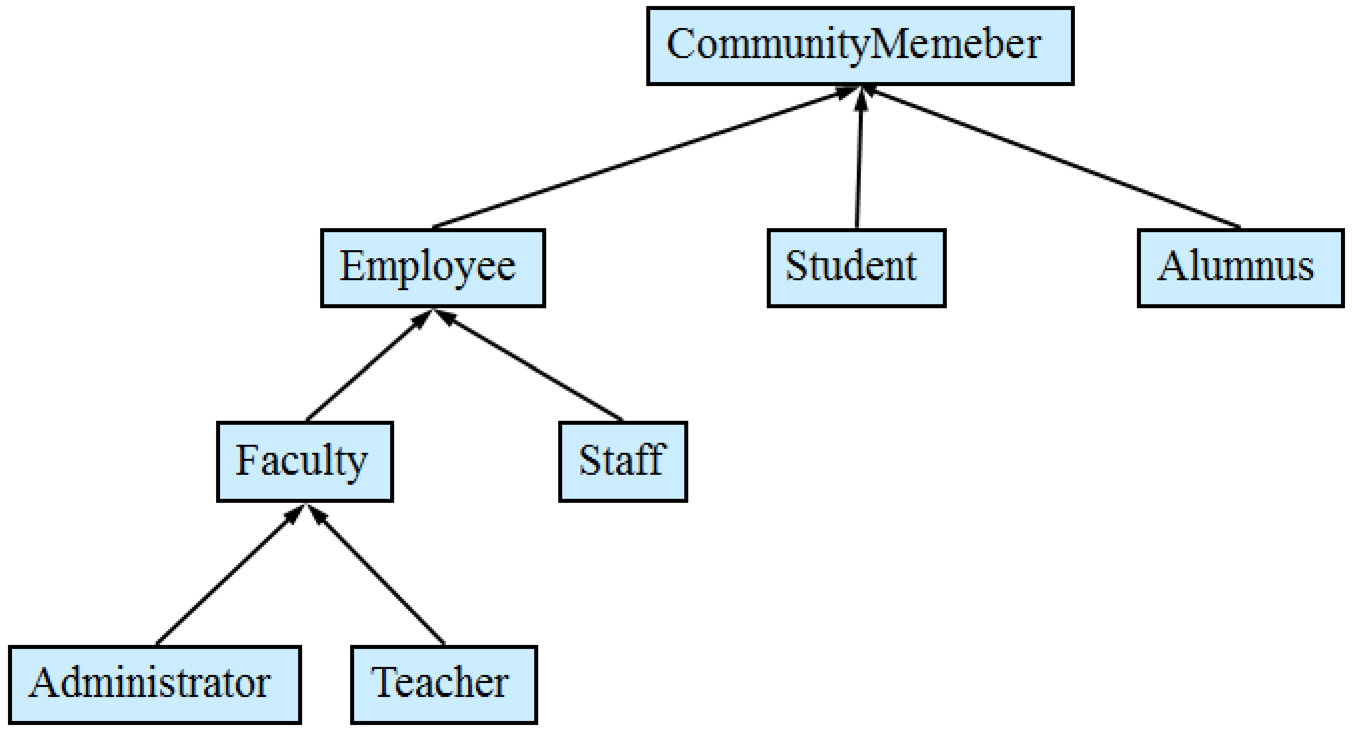
Chapter 9 – Object-Oriented Programming: Inheritance

Compiled by Prof. Fan Wu, MIS, CCU, Taiwan

9.1 Introduction

- Inheritance
  - New classes (also called derived class or subclass) created from existing classes (also called base class or supclass)
    - inherits data members and member functions from a previously defined base class

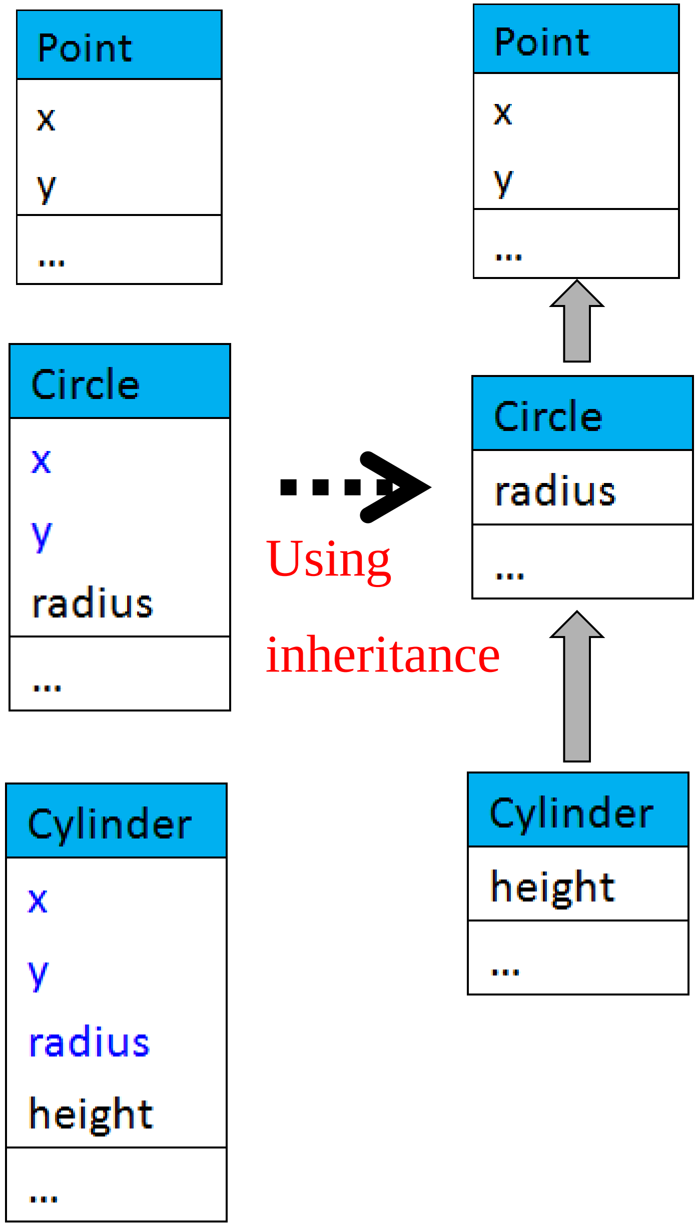


9.2 Base Classes and Derived Classes

- Objects of Base and derived classes
  - an object from a derived class (subclass) is also an object of a base class (superclass)
- derived class can only access non-private base class members

Case Study: Three-Level Inheritance Hierarchy

- Three level point/circle/cylinder hierarchy
  - Point
    - x-y coordinate pair
  - Circle
    - x-y coordinate pair,
    - radius
    - Skewed case: a circle with 0 radius will be a point
  - Cylinder
    - x-y coordinate pair,
    - radius,
    - height
    - Skewed case: a cylinder with 0 height will be a circle



Inheritance vs. Composition

- Inheritance
  - "Is a" relationship
    - Ex. Square is a rectangle
  - The class is created from existing one
    - It absorbs attributes and behaviors of its parents
- Cf. Composition
  - "Has a" relationship
    - Ex. A square has four sides and four angles
  - a class has more than one attributes through its contained object that is an object of other class

9.2 Base Classes and Derived Classes

Base class	Derived classes
`Student`	`GraduateStudent` `UndergraduateStudent`

`Shape`	`Circle`
	`Triangle`
	`Rectangle`

`Loan`	`CarLoan`
	`HomeImprovementLoan`
	`MortgageLoan`

`Employee`	`FacultyMember` `StaffMember`
`Account`	`CheckingAccount` `SavingsAccount`

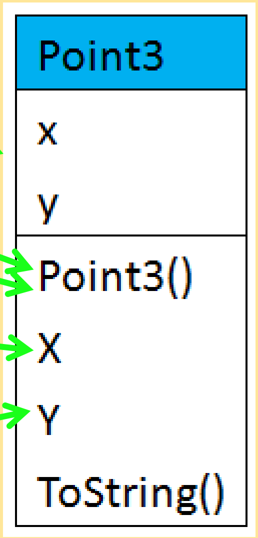
Member access specifiers

- Member access specifiers
  - Classes can limit the access to their member functions and data
  - The three types of access a class can grant are:
    - **Public**
      - 普遍級 *Accessible wherever the program has access to an object of the class*
    - **private**
      - 限制級 *Accessible only to member functions of the class*
    - **Protected**
      - 保護級 *Accessed only by subclass methods*

Methods in Base Classes and Derived Classes

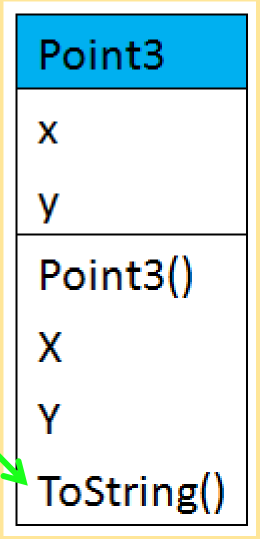
- Methods in base class can be inherited to derived class
- But the derived class can override its parent's method
  - override keyword is needed if a derived-class method overrides a base-class method
  - Overridden base class methods still can be accessed by "base" in the method of derived class
- Note: derived class can have its own constructor
  - The derived class first calls its base class' constructor, either explicitly or implicitly, then calls its own
    - "explicitly" is used specially when more than one constructors in its base class is defined
    - Otherwise, the base class's default constructor will be called implicitly

```
// Fig. 9.4: Point.cs
  using System;
  public class Point3   {
      private int x, y;
      public Point3()     {
      }
      public Point3( int xValue, int yValue )  {
          X = xValue;
          Y = yValue;
      }
      public int X   {
          get   {
              return x; }
          set {
              x = value; }
       }
      public int Y {
          get {
              return y; }
          set {
              y = value; }
      }
```



```
        public override string ToString()  {
            return "[" + X + ", " + Y + "]";
        }
    }
```

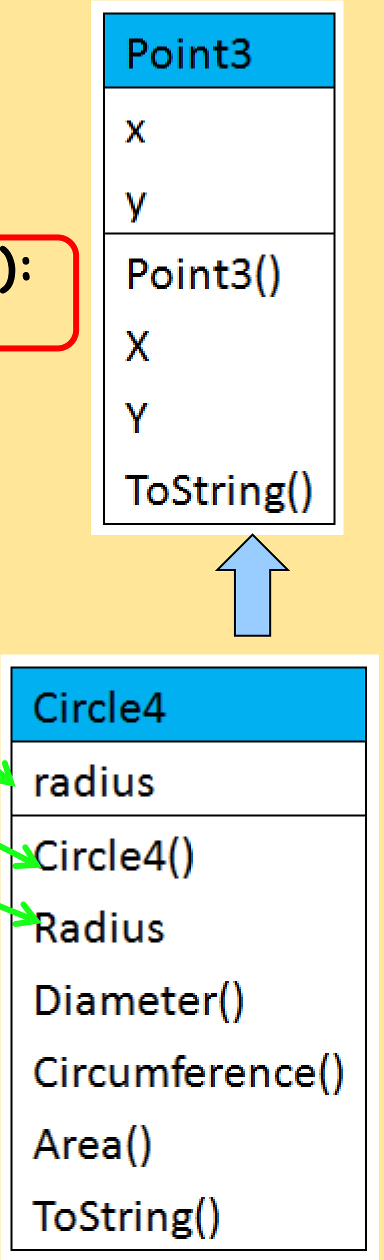
Override its base method (here the method did not call its base method)



```
  // Fig. 9.5: PointTest.cs
using System;
public class Circle4 : Point3 {
   private double radius;
   public Circle4() {
      // implicit call to Point constructor occurs here
   }
   public Circle4(int xValue, int yValue, double radiusValue ):
                 base( xValue, yValue) {
      Radius = radiusValue;
   }
   public double Radius  {
      get {
         return radius; }
      set {
         if ( value >= 0 )
            radius = value; }
   }
```

Inheritance syntax

Explicitly call its base constructor with two parameter

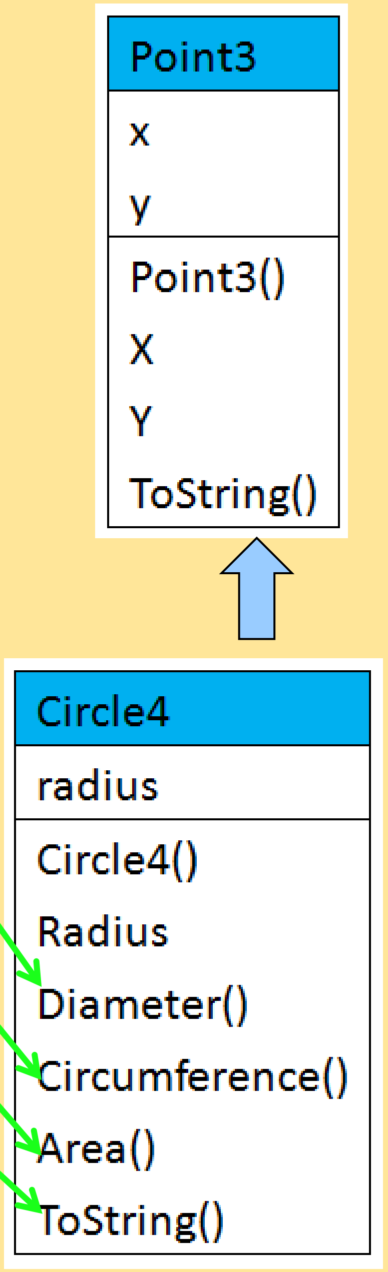


```
    public double Diameter()  {
       return Radius * 2;
    }
    public double Circumference() {
       return Math.PI * Diameter();
    }
    public virtual double Area() {
       return Math.PI * Math.Pow( Radius, 2 );
    }
```

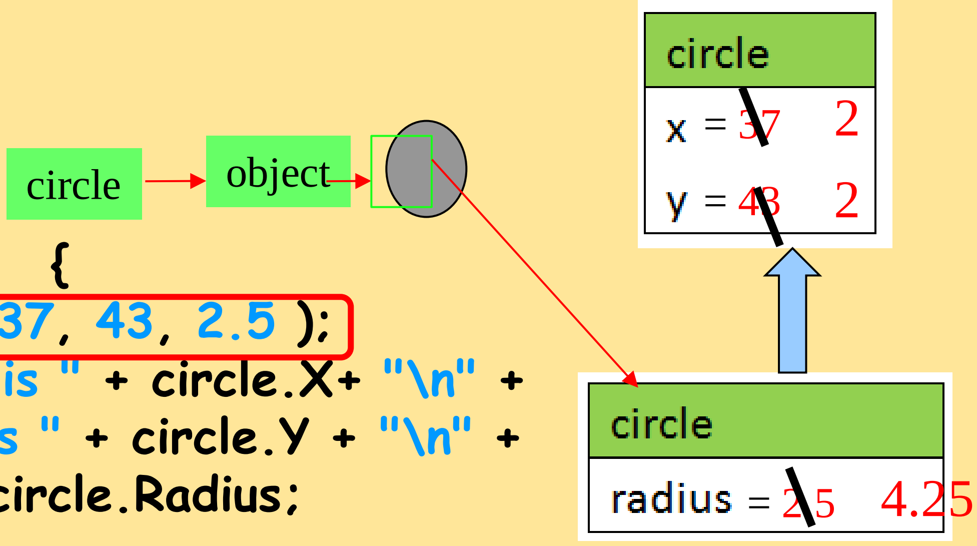
Override its base method

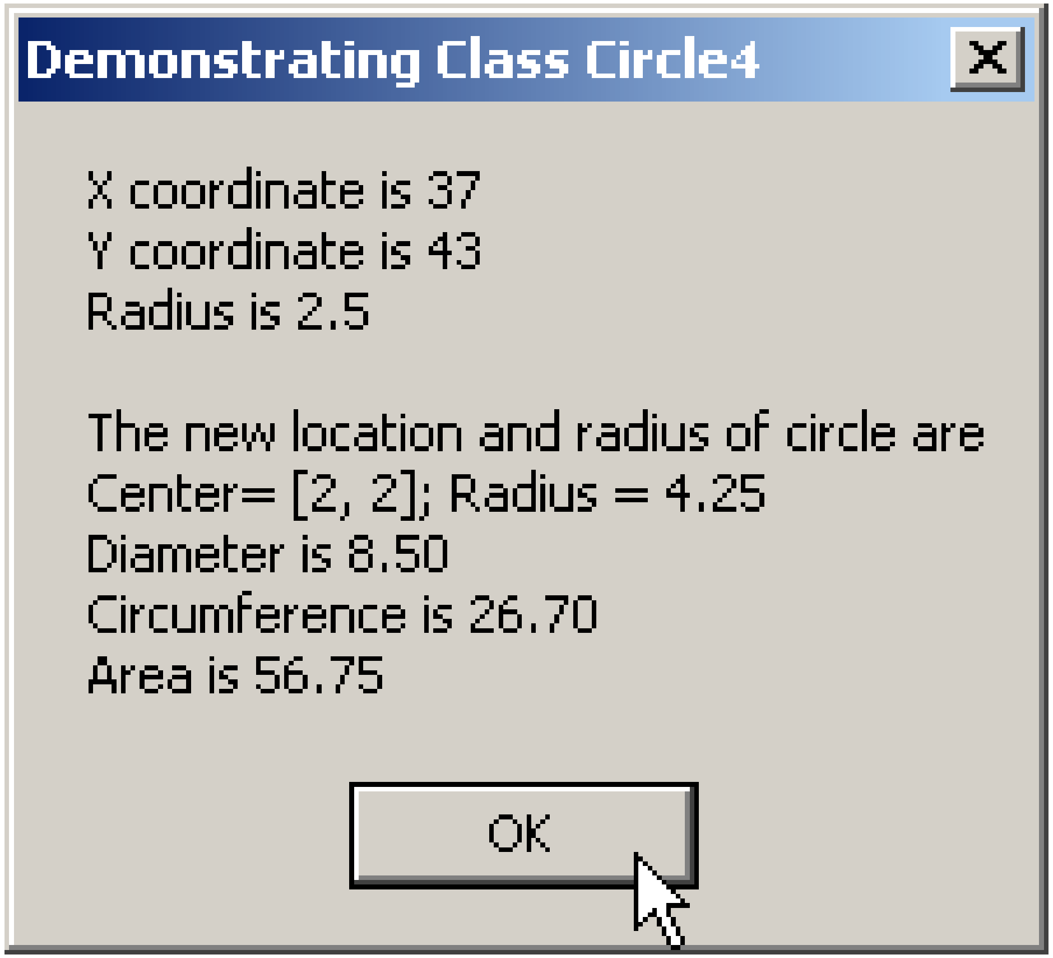
```
    public override string ToString() {
       return "Center= " + base.ToString() +
              "; Radius = " + Radius;
    }
}
```

Call its base method



```
// Fig. 9.14: CircleTest4.cs
using System;
using System.Windows.Forms;
class CircleTest4   {
   static void Main( string[] args )   {
      Circle4 circle = new Circle4( 37, 43, 2.5 );
      string output ="X coordinate is " + circle.X+ "\n" +
                     "Y coordinate is " + circle.Y + "\n" +
                     "Radius is " + circle.Radius;
      circle.X = 2;
      circle.Y = 2;
      circle.Radius = 4.25;
      output += "\n\n" +"The new location and radius of circle are " +
                "\n" + circle + "\n";
      output += "Diameter is " + String.Format( "{0:F}", circle.Diameter() )
                + "\n";
      output += "Circumference is " + String.Format( "{0:F}",
                 circle.Circumference() ) + "\n";
      output += "Area is " + String.Format( "{0:F}", circle.Area() );
      MessageBox.Show(output, "Demonstrating Class Circle4" );
   }
}
```





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

The new location and radius of circle are
Center= [2, 2]; Radius = 4.25
Diameter is 8.50
Circumference is 26.70
Area is 56.75
```

Constructors in Subclasses

- If instantiating a subclass object

  Step 1: Constructor is called from subclass

  Step 2: Then the constructor in subclass upward calls its superclass

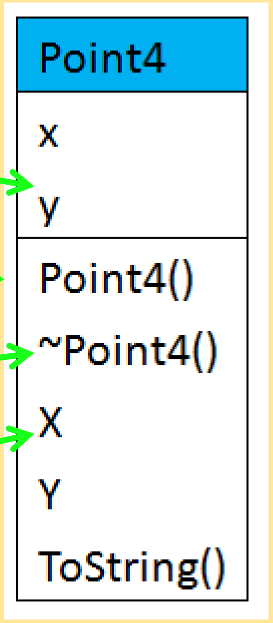
  - Implicitly, or explicitly (with base) calls its super class

- Note:
  - base, if used, must be the first line of the constructor of the subclass
  - The calls of constructors propagates until to the supreme super class
    - i.e., original subclass constructor's finishes its execution last

Destructors in Derived Classes

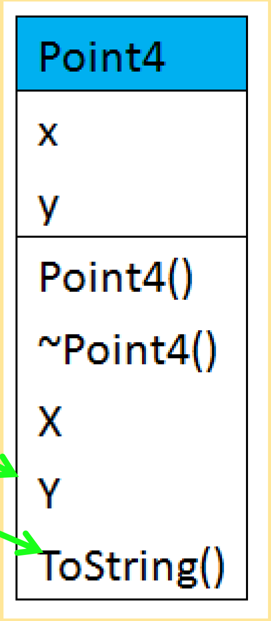
- **Destructor** method
  - Garbage collection
    - Return the memory allocated to the instance to OS
  - When subclass's destructor method is called
    - It should then invoke superclass's **finalize** method
- Chain of Destructor method calls
  - subclass's destructor finishes its execution before superclass's f destructor
  - After supreme superclass (Object) finalizer, the instance is removed from memory
- Cf: superclass's constructor finishes execution before subclass's constructor

```
// Fig. 9.17: Point4.cs
using System;
public class Point4 {
    private int x, y;
    public Point4() {
        Console.WriteLine( "Point4 constructor: {0}", this );
    }
    public Point4( int xValue, int yValue ) {
        X = xValue;
        Y = yValue;
        Console.WriteLine( "Point4 constructor: {0}", this );
    }
    ~Point4() {
        Console.WriteLine( "Point4 destructor: {0}", this );
    }
    public int X {
        get  {
            return x; }
        set  {
            x = value; }
    }
```

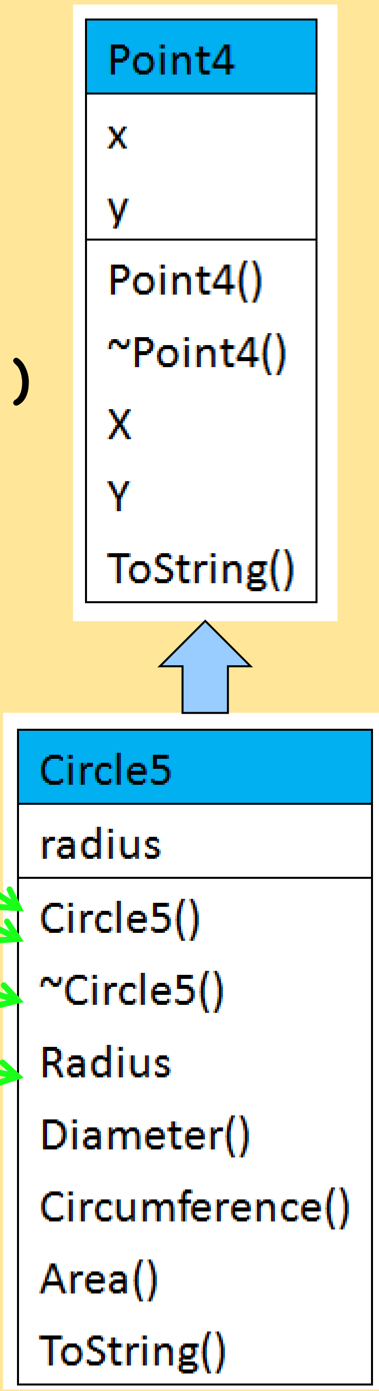


Call its ToString() if appearing in writeln()

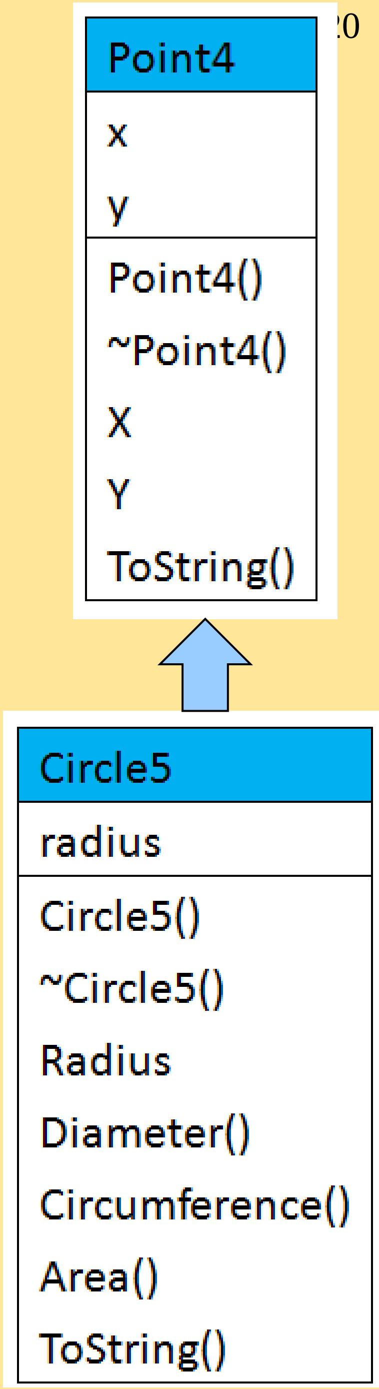
```
    public int Y {
        get  {
            return y; }
        set {
            y = value; }
    }
    public override string ToString() {
        return "[" + x + ", " + y + "]";
    }
}
```



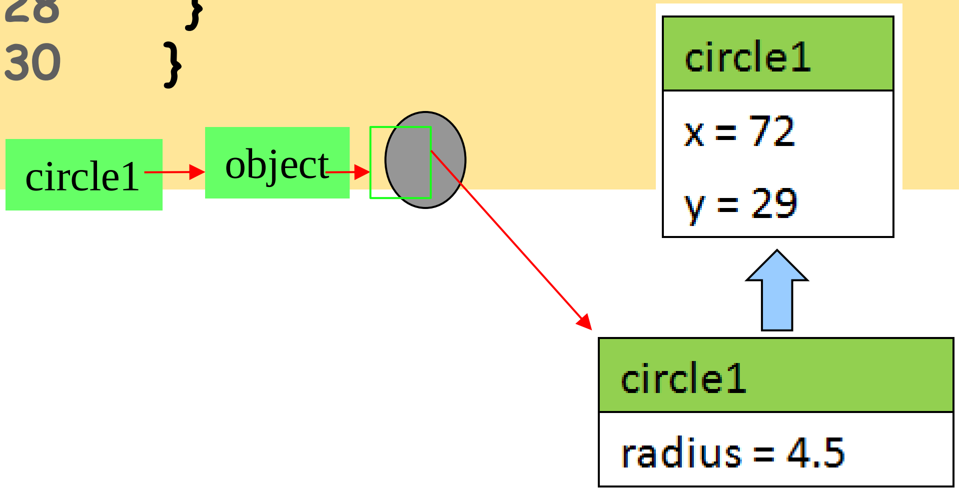
```
using System;
public class Circle5 : Point4 {
  private double radius;
   public Circle5() {
      Console.WriteLine( "Circle5 constructor: {0}", this );
   }
   public Circle5( int xValue, int yValue, double radiusValue )
      : base( xValue, yValue ) {
      Radius = radiusValue;
      Console.WriteLine( "Circle5 constructor: {0}", this );
   }
   ~Circle5() {
      Console.WriteLine( "Circle5 destructor: {0}", this );
   }
   public double Radius  {
      get {
         return radius; }
      set {
         if ( value >= 0 )
            radius = value; }
   }
```

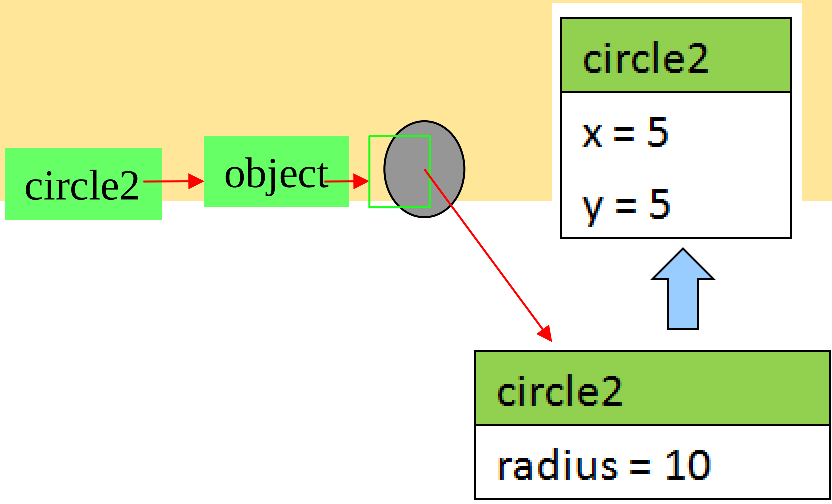


```
    public double Diameter() {
        return Radius * 2;
    }
    public double Circumference() {
        return Math.PI * Diameter();
    }
    public virtual double Area() {
        return Math.PI * Math.Pow( Radius, 2 );
    }
    public override string ToString() {
        return "Center = " + base.ToString() +
               "; Radius = " + Radius;
    }
}
```

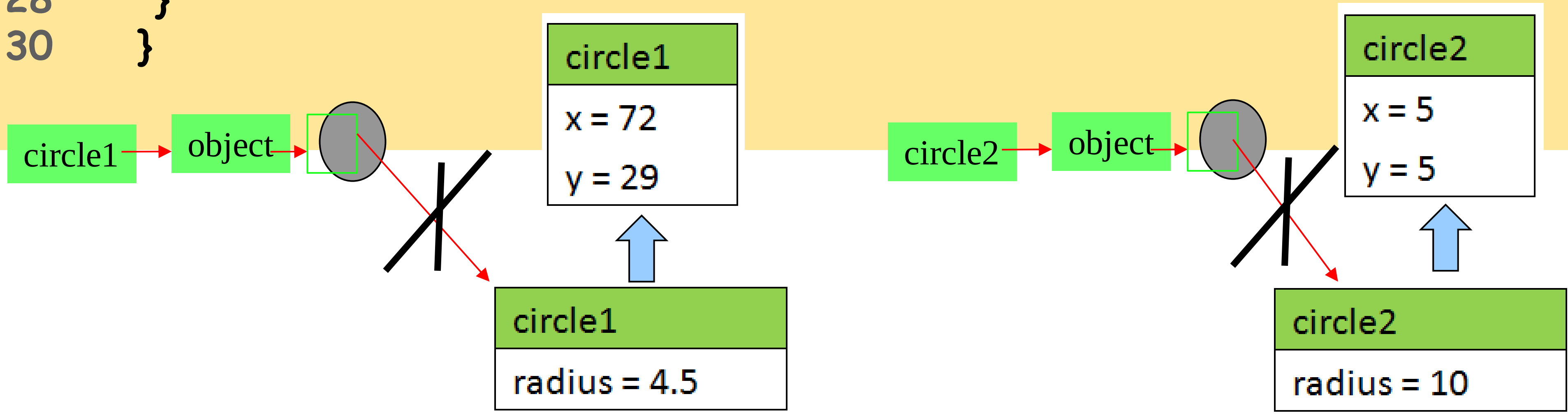


```
// Fig. 9.19: ConstructorAndDestructor.cs
using System;
class ConstructorAndFinalizer  {
   static void Main( string[] args ) {
      Circle5 circle1, circle2;
      circle1 = new Circle5( 72, 29, 4.5 );
      circle2 = new Circle5( 5, 5, 10 );
      Console.WriteLine();
      circle1 = null;
      circle2 = null;
      System.GC.Collect();
   }
}
```





```
// Fig. 9.19: ConstructorAndDestructor.cs
using System;
class ConstructorAndFinalizer  {
   static void Main( string[] args ) {
      Circle5 circle1, circle2;
      circle1  = new Circle5( 72, 29, 4.5 );
      circle2  = new Circle5( 5, 5, 10 );
      Console.WriteLine();
      circle1  = null;
      circle2  = null;
      System.GC.Collect();
   }
}
```



```csharp
// Fig. 9.19: ConstructorAndDestructor.cs
using System;
class ConstructorAndFinalizer  {
   static void Main( string[] args ) {
      Circle5 circle1, circle2;
      circle1 = new Circle5( 72, 29, 4.5 );
      circle2 = new Circle5( 5, 5, 10 );
      Console.WriteLine();
      circle1 = null;
      circle2 = null;
      System.GC.Collect();
   }
  }
```

```
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
```