C++ Programming

Trainer : Rajiv K

Email: rajiv.kamune@sunbeaminfo.com

Agenda

- **friend function**
- **Operator overloading**
- **Inheritance**
- **Mode of inheritance**

friend function

- If a function is declared as friend then **protected and private members of a class** can be used by the function.
- **friend keyword:** by using friend keyword compiler understands the function is a friend function.
- For accessing the private/protected data of a class you need to **give declaration of the friend function inside the class.**
- Friend function **do not get this pointer.**
- You can **declare main() as a friend function** but it is not recommended
- It cannot access the member names directly and has to use an object name and dot membership operator with the member name.

by using friend function we can access the private and protected data members of the class outside the class as well.

- Syntax:

```
1.    class class_name
2.    {
          //Class body
          //friend return_type function_name(parameter list);// you need to specify the function declaration with //friend keyword
3.    };
```

friend <data type> (object name); //declaration

friend function

Example:

```
class Point
{
	int x,y; //private
	public:
		Point():x(10),y(20)
		{
		}
		friend void display(Point p); friend <data type> (object name); //declaration
}
void display(Point p)
{
		cout<<“x=”<<p.x<<endl;
		cout<<“y=”<<p.y<<endl;
}
```

Operator Overloading

- operator is token in C/C++.
- It is used to generate expression.
- operator is keyword in C++.
- Types of operator:
  - Unary operator (++,--,&,!,~,sizeof())
  - Binary Operator (Arithmetic, relational, logical , bitwise, assignment)
  - Ternary operator (conditional)
- In C++, also we can not use operator with objects of user defined type directly.
- If we want to use operator with objects of user defined type then we should overload operator.
- To overload operator, we should define **operator function.**
- **We can define operator function using 2 ways:**
  - **Using member function** //inside class //no need to use friend function // only 1 argument
  - **Using non member function** //outside class //have to use friend function // 2 arguments

Need Of Operator Overloading

- we **extend the meaning of the operator**.
- If we want to **use operator** with the **object of user defined type**, then we need to **overload operator**.
- To **overload operator**, we need to **define operator function.**
- In C++, **operator is a keyword**
    - Suppose we want to use **plus(+) operator** with objects then we need to define **operator+( ) function.**

We define operator function either inside class (as a member function) or outside class (as a non-member function).	Point pt1(10,20), pt2(30,40), pt3;
	
	pt3 = pt1 + pt2; //pt3 = pt1.operator+(pt2); //using member function
	//or
	pt3 = pt1 + pt2; //pt3 = operator+(pt1, pt2); //using non member function

Operator Overloading

using member function

- **operator function must be member function**
- If we want to overload, binary operator using member function then **operator function should take only one parameter.**
  - Example : c3 = c1 + c2; //will be called as ------ c3 = c1.operator+(c2)

  Example :

```
Point operator+( Point &other ) //Member Function
        {
        Point temp;
        temp.xPos = this->xPos + other.xPos;
        temp.yPos = this->yPos + other.yPos;
        return temp;
        }
```

using non member function

- **Operator function must be global function**
- If we want to overload binary operator using non member function then **operator function should take two parameters.**
  - **Example :** c3 = c1 + c2; //will be called as ------c3 = operator+(c1,c2);

Example:

```
Point operator+( Point &pt1, Point &pt2 ) //Non Member Function
{
Point temp;
temp.xPos = pt1.xPos + pt2.xPos;
temp.yPos = pt1.yPos + pt2.yPos;
return temp;
}
```

We can not overloading following operator using member as well as non member function:

1. dot/member selection operator(.)
2. Pointer to member selection operator(.*)
3. Scope resolution operator(::)
4. Ternary/conditional operator(? :)
5. sizeof() operator
6. typeid() operator
7. static_cast operator
8. dynamic_cast operator
9. const_cast operator
10. reinterpret_cast operator

We can not overload following operators using non member function

- Assignment operator(=)
- Subscript / Index operator([])
- Function Call operator[()]
- Arrow / Dereferencing operator(→)

Inheritance

- The capability of a class to **derive properties and characteristics** from another class is called **Inheritance**.
- There must be **is-a relation** between classes to observe inheritance.
- Inheritance is a process of **creating the child class or derived class from parent class or base class**. parent class< - child class or   base class<- derived class
- We can say that child class is derived from base class.
- In other words we can say that Inheritance is a journey from **generalization to specialization.**

CONSTRUCTOR SEQUENCE:
1.BASE CLASS(PARENT CLASS)
2.DERIVED CLASS(CHILD CLASS)

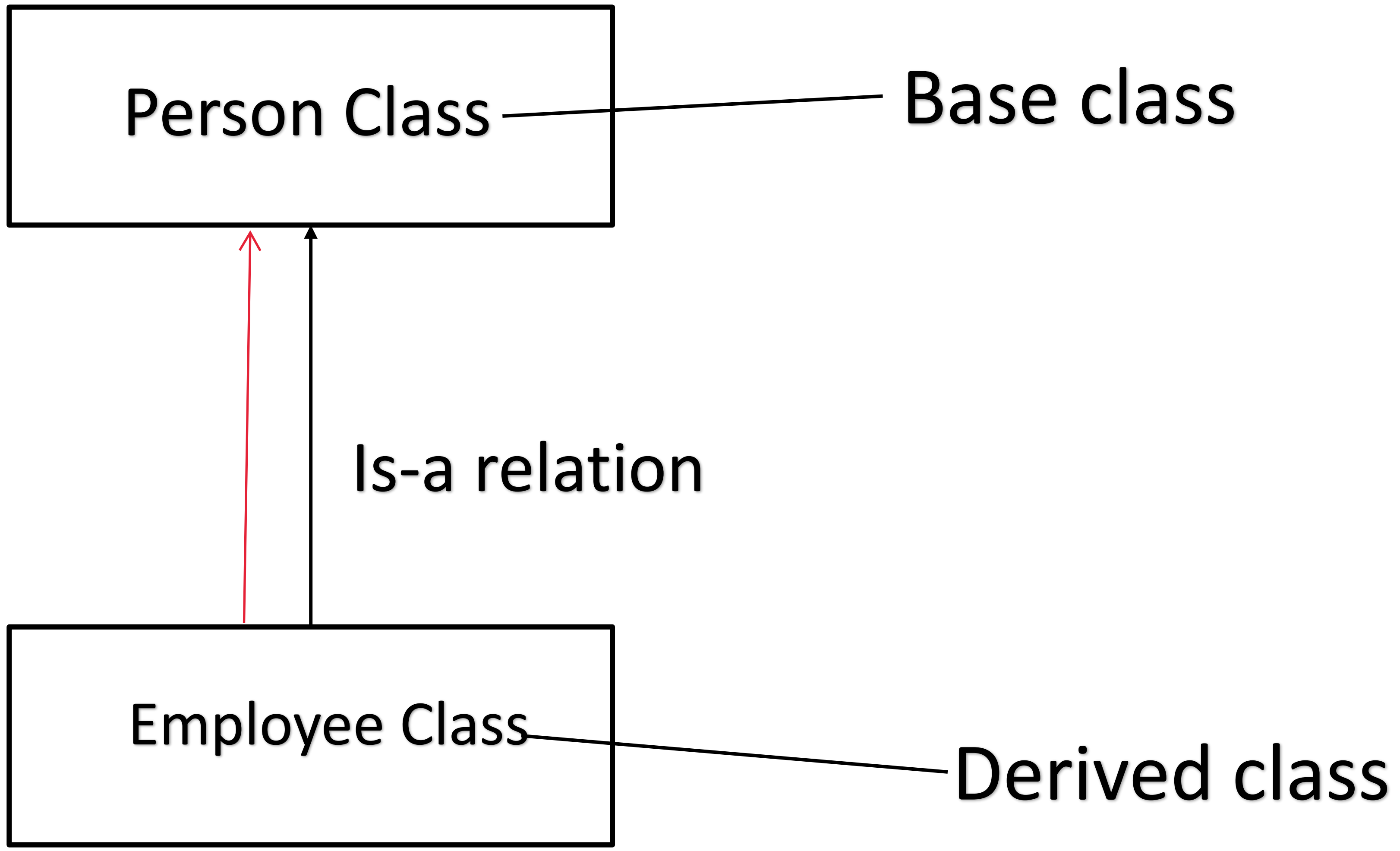
DESTRUCTOR SEQUENCE:
1.DERIVED CLASS
2.BASE CLASS

Inheritance

- **Base class:** The class **whose properties are inherited** by a derived class is called Base class or parent class.
- **Derived class:** The class that **inherits properties** from another class is called Derived class or child class.
- Inheritance is used to achieve the **code reusability.**
- If **data member in base class** is **non static** then it gets memory in **derived class object**.
- If we create object of derived class then **first base class constructor gets called** and **then derived class constructor gets called**.
- Destructor calling sequence is exactly opposite i.e. **first derived class destructor** gets called and **then base class destructor gets called.**

Inheritance

- If **“is-a” relationship** exists between two types(classes) then we should use inheritance e.g Employee is a Person.



Inheritance

- **Following function do not inherit into the derived class:**
  - 1. Constructor
  - 2. Destructor
  - 3. Copy constructor
  - 4. Assignment operator
  - 5. Friend function

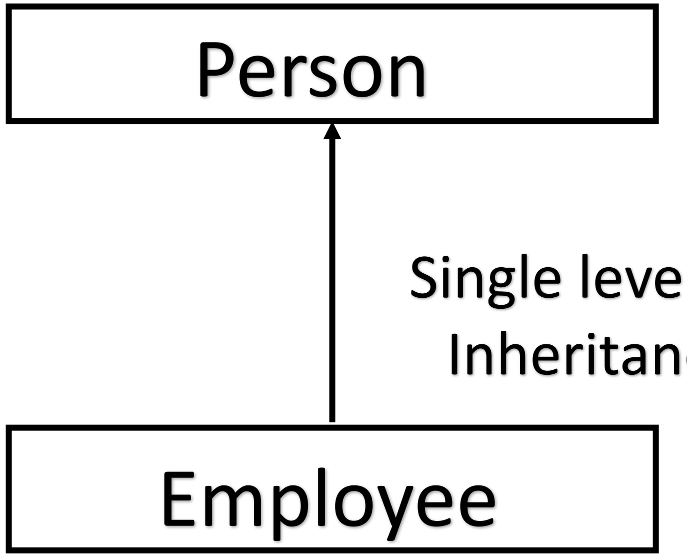
Inheritance

- **Types Of Inheritance**
  - 1. Single inheritance
  - 2. Multiple inheritance.
  - 3. Hierarchical inheritance
  - 4. Multilevel inheritance
- **1. Single Inheritance:**
  - If **single base class is having single derived class** then such inheritance is called single inheritance.
  - **Example:**

```
class Person        //Base class
{

};
class Employee : public Person        //Derived class
{

};
```

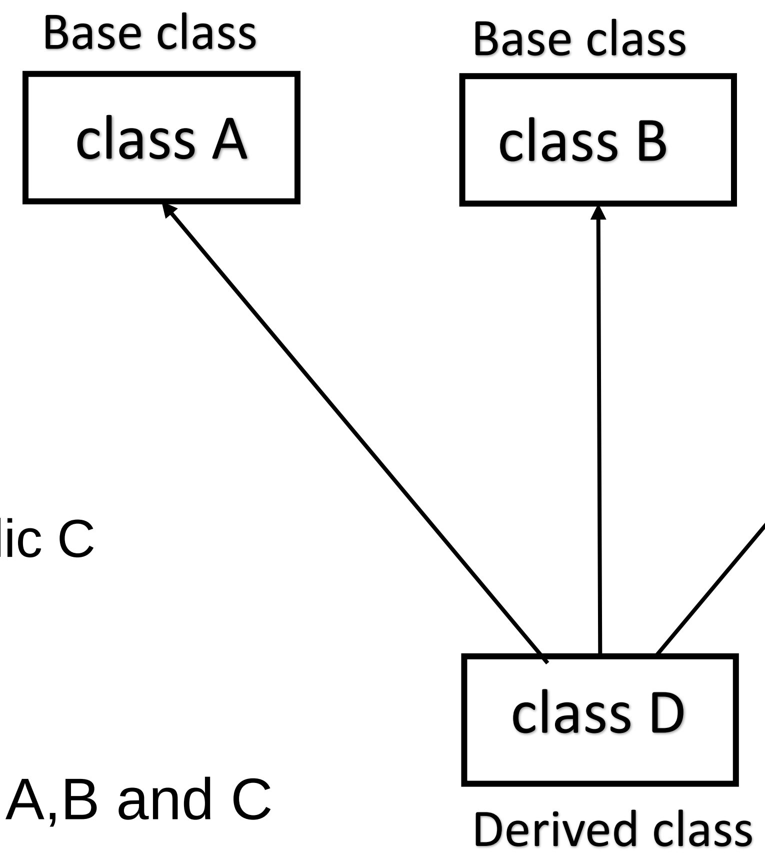


Inheritance

- **2. Multiple Inheritance :**
  - If **multiple Base classes are having single derived class** then such inheritance is called multiple inheritance.
  - **Example:**

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

class D is derived from class A,B and C

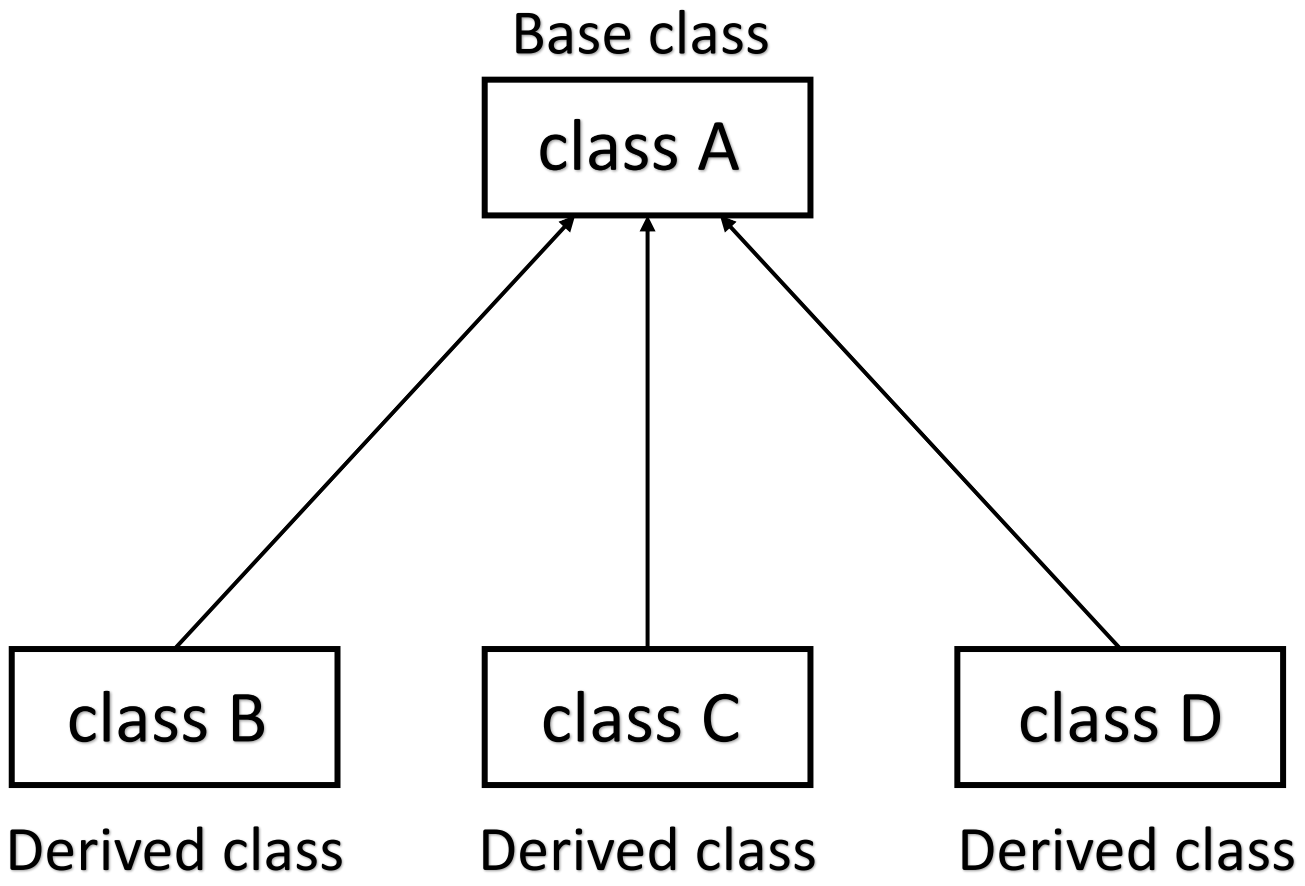


Inheritance

- **3 Hierarchical inheritance :**
  - If **single base class** is having **multiple derived classes** then such inheritance is called hierarchical inheritance.
- Example:

```
class A            //Base class
{
};
class B : public A    //Derived class
{
};
class C: public A     //Derived class
{
};
class D: public A    //Derived class
{
};
```

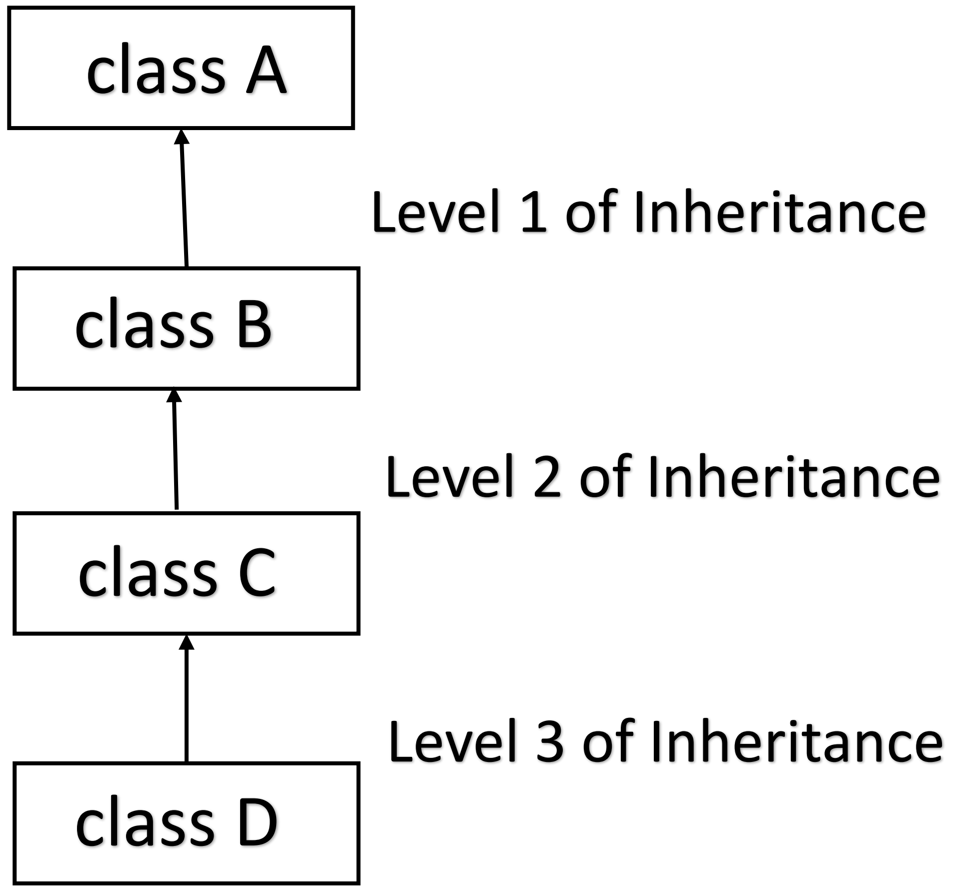
class B, C and D are derived from class A.



Inheritance

- **4. Multilevel inheritance :**
  - If **single inheritance** is having **number of levels** then it is called multilevel inheritance.
- **Example:**

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

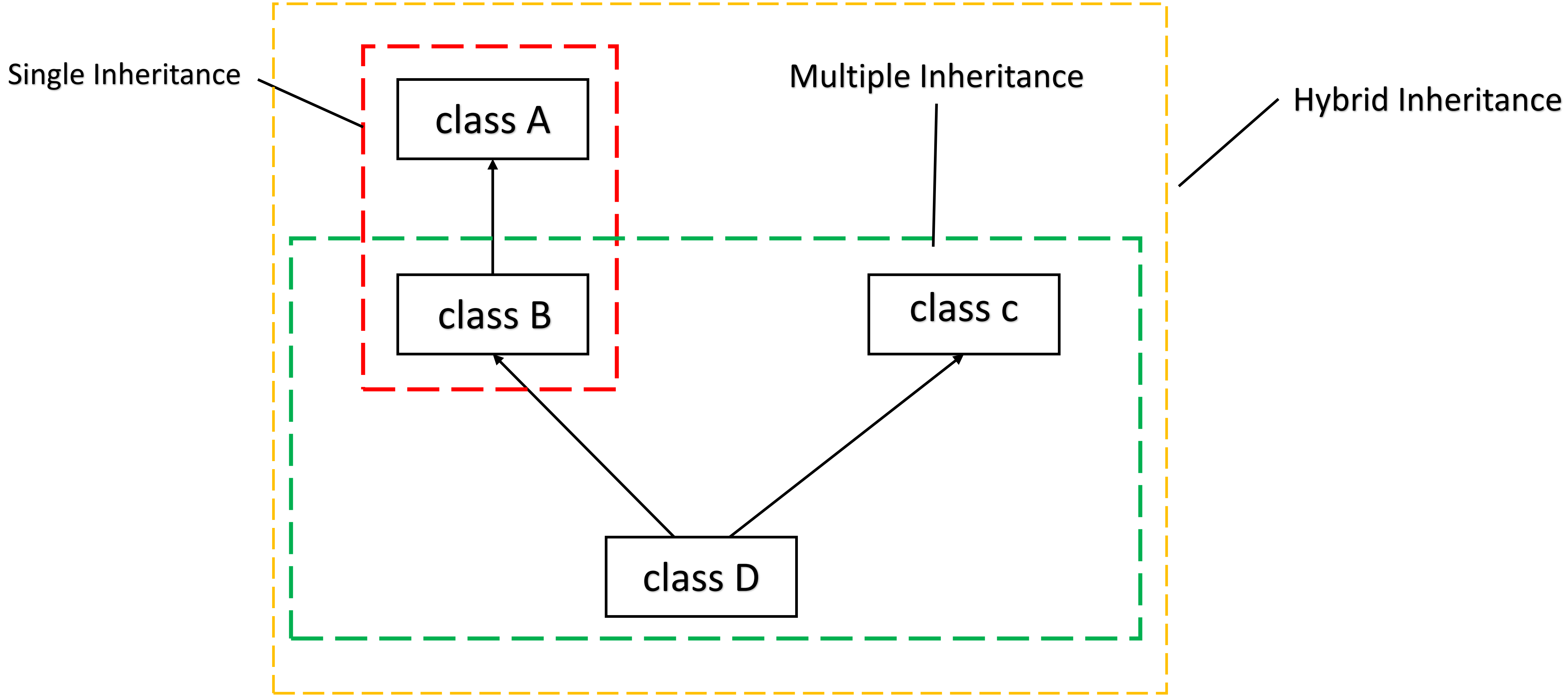


Inheritance

- **Hybrid Inheritance:**
  - The process of **combining more than one type of Inheritance together** while deriving subclasses in a program is called a Hybrid Inheritance.
- Example:

```
class A
{
        // block of statement(s)
};
class B: public A  // class B derived from a single class A -- follows single inheritance
{
        // block of statement(s)
};
class C
{
        // block of statement(s)
};
class D: public B, public C
// class D derived from two classes, class B and class C -- follows multiple inheritance
{
        // block of statement(s)
};
```

Inheritance



Inheritance

- Red outlined box shows **Single inheritance**.
- Green outlined box shows **Multiple inheritance**.
- So in yellow outline you could observe the combination of **single and multiple** inheritance together is hybrid inheritance.
- In above Example we could see **two different types of inheritances** (single inheritance and multiple inheritance)are **combined together**.

Mode of inheritance

- If we use private, protected and public keyword to manage visibility of the members of class then it is called as access specifier.
- But if we use these keywords to extends the class then it is called as mode of inheritance.
- C++ supports private, protected and public mode of inheritance. If we do not specify any mode, then default mode of inheritance is private.

Mode of inheritance

Mode of inheritance (read "--->" as becomes)

	Base	Derived
public mode:		
	Public --->	Public
	protected --->	Protected
	private --->	NA
protected mode:		
	Public --->	Protected
	protected --->	Protected
	private --->	NA
private mode:		
	Public --->	private
	protected --->	private
	private --->	NA

Thank You