Superclass	Subclasses
`Student`	`GraduateStudent` `UndergraduateStudent`

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

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

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

Fig. 9.1 Some simple inheritance examples .

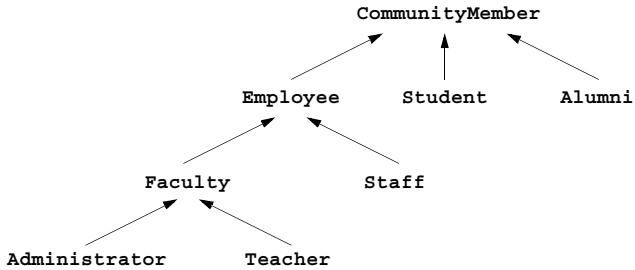

Fig. 9.2 An inheritance hierarchy for university **`CommunityMember`**s.

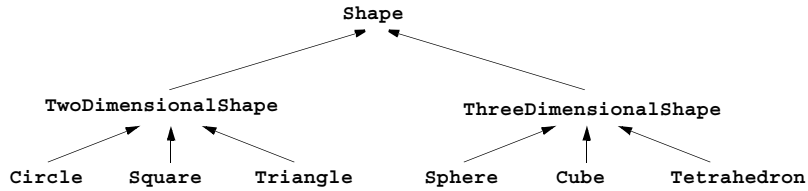

Fig. 9.3 A portion of a **Shape** class hierarchy.

```
1  // Fig. 9.4: Point.java
2  // Definition of class Point
3
4  public class Point {
5     protected int x, y; // coordinates of the Point
6
7     // No-argument constructor
8     public Point()
9     {
10       // implicit call to superclass constructor occurs here
11       setPoint( 0, 0 );
12    }
13
14    // Constructor
15    public Point( int a, int b )
16    {
17       // implicit call to superclass constructor occurs here
18       setPoint( a, b );
19    }
20
21    // Set x and y coordinates of Point
22    public void setPoint( int a, int b )
23    {
24       x = a;
25       y = b;
26    }
27
28    // get x coordinate
29    public int getX() { return x; }
30
31    // get y coordinate
32    public int getY() { return y; }
33
34    // convert the point into a String representation
35    public String toString()
36       { return "[" + x + ", " + y + "]"; }
37 }
```

Fig. 9.4 Assigning subclass references to superclass references (part 1 of 5).

```
38 // Fig. 9.4: Circle.java
39 // Definition of class Circle
40
41 public class Circle extends Point {  // inherits from Point
42    protected double radius;
43
44    // No-argument constructor
45    public Circle()
46    {
47       // implicit call to superclass constructor occurs here
48       setRadius( 0 );
49    }
```

Fig. 9.4 Assigning subclass references to superclass references (part 2 of 5).

```
50
51    // Constructor
52    public Circle( double r, int a, int b )
53    {
54       super( a, b );  // call to superclass constructor
55       setRadius( r );
56    }
57
58    // Set radius of Circle
59    public void setRadius( double r )
60       { radius = ( r >= 0.0 ? r : 0.0 ); }
61
62    // Get radius of Circle
63    public double getRadius() { return radius; }
64
65    // Calculate area of Circle
66    public double area() { return Math.PI * radius * radius; }
67
68    // convert the Circle to a String
69    public String toString()
70    {
71       return "Center = " + "[" + x + ", " + y + "]" +
72              "; Radius = " + radius;
73    }
74 }
```

Fig. 9.4 Assigning subclass references to superclass references (part 3 of 5).

```
// Fig. 9.4: Test.java
// Demonstrating the "is a" relationship
import java.text.DecimalFormat;
import javax.swing.JOptionPane;

public class InheritanceTest {
   public static void main( String args[] )
   {
      Point pointRef, p;
      Circle circleRef, c;
      String output;

      p = new Point( 30, 50 );
      c = new Circle( 2.7, 120, 89 );

      output = "Point p: " + p.toString() +
               "\nCircle c: " + c.toString();

      // use the "is a" relationship to refer to a Circle
      // with a Point reference
      pointRef = c;   // assign Circle to pointRef

      output += "\n\nCircle c (via pointRef): " +
                pointRef.toString();
```

Fig. 9.4 Assigning subclass references to superclass references (part 4 of 5).

```

      // Use downcasting (casting a superclass reference to a
      // subclass data type) to assign pointRef to circleRef
      circleRef = (Circle) pointRef;

      output += "\n\nCircle c (via circleRef): " +
                circleRef.toString();

      DecimalFormat precision2 = new DecimalFormat( "0.00" );
      output += "\nArea of c (via circleRef): " +
                precision2.format( circleRef.area() );

      // Attempt to refer to Point object
      // with Circle reference
      if ( p instanceof Circle ) {
         circleRef = (Circle) p;  // line 40 in Test.java
         output += "\n\ncast successful";
      }
      else
         output += "\n\np does not refer to a Circle";

      JOptionPane.showMessageDialog( null, output,
         "Demonstrating the \"is a\" relationship",
         JOptionPane.INFORMATION_MESSAGE );

      System.exit( 0 );
   }
}
```

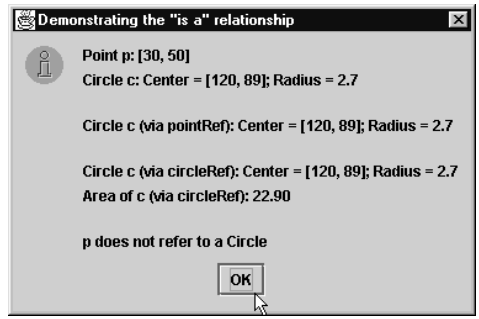

Fig. 9.4 Assigning subclass references to superclass references (part 5 of 5).

```
1  // Fig. 9.5: Point.java
2  // Definition of class Point
3  public class Point extends Object {
4     protected int x, y; // coordinates of the Point
5
6     // no-argument constructor
7     public Point()
8     {
9        x = 0;
10       y = 0;
11       System.out.println( "Point constructor: " + this );
12    }
13
14    // constructor
15    public Point( int a, int b )
16    {
17       x = a;
18       y = b;
19       System.out.println( "Point constructor: " + this );
20    }
21
22    // finalizer
23    protected void finalize()
24    {
25       System.out.println( "Point finalizer: " + this );
26    }
27
28    // convert the point into a String representation
29    public String toString()
30       { return "[" + x + ", " + y + "]"; }
31 }
```

Fig. 9.5 Order in which constructors and finalizers are called (part 1 of 4).

```
// Fig. 9.5: Circle.java
// Definition of class Circle
public class Circle extends Point {  // inherits from Point
   protected double radius;

   // no-argument constructor
   public Circle()
   {
      // implicit call to superclass constructor here
      radius = 0;
      System.out.println( "Circle constructor: " + this );
   }

   // Constructor
   public Circle( double r, int a, int b )
   {
      super( a, b );  // call the superclass constructor
      radius = r;
```

Fig. 9.5 Order in which constructors and finalizers are called (part 2 of 4).

```
      System.out.println( "Circle constructor: " + this );
   }

   // finalizer
   protected void finalize()
   {
      System.out.println( "Circle finalizer: " + this );
      super.finalize();  // call superclass finalize method
   }

   // convert the Circle to a String
   public String toString()
   {
      return "Center = " + super.toString() +
             "; Radius = " + radius;
   }
}
```

Fig. 9.5 Order in which constructors and finalizers are called (part 3 of 4).

```
// Fig. 9.5: Test.java
// Demonstrate when superclass and subclass
// constructors and finalizers are called.
public class Test {
   public static void main( String args[] )
   {
      Circle circle1, circle2;

      circle1 = new Circle( 4.5, 72, 29 );
      circle2 = new Circle( 10, 5, 5 );

      circle1 = null;  // mark for garbage collection
      circle2 = null;  // mark for garbage collection

      System.gc();     // call the garbage collector
   }
}
```

```
Point constructor: Center = [72, 29]; Radius = 0.0
Circle constructor: Center = [72, 29]; Radius = 4.5
Point constructor: Center = [5, 5]; Radius = 0.0
Circle constructor: Center = [5, 5]; Radius = 10.0
Circle finalizer: Center = [72, 29]; Radius = 4.5
Point finalizer: Center = [72, 29]; Radius = 4.5
Circle finalizer: Center = [5, 5]; Radius = 10.0
Point finalizer: Center = [5, 5]; Radius = 10.0
```

Fig. 9.5 Order in which constructors and finalizers are called (part 4 of 4).

```
1  // Fig. 9.6: Point.java
2  // Definition of class Point
3  package com.deitel.jhtp3.ch09;
4
5  public class Point {
6     protected int x, y; // coordinates of the Point
7
8     // no-argument constructor
9     public Point() { setPoint( 0, 0 ); }
10
11    // constructor
12    public Point( int a, int b ) { setPoint( a, b ); }
13
14    // Set x and y coordinates of Point
15    public void setPoint( int a, int b )
16    {
17       x = a;
18       y = b;
19    }
20
21    // get x coordinate
22    public int getX() { return x; }
23
24    // get y coordinate
25    public int getY() { return y; }
26
27    // convert the point into a String representation
28    public String toString()
29       { return "[" + x + ", " + y + "]"; }
30 }
```

Fig. 9.6 Testing class **`Point`** (part 1 of 3).

```
31 // Fig. 9.6: Test.java
32 // Applet to test class Point
33 import javax.swing.JOptionPane;
34 import com.deitel.jhtp3.ch09.Point;
35
36 public class Test {
37    public static void main( String args[] )
38    {
39       Point p = new Point( 72, 115 );
```

Fig. 9.6 Testing class **Point** (part 2 of 3).

```
40       String output;
41
42       output = "X coordinate is " + p.getX() +
43                "\nY coordinate is " + p.getY();
44
45       p.setPoint( 10, 10 );
46
47       // use implicit call to p.toString()
48       output += "\n\nThe new location of p is " + p;
49
50       JOptionPane.showMessageDialog( null, output,
51          "Demonstrating Class Point",
52          JOptionPane.INFORMATION_MESSAGE );
53       System.exit( 0 );
54    }
55 }
```

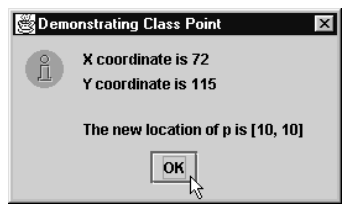

Fig. 9.6 Testing class **Point** (part 3 of 3).

```
1  // Fig. 9.7: Circle.java
2  // Definition of class Circle
3  package com.deitel.jhtp3.ch09;
4
5  public class Circle extends Point {  // inherits from Point
6     protected double radius;
7
8     // no-argument constructor
9     public Circle()
10    {
11       // implicit call to superclass constructor
12       setRadius( 0 );
13    }
14
```

Fig. 9.7 Testing class **Circle** (part 1 of 4).

```
15    // Constructor
16    public Circle( double r, int a, int b )
17    {
18       super( a, b );  // call the superclass constructor
19       setRadius( r );
20    }
21
22    // Set radius of Circle
23    public void setRadius( double r )
24       { radius = ( r >= 0.0 ? r : 0.0 ); }
25
26    // Get radius of Circle
27    public double getRadius() { return radius; }
28
29    // Calculate area of Circle
30    public double area()
31       { return Math.PI * radius * radius; }
32
33    // convert the Circle to a String
34    public String toString()
35    {
36       return "Center = " + super.toString() +
37              "; Radius = " + radius;
38    }
39 }
```

Fig. 9.7 Testing class **Circle** (part 2 of 4).

```
40 // Fig. 9.7: Test.java
41 // Applet to test class Circle
42 import javax.swing.JOptionPane;
43 import java.text.DecimalFormat;
44 import com.deitel.jhtp3.ch09.Circle;
45
46 public class Test {
47    public static void main( String args[] )
48    {
49       Circle c = new Circle( 2.5, 37, 43 );
50       DecimalFormat precision2 = new DecimalFormat( "0.00" );
51       String output;
52
53       output = "X coordinate is " + c.getX() +
54                "\nY coordinate is " + c.getY() +
55                "\nRadius is " + c.getRadius();
56
57       c.setRadius( 4.25 );
58       c.setPoint( 2, 2 );
59       output +=
60          "\n\nThe new location and radius of c are\n" + c +
61          "\nArea is " + precision2.format( c.area() );
62
```

Fig. 9.7 Testing class **`Circle`** (part 3 of 4).

```
63       JOptionPane.showMessageDialog( null, output,
64          "Demonstrating Class Circle",
65          JOptionPane.INFORMATION_MESSAGE );
66       System.exit( 0 );
67    }
68 }
```

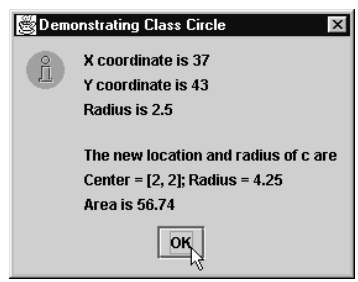

Fig. 9.7 Testing class **`Circle`** (part 4 of 4).

```
1  // Fig. 9.8: Cylinder.java
2  // Definition of class Cylinder
3  package com.deitel.jhtp3.ch09;
4
5  public class Cylinder extends Circle {
6     protected double height;  // height of Cylinder
7
8     // No-argument constructor
9     public Cylinder()
10    {
11       // implicit call to superclass constructor here
12       setHeight( 0 );
13    }
14
15    // constructor
16    public Cylinder( double h, double r, int a, int b )
17    {
18       super( r, a, b );
19       setHeight( h );
20    }
21
22    // Set height of Cylinder
23    public void setHeight( double h )
24       { height = ( h >= 0 ? h : 0 ); }
25
26    // Get height of Cylinder
27    public double getHeight() { return height; }
28
29    // Calculate area of Cylinder (i.e., surface area)
30    public double area()
31    {
32       return 2 * super.area() +
33              2 * Math.PI * radius * height;
34    }
35
36    // Calculate volume of Cylinder
37    public double volume() { return super.area() * height; }
38
39    // Convert the Cylinder to a String
40    public String toString()
41    {
42       return super.toString() + "; Height = " + height;
43    }
44 }
```

Fig. 9.8 Testing class **`Cylinder`** (part 1 of 3).

```
45 // Fig. 9.8: Test.java
46 // Application to test class Cylinder
47 import javax.swing.JOptionPane;
48 import java.text.DecimalFormat;
49 import com.deitel.jhtp3.ch09.Cylinder;
```

Fig. 9.8 Testing class **Cylinder** (part 2 of 3).

```
50
51 public class Test {
52    public static void main( String args[] )
53    {
54       Cylinder c = new Cylinder( 5.7, 2.5, 12, 23 );
55       DecimalFormat precision2 = new DecimalFormat( "0.00" );
56       String output;
57
58       output = "X coordinate is " + c.getX() +
59                "\nY coordinate is " + c.getY() +
60                "\nRadius is " + c.getRadius() +
61                "\nHeight is " + c.getHeight();
62
63       c.setHeight( 10 );
64       c.setRadius( 4.25 );
65       c.setPoint( 2, 2 );
66
67       output +=
68          "\n\nThe new location, radius " +
69          "and height of c are\n" + c +
70          "\nArea is " + precision2.format( c.area() ) +
71          "\nVolume is " + precision2.format( c.volume() );
72
73       JOptionPane.showMessageDialog( null, output,
74          "Demonstrating Class Cylinder",
75          JOptionPane.INFORMATION_MESSAGE );
76       System.exit( 0 );
77    }
78 }
```

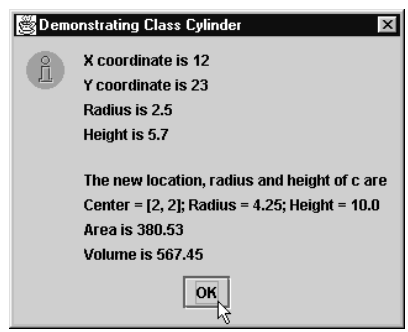

Fig. 9.8 Testing class **Cylinder** (part 3 of 3).

```
1  // Fig. 9.9: Employee.java
2  // Abstract base class Employee
3
4  public abstract class Employee {
5     private String firstName;
6     private String lastName;
7
8     // Constructor
9     public Employee( String first, String last )
10    {
11       firstName = first;
12       lastName = last;
13    }
14
15    // Return the first name
16    public String getFirstName() { return firstName; }
17
18    // Return the last name
19    public String getLastName() { return lastName; }
20
21    public String toString()
22       { return firstName + ' ' + lastName; }
23
24    // Abstract method that must be implemented for each
25    // derived class of Employee from which objects
26    // are instantiated.
27    public abstract double earnings();
28 }
```

Fig. 9.9 **`Employee`** class hierarchy using an **`abstract`** superclass (part 1 of 10).

```
29 // Fig. 9.9: Boss.java
30 // Boss class derived from Employee
31
32 public final class Boss extends Employee {
33    private double weeklySalary;
34
35    // Constructor for class Boss
36    public Boss( String first, String last, double s)
37    {
38       super( first, last );  // call superclass constructor
39       setWeeklySalary( s );
40    }
41
42    // Set the Boss's salary
43    public void setWeeklySalary( double s )
44       { weeklySalary = ( s > 0 ? s : 0 ); }
45
46    // Get the Boss's pay
47    public double earnings() { return weeklySalary; }
48
49    // Print the Boss's name
50    public String toString()
51    {
52       return "Boss: " + super.toString();
53    }
54 }
```

Fig. 9.9 **Employee** class hierarchy using an **abstract** superclass (part 2 of 10).

```
// Fig. 9.9: CommissionWorker.java
// CommissionWorker class derived from Employee

public final class CommissionWorker extends Employee {
   private double salary;       // base salary per week
   private double commission;   // amount per item sold
   private int quantity;        // total items sold for week

   // Constructor for class CommissionWorker
   public CommissionWorker( String first, String last,
                            double s, double c, int q)
   {
      super( first, last );  // call superclass constructor
      setSalary( s );
      setCommission( c );
      setQuantity( q );
   }

   // Set CommissionWorker's weekly base salary
   public void setSalary( double s )
      { salary = ( s > 0 ? s : 0 ); }

```

Fig. 9.9 **`Employee`** class hierarchy using an **`abstract`** superclass (part 3 of 10).

```
   // Set CommissionWorker's commission
   public void setCommission( double c )
      { commission = ( c > 0 ? c : 0 ); }

   // Set CommissionWorker's quantity sold
   public void setQuantity( int q )
      { quantity = ( q > 0 ? q : 0 ); }

   // Determine CommissionWorker's earnings
   public double earnings()
      { return salary + commission * quantity; }

   // Print the CommissionWorker's name
   public String toString()
   {
      return "Commission worker: " + super.toString();
   }
}
```

Fig. 9.9 **`Employee`** class hierarchy using an **`abstract`** superclass (part 4 of 10).

```
95  // Fig. 9.9: PieceWorker.java
96  // PieceWorker class derived from Employee
97
98  public final class PieceWorker extends Employee {
99     private double wagePerPiece; // wage per piece output
100    private int quantity;        // output for week
101
102    // Constructor for class PieceWorker
103    public PieceWorker( String first, String last,
104                        double w, int q )
105    {
106       super( first, last );  // call superclass constructor
107       setWage( w );
108       setQuantity( q );
109    }
110
111    // Set the wage
112    public void setWage( double w )
113       { wagePerPiece = ( w > 0 ? w : 0 ); }
114
```

Fig. 9.9 **Employee** class hierarchy using an **abstract** superclass (part 5 of 10).

```
115    // Set the number of items output
116    public void setQuantity( int q )
117       { quantity = ( q > 0 ? q : 0 ); }
118
119    // Determine the PieceWorker's earnings
120    public double earnings()
121       { return quantity * wagePerPiece; }
122
123    public String toString()
124    {
125       return "Piece worker: " + super.toString();
126    }
127 }
```

Fig. 9.9 **Employee** class hierarchy using an **abstract** superclass (part 6 of 10).

```
// Fig. 9.9: HourlyWorker.java
// Definition of class HourlyWorker

public final class HourlyWorker extends Employee {
   private double wage;   // wage per hour
   private double hours;  // hours worked for week

   // Constructor for class HourlyWorker
   public HourlyWorker( String first, String last,
                        double w, double h )
   {
      super( first, last );   // call superclass constructor
      setWage( w );
      setHours( h );
   }

   // Set the wage
   public void setWage( double w )
      { wage = ( w > 0 ? w : 0 ); }

   // Set the hours worked
   public void setHours( double h )
      { hours = ( h >= 0 && h < 168 ? h : 0 ); }

   // Get the HourlyWorker's pay
   public double earnings() { return wage * hours; }
```

Fig. 9.9 **Employee** class hierarchy using an **abstract** superclass (part 7 of 10).

```

   public String toString()
   {
      return "Hourly worker: " + super.toString();
   }
}
```

Fig. 9.9 **Employee** class hierarchy using an **abstract** superclass (part 8 of 10).

```
// Fig. 9.9: Test.java
// Driver for Employee hierarchy
import javax.swing.JOptionPane;
import java.text.DecimalFormat;

public class Test {
   public static void main( String args[] )
   {
      Employee ref;  // superclass reference
      String output = "";

      Boss b = new Boss( "John", "Smith", 800.00 );
      CommissionWorker c =
         new CommissionWorker( "Sue", "Jones",
                               400.0, 3.0, 150);
      PieceWorker p =
         new PieceWorker( "Bob", "Lewis", 2.5, 200 );
      HourlyWorker h =
         new HourlyWorker( "Karen", "Price", 13.75, 40 );

      DecimalFormat precision2 = new DecimalFormat( "0.00" );

      ref = b;  // Employee reference to a Boss
      output += ref.toString() + " earned $" +
                precision2.format( ref.earnings() ) + "\n" +
                b.toString() + " earned $" +
                precision2.format( b.earnings() ) + "\n";

      ref = c;  // Employee reference to a CommissionWorker
      output += ref.toString() + " earned $" +
                precision2.format( ref.earnings() ) + "\n" +
                c.toString() + " earned $" +
                precision2.format( c.earnings() ) + "\n";

      ref = p;  // Employee reference to a PieceWorker
```

Fig. 9.9 **`Employee`** class hierarchy using an **`abstract`** superclass (part 9 of 10).

```
      output += ref.toString() + " earned $" +
                precision2.format( ref.earnings() ) + "\n" +
                p.toString() + " earned $" +
                precision2.format( p.earnings() ) + "\n";

      ref = h;  // Employee reference to an HourlyWorker
      output += ref.toString() + " earned $" +
                precision2.format( ref.earnings() ) + "\n" +
                h.toString() + " earned $" +
                precision2.format( h.earnings() ) + "\n";

      JOptionPane.showMessageDialog( null, output,
         "Demonstrating Polymorphism",
         JOptionPane.INFORMATION_MESSAGE );
      System.exit( 0 );
   }
}
```

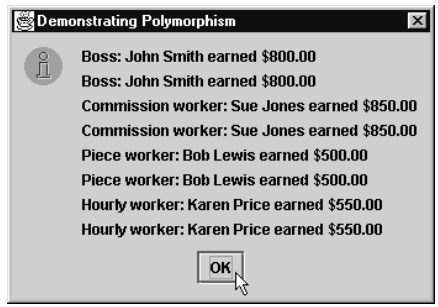

Fig. 9.9 **`Employee`** class hierarchy using an **`abstract`** superclass (part 10 of 10).

```
1  // Fig. 9.10: Shape.java
2  // Definition of abstract base class Shape
3
4  public abstract class Shape extends Object {
5     public double area() { return 0.0; }
6     public double volume() { return 0.0; }
7     public abstract String getName();
8  }
```

Fig. 9.10 Shape, point, circle, cylinder hierarchy (part 1 of 8).

```
// Fig. 9.10: Point.java
// Definition of class Point

public class Point extends Shape {
   protected int x, y; // coordinates of the Point

   // no-argument constructor
   public Point() { setPoint( 0, 0 ); }

   // constructor
   public Point( int a, int b ) { setPoint( a, b ); }

   // Set x and y coordinates of Point
   public void setPoint( int a, int b )
   {
      x = a;
      y = b;
   }

   // get x coordinate
   public int getX() { return x; }

   // get y coordinate
   public int getY() { return y; }

   // convert the point into a String representation
   public String toString()
      { return "[" + x + ", " + y + "]"; }

   // return the class name
   public String getName() { return "Point"; }
}
```

Fig. 9.10 Shape, point, circle, cylinder hierarchy (part 2 of 8).

```
41 // Fig. 9.10: Circle.java
42 // Definition of class Circle
43
44 public class Circle extends Point {  // inherits from Point
45    protected double radius;
46
47    // no-argument constructor
48    public Circle()
49    {
50       // implicit call to superclass constructor here
51       setRadius( 0 );
52    }
53
54    // Constructor
55    public Circle( double r, int a, int b )
56    {
57       super( a, b );  // call the superclass constructor
```

Fig. 9.10 Shape, point, circle, cylinder hierarchy (part 3 of 8).

```
58       setRadius( r );
59    }
60
61    // Set radius of Circle
62    public void setRadius( double r )
63       { radius = ( r >= 0 ? r : 0 ); }
64
65    // Get radius of Circle
66    public double getRadius() { return radius; }
67
68    // Calculate area of Circle
69    public double area() { return Math.PI * radius * radius; }
70
71    // convert the Circle to a String
72    public String toString()
73       { return "Center = " + super.toString() +
74                "; Radius = " + radius; }
75
76    // return the class name
77    public String getName() { return "Circle"; }
78 }
```

Fig. 9.10 Shape, point, circle, cylinder hierarchy (part 4 of 8).

```
// Fig. 9.10: Cylinder.java
// Definition of class Cylinder

public class Cylinder extends Circle {
   protected double height;  // height of Cylinder

   // no-argument constructor
   public Cylinder()
   {
      // implicit call to superclass constructor here
      setHeight( 0 );
   }

   // constructor
   public Cylinder( double h, double r, int a, int b )
   {
      super( r, a, b );   // call superclass constructor
      setHeight( h );
   }

   // Set height of Cylinder
   public void setHeight( double h )
      { height = ( h >= 0 ? h : 0 ); }

   // Get height of Cylinder
   public double getHeight() { return height; }
```

Fig. 9.10 Shape, point, circle, cylinder hierarchy (part 5 of 8).

```
   // Calculate area of Cylinder (i.e., surface area)
   public double area()
   {
      return 2 * super.area() +
             2 * Math.PI * radius * height;
   }

   // Calculate volume of Cylinder
   public double volume() { return super.area() * height; }

   // Convert a Cylinder to a String
   public String toString()
      { return super.toString() + "; Height = " + height; }

   // Return the class name
   public String getName() { return "Cylinder"; }
}
```

Fig. 9.10 Shape, point, circle, cylinder hierarchy (part 6 of 8).

```
// Fig. 9.10: Test.java
// Driver for point, circle, cylinder hierarchy
import javax.swing.JOptionPane;
import java.text.DecimalFormat;

public class Test {
   public static void main( String args[] )
   {
      Point point = new Point( 7, 11 );
      Circle circle = new Circle( 3.5, 22, 8 );
      Cylinder cylinder = new Cylinder( 10, 3.3, 10, 10 );

      Shape arrayOfShapes[];

      arrayOfShapes = new Shape[ 3 ];

      // aim arrayOfShapes[0] at subclass Point object
      arrayOfShapes[ 0 ] = point;

      // aim arrayOfShapes[1] at subclass Circle object
      arrayOfShapes[ 1 ] = circle;

      // aim arrayOfShapes[2] at subclass Cylinder object
      arrayOfShapes[ 2 ] = cylinder;

      String output =
         point.getName() + ": " + point.toString() + "\n" +
         circle.getName() + ": " + circle.toString() + "\n" +
         cylinder.getName() + ": " + cylinder.toString();

      DecimalFormat precision2 = new DecimalFormat( "0.00" );
```

Fig. 9.10 Shape, point, circle, cylinder hierarchy (part 7 of 8).

```
      // Loop through arrayOfShapes and print the name,
      // area, and volume of each object.
      for ( int i = 0; i < arrayOfShapes.length; i++ ) {
         output += "\n\n" +
            arrayOfShapes[ i ].getName() + ": " +
            arrayOfShapes[ i ].toString() +
            "\nArea = " +
            precision2.format( arrayOfShapes[ i ].area() ) +
            "\nVolume = " +
            precision2.format( arrayOfShapes[ i ].volume() );
      }

      JOptionPane.showMessageDialog( null, output,
         "Demonstrating Polymorphism",
         JOptionPane.INFORMATION_MESSAGE );

      System.exit( 0 );
   }
}
```

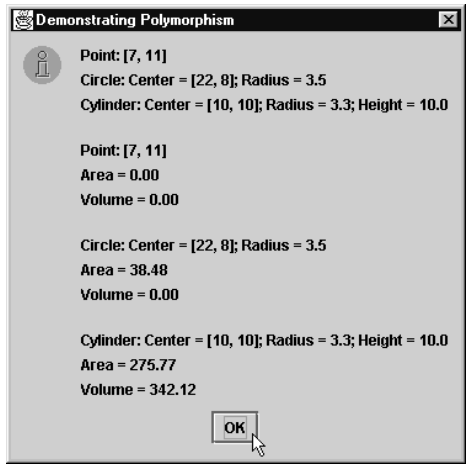

Fig. 9.10 Shape, point, circle, cylinder hierarchy (part 8 of 8).

```
1  // Fig. 9.11: Shape.java
2  // Definition of interface Shape
3
4  public interface Shape {
5     public abstract double area();
6     public abstract double volume();
7     public abstract String getName();
8  }
```

Fig. 9.11 Point, circle, cylinder hierarchy with a **Shape** interface (part 1 of 6).

```
// Fig. 9.11: Point.java
// Definition of class Point

public class Point extends Object implements Shape {
   protected int x, y; // coordinates of the Point

   // no-argument constructor
   public Point() { setPoint( 0, 0 ); }

   // constructor
   public Point( int a, int b ) { setPoint( a, b ); }

   // Set x and y coordinates of Point
   public void setPoint( int a, int b )
   {
      x = a;
      y = b;
   }

   // get x coordinate
   public int getX() { return x; }

   // get y coordinate
   public int getY() { return y; }

   // convert the point into a String representation
   public String toString()
      { return "[" + x + ", " + y + "]"; }

   // return the area
   public double area() { return 0.0; }

   // return the volume
   public double volume() { return 0.0; }

   // return the class name
   public String getName() { return "Point"; }
}
```

Fig. 9.11 Point, circle, cylinder hierarchy with a **Shape** interface (part 2 of 6).

```
// Fig. 9.11: Circle.java
// Definition of class Circle

public class Circle extends Point {  // inherits from Point
   protected double radius;

   // no-argument constructor
   public Circle()
   {
      // implicit call to superclass constructor here
      setRadius( 0 );
   }

   // Constructor
   public Circle( double r, int a, int b )
   {
      super( a, b );  // call the superclass constructor
      setRadius( r );
   }

   // Set radius of Circle
   public void setRadius( double r )
      { radius = ( r >= 0 ? r : 0 ); }

   // Get radius of Circle
   public double getRadius() { return radius; }

   // Calculate area of Circle
   public double area() { return Math.PI * radius * radius; }

   // convert the Circle to a String
   public String toString()
      { return "Center = " + super.toString() +
               "; Radius = " + radius; }

   // return the class name
   public String getName() { return "Circle"; }
}
```

Fig. 9.11 Point, circle, cylinder hierarchy with a **Shape** interface (part 3 of 6).

```
// Fig. 9.11: Cylinder.java
// Definition of class Cylinder

public class Cylinder extends Circle {
   protected double height;  // height of Cylinder

   // no-argument constructor
   public Cylinder()
   {
      // implicit call to superclass constructor here
      setHeight( 0 );
   }

   // constructor
   public Cylinder( double h, double r, int a, int b )
   {
      super( r, a, b );   // call superclass constructor
      setHeight( h );
   }

   // Set height of Cylinder
   public void setHeight( double h )
      { height = ( h >= 0 ? h : 0 ); }

   // Get height of Cylinder
   public double getHeight() { return height; }

   // Calculate area of Cylinder (i.e., surface area)
   public double area()
   {
      return 2 * super.area() +
             2 * Math.PI * radius * height;
   }

   // Calculate volume of Cylinder
   public double volume() { return super.area() * height; }

   // Convert a Cylinder to a String
   public String toString()
      { return super.toString() + "; Height = " + height; }

   // Return the class name
   public String getName() { return "Cylinder"; }
}
```

Fig. 9.11 Point, circle, cylinder hierarchy with a **`Shape`** interface (part 4 of 6).

```
// Fig. 9.11: Test.java
// Driver for point, circle, cylinder hierarchy
import javax.swing.JOptionPane;
import java.text.DecimalFormat;

public class Test {
   public static void main( String args[] )
   {
      Point point = new Point( 7, 11 );
      Circle circle = new Circle( 3.5, 22, 8 );
      Cylinder cylinder = new Cylinder( 10, 3.3, 10, 10 );

      Shape arrayOfShapes[];

      arrayOfShapes = new Shape[ 3 ];

      // aim arrayOfShapes[0] at subclass Point object
      arrayOfShapes[ 0 ] = point;

      // aim arrayOfShapes[1] at subclass Circle object
      arrayOfShapes[ 1 ] = circle;

      // aim arrayOfShapes[2] at subclass Cylinder object
      arrayOfShapes[ 2 ] = cylinder;

      String output =
         point.getName() + ": " + point.toString() + "\n" +
         circle.getName() + ": " + circle.toString() + "\n" +
         cylinder.getName() + ": " + cylinder.toString();

      DecimalFormat precision2 = new DecimalFormat( "#0.00" );

      // Loop through arrayOfShapes and print the name,
      // area, and volume of each object.
      for ( int i = 0; i < arrayOfShapes.length; i++ ) {
         output += "\n\n" +
            arrayOfShapes[ i ].getName() + ": " +
            arrayOfShapes[ i ].toString() +
            "\nArea = " +
            precision2.format( arrayOfShapes[ i ].area() ) +
            "\nVolume = " +
            precision2.format( arrayOfShapes[ i ].volume() );
      }

      JOptionPane.showMessageDialog( null, output,
         "Demonstrating Polymorphism",
         JOptionPane.INFORMATION_MESSAGE );

      System.exit( 0 );
   }
}
```

Fig. 9.11 Point, circle, cylinder hierarchy with a **Shape** interface (part 5 of 6).

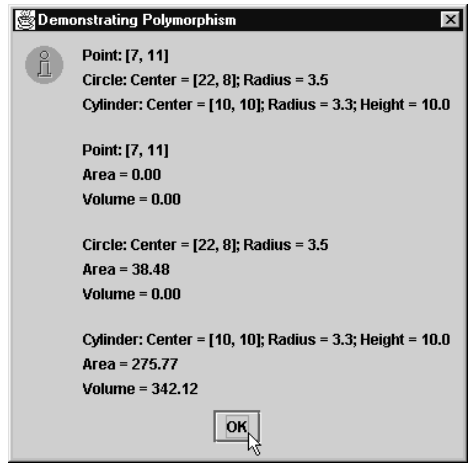

Fig. 9.11 Point, circle, cylinder hierarchy with a **`Shape`** interface (part 6 of 6).

```
// Fig. 9.12: Time.java
// Time class definition
import java.text.DecimalFormat;  // used for number formatting

// This class maintains the time in 24-hour format
public class Time extends Object {
   private int hour;     // 0 - 23
   private int minute;   // 0 - 59
   private int second;   // 0 - 59

   // Time constructor initializes each instance variable
   // to zero. Ensures that Time object starts in a
   // consistent state.
   public Time() { setTime( 0, 0, 0 ); }

   // Set a new time value using universal time. Perform
   // validity checks on the data. Set invalid values to zero.
   public void setTime( int h, int m, int s )
   {
      setHour( h );    // set the hour
      setMinute( m );  // set the minute
      setSecond( s );  // set the second
   }

   // set the hour
   public void setHour( int h )
      { hour = ( ( h >= 0 && h < 24 ) ? h : 0 ); }

   // set the minute
   public void setMinute( int m )
      { minute = ( ( m >= 0 && m < 60 ) ? m : 0 ); }

   // set the second
   public void setSecond( int s )
      { second = ( ( s >= 0 && s < 60 ) ? s : 0 ); }

   // get the hour
   public int getHour() { return hour; }

   // get the minute
   public int getMinute() { return minute; }

   // get the second
   public int getSecond() { return second; }

```

Fig. 9.12 Demonstrating an inner class in a windowed application (part 1 of 5).

```
46     // Convert to String in standard-time format
47     public String toString()
48     {
49        DecimalFormat twoDigits = new DecimalFormat( "00" );
50
51        return ( ( getHour() == 12 || getHour() == 0 ) ?
52                  12 : getHour() % 12 ) + ":" +
53                twoDigits.format( getMinute() ) + ":" +
54                twoDigits.format( getSecond() ) +
55                ( getHour() < 12 ? " AM" : " PM" );
56     }
57  }
```

Fig. 9.12 Demonstrating an inner class in a windowed application (part 2 of 5).

```
// Fig. 9.12: TimeTestWindow.java
// Demonstrating the Time class set and get methods
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class TimeTestWindow extends JFrame {
   private Time t;
   private JLabel hourLabel, minuteLabel, secondLabel;
   private JTextField hourField, minuteField,
                      secondField, display;
   private JButton exitButton;

   public TimeTestWindow()
   {
      super( "Inner Class Demonstration" );

      t = new Time();

      Container c = getContentPane();

      // create an instance of the inner class
      ActionEventHandler handler = new ActionEventHandler();

      c.setLayout( new FlowLayout() );
      hourLabel = new JLabel( "Set Hour" );
      hourField = new JTextField( 10 );
      hourField.addActionListener( handler );
      c.add( hourLabel );
      c.add( hourField );

      minuteLabel = new JLabel( "Set minute" );
      minuteField = new JTextField( 10 );
      minuteField.addActionListener( handler );
      c.add( minuteLabel );
      c.add( minuteField );

```

Fig. 9.12 Demonstrating an inner class in a windowed application (part 3 of 5).

```
95          secondLabel = new JLabel( "Set Second" );
96          secondField = new JTextField( 10 );
97          secondField.addActionListener( handler );
98          c.add( secondLabel );
99          c.add( secondField );
100
101         display = new JTextField( 30 );
102         display.setEditable( false );
103         c.add( display );
104
105         exitButton = new JButton( "Exit" );
106         exitButton.addActionListener( handler );
107         c.add( exitButton );
108      }
109
110      public void displayTime()
111      {
112         display.setText( "The time is: " + t );
113      }
114
115      public static void main( String args[] )
116      {
117         TimeTestWindow window = new TimeTestWindow();
118
119         window.setSize( 400, 140 );
120         window.show();
121      }
122
123      // INNER CLASS DEFINITION FOR EVENT HANDLING
124      private class ActionEventHandler implements ActionListener {
125         public void actionPerformed( ActionEvent e )
126         {
127            if ( e.getSource() == exitButton )
128               System.exit( 0 );   // terminate the application
129            else if ( e.getSource() == hourField ) {
130               t.setHour(
131                  Integer.parseInt( e.getActionCommand() ) );
132               hourField.setText( "" );
133            }
134            else if ( e.getSource() == minuteField ) {
135               t.setMinute(
136                  Integer.parseInt( e.getActionCommand() ) );
137               minuteField.setText( "" );
138            }
139            else if ( e.getSource() == secondField ) {
140               t.setSecond(
141                  Integer.parseInt( e.getActionCommand() ) );
142               secondField.setText( "" );
143            }
144
145            displayTime();
146         }
147      }
148   }
```

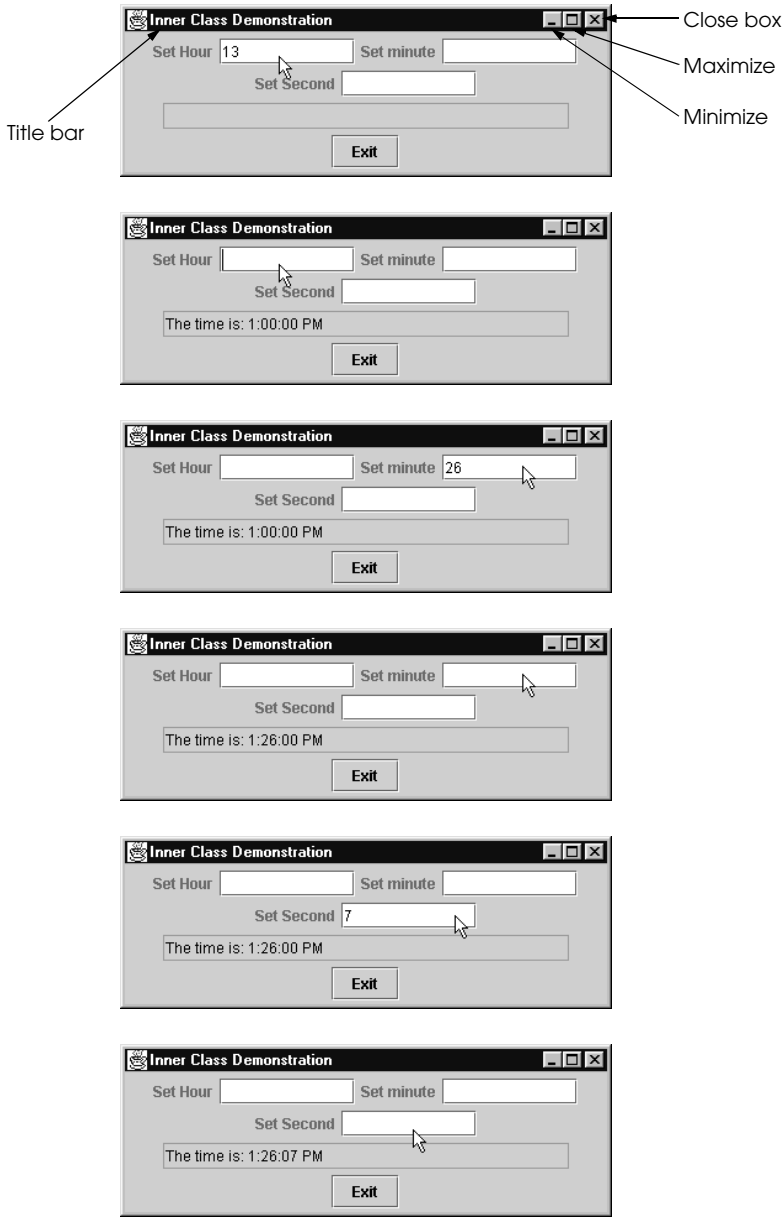

Fig. 9.12 Demonstrating an inner class in a windowed application (part 5 of 5).

```
1  // Fig. 9.13: TimeTestWindow.java
2  // Demonstrating the Time class set and get methods
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class TimeTestWindow extends JFrame {
8     private Time t;
9     private JLabel hourLabel, minuteLabel, secondLabel;
10    private JTextField hourField, minuteField,
11                       secondField, display;
12
13    public TimeTestWindow()
14    {
15       super( "Inner Class Demonstration" );
16
17       t = new Time();
18
19       Container c = getContentPane();
20
21       c.setLayout( new FlowLayout() );
22       hourLabel = new JLabel( "Set Hour" );
23       hourField = new JTextField( 10 );
```

Fig. 9.13 Demonstrating anonymous inner classes (part 1 of 4).

```
24          hourField.addActionListener(
25             new ActionListener() {  // anonymous inner class
26                public void actionPerformed( ActionEvent e )
27                {
28                   t.setHour(
29                      Integer.parseInt( e.getActionCommand() ) );
30                   hourField.setText( "" );
31                   displayTime();
32                }
33             }
34          );
35          c.add( hourLabel );
36          c.add( hourField );
37
38          minuteLabel = new JLabel( "Set minute" );
39          minuteField = new JTextField( 10 );
40          minuteField.addActionListener(
41             new ActionListener() {  // anonymous inner class
42                public void actionPerformed( ActionEvent e )
43                {
44                   t.setMinute(
45                      Integer.parseInt( e.getActionCommand() ) );
46                   minuteField.setText( "" );
47                   displayTime();
48                }
49             }
50          );
51          c.add( minuteLabel );
52          c.add( minuteField );
53
54          secondLabel = new JLabel( "Set Second" );
55          secondField = new JTextField( 10 );
56          secondField.addActionListener(
57             new ActionListener() {  // anonymous inner class
58                public void actionPerformed( ActionEvent e )
59                {
60                   t.setSecond(
61                      Integer.parseInt( e.getActionCommand() ) );
62                   secondField.setText( "" );
63                   displayTime();
64                }
65             }
66          );
67          c.add( secondLabel );
68          c.add( secondField );
69
70          display = new JTextField( 30 );
71          display.setEditable( false );
72          c.add( display );
73       }
74
```

Fig. 9.13 Demonstrating anonymous inner classes (part 2 of 4).

```
75      public void displayTime()
76      {
77         display.setText( "The time is: " + t );
78      }
79
80      public static void main( String args[] )
81      {
82         TimeTestWindow window = new TimeTestWindow();
83
84         window.addWindowListener(
85            new WindowAdapter() {
86               public void windowClosing( WindowEvent e )
87               {
88                  System.exit( 0 );
89               }
90            }
91         );
92
93         window.setSize( 400, 120 );
94         window.show();
95      }
96   }
```

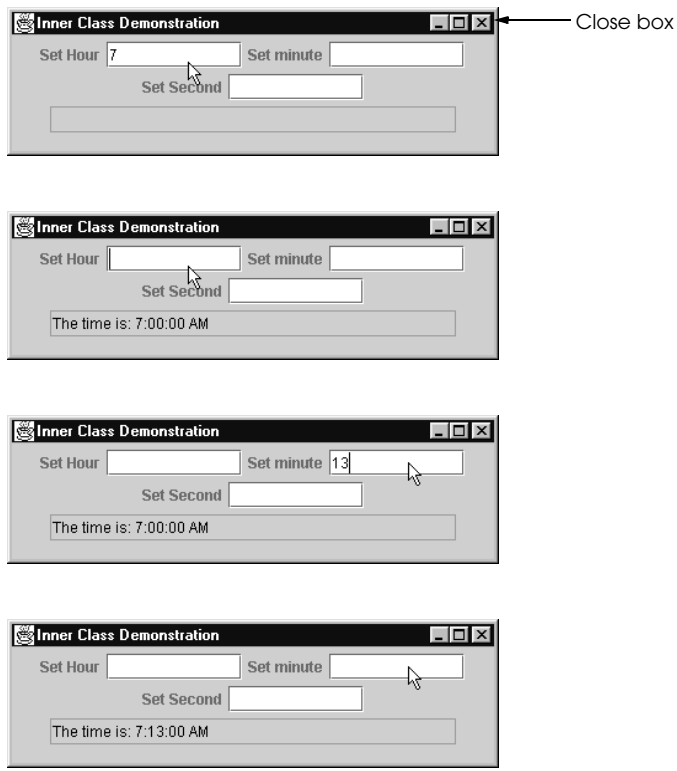

Fig. 9.13 Demonstrating anonymous inner classes (part 3 of 4).

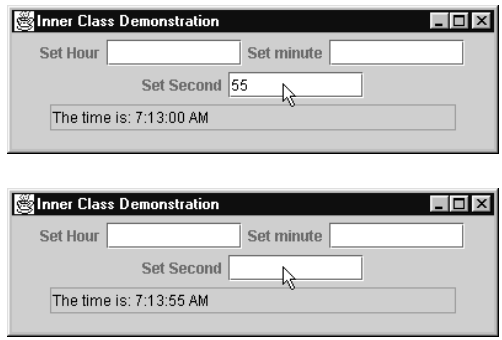

Fig. 9.13 Demonstrating anonymous inner classes (part 4 of 4).