Block 09

Book II Chapter 1 Inheritance.

INHERITANCE IS THE MAIN THING I'M THINKING OF WHEN I SAY

THAT MOST OOP TECHNIQUES ARE NOT APPROPRIATE TO FIRST-YEAR

PROJECTS, OR EVEN TO PROJECTS WITH LESS THAN A HANDFUL OF

PROGRAMMERS WORKING ON THEM. INHERITANCE IS DATA-INTENSIVE,

AND TRIES TO WRITE THE MOST GENERAL AND RE-USABLE DATA STRUCTURES

AND CODE, THEN TO "INHERIT" THAT CODE AND OVERRIDE PARTS OF IT

TO DO SPECIFIC JOBS.

TYPES OF INHERITANCE ARE PUBLIC, PROTECTED, AND PRIVATE. WE

GENERALLY USE PUBLIC.

PLEASE DON'T LET THIS MATERIAL GET TO BE A HEADACHE. IT'S

COMPLEX, BUT YOU HAVE YEARS TO PRACTICE WITH IT.

<A HREF="http://209.129.16.61/~hhaller/data/cisc192/modules/inherit2.cpp.txt">inherit2.cpp.txt</A>

Vocabulary.

Base Class: what you're inheriting from.

Derived Class: what is doing the inheriting.

From the point of view of the Derived Class:

Indirect Base Class

|

Direct base Class

|

Derived Class

Java only allows direct inheritance from a single

Base class, but C++ permits Multiple Inheritance.

For now, let's just do Public Inheritance:

Public Inheritance: Every object in a derived class is also an

object of that derived class's base class.

(That doesn't mean that every element of the derived class,

like int count, or char key, exists in the base class, but

rather that the derived object IS A object of the base class,

so class Atv IS A class Vehicle.

Vehicle

|

ATV

Derived classes can access non-private members of the base class. If you

only want members to have access within a class, or from derived classes,

make something "protected."

/\*-----------------------------------------------------------------

\* inheritance03.cpp - demonstrate protected access

Sun Jul 18 12:53:22 PDT 2010

-----------------------------------------------------------------\*/

#include <iostream>

using namespace std ;

class Imdabase

{

protected: int count ;

public: void setCount(int n) { count = n ; }

} ;

class Sonofbase : public Imdabase

{

public: int getCount(void) { return count ; }

} ;

int main (int argc, char \*argv[], char \*\*env)

{

Sonofbase s ;

s.setCount(999) ;

cout << s.getCount ()<< endl ;

//uncomment the following line for "protected: error" compiler error:

//cout << "Trying to access count externally: " << s.count << endl;

} // main() ends

===========================================

Here, we call a method from the base class from the

derived class.

/\*-----------------------------------------------------------------

\* inheritance03.cpp - demonstrate protected access

Sun Jul 18 12:53:22 PDT 2010

-----------------------------------------------------------------\*/

#include <iostream>

using namespace std ;

class Imdabase

{

protected: int count ;

public: void setCount(int n) { count = n ; }

} ;

class Sonofbase : protected Imdabase

{

public: int getCount(void) { return count ; }

// if I comment out the next line, then it won't compile,

// since this is "protected" inheritance, and thus the

// public method in the base isn't accessible: that means

// that the whole base is protected when you inherit from

// it, and you don't have to go back and re-think security

void setCount(int n) { Imdabase::setCount(n) ; } // call base class

} ;

int main (int argc, char \*argv[], char \*\*env)

{

Sonofbase s ;

s.setCount(999) ; // call base class method from derived class method

cout << s.getCount ()<< endl ;

//uncomment the following line for "protected: error" compiler error:

//cout << "Trying to access count externally: " << s.count << endl;

} // main() ends

With all forms of inheritance, private base class members are not

accessable directly from that class's derived classes, but they're

still inherited.

With "public" inheritance, base class members retain their original

security in derived classes.

If the derived class re-defines a base class member, the derived

class can refer to the original member in the base class with

notation like:

base\_class::member (rest of expression) thus:

/\*-----------------------------------------------------------------

\* inheritance04.cpp - demonstrate protected access

Sun Jul 18 12:53:22 PDT 2010

-----------------------------------------------------------------\*/

#include <iostream>

using namespace std ;

class Imdabase

{

protected: int count ;

public: void setCount(int n)

{

cout << "I'm the base class SetCount()\n" ;

count = n ;

}

} ;

class Sonofbase : public Imdabase

{

public:

void setCount( int n )

{

cout << "I'm the derived setCount()\n" ;

count = n ;

}

int getCount(void)

{

cout << "I'm the derived getCount returning: " << count << endl ;

return count ;

}

// here I'm using an extra parm so that NameMangling calls the

// base::method I want, ("max" is only used to differentiate)

void setCount(int n, int max) { Imdabase::setCount(n) ; } // call base class

} ;

int main (int argc, char \*argv[], char \*\*env)

{

Sonofbase s ;

s.setCount(999) ; // call derived class method

cout << s.getCount ()<< endl ;

s.setCount(4,100) ; // call base class method from derived class

cout << s.getCount ()<< endl ;

} // main() ends

==============================================

If you inherit from a class, that class's constructor

will run as its derived class is instantiated:

/\*-----------------------------------------------------------------

\* inherconstructors001.cpp - constructor actions in inheritance

\*

-----------------------------------------------------------------\*/

#include <iostream>

using namespace std ;

class one

{

private:

int count ;

public:

one() { cout << "class one constructor " << endl ; }

one(int n) : count( n )

{

cout << "Parameterized class one constructor: " << n << endl ;

}

} ;

class two : public one

{

public:

two() // this calls the default constructor automatically

// then the parameterized one

{

one::one(13) ;

cout << "class two constructor " << endl ;

}

} ;

class alpha

{

public:

alpha() { cout << "class alpha constructor\n" ; }

} ;

class baker : public alpha

{

public:

baker() { cout << "class baker constructor\n" ; }

} ;

// "diamond inheritance" from two classes:

class alphanumeric : public two, baker

{

public:

alphanumeric() { cout << "class alphanumeric constructor\n" ; }

} ;

int main (int argc, char \*argv[], char \*\*env)

{

}

/\*

OUTPUT:

class one constructor

Parameterized class one constructor: 13

class two constructor

class alpha constructor

class baker constructor

class alphanumeric constructor

\*/

=======================================

Again, this won't matter to you until you're a working programmer in a

corporate environment assigned to a big project.