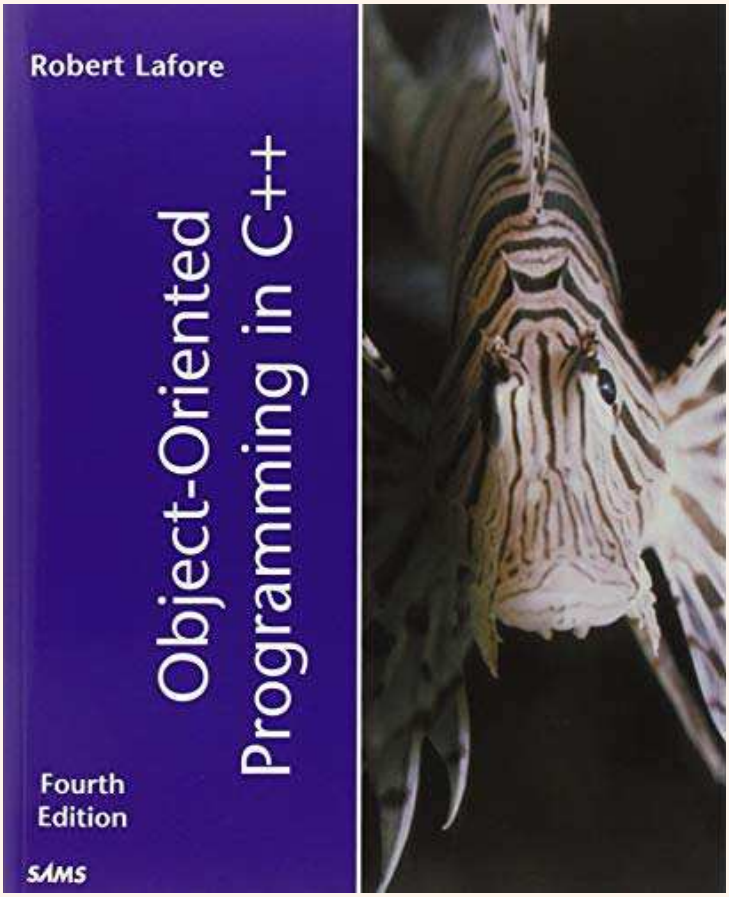


Chapter 9

Inheritance

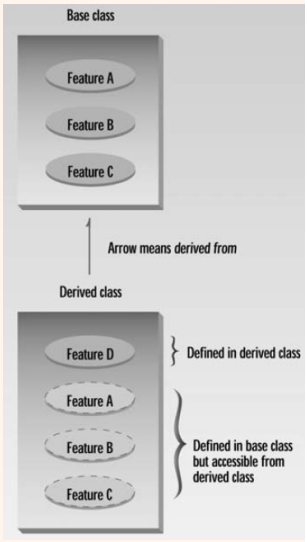
Animated Version

Topics

- Derived Class and Base Class
- Derived Class Constructors
- Overriding Member Functions
- Which Function Is Used?
- Inheritance in the English Distance Class
- Class Hierarchies
- Inheritance and Graphics Shapes
- Public and Private Inheritance
- Levels of Inheritance
- Multiple Inheritance
- `private` Derivation in `EMPMULT`
- Ambiguity in Multiple Inheritance
- Aggregation: Classes Within Classes
- Inheritance and Program Development

Introduction

- 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 refinements of its own.
- Permits code reusability:
  - saves time and money and increases a program's reliability.
  - ease of distributing class libraries.
  - 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.

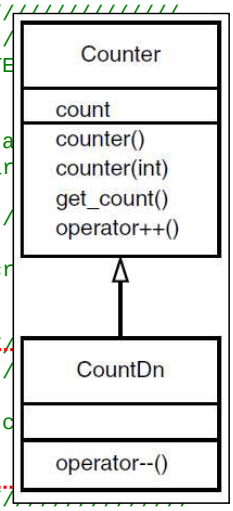


Derived Class and Base Class

- Specifying the Derived Class
  - `CountDn` is *derived from* the *base class* `Counter`
- Generalization in UML Class Diagrams
- Accessing Base Class Members
  - Substituting Base Class Constructors
    - if you don't specify a constructor, the derived class will use an appropriate constructor from the base class.
  - Substituting Base Class Member Functions

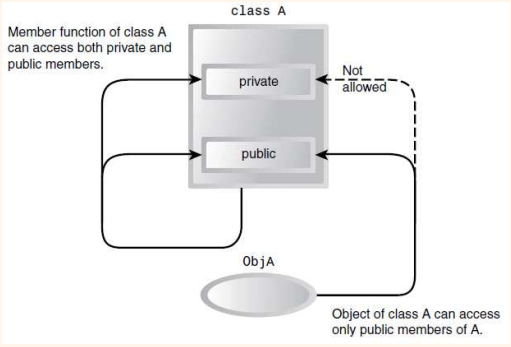
```
// counten.cpp
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////////
class Counter                                             /
   {   private:                                  //NOTE
         unsigned int count;
   public:
      Counter() : count(0) {  }                  //no-a
      Counter(int c) : count(c) {  }//1-ar

      unsigned int get_count() const              /
         { return count; }
      Counter operator ++ ()                      //incr
         { return Counter(++count); }
   };
////////////////////////////////////////////////////////////
class CountDn : public Counter                            /
   {  public:
      Counter operator -- ()                 //decr c
         { return Counter(--count); }
   };
////////////////////////////////////////////////////////////
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;
   }
```

```
Output:
c1=0    <------  after initialization
c1=3    <------  after ++c1, ++c1, ++c1
c1=1    <------  after --c1, --c1
```



Derived Class and Base Class (2)

- The protected Access Specifier
  - if you are writing a class that you suspect might be used, at any point in the future, as a base class for other classes, then any member data that the derived classes might need to access should be made protected rather than private . This ensures that the class is “inheritance ready.”
  - Dangers of protected: protected members considerably less secure than private members.
- Base Class Unchanged
- Other Terms:
  - Base class: superclass/parent class
  - Derived class: subclass/child class



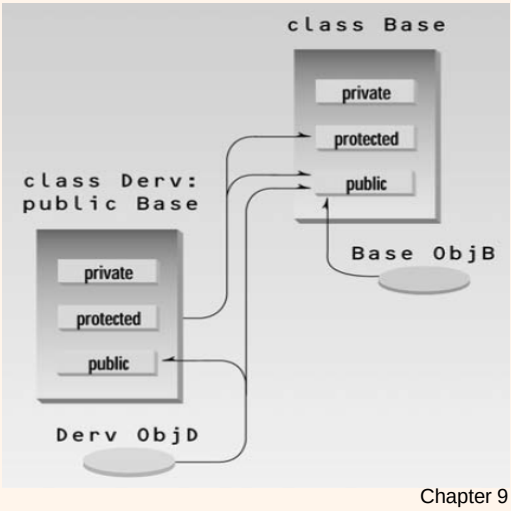


TABLE 9.1 Inheritance and Accessibility

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

Derived Class Constructors

- Calling a constructor from the initialization list calls the base-class constructor before the derived-class constructor starts to execute.

```
// counten2.cpp
// constructors in derived class
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////////
class Counter
   {
   protected:                          //NOTE: not private
      unsigned int count;              //count
   public:
      Counter() : count(0)             //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); }
   };
////////////////////////////////////////////////////////////
```

```
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); }
   };
////////////////////////////////////////////////////////////
int main()
   {
   CountDn c1;                         //class CountDn
   CountDn c2(100);

   cout << "\nc1=" << c1.get_count();   //display
   cout << "\nc2=" << c2.get_count();   //display

   ++c1; ++c1; ++c1;                   //increment c1
   cout << "\nc1=" << c1.get_count();   //display it

   --c2; --c2;                         //decrement c2
   cout << "\nc2=" << c2.get_count();   //display it

   CountDn c3 = --c2;                  //create c3 from c2
   cout << "\nc3=" << c3.get_count();   //display c3
   cout << endl;
   return 0;
   }
```

Overriding Member Functions

- Derived class can have member function with same name as base (overloaded) –> function overriding.
- Which Function Is Used?
  - Generally, the function in the derived class will be executed.
  - Use scope resolution to call base class's function.

```
// staken.cpp
// overloading functions in base and derived classes
#include <iostream>
using namespace std;
#include <process.h>              //for exit()
////////////////////////////////////////////////////////////
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--]; }
   };
```

```
////////////////////////////////////////////////////////////
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
         }
   };
////////////////////////////////////////////////////////////
int main()
   {
   Stack2 s1;

   s1.push(11);                   //
   s1.push(22);
   s1.push(33);

   cout << endl << s1.pop();      //pop some values from stack
   cout << endl << s1.pop();
   cout << endl << s1.pop();
   cout << endl << s1.pop();      //oops, popped one too many...
   cout << endl;
   return 0;
   }
```

Output:

```
33
22
11
Error: stack is empty
```


Inheritance in the English `Distance` Class

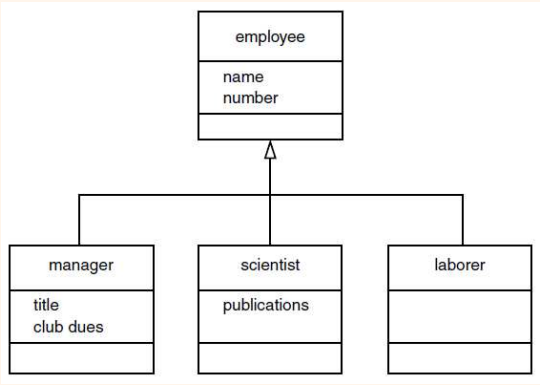
```
// englen.cpp
// inheritance using English Distances
#include <iostream>
using namespace std;
enum posneg { pos, neg };         //for sign in DistSign
////////////////////////////////////////////////////////////
    /
class Distance                    //English Distance class
   {
   protected:                     //NOTE: can't be private
      int feet;
      float inches;
   public:                        //no-arg constructor
      Distance() : feet(0), inches(0.0)
         {  }                     //2-arg constructor)
      Distance(int ft, float in) : feet(ft), inches(in)
         {  }
      void getdist()              //get length from user
         {
         cout << "\nEnter feet: ";  cin >> feet;
         cout << "Enter inches: ";  cin >> inches;
         }
      void showdist() const       //display distance
         { cout << feet << "\'-" << inches << '\"'; }
   };
////////////////////////////////////////////////////////////
    /
class DistSign : public Distance  //adds sign to Distance
   {
   private:
      posneg sign;                //sign is pos or neg
   public:
                                  //no-arg constructor
      DistSign() : Distance()     //call base constructor
         { sign = pos; }          //set the sign to +

                                  //2- or 3-arg constructor
      DistSign(int ft, float in, posneg sg=pos) :
            Distance(ft, in)      //call base constructor
         { sign = sg; }           //set the sign
```

```
      void getdist()              //get length from user
            {
            Distance::getdist();     //call base getdist()
            char ch;                 //get sign from user
            cout << "Enter sign (+ or -): ";  cin >> ch;
            sign = (ch=='+') ? pos : neg;
            }
      void showdist() const       //display distance
            {
            cout << ( (sign==pos) ? "(+)" : "(-)" );//show sign
            Distance::showdist();                   //ft and in
            }
   };
////////////////////////////////////////////////////////////
int main()
   {
   DistSign alpha;                //no-arg constructor
   alpha.getdist();               //get alpha from user

   DistSign beta(11, 6.25);       //2-arg constructor

   DistSign gamma(100, 5.5, neg); //3-arg constructor

                                  //display all distances
   cout << "\nalpha = ";  alpha.showdist();
   cout << "\nbeta = ";  beta.showdist();
   cout << "\ngamma = ";  gamma.showdist();
   cout << endl;
   return 0;
   }
```

```
Enter feet: 6
Enter inches: 2.5
Enter sign (+ or -): -

alpha = (-)6'-2.5"
beta = (+)11'-6.25"
gamma = (-)100'-5.5"
```

- Constructors in DistSign
- Member Functions in DistSign

Class Hierarchies




```
// employ.cpp
#include <iostream>
using namespace std;
const int LEN = 80;                 //maximum length of names
////////////////////////////////////////////////////////////
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;
         }
   };
```

```
////////////////////////////////////////////////////////////
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;
         }
   };

////////////////////////////////////////////////////////////
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;
         }
   };
```


Class Hierarchies

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

////////////////////////////////////////////////////////////
int main()
   {
   manager m1, m2;
   scientist s1;
   laborer l1;

   cout << endl;      //get data for several employees
   cout << "\nEnter data for manager 1";
   m1.getdata();

   cout << "\nEnter data for manager 2";
   m2.getdata();

   cout << "\nEnter data for scientist 1";
   s1.getdata();

   cout << "\nEnter data for laborer 1";
   l1.getdata();
                     //display data for several employees
   cout << "\nData on manager 1";
   m1.putdata();

   cout << "\nData on manager 2";
   m2.putdata();

   cout << "\nData on scientist 1";
   s1.putdata();

   cout << "\nData on laborer 1";
   l1.putdata();
   cout << endl;
   return 0;
   }
```

```
Enter data for manager 1
   Enter last name: Wainsworth
   Enter number: 10
   Enter title: President
   Enter golf club dues: 1000000
Enter data on manager 2
   Enter last name: Bradley
   Enter number: 124
   Enter title: Vice-President
   Enter golf club dues: 500000
Enter data for scientist 1
   Enter last name: Hauptman-Frenglish
   Enter number: 234234
   Enter number of pubs: 999
Enter data for laborer 1
   Enter last name: Jones
   Enter number: 6546544
```

The program then plays it back.

```
Data on manager 1
   Name: Wainsworth
   Number: 10
   Title: President
   Golf club dues: 1000000
Data on manager 2
   Name: Bradley
   Number: 124
   Title: Vice-President
   Golf club dues: 500000
Data on scientist 1
    Name: Hauptman-Frenglish
   Number: 234234
   Number of publications: 999
Data on laborer 1
   Name: Jones
   Number: 6546544
```

- Abstract Base class

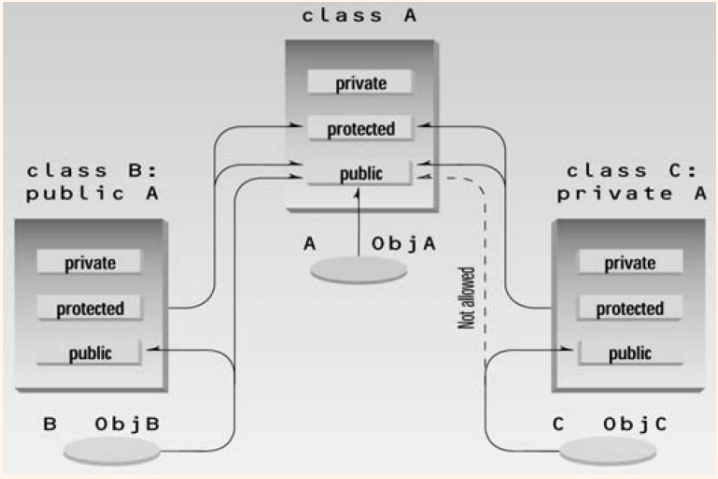
Public and Private Inheritance

- Public inheritance:
  - functions in the derived classes can access protected and public data/member functions in the base class.
  - objects of the derived class are able to access only public (not private/protected) data/member functions data of the base class.
- Private inheritance:
  - no member (public/private/protected) of the base class is accessible to objects of the derived class. They can only access the public members of their own derived class.
- Objects of the publicly derived class B can access public members of the base class A , while objects of the privately derived class C cannot; they can only access the public members of their own derived class.

Inheritance in the English `Distance` Class

```
// pubpriv.cpp
// tests publicly- and privately-derived classes
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////////////
class A                  //base class
   {
   private:
      int privdataA;     //(functions have the same access
   protected:            //rules as the data shown here)
      int protdataA;
   public:
      int pubdataA;
   };
////////////////////////////////////////////////////////////////
class B : public A       //publicly-derived class
   {
   public:
      void funct()
         {
         int a;
         a = privdataA;  //error: not accessible
         a = protdataA;  //OK
         a = pubdataA;   //OK
         }
   };
////////////////////////////////////////////////////////////////
class C : private A      //privately-derived class
   {
   public:
      void funct()
         {
         int a;
         a = privdataA;  //error: not accessible
         a = protdataA;  //OK
         a = pubdataA;   //OK
         }
   };
```

```
////////////////////////////////////////////////////////////////
int main()
   {
   int a;

   B objB;
   a = objB.privdataA;   //error: not accessible
   a = objB.protdataA;   //error: not accessible
   a = objB.pubdataA;    //OK (A public to B)

   C objC;
   a = objC.privdataA;   //error: not accessible
   a = objC.protdataA;   //error: not accessible
   a = objC.pubdataA;  //error: not accessible (A private to C)
   return 0;
   }
```

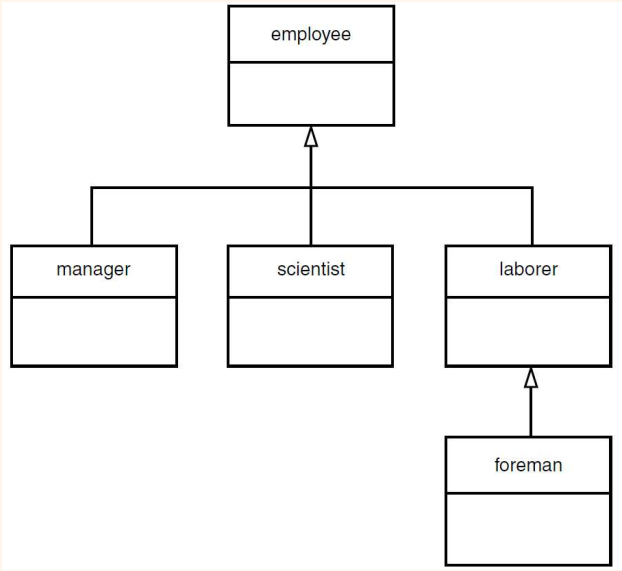


Access Specifiers: When to Use What

- Public inheritance:
  - Cases when a derived class exists to offer an improved—or a more specialized—version of the base class.
  - objects of the derived class access the public functions of the base class if they want to perform a basic operation, and to access functions in the derived class to perform the more specialized operations that the derived class provides.
- Private inheritance:
  - Cases when a derived class is created as a way of completely modifying the operation of the base class, hiding or disguising its original interface.

Levels of Inheritance

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




```
// employ2.cpp
// multiple levels of inheritance
#include <iostream>
using namespace std;
const int LEN = 80;                 //maximum length of names
////////////////////////////////////////////////////////////
class employee
 { 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;
         }
   };
////////////////////////////////////////////////////////////
class manager : public employee     //manager 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;
         }
   };
```

Levels of Inheritance (2)

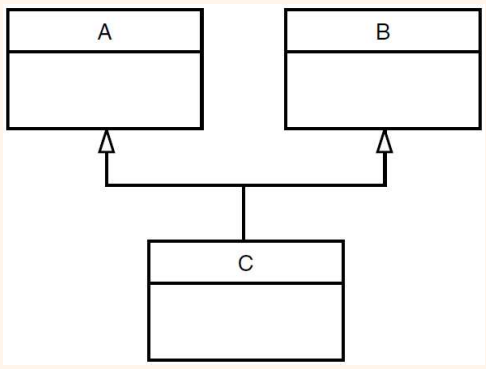
```
//////////////////////////////////////////////////////////////
class scientist : public employee  //scientist class
   {
   private:
      int pubs;                    //number of publications
   public:
      void getdata()
         {
         employee::getdata();
         cout << "   Enter number of pubs: "; cin >> pubs;
         }
      void putdata() const
         {
         employee::putdata();
         cout << "\n   Number of publications: " << pubs;
         }
   };
//////////////////////////////////////////////////////////////
class laborer : public employee     //laborer class
   {
   };
//////////////////////////////////////////////////////////////
class foreman : public laborer      //foreman class
   {
   private:
      float quotas;  //percent of quotas met successfully
   public:
      void getdata()
         {
         laborer::getdata();
         cout << "   Enter quotas: "; cin >> quotas;
         }
      void putdata() const
         {
         laborer::putdata();
         cout << "\n   Quotas: " << quotas;
         }
   };
```

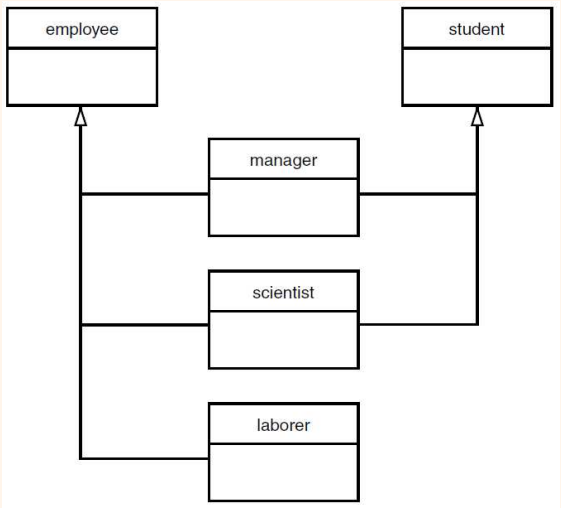
```
//////////////////////////////////////////////////////////////
int main()
   {
   laborer l1;
   foreman f1;

   cout << endl;
   cout << "\nEnter data for laborer 1";
   l1.getdata();
   cout << "\nEnter data for foreman 1";
   f1.getdata();

   cout << endl;
   cout << "\nData on laborer 1";
   l1.putdata();
   cout << "\nData on foreman 1";
   f1.putdata();
   cout << endl;
   return 0;
   }
```


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
   {
   };
```

```
class student
   { };
class employee
   { };
class manager : private employee, private student
   { };
class scientist : private employee, private student
   { };
class laborer : public employee
   { };
```





Multiple Inheritance (2)

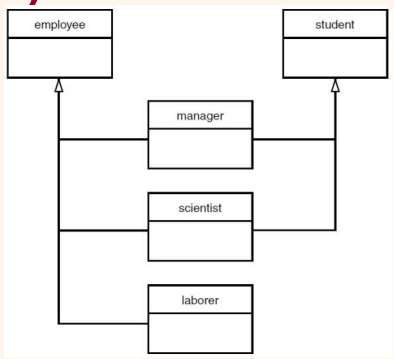
```
//empmult.cpp
//multiple inheritance with employees and degrees
#include <iostream>
using namespace std;
const int LEN = 80;           //maximum length of names
////////////////////////////////////////////////////////////
class student                 //educational background
 { private:
      char school[LEN];       //name of school or university
      char degree[LEN];       //highest degree earned
   public:
      void getedu()
         {
         cout << "   Enter name of school or university: ";
         cin >> school;
         cout << "   Enter highest degree earned \n";
        cout << "   (Highschool, Bachelor's, Master's, PhD): ";
         cin >> degree;
         }
      void putedu() const
         {
         cout << "\n   School or university: " << school;
         cout << "\n   Highest degree earned: " << degree;
         }
   };
////////////////////////////////////////////////////////////
class employee
 { 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;
         }
   };
```

```
////////////////////////////////////////////////////////////
class manager : private employee, private student  //management
   {
   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;
         student::getedu();
         }
      void putdata() const
         {
         employee::putdata();
         cout << "\n   Title: " << title;
         cout << "\n   Golf club dues: " << dues;
         student::putedu();
         }
   };
////////////////////////////////////////////////////////////
class scientist : private employee, private student
      //scientist
   {
   private:
      int pubs;     //number of publications
   public:
      void getdata()
         {
         employee::getdata();
         cout << "   Enter number of pubs: "; cin >> pubs;
         student::getedu();
         }
      void putdata() const
         {
         employee::putdata();
         cout << "\n   Number of publications: " << pubs;
         student::putedu();
         }
   };
```

Multiple Inheritance (2)

```
////////////////////////////////////////////////
class laborer : public employee              //la
   {
   };
////////////////////////////////////////////////
int main()
   {
   manager m1;
   scientist s1, s2;
   laborer l1;

   cout << endl;
   cout << "\nEnter data for manager 1";     //ge
   m1.getdata();                              //se

   cout << "\nEnter data for scientist 1";
   s1.getdata();

   cout << "\nEnter data for scientist 2";
   s2.getdata();

   cout << "\nEnter data for laborer 1";
   l1.getdata();

   cout << "\nData on manager 1";            //display data for
   m1.putdata();                              //several employees

   cout << "\nData on scientist 1";
   s1.putdata();

   cout << "\nData on scientist 2";
   s2.putdata();

   cout << "\nData on laborer 1";
   l1.putdata();
   cout << endl;
   return 0;
   }
```

```
Enter data for manager 1
   Enter last name: Bradley
   Enter number: 12
   Enter title: Vice-President
   Enter golf club dues: 100000
   Enter name of school or university: Yale
   Enter highest degree earned
   (Highschool, Bachelor's, Master's, PhD): Bachelor's

Enter data for scientist 1
   Enter last name: Twilling
   Enter number: 764
   Enter number of pubs: 99
   Enter name of school or university: MIT
   Enter highest degree earned
   (Highschool, Bachelor's, Master's, PhD): PhD

Enter data for scientist 2
   Enter last name: Yang
   Enter number: 845
   Enter number of pubs: 101
   Enter name of school or university: Stanford
   Enter highest degree earned
   (Highschool, Bachelor's, Master's, PhD): Master's

Enter data for laborer 1
   Enter last name: Jones
   Enter number: 48323
```



Constructors in Multiple Inheritance

```
// englmult.cpp
// multiple inheritance with English Distances
#include <iostream>
#include <string>
using namespace std;
////////////////////////////////////////////////////////////
class Type                          //type of lumber
 { private:
      string dimensions;
      string grade;
   public:                          //no-arg constructor
      Type() : dimensions("N/A"), grade("N/A")
         {  }                       //2-arg constructor
      Type(string di, string gr) : dimensions(di), grade(gr)
         {  }
      void gettype()                //get type from user
       { cout << "   Enter nominal dimensions (2x4 etc.): ";
         cin >> dimensions;
         cout << "   Enter grade (rough, const, etc.): ";
         cin >> grade;          }
      void showtype() const         //display type
       { cout << "\n   Dimensions: " << dimensions;
         cout << "\n   Grade: " << grade;     }
   };
////////////////////////////////////////////////////////////
class Distance                      //English Distance class
 { private:
      int feet;
      float inches;
   public:                          //no-arg constructor
      Distance() : feet(0), inches(0.0)
         {  }                       //constructor (two args)
      Distance(int ft, float in) : feet(ft), inches(in)
         {  }
      void getdist()                //get length from user
       { cout << "   Enter feet: ";  cin >> feet;
         cout << "   Enter inches: ";  cin >> inches; }
      void showdist() const         //display distance
         { cout  << feet << "\'-" << inches << '\"'; }
   };
```

```
////////////////////////////////////////////////////////////
class Lumber : public Type, public Distance
   {
   private:
      int quantity;                 //number of pieces
      double price;                 //price of each piece
   public:                          //constructor (no
     args)
      Lumber() : Type(), Distance(), quantity(0), price(0.0)
         {  }                       //constructor (6 args)
      Lumber( string di, string gr,       //args for Type
              int ft, float in,           //args for Distance
              int qu, float prc ) :       //args for our data
              Type(di, gr),               //call Type ctor
              Distance(ft, in),           //call Distance ctor
              quantity(qu), price(prc)    //initialize our data
           {  }
      void getlumber()
      {  Type::gettype();
         Distance::getdist();
         cout << "   Enter quantity: "; cin >> quantity;
         cout << "   Enter price per piece: "; cin >> price; }
      void showlumber() const
      {  Type::showtype();
         cout << "\n   Length: ";
         Distance::showdist();
         cout << "\n   Price for " << quantity
              << " pieces: $" << price * quantity;    }
   };
////////////////////////////////////////////////////////////
int main()
 { Lumber siding;                   //constructor (no args)
   cout << "\nSiding data:\n";
   siding.getlumber();              //get siding from user
                                    //constructor (6 args)
   Lumber studs( "2x4", "const", 8, 0.0, 200, 4.45F );
                                    //display lumber data
   cout << "\nSiding";  siding.showlumber();
   cout << "\nStuds";     studs.showlumber();
   cout << endl;
   return 0;
   }
```

Ambiguity in Multiple Inheritance (1)

- 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?
- problem is resolved using the scope-resolution operator.

```
// ambigu.cpp
// demonstrates 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 (2)

- if you derive a class from two classes that are each derived from the same class. This creates a diamond-shaped inheritance tree.
  - How do objects of the derived class access the correct base class function?
- experts recommend avoiding multiple inheritance unless necessary.

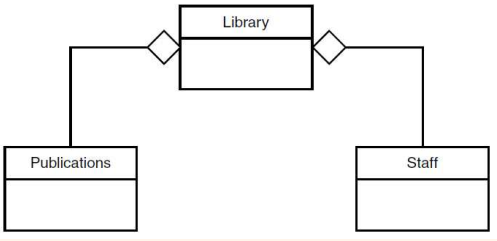
```
//diamond.cpp
//investigates diamond-shaped multiple inheritance
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////////
class A
   {
   public:
      virtual void func();
   };
class B : public A
   {  };
class C : public A
   {  };
class D : public B, public C
   {  };
////////////////////////////////////////////////////////////
int main()
   {
   D objD;
   objD.func();  //ambiguous: won't compile
   return 0;
   }
```


Aggregation: Classes Within Classes

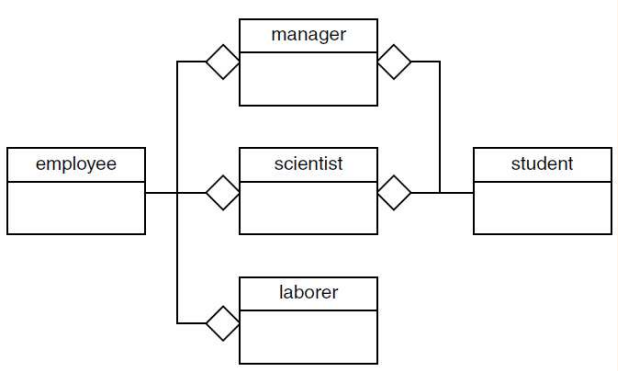
- If a class B is derived by inheritance from a class A, we can say that " B is a kind of A".
  - This is because B has all the characteristics of A , and in addition some of its own.
  - often called a "kind of" relationship.

```
class A
   {
   };
class B
   {
   A objA;  // define objA as an object of class A
   };
```

- Aggregation is called a "has a" relationship.
  - may occur when one object is an attribute of another.
  - In the UML, a special kind of association.
  - if class A contains objects of class B , and is organizationally superior to class B , it's a good candidate for aggregation.



Aggregation in the EMPCONT Program




```
class student
   {};
class employee
   {};
class manager
   {
   student stu;    // stu is an object of class student
   employee emp;   // emp is an object of class employee
   };
class scientist
   {
   student stu;    // stu is an object of class student
   employee emp;   // emp is an object of class employee
   };
class laborer
   {
   employee emp;   // emp is an object of class employee
   };
```

```
// empcont.cpp
// containership with employees and degrees
#include <iostream>
#include <string>
using namespace std;
////////////////////////////////////////////////////////////
class student                   //educational background
 { private:
      string school;            //name of school or university
      string degree;            //highest degree earned
   public:
      void getedu()
      { cout << "   Enter name of school or university: ";
        cin >> school;
        cout << "   Enter highest degree earned \n";
        cout << "   (Highschool, Bachelor's, Master's, PhD): ";
        cin >> degree;    }
      void putedu() const
      {  cout << "\n   School or university: " << school;
         cout << "\n   Highest degree earned: " << degree;
         }
   };
////////////////////////////////////////////////////////////
class employee
   {
   private:
      string name;              //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;
         }
   };
```

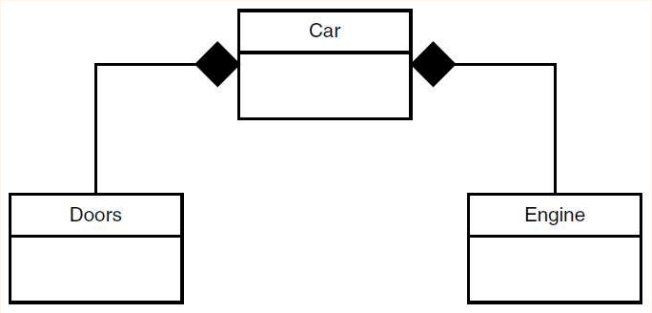
Aggregation in the EMPCONT Program (2)

```
////////////////////////////////////////////////////////////
class manager                   //management
 { private:
      string title;             //"vice-president" etc.
      double dues;              //golf club dues
      employee emp;             //object of class employee
      student stu;              //object of class student
   public:
      void getdata()
       { emp.getdata();
         cout << "   Enter title: ";          cin >> title;
         cout << "   Enter golf club dues: "; cin >> dues;
         stu.getedu();       }
      void putdata() const
       { emp.putdata();
         cout << "\n   Title: " << title;
         cout << "\n   Golf club dues: " << dues;
         stu.putedu();       }
   };
////////////////////////////////////////////////////////////
class scientist                 //scientist
   {
   private:
      int pubs;                 //number of publications
      employee emp;             //object of class employee
      student stu;              //object of class student
   public:
      void getdata()
         {
         emp.getdata();
         cout << "   Enter number of pubs: "; cin >> pubs;
         stu.getedu();
         }
      void putdata() const
         {
         emp.putdata();
         cout << "\n   Number of publications: " << pubs;
         stu.putedu();
         }
   };
```

```
////////////////////////////////////////////////////////////
class laborer                   //laborer
 { private:
      employee emp;             //object of class employee
   public:
      void getdata()
         { emp.getdata(); }
      void putdata() const
         { emp.putdata(); }
   };
////////////////////////////////////////////////////////////
int main()
   {
   manager m1;
   scientist s1, s2;
   laborer l1;

   cout << endl;
   cout << "\nEnter data for manager 1";    //get data for
   m1.getdata();                            //several employees
   cout << "\nEnter data for scientist 1";
   s1.getdata();
   cout << "\nEnter data for scientist 2";
   s2.getdata();
   cout << "\nEnter data for laborer 1";
   l1.getdata();

   cout << "\nData on manager 1";           //display data for
   m1.putdata();                            //several employees
   cout << "\nData on scientist 1";
   s1.putdata();
   cout << "\nData on scientist 2";
   s2.putdata();
   cout << "\nData on laborer 1";
   l1.putdata();
   cout << endl;
   return 0;
   }
```

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.
- A car is composed of doors (among other things). The doors can't belong to some other car, and they are born and die along with the car. A room is composed of a floor, ceiling, and walls.
- While aggregation is a "has a" relationship, composition is a "consists of" relationship.



Summary (1)

- A class, called the derived class, can inherit the features of another class, called the base class.
  - The derived class can add other features of its own, so it becomes a specialized version of the base class.
  - Inheritance provides a powerful way to extend the capabilities of existing classes, and to design programs using hierarchical relationships.
- Accessibility of base class members from derived classes and from objects of derived classes is an important issue.
  - Data or functions in the base class that are prefaced by the keyword protected can be accessed from derived classes but not by any other objects, including objects of derived classes.
  - Classes may be publicly or privately derived from base classes. Objects of a publicly derived class can access public members of the base class, while objects of a privately derived class cannot.
- A class can be derived from more than one base class. This is called multiple inheritance. A class can also be contained within another class.

Summary (2)

- In the UML, inheritance is called generalization. This relationship is represented in class diagrams by an open triangle pointing to the base (parent) class.
- Aggregation is a “has a” or “part-whole” relationship:
  - one class contains objects of another class. Aggregation is represented in UML class diagrams by an open diamond pointing to the “whole” part of the part-whole pair.
- Composition is a strong form of aggregation. Its arrowhead is solid rather than open.
- Inheritance permits the reusability of software: Derived classes can extend the capabilities of base classes with no need to modify—or even access the source code of—the base class. This leads to new flexibility in the software development process, and to a wider range of roles for software developers.