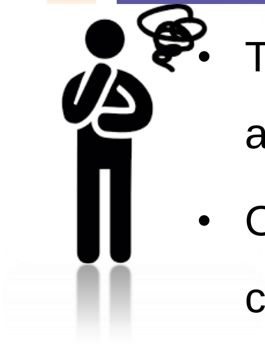
Java^{TM} Education & Technology Services

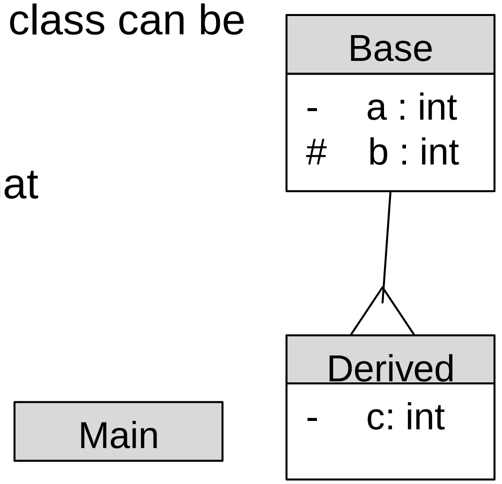
Object Oriented programming Using

C++

what are Access Specifiers?



- These access specifiers define how the members of the class can be accessed.
- Of course, any member of a class is accessible within that class(Inside any member function of that same class).



- Public - The members declared as Public are accessible from outside the Class through an object of the class.
- Protected - The members declared as Protected are accessible from outside the class BUT only in a class derived from it.
- Private - These members are only accessible from within the class. No outside Access is allowed.

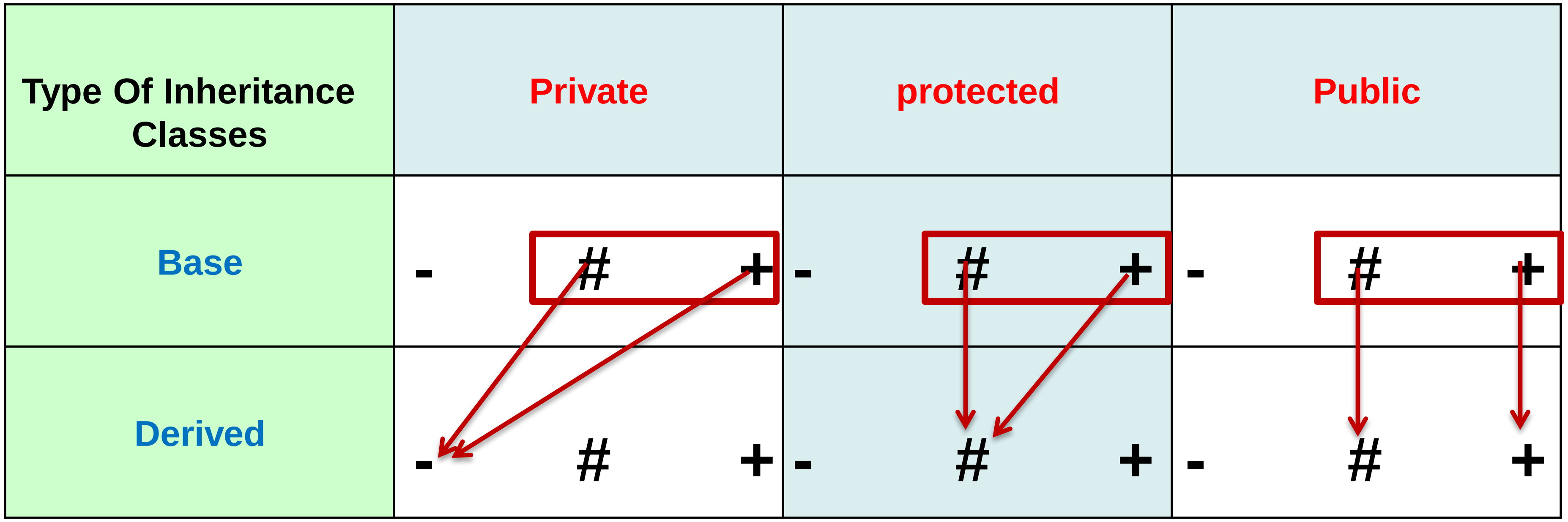
Types Of Inheritance

Types Of Inheritance

Derived Class Name : <Access Specifier> Base class Name

Ex: Manager: public Employee

Derived : ? Employee



Type Of Inheritance Classes	Private	protected	Public
Base	- # +	- # +	- # +
Derived	- # +	- # +	- # +

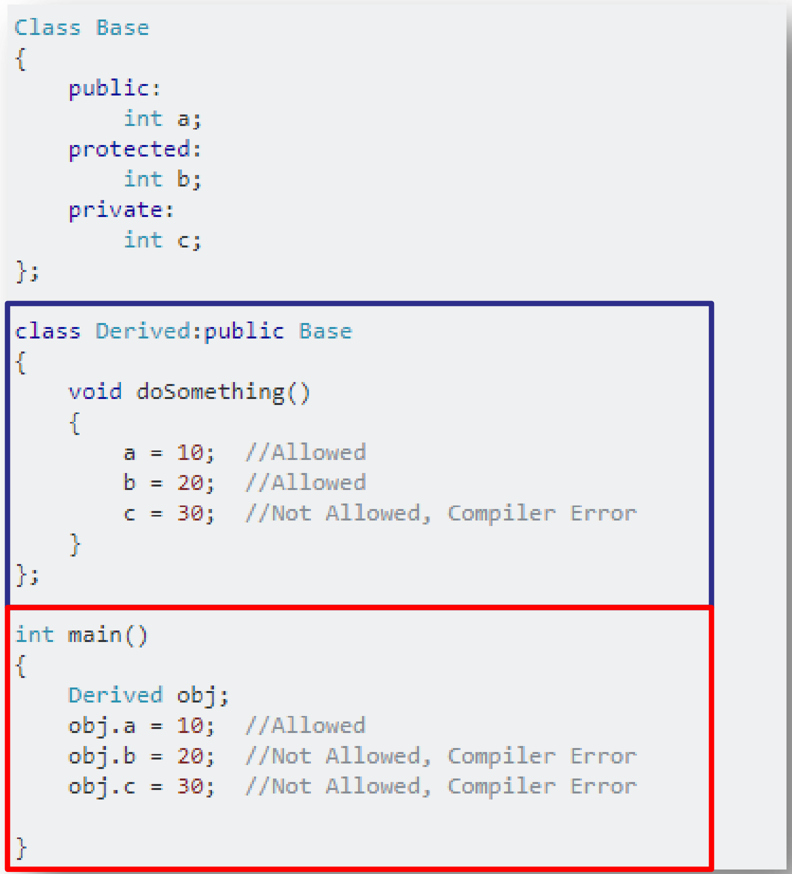
Types Of Inheritance

– Public Inheritance

- All Public members of the Base Class become Public Members of the derived class

  All Protected members of the Base Class become Protected Members of the Derived Class.

- No change in the Access of the members.



```
Class Base
{
    public:
        int a;
    protected:
        int b;
    private:
        int c;
};
```

```
class Derived:public Base
{
    void doSomething()
    {
        a = 10;  //Allowed
        b = 20;  //Allowed
        c = 30;  //Not Allowed, Compiler Error
    }
};
```

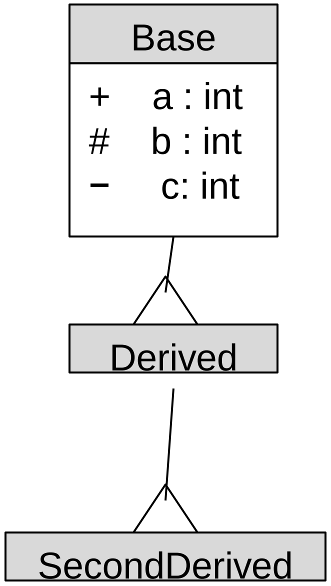
```
int main()
{
    Derived obj;
    obj.a = 10;  //Allowed
    obj.b = 20;  //Not Allowed, Compiler Error
    obj.c = 30;  //Not Allowed, Compiler Error

}
```

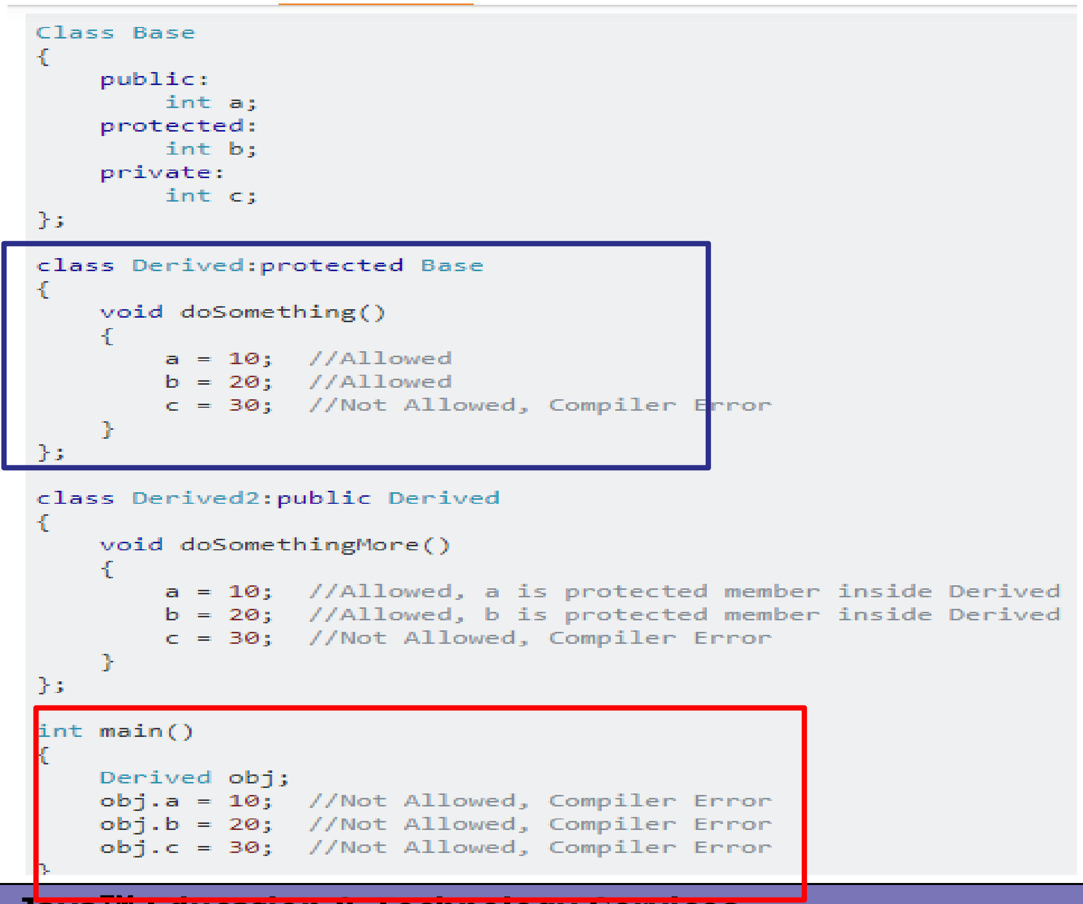
Types Of Inheritance

– Protected Inheritance

- All Public members of the Base Class become Protected Members of the derived class

  All Protected members of the Base Class become Protected Members of the Derived Class.





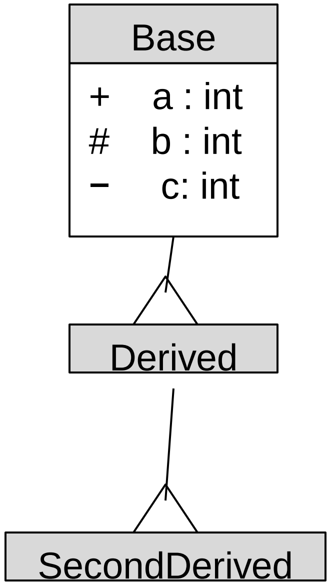
```
Class Base
{
    public:
        int a;
    protected:
        int b;
    private:
        int c;
};

class Derived:protected Base
{
    void doSomething()
    {
        a = 10;  //Allowed
        b = 20;  //Allowed
        c = 30;  //Not Allowed, Compiler Error
    }
};

class Derived2:public Derived
{
    void doSomethingMore()
    {
        a = 10;  //Allowed, a is protected member inside Derived
        b = 20;  //Allowed, b is protected member inside Derived
        c = 30;  //Not Allowed, Compiler Error
    }
};

int main()
{
    Derived obj;
    obj.a = 10;  //Not Allowed, Compiler Error
    obj.b = 20;  //Not Allowed, Compiler Error
    obj.c = 30;  //Not Allowed, Compiler Error
}
```

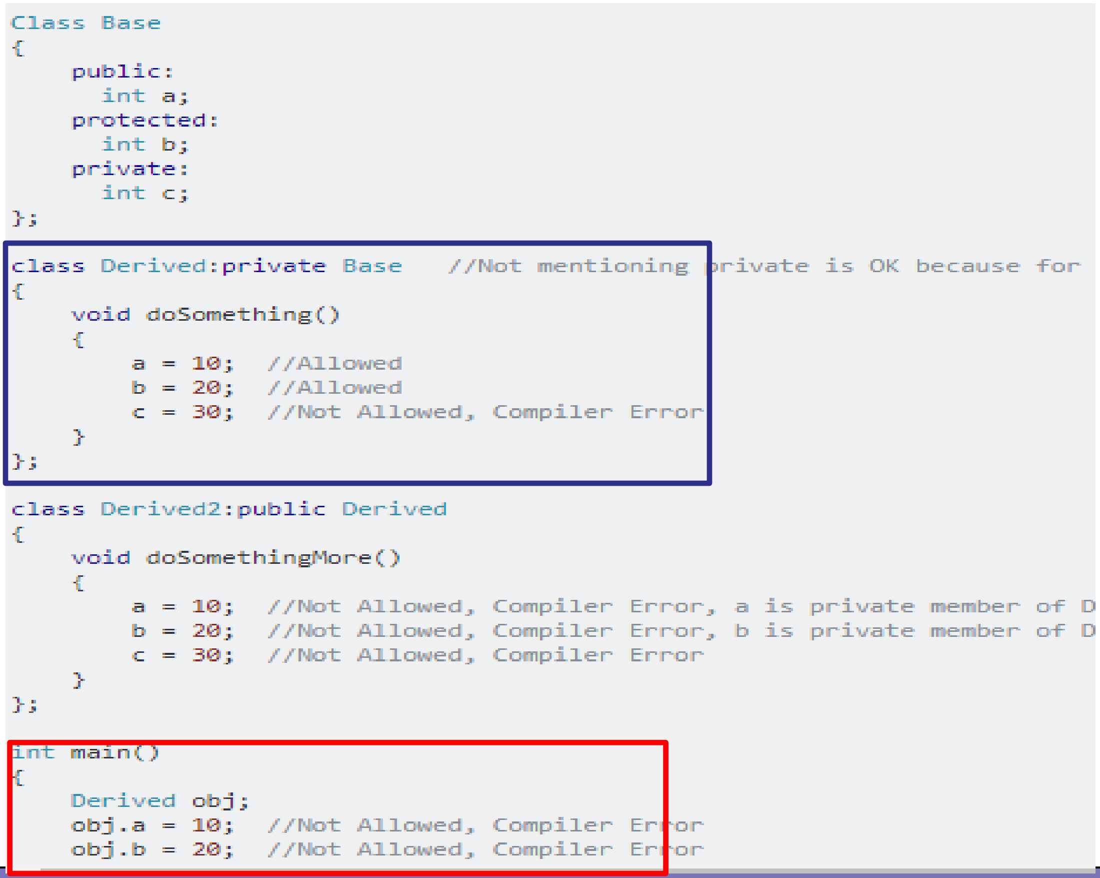
Types Of Inheritance

– Private Inheritance

- All Public members of the Base Class become Private Members of the derived class

  All Protected members of the Base Class become Private Members of the Derived Class.



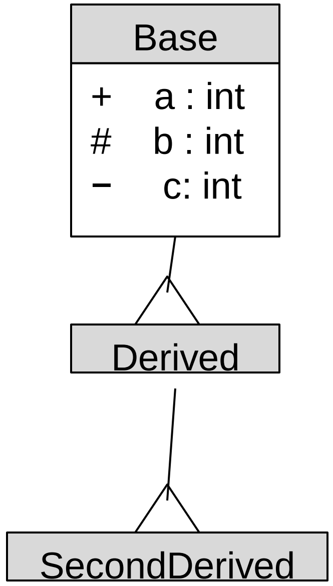


```
Class Base
{
    public:
      int a;
    protected:
      int b;
    private:
      int c;
};

class Derived:private Base   //Not mentioning private is OK because for
{
    void doSomething()
    {
        a = 10;  //Allowed
        b = 20;  //Allowed
        c = 30;  //Not Allowed, Compiler Error
    }
};

class Derived2:public Derived
{
    void doSomethingMore()
    {
        a = 10;  //Not Allowed, Compiler Error, a is private member of D
        b = 20;  //Not Allowed, Compiler Error, b is private member of D
        c = 30;  //Not Allowed, Compiler Error
    }
};

int main()
{
    Derived obj;
    obj.a = 10;  //Not Allowed, Compiler Error
    obj.b = 20;  //Not Allowed, Compiler Error
```

Types Of Inheritance

Accessibility in SecondDerived Class

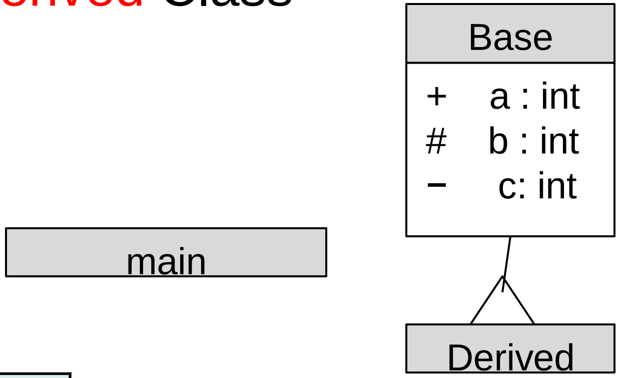
Inheritance Base & Derived / Class Base Members	Private	protected	Public
+a	NO	Yes	Yes
#b	NO	Yes	Yes
-c	NO	NO	NO



Types Of Inheritance

Accessibility in Main for object from Derived Class

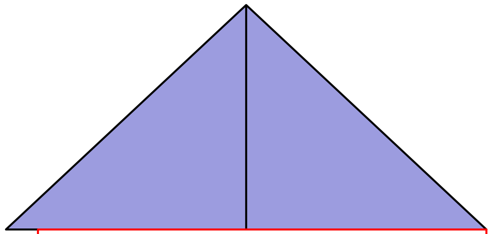
Derived obj;



Inheritance Base & Derived / Class Base Members	Private	protected	Public
+a	NO	NO	Yes
#b	NO	NO	NO
-c	NO	NO	NO

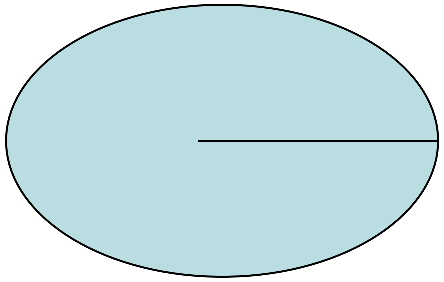
Types Of Inheritance

- Calculate the area of geometric shapes

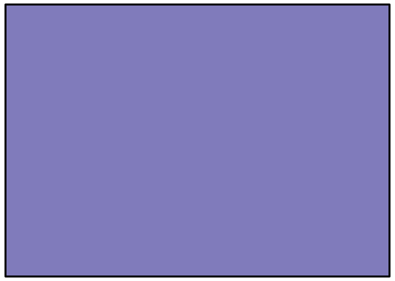


Triangle
Area =0.5 * d1 * d2

Rectangle
Area = d1 * d2

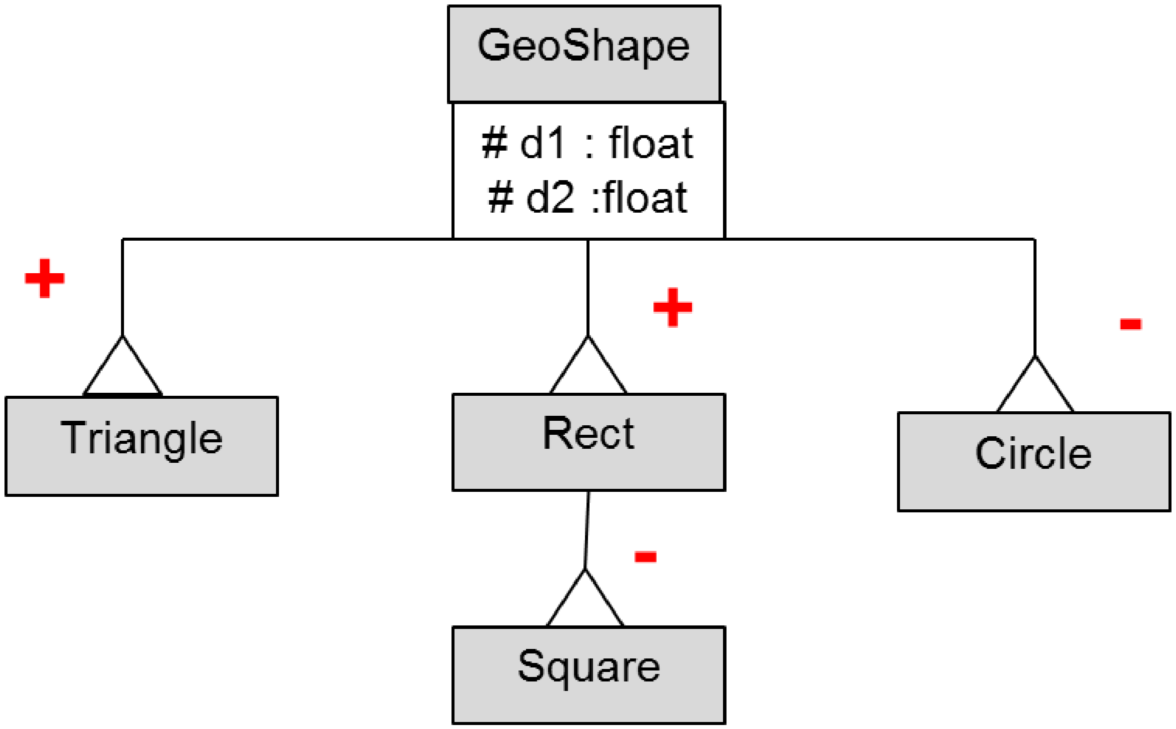


Circle
Area = 22/7 * r^2

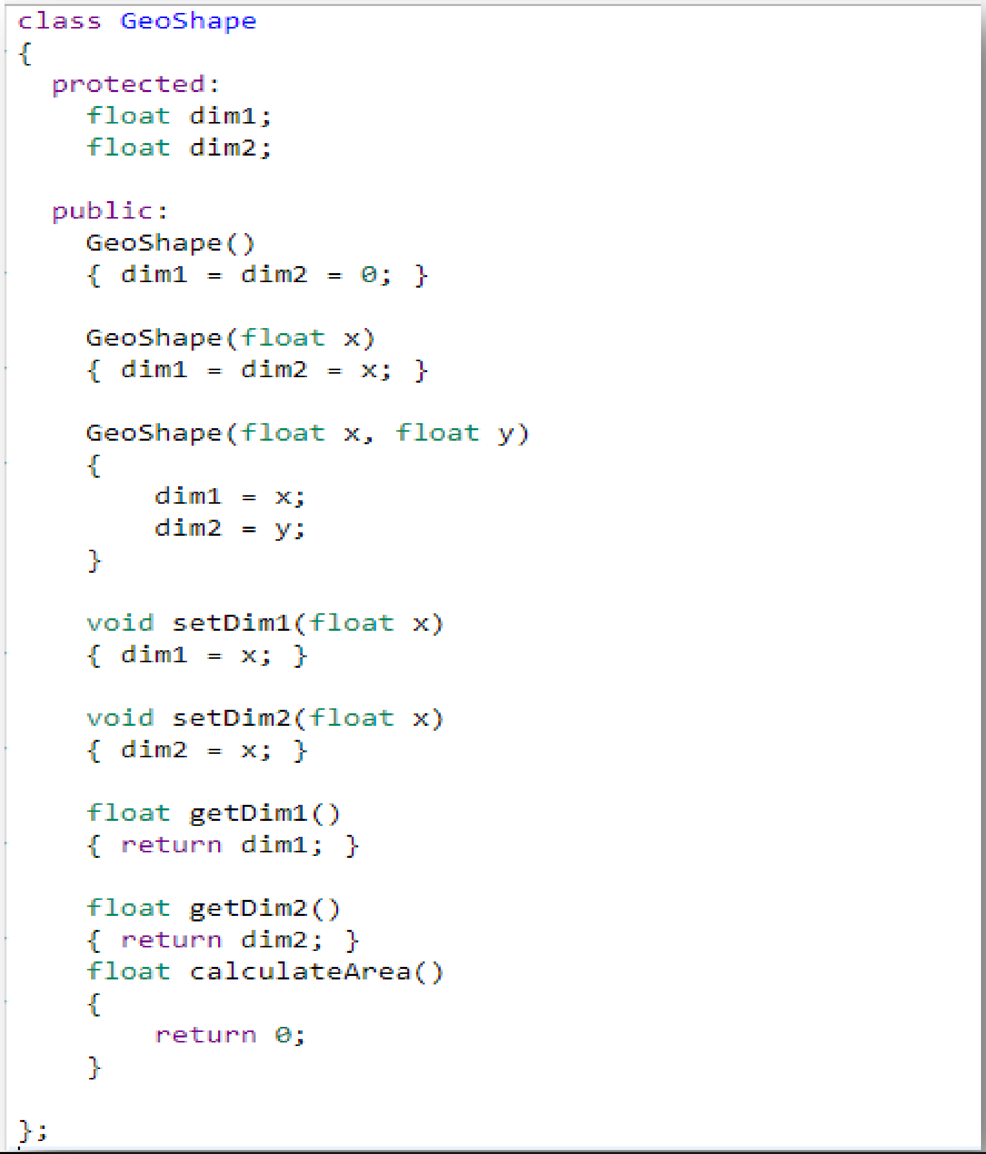


Square
Area = L ^ 2

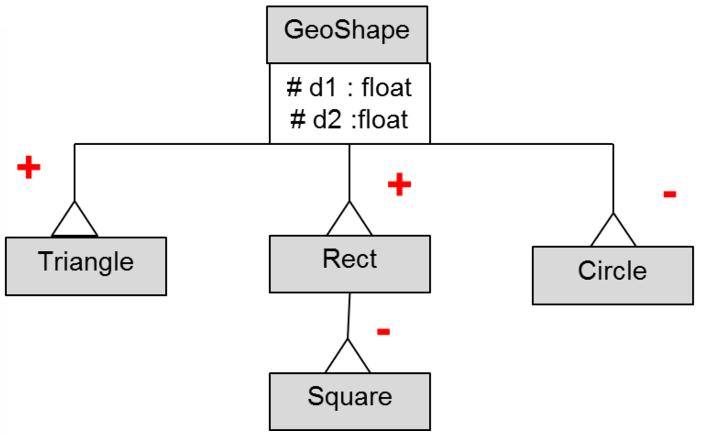
Types Of Inheritance



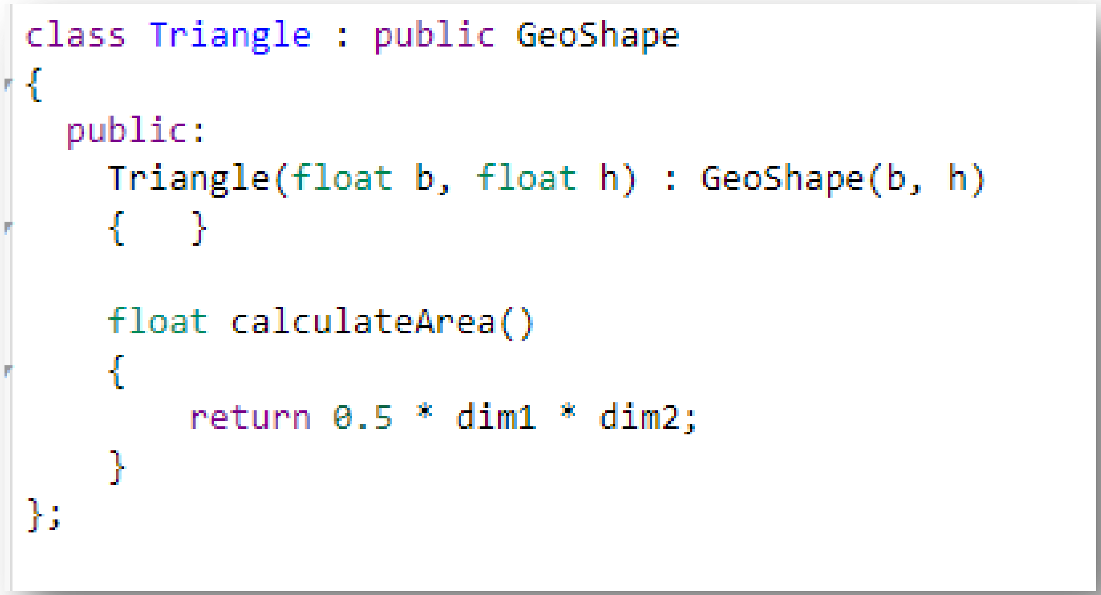
Types Of Inheritance



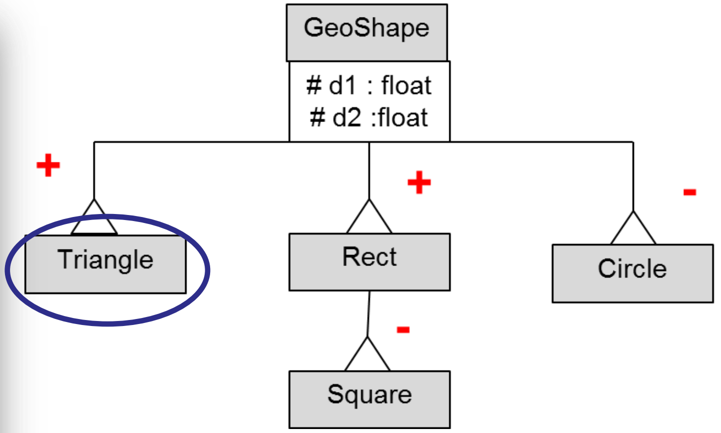
```
class GeoShape
{
  protected:
    float dim1;
    float dim2;

  public:
    GeoShape()
    { dim1 = dim2 = 0; }

    GeoShape(float x)
    { dim1 = dim2 = x; }

    GeoShape(float x, float y)
    {
        dim1 = x;
        dim2 = y;
    }

    void setDim1(float x)
    { dim1 = x; }

    void setDim2(float x)
    { dim2 = x; }

    float getDim1()
    { return dim1; }

    float getDim2()
    { return dim2; }
    float calculateArea()
    {
        return 0;
    }

};
```



Types Of Inheritance

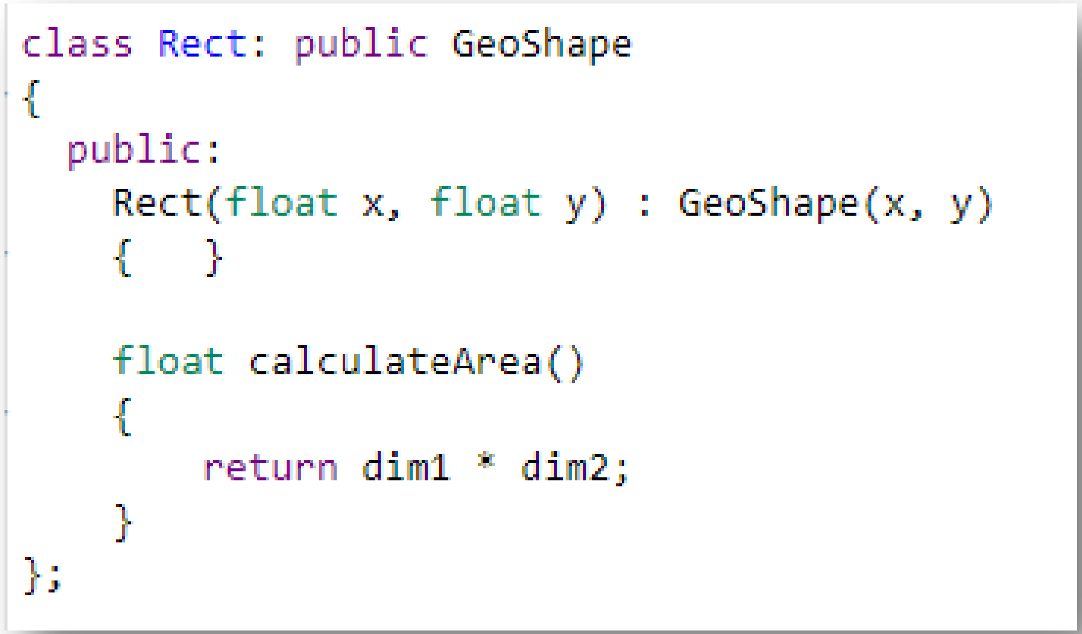


```
class Triangle : public GeoShape
{
  public:
    Triangle(float b, float h) : GeoShape(b, h)
    {   }

    float calculateArea()
    {
        return 0.5 * dim1 * dim2;
    }
};
```

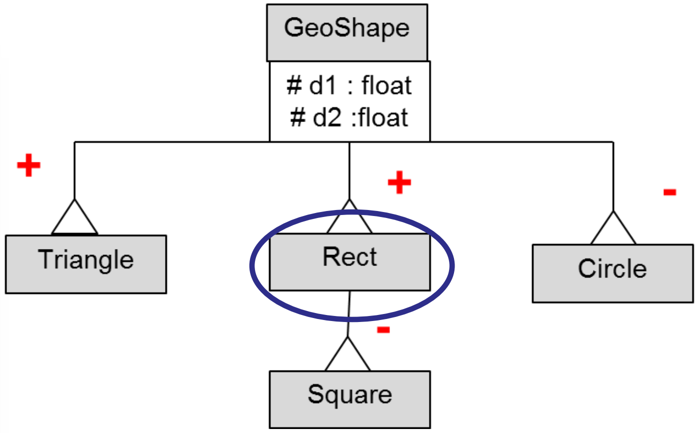


Object from Triangle can access the public members of Triangle and GeoShape.

Types Of Inheritance

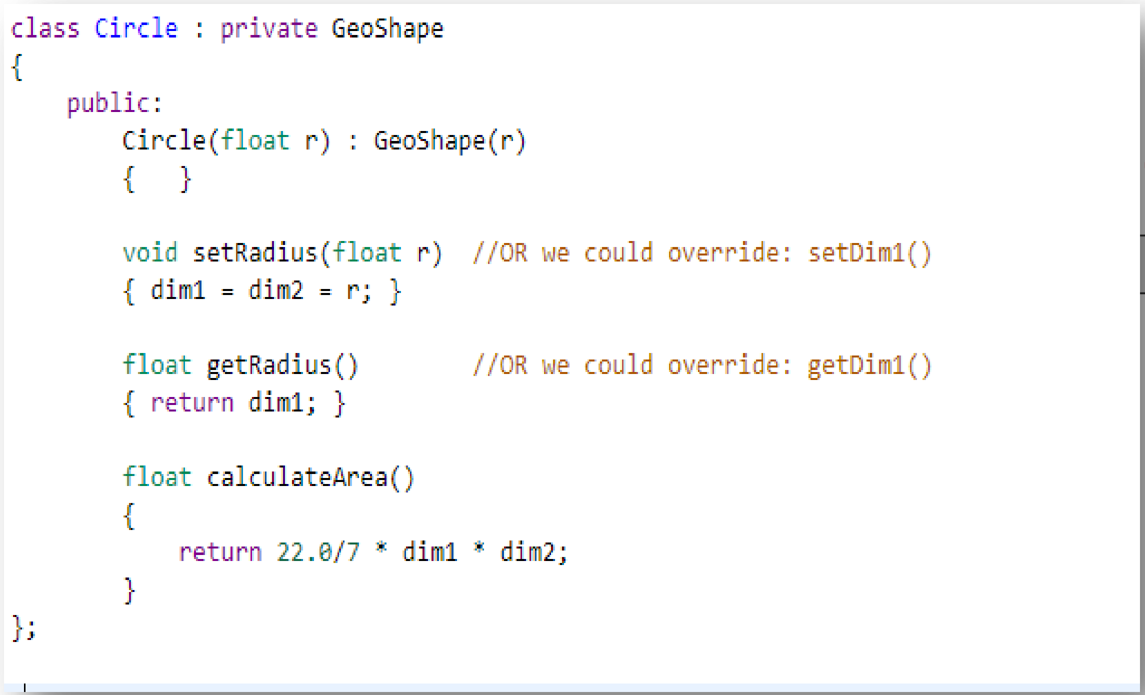


```
class Rect: public GeoShape
{
  public:
    Rect(float x, float y) : GeoShape(x, y)
    {   }

    float calculateArea()
    {
        return dim1 * dim2;
    }
};
```

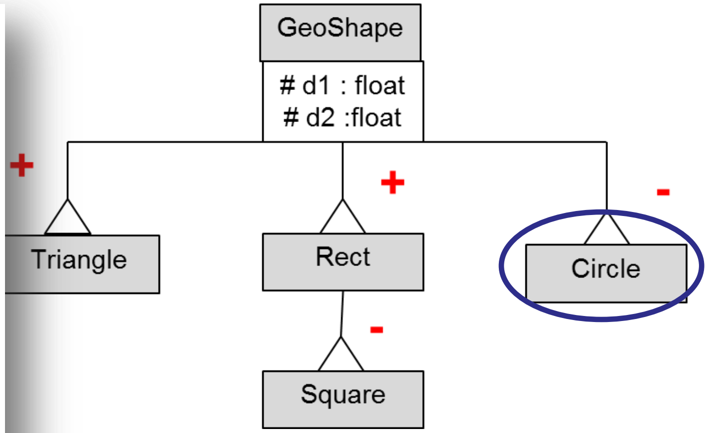


Object from Rect can access the public members of Rect and GeoShape.

Types Of Inheritance



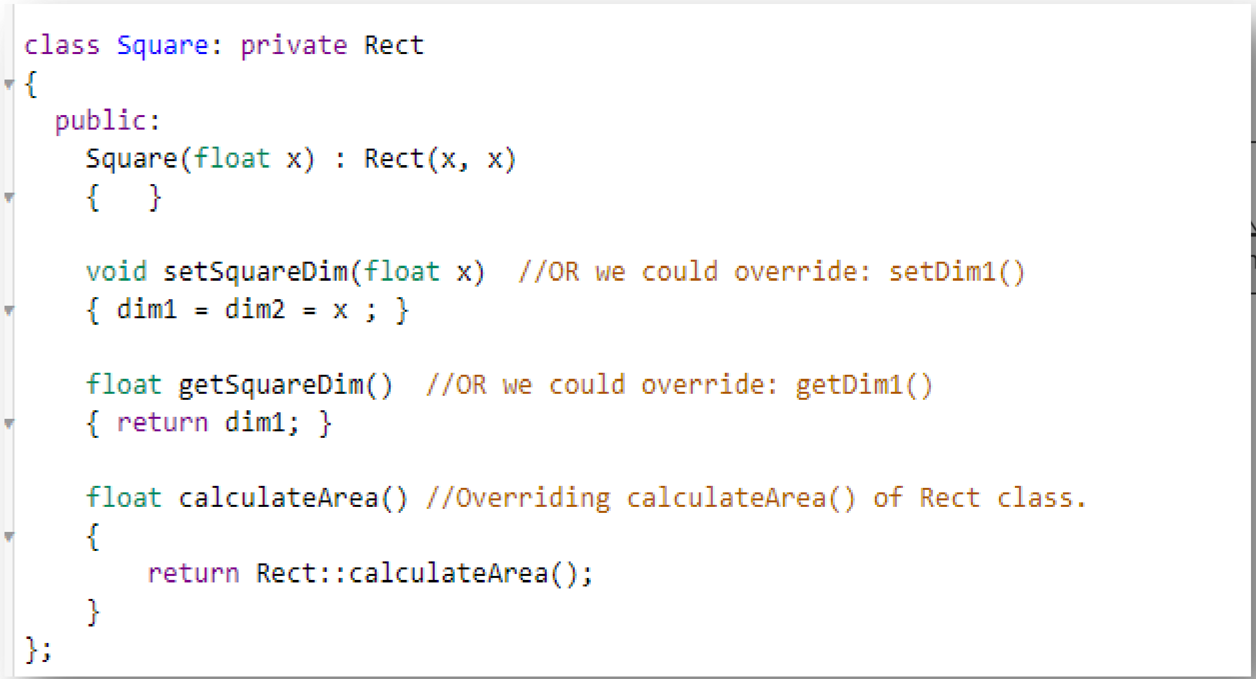
```
class Circle : private GeoShape
{
    public:
        Circle(float r) : GeoShape(r)
        {   }

        void setRadius(float r)  //OR we could override: setDim1()
        { dim1 = dim2 = r; }

        float getRadius()        //OR we could override: getDim1()
        { return dim1; }

        float calculateArea()
        {
            return 22.0/7 * dim1 * dim2;
        }
};
```



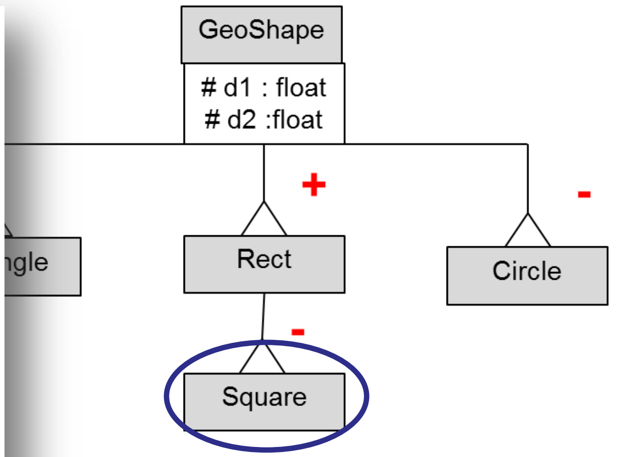
Object from Circle can access only the public members of Circle.

Circle c1;
❌ c1.setDim1(5);
❌ c1.getDim2();
c1. setRadius(5);

Types Of Inheritance



```
class Square: private Rect
{
  public:
    Square(float x) : Rect(x, x)
    {   }

    void setSquareDim(float x)  //OR we could override: setDim1()
    { dim1 = dim2 = x ; }

    float getSquareDim()  //OR we could override: getDim1()
    { return dim1; }

    float calculateArea() //Overriding calculateArea() of Rect class.
    {
        return Rect::calculateArea();
    }
};
```

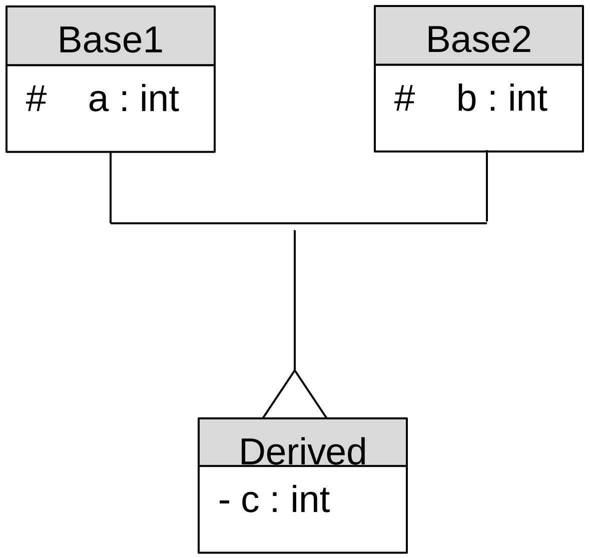


Object from Square can access only the public members of Square.

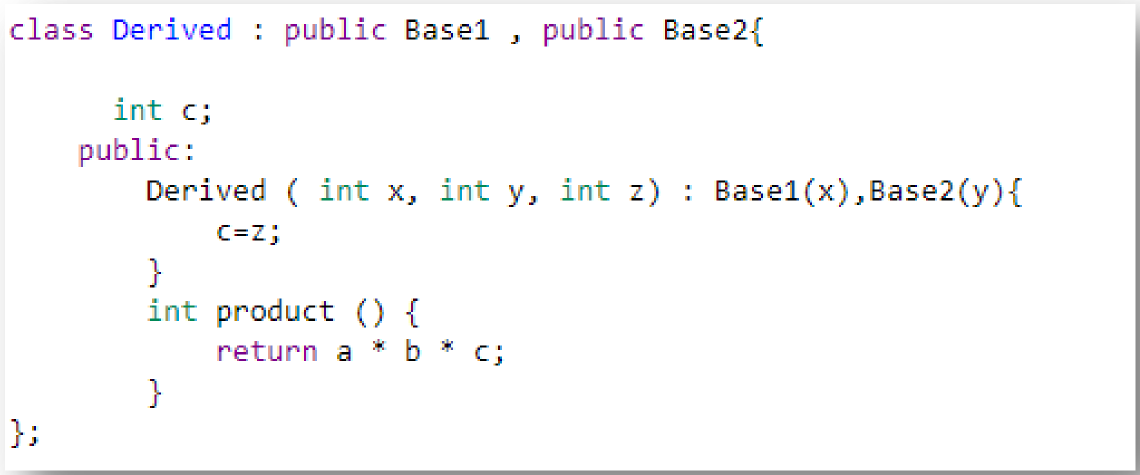
Square s1;
✗ s1.setDim1(5);
✗ s1.getDim2();
s1. setSquareDim(5);

Multiple Inheritance

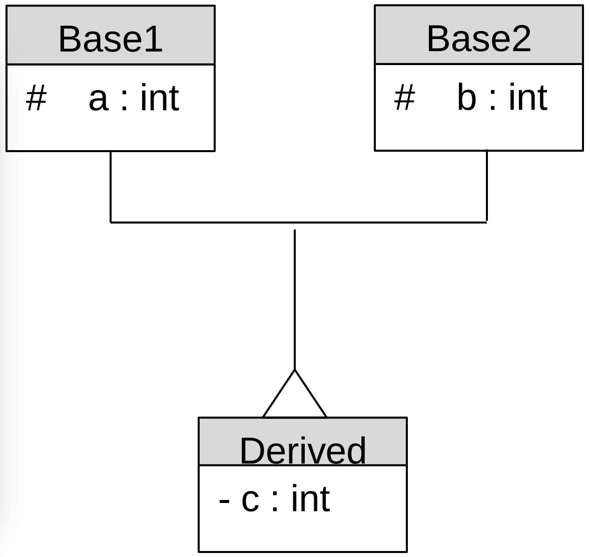
- Try this feature in C++ but Not use it .
- It is a wrong concept in OOP as at the end , we may have an object that carries all the tree.



Multiple Inheritance



```
class Derived : public Base1 , public Base2{

      int c;
    public:
        Derived ( int x, int y, int z) : Base1(x),Base2(y){
            c=z;
        }
        int product () {
            return a * b * c;
        }
};
```



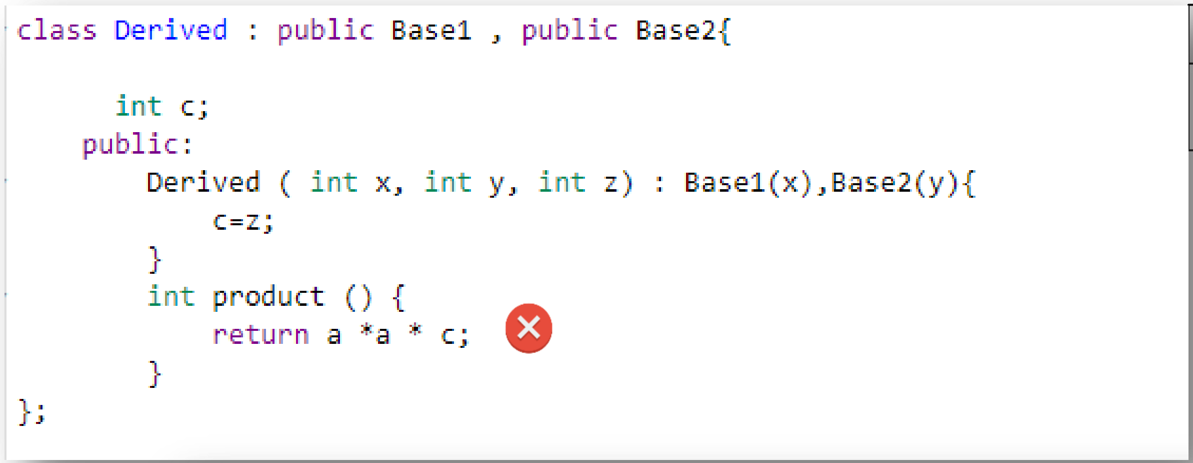
Object from Derived can access the public members of Derived, Base1 and Base2.

Derived d1;

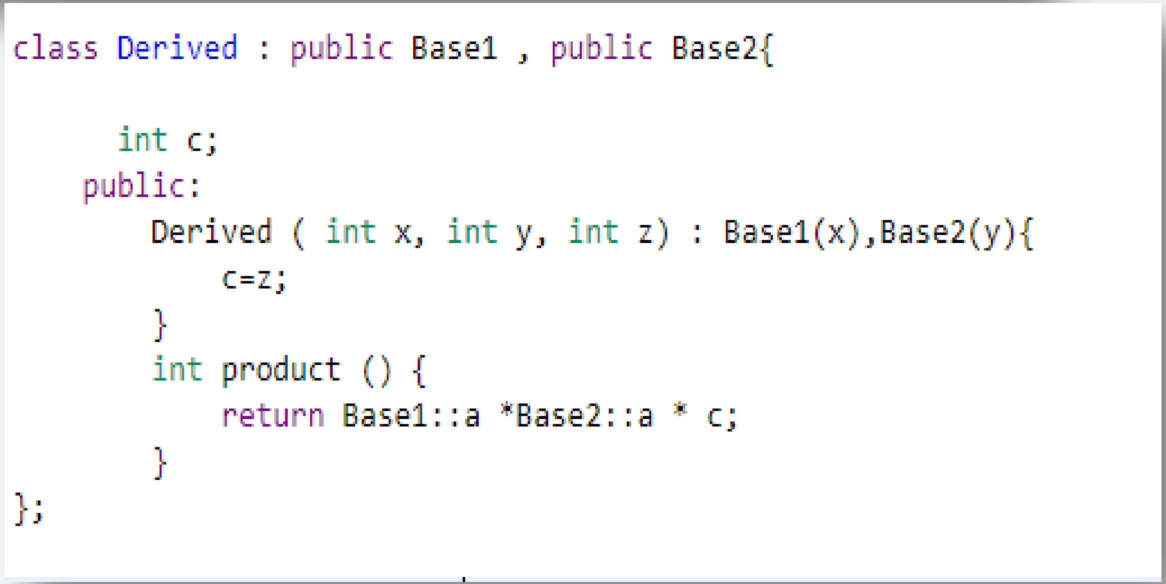
- Constructors order
- Destructor order

Multiple Inheritance

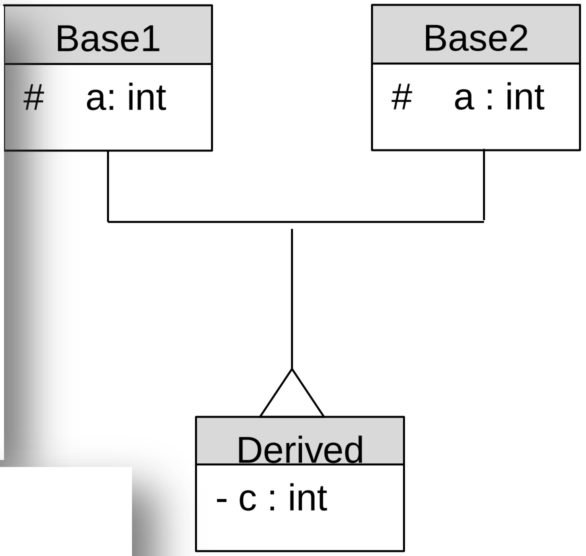
- Problem 1



```
class Derived : public Base1 , public Base2{

      int c;
    public:
        Derived ( int x, int y, int z) : Base1(x),Base2(y){
            c=z;
        }
        int product () {
            return a *a * c;
        }
};
```

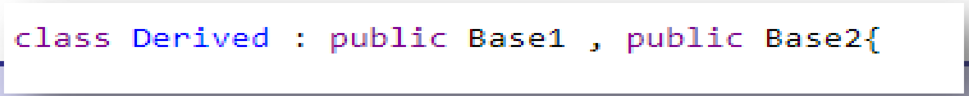


```
class Derived : public Base1 , public Base2{

      int c;
    public:
        Derived ( int x, int y, int z) : Base1(x),Base2(y){
            c=z;
        }
        int product () {
            return Base1::a *Base2::a * c;
        }
};
```



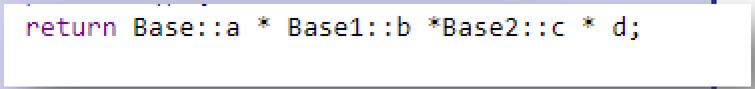
Multiple Inheritance

- Problem 2



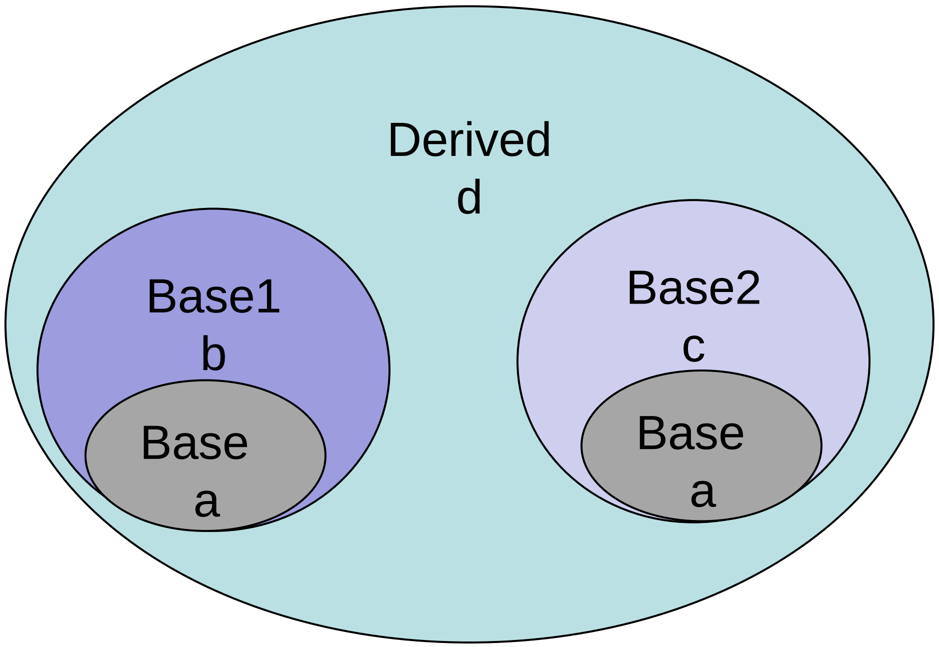
```
class Derived : public Base1 , public Base2{
```

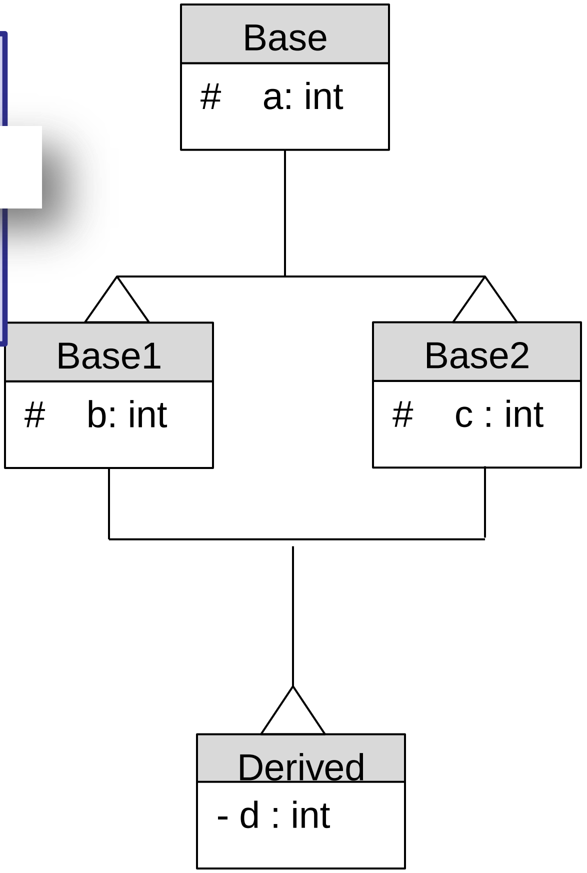
Derived d1;



```
return Base::a * Base1::b *Base2::c * d;
```

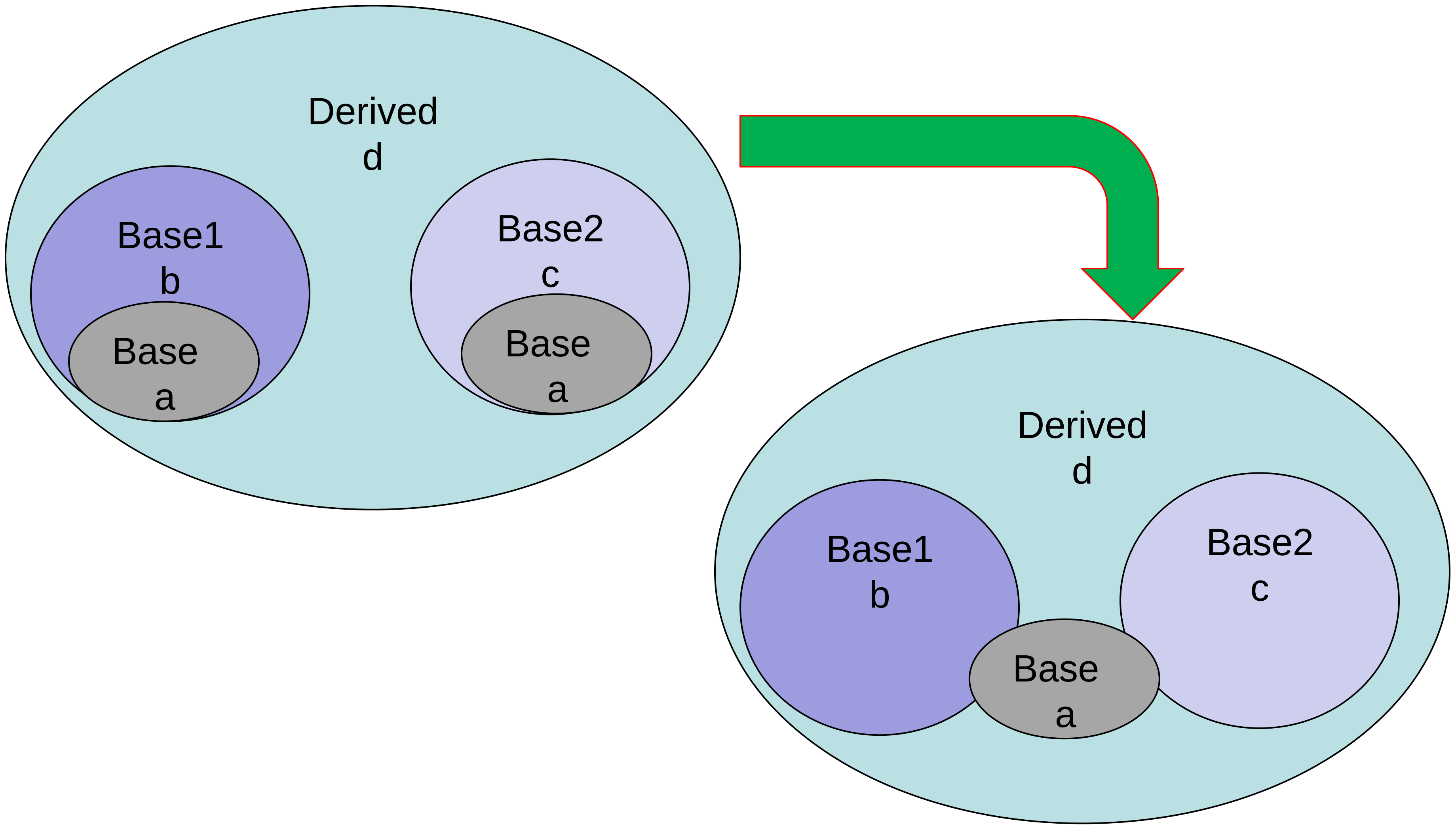
- Ambiguity there are two objects form Base in one object Derived





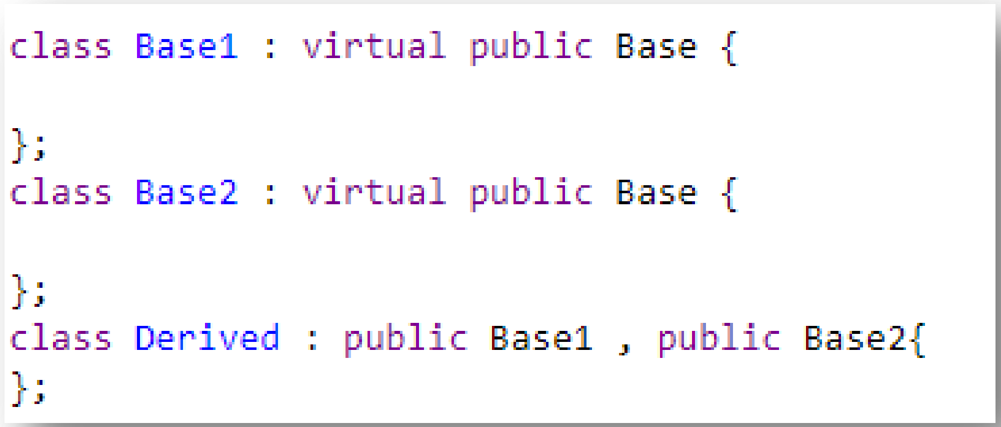
Multiple Inheritance

- Problem 2

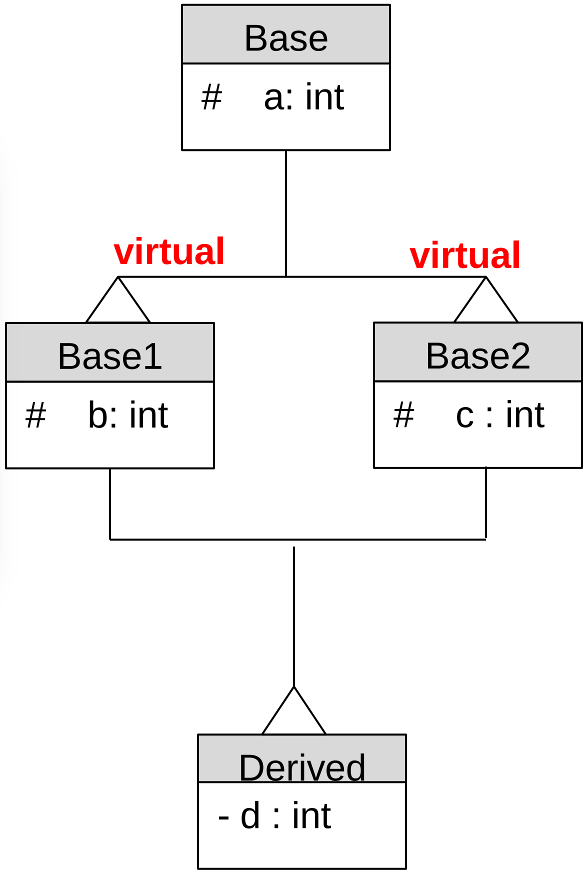


Multiple Inheritance

- Problem 2



```
class Base1 : virtual public Base {

};
class Base2 : virtual public Base {

};
class Derived : public Base1 , public Base2{
};
```



Lab Exercise

Lab Exercise

- 1st Assignment :
  - Geoshape Example
    - » try with it all the inheritance types