Inheritance & Polymorphism

Week 7

Gaddis: Chapter 15

CS 5301 Spring 2015

Jill Seaman

Inheritance

- A way to create a new class from an existing class
- The new class is a specialized version of the existing class
- Base class (or parent) the existing class
- Derived class (or child) inherits from the base class
- The derived class contains all the members from the base class (in addition to the ones in the derived class).

class Student {	<pre>class UnderGrad : public student {</pre>
} Base class	} Derived class 2

Access to private members

class Grade	class Test : p
<pre>private members: char letter; float score; void calcGrade(); public members: void setScore(float); float getScore(); char getLetter();</pre>	private memb int numQ float po int numM public membo Test (int
When Test class inherits from Grade class using public class access, it looks like this:	private memb int numQ float po int numM public membo Test (int void set float ge float ge

An instance of Test contains letter and score, but they are **not** directly accessible from inside (or outside) the Test member functions. class Test: public Grade rivate members: int numQuestions; float pointsEach; int numMissed; ublic members: Test(int, int);

orivate members: int numQuestions: float pointsEach; int numMissed; oublic members: Test(int, int); void setScore(float); float getScore(); float getLetter();

3

1

Constructors and Destructors in Base and Derived Classes

- Derived classes can have their own constructors and destructors
- When an object of a derived class is created,
 - 1. the base class's (default) constructor is executed first,
 - 2. followed by the derived class's constructor
- When an object of a derived class is destroyed,
 - 1. the derived class destructor is called first,
 - 2. then the base class destructor

Constructors and Destructors:

example

class BaseClass { public:	•
BaseClass()	
.,	is the Decollege constructor \s".
	is the BaseClass constructor.\n"; }
~BaseClass()	
{ cout << "This	is the BaseClass destructor.\n"; }
};	
class DerivedClass :	<pre>public BaseClass {</pre>
public:	
<pre>- DerivedClass()</pre>	
	is the DerivedClass constructor.\n"; }
~DerivedClass()	
()	is the DerivedClass destructor.\n"; }
•	is the belived class descractor. (h ,)
};	
int main() {	
	ow define a DerivedClass object.\n";
DerivedClass objec	t;
cout << "The progr	am is now going to end.\n";
}	
Output:	We will now define a DerivedClass object.
Oulpul.	This is the BaseClass constructor.
	This is the DerivedClass constructor.
	The program is now going to end.
	This is the DerivedClass destructor.

This is the DerivedClass destructor This is the BaseClass destructor.

Passing Arguments to a non-default Base Class Constructor

- Allows programmer to choose which base class constructor is called from the derived class constructor
- Specify arguments to base constructor in the derived constructor function header:

```
//assuming Square is derived from Rectangle:
Rectangle::Rectangle() { width = 0.0; length = 0.0; }
Square::Square(int side) : Rectangle(side, side) {
    // code for Square constr goes here, if any
}
```

 You must specify a call to a base class constructor if base class has no default constructor

Redefining Base Class Functions

- <u>Redefining function</u>: a function in a derived class that has the same name and parameter list as a function in the base class
- Not the same as overloading with overloading, parameter lists must be different
- Objects of base class use base class version of function; objects of derived class use derived class version of function.
- To call the base class version from the derived class version, you must prefix the name of the function with the base class name and the scope resolution operator:

Redefining Base Class Functions:

	1	
class Animal {		
private:		
string species;		
public:		
Animal() { species = "Animal";}	<pre>int main() {</pre>	
Animal(string spe)	Animal jaspe	er; // Animal()
{ species = spe ;}	Primate fred(4); // Primate(int)
<pre>void display()</pre>	jasper.displa	ay(); cout << endl;
<pre>{cout << "species " << species; }</pre>	fred.display(); cout << endl;
};	}	
class Primate: public Animal {		
private:		
int heartCham;		Output:
public:		species Animal
Primate() : Animal("Primate") { }		species Primate,
Primate(int in) : Animal ("Primate")		# of heart chambers 4
<pre>{ heartCham = in; }</pre>		
<pre>void display()</pre>		
{ Animal::display(); //calls base class display()		
<pre>cout << ", \n# of heart chambers " << heartCham;</pre>		
}		8
};		0

Include Guar	ĊĊŚ
#ifndef RECTANGLE_H	Rectangle.
#define RECTANGLE_H	
class Rectangle	
{	
private:	
double width;	
double length;	
public:	
<pre>void setWidth(double);</pre>	
<pre>void setLength(double);</pre>	
<pre>double getWidth() const;</pre>	
double getLength() const;	
double getArea() const;	
}; #endif	

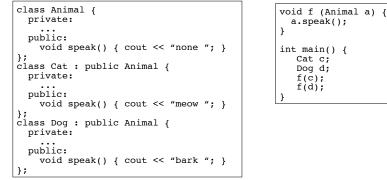
- These preprocessor directives prevent the header file from accidentally being included more than once.
- If you have a base class with 2 derived classes, and the derived classes are both included in a driver₉...

Polymorphism

- The Greek word poly means many, and the Greek word morphism means form.
- So, polymorphism means 'many forms'.
- In object-oriented programming (OOP), polymorphism refers to
 - identically named (and redefined) methods
 - that have different behavior depending on the (specific derived) type of object that they are called on.

10

Example of polymorphism?



- IF the output is "meow bark", yes, polymorphism.
 - The behavior of a in f would depend on its specific (derived) type.
- IF the output is "none none", no it's not.

Polymorphism in C++

- Polymorphism in C++ is supported through:
 - virtual methods AND
 - pointers to objects OR reference variables/ parameters.
- without these, C++ determines which method to invoke at <u>compile time</u> (using the variable type).
- when virtual methods and pointer/references are used together, C++ determines which method to invoke at <u>run time</u> (using the specific type of the instance currently referenced by the variable).

12

Virtual methods

- <u>Virtual member function</u>: function in a base class that expects to be redefined in derived class
- Function defined with key word virtual:

virtual void Y() $\{\dots\}$

- Supports <u>dynamic binding</u>: functions bound at run time to function that they call
- Without virtual member functions, C++ uses <u>static</u> (compile time) <u>binding</u>

Example virtual methods

```
class Animal {
  public:
  virtual void speak();
 int age();
class Cat : public Animal
  public:
 virtual void speak(); //redefining a virtual
 int age();
                       //redefining a normal function
1:
int main()
  Cat morris;
  Animal *pA = &morris; //using a pointer to get dynamic binding
  pA -> age(); // Animal::age() is invoked (base) (not virtual)
 pA -> speak(); // Cat::speak() is invoked (derived)
                                                              14
```

Virtual methods

- In compile-time binding, <u>the data type of the</u> <u>pointer</u> resolves which method is invoked.
- In run-time binding, <u>the type of the object</u> <u>pointed to</u> resolves which method is invoked.

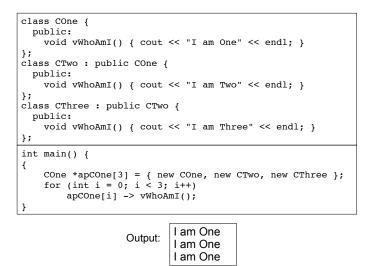
<pre>void f (Animal a.speak(); }</pre>	<mark>&</mark> a)	{
<pre>int main() { Cat c; Dog d; f(c); f(d); }</pre>		

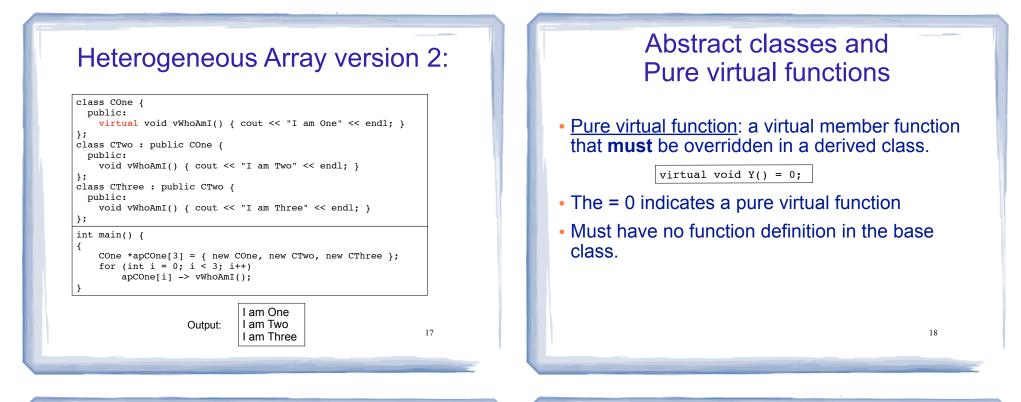
 Assuming speak is virtual, since a is passed by reference, the output is:
 Imeow bark

15

13

Heterogeneous Array version 1:





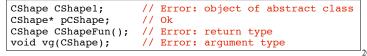
Abstract classes and Pure virtual functions

- <u>Abstract base class</u>: a class that can have no objects (instances).
- Serves as a basis for derived classes that will have objects
- A class becomes an abstract base class when one or more of its member functions is a pure virtual function.

Example: Abstract Class

```
class CShape {
  public:
    CShape ( ) { }
    virtual void vDraw ( ) const = 0; // pure virtual method
};
```

- An abstract class may **not** be used as an argument type, as a function return type, or as the type of an explicit conversion.
- Pointers and references to an abstract class may be declared.



Example: Abstract Class

 Pure virtual functions are inherited as pure virtual functions.

class CAbstractCircle : public CShape { private: int m iRadius; public: void vRotate (int) {} // CAbstractCircle ::vDraw() is a pure virtual function };

• Or else:

```
class CCircle : public CShape {
 private:
    int m iRadius;
 public:
    void vRotate (int) {}
    void vDraw(); //define here or in impl file
};
```

21

Heterogeneous collection: abstract base class

class Animal {		int	t main()	
private:		{		
string name;			Animal* animals[] = {	
public:			new Cat("Charlie"),	
Animal(string n) {name = n;}			new Cat("Scamp"),	
<pre>virtual void speak() = 0;</pre>			new Dog("Penny"),	
};			new Cat("Libby"),	
class Cat : public Animal {			new Cat("Patches"),	
public:			new Dog("Milo"),	
Cat(string n) : Animal(n) { };			<pre>new Pig("Wilbur") };</pre>	
void speak() {cout << "meow "; }				
};			for (int i=0; i< 7; i++) {	
class Dog : public Animal {			animals[i]->speak();	
public:			}	
Dog(string n) : Animal(n) { };		}		
void speak() {cout << "bark "; }				`
};	(Outpu	out:	
<pre>class Pig : public Animal { public:</pre>		meov	ow meow bark meow meow bark oink	i
<pre>Pig(string n) : Animal(n) { };</pre>				
void speak() {cout << "oink "; }				
};			22	
			22	

Sample Problem

Ship, CruiseShip, and CargoShip Classes : Design a Ship class that has the following members:

- A member variable for the <u>name</u> of the ship (a string)
- A member variable for the year the ship was built (a string)
- A constructor and appropriate accessors and mutators
- A virtual print function that displays the ship's name and the year it was built.

Design a CruiseShip class that is derived from the Ship class. The CruiseShip class has the following members:

- A member variable for the maximum number of passengers (an int)
- A constructor and appropriate accessors and mutators

 A print function that overrides the print function in the base class. The CruiseShip class's print function should display only the ship's name and the maximum number of passengers. 23

Sample Problem, cont.

Design a CargoShip class that is derived from the Ship class. The CargoShip class should have the following members:

- A member variable for the cargo capacity in tonnage (an int).
- A constructor and appropriate accessors and mutators.

 A print function that overrides the print function in the base class. The CargoShip class's print function should display only the ship's name and the ship's cargo capacity.

Demonstrate the classes in a program that has an array of Ship pointers. The array elements should be initialized with the addresses of dynamically allocated Ship, CruiseShip, and CargoShip objects. The program should then step through the array, calling each object's print function.