A Third Type of Access Modifier

`private`

Your toothbrush
(Only you can use it)

`protected`

`public`

Your toothpaste
(Anyone can use it)

A Third Type of Access Modifier

`private`

Your toothbrush
(Only you can use it)

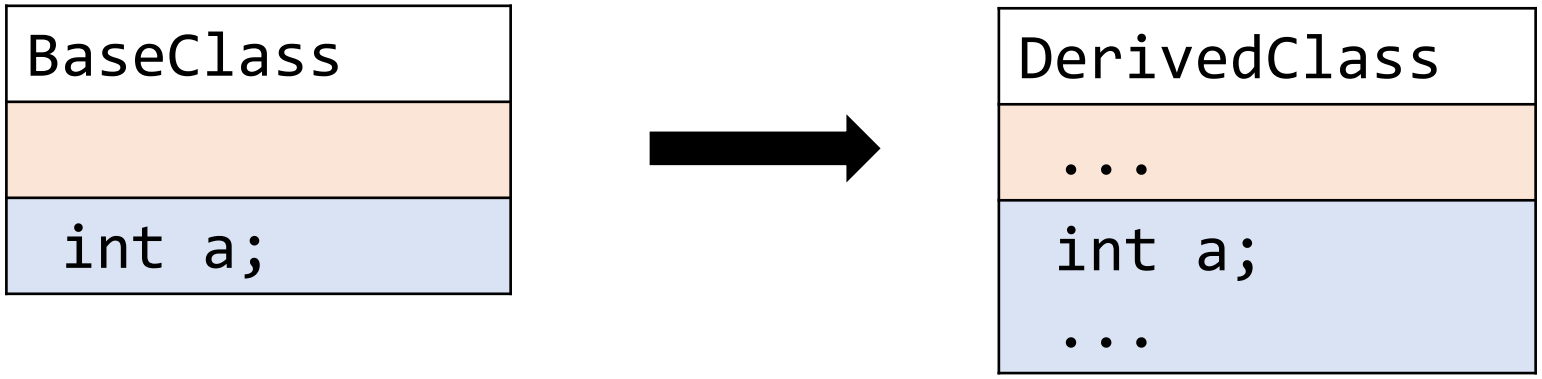
`protected`

Your home
(Your relatives can use it too)

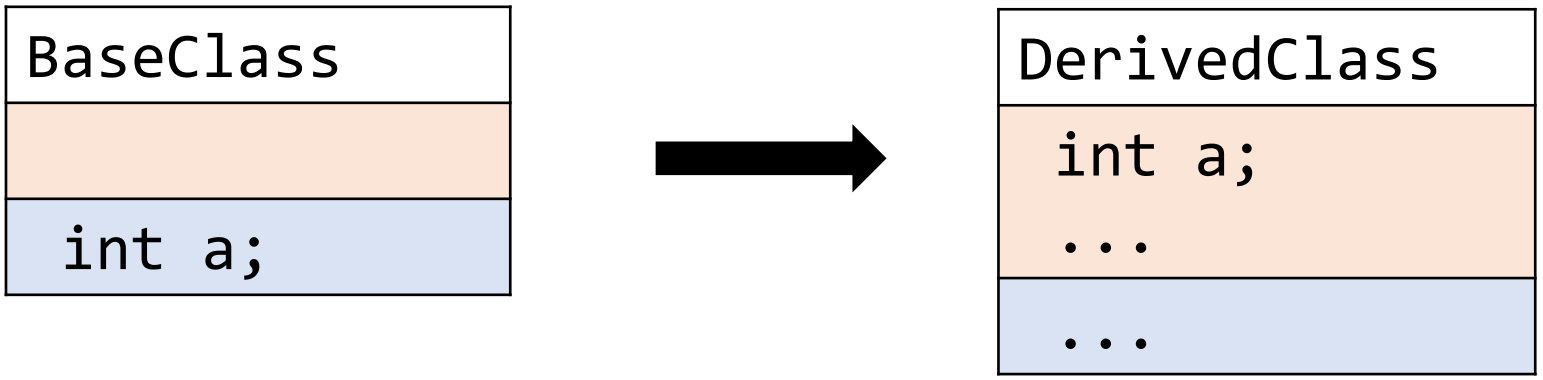
`public`

Your toothpaste
(Anyone can use it)

Public Inheritance

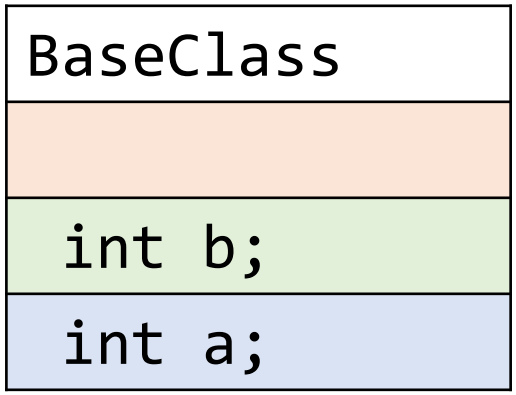


Private Inheritance



We need something that is private in both Base Class and Derived Class

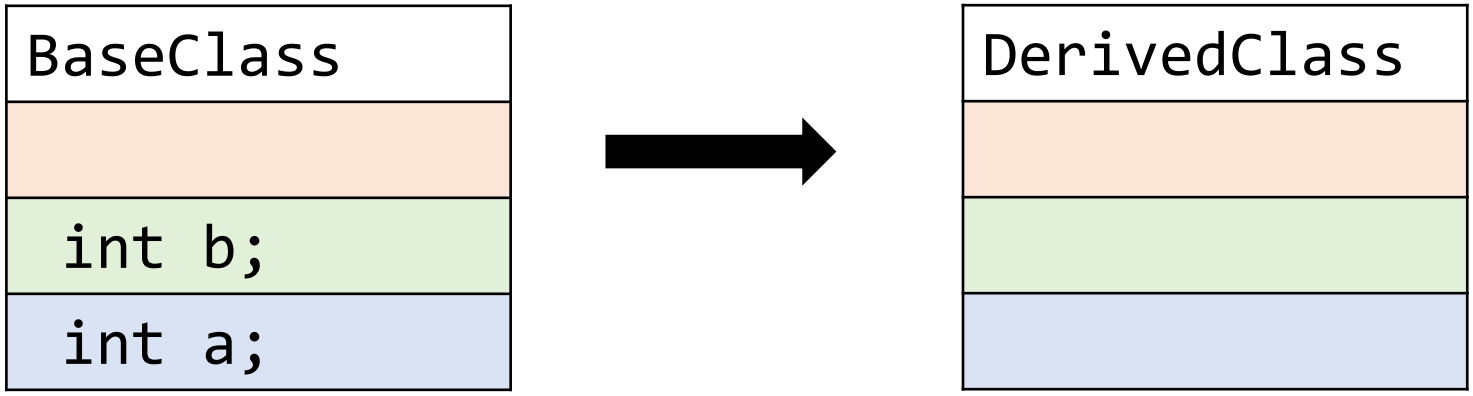
Protected Member



Protected member

‘b’ is inaccessible from outside of **BaseClass** (just like a private member)
But it’s accessible in its derived classes

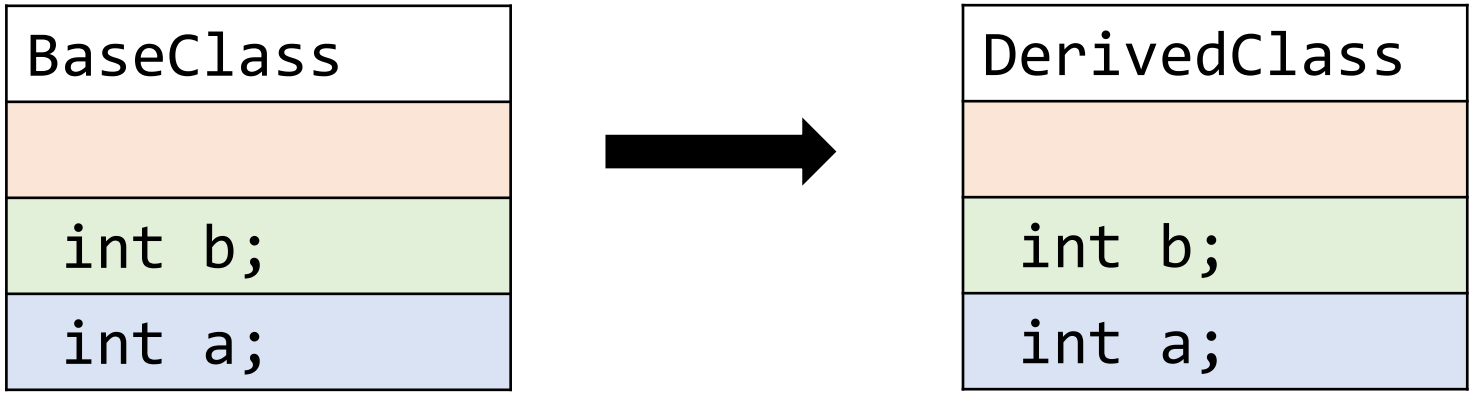
Public Inheritance




```
class DerivedClass : public BaseClass
{

}
```

Public Inheritance

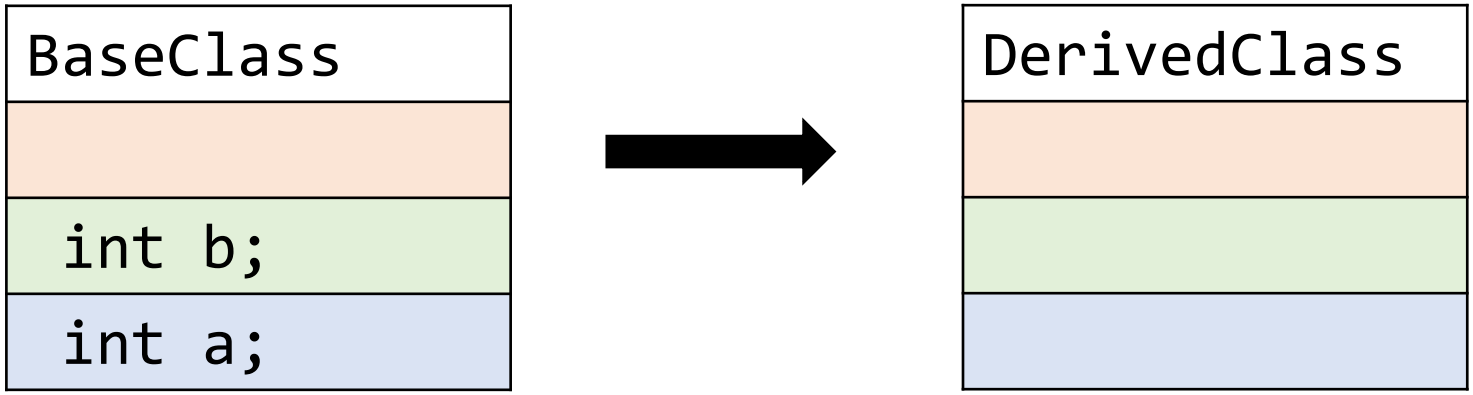



```
class DerivedClass : public BaseClass
{

}
```

'b' is inaccessible from outside of **DerivedClass** (just like a private member)
But it's accessible in its derived classes!

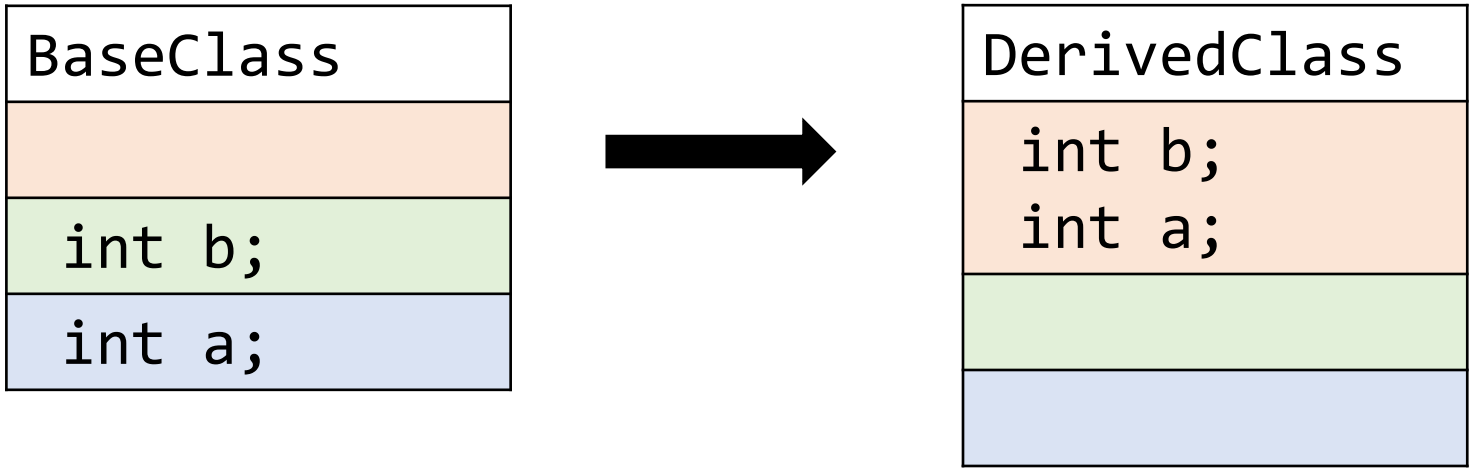
Private Inheritance




```
class DerivedClass : private BaseClass
{

}
```

Private Inheritance

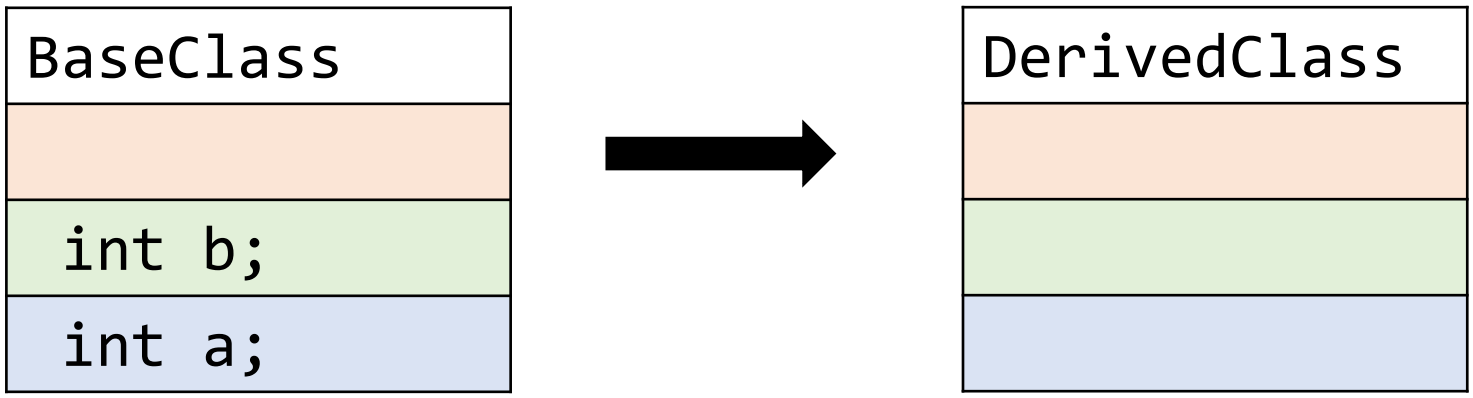



```
class DerivedClass : private BaseClass
{

}
```

Both ‘b’ and ‘a’ becomes private. **Neither** will be accessible from its derived class **or** outside class anymore.

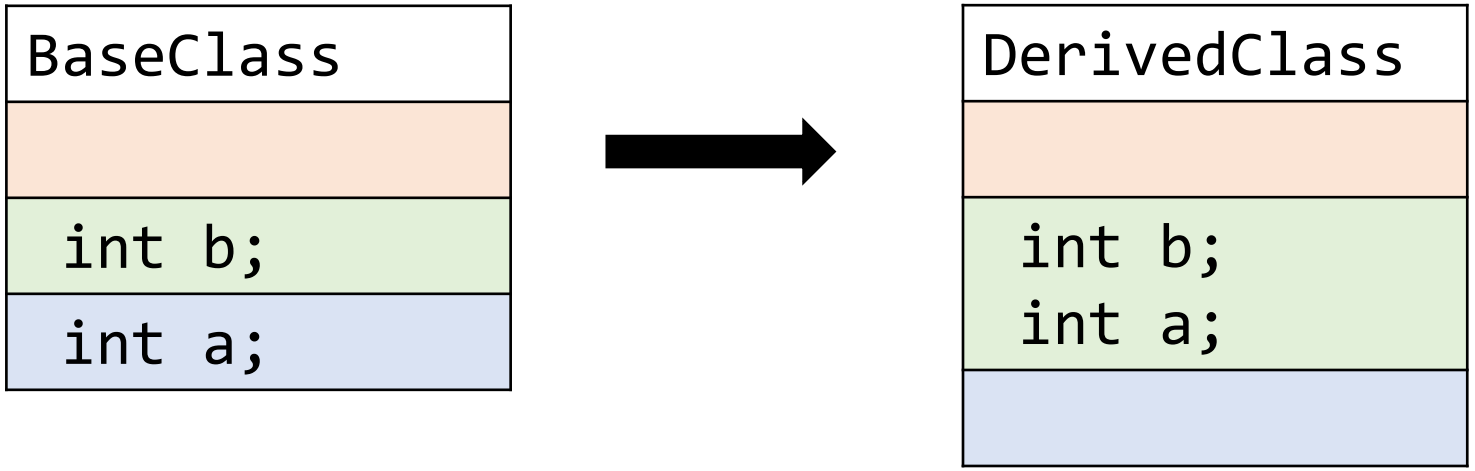
Protected Inheritance




```
class DerivedClass : protected BaseClass
{

}
```

Protected Inheritance

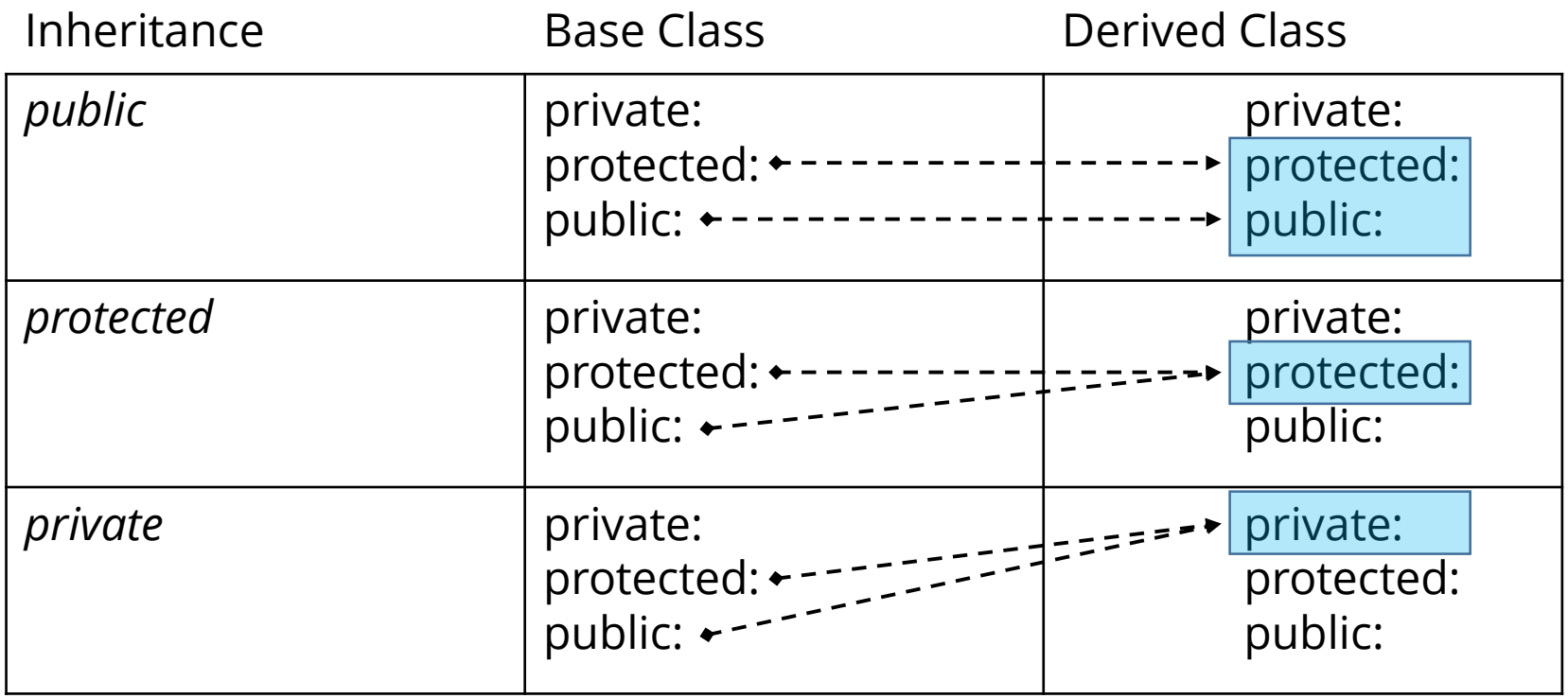



```
class DerivedClass : protected BaseClass
{

}
```

Both ‘b’ and ‘a’ becomes protected. **Neither** will be accessible from outside this DerivedClass**, except** its derived classes.

Access Specifier Summary (Final)



Source: https://flylib.com/books/en/2.937.1.175/1/

Modifying Behavior of Derived class

- Modification of functions
- Modification of variables
- Modification of visibility

Modifying Function Behavior (Overriding)

Modifying Function Behavior (Overriding)

```
class Point
{
    protected:
    int x;
    int y;
    public:
    void setPoint(int _x, int _y)
    {
        x = _x;
        y = _y;
    }

    void show()
    {
        cout << "(" << x << ", " << y <<")" << endl;
    }
};

Point p1;
p1.setPoint(3, 4);
p1.show();          //(3, 4)
```

Modifying Function Behavior (Overriding)

```
class Point3D : public Point
{
    int z;
    public:
    void setZ(int a, int b, int c)
    {
        setPoint(a, b);
        z = c;
    }
};
```

```
Point3D p2;
p2.setZ(5, 6, 7);
p2.show();          //(5, 6)
```

Modifying Function Behavior (Overriding)

```
class Point3D : public Point
{
    int z;
    public:
    void setZ(int a, int b, int c)
    {
        setPoint(a, b);
        z = c;
    }
    void show()
    {
        cout << "(" << x << ", " << y << ", " << z << ")" << endl;
    }
};
```

```
Point3D p2;
p2.setZ(5, 6, 7);
p2.show();          //(5, 6, 7)
```

Modifying Function Behavior (Overriding)

When a function is called, it tries to invoke the nearest definition.

Rule: The prototype of the overriding function must be exactly the same as the original function in Base class. i.e.:

- Same function name
- Same parameter list
- Same return type

Modifying Variable Behavior

This is not encouraged, but can be done.

```
class Point
{
    protected:
    int x;
    int y;
    int resourceID = 50;
    public:
    void setPoint(int _x, int _y)
    {
        x = _x;
        y = _y;
    }
    void show()
    {
        cout << resourceID << " (" << x << ", " << y <<")" << endl;
    }
};
```

Modifying Variable Behavior

```
class Point3D : public Point
{
    int z;
protected:
    public:
    void setZ(int a, int b, int c)
    {
        setPoint(a, b);
        z = c;
    }
    void show()
    {
        cout << resourceID << " (" << x << ", " << y << ", " << \
        z << ")" << endl;
    }
};
```

```
Point p1;
p1.setPoint(3, 4);
p1.show();          //50 (3, 4)
Point3D p2;
p2.setZ(5, 6, 7);
p2.show();          //50 (5, 6, 7)
```

Modifying Variable Behavior

```
class Point3D : public Point
{
    int z;
protected: int resourceID = 60;
    public:
    void setZ(int a, int b, int c)
    {
        setPoint(a, b);
        z = c;
    }
    void show()
    {
        cout << resourceID << " (" << x << ", " << y << ", " << \
        z << ")" << endl;
    }
};
```

```
Point p1;
p1.setPoint(3, 4);
p1.show();          //50 (3, 4)
Point3D p2;
p2.setZ(5, 6, 7);
p2.show();          //60 (5, 6, 7)
```

Accessing Members of Base Class

```
class Point3D : public Point
{
    int z;
protected: int resourceID = 60;
    public:
    void setZ(int a, int b, int c)
    {
        setPoint(a, b);
        z = c;
    }
    void show()
    {
        cout << Point::resourceID << " (" << x << ", " << y << ", " << \
        z << ")" << endl;
    }
};
```

```
Point p1;
p1.setPoint(3, 4);
p1.show();          //50 (3, 4)
Point3D p2;
p2.setZ(5, 6, 7);
p2.show();          //50 (5, 6, 7)
```

Modifying Visibility

Modifying Visibility

```
class Base
{
protected:
    void hiddenMethod()
    {
        cout << "This should not be publicly exposed!";
    }
public:
    void publicMethod()
    {
        cout << "This should be called from outside!";
    }
};

class Derived : public Base
{
protected:
    Base::publicMethod;
public:
    Base::hiddenMethod;
};
```

```
Base b;
b.hiddenMethod();    //Error!
b.publicMethod();    //OK

Derived d;
d.hiddenMethod();    //OK
d.publicMethod();    //Error!
```