```
// Fig. 10.19: MouseTracker.java
// Demonstrating mouse events.
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class MouseTracker extends Applet
             implements MouseListener, MouseMotionListener {
   private int xPos, yPos = -10;  // off screen for 1st paint
   private String s = "";

   public void init()
   {
      // applet listens to its own mouse events
      addMouseListener( this );
      addMouseMotionListener( this );
   }

   public void paint( Graphics g )
   {
      g.drawString( s + " @ [" + xPos + ", " + yPos + "]",
                    xPos, yPos );
   }

   private void setValues( String event, int x, int y )
   {
      s = event;
      xPos = x;
      yPos = y;
      repaint();
   }

   // MouseListener event handlers
   public void mouseClicked( MouseEvent e )
      { setValues( "Clicked", e.getX(), e.getY() ); }

   public void mousePressed( MouseEvent e )
      { setValues( "Pressed", e.getX(), e.getY() ); }

   public void mouseReleased( MouseEvent e )
      { setValues( "Released", e.getX(), e.getY() ); }

   public void mouseEntered( MouseEvent e )
      { showStatus( "Mouse in applet area" ); }

   public void mouseExited( MouseEvent e )
      { showStatus( "Mouse outside applet area" ); }
```

```
   // MouseMotionListener event handlers
   public void mouseDragged( MouseEvent e )
      { setValues( "Dragging", e.getX(), e.getY() ); }

   public void mouseMoved( MouseEvent e )
      { setValues( "Moving", e.getX(), e.getY() ); }
}
```

```java
// Fig. 10.21: Drag.java
// Using the MouseMotionAdapter class.
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class Drag extends Applet {
   private int xValue = -10, yValue = -10;

   public void init()
   {
      addMouseMotionListener( new MotionHandler( this ) );
   }

   public void paint( Graphics g )
   {
      g.drawString( "Drag the mouse to draw", 10, 20 );
      g.fillOval( xValue, yValue, 4, 4 );
   }

   // Override Component class update method to allow all ovals
   // to remain on the screen by not clearing the background.
   public void update( Graphics g ) { paint( g ); }

   // set the drawing coordinates and repaint
   public void setCoordinates( int x, int y )
   {
      xValue = x;
      yValue = y;
      repaint();
   }
}

// Class to handle only mouse drag events for the Drag applet
class MotionHandler extends MouseMotionAdapter {
   private Drag dragger;

   public MotionHandler( Drag d ) { dragger = d; }

   public void mouseDragged( MouseEvent e )
      { dragger.setCoordinates( e.getX(), e.getY() ); }
}
```

```java
// Fig. 10.22: MouseDetails.java
// Demonstrating mouse clicks and distinguishing mouse buttons.
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class MouseDetails extends Applet {
   private String s = "";
   private int xPos, yPos;

   public void init()
   {
      addMouseListener( new DetailHandler( this ) );
   }

   public void paint( Graphics g )
   {
      g.drawString( "Clicked @ [" + xPos + ", " + yPos + "]",
                    xPos, yPos );
   }

   public void setCoordinates( int x, int y )
   {
      xPos = x;
      yPos = y;
      repaint();
   }
}

class DetailHandler extends MouseAdapter {
   private MouseDetails details;

   public DetailHandler( MouseDetails m ) { details = m; }

   public void mouseClicked( MouseEvent e )
   {
      details.setCoordinates( e.getX(), e.getY() );

      String s = "Clicked " + e.getClickCount() + " time(s)";

      if ( e.isMetaDown() )       // Right mouse button
         s += " with right mouse button";
      else if ( e.isAltDown() )   // Middle mouse button
         s += " with center mouse button";
      else                        // Left mouse button
         s += " with left mouse button";

      details.showStatus( s );
```

```
    }
}
```

```
// Fig. 10.24: Key.java
// Demonstrating keystroke events.
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class Key extends Applet implements KeyListener, FocusList
ener {
  private String line1 = "";
  private String line2 = "";
  private String line3 = "";
  private TextArea editField;

  public void init()
  {
    editField = new TextArea("foo", 5, 40);
    // allow applet to process Key events
    editField.addKeyListener( this );
    editField.addFocusListener( this );

    add(editField);

    // make applet the active component for key events
    //requestFocus();
  }

  public void paint( Graphics g )
  {
    g.drawString( line1, 25, 25 );
    g.drawString( line2, 25, 40 );
    g.drawString( line3, 25, 55 );
  }

  public void keyPressed( KeyEvent e )
  {
    line1 = "Key pressed: " +
      e.getKeyText( e.getKeyCode() );
    setLines2and3( e );
  }

  public void keyReleased( KeyEvent e )
  {
    line1 = "Key released: " +
      e.getKeyText( e.getKeyCode() );
    setLines2and3( e );
  }

  public void keyTyped( KeyEvent e )
```

```
  {
    line1 = "Key typed: " + e.getKeyChar();
    System.out.println(line1);
    setLines2and3( e );
  }

  private void setLines2and3( KeyEvent e )
  {
    line2 = "This key is " +
      ( e.isActionKey() ? "" : "not " ) +
      "an action key";

    String temp =
      e.getKeyModifiersText( e.getModifiers() );

    line3 = "Modifier keys pressed: " +
      ( temp.equals( "" ) ? "none" : temp );
    repaint();
    requestFocus();
  }

  public void focusGained( FocusEvent e ) {
    System.out.println("Got focus");
  }

  public void focusLost( FocusEvent e ) {
    System.out.println("Lost focus");
  }
}
```

```
// Fig. 10.27: FlowLayoutDemo.java
// Demonstrating FlowLayout alignments.
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class FlowLayoutDemo extends Applet
             implements ActionListener {
   private Button left, center, right;

   public void init()
   {
      left = new Button( "Left" );
      left.addActionListener( this );
      add( left );

      center = new Button( "Center" );
      center.addActionListener( this );
      add( center );

      right = new Button( "Right" );
      right.addActionListener( this );
      add( right );
   }

   public void actionPerformed( ActionEvent e )
   {
      int align;

      if ( e.getSource() == left )
         align = FlowLayout.LEFT;
      else if ( e.getSource() == center )
         align = FlowLayout.CENTER;
      else
         align = FlowLayout.RIGHT;

      setLayout( new FlowLayout( align ) );
      validate();  // re-align attached components
   }
}
```

```
// Fig. 10.29: Border.java
// Demonstrating BorderLayout.
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class Border extends Applet implements ActionListener {
   private Button b[];
   private String names[] =
      { "Hide North", "Hide South", "Hide East",
        "Hide West", "Hide Center" };

   public void init()
   {
      // set layout to border layout
      setLayout( new BorderLayout( 5, 5 ) );

      // instantiate button objects
      b = new Button[ names.length ];

      for ( int i = 0; i < names.length; i++ ) {
         b[ i ] = new Button( names[ i ] );
         b[ i ].addActionListener( this );
      }

      // order not important
      add( b[ 0 ], BorderLayout.NORTH );  // North position
      add( b[ 1 ], BorderLayout.SOUTH );  // South position
      add( b[ 2 ], BorderLayout.EAST );      // East position
      add( b[ 3 ], BorderLayout.WEST );     // West position
      add( b[ 4 ], BorderLayout.CENTER );  // Center position
   }

   public void actionPerformed( ActionEvent e )
   {
      for ( int i = 0; i < b.length; i++ )
         if ( e.getSource() == b[ i ] )
            b[ i ].setVisible( false );
         else
            b[ i ].setVisible( true );

      validate();
   }
}
```

```
// Fig. 10.31: Grid.java
// Demonstrating GridLayout.
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class Grid extends Applet implements ActionListener {
   private Button b[];
   private String names[] =
      { "one", "two", "three", "four", "five", "six" };
   private boolean toggle = true;

   public void init()
   {
      // set layout to grid layout
      setLayout( new GridLayout( 2, 3, 5, 5 ) );

      // create and add buttons
      b = new Button[ names.length ];

      for (int i = 0; i < names.length; i++ ) {
         b[ i ] = new Button( names[ i ] );
         b[ i ].addActionListener( this );
         add( b[ i ] );
      }
   }

   public void actionPerformed( ActionEvent e )
   {
      if ( toggle )
         setLayout( new GridLayout( 3, 2 ) );
      else
         setLayout( new GridLayout( 2, 3, 5, 5 ) );

      toggle = !toggle;
      validate();
   }
}
```

```
// Fig. 10.32: PanelDemo.java
// Using a panel to help layout components.
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class PanelDemo extends Applet {
   private Panel buttonPanel;
   private Button buttons[];

   public void init()
   {
      buttonPanel = new Panel();
      buttons = new Button[ 5 ];

      buttonPanel.setLayout(
         new GridLayout( 1, buttons.length ) );

      for ( int i = 0; i < buttons.length; i++ ) {
         buttons[ i ] = new Button( "Button " + (i + 1) );
         buttonPanel.add( buttons[ i ] );
      }

      setLayout( new BorderLayout() );
      add( buttonPanel, BorderLayout.SOUTH );
   }
}
```

```
// Fig. 11.16: CardDeck.java
// Demonstrating CardLayout.
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class CardDeck extends Applet implements ActionListener {
   private Canvas c;
   private CardLayout cardManager;
   private Panel deck, card1, card2, card3, buttons;
   private Button b1, b2, b3, prevButton, nextButton,
                  lastButton, firstButton;

   public void init()
   {
      setLayout( new BorderLayout() );

      // create the panel with CardLayout
      deck = new Panel();
      cardManager = new CardLayout();
      deck.setLayout( cardManager );

      // set up card1 and add it to Panel deck
      b1 = new Button( "card one" );
      card1 = new Panel();
      card1.add( b1 );       // add a button
      deck.add( card1, b1.getLabel() );  // add card to deck

      // set up card2 and add it to Panel deck
      b2 = new Button( "card two" );
      c = new Canvas();
      c.setBackground( Color.green );
      c.setSize( 80, 80 );
      card2 = new Panel();
      card2.add( b2 );       // add a button
      card2.add( c );        // add a canvas
      deck.add( card2, b2.getLabel() );  // add card to deck

      // set up card3 and add it to Panel deck
      b3 = new Button( "card three" );
      card3 = new Panel();
      card3.setLayout( new BorderLayout() );   // set layout
      card3.add( new Button( "North" ), BorderLayout.NORTH );
      card3.add( new Button( "West" ), BorderLayout.WEST );
      card3.add( new Button( "East" ), BorderLayout.EAST );
      card3.add( new Button( "Pole" ), BorderLayout.SOUTH );
      card3.add( b3, BorderLayout.CENTER );
      deck.add( card3, b3.getLabel() );  // add card to deck
```

```
      // create and layout Panel buttons that will control deck
      buttons = new Panel();
      buttons.setLayout ( new GridLayout( 2, 2 ) );
      buttons.add( firstButton = new Button( "First card" ) );
      buttons.add( nextButton = new Button( "Next card" ) );
      buttons.add( prevButton = new Button( "Previous card" ) );
      buttons.add( lastButton = new Button( "Last card" ) );
      firstButton.addActionListener( this );
      nextButton.addActionListener( this );
      prevButton.addActionListener( this );
      lastButton.addActionListener( this );

      // add Panel deck and Panel buttons to the applet
      add( buttons, BorderLayout.WEST );
      add( deck, BorderLayout.EAST );
   }

   public void actionPerformed( ActionEvent e )
   {
      if ( e.getSource() == firstButton )
         cardManager.first( deck );    // show first card
      else if ( e.getSource() == prevButton )
         cardManager.previous( deck ); // show previous card
      else if ( e.getSource() == nextButton )
         cardManager.next( deck );     // show next card
      else if ( e.getSource() == lastButton )
         cardManager.last( deck );     // show last card

   }
}
```

```
// Fig. 11.20: GridBagDemo.java
// Demonstrating GridBagLayout.
import java.applet.Applet;
import java.awt.*;

public class GridBagDemo extends Applet {
  private Canvas c;
  private Choice cb;
  private TextArea ta;
  private TextField tf;
  private Button b1, b2, b3;
  private GridBagLayout gbLayout;
  private GridBagConstraints gbConstraints;

  public void init()
  {
    gbLayout = new GridBagLayout();
    setLayout( gbLayout );   // applet

    // instantiate gridbag constraints
    gbConstraints = new GridBagConstraints();

    ta = new TextArea( "Text Area", 5, 10 );
    cb = new Choice( );
    cb.add( "Linden" );
    cb.add( "Birch" );
    cb.add( "Ceder" );
    tf = new TextField( "TextField" );
    b1 = new Button( "Button 1" );
    b2 = new Button( "Button 2" );
    b3 = new Button( "Button 3" );
    c = new Canvas();
    c.setBackground( Color.blue );
    c.setSize( 10, 5 );

    // text area
    // weightx and weighty are both 0: the default
    // anchor for all components is CENTER: the default
    gbConstraints.fill = GridBagConstraints.BOTH;
    addComponent( ta, 0, 0, 1, 3 );

    // button b1
    // weightx and weighty are both 0: the default
    gbConstraints.fill = GridBagConstraints.HORIZONTAL;
    addComponent( b1, 0, 1, 2, 1 );

    // choice button
    // weightx and weighty are both 0: the default
```

```
    // fill is HORIZONTAL
    addComponent( cb, 2, 1, 2, 1 );

    // button b2
    gbConstraints.weightx = 1000;  // can grow wider
    gbConstraints.weighty = 1;     // can grow taller
    gbConstraints.fill = GridBagConstraints.BOTH;
    addComponent( b2, 1, 1, 1, 1 );

    // button b3
    // fill is BOTH
    gbConstraints.weightx = 0;
    gbConstraints.weighty = 0;
    addComponent( b3, 1, 2, 1, 1 );

    // textfield
    // weightx and weighty are both 0: fill is BOTH
    addComponent( tf, 3, 0, 2, 1 );

    // canvas
    // weightx and weighty are both 0: fill is BOTH
    addComponent( c, 3, 2, 1, 1 );
  }

  // addComponent is programmer defined
  private void addComponent( Component c, int row, int column,
                             int width, int height )
  {
    // set gridx and gridy
    gbConstraints.gridx = column;
    gbConstraints.gridy = row;

    // set gridwidth and gridheight
    gbConstraints.gridwidth = width;
    gbConstraints.gridheight = height;

    // set constraints
    gbLayout.setConstraints( c, gbConstraints );
    add( c );      // add component to applet
  }
}
```

```
// Fig. 11.3: TextAreaDemo.java
// Copying selected text from one text area to another.
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class TextAreaDemo extends Applet
      implements ActionListener, TextListener {
   private TextArea t1, t2;
   private Button copy;

   public void init()
   {
      String s = "This is a demo string to illustrate " +
                 "copying text from one TextArea to " +
                 "another TextArea using an external event";

      t1 = new TextArea( s, 5, 20, TextArea.SCROLLBARS_NONE );
      t1.addTextListener( this );
      add( t1 );

      copy = new Button( "Copy >>>" );
      copy.addActionListener( this );
      add( copy );

      t2 = new TextArea( 5, 20 );
      t2.setEditable( false );
      add( t2 );
   }

   public void actionPerformed( ActionEvent e )
   {
      t2.setText( t1.getSelectedText() );
   }

   public void textValueChanged( TextEvent e )
   {
      TextComponent source = ( TextComponent ) e.getSource();
      t2.setText( source.getText() );
   }
}
```

```java
// Fig. 11.4: ButtonHandler.java
// Handle event for a button in the MyCanvas applet.
import java.awt.event.*;

public class ButtonHandler implements ActionListener {
   private CustomCanvas canvas;
   private int shape;

   public ButtonHandler( CustomCanvas c, int s )
   {
      canvas = c;
      shape = s;
   }

   public void actionPerformed( ActionEvent e )
   {
      canvas.draw( shape );
   }
}
```

```
// Fig. 11.4: CustomCanvas.java
// A customized Canvas class.
import java.awt.*;

public class CustomCanvas extends Canvas {
   public final static int CIRCLE = 1, SQUARE = 2;
   private int shape;

   public void paint( Graphics g )
   {
      if ( shape == CIRCLE )
         g.fillOval( 50, 10, 60, 60 );
      else if ( shape == SQUARE )
         g.fillRect( 50, 10, 60, 60 );
   }

   public void draw( int s )
   {
      shape = s;
      repaint();
   }
}
```

```
// Fig. 11.4: MyCanvas.java
// Using a customized Canvas object.
import java.applet.Applet;
import java.awt.*;

public class MyCanvas extends Applet {
   private Panel p;
   private CustomCanvas c;
   private Button circle, square;

   public void init()
   {
      c = new CustomCanvas();   // instantiate canvas
      c.setBackground( Color.green );

      square = new Button( "Square" );
      square.addActionListener(
         new ButtonHandler( c, CustomCanvas.SQUARE ) );
      circle = new Button( "Circle" );
      circle.addActionListener(
         new ButtonHandler( c, CustomCanvas.CIRCLE ) );

      p = new Panel();
      p.setLayout( new GridLayout( 2, 1 ) );
      p.add( circle );
      p.add( square );

      setLayout( new BorderLayout() );
      add( p, BorderLayout.WEST );
      add( c, BorderLayout.CENTER );    // add canvas
   }
}
```

```
// Fig. 11.5: MyCanvas2.java
// Creating a self-contained canvas that processes
// its own mouse events.
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import com.deitel.jhtp2.ch11.SelfContainedCanvas;

public class MyCanvas2 extends Applet
      implements MouseMotionListener {
   private SelfContainedCanvas c;

   public void init()
   {
      c = new SelfContainedCanvas();
      c.setSize( 185, 70 );  // resize canvas
      c.setBackground( Color.yellow );
      add( c );    // add canvas to applet

      addMouseMotionListener( this );
   }

   public void mouseDragged( MouseEvent e )
   {
      showStatus( "Applet x: " + e.getX() +
                  "; Applet y: " + e.getY() );
   }

   public void mouseMoved( MouseEvent e )
   {
      showStatus( "Applet x: " + e.getX() +
                  "; Applet y: " + e.getY() );
   }
}
```

```java
// Fig. 11.5: SelfContainedCanvas.java
// A self-contained Canvas class that
// handles its own mouse events.
package com.deitel.jhtp2.ch11;

import java.awt.*;
import java.awt.event.*;

public class SelfContainedCanvas extends Canvas
      implements MouseListener, MouseMotionListener {
   private int x1, y1, x2, y2;

   public SelfContainedCanvas()
   {
      addMouseListener( this );
      addMouseMotionListener( this );
   }

   public void paint( Graphics g )
   {
      int x, y, width, height;

      // determine upper-left corner of bounding rectangle
      x = Math.min( x1, x2 );
      y = Math.min( y1, y2 );

      // determine width and height of bounding rectangle
      width = Math.abs( x1 - x2 );
      height = Math.abs( y1 - y2 );

      g.drawOval( x, y, width, height );
   }

   public void mousePressed( MouseEvent e )
   {
      x1 = e.getX();
      y1 = e.getY();
   }

   public void mouseReleased( MouseEvent e )
   {
      x2 = e.getX();
      y2 = e.getY();
      repaint();
   }

   public void mouseDragged( MouseEvent e )
   {
```

```
        x2 = e.getX();
        y2 = e.getY();
        repaint();
    }

    // These methods not used but must be defined because
    // we implement MouseListener and MouseMotionListener.
    public void mouseClicked( MouseEvent e ) { }
    public void mouseEntered( MouseEvent e ) { }
    public void mouseExited( MouseEvent e ) { }
    public void mouseMoved( MouseEvent e ) { }
}
```

```java
// Fig. 11.08: OvalCanvas.java
// A customized Canvas class.
import java.awt.*;

public class OvalCanvas extends Canvas {
   public static int WIDTH = 1, HEIGHT = 2;
   private int ovalWidth, ovalHeight;

   public OvalCanvas()
   {
      setOvalWidth( 100 );
      setOvalHeight( 100 );
   }

   public void paint( Graphics g )
   {
      g.drawOval( 0, 0, ovalWidth, ovalHeight );
   }

   public void setOvalWidth( int w )
   {
      ovalWidth = w;
      repaint();
   }

   public void setOvalHeight( int h )
   {
      ovalHeight = h;
      repaint();
   }
}
```

```
// Fig. 11.08: ScrollbarDemo.java
// Using Scrollbars to size an oval.
import java.applet.Applet;
import java.awt.*;

public class ScrollbarDemo extends Applet {
   private Scrollbar scrollHeight, scrollWidth;
   private OvalCanvas c;

   public void init()
   {
      setSize( 215, 235 );

      c = new OvalCanvas();
      c.setSize( 200, 200 );
      c.setBackground( Color.yellow );

      scrollHeight = new Scrollbar( Scrollbar.VERTICAL, 100,
                                    1, 0, 200 );
      scrollHeight.addAdjustmentListener(
         new ScrollbarHandler( c, OvalCanvas.HEIGHT ) );

      scrollWidth = new Scrollbar( Scrollbar.HORIZONTAL, 100,
                                   1, 0, 200 );
      scrollWidth.addAdjustmentListener(
         new ScrollbarHandler( c, OvalCanvas.WIDTH ) );

      setLayout( null );  // No layout

      add( scrollHeight );
      scrollHeight.setBounds( 200, 0, 15, 200 );

      add( scrollWidth );
      scrollWidth.setBounds( 0, 200, 200, 15 );

      add( c );
      c.setLocation( 0, 0 );
   }
}
```

```java
// Fig. 11.08: ScrollbarHandler.java
// Class to handle Scrollbar events.
import java.awt.event.*;

public class ScrollbarHandler implements AdjustmentListener {
   private OvalCanvas canvas;
   private int dimension;

   public ScrollbarHandler( OvalCanvas c, int d )
   {
      canvas = c;
      dimension = d;
   }

   public void adjustmentValueChanged( AdjustmentEvent e )
   {
      if ( dimension == OvalCanvas.WIDTH )
         canvas.setOvalWidth( e.getValue() );
      else if ( dimension == OvalCanvas.HEIGHT )
         canvas.setOvalHeight( e.getValue() );
   }
}
```

```java
// Fig. 11.09: CloseWindow.java
// Class to close a window.
package com.deitel.jhtp2.ch11;
import java.awt.event.*;

public class CloseWindow extends WindowAdapter {
   public void windowClosing( WindowEvent e )
   {
      e.getWindow().setVisible( false );
   }
}
```

```java
// Fig. 11.09: ColorFrame.java
// ColorFrame class allows user to select a color and
// changes the frame’s background to that color.
package com.deitel.jhtp2.ch11;
import java.awt.*;
import java.awt.event.*;

public class ColorFrame extends Frame implements ItemListener {
   private Color colors[] = { Color.white, Color.red,
      Color.green, Color.blue, Color.magenta };
   private String colorNames[] =
      { "White", "Red", "Green", "Blue", "Magenta" };
   private Choice colorChoice;

   public ColorFrame( String s )
   {
      // call base class constructor
      super( s );
      setSize( 275, 150 );
      addWindowListener( new CloseWindow() );

      colorChoice = new Choice();

      for ( int i = 0; i < colorNames.length; i++ )
         colorChoice.add( colorNames[ i ] );

      add( colorChoice, BorderLayout.NORTH );
      colorChoice.addItemListener( this );
   }

   public void itemStateChanged( ItemEvent e )
   {
      setBackground( colors[ colorChoice.getSelectedIndex() ] );
      repaint();
   }
}
```

```java
// Fig. 11.09: MyFrame.java
// Creating a subclass of Frame.
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import com.deitel.jhtp2.ch11.ColorFrame;

public class MyFrame extends Applet implements ActionListener {
   private ColorFrame f;
   private Button showFrame;

   public void init()
   {
      showFrame = new Button( "Show frame" );
      add( showFrame );
      showFrame.addActionListener( this );

      f = new ColorFrame( "Select a color" );
   }

   public void actionPerformed( ActionEvent e )
   {
      f.setVisible( true );  // show the frame
   }
}
```

```
// Fig. 11.10: CloseWindowAndExit.java
// Frame in which an applet can execute as an application.
package com.deitel.jhtp2.ch11;
import java.awt.event.*;

public class CloseWindowAndExit extends WindowAdapter {
   public void windowClosing( WindowEvent e )
   {
      System.exit( 0 );
   }
}
```

```
// Fig. 11.10: MyFrame.java
// Creating a subclass of Frame.
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import com.deitel.jhtp2.ch11.ColorFrame;
import com.deitel.jhtp2.ch11.CloseWindowAndExit;

public class MyFrame extends Applet implements ActionListener {
   private ColorFrame f;
   private Button showFrame;

   public void init()
   {
      showFrame = new Button( "Show frame" );
      add( showFrame );
      showFrame.addActionListener( this );

      f = new ColorFrame( "Select a color" );
   }

   public void actionPerformed( ActionEvent e )
   {
      f.setVisible( true );  // show the frame
   }

   // Allow this applet to run as an application also
   public static void main( String args[] )
   {
      // create application Frame and resize
      Frame app = new Frame( "Application" );
      app.setSize( 300, 100 );

      // register window handler to terminate application
      app.addWindowListener( new CloseWindowAndExit() );

      // create applet instance
      MyFrame m = new MyFrame();

      // initialize and start the applet
      m.init();
      m.start();

      // add applet to center of Frame
      app.add( m, BorderLayout.CENTER );

      // display the frame (everything attached is painted)
      app.setVisible( true );
```

```
    }
}
```

```
// Fig. 11.11: ScratchPad.java
// Incorporating menus into a subclass of Frame.
package com.deitel.jhtp2.ch11;
import java.awt.*;
import java.awt.event.*;

public class ScratchPad extends Frame
      implements ActionListener, ItemListener {
   private TextArea t;
   private String fontNames[] =
      { "TimesRoman", "Courier", "Helvetica" };
   private String colorNames[] =
      { "Black", "Red", "Green", "Blue" };
   private Color colorValues[] =
      { Color.black, Color.red, Color.green, Color.blue };

   private MenuBar bar;
   private Menu formatMenu, fontMenu, colorMenu;
   private MenuItem fonts[], colors[];
   private CheckboxMenuItem readOnly;

   public ScratchPad()
   {
      super( "ScratchPad Application" );
      setSize( 300, 200 );

      t = new TextArea( "", 2, 20,
                        TextArea.SCROLLBARS_VERTICAL_ONLY);
      add( t, BorderLayout.CENTER );

      t.setFont( new Font( "TimesRoman", Font.PLAIN, 12 ) );
      t.setForeground( colorValues[ 0 ] );

      // create menubar
      bar = new MenuBar();

      // create the format menu
      formatMenu = new Menu( "Format" );

      // create font menu
      fontMenu = new Menu( "Font" );

      fonts = new MenuItem[ fontNames.length ];

      for ( int i = 0; i < fonts.length; i++ ) {
         fonts[ i ] = new MenuItem( fontNames[ i ] );
         fontMenu.add( fonts[ i ] );
         fonts[ i ].addActionListener( this );
```

```
      }

      formatMenu.add( fontMenu );
      formatMenu.addSeparator();

      // create color menu
      colorMenu = new Menu( "Color" );

      colors = new MenuItem[ colorNames.length ];

      for ( int i = 0; i < colors.length; i++ ) {
         colors[ i ] = new MenuItem( colorNames[ i ] );
         colorMenu.add( colors[ i ] );
         colors[ i ].addActionListener( this );
      }

      formatMenu.add( colorMenu );
      formatMenu.addSeparator();

      // create "read-only" menu item
      readOnly = new CheckboxMenuItem( "Read-Only" );
      readOnly.addItemListener( this );

      formatMenu.add( readOnly );

      // add menu to menu bar
      bar.add( formatMenu );

      // set the menubar for the frame
      setMenuBar( bar );

      setVisible( true );
   }

   // Handle font and color menu selections
   public void actionPerformed( ActionEvent e )
   {
      for ( int i = 0; i < fonts.length; i++ )
         if ( e.getSource() == fonts[ i ] ) {
            t.setFont( new Font( fonts[ i ].getLabel(),
               Font.PLAIN, 12 ) );
            break;
         }

      for ( int i = 0; i < colors.length; i++ )
         if ( e.getSource() == colors[ i ] ) {
            t.setForeground( colorValues[ i ] );
            break;
```

```
         }
   }

   // Handle "read-only" menu selections
   public void itemStateChanged( ItemEvent e )
   {
      t.setEditable( ! t.isEditable() );
   }
}
```

```
// Fig. 11.11: ScratchPadTest.java
// Class to test ScratchPad.
import com.deitel.jhtp2.ch11.ScratchPad;
import com.deitel.jhtp2.ch11.CloseWindowAndExit;

public class ScratchPadTest {
   public static void main( String args[] )
   {
      ScratchPad e = new ScratchPad();
      e.addWindowListener( new CloseWindowAndExit() );
   }
}
```

```java
// Fig. 11.12: DialogBoxDemo.java
// Demonstrating modal and modeless Dialog boxes.
import java.awt.*;
import java.awt.event.*;
import com.deitel.jhtp2.ch11.CloseWindowAndExit;
import com.deitel.jhtp2.ch11.SelfContainedCanvas2;
import com.deitel.jhtp2.ch11.SelectColorDialog;
import com.deitel.jhtp2.ch11.MessageBox;

public class DialogBoxDemo extends Frame
             implements ActionListener, ItemListener {
   private MessageBox information;
   private MenuItem about, colors, exit;
   private MenuBar bar;
   private Menu helpMenu, fileMenu;
   private SelfContainedCanvas2 canvas;
   private SelectColorDialog selectColor;

   public DialogBoxDemo()
   {
      super( "Dialog boxes" );
      setSize( 275, 150 );

      // create Help menu
      helpMenu = new Menu( "Help" );
      about = new MenuItem( "About" );
      about.addActionListener( this );
      helpMenu.add( about );

      // create File menu
      fileMenu = new Menu( "File" );
      colors = new MenuItem( "Select color ..." );
      colors.addActionListener( this );
      fileMenu.add( colors );
      fileMenu.addSeparator();
      exit = new MenuItem( "Exit" );
      exit.addActionListener( this );
      fileMenu.add( exit );

      // setup MenuBar
      bar = new MenuBar();
      bar.add( fileMenu );
      bar.add( helpMenu );
      bar.setHelpMenu( helpMenu );
      setMenuBar( bar );

      // create About dialog
      information = new MessageBox(
```

```
         this, "Message", "Dialog box example", 200 );

      // create SelectColorDialog
      selectColor = new SelectColorDialog(
         this, "Click once on desired color" );

      // create SelfContainedCanvas2
      canvas = new SelfContainedCanvas2();
      add( canvas, BorderLayout.CENTER );

      setVisible( true );  // show window
   }

   public void actionPerformed( ActionEvent e )
   {
      if ( e.getSource() == about )
         information.setVisible( true );
      else if ( e.getSource() == colors )
         selectColor.setVisible( true );
      else if ( e.getSource() == exit )
         System.exit( 0 );
   }

   public void itemStateChanged( ItemEvent e )
   {
      canvas.setDrawingColor( selectColor.getColor() );
   }

   public static void main( String args[] )
   {
      DialogBoxDemo d = new DialogBoxDemo();
      d.addWindowListener( new CloseWindowAndExit() );
   }
}
```

```
// Fig. 11.12: MessageBox.java
// Class that implements a modal message dialog box.
package com.deitel.jhtp2.ch11;
import java.awt.*;
import java.awt.event.*;
import com.deitel.jhtp2.ch11.CloseWindow;

public class MessageBox extends Dialog
             implements ActionListener {
   private Label message;
   private Button close;

   public MessageBox( Frame parent, String title,
      String messageString, int width )
   {
      super( parent, title, true );
      setSize( width, 100 );
      setResizable( false );

      message = new Label( messageString, Label.CENTER );
      add( message, BorderLayout.CENTER );

      close = new Button( "Close" );
      close.addActionListener( this );
      add( close, BorderLayout.SOUTH );

      addWindowListener( new CloseWindow() );
   }

   public void actionPerformed( ActionEvent e )
   {
      setVisible( false );
   }
}
```

```
// Fig. 11.12: SelectColorDialog.java
// Modeless dialog that allows user to choose drawing color.
package com.deitel.jhtp2.ch11;
import java.awt.*;
import java.awt.event.*;
import com.deitel.jhtp2.ch11.CloseWindow;

public class SelectColorDialog extends Dialog
             implements ItemListener {
   private List colorList;
   private String colorNames[] =
      { "Black", "Blue", "Cyan", "Dark Gray", "Gray", "Green",
        "Light Gray", "Magenta", "Orange", "Pink", "Red",
        "White", "Yellow" };
   private Color colors[] =
      { Color.black, Color.blue, Color.cyan, Color.darkGray,
        Color.gray, Color.green, Color.lightGray, Color.magenta,
        Color.orange, Color.pink, Color.red, Color.white,
        Color.yellow };
   private Color currentColor;

   public SelectColorDialog( Frame parent, String title )
   {
      super( parent, title, false );
      setSize( 275, 150 );
      addWindowListener( new CloseWindow() );

      // create a list with 5 items visible
      // do not allow multiple selections
      colorList = new List( 5, false );
      colorList.addItemListener( this );
      colorList.addItemListener( ( ItemListener ) parent );

      // add items to the list
      for ( int i = 0; i < colorNames.length; i++ )
         colorList.add( colorNames[ i ] );

      // add list to applet
      add( colorList, BorderLayout.CENTER );

      currentColor = colors[ 0 ];  // black
   }

   public void itemStateChanged( ItemEvent e )
   {
      currentColor = colors[ colorList.getSelectedIndex() ];
   }
```

```
  public Color getColor() { return currentColor; }
}
```

```java
// Fig. 11.12: SelfContainedCanvas2.java
// Adds ability to set drawing color for a SelfContainedCanvas.
package com.deitel.jhtp2.ch11;
import java.awt.*;
import com.deitel.jhtp2.ch11.SelfContainedCanvas;

public class SelfContainedCanvas2 extends SelfContainedCanvas {
   private Color color = Color.black;

   public void paint( Graphics g )
   {
      g.setColor( color );
      super.paint( g );
   }

   public void setDrawingColor( Color c ) { color = c; }
}
```

```java
// Fig. 11.13: ScrollPaneTest.java
// Testing class ScrollPane
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
import com.deitel.jhtp2.ch11.SelfContainedCanvas;

public class ScrollPaneTest extends Applet {
   private SelfContainedCanvas drawCanvas;
   private ScrollPane drawCanvasScroller;

   public void init()
   {
      drawCanvas = new SelfContainedCanvas();
      drawCanvas.setSize( 1000, 1000 );
      drawCanvasScroller = new ScrollPane();
      drawCanvasScroller.add( drawCanvas );
      drawCanvasScroller.setSize( 275, 275 );
      add( drawCanvasScroller );

      Adjustable horizontal =
         drawCanvasScroller.getHAdjustable();
      horizontal.setUnitIncrement( 4 );
      Adjustable vertical =
         drawCanvasScroller.getVAdjustable();
      vertical.setUnitIncrement( 4 );
   }
}
```

```java
// Fig. 11.14: PopupTest.java
// Class to test popup menus.
import java.awt.*;
import com.deitel.jhtp2.ch11.SelfContainedCanvas3;
import com.deitel.jhtp2.ch11.CloseWindowAndExit;

public class PopupTest extends Frame {
   private SelfContainedCanvas3 canvas;

   public PopupTest()
   {
      super( "Popup Test Example" );

      canvas = new SelfContainedCanvas3();
      add( canvas, BorderLayout.CENTER );
      setSize( 300, 200 );
      setVisible( true );
   }

   public static void main( String args[] )
   {
      PopupTest p = new PopupTest();
      p.addWindowListener( new CloseWindowAndExit() );
   }
}
```

```
// Fig. 11.14: SelfContainedCanvas3.java
// Class that extends SelfContainedCanvas2 and adds
// a popup menu to select drawing color.
package com.deitel.jhtp2.ch11;
import java.awt.*;
import java.awt.event.*;

public class SelfContainedCanvas3 extends SelfContainedCanvas2
      implements ActionListener {
   private String colorNames[] =
      { "Black", "Blue", "Cyan", "Dark Gray", "Gray", "Green",
        "Light Gray", "Magenta", "Orange", "Pink", "Red",
        "White", "Yellow" };
   private Color colorValues[] =
      { Color.black, Color.blue, Color.cyan, Color.darkGray,
        Color.gray, Color.green, Color.lightGray, Color.magenta,
        Color.orange, Color.pink, Color.red, Color.white,
        Color.yellow };
   private MenuItem colors[];
   private PopupMenu popup;

   public SelfContainedCanvas3()
   {
      popup = new PopupMenu( "Colors" );
      colors = new MenuItem[ colorNames.length ];

      for ( int i = 0; i < colorNames.length; i++ ) {
         colors[ i ] = new MenuItem( colorNames[ i ] );
         popup.add ( colors[ i ] );
         colors[ i ].addActionListener( this );
      }

      add( popup );  // add popup to this canvas
      enableEvents( AWTEvent.MOUSE_EVENT_MASK );
   }

   public void actionPerformed( ActionEvent e )
   {
      for ( int i = 0; i < colors.length; i++ )
         if ( e.getSource() == colors[ i ] ) {
            setDrawingColor( colorValues[ i ] );
            break;
         }
   }

   public void processMouseEvent( MouseEvent e )
   {
      if ( e.isPopupTrigger() )
```

```
            popup.show( this, e.getX(), e.getY() );

        super.processMouseEvent( e );
    }
}
```