CSE-4301
Object Oriented Programming
2022-2023

Week-6

Inheritance

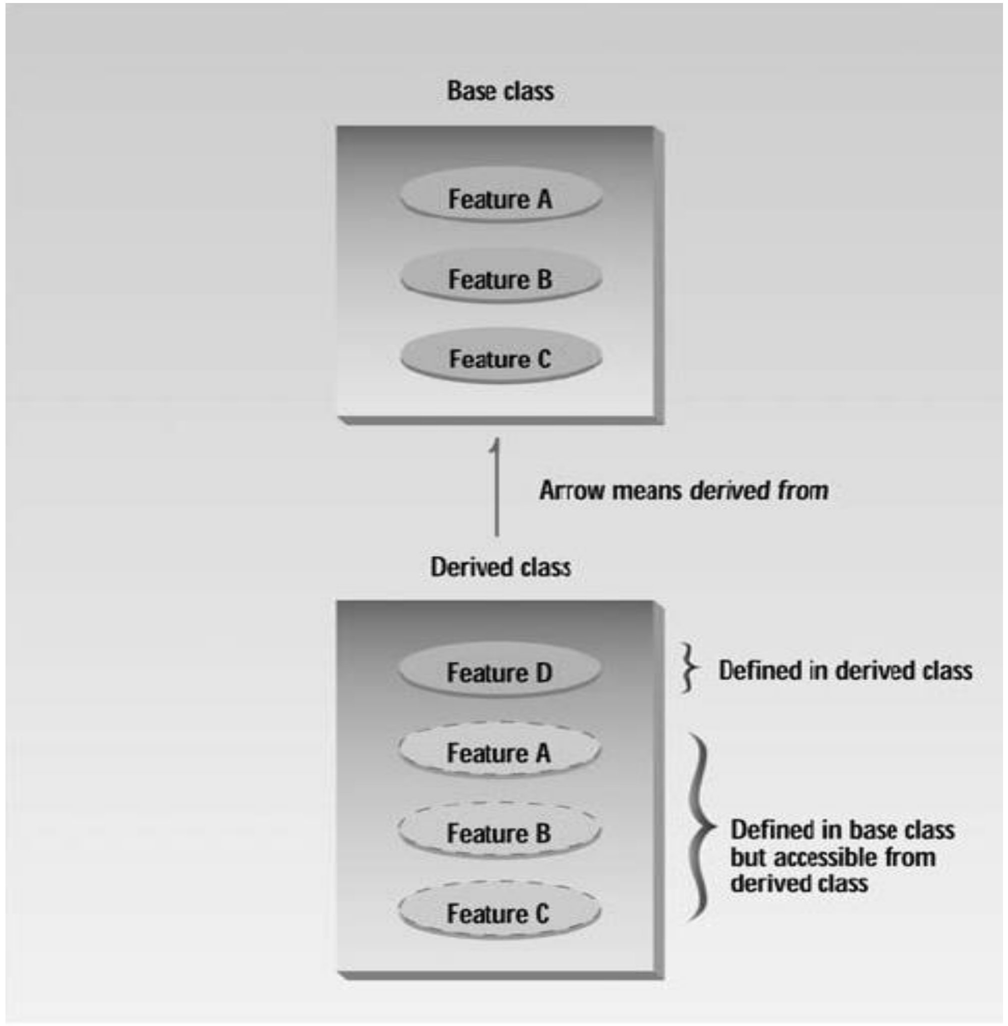
-Faisal Hussain
Assistant Professor, Department Of Computer Science And Engineering, IUT

Contents

Inheritance

- Inheritance is the process of creating new classes (called **derived classes**) from an existing class (called **Base Class**)

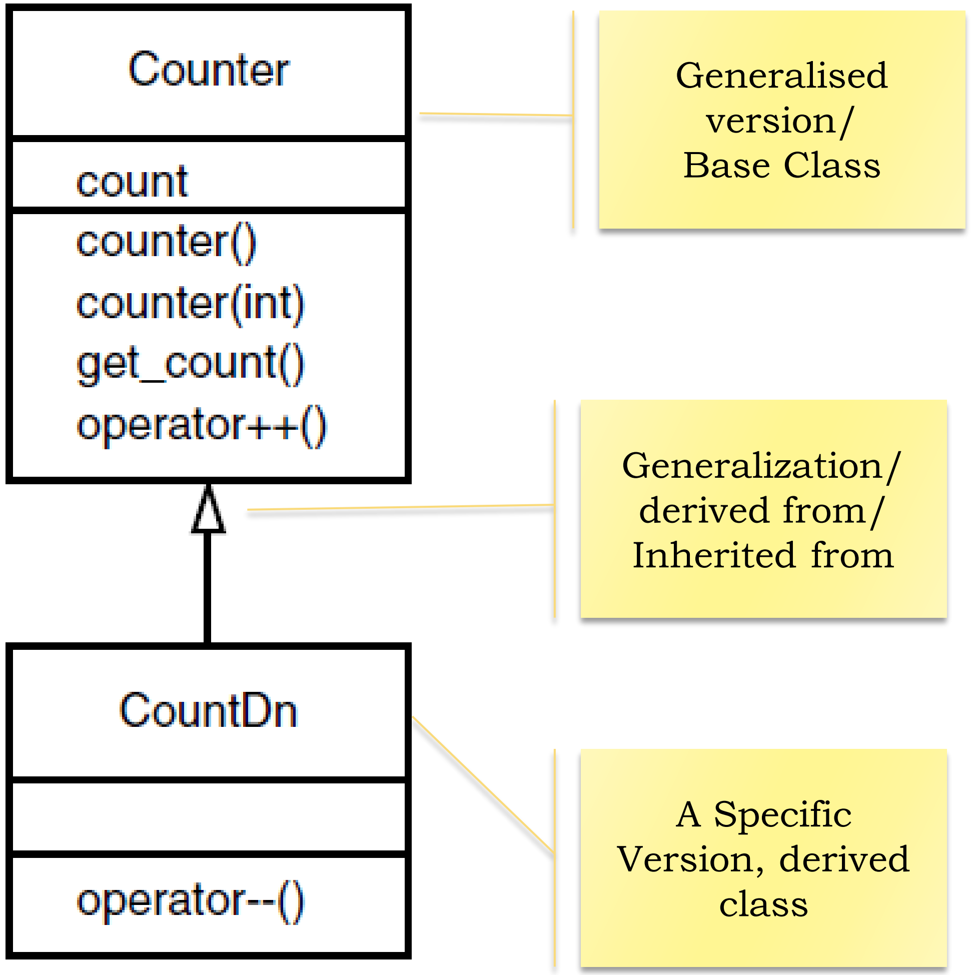
Derived Class and Base Class



- Inheritance is the process of creating new classes (called **derived classes** also known as subclass) from an existing class (called **Base Class** also known as superclass)
- **Derived Class** inherits all the features (member variable and function) of Base class.
- New features (member variables and functions) can be added.
- Moreover, some features of base class can be refined (see overriding)

Why Inheritance is important

- Code **Reusability.**
- The alternative of inheritance is – writing, debugging similar thing again and again.
- May not have access to the base class code.


Counter
count
counter()
counter(int)
get_count()
operator++()
CountDn
operator--()
Generalised version/ Base Class
Generalization/ derived from/ Inherited from
A Specific Version, derived class

Accessing Base Class Members

- Substituting Base Class Constructor

Accessibility

```
class derived : access specifier BaseClass
{
}
```

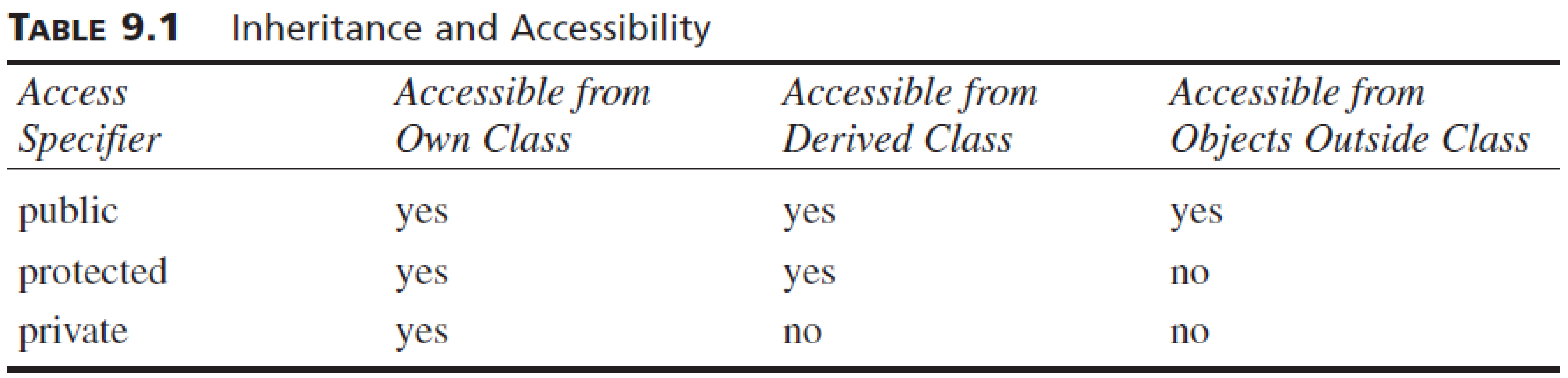


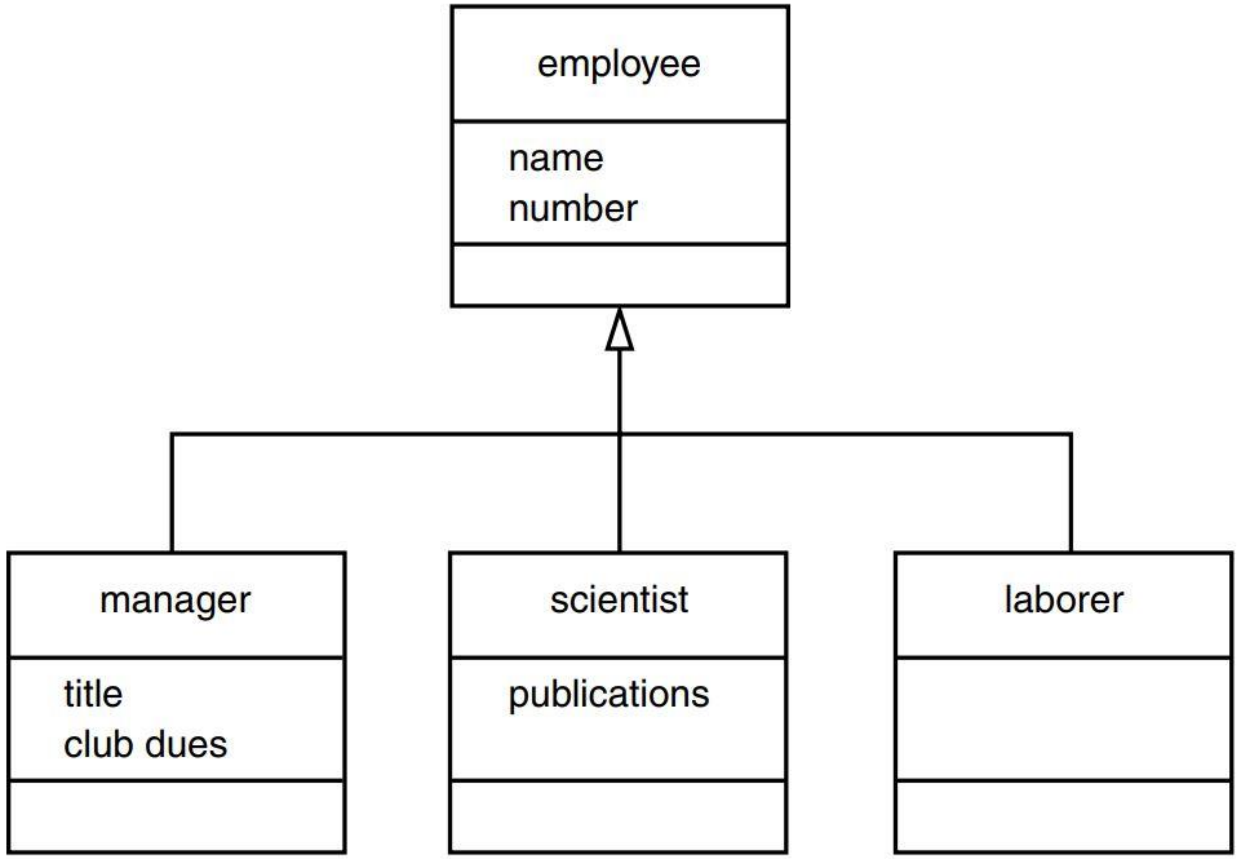
TABLE 9.1 Inheritance and Accessibility

Access Specifier	*Accessible from Own Class*	*Accessible from Derived Class*	*Accessible from Objects Outside Class*
public	yes	yes	yes
protected	yes	yes	no
private	yes	no	no

Overriding & Dominating Inherited Members

- Difference between **Overriding and Overloading ?**
- A function with **same name and same signature** (parameters and their type) can be defined in a derived class which is already present in the base class. The function of derived class override the function of the base class.
- An object of derive class will execute the **overriding function** [function which is defined in the derived class] (if present).
- **Using Scope resolution operator you can still call the base class function.

Class Hierarchies

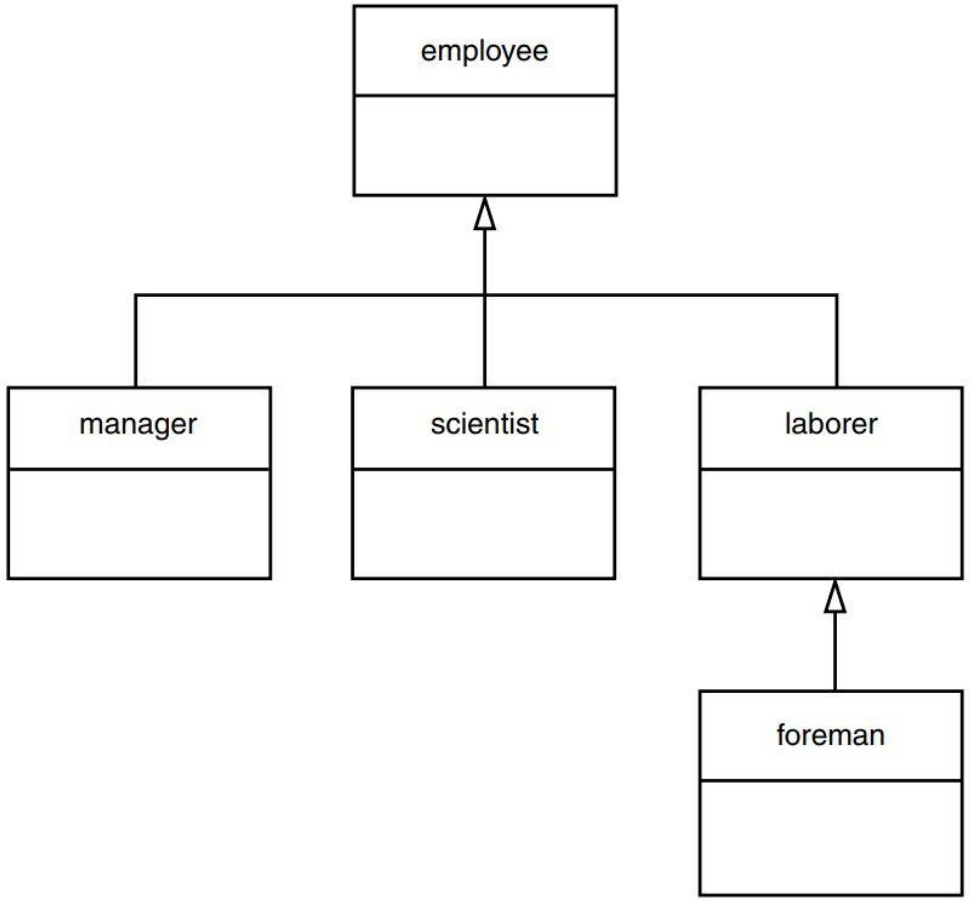


Type of Inheritance

- public Inheritance :
  - **public** member of base class -> **public** in derived class
  - **protected** member of base class -> **protected** in derived class
  - **private** member of base class -> **private** in derived class
- protected Inheritance :
  - **public** member of base class -> **protected** in derived class
  - **protected** member of base class -> **protected** in derived class
  - **private** member of base class -> **private** in derived class
- private Inheritance :
  - **public** member of base class -> **private** in derived class
  - **protected** member of base class -> **private** in derived class
  - **private** member of base class -> **private** in derived class

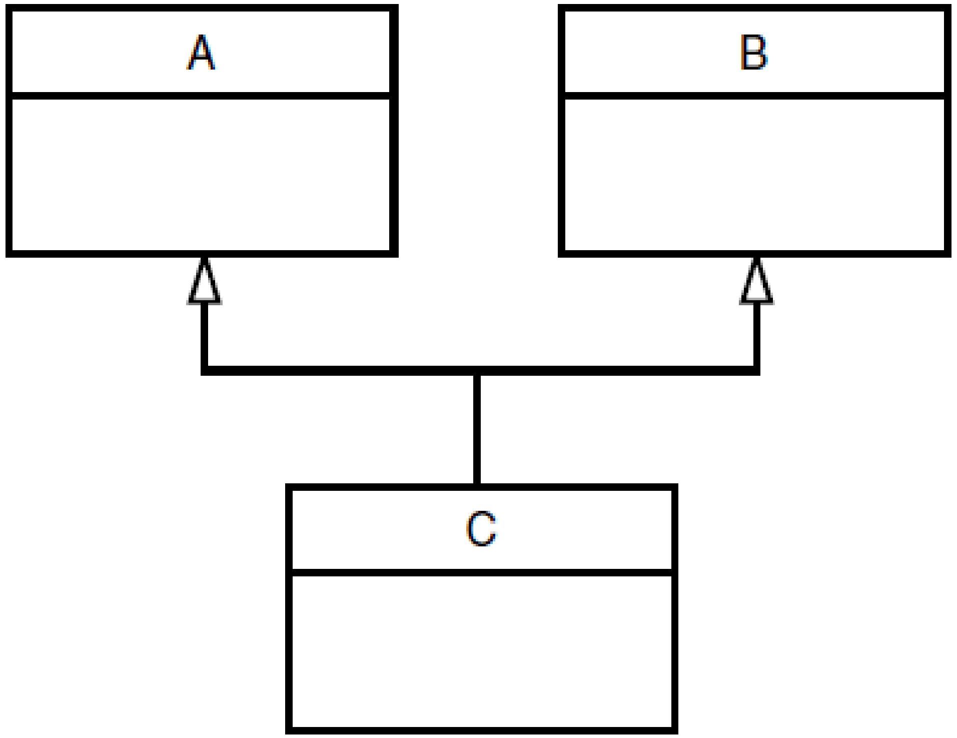
Level of Inheritance

```
class employee { };


class laborer : public employee
{
};


class foreman : public laborer
{
};
```



Multiple Inheritance

```
class A // base class A
{
};
class B // base class B
{
};
class C : public A, public B
// C is derived from A and B
{
};
```



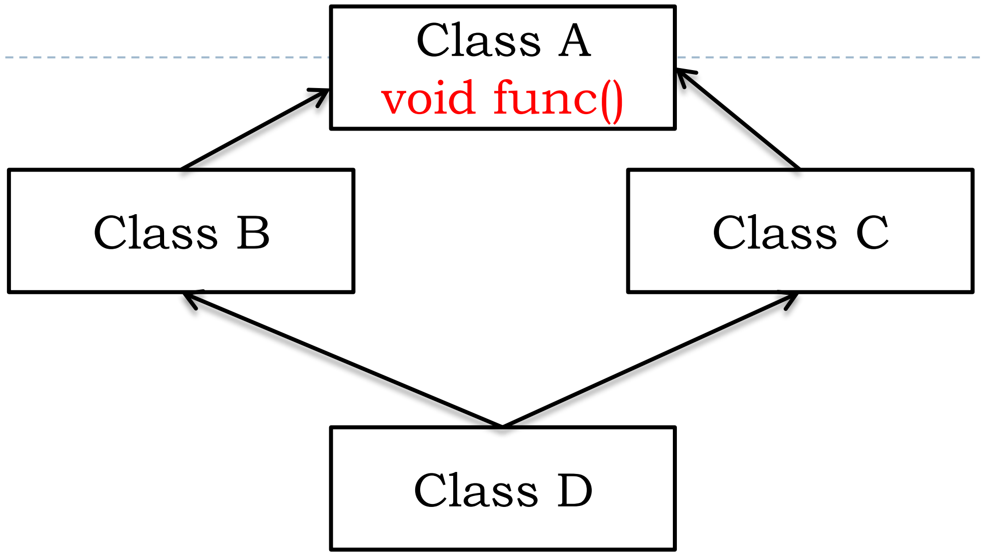
Ambiguity in Multiple Inheritance

```
#include <iostream>
using namespace std;
class A
{
public:
  void show() { cout << "Class A\n"; }
};
class B
{
public:
  void show() { cout << "Class B\n"; }
};
class C : public A, public B
{
};
```

```
int main()
{
C objC; //object of class C
// objC.show();
//ambiguous--will not compile
objC.A::show(); //OK
objC.B::show(); //OK
return 0;
}
```

Ambiguity in Multiple Inheritance (Diamond shape inheritance tree)

```
#include <iostream>
using namespace std;
class A
{
  public:
        void func();
};
class B : public A
{};
class C : public A
{};
class D : public B, public C
{};
```

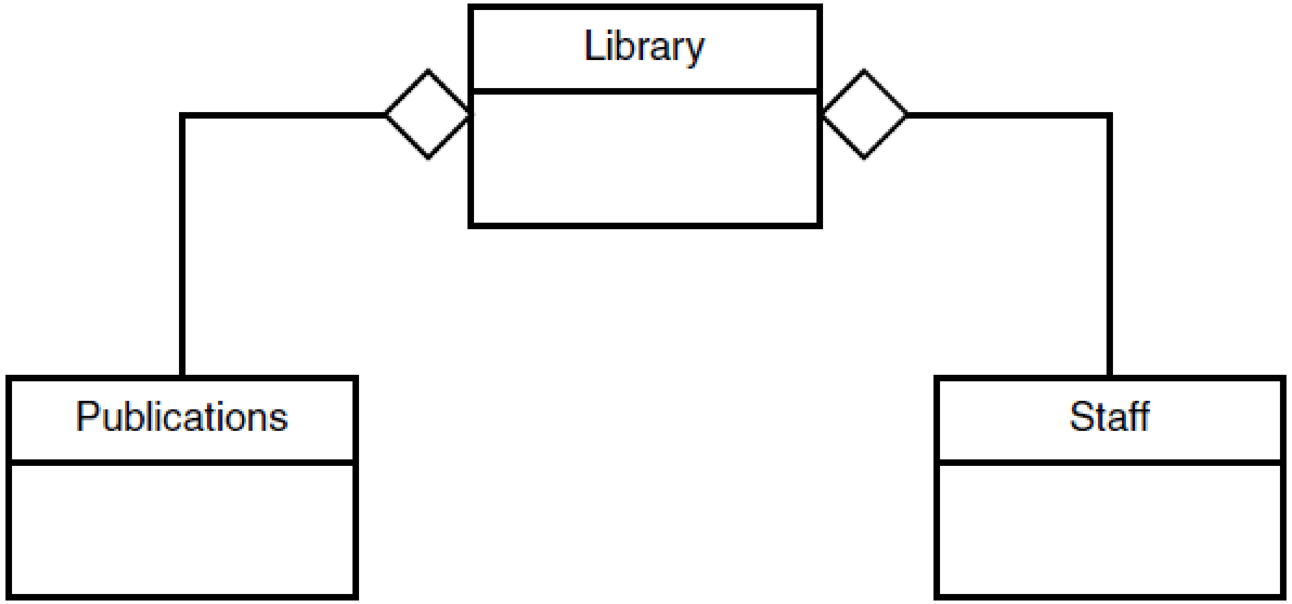


```
int main()
{
 D objD;
 objD.func();
 //ambiguous: won't compile
 return 0;
}
```

Aggregation: Classes Within Classes

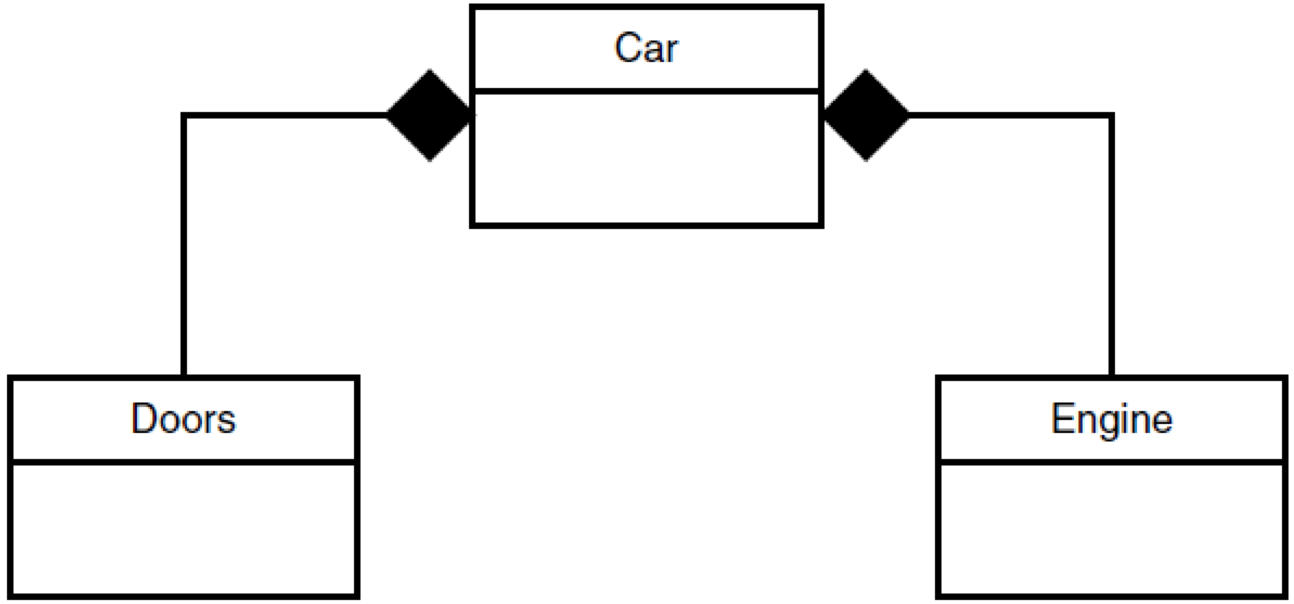
- **Has a** relationship.

```
class Publication
{ };
class Staff
{ };
class Library
{
        Publication p[1000];
        Staff s[50]
};
```



Composition: A Stronger Aggregation

- stronger form of aggregation. It has all the characteristics of aggregation, plus
- two more:
  - The part may belong to only one whole.
  - The lifetime of the part is the same as the lifetime of the whole.
- "Has a" -> "Consists of"



Reading Assignment

- **Chapter- 10,12**
  - **Schaum's Outline Programming with C++**

    --John R. Hubbard
- **Chapter- 9**
  - **Object Oriented Programming in C++ -- Robert Lafore**