Chapter 7 Object-Oriented Programming (Main Page)

Superclass	Subclasses
Student	GraduateStudent UndergraduateStudent
Shape	Circle Triangle Rectangle
Loan	CarLoan HomeImprovementLoan MortgageLoan
Employee	FacultyMember StaffMember
Account	CheckingAccount SavingsAccount

Fig. 7.1 Some simple inheritance examples.

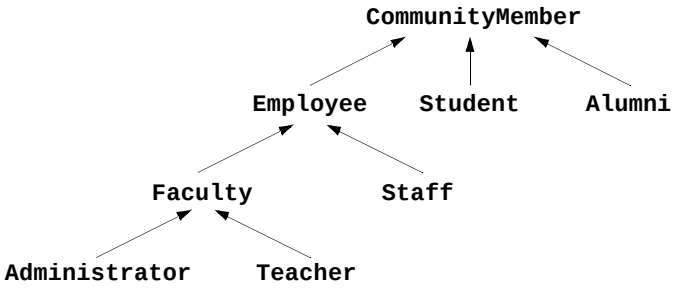


Fig. 7.2 An inheritance hierarchy for university community members.

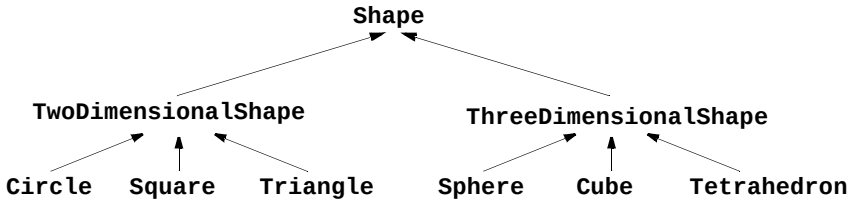


Fig. 7.3 A portion of a **Shape** class hierarchy.

```
1  // Fig. 7.4: 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() { setPoint( 0, 0 ); }
9
10    // Constructor
11    public Point( int a, int b ) { setPoint( a, b ); }
12
13    // Set x and y coordinates of Point
14    public void setPoint( int a, int b )
15    {
16       x = a;
17       y = b;
18    }
19
20    // get x coordinate
21    public int getX() { return x; }
22
23    // get y coordinate
24    public int getY() { return y; }
25
26    // convert the point into a String representation
27    public String toString()
28       { return "[" + x + ", " + y + "]"; }
29 }
```

Fig. 7.4 Assigning subclass references to superclass references (part 1 of 4).

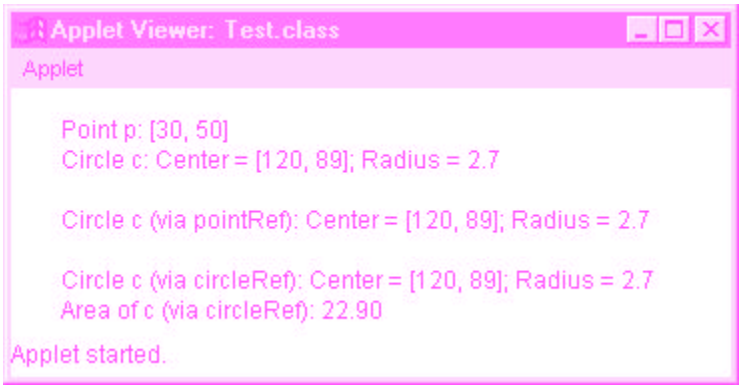
```
30 // Fig. 7.4: Circle.java
31 // Definition of class Circle
32
33 public class Circle extends Point {  // inherits from Point
34    protected double radius;
35
36    // No-argument constructor
37    public Circle()
38    {
39       // implicit call to superclass constructor occurs here
40       setRadius( 0 );
41    }
42
43    // Constructor
44    public Circle( double r, int a, int b )
45    {
46       super( a, b );  // explicit call to superclass constructor
47       setRadius( r );
48    }
49
50    // Set radius of Circle
51    public void setRadius( double r )
52       { radius = ( r >= 0.0 ? r : 0.0 ); }
53
54    // Get radius of Circle
55    public double getRadius() { return radius; }
56
57    // Calculate area of Circle
58    public double area() { return Math.PI * radius * radius; }
59
```

```
60     // convert the Circle to a String
61     public String toString()
62     {
63        return "Center = " + "[" + x + ", " + y + "]" +
64               "; Radius = " + radius;
65     }
66  }
```

Fig. 7.4 Assigning subclass references to superclass references (part 2 of 4).

```
67  // Fig. 7.4: Test.java
68  // Casting superclass references to subclass references
69  import java.awt.Graphics;
70  import java.applet.Applet;
71  import java.text.DecimalFormat;
72
73  public class Test extends Applet {
74     private Point pointRef, p;
75     private Circle circleRef, c;
76
77     public void init()
78     {
79        p = new Point( 30, 50 );
80        c = new Circle( 2.7, 120, 89 );
81     }
82
83     public void paint( Graphics g )
84     {
85        DecimalFormat precision2 = new DecimalFormat( "#.00" );
86
87        g.drawString( "Point p: " + p.toString(), 25, 25 );
88        g.drawString( "Circle c: " + c.toString(), 25, 40 );
89
90        // Attempt to treat a Circle as a Point
91        pointRef = c;   // assign Circle to pointRef
92        g.drawString( "Circle c (via pointRef): " +
93                      pointRef.toString(), 25, 70 );
94
95        // Treat a Circle as a Circle (with some casting)
96        circleRef = (Circle) pointRef;  // cast super to sub
97        g.drawString( "Circle c (via circleRef): " +
98                       circleRef.toString(), 25, 100);
99        g.drawString( "Area of c (via circleRef): " +
100                     precision2.format( circleRef.area() ),
101                     25, 115 );
102
103       // Attempt to refer to Point object
104       // with Circle reference
105       circleRef = (Circle) p;     // line 39 in Test.java
106    }
107 }
```

Fig. 7.4 Assigning subclass references to superclass references (part 3 of 4).




```
Exception occurred during event dispatching:
java.lang.ClassCastException:
        at Test.paint(Test.java:39)
        at java.awt.Component.dispatchEventImpl
           (Component.java:1401)
        at java.awt.Container.dispatchEventImpl
           (Container.java:833)
        at java.awt.Component.dispatchEvent
           (Component.java:1382)
        at java.awt.EventDispatchThread.run
           (EventDispatchThread.java:63)
```

Fig. 7.4 Assigning subclass references to superclass references (part 4 of 4).

```
1  // Fig. 7.5: Point.java
2  // Definition of class Point
3  public class Point {
4     protected int x, y; // coordinates of the Point
5
6     // no-argument constructor
7     public Point()
8     {
9        setPoint( 0, 0 );
10       System.out.println( "Point constructor: " +
11                           toString() );
12    }
13
14    // constructor
15    public Point( int a, int b )
16    {
17       setPoint( a, b );
18       System.out.println( "Point constructor: " +
19                           toString() );
20    }
21
22    // finalizer
23    protected void finalize() throws Throwable
24    {
25       System.out.println( "Point finalizer: " +
26                           toString() );
27       super.finalize();  // call superclass finalize method
28    }
29
```

```
30      // Set x and y coordinates of Point
31      public void setPoint( int a, int b )
32      {
33         x = a;
34         y = b;
35      }
```

Fig. 7.5 Order in which constructors and finalizers are called (part 1 of 6).

```
36
37      // get x coordinate
38      public int getX() { return x; }
39
40      // get y coordinate
41      public int getY() { return y; }
42
43      // convert the point into a String representation
44      public String toString()
45         { return "[" + x + ", " + y + "]"; }
46   }
```

Fig. 7.5 Order in which constructors and finalizers are called (part 2 of 6).

```
47   // Fig. 7.5: Circle.java
48   // Definition of class Circle
49   public class Circle extends Point {  // inherits from Point
50      protected double radius;
51
52      // no-argument constructor
53      public Circle()
54      {
55         // implicit call to superclass constructor here
56         setRadius( 0 );
57         System.out.println( "Circle constructor: " +
58                             toString() );
59      }
60
61      // Constructor
62      public Circle( double r, int a, int b )
63      {
64         super( a, b );  // call the superclass constructor
65         setRadius( r );
66         System.out.println( "Circle constructor: " +
67                             toString() );
68      }
69
70      // finalizer
71      protected void finalize() throws Throwable
72      {
73         System.out.println( "Circle finalizer: " +
74                             toString() );
75         super.finalize();  // call superclass finalize method
76      }
77
78      // Set radius of Circle
79      public void setRadius( double r )
80         { radius = ( r >= 0 ? r : 0 ); }
81
```

Fig. 7.5 Order in which constructors and finalizers are called (part 3 of 6).

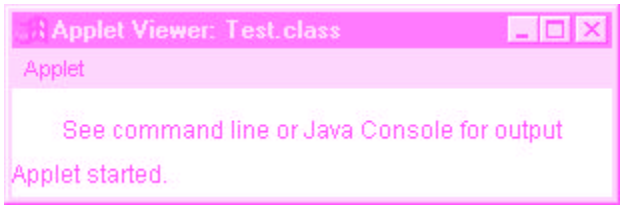
```
82      // Get radius of Circle
```

```
83      public double getRadius() { return radius; }
84
85      // Calculate area of Circle
86      public double area()
87         { return Math.PI * radius * radius; }
88
89      // convert the Circle to a String
90      public String toString()
91      {
92         return "Center = " + super.toString() +
93                "; Radius = " + radius;
94      }
95   }
```

Fig. 7.5 Order in which constructors and finalizers are called (part 4 of 6).

```
96   // Fig. 7.5: Test.java
97   // Demonstrate when superclass and subclass
98   // constructors and finalizers are called.
99   import java.awt.Graphics;
100  import java.applet.Applet;
101
102  public class Test extends Applet {
103     private Circle circle1, circle2;
104
105     public void init()
106     {
107        circle1 = new Circle( 4.5, 72, 29 );
108        circle2 = new Circle( 10, 5, 5 );
109     }
110
111     public void start()
112     {
113        circle2 = null; // Circle can now be garbage collected
114        circle1 = null; // Circle can now be garbage collected
115
116        System.gc();    // call the garbage collector
117     }
118
119     public void paint( Graphics g )
120     {
121        g.drawString(
122           "See command line or Java Console for output",
123           25, 25 );
124     }
125  }
```

Fig. 7.5 Order in which constructors and finalizers are called (part 5 of 6).



```
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. 7.5 Order in which constructors and finalizers are called (part 6 of 6).

```
1  // Fig. 7.6: Point.java
2  // Definition of class Point
3  package com.deitel.jhtp2.ch07;
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. 7.6 Testing class **Point** (part 1 of 3).

```
31 // Fig. 7.6: Test.java
32 // Applet to test class Point
33 import java.awt.Graphics;
34 import java.applet.Applet;
35 import com.deitel.jhtp2.ch07.Point;
36
37 public class Test extends Applet {
38    private Point p;
39
40    public void init()
41    {
42       p = new Point( 72, 115 );
43    }
```

```
44
```

Fig. 7.6 Testing class **Point** (part 2 of 3).

```
45     public void paint( Graphics g )
46     {
47        g.drawString( "X coordinate is " + p.getX(), 25, 25 );
48        g.drawString( "Y coordinate is " + p.getY(), 25, 40 );
49
50        p.setPoint( 10, 10 );
51        g.drawString( "The new location of p is " +
52                      p.toString(), 25, 70 );
53     }
54  }
```

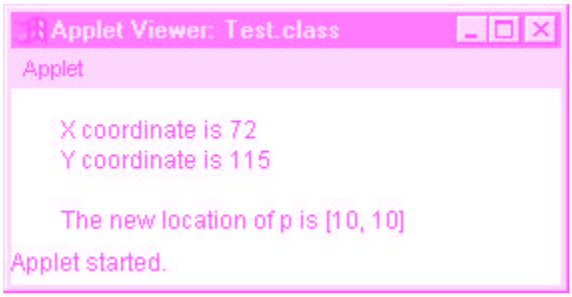


Fig. 7.6 Testing class **Point** (part 3 of 3).

```
1  // Fig. 7.7: Circle.java
2  // Definition of class Circle
3  package com.deitel.jhtp2.ch07;
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 super class constructor
12       setRadius( 0 );
13    }
14
```

Fig. 7.7 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. 7.7 Testing class **Circle** (part 2 of 4).

```
40  // Fig. 7.7: Test.java
41  // Applet to test class Circle
42  import java.awt.Graphics;
43  import java.applet.Applet;
44  import java.text.DecimalFormat;
45  import com.deitel.jhtp2.ch07.Circle;
46
47  public class Test extends Applet {
48     private Circle c;
49
50     public void init()
51     {
52        c = new Circle( 2.5, 37, 43 );
53     }
54
55     public void paint( Graphics g )
56     {
57        DecimalFormat precision2 = new DecimalFormat( "#.00" );
58
59        g.drawString( "X coordinate is " + c.getX(), 25, 25 );
60        g.drawString( "Y coordinate is " + c.getY(), 25, 40 );
61        g.drawString( "Radius is " + c.getRadius(), 25, 55 );
```

Fig. 7.7 Testing class **Circle** (part 3 of 4).

```
62
63        c.setRadius( 4.25 );
64        c.setPoint( 2, 2 );
65        g.drawString( "The new location and radius of c are ",
66                      25, 85 );
67        g.drawString( c.toString(), 40, 100 );
68        g.drawString( "Area is " +
69                      precision2.format( c.area() ), 25, 115 );
70     }
71  }
```

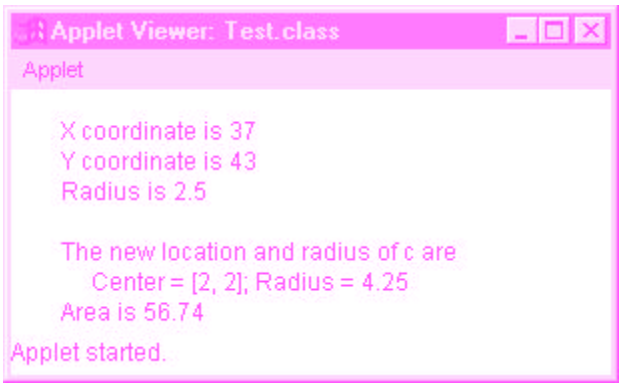


Fig. 7.7 Testing class `Circle` (part 4 of 4).

```
1  // Fig. 7.8: Cylinder.java
2  // Definition of class Cylinder
3  package com.deitel.jhtp2.ch07;
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    }
```

```
}
```

Fig. 7.8 Testing class **Cylinder** (part 1 of 3).

```
// Fig. 7.8: Test.java
// Applet to test class Cylinder
import java.awt.Graphics;
import java.applet.Applet;
import java.text.*;
import com.deitel.jhtp2.ch07.Cylinder;

public class Test extends Applet {
   private Cylinder c;

   public void init()
   {
      c = new Cylinder( 5.7, 2.5, 12, 23 );
   }

   public void paint( Graphics g )
   {
      DecimalFormat precision2 = new DecimalFormat( "#.00" );

      g.drawString( "X coordinate is " + c.getX(), 25, 25 );
      g.drawString( "Y coordinate is " + c.getY(), 25, 40 );
      g.drawString( "Radius is " + c.getRadius(), 25, 55 );
      g.drawString( "Height is " + c.getHeight(), 25, 70 );

      c.setHeight( 10 );
      c.setRadius( 4.25 );
      c.setPoint( 2, 2 );

      g.drawString( "The new location, radius and height" +
                    " of c are ", 25, 100 );
      g.drawString( c.toString(), 40, 115 );
      g.drawString( "Area is " +
                    precision2.format( c.area() ), 25, 130 );
      g.drawString( "Volume is " +
                    precision2.format( c.volume() ), 25, 145 );
   }
}
```

Fig. 7.8 Testing class **Cylinder** (part 2 of 3).

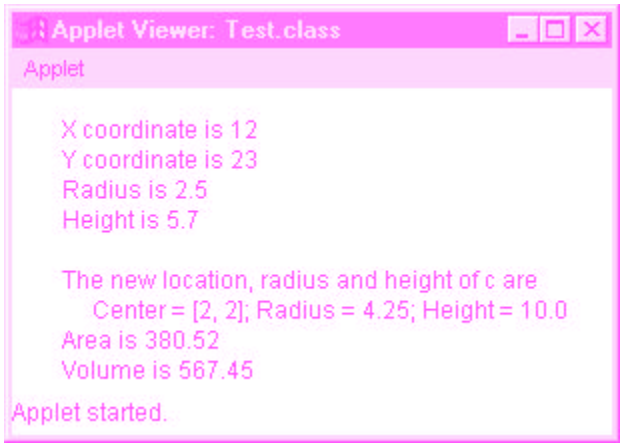


Fig. 7.8 Testing class **Cylinder** (part 3 of 3).

```
1  // Fig. 7.9: Employee.java
2  // Abstract superclass Employee
3
4  public abstract class Employee {
5     private String firstName;
6     private String lastName;
7
```

Fig. 7.9 **Employee** class hierarchy using an **abstract** superclass (part 1 of 8).

```
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    abstract double earnings();
28 }
```

Fig. 7.9 **Employee** class hierarchy using an **abstract** superclass (part 2 of 8).

```
29 // Fig. 7.9: 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. 7.9 **Employee** class hierarchy using an **abstract** superclass (part 3 of 8).

```
55  // Fig. 7.9: CommissionWorker.java
56  // CommissionWorker class derived from Employee
57
58  public final class CommissionWorker extends Employee {
59     private double salary;      // base salary per week
60     private double commission;  // amount per item sold
61     private int quantity;       // total items sold for week
62
63     // Constructor for class CommissionWorker
64     public CommissionWorker( String first, String last,
65                              double s, double c, int q)
66     {
67        super( first, last );  // call superclass constructor
68        setSalary( s );
69        setCommission( c );
70        setQuantity( q );
71     }
72
73     // Set CommissionWorker's weekly base salary
74     public void setSalary( double s )
75        { salary = ( s > 0 ? s : 0 ); }
76
77     // Set CommissionWorker's commission
78     public void setCommission( double c )
79        { commission = ( c > 0 ? c : 0 ); }
80
81     // Set CommissionWorker's quantity sold
82     public void setQuantity( int q )
83        { quantity = ( q > 0 ? q : 0 ); }
84
85     // Determine CommissionWorker's earnings
86     public double earnings()
87        { return salary + commission * quantity; }
88
89     // Print the CommissionWorker's name
90     public String toString()
91     {
92        return "Commission worker: " + super.toString();
93     }
```

```
94  }
```

Fig. 7.9 **Employee** class hierarchy using an **abstract** superclass (part 4 of 8).

```
95  // Fig. 7.9: 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
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. 7.9 **Employee** class hierarchy using an **abstract** superclass (part 5 of 8).

```
128 // Fig. 7.9: HourlyWorker.java
129 // Definition of class HourlyWorker
130
131 public final class HourlyWorker extends Employee {
132    private double wage;   // wage per hour
133    private double hours;  // hours worked for week
134
135    // Constructor for class HourlyWorker
136    public HourlyWorker( String first, String last,
137                         double w, double h )
138    {
139       super( first, last );   // call superclass constructor
140       setWage( w );
141       setHours( h );
142    }
143
144    // Set the wage
145    public void setWage( double w )
146       { wage = ( w > 0 ? w : 0 ); }
147
148    // Set the hours worked
149    public void setHours( double h )
150       { hours = ( h >= 0 && h < 168 ? h : 0 ); }
```

```
151
152    // Get the HourlyWorker's pay
153    public double earnings() { return wage * hours; }
154
155    public String toString()
156    {
157       return "Hourly worker: " + super.toString();
158    }
159 }
```

Fig. 7.9 **`Employee`** class hierarchy using an **`abstract`** superclass (part 6 of 8).

```
160 // Fig. 7.9: Test.java
161 // Driver for Employee hierarchy
162 import java.awt.Graphics;
163 import java.applet.Applet;
164 import java.text.DecimalFormat;
165
166 public class Test extends Applet {
167    private Employee ref;  // superclass reference
168    private Boss b;
169    private CommissionWorker c;
170    private PieceWorker p;
171    private HourlyWorker h;
172
173    public void init()
174    {
175       b = new Boss( "John", "Smith", 800.00 );
176       c = new CommissionWorker( "Sue", "Jones",
177                                 400.0, 3.0, 150);
178       p = new PieceWorker( "Bob", "Lewis", 2.5, 200 );
179       h = new HourlyWorker( "Karen", "Price", 13.75, 40 );
180    }
181
182    public void paint( Graphics g )
183    {
184       DecimalFormat precision2 = new DecimalFormat( "#.00" );
185
186       ref = b;  // superclass reference to subclass object
187       g.drawString( ref.toString() + " earned $" +
188                     precision2.format( ref.earnings() ),
189                     25, 25 );
190       g.drawString( b.toString() + " earned $" +
191                     precision2.format( b.earnings() ),
192                     25, 40 );
193
194       ref = c;  // superclass reference to subclass object
195       g.drawString( ref.toString() + " earned $" +
196                     precision2.format( ref.earnings() ),
197                     25, 55 );
198       g.drawString( c.toString() + " earned $" +
199                     precision2.format( c.earnings() ),
200                     25, 70 );
201
202       ref = p;  // superclass reference to subclass object
203       g.drawString( ref.toString() + " earned $" +
204                     precision2.format( ref.earnings() ),
205                     25, 85 );
206       g.drawString( p.toString() + " earned $" +
207                     precision2.format( p.earnings() ),
208                     25, 100 );
```

```
209
```

Fig. 7.9 **Employee** class hierarchy using an **abstract** superclass (part 7 of 8).

```
210       ref = h;  // superclass reference to subclass object
211       g.drawString( ref.toString() + " earned $" +
212                     precision2.format( ref.earnings() ),
213                     25, 115 );
214       g.drawString( h.toString() + " earned $" +
215                     precision2.format( h.earnings() ),
216                     25, 130 );
217    }
218 }
```

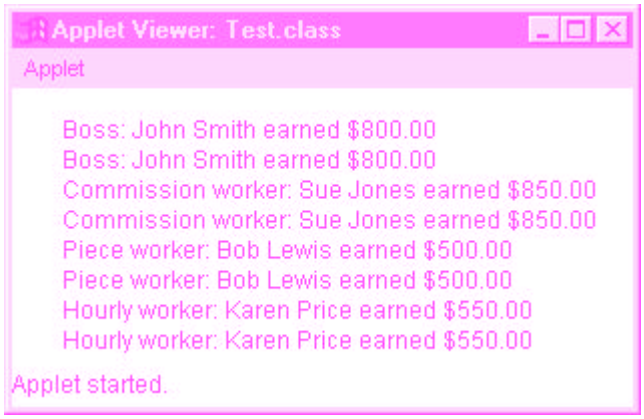


Fig. 7.9 **Employee** class hierarchy using an **abstract** superclass (part 8 of 8).

```
1  // Fig. 7.10: Shape.java
2  // Definition of abstract superclass Shape
3
4  public abstract class Shape {
5     public double area() { return 0.0; }
6     public double volume() { return 0.0; }
7     public abstract String getName();
8  }
```

Fig. 7.10 Shape, point, circle, cylinder hierarchy (part 1 of 6).

```
9  // Fig. 7.10: Point.java
10 // Definition of class Point
11
12 public class Point extends Shape {
13    protected int x, y; // coordinates of the Point
14
15    // no-argument constructor
16    public Point() { setPoint( 0, 0 ); }
17
18    // constructor
19    public Point( int a, int b ) { setPoint( a, b ); }
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
38     // return the class name
39     public String getName() { return "Point"; }
40  }
```

Fig. 7.10 Shape, point, circle, cylinder hierarchy (part 2 of 6).

```
41  // Fig. 7.10: 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
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. 7.10 Shape, point, circle, cylinder hierarchy (part 3 of 6).

```
79  // Fig. 7.10: Cylinder.java
80  // Definition of class Cylinder
81
82  public class Cylinder extends Circle {
83     protected double height;  // height of Cylinder
```

```
84
85     // no-argument constructor
86     public Cylinder()
87     {
88        // implicit call to superclass constructor here
89        setHeight( 0 );
90     }
91
92     // constructor
93     public Cylinder( double h, double r, int a, int b )
94     {
95        super( r, a, b );   // call superclass constructor
96        setHeight( h );
97     }
98
99     // Set height of Cylinder
100    public void setHeight( double h )
101       { height = ( h >= 0 ? h : 0 ); }
102
103    // Get height of Cylinder
104    public double getHeight() { return height; }
105
106    // Calculate area of Cylinder (i.e., surface area)
107    public double area()
108    {
109       return 2 * super.area() +
110              2 * Math.PI * radius * height;
111    }
112
113    // Calculate volume of Cylinder
114    public double volume() { return super.area() * height; }
115
116    // Convert a Cylinder to a String
117    public String toString()
118       { return super.toString() + "; Height = " + height; }
119
120    // Return the class name
121    public String getName() { return "Cylinder"; }
122 }
```

Fig. 7.10 Shape, point, circle, cylinder hierarchy (part 4 of 6).

```
123 // Fig. 7.10: Test.java
124 // Driver for point, circle, cylinder hierarchy
125 import java.awt.Graphics;
126 import java.applet.Applet;
127 import java.text.DecimalFormat;
128
129 public class Test extends Applet {
130    private Point point;
131    private Circle circle;
132    private Cylinder cylinder;
133    private Shape arrayOfShapes[];
134
135    public void init()
136    {
137       point = new Point( 7, 11 );
138       circle = new Circle( 3.5, 22, 8 );
139       cylinder = new Cylinder( 10, 3.3, 10, 10 );
140
141       arrayOfShapes = new Shape[ 3 ];
142
143       // aim arrayOfShapes[0] at subclass Point object
144       // aim arrayOfShapes[1] at subclass Circle object
```

```
145         // aim arrayOfShapes[2] at subclass Cylinder object
146         arrayOfShapes[ 0 ] = point;
147         arrayOfShapes[ 1 ] = circle;
148         arrayOfShapes[ 2 ] = cylinder;
149      }
150
151      public void paint( Graphics g )
152      {
153         g.drawString( point.getName() + ": " +
154                       point.toString(), 25, 25 );
155
156         g.drawString( circle.getName() + ": " +
157                       circle.toString(), 25, 40 );
158
159         g.drawString( cylinder.getName() + ": " +
160                       cylinder.toString(), 25, 55 );
161
162         DecimalFormat precision2 = new DecimalFormat( "#0.00" );
163         int yPos = 85;
164
165         // Loop through arrayOfShapes and print the name,
166         // area, and volume of each object.
167         for ( int i = 0; i < arrayOfShapes.length; i++ ) {
168            g.drawString( arrayOfShapes[ i ].getName() + ": " +
169                          arrayOfShapes[ i ].toString(),
170                          25, yPos );
171            yPos += 15;
```

Fig. 7.10 Shape, point, circle, cylinder hierarchy (part 5 of 6).

```
172            g.drawString( "Area = " + precision2.format(
173                          arrayOfShapes[ i ].area() ),
174                          25, yPos );
175            yPos += 15;
176            g.drawString( "Volume = " + precision2.format(
177                          arrayOfShapes[ i ].volume() ),
178                          25, yPos );
179            yPos += 30;
180         }
181      }
182   }
```

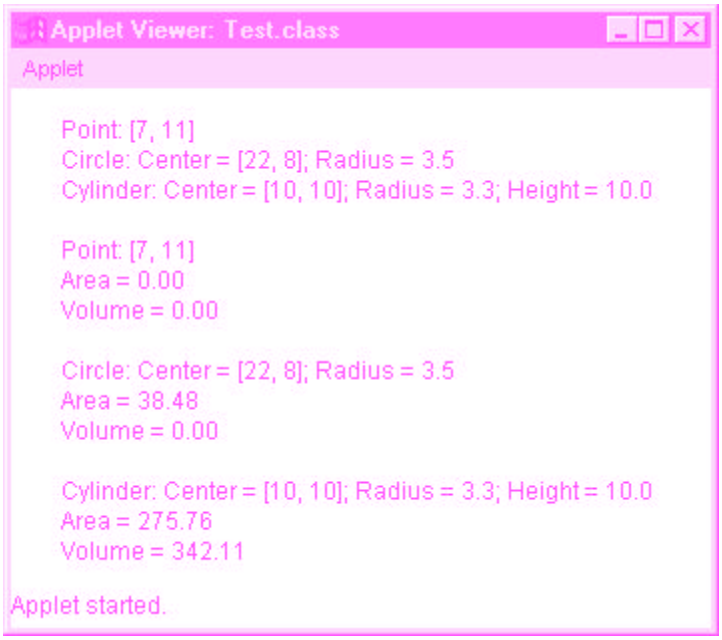


Fig. 7.10 Shape, point, circle, cylinder hierarchy (part 6 of 6).

```
1  // Fig. 7.11: Shape.java
2  // Definition of interface Shape
3
4  interface Shape {
5     double area();
6     double volume();
7     String getName();
8  }
```

Fig. 7.11 Point, circle, cylinder hierarchy with a **Shape** interface (part 1 of 6).

```
9  // Fig. 7.11: Point.java
10 // Definition of class Point
11
12 public class Point implements Shape {
13    protected int x, y; // coordinates of the Point
14
15    // no-argument constructor
16    public Point() { setPoint( 0, 0 ); }
17
18    // constructor
19    public Point( int a, int b ) { setPoint( a, b ); }
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
38     // return the area
39     public double area() { return 0.0; }
40
41     // return the volume
42     public double volume() { return 0.0; }
43
44     // return the class name
45     public String getName() { return "Point"; }
46  }
```

Fig. 7.11 Point, circle, cylinder hierarchy with a **Shape** interface (part 2 of 6).

```
47  // Fig. 7.11: Circle.java
48  // Definition of class Circle
49
50  public class Circle extends Point {  // inherits from Point
51     protected double radius;
52
53     // no-argument constructor
54     public Circle()
55     {
56        // implicit call to superclass constructor here
57        setRadius( 0 );
58     }
59
60     // Constructor
61     public Circle( double r, int a, int b )
62     {
63        super( a, b );  // call the superclass constructor
64        setRadius( r );
65     }
66
67     // Set radius of Circle
68     public void setRadius( double r )
69        { radius = ( r >= 0 ? r : 0 ); }
70
71     // Get radius of Circle
72     public double getRadius() { return radius; }
73
74     // Calculate area of Circle
75     public double area() { return Math.PI * radius * radius; }
76
77     // convert the Circle to a String
78     public String toString()
79        { return "Center = " + super.toString() +
80                 "; Radius = " + radius; }
81
82     // return the class name
83     public String getName() { return "Circle"; }
84  }
```

Fig. 7.11 Point, circle, cylinder hierarchy with a **Shape** interface (part 3 of 6).

```
85  // Fig. 7.11: Cylinder.java
86  // Definition of class Cylinder
```

```
87
88  public class Cylinder extends Circle {
89     protected double height;  // height of Cylinder
90
91     // no-argument constructor
92     public Cylinder()
93     {
94        // implicit call to superclass constructor here
95        setHeight( 0 );
96     }
97
98     // constructor
99     public Cylinder( double h, double r, int a, int b )
100    {
101       super( r, a, b );   // call superclass constructor
102       setHeight( h );
103    }
104
105    // Set height of Cylinder
106    public void setHeight( double h )
107       { height = ( h >= 0 ? h : 0 ); }
108
109    // Get height of Cylinder
110    public double getHeight() { return height; }
111
112    // Calculate area of Cylinder (i.e., surface area)
113    public double area()
114    {
115       return 2 * super.area() +
116              2 * Math.PI * radius * height;
117    }
118
119    // Calculate volume of Cylinder
120    public double volume() { return super.area() * height; }
121
122    // Convert a Cylinder to a String
123    public String toString()
124       { return super.toString() + "; Height = " + height; }
125
126    // Return the class name
127    public String getName() { return "Cylinder"; }
128 }
```

Fig. 7.11 Point, circle, cylinder hierarchy with a **Shape** interface (part 4 of 6).

```
129 // Fig. 7.11: Test.java
130 // Driver for point, circle, cylinder hierarchy
131 import java.awt.Graphics;
132 import java.applet.Applet;
133 import java.text.DecimalFormat;
134
135 public class Test extends Applet {
136    private Point point;
137    private Circle circle;
138    private Cylinder cylinder;
139    private Shape arrayOfShapes[];
140
141    public void init()
142    {
143       point = new Point( 7, 11 );
144       circle = new Circle( 3.5, 22, 8 );
145       cylinder = new Cylinder( 10, 3.3, 10, 10 );
146
147       arrayOfShapes = new Shape[ 3 ];
```

```
148
149         // aim arrayOfShapes[0] at subclass Point object
150         // aim arrayOfShapes[1] at subclass Circle object
151         // aim arrayOfShapes[2] at subclass Cylinder object
152         arrayOfShapes[ 0 ] = point;
153         arrayOfShapes[ 1 ] = circle;
154         arrayOfShapes[ 2 ] = cylinder;
155      }
156
157      public void paint( Graphics g )
158      {
159         g.drawString( point.getName() + ": " +
160                       point.toString(), 25, 25 );
161
162         g.drawString( circle.getName() + ": " +
163                       circle.toString(), 25, 40 );
164
165         g.drawString( cylinder.getName() + ": " +
166                       cylinder.toString(), 25, 55 );
167
168         DecimalFormat precision2 = new DecimalFormat( "#0.00" );
169         int yPos = 85;
170
171         // Loop through arrayOfShapes and print the name,
172         // area, and volume of each object.
173         for ( int i = 0; i < arrayOfShapes.length; i++ ) {
174            g.drawString( arrayOfShapes[ i ].getName() + ": " +
175                          arrayOfShapes[ i ].toString(),
176                          25, yPos );
177            yPos += 15;
```

Fig. 7.11 Point, circle, cylinder hierarchy with a **Shape** interface (part 5 of 6).

```
178            g.drawString( "Area = " + precision2.format(
179                          arrayOfShapes[ i ].area() ),
180                          25, yPos );
181            yPos += 15;
182            g.drawString( "Volume = " + precision2.format(
183                          arrayOfShapes[ i ].volume() ),
184                          25, yPos );
185            yPos += 30;
186         }
187      }
188   }
```

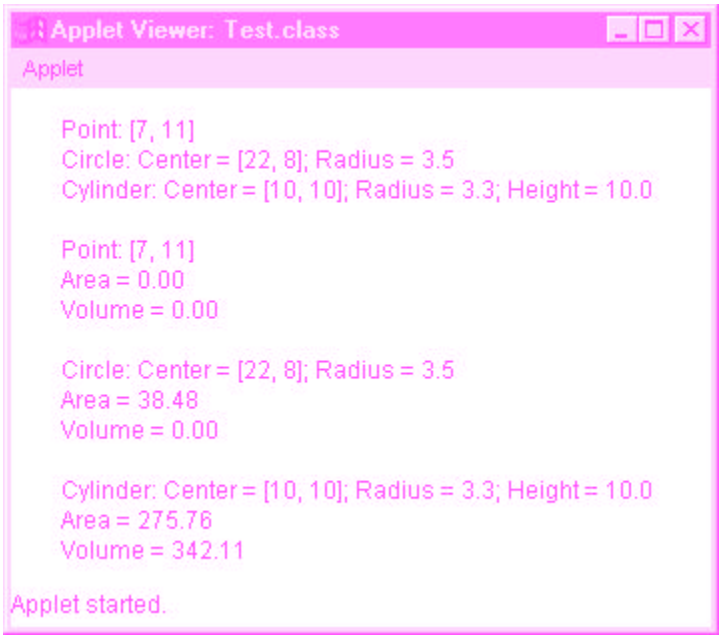


Fig. 7.11 Point, circle, cylinder hierarchy with a **Shape** interface (part 6 of 6).