```
// A demonstration of some Applet aspects

import java.awt.Graphics;
import java.applet.Applet;

public class AppletDemo extends Applet {
  private int start_count;
  private int paint_count;

  public void init() {
    start_count = 0;
    paint_count = 0;
  }

  public void start() {
    start_count++;
  }

  public void paint( Graphics g ) {
    paint_count++;
    g.drawString("Applet started " + start_count + " times", 25,
25);
    showStatus("Applet painted " + paint_count + " times");
  }
}
```

```
// Fig. 9.4: DrawSCB.java
// Demonstrating drawString, drawChars and drawBytes
import java.applet.Applet;
import java.awt.Graphics;

public class DrawSCB extends Applet {
   private String s = "Using drawString!";
   private char c[] = { 'c', 'h', 'a', 'r', 's', ' ', '8' };
   private byte b[] = { 'b', 'y', 't', 'e', 1, 2, 3 };

   public void paint( Graphics g )
   {
      // draw a string at location (100, 25) on the applet
      g.drawString( s, 100, 25 );

      // draw a series of characters at location (100, 50)
      g.drawChars( c, 2, 3,  100, 50 );

      // draw a series of bytes at location (100, 75)
      g.drawBytes( b, 0, 5, 100, 75 );
   }
}
```

```java
// Fig. 9.7: ShowColors.java
// Demonstrating setting and getting a Color.
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;

public class ShowColors extends Applet {
  private int red, green, blue; // for the first color
  private float red_f, green_f, blue_f;
  private Color c;

  public void init()
  {
    // set some values
    red = 100;
    blue = 255;
    green = 125;

    red_f = 0.1f;
    green_f = 0.61f;
    blue_f = 0.23f;

    c = Color.yellow;
  }

  public void paint ( Graphics g )
  {
    g.setColor( new Color( red, green, blue ) );
    g.drawString( "Color set with ints " +
                  g.getColor().toString() +
                  ": ABCDEFGHIJKLMNOPQRSTUVWXYZ", 25, 25 );

    g.setColor( new Color( red_f, green_f, blue_f ) );
    g.drawString( "Color set with floats " +
                  g.getColor().toString() +
                  ": ABCDEFGHIJKLMNOPQRSTUVWXYZ", 25, 50 );

    g.setColor( c );
    g.drawString( "Color set to constant " +
                  c.getRed() + " " + c.getGreen() + " " +
                  c.getBlue() +
                  ": ABCDEFGHIJKLMNOPQRSTUVWXYZ", 25, 75 );
  }
}
```

```
// Fig. 9.11: DemoFont.java
// Demonstrating the Font constants, the Font constructor
// and the setFont method
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Font;

public class DemoFont extends Applet {
   private Font font1, font2, font3;

   public void init()
   {
      // create a font object: 12-point bold times roman
      font1 = new Font( "Serif", Font.BOLD, 12 );

      // create a font object: 24-point italic courier
      font2 = new Font( "Monospaced", Font.ITALIC, 24 );

      // create a font object: 14-point plain helvetica
      font3 = new Font( "SansSerif", Font.PLAIN, 14 );
   }

   public void paint( Graphics g )
   {
      // set the current font to font1
      g.setFont( font1 );

      // draw a string in font font1
      g.drawString( "Serif 12 point bold.", 20, 20 );

      // change the current font to font2
      g.setFont( font2 );

      // draw a string in font font2
      g.drawString( "Monospaced 24 point italic.", 20, 40 );

      // change the current font to font3
      g.setFont( font3 );

      // draw a string in font font3
      g.drawString( "SansSerif 14 point plain.", 20, 60 );
  }
}
```

```
// Fig. 9.13: DemoFont2.java
// Demonstrating the Font methods for
// retrieving font information
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Font;

public class DemoFont2 extends Applet {
   private Font f;

   public void init()
   {
      // create a font object: 24-point bold italic Monospaced
      f = new Font( "Monospaced", Font.ITALIC + Font.BOLD, 24 );
   }

   public void paint( Graphics g )
   {
      int style, size;
      String s, name;

      g.setFont( f );          // set the current font to f
      style = f.getStyle();    // determine current font style

      if ( style == Font.PLAIN )
         s = "Plain ";
      else if ( style == Font.BOLD )
         s = "Bold ";
      else if ( style == Font.ITALIC )
         s = "Italic ";
      else    // bold + italic
         s = "Bold italic ";

      size = f.getSize();     // determine current font size
      s += size + " point ";
      name = f.getName();     // determine current font name
      s += name;
      g.drawString( s, 20, 40 );

      // display font family
      g.drawString( "Font family is " + f.getFamily(), 20, 60 );
  }
}
```

```java
// Fig. 9.15: DemoFont3.java
// Demonstrating isPlain, isBold and isItalic
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Font;

public class DemoFont3 extends Applet {
   private Font f;

   public void init()
   {
      // create a font object: 24-point bold italic Monospaced
      f = new Font( "Monospaced", Font.ITALIC + Font.BOLD, 24 );
   }

   public void paint( Graphics g )
   {
      String s;

      g.setFont( f );          // set the current font to f

      if ( f.isPlain() == true )
         s = "Font is plain.";
      else if ( f.isBold() == true && f.isItalic() == false )
         s = "Font is bold.";
      else if ( f.isItalic() == true && f.isBold() == false )
         s = "Font is italic.";
      else    // bold + italic
         s = "Font is bold italic.";

      g.drawString( s, 20, 40 );
  }
}
```

```
// Fig. 9.20: Line.java
// Demonstrating drawLine
import java.applet.Applet;
import java.awt.Graphics;

public class Line extends Applet {

   public void paint( Graphics g )
   {
      // draw a line from (10, 10) to (230, 95)
      g.drawLine( 10, 10, 230, 95 );
   }
}
```

```
// Fig. 9.22: RectangleTest.java
// Demonstrating drawRect and fillRect
import java.applet.Applet;
import java.awt.Graphics;

public class RectangleTest extends Applet {

   public void paint( Graphics g )
   {
      // draw a rectangle at location ( 10, 15 )
      g.drawRect( 10, 15, 100, 100 );

      // draw a filled rectangle at location ( 150, 15 )
      g.fillRect( 150, 15, 100, 100 );
   }
}
```

```java
// Fig. 9.25: Rectangle2.java
// Drawing rounded rectangles
import java.applet.Applet;
import java.awt.Graphics;

public class Rectangle2 extends Applet {

   public void paint( Graphics g )
   {
      // draw a rounded rectangle at (10, 35)
      g.drawRoundRect( 10, 35, 50, 50, 10, 20 );

      // draw a filled rounded rectangle at (80, 15)
      g.fillRoundRect( 80, 15, 60, 80, 50, 10 );

      // draw a rounded rectangle at (150, 55)
      g.drawRoundRect( 150, 55, 80, 20, 70, 70 );

      // draw a filled square at (240, 15)
      g.fillRoundRect( 240, 15, 80, 80, 0, 0 );

      // draw a circle at (330, 15)
      g.drawRoundRect( 330, 15, 80, 80, 80, 80 );
   }
}
```

```
// Fig. 9.27: Draw3D.java
// Drawing 3-D rectangles
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;

public class Draw3D extends Applet {

   public void paint( Graphics g )
   {
      g.setColor( Color.yellow );

      // draw a raised 3D rectangle at location (10, 10)
      g.draw3DRect( 10, 10, 100, 100, true );

      // draw a sunk 3D at location (130, 10)
      g.draw3DRect( 130, 10, 100, 100, false );

      // draw a filled raised 3D rectangle at (10, 120)
      g.fill3DRect( 10, 120, 100, 100, true );

      // draw a filled sunk 3D rectangle at (130, 120)
      g.fill3DRect( 130, 120, 100, 100, false );
   }
}
```

```java
// Fig. 9.30: DrawOval.java
// Drawing ovals
import java.applet.Applet;
import java.awt.Graphics;

public class DrawOval extends Applet {

   public void paint( Graphics g )
   {
      // draw an oval at location (10, 15)
      g.drawOval( 10, 15, 100, 70 );

      // draw an oval at location (160, 15)
      g.fillOval( 160, 15, 70, 130 );
   }
}
```

```
// Fig. 9.34: DrawArcs.java
// Drawing arcs
import java.applet.Applet;
import java.awt.Graphics;

public class DrawArcs extends Applet {

   public void paint( Graphics g )
   {
      // draw an arc
      g.drawArc( 15, 15, 80, 80, 0, 180 );

      // draw an arc
      g.drawArc( 100, 100, 80, 80, 0, 110 );

      // draw a solid arc
      g.fillArc( 100, 15, 70, 80, 0, 270 );

      // draw a solid arc
      g.fillArc( 15, 70, 70, 80, 0, -110 );

      // draw a solid arc
      g.fillArc( 190, 15, 80, 140, 0, -360 );
   }
}
```

```
// Fig. 9.36: DrawPoly.java
// Drawing polygons
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Polygon;

public class DrawPoly extends Applet {
   // coordinates for first polygon
   private int xValues[] = { 20, 40, 50, 30, 20, 15, 20 };
   private int yValues[] = { 20, 20, 30, 50, 50, 30, 20 };

   // coordinates for second polygon
   private int xValues2[] = { 70, 90, 100, 80, 70, 65,
                              60, 70 };

   private int yValues2[] = { 70, 70, 80, 100, 100, 80,
                              60, 70 };

   // coordinates for third polygon
   private int xValues3[] = { 120, 140, 150, 190 };
   private int yValues3[] = { 10, 40, 50, 30 };

   // create references to polygons
   private Polygon p4, p5, p6;

   public void init()
   {
      // instantiate polygon objects
      p4 = new Polygon( xValues, yValues, 7 );
      p5 = new Polygon();
      p6 = new Polygon();

      // add points to p5
      p5.addPoint( 165, 105 );
      p5.addPoint( 175, 120 );
      p5.addPoint( 270, 170 );
      p5.addPoint( 200, 190 );
      p5.addPoint( 130, 150 );
      p5.addPoint( 165, 105 );

      // add points to p6
      p6.addPoint( 240, 50 );
      p6.addPoint( 260, 70 );
      p6.addPoint( 250, 90 );
   }

   public void paint( Graphics g )
   {
```

```
      // draw a polygon of 8 points
      g.drawPolygon( xValues2, yValues2, 8 );

      // draw a polygon object
      g.drawPolygon( p4 );

      // draw a filled polygon object
      g.fillPolygon( p5 );

      // draw a filled polygon object
      g.fillPolygon( p6 );

      // draw a filled polygon of 4 points
      g.fillPolygon( xValues3, yValues3, 4 );
   }
}
```

```java
// Fig. 9.38: DemoCopyArea.java
// Demonstrate copying one area of the screen to another
// area of the screen
import java.applet.Applet;
import java.awt.Graphics;

public class DemoCopyArea extends Applet {

   // coordinates for first polygon
   int xValues[] = { 20, 40, 50, 30, 20, 15, 20 };
   int yValues[] = { 20, 20, 30, 50, 50, 30, 20 };

   // coordinates for second polygon
   int xValues2[] = { 70, 90, 100, 80, 70, 65, 60, 70 };
   int yValues2[] = { 70, 70, 80, 100, 100, 80, 60, 70 };

   public void paint( Graphics g )
   {
      // draw a polygon of 7 points
      g.drawPolygon( xValues, yValues, 7 );

      // draw a filled polygon of 8 points
      g.fillPolygon( xValues2, yValues2, 8 );

      // copy 100 x 100 area of applet to (140, 10)
      g.copyArea( 0, 0, 100, 100, 140, 10 );
   }
}
```

```java
// Fig. 9.39: PaintMode.java
// Demonstrating the XOR paint mode
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;

public class PaintMode extends Applet {

   public void paint( Graphics g )
   {
      // draw pink oval
      g.setColor( Color.pink );
      g.fillOval( 20, 10, 100, 50 );

      // draw a yellow rectangle over part of the oval
      g.setColor( Color.yellow );
      g.fillRect( 100, 10, 100, 50 );

      // draw an orange rectangle
      g.setColor( Color.orange );
      g.fillRect( 190, 10, 80, 50 );

      // set XOR mode to yellow
      g.setXORMode( Color.yellow );
      g.fillOval( 180, 25, 60, 20 );

      // draw a blue arc
      g.setColor( Color.blue );
      g.fillArc( 150, 20, 20, 20, 0, 360 );

      // draw a red square
      g.setColor( Color.red );
      g.fillRect( 120, 25, 20, 20 );
   }
}
```

```java
// Fig. 10.5: MyLabel.java
// Demonstrating the Label class.
import java.applet.Applet;
import java.awt.*;

public class MyLabel extends Applet {
   private Label label1, label2;

   public void init()
   {
      // call label constructor with no text
      label1 = new Label();

      // set the label's text
      label1.setText( "Started with no text" );

      // call label constructor with a string argument
      label2 = new Label( "This is read-only text" );

      // add label components to applet container
      add( label1 );
      add( label2 );
   }

   public void paint( Graphics g )
   {
      g.drawString( "label1's text is: " + label1.getText(),
         25, 40 );
      g.drawString( "label2's text is: " + label2.getText(),
         25, 55 );
   }
}
```

```
// Fig. 10.8: MyButtons.java
// Creating push buttons.
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class MyButtons extends Applet {
   private Button pushButton1, pushButton2;

   public void init()
   {
      // create buttons
      pushButton1 = new Button( "Button 1" );
      pushButton1.addActionListener(
         new Button1Handler( this ) );
      add( pushButton1 );

      pushButton2 = new Button( "Button 2" );
      pushButton2.addActionListener(
         new Button2Handler( this ) );
      add( pushButton2 );
   }
}

class Button1Handler implements ActionListener {
   Applet applet;

   public Button1Handler( Applet a ) { applet = a; }

   public void actionPerformed( ActionEvent e )
   {
      applet.showStatus( "You pressed: " +
         e.getActionCommand() );
   }
}

class Button2Handler implements ActionListener {
   Applet applet;

   public Button2Handler( Applet a ) { applet = a; }

   public void actionPerformed( ActionEvent e )
   {
      applet.showStatus( "You pressed: " +
         e.paramString() );
   }
}
```

```
// Fig. 10.10: MyTextfield.java
// Demonstrating the TextField class.
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class MyTextfield extends Applet {
   private TextField text1, text2, text3, text4;
   private TextFieldHandler handler;

   public void init()
   {
      setBackground( Color.lightGray );

      // setup handler object
      handler = new TextFieldHandler( this );

      // construct textfield with default sizing
      text1 = new TextField();
      text1.addActionListener( handler );
      add( text1 );

      // construct textfield with default text
      text2 = new TextField( "Enter text here" );
      text2.addActionListener( handler );
      add( text2 );

      // construct textfield with default text
      text3 = new TextField( "Hidden text" );
      text3.setEchoChar( '*' );
      text3.addActionListener( handler );
      add( text3 );

      // construct textfield with default text and
      // 40 visible elements and no event handler
      text4 = new TextField( "Uneditable text field", 40 );
      text4.setEditable( false );
      add( text4 );
   }
}

class TextFieldHandler implements ActionListener {
   private Applet applet;

   public TextFieldHandler( Applet a ) { applet = a; }

   public void actionPerformed( ActionEvent e )
   {
```

```
      applet.showStatus( "Text is: " +
         e.getActionCommand() );
   }
}
```

```
// Fig. 10.12: MyChoice.java
// Using a Choice button to select a font.
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class MyChoice extends Applet {
   private Choice fonts;
   private TextField t;

   public void init()
   {
      // Create a Choice button and add items to it
      fonts = new Choice();
      fonts.add( "Monospaced" );  // Courier
      fonts.add( "SansSerif" );   // Helvetica
      fonts.add( "Serif" );       // Times

      t = new TextField( fonts.getItem( 0 ), 30 );
      t.setEditable( false );
      t.setFont( new Font( fonts.getItem( 0 ),
         Font.PLAIN, 12 ) );

      fonts.addItemListener( new FontNameHandler( t ) );
      fonts.addItemListener( new SetTextFieldHandler( t ) );

      add( fonts );
      add( t );
   }
}

class SetTextFieldHandler implements ItemListener {
   private TextField field;

   public SetTextFieldHandler( TextField t ) { field = t; }

   public void itemStateChanged( ItemEvent e )
   {
      Choice choice = (Choice) e.getItemSelectable();

      field.setText( "Index: " + choice.getSelectedIndex() +
         "; Font: " + e.getItem() );
   }
}

class FontNameHandler implements ItemListener {
   private Component component;
```

```
   public FontNameHandler( Component c ) { component = c; }

   public void itemStateChanged( ItemEvent e )
   {
      component.setFont( new Font( e.getItem().toString(),
         component.getFont().getStyle(),
         component.getFont().getSize() ) );
   }
}
```

```
// Fig. 10.14: MyCheckbox.java
// Creating Checkbox buttons.
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class MyCheckbox extends Applet implements ItemListener {
   private TextField t;
   private Checkbox bold, italic;

   public void init()
   {
      t = new TextField( "Watch the font style change", 40 );
      t.setFont( new Font( "Serif", Font.PLAIN, 14 ) );
      add( t );

      // create checkbox objects
      bold = new Checkbox( "Bold" );
      bold.addItemListener( this );
      add( bold );

      italic = new Checkbox( "Italic" );
      italic.addItemListener( this );
      add( italic );
   }

   public void itemStateChanged( ItemEvent e )
   {
      // test state of each Checkbox
      int valBold =
         ( bold.getState() ? Font.BOLD : Font.PLAIN );
      int valItalic =
         ( italic.getState() ? Font.ITALIC : Font.PLAIN );

      t.setFont(
         new Font( "Serif", valBold + valItalic, 14 ) );
   }
}
```

```
// Fig. 10.15: RadioButton.java
// Creating radio buttons using CheckboxGroup and Checkbox.
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class RadioButton extends Applet
             implements ItemListener {
   private TextField t;
   private Font plainFont, boldFont, italicFont, boldItalicFont;
   private CheckboxGroup fontStyle;
   private Checkbox plain, bold, italic, boldItalic;

   public void init()
   {
      t = new TextField( "Watch the font style change", 40 );
      add( t );

      // Create CheckboxGroup to maintain radio buttons
      fontStyle = new CheckboxGroup();

      plain = new Checkbox( "Plain", fontStyle, true );
      plain.addItemListener( this );
      add( plain );
      bold = new Checkbox( "Bold", fontStyle, false );
      bold.addItemListener( this );
      add( bold );
      italic = new Checkbox( "Italic", fontStyle, false );
      italic.addItemListener( this );
      add( italic );
      boldItalic = new Checkbox( "Bold/Italic",
                                 fontStyle, false );
      boldItalic.addItemListener( this );
      add( boldItalic );

      plainFont = new Font( "Serif", Font.PLAIN, 14 );
      boldFont = new Font( "Serif", Font.BOLD, 14 );
      italicFont = new Font( "Serif", Font.ITALIC, 14 );
      boldItalicFont = new Font( "Serif",
                                 Font.BOLD + Font.ITALIC, 14 );
      t.setFont( plainFont );
   }

   public void itemStateChanged( ItemEvent e )
   {
      if ( e.getSource() == plain )
         t.setFont( plainFont );
      else if ( e.getSource() == bold )
```

```
         t.setFont( boldFont );
      else if ( e.getSource() == italic )
         t.setFont( italicFont );
      else if ( e.getSource() == boldItalic )
         t.setFont( boldItalicFont );
   }
}
```

```
// Fig. 10.16: MyList.java
// Creating a List.
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class MyList extends Applet
             implements ActionListener, 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 };

   public void init()
   {
      // create a list with 5 items visible
      // do not allow multiple selections
      colorList = new List( 5, false );
      colorList.addActionListener( this );
      colorList.addItemListener( this );

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

      // add list to applet
      add( colorList );
   }

   public void actionPerformed( ActionEvent e )
   {
      setBackground( colors[ colorList.getSelectedIndex() ] );
      repaint();     // update applet
   }

   public void itemStateChanged( ItemEvent e )
   {
      showStatus( colorList.getSelectedItem() + "; index " +
                  colorList.getSelectedIndex() );
   }
```

```
}
```

```java
// Fig. 10.17: MyList2.java
// Copying items from one List to another.
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class MyList2 extends Applet implements ActionListener {
   private List colorList, copyList;
   private Button copy;
   private String colorNames[] =
      { "Black", "Blue", "Cyan", "Dark Gray", "Gray", "Green",
        "Light Gray", "Magenta", "Orange", "Pink", "Red",
        "White", "Yellow" };

   public void init()
   {
      // create a list with 5 items visible
      // allow multiple selections
      colorList = new List( 5, true );

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

      add( colorList );

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

      // create a list with 5 items visible
      // do not allow multiple selections
      copyList = new List( 5, false );
      add( copyList );
   }

   public void actionPerformed( ActionEvent e )
   {
      String colors[];

      // get the selected states
      colors = colorList.getSelectedItems();

      // copy them to copyList
      for ( int i = 0; i < colors.length; i++ )
         copyList.add( colors[ i ] );
   }
```

```
}
```