```
6     <body>
7        <applet code="LabelTest.class" width="300" height="200"></applet>
8     </body>
9  </html>
```

HTML is called through the browser or by passing the path and name of HTML to 'appletviewer' in the Command windows.

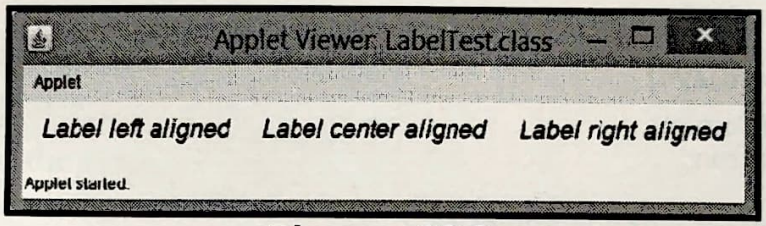

Diagram 23.3

Methods of the Label object:

Methods	Action
getText()	Returns a string containing this label's text
setText()	Changes the text of the label
getAlignment()	Returns an integer representing the alignment of the label 0 is Label.LEFT 1 is Label.CENTER 2 is Label.RIGHT
setAlignment()	Changes the alignment of the label to the given integer. Use the class variables listed in getAlignment()

Buttons

Buttons are GUI components. They are used to trigger an action within the GUI when clicked by a user. For example: A calculator applet might have a button for each number and operator or a dialog box might have the buttons OK and Cancel. A Button can be defined as a GUI component that, when pressed, causes and event to fire which in turn can be used to initiate some action.

To create a button, one of the following constructors can be used:

- **Button()**: Creates an empty button with no label
- **Button(String)**: Creates a button with the given string as its label displayed on the button face

Syntax:

```
Button btnName = new Button("Button Name");
add(btnName);
```

Here,

- The first statement creates an object of **Button** and passes **Button Name** as a parameter to the constructor

- The second statement adds the component to an Applet using add()

Example

The following example creates four buttons named Save, Edit, Delete and Reset and adds them to an Applet.

Code spec: [ButtonTest.java]

```
1  import java.applet.Applet;
2  import java.awt.Button;
3
4  public class ButtonTest extends Applet {
5      @Override
6      public void init() {
7          Button btnSave = new Button("Save");
8          add(btnSave);
9          Button btnEdit = new Button("Edit");
10         add(btnEdit);
11         Button btnDelete = new Button("Delete");
12         add(btnDelete);
13         Button btnReset = new Button("Reset");
14         add(btnReset);
15     }
16 }
```

Explanation:

Four objects of the button class are created. The text i.e. the name of the button is passed as a parameter to the constructor of the class. add() adds the component on the applet.

Being an applet, this class file needs to be embedded into the HTML, i.e. index.html.

The contents of the HTML: [index.html]

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Button Test</title>
5      </head>
6      <body>
7          <applet width="300" height="200" code="ButtonTest.class"></applet>
8      </body>
9  </html>
```

The HTML is called through the browser or by passing the path and name of the HTML to 'appletviewer' in the Command windows.

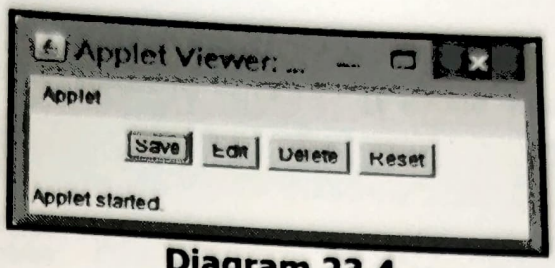

Diagram 23.4

Methods of the Button object:

Methods	Action
getLabel()	Returns the string set as the label of the Button
setLabel()	Adds or changes the label of a Button

Check Boxes

Check boxes are GUI components that can carry one of two states viz. ON or OFF [i.e. checked or unchecked, selected or unselected, true or false and so on]. Unlike buttons, check boxes are not normally used to trigger an action in a GUI, but are used to capture additional but optional features of some other action.

Check boxes can be used in one of two ways:

- **Nonexclusive:** From a series of check boxes in the GUI, **any** combination of them can be selected. Here, any checkbox can be checked or unchecked independently of the other checkboxes. Nonexclusive check boxes are created using the **Checkbox** class
- **Exclusive:** From a series of check boxes in the GUI, **only one** check box can be selected at any instant in time. Exclusive checkboxes are created by first creating a Radio Group and then assigning check boxes to the Radio Group. From within a radio group only one checkbox can be selected or checked at any instant in time

Create a checkbox using any one of its constructors, as follows:

- **Checkbox():** Creates an empty i.e. unselected checkbox
- **Checkbox(String):** Creates a check box with a label defined in the string
- **Checkbox(String, boolean):** Creates a checkbox with a label and is either selected or deselected based on whether the boolean argument is TRUE or FALSE
- **Checkbox(String, null, boolean):** Creates a checkbox that is named and is either selected or deselected based on whether the boolean argument is TRUE or FALSE. NULL is used as a placeholder for a group argument i.e. in case of check boxes that belong to a Radio group

Syntax:

```
Checkbox chk = new Checkbox("Shoes");
add(chk);
```

Here,

- The first statement creates an object of the Checkbox class and passes 'Shoes' as the checkbox label, as a parameter to its constructor
- The second statement adds checkbox to an applet using add()

Example

The following example displays checkboxes on an applet.

Code spec: [CheckboxTest.java]

```
1  import java.awt.*;
2
3  public class CheckboxTest extends java.applet.Applet {
4      @Override
5      public void init() {
6          setFont(new Font("Verdana", Font.ITALIC, 16));
7          Label lblHobbies = new Label("Hobbies: ", Label.LEFT);
8          add(lblHobbies);
9          Checkbox chkReading = new Checkbox("Reading");
10         add(chkReading);
11         Checkbox chkMusic = new Checkbox("Listening To Music", true);
12         add(chkMusic);
13         Checkbox chkDriving = new Checkbox("Driving");
14         add(chkDriving);
15     }
16 }
```

Explanation:

Three object of the Checkbox class are created. The text i.e. the options for the checkbox is passed as a parameter to the constructor of the class. add() adds the component on the applet.

Being an applet, this class file needs to be embedded into the HTML, i.e. index.html.

The contents of the HTML: [index.html]

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Checkbox Test</title>
5      </head>
6      <body>
7          <applet width="300" height="200" code="CheckboxTest.class"></applet>
8      </body>
9  </html>
```

The HTML is called through the browser or by passing the path and name of HTML to 'appletviewer' in the Command windows.

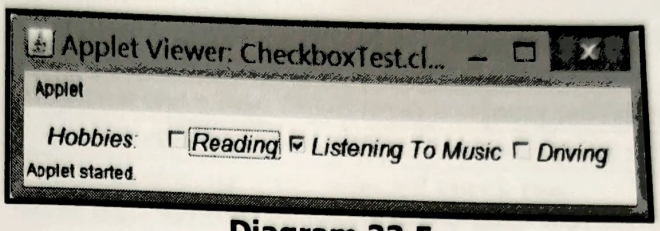

Diagram 23.5

Methods of the Checkbox object:

Methods	Action
getLabel()	Returns the string which is checkbox's label
setLabel()	Adds or changes the string which is the checkbox's label
getState()	Returns TRUE or FALSE depending on whether the checkbox is selected or not
setState()	Changes the checkbox's state to selected [TRUE] or unselected [FALSE]

Radio Buttons

A Radio button is a GUI component that looks like a hollow round circle. A **selected** radio button is round circle with a black spot in its center. Radio Buttons are created from **Checkbox**. In Radio Buttons only one in a series of radio buttons can be selected at a time. To create a series of radio buttons, an instance of **CheckboxGroup** must be created first.

```
CheckboxGroup cbg = new CheckboxGroup();
```

Once this parent container is created, instantiate and add individual radio buttons from **Checkbox** to it. This can be done by using **Checkbox(String, CheckboxGroup, boolean)**, which creates a radio button with a label that displays the string, the name of the checkbox group and a boolean value TRUE or FALSE which determines whether the radio button is selected or not.

Syntax:

```
CheckboxGroup chkgrp = new CheckboxGroup();
Checkbox chkbx = new Checkbox("itemname", chkgrp, false);
add(chkbx);
```

Here,

- The first statement creates an object of **CheckboxGroup** i.e. the parent container
- The second statement creates an object of **Checkbox** whose label is itemname and which belongs to chkgrp and is not selected
- The third statement adds the item to the group using **add()**

Example

This example displays radio buttons on an applet.

Code spec: [RadioButtonTest.java]

```
1  import java.awt.*;
2
3  public class RadioButtonTest extends java.applet.Applet {
4     @Override
5     public void init() {
6        setFont(new Font("Verdana", Font.ITALIC, 16));
7        Label lblGender = new Label("Gender: ", Label.LEFT);
8        add(lblGender);
9        CheckboxGroup chkgrp = new CheckboxGroup();
10       Checkbox chkMale = new Checkbox("Male", chkgrp, true);
11       add(chkMale);
12       Checkbox chkFemale = new Checkbox("Female", chkgrp, false);
13       add(chkFemale);
14    }
15 }
```

Explanation:

An object of CheckboxGroup is created. Two object of Checkbox are created. The text i.e. the options for the checkbox, the name of the group to which the check box belongs to and the value whether the checkbox should be selected are passed as the parameters to the constructor of the class. add() adds the component on the applet.

Being an applet, this class file needs to be embedded into an HTML, i.e. index.html.

The contents of an HTML: [index.html]

```
1 <!DOCTYPE html>
2 <html>
3    <head>
4       <title>Radio Button Test</title>
5    </head>
6    <body>
7       <applet code="RadioButtonTest.class" width=300 height=200></applet>
8    </body>
9 </html>
```

The HTML file is called through the browser or by passing the path and name of the HTML file to 'appletviewer' in the Command windows.

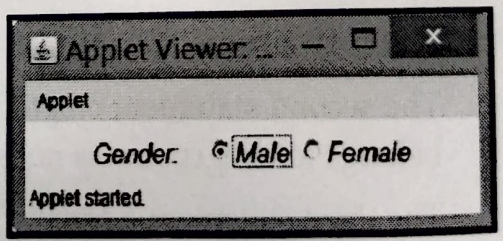

Diagram 23.6

Methods of the CheckboxGroup object:

Methods	Action
getCheckboxGroup()	Returns a handle to the parent container of the checkboxes
setCheckboxGroup()	Changes the group of any given checkbox
getSelectedCheckbox()	Returns the value TRUE from a selected checkbox
setSelectedCheckbox()	Sets the checkbox indentified in checkbox as selected
getCurrent()	Returns the currently selected check box
setCurrent()	Sets the currently selected check box

Choice Menus Or Choice Lists

Choice menus are GUI components that are a drop-down list of items from which the user can select any **one** item at a time. To create a choice menu, first create an instance of the **Choice** class and then add individual items to this object.

Syntax:

```
Choice ch = new Choice();
ch.add("apples");
ch.add("oranges");
```

Here,

- The first statement creates an object of **Choice**
- The second and third statement adds two items to the Choice list

Example

This example displays a dropdown Choice list in an applet.

Code spec: [ChoiceTest.java]

```
1  import java.awt.*;
2
3  public class ChoiceTest extends java.applet.Applet {
4     @Override
5     public void init() {
6        setFont(new Font("Verdana", Font.ITALIC, 16));
7        Label lblHobbies = new Label("Hobbies: ", Label.LEFT);
8        add(lblHobbies);
9        Choice chHobbies = new Choice();
10       chHobbies.add("Reading");
11       chHobbies.add("Listening to Music");
12       chHobbies.add("Driving");
13       add(chHobbies);
14    }
15 }
```

Explanation:

An object of Choice is created. The items are added to the list by add() of Choice. add() adds the component i.e. the Choice object on the applet.

Being an applet, this class file needs to be embedded into an HTML, i.e. index.html.

The contents of an HTML: [index.html]

```
1 <!DOCTYPE html>
2 <html>
3    <head>
4       <title>Text Field Test</title>
5    </head>
6    <body>
7       <applet code="TextFieldTest.class" width=300 height=200></applet>
8    </body>
9 </html>
```

The HTML file is called through the browser or by passing the path and name of the HTML file to 'appletviewer' in the Command windows.

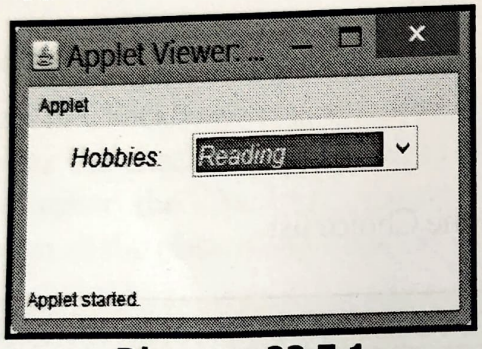

Diagram 23.7.1

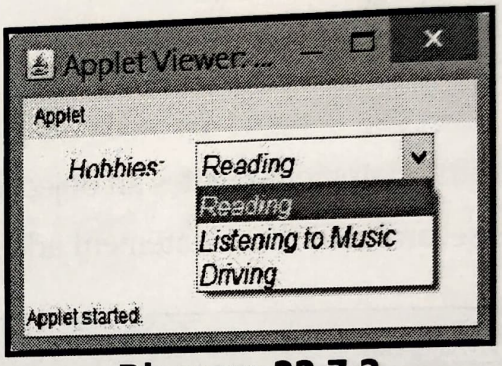

Diagram 23.7.2

Methods of the Choice object:

Methods	Action
getItem()	Returns the string at the given position [items inside a choice are held in an array whose index begins at 0]
getSelectedIndex()	Returns the array index value of the item selected by the user
select()	Selects an item at the stated array index position or selects a value within the array by identifying it based on the string passed.

Text Fields

Text fields are GUI components that provide an area, generally on a form, into which a user can enter / edit a single line of text. Text fields are generally used for capturing textual data from a user.

TextField is used to create text fields in a parent container, as follows:

- **TextField()**: Creates an empty text field which can be resized by a layout manager
- **TextField(int)**: Creates an empty text field. The integer value passed as an argument indicates the minimum number of characters to be displayed in the text field
- **TextField(String)**: Creates a text field which displays with the given string within it
- **TextField(String, int)**: Creates a text field which displays the value of the string within it and which has a specific size as determined by the integer value passed as an argument

Syntax:

```
TextField <Object Name> = new TextField(size);
```

The following example creates a TextField, which accepts 10 characters:

```
TextField txt = new TextField(10);
```

Syntax:

```
TextField <Object Name> = new TextField(<String Name>, size);
```

The following example creates a TextField, which accepts 10 characters and displays a 'Hello' in the text field when it is first instantiated in a container:

```
TextField txt = new TextField("Hello", 10);
```

Example

This example displays text fields in an applet.

Code spec: [TextFieldTest.java]

```
1  import java.awt.*;
2
3  public class TextFieldTest extends java.applet.Applet {
4     @Override
5     public void init() {
6        setFont(new Font("Verdana", Font.BOLD, 12));
7        Label lblFirstName = new Label("First Name");
8        add(lblFirstName);
9        TextField txtFirstName = new TextField("Enter First Name", 30);
10       add(txtFirstName);
11       Label lblLastName = new Label("Last Name");
12       add(lblLastName);
13       TextField txtLastName = new TextField("Enter Last Name", 30);
14       add(txtLastName);
15       Label lblUserName = new Label("User Name");
16       add(lblUserName);
```

```
        TextField txtUserName = new TextField("Enter User Name", 30);
17      add(txtUserName);
18      Label lblPassword = new Label("Password");
19      add(lblPassword);
20      TextField txtPassword = new TextField(30);
21      txtPassword.setEchoChar('*');
22      add(txtPassword);
23    }
24  }
25
```

Explanation:

Objects of TextField are created. Parameters are passed viz. the text which is displayed inside the text box and the size of the text box. add() adds the component on the applet. setEchoChar() is used to illustrate the creation of a text field, which can accept passwords from the users.

Being an applet, this class file needs to be embedded into an HTML, i.e. index.html.

The contents of an HTML: [index.html]

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Radio Button Test</title>
5    </head>
6    <body>
7      <applet code="TextFieldTest.class" width=300 height=200></applet>
8    </body>
9  </html>
```

The HTML file is called through the browser or by passing the path and name of the HTML file to 'appletviewer' in the Command windows.

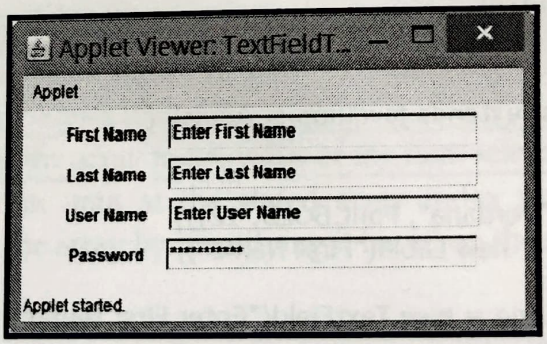

Diagram 23.8

Methods of the Text Field object:

Methods	Action
getText()	Returns the text that the text field contains

Methods of the Text Field object: [Continued]

Methods	Action
setText()	Places the string passed as an argument within the text field
setColumns()	Sets the width of the text field
select()	Selects the text between the two integer positions from within the text field. Positions start from 0
selectAll()	Selects all the text in the text field
isEditable()	Returns TRUE or FALSE based on whether the text field is editable or not
setEditable()	TRUE [the default] ensure that a text field is editable FALSE ensures that a text field is not editable
getEchoChar()	Returns the character used for masking data being keyed into a text field
setEchoCharacter()	Sets character to be used for masking data being keyed into a text field
echoCharIsSet()	Returns TRUE or FALSE based on whether the text field has a masking character set or not

Text Areas

Text Areas are GUI components that are editable text fields that can handle **more than one** line of textual input. Text Areas are created using **TextArea**. Text areas have horizontal and vertical scroll bars that enable the user to scroll through the text contained within TextArea.

TextArea can be instantiated by using one of the following constructors:

- **TextArea():** Creates an empty text area of an unspecified height and width
- **TextArea(int, int):** Creates an empty text area with the indicated number of lines [first argument] and the indicated number of characters per line [second argument]
- **TextArea(String):** Creates a text area which displays the String within it, of unspecified height and width
- **TextArea(String, int, int):** Creates a text area which displays the String within it, of a specified number of lines and characters contained on each line

Example

The following example displays a text area in an applet.

Code spec for TextAreaTest.java:

```
1 import java.awt.*;
2
3 public class TextAreaTest extends java.applet.Applet {
```

```
4     String letter = "";
5     TextArea txtareaDetails;
6
7     @Override
8     public void init() {
9        letter = this.getParameter("details");
10       txtareaDetails = new TextArea(letter, 10, 35);
11       add(txtareaDetails);
12    }
13 }
```

Explanation:

An object of **TextArea** is created by passing three arguments to the constructor. The string letter, the number of lines and width in characters.

Being an applet, the class file needs to be embedded into an HTML, i.e. index.html.

The contents of an HTML: [index.html]

```
1 <!DOCTYPE html>
2 <html>
3    <head>
4       <title>Text Area Test</title>
5    </head>
6    <body>
7       <applet code="TextAreaTest.class" width=300 height=200></applet>
8    </body>
9 </html>
```

An HTML is called through the browser or by passing the path and name of the HTML file to 'appletviewer' in the Command windows.

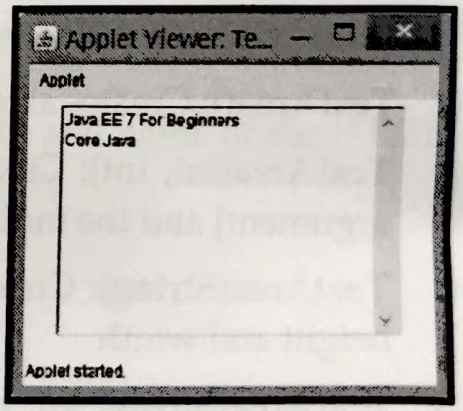

Diagram 23.9

As both **TextArea** and **TextField** inherit from **TextComponent**, one can use setText(), getText(), setEditable() and isEditable() of **TextField**, with respect to **TextArea**.

Methods of the TextArea object:

Methods	Action
insertText()	Inserts the string at the character index indicated by the integer
replaceText()	Replaces the text with the string provided between the given integer positions with the string

Scrolling Lists

Scrolling lists are GUI components whose functionality is similar to **Choice** menus or **Choice** lists with two significant differences:

- A scrolling list can be created where more than one item can be selected from it at a time
- Scrolling lists do not pop-up when selected. Instead, multiple items are displayed within a scrolling list using scroll bars

Scrolling lists are created using the **List** class. This is done by instantiating an object of **List** and then adding individual items to the list. The list class has the following constructors:

- **List()**: Creates an empty scrolling list that enables only one item to be selected at a time
- **List(int, boolean)**: Creates a scrolling list which displays a specific number of items within it, from multiple items, as indicated in **int**. The **boolean** argument indicates whether multiple items can be selected or not

After creating a scrolling list and populating it with items, the list should be added to its parent container for display using **add()**.

Example

The following example creates a list having list items and displays this in an Applet.

Code spec for ScrollingListTest.java:

```
1  import java.awt.*;
2
3  public class ScrollingListTest extends java.applet.Applet {
4     List ltHobbies = new List(6, true);
5     @Override
6     public void init() {
7        ltHobbies.add("Reading");
8        ltHobbies.add("Listening Music");
9        ltHobbies.add("Dancing");
10       ltHobbies.add("Driving");
11       ltHobbies.add("Collecting Stamps");
12       add(ltHobbies);
13    }
14 }
```

Explanation:

An object of List is created by passing two arguments to the constructor of List viz. number of items to be displayed and a boolean value. The boolean value indicates whether multiple items can be selected from the list or not.

Being an applet, this class file needs to be embedded into an HTML, i.e. index.html.

The contents of an HTML: [index.html]

```
1 <!DOCTYPE html>
2 <html>
3    <head>
4       <title>Scrolling List Test</title>
5    </head>
6    <body>
7       <applet code="ScrollingListTest.class" width=300 height=200></applet>
8    </body>
9 </html>
```

The HTML file is called through the browser or by passing the path and name of the HTML file to 'appletviewer' in the Command windows.

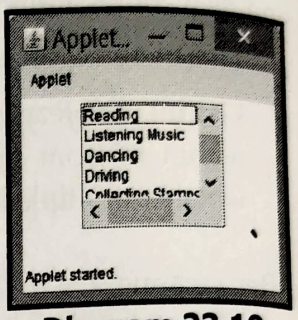

Diagram 23.10

Scrolling lists have several methods that work exactly the same as methods for choice lists such as getItem(), countItems(), getSelectedIndex(), getSelectedItem() and select().

Methods associated to Scrolling List:

Methods	Action
getSelectedIndexes()	Returns an array of integers containing the index position of each selected item
getSelectedItems()	Returns an array of strings containing the text of each selected item

Scrollbars

Scrollbars are GUI components that enable values to be displayed by sliding a box between two arrows as shown in diagram 23.11. Several GUI components have built-in scrollbar functionality such as text areas and scrolling lists. The Scrollbar class is used for creating scrollbars. A scrollbar can be oriented either horizontally or vertically as required.

To create a scrollbar use any one of the following constructors:

- **Scrollbar():** Creates a vertical scroll bar with its initial maximum and minimum values equal to 0
- **Scrollbar.HORIZONTAL(int):** Creates a scrollbar with minimum and maximum values of 0 but with the indicated orientation. Scrollbar class **constants** are used to set a scroll bar orientation i.e. Scrollbar.HORIZONTAL or Scrollbar.VERTICAL
- **Scrollbar(int, int, int, int, int):** Takes five arguments

1. Orientation can be either Scrollbar.HORIZONTAL or Scrollbar.VERTICAL

2. The initial value of the scrollbar, which must be equal to or between the maximum / minimum values specified of the bar
3. The overall width or height of the box used to scroll through the scroll bar. This can be equal to 0 when wanting to use its default size
4. The minimum value of the scrollbar
5. The maximum value of the scrollbar

Example

The following example displays a vertical scrollbar on an applet.

Code spec: [ScrollBarTest.java]

```
1 import java.awt.*;
2
3 public class ScrollBarTest extends java.applet.Applet {
4    Scrollbar scrollBarHorizontal = new Scrollbar(Scrollbar.HORIZONTAL, 10, 0, 1,
     100);
5    @Override
6    public void init() {
7       add(scrollBarHorizontal);
8    }
9 }
```

Explanation:

An object of **Scrollbar** is created by passing five arguments to the constructor of **Scrollbar** viz. orientation of the scrollbar i.e. Horizontal, the initial value of the scrollbar, the width and height of the scrollbar, the minimum value of the scrollbar and the maximum value of the scrollbar. add() adds the component i.e. the Scrollbar object on the applet.

Being an applet, this class file needs to be embedded into an HTML, i.e. index.html.

The contents of an HTML: [index.html]

```
1 <!DOCTYPE html>
2 <html>
3    <head>
4       <title>Scrolling Bar Test</title>
5    </head>
6    <body>
7       <applet code="ScrollingBarTest.class" width=300 height=200></applet>
8    </body>
9 </html>
```

The HTML file is called through the browser or by passing the path and name of the HTML file to 'appletviewer' in the Command windows.

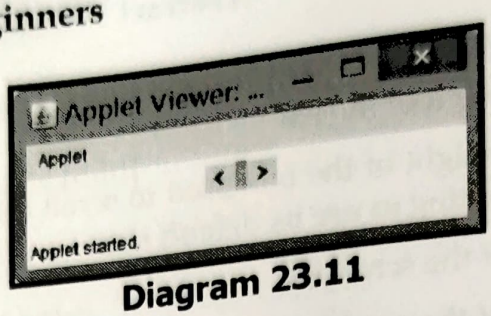

Diagram 23.11

Methods of a Scrollbar:

Methods	Action
getValue()	Returns the position of the box within the scroll bar
setValue()	Sets the position of the box within the scrollbar

Working With Panels

A **Panel** is GUI component which is a subclass of **Container**. It does not add any new methods. It simply implements **Container**. A **Panel** may be thought of as a recursively nestable, GUI screen component. **Panel** is the superclass of **Applet**. Whenever screen output is directed to an applet, it is drawn on the surface of a Panel object. A **Panel** can be conceptualized as a window that does not contain a title bar, menu bar or border.

Example

The following example shows how to create panels and nest sub-panels within them.

Code spec: [PanelTest.java]

```
1  import java.awt.*;
2
3  public class PanelTest extends java.applet.Applet {
4     @Override
5     public void init() {
6     /* Creating the main panels */
7        Panel mainPanel1 = new Panel();
8        Panel mainPanel2 = new Panel();
9
10    /* Creating the sub-panels */
11       Panel subPanel1 = new Panel();
12       Panel subPanel2 = new Panel();
13
14    /* Adding a Button directly to the Applet */
15       add(new Button("Applet Button"));
16
17    /* Adding the main panels to the Applet */
18       add(mainPanel1);
19       add(mainPanel2);
20
```

```
21      /* Giving main Panel 1 a button and a sub-panel */
22         mainPanel1.add(new Button("Main Panel 1 Button"));
23         mainPanel1.add(subPanel1);
24
25      /* Giving main Panel 2 a button and a sub-panel */
26         mainPanel2.add(new Button("Main Panel 2 Button"));
27         mainPanel2.add(subPanel2);
28
29      /* Giving each sub panel a button */
30         subPanel1.add(new Button("Sub Panel 1 Button"));
31         subPanel2.add(new Button("Sub Panel 2 Button"));
32      }
33  }
```

Explanation:

Four objects of **Panel** are created. A button is added directly to the applet. The two panels are added to the applet using add(). Buttons and panels are added to the two main panels. One button each are added to the sub panels respectively.

Being an applet, this class file needs to be embedded into an HTML, i.e. index.html.

The contents of an HTML: [index.html]

```
1  <!DOCTYPE html>
2  <html>
3     <head>
4        <title>Panel Test</title>
5     </head>
6     <body>
7        <applet code="PanelTest.class" width=300 height=200></applet>
8     </body>
9  </html>
```

The HTML file is called through the browser or by passing the path and name of the HTML file to 'appletviewer' in the Command windows.

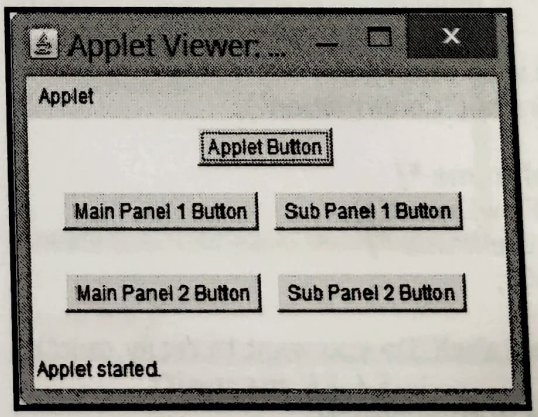

Diagram 23.12

Working With Frames

A Frame is a GUI component that is a window derived from the **Frame** class. A second object derived from Frame is an Applet. Frame is used to create child windows within applets and top-level or child windows for applications. A frame is a standard-style window.

A Frame can be instantiated using these two constructors:

- **Frame()**: Creates a standard window without a title
- **Frame(String title)**: Creates a window which displays the **title** specified

There are several methods available for use when working with windows spawned from Frame:

Methods	Action
dispose()	To close frame window
getTitle()	To get the title of the frame window
isResizable()	It takes the Boolean values TRUE or FALSE which in turn determine whether the window is user resizable or not
resize()	It sets a new dimension to the window spawned from frame based on the size specified as a number of pixels
setTitle()	To set the title of the frame window

Example

The following applet contains a simple Frame with a label and a text field.

Code spec: [FrameTest.java]

```
1  import java.awt.*;
2
3  public class FrameTest extends java.applet.Applet {
4     @Override
5     public void init() {
6     /* Constructing frame with title */
7        Frame frm = new Frame("Confirmation");
8
9     /* Setting the layout of frame */
10       frm.setLayout(new FlowLayout());
11    /* Setting the size for the frame */
12       frm.setSize(300, 150);
13
14       Label lblName = new Label("Do you want to really quit");
15       frm.add(lblName);
16       Button btnYes = new Button("Yes");
17       Button btnNo = new Button("No");
18       frm.add(btnYes);
```

```
19          frm.add(btnNo);
20
21      /* Showing the frame in Applet */
22          frm.setVisible(true);
23      }
24  }
```

Explanation:

An object of **Frame** is created and is passed the text to be displayed as the title of the frame. A layout of the frame is set via **setLayout()**. The Layout is the **FlowLayout**. The size of the frame is set in pixels via **setSize()**. A label stating **"Do you want to really quit"** and two buttons of **Yes** and **No** are added to the frame via add(). The frame is shown in the browser via setVisible().

Being an applet, this class file needs to be embedded into an HTML, i.e. index.html.

The contents of an HTML: [index.html]

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Frame Test</title>
5      </head>
6      <body>
7          <applet code="FrameTest.class" width=300 height=200></applet>
8      </body>
9  </html>
```

The HTML file is called through the browser or by passing the path and name of the HTML file to 'appletviewer' in the Command windows.

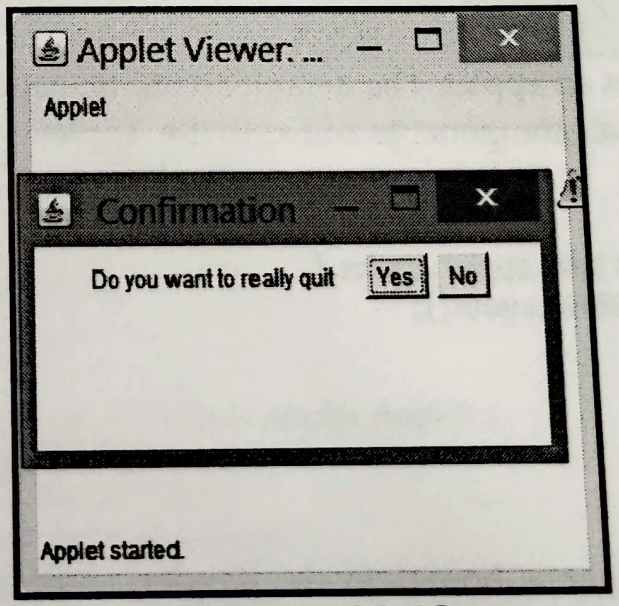

Diagram 23.13

Layouts

A layout is a GUI process that refers to the need of arranging / placing of components in a container such as data capture objects displayed within a form. In any programming environment, when designing an aesthetic GUI, a programmer has to specify, the top/left, row/column, co-ordinates of all objects that are displayed in the GUI. This is always with respect to 0,0 which is the top/left co-ordinates of the GUI window itself. Based on this data delivered by the programmer, the environment will create and display the GUI appropriately on a VDU screen.

Java has introduced a unique way of creating GUI's and displaying them on a VDU. Java has an interface called **LayoutManager**, which belongs to the **java.awt** package. This is used for placing objects in a container for display on screen **without** specifying their row, column co-ordinates. Java's layout manager determines how components will be arranged when they are added to a container such as a Frame or canvas.

The AWT package includes classes of the following layout managers, which implements LayoutManager:

1. FlowLayout
2. BorderLayout
3. GridLayout
4. CardLayout

To add a specific layout manager for a GUI, an object of one of the above classes is created. After spawning the layout manager, the layout manager is bound to the container of the GUI by using Container.setLayout(). A layout manager must be established before any components are added to any container. If no layout manager is specified, the container uses the Flow Layout manager by default.

The following example uses an applet. The default layout manager and setLayout() is used to control the arrangement of all components added to the Applet:

```
1 import java.awt.*;
2
3 public class Starter extends java.applet.Applet {
4    FlowLayout flow = new FlowLayout();
5    public void init() {
6       setLayout(flow);
7    }
8 }
```

After the flow layout manager is bound to the Applet, components can be added to the Applet.

Flow Layout

FlowLayout is the simplest of the Java layout managers. It lays out components in a container exactly as the way words are laid out on a page, from left to right until there is no more room, then onto the next line.

By default, the components on each row will be centered when using FlowLayout() constructor with no arguments. If the user wants the components to be aligned along the left or right edge of the container, the FlowLayout.LEFT or FlowLayout.RIGHT class constants should be the constructor's only argument as shown below. The FlowLayout.CENTER constant is used to specify centered components.

The following example ensures that components added to the container are justified to its right hand side:

```
FlowLayout flowRight = new FlowLayout(FlowLayout.RIGHT);
```

The **FlowLayout(int, int, int)** constructor takes the following three arguments, in order:

- The alignment, which must either be FlowLayout.CENTER or FlowLayout.LEFT or FlowLayout.RIGHT
- The horizontal gap between components, specified in pixels
- The vertical gap between components, specified in pixels

The following constructor creates a flow layout manager with centered components, a horizontal gap of 30 pixels and a vertical gap of 10:

```
FlowLayout flow = new FlowLayout(FlowLayout.CENTER, 30, 10);
```

Example

The following example displays six buttons arranged by the flow layout manager in an Applet. The buttons are centered within the applet.

Code spec: [FlowLayoutTest.java]

```
1  import java.awt.*;
2
3  public class FlowLayoutTest extends java.applet.Applet {
4      Button btnNew = new Button("New");
5      Button btnEdit = new Button("Edit");
6      Button btnDelete = new Button("Delete");
7      Button btnSave = new Button("Save");
8      Button btnReset = new Button("Reset");
9      FlowLayout flow = new FlowLayout(FlowLayout.CENTER);
10
11     @Override
```

```
12      public void init() {
13          setLayout(flow);
14          add(btnNew);
15          add(btnEdit);
16          add(btnDelete);
17          add(btnSave);
18          add(btnReset);
19      }
20  }
```

Explanation:

An object of FlowLayout is created and is passed the argument of alignment as CENTER. A layout of the frame is set via setLayout().

Being an applet, this class file needs to be embedded into an HTML, i.e. index.html.

The contents of an HTML: [index.html]

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Flow Layout Test</title>
5      </head>
6      <body>
7          <applet code="FlowLayoutTest.class" width=300 height=200></applet>
8      </body>
9  </html>
```

The HTML file is called through the browser or by passing the path and name of the HTML file to 'appletviewer' in the Command windows.

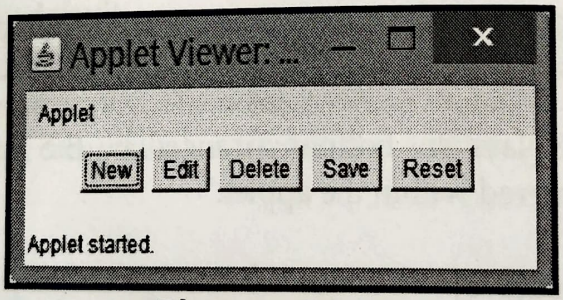

Diagram 23.14

Grid Layout

The grid layout manager is a component of a GUI that arranges GUI components within a grid of specified rows and columns. GUI components are first added to the top row of the grid, beginning with the leftmost grid cell and continuing to the right. When all of the cells in the top row are full, the next component is added to the leftmost cell in the second row of the grid - if there is a second row and so on.

Grid layout is created using an object instantiated from **GridLayout**. Two arguments are required by the **GridLayout** constructor - the number of rows and the number of columns.

The following statement creates a grid layout manager with 10 rows and 3 columns:

```
GridLayout grid = new GridLayout(10, 3);
```

As with flow layout, the user can specify a vertical and horizontal gap in pixels between GUI components using two extra arguments.

The following statement creates a grid layout with 10 rows, 3 columns, a horizontal gap of 5 pixels and a vertical gap of 8 pixels between its components:

```
GridLayout grid = new GridLayout(10, 3, 5, 8);
```

REMINDER

If no gap is specified when using the grid layout manager there will be no gap between the components when displayed in the GUI.

Example

The following example displays an applet that uses the GridLayout manager with a grid of 4 rows, 3 columns and a 7-pixel gap between GUI components in both the vertical and horizontal directions as shown in diagram 23.15.

Code spec: [GridLayoutTest.java]

```
 1 import java.awt.*;
 2
 3 public class GridLayoutTest extends java.applet.Applet {
 4    GridLayout grid = new GridLayout(4, 3, 7, 7);
 5    Button btnJanuary = new Button("January");
 6    Button btnFebruary = new Button("February");
 7    Button btnMarch = new Button("March");
 8    Button btnApril = new Button("April");
 9    Button btnMay = new Button("May");
10    Button btnJune = new Button("June");
11    Button btnJuly = new Button("July");
12    Button btnAugust = new Button("August");
13    Button btnSeptember = new Button("September");
14    Button btnOctober = new Button("October");
15    Button btnNovember = new Button("November");
16    Button btnDecember = new Button("December");
17
18    @Override
19    public void init() {
20       setLayout(grid);
```

```
21          add(btnJanuary);
22          add(btnFebruary);
23          add(btnMarch);
24          add(btnApril);
25          add(btnMay);
26          add(btnJune);
27          add(btnJuly);
28          add(btnAugust);
29          add(btnSeptember);
30          add(btnOctober);
31          add(btnNovember);
32          add(btnDecember);
33      }
34  }
```

Explanation:

The above code creates buttons and adds the buttons to the grid. The first three buttons, January, February, March are displayed in the first row. The next three buttons are displayed on the next row and so on.

An object of **GridLayout** is created and is passed arguments viz. the number of rows, the number of columns, the pixel gap between components in the vertical direction and the pixel gap between components in the horizontal directions. A layout of the frame is set via setLayout().

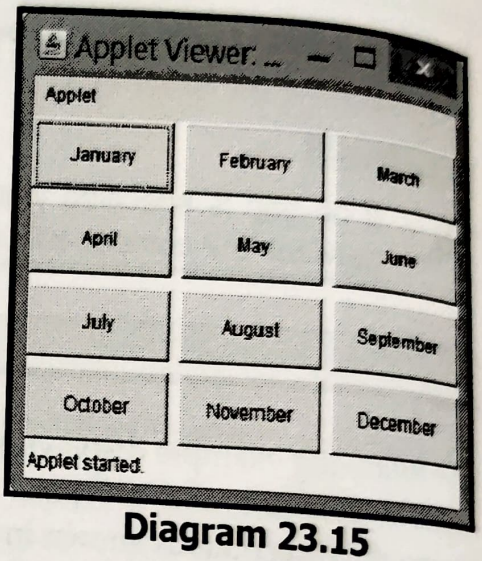

Diagram 23.15

Being an applet, this class file needs to be embedded into an HTML, i.e. index.html.

The contents of an HTML: [index.html]

```
<!DOCTYPE html>
<html>
  <head>
    <title>Grid Layout Test</title>
  </head>
  <body>
    <applet code="GridLayoutTest.class" width=300 height=200></applet>
  </body>
</html>
```

e HTML file is called through the browser or by passing the path and name of the HTML to 'appletviewer' in the Command windows.

Border Layout

A Border layout is a GUI component which is created using **BorderLayout**. The border layout manager divides its container into five sections: north, south, east, west and center. When using the border layout with a container, components placed in the four compass points will take up as much space as they need, the center gets whatever space is left over.

A border layout for a container is created with either of the following constructors:

- **BorderLayout()**: Creates a border layout with no gap between any of the components
- **BorderLayout(int, int)**: Specifies the horizontal gap and vertical gap in pixels between components

After creating a border layout and binding it as a container's layout manager, components are added using a different call to add() as shown below.

Syntax:

```
add(String, component)
```

The first argument is a string which specifies which part of the border layout the component will be assigned to. The component is then added to its container. There are five possible values: "North", "South", "East", "West" or "Center".

The following statement adds a button called **quitButton** to the north section of a border layout:

```
add("North", quitButton);
```

Example

The following code displays an applet using border layout.

Code spec: [BorderLayoutTest.java]

```
 1 import java.awt.*;
 2
 3 public class BorderLayoutTest extends java.applet.Applet {
 4     BorderLayout border = new BorderLayout();
 5     Button btnNorth = new Button("North");
 6     Button btnSouth = new Button("South");
 7     Button btnEast = new Button("East");
 8     Button btnWest = new Button("West");
 9     Button btnCenter = new Button("Center");
10
11     @Override
12     public void init() {
```

```
13          setLayout(border);
14          add("North", btnNorth);
15          add("South", btnSouth);
16          add("East", btnEast);
17          add("West", btnWest);
18          add("Center", btnCenter);
19      }
20  }
```

Explanation:

The above code adds the **North** button at the top most corner of the applet, the **South** button at bottom, the **East** button on the right-hand corner of the applet, **West** button on the left-hand corner of the applet and the **Center** button occupies the entire central position.

Being an applet, this class file needs to be embedded into an HTML, i.e. index.html.

The contents of an HTML: [index.html]

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Border Layout Test</title>
5      </head>
6      <body>
7          <applet code="BorderLayoutTest.class" width=300 height=200></applet>
8      </body>
9  </html>
```

The HTML file is called through the browser or by passing the path and name of the HTML file to 'appletviewer' in the Command windows.

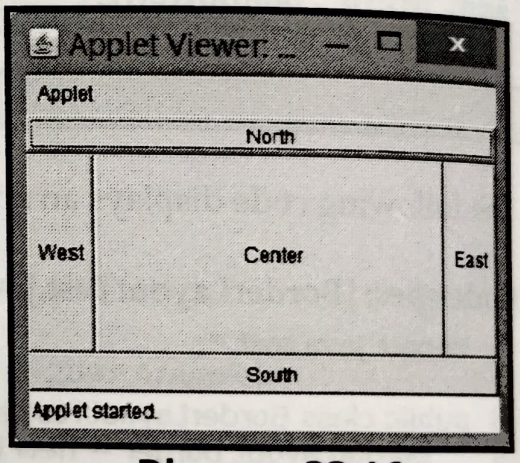

Diagram 23.16

Card Layout

Java's AWT includes one last layout manager used to place/position components within a GUI. The card layout manager can be mixed with the other managers, by nesting one container inside another.

The card layout manager differs from the other Java layout managers because it actually hides some of the GUI components from being viewed when it is used. A **card layout** is a group of containers that are displayed one at a time with each container in the group being known as a **card**.

Creating And Setting A Card Layout

A card layout is created from CardLayout with a simple constructor call:

```
CardLayout <ObjectName> = new CardLayout();
```

Solution:

```
CardLayout card = new CardLayout();
```

Creates an object of CardLayout.

setLayout() is used to bind the card layout manager to the container:

```
setLayout(card);
```

Adding A Card Layout

After binding the card layout manager to the container use add() to add cards to the container.

Syntax:

```
add(String, container);
```

- **String:** passed to add() of the card layout manager is a string that represents a user defined name for the card
- **container:** specifies the container to which the card will be added. All the necessary components must be added to the card before the card is added to the container

The following statement adds a panel called **options** to a container and gives this card the name "Options Card":

```
add("Options Card", options);
```

Displaying The Card

After adding a card to the container such as an Applet window, show() of the CardLayout class displays the card. show() takes two arguments:

- The container name in to which all the cards have been added. If the container is an Applet, the **this** keyword can be used as an argument
- The name of the card

The following statement calls show() of a card layout manager added to an Applet to display the "Finance Department" card on the Applet:

```
card.show(this, "Finance Department");
```

The this keyword refers to the class that contains this statement [i.e. the Applet] and "Finance Department" is the name of the card.

REMINDER

When using card layouts, only one card in a Card Layout can be viewed at a time.

Example

The following is an example, which displays series of text using a card layout.

Code spec: [CardLayoutTest.java]

```
1  import java.awt.*;
2
3  public class CardLayoutTest extends java.applet.Applet {
4     CardLayout card = new CardLayout();
5  /* An array of Labels to display static text */
6     Label[] lbl = new Label[5];
7
8     @Override
9     public void init() {
10    /* Assigning the static text to each element in the array */
11       lbl[0] = new Label("Personal Details");
12       lbl[1] = new Label("Financial Details");
13       lbl[2] = new Label("Contact Details");
14       lbl[3] = new Label("Dispatch Details");
15       lbl[4] = new Label("Credit Details");
16       setLayout(card);
17
18       for (int i = 0; i < 5; i++) {
19       /* Adding each element to the Card Layout */
20          add("Card " + i, lbl[i]);
21       /* Displaying each element of the Card on to the applet using the show()
         method */
22          card.show(this, "Card " + i);
23       }
24    }
25 }
```

Explanation:

The **CardLayoutTest** Applet features a card layout with five cards. Each card within the array of cards that Applet holds is assigned a static text message.

This is done using an array of Labels, specifying some text to be displayed on each label and then binding a label to a card. This is done by using add(). show() is used to display each element from the card on to the applet.

Being an applet, this class file needs to be embedded into an HTML, i.e. index.html.

The contents of an HTML: [index.html]

```
1  <!DOCTYPE html>
2  <html>
3     <head>
4        <title>Card Layout Test</title>
5     </head>
6     <body>
7        <applet code="CardLayoutTest.class" width=300 height=200></applet>
8     </body>
9  </html>
```

The HTML file is called through the browser or by passing the path and name of the HTML file to 'appletviewer' in the Command windows.

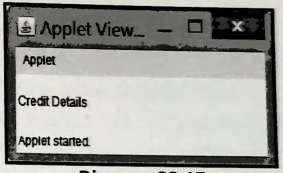

Diagram 23.17

REMINDER

Since one card in a Card Layout can be viewed at a time, the above diagram displays only the last card. To display each card in sequence, an interval period between the display of cards has to be specified.

The Book CDROM holds the complete application source code for the following applications:

- Codespecs \ Section_04 \ Chapter23_CDS \ **LabelAWT\LabelTest.java**
- Codespecs \ Section_04 \ Chapter23_CDS \ **LabelAWT\index.html**
- Codespecs \ Section_04 \ Chapter23_CDS \ **ButtonAWT\ButtonTest.java**
- Codespecs \ Section_04 \ Chapter23_CDS \ **ButtonAWT\index.html**
- Codespecs \ Section_04 \ Chapter23_CDS \ **CheckBoxAWT\CheckboxTest.java**
- Codespecs \ Section_04 \ Chapter23_CDS \ **CheckBoxAWT\index.html**