Chapter 9 - Object-Oriented Programming: Inheritance

Outline

9.1 Introduction

- Inheritance
  - Software reusability
  - Create new class from existing class
    - Absorb existing class's data and behaviors
    - Enhance with new capabilities
  - Subclass extends superclass
    - Subclass
      - More specialized group of objects
      - Behaviors inherited from superclass
        - Can customize
      - Additional behaviors

9.1 Introduction

- Class hierarchy
  - Direct superclass
    - Inherited explicitly (one level up hierarchy)
  - Indirect superclass
    - Inherited two or more levels up hierarchy
  - Single inheritance
    - Inherits from one superclass
  - Multiple inheritance
    - Inherits from multiple superclasses
      - Java does not support multiple inheritance

9.1 Introduction

- Abstraction
  - Focus on commonalities among objects in system
- “is-a” vs. “has-a”
  - “is-a”
    - Inheritance
    - subclass object treated as superclass 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

9.2 Superclasses and Subclasses

- Superclasses and subclasses
  - Object of one class "is an" object of another class
    - Example: Rectangle is quadrilateral.
      - Class `Rectangle` inherits from class `Quadrilateral`
      - `Quadrilateral`: superclass
      - `Rectangle`: subclass
  - Superclass typically represents larger set of objects than subclasses
    - Example:
      - superclass: `Vehicle`
        - Cars, trucks, boats, bicycles, …
      - subclass: `Car`
        - Smaller, more-specific subset of vehicles

9.2 Superclasses and Subclasses (Cont.)

- Inheritance examples

Superclass	Subclasses
`Student`	`GraduateStudent`, `UndergraduateStudent`
`Shape`	`Circle`, `Triangle`, `Rectangle`
`Loan`	`CarLoan`, `HomeImprovementLoan`, `MortgageLoan`
`Employee`	`Faculty`, `Staff`
`BankAccount`	`CheckingAccount`, `SavingsAccount`

Fig. 9.1 Inheritance examples.

9.2 Superclasses and Subclasses (Cont.)

- Inheritance hierarchy
  - Inheritance relationships: tree-like hierarchy structure
  - Each class becomes
    - superclass
      - Supply data/behaviors to other classes

    OR
    - subclass
      - Inherit data/behaviors from other classes

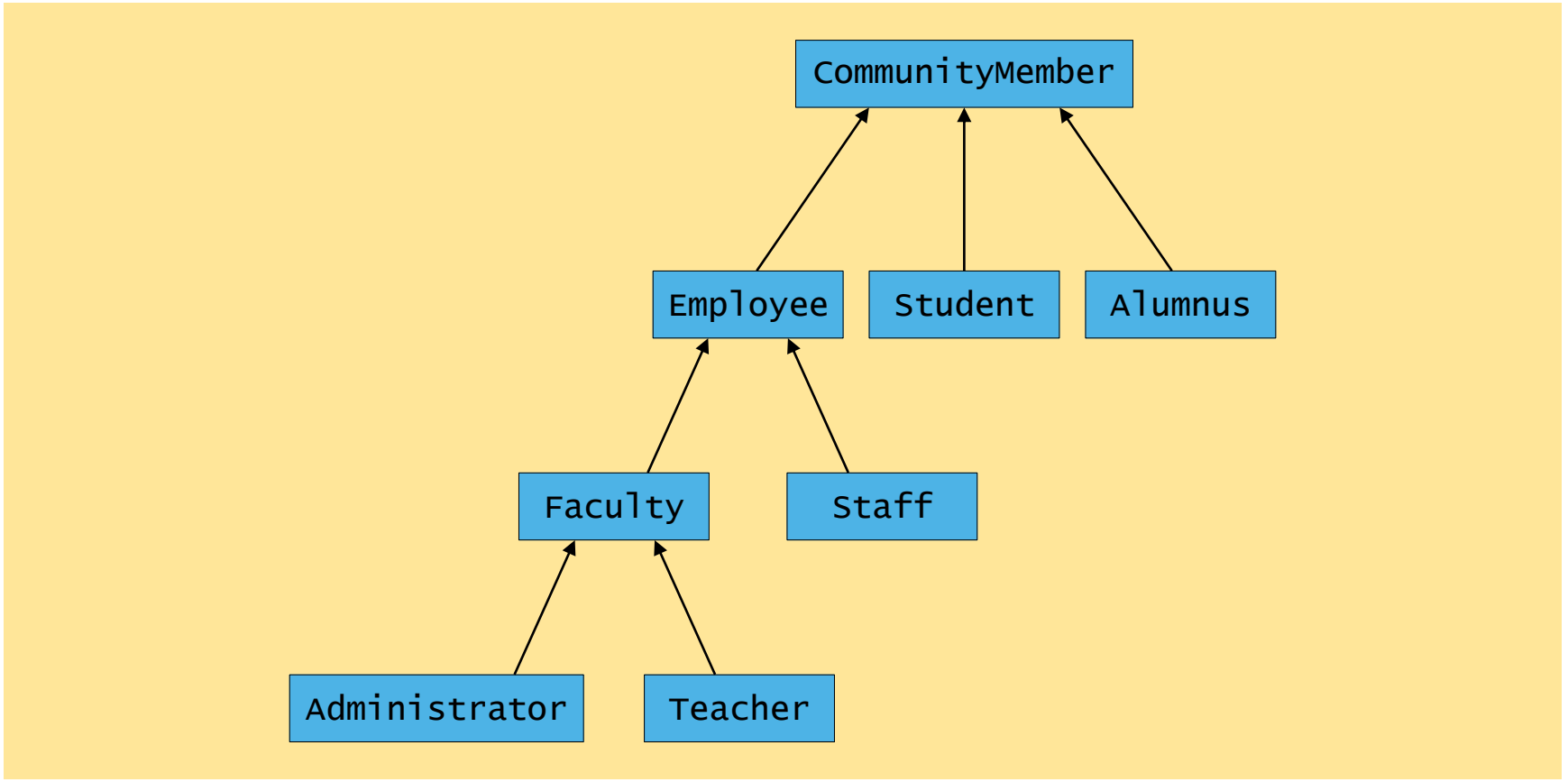


Fig. 9.2 Inheritance hierarchy for university `CommunityMember`s.

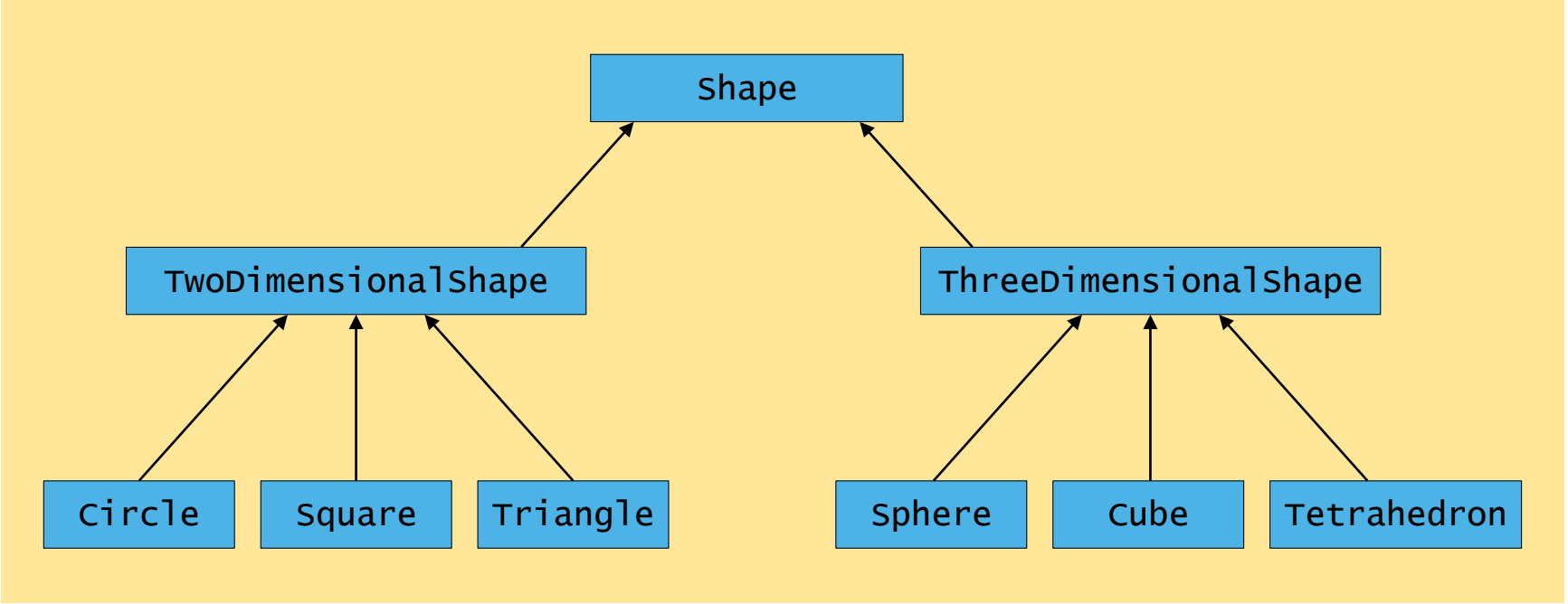


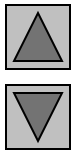
Fig. 9.3 Inheritance hierarchy for Shapes.

9.3 `protected` Members

- `protected` access
  - Intermediate level of protection between `public` and `private`
  - `protected` members accessible to
    - superclass members
    - subclass members
    - Class members in the same package
  - Subclass access superclass member
    - Keyword `super` and a dot (.)

9.4 Relationship between Superclasses and Subclasses

- Superclass and subclass relationship
  - Example: Point/circle inheritance hierarchy
    - Point
      - x-y coordinate pair
    - Circle
      - x-y coordinate pair
      - Radius



Outline

```
1  // Fig. 9.4: Point.java
2  // Point class declaration represe
3
4  public class Point {
5     private int x; // x part of coo
6     private int y; // y part of coordinate pair
7
8     // no-argument constructor
9     public Point()
10    {
11       // implicit call to Object constructor occurs here
12    }
13
14    // constructor
15    public Point( int xValue, int yValue )
16    {
17       // implicit call to Object constructor occurs here
18       x = xValue;  // no need for validation
19       y = yValue;  // no need for validation
20    }
21
22    // set x in coordinate pair
23    public void setX( int xValue )
24    {
25       x = xValue;  // no need for validation
26    }
27
```

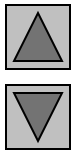
Maintain x- and y-coordinates as `private` instance variables.

Implicit call to Object constructor

Point.java

Lines 5-6
Maintain x- and y-coordinates as private instance variables.

Line 11
Implicit call to Object constructor



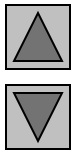
Outline

Point.java

Lines 47-50
Override method `toString` of class `Object`.

```
28    // return x from coordinate pair
29    public int getX()
30    {
31       return x;
32    }
33
34    // set y in coordinate pair
35    public void setY( int yValue )
36    {
37       y = yValue;  // no need for validation
38    }
39
40    // return y from coordinate pair
41    public int getY()
42    {
43       return y;
44    }
45
46    // return String representation of Point object
47    public String toString()
48    {
49       return "[" + x + ", " + y + "]";
50    }
51
52 } // end class Point
```

Override method `toString` of class `Object`



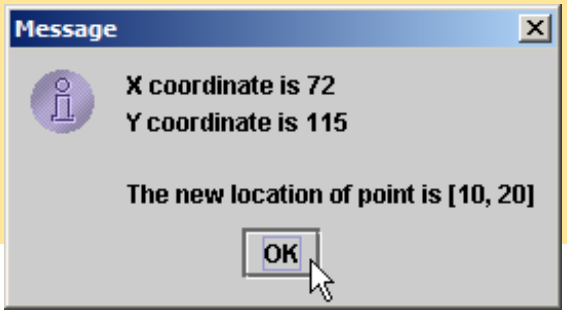
Outline

```
1  // Fig. 9.5: PointTest.java
2  // Testing class Point.
3  import javax.swing.JOptionPane;
4
5  public class PointTest {
6
7     public static void main( String[] args )
8     {
9        Point point = new Point( 72, 115 ); // create Point object
10
11       // get point coordinates
12       String output = "X coordinate is "
13          "\nY coordinate is " + point.get
14
15       point.setX( 10 );  // set x-coordinate
16       point.setY( 20 );  // set y-coordinate
17
18       // get String representation of new point value
19       output += "\n\nThe new location of point is " + point;
20
21       JOptionPane.showMessageDialog( null, output ); // display output
22
23       System.exit( 0 );
24
25    } // end main
26
27 } // end class PointTest
```

Instantiate `Point` object

Change the value of point's x- and y- coordinates

Implicitly call point's `toString` method

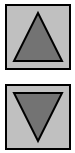


`PointTest.java`

Line 9
Instantiate `Point` object

Lines 15-16
Change the value of point's x- and y-

Implicitly call point's `toString` method



Outline

```
1  // Fig. 9.6: Circle.java
2  // Circle class contains x-y coordinate pair and radius.
3
4  public class Circle {
5     private int x;          // x-coo
6     private int y;          // y-coo
7     private double radius;  // Circle's radius
8
9     // no-argument constructor
10    public Circle()
11    {
12       // implicit call to Object constructor occurs here
13    }
14
15    // constructor
16    public Circle( int xValue, int yValue, double radiusValue )
17    {
18       // implicit call to Object constructor occurs here
19       x = xValue;  // no need for validation
20       y = yValue;  // no need for validation
21       setRadius( radiusValue );
22    }
23
24    // set x in coordinate pair
25    public void setX( int xValue )
26    {
27       x = xValue;  // no need for validation
28    }
29
```

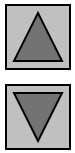
Maintain x-y coordinates and `radius` as `private` instance variables.

Note code similar to `Point` code.

`Circle.java`

Lines 5-7
Maintain x- and y-coordinates and radius as `private` instance variables.

Lines 25-28
Note code similar to `Point` code.



Outline

rcle.java

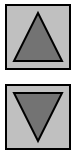
Lines 31-47
Note code similar to `Point` code.

Line 51
Ensure non-negative value for `radius`.

```
30    // return x from coordinate pair
31    public int getX()
32    {
33       return x;
34    }
35
36    // set y in coordinate pair
37    public void setY( int yValue )
38    {
39       y = yValue;  // no need for validation
40    }
41
42    // return y from coordinate pair
43    public int getY()
44    {
45       return y;
46    }
47
48    // set radius
49    public void setRadius( double radiusValue )
50    {
51       radius = ( radiusValue < 0.0 ? 0.0 : radiusValue );
52    }
53
54    // return radius
55    public double getRadius()
56    {
57       return radius;
58    }
59
```

Note code similar to `Point` code.

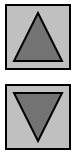
Ensure non-negative value for `radius`.



Outline

Circle.java

```
60    // calculate and return diameter
61    public double getDiameter()
62    {
63       return 2 * radius;
64    }
65
66    // calculate and return circumference
67    public double getCircumference()
68    {
69       return Math.PI * getDiameter();
70    }
71
72    // calculate and return area
73    public double getArea()
74    {
75       return Math.PI * radius * radius;
76    }
77
78    // return String representation of Circle object
79    public String toString()
80    {
81       return "Center = [" + x + ", " + y + "]; Radius = " + radius;
82    }
83
84 } // end class Circle
```



Outline

```
1  // Fig. 9.7: CircleTest.java
2  // Testing class Circle.
3  import java.text.DecimalFormat;
4  import javax.swing.JOptionPane;
5
6  public class CircleTest {
7
8     public static void main( String[] args )
9     {
10       Circle circle = new Circle( 37, 43, 2.5 ); // create Circle object
11
12       // get Circle's initial x-y coordinates and radius
13       String output = "X coordinate is " + circle.getX() +
14          "\nY coordinate is " + circle.getY() +
15          "\nRadius is " + circle.getRadius();
16
17       circle.setX( 35 );          // set new x-coordinate
18       circle.setY( 20 );          // set new y-coordinate
19       circle.setRadius( 4.25 );   // set
20
21       // get String representation of ne
22       output += "\n\nThe new location and radi
23          circle.toString();
24
25       // format floating-point values with 2 digits of precision
26       DecimalFormat twoDigits = new DecimalFormat( "0.00" );
27
```

Create `Circle` object.

Explicitly call circle's `toString` method

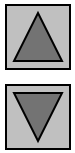
o modify `private` instance variable.

`CircleTest.java`

e 10
Create `Circle` object

Lines 17-19
Use set methods to modify `private` instance variable

Line 23
Explicitly call circle's `toString` method



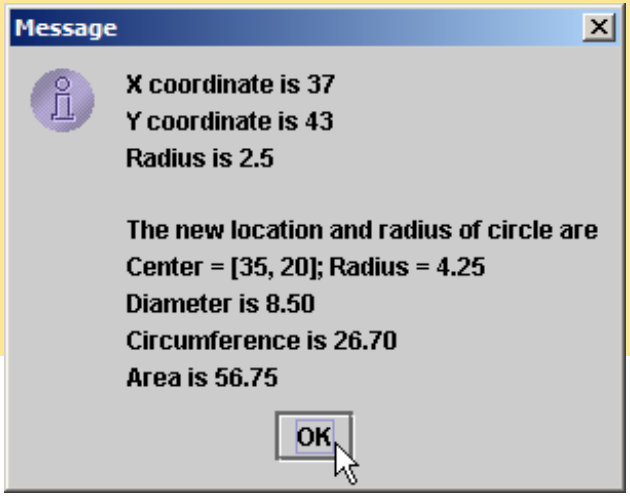
Outline

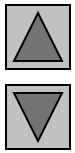
CircleTest.java

```
28        // get Circle's diameter
29        output += "\nDiameter is " +
30           twoDigits.format( circle.getDiameter() );
31
32        // get Circle's circumference
33        output += "\nCircumference is " +
34           twoDigits.format( circle.getCircumference() );
35
36        // get Circle's area
37        output += "\nArea is " + twoDigits.format( circle.getArea() );
38
39        JOptionPane.showMessageDialog( null, output ); // display output
40
41        System.exit( 0 );
42
43     } // end main
44
45  } // end class CircleTest
```

Use get methods to obtain circle's diameter, circumference and area.

Use get methods to obtain circle's diameter, circumference and area.





Outline

```
1  // Fig. 9.8: Circle2.java
2  // Circle2 class inherits from Point.
3
4  public class Circle2 extends Point {
5     private double radius;  // Circle2's radius
6
7     // no-argument constructor
8     public Circle2()
9     {
10       // implicit call to Point constructor occurs here
11    }
12
13    // constructor
14    public Circle2( int xValue, int yVal
15    {
16       // implicit call to Point constru
17       x = xValue;  // not allowed: x pr
18       y = yValue;  // not allowed: y private in Point
19       setRadius( radiusValue );
20    }
21
22    // set radius
23    public void setRadius( double radiusValue )
24    {
25       radius = ( radiusValue < 0.0 ? 0.0 : radiusValue );
26    }
27
```

Class `Circle2` extends class `Point`.

Maintain `private` instance variable `radius`.

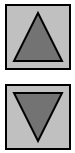
Attempting to access superclass `Point`'s `private` instance variables `x` and `y` results in syntax errors.

`Circle2.java`

Line 4
Class `Circle2` extends class `Point`.

Line 5
Maintain `private` instance variable `radius`.

Lines 17-18
Attempting to access superclass `Point`'s `private` instance variables `x` and `y` results in syntax errors.



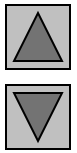
Outline

```
34       // calculate and return diameter
35       public double getDiameter()
36       {
37          return 2 * radius;
38       }
39
40       // calculate and return circumference
41       public double getCircumference()
42       {
43          return Math.PI * getDiameter();
44       }
45
46       // calculate and return area
47       public double getArea()
48       {
49          return Math.PI * radius * radius;
50       }
51
52       // return String representation of Circle object
53       public String toString()
54       {
55          // use of x and y not allowed: x and y private
56          return "Center = [" + x + ", " + y + "]; Radius = " + radius;
57       }
58
59    } // end class Circle2
```

Attempting to access superclass `Point`'s `private` instance variables `x` and `y` results in syntax errors.

`Circle2.java`

Line 56
Attempting to access superclass `Point`'s `private` instance variables `x` and `y` results in syntax errors.



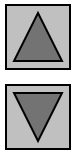
Outline

```
Circle2.java:17: x has private access in Point
      x = xValue;  // not allowed: x private in Point
      ^
Circle2.java:18: y has private access in Point
      y = yValue;  // not allowed: y private in Point
      ^
Circle2.java:56: x has private access in Point
      return "Center = [" + x + ", " + y + "]; Radius = " + radius;
                            ^
Circle2.java:56: y has private access in Point
      return "Center = [" + x + ", " + y + "]; Radius = " + radius;
                                   ^
4 errors
```

Attempting to access superclass `Point`'s `private` instance variables `x` and `y` results in syntax errors.

`Circle2.java` output

Attempting to access superclass `Point`'s `private` instance variables `x` and `y` results in syntax errors.



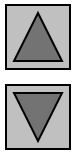
Outline

```
1  // Fig. 9.9: Point2.java
2  // Point2 class declaration represent
3
4  public class Point2 {
5     protected int x;  // x part of coo
6     protected int y;  // y part of coo
7
8     // no-argument constructor
9     public Point2()
10    {
11       // implicit call to Object constructor occurs here
12    }
13
14    // constructor
15    public Point2( int xValue, int yValue )
16    {
17       // implicit call to Object constructor occurs here
18       x = xValue;  // no need for validation
19       y = yValue;  // no need for validation
20    }
21
22    // set x in coordinate pair
23    public void setX( int xValue )
24    {
25       x = xValue;  // no need for validation
26    }
27
```

Maintain x- and y- coordinates as protected instance variables, accessible to subclasses.

Point2.java

Lines 5-6
Maintain x- and y- coordinates as protected instance variables, accessible to subclasses.



Outline

Point2.java

```
28    // return x from coordinate pair
29    public int getX()
30    {
31       return x;
32    }
33
34    // set y in coordinate pair
35    public void setY( int yValue )
36    {
37       y = yValue;  // no need for validation
38    }
39
40    // return y from coordinate pair
41    public int getY()
42    {
43       return y;
44    }
45
46    // return String representation of Point2 object
47    public String toString()
48    {
49       return "[" + x + ", " + y + "]";
50    }
51
52 } // end class Point2
```

```
1  // Fig. 9.10: Circle3.java
2  // Circle3 class inherits from Point2 and ha
3  // protected members x and y.
4
5  public class Circle3 extends Point2 {
6     private double radius; // Circle3's radius
7
8     // no-argument constructor
9     public Circle3()
10    {
11       // implicit call to Point2 constructor occurs here
12    }
13
14    // constructor
15    public Circle3( int xValue, int
16    {
17       // implicit call to Point2 c
18       x = xValue; // no need for
19       y = yValue; // no need for
20       setRadius( radiusValue );
21    }
22
23    // set radius
24    public void setRadius( double radiusValue )
25    {
26       radius = ( radiusValue < 0.0 ? 0.0 : radiusValue );
27    }
28
```

Class `Circle3` inherits from

Maintain `private` instance variables `radius`.

Implicitly calls superclass's default constructor.

Modify inherited instance variables `x` and `y`, declared `protected` in superclass `Point2`.

Outline

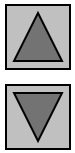
`Circle3.java`

Line 5
Class `Circle3` inherits from class `Point2`.

Line 6
Maintain `private` instance variables `radius`.

Lines 11 and 17
Implicitly call superclass's default constructor.

Lines 18-19
Modify inherited instance variables `x` and `y`, declared `protected` in superclass `Point2`.



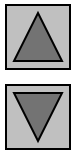
Outline

Circle3.java

Line 56
Access inherited instance variables `x` and `y`, declared `protected` in superclass `Point2`.

```
29      // return radius
30      public double getRadius()
31      {
32         return radius;
33      }
34
35      // calculate and return diameter
36      public double getDiameter()
37      {
38         return 2 * radius;
39      }
40
41      // calculate and return circumference
42      public double getCircumference()
43      {
44         return Math.PI * getDiameter();
45      }
46
47      // calculate and return area
48      public double getArea()
49      {
50         return Math.PI * radius * radius;
51      }
52
53      // return String representation of Circle3 object
54      public String toString()
55      {
56         return "Center = [" + x + ", " + y + "]; Radius = " + radius;
57      }
58
59   } // end class Circle3
```

Access inherited instance variables `x` and `y`, declared `protected` in superclass `Point2`.



Outline

```
1  // Fig. 9.11: CircleTest3.java
2  // Testing class Circle3.
3  import java.text.DecimalFormat;
4  import javax.swing.JOptionPane;
5
6  public class CircleTest3 {
7
8     public static void main( String[] args )
9     {
10       // instantiate Circle object
11       Circle3 circle = new Circle3( 37, 43, 2.5 );
12
13       // get Circle3's initial x-y coordinates and radius
14       String output = "X coordinate is " + circle.getX() +
15          "\nY coordinate is " + circle.getY() +
16          "\nRadius is " + circle.getRadius();
17
18       circle.setX( 35 );          // set new x-coordinate
19       circle.setY( 20 );          // set new y-coordinate
20       circle.setRadius( 4.25 );   // set new rad
21
22       // get String representation of
23       output += "\n\nThe new location a
24          circle.toString();
25
```

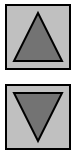
Create `Circle3` object.

Use `Circle3` get method to access `private` instance variables.

Use inherited set methods to modify inherited

Use `Circle3` set method to modify `private` data `radius`.

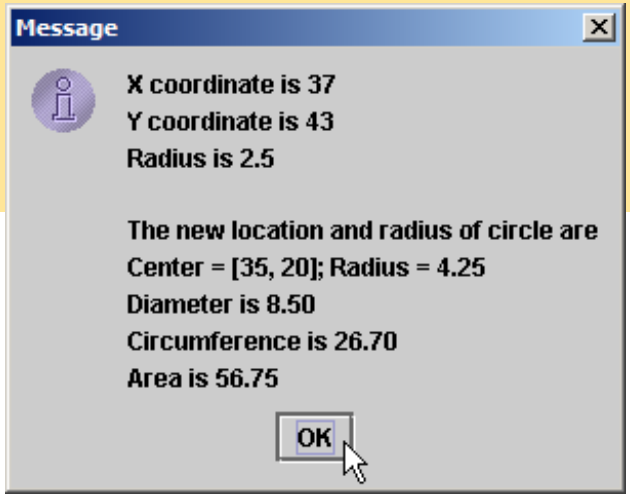
`Circletest3.java`
Line 11
Create `Circle3` object.
Lines 14-15
Use inherited get methods to access inherited `protected` instance
`private` instance variables.
Lines 18-19
modify inherited
`protected` data x and y.
Line 20
Use `Circle3` set method to modify `private` data `radius`.



Outline

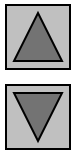
Circletest3.java

```
26        // format floating-point values with 2 digits of precision
27        DecimalFormat twoDigits = new DecimalFormat( "0.00" );
28
29        // get Circle's diameter
30        output += "\nDiameter is " +
31           twoDigits.format( circle.getDiameter() );
32
33        // get Circle's circumference
34        output += "\nCircumference is " +
35           twoDigits.format( circle.getCircumference() );
36
37        // get Circle's area
38        output += "\nArea is " + twoDigits.format( circle.getArea() );
39
40        JOptionPane.showMessageDialog( null, output ); // display output
41
42        System.exit( 0 );
43
44     } // end method main
45
46  } // end class CircleTest3
```



9.4 Relationship between Superclasses and Subclasses (Cont.)

- Using `protected` instance variables
  - Advantages
    - subclasses can modify values directly
    - Slight increase in performance
      - Avoid set/get function call overhead
  - Disadvantages
    - No validity checking
      - subclass can assign illegal value
    - Implementation dependent
      - subclass methods more likely dependent on superclass implementation
      - superclass implementation changes may result in subclass modifications
        - Fragile (brittle) software



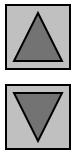
Outline

```
1  // Fig. 9.12: Point3.java
2  // Point class declaration represen
3
4  public class Point3 {
5     private int x;  // x part of coo
6     private int y;  // y part of coordinate pair
7
8     // no-argument constructor
9     public Point3()
10    {
11       // implicit call to Object constructor occurs here
12    }
13
14    // constructor
15    public Point3( int xValue, int yValue )
16    {
17       // implicit call to Object constructor occurs here
18       x = xValue;  // no need for validation
19       y = yValue;  // no need for validation
20    }
21
22    // set x in coordinate pair
23    public void setX( int xValue )
24    {
25       x = xValue;  // no need for validation
26    }
27
```

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

`Point3.java`

Lines 5-6
Better software-engineering practice: `private` over `protected` when possible.



Outline

Point3.java

Line 49
Invoke `public` methods to access `private` instance variables.

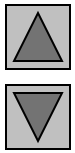
```
28    // return x from coordinate pair
29    public int getX()
30    {
31       return x;
32    }
33
34    // set y in coordinate pair
35    public void setY( int yValue )
36    {
37       y = yValue;  // no need for validation
38    }
39
40    // return y from coordinate pair
41    public int getY()
42    {
43       return y;
44    }
45
46    // return String representation of Point3
47    public String toString()
48    {
49       return "[" + getX() + ", " + getY() + "]";
50    }
51
52 } // end class Point3
```

Invoke `public` methods to access `private` instance variables.

```
1  // Fig. 9.13: Circle4.java
2  // Circle4 class inherits from Point3 and ac
3  // private x and y via Point3's public metho
4
5  public class Circle4 extends Point3 {
6
7     private double radius; // Circle4's radius
8
9     // no-argument constructor
10    public Circle4()
11    {
12       // implicit call to Point3 constructor occurs here
13    }
14
15    // constructor
16    public Circle4( int xValue, int yValue, double radiusValue )
17    {
18       super( xValue, yValue );  // call Point3 constructor explicitly
19       setRadius( radiusValue );
20    }
21
22    // set radius
23    public void setRadius( double radiusValue )
24    {
25       radius = ( radiusValue < 0.0 ? 0.0 : radiusValue );
26    }
27
```

Class `Circle4` inherits from class `Point3`.

Maintain `private` instance variable `radius`.

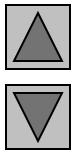


Outline

`Circle4.java`

Line 5
Class `Circle4` inherits from class `Point3`.

Line 7
Maintain `private` instance variable `radius`.



Outline

```
28      // return radius
29      public double getRadius()
30      {
31         return radius;
32      }
33
34      // calculate and return diameter
35      public double getDiameter()
36      {
37         return 2 * getRadius();
38      }
39
40      // calculate and return circumference
41      public double getCircumference()
42      {
43         return Math.PI * getDiameter();
44      }
45
46      // calculate and return area
47      public double getArea()
48      {
49         return Math.PI * getRadius() * getRadius();
50      }
51
52      // return String representation of Circle4 object
53      public String toString()
54      {
55         return "Center = " + super.toString() + "; Radius = " + getRadius();
56      }
57
58   } // end class Circle4
```

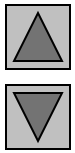
Invoke method `getRadius` rather than directly accessing instance variable `radius`.

Redefine class `Point3`'s method toString.

`Circle4.java`

Line 37, 49 and 55
Invoke method `getRadius` rather than directly accessing instance variable `radius`.

Lines 53-56
Redefine class `Point3`'s method toString.



Outline

```
1  // Fig. 9.14: CircleTest4.java
2  // Testing class Circle4.
3  import java.text.DecimalFormat;
4  import javax.swing.JOptionPane;
5
6  public class CircleTest4 {
7
8     public static void main( String[] args )
9     {
10       // instantiate Circle object
11       Circle4 circle = new Circle4( 37, 43, 2.5 );
12
13       // get Circle4's initial x-y coordinates and radius
14       String output = "X coordinate is " + circle.getX() +
15          "\nY coordinate is " + circle.getY() +
16          "\nRadius is " + circle.getRadius();
17
18       circle.setX( 35 );          // set new x-coordinate
19       circle.setY( 20 );          // set new y-c
20       circle.setRadius( 4.25 ); // set new rad
21
22       // get String representation of new circ
23       output += "\n\nThe new location and radi
24          circle.toString();
25
```

Create `Circle4` object.

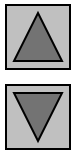
Use inherited get methods to access inherited `private`

Use `Circle4` get method to access `private` instance variable radius.

Use inherited seta methods to modify inherited `private`

Use `Circle4` set method to modify `private` instance variable radius.

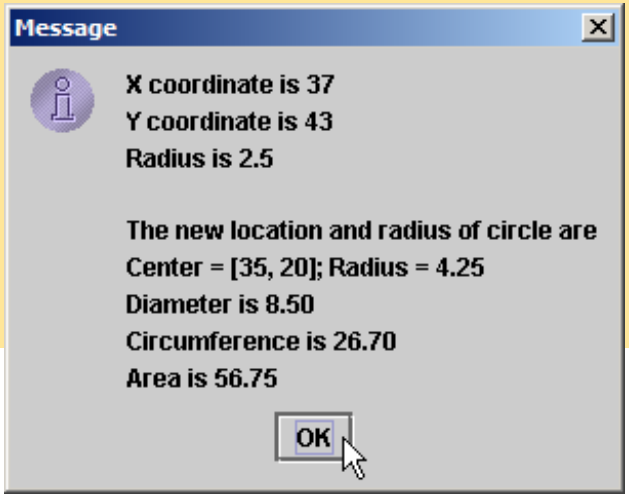
`Circletest4.java`
Line 11
Create `Circle4` object.
Lines 14 and 15
Use inherited get methods to access inherited
variable radius.
Lines 18-19
Use inherited seta
ethods to modify
herited `private`
stance variables x and
.
Line 20
Use `Circle4` set
method to modify
`private` instance
variable radius.



Outline

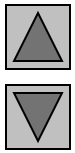
Circletest4.java

```
26        // format floating-point values with 2 digits of precision
27        DecimalFormat twoDigits = new DecimalFormat( "0.00" );
28
29        // get Circle's diameter
30        output += "\nDiameter is " +
31           twoDigits.format( circle.getDiameter() );
32
33        // get Circle's circumference
34        output += "\nCircumference is " +
35           twoDigits.format( circle.getCircumference() );
36
37        // get Circle's area
38        output += "\nArea is " + twoDigits.format( circle.getArea() );
39
40        JOptionPane.showMessageDialog( null, output ); // display output
41
42        System.exit( 0 );
43
44     } // end main
45
46  } // end class CircleTest4
```



9.5 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

```
1  // Fig. 9.15: Cylinder.java
2  // Cylinder class inherits from Circle4.
3
4  public class Cylinder extends Circle4 {
5     private double height;  // Cylinder's height
6
7     // no-argument constructor
8     public Cylinder()
9     {
10       // implicit call to Circle4 constructor occurs here
11    }
12
13    // constructor
14    public Cylinder( int xValue, int yValue, double radiusValue,
15       double heightValue )
16    {
17       super( xValue, yValue, radiusValue ); // call Circle4 constructor
18       setHeight( heightValue );
19    }
20
21    // set Cylinder's height
22    public void setHeight( double heightValue )
23    {
24       height = ( heightValue < 0.0 ? 0.0 : heightValue );
25    }
26
```

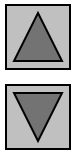
Maintain `private` instance variable `height`.

Class `Cylinder` extends class `Circle4`.

`Cylinder.java`

Line 4
Class `Cylinder` extends class `Circle4`.

Line 5
Maintain `private` instance variable `height`.



Outline

```
27    // get Cylinder's height
28    public double getHeight()
29    {
30       return height;
31    }
32
33    // override Circle4 method getArea to ca
34    public double getArea()
35    {
36       return 2 * super.getArea() + getCircumference() * getHeight();
37    }
38
39    // calculate Cylinder volume
40    public double getVolume()
41    {
42       return super.getArea() * getHeight();
43    }
44
45    // return String representation of Cylinde
46    public String toString()
47    {
48       return super.toString() + "; Height = " + getHeight();
49    }
50
51 } // end class Cylinder
```

Redefine superclass

Invoke superclass `Circle4`'s `getArea` method using keyword super.

Redefine class `Circle4`'s

Invoke superclass `Circle4`'s `toString` method using keyword super.

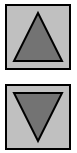
`Cylinder.java`

Line 34 and 42
Redefine superclass `Circle4`'s method `getArea` to return `Cylinder` surface area.

Line 36
Invoke superclass `Circle4`'s `getArea` method using keyword super.

Lines 46-49
Redefine class `Circle4`'s method `toString`.

Line 48
Invoke superclass `Circle4`'s `toString` method using keyword super.



Outline

```
1  // Fig. 9.16: CylinderTest.java
2  // Testing class Cylinder.
3  import java.text.DecimalFormat;
4  import javax.swing.JOptionPane;
5
6  public class CylinderTest {
7
8     public static void main( String[] args )
9     {
10       // create Cylinder object
11       Cylinder cylinder = new Cylinder( 12, 23, 2.5, 5.7 );
12
13       // get Cylinder's initial x-y coordinate
14       String output = "X coordinate is " + cyl
15          "\nY coordinate is " + cylinder.getY() + "\nRadius is " +
16          cylinder.getRadius() + "\nHeight is " + cylinder.getHeight();
17
18       cylinder.setX( 35 );              // set new
19       cylinder.setY( 20 );              // set new
20       cylinder.setRadius( 4.25 );       // set new
21       cylinder.setHeight( 10.75 );      // set new
22
23       // get String representation of new cylind
24       output +=
25          "\n\nThe new location, radius and height of cylinder are\n" +
26          cylinder.toString();
27
```

Invoke indirectly inherited Point3 get methods.

Invoke directly i... Circle4 get method.

Invoke Cylinder get method.

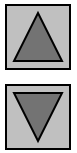
Invoke indirectly inherited Point3 set methods.

Invoke directly inherited C...

Invoke Cylinder set method.

Invoke overridden toString method.

CylinderTest.java
Lines 14 and 15
Invoke indirectly inherited Point3 get methods.
Line 16
...nherited ...ethod.
...nvoke Cylinder get method.
Lines 18-19
Invoke indirectly inherited Point3 set methods.
Line 20
Invoke directly inherited Circle4 set method.
...e 21
...oke Cylinder set method.
Line 26
Invoke overridden toString method.
method.



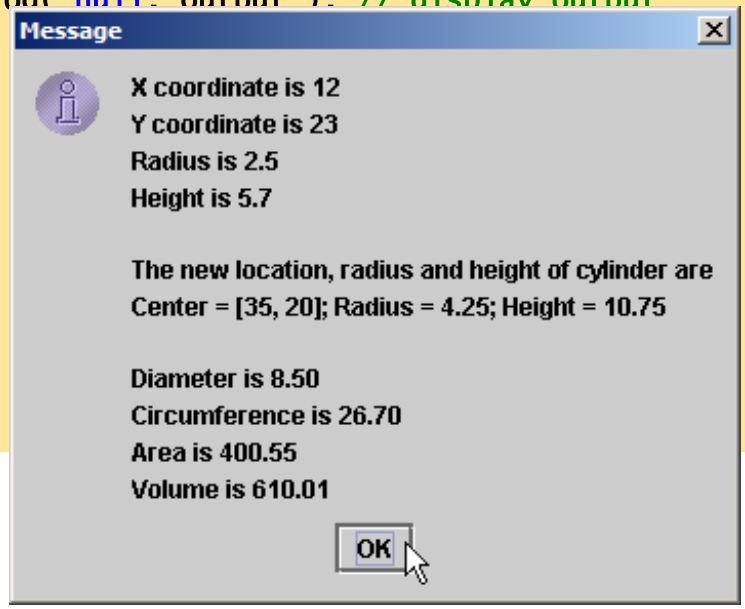
Outline

CylinderTest.java

Line 40
Invoke overridden `getArea` method.

```
28          // format floating-point values with 2 digits of precision
29          DecimalFormat twoDigits = new DecimalFormat( "0.00" );
30
31          // get Cylinder's diameter
32          output += "\n\nDiameter is " +
33             twoDigits.format( cylinder.getDiameter() );
34
35          // get Cylinder's circumference
36          output += "\nCircumference is " +
37             twoDigits.format( cylinder.getCircumference() );
38
39          // get Cylinder's area
40          output += "\nArea is " + twoDigits.format( cylinder.getArea() );
41
42          // get Cylinder's volume
43          output += "\nVolume is " + twoDigits.format( cylinder.getVolume
44
45          JOptionPane.showMessageDialog( null, output ); // display output
46
47          System.exit( 0 );
48
49       } // end main
50
51    } // end class CylinderTest
```

Invoke overridden `getArea` method.

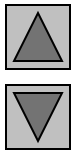


9.6 Constructors and Finalizers in Subclasses

- Instantiating subclass object
  - Chain of constructor calls
    - subclass constructor invokes superclass constructor
      - Implicitly or explicitly
    - Base of inheritance hierarchy
      - Last constructor called in chain is `Object`'s constructor
      - Original subclass constructor's body finishes executing last
      - Example: `Point3`/`Circle4`/`Cylinder` hierarchy
        - `Point3` constructor called second last (last is `Object` constructor)
        - `Point3` constructor's body finishes execution second (first is `Object` constructor's body)

9.6 Constructors and Destructors in Derived Classes

- Garbage collecting subclass object
  - Chain of `finalize` method calls
    - Reverse order of constructor chain
    - Finalizer of subclass called first
    - Finalizer of next superclass up hierarchy next
      - Continue up hierarchy until final superreached
        - After final superclass (`Object`) finalizer, object removed from memory



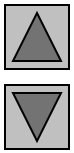
Outline

Point.java

Lines 12, 22 and 28 Constructor and finalizer output messages to demonstrate method call order.

```
1  // Fig. 9.17: Point.java
2  // Point class declaration represents an x-y coordinate pair.
3
4  public class Point {
5     private int x;  // x part of coordinate pair
6     private int y;  // y part of coordinate pair
7
8     // no-argument constructor
9     public Point()
10    {
11       // implicit call to Object constructor occurs here
12       System.out.println( "Point no-argument constructor: " + this );
13    }
14
15    // constructor
16    public Point( int xValue, int yValue )
17    {
18       // implicit call to Object constructor occurs here
19       x = xValue;  // no need for validation
20       y = yValue;  // no need for validation
21
22       System.out.println( "Point constructor: " + this );
23    }
24
25    // finalizer
26    protected void finalize()
27    {
28       System.out.println( "Point finalizer: " + this );
29    }
30
```

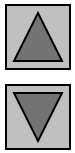
Constructor and finalizer output messages to demonstrate method call order.



Outline

Point.java

```
31      // set x in coordinate pair
32      public void setX( int xValue )
33      {
34         x = xValue;  // no need for validation
35      }
36
37      // return x from coordinate pair
38      public int getX()
39      {
40         return x;
41      }
42
43      // set y in coordinate pair
44      public void setY( int yValue )
45      {
46         y = yValue;  // no need for validation
47      }
48
49      // return y from coordinate pair
50      public int getY()
51      {
52         return y;
53      }
54
55      // return String representation of Point4 object
56      public String toString()
57      {
58         return "[" + getX() + ", " + getY() + "]";
59      }
60
61   } // end class Point
```



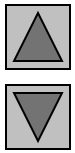
Outline

Circle.java

Lines 12, 21 and 29 Constructor and finalizer output messages to

Constructor and finalizer output messages to demonstrate method call order.

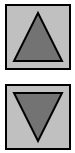
```
1  // Fig. 9.18: Circle.java
2  // Circle5 class declaration.
3
4  public class Circle extends Point {
5
6     private double radius;  // Circle's radius
7
8     // no-argument constructor
9     public Circle()
10    {
11       // implicit call to Point constructor occurs here
12       System.out.println( "Circle no-argument constructor: " + this );
13    }
14
15    // constructor
16    public Circle( int xValue, int yValue, double radiusValue )
17    {
18       super( xValue, yValue );  // call Point constructor
19       setRadius( radiusValue );
20
21       System.out.println( "Circle constructor: " + this );
22    }
23
24    // finalizer
25    protected void finalize()
26    {
27       System.out.println( "Circle finalizer: " + this );
28
29       super.finalize();  // call superclass finalize method
30    }
31
```



Outline

Circle.java

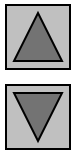
```
32       // set radius
33       public void setRadius( double radiusValue )
34       {
35          radius = ( radiusValue < 0.0 ? 0.0 : radiusValue );
36       }
37
38       // return radius
39       public double getRadius()
40       {
41          return radius;
42       }
43
44       // calculate and return diameter
45       public double getDiameter()
46       {
47          return 2 * getRadius();
48       }
49
50       // calculate and return circumference
51       public double getCircumference()
52       {
53          return Math.PI * getDiameter();
54       }
```



Outline

Circle.java

```
55
56       // calculate and return area
57       public double getArea()
58       {
59          return Math.PI * getRadius() * getRadius();
60       }
61
62       // return String representation of Circle5 object
63       public String toString()
64       {
65          return "Center = " + super.toString() + "; Radius = " + getRadius();
66       }
67
68    } // end class Circle
```



Outline

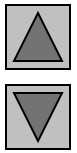
ConstructorFinalizerTest.java

- e 12
- int object goes in and out of scope immediately.
- 18
- o
- cts to
- rder of subclass and superclass constructor/finalizer method calls.

```
1  // Fig. 9.19: ConstructorFinalizerTest.java
2  // Display order in which superclass and subclass
3  // constructors and finalizers are called.
4
5  public class ConstructorFinalizerTest {
6
7     public static void main( String args[] )
8     {
9        Point point;
10       Circle circle1, circle2;
11
12       point = new Point( 11, 22 );
13
14       System.out.println();
15       circle1 = new Circle( 72, 29, 4.5 );
16
17       System.out.println();
18       circle2 = new Circle( 5, 7, 10.67 );
19
20       point = null;    // mark for garbage collection
21       circle1 = null;  // mark for garbage collection
22       circle2 = null;  // mark for garbage collection
23
24       System.out.println();
25
```

Point object goes in and out of scope immediately.

Instantiate two Circle objects to demonstrate order of subclass and superclass constructor/finalizer method calls.



Outline

ConstructorFinalizerTest.java

```
26         System.gc();  // call the garbage collector
27
28      } // end main
29
30   } // end class ConstructorFinalizerTest
```

```
Point constructor: [11, 22]

Point constructor: Center = [72, 29]; Radius = 0.0
Circle constructor: Center = [72, 29]; Radius = 4.5

Point constructor: Center = [5, 7]; Radius = 0.0
Circle constructor: Center = [5, 7]; Radius = 10.67

Point finalizer: [11, 22]
Circle finalizer: Center = [72, 29]; Radius = 4.5
Point finalizer: Center = [72, 29]; Radius = 4.5
Circle finalizer: Center = [5, 7]; Radius = 10.67
Point finalizer: Center = [5, 7]; Radius = 10.67
```

Subclass `Circle` constructor body executes after superclass `Point4`'s constructor finishes execution.

Finalizer for `Circle` object called in reverse order of constructors.

9.9 Software Engineering with Inheritance

- Customizing existing software
  - Inherit from existing classes
    - Include additional members
    - Redefine superclass members
    - No direct access to superclass'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