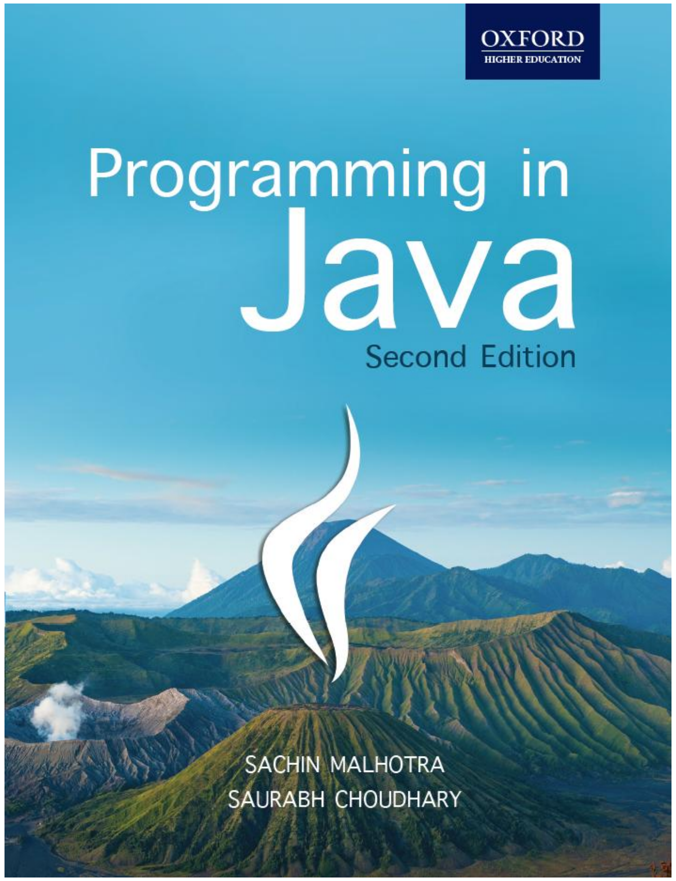
OXFORD
HIGHER EDUCATION



Programming in Java, 2e

Sachin Malhotra

Saurabh Choudhary

Chapter 14

Abstract Window Toolkit

Objectives

- To use the set of GUI components
- To use event-handling model for different components
- To study layout managers, for flexible window layouts that don't depend on a particular window size or screen resolution

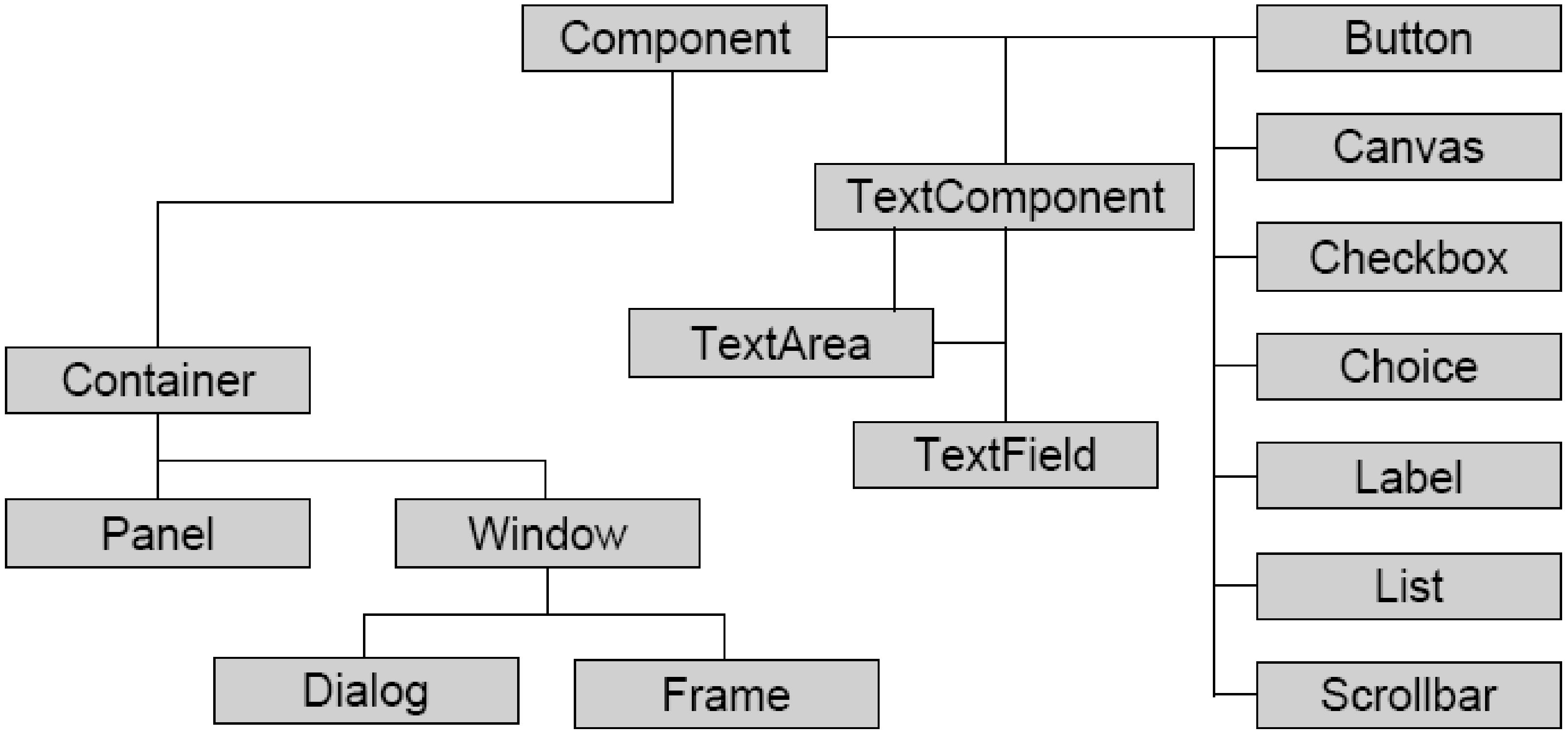
Introduction

- The Java Foundation Classes (JFC) provide two frameworks for building GUI-based application and interestingly both rely on the same event handling model:
  - AWT
  - Swing
- AWT relies on the underlying operating system on a specific platform to represent its GUI components (i.e components in AWT are called Heavyweight),
- Swing implements a new set of lightweight GUI components that are written in Java and has a pluggable look and feel.
- These lightweight components are not dependent on the underlying window system.

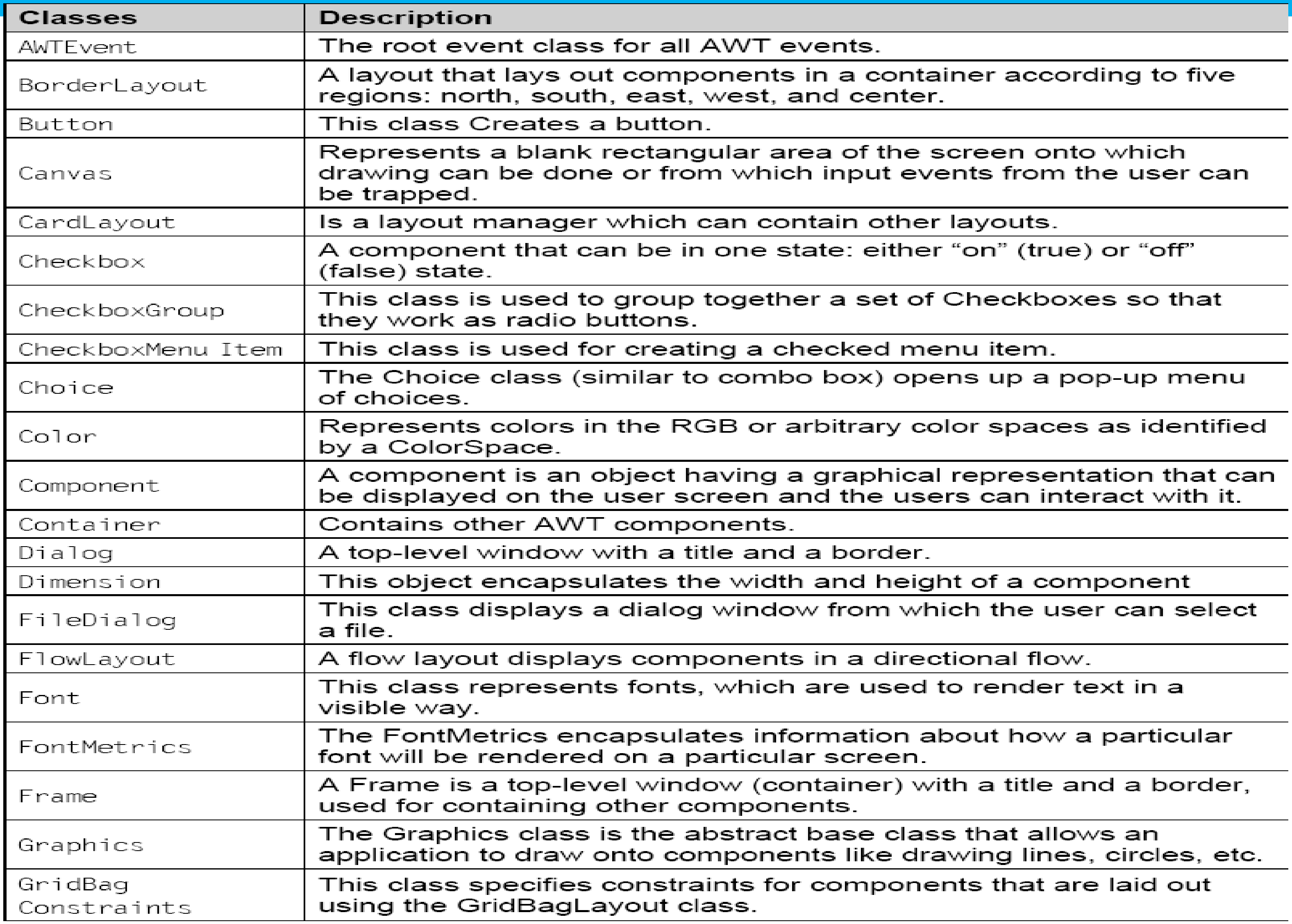
Components and Containers

- A graphical user interface is developed with the help of graphical elements like
  - buttons, scrollbars, lists, textfields, etc.
- These elements are called components.
- In AWT, these components are instances of the respective *Component* classes.
- Components cannot exist alone; they are found within containers.
- Actually, containers are themselves components, thus they can be placed inside other containers.
- In AWT, all containers are objects of class *Container* or one of its subtypes.

Hierarchy of classes in AWT



Few classes in AWT



Classes	Description
AWTEvent	The root event class for all AWT events.
BorderLayout	A layout that lays out components in a container according to five regions: north, south, east, west, and center.
Button	This class Creates a button.
Canvas	Represents a blank rectangular area of the screen onto which drawing can be done or from which input events from the user can be trapped.
CardLayout	Is a layout manager which can contain other layouts.
Checkbox	A component that can be in one state: either “on” (true) or “off” (false) state.
CheckboxGroup	This class is used to group together a set of Checkboxes so that they work as radio buttons.
CheckboxMenu Item	This class is used for creating a checked menu item.
Choice	The Choice class (similar to combo box) opens up a pop-up menu of choices.
Color	Represents colors in the RGB or arbitrary color spaces as identified by a ColorSpace.
Component	A component is an object having a graphical representation that can be displayed on the user screen and the users can interact with it.
Container	Contains other AWT components.
Dialog	A top-level window with a title and a border.
Dimension	This object encapsulates the width and height of a component
FileDialog	This class displays a dialog window from which the user can select a file.
FlowLayout	A flow layout displays components in a directional flow.
Font	This class represents fonts, which are used to render text in a visible way.
FontMetrics	The FontMetrics encapsulates information about how a particular font will be rendered on a particular screen.
Frame	A Frame is a top-level window (container) with a title and a border, used for containing other components.
Graphics	The Graphics class is the abstract base class that allows an application to draw onto components like drawing lines, circles, etc.
GridBag Constraints	This class specifies constraints for components that are laid out using the GridBagLayout class.

Few classes in AWT (contd.)

GridBagLayout	This layout is a flexible layout manager that aligns components vertically, horizontally or along their baseline without requiring that the components be of the same size.
GridLayout	This layout lays out components in a rectangular grid.
Image	This abstract class represents graphical images.
Insets	Specifies how much space must be left at all edges of a container.
Label	Used for placing text in a container.
List	A scrollable list of String items.
MediaTracker	Used for tracking the status of a number of media objects
Menu	A pull-down menu deployed from a menu bar.
MenuBar	Menus are added on a MenuBar attached on to a Frame.
MenuItem	All items in a menu are MenuItems.
MenuShortcut	This class represents a keyboard shortcut for a MenuItem.
MouseInfo	provides methods for getting information about the mouse location, etc.
Panel	It is a container class.
Point	Represents a location in (x,y) coordinate space.
PopupMenu	A menu which can be dynamically popped up at any specified position within a component.
Scrollbar	The Scrollbar class embodies a scroll bar either vertical or horizontal
ScrollPane	A container class which implements automatic horizontal and/or vertical scrolling for a single child component.
TextArea	An object that allow user to enter/edit multi-line input.
TextComponent	Super class of TextField and TextArea.
TextField	It is a text component that allows the user to enter/edit a single line of text.
Window	It is the parent of Dialog and Frame and represents a top-level window with no borders and no menubar.

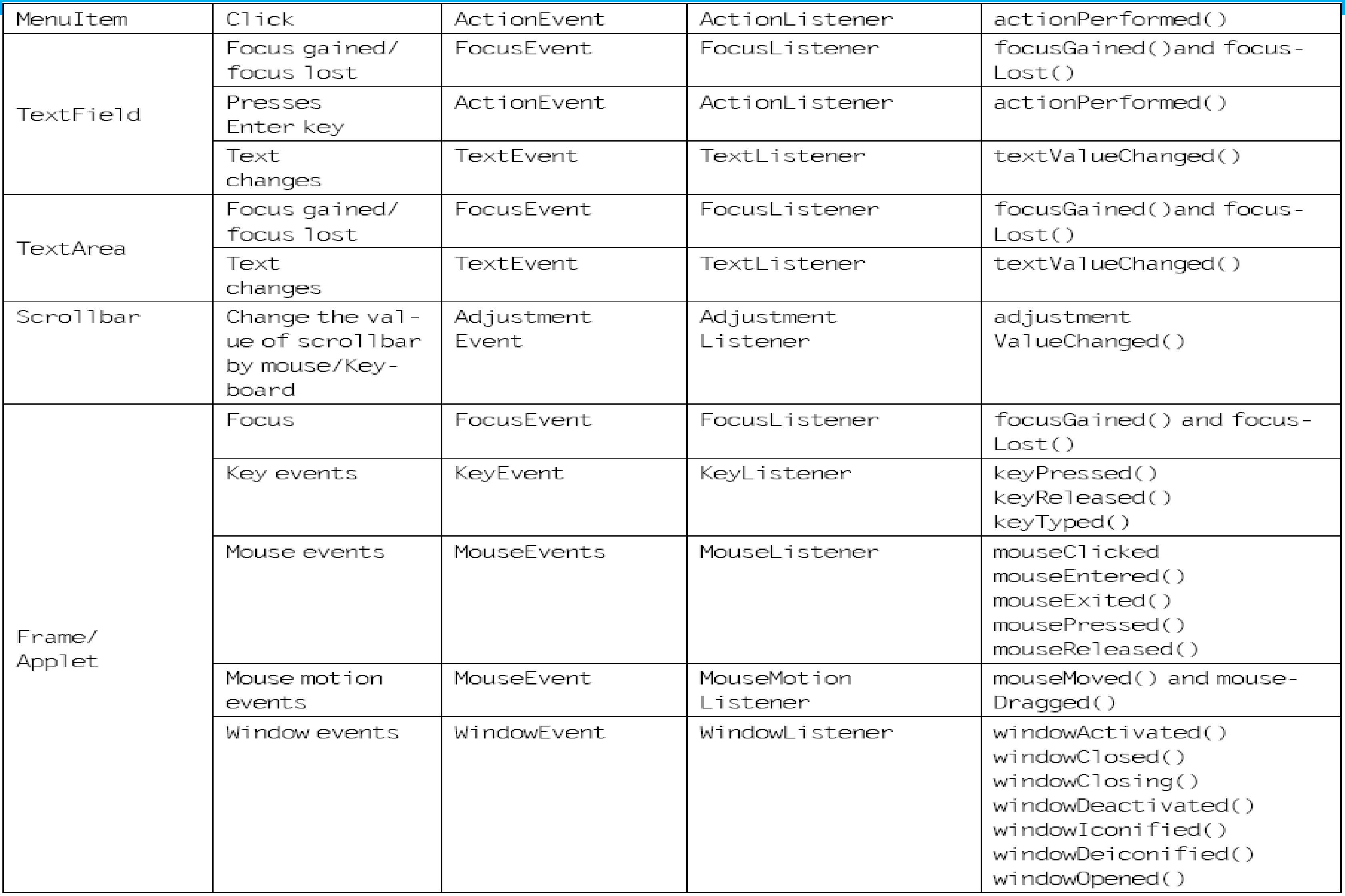
Component

- subclass of the Object class & super class of classes such as
  - Button,
  - Label,
  - CheckBox,
  - RadioButton, Choice, Canvas, etc
- Componenets are added to a window using the add() method
  - Component add(Component ComObj)
  - ComObj is the object of the Component, which is to be added
  - This method returns the reference to the ComObj.
- If you wish to remove a Component from a window, use remove() method
  - void remove(Component ComObj)2

Components as Event Generator

Components	Event	Events Type	Event Listeners	Method Name
Button	Click	ActionEvent	ActionListener	actionPerformed()
	Focus gained/ focus lost	FocusEvent	FocusListener	focusGained()and focus-Lost()
Checkbox	Selection/ Deselection	ItemEvent	ItemListener	itemStateChanged()
Choice	Selection/ Deselection	ItemEvent	ItemListener	itemStateChanged()
List	Selection/ Deselection	ItemEvent	ItemListener	itemStateChanged()
	Double clicking on an item	ActionEvent	ActionListener	actionPerformed()

Components as Event Generator



MenuItem	Click	ActionEvent	ActionListener	actionPerformed()
TextField	Focus gained/ focus lost	FocusEvent	FocusListener	focusGained()and focus-Lost()
	Presses Enter key	ActionEvent	ActionListener	actionPerformed()
	Text changes	TextEvent	TextListener	textValueChanged()
TextArea	Focus gained/ focus lost	FocusEvent	FocusListener	focusGained()and focus-Lost()
	Text changes	TextEvent	TextListener	textValueChanged()
Scrollbar	Change the val-ue of scrollbar by mouse/Key-board	Adjustment Event	Adjustment Listener	adjustment ValueChanged()
Frame/ Applet	Focus	FocusEvent	FocusListener	focusGained() and focus-Lost()

	Key events	KeyEvent	KeyListener	keyPressed()
				keyReleased()
				keyTyped()

	Mouse events	MouseEvents	MouseListener	mouseClicked
				mouseEntered()
				mouseExited()
				mousePressed()
				mouseReleased()

	Mouse motion events	MouseEvent	MouseMotion Listener	mouseMoved() and mouse-Dragged()

	Window events	WindowEvent	WindowListener	windowActivated()
				windowClosed()
				windowClosing()
				windowDeactivated()
				windowIconified()
				windowDeiconified()
				windowOpened()

Button

- The Button class belongs to **java.awt** package
  - public class **Button** extends Component implements Accessible
- This class creates a button which when pushed or pressed generates an event.
- The two constructors belonging to this Button class are:
  - Button() throws HeadlessException
  - Button(String str)throws HeadLessException;
- To create a button
  - Button buttonName = new Button(Str);
  - ‘buttonname’ is the name you give to the button object and ‘Str’ is the text you want to appear on the button.
- Once the object for Button is created, it needs to be added to the applet or any other container using
  - add(buttonname);
  - void setLabel(String str) for changing the button’s label
  - String getLabel() for getting the Buttons label’s text

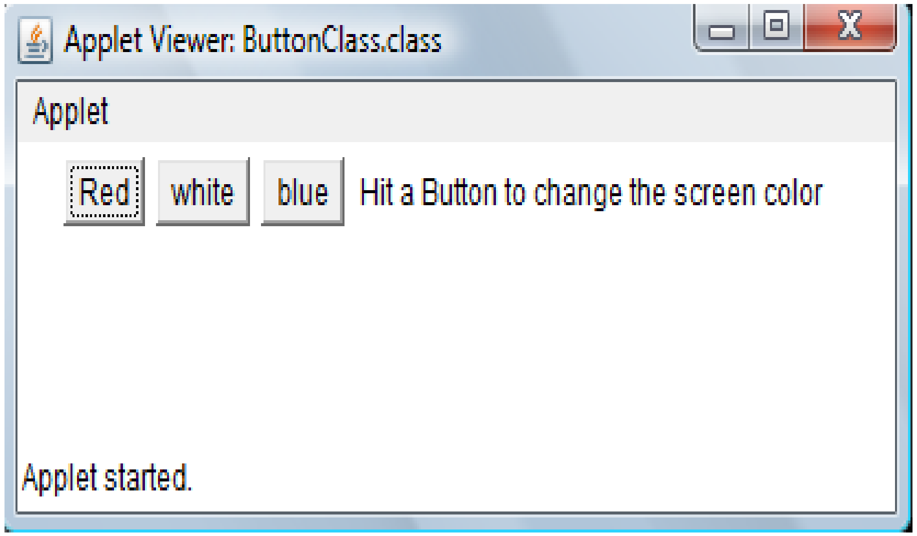
Button Example

```
/*<applet code=ButtonClass.class width=400
    height=150></applet>*/
    import java.applet.*;
    import java.awt.*;
    import java.awt.event.*;
    public class ButtonClass extends Applet implements
    ActionListener{
    Button red, white, blue;
    Label hit;
    public void init(){
    red = new Button(“Red”);
    white = new Button(“white”);
    blue = new Button(“blue”);
    hit = new Label(“Hit a Button to change the screen color”);
    add(red);     add(white);      add(blue);       add(hit);
```

Button Example (contd.)

```
red.addActionListener(this);
white.addActionListener(this);
blue.addActionListener(this);}
public void actionPerformed(ActionEvent ae){
String str = ae.getActionCommand();
if (str.equals(“Red”)) {
setBackground(Color.red);}
else if (str.equals(“white”)) {
setBackground(Color.white);}
else if (str.equals(“blue”)){
setBackground(Color.blue);}
repaint();}}
```

The Output

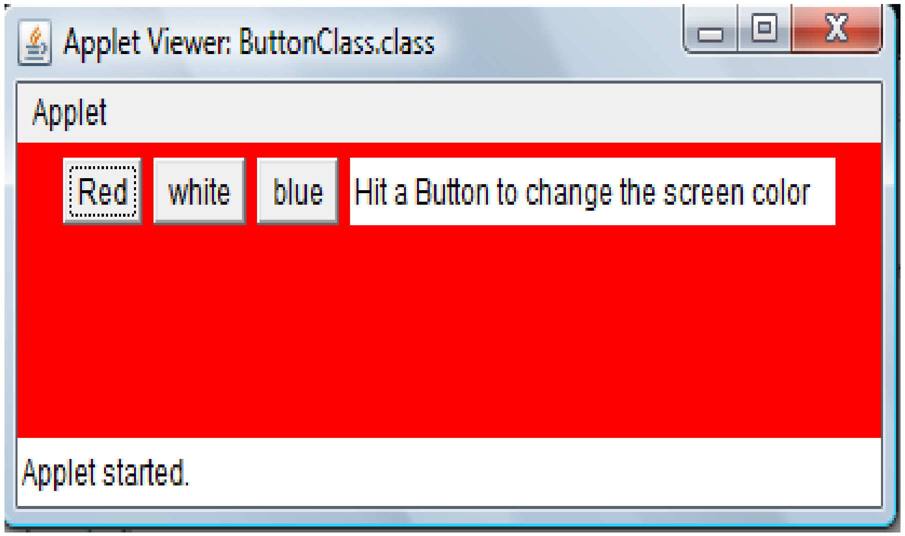


Applet Viewer: ButtonClass.class

Applet

Red | white | blue | Hit a Button to change the screen color

Applet started.

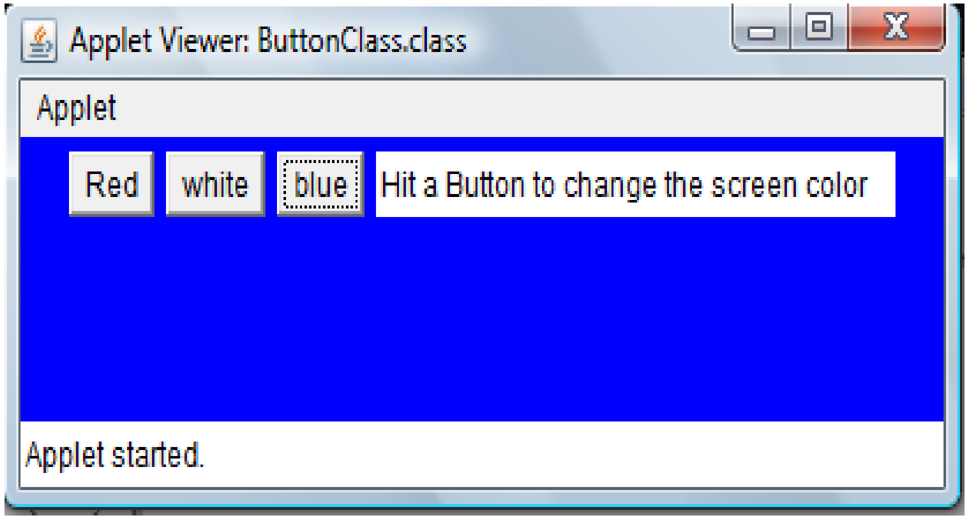


Applet Viewer: ButtonClass.class

Applet

Red | white | blue | Hit a Button to change the screen color

Applet started.



Applet Viewer: ButtonClass.class

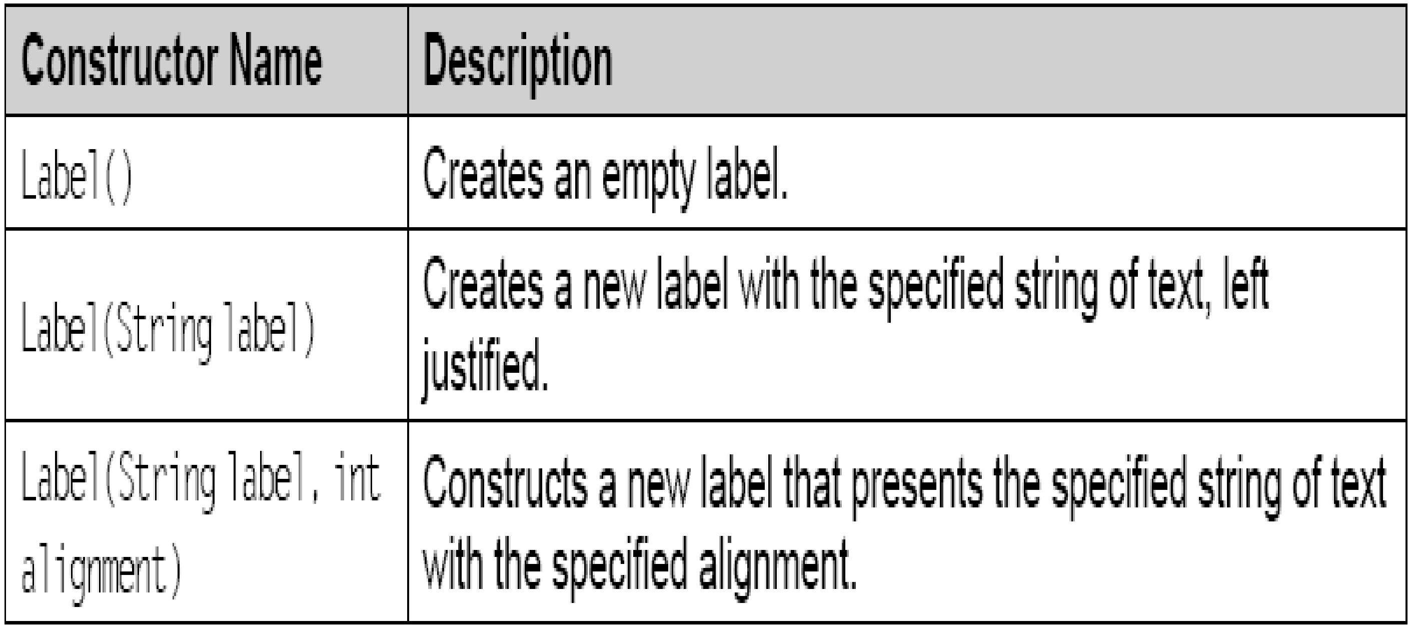
Applet

Red | white | blue | Hit a Button to change the screen color

Applet started.

Label

- Labels consist of a text string for display only and they never call an action method.
- A Label can be justified LEFT, RIGHT, or CENTERED.
  - new Label(“This label is for demonstration.”, Label.RIGHT);

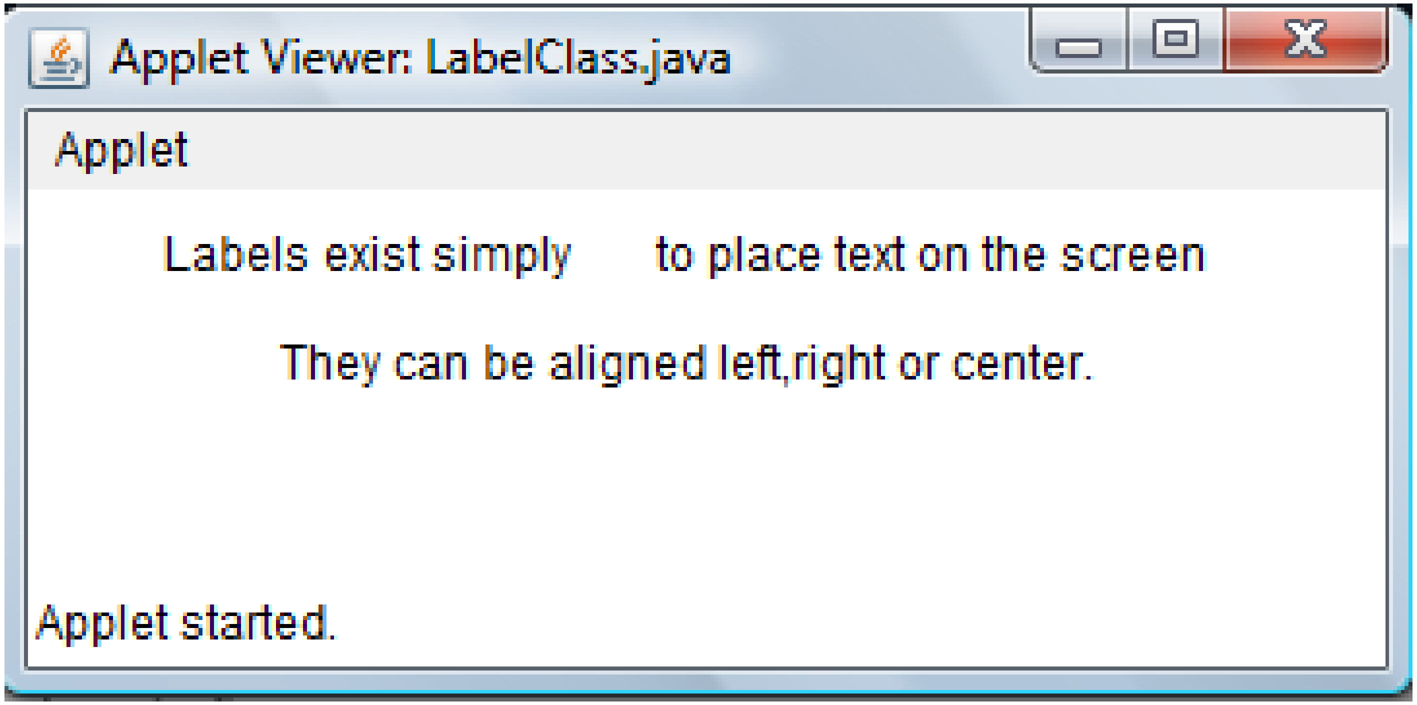


Constructor Name	Description
Label()	Creates an empty label.
Label(String label)	Creates a new label with the specified string of text, left justified.
Label(String label, int alignment)	Constructs a new label that presents the specified string of text with the specified alignment.

Label Example

```
/*<applet code=”LabelClass” width=350
height=100></applet>*/
import java.applet.*;
import java.awt.*;
public class LabelClass extends Applet {
public void init(){
Label firstLabel = new Label(“Labels exist simply “);
add(firstLabel);
Label secLabel = new Label(“to place text on the screen”);
add(secLabel);
Label thirdLabel = new Label(“They can be aligned left, right
or center.”);
add(thirdLabel);}}
```

The Output



```
Labels exist simply     to place text on the screen
They can be aligned left,right or center.
```

Applet started.

Checkbox

- Checkboxes are used as on-off or yes-no switches
- if you click on an unchecked checkbox, it will get checked and vice versa.
- Constructors of Checkbox
  - Checkbox()
  - Checkbox(String str)
  - Checkbox(String str, boolean on)
  - Checkbox(String str, CheckBoxGroup cbg, boolean on)

Methods of Checkbox class

Method Names	Description
void addItemListener (ItemListener il)	Adds the specified item listener to receive item events from this checkbox.
CheckboxGroup getCheckboxGroup()	Returns the associated checkbox's group.
ItemListener[] getItemListeners()	Returns an array of all the item listeners registered with this checkbox.
String getLabel()	Returns String in the form of Checkbox text.
Object[] getSelectedObjects()	Returns an array (length 1) containing the checkbox label or null if the checkbox is not selected.
boolean getState()	Returns Boolean in the form of true or false, depending on whether the Checkbox is selected or unselected.
protected String paramString()	Returns a string representing the state of the checkbox.
protected void processEvent(AWTEvent awte)	Processes events on this checkbox.
protected void processItemEvent(ItemEvent ie)	Processes item events occurring on this checkbox by dispatching them to any registered ItemListener objects.
void removeItemListener (ItemListener l)	Removes the association between item listener and checkbox
void setCheckboxGroup (CheckboxGroup ckbg)	Sets this checkbox's group to the specified checkbox group.
void setLabel(String label)	We can change the Checkbox label and set it to the argument, 'label'.
void setState(boolean state)	changes the checkbox's state to true (for selected) or false(for unselected).

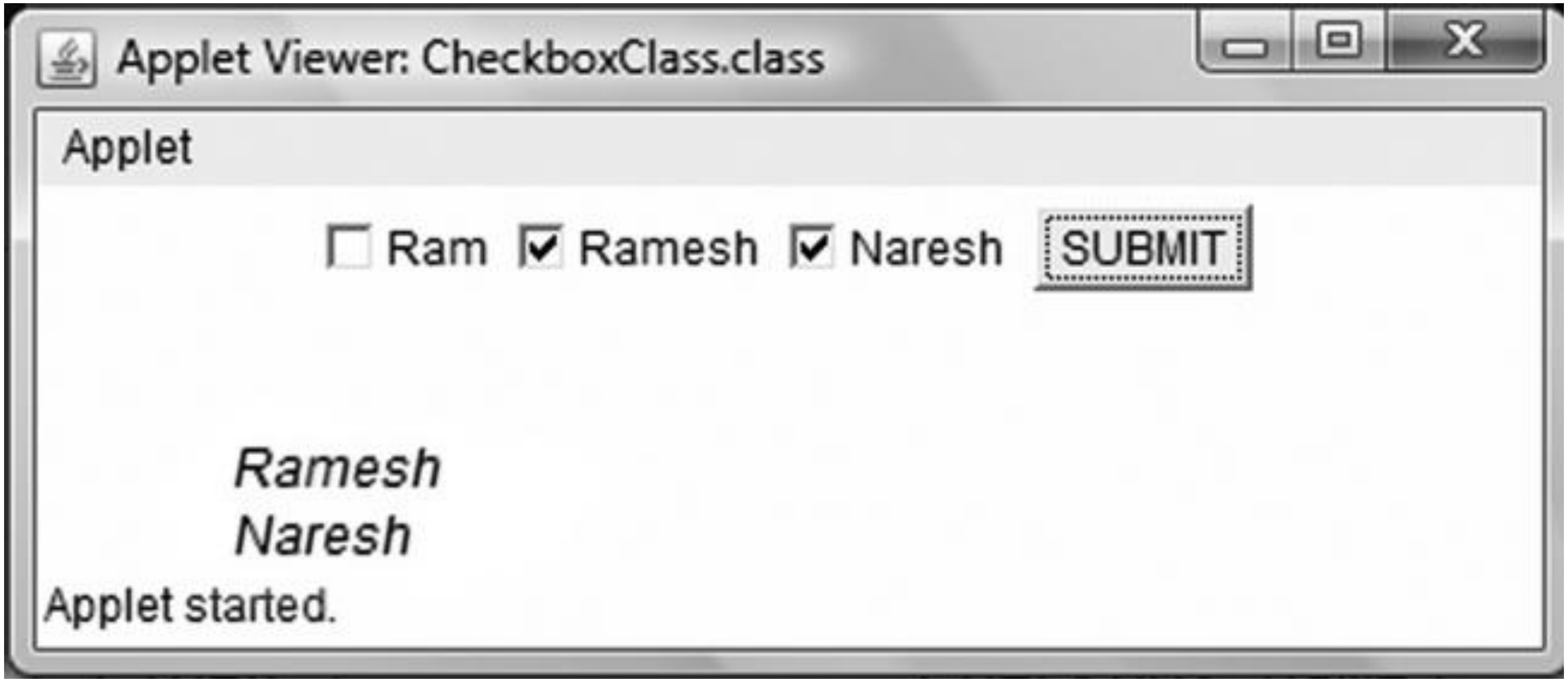
Checkbox Example

```
/*<applet code=CheckboxClass.class width=400
height=100></applet>*/
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class CheckboxClass zextends Applet implements
ActionListener {
Button submit;
Checkbox name1;
Checkbox name2;
Checkbox name3;
public void init(){
name1 = new Checkbox (“Ram”,null,false);
name2 = new Checkbox (“Ramesh”,null,false);
name3 = new Checkbox (“Naresh”,null,false);
```

Checkbox Example

```
Font f = new Font (“Arial”,Font.ITALIC,14);
submit = new Button(“SUBMIT”);
add(name1); add(name2);     add(name3);     add(submit);
submit.addActionListener(this);         }
public void actionPerformed(ActionEvent ae) {
String str = ae.getActionCommand();
if (str.equals(“SUBMIT”))       repaint();}
public void paint (Graphics g) {
g.setFont(f);
g.setColor(Color.blue);
if (name1.getState())
g.drawString(“Ram”,50,60);
if (name2.getState())
g.drawString(“Ramesh”,50,80);
if (name3.getState())
g.drawString(“Naresh”,50,100);         }}
```

The Output



```
Ramesh
Naresh
Applet started.
```

Radio Buttons

- are special kind of checkboxes where only one box can be selected at a time.
- The CheckboxGroup class is used to group together a set of checkboxes
  - CheckboxGroup fruits = new CheckboxGroup();
- After creating checkbox group, the individual checkboxes are added to that group.
  - add(new Checkbox(“mango”, fruits, false));
  - add(new Checkbox(“papaya”, fruits, false));
  - add(new Checkbox(“guava”, fruits, false));
  - add(new Checkbox(“apple”, true, yes));

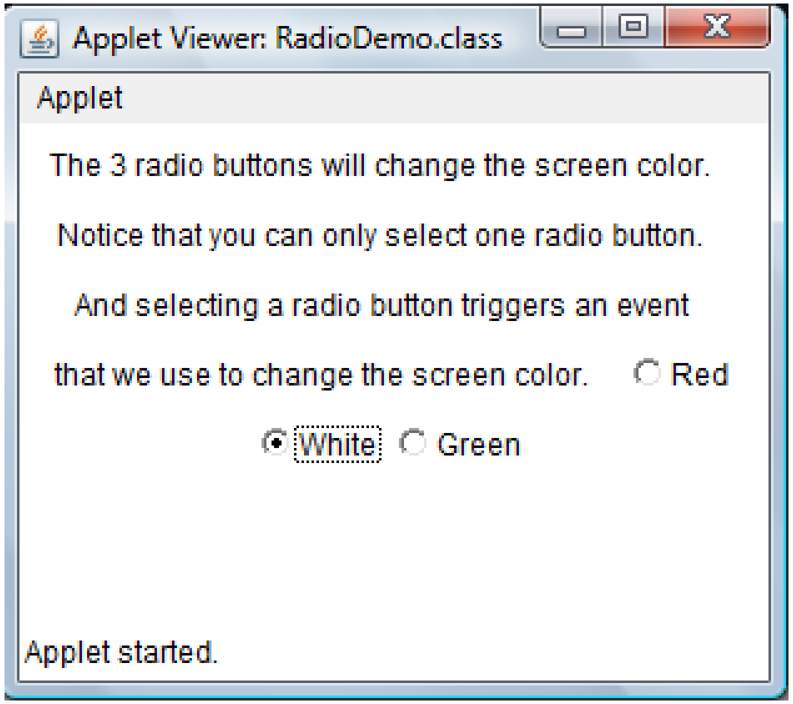
Radio Button Example

```
/*<applet code=”RadioDemo.class” width=300
height=200></applet>*/
import java.applet.*; import java.awt.*;
import java.awt.event.*;
public class RadioDemo extends Applet implements ItemListener{
Checkbox red, white, green;  CheckboxGroup cbg;
public void init(){
add(new Label(“The 4 radio buttons will change the screen color.”));
cbg = new CheckboxGroup();
red = new Checkbox(“Red”,cbg,false);
white = new Checkbox(“White”,cbg,false);
green = new Checkbox(“Green”,cbg,false);
add(new Label(“Notice that you can only select one radio
button.”));
add(new Label(“And selecting a radio button triggers an event”));
```

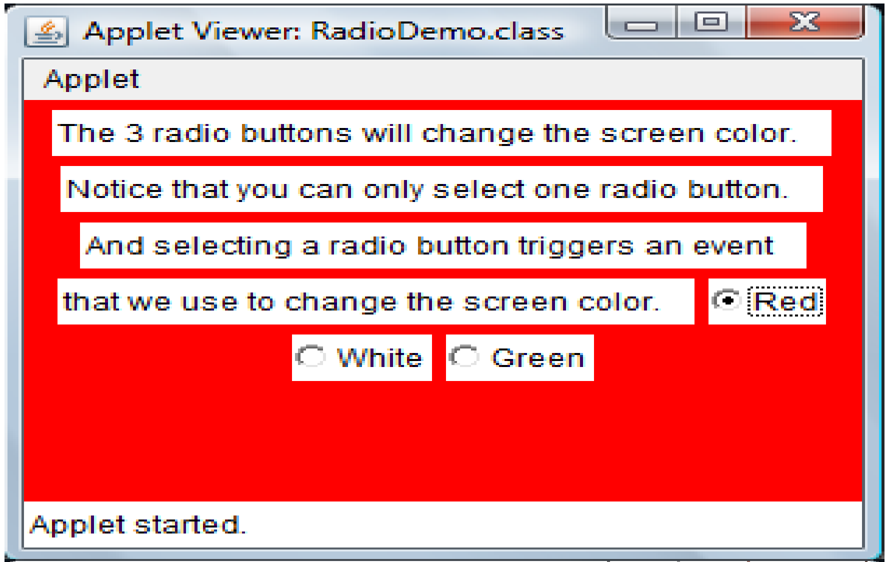
Radio Button Example

```
add(new Label("that we use to change the screen color."));
add(red);
add(white);  add(green);
red.addItemListener(this);
white.addItemListener(this);
green.addItemListener(this); }
public void itemStateChanged(ItemEvent ie){
String str = (String) ie.getItem();
if (str.equals("Red")) {
        setBackground(Color.red);}
else if (str.equals("White")) {
        setBackground(Color.white);}
else if (str.equals("Green")){
        setBackground(Color.green);}
repaint();}}
```

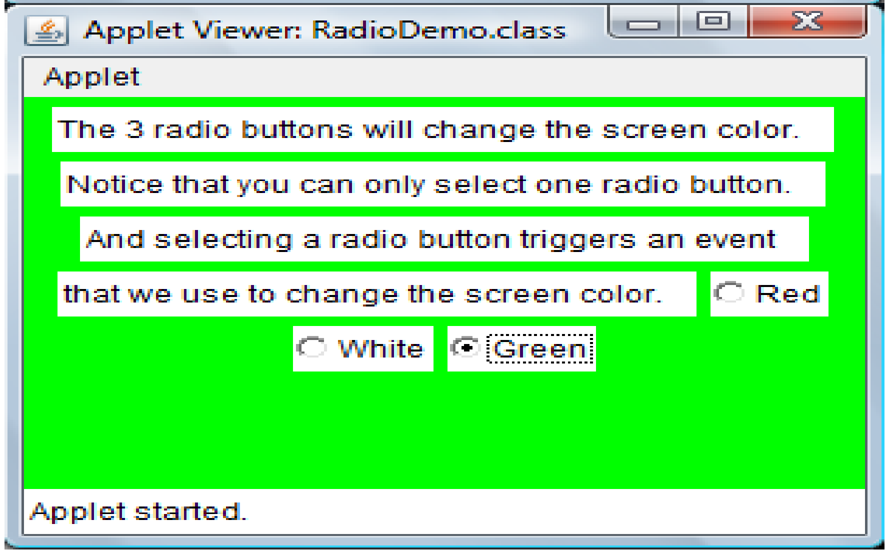
The Output



```
Applet Viewer: RadioDemo.class
Applet
The 3 radio buttons will change the screen color.
Notice that you can only select one radio button.
And selecting a radio button triggers an event
that we use to change the screen color.   ○ Red
◉ White  ○ Green
Applet started.
```



```
Applet Viewer: RadioDemo.class
Applet
The 3 radio buttons will change the screen color.
Notice that you can only select one radio button.
And selecting a radio button triggers an event
that we use to change the screen color.   ◉ Red
○ White  ○ Green
Applet started.
```



```
Applet Viewer: RadioDemo.class
Applet
The 3 radio buttons will change the screen color.
Notice that you can only select one radio button.
And selecting a radio button triggers an event
that we use to change the screen color.   ○ Red
○ White  ◉ Green
Applet started.
```

List

- provides a multiple choice, scrolling list of values that may be selected alone or together
- List class has the following constructors:
  - List()
  - List (int no_of_rows)
  - List(int no_of_rows, boolean multi_select)
- To create a List and add items to it
  - List anyList = new List(10, true);
  - anyList.add(“apple”);
  - anyList.add(“mango”);
  - anyList.add(“guava”);
  - thelist.add(“Coffee”, 0); // adds at the first position

Few Methods of List class

Method Name	Particulars
`void add(String item)`	Adds the specified item to the end of scrolling list.
`void add(String item, int index)`	Adds the specified item to the scrolling list at the position indicated by the index.
`void addActionListener ActionListener l)`	Adds the specified action listener to receive action events from this list.
`void addItemListener(ItemListener il)`	Adds the specified item listener to receive item events from this list.
`void deselect(int index)`	De-selects the item at the specified index.
`AccessibleContext getAccessibleContext()`	Gets the AccessibleContext associated with this List.
`ActionListener[] getActionListeners()`	Returns an array of all the action listeners registered on this list.
`String getItem(int index)`	Gets the item associated with the specified index.
`int getItemCount()`	Gets the number of items in the list.
`ItemListener[] getItemListeners()`	Returns an array of all the item listeners registered on this list.
`String[] getItems()`	Gets the items in the list.

Example

```
/*<applet code=ShopList.class width=600 height=600></applet>*/
import java.applet.*;import java.awt.*;import java.awt.event.*;
public class ShopList extends Applet implements ActionListener {
List original; List copy;
public void init(){
original= new List(8,false);
copy= new List(10,false);
populateList();
add(original);
Button b1 = new Button(“>>>>”);
add(b1);        add(copy);
Button b2 = new Button(“Clear”);
add(b2);
add(new Label(“Select an item from the list on the left and hit >>>> to
place it in the other list”));
b1.addActionListener(this);     b2.addActionListener(this);}
```

Example

```
public void populateList(){
        original.add(“Grocery”);
        original.add(“Fruits”);
        original.add (“Ice-cream”);
        original.add(“Vegetables”);
        original.add(“Garments”);
        original.add(“Baby Food”);}
public void actionPerformed(ActionEvent ae){
        String str = ae.getActionCommand();
        if (str.equals(“>>>>”) && original.getSelectedIndex ()>=0) {
                copy.add(original.getSelectedItem());
                original.remove(original.getSelectedIndex());}
        else if(str.equals(“Clear”))  {
                original.removeAll();
                copy.removeAll();
                populateList();                }
                repaint();                     }}
```

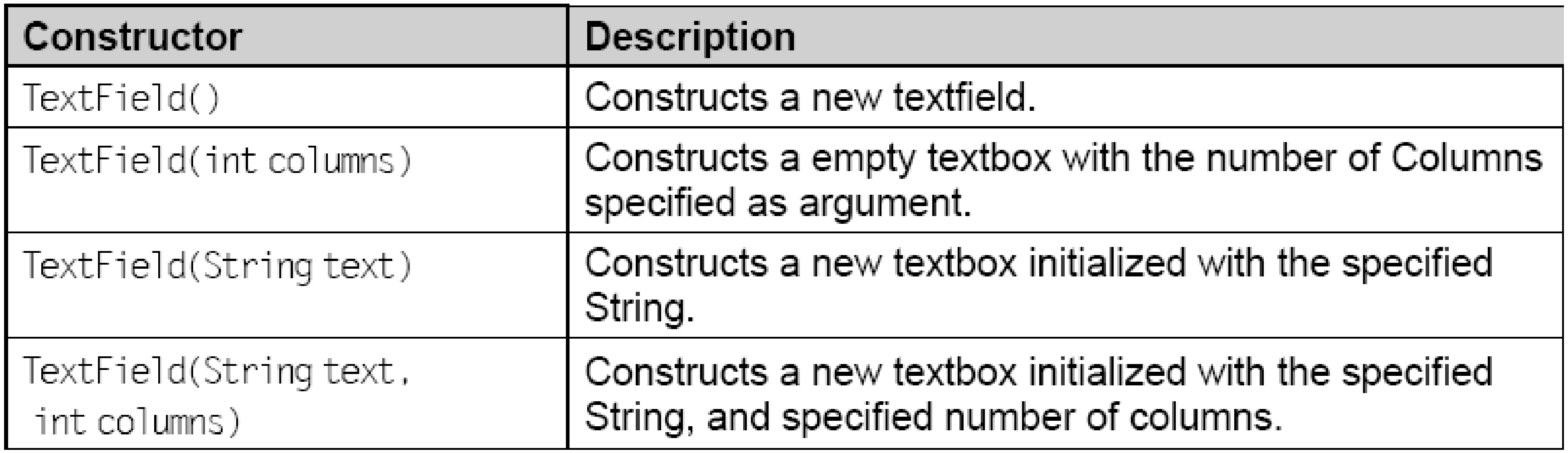
Choice box

- provides a pop-up menu of text string choices
- current choice is displayed on top.
  - Choice c = new Choice();
- the add method enables you to add new entries.
  - c.add(“Red”);
  - c.add(“Green”);
- The currently selected item can be changed by using select() method.
- The selection can be made based on name or index. For e.g.
  - c.select(“Red”);
  - c.select(0);
  - *getSelectedIndex()* - return the position of the selected item
  - *getSelectedItem()* **-** returns the name of the selected item
- The Listener for handling Choice change events is ItemListener.

TextField and TextArea

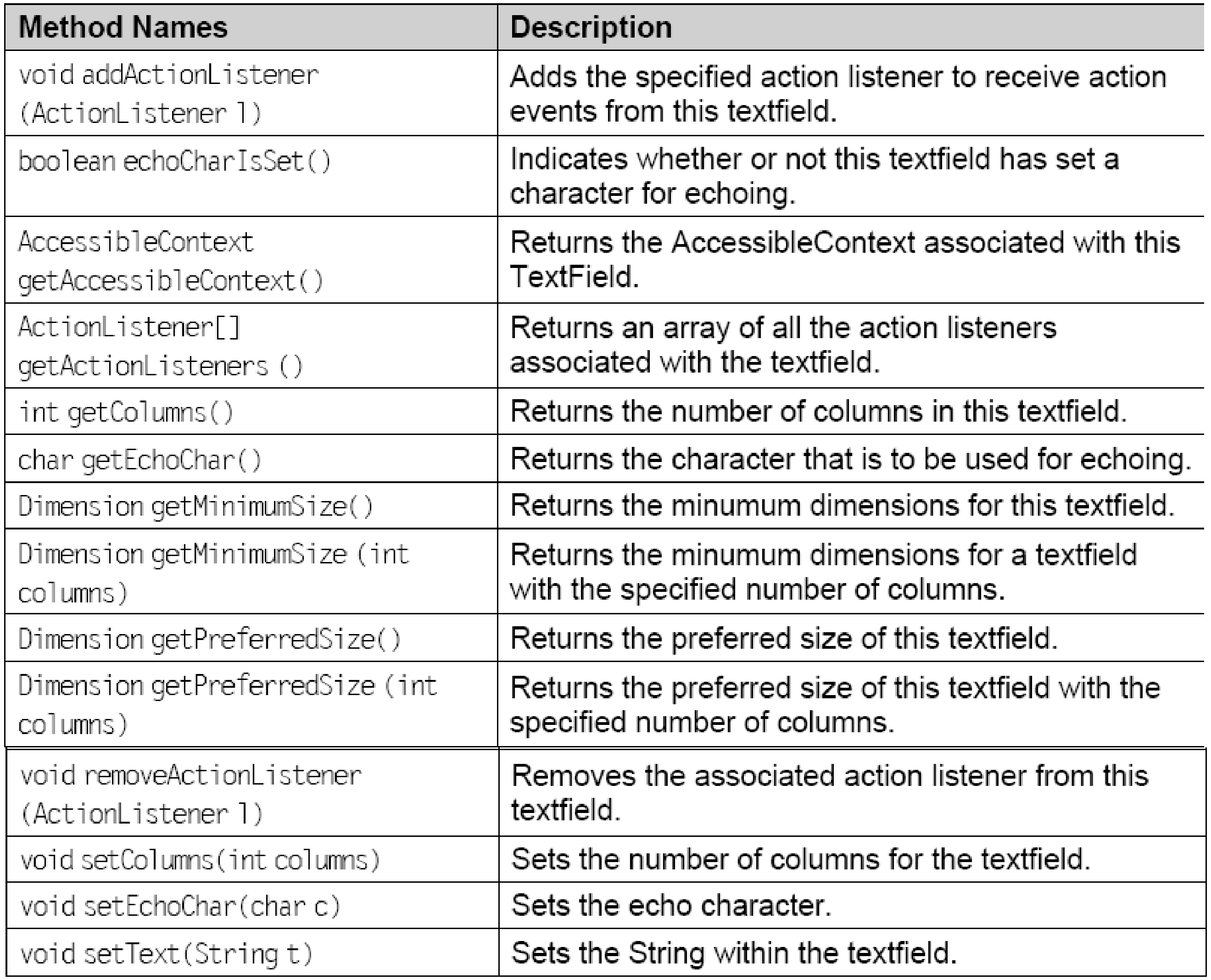
- The TextField class handles single line of text.
- The TextArea is used for handling multiple lines of text.

Table 14.7 Constructor of TextField



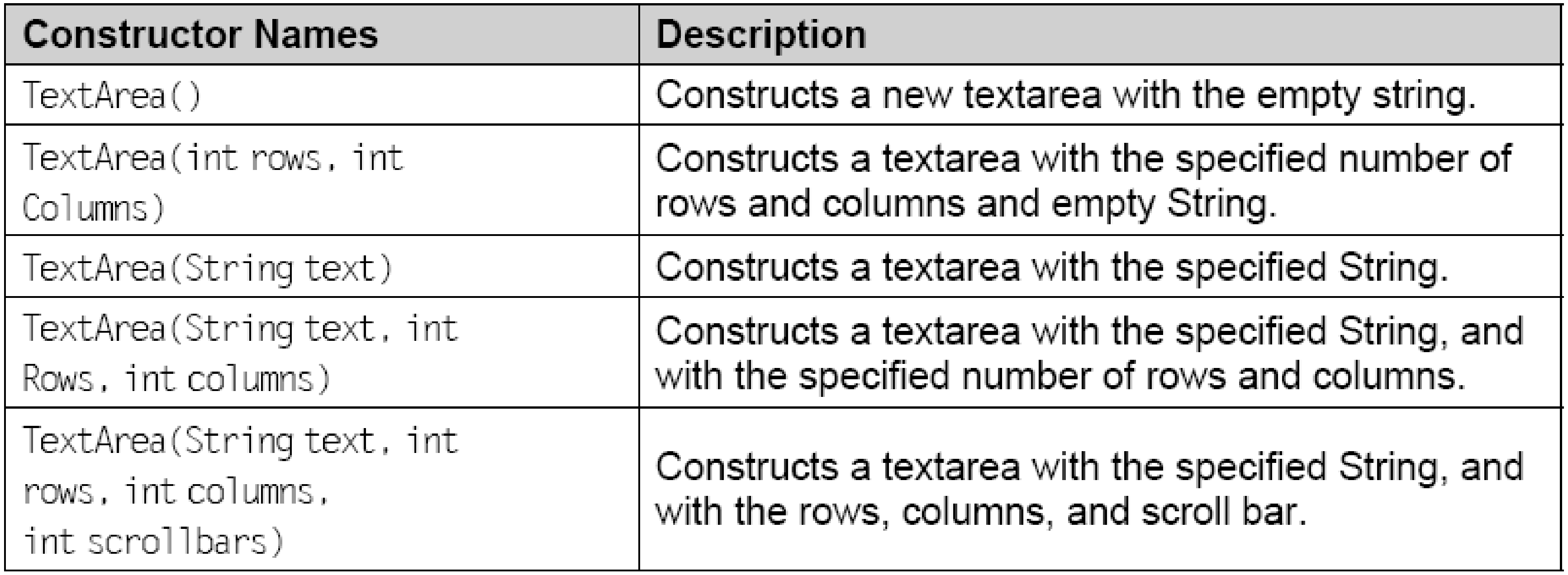
Constructor	Description
`TextField()`	Constructs a new textfield.
`TextField(int columns)`	Constructs a empty textbox with the number of Columns specified as argument.
`TextField(String text)`	Constructs a new textbox initialized with the specified String.
`TextField(String text, int columns)`	Constructs a new textbox initialized with the specified String, and specified number of columns.

Methods of TextField



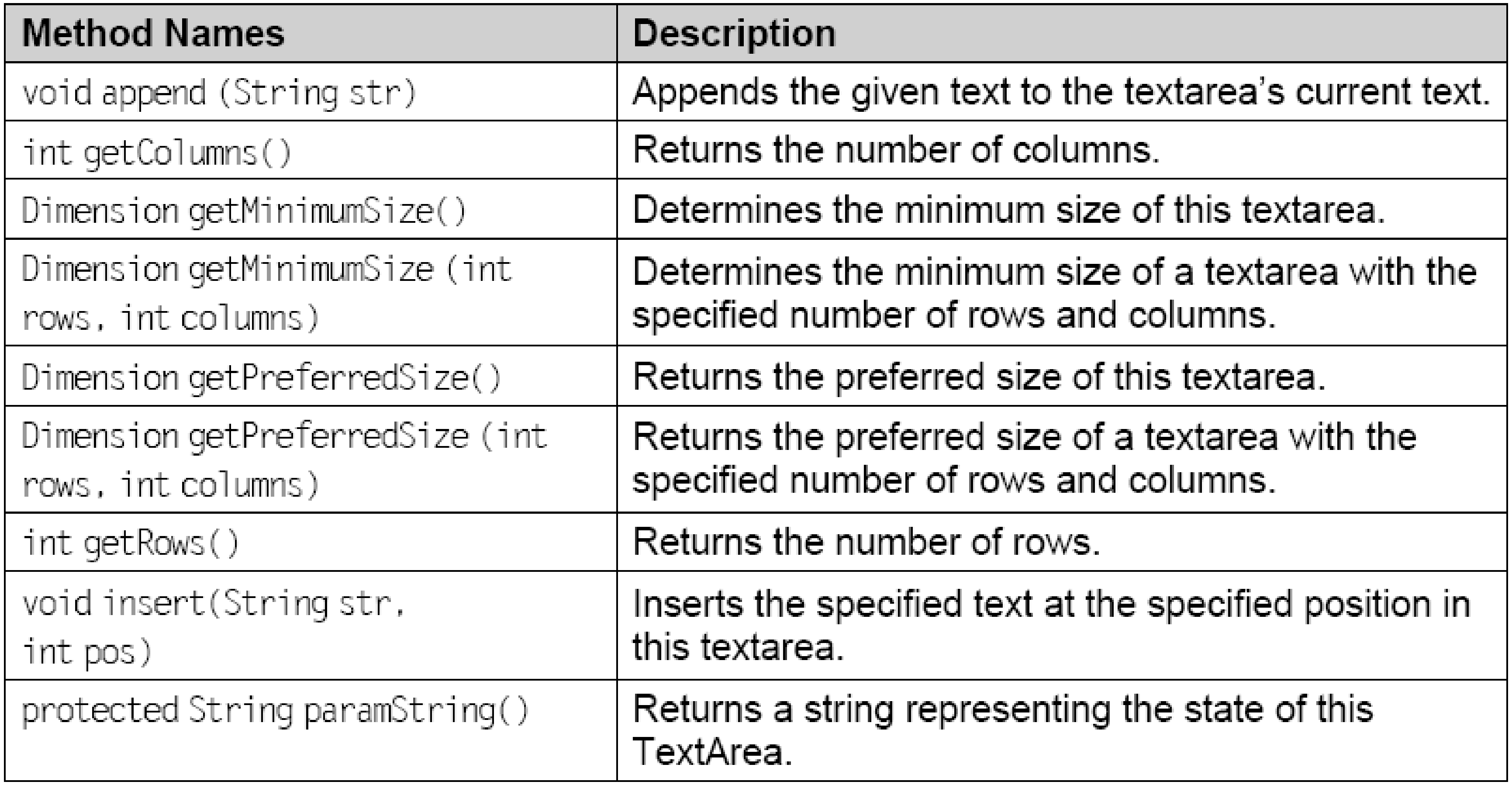
Method Names	Description
void addActionListener (ActionListener l)	Adds the specified action listener to receive action events from this textfield.
boolean echoCharIsSet()	Indicates whether or not this textfield has set a character for echoing.
AccessibleContext getAccessibleContext()	Returns the AccessibleContext associated with this TextField.
ActionListener[] getActionListeners ()	Returns an array of all the action listeners associated with the textfield.
int getColumns()	Returns the number of columns in this textfield.
char getEchoChar()	Returns the character that is to be used for echoing.
Dimension getMinimumSize()	Returns the minumum dimensions for this textfield.
Dimension getMinimumSize (int columns)	Returns the minumum dimensions for a textfield with the specified number of columns.
Dimension getPreferredSize()	Returns the preferred size of this textfield.
Dimension getPreferredSize (int columns)	Returns the preferred size of this textfield with the specified number of columns.
void removeActionListener (ActionListener l)	Removes the associated action listener from this textfield.
void setColumns(int columns)	Sets the number of columns for the textfield.
void setEchoChar(char c)	Sets the echo character.
void setText(String t)	Sets the String within the textfield.

TextArea



Constructor Names	Description
`TextArea()`	Constructs a new textarea with the empty string.
`TextArea(int rows, int Columns)`	Constructs a textarea with the specified number of rows and columns and empty String.
`TextArea(String text)`	Constructs a textarea with the specified String.
`TextArea(String text, int Rows, int columns)`	Constructs a textarea with the specified String, and with the specified number of rows and columns.
`TextArea(String text, int rows, int columns, int scrollbars)`	Constructs a textarea with the specified String, and with the rows, columns, and scroll bar.

Few methods of TextArea



Method Names	Description
void append (String str)	Appends the given text to the textarea's current text.
int getColumns()	Returns the number of columns.
Dimension getMinimumSize()	Determines the minimum size of this textarea.
Dimension getMinimumSize (int rows, int columns)	Determines the minimum size of a textarea with the specified number of rows and columns.
Dimension getPreferredSize()	Returns the preferred size of this textarea.
Dimension getPreferredSize (int rows, int columns)	Returns the preferred size of a textarea with the specified number of rows and columns.
int getRows()	Returns the number of rows.
void insert(String str, int pos)	Inserts the specified text at the specified position in this textarea.
protected String paramString()	Returns a string representing the state of this TextArea.

Container classes

- **Window**
  - Window is a top-level display surface. An object of Window class is not Attached to nor embedded within another container. An instance of the Window does not have border, title bar or menu.
- **Frame**
  - Frame is a top-level window with a border and title. An instance of the Frame class may have a menu bar, title bar and borders. It is otherwise like an object of the Window class.
- **Dialog**
  - Dialog is top-level display surface (a window) with a border and title. An object of the Dialog class cannot exist without an associated object of the Frame class.
- **Panel**
  - Panel is generic container for holding components. An instance of the Panel class provides a container to which components can be added. It does not add any new method; it simply implements the Container.

Layouts

- FlowLayout
- BorderLayout
- GridLayout
- GridbagLayout

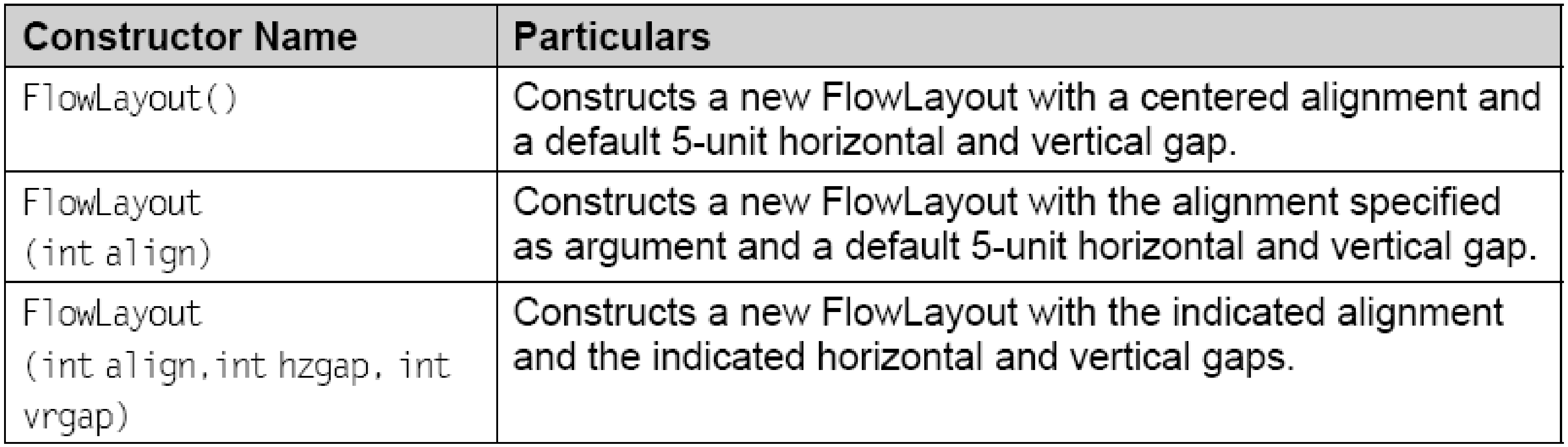
FlowLayout

- arranges components from left-to-right and top-to-bottom, centering components horizontally.
- The direction of flow is determined by the container’s componentOrientation property.
  - **ComponentOrientation.LEFT_TO_RIGHT :** Items run left to right and lines flow top to bottom, e.g. English, French, etc.
  - **ComponentOrientation.RIGHT_TO_LEFT :** Items run right to left and lines flow top to bottom, e.g. Arabic, Hebrew, etc.
- There is five pixel gap between the components arranged in this layout.
- the default layout for the Applet.

Fields of FlowLayout

Field Name	Particulars
`static int CENTER`	This value shows that each row of components should be centered.
`static int LEADING`	This value shows that each row of components should be justified to the leading edge of the container's orientation, e.g. to the right in right-to-left orientations.
`static int LEFT`	This value shows that each row of components should be left-justified.
`static int RIGHT`	This value shows that each row of components should be right-justified.
`static int TRAILING`	This value shows that each row of components should be justified to the trailing edge of the container's orientation, e.g. to the left in right-to-left orientations.

Constructors of FlowLayout

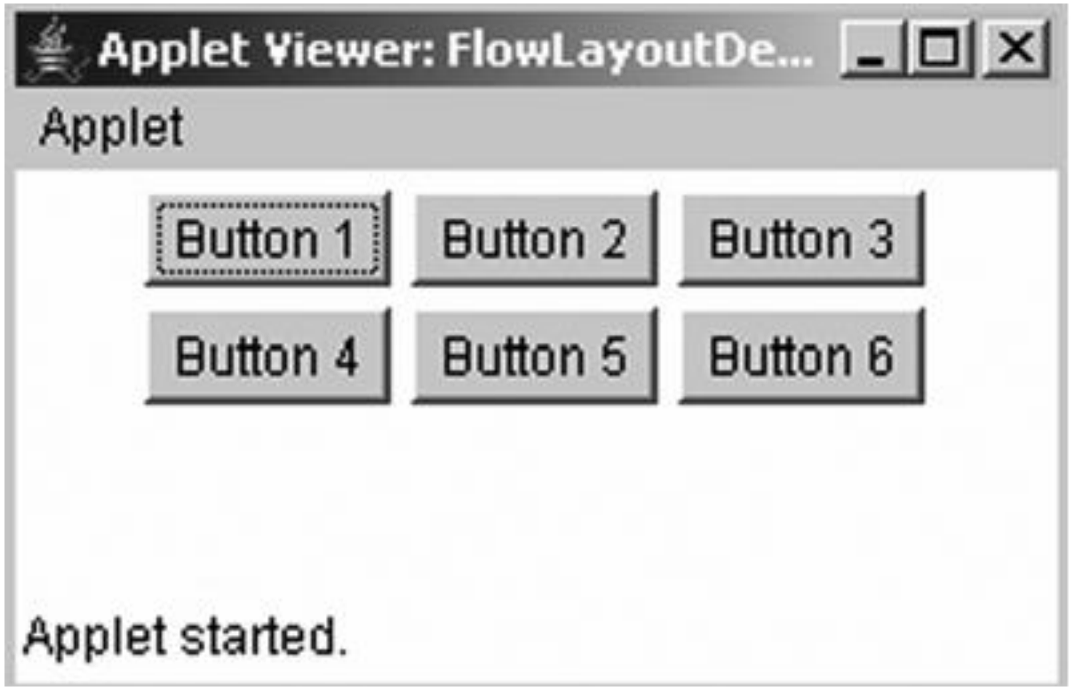


Constructor Name	Particulars
`FlowLayout()`	Constructs a new FlowLayout with a centered alignment and a default 5-unit horizontal and vertical gap.
`FlowLayout (int align)`	Constructs a new FlowLayout with the alignment specified as argument and a default 5-unit horizontal and vertical gap.
`FlowLayout (int align,int hzgap, int vrgap)`	Constructs a new FlowLayout with the indicated alignment and the indicated horizontal and vertical gaps.

Example

```
/*<applet code=FlowLayoutDemo.class width=600
height=350></applet>*/
import java.applet.Applet;   import java.awt.*;
public class FlowLayoutDemo extends Applet{
LayoutManager flowLayout;
Button [] Buttons;
public FlowLayoutDemo() {
int i;
flowLayout = new FlowLayout ();
setLayout (flowLayout);
Buttons = new Button [6];
for (i = 0; i < 6; i++) {
Buttons[i] = new Button ();
Buttons[i].setLabel (“Button “ + (i + 1));
add (Buttons[i]);}}}
```

The Output

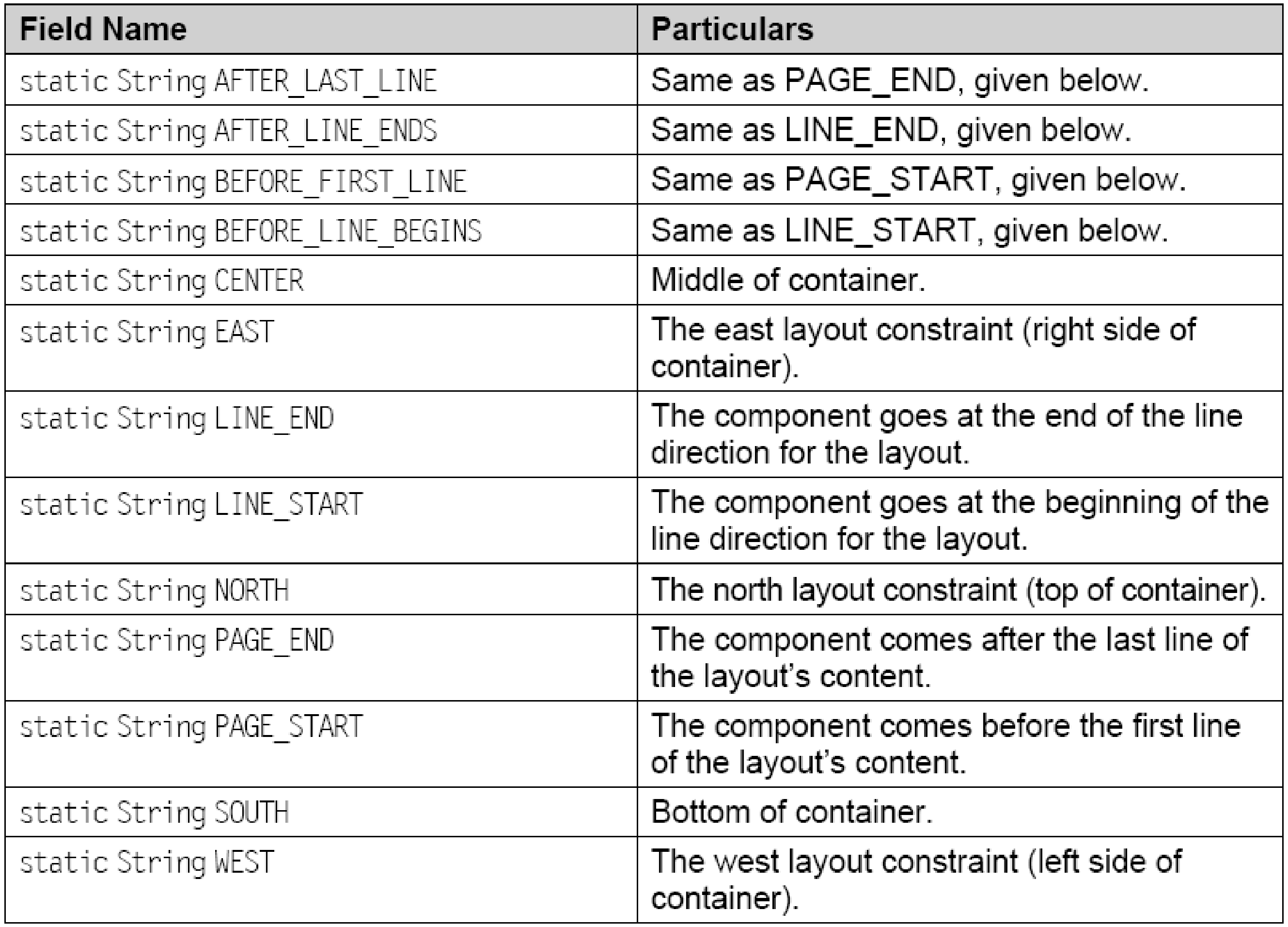


Applet started.

BorderLayout

- This is the default layout of the Frame.
  - public class **BorderLayout** extends Object implements LayoutManager2, Serializable
- There can be only one component in each region and the regions are identified as constants:
  - NORTH, SOUTH, EAST, WEST, and CENTER.
- Any of these five constant names can be used while adding a component to a container.
  - Panel pnl = new Panel();
  - pnl.setLayout(new BorderLayout());
  - pnl.add(new Button(“submit”), BorderLayout.NORTH);

Fields of BorderLayout

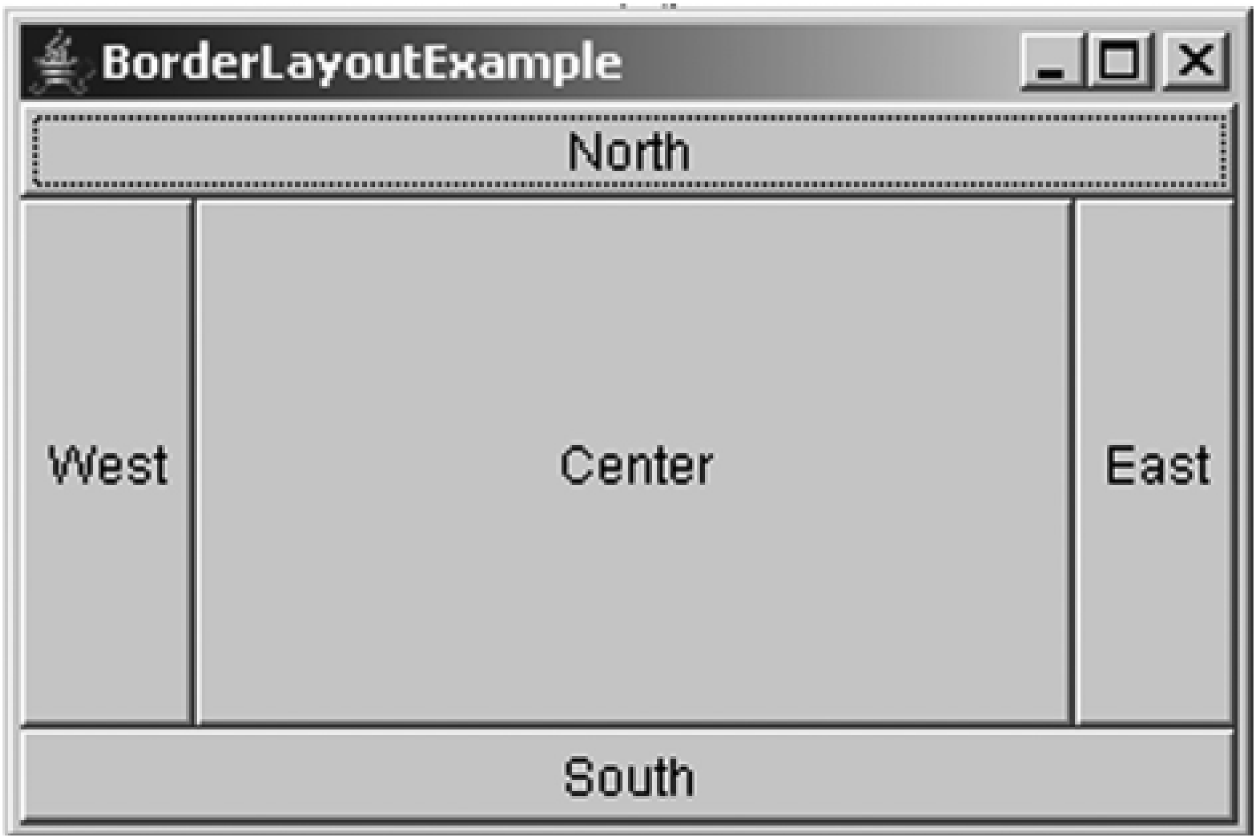


Field Name	Particulars
static String AFTER_LAST_LINE	Same as PAGE_END, given below.
static String AFTER_LINE_ENDS	Same as LINE_END, given below.
static String BEFORE_FIRST_LINE	Same as PAGE_START, given below.
static String BEFORE_LINE_BEGINS	Same as LINE_START, given below.
static String CENTER	Middle of container.
static String EAST	The east layout constraint (right side of container).
static String LINE_END	The component goes at the end of the line direction for the layout.
static String LINE_START	The component goes at the beginning of the line direction for the layout.
static String NORTH	The north layout constraint (top of container).
static String PAGE_END	The component comes after the last line of the layout's content.
static String PAGE_START	The component comes before the first line of the layout's content.
static String SOUTH	Bottom of container.
static String WEST	The west layout constraint (left side of container).

Example

```
import java.awt.*;
public class BLayoutDemo extends Frame {
public BLayoutDemo(String title) {
super(title);
add(new Button(“North”),BorderLayout.NORTH);
add(new Button(“South”),BorderLayout.SOUTH);
add(new Button(“East”),BorderLayout.EAST);
add(new Button(“West”),BorderLayout.WEST);
add(new Button(“Center”),BorderLayout.CENTER);
setSize(400, 270);
setVisible(true);}
public static void main(String[] args) {
BLayoutDemo blaypout = new BLayoutDemo(“Border Layout
Example”);}}
```

The Output



CardLayout

- Each component is treated as a card by cardLayout object.
- Each card kept on another like a stack and only one card can be visible at a time.
- When the container is displayed after adding the first component, then the first component is visible.
- The ordering of cards is determined by the container’s own internal ordering of its component objects.
- CardLayout defines a set of methods that allow an application to flip through these cards sequentially, or to show a specified card.

Example

```
import java.awt.*;
import java.awt.event.*;
public class CardDemo extends Frame implements
ActionListener {
Panel cardPanel;     Panel p1, p2, p3;                    Panel
buttonP;
Button B1,B2,B3;     CardLayout cLayout;
public void cardDemo(){
cardPanel = new Panel();
cLayout = new CardLayout();
cardPanel.setLayout(cLayout);
p1 = new Panel();
p1.setBackground(Color.red);
p2 = new Panel();
p2.setBackground(Color.yellow);
```

Example (contd.)

```
p3 = new Panel();
p3.setBackground(Color.green);
B1 = new Button(“Red”);
B1.addActionListener(this);
B2 = new Button(“Yellow”);
B2.addActionListener(this);
B3 = new Button(“Green”);
B3.addActionListener(this);
buttonP = new Panel();
buttonP.add(B1);
buttonP.add(B2);
buttonP.add(B3);
cardPanel.add(p1, “B1”);
cardPanel.add(p2, “B2”);
cardPanel.add(p3, “B3”);
setLayout(new BorderLayout());
```

Example (contd.)

```
add(buttonP, BorderLayout.SOUTH);
add(cardPanel, BorderLayout.CENTER);
setVisible(true);            setSize(300,200);
setTitle(“DemoCard”);
addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent we){
        System.exit(0);}});}
public void actionPerformed(ActionEvent e){
if (e.getSource() == B1)
        cLayout.show(cardPanel, “B1”);
if (e.getSource() == B2)
        cLayout.show(cardPanel, “B2”);
if (e.getSource() == B3)
        cLayout.show(cardPanel, “B3”);}
public static void main(String a[]){
        CardDemo demo=new CardDemo();
        demo.cardDemo();} }
```

GridLayout

- public class GridLayout extends Object implements LayoutManager, Serializable
- lays out a container’s components in a rectangular grid.

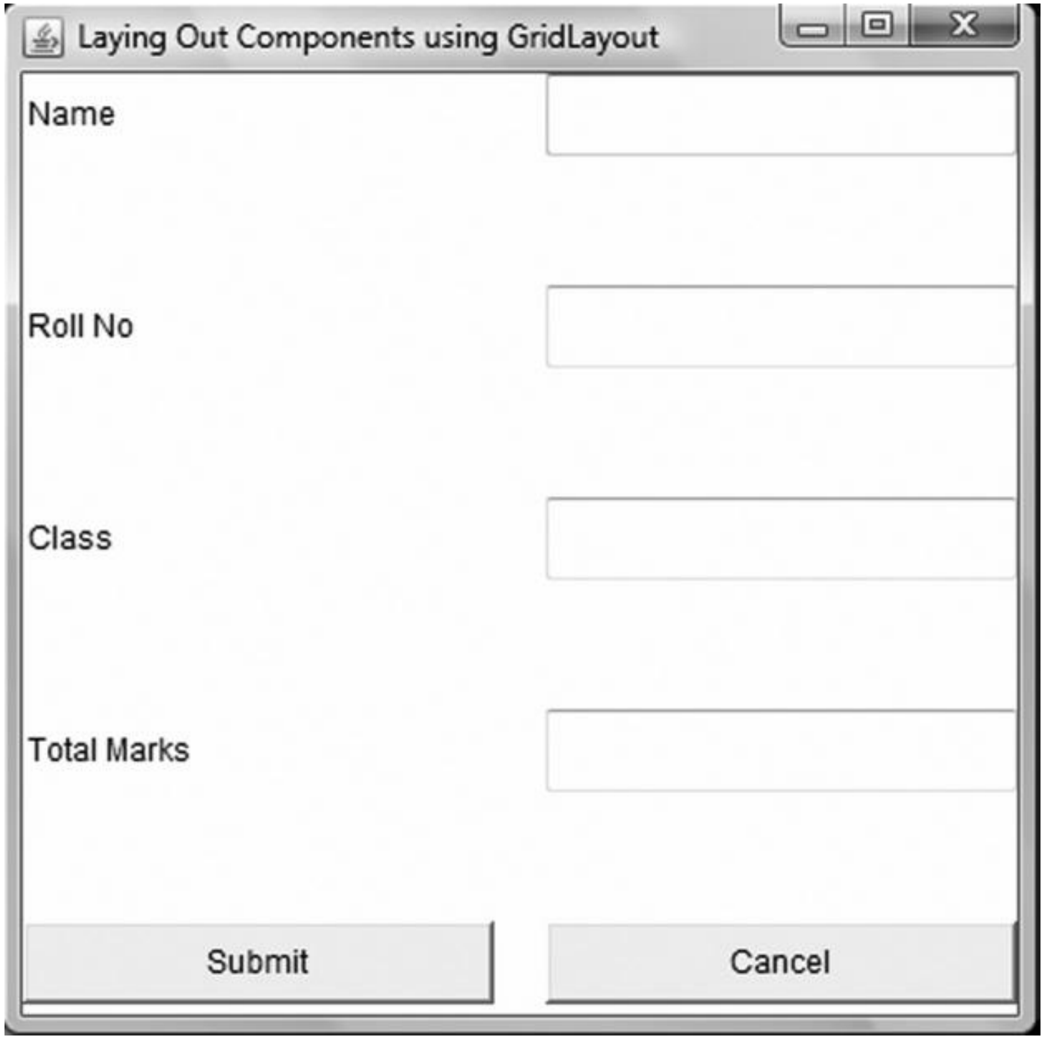
Example

```
import java.awt.event.*;
import java.awt.*;
class GridLayoutDemo extends Frame{
public GridLayoutDemo() {
super(“Laying Out Components using GridLayout”);
Panel p = new Panel(new GridLayout(5,2, 20,50));
p.add(new Label(“Name”));
p.add(new TextField(5));
p.add(new Label(“Roll No”));
p.add(new TextField(3));
p.add(new Label(“Class”));
p.add(new TextField(3));
p.add(new Label(“Total Marks”));
p.add(new TextField(3));
```

Example (contd.)

```
p.add(new Button(“Submit”));
p.add(new Button(“Cancel”));
add(p);
setSize(400,400);
setVisible(true);
addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent e){
System.exit(0);}});}
public static void main(String[] args) {
GridLayoutDemo g=new GridLayoutDemo();
}}
```

The Output


Laying Out Components using GridLayout
Name
Roll No
Class
Total Marks
Submit
Cancel

GridBagLayout

- We can arrange components in horizontal as well in vertical direction or by positioning them within a cell of a grid
- components need not be of same size in a row.
- each row can contain dissimilar number of columns.
- GridBagConstraint contains the constraint which includes the
  - height, width of a cell, placement and alignment of components.
- GridBagLayout object maintains a rectangular grid of cell.
- Component can occupy one or more cells, a.k.a its display area.

GridBagLayout

- ComponentOrientation class controls the orientation of the grid.
- We need to customize GridBagConstraints objects to use GridBagLayout effectively associated with its components.
- Customization of a GridBagConstraints object can be doneby setting one or more of its instance variables.
  - gridx & gridy
    - The initial address of cell of a grid is gridx = 0 and gridy = 0.
  - gridwidth & gridheight
    - gridwidth constraint specifies the number of cells in a row and gridheight specifies number of columns in display area of the components. The default value is 1.

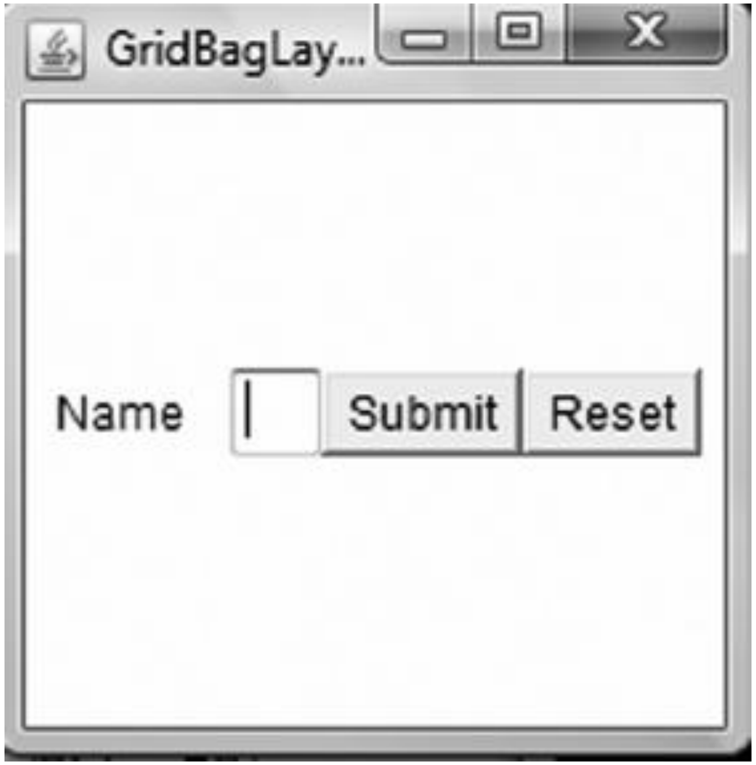
GridBagLayout (contd.)

- fill
  - GridBagConstraints.NONE (default value–does not grow when the window is resized)
  - GridBagConstraints.HORIZONTAL (this value fills all the horizontal display area of a component, but it does not change height).
  - GridBagConstraints.VERTICAL (it changes the height of a component, but does not change its width)
  - GridBagConstraints.BOTH (makes the component fill its display area horizontally and vertically, both).
- ipadx and ipady
  - for internal padding of components in given layout.
- Insets
  - used for spacing between the component and the edges of its display area
- anchor
  - specifies the position of a component in its display area
- weight x & weight y
  - used to distribute space (horizontal and vertical)

Example

```
import java.awt.*;
public class GBLayoutDemo1 extends Frame{
public GBLayoutDemo1(){
setLayout(new GridBagLayout());
setTitle(“GridBagLayout Without Constraints”);
Label l=new Label(“Name”);
add(l);
TextField t=new TextField();
add(t);
Button b=new Button(“Submit”);
add(b);
Button b1=new Button(“Reset”);
add(b1);
setSize(200,200);        setVisible(true);}
public static void main(String args[]){
GBLayoutDemo1 d=new GBLayoutDemo1();}}
```

The Output



Menu

- Menu is a class which inherits MenuItem class and two interfaces:
  - MenuContainer and
  - Accessible.
- Menubar deploys a menu object which is a dropdown menu component.
  - It shows a list of menu choices.
- To implement this concept we use three classes:
  - MenuBar,
  - Menu, and
  - MenuItem.

Example

```
import java.awt.event.*;
import java.awt.*;
public class DemoMenu extends Frame implements ActionListener{
public void demoMenu() {
setTitle(“MenuDemo”);
setSize(250,150);
MenuBar menuBar = new MenuBar();
setMenuBar(menuBar);
MenuShortcut n=new MenuShortcut(KeyEvent.VK_N);
MenuShortcut o=new MenuShortcut(KeyEvent.VK_O);
MenuShortcut x=new MenuShortcut(KeyEvent.VK_X);
Menu fileMenu = new Menu(“File”);
Menu editMenu = new Menu(“Edit”);
MenuItem newAction = new MenuItem(“New”,n);
MenuItem openAction = new MenuItem(“Open”,o);
MenuItem exitAction = new MenuItem(“Exit”,x);
```

Example (contd.)

```
MenuItem cutAction = new MenuItem(“Cut”);
MenuItem copyAction = new MenuItem(“Copy”);
MenuItem pasteAction = new menuItem(“Paste”);
newAction.addActionListener(this);
openAction.addActionListener(this);
exitAction.addActionListener(this);
fileMenu.addSeparator();
fileMenu.add(newAction);
fileMenu.addSeparator();
fileMenu.add(openAction);
fileMenu.addSeparator();
fileMenu.add(exitAction);
menuBar.add(fileMenu);
cutAction.addActionListener(this);
copyAction.addActionListener(this);
```

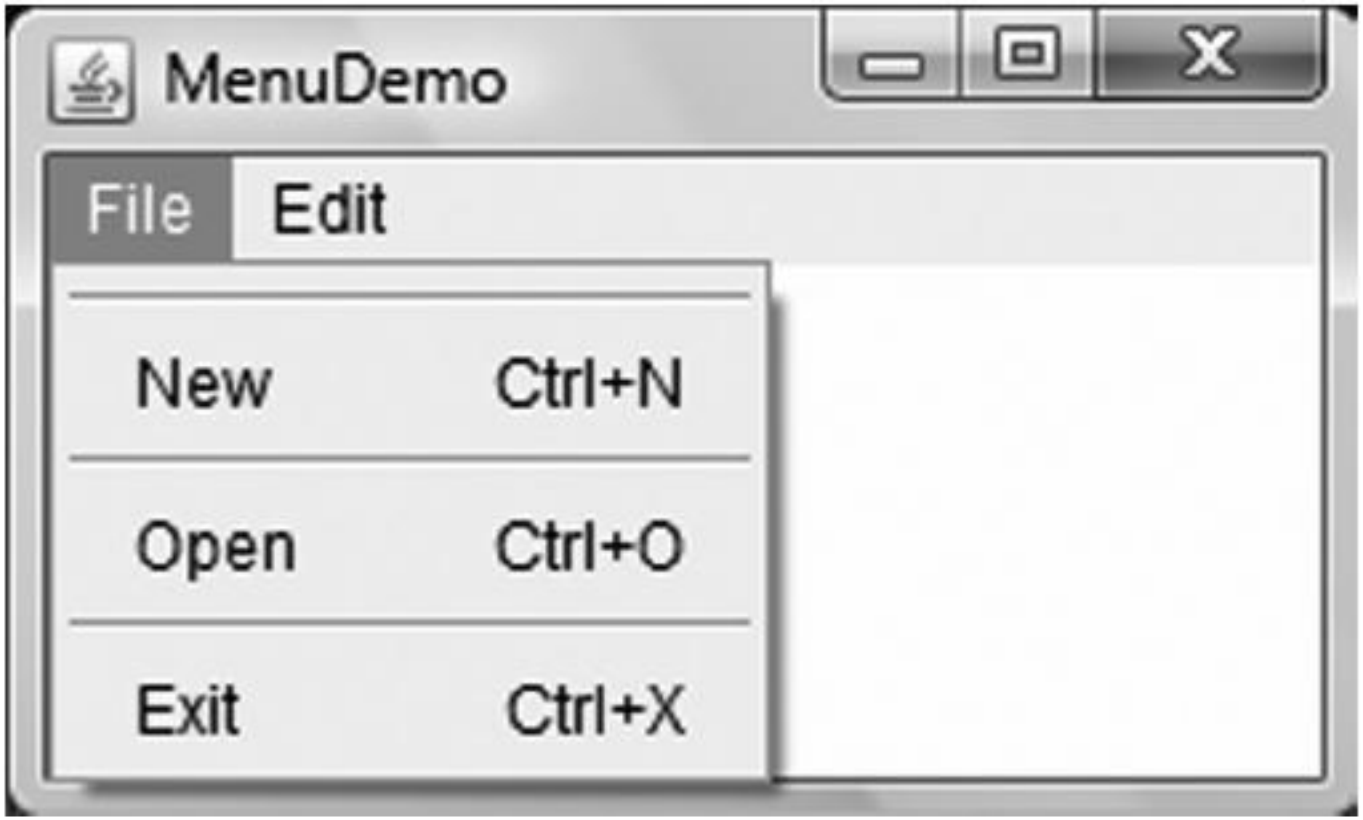
Example (contd.)

```
pasteAction.addActionListener(this);
editMenu.add(cutAction);
editMenu.addSeparator();
editMenu.add(copyAction);
editMenu.addSeparator();
editMenu.add(pasteAction);
editMenu.addSeparator();
menuBar.add(editMenu);
setVisible(true);
addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent we){
System.exit(0);}});                     }
public void actionPerformed(ActionEvent e) {
String action=e.getActionCommand();
if(action.equals("New")){
System.out.println("New");}
```

Example (contd.)

```
else if(action.equals(“Open”)){
System.out.println(“File”);}
else if(action.equals(“Exit”)){
System.exit(0);}
else if(action.equals(“Cut”)){
System.out.println(“Cut”);}
else if(action.equals(“Copy”)){
System.out.println(“Copy”);}
else if(action.equals(“Paste”)){
System.out.println(“Paste”);}}
public static void main(String[] args) {
DemoMenu demo= new DemoMenu();
demo.demoMenu();} }
```

The Output



Scrollbar

- Scrollbars are used to select continuous values through a range of integer values (the range set between maximum and minimum).
- These scrollbars can either be set horizontally or vertically.
- The scrollbar's maximum and minimum values can be set along with line increments and page increments.
  - Scrollbar() throws HeadlessException
  - Scrollbar(int direction) throws HeadlessException
  - Scrollbar(int direction, int initValue, int pageSize, int min, int max) throws HeadlessException

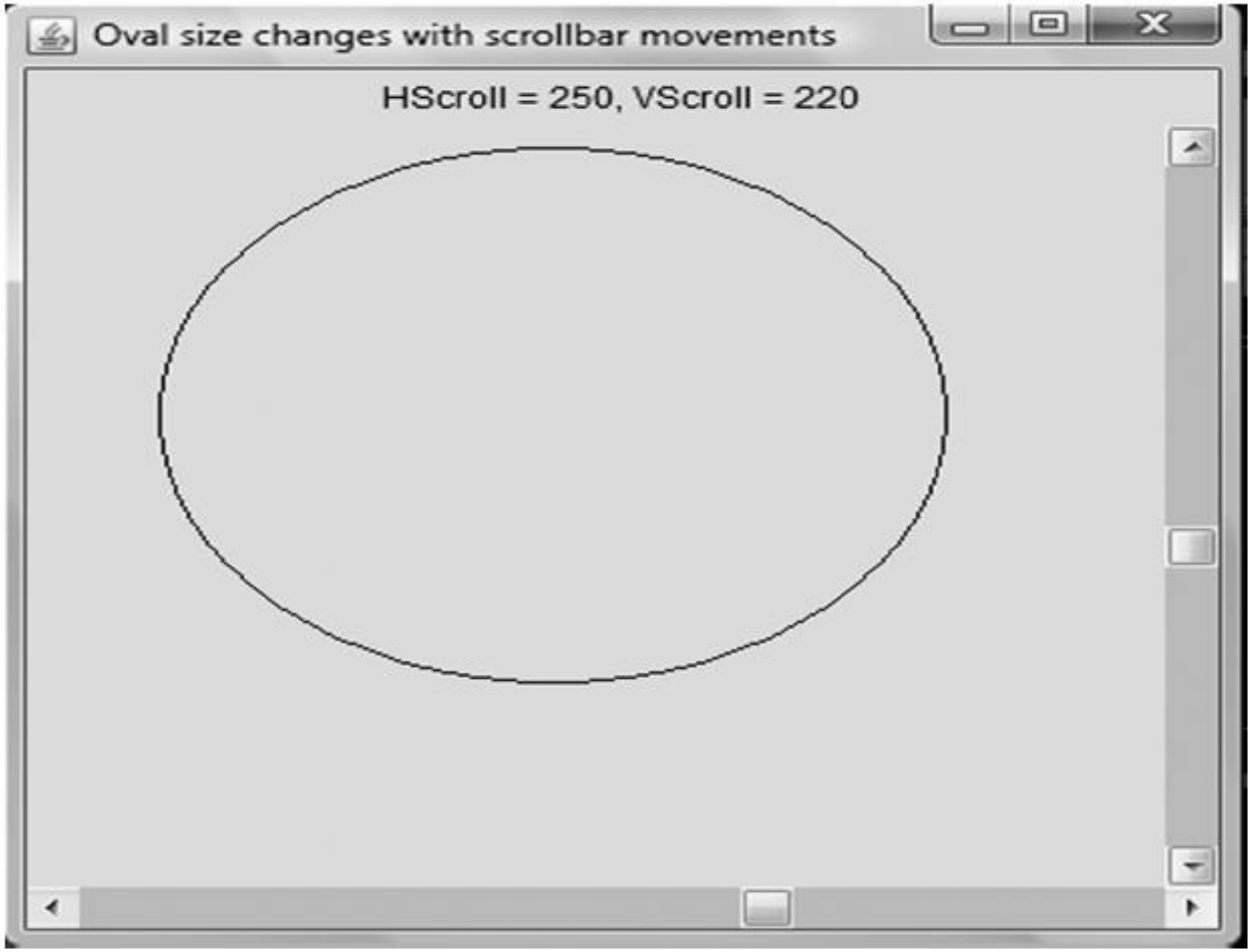
Example

```
import java.awt.*;
import java.awt.event.*;
public class ScrollbarDemo extends Frame implements AdjustmentListener
{	Scrollbar HScroll, VScroll;
	Label lbl;
	int X=100,Y=150;
	public ScrollbarDemo () {
	HScroll = new Scrollbar (Scrollbar.HORIZONTAL);
	VScroll = new Scrollbar (Scrollbar.VERTICAL);
	lbl = new Label ("",Label.CENTER);
	HScroll.setMaximum (400);
	VScroll.setMaximum (400);
	setBackground (Color.cyan);
	setTitle("Oval size changes with scrollbar movements");
	setLayout (new BorderLayout());
	add (lbl,BorderLayout.NORTH);
	add (HScroll,BorderLayout.SOUTH);
```

Example (contd.)

```
add (VScroll, BorderLayout.EAST);
HScroll.addAdjustmentListener (this);
VScroll.addAdjustmentListener (this);
HScroll.setValue (X);     VScroll.setValue (Y);
lbl.setText (“HScroll = “ + HScroll.getValue() + “, VScroll = “ + VScroll
getValue());
setSize(500,500);          setVisible(true);
addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent e){        System.exit(0);} });}
public void adjustmentValueChanged(AdjustmentEvent e) {
X=HScroll.getValue();    Y= VScroll.getValue ();
lbl.setText (“HScroll = “ + X + “, VScroll = “ + Y);
repaint();}
public void paint (Graphics g) {
g.drawOval (50, 60, X, Y); }
public static void main(String args[])        {
ScrollbarDemo d=new ScrollbarDemo();}}
```

The Output



Oval size changes with scrollbar movements

HScroll = 250, VScroll = 220

Summary

- In this chapter we have emphasized on Graphical User Interface (GUI) for input and output.
- Java has a package named as **java.awt**, having various classes responsible for generating various GUI frameworks.
- These components include Button, Scrollbar, Choicebox, List, TextField, etc.
- AWT defines ways to lay the AWT components in containers.
- There are many layout managers in AWT like
  - FlowLayout,
  - GridLayout,
  - GridBagLayout,
  - CardLayout, etc,
- Menu is a class that inherits MenuItem class and two interfaces: MenuContainer and Accessible.