Inheritance

Ho Dac Hung

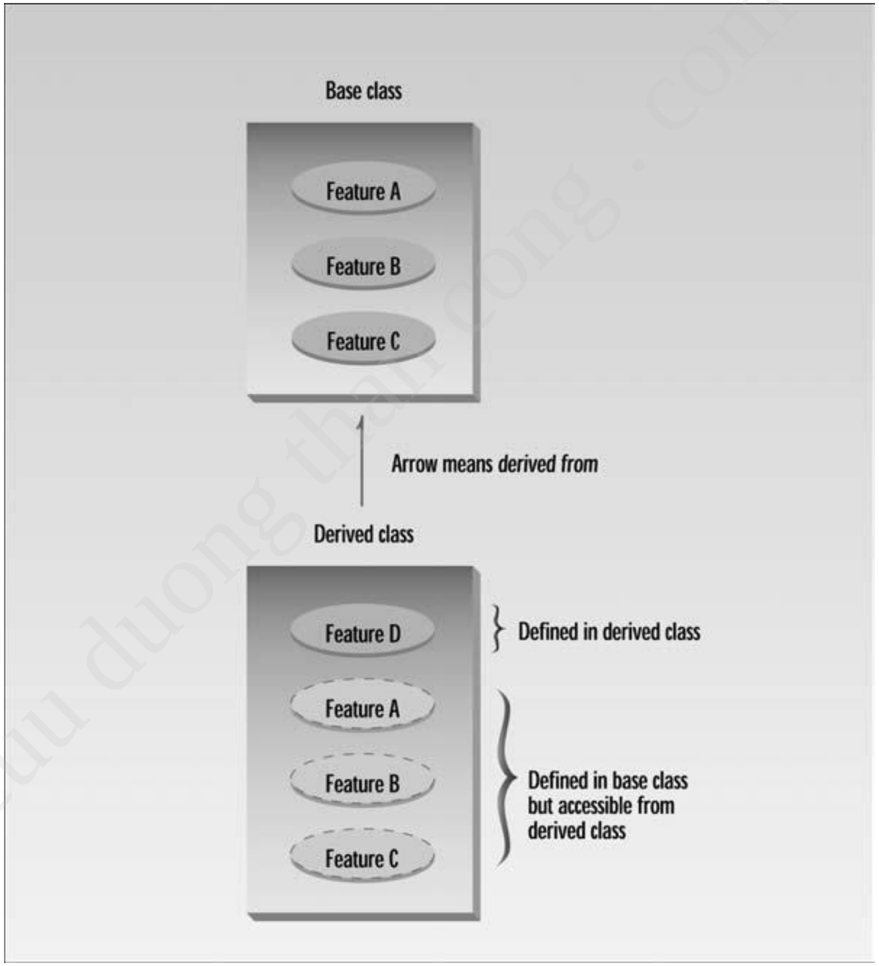
cuu duong than cong . com

Inheritance

- Inheritance is probably the most powerful feature of object-oriented programming, after classes
- themselves. Inheritance is the process of creating new classes, called derived classes, from
- existing or base classes. The derived class inherits all the capabilities of the base class but
- can add embellishments and refinements of its own. The base class is unchanged by this
- process.

cuu duong than cong . com

Inheritance



cuu duong than cong . com

Inheritance

- Inheritance is an essential part of OOP. Its big payoff is that it permits code reusability. Once a base class is written and debugged, it need not be touched again, but, using inheritance, can nevertheless be adapted to work in different situations. Reusing existing code saves time andmoney and increases a program's reliability. Inheritance can also help in the original conceptualization of a programming problem, and in the overall design of the program.

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Inheritance

- An important result of reusability is the ease of distributing class libraries. A programmer can use a class created by another person or company, and, without modifying it, derive other classes from it that are suited to particular situations.

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Derived Class and Base Class

- We could insert a decrement routine directly into the source code of the Counter class. However, there are several reasons that we might not want to do this.
- We can use inheritance to create a new class based on Counter, without modifying Counter itself.

cuu duong than cong. com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
class Counter //base class
{
    protected: //NOTE: not private
        unsigned int count; //count
    public:
        Counter() : count(0) //no-arg constructor
        { }
        Counter(int c) : count(c) //1-arg constructor
        { }
        unsigned int get_count() const //return count
        { return count; }
        Counter operator ++ () //incr count (prefix)
        { return Counter(++count); }
};
```

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
class CountDn : public Counter //derived class
{
    public:
        Counter operator -- () //decr count (prefix)
        { return Counter(--count); }
};
```

cuu duong than cong . com

```
int main()
{
    CountDn c1; //c1 of class CountDn
    cout << “\nc1=” << c1.get_count(); //display c1
    ++c1; ++c1; ++c1; //increment c1, 3 times
    cout << “\nc1=” << c1.get_count(); //display it
    --c1; --c1; //decrement c1, twice
    cout << “\nc1=” << c1.get_count(); //display it
    cout << endl;
    return 0;
}
```

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Accessing Base Class Members

- An important topic in inheritance is knowing when a member function in the base class can be used by objects of the derived class. This is called accessibility.

cuu duong than cong. com

Accessing Base Class Members

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

cuu duong than cong . com

Derived Class Constructors

- To initialize any variables, whether they’re in the derived class or the base class, before any statements in either the derived or base-class constructors are executed. By calling the baseclass constructor before the derived-class constructor starts to execute, we accomplish this.

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
class Counter
{
    protected: //NOTE: not private
        unsigned int count; //count
    public:
        Counter() : count() //constructor, no args
        { }
        Counter(int c) : count(c) //constructor, one arg
        { }
        unsigned int get_count() const //return count
        { return count; }
        Counter operator ++ () //incr count (prefix)
        { return Counter(++count); }
};
```

cuuduongthancong . com

```
class CountDn : public Counter
{
    public:
        CountDn() : Counter() //constructor, no args
        { }
        CountDn(int c) : Counter(c) //constructor, 1 arg
        { }
        CountDn operator -- () //decr count (prefix)
        { return CountDn(--count); }
};
```

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Overriding Member Functions

- You can use member functions in a derived class that override—that is, have the same name as—those in the base class. You might want to do this so that calls in your program work the same way for objects of both base and derived classes.

cuu duong than cong . com

```
class Stack
{
    protected: //NOTE: can’t be private
        enum { MAX = 3 }; //size of stack array
        int st[MAX]; //stack: array of integers
        int top; //index to top of stack
    public:
        Stack() //constructor
        { top = -1; }
        void push(int var) //put number on stack
        { st[++top] = var; }
        int pop() //take number off stack
        { return st[top--]; }
};
```

cuuduongthancong.com

```
class Stack2 : public Stack
{
    public:
        void push(int var) //put number on stack
        {
            if(top >= MAX-1) //error if stack full
            { cout << “\nError: stack is full”; exit(1); }
            Stack::push(var); //call push() in Stack class
        }
        int pop() //take number off stack
        {
            if(top < 0) //error if stack empty
            { cout << “\nError: stack is empty\n”; exit(1); }
            return Stack::pop(); //call pop() in Stack class
        }
};
```

cuu duong than cong . com

```
int main()
{
    Stack2 s1;
    s1.push(11); //push some values onto stack
    s1.push(22);
    s1.push(33);
    cout << endl << s1.pop();
    cout << endl << s1.pop();
    cout << endl << s1.pop();
    cout << endl << s1.pop();
    cout << endl;
    return 0;
}
```

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Which Function Is Used?

- When the same function exists in both the base class and the derived class, the function in the derived class will be executed.

cuu duong than cong . com

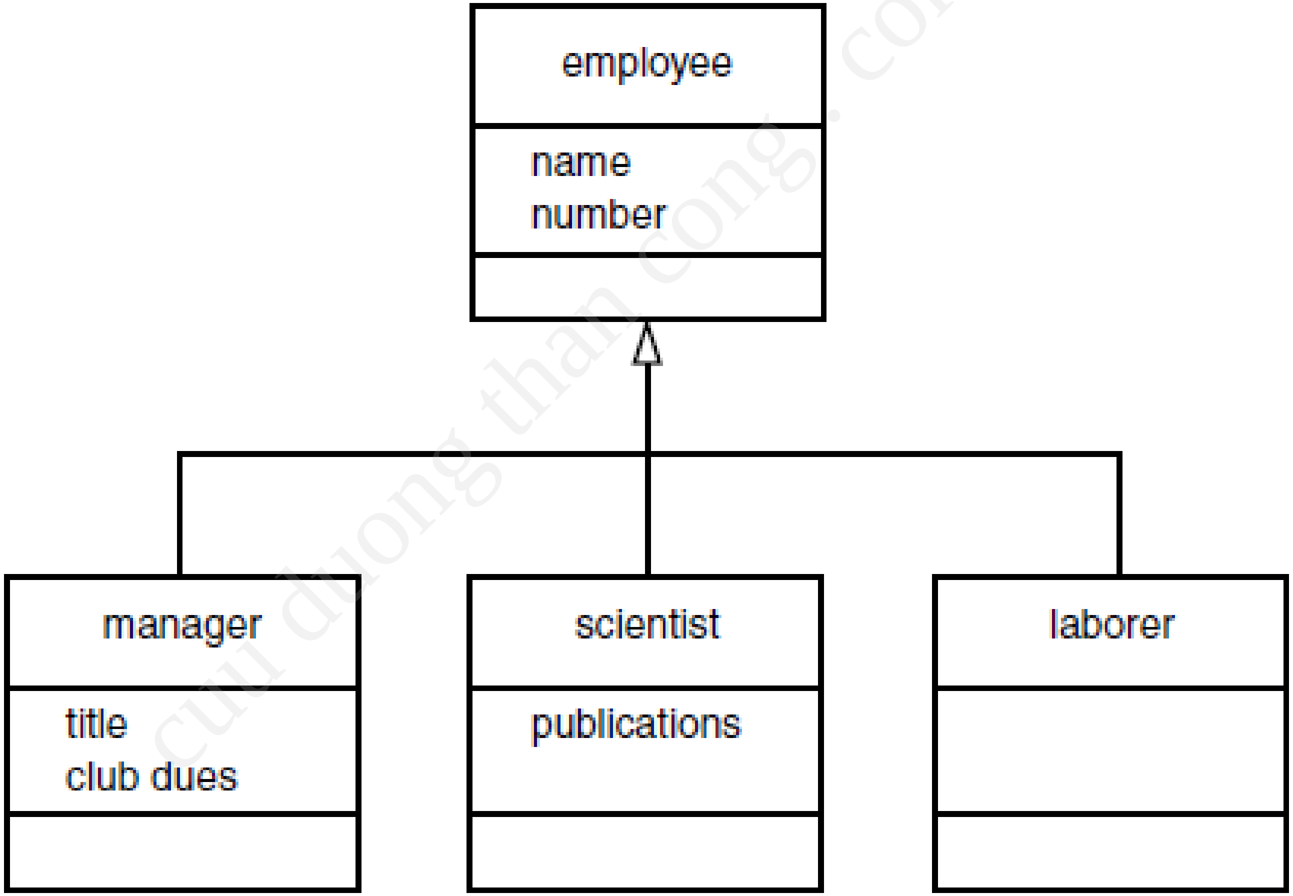
Class Hierarchies

- We’ve simplified the situation so that only three kinds of employees are represented. Managers manage, scientists perform research to develop better widgets, and laborers operate the dangerous widget-stamping presses.

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Class Hierarchies



cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
class employee //employee class
{
    private:
        char name[LEN]; //employee name
        unsigned long number; //employee number
    public:
        void getdata()
        {
            cout << “\n Enter last name: “; cin >> name;
            cout << “ Enter number: “; cin >> number;
        }
        void putdata() const
        {
            cout << “\n Name: “ << name;
            cout << “\n Number: “ << number;
        }
};
```

cuu duong than cong . com

```
class manager : public employee //management class
{
    private:
        char title[LEN]; //”vice-president” etc.
        double dues; //golf club dues
    public:
        void getdata()
        {
            employee::getdata();
            cout << “ Enter title: “; cin >> title;
            cout << “ Enter golf club dues: “; cin >> dues;
        }
        void putdata() const
        {
            employee::putdata();
            cout << “\n Title: “ << title;
            cout << “\n Golf club dues: “ << dues;
        }
};
```

cuu duong than cong . com

```
class scientist : public employee //scientist class
{
private:
int pubs; //number of publications
public:
void getdata()
{
employee::getdata();
cout << “ Enter number of pubs: “; cin >> pubs;
}
void putdata() const
{
employee::putdata();
cout << “\n Number of publications: “ << pubs;
}
};
```

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
class laborer : public employee //laborer class
{
};
```

cuu duong than cong . com

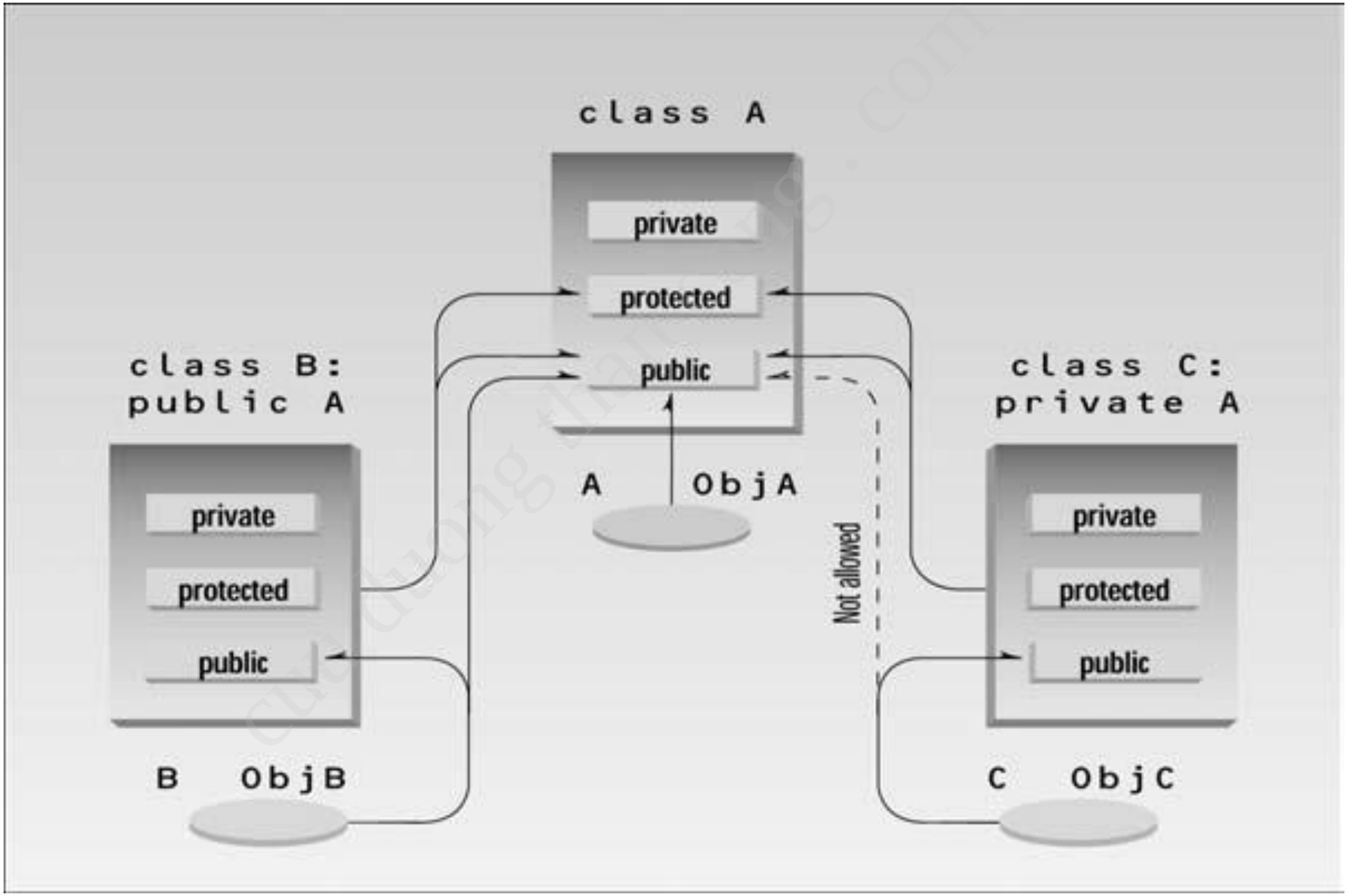
“Abstract” Base Class

- Classes used only for deriving other classes, are sometimes loosely called abstract classes, meaning that no actual instances (objects) of this class are created.

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

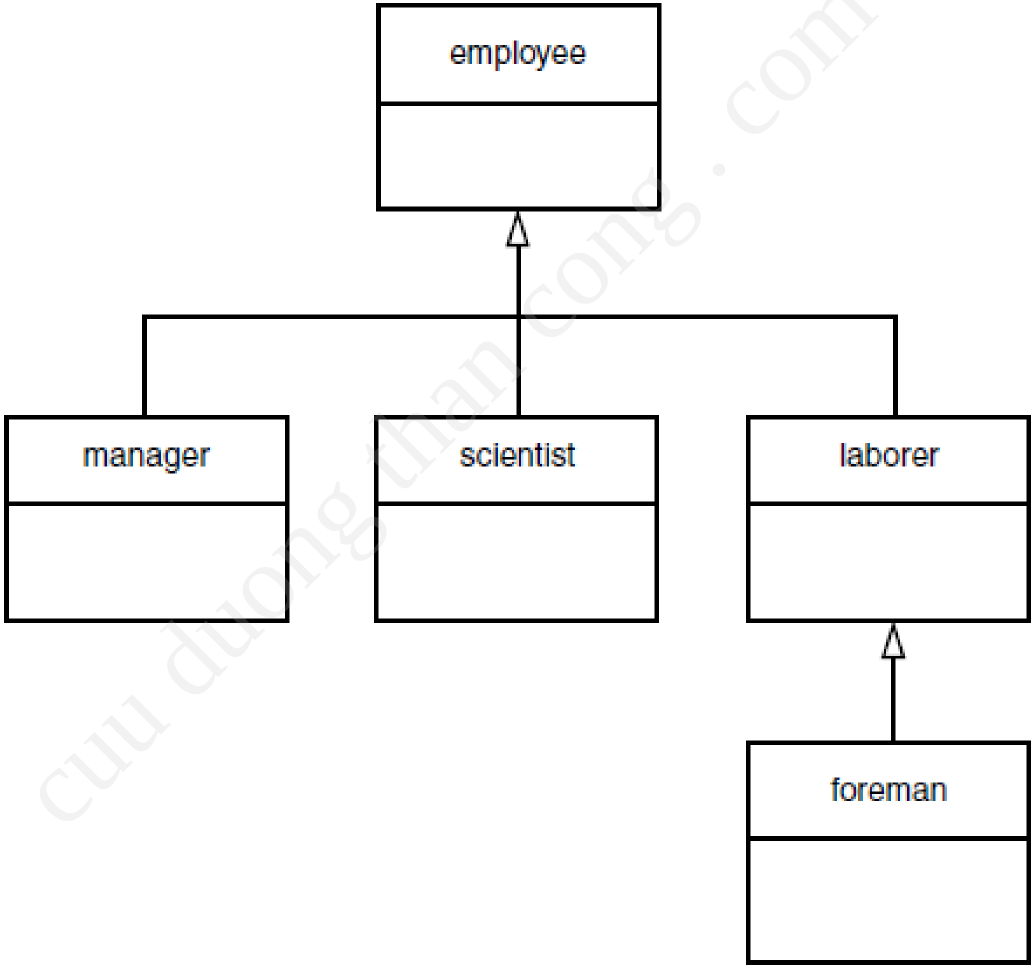
Public and Private Inheritance



cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

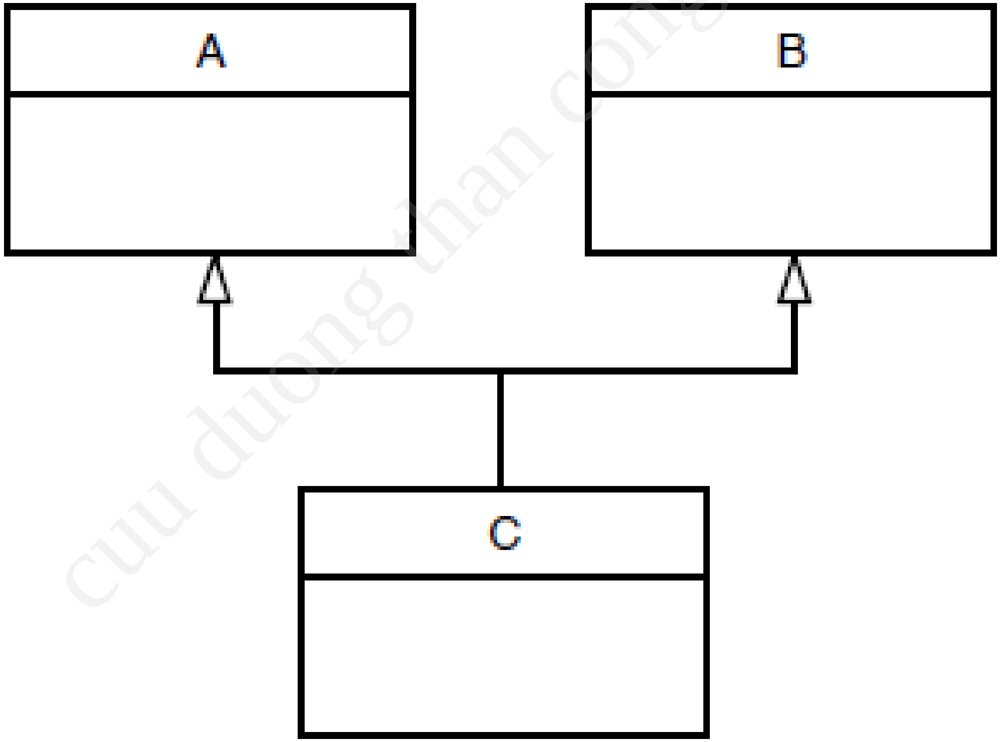
Levels of Inheritance



cuu duong than cong . com

Multiple Inheritance

- A class can be derived from more than one base class. This is called multiple inheritance.



cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

Ambiguity in Multiple Inheritance

- Two base classes have functions with the same name, while a class derived from both base classes has no function with this name. How do objects of the derived class access the correct base class function?

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

```
class A
{
    public:
        void show() { cout << “Class A\n”; }
};
class B
{
    public:
        void show() { cout << “Class B\n”; }
};
class C : public A, public B
{
};
```

cuu duong than cong . com

Ambiguity in Multiple Inheritance

- Another kind of ambiguity arises if you derive a class from two classes that are each derived from the same class. This creates a diamond-shaped inheritance tree.

cuu duong than cong . com

CuuDuongThanCong.com https://fb.com/tailieudientucntt

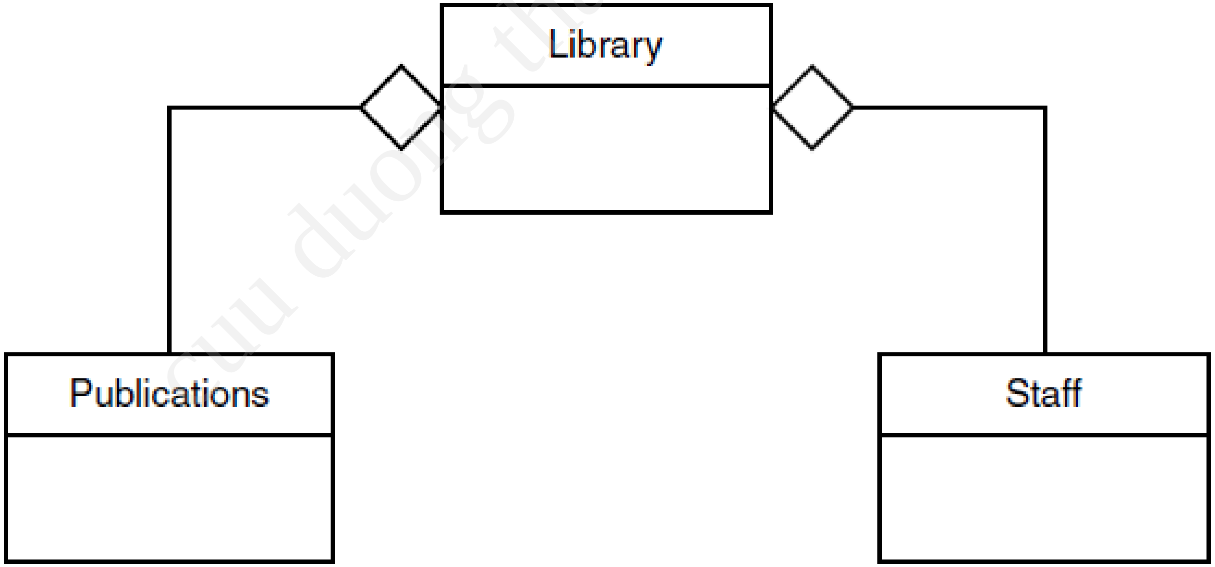
```
class A
{
    public:
        void func();
};
class B : public A
{ };
class C : public A
{ };
class D : public B, public C
{ };
```

cuu duong than cong . com

Aggregation: Classes Within Classes

- Aggregation is called a “has a” relationship. We say a library has a book or an invoice has an item line. Aggregation is also called a “part-whole” relationship: the book is part of the library.

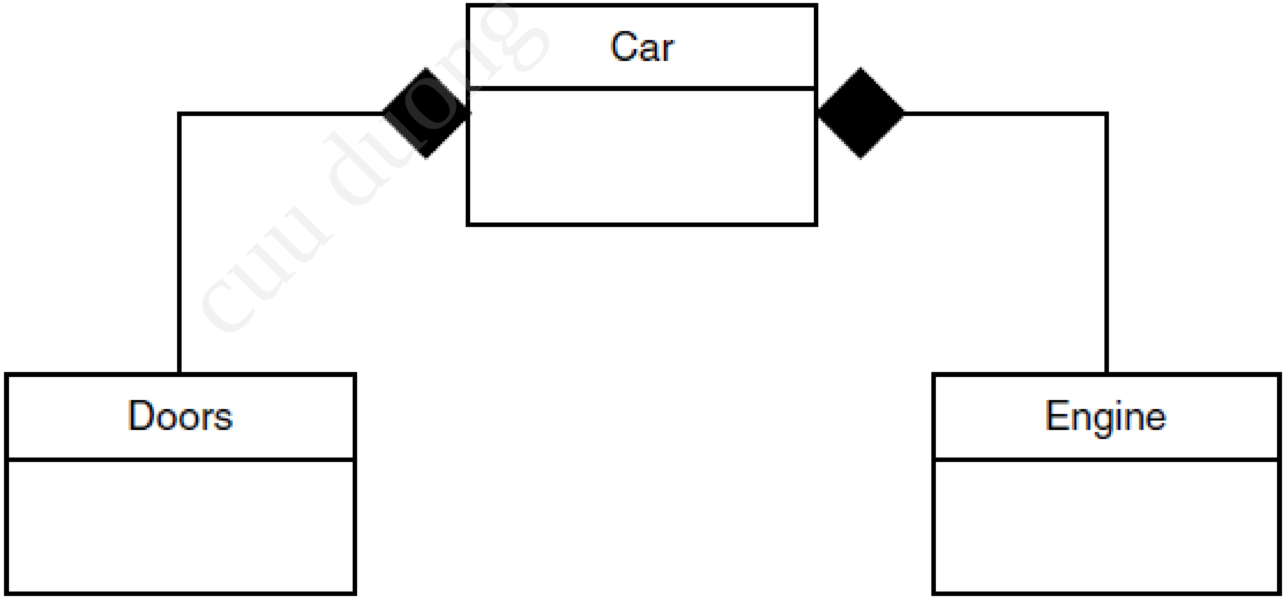
cuu duong than cong . com



CuuDuongThanCong.com https://fb.com/tailieudientucntt

Composition: A Stronger Aggregation

- Composition is a 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.



cuu duong than cong . com