# Derived Classes and Inheritance

Chapter 9 D&D

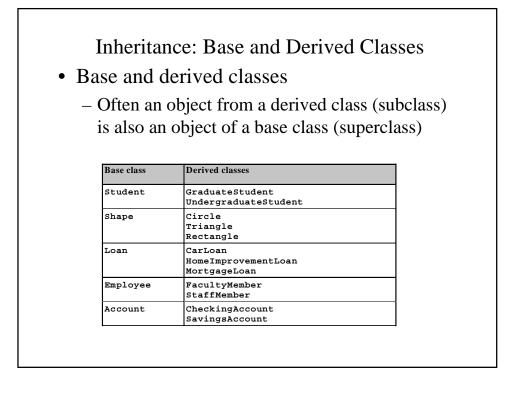
## **Derived Classes**

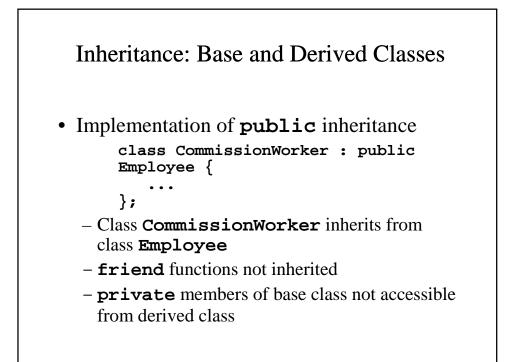
- It is sometimes the case that we have a class is *nearly* what we need.
- *Derived classes* acquire the properties of an existing class.
- The original class is called the *base class*.

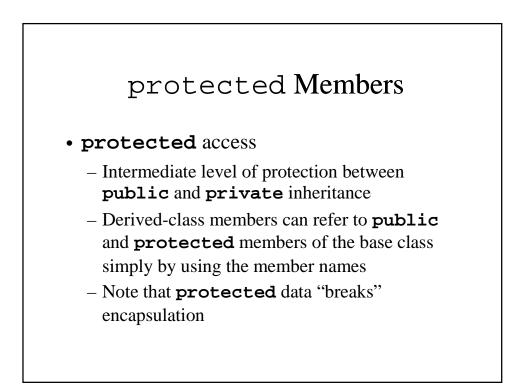
## Inheritance

#### • Inheritance

- New classes created from existing classes
- Derived class
  - Class that inherits data members and member functions from a previously defined base class
- Single inheritance
  - Class inherits from one base class
- Multiple inheritance
  - Class inherits from multiple base classes
- Types of inheritance
  - public: Derived objects are accessible by the base class objects
  - private: Derived objects are inaccessible by the base class
  - protected: Derived classes and friends can access protected members of the base class







# **Derived Classes**

- A derived class inherits member functions of base class.
- A derived class can be used anywhere the base class is expected.

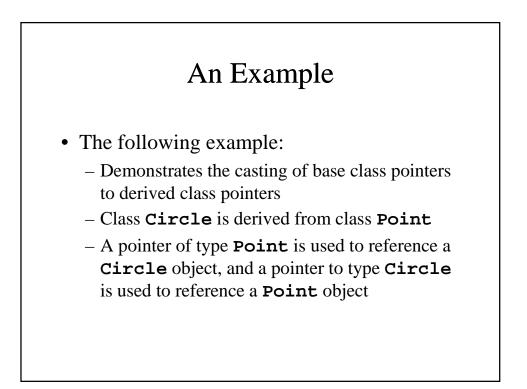
## **Derived Classes**

- A derived class inherits member functions of base class.
- A derived class can be used anywhere the base class is expected.
- However, a base class CANNOT be used anywhere the derived class is expected.

Casting Base-Class Pointers to Derived Class Pointers

- Downcasting a pointer
  - Use an explicit cast to convert a base-class pointer to a derived-class pointer
  - If pointer is going to be dereferenced, the type of the pointer must match the type of object to which the pointer points
  - Format:

# derivedPtr = static\_cast< DerivedClass \* > basePtr;



```
1 // Fig. 9.4: point.h
 2 // Definition of class Point
 3 #ifndef POINT H
 4 #define POINT_H
 5
 6 #include <iostream>
 7
 8 using std::ostream;
 9
10 class Point {
11 friend ostream &operator<<( ostream &, const Point & );</pre>
12 public:
      Point( int = 0, int = 0 );
13
                                            // default constructor
     void setPoint( int, int ); // set coordinates
int getX() const { return x; } // get x coordinate
int getY() const { return y; } // get y coordinate
14
15
16
                      // accessible by derived classes
// x and y coordinates of the Point
17 protected:
18
      int x, y;
19 };
20
21 #endif
22 // Fig. 9.4: point.cpp
23 // Member functions for class Point
24 #include <iostream>
25 #include "point.h"
26
27 // Constructor for class Point
28 Point::Point( int a, int b ) { setPoint( a, b ); }
29
30 // Set x and y coordinates of Point
31 void Point::setPoint( int a, int b )
32 {
33
        \mathbf{x} = \mathbf{a}:
```

```
34
      y = b;
35 }
36
37 // Output Point (with overloaded stream insertion operator)
38 ostream &operator<<( ostream &output, const Point &p )
39 {
40 output << '[' << p.x << ", " << p.y << ']';
41
     return output; // enables cascaded calls
42
43 }
44 // Fig. 9.4: circle.h
45 // Definition of class Circle
46 #ifndef CIRCLE_H
47 #define CIRCLE H
48
49 #include <iostream>
50
51 using std::ostream;
52
53 #include <iomanip>
54
55 using std::ios;
 56 using std::setiosflags;
57 using std::setprecision;
58
59 #include "point.h"
60
61 class Circle : public Point { // Circle inherits from Point
 62 friend ostream &operator<<( ostream &, const Circle & );</pre>
63 public:
64 // default constructor
```

```
65
    Circle( double r = 0.0, int x = 0, int y = 0 );
66
    void setRadius( double ); // set radius
67
    double getRadius() const; // return radius
68
69 double area() const;
                                // calculate area
70 protected:
71 double radius;
72 };
73
74 #endif
75 // Fig. 9.4: circle.cpp
76 // Member function definitions for class Circle
77 #include "circle.h"
78
79 // Constructor for Circle calls constructor for Point
80 // with a member initializer then initializes radius.
81 Circle::Circle( double r, int a, int b )
82 : Point( a, b )
                         // call base-class constructor
83 { setRadius( r ); }
84
85 // Set radius of Circle
86 void Circle::setRadius( double r )
87
    { radius = ( r >= 0 ? r : 0 ); }
88
```

```
89 // Get radius of Circle
90 double Circle::getRadius() const { return radius; }
91
92 // Calculate area of Circle
93 double Circle::area() const
    { return 3.14159 * radius * radius; }
94
95
96 // Output a Circle in the form:
97 // Center = [x, y]; Radius = #.##
98 ostream &operator<<( ostream &output, const Circle &c )
99 {
100 output << "Center = " << static cast< Point >( c )
101
            << "; Radius = "
102
             << setiosflags( ios::fixed | ios::showpoint )
103
             << setprecision( 2 ) << c.radius;
104
105 return output; // enables cascaded calls
106}
107// Fig. 9.4: fig09_04.cpp
108// Casting base-class pointers to derived-class pointers
109#include <iostream>
110
111using std::cout;
112using std::endl;
113
114#include <iomanip>
115
116#include "point.h"
117#include "circle.h"
118
119 int main()
120{
121
      Point *pointPtr = 0, p( 30, 50 );
```

```
122 Circle *circlePtr = 0, c( 2.7, 120, 89 );
123
124 cout << "Point p: " << p << "\nCircle c: " << c << '\n';
125
126 // Treat a Circle as a Point (see only the base class part)
127 pointPtr = &c; // assign address of Circle to pointPtr
128 cout << "\nCircle c (via *pointPtr): "</pre>
129
           << *pointPtr << '\n';
130
131 // Treat a Circle as a Circle (with some casting)
132 // cast base-class pointer to derived-class pointer
133 circlePtr = static_cast< Circle * >( pointPtr );
134 cout << "\nCircle c (via *circlePtr):\n" << *circlePtr
135
          << "\nArea of c (via circlePtr): "
136
          << circlePtr->area() << '\n';
137
138 // DANGEROUS: Treat a Point as a Circle
139 pointPtr = &p; // assign address of Point to pointPtr
140
141 // cast base-class pointer to derived-class pointer
142 circlePtr = static_cast< Circle * >( pointPtr );
143 cout << "\nPoint p (via *circlePtr):\n" << *circlePtr
144
           << "\nArea of object circlePtr points to: "
145
          << circlePtr->area() << endl;
146 return 0;
147}
```

Point p: [30, 50] Circle c: Center = [120, 89]; Radius = 2.70 Circle c (via \*circlePtr): Center = [120, 89]; Radius = 2.70 Area of c (via circlePtr): 22.90 Point p (via \*circlePtr): Center = [30, 50]; Radius = 0.00 Area of object circlePtr points to: 0.00

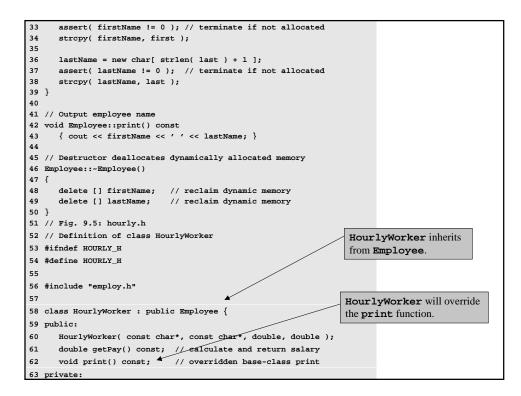
## Using Member Functions

- Derived class member functions
  - Cannot directly access private members of their base class
    - Maintains encapsulation
  - Hiding private members is a huge help in testing, debugging and correctly modifying systems

## Overriding Base-Class Members in a Derived Class

- To override a base-class member function
  - In the derived class, supply a new version of that function with the same signature
    - same function name, different definition
  - When the function is then mentioned by name in the derived class, the derived version is automatically called
  - The scope-resolution operator may be used to access the base class version from the derived class

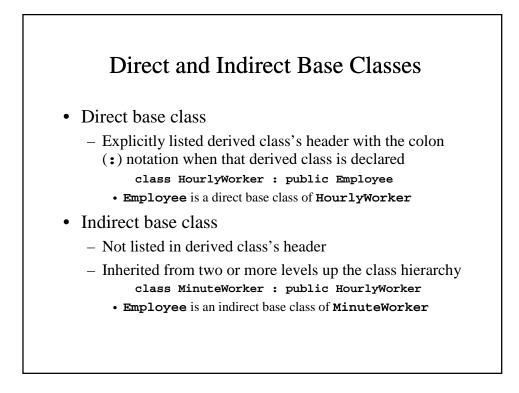
```
1 // Fig. 9.5: employ.h
 2 // Definition of class Employee
 3 #ifndef EMPLOY_H
 4 #define EMPLOY_H
 5
 6 class Employee {
 7 public:
 8
      Employee( const char *, const char * ); // constructor
      void print() const; // output first and last name
 9
 10
      ~Employee();
                          // destructor
11 private:
     char *firstName;
 12
                           // dynamically allocated string
     char *lastName;
                           // dynamically allocated string
13
14 };
15
16 #endif
17 // Fig. 9.5: employ.cpp
18 // Member function definitions for class Employee
19 #include <iostream>
20
21 using std::cout;
22
23 #include <cstring>
24 #include <cassert>
25 #include "employ.h"
26
27 // Constructor dynamically allocates space for the
28 // first and last name and uses strcpy to copy
29 // the first and last names into the object.
30 Employee::Employee( const char *first, const char *last )
31 {
32
      firstName = new char[ strlen( first ) + 1 ];
```



```
64
    double wage;
                              // wage per hour
65
     double hours;
                              // hours worked for week
66 };
67
68 #endif
69 // Fig. 9.5: hourly.cpp
70 // Member function definitions for class HourlyWorker
71 #include <iostream>
72
73 using std::cout;
74 using std::endl;
75
76 #include <iomanip>
 77
78 using std::ios;
79 using std::setiosflags;
80 using std::setprecision;
81
82 #include "hourly.h"
83
84 // Constructor for class HourlyWorker
85 HourlyWorker::HourlyWorker( const char *first,
                              const char *last,
86
87
                              double initHours, double initWage )
88
     : Employee( first, last ) // call base-class constructor
89 {
90
     hours = initHours; // should validate
      wage = initWage; // should validate
91
92 }
93
94 // Get the HourlyWorker's pay
95 double HourlyWorker::getPay() const { return wage * hours; }
```

```
96
97 // Print the HourlyWorker's name and pay
98 void HourlyWorker::print() const
99 {
100 cout << "HourlyWorker::print() is executing\n\n";</pre>
101 Employee::print(); // call base-class print function
102
103 cout << " is an hourly worker with pay of $"
104
         << setiosflags( ios::fixed | ios::showpoint )
105
          << setprecision( 2 ) << getPay() << endl;
106}
107// Fig. 9.5: fig09_05.cpp
108// Overriding a base-class member function in a
109// derived class.
110#include "hourly.h"
111
112int main()
113{
114 HourlyWorker h( "Bob", "Smith", 40.0, 10.00 );
115 h.print();
116 return 0;
117}
HourlyWorker::print() is executing
Bob Smith is an hourly worker with pay of $400.00
```

Base class member access specifier	Type of inheritance		
	<b>public</b> inheritance	protected inheritance	<b>private</b> inheritance
Public	public in derived class. Can be accessed directly in derived class by member or non- member functions	<b>protected</b> in derived class. Can be accessed directly only by all member functions	<b>private</b> in derived class. Can be accessed directly only by all member functions
Protected	<b>protected</b> in derived class. Can be accessed directly in derived class only by member functions (like private)	<b>protected</b> in derived class. Can be accessed directly only by all member functions	<b>private</b> in derived class. Can be accessed directly only by all member functions
	Can be accessed by non-static member functions and friend functions through public or protected member functions of the base class.	Can be accessed by non-static member functions and friend functions through public or protected member functions of the base class.	Can be accessed by non-static member functions and friend functions through public or protected member functions of the base class.



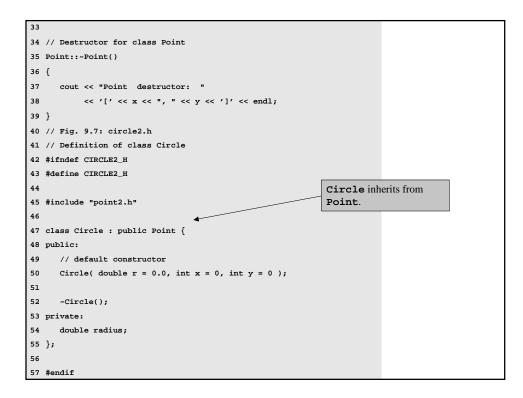
#### Using Constructors and Destructors in Derived Classes

- Base class initializer
  - Uses member-initializer syntax
  - Can be provided in the derived class constructor to call the base-class constructor explicitly
    - · Otherwise base class's default constructor called implicitly
  - Base-class constructors and base-class assignment operators are not inherited by derived classes
    - Derived-class constructors and assignment operators, however, can call base-class constructors and assignment operators

#### Using Constructors and Destructors in Derived Classes

- A derived-class constructor
  - Calls the constructor for its base class first to initialize its base-class members
  - If the derived-class constructor is omitted, its default constructor calls the base-class' default constructor
- Destructors are called in the reverse order of constructor calls
  - So a derived-class destructor is called before its baseclass destructor

```
1 // Fig. 9.7: point2.h
 2 // Definition of class Point
 3 #ifndef POINT2_H
 4 #define POINT2 H
 5
 6 class Point {
 7 public:
 8 Point( int = 0, int = 0 ); // default constructor
      ~Point(); // destructor
otected: // accessible by derived classes
 9
10 protected:
11 int x, y; // x and y coordinates of Point
12 };
13
14 #endif
15 // Fig. 9.7: point2.cpp
16 // Member function definitions for class Point
17 #include <iostream>
18
19 using std::cout;
20 using std::endl;
21
22 #include "point2.h"
23
24 // Constructor for class Point
25 Point::Point( int a, int b )
26 {
27 x = a;
28
     y = b;
29
30
     cout << "Point constructor: "
      << '[' << x << ", " << y << ']' << endl;</pre>
31
32 }
```



```
58 // Fig. 9.7: circle2.cpp
59 // Member function definitions for class Circle
60 #include <iostream>
61
62 using std::cout;
63 using std::endl;
64
65 #include "circle2.h"
66
67 // Constructor for Circle calls constructor for Point
68 Circle::Circle( double r, int a, int b )
69 : Point( a, b ) </r>
                                                                 Constructor for Circle
70 {
                                                                 calls constructor for
71 radius = r; // should validate
                                                                 Point, first. Uses
72 cout << "Circle constructor: radius is "
                                                                 member-initializer syntax.
73
         << radius << " [" << x << ", " << y << ']' << endl;
74 }
75
76 // Destructor for class Circle
                                                                     Destructor for Circle
77 Circle::~Circle()
                       ←
                                                                     calls destructor for Point,
78 {
                                                                     last.
79
    cout << "Circle destructor: radius is "</pre>
80
         << radius << " [" << x << ", " << y << ']' << endl;
81 }
```

82 // Fig. 9.7: fig09_07.cpp				
83 // Demonstrate when base-class and derived-class	<pre>// Demonstrate when base-class and derived-class</pre>			
4 // constructors and destructors are called.				
85 #include <iostream></iostream>				
86				
87 using std::cout;				
88 using std::endl;				
89				
90 #include "point2.h"				
91 #include "circle2.h"				
92				
93 int main()				
94 {				
95 // Show constructor and destructor calls for Point				
96 {				
97 Point p( 11, 22 );				
98 }				
99				
100 cout << endl;				
101 Circle circle1( 4.5, 72, 29 );				
102 cout << endl;				
103 Circle circle2( 10, 5, 5 );				
104 cout << endl;				
105 return 0;				
106}				

```
Point constructor: [11, 22]
Point destructor: [11, 22]
Point constructor: [11, 22]
Point constructor: [72, 29]
Circle constructor: radius is 4.5 [72, 29]
Point constructor: radius is 10 [5, 5]
Circle destructor: radius is 10 [5, 5]
Point destructor: radius is 4.5 [72, 29]
Point destructor: radius is 4.5 [72, 29]
```

Program Output

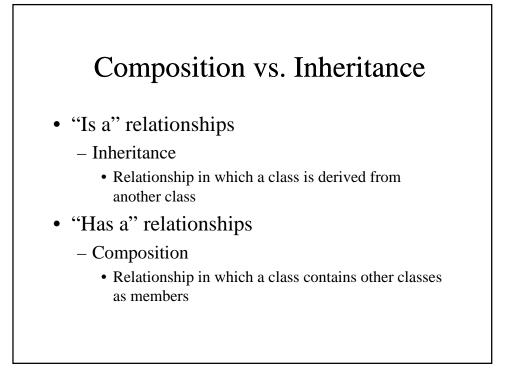
#### Implicit Derived-Class Object to Base-Class Object Conversion

- Assignment of derived and base classes
  - Derived-class type and base-class type are different
  - Derived-class object can be treated as a base-class object
    - Derived class has members corresponding to all of the base class's members
    - · Derived-class has more members than the base-class object
    - Base-class can be assigned a derived-class
  - Base-class object cannot be treated as a derived-class object
    - Would leave additional derived class members undefined
    - Derived-class cannot be assigned a base-class
    - Assignment operator can be overloaded to allow such an assignment

#### Implicit Derived-Class Object to Base-Class Object Conversion

• Mixing base and derived class pointers and objects

- Referring to a base-class object with a base-class pointer
  Allowed
- Referring to a derived-class object with a derived-class pointer
  - Allowed
- Referring to a derived-class object with a base-class pointer
  - Possible syntax error
  - Code can only refer to base-class members, or syntax error
- Referring to a base-class object with a derived-class pointer
  - Syntax error
  - The derived-class pointer must first be cast to a base-class pointer



# Point, Circle, Cylinder

- Point, circle, cylinder hierarchy
  - Point class is base class
  - Circle class is derived from Point class
  - Cylinder class is derived from Circle class

```
// Fig. 9.8: point2.h
 2 // Definition of class Point
 3 #ifndef POINT2_H
 4 #define POINT2_H
 5
 6 #include <iostream>
 7
8 using std::ostream;
9
10 class Point {
11 friend ostream &operator<<( ostream &, const Point & );</pre>
12 public:
                                       // default constructor
     Point( int = 0, int = 0 );
13
     void setPoint( int, int ); // set coordinates
int getX() const { return x; } // get x coordinate
14
15
16
    int getY() const { return y; } // get y coordinate
                   // accessible to derived classes
17 protected:
                  // coordinates of the point
18
      int x, y;
19 };
20
21 #endif
22 // Fig. 9.8: point2.cpp
23 // Member functions for class Point
24 #include "point2.h"
25
26 // Constructor for class Point
27 Point::Point( int a, int b ) { setPoint( a, b ); }
28
29 // Set the x and y coordinates
30 void Point::setPoint( int a, int b )
31 {
32
```

```
33 y = b;
34 }
35
36 // Output the Point
37 ostream &operator<<( ostream &output, const Point &p )
38 {
39 output << '[' << p.x << ", " << p.y << ']';
40
41 return output; // enables cascading
42 }
```

```
1 // Fig. 9.9: circle2.h
 2 // Definition of class Circle
 3 #ifndef CIRCLE2 H
 4 #define CIRCLE2 H
 5
 6 #include <iostream>
 7
 8 using std::ostream;
 9
10 #include "point2.h"
11
12 class Circle : public Point {
      friend ostream &operator<<( ostream &, const Circle & );
13
14 public:
     // default constructor
Circle( double r = 0.0, int x = 0, int y = 0 );
15
16
     void setRadius( double ); // set radius
double getRadius() const; // return radius
double area() const; // calculate area
17
18
19
                                         // calculate area
20 protected: // accessible to derived classes
21 double radius; // radius of the Circle
22 };
23
24 #endif
25 // Fig. 9.9: circle2.cpp
26 // Member function definitions for class Circle
27 #include <iomanip>
28
29 using std::ios;
30 using std::setiosflags;
31 using std::setprecision;
32
33 #include "circle2.h"
```

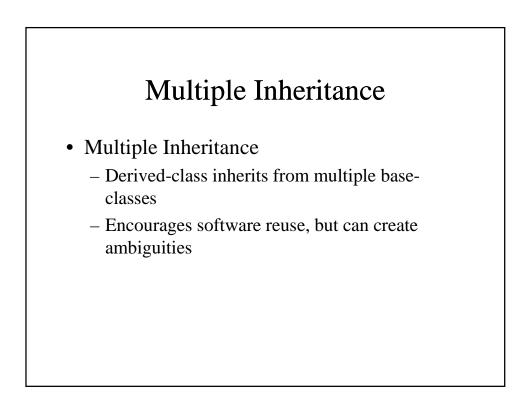
```
34
35 // Constructor for Circle calls constructor for Point
36 // with a member initializer and initializes radius
 37 Circle::Circle( double r, int a, int b )
38 : Point( a, b )
                          // call base-class constructor
39 { setRadius( r ); }
40
41 // Set radius
42 void Circle::setRadius( double r )
43 { radius = ( r >= 0 ? r : 0 ); }
44
 45 // Get radius
46 double Circle::getRadius() const { return radius; }
47
48 // Calculate area of Circle
 49 double Circle::area() const
50 { return 3.14159 * radius * radius; }
51
52 // Output a circle in the form:
53 // Center = [x, y]; Radius = #.##
 54 ostream &operator<<( ostream &output, const Circle &c )
55 {
56 output << "Center = " << static_cast< Point > ( c )
57
            << "; Radius = "
58
             << setiosflags( ios::fixed | ios::showpoint )
            << setprecision( 2 ) << c.radius;
59
60
61 return output; // enables cascaded calls
62 }
```

```
1 // Fig. 9.10: cylindr2.h
2 // Definition of class Cylinder
3 #ifndef CYLINDR2_H
4 #define CYLINDR2_H
 5
6 #include <iostream>
8 using std::ostream;
 9
10 #include "circle2.h"
11
12 class Cylinder : public Circle {
13
    friend ostream &operator<<( ostream &, const Cylinder & );
14
15 public:
16 // default constructor
17
    Cylinder( double h = 0.0, double r = 0.0,
18
              int x = 0, int y = 0);
19
20
    void setHeight( double ); // set height
21
     double getHeight() const; // return height
                               // calculate and return area
22
     double area() const;
     double volume() const;
23
                               // calculate and return volume
24
25 protected:
26 double height;
                              // height of the Cylinder
27 };
28
29 #endif
```

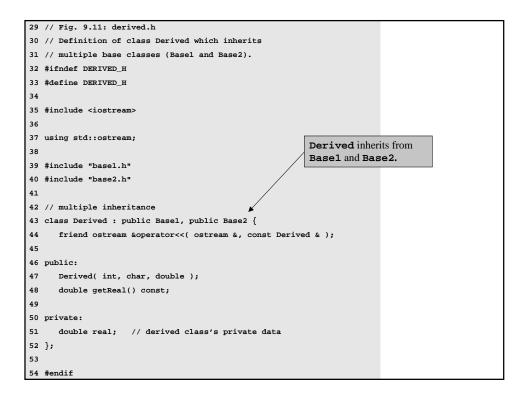
```
30 // Fig. 9.10: cylindr2.cpp
31 // Member and friend function definitions
 32 // for class Cylinder.
33 #include "cylindr2.h"
34
35 // Cylinder constructor calls Circle constructor
36 Cylinder::Cylinder( double h, double r, int x, int y )
37 : Circle( r, x, y ) // call base-class constructor
38 { setHeight( h ); }
39
40 // Set height of Cylinder
41 void Cylinder::setHeight( double h )
42
    { height = ( h >= 0 ? h : 0 ); }
43
44 // Get height of Cylinder
45 double Cylinder::getHeight() const { return height; }
 46
47 // Calculate area of Cylinder (i.e., surface area)
48 double Cylinder::area() const
 49 {
50 return 2 * Circle::area() +
           2 * 3.14159 * radius * height;
51
                                                           Circle::area() is
52 }
                                                           overidden.
53
54 // Calculate volume of Cylinder
55 double Cylinder::volume() const
56 { return Circle::area() * height; }
57
58 // Output Cylinder dimensions
59 ostream &operator<<( ostream &output, const Cylinder &c )
60 {
```

```
61
      output << static_cast< Circle >( c )
            << "; Height = " << c.height;
62
63
64
     return output; // enables cascaded calls
65 }
66 // Fig. 9.10: fig09_10.cpp
67 // Driver for class Cylinder
68 #include <iostream>
69
70 using std::cout;
71 using std::endl;
72
73 #include "point2.h"
74 #include "circle2.h"
75 #include "cylindr2.h"
76
77 int main()
78 {
79
     // create Cylinder object
    Cylinder cyl( 5.7, 2.5, 12, 23 );
80
81
                                                              X coordinate is 12
82
     // use get functions to display the Cylinder
                                                               Y coordinate is 23
     cout << "X coordinate is " << cyl.getX()</pre>
83
          << "\nY coordinate is " << cyl.getY()
                                                               Radius is 2.5
84
85
           << "\nRadius is " << cyl.getRadius()
                                                              Height is 5.7
           << "\nHeight is " << cyl.getHeight() << "\n\n";
86
87
     // use set functions to change the Cylinder's attributes
88
89
      cvl.setHeight( 10 );
90
      cyl.setRadius( 4.25 );
      cyl.setPoint( 2, 2 );
91
```

```
92
      cout << "The new location, radius, and height of cyl are:\n"
93
            << cyl << '\n';
94
95
      cout << "The area of cyl is:\n"
96
           << cyl.area() << '\n';
97
98
      // display the Cylinder as a Point
99
      Point &pRef = cyl; // pRef "thinks" it is a Point
100
     cout << "\nCylinder printed as a Point is: "</pre>
101
            << pRef << "\n\n";
102
103
     // display the Cylinder as a Circle
104
      Circle &circleRef = cyl; // circleRef thinks it is a Circle
105 cout << "Cylinder printed as a Circle is:\n" << circleRef
106
            << "\nArea: " << circleRef.area() << endl;
107
108 return 0;
109}
X coordinate is 12
Y coordinate is 23
Radius is 2.5
Height is 5.7
The new location, radius, and height of cyl are:
Center = [2, 2]; Radius = 4.25; Height = 10.00
The area of cyl is:
380.53
Cylinder printed as a Point is: [2, 2]
Cylinder printed as a Circle is:
Center = [2, 2]; Radius = 4.25
Area: 56.74
```



```
1 // Fig. 9.11: base1.h
 2 // Definition of class Basel
 3 #ifndef BASE1_H
 4 #define BASE1_H
 5
 6 class Base1 {
 7 public:
    Basel( int x ) { value = x; }
 8
 9
    int getData() const { return value; }
10 protected: // accessible to derived classes
11 int value; // inherited by derived class
12 };
 13
14 #endif
15 // Fig. 9.11: base2.h
16 // Definition of class Base2
17 #ifndef BASE2 H
18 #define BASE2_H
19
20 class Base2 {
21 public:
22 Base2( char c ) { letter = c; }
23 char getData() const { return letter; }
24 protected: // accessible to derived classes
25 char letter; // inherited by derived class
26 };
27
28 #endif
```



```
55 // Fig. 9.11: derived.cpp
56 // Member function definitions for class Derived
57 #include "derived.h"
58
59 // Constructor for Derived calls constructors for
60 // class Base1 and class Base2.
 61 // Use member initializers to call base-class constructors
62 Derived::Derived( int i, char c, double f )
63
      : Base1( i ), Base2( c ), real ( f ) { }
64
65 // Return the value of real
66 double Derived::getReal() const { return real; }
 67
68 // Display all the data members of Derived
69 ostream &operator<<( ostream &output, const Derived &d )
70 {
     output << " Integer: " << d.value
71
           << "\n Character: " << d.letter
72
73
             << "\nReal number: " << d.real;
74
75
     return output; // enables cascaded calls
76 }
 77 // Fig. 9.11: fig09_11.cpp
78 // Driver for multiple inheritance example
79 #include <iostream>
 80
81 using std::cout;
82 using std::endl;
83
84 #include "base1.h"
85 #include "base2.h"
```

```
86 #include "derived.h"
87
88 int main()
89 {
90 Basel b1( 10 ), *baselPtr = 0; // create Basel object
     Base2 b2( 'Z' ), *base2Ptr = 0; // create Base2 object
Derived d( 7, 'A', 3.5 ); // create Derived object
91
92
                                      // create Derived object
93
94
      // print data members of base class objects
95
      cout << "Object b1 contains integer " << b1.getData()</pre>
96
          << "\nObject b2 contains character " << b2.getData()
            << "\nObject d contains:\n" << d << "\n\n";
97
98
99
     // print data members of derived class object
100
      // scope resolution operator resolves getData ambiguity
     cout << "Data members of Derived can be"
101
102
           << " accessed individually:"
 103
           << "\n Integer: " << d.Basel::getData()
          << "\n Character: " << d.Base2::getData()
104
           << "\nReal number: " << d.getReal() << "\n\n";
105
106
107 cout << "Derived can be treated as an "
108
           << "object of either base class:\n";
109
110 // treat Derived as a Basel object
111 base1Ptr = &d;
112
      cout << "baselPtr->getData() yields "
113
            << baselPtr->getData() << '\n';
114
115 // treat Derived as a Base2 object
116 base2Ptr = &d;
```