Chapter 10 – Object-Oriented Programming: Inheritance

Outline

Some slides modified by LL

10.1. Introduction

- Inheritance:
  - A new class (NC) can be created by absorbing the methods and variables of an existing class
  - NC then adds its own methods to enhance its capabilities
  - NC is called a derived class
    - Because NC derives (by inheritance) methods and variables from a base class
  - Objects of derived class are objects of base class but not vice versa
    - E.g. a ‘car’ object derived from a ‘vehicle’ object
  - “Is-a” relationship: a derived class object can be treated as a base class object (e.g., a car is-a vehicle)
  - “Has-a” relationship: class object has object references as members (e.g., a car has-a body, a car has-an engine)
  - A derived class can only access non-private base class members
    - Unless it inherits accessor functions that allow for such access

10.2. Base Classes and Derived Classes

- An object of one class often is an object of another class
  - E.g., a car is a vehicle
- Every derived-class object is an object of its base class
- Inheritance forms a tree-like hierarchy (see slide +2)
- Specifying that class `car` is derived from class `vehicle`:
  - `class car : vehicle`
- Composition:
  - Formed by “has-a” relationships
- Class constructors are *not* inherited

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

Fig. 10.2 Inheritance examples.

Some slides modified by LL

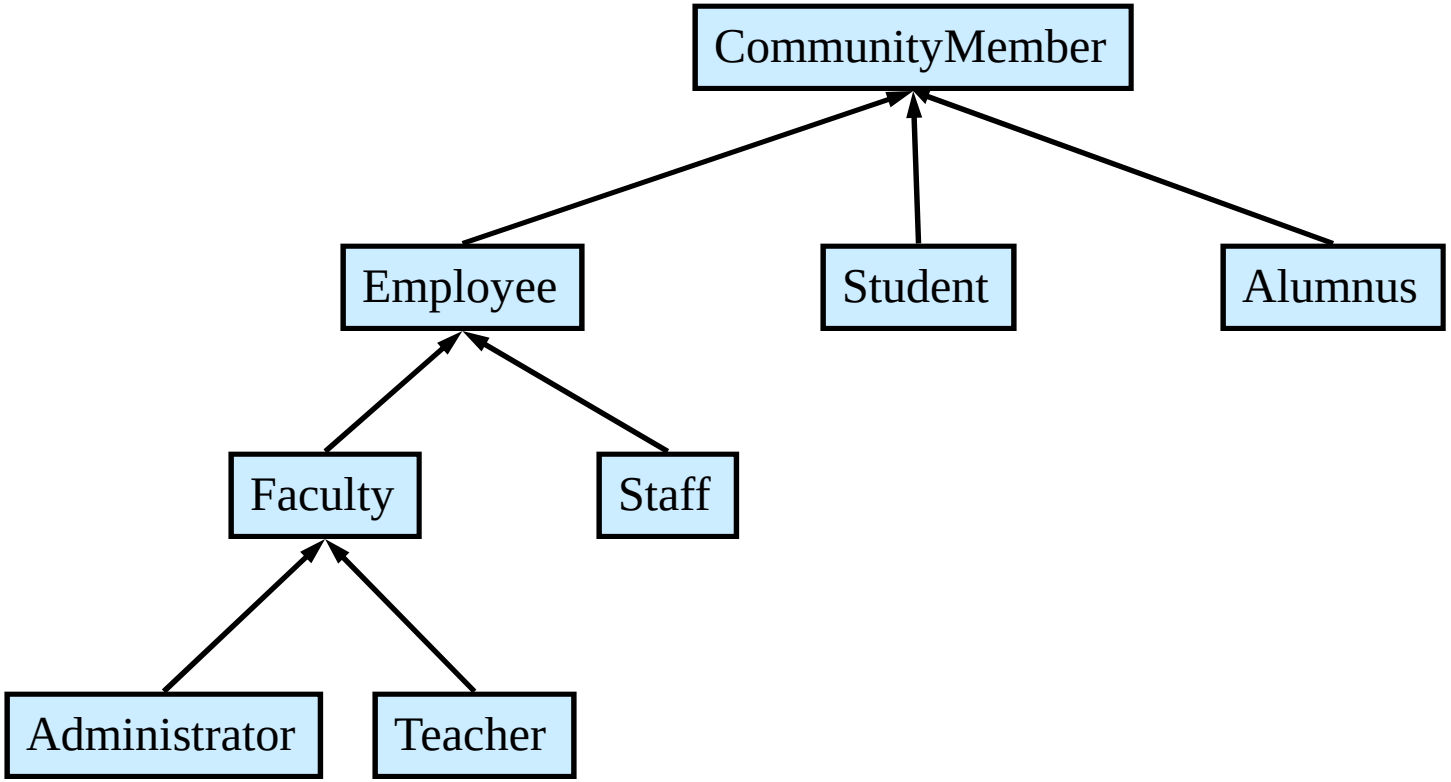


Fig. 10.2 Inheritance hierarchy for university **`CommunityMember`**s.

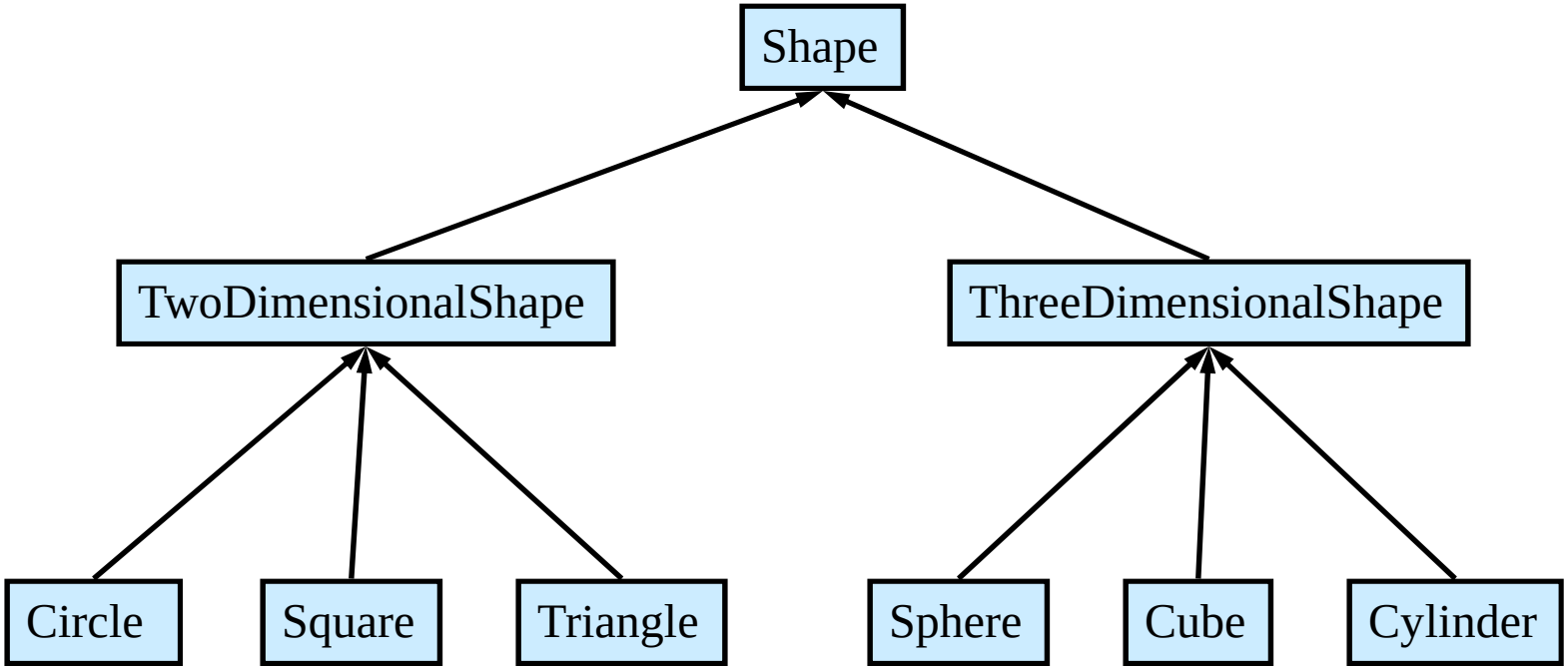
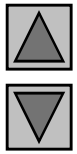


Fig. 10.3 Portion of a **Shape** class hierarchy.

10.3. `protected` [and `internal`] Members

- We know `public` and `private` members of a base class
  - public member: accessible anywhere that the program has a reference to an object of that base class or one of its derived classes
  - private member: accessible only within the body of that base class
- Now two intermediate (between public and private) levels of protection for members of a base class :
  - `protected` member: accessible by base class or any class derived from that base class
  - `internal` members: accessible in any part of the assembly in which the member is declared
    - Recall: The assembly = a package containing the MS Intermediate Language (MISL) code that a *project* has been *compiled into*, plus any other info that is needed for its classes
- Overridden base class members can be accessed using:
  - `base.member` (e.g., `base.ToString`) - '`base`' is the keyword

- Use a point-circle hierarchy to represent relationship between base and derived classes
- The first thing a derived class does is call its base class' constructor
  - Calls either explicitly or implicitly
- `override` keyword is needed if a derived-class method overrides a base-class method
  - E.g, `public override double Area()` [sl.34]
- If a base class method is going to be overridden it must be declared `virtual`
  - `E.g., public virtual double area()` [sl.18]



Outline

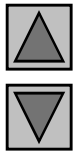
Point.cs

```
1    // Fig. 9.4: Point.cs [textbook ed. 1]
2    // Point class represents an x-y coordinate pair.
3
4    using System;
5
6    // Point class definition implicitly inherits from Object (System.Object)
7    public class Point
8    {
9       // point coordinates
10      private int x, y;
11
12      // default (no-argument) constructor
13      public Point()
14      {
15         // implicit call to Object constructor occurs here
16      }
17
18      // constructor
19      public Point( int xValue, int yValue )
20      {
21         // implicit call to Object constructor occurs here
22         X = xValue;
23         Y = yValue;
24      }
25
26      // property X
27      public int X
28      {
29         get
30         {
31            return x;
32         }
33
```

X and Y coordinates, declared private so other classes cannot directly access them

Default point constructor with implicit call to System's Object constructor

Constructor to set coordinates to parameters, also has implicit call to System's Object constructor



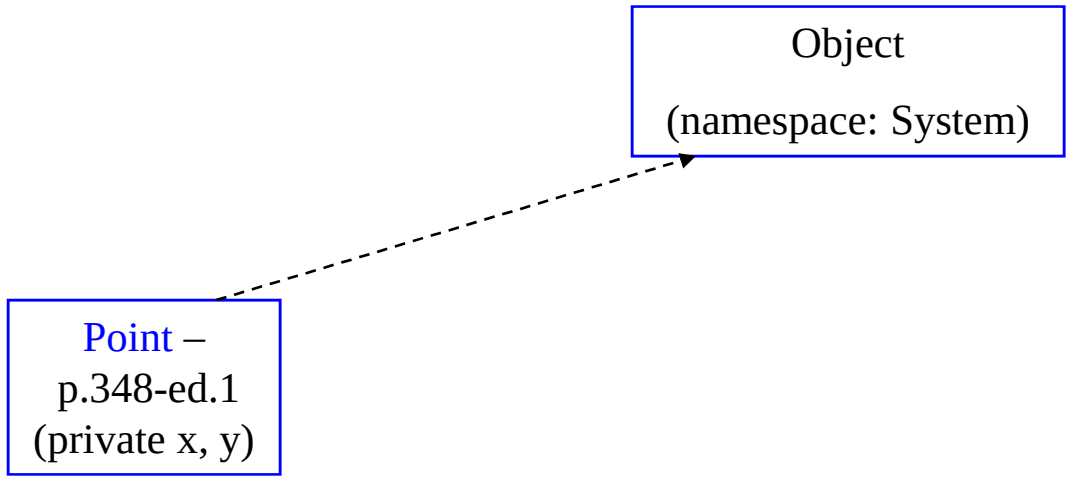
Outline

Point.cs

```
34          set
35          {
36             x = value; // no need for validation
37          }
38
39       } // end property X
40
41       // property Y
42       public int Y
43       {
44          get
45          {
46             return y;
47          }
48
49          set
50          {
51             y = value; // no need for validation
52          }
53
54       } // end property Y
55
56       // return string representation of Point
57       public override string ToString()
58       {
59          return "[" + x + ", " + y + "]";
60       }
61
62    } // end class Point
```

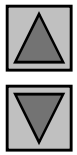
Definition of overridden method ToString (overrides System's Object.ToString)

Program Output



--------> implicit inheritance

Slide added by L. Lilien
Some slides modified by LL



Outline

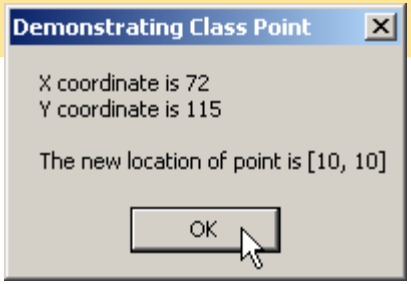
PointTest.cs

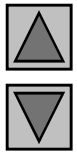
```
1    // Fig. 9.5: PointTest.cs [textbook ed. 1]
2    // Testing class Point.
3
4    using System;
5    using System.Windows.Forms;
6
7    // PointTest class definition
8    class PointTest
9    {
10      // main entry point for application
11      static void Main( string[] args )
12      {
13         // instantiate Point object
14         Point point = new Point( 72, 115 );
15
16         // display point coordinates via X and Y properties
17         string output = "X coordinate is " + point.X +
18            "\n" + "Y coordinate is " + point.Y;
19
20         point.X = 10; // set x-coordinate via X property
21         point.Y = 10; // set y-coordinate via Y property
22
23         // display new point value
24         output += "\n\nThe new location of point is " + point;
25
26         MessageBox.Show( output, "Demonstrating Class Point" );
27
28      } // end method Main
29
30   } // end class PointTest
```

Create a Point object

Calls the ToString method of class Point *implicitly* (converts '`point`' to string, bec. output is a string)

Change coordinates of Point object





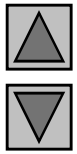
Outline

Circle.cs

```
1    // Fig. 9.6: Circle.cs [textbook ed. 1]
2    // Circle class contains x-y coordinate pair and radius.
3
4    using System;
5
6    // Circle class definition implicitly inherits from Object
7    public class Circle
8    {
9       private int x, y;    // coordinates of Circle's ce
10      private double radius; // Circle's radius
11
12      // default constructor
13      public Circle()
14      {
15         // implicit call to Object constructor occurs here
16      }
17
18      // constructor
19      public Circle( int xValue, int yValue, double radiusValue )
20      {
21         // implicit call to Object constructor occurs here
22         x = xValue;
23         y = yValue;
24         Radius = radiusValue;
25      }
26
27      // property X
28      public int X
29      {
30         get
31         {
32            return x;
33         }
34
```

Declare coordinates and radius of circle as private

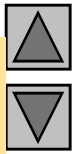
Circle constructors



Outline

Circle.cs

```
35          set
36          {
37             x = value;  // no need for validation
38          }
39
40       } // end property X
41
42       // property Y
43       public int Y
44       {
45          get
46          {
47             return y;
48          }
49
50          set
51          {
52             y = value;  // no need for validation
53          }
54
55       } // end property Y
56
57       // property Radius
58       public double Radius
59       {
60          get
61          {
62             return radius;
63          }
64
65          set
66          {
67             if ( value >= 0 )   // validation needed
68                radius = value;
69          }
```

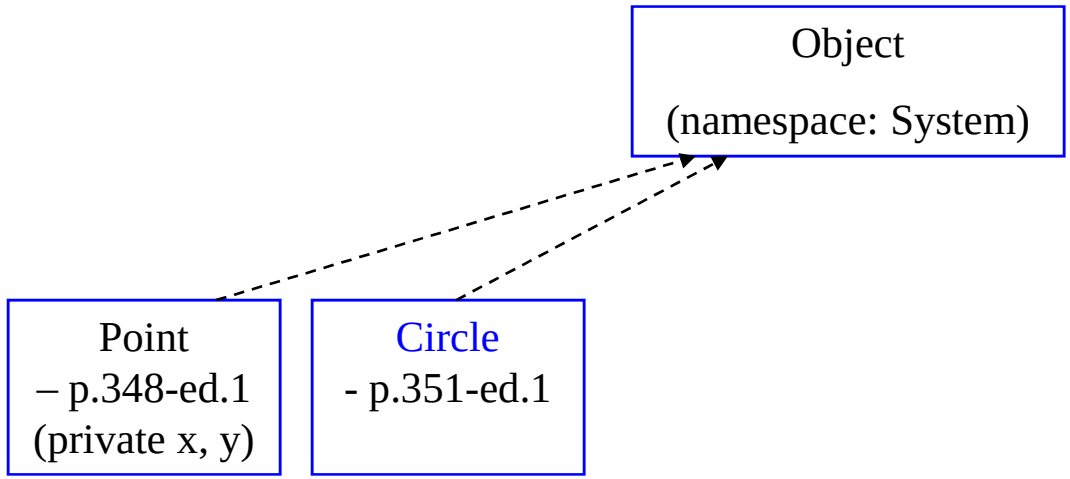


Outline

Circle.cs

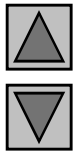
```
70
71      } // end property Radius
72
73      // calculate Circle diameter
74      public double Diameter()
75      {
76         return radius * 2;
77      }
78
79      // calculate Circle circumference
80      public double Circumference()
81      {
82         return Math.PI * Diameter();
83      }
84
85      // calculate Circle area
86      public double Area()
87      {
88         return Math.PI * Math.Pow( radius, 2 );
89      }
90
91      // return string representation of Circle
92      public override string ToString()
93      {
94         return "Center = [" + x + ", " + y + "]" +
95            "; Radius = " + radius;
96      }
97
98        } // end class Circle
```

Definition of overridden method ToString



implicit inheritance

Slide added by L. Lilien
Some slides modified by LL



Outline

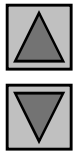
CircleTest.cs

```
1   // Fig. 9.7: CircleTest.cs [textbook ed. 1]
2   // Testing class Circle.
3
4   using System;
5   using System.Windows.Forms;
6
7   // CircleTest class definition
8   class CircleTest
9   {
10     // main entry point for application.
11     static void Main( string[] args )
12     {
13        // instantiate Circle
14        Circle circle = new Circle( 37, 43, 2.5 );
15
16        // get Circle's initial x-y coordinates and radius
17        string output = "X coordinate is " + circle.X +
18           "\nY coordinate is " + circle.Y + "\nRadius is " +
19           circle.Radius;
20
21        // set Circle's x-y coordinates and radius to new values
22        circle.X = 2;
23        circle.Y = 2;
24        circle.Radius = 4.25;
25
26        // display Circle's string representation
27        output += "\n\nThe new location and radius of " +
28           "circle are \n" + circle + "\n";
29
30        // display Circle's diameter
31        output += "Diameter is " +
32           String.Format( "{0:F}", circle.Diameter() ) + "\n";
33
```

Create a Circle object

Change coordinates and radius of Circle object

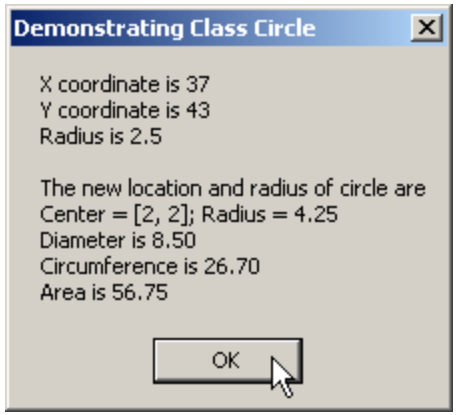
Implicit call to circle's ToString method (converts `circle` to string bec. `output` is a string)

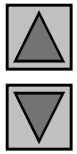


Outline

```
34          // display Circle's circumference
35          output += "Circumference is " +
36             String.Format( "{0:F}", circle.Circumference() ) + "\n";
37
38          // display Circle's area
39          output += "Area is " +
40             String.Format( "{0:F}", circle.Area() );
41
42          MessageBox.Show( output, "Demonstrating Class Circle" );
43
44       } // end method Main
45
46    } // end class CircleTest
```

Call Circle's Circumference and Area methods for output





Outline

Circle2.cs

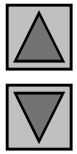
```
1    // Fig. 9.8: Circle2.cs [textbook ed. 1]
2    // Circle2 class that inherits from cl
3
4    using System;
5
6    // Circle2 class definition inherits from Point
7    class Circle2 : Point
8    {
9       private double radius; // Circle2's radius
10
11      // default constructor
12      public Circle2()
13      {
14         // implicit call to Point constructor occurs here (not to
                  Object constructor as before)
15      }
16
17      // constructor
18      public Circle2( int xValue, int yValue, double radiusValue )
19      {
20         // implicit call to Point construc
21         x = xValue;
22         y = yValue;
23         Radius = radiusValue;
24      }
25
26      // property Radius
27      public double Radius
28      {
29         get
30         {
31            return radius;
32         }
33
```

Declare class Circle to derive from class Point

Declare radius as private

Implicit calls to base class constructor

Attempt to directly change private base class methods results in an error

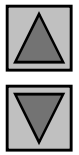


Outline

Circle2.cs

```
34          set
35          {
36             if ( value >= 0 )
37                radius = value;
38          }
39
40       } // end property Radius
41
42       // calculate Circle diameter
43       public double Diameter()
44       {
45          return radius * 2;
46       }
47
48       // calculate Circle circumference
49       public double Circumference()
50       {
51          return Math.PI * Diameter();
52       }
53
54       // calculate Circle area
55       public virtual double area()
56       {
57          return Math.PI * Math.Pow( radius, 2 );
58       }
59
60       // return string representation Circle
61       public override string ToString()
62       {
63          return "Center = [" + x + ", " + y + "]" +
64             "; Radius = " + radius;
65       }
66
67    } // end class Circle2
```

Attempt to directly access private base class members results in an error

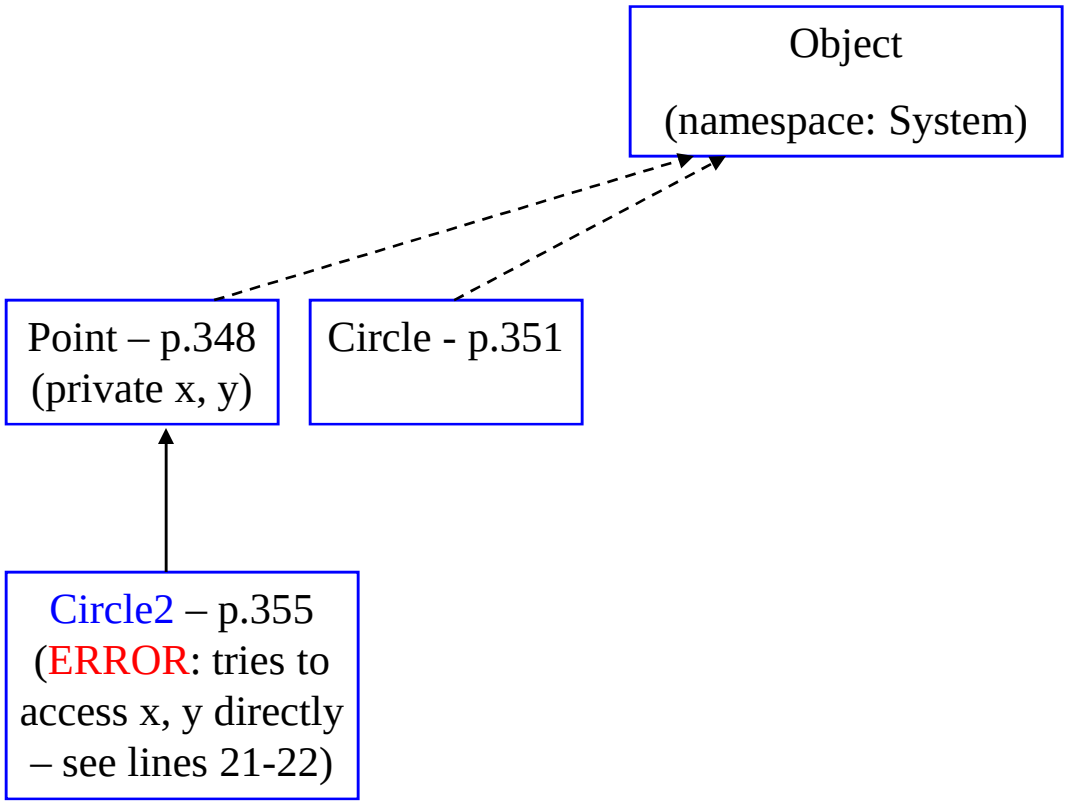


Outline

Circle2.cs program output

Task List - 3 Build Error tasks shown (filtered)

!		Description	File	Line
		Click here to add a new task		
!		'Circle2.Point.x' is inaccessible due to its protection level	C:\...\Circle2.cs	23
!		'Circle2.Point.y' is inaccessible due to its protection level	C:\...\Circle2.cs	24
!		'Circle2.Point.x' is inaccessible due to its protection level	C:\...\Circle2.cs	65

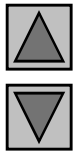


implicit inheritance

explicit inheritance

Slide added by L. Lilien

Some slides modified by LL

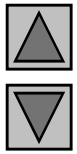


Outline

Point2.cs

```
1    // Fig. 9.9: Point2.cs [textbook ed. 1]
2    // Point2 class contains an x-y coordinate pair as protected data.
3
4    using System;
5
6    // Point2 class definition implicitly inherits from Object
7    public class Point2
8    {
9       // point coordinate
10      protected int x, y;
11
12      // default constructor
13      public Point2()
14      {
15         // implicit call to Object co
16      }
17
18      // constructor
19      public Point2( int xValue, int yValue )
20      {
21         // implicit call to Object constructor occurs here
22         X = xValue;
23         Y = yValue;
24      }
25
26      // property X
27      public int X
28      {
29         get
30         {
31            return x;
32         }
33
```

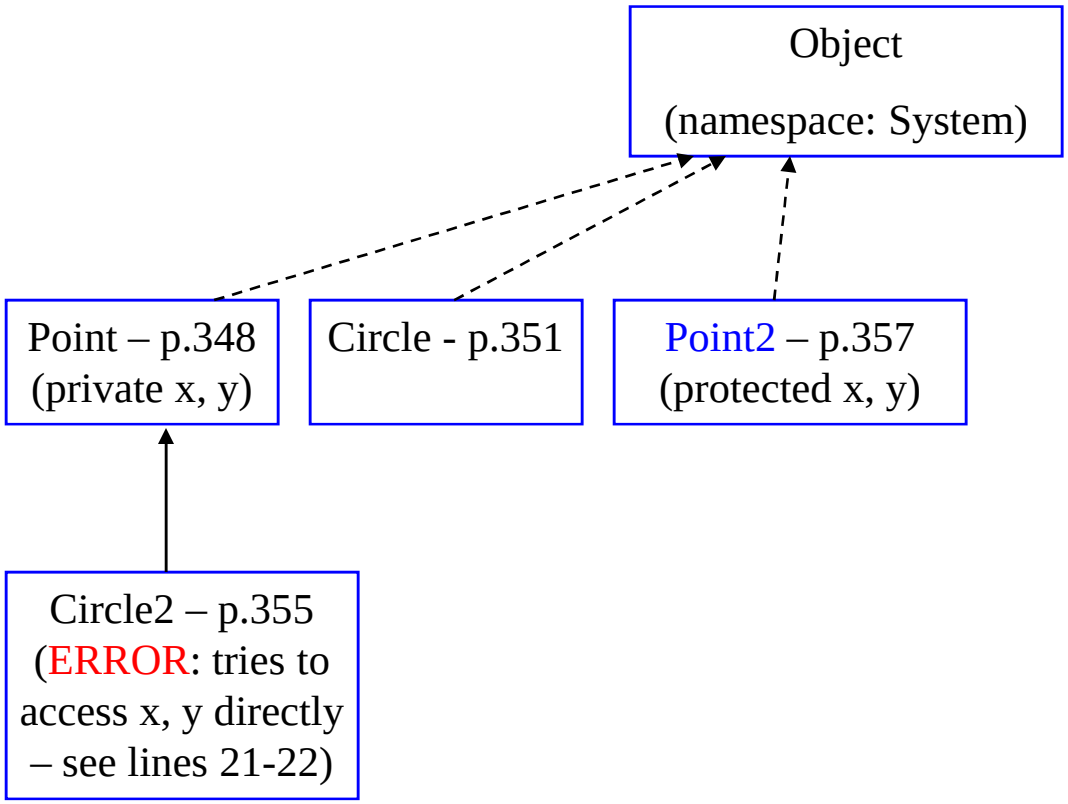
Declare coordinates as `protected` so derived classes can directly access them (they were `private` in Point.cs of Fig. 9.4)



Outline

Point2.cs

```
34          set
35          {
36             x = value; // no need for validation
37          }
38
39       } // end property X
40
41       // property Y
42       public int Y
43       {
44          get
45          {
46             return y;
47          }
48
49          set
50          {
51             y = value; // no need for validation
52          }
53
54       } // end property Y
55
56       // return string representation of Point2
57       public override string ToString()
58       {
59          return "[" + x + ", " + y + "]";
60       }
61
62    } // end class Point2
```

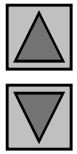


implicit inheritance

explicit inheritance

Slide added by L. Lilien

Some slides modified by LL



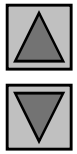
Outline

Circle3.cs

```
1    // Fig. 9.10: Circle3.cs [textbook ed. 1]
2    // Circle3 class that inherits from class Point2.
3         // Circle2 inheriting from class Point caused error.
4    using System;
5
6    // Circle3 class definition inherits from Point2
7    public class Circle3 : Point2
8    {
9       private double radius; // Circle's radius
10
11      // default constructor
12      public Circle3()
13      {
14         // implicit call to Point constructor occurs here
15      }
16
17      // constructor
18      public Circle3(
19         int xValue, int yValue, double radiusValue )
20      {
21         // implicit call to Point constructor occurs here
22         x = xValue;
23         y = yValue;
24         Radius = radiusValue;
25      }
26
27      // property Radius
28      public double Radius
29      {
30         get
31         {
32            return radius;
33         }
34
```

Class Circle3 inherits from Point2

Directly changing `protected` base class members does not result in error (as it did for `private` base class members before)

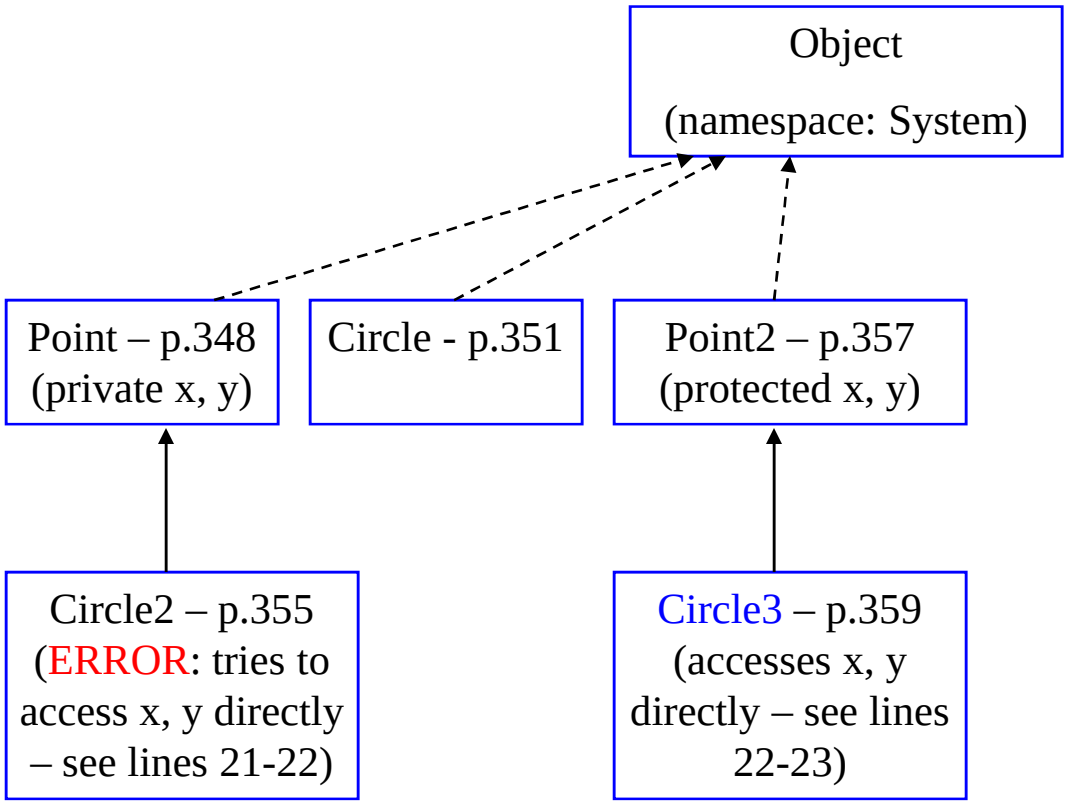


Outline

Circle3.cs

```
35          set
36          {
37             if ( value >= 0 )
38                radius = value;
39          }
40
41       } // end property Radius
42
43       // calculate Circle diameter
44       public double Diameter()
45       {
46          return radius * 2;
47       }
48
49       // calculate circumference
50       public double Circumference()
51       {
52          return Math.PI * Diameter();
53       }
54
55       // calculate Circle area
56       public virtual double Area()
57       {
58          return Math.PI * Math.Pow( radius, 2 );
59       }
60
61       // return string representation of Circle3
62       public override string ToString()
63       {
64          return "Center = [" + x + ", " + y + "]" +
65             "; Radius = " + radius;
66       }
67
68    } // end class Circle3
```

Directly accessing `protected` members does not result in error (as it did for `private` base class members before)

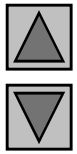


implicit inheritance

explicit inheritance

Slide added by L. Lilien

Some slides modified by LL



Outline

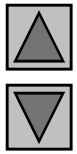
CircleTest3.cs

```
1    / Fig. 9.11: CircleTest3.cs [textbook ed. 1]
2    // Testing class Circle3.
3
4    using System;
5    using System.Windows.Forms;
6
7    // CircleTest3 class definition
8    class CircleTest3
9    {
10      // main entry point for application
11      static void Main( string[] args )
12      {
13         // instantiate Circle3
14         Circle3 circle = new Circle3( 37, 43, 2.5 );
15
16         // get Circle3's initial x-y coordinates and radius
17         string output = "X coordinate is " + circle.X + "\n" +
18            "Y coordinate is " + circle.Y + "\nRadius is " +
19            circle.Radius;
20
21         // set Circle3's x-y coordinates and radius to new values
22         circle.X = 2;
23         circle.Y = 2;
24         circle.Radius = 4.25;
25
26         // display Circle3's string representation
27         output += "\n\n" +
28            "The new location and radius of circle are " +
29            "\n" + circle + "\n";
30
31         // display Circle3's Diameter
32         output += "Diameter is " +
33            String.Format( "{0:F}", circle.Diameter() ) + "\n";
34
```

Change coordinates and radius of Circle3 object

Create new Circle3 object

Implicit call to Circle3's ToString method (to convert `circle` into a string bec. `output` is a string)

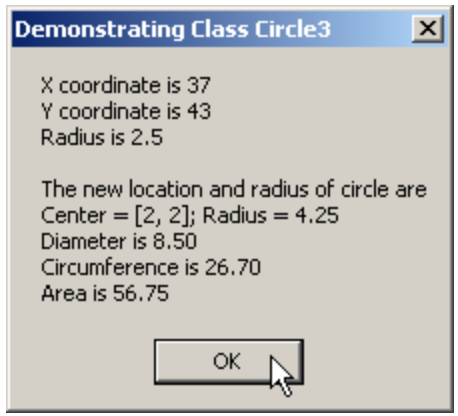


Outline

```
35          // display Circle3's Circumference
36          output += "Circumference is " +
37             String.Format( "{0:F}", circle.Circumference() ) + "\n";
38
39          // display Circle3's Area
40          output += "Area is " +
41             String.Format( "{0:F}", circle.Area() );
42
43          MessageBox.Show( output, "Demonstrating Class Circle3" );
44
45       } // end method Main
46
47    } // end class CircleTest3
```

cs

Call Circle's Circumference and Area methods for output



Problems with `protected` variables

- `Point2` used `protected` instance variables x, y to allow `Circle3` (and other derived-class objects) direct access to x, y
  - Also (a bit) faster execution for direct access than access via `set` / `get` accessors
- Problems with `protected` instance variables

  1) derived-class objects can assign illegal value to the `protected` data

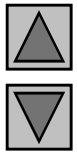
  2) software becomes *brittle* / *fragile*:

  change of base-class implementation forces changes of implementations of derived-classes

  — Here: changing names of `x, y` to, e.g., `xCoord, yCoord`

- Let's write Point3.cs and Circle4.cs that work correctly with `private`, not `protected` instance variables x, y

Some slides modified by LL

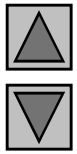


Outline

Point3.cs

```
1    // Fig. 9.12: Point3.cs [textbook ed. 1]
2    // Point3 class represents an x-y coordinate pair.
3
4    using System;
5
6    // Point3 class definition implicitly inherits from Object
7    public class Point3
8    {
9       // point coordinate
10      private int x, y;
11
12      // default constructor
13      public Point3()
14      {
15         // implicit call to Object constructor
16      }
17
18      // constructor
19      public Point3( int xValue, int yValue )
20      {
21         // implicit call to Object constructor occurs here
22         X = xValue;   // use property X
23         Y = yValue;   // use property Y
24      }
25
26      // property X
27      public int X
28      {
29         get
30         {
31            return x;
32         }
33
```

Declare coordinates as `private` (as in Point.cs – better s/w engineering if `private` – see p.291/3, 361/-1)



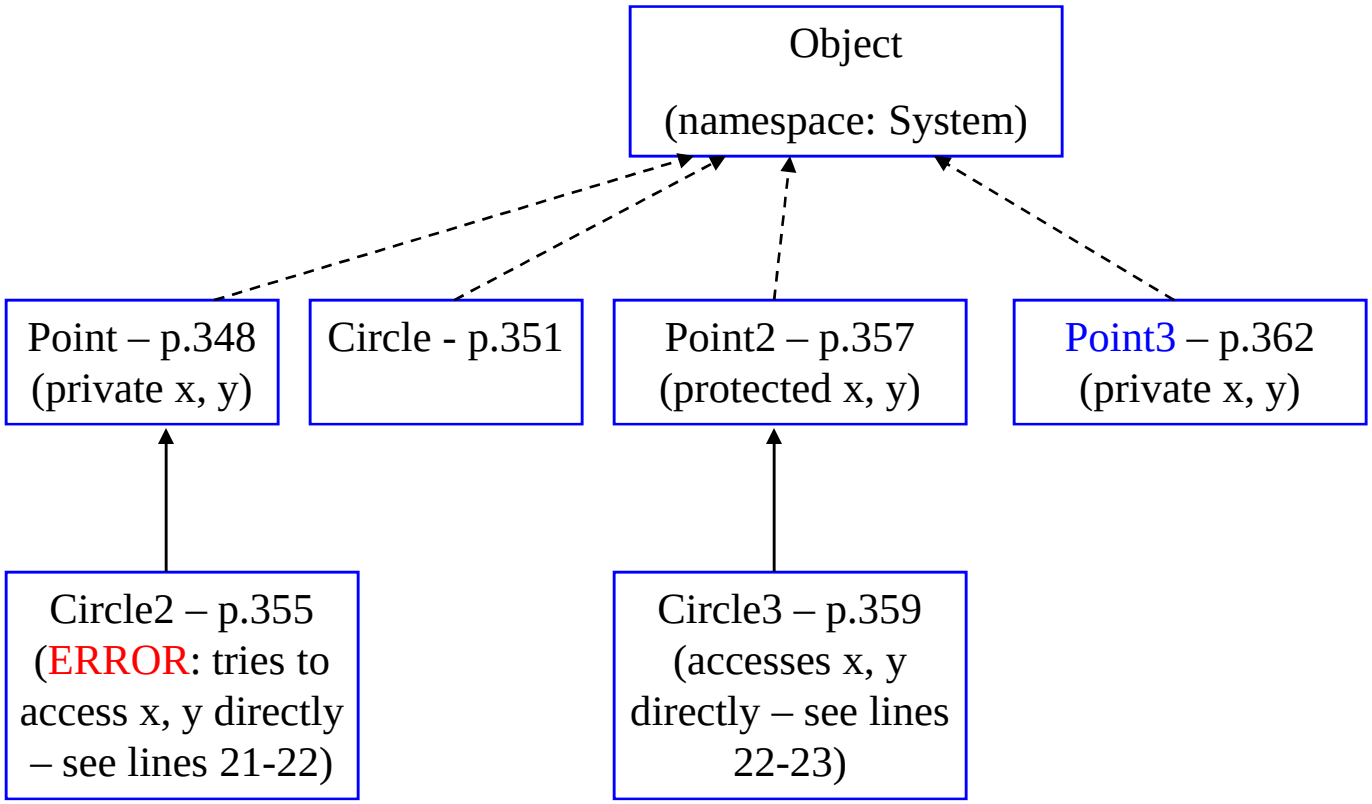
Outline

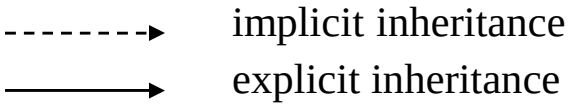
Point3.cs

```
34          set
35          {
36             x = value; // no need for validation
37          }
38
39       } // end property X
40
41       // property Y
42       public int Y
43       {
44          get
45          {
46             return y;
47          }
48
49          set
50          {
51             y = value; // no need for validation
52          }
53
54       } // end property Y
55
56       // return string representation of Point3
57       public override string ToString()
58       {
59          return "[" + X + ", " + Y + "]"; // uses properties X and Y
                    // unlike Point.cs (which used variables x and y)
                    // Better s/w engineering if x and y are private
                    // but accessible as needed!!!
60       }
61
62    } // end class Point3
```

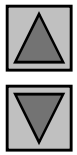
Methods to set x and y coordinates

Overridden ToString method





Slide added by L. Lilien
Some slides modified by LL



Outline

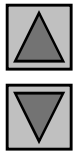
Circle4.cs

```
1    // Fig. 9.13: Circle4.cs [textbook ed. 1]
2    // Circle4 class that inherits from class Point3.
3          // Shows how Class 4 uses Point3 methods to manipulate
           // private Point3 data
4    using System;
5
6    // Circle4 class definition inherits from Point3
7    public class Circle4 : Point3
8    {
9       private double radius;
10
11      // default constructor
12      public Circle4()
13      {
14         // implicit call to Point constructor occurs here
15      }
16
17      // constructor
18      public Circle4( int xValue, int yValue, double radiusValue )
19         : base( xValue, yValue )
20      {
21         Radius = radiusValue;
22      }
23
24      // property Radius
25      public double Radius
26      {
27         get
28         {
29            return radius;
30         }
31
32         set
33         {
34            if ( value >= 0 )   // validation needed
35               radius = value;
```

Constructor with implicit call to base class constructor

Constructor with explicit call to base class constructor (compare Line 19 to Lines 22-23 in Circle3)

Explicit call to base class constructor



Outline

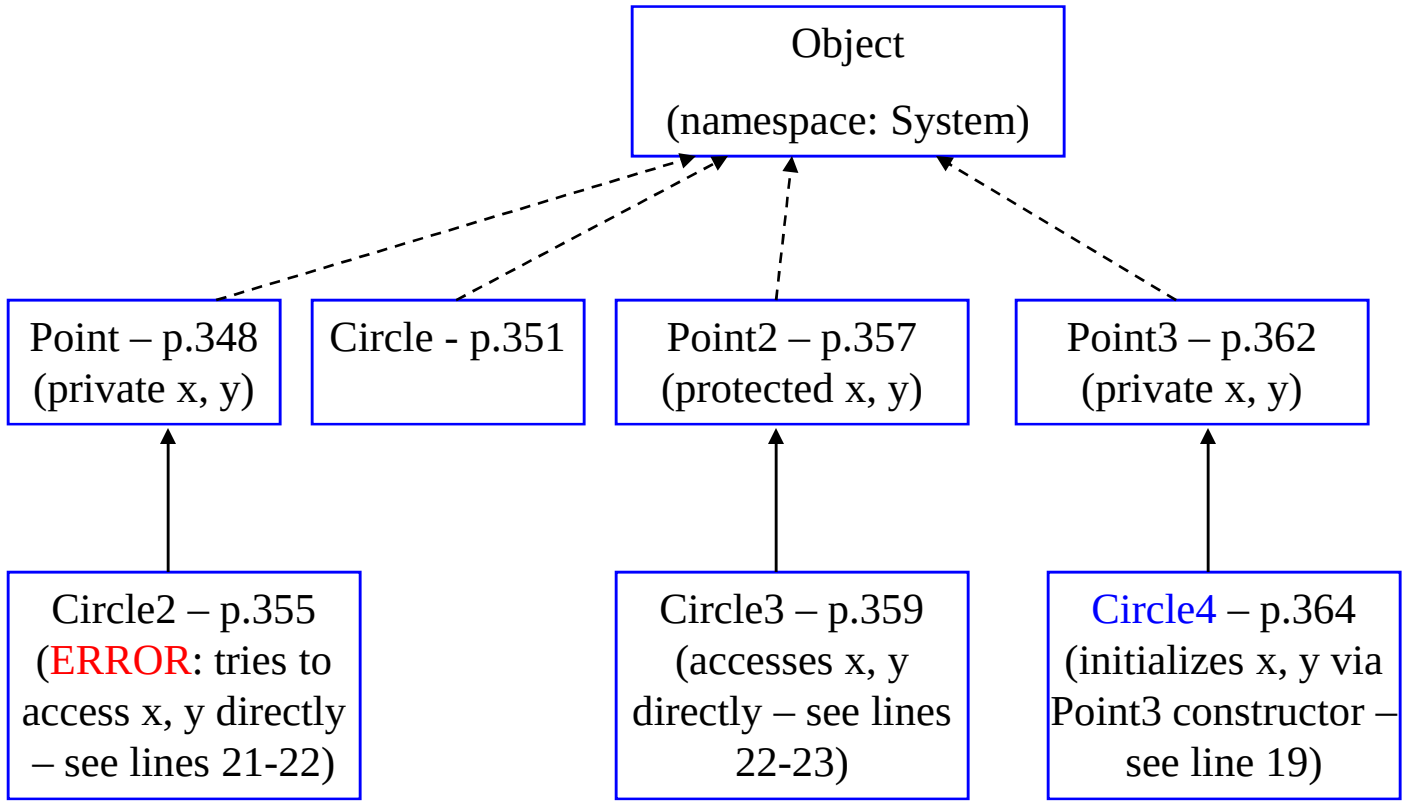
Circle4.cs

```
36         }
37
38      } // end property Radius
39
40      // calculate Circle diameter
41      public double Diameter()
42      {
43         return Radius * 2;     // use property Radius
44      }
45
46      // calculate Circle circumference
47      public double Circumference()
48      {
49         return Math.PI * Diameter();
50      }
51
52      // calculate Circle area
53      public virtual double Area()
54      {
55         return Math.PI * Math.Pow( Radius, 2 );  // use propert
56      }
57
58      // return string representation of Circle4
59      public override string ToString()
60      {
61         // use base reference to return Point string representation
62         return "Center= " + base.ToString() +
63            "; Radius = " + Radius;  // use property Radius
64      }
65
66   } // end class Circle4
```

Method area declared virtual so it can be overridden

Circle4's ToString method overrides Point3's ToString method

Call Point3's ToString method to display coordinates

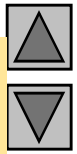


implicit inheritance

explicit inheritance

Slide added by L. Lilien

Some slides modified by LL



Outline

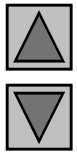
CircleTest4.cs

```
1    // Fig. 9.14: CircleTest4.cs [textbook ed. 1]
2    // Testing class Circle4.
3
4    using System;
5    using System.Windows.Forms;
6
7    // CircleTest4 class definition
8    class CircleTest4
9    {
10      // main entry point for application
11      static void Main( string[] args )
12      {
13         // instantiate Circle4
14         Circle4 circle = new Circle4( 37, 43, 2.9 );
15
16         // get Circle4's initial x-y coordinates and radius
17         string output = "X coordinate is " + circle.X + "\n" +
18            "Y coordinate is " + circle.Y + "\n" +
19            "Radius is " + circle.Radius;
20
21 // set Circle4's x-y coordinates and radius to new values
     // via properties: X, Y, Radius -- not via private: x, y, radius
22         circle.X = 2;
23         circle.Y = 2;
24         circle.Radius = 4.25;
25
26         // display Circle4's string representation
27         output += "\n\n" +
28            "The new location and radius of circle are " +
29            "\n" + circle + "\n";
30
31         // display Circle4's Diameter
32         output += "Diameter is " +
33            String.Format( "{0:F}", circle.Diameter() ) + "\n";
34
```

Change coordinates and radius of Circle4 object

Create new Circle4 object

Implicit call to Circle4's ToString method

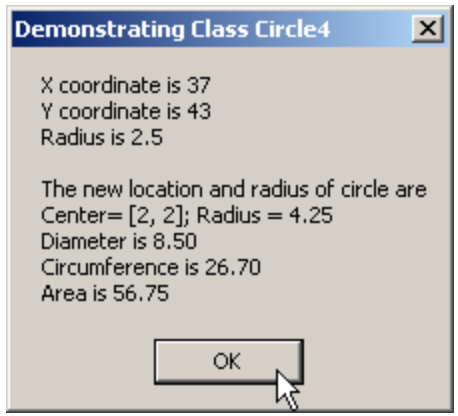


Outline

```
35         // display Circle4's Circumference
36         output += "Circumference is " +
37            String.Format( "{0:F}", circle.Circumference() ) + "\n";
38
39         // display Circle4's Area
40         output += "Area is " +
41            String.Format( "{0:F}", circle.Area() );
42
43         MessageBox.Show( output, "Demonstrating Class Circle4" );
44
45      } // end method Main
46
47   } // end class CircleTest4
```

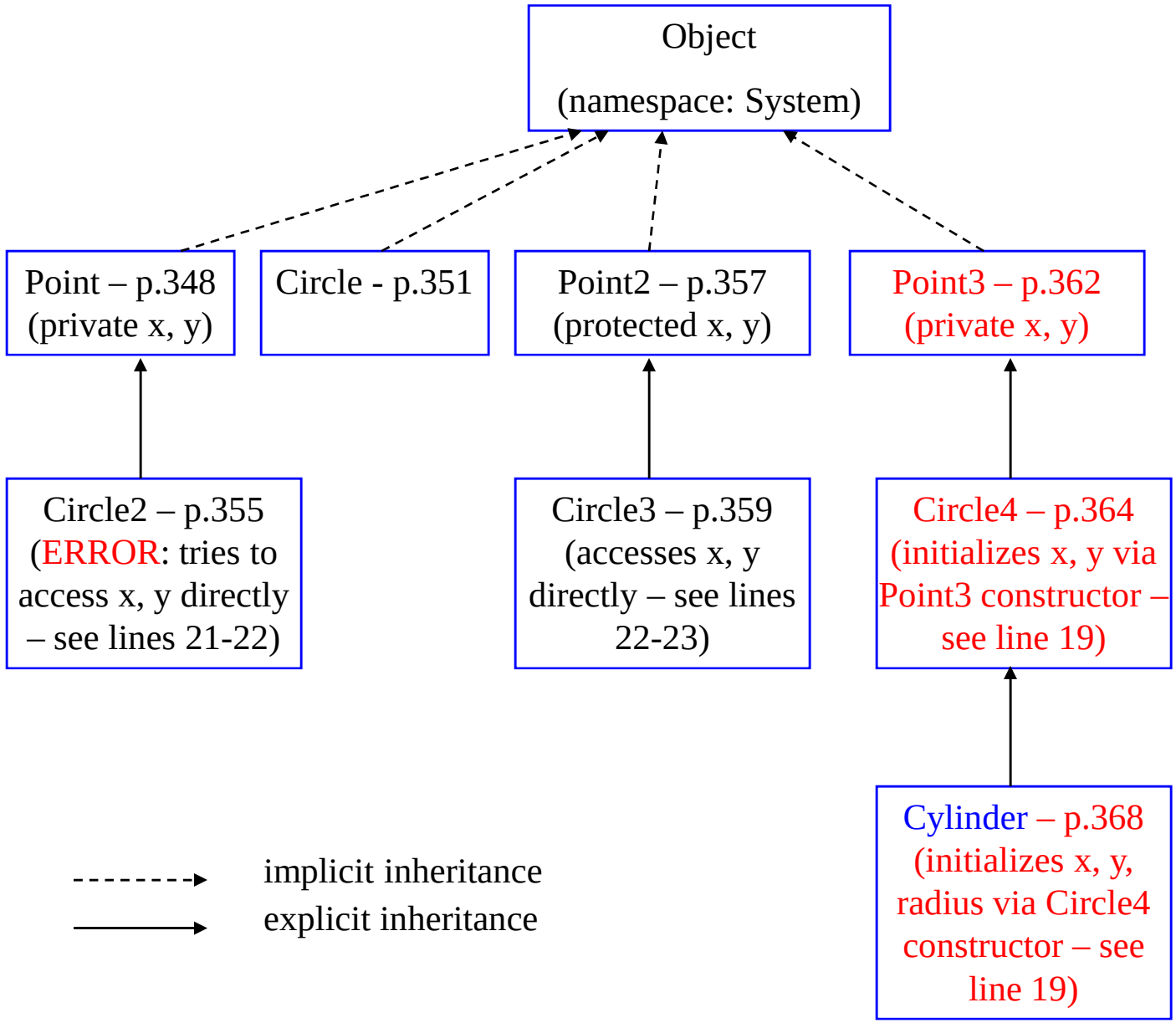
cs

Call Circle's Circumference and Area methods for output

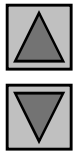


[9.6. in textbook ed. 1]
Case Study:
Three-Level Inheritance Hierarchy

- Three-level inheritance example:
  - Class **`Cylinder`** inherits from class **`Circle4`**
  - Class **`Circle4`** inherits from class **`Point3`**



Slide added by L. Lilien
Some slides modified by LL



Outline

Cylinder.cs

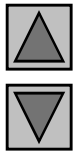
```
1    // Fig. 9.15: Cylinder.cs [textbook ed. 1]
2    // Cylinder class inherits from class Circle4.
3
4    using System;
5
6    // Cylinder class definition inherits from Circle4
7    public class Cylinder : Circle4
8    {
9       private double height;
10
11      // default constructor
12      public Cylinder()
13      {
14         // implicit call to Circle4 constructor occurs here
15      }
16
17      // four-argument constructor
18      public Cylinder( int xValue, int yValue, double radius
19         double heightValue ) : base( xValue, yValue, radiusValue )
20      {
21         Height = heightValue; // set Cylinder height
22      }
23
24      // property Height
25      public double Height
26      {
27         get
28         {
29            return height;
30         }
31
32         set
33         {
34            if ( value >= 0 ) // validate height
35               height = value;
```

Class Cylinder inherits from class Circle4

Declare variable height as private

Constructor that implicitly calls base class constructor

Constructor that explicitly calls base class constructor



Outline

Cylinder.cs

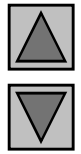
```
36          }
37
38       } // end property Height
39
40       // override Circle4 method Area to calculate Cylinder area
41       public override double Area()
42       {
43          return 2 * base.Area() + base.Circumference() * Height;
44       }
45
46       // calculate Cylinder volume
47       public double Volume()
48       {
49          return base.Area() * Height;
50       }
51
52       // convert Cylinder to string
53       public override string ToString()
54       {
55          return base.ToString() + "; Height = " + Height;
56       }
57
58    } // end class Cylinder
```

Method Area overrides Circle4's Area method (can override since Circle4 has `virtual` Area)

Calculate volume of cylinder

Overridden ToString method

Call Circle4's ToString method to get its output



Outline

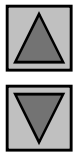
CylinderTest.cs

```
1  // Fig. 9.16: CylinderTest.cs [textbook ed. 1]
2  // Tests class Cylinder.
3
4  using System;
5  using System.Windows.Forms;
6
7  // CylinderTest class definition
8  class CylinderTest
9  {
10    // main entry point for application
11    static void Main( string[] args )
12    {
13       // instantiate object of class Cylinder
14       Cylinder cylinder = new Cylinder(12, 23, 2.5, 5.7);
15
16       // properties get initial x-y coordinate, radius and height
17       string output = "X coordinate is " + cylinder.X + "\n" +
18          "Y coordinate is " + cylinder.Y + "\nRadius is " +
19          cylinder.Radius + "\n" + "Height is " + cylinder.Height;
20
21       // properties set new x-y coordinate, radius and height
22       cylinder.X = 2;
23       cylinder.Y = 2;
24       cylinder.Radius = 4.25;
25       cylinder.Height = 10;
26
27       // get new x-y coordinate and radius
28       output += "\n\nThe new location, radius and height of " +
29          "cylinder are\n" + cylinder + "\n\n";
30
31       // display Cylinder's Diameter
32       output += "Diameter is " +
33          String.Format( "{0:F}", cylinder.Diameter() ) + "\n";
34
```

Create new cylinder

Change coordinates, radius and height

Implicit call to ToString

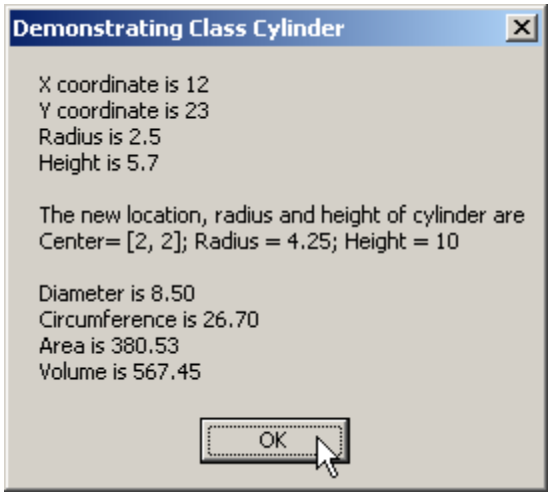


Outline

CylinderTest.cs

```
35          // display Cylinder's Circumference
36          output += "Circumference is " +
37             String.Format( "{0:F}", cylinder.Circumference() ) + "\n";
38
39          // display Cylinder's Area
40          output += "Area is " +
41             String.Format( "{0:F}", cylinder.Area() ) + "\n";
42
43          // display Cylinder's Volume
44          output += "Volume is " +
45             String.Format( "{0:F}", cylinder.Volume() );
46
47          MessageBox.Show( output, "Demonstrating Class Cylinder" );
48
49       } // end method Main
50
51    } // end class CylinderTest
```

Call methods Circumference, Area and Volume



Benefits of inheritance

- Using inheritance hierarchy in the above point-circle-cylinder example,

  we were able to develop Circle4 (inheriting from Point3), and Cylinder (inheriting from Circle4) much more quickly than

  if we had developed these classes from scratch

- Inheritance avoids duplicating code and associated code-maintenance problems
  - Such as the need to modify implementations all derived classes when the implementations of the base class changes

Slide added by L. Lilien
Some slides modified by LL

10.5. Constructors and Destructors in Derived Classes

- Instantiating a derived class DC, causes base-class constructor to be called, implicitly or explicitly
- When a constructor of a derived class DC is called, it invokes the derived class' base-class destructor first
  Only then it performs its own tasks
  - Causes chain reaction when a base class is also a derived class
  - The `Object` class constructor (at the top of the inheritance chain) is called last

  BUT

  The `Object` class constructor actually finishes first
  - The DC class constructor (called first) finishes last

- Consider example:

  Object - Point3 – Circle4 - Cylinder

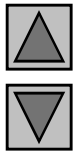
- When a destructor (or finalizer) of a derived class DC is called, it performs its own tasks first
  Then invokes the derived class' base-class destructor
  - Causes chain reaction when a base class is also a derived class
  - The DC class destructor (called first) finishes its tasks first
  - The `Object` class destructor (at the top of the inheritance chain) is called last

  AND

  The `Object` class destructor actually finishes its tasks last

- Consider example:
  Object - Point3 – Circle4 - Cylinder

- NOTE: unlike C or C++, C# performs memory management automatically => memory leaks are rare in C#
  - Memory mgmt via *garbage collection* of objects for which no references exist (p.312/1)



Outline

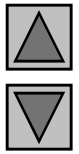
Point4.cs

```
1    // Fig. 9.17: Point4.cs [textbook ed. 1]
2    // Point4 class represents an x-y coordinate pair.
3
4    using System;
5
6    // Point4 class definition
7    public class Point4
8    {
9       // point coordinate
10      private int x, y;
11
12      // default constructor
13      public Point4()
14      {
15         // implicit call to Object constructor occurs here
16         Console.WriteLine( "Point4 constructor: {0}", this );
17      }
18
19      // constructor
20      public Point4( int xValue, int yValue )
21      {
22         // implicit call to Object constructor occurs here
23         X = xValue;
24         Y = yValue;
25         Console.WriteLine( "Point4 constructor: {0}", this );
26      }
27
28      // destructor
29      ~Point4()
30      {
31         Console.WriteLine( "Point4 destructor: {0}", this );
32      }
33
34      // property X
35      public int X
```

Constructors with output messages and implicit calls to base class constructor

Output statements use reference this to implicitly call ToString method

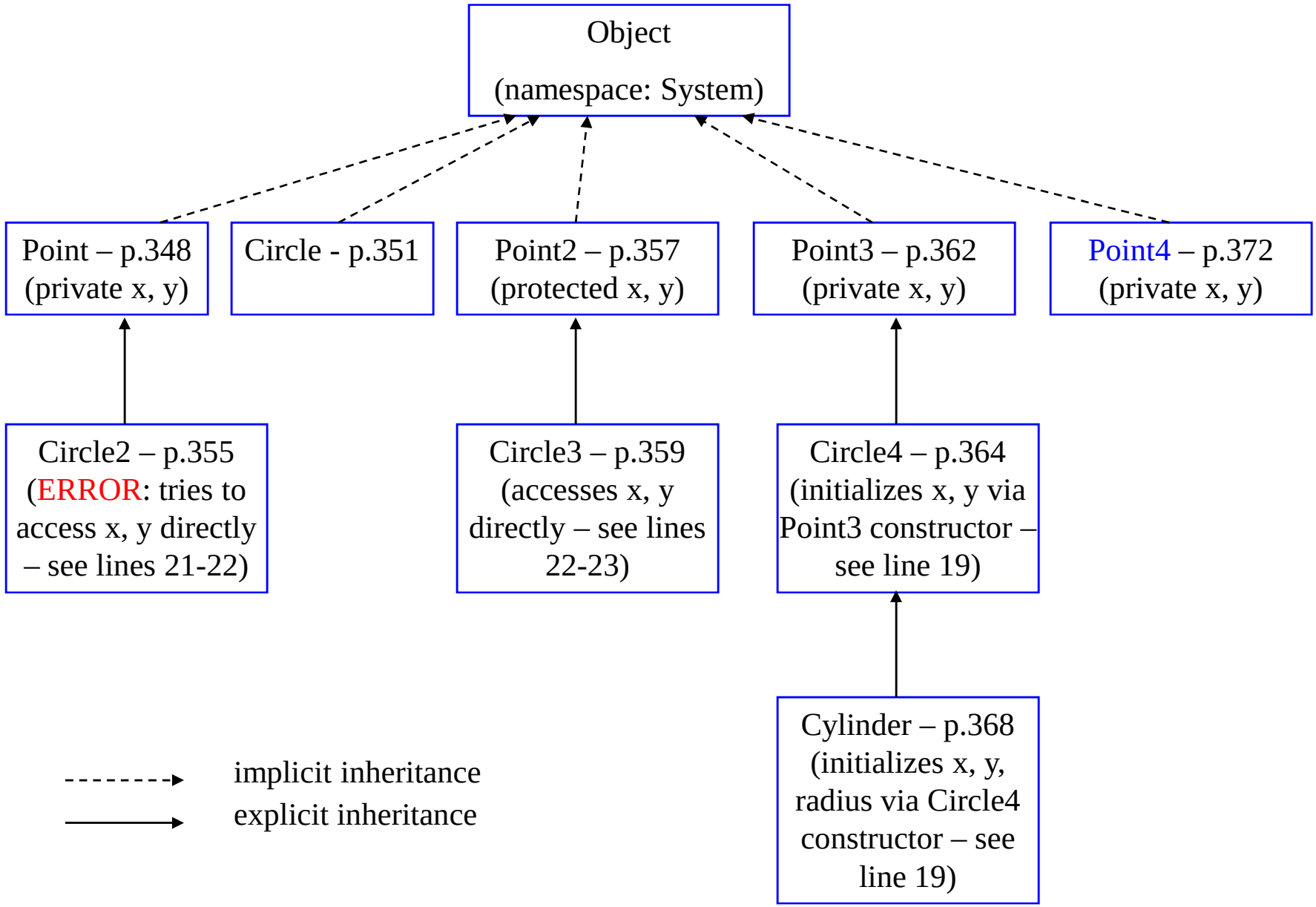
Destructor with output message



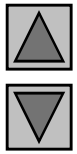
Outline

Point4.cs

```
36      {
37         get
38         {
39            return x;
40         }
41
42         set
43         {
44            x = value; // no need for validation
45         }
46
47      } // end property X
48
49      // property Y
50      public int Y
51      {
52         get
53         {
54            return y;
55         }
56
57         set
58         {
59            y = value; // no need for validation
60         }
61
62      } // end property Y
63
64      // return string representation of Point4
65      public override string ToString()
66      {
67         return "[" + x + ", " + y + "]";
68      }
69
70   } // end class Point4
```



Slide added by L. Lilien
Some slides modified by LL



Outline

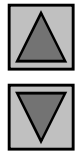
Circle5.cs

```
1    // Fig. 9.18: Circle5.cs [textbook ed. 1]
2    // Circle5 class that inherits from class Point4.
3
4    using System;
5
6    // Circle5 class definition inherits from Point4
7    public class Circle5 : Point4
8    {
9       private double radius;
10
11      // default constructor
12      public Circle5()
13      {
14         // implicit call to Point3 constructor occurs here
15         Console.WriteLine( "Circle5 constructor: {0}", this );
16      }
17
18      // constructor
19      public Circle5( int xValue, int yValue, double radiusValue )
20         : base( xValue, yValue )
21      {
22         Radius = radiusValue;
23         Console.WriteLine( "Circle5 constructor: {0}", this );
24      }
25
26      // destructor overrides version in class Point4
27      ~Circle5()
28      {
29         Console.WriteLine( "Circle5 destructor: {0}", this );
30      }
31
32      // property Radius
33      public double Radius
34      {
```

Constructors with implict and explicit (in turn) calls to base class and output statements

Output statements use reference this to implicitly call Circle5's ToString method

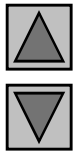
Destructor with output message



Outline

Circle5.cs

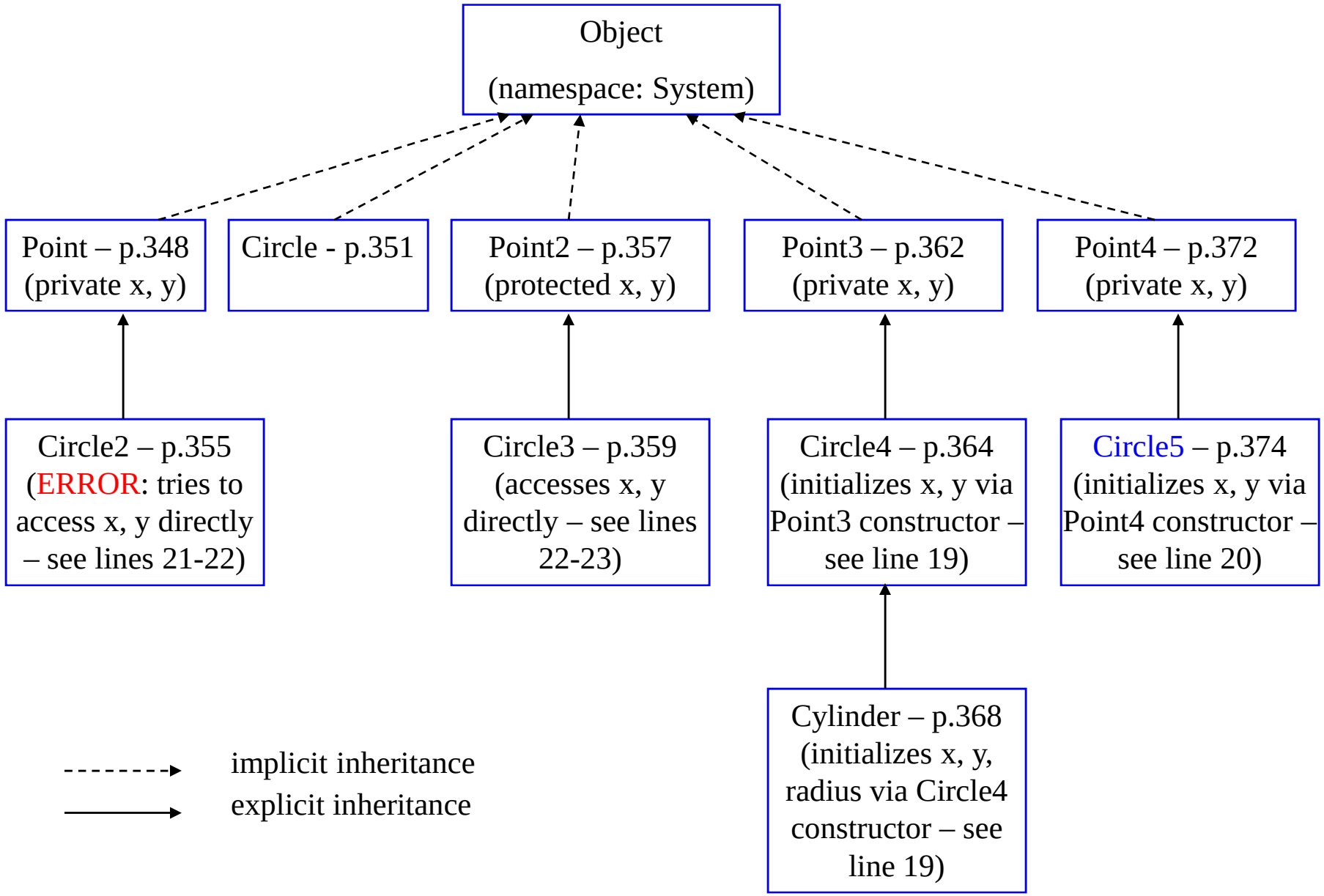
```
35          get
36          {
37             return radius;
38          }
39
40          set
41          {
42             if ( value >= 0 )
43                radius = value;
44          }
45
46       } // end property Radius
47
48       // calculate Circle5 diameter
49       public double Diameter()
50       {
51          return Radius * 2;
52       }
53
54       // calculate Circle5 circumference
55       public double Circumference()
56       {
57          return Math.PI * Diameter();
58       }
59
60       // calculate Circle5 area
61       public virtual double Area()
62       {
63          return Math.PI * Math.Pow( Radius, 2 );
64       }
65
66       // return string representation of Circle5
67       public override string ToString()
68       {
```



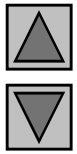
Outline

Circle5.cs

```
69          // use base reference to return Point3 string
70          return "Center = " + base.ToString() +
71             "; Radius = " + Radius;
72       }
73
74    } // end class Circle5
```



Slide added by L. Lilien
Some slides modified by LL



Outline

ConstructorAndDestructor.cs

```
1    // Fig. 9.19: ConstructorAndDestructor.cs [textbook ed. 1]
2    // Display order in which base-class and derived-class constructors
3    // and destructors are called.
4
5    using System;
6
7    // ConstructorAndFinalizer class definition
8    class ConstructorAndFinalizer
9    {
10      // main entry point for application.
11      static void Main( string[] args )
12      {
13         Circle5 circle1, circle2;  // Declares without initialization
14
15         // instantiate objects
16         circle1 = new Circle5( 72, 29, 4.5 );
17         circle2 = new Circle5( 5, 5, 10 );
18
19         Console.WriteLine(); // Output empty line
20
21         // mark objects for garbage collection
22         circle1 = null;
23         circle2 = null;
24
25         // inform garbage collector to execute
26         System.GC.Collect();
27
28      } // end method Main
29
30   } // end class ConstructorAndDestructor
```

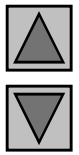
Create two objects of type Circle5

Remove references to Circle5 objects

Run the garbage collector

```
Point4 constructor: Center = [72, 29]; Radius = 0
Circle5 constructor: Center = [72, 29]; Radius = 4.5
Point4 constructor: Center = [5, 5]; Radius = 0
Circle5 constructor: Center = [5, 5]; Radius = 10

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



Outline

ConstructorAndDestructor.cs program output

[LL:] Notes:

1) Q: Why the text "; Radius = 0" is output by `Point 4` constructor in the 1st line?

   A: When constructing a `Circle5` object, the `this` reference used in the bodies of both `Circle5` *and* `Point4` constructors refers to the `Circle5` object being constructed. Hence, when `ToString` is invoked, the `Circle5.ToString` is used. (Yes, even within `Point4` constructor `Circle5.ToString` is used, not `Point4.ToString` !).

   `Circle5.ToString` always prints "; Radius = <value>"

   Since `Point4` constructor executes before `Circle5` constructor body, radius is not initialized by `Circle5` constructor yet. It has default value (zero) for `double`.

2) This also explains why similar text is output by other `Point4` constructors and destructors.

3) `Point4` constructor calls `Object` constructor *before* it executes its own body – not shown in output, since `Object` constructor prints nothing

4) `Point4` destructor calls `Object` destructor *after* it executes its own body – not shown in output, since `Object` destructor prints nothing

10.6. Software Engineering with Inheritance

- Derived classes inherit from the existing class
  - Member variables
  - Properties
  - Methods
- Can customize derived classes to meet needs by:
  - Creating new member variables
  - Creating new properties
  - Creating new methods
  - Overriding base-class members
- .NET Framework Class Library(FCL) allows full reuse of software through inheritance
  - Big benefits in software development (faster, more reliable, …)
- Read Section 10.6
- Read Section 10.8 Wrap-up(p.)

Some slides modified by LL

10.7. Class *object*

- All classes inherit (directly or not) from the object *class*
- 7 methods of the object *class* (inherited by all objects!)

See: http://msdn2.microsoft.com/enus/library/system.object_methods.aspx\

a) Public Methods

- Equals Overloaded. Determines whether two Object instances are equal.
- GetHashCode Serves as a hash function for a particular type.
- GetType Gets the Type of the current instance.
- ReferenceEquals Determines whether the specified **Object** instances are the same instance.
- ToString Returns a String that represents the current **Object**.

b) Protected Methods

- Finalize Allows an **Object** to attempt to free resources and perform other cleanup operations before the **Object** is reclaimed by garbage collection.
- MemberwiseClone Creates a shallow copy of the current **Object**.

The End

Some slides modified by LL