IS 0020
Program Design and Software Tools

Inheritance

Lecture 8

March 2, 2004

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

Introduction

- 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

Introduction

- 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

Introduction

- 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

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

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`

Base Classes and Derived Classes

- Inheritance hierarchy
  - 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

Fig. 9.2 Inheritance hierarchy for university **`CommunityMember`**s.

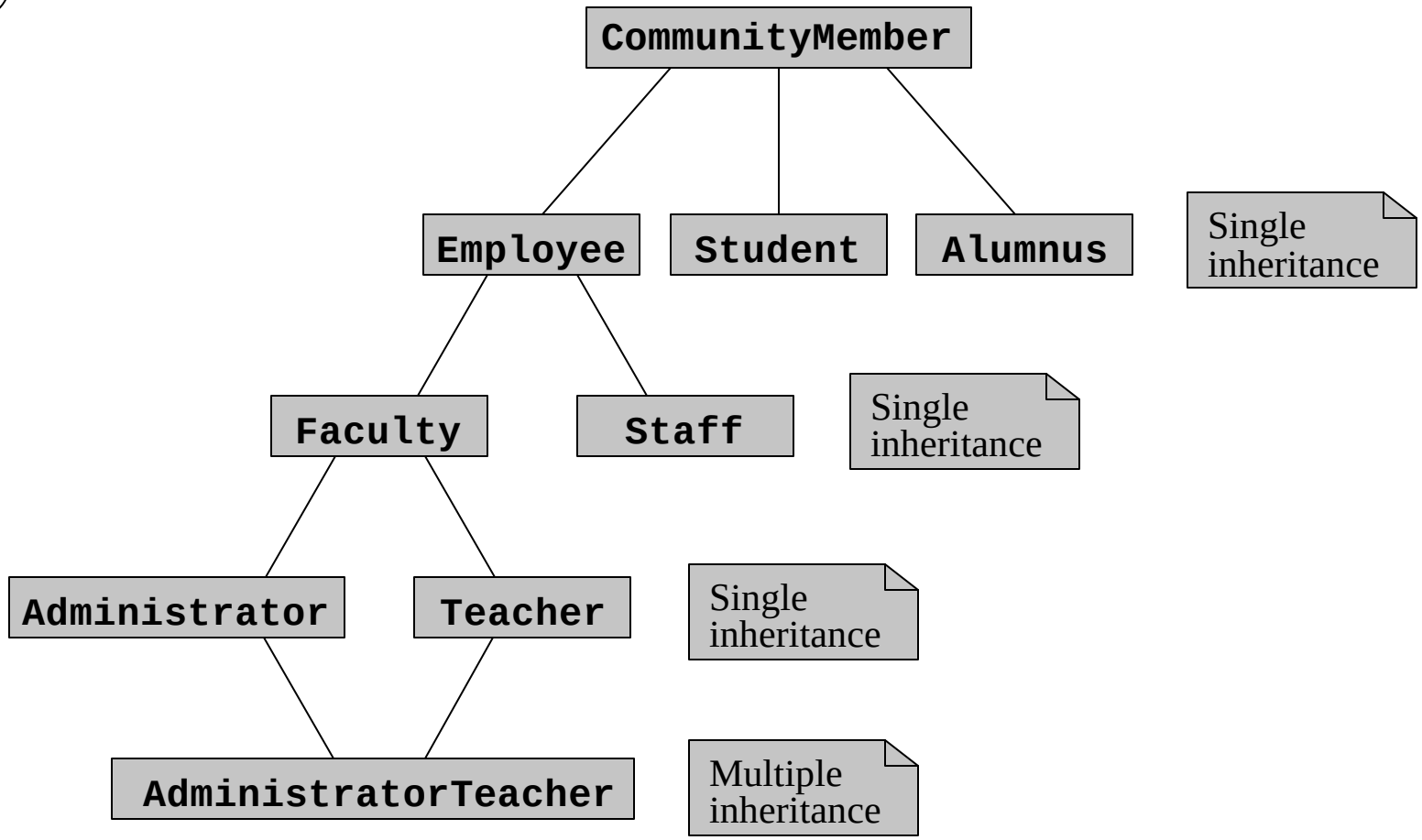
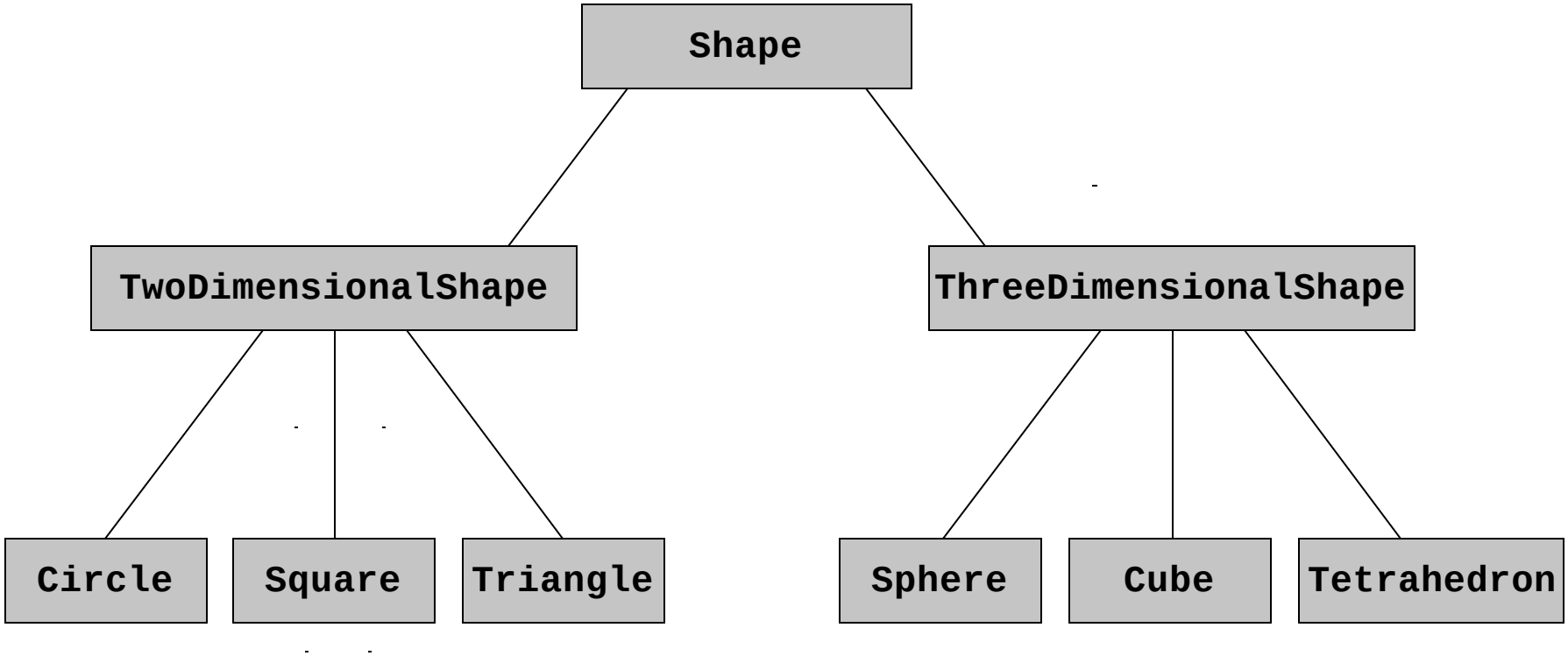


Fig. 9.3 Inheritance hierarchy for Shapes.



Base Classes and Derived Classes

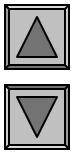
- **`public`** inheritance
  - Specify with:
    `Class TwoDimensionalShape : public Shape`
    - Class **`TwoDimensionalShape`** inherits from class **`Shape`**
  - Base class **`private`** members
    - Not accessible directly
    - Still inherited
      - Manipulate through inherited member functions
  - Base class **`public`** and **`protected`** members
    - Inherited with original member access
  - **`friend`** functions
    - Not inherited

`protected` Members

- **`protected`** access
  - Intermediate level of protection between **`public`** and **`private`**
  - **`protected`** members accessible to
    - Base class members
    - Base class **`friend`**s
    - Derived class members
    - Derived class **`friend`**s
  - Derived-class members
    - Refer to **`public`** and **`protected`** members of base class
      - Simply use member names

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

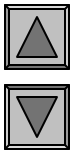


Outline

point.h (1 of 1)

```
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
```

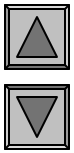
Maintain **x**- and **y**-coordinates as **private** data members.



Outline

point.cpp (1 of 3)

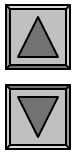
```
1  // Fig. 9.5: point.cpp
2  // Point class member-function definitions.
3  #include <iostream>
4
5  using std::cout;
6
7  #include "point.h"   // Point class definition
8
9  // default constructor
10 Point::Point( int xValue, int yValue )
11 {
12    x = xValue;
13    y = yValue;
14
15 } // end Point 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

point.cpp (2 of 3)

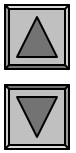
```
24 // return x from coordinate pair
25 int Point::getX() const
26 {
27    return x;
28
29 } // end function getX
30
31 // set y in coordinate pair
32 void Point::setY( int yValue )
33 {
34    y = yValue; // no need for validation
35
36 } // end function setY
37
38 // return y from coordinate pair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



Outline

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

point.cpp (3 of 3)



Outline

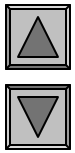
pointtest.cpp
(1 of 2)

```
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()
16         << "\nY coordinate is " << point.g
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
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.



Outline

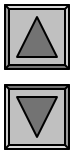
```
26     return 0;  // indicates successful termination
27
28  } // end main
```

```
X coordinate is 72
Y coordinate is 115

The new location of point is [10, 10]
```

pointtest.cpp
(2 of 2)

pointtest.cpp
output (1 of 1)

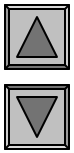


Outline

circle.h (1 of 2)

```
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 = 0.0 );
12
13     void setX( int );           // set x in coordinate pair
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.



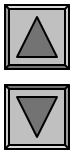
Outline

circle.h (2 of 2)

```
26    void print() const;          // output Circle object
27
28 private:
29    int x;            // x-coordina
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.

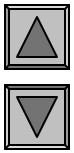
Note code similar to **Point** code.



Outline

circle.cpp (1 of 4)

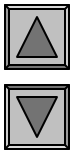
```
1  // Fig. 9.8: circle.cpp
2  // Circle class member-function definitions.
3  #include <iostream>
4
5  using std::cout;
6
7  #include "circle.h"   // Circle class definition
8
9  // default constructor
10 Circle::Circle( int xValue, int yValue, double radiusValue )
11 {
12    x = xValue;
13    y = 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

circle.cpp (2 of 4)

```
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
```

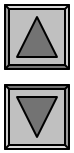


Outline

circle.cpp (3 of 4)

```
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
```

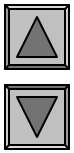
Ensure non-negative value for **`radius`**.



Outline

circle.cpp (4 of 4)

```
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 << ']'
85         << "; Radius = " << radius;
86
87 } // end function print
```

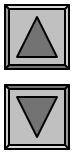


Outline

circletest.cpp
(1 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 defin
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.



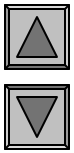
Outline

circletest.cpp
(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 point value
29    cout << "\n\nThe new location and
30    circle.print();
31
32    // display floating-point values w
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.

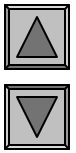


Outline

circletest.cpp
output (1 of 1)

```
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

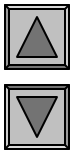
circle2.h (1 of 2)

```
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, dou
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;  // Circle2's radius
```

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

Keyword **public** indicates type of inheritance.

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

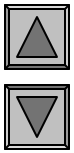


Outline


```
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, i                    e )
11 {
12    x = xValue;
13    y = yValue;
14    setRadius( radiusValue );
15
16 } // end Circle2 constructor
17
```

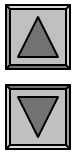
Attempting to access base class **`Point`**'s **`private`** data members **`x`** and **`y`** results in syntax errors.



Outline

circle2.cpp (2 of 3)

```
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
```

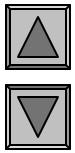


Outline

circle2.cpp (3 of 3)

```
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 << ']'
57         << "; Radius = " << radius;
58
59 } // end function print
```

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



Outline

circle2.cpp
output (1 of 1)

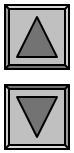
```
C:\cpphtp4\examples\ch09\CircleTest\circle2.cpp(12) : error C2248: 'x' :
cannot access private member declared in class 'Point'
        C:\cpphtp4\examples\ch09\circletest\point.h(20) :
        see declaration of 'x'

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.

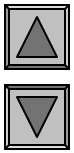


Outline

point2.h (1 of 1)

```
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;  // c
18
19 protected:
20    int x;  // x part of coor
21    int y;  // y part of coordinate pair
22
23 }; // end class Point2
24
25 #endif
```

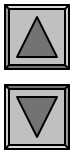
Maintain **x**- and **y**-coordinates as **protected** data, accessible to derived classes.



Outline

point2.cpp (1 of 3)

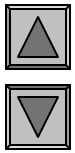
```
1  // Fig. 9.13: point2.cpp
2  // Point2 class member-function definitions.
3  #include <iostream>
4
5  using std::cout;
6
7  #include "point2.h"   // Point2 class definition
8
9  // default constructor
10 Point2::Point2( int xValue, int yValue )
11 {
12    x = xValue;
13    y = yValue;
14
15 } // 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
```



Outline

point2.cpp (2 of 3)

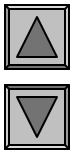
```
24 // return x from coordinate pair
25 int Point2::getX() const
26 {
27    return x;
28
29 } // end function getX
30
31 // set y in coordinate pair
32 void Point2::setY( int yValue )
33 {
34    y = yValue; // no need for validation
35
36 } // end function setY
37
38 // return y from coordinate pair
39 int Point2::getY() const
40 {
41    return y;
42
43 } // end function getY
44
```



Outline

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

point2.cpp (3 of 3)



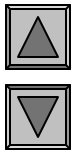
Outline

circle3.h (1 of 2)

```
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 c
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`**.

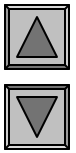
Maintain **`private`** data member **`radius`**.



Outline

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

circle3.h (2 of 2)



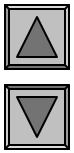
Outline

circle3.cpp (1 of 3)

```
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,                    )
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.

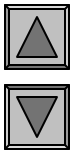
protected in base class **Point2**.



Outline

circle3.cpp (2 of 3)

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

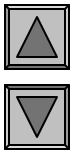


Outline

circle3.cpp (3 of 3)

```
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
```

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



Outline

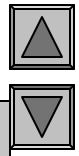
circletest3.cpp
(1 of 2)

```
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 objec
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 **`Circle3`** object.

Use inherited get functions to access inherited **`protected`** data.

Use **`Circle3`** get function to access **`private`** data **`radius`**.



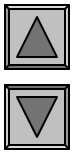
Outline

```
24    circle.setX( 2 );          // set new x-coordinate
25    circle.setY( 2 );          // set new y-coordinate
26    circle.setRadius( 4.25 );  // set new ra
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`**.

circletest3.cpp
(2 of 2)



Outline

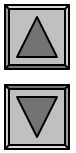
circletest3.cpp
output (1 of 1)

```
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
```

Relationship between Base Classes and Derived Classes

- 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

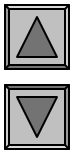


Outline

point3.h (1 of 1)

```
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;    // return y from coordinate pair
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.

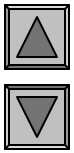


Outline

point3.cpp (1 of 3)

```
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
15 } // end Point3 constructor
16
17 // set x in coordinate pair
18 void Point3::setX( int xValue )
19 {
20    x = xValue; // no need for validation
21
22 } // end function setX
23
```

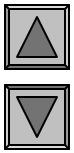
Member initializers specify values of **x** and **y**.



Outline

point3.cpp (2 of 3)

```
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
```

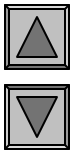


Outline

point3.cpp (3 of 3)

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

Invoke non-**`private`** member functions to access **`private`** data.



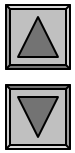
Outline

circle4.h (1 of 2)

```
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`**.

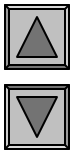
Maintain **`private`** data member **`radius`**.



Outline

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

circle4.h (2 of 2)

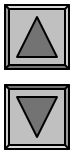


Outline

circle4.cpp (1 of 3)

```
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`**.

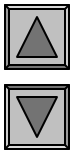


Outline

circle4.cpp (2 of 3)

```
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`**.



Outline

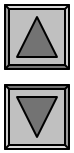
circle4.cpp (3 of 3)

```
45 // calculate and return area
46 double Circle4::getArea() const
47 {
48    return 3.14159 * getRadius() * getRadius();
49
50 } // end function getArea
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
```

Redefine class **Point3**'s member function **print**.

Invoke function **getRadius** ng

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



Outline

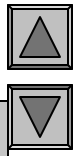
circletest4.cpp
(1 of 2)

```
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 def
14
15 int main()
16 {
17    Circle4 circle( 37, 43, 2.5 ); // instantiate Circle4 objec
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 **`Circle4`** object.

Use inherited get functions to access inherited **`protected`** data.

Use **`Circle3`** get function to access **`private`** data **`radius`**.



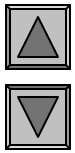
Outline

```
24    circle.setX( 2 );          // set new x-coordinate
25    circle.setY( 2 );          // set new y-coordinate
26    circle.setRadius( 4.25 );  // set new ra
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`**.

circletest4.cpp
(2 of 2)



Outline

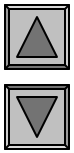
```
X coordinate is 37
Y coordinate is 43
Radius is 2.5

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
```

circletest4.cpp
output (1 of 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



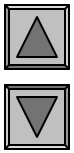
Outline

cylinder.h (1 of 2)

```
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 c
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;        // o
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**.



Outline

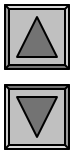
```
26
27  #endif
```

cylinder.h (2 of 2)

cylinder.cpp
(1 of 3)

```
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`**.



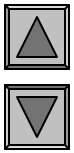
Outline

cylinder.cpp
(2 of 3)

```
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
```

Redefine base class
er function
area.

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



Outline

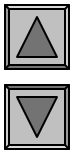
cylinder.cpp
(3 of 3)

```
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
```

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

Redefine class **`Circle4`**'s **`int`**.

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



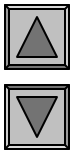
Outline

cylindertest.cpp
(1 of 3)

```
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
```

Invoke indirectly inherited **`Point3`** member functions

Invoke **`Cylinder`** member function.



Outline

```
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
```

Invoke indirectly inherited **Point3** member functions.

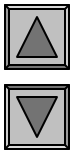
Invoke directly inherited

Invoke **Cylinder** member function.

Invoke redefined **print** function.

Invoke redefined **getArea** function.

cylindertest.cpp
(2 of 3)



Outline

```
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
```

cylindertest.cpp
(3 of 3)

cylindertest.cpp
output (1 of 1)

Constructors and Destructors in Derived Classes

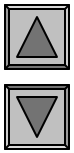
- Instantiating derived-class object
  - Chain of constructor calls
    - Derived-class constructor invokes base class constructor
      - Implicitly or explicitly
    - Base of inheritance hierarchy
      - Last constructor called in chain
      - First constructor body to finish executing
      - Example: **`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

Constructors and Destructors in Derived Classes

- 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

Constructors and Destructors in Derived Classes

- Base-class constructors, destructors, assignment operators
  - Not inherited by derived classes
  - Derived class constructors, assignment operators can call
    - Constructors
    - Assignment operators

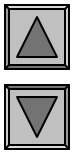


Outline

point4.h (1 of 1)

```
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
```

Constructor and destructor output messages to demonstrate function call order.



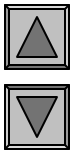
Outline

point4.cpp (1 of 3)

```
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;
```

Output message to demonstrate constructor function call order.

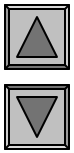
Output message to demonstrate destructor function call order.



Outline

point4.cpp (2 of 3)

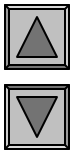
```
26
27 } // end Point4 destructor
28
29 // set x in coordinate pair
30 void Point4::setX( int xValue )
31 {
32    x = xValue; // no need for validation
33
34 } // end function setX
35
36 // return x from coordinate pair
37 int Point4::getX() const
38 {
39    return x;
40
41 } // end function getX
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

point4.cpp (3 of 3)

```
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() << ']';
61
62 } // end function print
```

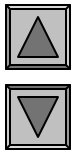


Outline

circle5.h (1 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
```

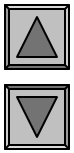
Constructor and destructor output messages to demonstrate function call order.



Outline

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

circle5.h (2 of 2)

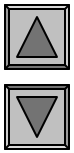


Outline

circle5.cpp (1 of 4)

```
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
20 } // end Circle5 constructor
21
```

Output message to demonstrate constructor function call order.

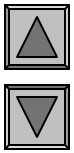


Outline

circle5.cpp (2 of 4)

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

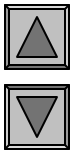
Output message to demonstrate destructor function call order.



Outline

circle5.cpp (3 of 4)

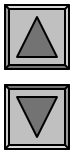
```
45 // calculate and return diameter
46 double Circle5::getDiameter() const
47 {
48    return 2 * getRadius();
49
50 } // end function getDiameter
51
52 // calculate and return circumference
53 double Circle5::getCircumference() const
54 {
55    return 3.14159 * getDiameter();
56
57 } // end function 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

circle5.cpp (4 of 4)

```
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
```



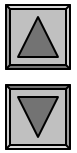
Outline

fig09_29.cpp
(1 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.



Outline

```
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

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

Circle5 destructor: Center = [5, 5]; Radius =
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.

fig09_29.cpp
(2 of 2)

fig09_29.cpp
output (1 of 1)

`public`, `protected` and `private` Inheritance

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 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.	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.

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