

# 1 Object-Oriented Programming: Inheritance

## 1.1 Introduction

- Inheritance
  - Software reusability
  - Create new class from existing class
    - \* Absorb existing class's data and behaviors
    - \* Enhance with new capabilities
  - Derived class inherits from base class
    - \* Derived class
      - More specialized group of objects
      - Behaviors inherited from base class; can customize
      - Additional behaviors
- Class hierarchy
  - Direct base class; inherited explicitly (one level up hierarchy)
  - Indirect base class; inherited two or more levels up hierarchy
  - Single inheritance; inherits from one base class
  - Multiple inheritance; Inherits from multiple base classes (Base classes possibly unrelated ); Chapter 22
- Three types of inheritance
  - **public**
    - \* Every object of derived class also object of base class
      - Base-class objects not objects of derived classes
      - Example: All cars vehicles, but not all vehicles cars
    - \* Can access non-**private** members of base class
      - Derived class can effect change to **private** base-class members
      - Through inherited non-**private** member functions
  - **private**
    - \* Alternative to composition
    - \* Chapter 17

- **protected**
  - \* Rarely used
- Abstraction
  - Focus on commonalities among objects in system; "is-a" vs. "has-a"
  - "is-a"
    - \* Inheritance
    - \* Derived class object treated as base class object
    - \* Example: Car *is a* vehicle; Vehicle properties/behaviors also car properties/behaviors
  - "has-a"
    - \* Composition
    - \* Object contains one or more objects of other classes as members
    - \* Example: Car *has a* steering wheel

## 1.2 Base Classes and Derived Classes

- Base classes and derived classes
  - Object of one class "is an" object of another class
    - \* Example: Rectangle is quadrilateral.
      - Class **Rectangle** inherits from class **Quadrilateral**
      - **Quadrilateral**: base class
      - **Rectangle**: derived class
  - Base class typically represents larger set of objects than derived classes
    - \* Example:
      - Base class: **Vehicle**  
Cars, trucks, boats, bicycles, ...
      - Derived class: **Car**  
Smaller, more-specific subset of vehicles
- Inheritance examples (see Fig. 1)
- Inheritance hierarchy (see Fig. 2 Top)

## 9.2 Base Classes and Derived Classes

- Inheritance examples

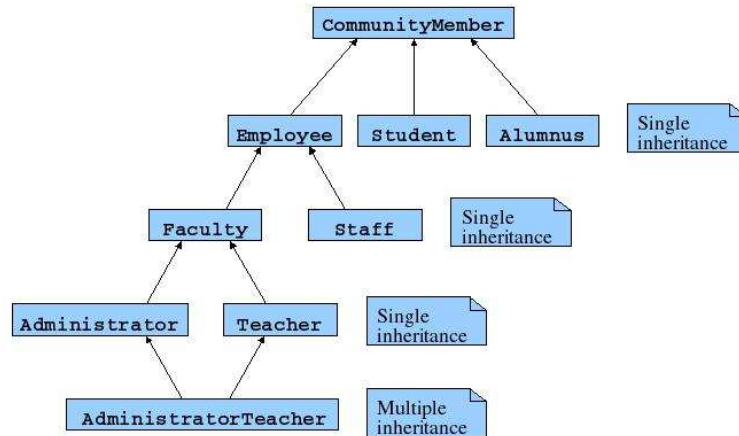
Base class	Derived classes
Student	GraduateStudent UndergraduateStudent
Shape	Circle Triangle Rectangle
Loan	CarLoan HomeImprovementLoan MortgageLoan
Employee	FacultyMember StaffMember
Account	CheckingAccount SavingsAccount

© 2003 Prentice Hall, Inc. All rights reserved.

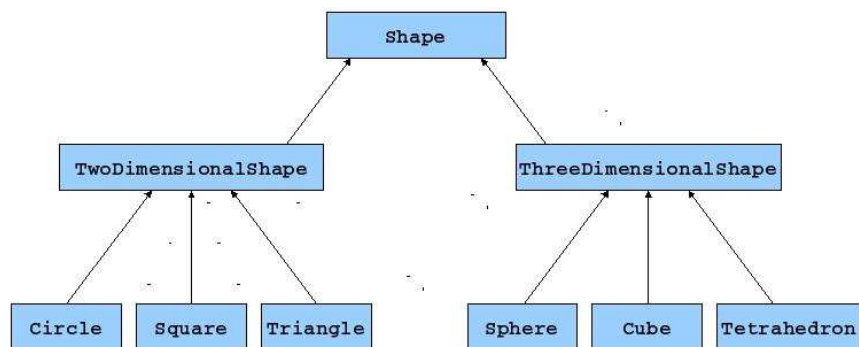


Figure 1: Inheritance examples

- Inheritance relationships: tree-like hierarchy structure
- Each class becomes
  - \* Base class; supply data/behaviors to other classes
  - \* OR
  - \* Derived class; inherit data/behaviors from other classes
- public inheritance
  - Specify with:
  - **Class TwoDimensionalShape : public Shape**  
Class **TwoDimensionalShape** inherits from class **Shape** (see Fig. 2 Bottom)
  - Base class **private** members
    - \* Not accessible directly
    - \* Still inherited; manipulate through inherited member functions

Fig. 9.2 Inheritance hierarchy for university **CommunityMembers**.

© 2003 Prentice Hall, Inc. All rights reserved.

Fig. 9.3 Inheritance hierarchy for **Shapes**.

© 2003 Prentice Hall, Inc. All rights reserved.

Figure 2: Inheritance hierarchy for university **CommunityMembers** and Inheritance hierarchy for **Shapes**

- Base class **public** and **protected** members; inherited with original member access
- **friend** functions; not inherited

### 1.3 protected Members

Protected access

- Intermediate level of protection between **public** and **private**
- **protected** members accessible to
  - Base class members
  - Base class **friends**
  - Derived class members
  - Derived class **friends**
- Derived-class members
  - Refer to **public** and **protected** members of base class; simply use member names

### 1.4 Relationship between *Base Classes* and *Derived Classes*

- Base class and derived class relationship
- Example: Point/circle inheritance hierarchy
  - Point  
x-y coordinate pair
  - Circle  
x-y coordinate pair  
Radius

```
1 // Fig. 9.4: point.h
2 // Point class definition represents an x-y coordinate pair.
3 #ifndef POINT_H
4 #define POINT_H
5
6 class Point {
7
8 public:
9     Point( 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;       // return y from coordinate pair
16
17     void print() const;      // output Point object
18
19 private:
20     int x; // x part of coordinate pair
21     int y; // y part of coordinate pair
22
23 }; // end class Point
24
25 #endif
```

Maintain **x**- and **y**-coordinates as **private** data members.

Figure 3: **Point** class header file

```

1 // Fig. 9.5: point.cpp
2 // Point class member-function definitions.
3 #include <iostream>
4
5 using std::cout;
6
7 #include "point.h" // Point class definition
8
9 // default constructor
10 Point::Point( int xValue, int yValue )
11 {
12     x = xValue;
13     y = yValue;
14 }
15 // end Point constructor
16
17 // set x in coordinate pair
18 void Point::setX( int xValue )
19 {
20     x = xValue; // no need for validation
21 }
22 // end function setX
23

```



[Outline](#)

15

point.cpp (1 of 3)

© 2003 Prentice Hall, Inc.  
All rights reserved.

```

24 // return x from coordinate pair
25 int Point::getX() const
26 {
27     return x;
28 }
29 // end function getX
30
31 // set y in coordinate pair
32 void Point::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 Point::getY() const
40 {
41     return y;
42 }
43 // end function getY
44

```



[Outline](#)

16

point.cpp (2 of 3)

© 2003 Prentice Hall, Inc.  
All rights reserved.

Figure 4: **Point** class represents an xy-coordinate pair. (part 1 of 2)

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



Outline

17

point.cpp (3 of 3)

© 2003 Prentice Hall, Inc.  
All rights reserved.

Figure 5: **Point** class represents an xy-coordinate pair. (part 2 of 2)



```

1 // Fig. 9.6: pointtest.cpp
2 // Testing class Point.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include "point.h" // Point class definition
9
10 int main()
11 {
12     Point point( 72, 115 ); // instantiate Point object
13
14     // display point coordinates
15     cout << "X coordinate is " << point.getX() << endl;
16     cout << "Y coordinate is " << point.getY() << endl;
17
18     point.setX( 10 ); // set x-coordinate
19     point.setY( 10 ); // set y-coordinate
20
21     // display new point value
22     cout << "\n\nThe new location of point is [10, 10]" << endl;
23     point.print();
24     cout << endl;
25

```

Create a **Point** object.

Invoke set functions to modify **private** data.

Invoke public function **print** to display new coordinates.

18

Outline

pointtest.cpp (1 of 2)

© 2003 Prentice Hall, Inc.  
All rights reserved.

```

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

```

X coordinate is 72  
Y coordinate is 115

The new location of point is [10, 10]

19

Outline

pointtest.cpp (2 of 2)

pointtest.cpp output (1 of 1)

© 2003 Prentice Hall, Inc.  
All rights reserved.

Figure 6: **Point** class test program.

## 1.4.1 Creating a Circle class without using inheritance

```
1 // Fig. 9.7: circle.h
2 // Circle class contains x-y coordinate pair and radius.
3 #ifndef CIRCLE_H
4 #define CIRCLE_H
5
6 class Circle {
7
8 public:
9
10 // default constructor
11 Circle( int = 0, int = 0, double = 0
12
13 void setX( int ); // set x
14 int getX() const; // return x from coordinate pair
15
16 void setY( int ); // set y in coordinate pair
17 int getY() const; // return y from coordinate pair
18
19 void setRadius( double ); // set radius
20 double getRadius() const; // return radius
21
22 double getDiameter() const; // return diameter
23 double getCircumference() const; // return circumference
24 double getArea() const; // return area
25
```

Note code similar to Point code.

20

Outline

circle.h (1 of 2)

© 2003 Prentice Hall, Inc.  
All rights reserved.

```
26 void print() const; // output Circle object
27
28 private:
29 int x; // x-coordinate
30 int y; // y-coordinate of Circle's center
31 double radius; // Circle's radius
32
33 }; // end class Circle
34
35 #endif
```

Maintain x-y coordinates and radius as private data members.

Note code similar to Point code.

21

Outline

circle.h (2 of 2)

© 2003 Prentice Hall, Inc.  
All rights reserved.

Figure 7: Circle class header file.

```

1 // Fig. 9.8: circle.cpp
2 // Circle class member-function definitions.
3 #include <iostream>
4
5 using std::cout;
6
7 #include "circle.h" // Circle class definition
8
9 // default constructor
10 Circle::Circle( int xValue, int yValue, double radiusValue )
11 {
12     x = xValue;
13     y = yValue;
14     setRadius( radiusValue );
15 }
16 // end Circle constructor
17
18 // set x in coordinate pair
19 void Circle::setX( int xValue )
20 {
21     x = xValue; // no need for validation
22 }
23 // end function setX
24

```



[Outline](#)

22

circle.cpp (1 of 4)

© 2003 Prentice Hall, Inc.  
All rights reserved.

```

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

```



[Outline](#)

23

circle.cpp (2 of 4)

© 2003 Prentice Hall, Inc.  
All rights reserved.

Figure 8: **Circle** class contains an xy-coordinate pair and a radius. (part 1 of 2)

```
46 // set radius
47 void Circle::setRadius( double radiusValue )
48 {
49     radius = ( radiusValue < 0.0 ? 0.0 : radiusValue );
50 }
51 // end function setRadius
52
53 // return radius
54 double Circle::getRadius() const
55 {
56     return radius;
57 }
58 // end function getRadius
59
60 // calculate and return diameter
61 double Circle::getDiameter() const
62 {
63     return 2 * radius;
64 }
65 // end function getDiameter
66
```

Ensure non-negative value for radius.

24  
Outline  
circle.cpp (3 of 4)

© 2003 Prentice Hall, Inc.  
All rights reserved.

```
67 // calculate and return circumference
68 double Circle::getCircumference() const
69 {
70     return 3.14159 * getDiameter();
71 }
72 // end function getCircumference
73
74 // calculate and return area
75 double Circle::getArea() const
76 {
77     return 3.14159 * radius * radius;
78 }
79 // end function getArea
80
81 // output Circle object
82 void Circle::print() const
83 {
84     cout << "Center = [" << x << ", " << y << "],\n";
85     << "Radius = " << radius;
86 }
87 // end function print
```

25  
Outline  
circle.cpp (4 of 4)

© 2003 Prentice Hall, Inc.  
All rights reserved.

Figure 9: Circle class contains an xy-coordinate pair and a radius. (part 2 of 2)

```

1 // Fig. 9.9: circletest.cpp
2 // Testing class Circle.
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 "circle.h" // Circle class definition
14
15 int main()
16 {
17     Circle circle( 37, 43, 2.5 ); // instantiate Circle 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

```

Create Circle object.

26

Outline

circletest.cpp  
(1 of 2)

© 2003 Prentice Hall, Inc.  
All rights reserved.

```

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 point value
29     cout << "\n\nThe new location and
30     circle.print();
31
32     // display floating-point values with
33     cout << fixed << setprecision( 2 )
34
35     // display Circle's diameter
36     cout << "\nDiameter is " << circle.getDiameter();
37
38     // display Circle's circumference
39     cout << "\nCircumference is " << circle.getCircumference();
40
41     // display Circle's area
42     cout << "\nArea is " << circle.getArea();
43
44     cout << endl;
45
46     return 0; // indicates successful termination
47
48 } // end main

```

Use set functions to modify private data.

Invoke public function print to display new coordinates.

27

Outline

circletest.cpp  
(2 of 2)

© 2003 Prentice Hall, Inc.  
All rights reserved.

Figure 10: Circle class test program. (part 1 of 2)

## 1.4.2 Point/Circle Hierarchy using Inheritance

```
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](#)

28

circletest.cpp  
output (1 of 1)

© 2003 Prentice Hall, Inc.  
All rights reserved.

```
1 // Fig. 9.10: circle2.h
2 // Circle2 class contains x-y coordinate pair and radius.
3 #ifndef CIRCLE2_H
4 #define CIRCLE2_H
5
6 #include "point.h" // Point class
7
8 class Circle2 : public Point {
9
10 public:
11
12     // default constructor
13     Circle2( int = 0, int = 0, double = 0, double = 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; // print circle's data
23
24 private:
25     double radius; // Circle2's radius
```

Class **Circle2** inherits from class **Point**.

Keyword **public** indicates type of inheritance.

Maintain **private** data member **radius**.



[Outline](#)

29

circle2.h (1 of 2)

© 2003 Prentice Hall, Inc.  
All rights reserved.

Figure 11: **Circle** class test program. (part 2 of 2) and **Circle2** class header file. (part 1 of 2)

```


26
27 }; // end class Circle2
28
29 #endif


1 // Fig. 9.11: circle2.cpp
2 // Circle2 class member-function definitions.
3 #include <iostream>
4
5 using std::cout;
6
7 #include "circle2.h" // Circle2 class definition
8
9 // default constructor
10 Circle2::Circle2( int xValue, int yValue, double radiusValue )
11 {
12     x = xValue;
13     y = yValue;
14     setRadius( radiusValue );
15
16 } // end Circle2 constructor
17

```

Attempting to access base class Point's private data members **x** and **y** results in syntax errors.

30

 Outline

 circle2.h (2 of 2)

circle2.cpp (1 of 3)


© 2003 Prentice Hall, Inc.  
All rights reserved.


```

18 // set radius
19 void Circle2::setRadius( double radiusValue )
20 {
21     radius = ( radiusValue < 0.0 ? 0.0 : radiusValue );
22
23 } // end function setRadius
24
25 // return radius
26 double Circle2::getRadius() const
27 {
28     return radius;
29
30 } // end function getRadius
31
32 // calculate and return diameter
33 double Circle2::getDiameter() const
34 {
35     return 2 * radius;
36
37 } // end function getDiameter
38

```

31

 Outline

 circle2.cpp (2 of 3)

© 2003 Prentice Hall, Inc.  
All rights reserved.

Figure 12: **Circle2** class header file (part 2 of 2) and Private base-class data can not be accessed from derived class. (part 1 of 2)



```

39 // calculate and return circumference
40 double Circle2::getCircumference() const
41 {
42     return 3.14159 * getDiameter();
43 }
44 // end function getCircumference
45
46 // calculate and return area
47 double Circle2::getArea() const
48 {
49     return 3.14159 * radius * radius;
50 }
51 // end function getArea
52
53 // output Circle2 object
54 void Circle2::print() const
55 {
56     cout << "Center = [" << x << ", " << y << "]\n";
57     cout << "Radius = " << radius;
58 }
59 // end function print

```

Attempting to access base class **Point**'s **private** data members **x** and **y** results in syntax errors.

```

C:\cpphtp4\examples\ch09\CircleTest\circle2.cpp(12) : error C2248: 'x' :
cannot access private member declared in class 'Point'
C:\cpphtp4\examples\ch09\circleTest\point.h(20) :
see declaration of 'x'

C:\cpphtp4\examples\ch09\CircleTest\circle2.cpp(13) : error C2248: 'y' :
cannot access private member declared in class 'Point'
C:\cpphtp4\examples\ch09\circleTest\point.h(21) :
see declaration of 'y'

C:\cpphtp4\examples\ch09\CircleTest\circle2.cpp(56) : error C2248: 'x' :
cannot access private member declared in class 'Point'
C:\cpphtp4\examples\ch09\circleTest\point.h(20) :
see declaration of 'x'

C:\cpphtp4\examples\ch09\CircleTest\circle2.cpp(56) : error C2248: 'y' :
cannot access private member declared in class 'Point'
C:\cpphtp4\examples\ch09\circleTest\point.h(21) :
see declaration of 'y'

```

Attempting to access base class **Point**'s **private** data members **x** and **y** results in syntax errors.

Figure 13: Private base-class data can not be accessed from derived class. (part 2 of 2)



### 1.4.3 Point/Circle Hierarchy using protected data

```
1 // Fig. 9.12: point2.h
2 // Point2 class definition represents an x-y coordinate pair.
3 #ifndef POINT2_H
4 #define POINT2_H
5
6 class Point2 {
7
8 public:
9     Point2( 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; // return y from coordinate pair
16
17     void print() const; // print coordinate pair
18
19 protected:
20     int x; // x part of coordinate pair
21     int y; // y part of coordinate pair
22
23 }; // end class Point2
24
25 #endif
```

Maintain x- and y-coordinates as protected data, accessible to derived classes.

34

Outline

point2.h (1 of 1)

© 2003 Prentice Hall, Inc.  
All rights reserved.

```
1 // Fig. 9.13: point2.cpp
2 // Point2 class member-function definitions.
3 #include <iostream>
4
5 using std::cout;
6
7 #include "point2.h" // Point2 class definition
8
9 // default constructor
10 Point2::Point2( int xValue, int yValue )
11 {
12     x = xValue;
13     y = yValue;
14
15 } // end Point2 constructor
16
17 // set x in coordinate pair
18 void Point2::setX( int xValue )
19 {
20     x = xValue; // no need for validation
21
22 } // end function setX
23
```

35

Outline

point2.cpp (1 of 3)

© 2003 Prentice Hall, Inc.  
All rights reserved.

Figure 14: Point2 class header file.

```

24 // return x from coordinate pair
25 int Point2::getX() const
26 {
27     return x;
28 }
29 // end function getX
30
31 // set y in coordinate pair
32 void Point2::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 Point2::getY() const
40 {
41     return y;
42 }
43 // end function getY
44

```



[Outline](#)

36



point2.cpp (2 of 3)

© 2003 Prentice Hall, Inc.  
All rights reserved.

```

45 // output Point2 object
46 void Point2::print() const
47 {
48     cout << '[' << x << ", " << y << ']'<< endl;
49 }
50 // end function print

```



[Outline](#)

37



point2.cpp (3 of 3)

© 2003 Prentice Hall, Inc.  
All rights reserved.

Figure 15: **Point2** class represents an xy-coordinate pair as **protected** data.

```
1 // Fig. 9.14: circle3.h
2 // Circle3 class contains x-y coordinate pair and radius.
3 #ifndef CIRCLE3_H
4 #define CIRCLE3_H
5
6 #include "point2.h" // Point2
7
8 class Circle3 : public Point2 {
9
10 public:
11
12 // default constructor
13 Circle3( 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; // Circle3's radius
```

Class Circle3 inherits from class Point2.

Maintain private data member radius.



Outline

38

circle3.h (1 of 2)

© 2003 Prentice Hall, Inc.  
All rights reserved.

```
26
27 }; // end class Circle3
28
29 #endif
```



Outline

39

circle3.h (2 of 2)

© 2003 Prentice Hall, Inc.  
All rights reserved.

Figure 16: Circle3 class header file.

```

1 // Fig. 9.15: circle3.cpp
2 // Circle3 class member-function definitions.
3 #include <iostream>
4
5 using std::cout;
6
7 #include "circle3.h" // Circle3 class definition
8
9 // default constructor
10 Circle3::Circle3( int xValue, int yValue, double radiusValue )
11 {
12     x = xValue;
13     y = yValue;
14     setRadius( radiusValue );
15
16 } // end Circle3 constructor
17
18 // set radius
19 void Circle3::setRadius( double radiusValue )
20 {
21     radius = ( radiusValue < 0.0 ? 0.0 : radiusValue );
22
23 } // end function setRadius
24

```

Constructor first implicitly calls base class's default constructor.  
protected in base class Point2.

```

25 // return radius
26 double Circle3::getRadius() const
27 {
28     return radius;
29 } // end function getRadius
30
31 // calculate and return diameter
32 double Circle3::getDiameter() const
33 {
34     return 2 * radius;
35 } // end function getDiameter
36
37 // calculate and return circumference
38 double Circle3::getCircumference() const
39 {
40     return 3.14159 * getDiameter();
41 } // end function getCircumference
42
43
44
45



```

Figure 17: **Circle3** class that inherits from class **Point2**.

```

46 // calculate and return area
47 double Circle3::getArea() const
48 {
49     return 3.14159 * radius * radius;
50 }
51 // end function getArea
52
53 // output Circle3 object
54 void Circle3::print() const
55 {
56     cout << "Center = [" << x << ", " << y << " ]'
57         << "; Radius = " << radius;
58 }
59 // end function print

```

 Outline 42  
 circle3.cpp (3 of 3)



Access inherited data members **x** and **y**, declared **protected** in base class **Point2**.

© 2003 Prentice Hall, Inc.  
All rights reserved.

```

1 // Fig. 9.16: circletest3.cpp
2 // Testing class Circle3.
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 "circle3.h" // Circle3 class def.
14
15 int main()
16 {
17     Circle3 circle( 37, 43, 2.5 ); // instantiate Circle3 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 43  
 circletest3.cpp (1 of 2)

Create **Circle3** object.

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

Figure 18: Protected base-class data can be accessed from derived class. (part 1 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 point 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 Circle3's diameter
36 cout << "\nDiameter is " << circle.getDiameter();
37
38 // display Circle3's circumference
39 cout << "\nCircumference is " << circle.getCircumference();
40
41 // display Circle3'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 44

circletest3.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 45

circletest3.cpp output (1 of 1)

© 2003 Prentice Hall, Inc.  
All rights reserved.

Figure 19: Protected base-class data can be accessed from derived class. (part 2 of 2)

- Using protected data members
  - Advantages
    - \* Derived classes can modify values directly
    - \* Slight increase in performance; avoid set/get function call overhead
  - Disadvantages
    - \* No validity checking; derived class can assign illegal value
    - \* Implementation dependent
      - Derived class member functions more likely dependent on base class implementation
      - Base class implementation changes may result in derived class modifications; fragile (brittle) software