Introduction to AWT: working with frames, graphics, color, font. AWT Control fundamentals. Swing overview. Java database connectivity: JDBC overview, creating and executing queries, dynamic queries.

Control Fundamentals

The AWT supports the following types of controls:

- Labels
- Push buttons
- Check boxes
- Choice lists
- Lists
- Scroll bars
- Text Editing

Adding and Removing Controls

- Component add(Component compObj)
- void remove(Component obj)
-

1.Labels

CONSTRUCTORS

- Label( )
- Label(String str)
- Label(String str, int how)

The first version creates a blank label. The second version creates a label that contains the string specified by str. This string is left-justified. The third version creates a label that contains the string specified by str using the alignment specified by how. The value of how must be one of these three constants: **Label.LEFT, Label.RIGHT, or Label.CENTER.**

METHODS

- void setText(String str)- set or change the text in a label
- String getText( )-the current label is returned.
- void setAlignment(int how)- set the alignment of the string within the label
- int getAlignment( )-obtain the current alignment

The following example creates three labels and adds them to an applet window:

```
// Demonstrate Labels
import java.awt.*;
import java.applet.*;
/*
<applet code="LabelDemo" width=300 height=200>
</applet>
*/

public class LabelDemo extends Applet {
  public void init() {
    Label one = new Label("One");
    Label two = new Label("Two");
    Label three = new Label("Three");

    // add labels to applet window
    add(one);
    add(two);
    add(three);
  }
}
```

2. Buttons

CONSTRUCTORS

- Button( )
- Button(String str)

METHODS

- void setLabel(String str)
- String getLabel( )

Each time a button is pressed, an action event is generated. This is sent to any listeners that previously registered an interest in receiving action event notifications from that component. Each listener implements the **ActionListener interface.** That interface defines the **actionPerformed( )** method, which is called when an event occurs.

```
public class ButtonDemo extends Applet implements ActionListener {
  String msg = "";
  Button yes, no, maybe;

  public void init() {
    yes = new Button("Yes");
    no = new Button("No");
    maybe = new Button("Undecided");

    add(yes);
    add(no);
    add(maybe);

    yes.addActionListener(this);
    no.addActionListener(this);
    maybe.addActionListener(this);
  }

  public void actionPerformed(ActionEvent ae) {
    String str = ae.getActionCommand();

    if(str.equals("Yes")) {
      msg = "You pressed Yes.";
    }
    else if(str.equals("No")) {
      msg = "You pressed No.";
    }
    else {
      msg = "You pressed Undecided.";
    }

    repaint();
  }

  public void paint(Graphics g) {
     g.drawString(msg, 6, 100);
  }
}
```

3. Check Boxes

CONSTRUCTORS

- Checkbox( )
- Checkbox(String str)
- Checkbox(String str, boolean on)
- Checkbox(String str, boolean on, CheckboxGroup cbGroup)
- Checkbox(String str, CheckboxGroup cbGroup, boolean on)

The fourth and fifth forms create a check box whose label is specified by str and whose group is specified by cbGroup. If this check box is not part of a group, then cbGroup must be null.

METHODS

- boolean getState( )
- void setState(boolean on)
- String getLabel( )
- void setLabel(String str)

To retrieve the current state of a check box, call getState( ). To set its state, call setState( ). You can obtain the current label associated with a check box by calling getLabel( ). To set the label, call setLabel( ).

```
public class CheckboxDemo extends Applet implements ItemListener {
  String msg = "";
  Checkbox winXP, win7, solaris, mac;

  public void init() {
    winXP = new Checkbox("Windows XP", null, true);
    win7 = new Checkbox("Windows 7");
    solaris = new Checkbox("Solaris");
    mac = new Checkbox("Mac OS");

    add(winXP);
    add(win7);
    add(solaris);
    add(mac);



    winXP.addItemListener(this);
    win7.addItemListener(this);
    solaris.addItemListener(this);
    mac.addItemListener(this);
  }

  public void itemStateChanged(ItemEvent ie) {
    repaint();
  }

  // Display current state of the check boxes.
  public void paint(Graphics g) {
    msg = "Current state: ";
    g.drawString(msg, 6, 80);
    msg = "  Windows XP: " + winXP.getState();
    g.drawString(msg, 6, 100);
    msg = "  Windows 7: " + win7.getState();
    g.drawString(msg, 6, 120);
    msg = "  Solaris: " + solaris.getState();
    g.drawString(msg, 6, 140);
    msg = "  Mac OS: " + mac.getState();
    g.drawString(msg, 6, 160);
  }
}
```

Figure 25-2 Sample output from the **CheckboxDemo** applet

4.Choice Controls

The Choice class is used to create a pop-up list of items from which the user may choose.

CONSTRUCTOR

- Choice()

METHODS

- void add(String name)- name is the name of the item being added. Items are added to the list in the order in which calls to add( ) occur.
- String getSelectedItem( )
- int getSelectedIndex( )
  The getSelectedItem( ) method returns a string containing the name of the item. getSelectedIndex( ) returns the index of the item

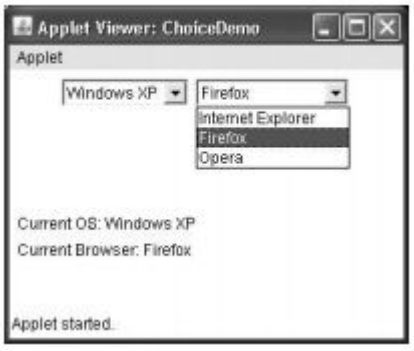
- int getItemCount( )
- void select(int index)
- void select(String name)
  To obtain the number of items in the list, call getItemCount( ). You can set the currently selected item using the select( ) method with either a zero-based integer index or a string that will match a name in the list.

- String getItem(int index)-obtain the name associated with the item at that index

Each listener implements the **ItemListener interface**. That interface defines the **itemStateChanged( )** method.

```
public class ChoiceDemo extends Applet implements ItemListener {
  Choice os, browser;
  String msg = "";

  public void init() {
    os = new Choice();
    browser = new Choice();

    // add items to os list
    os.add("Windows XP");
    os.add("Windows 7");
    os.add("Solaris");
    os.add("Mac OS");

    // add items to browser list
    browser.add("Internet Explorer");
    browser.add("Firefox");
    browser.add("Opera");

    // add choice lists to window
    add(os);
    add(browser);

    // register to receive item events
    os.addItemListener(this);
    browser.addItemListener(this);
  }

  public void itemStateChanged(ItemEvent ie) {
    repaint();
  }

  // Display current selections.
  public void paint(Graphics g) {
    msg = "Current OS: ";
    msg += os.getSelectedItem();
    g.drawString(msg, 6, 120);
    msg = "Current Browser: ";
    msg += browser.getSelectedItem();
    g.drawString(msg, 6, 140);
  }
}
```

Sample output is shown in Figure 25-4.



5. Lists

The List class provides a compact, multiple-choice, scrolling selection list.

CONSTRUCTORS

- List( )
- List(int numRows)
- List(int numRows, boolean multipleSelect)

numRows specifies the number of entries in the list that will always be visible

if multipleSelect is true, then the user may select two or more items at a time. If it is false, then only one item may be selected.

METHODS

To add a selection to the list, call add( ). It has the following two forms:

- void add(String name)
- void add(String name, int index)

name is the name of the item added to the list. The first form adds items to the end of the list. The second form adds the item at the index specified by index. Indexing begins at zero.

- String getSelectedItem( )
- int getSelectedIndex( )

The getSelectedItem( ) method returns a string containing the name of the item. If more than one item is selected, or if no selection has yet been made, null is returned. getSelectedIndex( ) returns the index of the item.

- String[ ] getSelectedItems( )

getSelectedItems( ) returns an array containing the names of the currently selected items.

- int[ ] getSelectedIndexes( )

getSelectedIndexes( ) returns an array containing the indexes of the currently selected items.

- int getItemCount( )
- void select(int index)
- String getItem(int index)

Here, index specifies the index

Lists implement the ActionListener interface. **getActionCommand( )** method can be used to retrieve the name of the newly selected item.

```
// Demonstrate Lists.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
  <applet code="ListDemo" width=300 height=180>
  </applet>
*/

public class ListDemo extends Applet implements ActionListener {
  List os, browser;
  String msg = "";

  public void init() {
    os = new List(4, true);
    browser = new List(4, false);
```

```
    // add items to os list
    os.add("Windows XP");
    os.add("Windows 7");
    os.add("Solaris");
    os.add("Mac OS");

    // add items to browser list
    browser.add("Internet Explorer");
    browser.add("Firefox");
    browser.add("Opera");

    browser.select(1);

    // add lists to window
    add(os);
    add(browser);

    // register to receive action events
    os.addActionListener(this);
    browser.addActionListener(this);
  }

  public void actionPerformed(ActionEvent ae) {
    repaint();
  }

  // Display current selections.
  public void paint(Graphics g) {
    int idx[];

    msg = "Current OS: ";
    idx = os.getSelectedIndexes();
    for(int i=0; i<idx.length; i++)
      msg += os.getItem(idx[i]) + "  ";
    g.drawString(msg, 6, 120);
    msg = "Current Browser: ";
    msg += browser.getSelectedItem();
    g.drawString(msg, 6, 140);
  }
}
```

6. Scroll Bars

Scroll bars are used to select continuous values between a specified minimum and maximum. Scroll bars may be oriented horizontally or vertically.

CONSTRUCTORS

- Scrollbar( )
- Scrollbar(int style)
- Scrollbar(int style, int initialValue, int thumbSize, int min, int max)

❖ The first form creates a vertical scroll bar.

❖ The second and third forms allow you to specify the orientation of the scroll bar.

❖ If style is Scrollbar.VERTICAL, a vertical scroll bar is created.

❖ If style is Scrollbar.HORIZONTAL, the scroll bar is horizontal.

❖ In the third form of the constructor, the initial value of the scroll bar is passed in initialValue. The number of units represented by the height of the thumb is passed in

thumbSize. The minimum and maximum values for the scroll bar are specified by min and max.

METHODS

- void setValues(int initialValue, int thumbSize, int min, int max)
- int getValue( ) -obtain the current value of the scroll bar
- void setValue(int newValue)-newValue specifies the new value for the scroll bar
- int getMinimum( )
- int getMaximum( )
- void setUnitIncrement(int newIncr)
- void setBlockIncrement(int newIncr)

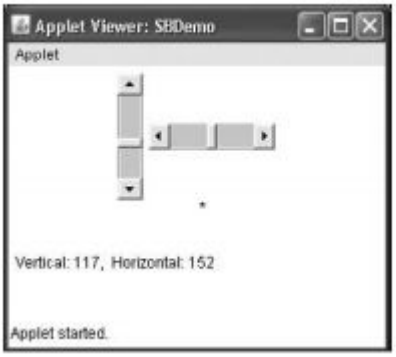
By default, 1 is the increment added to or subtracted from the scroll bar each time it is scrolled up or down one line. You can change this increment by calling setUnitIncrement( ). By default, page-up and page-down increments are 10.

To process scroll bar events, you need to implement the **AdjustmentListener i**nterface. **getAdjustmentType( )** method can be used to determine the type of the adjustment. The types of adjustment events are as follows:

BLOCK_DECREMENT	A page-down event has been generated.
BLOCK_INCREMENT	A page-up event has been generated.
TRACK	An absolute tracking event has been generated.
UNIT_DECREMENT	The line-down button in a scroll bar has been pressed.
UNIT_INCREMENT	The line-up button in a scroll bar has been pressed.

```
// Demonstrate scroll bars.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
  <applet code="SBDemo" width=300 height=200>
  </applet>
*/

public class SBDemo extends Applet
  implements AdjustmentListener, MouseMotionListener {
  String msg = "";
  Scrollbar vertSB, horzSB;
```

```
public void init() {
  int width = Integer.parseInt(getParameter("width"));
  int height = Integer.parseInt(getParameter("height"));

  vertSB = new Scrollbar(Scrollbar.VERTICAL,
                         0, 1, 0, height);
  vertSB.setPreferredSize(new Dimension(20, 100));

  horzSB = new Scrollbar(Scrollbar.HORIZONTAL,
                          0, 1, 0, width);
  horzSB.setPreferredSize(new Dimension(100, 20));

  add(vertSB);
  add(horzSB);

  // register to receive adjustment events
  vertSB.addAdjustmentListener(this);
  horzSB.addAdjustmentListener(this);

  addMouseMotionListener(this);
}

public void adjustmentValueChanged(AdjustmentEvent ae) {
  repaint();
}

// Update scroll bars to reflect mouse dragging.
public void mouseDragged(MouseEvent me) {
  int x = me.getX();
  int y = me.getY();
  vertSB.setValue(y);
  horzSB.setValue(x);
  repaint();
}


  // Necessary for MouseMotionListener
  public void mouseMoved(MouseEvent me) {
  }

  // Display current value of scroll bars.
  public void paint(Graphics g) {
     msg = "Vertical: " + vertSB.getValue();
     msg += ",  Horizontal: " + horzSB.getValue();
     g.drawString(msg, 6, 160);

     // show current mouse drag position
     g.drawString("*", horzSB.getValue(),
                  vertSB.getValue());
  }
}
```



7.TextField

CONSTRUCTORS

- TextField( )

- TextField(int numChars)
- TextField(String str)
- TextField(String str, int numChars)

METHODS

- String getText( )
- void setText(String str)
- String getSelectedText( )- obtain the currently selected text
- void select(int startIndex, int endIndex)
- boolean isEditable( )
- void setEditable(boolean canEdit)

isEditable( ) returns true if the text may be changed and false if not. In setEditable( ), if canEdit is true, the text may be changed. If it is false, the text cannot be altered.

There may be times when you will want the user to enter text that is not displayed, such as a password. You can disable the echoing of the characters as they are typed by calling setEchoChar( )

- void setEchoChar(char ch)
- boolean echoCharIsSet( )
- char getEchoChar( )

ch specifies the character to be echoed. If ch is zero, then normal echoing is restored.

```
// Demonstrate text field.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
  <applet code="TextFieldDemo" width=380 height=150>
  </applet>
*/

public class TextFieldDemo extends Applet
  implements ActionListener {

  TextField name, pass;

  public void init() {
    Label namep = new Label("Name: ", Label.RIGHT);
    Label passp = new Label("Password: ", Label.RIGHT);
    name = new TextField(12);
    pass = new TextField(8);
    pass.setEchoChar('?');
```

```
    add(namep);
    add(name);
    add(passp);
    add(pass);

    // register to receive action events
    name.addActionListener(this);
    pass.addActionListener(this);
  }

  // User pressed Enter.
  public void actionPerformed(ActionEvent ae) {
    repaint();
  }

  public void paint(Graphics g) {
    g.drawString("Name: " + name.getText(), 6, 60);
    g.drawString("Selected text in name: "
                 + name.getSelectedText(), 6, 80);
    g.drawString("Password: " + pass.getText(), 6, 100);
  }
}
```

8. TextArea

CONSTRUCTORS

- TextArea( )
- TextArea(int numLines, int numChars)
- TextArea(String str)
- TextArea(String str, int numLines, int numChars)
- TextArea(String str, int numLines, int numChars, int sBars)

Here, numLines specifies the height, in lines, of the text area, and numChars specifies its width, in characters. Initial text can be specified by str. In the fifth form, you can specify the scroll bars that you want the control to have. sBars must be one of these values:

SCROLLBARS_BOTH	SCROLLBARS_NONE
SCROLLBARS_HORIZONTAL_ONLY	SCROLLBARS_VERTICAL_ONLY

METHODS

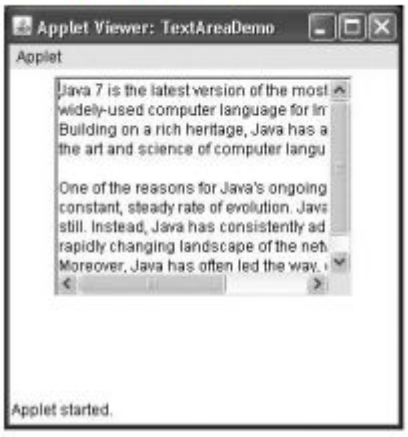
- void append(String str)
- void insert(String str, int index)
- void replaceRange(String str, int startIndex, int endIndex)

The append( ) method appends the string specified by str to the end of the current text. insert( ) inserts the string passed in str at the specified index. To replace text, call replaceRange( ). It replaces the characters from startIndex to endIndex–1, with the replacement text passed in str.

The following program creates a **TextArea** control:

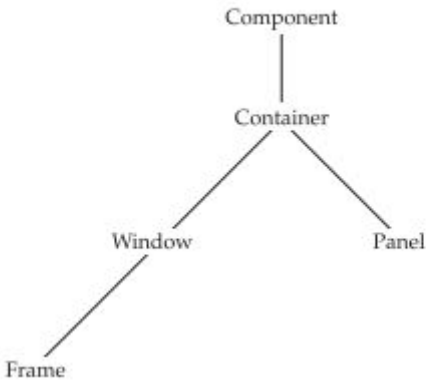
```
// Demonstrate TextArea.
import java.awt.*;
import java.applet.*;
/*
<applet code="TextAreaDemo" width=300 height=250>
</applet>
*/

public class TextAreaDemo extends Applet {
  public void init() {
    String val =
      "Java 7 is the latest version of the most\n" +
      "widely-used computer language for Internet programming.\n" +
      "Building on a rich heritage, Java has advanced both\n" +
      "the art and science of computer language design.\n\n" +
      "One of the reasons for Java's ongoing success is its\n" +
      "constant, steady rate of evolution. Java has never stood\n" +
      "still. Instead, Java has consistently adapted to the\n" +
      "rapidly changing landscape of the networked world.\n" +
      "Moreover, Java has often led the way, charting the\n" +
      "course for others to follow.";

    TextArea text = new TextArea(val, 10, 30);
    add(text);
  }
}
```

Sample output from the **TextAreaDemo** applet:



Introducing the AWT: Working with Windows, Graphics, and Text

Window Fundamentals



The AWT defines windows according to a class hierarchy that adds functionality and specificity with each level.
The two most common windows are those derived from Panel, which is used by applets, and those derived from Frame, which creates a standard application window.

Component

At the top of the AWT hierarchy is the Component class. Component is an abstract class that encapsulates all of the attributes of a visual component.

Container

The Container class is a subclass of Component. It has additional methods that allow other Component objects to be nested within it. Other Container objects can be stored inside of a Container .

Panel

The Panel class is a concrete subclass of Container. A Panel may be thought of as a recursively nestable, concrete screen component. Panel is the superclass for Applet. When screen output is directed to an applet, it is drawn on the surface of a Panel object. In essence, a Panel is a window that does not contain a title bar, menu bar, or border.

Window

The Window class creates a top-level window. A top-level window is not contained within any other object; it sits directly on the desktop.

Frame

Frame encapsulates what is commonly thought of as a “window.” It is a subclass of Window and has a title bar, menu bar, borders, and resizing corners.

Canvas

Canvas encapsulates a blank window upon which you can draw.

I.Working with Frames

CONSTRUCTORS

- Frame( )
- Frame(String title)

METHODS

1.Setting the Window’s Dimensions

- void setSize(int newWidth, int newHeight)
- void setSize(Dimension newSize)
- Dimension getSize( ) - returns the current size of the window contained within the width and height fields of a Dimension object.

2.Hiding and Showing a Window

- void setVisible(boolean visibleFlag)

The component is visible if the argument to this method is true. Otherwise, it is hidden

3.Setting a Window’s Title

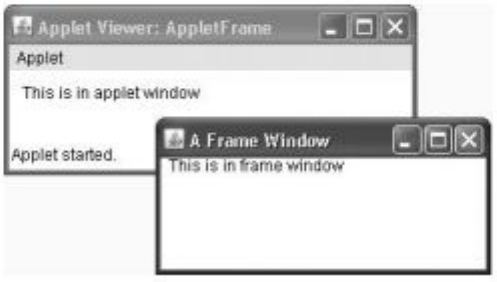
- void setTitle(String newTitle)

4.Closing a Frame Window

When using a frame window, your program must remove that window from the screen when it is closed, by calling **setVisible(false)**. To intercept a window-close event, you must implement the **windowClosing( )** method of the WindowListener interface. Inside windowClosing( ), you must remove the window from the screen.

```
// Create frame window.
public class AppletFrame extends Applet {
  Frame f;
  public void init() {
    f = new SampleFrame("A Frame Window");

    f.setSize(250, 250);
    f.setVisible(true);
  }

  public void start() {
    f.setVisible(true);
  }

  public void stop() {
    f.setVisible(false);
  }

  public void paint(Graphics g) {
    g.drawString("This is in applet window", 10, 20);
  }
}
```

Sample output from this program is shown here:



II.Working with Graphics

A graphics context is encapsulated by the Graphics class and is obtained in two ways:

• It is passed to a method, such as paint( ) or update( ), as an argument.

• It is returned by the getGraphics( ) method of Component.

1.Drawing Lines

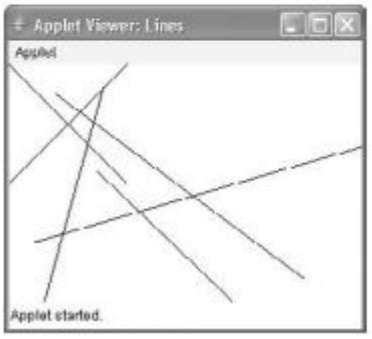
- void drawLine(int startX, int startY, int endX, int endY)

drawLine( ) displays a line in the current drawing color that begins at startX, startY and ends at endX, endY.

```
// Draw lines
import java.awt.*;
import java.applet.*;
/*
<applet code="Lines" width=300 height=200>
</applet>
*/
public class Lines extends Applet {
  public void paint(Graphics g) {
    g.drawLine(0, 0, 100, 100);
```

```
    g.drawLine(0, 100, 100, 0);
    g.drawLine(40, 25, 250, 180);
    g.drawLine(75, 90, 400, 400);
    g.drawLine(20, 150, 400, 40);
    g.drawLine(5, 290, 80, 19);
  }
}
```

Sample output from this program is shown here:



2.Drawing Rectangles

- void drawRect(int top, int left, int width, int height)
- void fillRect(int top, int left, int width, int height)

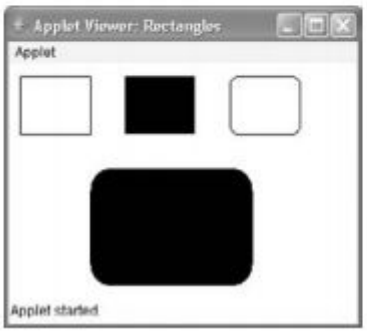
To draw a rounded rectangle

- void drawRoundRect(int top, int left, int width, int height, int xDiam, int yDiam)
- void fillRoundRect(int top, int left, int width, int height, int xDiam, int yDiam)

```
// Draw rectangles
import java.awt.*;
import java.applet.*;
/*
<applet code="Rectangles" width=300 height=200>
</applet>
*/
```

```
public class Rectangles extends Applet {
  public void paint(Graphics g) {
    g.drawRect(10, 10, 60, 50);
    g.fillRect(100, 10, 60, 50);
    g.drawRoundRect(190, 10, 60, 50, 15, 15);
    g.fillRoundRect(70, 90, 140, 100, 30, 40);
  }
}
```

Sample output from this program is shown here:

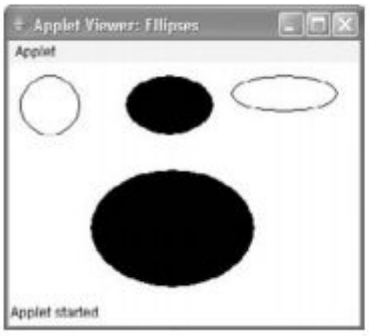


3.Drawing Ellipses and Circles

- void drawOval(int top, int left, int width, int height)
- void fillOval(int top, int left, int width, int height)

```
// Draw Ellipses
import java.awt.*;
import java.applet.*;
/*
<applet code="Ellipses" width=300 height=200>
</applet>
*/

public class Ellipses extends Applet {
  public void paint(Graphics g) {
    g.drawOval(10, 10, 50, 50);
    g.fillOval(100, 10, 75, 50);
    g.drawOval(190, 10, 90, 30);
    g.fillOval(70, 90, 140, 100);
  }
}
```

Sample output from this program is shown here:

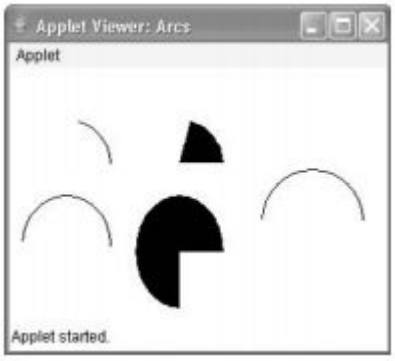


4.Drawing Arcs

- void drawArc(int top, int left, int width, int height, int startAngle, int sweepAngle)
- void fillArc(int top, int left, int width, int height, int startAngle, int sweepAngle)

The arc is bounded by the rectangle whose upper-left corner is specified by top, left and whose width and height are specified by width and height. The arc is drawn from startAngle through the angular distance specified by sweepAngle. Angles are specified in degrees. Zero degrees is on the horizontal, at the three o'clock position. The arc is drawn counterclockwise if sweepAngle is positive, and clockwise if sweepAngle is negative.

```
// Draw Arcs
import java.awt.*;
import java.applet.*;
/*
<applet code="Arcs" width=300 height=200>
</applet>
*/

public class Arcs extends Applet {
  public void paint(Graphics g) {
    g.drawArc(10, 40, 70, 70, 0, 75);
    g.fillArc(100, 40, 70, 70, 0, 75);
    g.drawArc(10, 100, 70, 80, 0, 175);
    g.fillArc(100, 100, 70, 90, 0, 270);
    g.drawArc(200, 80, 80, 80, 0, 180);
  }
}
```

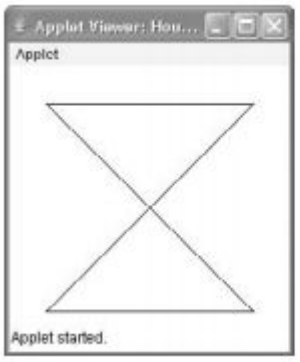


5.Drawing Polygons

- void drawPolygon(int x[ ], int y[ ], int numPoints)
- void fillPolygon(int x[ ], int y[ ], int numPoints)

The polygon's endpoints are specified by the coordinate pairs contained within the x and y arrays. The number of points defined by x and y is specified by numPoints.

```
// Draw Polygon
import java.awt.*;
import java.applet.*;
/*
<applet code="HourGlass" width=230 height=210>
</applet>
*/

public class HourGlass extends Applet {
  public void paint(Graphics g) {
    int xpoints[] = {30, 200, 30, 200, 30};
    int ypoints[] = {30, 30, 200, 200, 30};
    int num = 5;

    g.drawPolygon(xpoints, ypoints, num);
  }
}
```



III.Working with Color

CONSTRUCTORS

- Color(int red, int green, int blue)

The first constructor takes three integers that specify the color as a mix of red, green, and blue. These values must be between 0 and 255,

- Color(int rgbValue)

The second color constructor takes a single integer that contains the mix of red, green, and blue packed into an integer. The integer is organized with red in bits 16 to 23, green in bits 8 to 15, and blue in bits 0 to 7.

- Color(float red, float green, float blue)

The final constructor, Color(float, float, float), takes three float values (between 0.0 and 1.0) that specify the relative mix of red, green, and blue.

METHODS

1.Using Hue, Saturation, and Brightness

- static int HSBtoRGB(float hue, float saturation, float brightness)
- static float[ ] RGBtoHSB(int red, int green, int blue, float values[ ])

HSBtoRGB( ) returns a packed RGB value compatible with the Color(int) constructor.
RGBtoHSB( ) returns a float array of HSB values corresponding to RGB integers.

2.getRed( ), getGreen( ), getBlue( )

- int getRed( )
- int getGreen( )
- int getBlue( )
- int getRGB( )

3.Setting the Current Graphics Color

- void setColor(Color newColor)
- Color getColor( )- obtain the current color

```
// Demonstrate color.
import java.awt.*;
import java.applet.*;
/*
<applet code="ColorDemo" width=300 height=200>
</applet>
*/

public class ColorDemo extends Applet {
  // draw lines
  public void paint(Graphics g) {
    Color c1 = new Color(255, 100, 100);
    Color c2 = new Color(100, 255, 100);
    Color c3 = new Color(100, 100, 255);

    g.setColor(c1);
    g.drawLine(0, 0, 100, 100);
    g.drawLine(0, 100, 100, 0);

    g.setColor(c2);
    g.drawLine(40, 25, 250, 180);
    g.drawLine(75, 90, 400, 400);

    g.setColor(c3);
    g.drawLine(20, 150, 400, 40);
    g.drawLine(5, 290, 80, 19);

    g.setColor(Color.red);
    g.drawOval(10, 10, 50, 50);
    g.fillOval(70, 90, 140, 100);

    g.setColor(Color.blue);
    g.drawOval(190, 10, 90, 30);
    g.drawRect(10, 10, 60, 50);

    g.setColor(Color.cyan);
    g.fillRect(100, 10, 60, 50);
    g.drawRoundRect(190, 10, 60, 50, 15, 15);
  }
}
```

IV.Working with Fonts

Constructors

- Font(String fontName, int fontStyle, int pointSize)

fontName specifies the name of the desired font.
The style of the font is specified by fontStyle. It may consist of one or more of these three constants: Font.PLAIN, Font.BOLD, and Font.ITALIC. To combine styles, OR them together. For example, Font.BOLD | Font.ITALIC specifies a bold, italics style.

METHODS

Method	Description
static Font decode(String *str*)	Returns a font given its name.
boolean equals(Object *FontObj*)	Returns **true** if the invoking object contains the same font as that specified by *FontObj*. Otherwise, it returns **false**.
String getFamily( )	Returns the name of the font family to which the invoking font belongs.
static Font getFont(String *property*)	Returns the font associated with the system property specified by *property*. **null** is returned if *property* does not exist.
static Font getFont(String *property*, Font *defaultFont*)	Returns the font associated with the system property specified by *property*. The font specified by *defaultFont* is returned if *property* does not exist.
String getFontName( )	Returns the face name of the invoking font.
String getName( )	Returns the logical name of the invoking font.
int getSize( )	Returns the size, in points, of the invoking font.
int getStyle( )	Returns the style values of the invoking font.
int hashCode( )	Returns the hash code associated with the invoking object.
boolean isBold( )	Returns **true** if the font includes the **BOLD** style value. Otherwise, **false** is returned.
boolean isItalic( )	Returns **true** if the font includes the **ITALIC** style value. Otherwise, **false** is returned.
boolean isPlain( )	Returns **true** if the font includes the **PLAIN** style value. Otherwise, **false** is returned.
String toString( )	Returns the string equivalent of the invoking font.

The Font class defines these variables:

Variable	Meaning
String name	Name of the font
float pointSize	Size of the font in points
int size	Size of the font in points
int style	Font style

- void setFont(Font fontObj)-To use a font that you have created, you must select it using setFont( ), which is defined by Component.
- Font getFont( )- obtain information about the currently selected font.
- String[ ] getAvailableFontFamilyNames( )-Determine the Available Fonts on your machine.
- Font[ ] getAllFonts( )- returns an array of Font objects for all of the available fonts.
- static GraphicsEnvironment getLocalGraphicsEnvironment( )-Since these methods are members of GraphicsEnvironment, you need a GraphicsEnvironment reference to call them. You can obtain this reference by using the getLocalGraphicsEnvironment( ) static method, which is defined by GraphicsEnvironment.

```
// Display Fonts
/*
<applet code="ShowFonts" width=550 height=60>
</applet>
*/
import java.applet.*;
import java.awt.*;

public class ShowFonts extends Applet {
  public void paint(Graphics g) {
    String msg = "";
    String FontList[];

    GraphicsEnvironment ge =
      GraphicsEnvironment.getLocalGraphicsEnvironment();
    FontList = ge.getAvailableFontFamilyNames();
    for(int i = 0; i < FontList.length; i++)
      msg += FontList[i] + " ";

     g.drawString(msg, 4, 16);
  }
}
```

Swing

The Swing component classes are:

JButton	JCheckBox	JComboBox	JLabel
JList	JRadioButton	JScrollPane	JTabbedPane
JTable	JTextField	JToggleButton	JTree

1.JLabel and ImageIcon
CONSTRUCTORS

- JLabel(Icon icon)
- JLabel(String str)
- JLabel(String str, Icon icon, int align)
- ImageIcon(String filename)

Values for align are:
LEFT, RIGHT, CENTER, LEADING, or TRAILING.

METHODS

- void setIcon(Icon icon)
- void setText(String str)
- Icon getIcon( )
- String getText( )

```
  private void makeGUI() {

    // Create an icon.
    ImageIcon ii = new ImageIcon("france.gif");

    // Create a label.
    JLabel jl = new JLabel("France", ii, JLabel.CENTER);

    // Add the label to the content pane.
    add(jl);
  }
}
```

2.JTextField

CONSTRUCTORS

- JTextField(int cols)
- JTextField(String str, int cols)
- JTextField(String str)

str is the string to be initially presented, and cols is the number of columns in the text field. If no string is specified, the text field is initially empty. If the number of columns is not specified, the text field is sized to fit the specified string.

```
  private void makeGUI() {

    // Change to flow layout.
    setLayout(new FlowLayout());

    // Add text field to content pane.
    jtf = new JTextField(15);
    add(jtf);
    jtf.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        // Show text when user presses ENTER.
        showStatus(jtf.getText());
      }
    });
  }
}
```

3.The Swing Buttons

Swing defines four types of buttons: JButton, JToggleButton, JCheckBox, and JRadioButton.

AbstractButton contains many methods that allow you to control the behavior of buttons. For example, you can define different icons that are displayed for the button when it is disabled, pressed, or selected. Another icon can be used as a rollover icon, which is displayed when the mouse is positioned over a button. The following methods set these icons:

- void setDisabledIcon(Icon di)
- void setPressedIcon(Icon pi)
- void setSelectedIcon(Icon si)
- void setRolloverIcon(Icon ri)

Here, di, pi, si, and ri are the icons to be used for the indicated purpose.

3.1 JButton

CONSTRUCTORS

- JButton(Icon icon)
- JButton(String str)
- JButton(String str, Icon icon)

String getActionCommand( )-Obtain the action command.

```
ImageIcon france = new ImageIcon("france.gif");
JButton jb = new JButton(france);
jb.setActionCommand("France");
jb.addActionListener(this);
add(jb);
```

3.2 JToggleButton

CONSTRUCTORS

- JToggleButton(String str)

METHODS

- Object getItem( )- obtain a reference to the JToggleButton instance that generated the event.
- boolean isSelected( )-It returns true if the button is selected and false otherwise.

```
    // Create a label.
    jlab = new JLabel("Button is off.");

    // Make a toggle button.
    jtbn =  new JToggleButton("On/Off");

    // Add an item listener for the toggle button.
    jtbn.addItemListener(new ItemListener() {
      public void itemStateChanged(ItemEvent ie) {
        if(jtbn.isSelected())
          jlab.setText("Button is on.");
        else
          jlab.setText("Button is off.");
      }

    });

    // Add the toggle button and label to the content pane.
    add(jtbn);
    add(jlab);
  }
}
```

4.Check Boxes

CONSTRUCTOR

- JCheckBox(String str)

It creates a check box that has the text specified by str as a label.
When the user selects or deselects a check box, an ItemEvent is generated. You can obtain a reference to the JCheckBox that generated the event by calling getItem( ) on the ItemEvent passed to the itemStateChanged( ) method defined by ItemListener.

```
private void makeGUI() {

  // Change to flow layout.
  setLayout(new FlowLayout());

  // Add check boxes to the content pane.
  JCheckBox cb = new JCheckBox("C");
  cb.addItemListener(this);
  add(cb);

  cb = new JCheckBox("C++");
  cb.addItemListener(this);
  add(cb);

  cb = new JCheckBox("Java");
  cb.addItemListener(this);
  add(cb);

  cb = new JCheckBox("Perl");
  cb.addItemListener(this);
  add(cb);
```

5.Radio Buttons

CONSTRUCTOR

- JRadioButton(String str)

str is the label for the button.

METHODS

Elements are then added to the button group via the following method:

- void add(AbstractButton ab)

Here, ab is a reference to the button to be added to the group.

```
private void makeGUI() {

  // Change to flow layout.
  setLayout(new FlowLayout());

  // Create radio buttons and add them to content pane.
  JRadioButton b1 = new JRadioButton("A");
  b1.addActionListener(this);
  add(b1);

  JRadioButton b2 = new JRadioButton("B");
  b2.addActionListener(this);
  add(b2);

  JRadioButton b3 = new JRadioButton("C");
  b3.addActionListener(this);
  add(b3);
```

6.JTabbedPane

The general procedure to use a tabbed pane is outlined here:

1. Create an instance of JTabbedPane.
2. Add each tab by calling addTab( ).

   Tabs are added by calling addTab( ).

   void addTab(String name, Component comp)
3. . Add the tabbed pane to the content pane.

```
private void makeGUI() {

  JTabbedPane jtp = new JTabbedPane();
  jtp.addTab("Cities", new CitiesPanel());
  jtp.addTab("Colors", new ColorsPanel());
  jtp.addTab("Flavors", new FlavorsPanel());
  add(jtp);
 }
}
```

7.JScrollPane

CONSTRUCTOR

- **JScrollPane(Component comp)**

The general procedure to use a SCROLL pane is outlined here:

1. Create the component to be scrolled.
2. . Create an instance of JScrollPane, passing to it the object to scroll.
3. . Add the scroll pane to the content pane.


```
    // Add 400 buttons to a panel.
    JPanel jp = new JPanel();
    jp.setLayout(new GridLayout(20, 20));
    int b = 0;

    for(int i = 0; i < 20; i++) {
      for(int j = 0; j < 20; j++) {
        jp.add(new JButton("Button " + b));
        ++b;
      }
    }

    // Create the scroll pane.
    JScrollPane jsp = new JScrollPane(jp);

    // Add the scroll pane to the content pane.
    // Because the default border layout is used,
    // the scroll pane will be added to the center.
    add(jsp, BorderLayout.CENTER);
  }
}
```