

Chapter 1...

Introduction to Abstract Windowing Toolkit (AWT) and Swings

Weightage of Marks = 24, Teaching Hours = 16

Contents

- 1.1 Introduction
 - 1.1.1 AWT Classes
- 1.2 Window Fundamentals (Component, Container, Window, Frame, Panel)
- 1.3 Creating Windowed Programs and Applets
- 1.4 AWT Controls and Layout Managers
 - 1.4.1 Understanding the Use of AWT Controls
 - 1.4.1.1 Labels
 - 1.4.1.2 Buttons
 - 1.4.1.3 CheckBox
 - 1.4.1.4 CheckBoxGroup
 - 1.4.1.5 Choice Controls
 - 1.4.1.6 Lists
 - 1.4.1.7 ScrollBars
 - 1.4.1.8 TextField
 - 1.4.1.9 TextArea
 - 1.4.1.10 Radio Buttons
 - 1.4.2 Understanding the Use of Layout Managers
 - 1.4.2.1 FlowLayout
 - 1.4.2.2 BorderLayout
 - 1.4.2.3 GridLayout
 - 1.4.2.4 CardLayout
 - 1.4.2.5 GridBagLayout
 - 1.4.3 Menu Bars and Menus
 - 1.4.3.1 Menu Bars
 - 1.4.3.2 Menus
 - 1.4.3.3 Pop-up Menus
 - 1.4.4 Dialog Boxes
 - 1.4.4.1 File Dialog

- 1.5 Introduction to Swing
 - 1.5.1 Swing Features
 - 1.5.2 Working with Swing
 - 1.5.3 Advantages and Disadvantages of Swing
 - 1.5.4 Model-View-Controller (MVC) Architecture
 - 1.5.5 Swing Components
 - 1.5.5.1 JFrame
 - 1.5.5.2 JPanel
 - 1.5.5.3 JApplet
 - 1.5.5.4 JDialog
 - 1.5.5.5 JLabel
 - 1.5.5.6 Icons
 - 1.5.5.7 JTextField
 - 1.5.5.8 JTextArea
 - 1.5.5.9 JButton
 - 1.5.5.10 JRadioButton
 - 1.5.5.11 JToggleButton
 - 1.5.5.12 JCheckBox
 - 1.5.5.13 JList
 - 1.5.5.14 Combo Boxes
 - 1.5.5.15 Progress Bar
 - 1.5.5.16 Tool Tips
 - 1.5.5.17 Separators (JSeparator)
 - 1.5.5.18 Tabbed Pane (JTabbedPane)
 - 1.5.5.19 JScrollPane
 - 1.5.5.20 JTree
 - 1.5.5.21 JTable

- Important Points
- Practice Questions

Objectives

- To Design and Develop Graphical User Interface (GUI) Programs using AWT and Swing Component
 - To arrange the GUI Components using different Layout Managers
-

1.1 INTRODUCTION

- Java AWT (Abstract Windowing Toolkit) is an API to develop GUI or window-based application in Java.
- Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavy weight i.e. its components uses the resources of system.
- The AWT is now part of the Java Foundation Classes (JFC) - the standard API for providing a Graphical User Interface (GUI) for a Java program.
- The main purpose for using the AWT is using for all the components displaying on the screen. AWT defines all the windows according to a class hierarchy those are useful at a specific level or we can say arranged according to their functionality.
- This chapter will help us to learn how graphics classes can be reused provided in JDK (Java Development Kit) for constructing our own (GUI) applications.
- There are two sets of java API for graphics programming i.e., AWT (Abstract Windowing Toolkit) and Swing.
 1. **AWT API** was introduced in JDK 1.0. Most of the AWT components have become obsolete and should be replaced by newer Swing components.
 2. **Swing API** a much more comprehensive set of graphics libraries that enhances the AWT, was introduced as part of Java Foundation Class (JFC) after the release of JDK 1.1. JFC, which consists of Swing, Java2D, Accessibility API, Internationalization, and Pluggable Look-and-Feel Support, was an add-on to JDK 1.1 but has been integrated into core java since JDK 1.2.
- Java applets are small .class files that run within a java enabled web browser. Most web browsers today have a built in JRE (Java Runtime Environment) that is either active by default or can be set as active.
- A web browser within its JRE set to active is a Java Enabled Web Browser. This is what makes an applet different from a standalone java program. A web browser and JDK's Applet viewer can be used to execute JavaApplet.class file.
- Java Graphics APIs (AWT and Swing) provide a huge set of reusable GUI components, such as button, text field, label, choice, panel and frame for building GUI applications. AWT is huge! It consists of 12 packages (Swing is even bigger, with 18 packages as of JDK 1.7!), but only two packages java.awt and java.awt.event - are commonly-used.
 1. The java.awt package contains the core AWT graphics classes:
 - GUI Component classes such as Button, TextField, and Label.
 - GUI Container classes such as Frame, Panel, Dialog and ScrollPane.
 - Layout managers such as FlowLayout, BorderLayout and GridLayout.
 - Custom graphics classes (such as Graphics, Color and Font).

2. The java.awt.event package supports event handling:
 - Event classes such as ActionEvent, MouseEvent, KeyEvent and WindowEvent.
 - Event Listener Interfaces such as ActionListener, MouseListener, KeyListener and WindowListener.
 - Event Listener Adapter classes such as MouseAdapter, KeyAdapter, and WindowAdapter.
- AWT provides a platform-independent and device-independent interface to develop graphic programs that runs on all platforms, such as Windows, Mac and Linux.

1.1.1 AWT Classes

- The AWT classes are contained in the java.awt package. It is one of Java's largest packages.
- Fortunately, because it is logically organized in a top-down, hierarchical fashion, it is easier to understand and use than you might at first believe.
- The hierarchy of Java AWT classes are given below:

Class	Description
AWTEvent	Encapsulates AWT events.
AWTEventMulticaster	Dispatches events to multiple listeners.
BorderLayout	The border layout manager. Border layouts use five components i.e. North, South, East, West and Centre.
Button	Creates a push button control.
Canvas	A blank, semantics-free window.
CardLayout	The card layout manager. Card layouts emulate indexcards. Only the one on top is showing.
Checkbox	Creates a check box control.
CheckboxGroup	Creates a group of check box controls.
CheckboxMenuItem	Creates an on/off menu item.
Choice	Creates a pop-up list.
Color	Manages colors in a portable, platform-independent fashion.
Component	An abstract super-class for various AWT components.

contd. ...

Container	A subclass of Component that can hold other components.
Cursor	Encapsulates a bitmapped cursor.
Dialog	Creates a dialog window.
Dimension	Specifies the dimensions of an object. The width is stored in width, and the height is stored in height.
Event	Encapsulates events.
EventQueue	Queues events.
FileDialog	Creates a window from which a file can be selected.
FlowLayout	The flow layout manager. Flow layout positions components left to right, top to bottom.
Font	Encapsulates a type font.
FontMetrics	Encapsulates various information related to a font.
Frame	Creates a standard window that has a title bar, resize corners, and a menu bar.
TextArea	Creates a multiline edit control.
TextComponent	A superclass for TextArea and TextField.
TextField	Creates a single-line edit control.
Toolkit	Abstract class implemented by the AWT.
Window	Creates a window with no frame, no menu bar.

1.2 WINDOW FUNDAMENTALS

(COMPONENT, CONTAINER, WINDOW, FRAME, PANEL)

- Window is a rectangular area which is displayed on the screen. In different window we can execute different program and display different data.
- Window provides us with multitasking environment. A window must have a frame, dialog or another window defined as its owner when it's constructed.

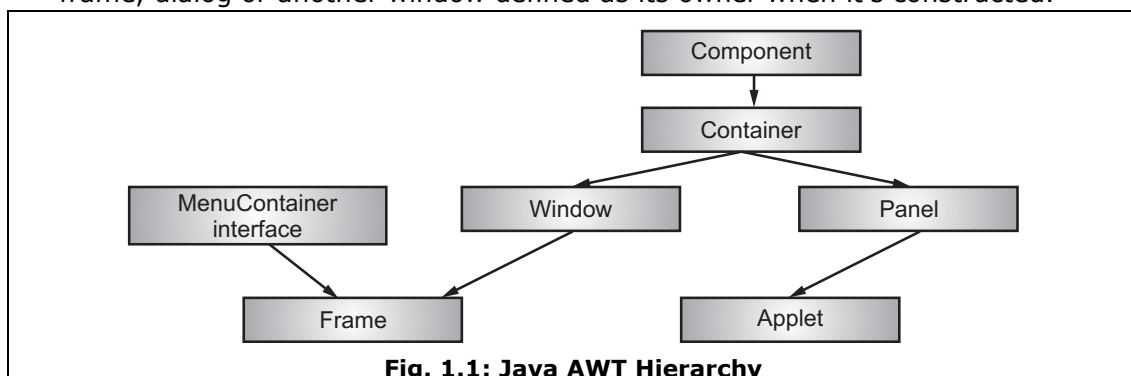


Fig. 1.1: Java AWT Hierarchy

- Fig. 1.1 shows following components of Window:
 1. **Component:** At the top of the AWT hierarchy is the Component class. Component is an abstract class that encapsulates all of the attributes of a visual component. All user interface elements that are displayed on the screen and that interact with the user are subclasses of Component. It defines over a hundred public methods that are responsible for managing events, such as mouse and keyboard input, positioning and sizing the window, and repainting. A Component object is responsible for remembering the current foreground and background colors and the currently selected text font.
 2. **Container:** The Container class is a subclass of Component. It has additional methods that allow other Component objects to be nested within it. Other Container objects can be stored inside of a Container (since, they are themselves instances of Component). This makes for a multileveled containment system. A container is responsible for laying out (that is, positioning) any components that it contains. It does this through the use of various layout managers.
 3. **Panel:** The Panel class is a concrete subclass of Container. It doesn't add any new methods; it simply implements Container. A Panel may be thought of as a recursively nestable, concrete screen component. Panel is the super-class for Applet. When screen output is directed to an applet, it is drawn on the surface of a Panel object. In essence, a Panel is a window that does not contain a title bar, menu bar, or border. This is why we don't see these items when an applet is run inside a browser. When we run an applet using an applet viewer, the applet viewer provides the title and border. Other components can be added to a Panel object by its add() method (inherited from Container). Once, these components have been added, we can position and resize them manually using the setLocation(), setSize(), or setBounds() methods defined by Component.
 4. **Window:** The Window class creates a top-level window. A top-level window is not contained within any other object; it sits directly on the desktop. Generally, we won't create Window objects directly. Instead, we will use a subclass of Window called Frame.
 5. **Frame:** Frame encapsulates what is commonly thought of as a "window." It is a subclass of Window and has a title bar, menu bar, borders, and resizing corners. If we create a Frame object from within an applet, it will contain a warning message, such as "Java Applet Window," to the user that an applet window has been created. This message warns users that the window they see was started by an applet and not by software running on their computer. When a Frame window is created by a program rather than an applet, a normal window is created.
 6. **Canvas:** Canvas encapsulates a blank window upon which we can draw.

Working with Frame Window:

- A Frame is a top-level window with a title and a border.
- The type of window you will be creating for stand alone application is subclass of Frame.
- Two of Frame's constructors:
 1. `Frame()`: The first form creates a standard window that does not contain a title.
 2. `Frame(String title)`: The second form creates a window with the title specified by title.

Setting the Windows Dimensions:

- The `setSize()` method is used to set the size of the window.
- It's general structure/syntax is shown here:

```
void setSize(int newWidth, int newHeight)
void setSize(Dimension newSize)
```
- The new size of the Frame is specified by `newWidth` and `newHeight` or by the width and height fields of the `Dimension` object passed in `newSize`. The dimensions are specified in terms of pixels.
- The `getSize()` method is used to obtain the current size of a window. Its general syntax is,

```
Dimension getSize( )
```
- This method returns the current size of the Frame as the width and height fields of a `Dimension` object.

Showing and Hiding Frame:

- After a frame has been created, it will not be visible until you call `setVisible()` method. Its general structure/syntax is shown here:

```
void setVisible(boolean visibleFlag)
```
- The Frame is visible if the argument to this method is true. Otherwise, it is hidden.

Setting a Frame's Title:

- You can set the title of a frame window using `setTitle()`, which has this general form:

```
void setTitle(String newTitle)
```

Program 1.1: Frame Demo.

```
import java.awt.*;

class MyFrame extends Frame
{
    MyFrame(String title)
    {
        setVisible(true);
        setSize(100,100);
        setTitle(title);
    }

    public static void main(String args[])
    {
        MyFrame f= new MyFrame("My Frame");
    }
}
```

Closing a Frame Window:

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

1.3 CREATING WINDOWED PROGRAMS AND APPLETS

- Although creating applets is the most common use for Java's AWT, it is possible to create stand-alone AWT-based applications, too. To do this, simply we need to create an instance of the window or windows inside `main()`. For example, the following program creates a simple frame window.
- A Frame provides the "main window" for the GUI application, which has a title bar (containing an icon, a title, the minimize, maximize/restore-down and close buttons), an optional menu bar, and the content display area.

- To write a GUI program, we typically start with a subclass extending from `java.awt.Frame` to inherit the main window as follows:

```
import java.awt.*;
class myFrame extends Frame
{
    myFrame(String s)
    {
        super(s);
        setVisible(true);
        setSize(500,300);
    }
    public static void main(String[] args)
    {
        myFrame f= new myFrame("Demonstrating Frame");
    }
}
```

Frame Window in an Applet:

- It is also possible to create a Frame Window within an applet window.
 - The following applet (Program 1.2) creates a subclass of `Frame` called `MyFrame`. A window of this subclass is instantiated within the `init()` method of `AppletFrame`. Note that `MyFrame` calls `Frame`'s constructor. This causes a standard frame window to be created with the title passed in `title`.
 - This example overrides the applet window's `start()` and `stop()` methods so that they show and hide the child window, respectively. It also causes the child window to be shown when the browser returns to the applet.
-

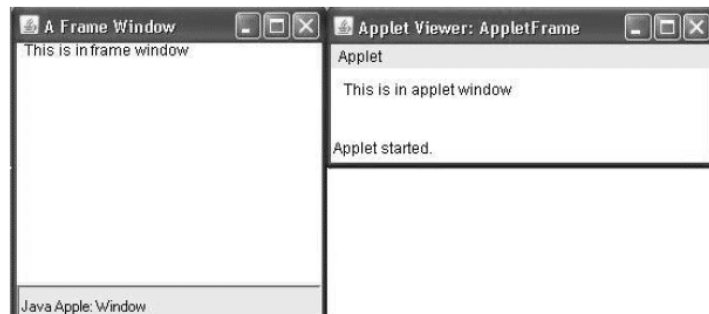
Program 1.2: Frame in Applet.

```
import java.awt.*;
import java.applet.*;
class SampleFrame extends Frame
{
    SampleFrame(String title)
    {
        super(title);
    }
    public void paint(Graphics g)
    {
        g.drawString("This is in frame window", 10, 40);
    }
}
```

```
public class AppletFrame extends Applet
{
    SampleFrame f;
    public void init()
    {
        f = new SampleFrame("A Frame Window");
        f.setSize(250, 250);
        f.setVisible(true);
    }
    public void start()
    {
        f.setVisible(true);
    }
    public void stop()
    {
        f.setVisible(false);
    }
    public void paint(Graphics g)
    {
        g.drawString("This is in applet window", 10, 20);
    }
}
/* <applet code="AppletFrame" width=300 height=50>
</applet> */
```

Output:

Output:



Displaying Information Within a Window:

- A window is a container for information. One of the important advantage of a window is the ability to present a high quality text and graphics.
-

Program 1.3: Displaying information within window.

```
import java.awt.*;
class MyFrameDemo extends Frame
{
    public void paint(Graphics g)
    {
        setForeground(Color.red);
        setBackground(Color.cyan);
        g.drawString("This is my frame", 130, 170);
    }
    public static void main(String[] args)
    {
        MyFrameDemo f= new MyFrameDemo();
        f.setSize(500,400);
        f.setVisible(true);
    }
}
```

1.4 AWT CONTROLS AND LAYOUT MANAGERS

- Controls are components that allow a user to interact with his/her application in various ways—for example; a commonly used control is the push button.
- A layout manager automatically positions components within a container. Thus, the appearance of a window is determined by a combination of the controls that it contains and the layout manager used to position them.
- In addition to the controls, a frame window can also include a standard-style menu bar. Each entry in a menu bar activates a drop-down menu of options from which the user can choose. A menu bar is always positioned at the top of a window.
- Although different in appearance, menu bars are handled in much the same way as are the other controls. While it is possible to manually position components within a window, doing so is quite tedious. The layout manager automates this task.

Adding and Removing Controls:

- In order to include a control in a window, we must add it to the window. So, we must first create an instance of the desired control and then add it to a window by calling `add()`, which is defined by `Container`.
- The `add()` method has several forms. The following form/syntax is the one that is used for the first part of this chapter:

```
Component add(Component compObj)
```

- Here, `compObj` is an instance of the control that we want to add. A reference to `compObj` is returned. Once, a control has been added, it will automatically be visible whenever its parent window is displayed.
- Sometimes, we will want to remove a control from a window when the control is no longer needed. For doing this, call `remove()`.

- This method is also defined by `Container`. It has following general form/syntax:

```
void remove(Component obj)
```

- Here, `obj` is a reference to the control that we want to remove. We can remove all controls by calling `removeAll()`.

1.4.1 Understanding the Use of AWT Controls

- Controls are the components that allow a user to interact with java application in various ways.
- Controls are the most common ways that users give instructions to Java programs.
- The AWT supports the following types of controls:
 1. Labels
 2. Push buttons
 3. Check boxes
 4. Choice lists
 5. Lists
 6. Scroll bars
 7. Text Area
 8. Text Field
- Above controls are subclasses of `Component`.

Note: It is also possible to add controls to applet window also, so some controls are demonstrated using frame window and some using applet window.

1.4.1.1 Labels

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

- Label defines the following constructors:
 1. `Label()`: Creates a blank/empty label.
 2. `Label(String str)`: Creates a label that contains the string specified by `str`. This string is left-justified.
 3. `Label(String str, int how)`: Creates a label that contains the string specified by `str` using the alignment specified by `how`. The value of `how` must be one of these three constants:
 - (i) `Label.LEFT`,
 - (ii) `Label.RIGHT`, and
 - (iii) `Label.CENTER`.
- We can set or change the text in a label by using the `setText()` method. We can obtain the current label by calling `getText()`.
- Syntax for `setText()`:

```
void setText(String str)
```
- Syntax for `getText()`:

```
String getText( )
```
- For `setText()`, `str` specifies the new label. For `getText()`, the current label is returned.
- You can set the alignment of the string within the label by calling `setAlignment()`.
Syntax: `void setAlignment(int how)`
- To obtain the current alignment, call `getAlignment()` method.
Syntax: `int getAlignment()`
Here, `how` must be one of the alignment constants shown earlier.
- The following example creates three labels and adds them to frame window.

Program 1.4: Program to demonstrate label.

```
import java.awt.*;
class myFrame extends Frame
{
    myFrame(String s)
    {
        super(s);
        setVisible(true);
        setSize(500,300);
        Label one = new Label("Label One");
        Label two = new Label("Label Two");
        Label three = new Label("Label Three");
```

```
        // add labels to applet window
        add(Label one);
        add(Label two);
        add(Label three);
    }
    public static void main(String[] args)
    {
        myFrame f= new myFrame("Demonstrating Frame");
    }
}
```

1.4.1.2 Buttons

- The most widely used control is the push button.
- A push button is a component that contains a label and that generates an event when it is pressed.
- Push buttons are objects of type Button. Button defines these two constructors:
 1. Button(): Creates an empty button.
 2. Button(String str): Creates a button that contains str as a label.
- After a button has been created, we can set its label by calling setLabel(). We can retrieve its label by calling getLabel().

Syntax: void setLabel(String str)

- Here, str becomes the new label for the button.

Syntax: String getLabel()

Program 1.5: Program to demonstrate buttons.

```
import java.awt.*;
class myFrame extends Frame
{
    Button ok, cancel;
    myFrame(String s)
    {
        super(s);
        setVisible(true);
        setSize(500,300);
        ok = new Button("OK");
        cancel = new Button("Cancel");
        add(ok);
        add(cancel);
    }
    public static void main(String[] args)
    {
        myFrame f= new myFrame("Demonstrating Frame");
    }
}
```

1.4.1.3 CheckBox

- A check box is a control that is used to turn an option on (true) or off (false).
- It consists of a small box that can either contain a check mark or not.
- There is a label associated with each check box that describes what option the box represents. We change the state of a check box by clicking on it.
- Check boxes can be used individually or as part of a group. Check boxes are objects of the `Checkbox` class.
- `Checkbox` supports following constructors:
 1. `Checkbox()`: Creates a check box whose label is initially blank.
 2. `Checkbox(String str)`: Creates a check box whose label is specified by `str`. The state of the check box is unchecked.
 3. `Checkbox(String str, boolean on)`: Allows us to set the initial state of the check box. If `on` is true, the check box is initially checked; otherwise, it is cleared.
 4. `Checkbox(String str, boolean on, CheckboxGroup cbGroup)`: Constructs a `Checkbox` with the specified label, set to the specified state, and in the specified check box group.
 5. `Checkbox(String str, CheckboxGroup cbGroup, boolean on)`: Create a check box whose label is specified by `str` and whose group is specified by `cbGroup`. If this check box is not part of a group, then `cbGroup` must be null.
- The value of `on` determines the initial state of the check box. In order to retrieve the current state of a check box, call `getState()`.
Syntax: `boolean getState()`
- For setting its state, call `setState()`.
Syntax: `void setState(boolean on)`
- We can obtain the current label associated with a check box by calling `getLabel()`.
Syntax: `String getLabel()`
- For setting the label, `setLabel()` is used.
Syntax: `void setLabel(String str)`
- Here, if `on` is true, the box is checked. If it is false, the box is cleared. The string passed in `str` becomes the new label associated with the invoking checkbox.

Program 1.6: Program to demonstrate checkboxes.

```
import java.awt.*;
import java.applet.*;
public class CheckboxDemo extends Applet
{
String msg = "";
Checkbox Win98, winNT, solaris, mac;
public void init()
{
Win98 = new Checkbox("Windows 98/XP", null, true);
winNT = new Checkbox("Windows NT/2000");
solaris = new Checkbox("Solaris");
mac = new Checkbox("MacOS");
add(Win98);
add(winNT);
add(solaris);
add(mac);
}
public void paint(Graphics g)
{
}
}
/*<applet code="CheckboxDemo" width=250 height=200>
</applet>*/
```

1.4.1.4 CheckBoxGroup

- The CheckBoxGroup class is used to group the set of check box.
- It is possible to create a set of mutually exclusive check boxes in which one and only one check box in the group can be checked at any one time.
- These check boxes are often called radio buttons, because they act like the station selector on a car radio—only one station can be selected at any onetime.
- For creating a set of mutually exclusive check boxes, we must first define the group to which they will belong and then specify that group when we construct the check boxes.
- Check box groups are objects of type CheckBoxGroup. Only the default constructor is defined, which creates an empty group.

- We can determine which check box in a group is currently selected by calling `getSelectedCheckbox()`.
Syntax: `Checkbox getSelectedCheckbox()`
- We can set a check box by calling `setSelectedCheckbox()`.
Syntax: `void setSelectedCheckbox(Checkbox wh)`
- Here, wh i.e. which is the check box that we want to be selected. The previously selected check box will be turned off. Here is a program that uses check boxes that are part of a group:

Program 1.7: Program to demonstrate Checkbox group.

```
import java.awt.*;
class myFrame extends Frame
{
    Checkbox Win98, winNT, solaris, mac;
    CheckboxGroup cbg;
    myFrame(String s)
    {
        super(s);
        setVisible(true);
        setSize(500,300);
        cbg = new CheckboxGroup();
        Win98 = new Checkbox("Windows 98/XP", cbg, true);
        winNT = new Checkbox("Windows NT/2000", cbg, false);
        solaris = new Checkbox("Solaris", cbg, false);
        mac = new Checkbox("MacOS", cbg, false);
        add(Win98);
        add(winNT);
        add(solaris);
        add(mac);
    }
    public void paint(Graphics g)
    {
        msg = "Current selection: ";
        msg += cbg.getSelectedCheckbox().getLabel();
        g.drawString(msg, 6, 100);
    }
    public static void main(String[] args)
    {
        myFrame f= new myFrame("Demonstrating Frame");
    }
}
```

1.4.1.5 Choice Controls

- The Choice class is used to create a pop-up list of items from which the user may choose. Thus, a Choice control is a form of menu.
- When inactive, a Choice component takes up only enough space to show the currently selected item. When the user clicks on it, the whole list of choices pops up, and a new selection can be made. Each item in the list is a string that appears as a left justified label in the order it is added to the Choice object.
- Choice only defines the default constructor, which creates an empty list. In order to add a selection to the list, `add()` is used.

Syntax: `void add(String name)`

- Here, `name` is the name of the item being added. Items are added to the list in the order in which calls to `add()` occur.
- In order to determine which item is currently selected, we may call either any of the following methods:
- The `getSelectedItem()` method returns a string containing the name of the item.

Syntax: `String getItem()`

- The `getSelectedItemIndex()` returns the index of the item.

Syntax: `int getItemIndex()`

- The first item is at index 0. By default, the first item added to the list is selected.
- For obtaining the number of items in the list, call `getItemCount()`.

Syntax: `int getItemCount()`

- We can set the currently selected item using the `select()` method with either a zero-based integer index or a string that will match a name in the list. These methods are shown here,

Syntax: `void select(int index)`

And

`void select(String name)`

- Given an index, we can obtain the name associated with the item at that index by calling `getItem()`, which has following general form syntax:

`String getItem(int index)`

Here, `index` specifies the index of the desired item.

Program 1.8: Program to demonstrate choice controls.

```
import java.awt.*;
import java.applet.*;
public class ChoiceDemo extends Applet
{
    Choice os, browser;
    String msg = "";
    public void init()
    {
        os = new Choice();
        browser = new Choice();
        os.add("Windows 98/XP");
        os.add("Windows NT/2000");
        os.add("Solaris");
        os.add("MacOS");
        browser.add("Netscape 6.x");
        browser.add("Internet Explorer 6.0");
        browser.add("Linux 2.4");
        browser.add("Google chrome");
        browser.add("Mozilla");
        browser.select("Netscape 6.x");
        add(os);
        add(browser);
    }
    public void paint(Graphics g)
    {
    }
}
/*<applet code="ChoiceDemo" width=300 height=180></applet>*/
```

1.4.1.6 Lists

- The List class provides a compact, multiple-choice, scrolling selection list.
- Unlike the Choice object, which shows only the single selected item in the menu, a List object can be constructed to show any number of choices in the visible Window. It can also be created to allow multiple selections.

- List provides following constructors:
 1. `List()`: Creates a List control that allows only one item to be selected at any one time.
 2. `List(int numRows)`: The value of `numRows` specifies the number of entries in the list that will always be visible (others can be scrolled into view as needed).
 3. `List(int numRows, boolean multipleSelect)`: If `multipleSelect` is true, then the user may select two or more items at a time. If it is false, then only one item may be selected.
- For adding a selection to the list, we can call `add()`. It has the following two forms:

```
void add(String name)
```

OR

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

Here, `name` is the name of the item added to the list. The first form adds items to the end of the list. The second form adds the item at the index specified by `index`. Indexing begins at zero. We can specify `-1` to add the item to the end of the list.

- For lists that allow only single selection, we can determine which item is currently selected by calling either `getSelectedItem()` or `getSelectedIndex()` methods.
- The `getSelectedItem()` method returns a string containing the name of the item. If more than one item is selected or if no selection has yet been made, null is returned.

Syntax: `String getItem()`

- The `getSelectedIndex()` returns the index of the item. The first item is at index 0. If more than one item is selected, or if no selection has yet been made, `-1` is returned.

Syntax: `int getSelectedIndex()`

- For lists that allow multiple selection, we must use either `getSelectedItems()` or `getSelectedIndexes()`.
- The `getSelectedItems()` returns an array containing the names of the currently selected items.

Syntax: `String[] getSelectedItems()`

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

Syntax: `int[] getSelectedIndexes()`

- In order to obtain the number of items in the list, call `getItemCount()`.

- We can set the currently selected item by using the `select()` method with a zero-based integer index.
- The `getItemCount()` used to obtain the number of items in the list.

Syntax: `int getItemCount()`

- The `select(int index)` method is used to set the currently selected item.

Syntax: `void select(int index)`

- Given an index, we can obtain the name associated with the item at that index by calling `getItem()`, which has following general form:

Syntax: `String getItem(int index)`

Here, `index` specifies the index of the desired item.

Program 1.9: Program to demonstrate lists.

```
import java.awt.*;
class myFrame extends Frame
{
    List l;
    myFrame(String s)
    {
        super(s);
        setVisible(true);
        setSize(500,300);
        l= new List(3);
        l.add("India");
        l.add("America");
        l.add("Australia");
        l.add("Pakistan");
        add(l);
    }
    public static void main(String[] args)
    {
        myFrame f= new myFrame("Demonstrating Frame");
    }
}
```

1.4.1.7 Scroll Bars

- Scrollbar control represents a scroll bar component in order to enable user to select from range of values.
- Scrollbars allows users to scroll the components. Scroll bars are used to select continuous values between a specified minimum and maximum. Scroll bars may be oriented horizontally or vertically.
- A scroll bar is actually a composite of several individual parts. Each end has an arrow that we can click to move the current value of the scroll bar one unit in the direction of the arrow.
- The current value of the scroll bar relative to its minimum and maximum values is indicated by the slider box (or thumb) for the scroll bar.
- The slider box can be dragged by the user to a new position. The scroll bar will then reflect this value. In the background space on either side of the thumb, the user can click to cause the thumb to jump in that direction by some increment larger than 1.
- Typically, this action translates into some form of page up and page down. Scroll bars are encapsulated by the Scrollbar class.
- Scrollbar defines the following constructors:
 1. `Scrollbar()`: Creates a vertical scroll bar.
 2. `Scrollbar(int style)`: Allow us to specify the orientation of the scroll bar. If `style` is `Scrollbar.VERTICAL`, a vertical scroll bar is created. If `style` is `Scrollbar.HORIZONTAL`, the scroll bar is horizontal.
 3. `Scrollbar(int style, int iValue, int tSize, int min, int max)`: In this constructor, the initial value of the scroll bar is passed in `iValue`. The number of units represented by the height of the thumb is passed in `tSize`.
- The minimum and maximum values for the scroll bar are specified by `min` and `max`. If we construct a scroll bar by using one of the first two constructors, then we need to set its parameters by using `setValues()`, shown here, before it can be used:

```
void setValues(int iValue, int tSize, int min, int max)
```

- The parameters have the same meaning as they have in the third constructor just described. In order to obtain the current value of the scroll bar, call `getValue()`. It returns the current setting. For setting the current value, we can use `setValue()`. These methods are as follows:

Syntax: `int getValue()`

OR

```
void setValue(int newValue)
```

Here, `newValue` specifies the new value for the scroll bar. When we set a value, the slider box inside the scroll bar will be positioned to reflect the new value.

- We can also retrieve the minimum and maximum values via `getMinimum()` and `getMaximum()` methods:

Syntax: `int getMinimum()`

And

`int getMaximum()`

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

Syntax: `void setUnitIncrement(int newIncr)`

- You can change this value by calling `setBlockIncrement()`.

Syntax: `void setBlockIncrement(int newIncr)`

Program 1.10: Program to demonstrate scroll bar.

```
import java.awt.*;
import java.applet.*;
public class SBDemo extends Applet
{
    String msg = "";
    Scrollbar vertSB, horzSB;
    public void init()
    {
        int width = Integer.parseInt(getParameter("width"));
        int height = Integer.parseInt(getParameter("height"));
        vertSB = new Scrollbar(Scrollbar.VERTICAL,0, 1, 0, height);
        horzSB = new Scrollbar(Scrollbar.HORIZONTAL,0, 1, 0, width);
        add(vertSB);
        add(horzSB);
    }
    public void paint(Graphics g)
    {
        msg = "Vertical: " + vertSB.getValue();
        msg += ", Horizontal: " + horzSB.getValue();
        g.drawString(msg, 6, 160);
    }
}
/*<applet code="SBDemo" width=300 height=200>
</applet>*/
```

1.4.1.8 TextField

- The `TextField` class implements a single-line text-entry area, usually called an edit control.
- Text fields allow the user to enter strings and to edit the text using the arrow keys, cut and paste keys, and mouse selections.
- `TextField` is a subclass of `TextComponent`. `TextField` defines the following constructors:
 1. `TextField()`: Creates a default text field.
 2. `TextField(int numChars)`: Creates a text field that is `numChars` characters wide.
 3. `TextField(String str)`: Initializes the textfield with the string contained in `str`.
 4. `TextField(String str, int numChars)`: Initializes a text field and sets its width.
- `TextField` (and its superclass `TextComponent`) provides several methods that allow us to utilize a text field.
- In order to obtain the string currently contained in the text field, we can use `getText()`.
Syntax: `String getText()`
- For setting the text, we call `setText()`.
Syntax: `void setText(String str)`
Here, `str` is the new string. The user can select a portion of the text in a text field.
- Also, we can select a portion of text under program control by using `select()`.
- Our program can obtain the currently selected text by calling the `getSelectedText()`.
Syntax: `String getSelectedText()`
- The `getSelectedText()` returns the selected text. The `select()` method selects the characters beginning at `startIndex` and ending at `endIndex-1`.
Syntax: `void select(int startIndex, int endIndex)`
- We can control whether the contents of a text field may be modified by the user by calling `setEditable()`.
Syntax: `void setEditable(boolean canEdit)`
- We can determine editability by calling `isEditable()`.
Syntax: `boolean isEditable()`
- The `isEditable()` returns true if the text may be changed and false if not.

- In `setEditable()`, if `canEdit` is true, the text may be changed. If it is false, the text cannot be altered. There may be times when we will want the user to enter text that is not displayed, such as a password.
- You can disable the echoing of the characters as they are typed by calling `setEchoChar()`. This method specifies a single character that the `TextField` will display when characters are entered (thus, the actual characters typed will not be shown).

Syntax: `void setEchoChar(char ch)`

Here, `ch` specifies the character to be echoed.

- We can check a text field to see if it is in this mode with the `echoCharIsSet()` method.

Syntax: `boolean echoCharIsSet()`

- We can retrieve the echo character by calling the `getEchoChar()` method.

Syntax: `char getEchoChar()`

Program 1.11: Program to demonstrate textfield.

```
import java.awt.*;
class myFrame extends Frame
{
    TextField name, pass;
    myFrame(String s)
    {
        super(s);
        setVisible(true);
        setSize(500,300);
        Label namep = new Label("Name: ", Label.RIGHT);
        Label passp = new Label("Password: ", Label.RIGHT);
        name = new TextField(12);
        pass = new TextField(8);
        pass.setEchoChar('?');
        add(namep);
        add(name);
        add(passp);
        add(pass);
    }
    public static void main(String[] args)
    {
        myFrame f= new myFrame("Demonstrating Frame");
    }
}
```

1.4.1.9 TextArea

- Sometimes, a single line of text input is not enough for a given task. To handle these situations, the AWT includes a simple multi-line editor called TextArea.
- Following are the constructors for TextArea:
 1. `TextArea()`: Constructs a new text area with the empty string.
 2. `TextArea(int numLines, int numChars)`: Here, `numLines` specifies the height, in lines, of the text area; and `numChars` specifies its width, in characters.
 3. `TextArea(String str)`: Initial text can be specified by `str`.
 4. `TextArea(String str, int numLines, int numChars)`: Specify the scroll bars that we want the control to have.
 5. `TextArea(String str, int numLines, int numChars, int sBars)`: We can `sBars` must be one of these values:

`SCROLLBARS_BOTH`

`SCROLLBARS_NONE`

`SCROLLBARS_HORIZONTAL_ONLY`

`SCROLLBARS_VERTICAL_ONLY`

- `TextArea` is a subclass of `TextComponent`. Therefore, it supports the `getText()`, `setText()`, `getSelectedText()`, `select()`, `isEditable()`, and `setEditable()` methods described in the preceding section.
- `TextArea` adds the following methods:
- The `append()` method appends the string specified by `str` to the end of the current text.

Syntax: `void append(String str)`
- The `insert()` inserts the string passed in `str` at the specified index.

Syntax: `void insert(String str, int index)`
- In order to replace text, we call `replaceRange()`. It replaces the characters from `startIndex` to `endIndex-1`, with the replacement text passed in `instr`. Text areas are almost self-contained controls.

Syntax: `void replaceRange(String str, int startIndex, int endIndex)`

Program 1.12: Program to demonstrate textarea.

```
import java.awt.*;
import java.applet.*;
public class TextAreaDemo extends Applet
{
    public void init()
    {
        String val = "There are two ways of constructing " +
            "a software design.\n" +
            "One way is to make it so simple\n" +
            "that there are obviously no deficiencies.\n" +
            "And the other way is to make it so complicated\n" +
            "that there are no obvious deficiencies.\n\n";
        TextArea text = new TextArea(val, 10, 30);
        add(text);
    }
}
/*<applet code="TextAreaDemo" width=300 height=250>
</applet>*/
```

1.4.1.10 Radio Buttons

- Radio Buttons are checkboxes in which we can check only one at a time from the group. So, for that we have to group the individual checkboxes.

Methods:

1. `getSelectedCheckbox()`: To determine which checkbox in a group is selected.

Syntax: `Checkbox getSelectedCheckbox()`

2. `setSelectedCheckbox()`: To set a checkbox as selected.

Syntax: `void setSelectedCheckbox(Checkbox ckb)`

Program 1.13: Program to demonstrate radio buttons.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*<applet code="radiodemo" width=250 height=150>
</applet>
*/
```

```
public class radiodemo extends Applet
{
    Checkbox m, f;
    CheckboxGroup cbg;
    public void init()
    {
        cbg=new CheckboxGroup();
        m=new Checkbox("male",cbg,true);
        f=new Checkbox("female",cbg,false);
        add(m);
        add(f);
    }
}
```

1.4.2 Understanding the Use of Layout Managers

- Layout means the arrangement of components within the container. In other way we can say that placing the components at a particular position within the container.
- The task of layouting the controls are done automatically by the Layout Manager.
- All of the components that we have shown so far have been positioned by the default layout manager. A layout manager automatically arranges our controls within a window by using some type of algorithm.
- If we have programmed for other GUI environments, such as Windows, then we are accustomed to laying out our controls by hand. While it is possible to layout Java controls by hand, too, we generally won't want to, for two main reasons.
 1. First, it is very tedious to manually lay out a large number of components.
 2. Sometimes, the width and height information is not yet available when we need to arrange some control, because the native toolkit components haven't been realized. This is a chicken-and-egg situation; it is pretty confusing to figure out when it is okay to use the size of a given component to position it relative to another. Each Container object has a layout manager associated with it.
- A layout manager is an instance of any class that implements the `LayoutManager` interface.
- The layout manager is set by the `setLayout()` method. If no call to `setLayout()` is made, then the default layout manager is used.

- Whenever, a container is resized (or sized for the first time), the layout manager is used to position each of the components within it.
- The `setLayout()` method has the following general form:

Syntax: `void setLayout(LayoutManager layoutObj)`

Here, `layoutObj` is a reference to the desired layout manager. If we wish to disable the layout manager and position components manually, pass null for `layoutObj`.

- If we do this, we will need to determine the shape and position of each component manually, using the `setBounds()` method defined by `Component`. Normally, we will want to use a layout manager.
- Each layout manager keeps track of a list of components that are stored by their names. The layout manager is notified each time we add a component to a container.
- Whenever, the container needs to be resized, the layout manager is consulted via its minimum `LayoutSize()` and preferred `LayoutSize()` methods.
- Each component that is being managed by a layout manager contains the `getPreferredSize()` and `getMinimumSize()` methods. These return the preferred and minimum size required to display each component.
- The layout manager will honor these requests if at all possible, while maintaining the integrity of the layout policy. We may override these methods for controls that we subclass.
- The layout manager automatically positions all the components within the container. Default values are provided otherwise.
- Java has several predefined Layout Manager Classes. Layout manager classes are founded in the `java.awt` package same as the classes used for displaying graphical components.
- Several layout managers of which are described next sections. We can use the layout manager that best fits our application.
- The `java.awt` package provides five layout manager classes as given below:
 1. `FlowLayout`: Arranges components in variable-length rows.
 2. `BorderLayout`: Arranges components along the sides of the container and in the middle.
 3. `CardLayout`: Arrange components in "cards" Only one card is visible at a time.
 4. `GridBagLayout`: Aligns components horizontally and vertically; components can be of different sizes.
 5. `GridLayout`: Arranges components in fixed-length rows and columns.

1.4.2.1 FlowLayout

- FlowLayout is the default layout manager. This is the layout manager that the preceding examples have used.
- FlowLayout implements a simple layout style, which is similar to how words flow in a text editor.
- Components are laid out from the upper-left corner, left to right and top to bottom. When no more components fit on a line, the next one appears on the next line. A small space is left between each component, above and below, as well as left and right.
- Here, are the constructors for FlowLayout:
 1. `FlowLayout()`: Creates the default layout, which centres components and leaves five pixels of space between each component.
 2. `FlowLayout(int how)`: Specify how each line is aligned. Valid values for how are as follows:

```
FlowLayout.LEFT
```

```
FlowLayout.CENTER
```

```
FlowLayout.RIGHT
```

- These values specify left, center, and right alignment, respectively.
- We can change this with the `setAlignment()` method shown in Fig. 1.2.
- For example, we might use this line of code:

```
layout.setAlignment(FlowLayout.LEFT);
```


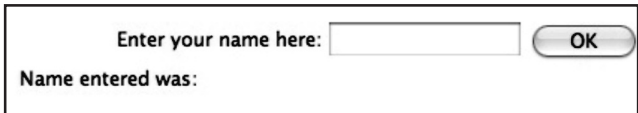
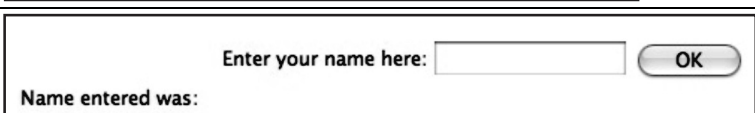
Alignment	Applet
<code>FlowLayout.LEFT</code>	 <p>Enter your name here: <input type="text"/> <input type="button" value="OK"/></p> <p>Name entered was:</p>
<code>FlowLayout.CENTER</code>	 <p>Enter your name here: <input type="text"/> <input type="button" value="OK"/></p> <p>Name entered was:</p>
<code>FlowLayout.RIGHT</code>	 <p>Enter your name here: <input type="text"/> <input type="button" value="OK"/></p> <p>Name entered was:</p>

Fig. 1.2

3. `FlowLayout(int how, int horz, int vert)`: Allows us to specify the horizontal and vertical space left between components in `horz` and `vert`, respectively.
- Here, is a version of the CheckboxDemo applet shown earlier, modified so that it uses left-aligned flowlayout.

Program 1.14: Program to demonstrate flow layout manager.

```
import java.awt.*;
class myFrame extends Frame
{
    TextField name, pass;
    myFrame(String s)
    {
        super(s);
        setVisible(true);
        setSize(500,300);
        setLayout(new FlowLayout());
        Label namep = new Label("Name: ", Label.RIGHT);
        Label passp = new Label("Password: ", Label.RIGHT);
        name = new TextField(12);
        pass = new TextField(8);
        pass.setEchoChar('?');
        add(namep);
        add(name);
        add(passp);
        add(pass);
    }
    public static void main(String[] args)
    {
        myFrame f= new myFrame("Demonstrating Frame");
    }
}
```

1.4.2.2 BorderLayout

- The BorderLayout class implements a common layout style for top-level windows.
- It has four narrow, fixed-width components at the edges and one large area in the center. The four sides are referred to as north (upper region), south (lower region), east (left region), and west (right region). The middle area is called the center (central region).

- Five zones, one for each component, as can be seen in Fig. 1.3.

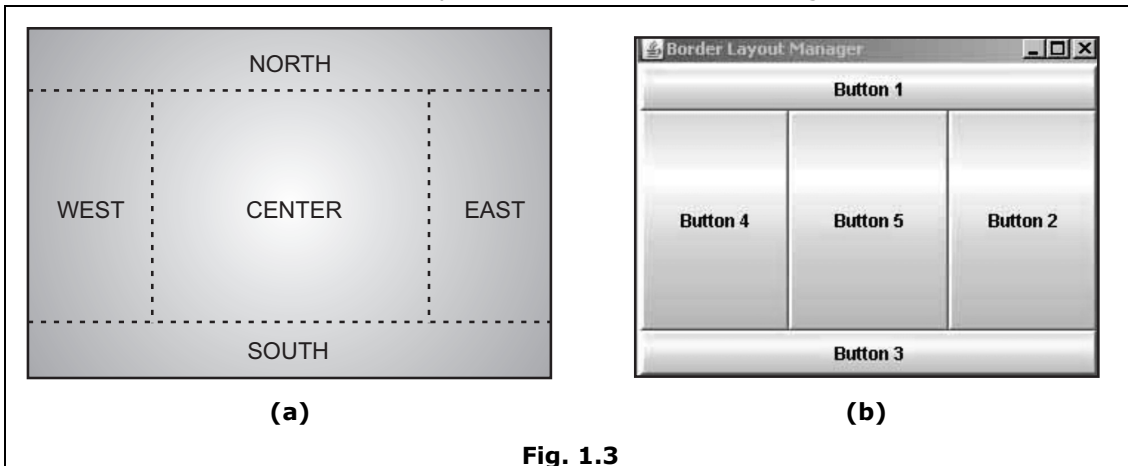


Fig. 1.3

- Here, are the constructors defined by BorderLayout:
 1. `BorderLayout()`: Creates a default border layout.
 2. `BorderLayout(int horz, int vert)`: Allows us to specify the horizontal and vertical space left between components in `horz` and `vert`, respectively.
 BorderLayout defines the following constants that specify the regions:


```

      BorderLayout.CENTER
      BorderLayout.SOUTH
      BorderLayout.EAST
      BorderLayout.WEST
      BorderLayout.NORTH
      
```
- When adding components, we will use these constants with the following form of `add()`, which is defined by Container:


```

      void add(Component compObj, Object region);
      
```
- Here, `compObj` is the component to be added, and `region` specifies where the component will be added.

Program 1.15: Program to demonstrate border layout.

```

import java.awt.*;
class myFrame extends Frame
{
    TextField name, pass;
    Button E,W,S,N,C;
    BorderLayout B1;
    myFrame(String s)
    {
        super(s);
        setVisible(true);
    }
}

```

```
setSize(500,300);
Bl=new BorderLayout();
setLayout(Bl);
E=new Button("EAST");
W=new Button("WEST");
S=new Button("SOUTH");
N=new Button("NORTH");
C=new Button("CENTER");
add(E, BorderLayout.EAST);
add(W, BorderLayout.WEST);
add(S, BorderLayout.SOUTH);
add(N, BorderLayout.NORTH);
add(C, BorderLayout.CENTER);
}
public static void main(String[] args)
{
    myFrame f= new myFrame("Demonstrating Frame");
}
}
```

1.4.2.3 GridLayout

- GridLayout lays out components in a two-dimensional grid. When we instantiate a GridLayout, we define the number of rows and columns.
- The Grid layout manager places items in rows (left to right) and columns (top to bottom). The number of rows and columns is defined when the Grid layout manager is created.



Fig. 1.4

- The constructors supported by GridLayout are shown here:
 1. `GridLayout()`: Creates a single-column grid layout.
 2. `GridLayout(int numRows, int numColumns)`: Creates a grid layout with the specified number of rows and columns.
 3. `GridLayout(int numRows, int numColumns, int horz, int vert)`: Allows us to specify the horizontal and vertical space left between components in `horz` and `vert`, respectively. Either `numRows` or `numColumns` can be zero. Specifying `numRows` as zero allows for unlimited-length columns. Specifying `numColumns` as zero allows for unlimited-length rows.
 - Here, is a sample program that creates a 4x4 grid and fills it in with 15 buttons, each labeled with its index:
-

Program 1.16: Program to demonstrate gridlayout.

```
import java.awt.*;
import java.applet.*;
public class GridLayoutDemo extends Applet
{
    static final int n = 4;
    public void init()
    {
        setLayout(new GridLayout(n, n));
        setFont(new Font("SansSerif", Font.BOLD, 24));
        for(int i = 0; i < n; i++)
        {
            for(int j = 0; j < n; j++)
            {
                int k = i * n + j;
                if(k > 0)
                    add(new Button("" + k));
            }
        }
    }
}
/*<applet code="GridLayoutDemo" width=300 height=200>
</applet>*/
```

1.4.2.4 CardLayout

- The CardLayout Manager provides the means to manage multiple components, displaying one at a time. Components are displayed in the order in which they are added to the layout or in an arbitrary order by using an assignable name.
- The CardLayout class is unique among the other layout managers in that it stores several different layouts. Each layout can be thought of as being on a separate index card in a deck that can be shuffled so that any card is on top at a given time. This can be useful for user interfaces with optional components that can be dynamically enabled and disabled upon user input.
- The class CardLayout arranges each component in the container as a card. Only one card is visible at a time, and the container acts as a stack of cards.
- We can prepare the other layouts and have them hidden, ready to be activated when needed.
- CardLayout provides these two constructors:
 1. `CardLayout()`: Creates a default card layout.
 2. `CardLayout(int horz, int vert)`: Allows us to specify the horizontal and vertical space left between components in horz and vert, respectively.
- Use of a card layout requires a bit more work than the other layouts. The cards are typically held in an object of type Panel.
- This panel must have CardLayout selected as its layout manager. The cards that form the deck also typically objects of type Panel. Thus, we must create a panel that contains the deck and a panel for each card in the deck. Next, we add to the appropriate panel the components that form each card. We then add these panels to the panel for which CardLayout is the layout manager.
- Finally, we add this panel to the main applet panel. Once, these steps are complete, we must provide some way for the user to select between cards. One common approach is to include one push button for each card in the deck.
- When card panels are added to a panel, they are usually given a name. Thus, most of the time, we will use this form of `add()` when adding cards to a panel:

```
void add(Component panelObj, Object name);
```

- Here, name is a string that specifies the name of the card whose panel is specified by panelObj. After we have created a deck, our program activates a card by calling one of the following methods defined by CardLayout:

```
void first(Container deck)
```

```
void last(Container deck)
```

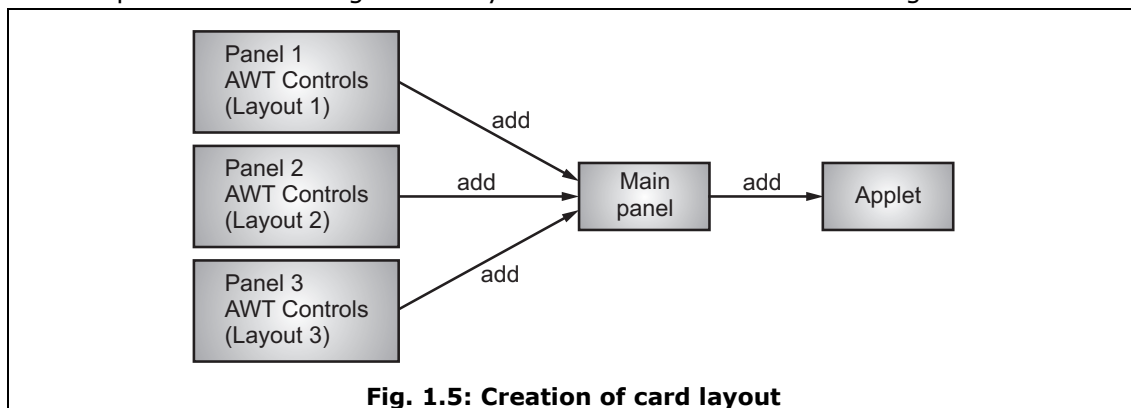
```
void next(Container deck)
```

```
void previous(Container deck)
```

```
void show(Container deck, String cardName)
```

Here, deck is a reference to the container (usually a panel) that holds the cards, and cardName is the name of a card. Calling first() causes the first card in the deck to be shown. For showing the last card, call last() and for the next card, call next().

- To show the previous card, call previous(). Both next() and previous() automatically cycle back to the top or bottom of the deck, respectively. The show() method displays the card whose name is passed in cardName.
- The following example creates a two-level card deck that allows the user to select an operating system. Windows-based operating systems are displayed in one card. Macintosh and Solaris are displayed in the other card.
- The process of creating a card layout is visualized as shown in Fig. 1.5.



Program 1.17: Program to demonstrate card layout.

```
import java.awt.*;
import java.awt.event.*;
//Class definition
class Panell extends Panel
{
    //Variable declaration
    Label la1,la2,la3;
    TextField te1,te2,te3;
    Button bu1, bu2, bu3, bu4, bu5;
    public Panell()
    {
        la1=new Label("Number 1:");
        la2=new Label("Number 2:");
        la3=new Label("Result:");
```

```
        te1=new TextField(10);
        te2=new TextField(10);
        te3=new TextField(10);
        te3.setEditable(false);
        bu1=new Button("Add");
        bu2=new Button("Sub");
        bu3=new Button("Mul");
        bu4=new Button("Div");
        bu5=new Button("Exit");
        add(la1);
        add(te1);
        add(la2);
        add(te2);
        add(la3);
        add(te3);
        add(bu1);
        add(bu2);
        add(bu3);
        add(bu4);
        add(bu5);
    }
}
class Panel2 extends Panel
{
    Checkbox ch1, ch2, ch3, ch4;
    public Panel2()
    {
        ch1=new Checkbox("Windows 8");
        ch2=new Checkbox("Linux Ubuntu");
        ch3=new Checkbox("Android");
        ch4=new Checkbox("macOS");
        setLayout(new GridLayout(2,2));
        add(ch1);
        add(ch2);
        add(ch3);
        add(ch4);
    }
}
```

```
class Panel3 extends Panel
{
    TextArea tax;
    TextField te1, te2;
    Label la1, la2;
    Button bu1, bu2, bu3, bu4;
    public Panel3()
    {
        tax=new TextArea(10,50);
        te1=new TextField(10);
        te2=new TextField(10);
        la1=new Label("Find What:");
        la2=new Label("Replace With:");
        bu1=new Button("Find");
        bu2=new Button("Replace");
        bu3=new Button("Replace All");
        bu4=new Button("Cancel");
        add(tax);
        add(la1);
        add(te1);
        add(la2);
        add(te2);
        add(bu1);
        add(bu2);
        add(bu3);
        add(bu4);
    }
}

class CardLayoutDemo extends Frame implements ActionListener
{
    Button bu1, bu2;
    Panel cards;
    CardLayout c;
    Panel1 pa1;
    Panel2 pa2;
    Panel3 pa3;
```

```
public CardLayoutDemo(String title)
{
    super(title);
    bu1=new Button("Next");//Object creation
    bu2=new Button("Previous");
    cards=new Panel();
    pa1=new Panel1();
    pa2=new Panel2();
    pa3=new Panel3();
    c=new CardLayout();
    cards.setLayout(c);
    cards.add(pa1,"Panel1");
    cards.add(pa2,"Panel2");
    cards.add(pa3,"Panel3");
    bu1.addActionListener(this);
    bu2.addActionListener(this);
    Panel p=new Panel();
    p.add(bu1);
    p.add(bu2);
    add("North",p);
    add("Center",cards);
    setSize(300,300);
    setVisible(true);
}
public void actionPerformed(ActionEvent e)
{
    if(e.getSource()==bu1)//Decision making statement
        c.next(cards);
    else
        c.previous(cards);
}
public static void main(String args[])
{
    new CardLayoutDemo("Card Layout Demo");
}
}
```

1.4.2.5 GridBagLayout

- The `java.awt.GridBagLayout` layout manager is the most powerful and flexible of all the predefined layout managers but more complicated to use.
- Unlike `GridLayout` where the components are arranged in a rectangular grid and each component in the container is forced to be the same size, in `GridBagLayout`, components are also arranged in rectangular grid but can have different sizes and can occupy multiple rows or columns, (See Fig. 1.6).
- The class `GridBagLayout` arranges components in a horizontal and vertical manner.

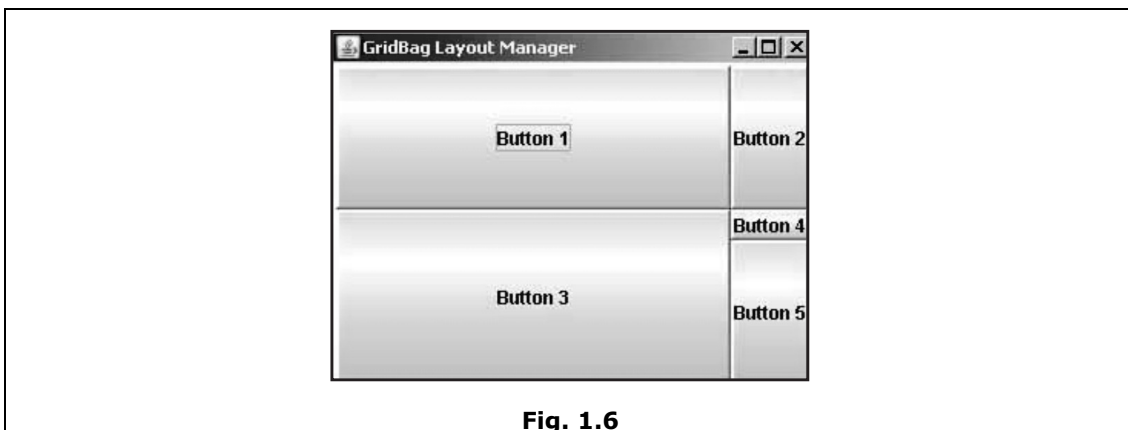


Fig. 1.6

- **Constructor:**
 1. `GridBagLayout()` : Creates a grid bag layout manager.
- `GridBagLayout` is one of the most flexible — and complex — layout managers the Java platform provides.
- A `GridBagLayout` places components in a grid of rows and columns, allowing specified components to span multiple rows or columns.
- Not all rows necessarily have the same height. Similarly, not all columns necessarily have the same width. `GridBagLayout` places components in rectangles (cells) in a grid, and then uses the components' preferred sizes to determine how big the cells should be.
- Fig. 1.7 shows the grid for the preceding applet. As you can see, the grid has three rows and three columns.
- The button in the second row spans all the columns; the button in the third row spans the two right columns.

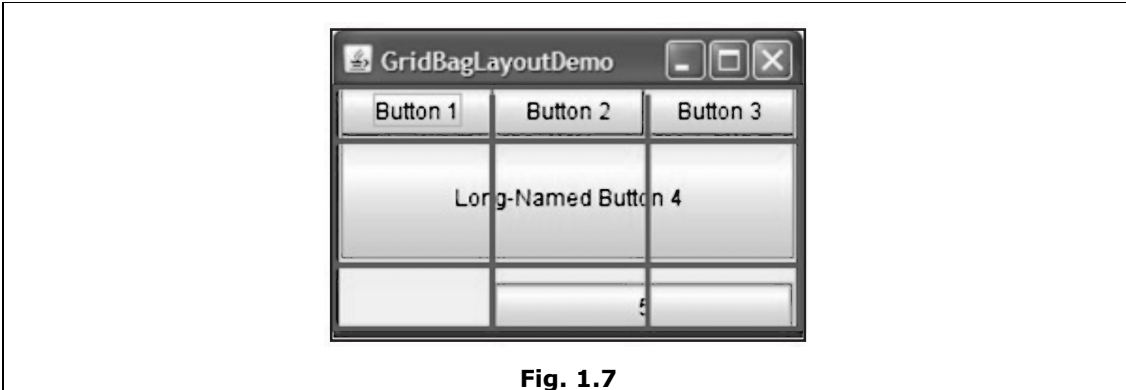


Fig. 1.7

- If you enlarge the window as shown in the Fig. 1.8, you will notice that the bottom row, which contains Button 5, gets all the new vertical space.
- The new horizontal space is split evenly among all the columns. This resizing behavior is based on weights the program assigns to individual components in the GridBagLayout.
- You will also notice that each component takes up all the available horizontal space — but not (as you can see with button 5) all the available vertical space. This behavior is also specified by the program.

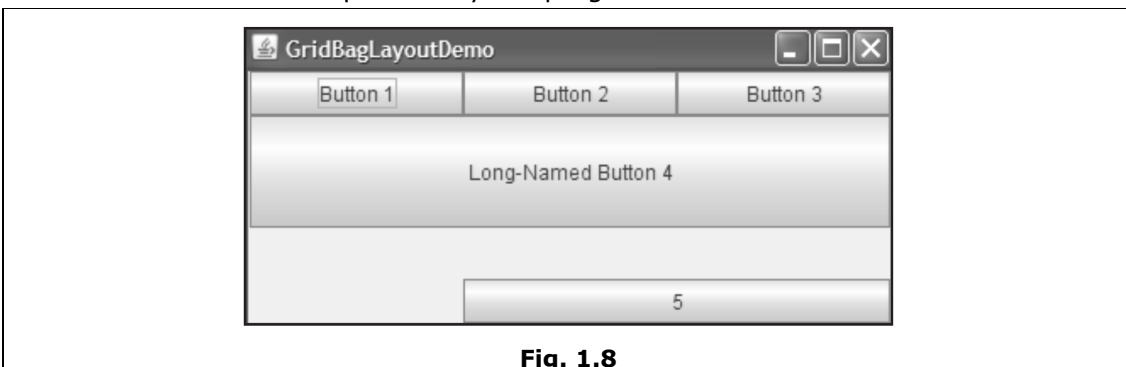


Fig. 1.8

- The way the program specifies the size and position characteristics of its components is by specifying constraints for each component.
- Instance variables to manipulate the GridBagLayout object Constraints are:

Variable	Role
<code>gridx</code> and <code>gridy</code>	These contain the coordinates of the origin of the grid. They allow a at a specific position of a component positioning. By default, they have GrigBagConstraint.RELATIVE value which indicates that a component can be stored to the right of previous.

contd. ...

gridwidth, gridheight	Define how many cells will occupy component (height and width), by the default is 1. The indication is relative to the other components of the line or the column. The GridBagConstraints. REMAINDER value specifies that the next component inserted will be the last of the line or the current column. The value GridBagConstraints. RELATIVE up the component after the last component of a row or column.
fill	Defines the fate of a component smaller than the grid cell. GridBagConstraints. NONE retains the original size: Default GridBagConstraints. HORIZONTAL expanded horizontally GridBagConstraints. VERTICAL GridBagConstraints: BOTH expanded vertically expanded to the dimensions of the cell.
ipadx, ipady	Used to define the horizontal and vertical expansion of components, not works if expansion is required by fill. The default value is (0,0).
anchor	When a component is smaller than the cell in which it is inserted, it can be positioned using this variable to define the side from which the control should be aligned within the cell. Possible variables NORTH, NORTHWEST, NORTHEAST, SOUTH, SOUTHWEST, SOUTHEAST, WEST and EAST.
weightx, weighty	Used to define the distribution of space in case of change of dimension.

Program 1.18: Program to demonstrate gridbag layout.

```
import java.awt.*;
import java.awt.event.*;
public class AwtLayoutDemo {
    private Frame mainFrame;
    private Label headerLabel;
    private Label statusLabel;
    private Panel controlPanel;
    private Label msgLabel;
    public AwtLayoutDemo(){
        prepareGUI();
    }
}
```

```
public static void main(String[] args){
    AwtLayoutDemo awtLayoutDemo = new AwtLayoutDemo();
    awtLayoutDemo.showGridBagLayoutDemo();
}
private void prepareGUI(){
    mainFrame = new Frame("Java AWT Examples");
    mainFrame.setSize(400,400);
    mainFrame.setLayout(new GridLayout(3, 1));
    mainFrame.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
        }
    });
    headerLabel = new Label();
    headerLabel.setAlignment(Label.CENTER);
    statusLabel = new Label();
    statusLabel.setAlignment(Label.CENTER);
    statusLabel.setSize(350,100);

    msgLabel = new Label();
    msgLabel.setAlignment(Label.CENTER);
    msgLabel.setText("Welcome to Tutorialspoint AWT Tutorial.");

    controlPanel = new Panel();
    controlPanel.setLayout(new FlowLayout());

    mainFrame.add(headerLabel);
    mainFrame.add(controlPanel);
    mainFrame.add(statusLabel);
    mainFrame.setVisible(true);
}
private void showGridBagLayoutDemo(){
    headerLabel.setText("Layout in action: GridBagLayout");

    Panel panel = new Panel();
    panel.setBackground(Color.darkGray);
    panel.setSize(300,300);
    GridBagLayout layout = new GridBagLayout();
```

```
panel.setLayout(layout);
GridBagConstraints gbc = new GridBagConstraints();

gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.gridx = 0;
gbc.gridy = 0;
panel.add(new Button("Button 1"), gbc);

gbc.gridx = 1;
gbc.gridy = 0;
panel.add(new Button("Button 2"), gbc);

gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.ipady = 20;
gbc.gridx = 0;
gbc.gridy = 1;
panel.add(new Button("Button 3"), gbc);

gbc.gridx = 1;
gbc.gridy = 1;
panel.add(new Button("Button 4"), gbc);

gbc.gridx = 0;
gbc.gridy = 2;
gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.gridwidth = 2;
panel.add(new Button("Button 5"), gbc);

controlPanel.add(panel);

mainFrame.setVisible(true);
}
}
```

1.4.3 Menu Bars and Menus

- A top-level window can have a menu bar associated with it. A menu bar displays a list of top-level menu choices. Each choice is associated with a drop down menu.

- Fig. 1.9 shows menu hierarchy.

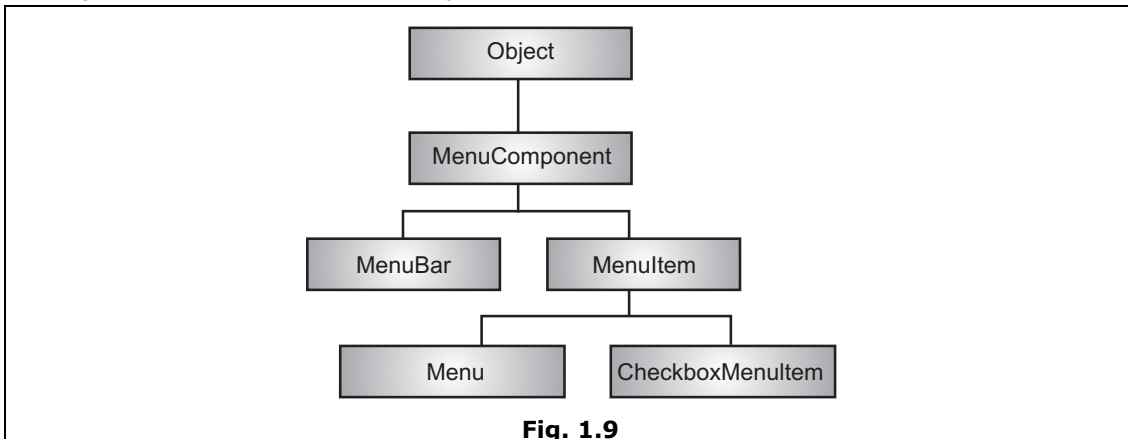


Fig. 1.9

- Menu controls are listed in following table:

Sr. No.	Control and Description
1.	MenuComponent: It is the top level class for all menu related controls.
2.	MenuBar: The MenuBar object is associated with the top-level window.
3.	MenuItem: The items in the menu must belong to the MenuItem or any of its subclass.
4.	Menu: The Menu object is a pull-down menu component which is displayed from the menu bar.
5.	CheckboxMenuItem: CheckboxMenuItem is subclass of MenuItem.
6.	PopupMenu: PopupMenu can be dynamically popped up at a specified position within a component.

1.4.3.1 Menu Bar

- In general, a menu bar contains one or more Menu objects. Each Menu object contains a list of MenuItem objects. Each MenuItem object represents something that can be selected by the user.
- Since, Menu is a subclass of MenuItem, a hierarchy of nested submenus can be created. It is also possible to include checkable menu items.
- These are menu options of type CheckboxMenuItem and will have a check mark next to them when they are selected.
- For creating a menu bar, we first create an instance of MenuBar. This class only defines the default constructor MenuBar() which Creates a new menu bar. Next, create instances of Menu that will define the selections displayed on the bar.

Program 1.19: Program to demonstrate menu bar.

```
import java.awt.*;
import java.awt.event.*;
public class AWTMenuDemo {
    private Frame mainFrame;
    private Label headerLabel;
    private Label statusLabel;
    private Panel controlPanel;
    public AWTMenuDemo(){
        prepareGUI();
    }
    public static void main(String[] args){
        AWTMenuDemo awtMenuDemo = new AWTMenuDemo();
        awtMenuDemo.showMenuDemo();
    }
    private void prepareGUI(){
        mainFrame = new Frame("Java AWT Examples");
        mainFrame.setSize(400,400);
        mainFrame.setLayout(new GridLayout(3, 1));
        mainFrame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent windowEvent){
                System.exit(0);
            }
        });
        headerLabel = new Label();
        headerLabel.setAlignment(Label.CENTER);
        statusLabel = new Label();
        statusLabel.setAlignment(Label.CENTER);
        statusLabel.setSize(350,100);
        controlPanel = new Panel();
        controlPanel.setLayout(new FlowLayout());
        mainFrame.add(headerLabel);
        mainFrame.add(controlPanel);
        mainFrame.add(statusLabel);
        mainFrame.setVisible(true);
    }
}
```

```
private void showMenuDemo(){
    //create a menu bar
    final MenuBar menuBar = new MenuBar();

    //create menus
    Menu fileMenu = new Menu("File");
    Menu editMenu = new Menu("Edit");
    final Menu aboutMenu = new Menu("About");

    //create menu items
    MenuItem newMenuItem =
        new MenuItem("New",new MenuShortcut(KeyEvent.VK_N));
    newMenuItem.setActionCommand("New");

    MenuItem openMenuItem = new MenuItem("Open");
    openMenuItem.setActionCommand("Open");

    MenuItem saveMenuItem = new MenuItem("Save");
    saveMenuItem.setActionCommand("Save");

    MenuItem exitMenuItem = new MenuItem("Exit");
    exitMenuItem.setActionCommand("Exit");

    MenuItem cutMenuItem = new MenuItem("Cut");
    cutMenuItem.setActionCommand("Cut");

    MenuItem copyMenuItem = new MenuItem("Copy");
    copyMenuItem.setActionCommand("Copy");

    MenuItem pasteMenuItem = new MenuItem("Paste");
    pasteMenuItem.setActionCommand("Paste");

    MenuItemListener menuItemListener = new MenuItemListener();

    newMenuItem.addActionListener(menuItemListener);
    openMenuItem.addActionListener(menuItemListener);
    saveMenuItem.addActionListener(menuItemListener);
    exitMenuItem.addActionListener(menuItemListener);
    cutMenuItem.addActionListener(menuItemListener);
    copyMenuItem.addActionListener(menuItemListener);
    pasteMenuItem.addActionListener(menuItemListener);

    final CheckboxMenuItem showWindowMenu =
        new CheckboxMenuItem("Show About", true);
```

```
showWindowMenu.addItemListener(new ItemListener() {
    public void itemStateChanged(ItemEvent e) {
        if(showWindowMenu.getState()){
            menuBar.add(aboutMenu);
        }else{
            menuBar.remove(aboutMenu);
        }
    }
});

//add menu items to menus
fileMenu.add(newMenuItem);
fileMenu.add(openMenuItem);
fileMenu.add(saveMenuItem);
fileMenu.addSeparator();
fileMenu.add(showWindowMenu);
fileMenu.addSeparator();
fileMenu.add(exitMenuItem);

editMenu.add(cutMenuItem);
editMenu.add(copyMenuItem);
editMenu.add(pasteMenuItem);

//add menu to menubar
menuBar.add(fileMenu);
menuBar.add(editMenu);
menuBar.add(aboutMenu);

//add menubar to the frame
mainFrame.setMenuBar(menuBar);
mainFrame.setVisible(true);
}

class MenuItemListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        statusLabel.setText(e.getActionCommand()
            + " MenuItem clicked.");
    }
}
}
```

1.4.3.2 Menu

- The Menu class represents pull-down menu component which is deployed from a menu bar.
- Following are the constructors for Menu:

1. Menu()
2. Menu(String optionName)
3. Menu(String optionName, boolean removable)

Here, optionName specifies the name of the menu selection. If removable is true, the pop-up menu can be removed and allowed to float free. Otherwise, it will remain attached to the menu bar.

The first form creates an empty menu. Individual menu items are of type MenuItem. It defines these constructors:

1. MenuItem(): Constructs a new MenuItem with an empty label and no keyboard shortcut.
2. MenuItem(String itemName): Creates a menuItem where itemName is the name shown in the menu.
3. MenuItem(String itemName, MenuShortcut keyAccel): Create a menu item with an associated keyboard shortcut.

Here, itemName is the name shown in the menu, and keyAccel is the menu shortcut for this item.

- We can disable or enable a menu item by using the setEnabled() method. Its form/syntax is shown below:

```
void setEnabled(boolean enabledFlag)
```

Program 1.20: Program to demonstrate menu.

```
import java.awt.*;
import java.awt.event.*;

public class AWTMenuDemo {
    private Frame mainFrame;
    private Label headerLabel;
    private Label statusLabel;
    private Panel controlPanel;

    public AWTMenuDemo(){
        prepareGUI();
    }
}
```

```
public static void main(String[] args){
    AWTMenuDemo awtMenuDemo = new AWTMenuDemo();
    awtMenuDemo.showMenuDemo();
}

private void prepareGUI(){
    mainFrame = new Frame("Java AWT Examples");
    mainFrame.setSize(400,400);
    mainFrame.setLayout(new GridLayout(3, 1));
    mainFrame.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
        }
    });
    headerLabel = new Label();
    headerLabel.setAlignment(Label.CENTER);
    statusLabel = new Label();
    statusLabel.setAlignment(Label.CENTER);
    statusLabel.setSize(350,100);

    controlPanel = new Panel();
    controlPanel.setLayout(new FlowLayout());

    mainFrame.add(headerLabel);
    mainFrame.add(controlPanel);
    mainFrame.add(statusLabel);
    mainFrame.setVisible(true);
}

private void showMenuDemo(){
    //create a menu bar
    final MenuBar menuBar = new MenuBar();

    //create menus
    Menu fileMenu = new Menu("File");
    Menu editMenu = new Menu("Edit");
    final Menu aboutMenu = new Menu("About");
```

```
//create menu items
MenuItem newMenuItem =
    new MenuItem("New",new MenuShortcut(KeyEvent.VK_N));
newMenuItem.setActionCommand("New");

MenuItem openMenuItem = new MenuItem("Open");
openMenuItem.setActionCommand("Open");

MenuItem saveMenuItem = new MenuItem("Save");
saveMenuItem.setActionCommand("Save");

MenuItem exitMenuItem = new MenuItem("Exit");
exitMenuItem.setActionCommand("Exit");

MenuItem cutMenuItem = new MenuItem("Cut");
cutMenuItem.setActionCommand("Cut");

MenuItem copyMenuItem = new MenuItem("Copy");
copyMenuItem.setActionCommand("Copy");

MenuItem pasteMenuItem = new MenuItem("Paste");
pasteMenuItem.setActionCommand("Paste");

MenuItemListener menuItemListener = new MenuItemListener();

newMenuItem.addActionListener(menuItemListener);
openMenuItem.addActionListener(menuItemListener);
saveMenuItem.addActionListener(menuItemListener);
exitMenuItem.addActionListener(menuItemListener);
cutMenuItem.addActionListener(menuItemListener);
copyMenuItem.addActionListener(menuItemListener);
pasteMenuItem.addActionListener(menuItemListener);

final CheckboxMenuItem showWindowMenu =
    new CheckboxMenuItem("Show About", true);
showWindowMenu.addItemListener(new ItemListener() {
    public void itemStateChanged(ItemEvent e) {
        if(showWindowMenu.getState()){
            menuBar.add(aboutMenu);
        }else{
            menuBar.remove(aboutMenu);
        }
    }
});
```

```
//add menu items to menus
fileMenu.add(newMenuItem);
fileMenu.add(openMenuItem);
fileMenu.add(saveMenuItem);
fileMenu.addSeparator();
fileMenu.add(showWindowMenu);
fileMenu.addSeparator();
fileMenu.add(exitMenuItem);

editMenu.add(cutMenuItem);
editMenu.add(copyMenuItem);
editMenu.add(pasteMenuItem);

//add menu to menubar
menuBar.add(fileMenu);
menuBar.add(editMenu);
menuBar.add(aboutMenu);

//add menubar to the frame
mainFrame.setMenuBar(menuBar);
mainFrame.setVisible(true);
}

class MenuItemListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        statusLabel.setText(e.getActionCommand()
            + " MenuItem clicked.");
    }
}
}
```

-
- If the argument `enabledFlag` is true, the menu item is enabled. If false, the menu item is disabled.
 - We can determine an item's status by calling `isEnabled()`. Its form/syntax is shown below:

```
boolean isEnabled( )
```

- The `isEnabled()` returns true if the menu item on which it is called is enabled. Otherwise, it returns false.

- We can change the name of a menu item by calling `setLabel()`. Its form/syntax is shown below:

```
void setLabel(String newName)
```

Here, `newName` becomes the new name of the invoking menu item.

- We can retrieve the current name by using `getLabel()`. Its form/syntax is shown below:

```
String getLabel( )
```

`getLabel()` returns the current name.

- We can create a checkable menu item by using a subclass of `MenuItem` called `CheckboxMenuItem()`. It has following constructors:
 1. `CheckboxMenuItem()`: Create a check box menu item with an empty label.
 2. `CheckboxMenuItem(String itemName)`: Create a check box menu item with the specified item name.
 3. `CheckboxMenuItem(String itemName, boolean on)`: Create a check box menu item with the specified label and state. Here, `itemName` is the name shown in the menu. Checkable items operate as toggles. Each time one is selected, its state changes.
- In the first two forms, the checkable entry is unchecked. In the third form, if `on` is true, the checkable entry is initially checked. Otherwise, it is cleared.
- We can obtain the status of a checkable item by calling `getState()`.
- We can set it to a known state by using `setState()`. Its form/syntax is shown below:

```
void setState(boolean checked)
```

- If the item is checked, `getState()` returns true, otherwise, it returns false. Its form/syntax is shown below:

```
boolean getState( )
```

- For checking an item, pass true to `setState()`. To clear an item, pass false. Once we have created a menu item, we must add the item to a `Menu` object by using `add()`, which has the following general form:

```
MenuItem add(MenuItem item)
```

- Here, `item` is the item being added. Items are added to a menu in the order in which the calls to `add()` take place. The item is returned. Once we have added all items to a `Menu` object, we can add that object to the menu bar by using this version of `add()` defined by `MenuBar`.

```
Menu add(Menu menu)
```

- Here, menu is the menu being added. The menu is returned. Menus only generate events when an item of type MenuItem or CheckboxMenuItem is selected. They do not generate events when a menu bar is accessed to display a drop-down menu, for example. Each time a menu item is selected, an ActionEvent object is generated. Each time a check box menu item is checked or unchecked, an ItemEvent object is generated. Thus, we must implement the ActionListener and ItemListener interfaces in order to handle these menu events.
- The getItem() method of ItemEvent returns a reference to the item that generated this event.
- The **general form/syntax** of this method is shown below:

```
Object getItem( )
```

1.4.3.3 Pop-up Menu

- Pop-up menu represents a menu which can be dynamically popped up at a specified position within a component.
- **Constructors:**
 1. PopupMenu() : Creates a new popup menu with an empty name.
 2. PopupMenu(String label) : Creates a new popup menu with the specified name.
- Following is an example that adds a series of nested menus to a pop-up window. The item selected is displayed in the window. The state of the two check box menu items is also displayed.

Program 1.21: Program to demonstrate pop-up menu.

```
import java.awt.*;
import java.applet.*;
class MenuFrame extends Frame
{
String msg = "";
CheckboxMenuItem debug, test;
MenuFrame(String title)
{
super(title);
// create menu bar and add it to frame
MenuBar mbar = new MenuBar();
setMenuBar(mbar);
// create the menu items
Menu file = new Menu("File");
```

```
MenuItem item1, item2, item3, item4, item5;
file.add(item1 = new MenuItem("New..."));
file.add(item2 = new MenuItem("Open..."));
file.add(item3 = new MenuItem("Close"));
file.add(item4 = new MenuItem("-"));
file.add(item5 = new MenuItem("Quit..."));
mbar.add(file);

Menu edit = new Menu("Edit");
MenuItem item6, item7, item8, item9;
edit.add(item6 = new MenuItem("Cut"));
edit.add(item7 = new MenuItem("Copy"));
edit.add(item8 = new MenuItem("Paste"));
edit.add(item9 = new MenuItem("-"));

Menu sub = new Menu("Special");
MenuItem item10, item11, item12;
sub.add(item10 = new MenuItem("First"));
sub.add(item11 = new MenuItem("Second"));
sub.add(item12 = new MenuItem("Third"));
edit.add(sub);

// these are checkable menu items
debug = new CheckboxMenuItem("Debug");
edit.add(debug);
test = new CheckboxMenuItem("Testing");
edit.add(test);
mbar.add(edit);
}
}

public class MenuDemo extends Applet
{
    Frame f;

    public void init()
    {
        f = new MenuFrame("Menu Demo");
        f.setVisible(true);
    }
}
```

```
int width = Integer.parseInt(getParameter("width"));
int height = Integer.parseInt(getParameter("height"));
setSize(width, height);
f.setSize(width, height);
}
public void start()
{
f.setVisible(true);
}
public void stop()
{
f.setVisible(false);
}
}
/*<applet code="MenuDemo" width=250 height=250>
</applet>*/
```

1.4.4 Dialog Boxes

- A dialog box is a GUI object in which you can place messages that you want to display on the screen.
- Often, we will want to use a dialog box to hold a set of related controls. Dialog boxes are primarily used to obtain user input.
- Dialog boxes are similar to frame windows, except that dialog boxes are always child windows of a top-level window. Also, dialog boxes don't have menu bars.
- In other respects, dialog boxes function like frame windows, (We can add controls to them, for example, in the same way that we add controls to a frame window).
- Dialog boxes may be modal or modeless. When a modal dialog box is active, all input is directed to it until it is closed. This means that we cannot access other parts of our program until we have closed the dialog box.
- When a modeless dialog box is active, input focus can be directed to another window in our program. Thus, other parts of our program remain active and accessible.
- Dialog boxes are of type Dialog. Two commonly used constructors are shown here:

1. Dialog(Frame parentWindow, boolean mode)
2. Dialog(Frame parentWindow, String title, boolean mode)

Here, parentWindow is the owner of the dialog box. If mode is true, the dialog box is modal. Otherwise, it is modeless. The title of the dialog box can be passed in title. Generally, we will subclass Dialog, adding the functionality required by your application.

Program 1.22: Program to demonstrate dialog box.

```
import java.awt.*;
import java.awt.event.*;
class AboutDialog extends Dialog
{
    public AboutDialog(Frame parent)
    {
        super(parent, true);
        setBackground(Color.gray);
        setLayout(new BorderLayout());

        Panel p = new Panel();
        p.add(new Button("Close"));
        p.add(new Button("Help"));
        add("South", p);
    }
    public boolean action(Event evt, Object arg)
    {
        if(arg.equals("Close"))
        {
            dispose();
            return true;
        }
        return false;
    }
    public void paint(Graphics g)
    {
        g.setColor(Color.white);
        g.drawString("My Application", 25, 70);
        g.drawString("Version 1.0", 25, 90);
    }
    public static void main(String args[ ])
    {
        new AboutDialog();
    }
}
```

1.4.4.1 FileDialog

- Java provides a built-in dialog box that lets the user specify a file.
 - To create a file dialog box, instantiate an object of type FileDialog. This causes a file dialog box to be displayed. Usually, this is the standard file dialog box provided by the operating system.
 - FileDialog provides following constructors:
 1. `FileDialog(Frame parent, String boxName)`: It creates File Dialog box where parent is the owner of the dialog box and boxName is the name displayed in the box's title bar.
 2. `FileDialog(Frame parent, String boxName, int how)`: It creates File Dialog box where parent is the owner of the dialog box and boxName is the name displayed in the box's title bar. If how is `FileDialog.LOAD`, then the box is selecting a file for reading. If how is `FileDialog.SAVE`, the box is selecting a file for writing.
 3. `FileDialog(Frame parent)`: This creates a dialog box for selecting a file for reading.
 - `FileDialog()` provides methods that allows us to determine the name of the file and its path as selected by the user. Here are given below:

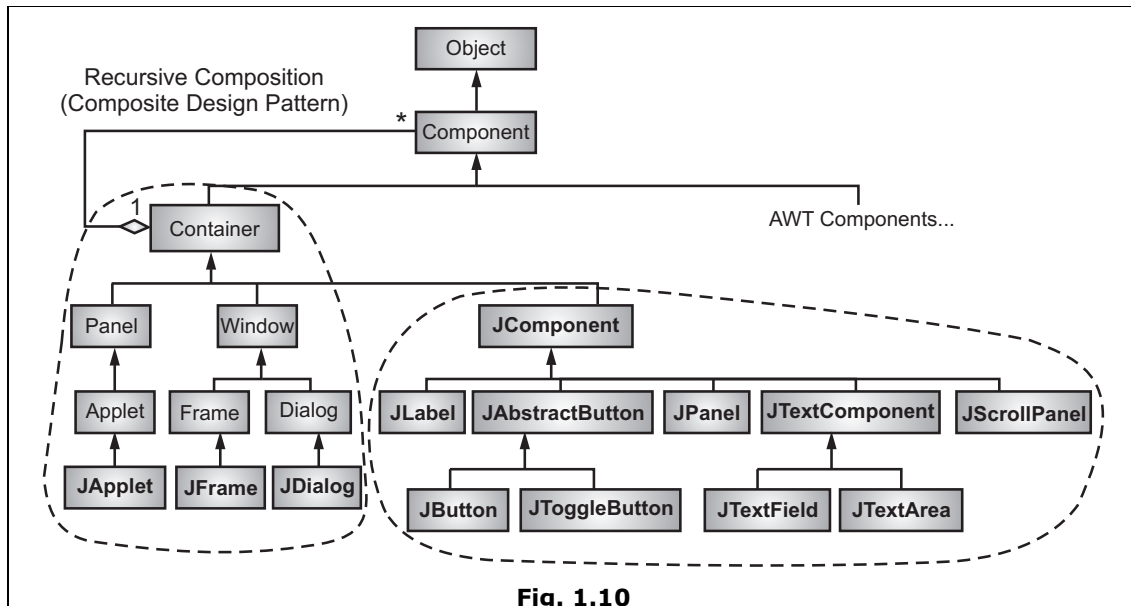
```
String getDirectory( )
String getFile( )
```
 - These methods return the directory and the filename, respectively.
-

Program 1.23: Program to demonstrate standard file dialog box.

```
import java.awt.*;
class SampleFrame extends Frame
{
    SampleFrame(String title)
    {
        super(title);
    }
}
class FileDialogDemo
{
    public static void main(String args[])
    {
        Frame f = new SampleFrame("File Dialog Demo");
        f.setVisible(true);
        f.setSize(100, 100);
        FileDialog fd = new FileDialog(f, "File Dialog");
        fd.setVisible(true);
    }
}
```

1.5 INTRODUCTION TO SWING

- Swing is a Java foundation classes library and it is an extension to do Abstract Windowing Toolkit (AWT).
- Swing is a extension to the AWT components which provides feature of pluggable look and feel for the components. It provides classes to design lightweight components.
- Swing is the next-generation GUI toolkit which Sun Microsystems is developing to enable enterprise development in Java.
- Swing is actually part of a large family of Java products which is known as Java Foundation Classes (JFC).
- Swing is used to create large-scale Java applications with a wide array of powerful components which can you easily extend or modify these components to control their appearance and behavior based on the current "look and feel" library that is being used.
- Swing is a set of classes, this provides more powerful and flexible components than are possible with the AWT. In addition to the familiar components, such as buttons, check boxes, and labels, Swing supplies several exciting additions, including tabbed panes, scroll panes, trees, and tables.
- Even familiar components such as buttons have more capabilities in Swing. For example, a button may have both an image and a text string associated with it. Also, the image can be changed as the state of the button changes.
- Unlike AWT components, Swing components are not implemented by platform-specific code. Instead, they are written entirely in Java and, therefore, are platform-independent. The term lightweight is used to describe such elements.
- The number of classes and interfaces in the Swing packages is substantial. Swing is the set of packages built on top of the AWT that provide us with a great number of pre-built classes that is, over 250 classes and 40 UI components.
- Fig. 1.10 shows the class hierarchy of the swing GUI classes.
- Every swing controls inherits properties from following Component class hierarchy. A Container is the abstract base class for the non menu user-interface controls of swing.
- Component represents an object with graphical representation. A Container is a component that can contain other SWING components.
- A JComponent is a base class for all swing UI components. In order to use a swing component that inherits from JComponent, component must be in a containment hierarchy whose root is a top-level Swing container.



- Class Description of Fig. 1.10
 1. **AbstractButton:** Abstract super-class for Swing buttons.
 2. **ButtonGroup:** Encapsulates a mutually exclusive set of buttons.
 3. **ImageIcon:** Encapsulates an icon.
 4. **JApplet:** The Swing version of Applet.
 5. **JButton:** The Swing push button class.
 6. **JCheckBox:** The Swing check box class.
 7. **JComboBox:** Encapsulates a combo box, (a combination of a dropdownlist and text field).
 8. **JLabel:** The Swing version of a label.
 9. **JRadioButton:** The Swing version of a radio button.
 10. **JScrollPane:** Encapsulates a scrollable window.
 11. **JTabbedPane:** Encapsulates a tabbed window.
 12. **JTable:** Encapsulates a table-based control.
 13. **JTextField:** The Swing version of a text field.
 14. **JTree:** Encapsulates a tree-based control.

1.5.1 Swing Features

- Various features of swing are listed below:
 1. **Light Weight:** Swing component are independent of native Operating System's API as Swing API controls are rendered mostly using pure JAVA code instead of underlying operating system calls.
 2. **Rich controls:** Swing provides a rich set of advanced controls like Tree, TabbedPane, slider, color picker, table controls and so on.
 3. **Borders:** We can draw borders in many different styles around components using the `setborder()` method.
 4. **Easy mouseless operation:** It is easy to connect keystrokes to components.
 5. **Tooltips:** We can use the `setToolTipText` method of `JComponent` to give components a tooltip, one of those small windows that appear when the mouse hovers over a component and gives explanatory text.
 6. **Easy Scrolling:** We can connect scrolling to various components-something that was impossible in AWT.
 7. **Pluggable look and feel:** We can set the appearance of applets and applications to one of three standard looks, i.e., Windows, Motif (Unix) or Metal (Standard swing look).
 8. **Highly customizable:** Swing controls can be customized in very easy way as visual appearance is independent of internal representation.
 9. **New layout managers:** Swing introduces the `BoxLayout` and `Overlay Layout` layout managers.

1.5.2 Working with Swing

- While designing any GUI, we need to have a main window on which we can place or position the different visual components.
- In swing the main window, also called a top level container is the root of a hierarchy, which contains all the other swing components that appear inside the window. All swing applications have at least one top level container.
- Every top level container has one intermediate container called content pane.
- This content pane contains all the visible components in the GUI window.
- The content plane is the base pane upon which all other components or container objects are placed. One exception to this rule is, if there is a menu bar in the top level container it will be placed outside the content pane.

- Fig. 1.11 illustrates the relationship between the components discussed above.

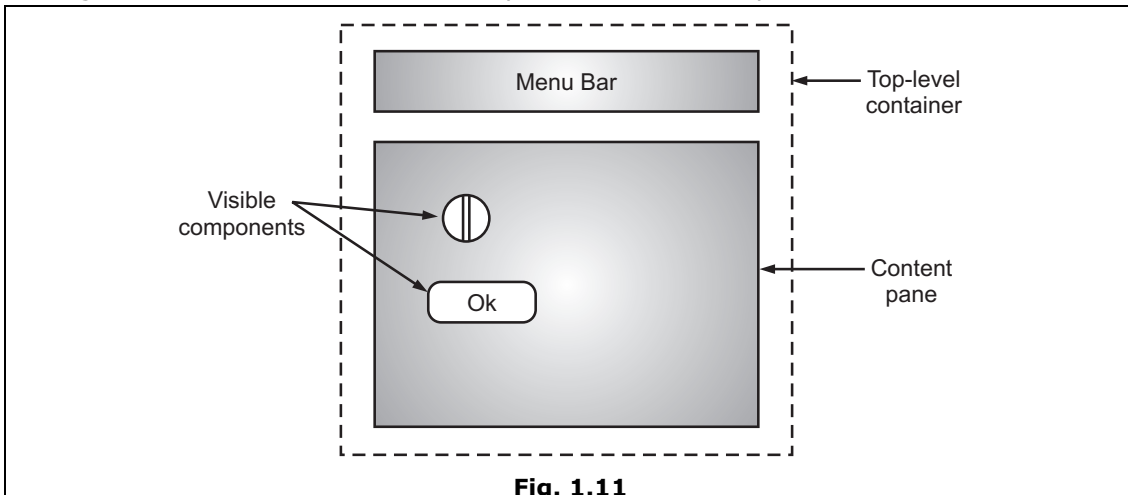


Fig. 1.11

- The general procedure to create top-level container is:
 1. Create the container,
 2. Set its size (in the case of frames and panels), and
 3. Set in visibility (in the case of frames and panels).
- All swing components names start with J. For instance the swing button class is named as JButton commonly used top-level containers are:
- Just like AWT application, a Swing application requires a *top-level container*. There are three top-level containers in Swing:
 - 1. JFrame:** Used for the application's main window (with an icon, a title, minimize/maximize/close buttons, an optional menu-bar, and a content-pane), as illustrated.
 - 2. JDialog:** Used for secondary pop-up window (with a title, a close button, and a content-pane).
 - 3. JApplet:** Used for the applet's display-area (content-pane) inside a browser's window.
- Similarly to AWT, there are *secondary containers* (such as JPanel) which can be used to group and layout relevant components.
- Panels are an example of intermediate containers.

1.5.3 Advantages and Disadvantages of Swing

Advantages:

1. Swing provides paint debugging support for when you build your own component.
2. Swing components are lightweight than AWT.

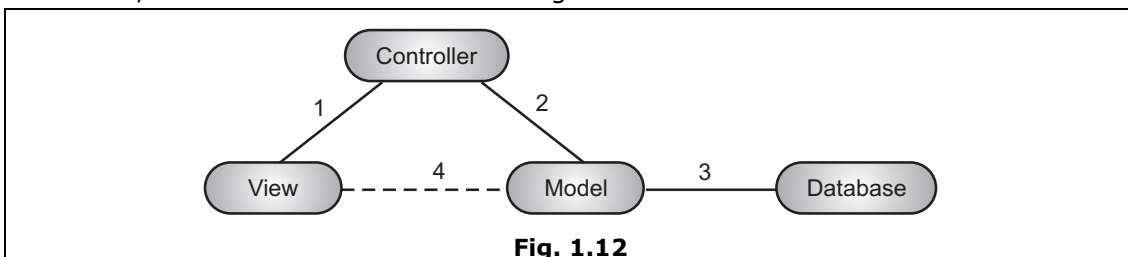
3. Swing components follow the Model-View-Controller (MVC) paradigm and thus can provide a much more flexible UI.
4. Swing provides both additional components like JTable, JTree etc and added functionality to replacement of AWT components.
5. Swing provides built-in double buffering.

Disadvantages:

1. It can be slower than AWT (all components are drawn), when you are not careful about programming.
2. Swing components might not behave exactly like native components which look like native components.
3. It requires Java 2 or a separate JAR file.

1.5.4 Model-View-Controller (MVC) Architecture

- The Model-View-Controller (MVC) architectural pattern is used in software engineering to allow for the separation of three common features of GUI applications:
 1. The data access (typically via a database),
 2. The business logic (how the data will be used), and
 3. User interaction (how the data and actions will be visually presented).
- Towards this end one defines three components of such an architecture, the model, view and controller with the Fig. 1.12.

**Fig. 1.12****The components of MVC are:**

- 1. Model:** This provides the means by which data is retrieved and manipulated.
- 2. View:** This represents the visual interface components of the Graphical User Interface (GUI) application which interact with the user. The interaction is of an event-driven nature where actions are initiated via keyboard and mouse.
- 3. Controller:** This joins the Model with the View and is the heart of the control logic by associating user-generated events with data actions.

- The interactions correspond to the Fig. 1.12 as follows:
 1. All actions begin in the view through events generated by the user. The controller provides listeners for the events. The controller also has the ability to manipulate the state of the view objects in a way which offers a response to the events generated by the user.
 2. The controller interacts with the model by either requesting information from the data source based on user-generated events, or by modifying the data based on these events.
 3. The model provides the programming interface which the controller must use to access the database, i.e., the controller does not interact directly with the database. In particular the notion of "connection" is never seen in the controller. Furthermore, the model also provides means to avoid, in most cases, direct SQL queries to the database.
 4. The view interacts with the model only to know about type information and other data abstractions held in the model. The relatively weak link indicates that the View/Model interaction should be "minimal" because the manipulation of data is to be done within the event handlers of the controller.

MVC in Java Swing GUIs:

- The basis of most Java Swing GUI applications is a single object created from an instance of a JFrame class extension.
- Another common auxiliary view class is a JDialog extension.
- Both JFrame and JDialog objects are swing Containers in that they hold non-container objects, such as JButton and others, which are responsible for capturing and presenting data and generating events.

MVC Package Decomposition:

- Although there is no official way to separate classes according to the MVC decomposition, we try to put corresponding classes into three separate packages i.e., models, views, controller.
 1. The controller package holds at least one Controller class which contains the main function used to initiate the application.
 2. The views package holds at least one class, TheFrame, an extension of the JFrame class.
 3. The models package holds classes, typically in which JDBC is used to access database records, tables and connection.

View/Controller Event Handling:

- A Java Swing event generation begins with an object which generates the event (like a JButton) and ends with an object which executes the action for the event.
- For example, suppose we have class TheFrame representing the JFrame for our GUI application containing a JButton object, button.
- We want the controller to be the object which handles the button's click event.

Swing vs AWT:

AWT (Abstract Windowing Toolkit)	Swing
1. A rich set of user interface components.	1. All the features of AWT.
2. A robust event-handling model.	2. 100% Pure Java certified versions of the existing AWT component set (Button, Scrollbar, Label, etc.).
3. Graphics and imaging tools, including shape, color, and font classes.	3. A rich set of higher-level components (such as tree view, list box, and tabbed panes).
4. Layout managers, for flexible window layouts that don't depend on a particular window size or screen resolution.	4. Pure Java design, no reliance on peers.
5. Data transfer classes, for cut-and-paste through the native platform clipboard.	5. Pluggable Look and Feel.
<p>6. Components Difference: The Abstract Windowing Toolkit (AWT) is the original GUI toolkit shipped with the Java Development Kit (JDK). The AWT provides a basic set of graphical interface components similar to those available with HTML forms.</p>	<p>Swing is the latest GUI toolkit, and provides a richer set of interface components than the AWT. In addition, Swing components offer the following advantages over AWT components:</p> <ul style="list-style-type: none"> ○ The behavior and appearance of Swing components is consistent across platforms, whereas AWT components will differ from platform to platform ○ Swing components can be given their own "look and feel" ○ Swing uses a more efficient event model than AWT; therefore, Swing components can run more quickly than their AWT counterparts <p>On the other hand, Swing components can take longer to load than AWT components.</p>

contd. ...

<p>7. Advantages (Pros):</p> <p>(i) Speed: Use of native peers speeds component performance.</p> <p>(ii) Applet Portability: Most Web browsers support AWT classes so AWT applets can run without the Java plugin.</p> <p>(iii) Look and Feel: AWT components more closely reflect the look and feel of the OS they run on.</p>	<p>(i) Portability: Pure Java design provides for fewer platform specific limitations.</p> <p>(ii) Behavior: Pure Java design allows for a greater range of behavior for Swing components since they are not limited by the native peers that AWT uses.</p> <p>(iii) Features: Swing supports a wider range of features like icons and pop-up tool-tips for components.</p> <p>(iv) Vendor Support: Swing development is more active. Sun puts much more energy into making Swing robust</p> <p>(v) Look and Feel: The pluggable look and feel lets you design a single set of GUI components that can automatically have the look and feel of any OS platform (Microsoft Windows, Solaris, Macintosh, etc.). It also makes it easier to make global changes to your Java programs that provide greater accessibility, (like picking a hi-contrast color scheme or changing all the fonts in all dialogs, etc.).</p>
<p>8. Disadvantages (Cons):</p> <p>(i) Portability: Use of native peers creates platform specific limitations. Some components may not function at all on some platforms.</p> <p>(ii) Third Party Development: The majority of component makers, including Borland and Sun, base new component development on Swing components. There is a much smaller set of AWT components available, thus placing the burden on the programmer to create his or her own AWT-based components.</p> <p>(iii) Features: AWT components do not support features like icons and tool-tips.</p>	<p>(i) Applet Portability: Most Web browsers do not include the Swing classes, so the Java plugin must be used.</p> <p>(ii) Performance: Swing components are generally slower and buggier than AWT, due to both the fact that they are pure Java and to video issues on various platforms. Since Swing components handle their own painting (rather than using native API's like DirectX on Windows) you may run into graphical glitches.</p> <p>(iii) Look and Feel: Even when Swing components are set to use the look and feel of the OS they are run on, they may not look like their native counterparts.</p>

1.5.5 Swing Components

- Swing components are not implemented by platform specific code. Instead they are written entirely in Java and therefore, are platform independent.
- Swing component is called light weight, as it does not depend on any non-java system classes. Swing components have their own view supported by java's look and feel classes.
- Swing components have pluggable look and feel so that with a single set of components you can achieve the same look and feel on any operating system platform.
- The swing related classes are contained in javax.swing and its subpackages, such as javax.swing.tree. To use a swing class, either use an import statement for a specific class to be imported from within a swing or import all classes in the swing package as:

```
import javax.swing.*;
```

1.5.5.1 JFrame

- The frame is a top-level container or window, on which other swing components are placed.
- A JFrame component is used to create windows in a swing program. It extends java.awt.Frame class. Every JFrame object has a root pane.
- Some of the constructors of the JFrame class are shown below:
 1. `JFrame()`: This constructor creates a new frame without title.
 2. `JFrame(string title)`: This constructor constructs the new frame with title.
- When adding components to a JFrame window, we must add it to its content pane and not directly to the JFrame window.
- For example: To add a button for b we would write `frame.getContentPane().add(b);`

- **Methods:**

<code>Container getContentPane()</code>	This method returns, a Content Pane for the JFrame.
<code>void setLayout(LayoutManager)</code>	This method sets the LayoutManager for the JFrame.
<code>void setJMenuBar(JMenuBar)</code>	This method sets the JMenu Bar to the JFrame.
<code>Void setIconImage(Image image)</code>	Prints icon image on JFrame.

Program 1.24: Program to demonstrate JFrame.

```
import java.awt.*;
import javaX.swing.*;
public class frameDemo extends JFrame
{
    public FrameDemo(string title)
    {
        Super(title);
    }
    public static void main(string args[])
    {
        FrameDemo fd = new FrameDemo("First frame");
        fd.setVisible(true);
        fd.setSize(300, 300);
    }
}
```

- Here, we have imported the relevant packages necessary to create the frame window. Then in the constructor we invoked super() which in turn calls the constructor of JFrame and creates the frame. We then sets the size of the frame and changed its visibility to true in order to display it when it is executed.

1.5.5.2 JPanel

- The JPanel component which is an intermediate container, is used to group smaller lightweight components together. By default panels are opaque which means that they work similar to content panes.
- JPanel objects have FlowLayout as their default layout.
- Some of the constructors of JPanel are:

```
JPanel();
JPanel(LayoutManager LM)
```

- Most methods of the JPanel API are derived from its super classes, container JComponent and component.

- **Methods:**

Container getContentPane()	This method returns, a Content Pane for the JPanel.
void setLayout(LayoutManager)	This method sets the LayoutManager for the JPanel.
void setJMenuBar(JMenuBar)	This method sets the JMenuBar to the JFrame.
void setIconImage(Image image)	Prints icon image on JFrame.

Program 1.25: Program to demonstrate JPanel.

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
class JPanelExample extends JFrame
{
    JPanel panel1, panel2 ;
    JTextField txtData;
    JButton[] btnData = new JButton[12];
    JPanelExample()
    {
        panel1=new JPanel();
        txtData = new JTextField(20);
        panel1.add(txtData);
        add(panel1, "North");
        panel2=new JPanel(new GridLayout(3,4));
        for(int i=0;i<=9;i++)
        {
            btnData[i]=new JButton(""+i);
            panel2.add(btnData[i]);
        }
        add(panel2, "Center");
    }
}
class JPanelJavaExample
{
    public static void main(String[] args)
    {
        JPanelExample frame = new JPanelExample();
        frame.setTitle("JPanel Java Example");
        frame.setBounds(100,200,220,200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);
    }
}
```

1.5.5.3 JApplet

- The extended version of java.applet.Applet is the Java Foundation class of Applet.
- Applets that use swings must be subclasses of JApplet. JApplet is rich with functionality that is not found in Applet.
- For example, JApplet supports various panes such as root pane glass pane and content pane.

1. Root Pane:

- Swing containers like JApplet, JFrame, JWindow and JDialog delegate their duties to the root pane represented by the class JRootPane.
- The root pane is made up of a content pane, layered pane, glass pane and an optional menu bar, other GUI components must be added to one of these roots pane members, rather than to the root pane itself.
- Fig. 1.13 shows how root pane members are positioned inside the root pane.
- The root pane and its member are all considered fundamental to Swing/GUI design.
- A reference to the underlying root pane can be achieved by using the `getRootPane()` method.

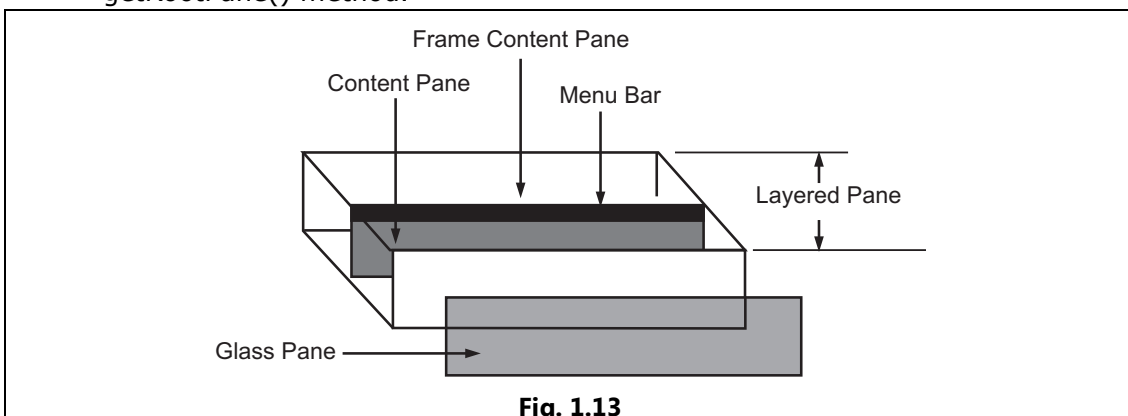


Fig. 1.13

2. Content Pane:

- This pane actually sits on one of the layers of a specially layered pane.
- The layered pane in Fig. 1.13 contains several layers meant for displaying different Swing components, depending on the GUI's requirements.
- Fig. 1.13 shows the position of the content pane in the specially layered pane. The menu bar and the content pane sit on the Frame-Content-Layer.
- Numbered Swing components are added to the content pane.

3. Glass Pane:

- It is a member of the multi-layered, root pane.
- It is used to draw over an area that already contains some components.

- It can also be used to catch mouse events because it sits over the top of the content pane and menu bar.
 - Fig. 1.13 shows the position of the glass pane in a root pane. Because glass pane is a member of the root pane, it already exists in a Swing container.
 - The glass pane is not visible by default.
-

Program 1.26: Program to demonstrate glass pane.

```
import java.awt.*;
import java.swing.*;
public class swingApplet extends JApplet
{
    public void paintGraphics(G)
    {
        G.drawString("Hello", 10,20);
    }
}
/* <Applet Code = swingApplet height = 300 width = 500> */
```

- The content pane makes swing applets different from regular applets in the following ways:
 - Components are added to the content pane of swing applets, not directly to the applets.
 - The layout manager is set for the content pane of a swing Applet, not directly for the applet.
 - The default layout manager for the content pane of a swing based applet is Border-Layout. This differs from the default Layout Manager for Applet which is Flow-Layout.
 - The code for the paint() method is not directly put in JApplet object. Instead paintComponent() method is used.

1.5.5.4 JDialog

- A dialog can be either modal or modeless.
- A modal dialog blocks user input to all other windows in the same application when it is visible. You have to close a modal dialog before other windows in the same application can get focus. A modeless one does not block user input.
- This class extends java.awt.Dialog class. Dialogs are used to accept some inputs from User.
- Default layout manager for JDialog class is BorderLayout.
- **Package :** javax.swing

- **Constructors :**

1. `JDialog(Frame parent)` : This constructor creates a new `JDialog` object which appears on specified parent `Frame`.
 2. `JDialog (Frame parent, Boolean modal)` : This constructor creates a new `JDialog` object which appears on the specified parent `frame`.
- If we pass second parameter as 'true' then we can not work on the parent window when dialog is visible, such a Dialog boxes are called as Modal dialog boxes.
 - If we pass 'false' then we can work on the parent window when dialog is visible such a dialog box is called as non-modal dialog box.
 1. `JDialog (Frame parent, String title)`: This constructor creates a dialog box which have some title.
 2. `JDialog (Frame parent, Siring title, Boolean modal)`: This constructor creates a Dialogbox which appears on specified parent frame, with title specified by the user and modal or non-modal nature is specified.

- **Methods :**

Name	Description
<code>void hide ()</code>	This method is used to hide the Dialog box.
<code>void show()</code>	This method is used to show the Dialog box.
<code>Container getContentPane()</code>	This method returns, a Content Pane for the <code>JDialog</code> .
<code>void setLayout (LayoutManager)</code>	This method sets the <code>LayoutManager</code> for the Dialog.
<code>void setJMenuBar (JMenuBar)</code>	This method sets the <code>JMenu Bar</code> to the <code>JDialog</code> box.
<code>boolean isModal ()</code>	This method checks whether, the dialog is modal or non-modal.

Program 1.27: Program to demonstrate `JDialog`.

```
import javax.swing.*;
import java.awt.*;
class JDialogExample extends JFrame
{
    JDialog dl;
    public JDialogExample()
    {
        createAndShowGUI();
    }
}
```

```
private void createAndShowGUI()
{
    setTitle("JDialog Example");
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setLayout(new FlowLayout());

    // Must be called before creating JDialog for
    // the desired effect
    JDialog.setDefaultLookAndFeelDecorated(true);

    // A perfect constructor, mostly used.
    // A dialog with current frame as parent
    // a given title, and modal
    dl=new JDialog(this,"This is title",true);

    // Set size
    dl.setSize(400,400);

    // Set some layout
    dl.setLayout(new FlowLayout());

    dl.add(new JButton("Button"));
    dl.add(new JLabel("Label"));
    dl.add(new JTextField(20));

    setSize(400,400);
    setVisible(true);

    // Like JFrame, JDialog isn't visible, you'll
    // have to make it visible
    // Remember to show JDialog after its parent is
    // shown so that its parent is visible
    dl.setVisible(true);
}

public static void main(String args[])
{
    new JDialogExample();
}
}
```

1.5.5.5 JLabel

- One of the simplest Swing component is JLabel. A JLabel object is a component for placing text in a container.
- This class is used to create single line read only text which describes the other component.
- Labels can display text as well as Images.
- **Package:** javax.swing
- **Constructors:**
 1. JLabel() : Create a empty label having no text and icon.
 2. JLabel(String str) : Create a label having some text.
 3. JLabel(ImageIcon i) : Creates a label having icon image.
 4. JLabel(String str, ImageIcon i) : Creates a label having string text and icon image.
- **Methods:**

Name	Description
void setHorizontalAlignment(int a)	This method sets the alignment of the text.
String getText()	This method returns the text associated with the Label.
void setText(String s)	This method is used to set the text to the Label.
void set icon(Icon i)	This method sets the Icon to the icon.

Program 1.28: Program to demonstrate JLabel.

```
import java.awt.Dimension;
import java.awt.Frame;
import java.awt.Rectangle;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextArea;
public class Frame1 extends JFrame
{
    private JLabel jLabel1 = new JLabel();
    private JLabel jLabel2 = new JLabel();
    public Frame1()
    {
```

```
        try
        {
            jbInit();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
    private void jbInit() throws Exception
    {
        this.getContentPane().setLayout( null );
        this.setSize( new Dimension(400, 300) );
        jLabel1.setText("UserName");
        jLabel1.setBounds(new Rectangle(40, 55, 80, 25));
        jLabel2.setText("Password");
        jLabel2.setBounds(new Rectangle(40, 95, 60, 25));
        this.getContentPane().add(jLabel2, null);
        this.getContentPane().add(jLabel1, null);
    }
    public static void main(String args[])
    {
        Frame1 frame=new Frame1();
        frame.setVisible(true);
    }
}
```

1.5.5.6 Icons

- Icons are encapsulated by the ImageIcon class, which prints an icon from an image.
- Constructors are:
 1. ImageIcon(String filename): This constructor creates object with image which is specified by filename.
 2. ImageIcon(URL url): This constructor creates object with image in the resource identified by url.

- **Methods:**

1. `int getIconHeight()`: Returns the height of the icon in pixels.
 2. `int getIconWidth()`: Returns the width of the icon in pixels.
 3. `void paintIcon(Component comp, Graphics g, int x, int y)`: Paints the icon at position `x`, `y` on the graphics context `g`. Additional information about the paint operation can be provided in component.
-

Program 1.29: Program of demonstrate icons.

```
import java.awt.*;
import javax.swing.*;
/*
<applet code="JLabelDemo" width=250 height=150>
</applet>
*/
public class JLabelDemo extends JApplet {
public void init()
{
// Get content pane
Container contentPane = getContentPane();
// Create an icon
ImageIcon ii = new ImageIcon("Lotus.gif");
// Create a label
JLabel jl = new JLabel("Lotus", ii, JLabel.CENTER);
// Add label to the content pane
contentPane.add(jl);
}
}
```

1.5.5.7 JTextField

- The `JTextField` component allows us to enter/edit a single line of text.
- Following are the important constructors of the subclasses of the `JTextField` class.
 1. `JTextField()`: Constructs a new `TextField`.
 2. `JTextField(Document doc, String text, int columns)`: Constructs a new `JTextField` that uses the given text storage model and the given number of columns.
 3. `JTextField(int columns)`: Constructs a new empty `TextField` with the specified number of columns.

4. `JTextField(String text)`: Constructs a new `TextField` initialized with the specified text.
 5. `JTextField(String text, int columns)`: Constructs a new `TextField` initialized with the specified text and columns.
-

Program 1.30: Program to demonstrate `JTextField`.

```
import java.awt.*;
import javax.swing.*;
public class TextFieldDemo extends JFrame
{
    public TextFieldDemo()
    {
        Container con = getContentPane();
        con.setLayout(new FlowLayout());
        JLabel jl = new JLabel ("TextField");
        con.add(jl);
        JTextField tf1 = new JTextField(40);
        con.add(tf1);
        setSize(600, 600);
        setVisible(true);
    }
    public static void main(String args[])
    {
        new TextFieldDemo();
    }
}
```

- The above Program 1.30, create a frame window in Swing. We then set its existing layout to `FlowLayout`. We then create a Swing label along with a text field component and add them to the content pane.

1.5.5.8 JTextArea

- The `JTextArea` is used to accept several lines of text from the user and it has capabilities not found in the AWT class.
- `JTextArea` class itself does not handle scrolling but implements the `Scrollable` interface which allows it to be placed inside a `JScrollPane` and thus implement scrolling.
- Text wrapping is possible through the use of a bound property of `JTextArea` class.

- A JTextArea can be created using one of the following constructors:
 1. JTextArea() : Constructs a new TextArea.
 2. JTextArea(Document doc): Constructs a new JTextArea with the given document model and defaults for all of the other arguments (null, 0, 0).
 3. JTextArea(Document doc, String text, int rows, int columns): Constructs a new JTextArea with the specified number of rows and columns and the given model.
 4. JTextArea(int rows, int columns): Constructs a new empty TextArea with the specified number of rows and columns.
 5. JTextArea(String text): Constructs a new TextArea with the specified text displayed.
 6. JTextArea(String text, int rows, int columns): Constructs a new TextArea with the specified text and number of rows and columns.
-

Program 1.31: Program to demonstrate JTextArea.

```
import java.awt.*;
import javax.swing.*;

public class TextAreaappl extends JFrame {
    public TextAreaappl() {
        Container con = getContentPane();
        con.setLayout(new FlowLayout());
        JLabel j11 = new JLabel ("TextArea");
        con.add(j11);
        JTextArea ta1 = new JTextArea(10, 20);
        con.add(ta1);
        setSize(600, 600);
        setVisible(true);
    }
    public static void main(String args[])
    {
        new TextAreaappl();
    }
}
```

- The above program creates a standalone Swing application having a frame window with a text area and a label. To create the text area that is ten rows and twenty columns wide we use the constructor:

```
JTextArea ta1= new JTextArea(10, 20);
```

1.5.5.9 JButton

- This class is used to create a push buttons.
- **Package:** javax.swing
- **Constructors:**
 1. JButton() : Create a empty button having no title and icon.
 2. JButton(String str) : Create a button having label.
 3. JButton(Icon i) : Creates a button having icon image.
 4. JButton(String str, Icon i): Creates a button having siring label and icon image.
- **Methods:** This class is subclass of Abstract Button class. So it uses following important methods of abstract button class.

Name	Description
void addActionListener(ActionListener obj)	This method is used to register the push button to throw an event.
String getText()	This method returns the text associated with the button.
void setText(String s)	This method is used to set new Label of the button
void setHorizontalTextPosition(int pos)	Sets the horizontal text position relative to the graphics.
void setVerticalTextPosition(int pos)	Sets the vertical text position relative to the graphics.
void setIcon(Icon i)	This method sets the Icon to the button.
void setMnemonic(char c)	Sets the mnemonic character to the button.

Program 1.32: Program of demonstrate JButton.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="JButtonDemo" width=250 height=300>
</applet>
*/
```

```
public class JButtonDemo extends JApplet implements ActionListener
{
    JTextField jtf;
    public void init()
    {
        // Get content pane
        Container contentPane = getContentPane();
        contentPane.setLayout(new FlowLayout());
        // Add buttons to content pane
        ImageIcon lotus = new ImageIcon("lotus.gif");
        JButton jlb = new JButton("Lotus");
        jlb.setActionCommand("Lotus");
        jlb.addActionListener(this);
        contentPane.add(jlb);
        ImageIcon rose = new ImageIcon("rose.gif");
        jlb = new JButton("Rose");
        jlb.setActionCommand("Rose");
        jlb.addActionListener(this);
        contentPane.add(jlb);
        // Add text field to content pane
        jtf = new JTextField(15);
        contentPane.add(jtf);
    }
    public void actionPerformed(ActionEvent ae)
    {
        jtf.setText(ae.getActionCommand());
    }
}
```

1.5.5.10 JRadioButton

- This class is used to create radio button with a text or icon.
- Radio buttons are supported by the JRadioButton class, which is a concrete implementation of AbstractButton.
- Once, a radio button is created they must be kept in one group, which is created by using the ButtonGroup class. Radio buttons must be configured into a group.
- Only one of the buttons in that group can be selected at any time.
- **Package:** javax.swing

Constructors:

1. `JRadioButton()`: Create a empty radio button having no title and icon.
2. `JRadioButton (String l)`: Create a radio button having label.
3. `JRadioButton (Icon i)`: Creates a radio button having icon image.
4. `JRadioButton (String l, Icon i)`: Creates a radio button having string label and icon image.
5. `JRadioButton (String l, Icon i, boolean selected)`: Creates a radio button having string label and icon image and default selected if true is passed.
6. `JRadioButton (String l, boolean selected)`: Creates a radio button having string label and option for default selected policy.

Button Group:

- This class is useful to group the component such as radio buttons.
 - If radio button is not grouped, then they work as checkbox. And multiple selection is possible.
 - To avoid that we group the Radio button object. This class provides the default constructor `ButtonGroup` by using this constructor object of button group class is created.
 - Then different component are added to the button group object by using `add()` method.
-

Program 1.33: Program to demonstrate `JRadioButton`.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="JRadioButtonDemo" width=300 height=50>
</applet>
*/
public class JRadioButtonDemo extends JApplet implements
ActionListener
{
    JTextField tf;
    public void init()
    {
        Container contentPane = getContentPane();
```

```

contentPane.setLayout(new FlowLayout());
JRadioButton b1 = new JRadioButton("C++");
b1.addActionListener(this);
contentPane.add(b1);
JRadioButton b2 = new JRadioButton("Java");
b2.addActionListener(this);
contentPane.add(b2);
JRadioButton b3 = new JRadioButton("SmallTalk");
b3.addActionListener(this);
contentPane.add(b3);
ButtonGroup bg = new ButtonGroup();
bg.add(b1);
bg.add(b2);
bg.add(b3);
tf = new JTextField(5);
contentPane.add(tf);
}
public void actionPerformed(ActionEvent ae)
{
tf.setText(ae.getActionCommand());
}

```

1.5.5.11 JToggleButton

- A toggle button is two-states button that allows user to switch on and off.
- To create a toggle button in Swing you use JToggleButton class.
- JToggleButton, when goes to the inward push state as long as the user has pressed the left mouse key and when he free the left mouse key, the button comes to its normal state.
- Here, are the most common used constructors of the JToggleButton class:

JToggleButton Constructors	Meanings
public JToggleButton()	Creates a toggle button without text and icon. The state of toggle button is not selected.
public JToggleButton(Icon icon)	Creates a toggle button with icon.
public JToggleButton(Icon icon, boolean selected)	Creates a toggle button with icon and initialize the state of toggle button by the boolean parameter selected.

contd. ...

<code>public JToggleButton(String text)</code>	Creates a toggle button with text.
<code>public JToggleButton(String text, boolean selected)</code>	Creates a toggle button with text and initialize the state of the toggle button.
<code>public JToggleButton(String text, Icon icon)</code>	Creates a toggle button which displays both text and icon.
<code>public JToggleButton(String text, Icon icon, boolean selected)</code>	Creates a toggle button which displays both text and icon. The state of toggle button can be initialized.

Program 1.34: Program to demonstrate JToggleButton.

```
import java.awt.BorderLayout;
import java.awt.Container;
import javax.swing.JFrame;
import javax.swing.JToggleButton;
public class ToggleButtonSample {
    public static void main(String args[]) {
        JFrame f = new JFrame("JToggleButton Sample");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Container content = f.getContentPane();
        content.add(new JToggleButton("North"), BorderLayout.NORTH);
        content.add(new JToggleButton("East"), BorderLayout.EAST);
        content.add(new JToggleButton("West"), BorderLayout.WEST);
        content.add(new JToggleButton("Center"), BorderLayout.CENTER);
        content.add(new JToggleButton("South"), BorderLayout.SOUTH);
        f.setSize(300, 200);
        f.setVisible(true);
    }
}
```

1.5.5.12 JCheckBox

- A checkbox is a control that may be turned ON or OFF by the user to indicate some option.
- The class JCheckBox is an implementation of a check box - an item that can be selected or deselected, and which displays its state to the user.
- The JCheckBox class is used to create CheckBox in Swing.

- **CheckBox** has the following constructors:
 1. `JCheckBox()`: Creates an initially unselected check box button with no text, no icon.
 2. `JCheckBox(Action a)`: Creates a check box where properties are taken from the Action supplied.
 3. `JCheckBox(Icon icon)`: Creates an initially unselected check box with an icon.
 4. `JCheckBox(Icon icon, boolean selected)`: Creates a check box with an icon and specifies whether or not it is initially selected.
 5. `JCheckBox(String text)`: Creates an initially unselected check box with text.
 6. `JCheckBox(String text, boolean selected)`: Creates a check box with text and specifies whether or not it is initially selected.
 7. `JCheckBox(String text, Icon icon)`: Creates an initially unselected check box with the specified text and icon.
 8. `JCheckBox(String text, Icon icon, boolean selected)`: Creates a check box with text and icon, and specifies whether or not it is initially selected.
-

Program 1.35: Program to demonstrate `JCheckBox`.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class HobbyOfStudent extends JPanel implements ActionListener,
ItemListener{
    JCheckBox ch1 = new JCheckBox("Playing", false);
    JCheckBox ch2 = new JCheckBox("Music", false);
    JCheckBox ch3 = new JCheckBox("Painting", false);
    JCheckBox ch4 = new JCheckBox("Reading", false);
    JLabel jl = new JLabel("What's student hobby?");
    JButton exitbtn = new JButton("Exit");
    public HobbyOfStudent()
    {
        setLayout(new GridLayout(7,1));
        ch1.setFont(new Font("Arial", Font.BOLD | Font.ITALIC, 14));
        ch2.setFont(new Font("Arial", Font.BOLD | Font.ITALIC, 14));
        ch3.setFont(new Font("Arial", Font.BOLD | Font.ITALIC, 14));
        ch4.setFont(new Font("Arial", Font.BOLD | Font.ITALIC, 14));
```

```
    ch1.addItemListener(this);
    ch2.addItemListener(this);
    ch3.addItemListener(this);
    ch4.addItemListener(this);
    add(j1);
    add(ch1);
    add(ch2);
    add(ch3);
    add(ch4);
    add(exitbtn);
    exitbtn.addActionListener(this);
    {
    public void actionPerformed(ActionEvent e)
    {
        if(e.getSource().equals(exitbtn)
        {
            System.exit(0);
        }
    }
    public void itemStateChanged(ItemEvent e) {
        String selected= ((JCheckBox)e.getSource()).getText();
        System.out.println(selected);
    }
    }
class HobbyTest extends JFrame
{
    HobbyTest()
    {
        super();
        getContentPane().add(new HobbyOfStudent());
        setSize(600,600);
        setVisible(true);
    }
    public static void main(String args[]) {
        new HobbyTest();
    }
}
```

1.5.5.13 JList

- In Swing, the JList component is the implementation of the AWT List class.
- JList consists of a range of elements arranged one after another, which can be selected individually or in a group.
- JList class, unlike its AWT counterpart, is capable of displaying not just the strings, but also icons.
- Some of the constructors used for creating JList are explained below:
 1. `public JList()`: Constructs a JList with an empty model.
 2. `public JList(ListModel dataModel)`: Constructs a JList() that displays the elements in the specified, non-null list model.
 3. `public JList(Object [] listData)`: Constructs a JList() that displays the elements of the specified array "listData".
- JList() does not support scrolling. To create a scrolling list the JList() is implemented as the viewport view of a JScrollPane.

```
JScrollPane myScrollPane = new JScrollPane();  
myScrollPane.getViewport().setView(dataList);
```

OR

```
JScrollPane myScrollPane = new JScrollPane(dataList);
```

- JList does not provide any special support for handling double or triple mouse clicks. However, using the MouseListener makes it easy to handle these events. The `locationToIndex()` method is used to determine the cell that was checked.
-

Program 1.36: Program to demonstrate JList.

```
import java.awt.*;  
import java.awt.event.*;  
import javax.swing.*;  
import javax.swing.event.*;  
  
class Employees extends JFrame implements ListSelectionListener  
{  
    String Employee[] = {"Amar Salunkhe", "Akbar Shaikh",  
        "Amol Mahabal", "Kiran Velankar"};  
    JList departments = new JList(Employees);  
    JLabel lTE = new  
        JLabel("Who is your favourite Technical Editor?");  
    JTextField jt = new JTextField(40);
```

```
public Employees(String s) {
    super(s);
    JPanel p = (JPanel) getContentPane();
    p.setLayout(new BorderLayout());
    p.add("North", lte);
    departments.setSelectionMode(
        ListSelectionModel.SINGLE_SELECTION);
    departments.setSelectedIndex(0);
    departments.addListSelectionListener(this);
    departments.setBackground(Color.LightRed);
    departments.setForeground(Color.black);
    p.setBackground(Color.white);
    p.setForeground(Color.black);
    p.add("Center", new JScrollPane(departments));
    p.add("South", jt)
}

public static void main(String args[])
{
    Employees e1 = new Employees ("Editor of Engineering Books");
    e1.setSize(500,500);
    e1.show();
}

public void valueChanged(ListSelectionEvent e)
{
    if(e.getValueIsAdjusting() == false)
    {
        if(departments.getSelectedIndex() != -1)
        {
            jt.setText((String)departments.getSelectedValue());
        }
    }
}
}
```

- In JList, events are handled by implementing the ListSelectionListener interface.
 1. `public void addListSelectionListener(ListSelectionListener listener)`: This method adds a listener to the list.
 2. `public void valueChanged(ListSelectionEvent e)`: This method is called whenever the value of the selection changes.

1.5.5.14 Combo Boxes

- Swing provides a combo box (a combination of a text field and a dropdown list) through the JComboBox class, which extends JComponent. A combobox normally displays one entry.
- However, it can also display a drop-down list that allows a user to select a different entry. We can also type our selection into the text field.
- The JComboBox component, similar to Choice component in AWT, is a combination of textfield and drop down list of items. User can make selection by clicking at an item from the list or by typing into the box.
- Three of JComboBox's constructors are shown here:
 1. `JComboBox()`: This constructor creates an empty JComboBox instance.
 2. `public JComboBox (ComboBoxModel asModel)`: Creates a JComboBox that takes its items from an existing ComboBoxModel. asModel is the ComboBoxModel that provides the displayed list of items.
 3. `public JComboBox (Object[] items)`: Creates a JComboBox that contains the elements of the specified array
- Items are added to the list of choices via the `addItem()` method, whose signature/syntax is shown here:

```
void addItem(Object obj)
```

Here, obj is the object to be added to the combo box.

Methods:

1. `public void setEditable(boolean aFlag)`: It determines whether the JComboBox field is editable or not?
2. `public boolean isEditable()`: It returns true if the JComboBox is editable. By default, a combo box is not editable.
3. `public void setMaximumRowCount(int count)`: It sets the maximum number of rows the JComboBox displays. If the number of objects in the model is greater than 'count', the combo box uses a scrollbar.
4. `public void setSelectedItem(Object anObject)`: It sets the selected item in the combo box display area to the object in the argument. If an Object is in the list, the display area shows an Object selected.

5. `public void insertItemAt(Object anObject, int index)`: It inserts an item 'anObject' into the item list at a given 'index'.
 6. `public void removeItem(Object anObject)`: It removes an item 'anObject' from the item list.
 7. `public void removeItemAt(int anIndex)`: It removes the item at 'anIndex'.
- The following example contains a combo box and a label. The label displays an icon. The combo box contains entries for colors Green, Red, Yellow and Black. When a country is selected, the label is updated to display the color for that particular color. Color jpeg images are already stored in the current directory.
-

Program 1.37: Program to demonstrate JComboBox.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="JComboBoxDemo" width=300 height=100>
</applet>
*/
public class JComboBoxDemo extends JApplet
implements ItemListener
{
    JLabel jl;
    ImageIcon green, red, black, yellow;
    public void init()
    {
        Container contentPane = getContentPane();
        contentPane.setLayout(new FlowLayout());
        JComboBox jc = new JComboBox();
        jc.addItem("Green");
        jc.addItem("Red");
        jc.addItem("Black");
        jc.addItem("Yellow");
        jc.addItemListener(this);
        contentPane.add(jc);
        jl = new JLabel(new ImageIcon("green.jpg"));
        contentPane.add(jl);
    }
    public void itemStateChanged(ItemEvent ie)
    {
        String s = (String)ie.getItem();
        jl.setIcon(new ImageIcon(s + ".jpg"));
    }
}
```

1.5.5.15 Progress Bar

- Progress indicators or progress bars are the new controls that give the users some indications of the progress of an operation.
- A `JProgressBar` is a Swing component that indicates progress. A `ProgressMonitor` is a dialog box that contains a progress bar.
- The class `JProgressBar` is a component which visually displays the progress of some task.
- A progress bar is a widget that displays progress of a lengthy task, for instance file download or transfer.
- To create a progress bar in swing you use the `JProgress` bar class. With `JProgressBar` you can either create vertical or horizontal progress bar.
- In addition `JProgressBar` class allows you to create another kind of progress bar which is known as indeterminate progress bar. The indeterminate progress bar is used to display progress with unknown progress or the progress cannot express by percentage.
- Here are the constructors of `JProgressBar` class:
 1. `public JProgressBar()`: Creates a horizontal progress bar.
 2. `public JProgressBar(BoundedRangeModel model)`: Creates a progress bar using `BoundedRangeModel` instance to hold data.
 3. `public JProgressBar(int orient)`: Creates a progress bar with a given orientation determined by `SwingConstants.VERTICAL` or `SwingConstants.HORIZONTAL`.
 4. `public JProgressBar(int min, int max)`: Creates a progress bar with minimum and maximum values.
 5. `public JProgressBar(int orient, int min, int max)`: Creates a progress bar with minimum and maximum values and progress bar's orientation.
- This is the class which creates the progress bar using its constructor `JProgressBar()` to show the status of your process completion.
- The constructor `JProgressBar()` takes two arguments as parameters in which, first is the initial value of the progress bar which is shown in the starting and another argument is the counter value by which the value of the progress bar is incremented. Here, the value of the progress bar is incremented by 20.

setStringPainted(boolean):

- This is the method of the `JProgressBar` class which shows the complete process in percent on the progress bar. It takes a boolean value as a parameter. If you pass the true then the value will be seen on the progress bar otherwise not seen.

setValue():

- This is the method of the JProgressBar class which sets the value to the progress bar.

Timer():

- This the constructor of the Timer class which starts the timer for timing. This constructor takes two argument as parameter first is the interval (in milliseconds) of the timer and second one is the listener object. Time is started using the start() method of the Timer class.
-

Program 1.38: Program to demonstrate progress bar.

```
import java.awt.*;
import javax.swing.*;
public class Main {
    public static void main(String[] args) {
        final int MAX = 100;
        final JFrame frame = new JFrame("JProgress Demo");

        // creates progress bar
        final JProgressBar pb = new JProgressBar();
        pb.setMinimum(0);
        pb.setMaximum(MAX);
        pb.setStringPainted(true);

        // add progress bar
        frame.setLayout(new FlowLayout());
        frame.getContentPane().add(pb);

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 200);
        frame.setVisible(true);

        // update progressbar
        for (int i = 0; i <= MAX; i++) {
            final int currentValue = i;
            try {
                SwingUtilities.invokeLater(new Runnable() {
                    public void run() {
                        pb.setValue(currentValue);
                    }
                });
                java.lang.Thread.sleep(100);
            } catch (InterruptedException e) {
                JOptionPane.showMessageDialog(frame, e.getMessage());
            }
        }
    }
}
```

OR

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JProgressBar;

public class Main extends JFrame {
    JProgressBar current = new JProgressBar(0, 2000);
    int num = 0;
    public Main() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel pane = new JPanel();
        current.setValue(0);
        current.setStringPainted(true);
        pane.add(current);
        setContentPane(pane);
    }

    public void iterate() {
        while (num < 2000) {
            current.setValue(num);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
            }
            num += 95;
        }
    }

    public static void main(String[] arguments) {
        Main frame = new Main();
        frame.pack();
        frame.setVisible(true);
        frame.iterate();
    }
}
```

1.5.5.16 Tool Tips

- Tooltips are part of the internal application's help system. The Swing shows a small rectangular window, if we hover a mouse pointer over an object.
 - Creating a tool tip for any JComponent object is easy. Use the `setToolTipText()` method to set up a tool tip for the component.
 - For example, to add tool tips to three buttons, you add only three lines of code:

```
b1.setToolTipText("Click this button to disable the middle button.");
b2.setToolTipText("This middle button does not react when you click it.");
b3.setToolTipText("Click this button to enable the middle button.");
```
 - When the user of the program pauses with the cursor over any of the program's buttons, the tool tip for the button comes up.
 - For components such as tabbed panes that have multiple parts, it often makes sense to vary the tool tip text to reflect the part of the component under the cursor. For example, a tabbed pane might use this feature to explain what will happen when you click the tab under the cursor.
 - When you implement a tabbed pane, you can specify the tab-specific tool tip text in an argument passed to the `addTab` or `setToolTipTextAt()` method.
-

Program 1.39: Program to demonstrate tooltip.

```
import javax.swing.*;
import java.awt.*;

public class TooltipExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Tool Tip Demo");
        frame.setSize(200, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel label = new JLabel("Hover on me!");
        // Setting tool tip for our Swing JLabel component
        label.setToolTipText("My JLabel Tool Tip");
        frame.getContentPane().setLayout(new
            FlowLayout(FlowLayout.CENTER));
        frame.getContentPane().add(label);
        frame.setVisible(true);
    }
}
```

1.5.5.17 Separators (JSeparator)

- The JSeparator class provides a horizontal or vertical dividing line or empty space. It's most commonly used in menus and tool bars.
- In fact, you can use separators without even knowing that a JSeparator class exists, since menus and tool bars provide convenience methods that create and add separators customized for their containers.
- Separators are somewhat similar to borders, except that they are genuine components and, as such, are drawn inside a container, rather than around the edges of a particular component.
- Here, is a picture of a menu that has one separator, used to divide the menu into two groups of items.
- The code to add the menu items and separators to the menu is extremely simple, boiling down to something like this:

```
menu.add(menuItem1);
menu.add(menuItem2);
menu.add(menuItem3);
menu.addSeparator();
menu.add(rbMenuItem1);
menu.add(rbMenuItem2);
menu.addSeparator();
menu.add(cbMenuItem1);
menu.add(cbMenuItem2);
menu.addSeparator();
menu.add(submenu);
```

- Adding separators to a tool bar is similar.

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JSeparator;
```

Program 1.40: Program for JSeparator.

```
public class MenuSeparator extends JFrame {
    public MenuSeparator() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
JMenuBar bar = new JMenuBar();
JMenu menu = new JMenu("File");
bar.add(menu);
menu.add(new JMenuItem("Close"));
menu.add(new JSeparator()); // SEPARATOR
menu.add(new JMenuItem("Exit"));
setJMenuBar(bar);
getContentPane().add(new JLabel("A placeholder"));
pack();
setSize(300, 300);
setVisible(true);
}
public static void main(String arg[]) {
    new MenuSeparator();
}
}
```

1.5.5.18 **Tabbed Panes (JTabbedPane)**

- A tabbed pane is a component that appears as a group of folders in a file cabinet. Each folder has a title.
- When a user selects a folder, its contents become visible. Only one of the folders may be selected at a time.
- Tabbed panes are commonly used for setting configuration options. Tabbed panes are encapsulated by the JTabbedPane class, which extends JComponent.
- There are three constructors of JTabbedPane:
 1. `JTabbedPane()` creates an empty: TabbedPane with a default tab placement of `JTabbedPane.TOP`.
 2. `JTabbedPane(int tabPlacement)`: Creates an empty TabbedPane with the specified tab placement of any of the following:

```
JTabbedPane.TOP
JTabbedPane.BOTTOM
JTabbedPane.LEFT
JTabbedPane.RIGHT
```
 3. `JTabbedPane(int tabPlacement, int tabLayoutPolicy)`: Creates an empty TabbedPane with the specified tab placement and tab layout policy.

- Tab placements are listed above. Tab layout policy may be either of the following:

```
JTabbedPane.WRAP_TAB_LAYOUT
```

```
JTabbedPane.SCROLL_TAB_LAYOUT
```

- Tabs are defined via the following method:

```
void addTab(String str, Component comp)
```

- Here, `str` is the title for the tab, and `comp` is the component that should be added to the tab. Typically, a `JPanel` or a subclass of it is added.
 - The general procedure to use a tabbed pane in an applet is outlined here:
 1. Create a `JTabbedPane` object.
 2. Call `addTab()` to add a tab to the pane. (The arguments to this method define the title of the tab and the component it contains.)
 3. Repeat step 2 for each tab.
 4. Add the tabbed pane to the content pane of the applet.
 - The following example illustrates how to create a tabbed pane. The first tab is titled Languages and contains four buttons. Each button displays the name of a language. The second tab is titled Colors and contains three checkboxes. Each check box displays the name of a color. The third tab is titled Flavors and contains one combo box. This enables the user to select one of three flavors.
-

Program 1.41: Program to demonstrate `JTabbedPane`.

```
import javax.swing.*;
/*
<applet code="JTabbedPaneDemo" width=400 height=100>
</applet>
*/
public class JTabbedPaneDemo extends JApplet
{
public void init()
{
JTabbedPane jtp = new JTabbedPane();
jtp.addTab("Languages", new LangPanel());
jtp.addTab("Colors", new ColorsPanel());
jtp.addTab("Flavors", new FlavorsPanel());
getContentPane().add(jtp);
}
}
```

```
class LangPanel extends JPanel
{
public LangPanel()
{
    JButton b1 = new JButton("Marathi");
    add(b1);
    JButton b2 = new JButton("Hindi");
    add(b2);
    JButton b3 = new JButton("Bengali");
    add(b3);
    JButton b4 = new JButton("Tamil");
    add(b4);
}
}
class ColorsPanel extends JPanel
{
public ColorsPanel()
{
    JCheckBox cb1 = new JCheckBox("Red");
    add(cb1);
    JCheckBox cb2 = new JCheckBox("Green");
    add(cb2);
    JCheckBox cb3 = new JCheckBox("Blue");
    add(cb3);
}
}
class FlavorsPanel extends JPanel
{
public FlavorsPanel()
{
    JComboBox jcb = new JComboBox();
    jcb.addItem("Vanilla");
    jcb.addItem("Chocolate");
    jcb.addItem("Strawberry");
    add(jcb);
}
}
```

1.5.5.19 JScrollPane

- A scroll pane is a component that presents a rectangular area in which a component may be viewed.
- Horizontal and/or vertical scroll bars may be provided if necessary.
- Scroll panes are implemented in Swing by the JScrollPane class.
- Constructors:
 1. `JScrollPane(Component comp)`: This creates a scroll pane with the component specified with `comp`.
 2. `JScrollPane(int vsb, int hsb)`: This creates a scroll panel with the `vsb` and `hsb` are `int` constants that define when vertical and horizontal scroll bars for this scroll panel are shown.
 3. `JScrollPane(Component comp, int vsb, int hsb)`: This constructor scroll panel creates a combination of above both constructor.

- Constants of scroll pane are:

Constant	Description
<code>HORIZONTAL_SCROLLBAR_ALWAYS</code>	: Always provide horizontal scroll bar.
<code>HORIZONTAL_SCROLLBAR_AS_NEEDED</code>	: Provide horizontal scroll bar, if needed.
<code>VERTICAL_SCROLLBAR_ALWAYS</code>	: Always provide vertical scroll bar.
<code>VERTICAL_SCROLLBAR_AS_NEEDED</code>	: Provide vertical scroll bar, if needed.

- Here, are the steps that you should follow to use a scroll pane in an applet:
 1. Create a `JComponent` object.
 2. Create a `JScrollPane` object, (the arguments to the constructor specify the component and the policies for vertical and horizontal scroll bars.)
 3. Add the scroll pane to the content pane of the applet.

Program 1.42: Program to demonstrate `JScrollPane`.

```
import java.awt.*;
import javax.swing.*;
/*
<applet code="JScrollPaneDemo" width=300 height=250>
</applet>
*/
public class JScrollPaneDemo extends JApplet
{
    public void init()
    {
        container contentPane = getContentPane();
        contentPane.setLayout(new BorderLayout());
        JPanel jp = new JPanel();
        jp.setLayout(new GridLayout(20, 20));
        int b = 0;
```

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

int v = ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED;
int h = ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED;
JScrollPane jsp = new JScrollPane(jp, v, h);
contentPane.add(jsp, BorderLayout.CENTER);
}
}
```

1.5.5.20 JTree

- A tree is a component that presents a hierarchical view of data. A user has the ability to expand or collapse individual sub-trees in this display.
- Any computer user who had used Windows Explorer or File Manager will recall seeing a tree-like structure depicting files and folders, (See Fig. 1.14).
- This tree-like structure will display folders and subfolders like branches of a tree one below the other.

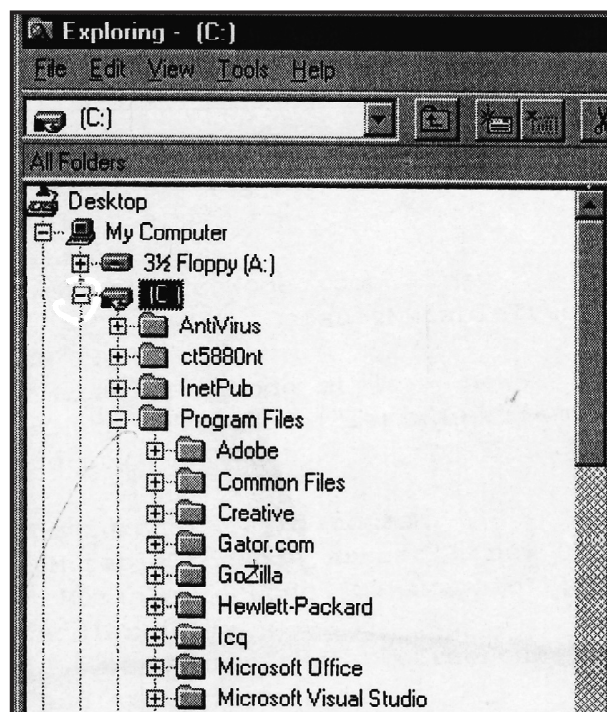


Fig. 1.14

- Similar, tree-like structures can be created in Java using JTree.
- Trees are implemented in Swing by the JTree class, which extends JComponent.
- Some of its constructors are shown here:
 1. `JTree(Hashtable ht)`: Creates a tree in which each element of the hash table `ht` is a child node.
 2. `JTree(Object obj[])`: Each element of the array `obj` is a child node in the second form.
 3. `JTree(TreeNode tn)`: The tree node `tn` is the root of the tree in the third form.
 4. `JTree(Vector v)`: Uses the elements of vector `v` as child nodes. A JTree object generates events when a node is expanded or collapsed.
- The `addTreeExpansionListener()` and `removeTreeExpansionListener()` methods allow listeners to register and unregister for these notifications.
- The signatures/syntax of these methods are shown below:

```
void addTreeExpansionListener(TreeExpansionListener tel)
void removeTreeExpansionListener(TreeExpansionListener tel)
```

Here, `tel` is the listener object.

- The `getPathForLocation()` method is used to translate a mouse click on a specific point of the tree to a tree path. Its syntax is shown here:

```
TreePath getPathForLocation(int x, int y)
```

Here, `x` and `y` are the coordinates at which the mouse is clicked. The return value is a `TreePath` object that encapsulates information about the tree node that was selected by the user.

- The `TreePath` class encapsulates information about a path to a particular node in a tree. It provides several constructors and methods.
- The `DefaultMutableTreeNode` class implements the `MutableTreeNode` interface. It represents a node in a tree. One of its constructors is shown here:

```
DefaultMutableTreeNode(Object obj)
```

Here, `obj` is the object to be enclosed in this tree node. The new `TreeNode` doesn't have a parent or children.

- To create a hierarchy of tree nodes, the `add()` method of `DefaultMutableTreeNode` can be used. Its syntax is:

```
void add(MutableTreeNode child)
```

Here, `child` is a mutable tree node that is to be added as a child to the current node.

- Tree expansion events are described by the class `TreeExpansionEvent` in the `javax.swing.event` package. The `getPath()` method of this class returns a `TreePath` object that describes the path to the changed node. Its syntax is shown here:

```
TreePath getPath( )
```

Here, `tee` is the tree expansion event. The first method is called when a sub-tree is hidden, and the second method is called when a sub-tree becomes visible.

- Here, are the steps that we should follow to use a tree in an applet:
 1. Create a `JTree` object.
 2. Create a `JScrollPane` object, (The arguments to the constructor specify the tree and the policies for vertical and horizontal scroll bars).
 3. Add the tree to the scroll pane.
 4. Add the scroll pane to the content pane of the applet.
-

Program 1.43: Program to demonstrate `JTree`.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.tree.*;
/*
<applet code="JTreeEvents" width=400 height=200>
</applet>
*/
public class JTreeEvents extends JApplet
{
    JTree tree;
    JTextField jtf;
    public void init()
    {
        Container contentPane=getContentPane();
        contentPane.setLayout(new BorderLayout());
        DefaultMutableTreeNode top=new
        DefaultMutableTreeNode("Options");
        DefaultMutableTreeNode a= new DefaultMutableTreeNode("A");
```

```
top.add(a);
DefaultMutableTreeNode a1=new DefaultMutableTreeNode("A1");
a.add(a1);
DefaultMutableTreeNode a2=new DefaultMutableTreeNode("A2");
a.add(a2);
DefaultMutableTreeNode b= new DefaultMutableTreeNode("B");
top.add(b);
DefaultMutableTreeNode b1=new DefaultMutableTreeNode("B1");
b.add(b1);
DefaultMutableTreeNode b2=new DefaultMutableTreeNode("B2");
b.add(b2);
DefaultMutableTreeNode b3=new DefaultMutableTreeNode("B3");
b.add(b3);
tree=new JTree(top);
int v=ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED;
int h=ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED;
JScrollPane jsp=new JScrollPane(tree,v,h);
contentPane.add(jsp, BorderLayout.CENTER);
jtf=new JTextField("",20);
contentPane.add(jtf, BorderLayout.SOUTH);
tree.addMouseListener(new MouseAdapter()
{
public void mouseClicked(MouseEvent me)
{
doMouseClicked(me);
}
});
void doMouseClicked(MouseEvent me)
{
TreePath tp=tree.getPathForLocation(me.getX(),me.getY());
if(tp!=null)
jtf.setText(tp.toString());
else
jtf.setText("");
}
}
```

1.5.5.21 JTable

- A table is a component that displays rows and columns of data. JTable class use to create tables for a GUI based application.
- We can drag the cursor on column boundaries to resize columns. We can also drag a column to a new position.
- Tables are implemented by the JTable class, which extends JComponent.
- One of its constructors is shown here:

```
JTable(Object data[ ][ ], Object colHeads[ ])
JTable(int numRows, int numColumns)
JTable(Vector rowData, Vector columnData)
```

Here, data is a two-dimensional array of the information to be presented and colHeads is a one-dimensional array with the column headings. The 'numRows' and 'numColumns' are values with which the table is to be created. The 'rowData' and 'columnData' are the vector values by which the table is constructed.

- Here, are the steps for using a table in an applet:
 1. Create a JTable object.
 2. Create a JScrollPane object (The arguments to the constructor specify the table and the policies for vertical and horizontal scroll bars).
 3. Add the table to the scroll pane.
 4. Add the scroll pane to the content pane of the applet.
- The following example illustrates how to create and use a table. The content pane of the JApplet object is obtained and a border layout is assigned as its layout manager.
- A one-dimensional array of strings is created for the column headings. This table has three columns. A two-dimensional array of strings is created for the table cells. We can see that each element in the array is an array of three strings. These arrays are passed to the JTable constructor. The table is added to a scroll pane and then the scroll pane is added to the content pane.

Program 1.44: Program to demonstrate JTable.

```
import java.awt.*;
import javax.swing.*;
/*
<applet code="JTableDemo" width=400 height=200>
</applet>
*/
```

```
public class JTableDemo extends JApplet
{
public void init()
{
Container contentPane = getContentPane();
contentPane.setLayout(new BorderLayout());
final String[] colHeads = { "Name", "Phone", "Fax" };
final Object[][] data = {{ "Prمود", "4567", "8675" },
{ "Tausif", "7566", "5555" },
{ "Nitin", "5634", "5887" },
{ "Amol", "7345", "9222" },
{ "Vijai", "1237", "3333" },
{ "Ranie", "5656", "3144" },
{ "Mangesh", "5672", "2176" },
{ "Suhail", "6741", "4244" },
{ "Nilofer", "9023", "5159" },
{ "Jinnie", "1134", "5332" },
{ "Heena", "5689", "1212" },
{ "Saurav", "9030", "1313" },
{ "Raman", "6751", "1415" }
};
JTable table = new JTable(data, colHeads);
int v = ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED;
int h = ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED;
JScrollPane jsp = new JScrollPane(table, v, h);
contentPane.add(jsp, BorderLayout.CENTER);
}
}
```

Important Points

- AWT contains number of classes, which all are placed in java.awt package.
- The two most common windows are Panel, which is used by applets and Frame, which creates a standard window.
- Component is an abstract class that contain all of the attributes required for a visual component.
- The Container class is a subclass of Component.
- A container is responsible for positioning any components that it contains.
- A Panel may be thought of as component. Panel is the superclass for Applet.
- A Panel is a window that does not contain a title bar, menu bar or border.
- The Window class creates a top-level window.
- Frame is commonly nothing but “window.” It is a subclass of Window and has a title bar, menu bar, borders and resizing corners.
- Canvas encapsulates a blank window upon which you can draw.
- The AWT color system allows you to define any color you want. Color is represented by the Color class.
- The AWT supports various type of fonts. The AWT allow for dynamic selection of fonts.
- Fonts are represented by the Font class.
- AWT includes the FontMetrics class, which encapsulates various information about a font.
- A push button is a component that contains a label and that generates an event when it is pressed.
- The TextField class allow the user to enter strings and to edit the text using the arrow keys, cut and paste keys.
- Sometimes, a single line of text input is not enough for a given task. For example, for entering address or brief description. To handle these situations, the AWT includes a simple multiline editor called TextArea.
- CheckBox consists of a small box that can either contain a check mark or not, (i.e. checked or unchecked).
- A Choice control takes enough space to show the item. When the user clicks on it, the whole list of items pops up and a selection can be made.
- The List class provides a multiple-choice, scrolling selection list.
- A layout manager automatically arranges your controls within a window.
- FlowLayout is the default layout manager for Panel or Applet.

- BorderLayout layout divide your window in five parts as north, south, center, east, west.
- 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.
- A menu bar displays a list of menu choices. Each choice is associated with a drop-down menu.
- A menu bar contains one or more Menu objects. Each Menu object contains a list of MenuItem objects. Each MenuItem object represents something that can be selected by the user. Since Menu is a subclass of MenuItem.
- Dialog boxes are primarily used to obtain user input.
- To create a file dialog box, instantiate an object of type FileDialog.
- Java foundation classes are nothing but class libraries specially designed for building GUI's. Foundation classes simply the designing process and reduce the time taken for coding.
- Swing is just one part of the Java Foundation Classes (JFC).
- A component is called light weight, when it does not depend on any non-java system classes. Swing components have their own view supported by java's look and feel classes.
- A JFrame component is used to create windows in a swing program.
- The JPanel component which is an intermediate container, is used to group smaller lightweight components together.
- Swing containers like JApplet, JFrame, JWindow and JDialog delegate their duties to the root pane represented by the class JRootPane.
- Dialogs are used to accept some inputs from User.
- Default layout manager for JDialog class is BorderLayout.
- Labels can display text as well as Images.
- Icons are encapsulated by the ImageIcon class, which prints an icon from an image.
- The JTextField component allows us to enter/edit a single line of text.
- The JTextArea is used to accept several lines of text from the user.
- JButton class is used to create a push buttons.
- JRadioButton class is used to create radio button with a text or icon.
- ButtonGroup class is useful to group the component such as radio buttons. If radio button is not grouped, then they work as checkbox. And multiple selection is possible.
- A checkbox is a control that may be turned ON or OFF by the user to indicate some option.

- JList class, unlike its AWT counterpart, is capable of displaying not just the strings, but also icons.
- A JComboBox is a combination of a text field and drop-down list that lets user either type in a value or select it from a list that is displayed when the user asks for it.
- A JTabbedPane is a component that appears as a group of folders in a file cabinet. Each folder has a title.
- A JScrollPane is a component that presents a rectangular area in which a component may be viewed.
- JTree object does not actually contain your data, it only provides a view of the data. JTree displays its data vertically. Every row in the hierarchy of the tree is termed as a node. Every tree has a root node from which all nodes descend.
- The JTable class in Swing enables us to create tables.

Practice Questions

1. What is AWT?
2. Explain what is the meaning of import java.awt. * line in Java.
3. List out the components, who supports ActionEvent ActionListener events?
4. What are different AWT controls? Explain any one in detail.
5. Construct a list to display the book list as shown below:
 - (a) Java in 21 days
 - (b) C++ Made Easy
 - (c) C Programming
 - (d) System Analysis and Design
 - (e) COBOL.
6. What is Frame? Explain in detail.
7. What is Panel? Explain in detail.
8. What is Layout?
9. List out different layouts and explain any one in detail.
10. Explain check box control with suitable programming example.
11. Write a program to design a form using components textbox, text field, checkbox, buttons, list and handle various events related to each component.
12. Write a program to design a calculator using Java components and handle various events related to each component and apply proper layout to it.
13. Write a program to demonstrate use of Grid Layout.

14. Write a program to demonstrate use of Flow Layout.
15. Write a program to demonstrate use of Card Layout.
16. Write a program to demonstrate use of Border Layout.
17. Write a program to create a menu bar with various menu items and sub menu items. Also create a checkable menu item. On clicking a menu Item display a suitable Dialog box.
18. Write a program to increase the font size of a font displayed when the value of thumb in scrollbar increases at the same time it decreases the size of the font when the value of font decreases.
19. What is swing?
20. Enlist various features of swing?
21. With the help of diagram describe following swing components:
 - (i) JComboBox
 - (ii) JCheckBox
 - (iii) JProgressBar
22. With the help of diagram describe MVC architecture.

