

Deitel & Deitel Book

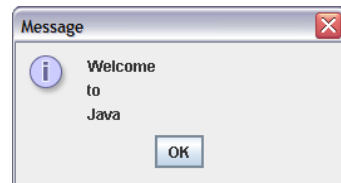
All code:

```
/*
 * (C) Copyright 1992-2005 by Deitel & Associates, Inc. and
 * Pearson Education, Inc. All Rights Reserved.
 *
 * DISCLAIMER: The authors and publisher of this book have used their
 * best efforts in preparing the book. These efforts include the
 * development, research, and testing of the theories and programs
 * to determine their effectiveness. The authors and publisher make
 * no warranty of any kind, expressed or implied, with regard to these
 * programs or to the documentation contained in these books. The authors
 * and publisher shall not be liable in any event for incidental or
 * consequential damages in connection with, or arising out of, the
 * furnishing, performance, or use of these programs.
 */
```

Section 3.9

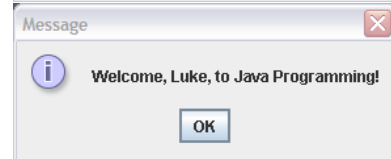
// Fig. 3.17: Dialog1.java Printing multiple lines in dialog box.

```
import javax.swing.JOptionPane; // import class JOptionPane
public class Dialog1 {
    public static void main( String args[] ) {
        // display a dialog with the message
        JOptionPane.showMessageDialog( null, "Welcome\nto\nJava" );
    } // end main
} // end class Dialog1
```



// Fig. 3.18: NameDialog.java Basic input with a dialog box.

```
import javax.swing.JOptionPane;
public class NameDialog {
    public static void main( String args[] ) {
        // prompt user to enter name
        String name =
            JOptionPane.showInputDialog( "What is your name?" );
        // create the message
        String message =
            String.format( "Welcome, %s, to Java Programming!", name );
        // display the message to welcome the user by name
        JOptionPane.showMessageDialog( null, message );
    } // end main } // end class NameDialog
```



Section 4.14

// Fig. 4.19: DrawPanel.java

```
// Draws two crossing lines on a panel.
import java.awt.Graphics;
import javax.swing.JPanel;
public class DrawPanel extends JPanel
{
    // draws an X from the corners of the panel
    public void paintComponent( Graphics g )
    {
        // call paintComponent to ensure the panel displays correctly
        super.paintComponent( g );

        int width = getWidth(); // total width
        int height = getHeight(); // total height
        // draw a line from the upper-left to the lower-right
        g.drawLine( 0, 0, width, height );
        // draw a line from the lower-left to the upper-right
        g.drawLine( 0, height, width, 0 );
    } // end method paintComponent
} // end class DrawPanel
```

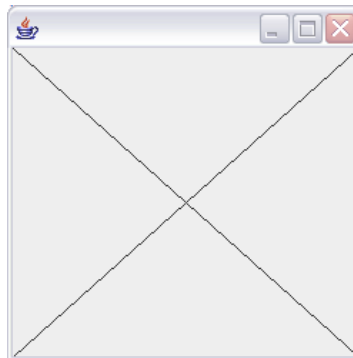
// Fig. 4.20: DrawPanelTest.java

```
// Application to display a DrawPanel.
import javax.swing.JFrame;
public class DrawPanelTest
{
    public static void main( String args[] )
    {
        // create a panel that contains our drawing
        DrawPanel panel = new DrawPanel();

        // create a new frame to hold the panel
        JFrame application = new JFrame();

        // set the frame to exit when it is closed
        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

        application.add( panel ); // add the panel to the frame
        application.setSize( 250, 250 ); // set the size of the frame
        application.setVisible( true ); // make the frame visible
    } // end main
} // end class DrawPanelTest
```

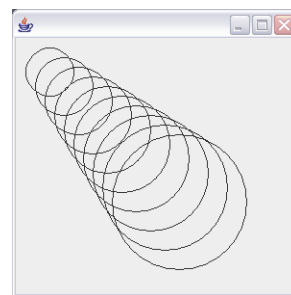
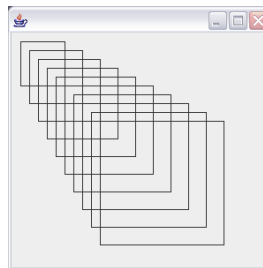
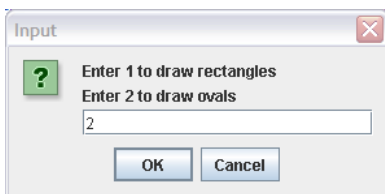
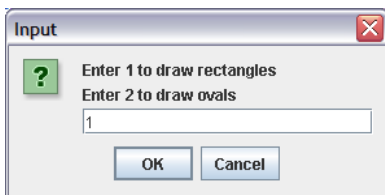


Section 5.10

// Fig. 5.26: Shapes.java

```
// Demonstrates drawing different shapes.
import java.awt.Graphics;
import javax.swing.JPanel;

public class Shapes extends JPanel
{
    private int choice; // user's choice of which shape to draw
    // constructor sets the user's choice
    public Shapes( int userChoice )
    {
        choice = userChoice;
    } // end Shapes constructor
    // draws a cascade of shapes starting from the top left corner
    public void paintComponent( Graphics g )
    {
        super.paintComponent( g );
        for ( int i = 0; i < 10; i++ )
        {
            // pick the shape based on the user's choice
            switch ( choice )
            {
                case 1: // draw rectangles
                    g.drawRect( 10 + i * 10, 10 + i * 10,
                               50 + i * 10, 50 + i * 10 );
                    break;
                case 2: // draw ovals
                    g.drawOval( 10 + i * 10, 10 + i * 10,
                               50 + i * 10, 50 + i * 10 );
                    break;
            } // end switch
        } // end for
    } // end method paintComponent
} // end class Shapes
```



// Fig. 5.27: ShapesTest.java

```
// Test application that displays class Shapes.
import javax.swing.JFrame;
import javax.swing.JOptionPane;
public class ShapesTest
{
    public static void main( String args[] )
    {
        // obtain user's choice
        String input = JOptionPane.showInputDialog(
            "Enter 1 to draw rectangles\n" +
            "Enter 2 to draw ovals" );
        int choice = Integer.parseInt( input ); // convert input to int
        // create the panel with the user's input
        Shapes panel = new Shapes( choice );
        JFrame application = new JFrame(); // creates a new JFrame
        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        application.add( panel ); // add the panel to the frame
        application.setSize( 300, 300 ); // set the desired size
        application.setVisible( true ); // show the frame
    } // end main
} // end class ShapesTest
```

Section 6.13

// Fig. 6.16: DrawSmiley.java

```
// Demonstrates filled shapes.
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JPanel;
public class DrawSmiley extends JPanel
{
    public void paintComponent( Graphics g )
    {
        super.paintComponent( g );
        // draw the face
        g.setColor( Color.YELLOW );
        g.fillOval( 10, 10, 200, 200 );

        // draw the eyes
        g.setColor( Color.BLACK );
        g.fillOval( 55, 65, 30, 30 );
        g.fillOval( 135, 65, 30, 30 );

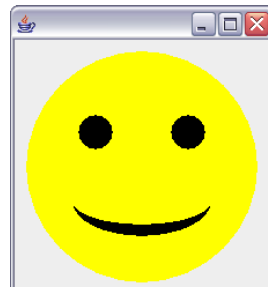
        // draw the mouth
        g.fillOval( 50, 110, 120, 60 );

        // "touch up" the mouth into a smile
        g.setColor( Color.YELLOW );
        g.fillRect( 50, 110, 120, 30 );
        g.fillOval( 50, 120, 120, 40 );
    } // end method paintComponent
} // end class DrawSmiley
```

// Fig. 6.17: DrawSmileyTest.java

```
// Test application that displays a smiley face.
import javax.swing.JFrame;
public class DrawSmileyTest
{
    public static void main( String args[] )
    {
        DrawSmiley panel = new DrawSmiley();
        JFrame application = new JFrame();

        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        application.add( panel );
        application.setSize( 230, 250 );
        application.setVisible( true );
    } // end main
} // end class DrawSmileyTest
```



Section 7.13

// Fig. 7.22: DrawRainbow.java

```
// Demonstrates using colors in an array.
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JPanel;
public class DrawRainbow extends JPanel
{
    // Define indigo and violet
    final Color VIOLET = new Color( 128, 0, 128 );
    final Color INDIGO = new Color( 75, 0, 130 );

    // colors to use in the rainbow, starting from the innermost
    // The two white entries result in an empty arc in the center
    private Color colors[] =
        { Color.WHITE, Color.WHITE, VIOLET, INDIGO, Color.BLUE,
          Color.GREEN, Color.YELLOW, Color.ORANGE, Color.RED };

    // constructor
    public DrawRainbow()
    {
        setBackground( Color.WHITE ); // set the background to white
    } // end DrawRainbow constructor

    // draws a rainbow using concentric circles
    public void paintComponent( Graphics g )
    {
        super.paintComponent( g );

        int radius = 20; // radius of an arch

        // draw the rainbow near the bottom-center
        int centerX = getWidth() / 2;
        int centerY = getHeight() - 10;

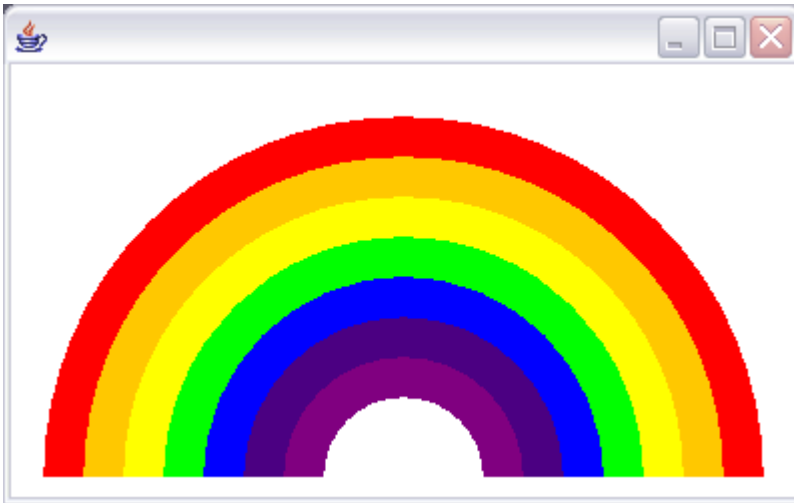
        // draws filled arcs starting with the outermost
        for ( int counter = colors.length; counter > 0; counter-- )
        {
            // set the color for the current arc
            g.setColor( colors[ counter - 1 ] );

            // fill the arc from 0 to 180 degrees
            g.fillArc( centerX - counter * radius,
                      centerY - counter * radius,
                      counter * radius * 2, counter * radius * 2, 0, 180 );
        } // end for
    } // end method paintComponent
} // end class DrawRainbow
```

// Fig. 7.23: DrawRainbowTest.java

```
// Test application to display a rainbow.
import javax.swing.JFrame;
public class DrawRainbowTest
{
    public static void main( String args[] )
    {
        DrawRainbow panel = new DrawRainbow();
        JFrame application = new JFrame();

        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        application.add( panel );
        application.setSize( 400, 250 );
        application.setVisible( true );
    } // end main
} // end class DrawRainbowTest
```



Section 8.18

// Fig. 8.22: DrawPanel2.java

```
// Program that uses class MyLine
// to draw random lines.
import java.awt.Color;
import java.awt.Graphics;
import java.util.Random;
import javax.swing.JPanel;

public class DrawPanel2 extends JPanel
{
    private Random randomNumbers = new Random();
    private MyLine lines[]; // array on lines

    // constructor, creates a panel with random shapes
    public DrawPanel2()
    {
        setBackground( Color.WHITE );

        lines = new MyLine[ 5 + randomNumbers.nextInt( 5 ) ];

        // create lines
        for ( int count = 0; count < lines.length; count++ )
        {
            // generate random coordinates
            int x1 = randomNumbers.nextInt( 300 );
            int y1 = randomNumbers.nextInt( 300 );
            int x2 = randomNumbers.nextInt( 300 );
            int y2 = randomNumbers.nextInt( 300 );

            // 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
    } // end DrawPanel2 constructor

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

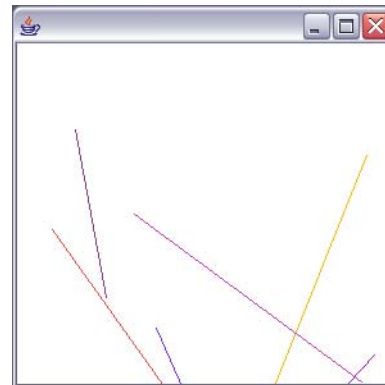
        // draw the lines
        for ( MyLine line : lines )
            line.draw( g );
    } // end method paintComponent
} // end class DrawPanel2
```

// Fig. 8.23: TestDraw.java

```
// Test application to display a DrawPanel2.
import javax.swing.JFrame;

public class TestDraw
{
    public static void main( String args[] )
    {
        DrawPanel2 panel = new DrawPanel2();
        JFrame application = new JFrame();

        application.setDefaultCloseOperation(
JFrame.EXIT_ON_CLOSE );
        application.add( panel );
        application.setSize( 300, 300 );
        application.setVisible( true );
    } // end main
} // end class TestDraw
```



Section 9.8

// Fig 9.19: LabelDemo.java

```
// Demonstrates the use of labels.
import java.awt.BorderLayout;
import javax.swing.ImageIcon;
import javax.swing.JLabel;
import javax.swing.JFrame;
public class LabelDemo
{
    public static void main( String args[] )
    {
        // Create a label with plain text
        JLabel northLabel = new JLabel( "North" );

        // create an icon from an image so we can put it on a JLabel
        ImageIcon labelIcon = new ImageIcon(
"C:\\eclipse\\workspace\\Swing\\src\\GUItip.gif" );
        // create a label with an Icon instead of text
        JLabel centerLabel = new JLabel( labelIcon );
        // create another label with an Icon
        JLabel southLabel = new JLabel( labelIcon );
        // set the label to display text (as well as an icon)
        southLabel.setText( "South" );
        // create a frame to hold the labels
        JFrame application = new JFrame();
        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        // add the labels to the frame; the second argument specifies
        // where on the frame to add the label
        application.add( northLabel, BorderLayout.NORTH );
        application.add( centerLabel, BorderLayout.CENTER );
        application.add( southLabel, BorderLayout.SOUTH );
        application.setSize( 300, 300 ); // set the size of the frame
        application.setVisible( true ); // show the frame
    } // end main } // end class LabelDemo
```

