

0.0.1 Point/Circle Hierarchy using private data

Class **Point3** (Figs. 1-2) declares data members **x** and **y** as **private** and exposes member functions **setX**, **getX**, **setY**, **getY** and **print** for manipulating these values.

1 Case Study: Three-Level Inheritance Hierarchy

Three level point/circle/cylinder hierarchy

- Point
 - x-y coordinate pair
- Circle
 - x-y coordinate pair
 - Radius
- Cylinder
 - x-y coordinate pair
 - Radius
 - Height

Derive class **Cylinder** from class **Circle4**. Class **Cylinder** should redefine member functions **getArea** and **print** member functions. Figs. 7-8 present class **Cylinder**, which inherits from class **Circle4**. We were able to develop classes **Circle4** and **Cylinder** much more quickly by using inheritance than if we had developed these classes "from scratch". Inheritance avoids duplicating code and the associated code-maintenance problems.

```

1 // Fig. 9.17: point3.h
2 // Point3 class definition represents an x-y coordinate pair.
3 #ifndef POINT3_H
4 #define POINT3_H
5
6 class Point3 {
7
8 public:
9     Point3( int = 0, int = 0 ); // default constructor
10
11     void setX( int ); // set x in coordinate pair
12     int getX() const; // return x from coordinate pair
13
14     void setY( int ); // set y in coordinate pair
15     int getY() const; //
16
17     void print() const; //
18
19 private:
20     int x; // x part of coordinate pair
21     int y; // y part of coordinate pair
22
23 }; // end class Point3
24
25 #endif

```

Better software-engineering practice: **private** over **protected** when possible.

```



1 // Fig. 9.18: point3.cpp
2 // Point3 class member-function definitions.
3 #include <iostream>
4
5 using std::cout;
6
7 #include "point3.h" // Point3 class definition
8
9 // default constructor
10 Point3::Point3( int xValue, int yValue )
11     : x( xValue ), y( yValue )
12 {
13     // empty body
14 } // end Point3 constructor
15
16 // set x in coordinate pair
17 void Point3::setX( int xValue )
18 {
19     x = xValue; // no need for validation
20 } // end function setX
21
22
23

```

Member initializers specify values of **x** and **y**.



Figure 1: **Point3** class header file. Point/Circle Hierarchy Using **private** Data

```
24 // return x from coordinate pair
25 int Point3::getX() const
26 {
27     return x;
28 }
29 // end function getX
30
31 // set y in coordinate pair
32 void Point3::setY( int yValue )
33 {
34     y = yValue; // no need for validation
35 }
36 // end function setY
37
38 // return y from coordinate pair
39 int Point3::getY() const
40 {
41     return y;
42 }
43 // end function getY
44
```

49  Outline
 point3.cpp (2 of 3)

© 2003 Prentice Hall, Inc.
All rights reserved.

```
45 // output Point3 object
46 void Point3::print() const
47 {
48     cout << '[' << getX() << ", " << getY() << ']'<< endl;
49 }
50 // end function print
```

50  Outline
 point3.cpp (3 of 3)

Invoke non-private member functions to access private data.

© 2003 Prentice Hall, Inc.
All rights reserved.

Figure 2: **Point3** class uses member functions to manipulate its **private** data.

```
1 // Fig. 9.19: circle4.h
2 // Circle4 class contains x-y coordinate pair and radius.
3 #ifndef CIRCLE4_H
4 #define CIRCLE4_H
5
6 #include "point3.h" // Point3
7
8 class Circle4 : public Point3 {
9
10 public:
11
12 // default constructor
13 Circle4( int = 0, int = 0, double = 0.0 );
14
15 void setRadius( double ); // set radius
16 double getRadius() const; // return radius
17
18 double getDiameter() const; // return diameter
19 double getCircumference() const; // return circumference
20 double getArea() const; // return area
21
22 void print() const; //
23
24 private:
25 double radius; // Circle4's radius
```

Class Circle4 inherits from class Point3.

Maintain private data member radius.

```
26
27 }; // end class Circle4
28
29 #endif
```

Figure 3: Circle4 class header file.

```
1 // Fig. 9.20: circle4.cpp
2 // Circle4 class member-function definitions.
3 #include <iostream>
4
5 using std::cout;
6
7 #include "circle4.h" // Circle4 cl
8
9 // default constructor
10 Circle4::Circle4( int xValue, int yV
11 : Point3( xValue, yValue ) // call base-class constructor
12 {
13     setRadius( radiusValue );
14
15 } // end Circle4 constructor
16
17 // set radius
18 void Circle4::setRadius( double radiusValue )
19 {
20     radius = ( radiusValue < 0.0 ? 0.0 : radiusValue );
21
22 } // end function setRadius
23
```

Base-class initializer syntax passes arguments to base class Point3.

```
24 // return radius
25 double Circle4::getRadius() const
26 {
27     return radius;
28
29 } // end function getRadius
30
31 // calculate and return diameter
32 double Circle4::getDiameter() const
33 {
34     return 2 * getRadius();
35
36 } // end function getDiameter
37
38 // calculate and return circumference
39 double Circle4::getCircumference() const
40 {
41     return 3.14159 * getDiameter();
42
43 } // end function getCircumference
44
```

Invoke function getRadius rather than directly accessing data member radius.

Figure 4: Circle4 class that inherits from class Point3, which does not provide protected data. (part 1 of 2)

```

45 // calculate and return area
46 double Circle4::getArea() const
47 {
48     return 3.14159 * getRadius() * getRadius();
49 } // end function getArea
50
51
52 // output Circle4 object
53 void Circle4::print() const
54 {
55     cout << "Center = ";
56     Point3::print(); // invoke P
57     cout << "; Radius = " << getRadius();
58
59 } // end function print

```

▲ Outline 55
▼
circle4.cpp (3 of 3)

Redefine class **Point3**'s member function **print**.
Invoke function **getRadius**
Invoke base-class **Point3**'s **print** function using binary scope-resolution operator (**::**).

© 2003 Prentice Hall, Inc.
All rights reserved.

```

1 // Fig. 9.21: circletest4.cpp
2 // Testing class Circle4.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7 using std::fixed;
8
9 #include <iomanip>
10
11 using std::setprecision;
12
13 #include "circle4.h" // Circle4 class definition
14
15 int main()
16 {
17     Circle4 circle( 37, 43, 2.5 ); // instantiate Circle4 object
18
19     // display point coordinates
20     cout << "X coordinate is " << circle.getX()
21         << "\nY coordinate is " << circle.getY()
22         << "\nRadius is " << circle.getRadius();
23

```

▲ Outline 56
▼
circletest4.cpp (1 of 2)

Create **Circle4** object.
Use inherited get functions to access inherited **protected**
Use **Circle3** get function to access **private** data **radius**.

© 2003 Prentice Hall, Inc.
All rights reserved.

Figure 5: **Circle4** class that inherits from class **Point3**, which does not provide **protected** data. (part 2 of 2)

```

24 circle.setX( 2 ); // set new x-coordinate
25 circle.setY( 2 ); // set new y-coordinate
26 circle.setRadius( 4.25 ); // set new radius
27
28 // display new circle value
29 cout << "\n\nThe new location and radius
30 circle.print();
31
32 // display floating-point values with 2 digits of precision
33 cout << fixed << setprecision( 2 );
34
35 // display Circle4's diameter
36 cout << "\nDiameter is " << circle.getDiameter();
37
38 // display Circle4's circumference
39 cout << "\nCircumference is " << circle.getCircumference();
40
41 // display Circle4's area
42 cout << "\nArea is " << circle.getArea();
43
44 cout << endl;
45
46 return 0; // indicates successful termination
47
48 } // end main

```

Use inherited set functions to modify inherited
 Use Circle3 set function to modify private data radius.

Outline 57
 circletest4.cpp (2 of 2)

© 2003 Prentice Hall, Inc. All rights reserved.

```

X coordinate is 37
Y coordinate is 43
Radius is 2.5

The new location and radius of circle are
Center = [2, 2]; Radius = 4.25
Diameter is 8.50
Circumference is 26.70
Area is 56.74

```

Outline 58
 circletest4.cpp output (1 of 1)

© 2003 Prentice Hall, Inc. All rights reserved.

Figure 6: Base class **private** data is accessible to a derived class via **public** or **protected** member function inherited by the derived class.

```
1 // Fig. 9.22: cylinder.h
2 // Cylinder class inherits from class Circle4.
3 #ifndef CYLINDER_H
4 #define CYLINDER_H
5
6 #include "circle4.h" // Circle4
7
8 class Cylinder : public Circle4 {
9
10 public:
11
12 // default constructor
13 Cylinder( int = 0, int = 0, double = 0.0, double = 0.0 );
14
15 void setHeight( double ); // set Cylinder's height
16 double getHeight() const; // return Cylinder's height
17
18 double getArea() const; // return Cylinder's area
19 double getVolume() const; // return Cylinder's volume
20 void print() const; // print Cylinder's data
21
22 private:
23 double height; // Cylinder's height
24
25 }; // end class Cylinder
```

Class **Cylinder** inherits from class **Circle4**.

Maintain private data member **height**.

```
26
27 #endif

1 // Fig. 9.23: cylinder.cpp
2 // Cylinder class inherits from class Circle4.
3 #include <iostream>
4
5 using std::cout;
6
7 #include "cylinder.h" // Cylinder class definition
8
9 // default constructor
10 Cylinder::Cylinder( int xValue, int yValue,
11 double heightValue )
12 : Circle4( xValue, yValue, radiusValue )
13 {
14 setHeight( heightValue );
15
16 } // end Cylinder constructor
17
```

Base-class initializer syntax passes arguments to base class **Circle4**.

Figure 7: **Cylinder** class header file.


```

18 // set Cylinder's height
19 void Cylinder::setHeight( double heightValue )
20 {
21     height = ( heightValue < 0.0 ? 0.0 : heightValue );
22 }
23 // end function setHeight
24
25 // get Cylinder's height
26 double Cylinder::getHeight() const
27 {
28     return height;
29 }
30 // end function getHeight
31
32 // redefine Circle4 function getArea to
33 double Cylinder::getArea() const
34 {
35     return 2 * Circle4::getArea() +
36         getCircumference() * getHeight();
37 }
38 // end function getArea
39

```

Outline 62
cylinder.cpp (2 of 3)

Invoke base-class Circle4's getArea function using binary scope-resolution operator (::).
Redefine base class Circle4's getArea function using binary scope-resolution operator (::).

© 2003 Prentice Hall, Inc.
All rights reserved.

```

40 // calculate Cylinder volume
41 double Cylinder::getVolume() const
42 {
43     return Circle4::getArea() * getHeight();
44 }
45 // end function getVolume
46
47 // output Cylinder object
48 void Cylinder::print() const
49 {
50     Circle4::print();
51     cout << " Height = " << getHeight();
52 }
53 // end function print

```

Outline 63
cylinder.cpp (3 of 3)

Invoke base-class Circle4's getArea function using binary scope-resolution operator (::).
Redefine class Circle4's print function using binary scope-resolution operator (::).
Invoke base-class Circle4's print function using binary scope-resolution operator (::).

© 2003 Prentice Hall, Inc.
All rights reserved.

Figure 8: **Cylinder** class inherits from class **Circle4** and redefines member function **getArea**.

```

1 // Fig. 9.24: cylindertest.cpp
2 // Testing class Cylinder.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7 using std::fixed;
8
9 #include <iomanip>
10
11 using std::setprecision;
12
13 #include "cylinder.h" // Cylinder class definition
14
15 int main()
16 {
17     // instantiate Cylinder object
18     Cylinder cylinder( 12, 23, 2.5, 5.7 );
19
20     // display point coordinates
21     cout << "X coordinate is " << cylinder.getX()
22          << "\nY coordinate is " << cylinder.getY()
23          << "\nRadius is " << cylinder.getRadius()
24          << "\nHeight is " << cylinder.getHeight();
25

```

64
 Outline
 cylindertest.cpp
 (1 of 3)

Invoke indirectly inherited
 Point3 member functions.
 Invoke directly inherited
 Cylinder member
 function.

© 2003 Prentice Hall, Inc.
 All rights reserved.

```

26     cylinder.setX( 2 ); // set new x-coordinate
27     cylinder.setY( 2 ); // set new
28     cylinder.setRadius( 4.25 ); // set new
29     cylinder.setHeight( 10 ); // set new
30
31     // display new cylinder value
32     cout << "\n\nThe new location and radius
33     cylinder.print();
34
35     // display floating-point values
36     cout << fixed << setprecision( 2 );
37
38     // display cylinder's diameter
39     cout << "\n\nDiameter is " << cylinder.getDiameter();
40
41     // display cylinder's circumference
42     cout << "\nCircumference is "
43          << cylinder.getCircumference();
44
45     // display cylinder's area
46     cout << "\nArea is " << cylinder.getArea();
47
48     // display cylinder's volume
49     cout << "\nVolume is " << cylinder.getVolume();
50

```

65
 Outline
 cylindertest.cpp
 (2 of 3)

Invoke redefined print
 function.

Invoke redefined getArea
 function.

© 2003 Prentice Hall, Inc.
 All rights reserved.

Figure 9: Point/Circle/Cylinder hierarchy test program. (part 1 of 2)

```

51     cout << endl;
52
53     return 0; // indicates successful termination
54
55 } // end main

```

```

X coordinate is 12
Y coordinate is 23
Radius is 2.5
Height is 5.7

The new location and radius of circle are
Center = [2, 2]; Radius = 4.25; Height = 10

Diameter is 8.50
Circumference is 26.70
Area is 380.53
Volume is 567.45

```

Outline 66

cylindertest.cpp (3 of 3)

cylindertest.cpp output (1 of 1)

© 2003 Prentice Hall, Inc.
All rights reserved.

Figure 10: **Point/Circle/Cylinder** hierarchy test program. (part 2 of 2)

1.1 Constructors and Destructors in Derived Classes

- Instantiating derived-class object
 - Chain of constructor calls
 - * Derived-class constructor invokes base class constructor
 - Implicitly or explicitly
 - * Base of inheritance hierarchy
 - Last constructor called in chain
 - First constructor body to finish executing
 - Example: **Point3/Circle4/Cylinder** hierarchy
 - Point3** constructor called last
 - Point3** constructor body finishes execution first
 - * Initializing data members
 - Each base-class constructor initializes data members Inherited by derived class
- Destroying derived-class object

- Chain of destructor calls
 - * Reverse order of constructor chain
 - * Destructor of derived-class called first
 - * Destructor of next base class up hierarchy next
 - Continue up hierarchy until final base reached; After final base-class destructor, object removed from memory
- Base-class constructors, destructors, assignment operators
 - Not inherited by derived classes
 - Derived class constructors, assignment operators can call
 - * Constructors
 - * Assignment operators

Next example revisits the point/circle hierarchy by defining class **Point4** (11-12) and class **Circle5** (13-15) that contain constructors and destructors, each of which prints a message when it is invoked.

```

1 // Fig. 9.25: point4.h
2 // Point4 class definition represents an x-y coordinate pair.
3 #ifndef POINT4_H
4 #define POINT4_H
5
6 class Point4 {
7
8 public:
9     Point4( int = 0, int = 0 ); // default constructor
10    ~Point4(); // destructor
11
12    void setX( int ); // set x in coordinate pair
13    int getX() const; // return x from coordinate pair
14
15    void setY( int ); // set y in coordinate pair
16    int getY() const; // return y from coordinate pair
17
18    void print() const; // output Point3 object
19
20 private:
21     int x; // x part of coordinate pair
22     int y; // y part of coordinate pair
23
24 }; // end class Point4
25
26 #endif

```

point4.h (1 of 1)

70

Outline

Constructor and destructor output messages to demonstrate function call order.

© 2003 Prentice Hall, Inc.
All rights reserved.

```

1 // Fig. 9.26: point4.cpp
2 // Point4 class member-function definitions.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include "point4.h" // Point4 class definition
9
10 // default constructor
11 Point4::Point4( int xValue, int yValue )
12     : x( xValue ), y( yValue )
13 {
14     cout << "Point4 constructor: ";
15     print();
16     cout << endl;
17 }
18 // end Point4 constructor
19
20 // destructor
21 Point4::~Point4()
22 {
23     cout << "Point4 destructor: ";
24     print();
25     cout << endl;

```

point4.cpp (1 of 3)

71

Outline

Output message to demonstrate constructor function call order.

Output message to demonstrate destructor function call order.

© 2003 Prentice Hall, Inc.
All rights reserved.

Figure 11: **Point4** class header file and **Point4** base class contains a constructor and a destructor. (part 1 of 2)

```

26
27 } // end Point4 destructor
28
29 // set x in coordinate pair
30 void Point4::setX( int xValue )
31 {
32     x = xValue; // no need for validation
33
34 } // end function setX
35
36 // return x from coordinate pair
37 int Point4::getX() const
38 {
39     return x;
40
41 } // end function getX
42
43 // set y in coordinate pair
44 void Point4::setY( int yValue )
45 {
46     y = yValue; // no need for validation
47
48 } // end function setY
49

```



[Outline](#)

72

point4.cpp (2 of 3)

© 2003 Prentice Hall, Inc.
All rights reserved.

```

50 // return y from coordinate pair
51 int Point4::getY() const
52 {
53     return y;
54
55 } // end function getY
56
57 // output Point4 object
58 void Point4::print() const
59 {
60     cout << '[' << getX() << ", " << getY() << ']'<< endl;
61
62 } // end function print

```



[Outline](#)

73

point4.cpp (3 of 3)

© 2003 Prentice Hall, Inc.
All rights reserved.

Figure 12: **Point4** base class contains a constructor and a destructor. (part 2 of 2)

```

1 // Fig. 9.27: circle5.h
2 // Circle5 class contains x-y coordinate pair and radius.
3 #ifndef CIRCLE5_H
4 #define CIRCLE5_H
5
6 #include "point4.h" // Point4 class definition
7
8 class Circle5 : public Point4 {
9
10 public:
11
12 // default constructor
13 Circle5( int = 0, int = 0, double = 0.0 );
14
15 ~Circle5(); // destructor
16 void setRadius( double ); // set radius
17 double getRadius() const; // return radius
18
19 double getDiameter() const; // return diameter
20 double getCircumference() const; // return circumference
21 double getArea() const; // return area
22
23 void print() const; // output Circle5 object
24

```



Outline

74

circle5.h (1 of 2)

Constructor and destructor
output messages to
demonstrate function call
order.

© 2003 Prentice Hall, Inc.
All rights reserved.

```

25 private:
26     double radius; // Circle5's radius
27
28 }; // end class Circle5
29
30 #endif

```



Outline

75

circle5.h (2 of 2)

© 2003 Prentice Hall, Inc.
All rights reserved.

Figure 13: Circle5 class header file.

```
1 // Fig. 9.28: circle5.cpp
2 // Circle5 class member-function definitions.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include "circle5.h" // Circle5 class definition
9
10 // default constructor
11 Circle5::Circle5( int xValue, int yValue, double radiusValue )
12 : Point4( xValue, yValue ) // call base
13 {
14     setRadius( radiusValue );
15
16     cout << "Circle5 constructor: ";
17     print();
18     cout << endl;
19 } // end Circle5 constructor
20
21
```

Output message to demonstrate constructor function call order.

```
22 // destructor
23 Circle5::~Circle5()
24 {
25     cout << "Circle5 destructor: ";
26     print();
27     cout << endl;
28 } // end Circle5 destructor
29
30 // set radius
31 void Circle5::setRadius( double radiusValue )
32 {
33     radius = ( radiusValue < 0.0 ? 0.0 : radiusValue );
34 } // end function setRadius
35
36 // return radius
37 double Circle5::getRadius() const
38 {
39     return radius;
40 } // end function getRadius
41
42
43
44
```

Output message to demonstrate destructor function call order.

Figure 14: **Circle5** class inherits from class **Point4**. (part 1 of 2)


```

45 // calculate and return diameter
46 double Circle5::getDiameter() const
47 {
48     return 2 * getRadius();
49 }
50 } // end function getDiameter
51
52 // calculate and return circumference
53 double Circle5::getCircumference() const
54 {
55     return 3.14159 * getDiameter();
56 }
57 } // end function getCircumference
58
59 // calculate and return area
60 double Circle5::getArea() const
61 {
62     return 3.14159 * getRadius() * getRadius();
63 }
64 } // end function getArea
65

```



[Outline](#)

78

circle5.cpp (3 of 4)

© 2003 Prentice Hall, Inc.
All rights reserved.

```

66 // output Circle5 object
67 void Circle5::print() const
68 {
69     cout << "Center = ";
70     Point4::print(); // invoke Point4's print function
71     cout << "; Radius = " << getRadius();
72 }
73 } // end function print

```



[Outline](#)

79

circle5.cpp (4 of 4)

© 2003 Prentice Hall, Inc.
All rights reserved.

Figure 15: **Circle5** class inherits from class **Point4**. (part 2 of 2)

```

1 // Fig. 9.29: fig09_29.cpp
2 // Display order in which base-class and derived-class
3 // constructors are called.
4 #include <iostream>
5
6 using std::cout;
7 using std::endl;
8
9 #include "circle5.h" // Circle5 class definition
10
11 int main()
12 {
13     { // begin new scope
14
15         Point4 point( 11, 22 );
16
17     } // end scope
18
19     cout << endl;
20     Circle5 circle1( 72, 29, 4.5 );
21
22     cout << endl;
23     Circle5 circle2( 5, 5, 10 );
24
25     cout << endl;

```

Point4 object goes in and out of scope immediately.

Instantiate two Circle5 objects to demonstrate order of derived-class and base-class constructor/destructor function calls.

80

Outline

fig09_29.cpp (1 of 2)

© 2003 Prentice Hall, Inc. All rights reserved.

```

26
27     return 0; // indicates successful termination
28
29 } // end main

```

```

Point4 constructor: [11, 22]
Point4 destructor: [11, 22]

Point4 constructor: [72, 29]
Circle5 constructor: Center = [72, 29]; Radius = 4.5

Point4 constructor: [5, 5]
Circle5 constructor: Center = [5, 5]; Radius = 10

Circle5 destructor: Center = [5, 5]; Radius = 10
Point4 destructor: [5, 5]
Circle5 destructor: Center = [72, 29]; Radius = 4.5
Point4 destructor: [72, 29]

```

Point4 constructor called for object in block; destructor

Derived-class Circle5 constructor body executes

Derived-class Circle5 constructor body executes

Destructors for Circle5 object called in reverse order

Destructors for Circle5 object called in reverse order of constructors.

81

Outline

fig09_29.cpp (2 of 2)

fig09_29.cpp output (1 of 1)

© 2003 Prentice Hall, Inc. All rights reserved.

Figure 16: Constructor and destructor call order.

1.2 "Uses A" and "Knows A" Relationships

- "Uses a"
 - Object uses another object
 - * Call non-**private** member function; using pointer, reference or object name
- "Knows a" (association)
 - Object aware of another object; contain pointer handle or reference handle
 - Knowledge networks

1.3 public, protected and private Inheritance

9.8 public, protected and private Inheritance

83

Base class member access specifier	Type of inheritance		
	public inheritance	protected inheritance	private inheritance
Public	public in derived class. Can be accessed directly by any non- static member functions, friend functions and non-member functions.	protected in derived class. Can be accessed directly by all non- static member functions and friend functions.	private in derived class. Can be accessed directly by all non- static member functions and friend functions.
Protected	protected in derived class. Can be accessed directly by all non- static member functions and friend functions.	protected in derived class. Can be accessed directly by all non- static member functions and friend functions.	private in derived class. Can be accessed directly by all non- static member functions and friend functions.
Private	Hidden in derived class. Can be accessed by non- static member functions and friend functions through public or protected member functions of	Hidden in derived class. Can be accessed by non- static member functions and friend functions through public or protected member functions of the base class.	Hidden in derived class. Can be accessed by non- static member functions and friend functions through public or protected member functions of the base class.

© 2003 Prentice Hall, Inc. All rights reserved.



Figure 17: Summary of base-class member accessibility in a derived class.

1.4 Software Engineering with Inheritance

Customizing existing software

- Inherit from existing classes
 - Include additional members
 - Redefine base-class members
 - No direct access to base class's source code; Link to object code
- Independent software vendors (ISVs)
 - Develop proprietary code for sale/license; available in object-code format
 - Users derive new classes; without accessing ISV proprietary source code