9

Object-Oriented Programming: Inheritance

Introducing
JAVA SE 6 (Mustang) • Web Services • Ajax-Enabled Web Applications • JavaServer Faces

Say not you know another entirely, till you have divided an inheritance with him.

—Johann Kasper Lavater

This method is to define as the number of a class the class of all classes similar to the given class.

—Bertrand Russell

Good as it is to inherit a library, it is better to collect one.

—Augustine Birrell

Save base authority from others' books.

—William Shakespeare

OBJECTIVES

In this chapter you will learn:

- How inheritance promotes software reusability.
- The notions of superclasses and subclasses.
- To use keyword **`extends`** to create a class that inherits attributes and behaviors from another class.
- To use access modifier **`protected`** to give subclass methods access to superclass members.
- To access superclass members with **`super`**.
- How constructors are used in inheritance hierarchies.
- The methods of class `Object`, the direct or indirect superclass of all classes in Java.

Self-Review Exercises

9.1 Fill in the blanks in each of the following statements:

a) __________ is a form of software reusability in which new classes acquire the members of existing classes and embellish those classes with new capabilities.

ANS: Inheritance

b) A superclass's __________ members can be accessed in the superclass declaration and in subclass declarations.

ANS: `public` and `protected`.

c) In a(n) __________ relationship, an object of a subclass can also be treated as an object of its superclass.

ANS: "is-a" or inheritance

d) In a(n) __________ relationship, a class object has references to objects of other classes as members.

ANS: "has-a" or composition

e) In single inheritance, a class exists in a(n) __________ relationship with its subclasses.

ANS: hierarchical

f) A superclass's __________ members are accessible anywhere that the program has a reference to an object of that superclass or to an object of one of its subclasses.

ANS: `public`

g) When an object of a subclass is instantiated, a superclass __________ is called implicitly or explicitly.

ANS: constructor

h) Subclass constructors can call superclass constructors via the __________ keyword.

ANS: `super`

9.2 State whether each of the following is *true* or *false*. If a statement is *false*, explain why.

a) Superclass constructors are not inherited by subclasses.

ANS: True.

b) A "has-a" relationship is implemented via inheritance.

ANS: False. A "has-a" relationship is implemented via composition. An "is-a" relationship is implemented via inheritance.

c) A `Car` class has an "is-a" relationship with the `SteeringWheel` and `Brakes` classes.

ANS: False. This is an example of a "has-a" relationship. Class `Car` has an "is-a" relationship with class `Vehicle`.

d) Inheritance encourages the reuse of proven high-quality software.

ANS: True.

e) When a subclass redefines a superclass method by using the same signature, the subclass is said to overload that superclass method.

ANS: False. This is known as overriding, not overloading—an overloaded method has the same name, but a different signature.

Exercises

9.3 Many programs written with inheritance could be written with composition instead, and vice versa. Rewrite class `BasePlusCommissionEmployee4` (Fig. 9.13) of the `CommissionEmployee3`–`BasePlusCommissionEmployee4` hierarchy to use composition rather than inheritance. After you do this, assess the relative merits of the two approaches for the `CommissionEmployee3` and `BasePlusCommissionEmployee4` problems, as well as for object-oriented programs in general. Which approach is more natural? Why?

ANS: For a relatively short program like this one, either approach is acceptable. But as programs become larger with more and more objects being instantiated, inheritance becomes preferable because it makes the program easier to modify and promotes the reuse of code. The inheritance approach is more natural because a base-salaried commission employee *is a* commission employee. Composition is defined by the "has-a" relationship, and clearly it would be strange to say that "a base-salaried commission employee *has a* commission employee."

```java
// Exercise 9.3 Solution: BasePlusCommissionEmployee4.java
// BasePlusCommissionEmployee4 using composition.

public class BasePlusCommissionEmployee4
{
   private CommissionEmployee3 commissionEmployee; // composition
   private double baseSalary; // base salary per week

   // six-argument constructor
   public BasePlusCommissionEmployee4( String first, String last,
      String ssn, double sales, double rate, double salary )
   {
      commissionEmployee =
         new CommissionEmployee3( first, last, ssn, sales, rate );
      setBaseSalary( salary ); // validate and store base salary
   } // end six-argument BasePlusCommissionEmployee4 constructor

   // set first name
   public void setFirstName( String first )
   {
      commissionEmployee.setFirstName( first );
   } // end method setFirstName

   // return first name
   public String getFirstName()
   {
      return commissionEmployee.getFirstName();
   } // end method getFirstName

   // set last name
   public void setLastName( String last )
   {
      commissionEmployee.setLastName( last );
   } // end method setLastName

   // return last name
   public String getLastName()
   {
      return commissionEmployee.getLastName();
   } // end method getLastName

   // set social security number
   public void setSocialSecurityNumber( String ssn )
   {
      commissionEmployee.setSocialSecurityNumber( ssn );
   } // end method setSocialSecurityNumber

   // return social security number
   public String getSocialSecurityNumber()
   {
      return commissionEmployee.getSocialSecurityNumber();
   } // end method getSocialSecurityNumber

   // set commission employee's gross sales amount
   public void setGrossSales( double sales )
   {
      commissionEmployee.setGrossSales( sales );
   } // end method setGrossSales

   // return commission employee's gross sales amount
   public double getGrossSales()
   {
      return commissionEmployee.getGrossSales();
   } // end method getGrossSales

   // set commission employee's rate
   public void setCommissionRate( double rate )
   {
      commissionEmployee.setCommissionRate( rate );
   } // end method setCommissionRate

   // return commission employee's rate
   public double getCommissionRate()
   {
      return commissionEmployee.getCommissionRate();
   } // end method getCommissionRate

   // set base-salaried commission employee's base salary
   public void setBaseSalary( double salary )
   {
      baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
   } // end method setBaseSalary

   // return base-salaried commission employee's base salary
   public double getBaseSalary()
   {
      return baseSalary;
   } // end method getBaseSalary

   // calculate base-salaried commission employee's earnings
   public double earnings()
   {
      return getBaseSalary() + commissionEmployee.earnings();
   } // end method earnings

   // return String representation of BasePlusCommissionEmployee4
```

```
    public String toString()
    {
       return String.format( "%s %s\n%s: %.2f", "base-salaried",
          commissionEmployee.toString(), "base salary", getBaseSalary() );
    } // end method toString
} // end class BasePlusCommissionEmployee4
```

9.4 Discuss the ways in which inheritance promotes software reuse, saves time during program development and helps prevent errors.

ANS: Inheritance allows developers to create subclasses that reuse code declared already in a superclass. Avoiding the duplication of common functionality between several classes by building an inheritance hierarchy to contain the classes can save developers a considerable amount of time. Similarly, placing common functionality in a single superclass, rather than duplicating the code in multiple unrelated classes, helps prevent the same errors from appearing in multiple source-code files. If several classes each contain duplicate code containing an error, the software developer has to spend time correcting each source-code file with the error. However, if these classes take advantage of inheritance, and the error occurs in the common functionality of the superclass, the software developer needs to modify only the superclass's source-code file.

9.5 Draw an inheritance hierarchy for students at a university similar to the hierarchy shown in Fig. 9.2. Use `Student` as the superclass of the hierarchy, then extend `Student` with classes `UndergraduateStudent` and `GraduateStudent`. Continue to extend the hierarchy as deep (i.e., as many levels) as possible. For example, `Freshman`, `Sophomore`, `Junior` and `Senior` might extend `UndergraduateStudent`, and `DoctoralStudent` and `MastersStudent` might be subclasses of `GraduateStudent`. After drawing the hierarchy, discuss the relationships that exist between the classes. [*Note:* You do not need to write any code for this exercise.]

ANS:

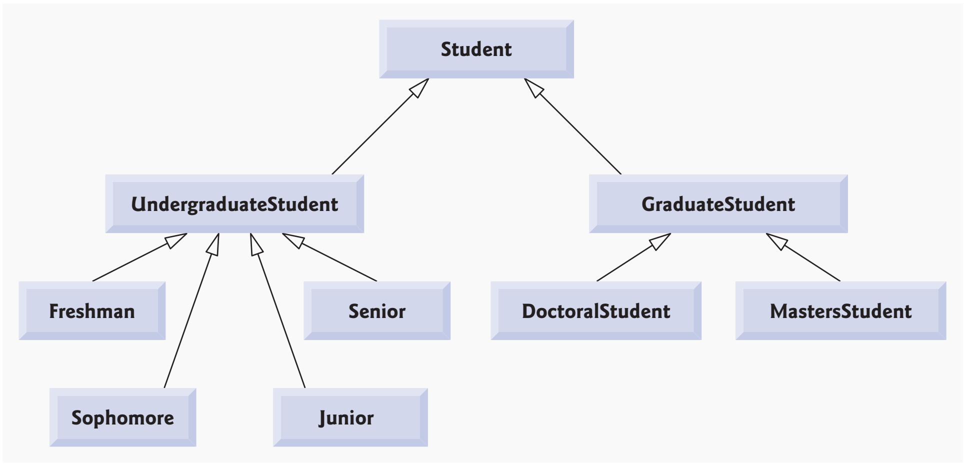


Fig. 9.1 |

This hierarchy contains many "is-a" (inheritance) relationships. An `UndergraduateStudent` *is a* `Student`. A `GraduateStudent` *is a* `Student` too. Each of the classes `Freshman`, `Sophomore`, `Junior` and Senior *is an* UndergraduateStudent and *is a* Student. Each of the classes DoctoralStudent and MastersStudent *is a* GraduateStudent and *is a* Student.

9.6 The world of shapes is much richer than the shapes included in the inheritance hierarchy of Fig. 9.3. Write down all the shapes you can think of—both two-dimensional and three-dimensional—and form them into a more complete Shape hierarchy with as many levels as possible. Your hierarchy should have class Shape at the top. Class TwoDimensionalShape and class ThreeDimensionalShape should extend Shape. Add additional subclasses, such as Quadrilateral and Sphere, at their correct locations in the hierarchy as necessary.

ANS: [*Note:* Solutions may vary.]

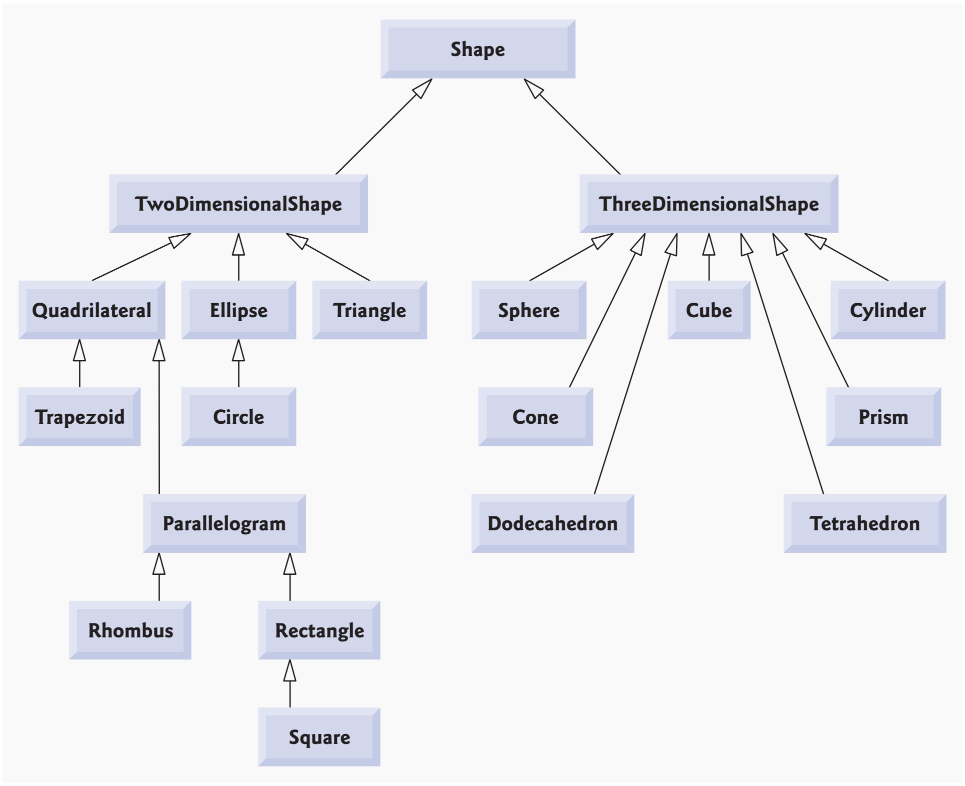


Fig. 9.2 |

9.7 Some programmers prefer not to use protected access, because they believe it breaks the encapsulation of the superclass. Discuss the relative merits of using protected access vs. using private access in superclasses.

ANS: private instance variables are hidden in the subclass and are accessible only through the public or protected methods of the superclass. Using protected access enables the subclass to manipulate the protected members without using the access methods of the superclass. If the superclass members are private, the methods of the superclass must be used to access the data. This may result in a decrease in performance due to the extra method calls.

9.8 Write an inheritance hierarchy for classes `Quadrilateral`, `Trapezoid`, `Parallelogram`, `Rectangle` and `Square`. Use `Quadrilateral` as the superclass of the hierarchy. Make the hierarchy as deep (i.e., as many levels) as possible. Specify the instance variables and methods for each class. The `private` instance variables of `Quadrilateral` should be the *x-y* coordinate pairs for the four end-points of the `Quadrilateral`. Write a program that instantiates objects of your classes and outputs each object's area (except `Quadrilateral`).

ANS:

```java
// Exercise 9.8 Solution: QuadrilateralTest.java
// Driver for Exercise 9.8

public class QuadrilateralTest
{
   public static void main( String args[] )
   {
      // NOTE: All coordinates are assumed to form the proper shapes
      // A quadrilateral is a four-sided polygon
      Quadrilateral quadrilateral =
         new Quadrilateral( 1.1, 1.2, 6.6, 2.8, 6.2, 9.9, 2.2, 7.4 );

      // A trapezoid is a quadrilateral having exactly two parallel sides
      Trapezoid trapezoid =
         new Trapezoid( 0.0, 0.0, 10.0, 0.0, 8.0, 5.0, 3.3, 5.0 );

      // A parallelogram is a quadrilateral with opposite sides parallel
      Parallelogram parallelogram =
         new Parallelogram( 5.0, 5.0, 11.0, 5.0, 12.0, 20.0, 6.0, 20.0 );

      // A rectangle is an equiangular parallelogram
      Rectangle rectangle =
         new Rectangle( 17.0, 14.0, 30.0, 14.0, 30.0, 28.0, 17.0, 28.0 );

      // A square is an equiangular and equilateral parallelogram
      Square square =
         new Square( 4.0, 0.0, 8.0, 0.0, 8.0, 4.0, 4.0, 4.0 );

      System.out.printf(
         "%s %s %s %s %s\n", quadrilateral, trapezoid, parallelogram,
         rectangle, square );
   } // end main
} // end class QuadrilateralTest
```

```
Coordinates of Quadrilateral are:
( 1.1, 1.2 ), ( 6.6, 2.8 ), ( 6.2, 9.9 ), ( 2.2, 7.4 )

Coordinates of Trapezoid are:
( 0.0, 0.0 ), ( 10.0, 0.0 ), ( 8.0, 5.0 ), ( 3.3, 5.0 )
Height is: 5.0
Area is: 36.75

Coordinates of Parallelogram are:
( 5.0, 5.0 ), ( 11.0, 5.0 ), ( 12.0, 20.0 ), ( 6.0, 20.0 )
Width is: 6.0
Height is: 15.0
Area is: 90.0

Coordinates of Rectangle are:
( 17.0, 14.0 ), ( 30.0, 14.0 ), ( 30.0, 28.0 ), ( 17.0, 28.0 )
Width is: 13.0
Height is: 14.0
Area is: 182.0

Coordinates of Square are:
( 4.0, 0.0 ), ( 8.0, 0.0 ), ( 8.0, 4.0 ), ( 4.0, 4.0 )
Side is: 4.0
Area is: 16.0
```

```java
// Exercise 9.8 Solution: Point.java
// Class Point definition

public class Point
{
   private double x; // x coordinate
   private double y; // y coordinate

   // two-argument constructor
   public Point( double xCoordinate, double yCoordinate )
   {
      x = xCoordinate; // set x
      y = yCoordinate; // set y
   } // end two-argument Point constructor

   // return x
   public double getX()
   {
      return x;
   } // end method getX

   // return y
   public double getY()
   {
      return y;
   } // end method getY

   // return string representation of Point object
   public String toString()
   {
      return String.format( "( %.1f, %.1f )", getX(), getY() );
   } // end method toString
} // end class Point
```

```
// Exercise 9.8 Solution: Quadrilateral.java
// Class Quadrilateral definition

public class Quadrilateral
{
   private Point point1; // first endpoint
   private Point point2; // second endpoint
   private Point point3; // third endpoint
   private Point point4; // fourth endpoint

   // eight-argument constructor
   public Quadrilateral( double x1, double y1, double x2, double y2,
      double x3, double y3, double x4, double y4 )
   {
      point1 = new Point( x1, y1 );
      point2 = new Point( x2, y2 );
      point3 = new Point( x3, y3 );
      point4 = new Point( x4, y4 );
   } // end eight-argument Quadrilateral constructor

   // return point1
   public Point getPoint1()
   {
      return point1;
   } // end method getPoint1

   // return point2
   public Point getPoint2()
   {
      return point2;
   } // end method getPoint2

   // return point3
   public Point getPoint3()
   {
      return point3;
   } // end method getPoint3

   // return point4
   public Point getPoint4()
   {
      return point4;
   } // end method getPoint4

   // return string representation of a Quadrilateral object
   public String toString()
   {
```

```java
      return String.format( "%s:\n%s",
         "Coordinates of Quadrilateral are", getCoordinatesAsString() );
   } // end method toString

   // return string containing coordinates as strings
   public String getCoordinatesAsString()
   {
      return String.format(
         "%s, %s, %s, %s\n", point1, point2, point3, point4 );
   } // end method getCoordinatesAsString
} // end class Quadrilateral
```

```java
// Exercise 9.8 Solution: Trapezoid.java
// Class Trapezoid definition

public class Trapezoid extends Quadrilateral
{
   private double height; // height of trapezoid

   // eight-argument constructor
   public Trapezoid( double x1, double y1, double x2, double y2,
      double x3, double y3, double x4, double y4 )
   {
      super( x1, y1, x2, y2, x3, y3, x4, y4 );
   } // end of eight-argument Trapezoid constructor

   // return height
   public double getHeight()
   {
      if ( getPoint1().getY() == getPoint2().getY() )
         return Math.abs( getPoint2().getY() - getPoint3().getY() );
      else
         return Math.abs( getPoint1().getY() - getPoint2().getY() );
   } // end method getHeight

   // return area
   public double getArea()
   {
      return getSumOfTwoSides() * getHeight() / 2.0;
   } // end method getArea

   // return the sum of the trapezoid's two sides
   public double getSumOfTwoSides()
   {
      if ( getPoint1().getY() == getPoint2().getY() )
         return Math.abs( getPoint1().getX() - getPoint2().getX() ) +
            Math.abs( getPoint3().getX() - getPoint4().getX() );
      else
         return Math.abs( getPoint2().getX() - getPoint3().getX() ) +
            Math.abs( getPoint4().getX() - getPoint1().getX() );
   } // end method getSumOfTwoSides

   // return string representation of Trapezoid object
   public String toString()
   {
      return String.format( "\n%s:\n%s%s: %s\n%s: %s\n",
         "Coordinates of Trapezoid are", getCoordinatesAsString(),
         "Height is", getHeight(), "Area is", getArea() );
   } // end method toString
} // end class Trapezoid
```

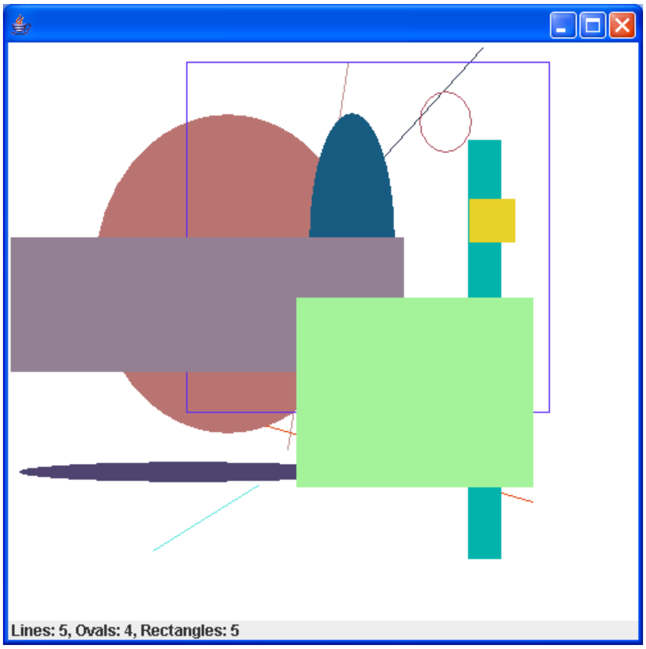
```java
// Exercise 9.8 Solution: Parallelogram.java
// Class Parallelogram definition

public class Parallelogram extends Trapezoid
{
   // eight-argument constructor
   public Parallelogram( double x1, double y1, double x2, double y2,
      double x3, double y3, double x4, double y4 )
   {
      super( x1, y1, x2, y2, x3, y3, x4, y4 );
   } // end eight-argument Parallelogram constructor

   // return width of parallelogram
   public double getWidth()
   {
      if ( getPoint1().getY() == getPoint2().getY() )
         return Math.abs( getPoint1().getX() - getPoint2().getX() );
      else
         return Math.abs( getPoint2().getX() - getPoint3().getX() );
   } // end method getWidth

   // return string representation of Parallelogram object
   public String toString()
   {
      return String.format( "\n%s:\n%s%s: %s\n%s: %s\n%s: %s\n",
         "Coordinates of Parallelogram are", getCoordinatesAsString(),
         "Width is", getWidth(), "Height is", getHeight(),
         "Area is", getArea() );
   } // end method toString
} // end class Parallelogram
```

```java
// Exercise 9.8 Solution: Rectangle.java
// Class Rectangle definition

public class Rectangle extends Parallelogram
{
   // eight-argument constructor
   public Rectangle( double x1, double y1, double x2, double y2,
      double x3, double y3, double x4, double y4 )
   {
      super( x1, y1, x2, y2, x3, y3, x4, y4 );
   } // end eight-argument Rectangle constructor

   // return string representation of Rectangle object
   public String toString()
   {
      return String.format( "\n%s:\n%s%s: %s\n%s: %s\n%s: %s\n",
         "Coordinates of Rectangle are", getCoordinatesAsString(),
         "Width is", getWidth(), "Height is", getHeight(),
         "Area is", getArea() );
   } // end method toString
} // end class Rectangle
```

```java
// Exercise 9.8 Solution: Square.java
// Class Square definition

public class Square extends Parallelogram
{
   // eight-argument constructor
   public Square( double x1, double y1, double x2, double y2,
      double x3, double y3, double x4, double y4 )
   {
      super( x1, y1, x2, y2, x3, y3, x4, y4 );
   } // end eight-argument Square constructor

   // return string representation of Square object
   public String toString()
   {
      return String.format( "\n%s:\n%s%s: %s\n%s: %s\n",
         "Coordinates of Square are", getCoordinatesAsString(),
         "Side is", getHeight(), "Area is", getArea() );
   } // end method toString
} // end class Square
```

(Optional) GUI and Graphics Case Study

9.1 Modify Exercise 8.1 to include a `JLabel` as a status bar that displays counts representing the number of each shape displayed. Class `DrawPanel` should declare a method that returns a `String` containing the status text. In `main`, first create the `DrawPanel`, then create the `JLabel` with the status text as an argument to the `JLabel`'s constructor. Attach the `JLabel` to the `SOUTH` region of the `JFrame`, as shown in Fig. 9.3.



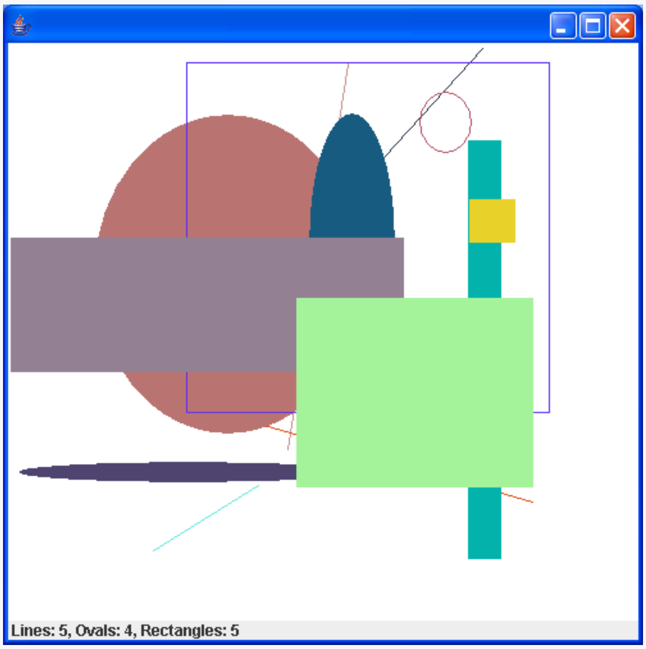
```
Lines: 5, Ovals: 4, Rectangles: 5
```

Fig. 9.3 | `JLabel` displaying shape statistics.

ANS: (Uses classes `MyLine`, `MyRect`, and `MyOval` from GUI and Graphics Case Study Exercise 8.1)

```java
// GCS Exercise 9.1 Solution: DrawPanel.java
// Program that uses classes MyLine, MyOval and MyRect to draw
// random shapes
import java.awt.Color;
import java.awt.Graphics;
import java.util.Random;
import javax.swing.JPanel;

public class DrawPanel extends JPanel
{
   private Random randomNumbers = new Random();

   private MyLine lines[]; // array on lines
   private MyOval ovals[]; // array of ovals
   private MyRect rectangles[]; // array of rectangles

   // String containing shape statistic information
   private String statusText;

   // constructor, creates a panel with random shapes
   public DrawPanel()
   {
      setBackground( Color.WHITE );

      lines = new MyLine[ 1 + randomNumbers.nextInt( 5 ) ];
      ovals = new MyOval[ 1 + randomNumbers.nextInt( 5 ) ];
      rectangles = new MyRect[ 1 + randomNumbers.nextInt( 5 ) ];

      // create lines
      for ( int count = 0; count < lines.length; count++ )
      {
         // generate random coordinates
         int x1 = randomNumbers.nextInt( 450 );
         int y1 = randomNumbers.nextInt( 450 );
         int x2 = randomNumbers.nextInt( 450 );
         int y2 = randomNumbers.nextInt( 450 );

         // generate a random color
         Color color = new Color( randomNumbers.nextInt( 256 ),
            randomNumbers.nextInt( 256 ), randomNumbers.nextInt( 256 ) );

         // add the line to the list of lines to be displayed
         lines[ count ] = new MyLine( x1, y1, x2, y2, color );
      } // end for

      // create ovals
      for ( int count = 0; count < ovals.length; count++ )
      {
         // generate random coordinates
         int x1 = randomNumbers.nextInt( 450 );
         int y1 = randomNumbers.nextInt( 450 );
         int x2 = randomNumbers.nextInt( 450 );
         int y2 = randomNumbers.nextInt( 450 );

         // generate a random color
         Color color = new Color( randomNumbers.nextInt( 256 ),
            randomNumbers.nextInt( 256 ), randomNumbers.nextInt( 256 ) );

         // get filled property
         boolean filled = randomNumbers.nextBoolean();

         // add the oval to the list of ovals to be displayed
         ovals[ count ] = new MyOval( x1, y1, x2, y2, color, filled );
      } // end for

      // create rectangles
      for ( int count = 0; count < rectangles.length; count++ )
      {
         // generate random coordinates
         int x1 = randomNumbers.nextInt( 450 );
         int y1 = randomNumbers.nextInt( 450 );
         int x2 = randomNumbers.nextInt( 450 );
         int y2 = randomNumbers.nextInt( 450 );

         // generate a random color
         Color color = new Color( randomNumbers.nextInt( 256 ),
            randomNumbers.nextInt( 256 ), randomNumbers.nextInt( 256 ) );

         // get filled property
         boolean filled = randomNumbers.nextBoolean();

         // add the rectangle to the list of rectangles to be displayed
         rectangles[ count ] =
            new MyRect( x1, y1, x2, y2, color, filled );
      } // end for

      // create the status bar text
      statusText = String.format( " %s: %d, %s: %d, %s: %d",
         "Lines", lines.length, "Ovals", ovals.length,
         "Rectangles", rectangles.length );
   } // end DrawPanel constructor

   // returns a String containing shape statistics on this panel
   public String getLabelText()
   {
      return statusText;
   } // end method getLabelText

   // for each shape array, draw the individual shapes
   public void paintComponent( Graphics g )
   {
      super.paintComponent( g );

      // draw the lines
      for ( MyLine line : lines )
         line.draw( g );

      // draw the ovals
      for ( MyOval oval: ovals )
         oval.draw( g );

      // drat the rectangles
      for ( MyRect rectangle : rectangles )
         rectangle.draw( g );
   } // end method paintComponent
} // end class DrawPanel
```

```java
// GCS Exercise 9.1 Solution: TestDraw.java
// Test application to display a DrawPanel
import java.awt.BorderLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;

public class TestDraw
{
   public static void main( String args[] )
   {
      DrawPanel panel = new DrawPanel();
      JFrame application = new JFrame();

      // create a JLabel containing the shape information
      JLabel statusLabel = new JLabel( panel.getLabelText() );

      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

      application.add( panel ); // add drawing to CENTER by default

      // add the status label to the SOUTH (bottom) of the frame
      application.add( statusLabel, BorderLayout.SOUTH );

      application.setSize( 500, 500 );
      application.setVisible( true );
   } // end main
} // end class TestDraw
```



```
Lines: 5, Ovals: 4, Rectangles: 5
```