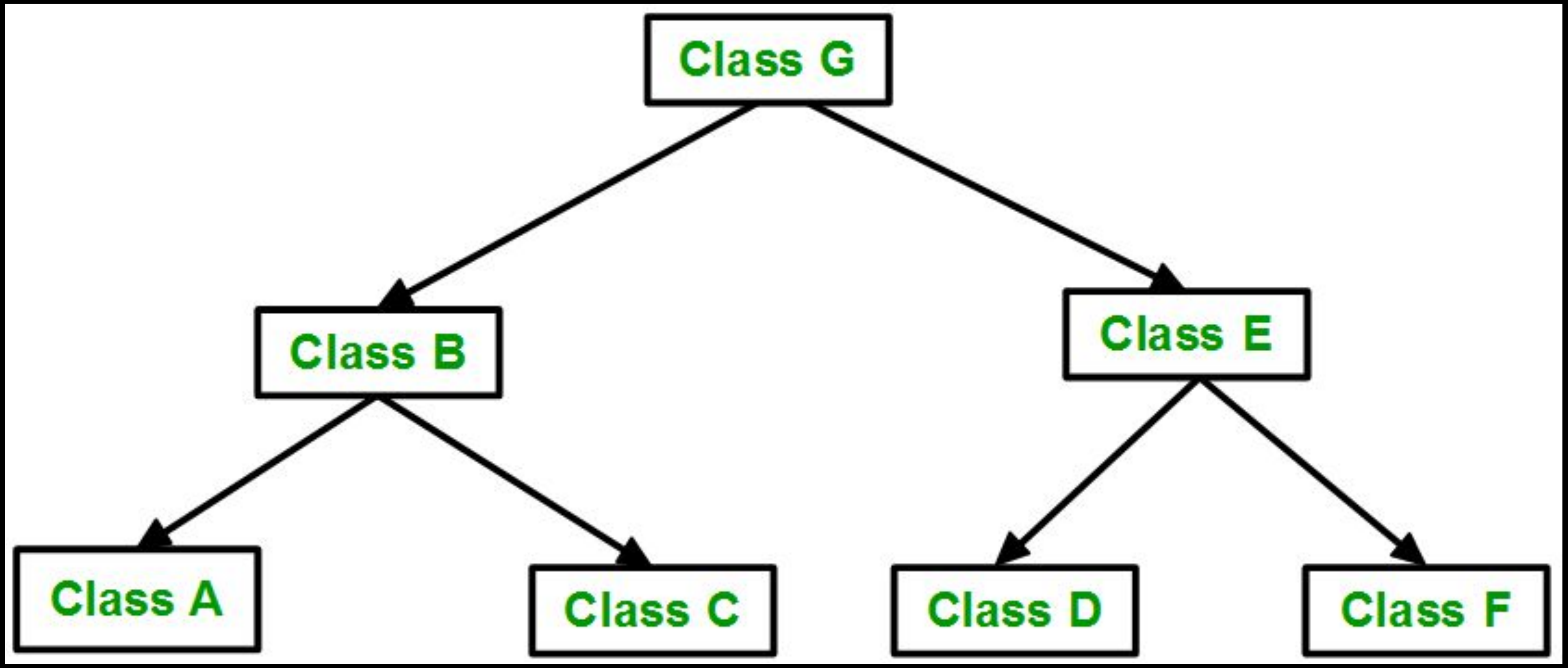
```
class C : public B
{


};

int main()
{
    C obj;
    obj.display();
    return 0;
}
```

Types of Inheritance

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.



Types of Inheritance

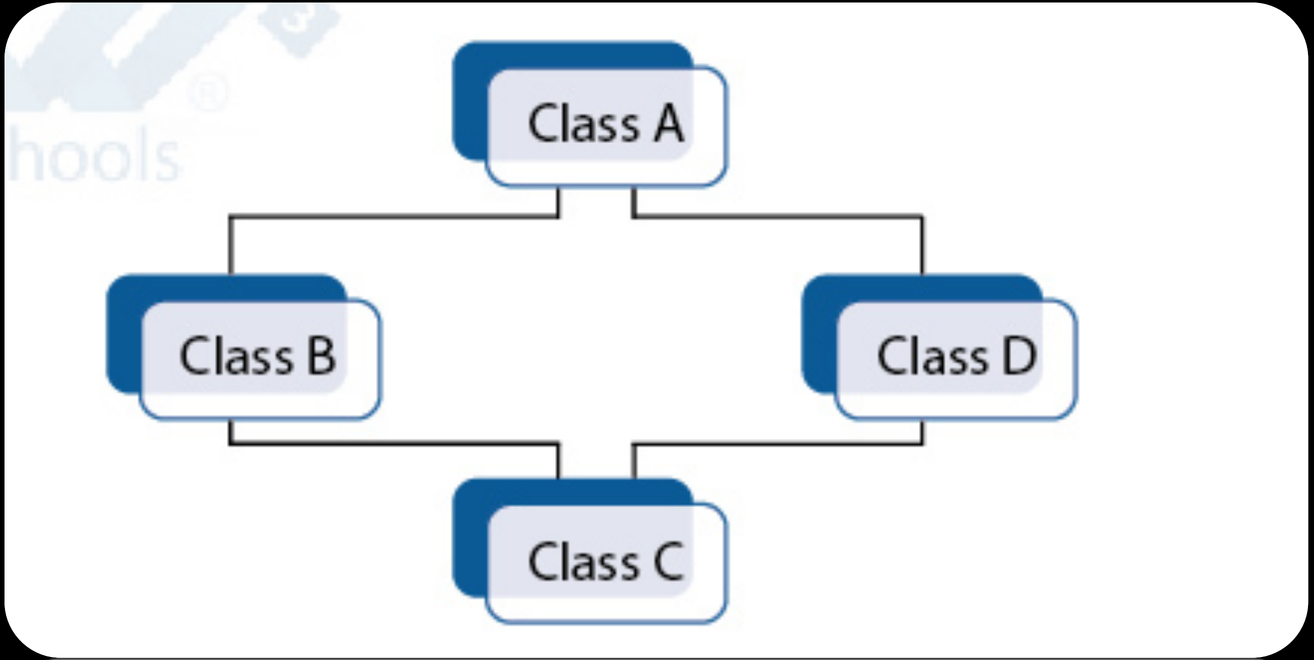
```
// C++ program to implement   Hierarchical Inheritance
#include <iostream>
using namespace std;
class Vehicle
{
  public:
    Vehicle()
    {
      cout << "This is a Vehicle" << endl;
    }
};
// first sub class
class Car: public Vehicle
{ };
// second sub class
class Bus: public Vehicle
{   };
// main function
int main()
{
    // creating object of sub class will
    // invoke the constructor of base class
    Car obj1;
    Bus obj2;
    return 0;
}
```

Output:

```
This is a Vehicle
This is a Vehicl
```

Types of Inheritance

Hybrid Inheritance: Hybrid Inheritance is implemented by combining more than one type of inheritance.



Access Specifier in Class Declaration

```
Class sample
{
Private:
//Visible to member function within its class but not in derived class
Protected:
//Visible to member function within its class and derived class
Public:
//visible to member functions within its class, derived classed and through object
};
```

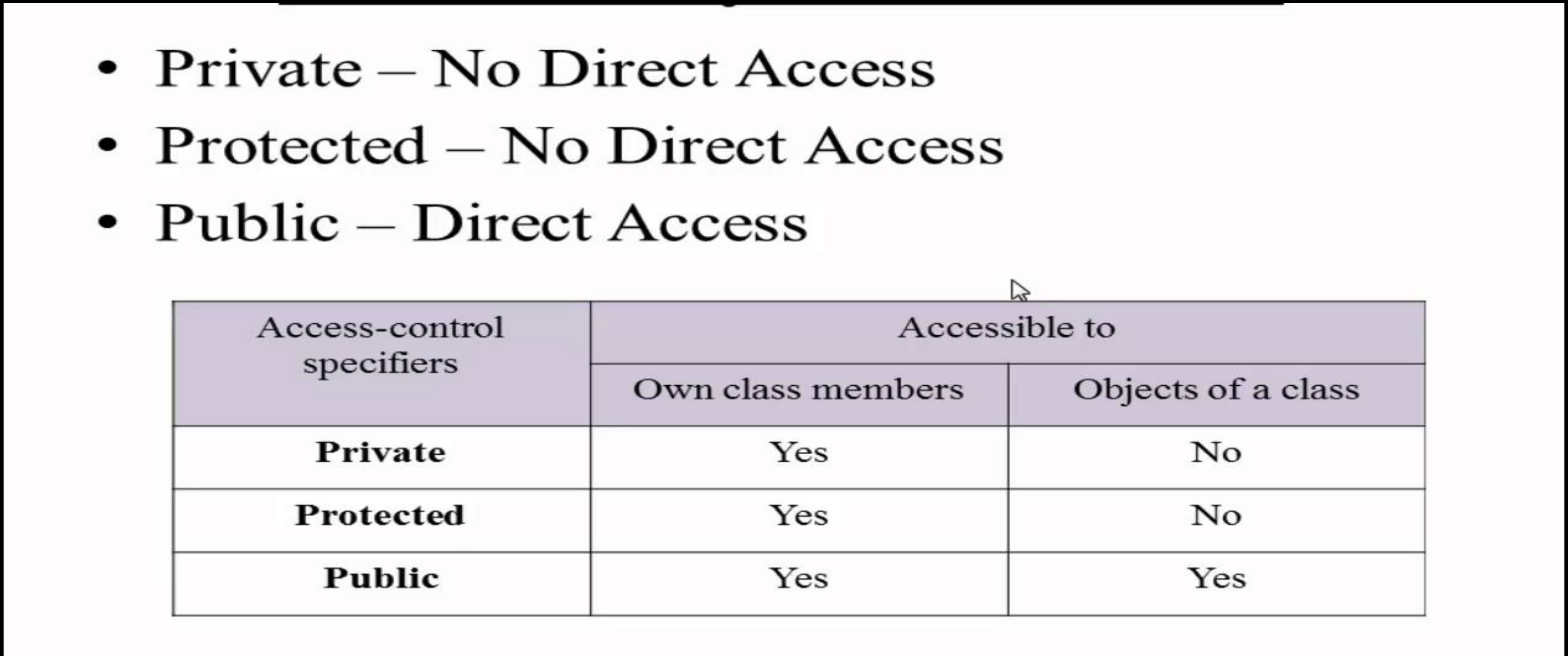
Access Specifier

Access modifiers or Access Specifiers in a class are used to set the accessibility of the class members. That is, it sets some restrictions on the class members not to get directly accessed by the outside functions.

There are 3 types of access modifiers available in C++:

- **Public**
- **Private**
- **Protected**

Access Specifier



- Private – No Direct Access
- Protected – No Direct Access
- Public – Direct Access

Access-control specifiers	Accessible to	
	Own class members	Objects of a class
Private	Yes	No
Protected	Yes	No
Public	Yes	Yes

INHERITANCE

HOW TO DECLARE DERIVED CLASS

Class *Derived_class_name* : Visiblity Mode *Base_class_name*

{

}

Public
Private
Protected

INHERITANCE

FOLLOWING ARE THE THREE POSSILBE STYLE OF DERIVATION

Public Derivative

```
class Derived_Class_name: public Base_class_name
{



}
```

INHERITANCE

FOLLOWING ARE THE THREE POSSILBE STYLE OF DERIVATION

Private Derivative

```
class Derived_Class_name: private Base_class_name
{



}
```

INHERITANCE

FOLLOWING ARE THE THREE POSSILBE STYLE OF DERIVATION

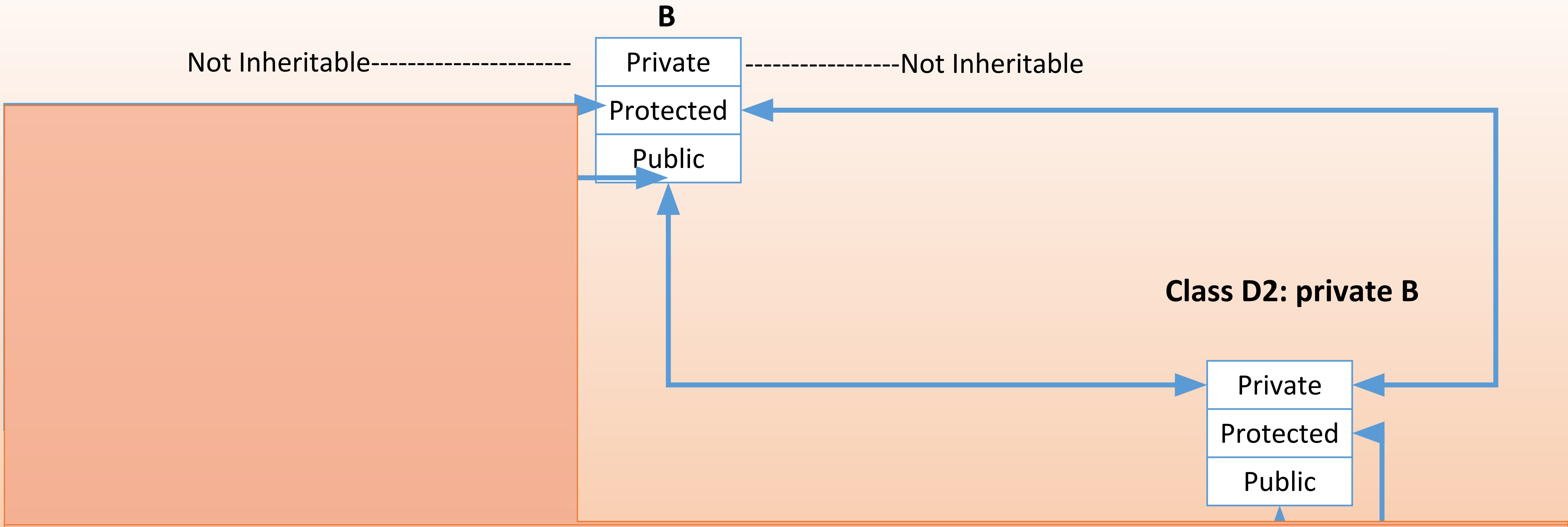
Protected Derivative

```
class Derived_Class_name: protected  Base_class_name
{


}
```

INHERITANCE

Base class member	Derivation access level	Derived class member	User of derived class
Private	Private Protected Public	Not accessible	Not accessible



B

Not Inheritable---------------------

Private

-----------------Not Inheritable

Protected

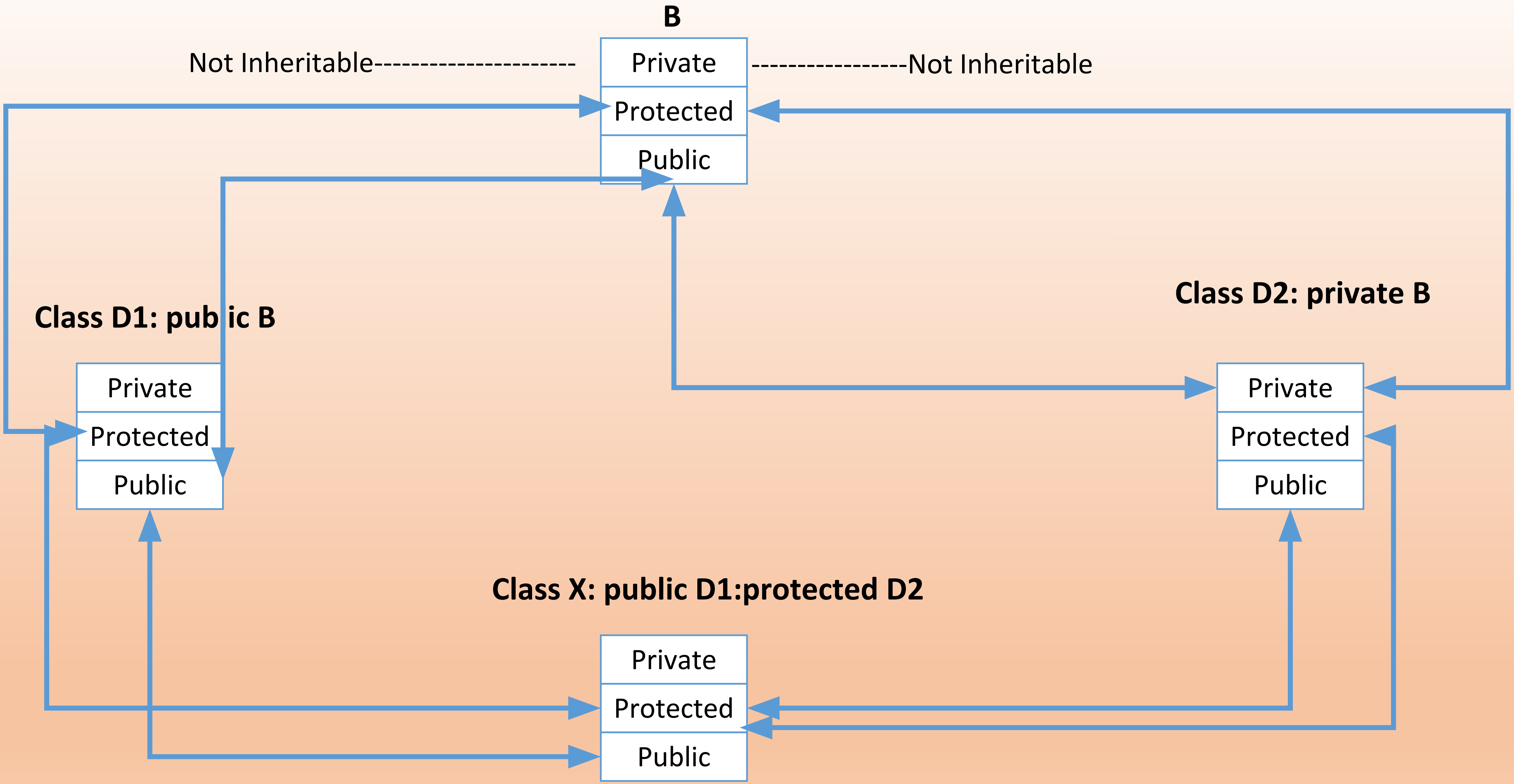
Public

Class D2: private B

Private

Protected

Public


B
Not Inheritable---------------------
Private
-----------------Not Inheritable
Protected
Public
Class D1: pub ic B
Private
Protected
Public
Class D2: private B
Private
Protected
Public
Class X: public D1:protected D2
Private
Protected
Public

Class Derivation
Constructors and Destructors

- Constructors and destructors are not inherited
  - Each derived class should define its constructors/destructor
  - If no constructor is written=> hidden constructor is generated and will call the base default constructor for the inherited portion and then apply the default initialization for any additional data members
- When a derived object is instantiated, memory is allocated for
  - Base object
  - Added parts
- Initialization occurs in two stages:
  - the base class constructors are invoked to initialize the base objects
  - the derived class constructor is used to complete the task
- The derived class constructor specifies appropriate base class constructor in the initialization list
  - If there is no constructor in base class, the compiler created default constructor used
- If the base class is derived, the procedure is applied recursively

Constructor Rules for Derived Classes

The default constructor and the destructor of the base class are always called when a new object of a derived class is created or destroyed.

```
class A {
  public:
    A ( )
      {cout<< “A:default”<<endl;}
    A (int a)
      {cout<<“A:parameter”<<endl;}
};
```

```
class B : public A
{
  public:
    B (int a)
        {cout<<“B”<<endl;}
};
```

```
B test(1);
```

output:

```
A:default
B
```

Constructor Rules for Derived Classes

You can also specify an constructor of the base class other than the default constructor

```
DerivedClassCon ( derivedClass args ) : BaseClassCon ( baseClass
args )
      {  DerivedClass constructor body   }
```

```
class A {
  public:
    A ( )
      {cout<< “A:default”<<endl;}
    A (int a)
      {cout<<“A:parameter”<<endl;}
};
```

```
class C : public A {
  public:
    C (int a) : A(a)
        {cout<<“C”<<endl;}
};
```

```
C test(1);
```

output:

```
A:parameter
C
```

Method Overriding

- As we know, inheritance is a feature of OOP that allows us to create derived classes from a base class. The derived classes inherit features of the base class.
- Suppose, the same function is defined in both the derived class and the based class. Now if we call this function using the object of the derived class, the function of the derived class is executed.
- This is known as **function overriding** in C++. The function in derived class overrides the function in base class.

Method Overriding

- To override a function you must have the same signature in child class. By signature I mean the data type and sequence of parameters. Here we don’t have any parameter in the parent function so we didn’t use any parameter in the child function.

- In function overriding, the function in parent class is called the overridden function and function in child class is called overriding function.

Requirements for Overriding a Function

1. Inheritance should be there. Function overriding cannot be done within a class. For this we require a derived class and a base class.
2. Function that is redefined must have exactly the same declaration in both base and derived class, that means same name, same return type and same parameter list.

Derived Function

Method Overriding

```
// C++ program to demonstrate function overriding

#include <iostream>
using namespace std;

class Base {
   public:
    void print() {
        cout << "Base Function" << endl;
    }
};
```

```
class Derived : public Base {
   public:
    void print() {
        cout << "Derived Function" << endl;
    }
};

int main() {
    Derived derived1;
    derived1.print();
    return 0;
}
```

Outpu: Derived Function

THANKS