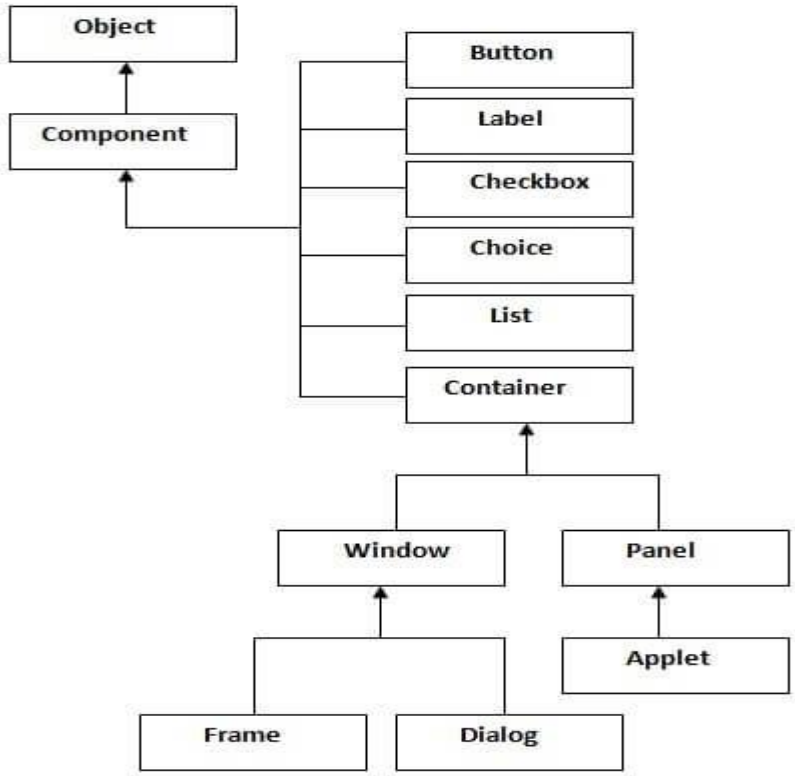
Abstract Window Toolkit (AWT):-

- **Java AWT** (Abstract Window Toolkit) is *an API to develop GUI or window-based applications* in java.
- Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.
- The java.awt package provides classes for AWT api such as TextField, Label, TextArea,RadioButton, CheckBox, Choice, List etc
- The AWT contains numerous classes and methods that allow you to create and manage windows.
- use the AWT when creating your own applets or stand-alone programs that run in a GUI environment, such as Windows.

- Most of the examples are contained in applets, so to run them, you need to use an applet viewer or a Java-compatible Web browser.

AWT Hierarchy:-The hierarchy of Java AWT classes are given below.



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 window.
- Much of the functionality of these windows is derived from their parent classes.

- ✓ **Component**
- ✓ Container
- ✓ **Panel**
- ✓ **Window**
- ✓ Frame
- ✓ **Canvas**

- **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.
- All user interface elements that are displayed on the screen and that interact with the user are subclasses of **Component**.
- It defines over a hundred public methods that are responsible for managing events, such as
    - mouse
    - keyboard input,
    - positioning ,
    - sizing the window,
    - repainting.
- A **Component** object is responsible for remembering the current foreground and background colors and the currently selected text font.

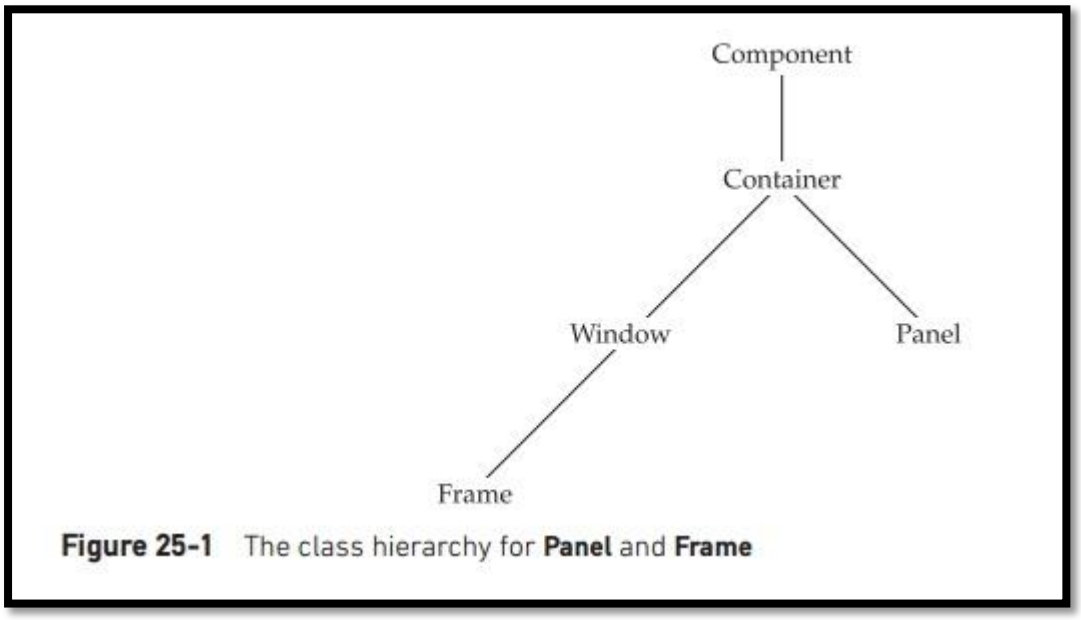


Figure 25-1 The class hierarchy for **Panel** and **Frame**

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.**

- This makes for a multileveled containment system.
- A container is responsible for laying out (that is, positioning) any components that it contains.
- It does this through the use of various layout managers.

Panel:-

- The **Panel** class is a concrete subclass of **Container**.
- It doesn't add any new methods; it simply implements **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.
- When you run an applet using an applet viewer, the applet viewer provides the title and border.

- Other components can be added to a **Panel** object by its **add( )** method (inherited from **Container**).
- Once these components have been added, you can position and resize them manually using the **setLocation( )**, **setSize( )**, or **setBounds( )** methods defined by **Component**.

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.
- Generally, you won't create **Window** objects directly. Instead, you will use a subclass of **Window** called **Frame**.

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:-

It is not part of the hierarchy for applet or frame windows. Canvas encapsulates a blank window upon which you can draw.

➢ Frame Windows:-

the applet, the type of window you will most often create is derived from **Frame**. A top-level window is called **Frame.** As mentioned, it creates a standard-style window.

Here are two of **Frame**'s **constructors:**

Frame( )
Frame(String *title*)

The first form creates a standard window that does not contain a title. The second form creates a window with the title specified by *title.* Notice that you cannot specify the dimensions of the window. Instead, you must set the size of the window after it has been created.

- **Setting the Window's Dimensions**

The **setSize( )** method is used to set the dimensions of the window.
shown here:

void setSize(int *newWidth*, int *newHeight*)

void setSize(Dimension *newSize*)

The new size of the window is specified by *newWidth* and *newHeight*, or by the **width** and **height** fields of the **Dimension** object passed in *newSize*. The dimensions are specified in terms of pixels.

The **getSize( )** method is used to obtain the current size of a window. Its signature is shown here:

- **Dimension getSize( )**

This method returns the current size of the window contained within the **width** and **height** fields of a **Dimension** object.

- **Hiding and Showing a Window**

After a frame window has been created, it will not be visible until you call **setVisible( )**. Its signature is shown here: void setVisible(boolean *visibleFlag*)
The component is visible if the argument to this method is **true**. Otherwise, it is hidden.

- **Setting a Window's Title**

You can change the title in a frame window using **setTitle( )**, which has this general form: void setTitle(String *newTitle*) Here, *newTitle* is the new title for the window.

- **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.

Example:-

```
import java.awt.*;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
public class WindowExample extends Frame{
    WindowExample(){
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e) {
                dispose();
            }
        });
        setSize(400,400);
        setLayout(null);
        setVisible(true);
    }
public static void main(String[] args) {
    new WindowExample();
}
```

Working with Graphics:-

- The AWT supports a rich assortment of graphics methods.
- All graphics are drawn relative to a window.
- This can be the main window of an applet, a child window of an applet, or a stand-alone application window.
- The origin of each window is at the top-left corner and is 0,0.
- Coordinates are specified in pixels.
- All output to a window takes place through a graphics context.
- A *graphics context* is encapsulated by the **Graphics** class and is obtained in two ways:

1. It is passed to an applet when one of its various methods, such as **paint( )** or **update( )**, is called.

2. It is returned by the **getGraphics( )** method of **Component**.
the same techniques will apply to any other window.

- The **Graphics** class defines a number of drawing functions.
- Each shape can be drawn edge-only or filled.

- Objects are drawn and filled in the currently selected graphics color, which is black by default.
- When a graphics object is drawn that exceeds the dimensions of the window, output is automatically clipped.
- Let's take a look at several of the drawing methods.

Drawing Lines:-

- Lines are drawn by means of the **drawLine( )** method, shown here:

- 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*.

The following applet draws several lines:

```
// Draw lines
import java.awt.*;
import java.applet.*;
/*
<applet code="Lines" width=300 height=200>
THE JAVA LIBRARY
</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);
        }
}
```

Drawing Rectangles:-

The **drawRect( )** and **fillRect( )** methods display an outlined and filled rectangle, respectively.

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

- The upper-left corner of the rectangle is at *top*,*left.*
- The dimensions of the rectangle are specified by *width* and *height.*

- To draw a rounded rectangle, use **drawRoundRect( )** or **fillRoundRect( )**, both shown here:

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*)

- Arounded rectangle has rounded corners.
- The upper-left corner of the rectangle is at *top*,*left.*
- The dimensions of the rectangle are specified by *width* and *height.*
- The diameter of the rounding arc along the X axis is specified by *xDiam.*
- The diameter of the rounding arc along the Y axis is specified by *yDiam.*
- The following applet draws several rectangles:

```
// 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);
        }
}
```

Drawing Ellipses and Circles:-

To draw an ellipse, use **drawOval( )**. To fill an ellipse, use **fillOval( )**.
These methods are:-

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

The ellipse is drawn within a bounding rectangle whose upper-left corner is specified by *top,left* and whose width and height are specified by *width* and *height*a square as the bounding rectangle.

```
// 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);
                }
}
```

Drawing Arcs:-

Arcs can be drawn with **drawArc( )** and **fillArc( )**,

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);
        }
}
```

Drawing Polygons:-

It is possible to draw arbitrarily shaped figures using **drawPolygon( )** and **fillPolygon( )**,

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.* There are alternative forms of these methods in which the polygon is specified by a **Polygon** object.

The following applet draws an hourglass shape:

```
// 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);
        }
}
```

➢ Color:-

- Java supports color in a portable, device-independent fashion.
- The AWT color system allows you to specify any color you want.
- It then finds the best match for that color, given the limits of the display hardware currently executing your program or applet.
- Thus, your code does not need to be concerned with the differences in the way color is supported by various hardware devices.
- Color is encapsulated by the Color class.

color constructors

To create your own colors,using one of the color constructors.the most commly used forms are shown here:

- **Color (int red, int green, int blue)**
- **Color (int rgbValue) //int rgbValue = (0xff000000 | (0xc0<<16) | (0x00 <<8) | 0x00);**
- **Color (float red, float green, float blue)**

The first constructors take three integers that specify the color as the mix of red,green and blue. these values must **be between 0 and 255.**

example:

```
new color(255,100,100); // light red.
```

- The second color constructor takes a single integer that contain 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.

example:

```
int newRed=(0xff000000 | (0xc0<<16) | (0x00 <<8) | 0x00);
color darkRed = new Color(newRed);
```

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

Color Methods

The color class defines several methods that help manipulate colors.they are:

- **Using Hue, Saturation, and Brightness:**

The Hue-Saturation- Brightness(HSB)color model is an alternatives to red-green-blue(rgb) for specifying particular colors.hue is wheel of color, value between 0.0 to 1.0.saturation is light pastels to intense hues, value between 0.0 to 1.0 . brightness: 1 is bright white and 0 is black, value between 0.0 to 1.0.

Color supplies two methods that let you convert between RGB andHSB. they are:

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

If values is not null, then this array is given the HSB values and returned. Otherwise, a new array is created and the HSB values are returned in it.

- **getRed(), getGreen(), and getBlue()**

you can obtain the red green,and blue components of a color independently using getRed(),getGreen(),and getBlue(), shown here:

syntax:

int getRed()
int getGreen()
int getBlue()

- **getRGB()**

To obtain a packed,RGB representation of a color,use getRGB().

syntax:

int getRGB()

Setting the current Graphics Color:-

By default ,graphics objects are drawn in the current foreground color.you cn change this color by calling the Graphics method setColor():

syntax:

void setColor(Color newColor)
here ,newColor specifies the new drawing color.

you can obtain current color by calling getColor,Shown here:

syntax:-
Color getColor()

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

        public classColorDemo extends Applet
```

```
        {
            //draw line
           public void main paint(Graphics g)
           {
               color c1 = new color(255,100,100);
                g.setColor(c1);
                g.drawLine(0,0,100,100);
               g.drawLine(0,100,100,0);

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

- **FontMetrics:-**

  - The FontMetrics class is used to return the specific parameters for a particular Font object.
  - An object of this class is created using the getFontMetrics() methods supported by the Component class and other classes, such as the Graphics and Toolkit classes.
  - The FontMetrics access methods provide access to the details of the implementation of a Font object.

- Given that the size of each font may differ and that fonts may be changed while your program is executing, there must be some way to determine the dimensions and various other attributes of the currently selected font.
- For example, to write one line of text after another implies that you have some way of knowing how tall the font is and how manypixels are needed between lines.
- To fill this need, the AWT includes the **FontMetrics** class, which encapsulates various information about a font. Let's begin by defining the commonterminology used when describing fonts:

Terminology	Description
Height	Height The top-to-bottom size of a line of text
Baseline	Baseline The line that the bottoms of characters are aligned to (not counting descent)
Ascent	Ascent The distance from the baseline to the top of a character
Descent	Descent The distance from the baseline to the bottom of a character
Leading	Leading The distance between the bottom of one line of text and the top of the next
Height	Height The top-to-bottom size of a line of text

Sampling of Methods Defined by FontMetrics

Method Description	Method Description
int bytesWidth(byte b[ ], int start, int numBytes)	Returns the width of numBytes characters held in array b beginning at start.
int charWidth(char c[ ], int start, int numChars)	Returns the width of numChars characters held in array c, beginning at start.
int charWidth(char c)	int charWidth(char c) Returns the width of c.
int charWidth(int c)	int charWidth(int c) Returns the width of c.
int getAscent( )	int getAscent( ) Returns the ascent of the font.
int getDescent( )	int getDescent( ) Returns the descent of the font.
Font getFont( )	Font getFont( ) Returns the font.
int getHeight( )	Returns the height of a line of text. This value can be used to output multiple lines of text in a window.
int getLeading( )	Returns the space between lines of text.
int getMaxAdvance( )	Returns the width of the widest character. –1 is returned if this value is not available.
int getMaxAscent( )	Returns the maximum ascent.
int getMaxDescent( )	Returns the maximum descent.
int[ ] getWidths( )	Returns the widths of the first 256 characters.
int stringWidth(String str)	Returns the width of the string specified by str.
String toString( )	Returns the string equivalent of the invoking object.

Displaying Multiple Lines of Text

- Perhaps the most common use of **FontMetrics** is to determine the spacing between lines of text.
- The second most common use is to determine the length of a string that is being displayed.
- display multiple lines of text, your program must manually keep track of the current output position.

- Each time a newline is desired, the Y coordinate must be advanced to the beginning of the next line.
- Each time a string is displayed, the X coordinate must be set to the point at which the string ends.
- This allows the next string to be written so that it begins at the end of the preceding one.

- To determine the spacing between lines, you can use the value returned by **getLeading( )**.
- To determine the total height of the font, add the value returned by **getAscent( )** to the value returned by **getDescent( )**.

```
// Demonstrate multiline output.
import java.applet.*;
import java.awt.*;
/*
<applet code="MultiLine" width=300 height=100>
</applet>
*/
public class MultiLine extends Applet {
int curX=0, curY=0; // current position
public void init() {
Font f = new Font("SansSerif", Font.PLAIN, 12);
setFont(f);
}
public void paint(Graphics g) {
        FontMetrics fm = g.getFontMetrics();
        nextLine("This is on line one.", g);
        nextLine("This is on line two.", g);
        sameLine(" This is on same line.", g);
        sameLine(" This, too.", g);
        nextLine("This is on line three.", g);
}
// Advance to next line.
void nextLine(String s, Graphics g) {
        FontMetrics fm = g.getFontMetrics();
        curY += fm.getHeight(); // advance to next line
        curX = 0;
        g.drawString(s, curX, curY);
        curX = fm.stringWidth(s); // advance to end of line
}
// Display on same line.
void sameLine(String s, Graphics g) {
        FontMetrics fm = g.getFontMetrics();
        g.drawString(s, curX, curY);
        curX += fm.stringWidth(s); // advance to end of line
}
}
```

Centering Text

Here is an example that centers text, left to right, top to bottom, in a window. It obtains the
ascent, descent, and width of the string and computes the position at which it must be displayed
to be centered.

```
// Center text.
import java.applet.*;
import java.awt.*;
/*
<applet code="CenterText" width=200 height=100>
</applet>
*/
public class CenterText extends Applet {
final Font f = new Font("SansSerif", Font.BOLD, 18);
public void paint(Graphics g) {
Dimension d = this.getSize();
g.setColor(Color.white);
g.fillRect(0, 0, d.width,d.height);
        g.setColor(Color.black);
g.setFont(f);
drawCenteredString("This is centered.", d.width, d.height, g);
g.drawRect(0, 0, d.width-1, d.height-1);
}
public void drawCenteredString(String s, int w, int h,
Graphics g) {
FontMetrics fm = g.getFontMetrics();
int x = (w - fm.stringWidth(s)) / 2;
int y = (fm.getAscent() + (h - (fm.getAscent()
+ fm.getDescent()))/2);
g.drawString(s, x, y);
}
}
```

Control Fundamentals:-

The AWT supports the following types of controls:

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

These controls are subclasses of **Component**.

Adding and Removing Controls:-

- To include a control in a window, you must add it to the window.
- To do this, you must first create an instance of the desired control and then add it to a window by calling **add( )**, which is defined by **Container**.
- The **add( )** method has several forms.

Component add(Component *compObj*)

Here, *compObj* is an instance of the control that you want to add. A reference to *compObj* is returned. Once a control has been added, it will automatically be visible whenever its parent window is displayed.

Sometimes you will want to remove a control from a window when the control is no longer needed. To do this, call **remove( )**.

void remove(Component *obj*)
Here, *obj* is a reference to the control you want to remove. You can remove all controls by calling **removeAll( )**.

Labels:-

- The easiest control to use is a label.
- A *label* is an object of type **Label**, and it contains a string, which it displays.
- Labels are passive controls that do not support any interaction with the user.

Label defines the following **constructors**:

- Label( ) throws HeadlessException
- Label(String *str*) throws HeadlessException
- Label(String *str*, int *how*) throws HeadlessException

these three constants:

- **Label.LEFT**,
- **Label.RIGHT**,
- **Label.CENTER**.

You can set or change the text in a label by using the **setText( )** method. You can obtain the current label by calling **getText( )**. **getText( )**, the current label is returned.

methods :-

- void setText(String *str*)
- String getText( )

set the alignment of the string within the label by calling **setAlignment( )**. To obtain the current alignment, call **getAlignment( )**.

Methods:-

- void setAlignment(int *how*)
- int getAlignment( )

Example:-

```
// 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);
        }
}
```

Buttons:-

- The most widely used control is the push button.
- A *push button* is a component that contains a label and that generates an event when it is pressed.
- Push buttons are objects of type **Button**.

Two constructors:-

Button( ) throws HeadlessException
Button(String *str*) throws HeadlessException

The first version creates an empty button. The second creates a button that contains *str* as a label. After a button has been created, you can set its label by calling **setLabel( )**. You can retrieve its label by calling **getLabel( )**.

methods :-

void setLabel(String *str*)
String getLabel( )

Handling Buttons:-

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. An **ActionEvent** object is supplied as the argument to this method.
- It contains both a reference to the button that generated the event and a reference to the *action command string* associated with the button.

- By default, the action command string is the label of the button.
- Usually, either the button reference or the action command string can be used to identify the button.

Example:-

```
// Demonstrate Buttons
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="ButtonDemo" width=250 height=150>
</applet>
*/
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);
        }
}
```

Check Boxes:-

Checkbox control is used to turn an option on(true) or off(false). There is label for each checkbox representing what the checkbox does.The state of a checkbox can be changed by clicking on it.

Class declaration

Following is the declaration for **java.awt.Checkbox** class:
public class Checkbox
  extends Component implements ItemSelectable,Accessible

Class constructors

S.N.	Constructor & Description
1	**Checkbox()** Creates a check box with an empty string for its label.
2	**Checkbox(String label)** Creates a check box with the specified label.
3	**Checkbox(String label, boolean state)** Creates a check box with the specified label and sets the specified state.
4	**Checkbox(String label, boolean state, CheckboxGroup group)** Constructs a Checkbox with the specified label, set to the specified state, and in the specified check box group.
5	**Checkbox(String label, CheckboxGroup group, boolean state)** Creates a check box with the specified label, in the specified check box group, and set to the specified state.

Class methods

S.N.	Method & Description
1	**void addItemListener(ItemListener l)** Adds the specified item listener to receive item events from this check box.
2	**void addNotify()**

	Creates the peer of the Checkbox.
3	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this Checkbox.
4	**CheckboxGroup getCheckboxGroup()** Determines this check box's group.
5	**ItemListener[] getItemListeners()** Returns an array of all the item listeners registered on this checkbox.
6	**String getLabel()** Gets the label of this check box.
7	**<T extends EventListener>T[] getListeners(Class<T> listenerType)** Returns an array of all the objects currently registered as FooListeners upon this Checkbox.
8	**Object[] getSelectedObjects()** Returns an array (length 1) containing the checkbox label or null if the checkbox is not selected.
9	**boolean getState()** Determines whether this check box is in the **on** or **off** state.
10	**protected String paramString()** Returns a string representing the state of this Checkbox.
11	**protected void processEvent(AWTEvent e)** Processes events on this check box.
12	**protected void processItemEvent(ItemEvent e)** Processes item events occurring on this check box by dispatching them to any registered ItemListener objects.
13	**void removeItemListener(ItemListener l)** Removes the specified item listener so that the item listener no longer receives item events from this check box.
14	**void setCheckboxGroup(CheckboxGroup g)** Sets this check box's group to the specified check box group.
15	**void setLabel(String label)** Sets this check box's label to be the string argument.

16	**void setState(boolean state)** Sets the state of this check box to the specified state.

Methods inherited

This class inherits methods from the following classes:

- java.awt.Component
- java.lang.Object

Example:-

```
// Demonstrate check boxes.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="CheckboxDemo" width=250 height=200>
</applet>
*/
public class CheckboxDemo extends Applet implements ItemListener {
	String msg = "";
	Checkbox winXP, winVista, solaris, mac;
	public void init() {
		winXP = new Checkbox("Windows XP", null, true);
		winVista = new Checkbox("Windows Vista");
		solaris = new Checkbox("Solaris");
		mac = new Checkbox("Mac OS");
		add(winXP);
		add(winVista);
		add(solaris);
		add(mac);
		winXP.addItemListener(this);
		winVista.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 Vista: " + winVista.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);
        }
}
```

CheckboxGroup:-

The CheckboxGroup class is used to group the set of checkbox.

Class declaration

Following is the declaration for **java.awt.CheckboxGroup** class:
public class CheckboxGroup
  extends Object
    implements Serializable

Class constructors

S.N.	Constructor & Description
1	**CheckboxGroup() ()** Creates a new instance of CheckboxGroup.

Class methods

S.N.	Method & Description
1	**Checkbox getCurrent()** Deprecated. As of JDK version 1.1, replaced by getSelectedCheckbox().
2	**Checkbox getSelectedCheckbox()** Gets the current choice from this check box group.
3	**void setCurrent(Checkbox box)** Deprecated. As of JDK version 1.1, replaced by setSelectedCheckbox(Checkbox).
4	**void setSelectedCheckbox(Checkbox box)** Sets the currently selected check box in this group to be the specified check box.
5	**String toString()** Returns a string representation of this check box group, including the value of its current selection.

Methods inherited

This class inherits methods from the following classes:

- java.lang.Object

Example:-

```
// Demonstrate check box group.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="CBGroup" width=250 height=200>
</applet>
*/
public class CBGroup extends Applet implements ItemListener {
        String msg = "";
        Checkbox winXP, winVista, solaris, mac;

        public void init() {
                cbg = new CheckboxGroup();
                winXP = new Checkbox("Windows XP", cbg, true);
                winVista = new Checkbox("Windows Vista", cbg, false);
                solaris = new Checkbox("Solaris", cbg, false);
                mac = new Checkbox("Mac OS", cbg, false);
                add(winXP);
                add(winVista);
                add(solaris);
                add(mac);
                winXP.addItemListener(this);
                winVista.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 selection: ";
                msg += cbg.getSelectedCheckbox().getLabel();
                g.drawString(msg, 6, 100);
        }

}
```

Lists:-

The List represents a list of text items.

The list can be configured to that user can choose either one item or multiple items.The List provides a compact,multiple-choice ,scrolling selection list.Unlike the choice object,which shows only the single selected item in the menu,aList object can be constructed to show any number of choices in the visible window.

Itcan also be constructed to allow multiple selections.

List provides these constructors:-

- **List()**
  Creates a new scrolling list. Initially there are no visible lines, and only one item can be selected from the list.

- **List(int numRows)**
  Creates a new scrolling list initialized with the specified number of visible lines. By default, multiple selections are not allowed.Parameter,rows specified by the number of items to show.

- **List(int numRows,boolean multipleMode)**
  Creates a new scrolling list initialized to display the specified number of rows. If the value of multipleMode is true, then the user can select multiple items from the list. If it is false, only one item at a time can be selected.Parameters,numRows specified by the number of items to show and multipleMode,if true, then multiple selections are allowed; otherwise, only one item can be selected at a time .

Lists provides these methods:-

- **add(String item) :**Adds the specified item to the end of scrolling list.
  syntax:
  void add(String item)

- **add(String item, int index) :**Adds the specified item to the the scrolling list at the position indicated by the index
  syntax:
  void add(String item, int index)

- **getSelectedIndex()**:Gets the index of the selected item on the list,
  syntax
  int getSelectedIndex()

- **getSelectedItem() :**Gets the selected item on this scrolling list.
  syntax:
  String getSelectedItem()

- **int[] getSelectedIndexes()** :Gets the selected indexes on the list.It return an array containing the indexes of the currentily selected items.

  syntax:
  int[] getSelectedIndexes()

- **String[] getSelectedItems() :**Gets the selected items on this scrolling list.
  syntax:
  String[] getSelectedItems()

- **void select(int index) :**Selects the item at the specified index in the scrolling list
  syntax:
  void select(int index)

Example:-

```
// 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 Vista");
		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);
	}
}
```

Scroll Bars:-

Scrollbar control represents a scroll bar component in order to enable user to select from range of values.

Class declaration

Following is the declaration for **java.awt.Scrollbar** class:
public class Scrollbar
  extends Component
    implements Adjustable, Accessible

Field

Following are the fields for **java.awt.Image** class:

- **static int HORIZONTAL** --A constant that indicates a horizontal scroll bar.
- **static int VERTICAL** --A constant that indicates a vertical scroll bar.

Class constructors

S.N.	Constructor & Description
1	**Scrollbar()** Constructs a new vertical scroll bar.
2	**Scrollbar(int orientation)** Constructs a new scroll bar with the specified orientation.
3	**Scrollbar(int orientation, int value, int visible, int minimum, int maximum)** Constructs a new scroll bar with the specified orientation, initial value, visible amount, and minimum and maximum values.

Class methods

S.N.	Method & Description
1	**void addAdjustmentListener(AdjustmentListener l)** Adds the specified adjustment listener to receive instances of AdjustmentEvent from this scroll bar.
2	**void addNotify()** Creates the Scrollbar's peer.
3	**int getBlockIncrement()** Gets the block increment of this scroll bar.
4	**int getLineIncrement()** Deprecated. As of JDK version 1.1, replaced by getUnitIncrement().
5	**int getMaximum()**

	Gets the maximum value of this scroll bar.
6	**int getMinimum()** Gets the minimum value of this scroll bar.
7	**int getOrientation()** Returns the orientation of this scroll bar.
8	**int getPageIncrement()** Deprecated. As of JDK version 1.1, replaced by getBlockIncrement().
9	**int getUnitIncrement()** Gets the unit increment for this scrollbar.
10	**int getValue()** Gets the current value of this scroll bar.
11	**Boolean** getValueIsAdjusting() Returns true if the value is in the process of changing as a result of actions being taken by the user.
12	**int getVisible()** Deprecated. As of JDK version 1.1, replaced by getVisibleAmount().
13	**int getVisibleAmount()** Gets the visible amount of this scroll bar.
14	**protected String paramString()** Returns a string representing the state of this Scrollbar.
15	**protected void processAdjustmentEvent(AdjustmentEvent e)** Processes adjustment events occurring on this scrollbar by dispatching them to any registered AdjustmentListener objects.
16	**protected void processEvent(AWTEvent e)** Processes events on this scroll bar.
17	**void removeAdjustmentListener(AdjustmentListener l)** Removes the specified adjustment listener so that it no longer receives instances of AdjustmentEvent from this scroll bar.
18	**void setBlockIncrement(int v)**

	Sets the block increment for this scroll bar.
19	**void setLineIncrement(int v)** Deprecated. As of JDK version 1.1, replaced by setUnitIncrement(int).
20	**void setMaximum(int newMaximum)** Sets the maximum value of this scroll bar.
21	**void setMinimum(int newMinimum)** Sets the minimum value of this scroll bar.
22	**void setOrientation(int orientation)** Sets the orientation for this scroll bar.
23	**void setPageIncrement(int v)** Deprecated. As of JDK version 1.1, replaced by setBlockIncrement().
24	**void setUnitIncrement(int v)** Sets the unit increment for this scroll bar.
25	**void setValue(int newValue)** Sets the value of this scroll bar to the specified value.
26	**void setValueIsAdjusting(boolean b)** Sets the valueIsAdjusting property.
27	**void setValues(int value, int visible, int minimum, int maximum)** Sets the values of four properties for this scroll bar: value, visibleAmount, minimum, and maximum.
28	**void setVisibleAmount(int newAmount)** Sets the visible amount of this scroll bar.
29	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this Scrollbar.
30	**AdjustmentListener[] getAdjustmentListeners()** Returns an array of all the adjustment listeners registered on this scrollbar.
31	**<T extends EventListener>T[] getListeners(Class<T> listenerType)** Returns an array of all the objects currently registered as FooListeners upon this Scrollbar.

Methods inherited

This class inherits methods from the following classes:

- java.awt.Component
- java.lang.Object

Example:-

```
// 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);
    horzSB = new Scrollbar(Scrollbar.HORIZONTAL,0, 1, 0, width);
    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());
}
}
```

TextField:-

- The textField component allows the user to edit single line of text.
- When the user types a key in the text field the event is sent to the TextField.
- The key event may be key pressed, Key released or key typed.
- The key event is passed to the registered KeyListener.
- It is also possible to for an ActionEvent if the ActionEvent is enabled on the textfield then ActionEvent may be fired by pressing the return key.

Class declaration

Following is the declaration for **java.awt.TextField** class:

```
public class TextField
  extends TextComponent
```

Class constructors

S.N.	Constructor & Description
1	**TextField()** Constructs a new text field.
2	**TextField(int columns)** Constructs a new empty text field with the specified number of columns.
3	**TextField(String text)** Constructs a new text field initialized with the specified text.
4	**TextField(String text, int columns)** Constructs a new text field initialized with the specified text to be displayed, and wide enough to hold the specified number of columns.

Class methods

S.N.	Method & Description
1	**void addActionListener(ActionListener l)** Adds the specified action listener to receive action events from this text field.
2	**void addNotify()** Creates the TextField's peer.

3	**boolean echoCharIsSet()** Indicates whether or not this text field has a character set for echoing.
4	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this TextField.
5	**ActionListener[] getActionListeners()** Returns an array of all the action listeners registered on this textfield.
6	**int getColumns()** Gets the number of columns in this text field.
7	**char getEchoChar()** Gets the character that is to be used for echoing.
8	**<T extends EventListener> T[] getListeners(Class<T> listenerType)** Returns an array of all the objects currently registered as FooListeners upon this TextField.
9	**Dimension getMinimumSize()** Gets the minumum dimensions for this text field.
10	**Dimension getMinimumSize(int columns) Gets the minumum dimensions for a text field with the specified number of columns.**
11	**Dimension getPreferredSize()** Gets the preferred size of this text field.
12	**Dimension getPreferredSize(int columns)** Gets the preferred size of this text field with the specified number of columns.
13	**Dimension minimumSize()** Deprecated. As of JDK version 1.1, replaced by getMinimumSize().
14	**Dimension minimumSize(int columns)** Deprecated. As of JDK version 1.1, replaced by getMinimumSize(int).
15	**protected String paramString()** Returns a string representing the state of this TextField.
16	**Dimension preferredSize()** Deprecated. As of JDK version 1.1, replaced by getPreferredSize().
17	**Dimension preferredSize(int columns)** Deprecated. As of JDK version 1.1, replaced by getPreferredSize(int).

18	**protected void processActionEvent(ActionEvent e)** Processes action events occurring on this text field by dispatching them to any registered ActionListener objects.
19	**protected void processEvent(AWTEvent e)** Processes events on this text field.
20	**void removeActionListener(ActionListener l)** Removes the specified action listener so that it no longer receives action events from this text field.
21	**void setColumns(int columns)** Sets the number of columns in this text field.
22	**void setEchoChar(char c)** Sets the echo character for this text field.
23	**void setEchoCharacter(char c)** Deprecated. As of JDK version 1.1, replaced by setEchoChar(char).
24	**void setText(String t)** Sets the text that is presented by this text component to be the specified text.

Methods inherited

This class inherits methods from the following classes:

- java.awt.TextComponent
- java.awt.Component
- java.lang.Object

Example:-

```
// 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);
        }
}
```

TextArea:-

- The TextArea control in AWT provide us multiline editor area.
- The user can type here as much as he wants.
- When the text in the text area become larger than the viewable area the scroll bar is automatically appears which help us to scroll the text up & down and right & left.

Class declaration

Following is the declaration for **java.awt.TextArea** class:

```
public class TextArea
  extends TextComponent
```

Field

Following are the fields for **java.awt.TextArea** class:

- **static int SCROLLBARS_BOTH** -- Create and display both vertical and horizontal scrollbars.
- **static int SCROLLBARS_HORIZONTAL_ONLY** -- Create and display horizontal scrollbar only.
- **static int SCROLLBARS_NONE** -- Do not create or display any scrollbars for the text area.
- **static int SCROLLBARS_VERTICAL_ONLY** -- Create and display vertical scrollbar only.

Class constructors

S.N.	Constructor & Description
1	**TextArea()**

	Constructs a new text area with the empty string as text.
2	**TextArea(int rows, int columns)** Constructs a new text area with the specified number of rows and columns and the empty string as text.
3	**TextArea(String text)** Constructs a new text area with the specified text.
4	**TextArea(String text, int rows, int columns)** Constructs a new text area with the specified text, and with the specified number of rows and columns.
5	**TextArea(String text, int rows, int columns, int scrollbars)** Constructs a new text area with the specified text, and with the rows, columns, and scroll bar visibility as specified.

Class methods

S.N.	Method & Description
1	**void addNotify()** Creates the TextArea's peer.
2	**void append(String str)** Appends the given text to the text area's current text.
3	**void appendText(String str)** Deprecated. As of JDK version 1.1, replaced by append(String).
4	**AccessibleContext getAccessibleContext()** Returns the AccessibleContext associated with this TextArea.
5	**int getColumns()** Returns the number of columns in this text area.
6	**Dimension getMinimumSize()** Determines the minimum size of this text area.
7	**Dimension getMinimumSize(int rows, int columns)** Determines the minimum size of a text area with the specified number of rows and columns.
8	**Dimension getPreferredSize()** Determines the preferred size of this text area.

9	**Dimension getPreferredSize(int rows, int columns)** Determines the preferred size of a text area with the specified number of rows and columns.
10	**int getRows()** Returns the number of rows in the text area.
11	**int getScrollbarVisibility()** Returns an enumerated value that indicates which scroll bars the text area uses.
12	**void insert(String str, int pos)** Inserts the specified text at the specified position in this text area.
13	**void insertText(String str, int pos)** Deprecated. As of JDK version 1.1, replaced by insert(String, int).
14	**Dimension minimumSize()** Deprecated. As of JDK version 1.1, replaced by getMinimumSize().
15	**Dimension minimumSize(int rows, int columns)** Deprecated. As of JDK version 1.1, replaced by getMinimumSize(int, int).
16	**protected String paramString()** Returns a string representing the state of this TextArea.
17	**Dimension preferredSize()** Deprecated. As of JDK version 1.1, replaced by getPreferredSize().
18	**Dimension preferredSize(int rows, int columns)** Deprecated. As of JDK version 1.1, replaced by getPreferredSize(int, int).
19	**void replaceRange(String str, int start, int end)** Replaces text between the indicated start and end positions with the specified replacement text.
20	**void replaceText(String str, int start, int end)** Deprecated. As of JDK version 1.1, replaced by replaceRange(String, int, int).
21	**void setColumns(int columns)** Sets the number of columns for this text area.
22	**void setRows(int rows)** Sets the number of rows for this text area.

Methods inherited

This class inherits methods from the following classes:

- java.awt.TextComponent
- java.awt.Component
- java.lang.Object

Example:-

```
// 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 SE 6 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);}}
```

Assignment:-

Q-1.Explain Fundamental of Window.
Q-2. Explain Frame Window in AWT.
Q-3.Explain Graphics.
Q-4.Explain color.
Q-5.Explain Font Metrics.
Q-6. Explain Controls.

Practical Assignment:-

Q-1 wrt a java code create Registration form.
Q-2 wrt a java code create poligone shape.
Q-3 wrt a java code create smile face shap.
Q-4wrt a java code create star shap.