08 Window-Based Applications in Java

IN THIS CHAPTER

In today's scenario, people prefer Graphical User Interface (GUI) based applications over Character User Interface (CUI) based applications as the former provide a more interactive user interface.

A GUI-based application is an application in which a user interacts through various components, such as text fields and buttons. Java provides Abstract Window Toolkit (AWT) and the Swing Application Programming Interface (API) to create interactive GUI-based applications. You can use the `java.awt` and `javax.swing` packages, which contain various classes, interfaces, and methods, to create GUI-based applications. Moreover, you can change the layout of GUI components by using the layout manager and add menus along with submenus to these applications.

This chapter introduces AWT and its components, which are added to frame windows to create GUI-based applications. It also explains how AWT components are managed in the frame windows. In addition, this chapter discusses the various layout managers of Java that are required to arrange AWT components in a frame window. We also learn how to create menus in a frame window, which help to arrange AWT components in a systematic manner. Moreover, we learn how to create dialog boxes, which are used to provide information to a user about any action that is being performed in a GUI-based application. Finally, we learn how to create images to make the GUI-based application attractive.

Now, let's start with a detailed introduction to AWT.

8.1 Introducing AWT

AWT is an API that helps to create GUI-based (Windows) applications. The AWT API provides various classes, stored in the java.awt package, to create GUI components in the application. Using the `java.awt` package, you can create and display buttons, labels, check boxes, text fields, and other user interface components that you want to display in a Windows application.

Table 1 lists noteworthy classes provided in the java.awt package that help you create a GUI-based application:

Table 1: Noteworthy Classes of the AWT Package

Class	Description
Component	Serves as an abstract superclass for various AWT components, such as buttons and check boxes.
Container	Contains AWT components, such as buttons and check boxes.
Panel	Creates a panel for a Windows application.
Window	Provides the container where a Windows application can attach other components, including other panels. The Window class is a subclass of the Container class.
Frame	Creates a frame for a Windows application.

Now, let's describe these classes in greater detail in the following sections.

■ Exploring the Component Class

The `Component` class is the base class of the `java.awt` package on which all the visual components are based. For example, the Button class is directly derived from the Component class. The `Component` class also contains many methods that are used to create GUI-based applications. Table 2 lists noteworthy methods available in the `Component` class:

Table 2: Noteworthy Methods of the Component Class

Method	Description
`Font getFont()`	Returns the font of the text of a component.
`void setFont(Font ft)`	Sets the specified font for the text of a component.
`Color getForeground()`	Returns the foreground color of a component.
`void setForeground(Color c)`	Sets the foreground color for a component.
`Color getBackground()`	Returns the background color of a component.
`void setBackground(Color c)`	Sets the background color for a component.
`String getName()`	Returns the name of a component.
`void setName(String name)`	Sets a new name for a component.
`int getHeight()`	Returns the height of a component in pixels as an integer value.
`int getWidth()`	Returns the width of a component in pixels as an integer value.
`Dimension getSize()`	Returns the size of a component.
`int getX()`	Returns the current `x` coordinate of a component.
`int getY()`	Returns the current `y` coordinate of a component.
`Point getLocation()`	Returns the location of a component in the form of a point.

Table 2: Noteworthy Methods of the Component Class

Method	Description
`void setLocation(int x, int y)`	Sets a component at a new location according to the specified `x` and `y` coordinates.
`void setSize(int width, int height)`	Sets a component size according to the specified width and height.
`void setVisible(boolean b)`	Sets the visibility of a component, that is, if the parameter `b` is set to true, then the component is visible; otherwise, it is hidden.
`void setEnabled(boolean b)`	Enables a component if the parameter `b` is set to true; otherwise, the component is disabled.
`void setBounds(int x, int y, int w, int h)`	Sets the component in rectangular bounds. The `x` and `y` parameters of the setBounds() method set the `x` and `y` coordinates of the rectangle and the `w` and `h` parameters set the width and height.

Next, let's discuss the `Container` class.

Exploring the Container Class

A generic AWT container object is a component that contains other AWT components. The components added to a container are tracked in a list. The order of the list defines the front-to-back stacking order of the components within the container. When a component is added to a container and no index is specified in the add() method to append the component at a specified location, it is added to the end of the list. The `Container` class is an abstract class that cannot be instantiated and some of its methods must be implemented by its subclass. Table 3 lists noteworthy methods of the `Container` class:

Table 3: Noteworthy Methods of the Container Class

Method	Description
`Component add(Component comp)`	Appends a component, passed as an argument, to the end of a container.
`Component add(Component comp, int index)`	Appends a component, passed as an argument, at the specified position in a container.
`Component getComponent(int x, int y)`	Retrieves the component with the x and y coordinates.
`void remove(int index)`	Removes the component, specified by index, from a container.
`LayoutManager getLayout()`	Gets the layout manager for a container.
`void setFont(Font f)`	Sets the font for the text of a container.
`void setLayout(LayoutManager l)`	Sets the layout for a container.
`void update(Graphics g)`	Updates a container.

We discuss the Panel, Window and Frame classes in the following sections.

■ Exploring the Panel Class

The `Panel` class is a subclass of the `Container` class and a superclass of the `Applet` class. A panel serves as a container for attaching other components or panels. You can add a title bar, menu bar, or border to a panel window. When screen output of a Java application is directed to an applet inside a browser, the screen output appears on the Panel object provided in the application. Moreover, because of the absence of a title bar, menu bar, or border, you do not see these items when an applet runs in a browser. However, when you run an applet by using an applet viewer, the applet viewer displays the title bar and border. You can add other components, such as buttons and check boxes, to the Panel object by using the `add()` method. After adding these components, you can position and resize them by using the various methods of the `Component` class, such as `setLocation()`, `setSize()`, or `setBounds()`.

■ Exploring the Window Class

The Window class is a subclass of the `Container` class. An object of the Window class is a top-level window with no menu bar or borders. The WindowAdapter class or WindowListener interface responds to events, such as `WINDOW_OPENED` and `WINDOW_CLOSED`, of the Window class. You should create an object of the Frame class, instead of creating an instance of the Window class. The Frame class is the subclass of the Window class and the Frame class object provides a normal window with a title bar.

■ Exploring the Frame Class

As the Frame class is a subclass of the Window class and contains a title bar and borders; therefore, when you create an object of the Frame class within an applet a warning message is displayed during execution.

The default layout of a frame window is BorderLayout. A frame window generates events, such as `WINDOW_OPENED`, `WINDOW_CLOSED`, `WINDOW_ACTIVATED`, and `WINDOW_DEACTIVATED`.

Now that you are familiar with the various classes of the java.awt package, let's learn how to create a frame window and set its various properties to display, hide, and close the window.

8.2 Using AWT Frames

In this section, you learn how to create a frame window by using the AWT API. Depending on a user's requirement, a Windows application may consist of several components, such as windows, text boxes, check boxes, and buttons. All these AWT components can be added in a window.

Now, let's create a frame in a Windows application by using the java.awt package. To do so, we take the following broad level steps:

- Create a `Frame` window
- Hide and display the `Frame` window
- Set the title of the `Frame` window
- Close the `Frame` window

Now, let's perform these steps one by one.

Creating a Frame Window

A frame window is created by extending the functionality of the `Frame` class. This window has a title bar, menu bar, borders, and resizing options, such as minimize, maximize, and close. Before creating a frame window, however, let's list noteworthy constructors of the `Frame` class that help to create a frame window, as shown in Table 4:

Table 4: Noteworthy Constructors of the Frame Class

Constructor	Description
`Frame()`	Creates a frame window without a title.
`Frame(String str)`	Creates a frame window with a specified title.

Table 5 lists noteworthy methods of the Frame class:

Table 5: Noteworthy Methods of the Frame Class

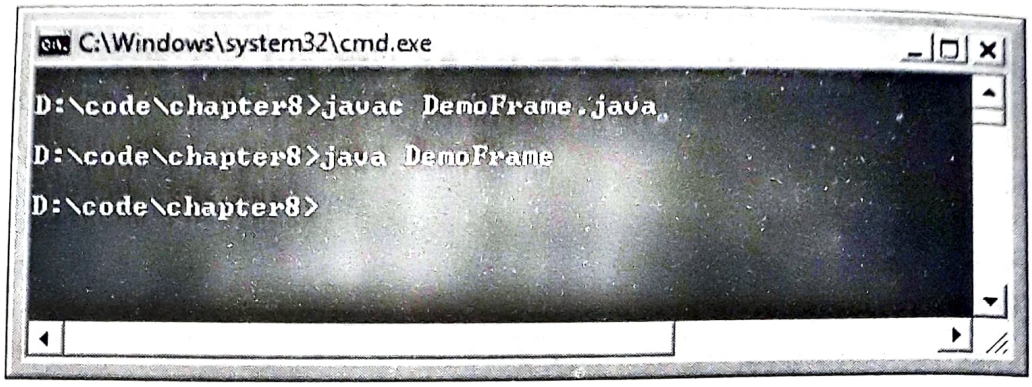
Method	Description
`String getTitle()`	Returns the title of a frame window.
`boolean isResizable()`	Returns true if a frame window is resizable; otherwise, the method returns false. By default, all frame windows are resizable.
`void setSize(int w, int h)`	Specifies the new size for a frame window according to the parameters passed on it.
`void remove (MenuComponent m)`	Removes the specified menu component from a frame window.
`void setMenuBar (MenuBar m)`	Sets the menu bar for a frame window.
`void setResizable(boolean resize)`	Sets the value to true or false as per the resize parameter. If the parameter is set to true, then the frame window can be resized by the user; otherwise, it cannot.
`void setTitle(String str)`	Sets the title for a frame window.
`void setVisible(boolean vb)`	Sets the value to true or false according to the vb parameter. If the vb parameter is set to true, then the frame window is visible; otherwise, it is not.

Listing 1 shows the creation of a frame window (you can find the `DemoFrame.java` file on the CD in the code\chapter8 folder):

▶ Listing 1: Creating a Frame Window

```
import java.awt.*;
class DemoFrame extends Frame
{
  public static void main (String args[]) {
      DemoFrame f = new DemoFrame();
      f.setSize (300,300);
  }
}
```

In Listing 1, the `java.awt` package is imported first to use its classes and methods and then the `DemoFrame` class is created, which extends the `Frame` class to use its functionality. In the `main()` method, we create an object of the `DemoFrame` class and also set the size of the frame window by using the `setSize()` method. Figure 1 shows the output of Listing 1:



▲ **Figure 1: Executing the DemoFrame Class**

Figure 1 shows that the program is compiled and runs successfully, which means that the frame window is created successfully. However, the question is, if the window has been successfully created, then why are we not able to view it? The answer is provided in the next section.

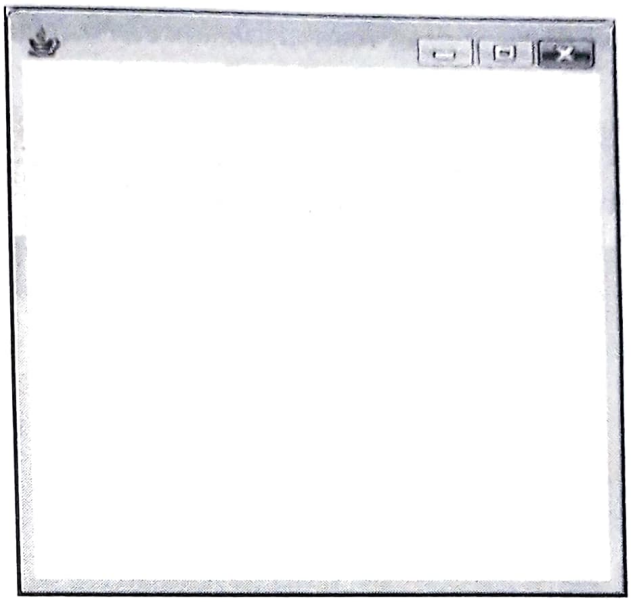
■ Hiding and Showing the Frame Window

In the previous section, we created a frame window, but are not able to view it. The reason for this is that we did not use the method required to display the window. We have to use the `setVisible()` method to display or hide a frame window. If the `setVisible()` method is set to true, then the frame window is visible; otherwise, it is hidden. Listing 2 shows use of the `setVisible()` method to display the frame window (you can find the `DemoFrame1.java` file on the CD in the code\chapter8 folder):

▶ **Listing 2: Program to Display a Frame Window**

```
import java.awt.*;

class DemoFrame1 extends Frame
{
  public static void main (String args [])
  {
      DemoFrame1 f = new DemoFrame1();
      f.setSize (300,300);
      f.setVisible (true); // making the window visible
  }
}
```

Listing 2 is similar to Listing 1, except for the class, which has been changed from `DemoFrame` to `DemoFrame1` and includes the `setVisible()` method to display the frame window. When you compile and execute the `DemoFrame1` class, the output appears, as shown in Figure 2:



▲ Figure 2: Displaying a Blank Frame Window

Figure 2 shows the frame window that we created in Listing 2. In this figure, we see the resizing options at the top-right corner of the window. Of the three resizing buttons, the `minimize` and `maximize` buttons are active, while the `close` button is inactive. We can activate the close button by using the `WindowListener` interface, discussed later in the chapter. You can, however, close the frame window now by pressing the `CTRL` and `C` keys together. This takes you back to the command prompt and closes the frame window.

We will use the `WindowListener` interface later in the chapter. Before that, however, let's learn how to set the title for a frame window.

■ Setting the Title of the Frame Window

The title of a frame window can be set by either specifying a string while instantiating the `Frame` class or by using the `setTitle()` method. Listing 3 shows how to set the title of a window through a constructor of the Frame class (you can find the `DemoFrame2.java` file on the CD in the code\chapter8 folder):

▶ **Listing 3: Setting a Window Title Through a Constructor**

```
import java.awt.*;
class DemoFrame2 extends Frame
{
  DemoFrame2(String str)
  {
      super(str);
  }
  public static void main(String args[])
  {
      DemoFrame2 f = new DemoFrame2("This is a Frame window");
      f.setSize (300,300);
      f.setVisible (true);
  }
}
```

In Listing 3, we create a class named `DemoFrame2`, in which the constructor of the Frame class is invoked. The constructor of the Frame class accepts the `str` parameter of the String type. Inside the `main()` method, we pass String as a parameter while instantiating the Frame object `f`. Figure 3 shows String passed as a parameter, as the title of the window, when the constructor of the `Frame` class is invoked in Listing 3:

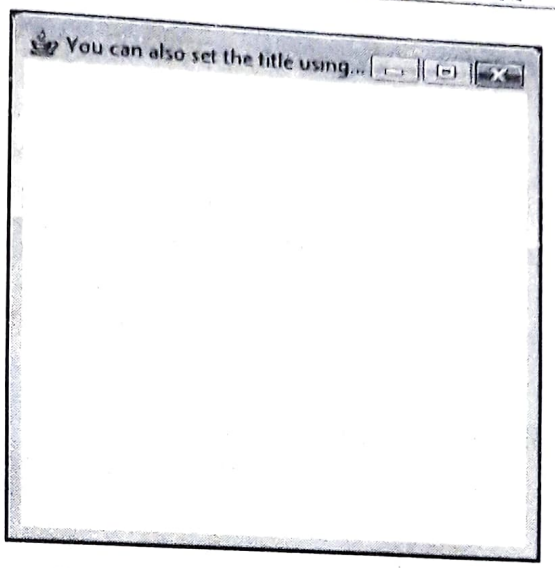
▲ **Figure 3: Displaying a Blank Window with a Title**

In this way, you can set the title for a frame window. Another way to set the title of a frame window is by using the `setTitle()` method. Listing 4 shows the use of the `setTitle()` method in setting the window title (you can find the `DemoFrame3.java` file on the CD in the code\chapter8 folder):

▶ **Listing 4: Using the `setTitle()` Method**

```
import java.awt.*;

class DemoFrame3 extends Frame
{
  public static void main(String args[])
  {
      DemoFrame3 f = new DemoFrame3();
      f.setSize (300,300);

      f.setVisible (true);
      f.setTitle ("You can also set the title using setTitle() ");
  }
}
```

Listing 4 is similar to Listing 2, except that in Listing 4, we use the `setTitle()` method to set the title of the frame window. Inside the `setTitle()` method, we pass a string that is displayed as the title of the frame window. Figure 4 shows the output of Listing 4:



▲ **Figure 4: Displaying the Title Set in a Blank Window**

In Figure 4, the string passed in the `setTitle()` method is displayed as the title of the frame window.

Next, let's learn how to close a frame window.

Closing a Frame Window

To close a frame window by clicking the close button on the top-right corner, we use the `windowClosing()` method of the `WindowListener` interface, as shown in Listing 5 (you can find the `CloseDemo.java` file on the CD in the code\chapter8 folder):

▶ **Listing 5: Closing a Frame Window**

```
import java.awt.*;
import java.awt.event.*;
class CloseDemo extends Frame {
 CloseDemo (String str) {
     super (str);
 }
 public static void main (String args []) {

     CloseDemo cd = new CloseDemo("Closing the Frame window");
     cd.setSize (300,300);
     cd.setVisible (true);
     cd.addWindowListener (new Demo());
 }
}
class Demo extends WindowAdapter
{
 public void windowClosing (WindowEvent e) {
     System.exit (0);
 }
}
```

In Listing 5, we import the `java.awt.event` package along with the java.awt package to use the classes and methods of both these packages. In this listing, the `CloseDemo` class extends the Frame class to use its functionality, and the `setSize()` method sets the size of

the frame window. The `setVisible()` method then sets the visibility of the frame window. We have set the visibility of the window to true. The reference of the Demo class is passed as an argument to the addWindowListener() method, which would help in closing the window. The `Demo class` extends `WindowAdapter`, which is an adapter class of the WindowListener interface. In addition, the `windowClosing()` method is used to close a frame window when a user clicks the cross (X) button on the window.

The Adapter class provides an implementation of all the methods contained in the listener interface.

Figure 5 shows the output of the Listing 5:

▲ **Figure 5: Closing a Blank Frame Window**

Now, clicking the cross (X) on the top-right corner of the frame window closes the frame window and brings us back to the command prompt screen.

In Listing 5, notice that we have created another class named `Demo`, whose reference is passed in the `addWindowListener()` method to close the frame window. This can also be done without creating the Demo class. In other words, you can close a frame window without creating another class. To do so, use the following code snippet:

```
cd.addWindowListener (new WindowAdapter()
{
public void windowClosing (WindowEvent e)
  {
      System.exit (0);
  }
});
```

In the preceding code snippet, `cd` is the object of the `CloseDemo` class, which is used to invoke the `addWindowListener()` method of the `WindowListener` interface. In the preceding code snippet, we have not created any class to extend the `WindowAdapter` class. However, to use the functionality of the `WindowAdapter` class, we create the object of this class by using the new keyword. We then use the `windowClosing()` method to close the window.

Since we have created the object of the WindowAdapter class, it means that internally a name is assigned to the WindowAdapter class inside the CloseDemo class. This type of an inner class is known as the Anonymous inner class.

Now, add the preceding snippet to Listing 5, as shown in Listing 6 (you can find the `CloseDemo1.java` file on the CD in the code\chapter8 folder):

▶ Listing 6: Implementing the `addWindowListener()` Method

```
import java.awt.*;
import java.awt.event.*;
class CloseDemo1 extends Frame
{
 CloseDemo1 (String str)
 {
     super (str);
 }

 public static void main (String args [])
 {
     CloseDemo1 cd = new CloseDemo1("Closing the Frame window");
     cd.setSize (300,300);
     cd.setVisible (true);
     cd.addWindowListener(new WindowAdapter()
     {
         public void windowClosing(WindowEvent e)
         {
             System.exit (0);
         }
     });
 }
}
```

In Listing 6, the name of the class has been changed from `CloseDemo`, in Listing 5, to `CloseDemo1`, and the rest is kept the same. The output of Listing 6 is the same as the output of Listing 5.

You now know how to create a frame window, set its title, and close the window. Next, let's discuss how events are handled in a frame window.

.3 Handling Events in Frames

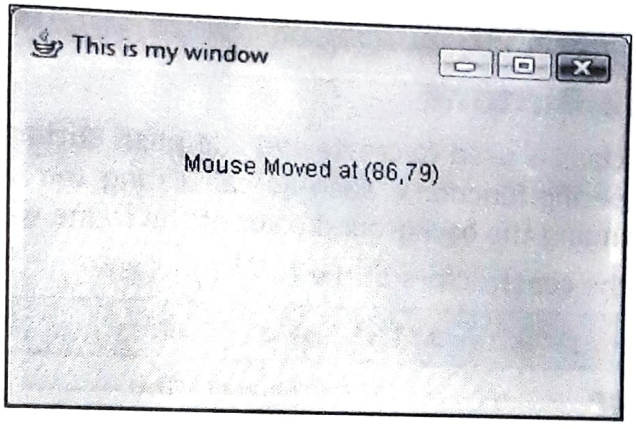
You have already studied the concept of handling events in *Chapter 7, Working with Applets*. However, here, we discuss the handling of events in a frame window. This implies that you can use and manage a frame window in the same way as done in the case of the main window of an applet. Each frame window has its own events, such as moving the mouse-pointer, dragging an object, or closing the frame window. Here, we create a simple program to demonstrate how mouse events are handled in a frame window. Listing 7 creates the `DemoFrame4` and `MyWindowAdapter` classes that help understand the concept of handling events in a frame window (you can find the `DemoFrame4.java` file on the CD in the code\chapter8 folder):

▶ **Listing 7: Handling Events in a Frame Window**

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

class DemoFrame4 extends Frame implements MouseMotionListener
{
 int x, y;
 String msg = " ";
 public DemoFrame4(String s)
 {
     super(s);
     setBackground(Color.pink);
     setForeground(Color.blue);
     setSize(200,200);
     setVisible (true);
     MyWindowAdapter adapter = new MyWindowAdapter(this);
     addWindowListener(adapter);
     addMouseMotionListener(this);
 }

 public void mouseDragged(MouseEvent me)
 {
     x = me.getX();
     y = me.getY();
     msg = "Mouse Dragged at (" + x + "," + y + ")";
     repaint();
 }

 public void mouseMoved(MouseEvent me)
 {
     x = me.getX();
     y = me.getY();
     msg = "Mouse Moved at (" + x + "," + y + ")";
     repaint();
 }
 public void paint(Graphics g)
 {
     g.drawString(msg, x, y);
 }
 public static void main(String args[])
 {
     DemoFrame4 d = new DemoFrame4("This is my window");
 }
}
class MyWindowAdapter extends WindowAdapter
{
 DemoFrame4 df;
 public MyWindowAdapter(DemoFrame4 df)
 {
     this.df = df;
 }
 public void windowClosing(WindowEvent we)
 {
     System.exit (0);
 }
}
```

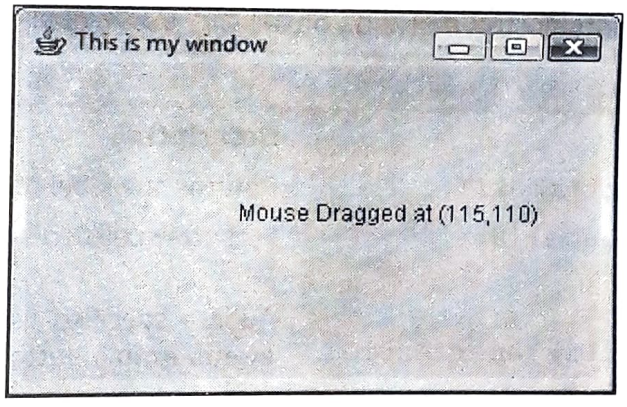
In Listing 7, the `DemoFrame4` class declares the `mouseDragged()` and `mouseMoved()` methods, which track mouse-pointer movements. Apart from this, the event of closing the frame window is also handled the `windowClosing()` method of the `WindowAdapter` interface. The `MyWindowAdapter` class is created to handle the events of the frame window, which is created in the `DemoFrame4` class.

Figure 6 shows the output of Listing 7:



▲ **Figure 6: Displaying Mouse-pointer Movements**

Dragging the mouse-pointer on the frame window displays the output as shown in Figure 7:



▲ **Figure 7: Dragging the Mouse-pointer**

Next, we discuss how components of the `Component` class are added in a frame window.

8.4 Using AWT Components

In Java, you can add components, such as labels, buttons, text boxes, and check boxes, in a frame window. The components allow a user to interact with an application. The most commonly used components are the push buttons. The appearance of a frame window is determined by the combination of the components contained in the frame window and by the layout manager, which is used to position the components. The implementation of layout managers is discussed later in the chapter. Here, we discuss how to add the following AWT components in a frame window:

- Push buttons
- Labels and text fields
- Check boxes and radio buttons

- Choice lists
- Lists
- Scroll bars
- Text area

These components are the subclasses of the Component class. Now, let's discuss implementation of each of these components.

Handling Push Buttons

The `Button` class is used to create and add push buttons to a frame window that can be used to perform specific functions, such as calculating the sum of two numbers on pressing the button or changing the background color of the frame window.

Table 6 lists the constructors of the `Button` class:

Table 6: Constructors of the Button Class

Constructor	Description
`Button()`	Constructs a button without a label.
`Button(String str)`	Constructs a button with the specific label.

Table 7 lists noteworthy methods of the `Button` class:

Table 7: Noteworthy Methods of the Button Class

Method	Description
`String getLabel()`	Returns the label of a button.
`void setLabel(String str)`	Sets the specified string as a button label.
`void addActionListener(ActionListener al)`	Adds a specified listener to a component to receive the events from a button.
`void removeActionListener(ActionListener al)`	Removes an action listener from a component so that it no longer receives the events from a button.
`String getActionCommand()`	Returns the command of the event caused by a button.
`String setActionCommand(string command)`	Sets the command name for the action of an event caused by a button.

Now, let's create a simple application, which shows how to add a push button in a frame window. In this application, we create the ButtonDemo class, which adds a push button and handles the events of the button, such as clicking of the button. Listing 8 shows how to add the push button in a frame window (you can find the `ButtonDemo.java` file on the CD in the code\chapter8 folder):

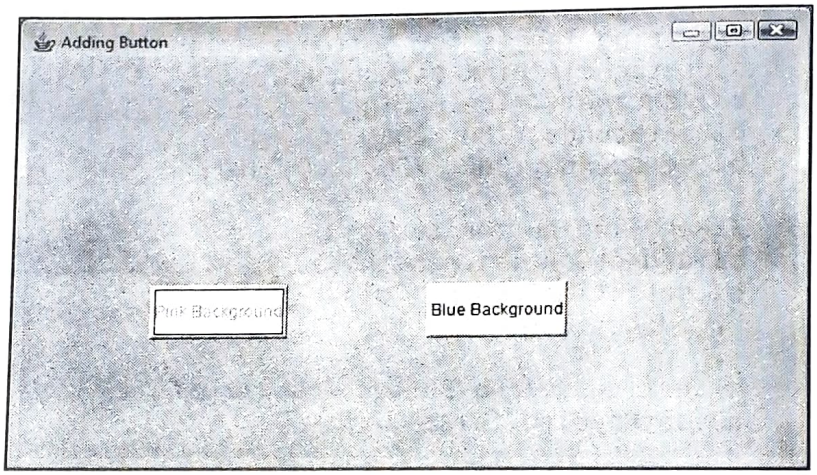
▶ **Listing 8: Adding Buttons to a Frame Window**

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

```
class ButtonDemo extends Frame implements ActionListener
{
 //Declaring buttons
 private Button b1;
 private Button b2;
 ButtonDemo()
 {
     this.setLayout (null);// setting layout to null
     //Instantiating Buttons
     b1 = new Button("Pink");
     b2 = new Button("Blue");

     //adding buttons to the frame
     add(b1);
     add(b2);

     //setting labels for button
     b1.setLabel("Pink Background");
     b2.setLabel("Blue Background");
     b1.setBounds (100, 200, 100, 40);
     b2.setBounds (300, 200, 100, 40);

     //Handling button events
     b1.addActionListener(this);
     b1.setActionCommand("pink");
     b2.addActionListener(this);

     //setting second button disenabled
     b2.setEnabled(false);

 }
 public void actionPerformed(ActionEvent e)
 {
     String action;
     action = e.getActionCommand();
     if(action.equals("pink"))
     {
         setBackground(Color.pink);
         b1.setEnabled(false);
         b2.setEnabled(true);
     }
     else if(action.equals("Blue Background"))
     {
         setBackground(Color.blue);
         b1.setEnabled(true);
         b2.setEnabled(false);
     }
 }
 public static void main (String args[])
 {

     ButtonDemo c = new ButtonDemo();
     c.setTitle ("Adding Button"); // setting frame title
     c.setSize (300,300);// setting frame size
     c.setVisible (true);
     //Handling closing of window
     c.addWindowListener (new WindowAdapter(){
```

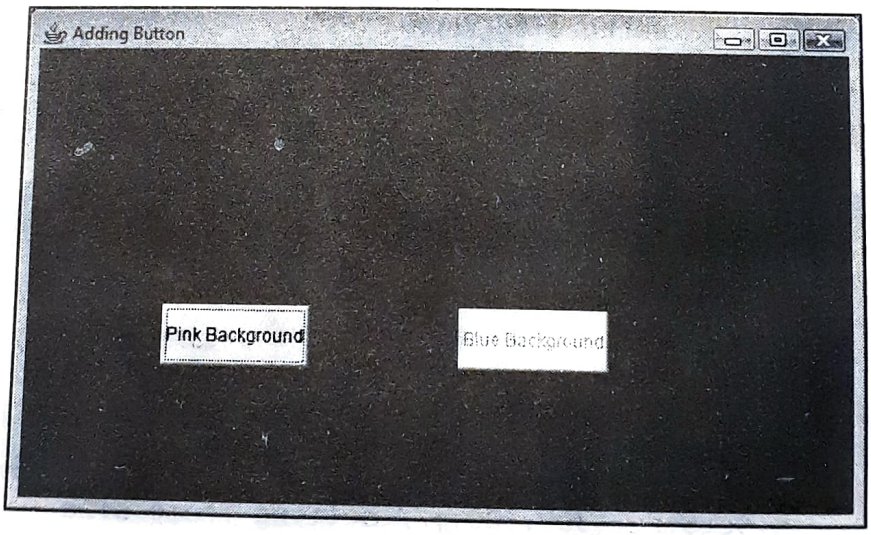
```
            public void windowClosing (WindowEvent e)
            {
                System.exit (0);
            }
        });
    }
}
```

In Listing 8, the java.awt.event package is imported along with the `java.awt` package to use their classes and methods. The `ButtonDemo` class extends the `Frame` class and implements the `ActionListener` interface. The `ButtonDemo` class creates two buttons, `b1` and `b2`. The action of clicking of the buttons is handled by invoking the actionPerformed() method. The background color of the frame window is changed, depending on which button is clicked. Figure 8 shows the output when the Pink Background button is clicked:



▲ **Figure 8: Displaying the Action Performed on Clicking a Button**

On clicking the Pink Background button, the background color of the frame window changes from white to pink. Moreover, the Pink Background button gets disabled and the Blue Background button gets enabled. Similarly, on clicking the Blue Background button, the background color changes from pink to blue, and the Pink Background button gets enabled, as shown in Figure 9:



▲ **Figure 9: Displaying the Change of the Background Color on Clicking a Button**

Now, let's learn how labels and text fields are added to a frame window.

Adding Labels and Text Fields

A label is a constant text, which is generally used to direct a user to do something. For example, in the form to create an e-mail address, you must have come across the text `Name`, in front of which you are provided blank space to enter your name. This `Name` text is known as the label and the blank space is known as a text field. The classes used to create a label and text field are `Label` and `TextField`, respectively. Table 8 lists the constructors of the `Label` class:

Table 8: Constructors of the Label Class

Constructor	Description
`Label()`	Constructs an empty label.
`Label(String str)`	Constructs a label with the specified text.
`Label(String str, int alignment)`	Constructs a label with a specified string, which is justified according to certain alignment constants, such as RIGHT, LEFT, or CENTER. Alignment constants are used with the class name, for example, Label.Right.

Table 9 lists noteworthy methods of the `Label` class:

Table 9: Noteworthy Methods of the Label Class

Method	Description
`String getText()`	Returns the text of a label.
`void setText(String str)`	Sets the specified string as the text of a label.
`void setAlignment(int Alignment)`	Sets a label to the specified alignment.
`int getAlignment()`	Returns the current alignment of a label.

Table 10 lists the constructors of the `TextField` class:

Table 10: Constructors of the TextField Class

Constructor	Description
`TextField()`	Constructs an empty text field.
`TextField(String str)`	Constructs a text field with the specified text.
`TextField(int col)`	Constructs a text field with the specified number of columns. The number of columns represents the maximum number of characters that can be entered in the text field.
`TextField(String str, int col)`	Constructs a text field with the specified text and number of columns.

Table 11 lists noteworthy methods of the `TextField` class:

Table 11: Noteworthy Methods of the TextField Class

Method	Description
`String getText()`	Retrieves text from a text field.
`void setText(String str)`	Sets the specified string in a text field.
`void addActionListener(ActionListener al)`	Adds the specified listener to a component to receive events.
`void removeActionListener(ActionListener al)`	Removes the action listener from a component so that it no longer receives events.
`void setEchoChar(Char c)`	Hides the text typed in a text field by a specified character. This method is used when you do not want to display the text to a user. For example, you must have noticed that whenever you login to your e-mail account, the password entered is displayed in a different format, such as *.
`void setColumns(int col)`	Sets the specified number of columns in a text field.
`int getColumns()`	Returns the number of the columns in a text field.

Now, let's create a simple frame window, which contains a label and a text field. Listing 9 shows how to add a label and text field in a frame window (you can find the `LabelTextDemo.java` file on the CD in the code\chapter8 folder):

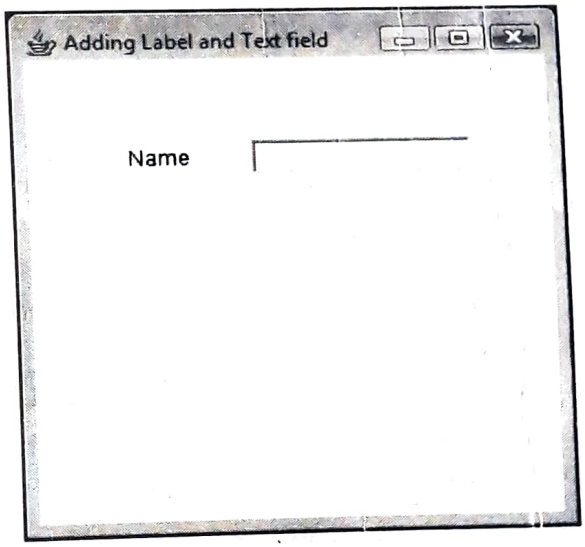
▶ **Listing 9: Adding a Label and Text Field**

```
import java.awt.*;
import java.awt.event.*;
class LabelTextDemo extends Frame
{

 Label lab;
 TextField t1;
 LabelTextDemo()
 {
     this.setLayout (null);

     lab = new Label ("Name");
     lab.setBounds (60, 60, 50, 50);

     t1 = new TextField (30);
     t1.setBounds (130, 75, 120, 20);

     this.add (lab);
     this.add (t1);

     addWindowListener (new WindowAdapter()
     {
         public void windowClosing (WindowEvent e)
         {
             System.exit (0);
         }
    });
```

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

    LabelTextDemo c = new LabelTextDemo ();
    c.setTitle ("Adding Label and Text field ");
    // Setting the frame title
    c.setSize (300,300);// setting the frame size
    c.setVisible (true);
 }
}
```

In Listing 9, a class named `LabelTextDemo` is created, which extends the `Frame` class to use its functionality. Inside this class, we first create the objects of the `Label` and `TextField` classes, similar to the way adopted to create the objects of the `Button` class in Listing 8, and then create the constructor of the `LabelTextDemo` class.

We allocate memory to the objects inside the constructor, and set the dimensions of the frame window. The objects are then added to the frame window by using the `add()` method. Figure 10 displays the output of Listing 9:



▲ **Figure 10: Displaying a Label and a Text Field**

Figure 10 shows a label and text box in a frame window.

■ Adding Check Boxes and Radio Buttons

A check box is a square-shaped box, which displays an option to a user. When the user selects the check box, a visual indication of some kind, such as a check mark, appears to specify that the check box is selected. If the check box is selected again, then the check box is cleared.

A radio button, on other hand, is a round-shaped button, and only one radio button can be selected from the given group of radio buttons as compared to check boxes, which you can select more than one at a time. Radio buttons are created by adding check boxes to a check box group. The class used to create radio buttons is known as the `CheckboxGroup` class, while the class used to create a check box is the `Checkbox` class. Table 12 lists noteworthy constructors of the `Checkbox` class:

Table 12: Noteworthy Constructors of the Checkbox Class

Constructor	Description
`Checkbox()`	Constructs a check box without a label.
`Checkbox(String str)`	Constructs a check box with a specified label.
`Checkbox(String str, boolean state)`	Constructs a check box with a specified label and also sets its state, that is, if the state is set to true, then the check box is selected; otherwise, it is not.

Table 13 lists noteworthy methods of the `Checkbox` Class:

Table 13: Noteworthy Methods of the Checkbox Class

Method	Description
`boolean getState()`	Returns true if a check box is in the selected state; otherwise, it returns false.
`void setState(boolean state)`	Sets the state of a check box as either true or false.
`void addItemListener (ActionListener al)`	Adds the specified item listener to a check box component to receive item events.
`void removeItemListener(ActionListener al)`	Removes an item listener from a check box component so that it no longer receives events.
`String getLabel()`	Returns the label of a check box.
`Object [] getSelectedObjects()`	Returns an array of length 1, which contains the check box label, if the check box is selected, or returns null if the check box is not selected.

The `CheckboxGroup` class has only one constructor, the `CheckboxGroup()`, which takes no parameter. Table 14 lists noteworthy methods of the `CheckboxGroup` class:

Table 14: Noteworthy Methods of the CheckboxGroup Class

Method	Description
Checkbox getSelectedCheckbox()	Returns the current selected check box.
void setSelectedCheckbox(Checkbox box)	Sets the currently selected check box in a check box group.

Now, let's create a simple frame window, which contains check boxes and radio buttons. Listing 10 shows how to add check boxes and radio buttons to a frame window (you can find the `CheckRadioDemo.java` file on the CD in the code\chapter8 folder):

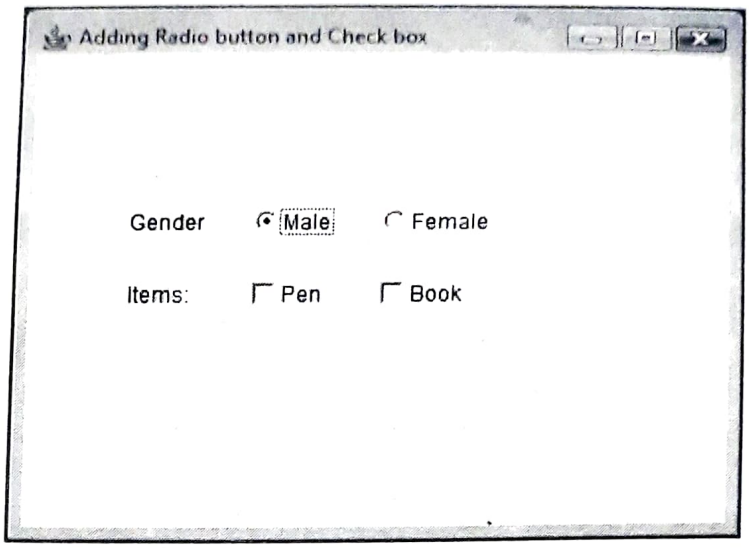
▶ ***Listing 10: Adding Check Boxes and Radio Buttons***

```
import java.awt.*;
import java.awt.event.*;
class CheckRadioDemo extends Frame
{
 Label lab1, lab2;
```

```
CheckboxGroup cb;
Checkbox c1, c2, c3, c4;

CheckRadioDemo ()
{
    this.setLayout (null);

    lab1 = new Label ("Gender");
    lab1.setBounds (60, 100, 50, 50);
    lab2 = new Label ("Items:");
    lab2.setBounds (60, 140, 50, 50);

    cb = new CheckboxGroup();
    c1 = new Checkbox ("Male", cb, true); /* passing cb object to
    checkbox class to make it radiobutton */
    c1.setBounds (130, 115, 60, 20);
    c2 = new Checkbox ("Female", cb,false);/* passing cb object to
    checkbox class to make it radiobutton */
    c2.setBounds (200, 115, 60, 20);
    c3 = new Checkbox ("Pen");
    c3.setBounds (130, 155, 50, 20);
    c4 = new Checkbox ("Book");
    c4.setBounds (200, 155, 50, 20);
    this.add (lab1);
    this.add (lab2);

    this.add (c1);
    this.add (c2);
    this.add (c3);
    this.add (c4);
    addWindowListener (new WindowAdapter()
    {
        public void windowClosing (WindowEvent e)
        {
            System.exit (0);
        }
    });
}
public static void main (String args [])
{
    CheckRadioDemo c = new CheckRadioDemo ();
    c.setTitle ("Adding Radio button and Check box");
    c.setSize (300,300);
    c.setVisible (true);
}
}
```

In Listing 10, we create a class named `CheckRadioDemo`, which extends the `Frame` class to use the methods of this class. Inside the `CheckRadioDemo` class, we first create objects of the `Label`, `CheckboxGroup`, and `Checkbox` classes and then create the constructor of the `CheckRadioDemo` class.

Inside the constructor, memory is allocated to the objects and the `CheckboxGroup` object is passed to the `c1` and `c2` objects to create radio buttons. The objects are then added to the frame window by using the `add()` method. Figure 11 shows the output of Listing 10:



▲ **Figure 11: Displaying Radio Buttons and Check Boxes in a Frame Window**

Figure 11 shows a group of radio button and check boxes in a frame window.

Handling a Choice List

The `Choice` class helps you to create a drop-down list. This class has only one constructor, `Choice()`, which is used to create an empty choice list. Table 15 lists noteworthy methods of the `Choice` class:

Table 15: Noteworthy Methods of the Choice Class

Method	Description
`void add(String item)`	Adds an item to a choice component.
`void addItemListener(ActionListener al)`	Adds the specified item listener to receive events from a choice component.
`void removeItemListener(ActionListener al)`	Removes the item listener from a choice component so that it no longer receives item events.
`int getItemCount()`	Returns the number of items in a choice component.
`String getItem(int index)`	Returns the string at the given index in a choice component.
`int getSelectedIndex()`	Returns the index of the currently selected item and returns -1 if no item is selected.
`void insert(String item, int index)`	Adds the specified item at the specified index in a choice component.
`void remove(int index)`	Deletes an item from a choice component at the specified index.
`void remove(String item)`	Deletes the first occurrence of the specified item from a choice component.

Now, let's create a simple program to understand the use of the `Choice` class. Listing 11 shows how to add a drop-down list by using the `Choice` class (you can find the `ChoiceDemo.java` file on the CD in the code\chapter8 folder):

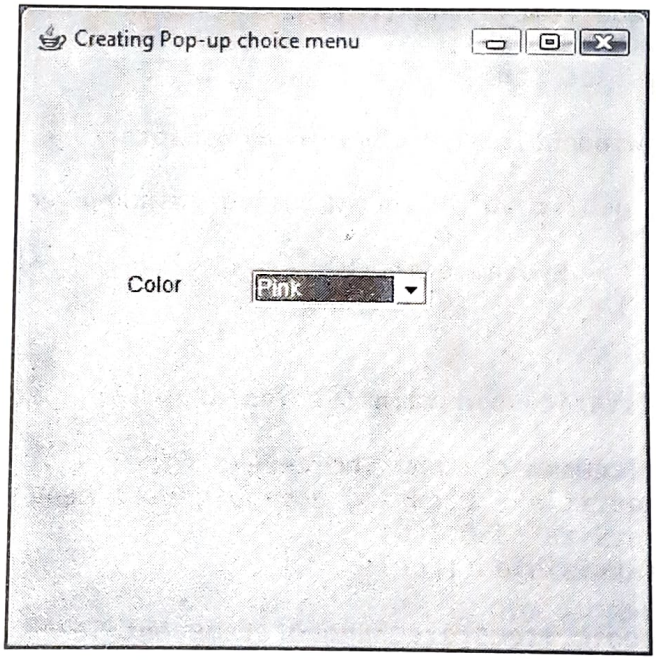
▶ **Listing 11: Creating a Drop-down List**

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

class ChoiceDemo extends Frame implements ItemListener
{
  Label lab;
  Choice ch;
  ChoiceDemo()
  {
      this.setLayout (null);

      lab = new Label ("Color");
      lab.setBounds (60, 130, 50, 50);
      ch = new Choice ();
      ch.add ("Red");
      ch.add ("Blue");
      ch.add ("Pink");
      ch.add ("White");
      ch.setBounds (130,145, 100, 20);
      ch.addItemListener(this);
      this.add (lab);
      this.add (ch);

      addWindowListener (new WindowAdapter()
      {
          public void windowClosing (WindowEvent e)
          {
              System.exit (0);
          }
      });
  }
  public static void main (String args [])
  {
      ChoiceDemo c = new ChoiceDemo ();
      c.setTitle ("Creating Pop-up choice menu");
      c.setSize (350,350);
      c.setVisible (true);
  }
  public void itemStateChanged(ItemEvent ie)
  {
      int s;
      s = ch.getSelectedIndex();
      if(s == 0)
      {
          setBackground(Color.red);
          lab.setBackground(Color.red);
      }
      else if(s == 1)
      {
          setBackground(Color.blue);
          lab.setBackground(Color.blue);
```

```
        }
        else if(s == 2)
        {
            setBackground(Color.pink);
            lab.setBackground(Color.pink);
        }
        else if(s == 3)
        {
            setBackground(Color.white);
            lab.setBackground(Color.white);
        }
    }
}
```

Listing 11 creates a class named `ChoiceDemo`, which extends the `Frame` class and implements the `ItemListener` interface to use its functionality. Inside the `ChoiceDemo` class, we first create the objects of the `Label` and `Choice` classes and then create the constructor of the `ChoiceDemo` class.

Inside the constructor, we first allocate memory to the objects, and then add those objects to the frame window by using the `add()` method. The `itemStateChanged()` method keeps track of the item selected in the drop-down list, and accordingly changes the background color of the frame window. Figure 12 shows the output of Listing 11, when a user select the pink color from the drop-down list:



▲ **Figure 12: Displaying the Choice list on a Frame Window**

Figure 12 shows that the background color of the frame window changes to pink when the Pink item is selected from the drop-down list.

■ Handling Lists

The `List` component displays a scrolling list of text items. You can set the type of selection for a list so that a user can select either one or multiple items. Table 16 lists noteworthy constructors of the `List` class:

Table 16: Noteworthy Constructors of the List Class

Constructor	Description
`List()`	Constructs a new scrolling list.
`List(int rows)`	Constructs a new scrolling list with the specified number of visible lines.
`List(int rows, boolean multipleMode)`	Constructs a new scrolling list to display the specified number of rows.

Table 17 lists noteworthy methods of the `List` class:

Table 17: Noteworthy Methods of the List Class

Method	Description
`void add(String item)`	Adds the specified item to the end of a scrolling list.
`void add(String item, int index)`	Adds the specified item at the specified index.
`void addActionListener(ActionListener l)`	Adds the specified action listener to handle the action events of a list.
`void addItemListener(ItemListener l)`	Adds the specified item listener to handle the item events of a list.
`String getItem(int index)`	Returns the item associated with the specified index value passed as an argument.
`String[] getItems()`	Returns all items in a list.
`void select(int index)`	Selects the item associated with the specified index value passed as an argument.

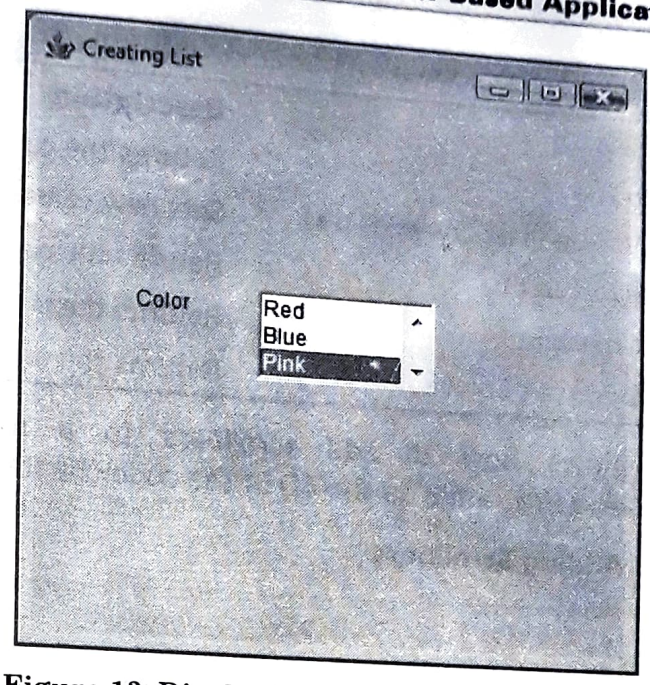
Listing 12 shows how to create a list and add it to a frame window (you can find the `ListDemo.java` file on the CD in the code\chapter8 folder):

▶ **Listing 12: Adding a List to a Frame Window**

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

class ListDemo extends Frame implements ItemListener
{
  List l;
  Label lab;
  ListDemo()
  {
      this.setLayout (null);
      lab = new Label ("Color");
      lab.setBounds (60, 130, 50, 50);
      l = new List();
      l.add("Red");
      l.add("Blue");
      l.add("Pink");
      l.add("White");
      l.addItemListener(this);
```

```
            l.setBounds(130,145, 100, 50);
            add(lab);
            add(l);

            addWindowListener(new WindowAdapter()
            {
                public void windowClosing (WindowEvent e)
                {
                    System.exit (0);
                }
            });
        }

        public void itemStateChanged(ItemEvent ie)
        {
            int s;
            s = l.getSelectedIndex();
            if(s == 0)
            {
                setBackground(Color.red);
                lab.setBackground(Color.red);
            }

            else if(s == 1)
            {
                setBackground(Color.blue);
                lab.setBackground(Color.blue);
            }

            else if(s == 2)
            {
                setBackground(Color.pink);
                lab.setBackground(Color.pink);
            }

            else if(s == 3)
            {
                setBackground(Color.white);
                lab.setBackground(Color.white);
            }
        }

        public static void main(String args[])
        {
            ListDemo d = new ListDemo();
            d.setTitle("Creating List");
            d.setSize (350,350);
            d.setVisible(true);
        }
    }
```

In Listing 12, the `ListDemo` class is declared, which creates a frame window containing a label and list. When the user selects an option from the list, the itemStateChanged() method is invoked. The background color of the frame window changes, depending on the selected index value. Figure 13 shows the output of Listing 12, when the pink color is selected:



▲ Figure 13: Displaying a List on a Frame Window

In Figure 13, the pink color is selected and the background of the frame window changes accordingly.

■ Adding a Scrollbar

A scrollbar consists of arrows (buttons at each end of the scrollbar), a thumb or bubble (the scrollable box you slide), and a track (the part of the scrollbar you slide the thumb in). The class used to create a scrollbar is known as the `Scrollbar` class. Table 18 lists the constructors of the `Scrollbar` class:

Table 18: Constructors of the Scrollbar Class

Constructor	Description
`Scrollbar()`	Constructs a new vertical scrollbar.
`Scrollbar(int orientation)`	Constructs a new scrollbar with the given orientation. Orientation here refers to the `VERTICAL` or `HORIZONTAL` position of the scrollbar. For example, `Scrollbar.VERTICAL` or `Scrollbar.HORIZONTAL`.
`Scrollbar(int orientation, int value, int length, int min, int max)`	Constructs a scrollbar with the given orientation, page length, starting value, and the minimum and maximum scrollbar values. The starting value represents the location of a scrollbar thumb. By default, the scrollbar thumb appears at the left side of a horizontal scrollbar and at the top in a vertical scroll bar. The minimum and maximum scrollbar values are used to set the thumb size of a scrollbar within a minimum and maximum range, respectively.

Table 19 lists noteworthy methods of the `Scrollbar` class:

Table 19: Noteworthy Methods of the Scrollbar Class

Method	Description
`int getValue()`	Returns the current value of a scrollbar.
`void setValue(int newvalue)`	Sets new values for a scrollbar.
`int getMaximum()`	Returns the `maximum` value of a scrollbar.
`int getMinimum()`	Returns the `minimum` value of a scrollbar.
`int getOrientation()`	Returns the orientation of a scrollbar.

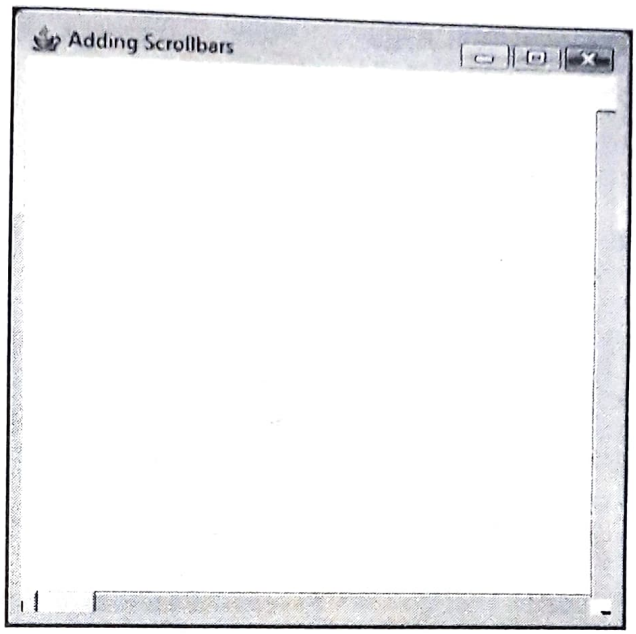
Listing 13 shows how to add scrollbars to a frame window (you can find the `ScrollbarDemo.java` file on the CD in the code\chapter8 folder):

▶ **Listing 13: Adding Scrollbars**

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

class ScrollbarDemo extends Frame {
 Scrollbar s1, s2;
 ScrollbarDemo()
 {
     this.setLayout (null);
     s1 = new Scrollbar (Scrollbar.VERTICAL);
     s1.setBounds(330,0,20,350);
     s2 = new Scrollbar (Scrollbar.HORIZONTAL);
     s2.setBounds(0,330,350,20);
     this.add (s1);
     this.add (s2);
     addWindowListener (new WindowAdapter()
     {
         public void windowClosing (WindowEvent e)
         {
             System.exit (0);
         }
     });
 }

 public static void main (String args [])
 {
     ScrollbarDemo c = new ScrollbarDemo ();
     c.setTitle ("Adding Scrollbars");
     c.setSize (350,350);
     c.setVisible (true);
 }
}
```

In Listing 13, we first create the objects of the `Scrollbar class` and then create the constructor of the `ScrollbarDemo` class.

Inside the constructor, memory is allocated to the objects of the `Scrollbar` class and the orientation is set for the scrollbars. The objects of the `Scrollbar` class are then added to the frame window by using the `add()` method. Figure 14 shows output of Listing 13:



▲ Figure 14: Displaying Scrollbars Added in a Frame Window

Figure 14 shows both the vertical and horizontal scrollbars in the frame window.

Adding a Text Area

A `text area` is similar to a text field, the only difference between the two being that a text area is used to get a multi-line text box, for example, adding personal details on an online form, such as one's contact details. Table 20 lists noteworthy constructors of the `TextArea` class:

Table 20: Noteworthy Constructors of the TextArea Class

Constructor	Description
`TextArea()`	Constructs a new text area.
`TextArea(int row, int col)`	Constructs a new text area with a specific number of rows and columns.
`TextArea(String str)`	Constructs a new text area with a specific string.

Table 21 lists noteworthy methods of the `TextArea` class:

Table 21: Noteworthy Methods of the TextArea Class

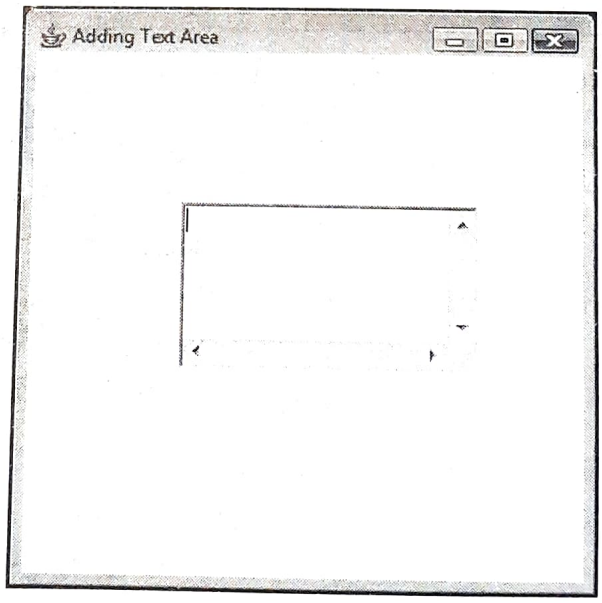
Method	Description
`String getText()`	Returns text from a text area.
`void setText(String text)`	Sets the specified text in a text area.
`void append(String text)`	Adds specific text to the current text in a text area.
`void insert(String text, int position)`	Adds the specified text at the specified position in a text area.

Listing 14 shows how to add a text area in a frame window (you can find the `TextAreaDemo.java` file on the CD in the code\chapter8 folder):

▶ **Listing 14: Adding a Text Area**

```
import java.awt.*;
import java.awt.event.*;
class TextAreaDemo extends Frame
{
 TextArea t;
 TextAreaDemo()
 {
     this.setLayout (null);
     t = new TextArea (2, 20);
     t.setBounds(100, 120, 180, 100);
     this.add (t);
     addWindowListener (new WindowAdapter()
     {
         public void windowClosing (WindowEvent e)
         {
             System.exit (0);
         }
     });
 }
 public static void main (String args [])
 {
     TextAreaDemo t = new TextAreaDemo ();
     t.setTitle ("Adding Text Area");
     t.setSize (350,350);


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

Listing 14 creates a class named `TextAreaDemo`, which extends the `Frame` class to use its functionality. Inside the `TextAreaDemo` class, we first create the object `t` of the TextArea class and then create the constructor of the TextAreaDemo class, in which memory is allocated to the `t` object and the bounds of the object are set. Figure 15 shows the output of Listing 14:



▲ **Figure 15: Displaying Text Area in a Frame Window**

Figure 15 shows a text area added in a frame window, in which a user can enter multi-line string values.

Next, let's discuss the implementation of layout managers.

8.5 Using Layout Managers

Java provides various layout managers, which are used to arrange components, such as buttons and radio buttons, in a frame window. A layout manager implements the `LayoutManager` interface of the `java.awt` package to arrange these components in the frame window. Before describing the various layout managers, however, let's first look at the `setLayout()` method, which is used to set a particular layout for a frame window. To take an example, suppose that you want to set the Border layout for the components in a frame window. To do so, use the following code snippet:

```
setLayout(new BorderLayout());
```

In the preceding code snippet, the Border layout, which is a type of layout manager, is set for a frame window. In the following sections, we will look at the following layout managers in detail:

- The FlowLayout Manager
- The BorderLayout Manager
- The GridLayout Manager
- The GridBagLayout Manager
- The CardLayout Manager

The FlowLayout Manager

The `FlowLayout` manager is the default layout for applets and panels. It arranges the components in a row. When one row is filled with components, they are automatically arranged in the next row, and so on. The Java class used to implement the `FlowLayout` in a frame window or an applet is known as the `FlowLayout` class. Table 22 lists the constructors of the `FlowLayout` class:

Table 22: Constructors of the FlowLayout Class

Constructor	Description
`FlowLayout()`	Creates a new flow layout with a centered alignment that has 5 pixel horizontal and vertical gaps between the components.
`FlowLayout(int align)`	Creates a new flow layout with a specified alignment that has 5 pixel horizontal and vertical gaps between the components. The alignment can be RIGHT, LEFT, or CENTER.
`FlowLayout(int align, int hgap, int vgap)`	Creates a new flow layout with the specified horizontal or vertical gap between the components.

Table 23 lists noteworthy methods of the `FlowLayout` class:

Table 23: Noteworthy Methods of the FlowLayout Class

Method	Description
`void setHgap(int hgap)`	Sets the horizontal gap between the components in a frame window.
`void setVgap(int vgap)`	Sets the vertical gap between the components in a frame window.
`int getAlignment()`	Returns the alignment of the flow layout between the components in a frame window.

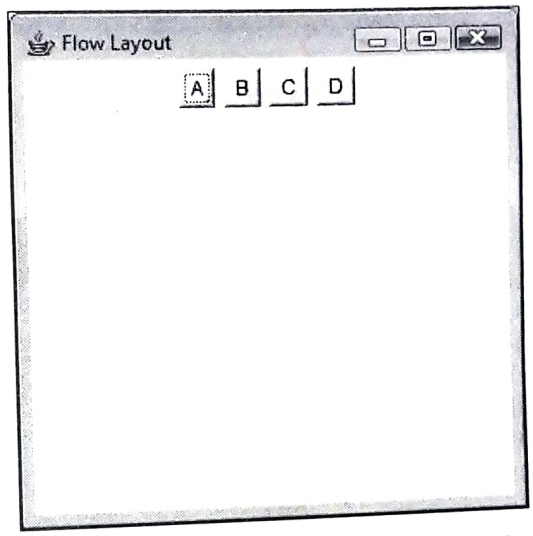
Listing 15 shows how to implement flow layout in a frame window (you can find the `FlowLayoutDemo.java` file on the CD in the code\chapter8 folder):

▶ **Listing 15: Using the FlowLayout Class**

```
import java.awt.*;
import java.awt.event.*;
class FlowLayoutDemo extends Frame
{

  Button b1, b2, b3, b4;
  FlowLayoutDemo()
  {
      this.setLayout (new FlowLayout (FlowLayout.CENTER));
      b1 = new Button("A");
      b2 = new Button("B");
      b3 = new Button("C");
      b4 = new Button("D");
      this.add(b1);
      this.add(b2);
      this.add(b3);
      this.add(b4);
      addWindowListener (new WindowAdapter()
      {
          public void windowClosing (WindowEvent e)
          {
              System.exit (0);
          }
      });
  }
  public static void main (String args [])
  {

      FlowLayoutDemo f = new FlowLayoutDemo ();
      f.setTitle ("Flow Layout");// setting frame title
      f.setSize (300,300);// setting frame size
      f.setVisible (true);

  }
}
```

In Listing 15, we create a class named `FlowLayoutDemo`, which extends the `Frame` class. Inside this class, we first create the objects of the Button class and then create the constructor of this class.

Inside the constructor, we first set the flow layout to center alignment for the components, followed by allocating memory to the `b1`, `b2`, `b3`, and `b4` objects of the Button class. The objects are then added to the frame window by using the `add()` method. In addition, the `f` object of the `FlowLayoutDemo` class is created inside the `main()` method. Figure 16 shows the output of Listing 15:



▲ **Figure 16: Displaying the Flow Layout of a Frame Window**

In Figure 16, note that the buttons created are arranged according to the flow layout specified in Listing 15.

The BorderLayout Manager

The BorderLayout manager is the default layout of a frame window. This layout allows you to arrange components on the four borders of the frame window. The border is identified with the name of the direction at which it located, that is, North, South, East, or West. The Java class that is used to implement the `BorderLayout` layout in a frame window or an applet is known as the BorderLayout class. Table 24 lists the constructors of the BorderLayout class:

Table 24: Constructors of the BorderLayout Class

Constructor	Description
`BorderLayout()`	Creates a new border layout of a frame window.
`BorderLayout(int hgap, int vgap)`	Creates a new border layout with specified horizontal and vertical gaps between the components.

Table 25 lists noteworthy methods of the BorderLayout class:

Table 25: Noteworthy Methods of BorderLayout Class

Method	Description
`void setHgap(int hgap)`	Sets the horizontal gap between components in pixels.
`void setVgap(int vgap)`	Sets the vertical gap between components in pixels.
`int getHgap()`	Returns the horizontal gap between components in pixels.
`int getVgap()`	Returns the vertical gap between components in pixels.

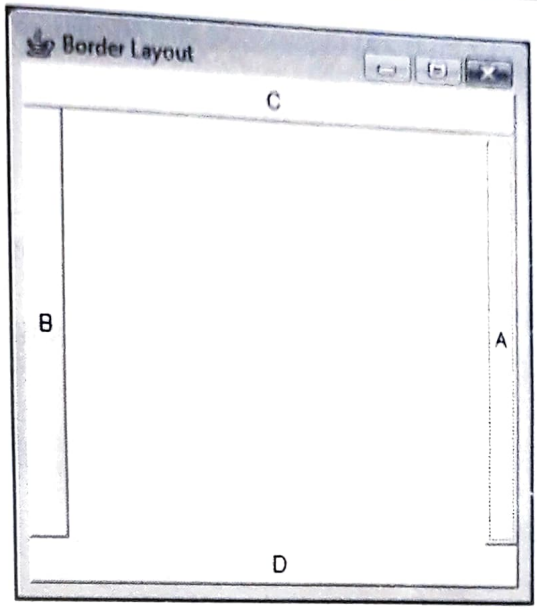
Listing 16 shows how to implement border layout in a form window (you can find the BorderLayoutDemo.java file on the CD in the code\chapter8 folder):

▶ **Listing 16: Using the BorderLayout Class**

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

class BorderLayoutDemo extends Frame
{
 Button b1, b2, b3, b4;
 BorderLayoutDemo ()
 {
     this.setLayout (new BorderLayout()); // Setting the layout
     b1 = new Button ("A");
     b2 = new Button ("B");
     b3 = new Button ("C");
     b4 = new Button ("D");
     this.add ("East",b1);
     this.add ("West", b2);
     this.add ("North", b3);
     this.add ("South", b4);

     addWindowListener (new WindowAdapter()
     {
         public void windowClosing (WindowEvent e)
         {
             System.exit (0);
         }
     });
 }

 public static void main (String args [])
 {

     BorderLayoutDemo b = new BorderLayoutDemo ();
     b.setTitle ("Border Layout");// setting frame title
     b.setSize (300,300);// setting frame size
     b.setVisible (true);

 }
}
```

In Listing 16, we create a class named BorderLayoutDemo, which extends the Frame class. The objects of the Button class are created in this class, followed by the creation of the constructor of the class.

Inside the constructor, we first set the layout of a frame window as BorderLayout, and then allocate memory to the objects of the Button class. Moreover, these objects are added to the frame window in the specified directions by using the add() method. The object of the BorderLayoutDemo class is created inside the main() method. Figure 17 shows the output of the Listing 16:



▲ Figure 17: Displaying BorderLayout on a Frame Window

In Figure 17, the buttons created in Listing 16 are arranged on the four sides of the frame window, according to the layout specified in Listing 16.

TEST YOUR KNOWLEDGE

Q1. Write a program to arrange the components in the border layout in a frame window.

Ans.

```
import java.awt.*;
import java.awt.event.*;
class TestYourKnowledge1 extends Frame
{
  Button b1, b2, b3, b4;
  TestYourKnowledge1()
  {
      this.setLayout (new BorderLayout()); // Setting the layout
      b1 = new Button ("Border 1");
      b2 = new Button ("Border 2");
      b3 = new Button ("Border 3");
      b4 = new Button ("Border 4");
      this.add ("East",b1);
      this.add ("West", b2);
      this.add ("North", b3);
      this.add ("South", b4);

      addWindowListener (new WindowAdapter()
      {
          public void windowClosing (WindowEvent e)
          {
              System.exit (0);
          }
      });
  }

  public static void main (String args [])
  {

      TestYourKnowledge1 b = new TestYourKnowledge1 ();
      b.setTitle ("Border Layout");// setting frame title
      b.setSize (300,300);// setting frame size
```

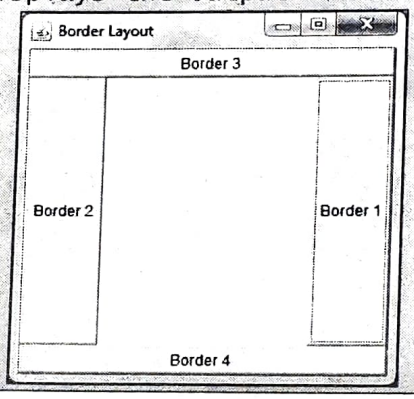
```
        b.setVisible (true);
    }
}
```

Execution:

```
D:\code\chapter8>javac TestYourKnowledge1.java
D:\code\chapter8>java TestYourKnowledge1
```

Output:

The following figure displays the output of the preceding program:



The GridLayout Manager

The GridLayout manager divides a container into a two-dimensional grid that contains several rows and columns. In a GridLayout manager, each component is given the same size and dimensions. The Java class used to implement the GridLayout in a frame window or in applets is known as the `GridLayout` class. Table 26 lists the constructors of the `GridLayout` class:

Table 26: Constructors of the GridLayout Class

Constructor	Description
`GridLayout()`	Creates a grid layout that has one column per component in one row.
`GridLayout(int rows, int cols)`	Creates a grid layout with the specified number of rows and columns.
`GridLayout(int rows, int cols, int hgap, int vgap)`	Creates a grid layout with the specified number of rows and columns, along with the specified horizontal and vertical gaps between the components.

Table 27 lists noteworthy methods of the `GridLayout` class:

Table 27: Noteworthy Methods of the GridLayout Class

Method	Description
`void setHgap(int hgap)`	Sets the horizontal gap between components.
`void setVgap(int vgap)`	Sets the vertical gap between components.

Table 27: Noteworthy Methods of the GridLayout Class

Method	Description
`void setColumns(int cols)`	Sets the specified number of columns in the layout.
`void setRows(int cols)`	Sets the specified number of rows in the layout.

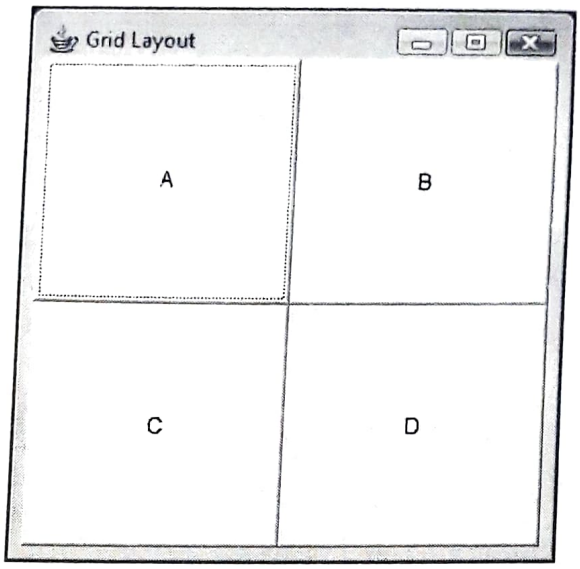
Listing 17 shows the usage of the `GridLayout` class (you can find the `GridLayoutDemo.java` file on the CD in the code\chapter8 folder):

▶ **Listing 17: Using the GridLayout Class**

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

class GridLayoutDemo extends Frame
{
  Button b1, b2, b3, b4;
  GridLayoutDemo()
  {
      this.setLayout (new GridLayout(2,2)); // Seting layout
      b1 = new Button ("A");
      b2 = new Button ("B");
      b3 = new Button ("C");
      b4 = new Button ("D");

      this.add (b1);
      this.add (b2);
      this.add (b3);
      this.add (b4);
      addWindowListener (new WindowAdapter()
      {
          public void windowClosing (WindowEvent e)
          {
              System.exit (0);
          }
      });
  }

  public static void main (String args [])
  {
      GridLayoutDemo g = new GridLayoutDemo();
      g.setTitle("Grid Layout");// setting frame title
      g.setSize(300,300);// setting frame size
      g.setVisible(true);
  }

}
```

In Listing 17, we create a class named `GridLayoutDemo`, which extends the Frame class. Inside the `GridLayoutDemo` class, we first create objects of the `Button` class and then create the constructor of the class.

Inside the constructor, we first set the layout to `GridLayout` with two rows and two columns. This is followed by allocating memory to the objects of the `Button` class. The objects are then added to the frame window by using the `add()` method. In addition, object g of the `GridLayoutDemo` class is created inside the `main()` method. Figure 18 shows the output of Listing 17:



▲ **Figure 18: Displaying the GridLayout on a Frame Window**

In Figure 18, note that the buttons created in the Listing 17 are arranged in a grid layout consisting of 2 rows and 2 columns.

■ The GridBagLayout Manager

In the previous section, you studied about the `GridLayout` manager, which allows you arrange the components of a frame window in rows and columns. The `GridBagLayout` class also allows you to arrange the components in rows and columns, but it is more flexible than the `GridLayout` layout as the components can span more than one row or column. The size and dimensions of these components can be set in a frame window. However, some constraints must be applied on the components regarding their size and position. Before we study the application of constraints on the components, let's have a look at the constructors and methods of the `GridBagLayout` class. This class has the `GridBagLayout()` constructor, which creates the GridBagLayout layout. Table 28 lists noteworthy methods of the `GridBagLayout` class:

Table 28: Noteworthy Methods of the GridBagLayout Class

Method	Description
`void removeLayoutComponent(Component comp)`	Removes the given component from the GridBagLayout layout.
`void setConstraints(Component comp, GridBagConstraints constraints)`	Sets constraints for the component in the GridBagLayout layout.

Table 29 lists the constructors of the `GridBagConstraints` class:

Table 29: Constructors of the GridBagConstraints Class

Constructor	Description
`GridBagConstraints()`	Creates a GridBagConstraints object.
`GridBagConstraints(int gridx, int gridy, int gridwidth, int gridheight, double weightx, double weighty, int anchor, int fill, Insets insets, int ipadx, int ipady))`	Creates a GridBagConstraints object with specific fields.

Table 30 lists the noteworthy fields of the `GridBagConstraints` class:

Table 30: Noteworthy Fields of the GridBagConstraints Class

Fields	Description
`int gridx`	Arranges a row and column at the upper left corner of the display area. The leftmost column in the display area has the address 0, that is gridx=0.
`int gridy`	Arranges the row and column at the top of the display area. The topmost column in the display area has the address 0, that is gridy=0.
`int anchor`	Arranges the component that is smaller than the display area, at some pre-defined location in the display area. That pre-defined location can be any one of the following: `FIRST_LINE_START`, `PAGE_START`, `FIRST_LINE_END`, `LINE_START`, `CENTER`, `LINE_END`, `LAST_LINE_START`, `PAGE_END`, and `LAST_LINE_END`. The default value for this field is CENTER.
`int gridheight`	Specifies the number of cells in a column in a component's display area.
`int gridwidth`	Specifies the number of cells in a row in a component's display area.
`double weightx`	Resizes a component horizontally when the display area is resized.
`double weighty`	Resizes the component vertically when the display area is resized.
`int fill`	Resizes the component according to the space available in the display area, that is, if the size of the component is smaller than the display area, then the component stretches and occupies the display area horizontally or vertically. The way the display area is filled depends on the constraints of the fill field. The constraint can be any one of the following: NONE: Indicates no resizing of a component HORIZONTAL: Resizes a component horizontally to fill the display area VERTICAL: Resizes a component vertically to fill the display area

	BOTH: Resizes a component both horizontally and vertically in the display area.
`Insets insets`	Specifies the external padding of components, that is, the amount of space between the component and the edges of the display area. The four parameters: top, left, bottom, and right, need to be passed to specify a component's padding. You can specify the parameters by using the Insets object. For example, consider the following statement: Insets inset = new Insets (0,0,0,0); In the preceding statement, 0,0,0,0 is the default setting, that is, there is no padding for the components.
`int ipadx`	Adds horizontal space within a component and increases the size of the component width wise.
`int ipady`	Adds vertical space within a component and increases the size of the component height wise.

Now, let's create a frame window and set its layout as `gridbag layout`. Listing 18 shows the use of the `GridBagLayout` class (you can find the `GridBagLayoutDemo.java` file on the CD in the code\chapter8 folder):

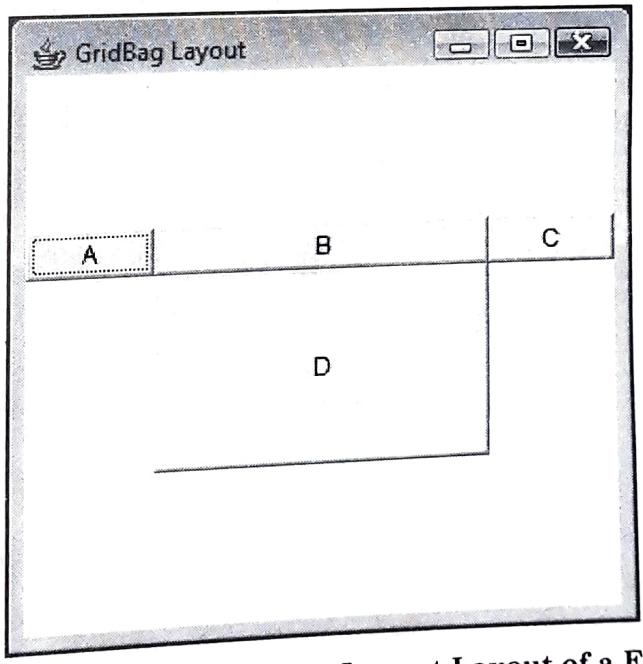
▶ Listing 18: Using the GridBagLayout Class

```
import java.awt.*;
import java.awt.event.*;
class GridBagLayoutDemo extends Frame
{
 Button b1, b2, b3, b4;
 GridBagConstraints cons;
 GridBagLayout gbag;
 GridBagLayoutDemo ()
 {
 gbag = new GridBagLayout(); // Creating GridBag Layout object
 this.setLayout (gbag); // Setting the Layout
 cons = new GridBagConstraints(); // Creating GridBagConstraints object
 b1 = new Button ("A");
 b2 = new Button ("B");
 b3 = new Button ("C");
 b4 = new Button ("D");
 cons.fill = GridBagConstraints.HORIZONTAL; // using the horizontal filling
 cons.gridx = 0; // Setting x,y coordinate to display first button
 cons.gridy = 0;
 cons.weightx = 1; // resizing the component when the frame is resized
 gbag.setConstraints (b1, cons);//seeting the above constraints to button
 cons.gridx = 1; // Setting x,y coordinate to display second button
 cons.gridy = 0;
 gbag.setConstraints (b2, cons);//seeting the above constraints to button
 cons.gridx = 2; // Setting x,y coordinate to display third button
 cons.gridy = 0;
 gbag.setConstraints (b3, cons);//seeting the above constraints to button
 cons.gridx = 1; // Setting x,y coordinate to display third button
 cons.gridy = 2;
 cons.ipadx = 100; // Setting 50 px for ipadx
 cons.ipady = 70;// Setting 100 px for ipady
 gbag.setConstraints (b4, cons);//seeting the above constraints to button
```

```
        this.add (b1);
        this.add (b2);
        this.add (b3);
        this.add (b4);

        addWindowListener (new WindowAdapter()
        {
            public void windowClosing (WindowEvent e)
            {
                System.exit (0);
            }
        });
        }

        public static void main (String args [])
        {
            GridBagLayoutDemo gb = new GridBagLayoutDemo ();
            gb.setTitle ("GridBag Layout");// setting frame title
            gb.setSize (300,300);// setting frame size
            gb.setVisible (true);
        }
    }
```

In Listing 18, we create a class named `GridBagLayoutDemo`, which extends the `Frame` class. Inside this class, we first create the objects `b1`, `b2`, `b3`, and `b4` of the Button class, the cons object of the `GridBagConstraints` class, the `gbag` object of the `GridBagLayout` class, followed by the constructor of this class.

Inside the constructor, we first set the layout as GridBagLayout, followed by the allocation of memory to the objects of the `Button` and `GridBagConstraints` classes. Figure 19 shows the output of Listing 18:



▲ **Figure 19: Displaying the GridBagLayout Layout of a Frame Window**

In Figure 19, the buttons have been arranged according to the constraints listed in Listing 18.

TEST YOUR KNOWLEDGE

Q2. Write a program to arrange the Java buttons in the GridBagLayout layout.

Ans.

```
import java.awt.*;
import java.awt.event.*;
class TestYourKnowledge2 extends Frame
{
 Button b1, b2, b3, b4;
 GridBagConstraints cons;
 GridBagLayout gbag;
 TestYourKnowledge2 ()
 {
 gbag = new GridBagLayout(); // Creating GridBag Layout object
 this.setLayout (gbag); // Setting the Layout
 cons = new GridBagConstraints(); // Creating GridBagConstraints
     object
 b1 = new Button ("Border 1");
 b2 = new Button ("Border 2");
 b3 = new Button ("Border 3");
 b4 = new Button ("Border 4");
 cons.fill = GridBagConstraints.HORIZONTAL; // using the horizontal
     filling
 cons.gridx = 0; // Setting x,y coordinate to display first button
 cons.gridy = 0;
 cons.weightx = 1; // resizing the component when the frame is resized
 gbag.setConstraints (b1, cons);//seeting the above constraints to
     button
 cons.gridx = 1; // Setting x,y coordinate to display second button
 cons.gridy = 0;
 gbag.setConstraints (b2, cons);//seeting the above constraints to
 button
 cons.gridx = 2; // Setting x,y coordinate to display third button
 cons.gridy = 0;
 gbag.setConstraints (b3, cons);//seeting the above constraints to
 button
 cons.gridx = 1; // Setting x,y coordinate to display third button
 cons.gridy = 2;
 cons.ipadx = 100; // Setting 50 px for ipadx
 cons.ipady = 70;// Setting 100 px for ipady
 gbag.setConstraints (b4, cons);//seeting the above constraints to
 button
 this.add (b1);
 this.add (b2);
 this.add (b3);
 this.add (b4);

 addWindowListener (new WindowAdapter()
 {
     public void windowClosing (WindowEvent e)
     {
         System.exit (0);
     }
 });
 }
 public static void main (String args [])
 {
     TestYourKnowledge2 gb = new TestYourKnowledge2 ();
```

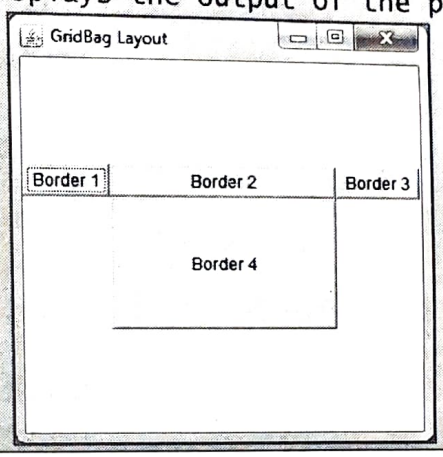
```
        gb.setTitle ("GridBag Layout");// setting frame title
        gb.setSize (300,300);// setting frame size
        gb.setVisible (true);
    }
}
```

Execution:

```
D:\code\chapter8>javac TestYourKnowledge2.java
D:\code\chapter8>java TestYourKnowledge2
```

Output:

The following figure displays the output of the preceding program:



■ The CardLayout Manager

The CardLayout manager displays the containers you pass to it in the form of cards. You can give each card a name and then move from card to card by using the CardLayout's `show()` method. Besides using the `show()` method, you can also display specific cards by using the `first()`, `last()`, `next()`, and `previous()` methods of the `CardLayout` class. Table 31 lists the constructors of the CardLayout class:

Table 31: Constructors of the CardLayout Class

Constructor	Description
`CardLayout()`	Creates a new card layout.
`CardLayout(int hgap, int vgap)`	Creates a new card layout with the given horizontal and vertical gaps.

Table 32 lists noteworthy methods of the CardLayout Class:

Table 32: Noteworthy Methods of the CardLayout Class

Method	Description
`void addLayoutComponent(Component comp, Object constraints)`	Adds the given component to the CardLayout class.
`void addLayoutComponent(String name, Component comp)`	Adds the component with the given name to the CardLayout class.

Table 32: Noteworthy Methods of the CardLayout Class

Method	Description
`void first(Container parent)`	Moves the invoking components to the first card of the current container.
`int getHgap()`	Gets the horizontal gap between components.
`float getLayoutAlignmentX (Container parent)`	Gets the alignment of a component along the x-axis.
`float getLayoutAlignmentY(Containe r parent)`	Gets the alignment of a component along the y-axis.
`int getVgap()`	Gets the vertical gap between components.
`void invalidateLayout(Container target)`	Invalidates the layout of the current container.
`void last(Container parent)`	Moves the invoking component to the last card of a current container.
`void layoutContainer(Container parent)`	Lays out the given container by using this card layout.
`Dimension maximumLayoutSize(Container target)`	Gets the maximum dimensions for the layout, in the given target container.
`Dimension minimumLayoutSize(Container parent)`	Calculates the minimum size for the given panel.
`void next(Container parent)`	Moves to the next card of the given container.
`Dimension preferredLayoutSize(Containe r parent)`	Determines the preferred size of a container argument by using the current card layout.
`void previous(Container parent)`	Moves to the previous card of the given container.
`void removeLayoutComponent(Compon ent comp)`	Removes the given component from the layout.
`void setHgap(int hgap)`	Sets the horizontal gap between components.
`void setVgap(int vgap)`	Sets the vertical gap between components.
`void show(Container parent, String name)`	Moves the invoking component to the current container with the given name.
`String toString()`	Gets a string representation of the current layout.

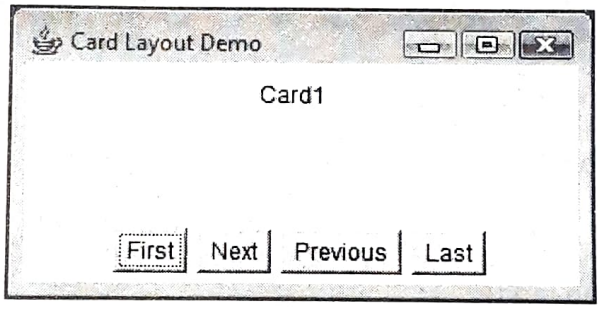
Now, let's create the `CardLayoutDemo` class that displays four buttons, namely first, next, previous, and last, along with a label to indicate the present card number. Listing 19 shows the implementation of the CardLayout manager (you can find the `CardLayoutDemo.java` file on the CD in the code\chapter8 folder):

▶ **Listing 19: Implementing the CardLayout Manager**

```
import java.awt.*;
import java.awt.event.*;
class CardLayoutDemo extends Frame
{
 int current = 1;
 Panel cp;
 CardLayout c;
 CardLayoutDemo()
 {
     setTitle("Card Layout Demo");
     setSize(300,150);
     cp = new Panel();
     c = new CardLayout();
     cp.setLayout(c);
     Panel p1 = new Panel();
     Panel p2 = new Panel();
     Panel p3 = new Panel();
     Panel p4 = new Panel();
     Label l1 = new Label("Card1");
     Label l2 = new Label("Card2");
     Label l3 = new Label("Card3");
     Label l 4 = new Label("Card4");
     p1.add(l1);
     p2.add(l2);
     p3.add(l3);
     p4.add(l4);
     cp.add(p1,"1");
     cp.add(p2,"2");
     cp.add(p3,"3");
     cp.add(p4,"4");
     Panel button = new Panel();
     Button b1 = new Button("First Card");
     Button b2 = new Button("Next Card");
     Button b3 = new Button("Previous Card");
     Button b4 = new Button("Last Card");
     button.add(b1);
     button.add(b2);
     button.add(b3);
     button.add(b4);
     b1.addActionListener(new ActionListener(){
         public void actionPerformed(ActionEvent ae){
             c.first(cp);
             current =1;
         }
     });
     b4.addActionListener(new ActionListener(){
         public void actionPerformed(ActionEvent ae){
             c.last(cp);
             current =4;
         }
     });
         b2.addActionListener(new ActionListener(){
         public void actionPerformed(ActionEvent ae){
             c.next(cp);
         }
     });
```

```
        b3.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent ae){
                c.previous(cp);
            }
        });
        addWindowListener(new WindowAdapter()
        {
            public void windowClosing (WindowEvent e)
            {
                System.exit (0);
            }
        });
        add(cp, BorderLayout.NORTH);
        add(button, BorderLayout.SOUTH);
    }
    public static void main(String args[])
    {
        CardLayoutDemo cle = new CardLayoutDemo();
        cle.setVisible(true);
    }
}
```

In Listing 19, the `CardLayoutDemo` class creates four panels and each panel comprises a label, which is appended to the `cp` panel. In addition, a `button` panel is created, which comprises four buttons. The message of the label changes depending on the button clicked by a user. For example, if the user clicks the `First` button, then the message of the first card is displayed. Figure 20 shows the output of the Listing 19:



▲ **Figure 20: Displaying the Components in the CardLayout Layout**

Now that you know how to implement the layout managers, let's discuss how to create a menu bar in a frame window and add menus to it.

8.6 Working with Menus and Menu Bar

Every GUI user knows about menus. They are those indispensable components that hide all the options of a program by category. Imagine if all the options a word processor could present were available as buttons, visible all at once. In such a scenario, there would be no space to enter text. Menus allow you store those options away in a compact way. This is a very attractive GUI technique, because space is always at a premium in windowed environments.

In AWT programming, a frame window needs to use menus. You can first create a `MenuBar` object and then add the menu bar object to a frame window through the `setMenuBar()` method. You can also create objects of the `Menu` class, which are in turn used to create the

individual menus (such as File and Edit) that appear in the menu bar, and you create objects of the `MenuItem` class to create the actual items in each menu (such as New, Open, Help and Exit).

You can also provide some extra options in menus, for example, submenus that open when you select a menu item, and check boxes that allow you toggle menu items on and off (such as Automatic Spell Checking and View Toolbar).

Table 33 lists noteworthy constructors of the Menu class:

Table 33: Noteworthy Constructors of the Menu Class

Constructor	Description
`Menu()`	Creates a new menu.
`Menu(String label)`	Creates a new menu with the specified label.

The individual menu items are of the MenuItem class. Table 34 lists the constructors of the MenuItem class:

Table 34: Constructors of the MenuItem Class

Constructor	Description
`MenuItem()`	Creates a new menu item with an empty label and no keyboard shortcut.
`MenuItem(String label)`	Creates a new menu item with the specified label but no keyboard shortcut.
`MenuItem(String label, MenuShortcut s)`	Creates a menu item with the specified label and keyboard shortcuts.

Table 35 describes noteworthy methods of the MenuItem class:

Table 35: Noteworthy Methods of the MenuItem Class

Methods	Description
`void deleteShortcut()`	Deletes a MenuShortcut object associated with a menu item.
`MenuShortcut getShortcut()`	Gets a MenuShortcut object associated with a menu item.
`void setShortcut(MenuShortcut s)`	Sets the MenuShortcut object associated with a menu item.
`boolean isEnabled()`	Checks whether or not a menu item is enabled.
`setLabel(String label)`	Sets the label for a menu item.
`void getLabel()`	Gets the label for a menu item.

Now, let's create a simple program in which a frame contains the File and Edit menus on the menu bar, and submenus for each menu. Listing 20 shows the implementation of the `Menu` and `MenuItem` classes (you can find the `MenuDemo.java` file on the CD in the code\chapter8 folder):

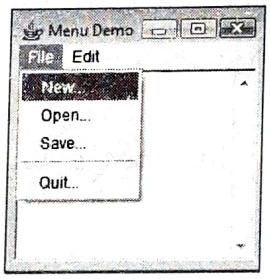
▶ Listing 20: The MenuDemo.java File

```
import java.awt.*;
import java.awt.event.*;
import java.awt.datatransfer.*;
import java.io.*;
class MenuDemo extends Frame
{
 String msg=" ";
 CheckboxMenuItem i13,i14;
 TextArea text;
 MenuDemo()
 {
     MenuBar mbar=new MenuBar();
     text=new TextArea(400,200);
     add(text);
     text.setEditable(true);
     Menu file=new Menu("File");
     MenuItem i1,i2,i3,i4,i5;
     file.add(i1=new MenuItem("New..."));
     file.add(i2=new MenuItem("Open..."));
     file.add(i3=new MenuItem("Save..."));
     file.add(i4=new MenuItem("-"));
     file.add(i5=new MenuItem("Quit..."));
     mbar.add(file);
     Menu edit=new Menu("Edit");
     MenuItem i6,i7,i8,i9,sep;
     edit.add(i6=new MenuItem("Cut"));
     edit.add(i7=new MenuItem("Copy"));
     edit.add(i8=new MenuItem("Paste"));
     edit.add(i9=new MenuItem("-"));
     Menu sub=new Menu("Align");
     MenuItem i10,i11,i12;
     sub.add(i10=new MenuItem("Left"));
     sub.add(i11=new MenuItem("Right"));
     sub.add(i12=new MenuItem("Center"));
     edit.add(sub);
     edit.add(sep=new MenuItem("-"));
     i13=new CheckboxMenuItem("VerticalScrollBar");
     edit.add(i13);
     i14=new CheckboxMenuItem("HorizontalScrollBar");
     edit.add(i14);
     mbar.add(edit);
     Mymenuhandler handler=new Mymenuhandler(this);
     i1.addActionListener(handler);
     i2.addActionListener(handler);
     i3.addActionListener(handler);
     i4.addActionListener(handler);
     i5.addActionListener(handler);
     i6.addActionListener(handler);
     i7.addActionListener(handler);
     i8.addActionListener(handler);
     i9.addActionListener(handler);
     i10.addActionListener(handler);
     i11.addActionListener(handler);
     i12.addActionListener(handler);
     i13.addItemListener(handler);
     i14.addItemListener(handler);
```

```
        MyWindowAdapter adapter=new MyWindowAdapter(this);
        addWindowListener(adapter);
        setMenuBar(mbar);
    }
    public void paint(Graphics g)
    {
    }
    public static void main(String args[])
    {
        MenuDemo cle = new MenuDemo();
        cle.setVisible(true);
        cle.setTitle("Menu Demo");
        cle.setSize(200,200);
    }
}
class MyWindowAdapter extends WindowAdapter
{
    MenuDemo md;
    public MyWindowAdapter(MenuDemo md)
    {
        this.md=md;
    }
    public void windowClosing(WindowEvent we)
    {
        System.exit(0);
    }
}
class Mymenuhandler implements ActionListener,ItemListener
{
    MenuDemo mdemo;
    FileDialog fd;
    String pm="you selected : ",msg=" ";
    String filename;
    public Mymenuhandler(MenuDemo mdemo)
    {
        this.mdemo=mdemo;
    }
    public void actionPerformed(ActionEvent ae)
    {
        String arg=(String)ae.getActionCommand();
        if(arg.equals("New..."))
        msg=pm+"new";
        else if(arg.equals("Open..."))
        msg=pm+"Open";
        else if(arg.equals("Save..."))
        msg=pm+"save";
        else if(arg.equals("Quit..."))
        msg=pm+"quit";
        else if(arg.equals("Edit"))
        msg=pm+"Edit";
        else if(arg.equals("Cut"))
        msg=pm+"Cut";
        else if(arg.equals("Copy"))
        msg=pm+"Copy";
        else if(arg.equals("Paste"))
        msg=pm+"Paste";
        else if(arg.equals("Left"))
        msg=pm+"Left";
```

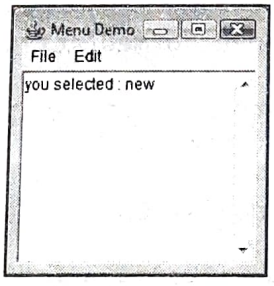
```
            else if(arg.equals("Right"))
            msg=pm+"Right";
            else if(arg.equals("Center"))
            msg=pm+"Center";
            else if(arg.equals("VerticalScrollBar"))
            msg=pm+"Vertical Scroll Bar";
            else if(arg.equals("HorizontalScrollBar"))
            msg=pm+"Horizontal Scroll Bar";
            mdemo.text.setText(msg);
            mdemo.repaint();
        }
        public void itemStateChanged(ItemEvent ie)
        {
            mdemo.repaint();
        }
    }
```

In Listing 20, the MenuDemo class creates a frame window, which contains the `mbar` object of the `MenuBar` class and the `text` object of the `TextArea` class. Depending on the menu selected, a text message is displayed in the text area in the frame window. Figure 21 shows the output of Listing 20:



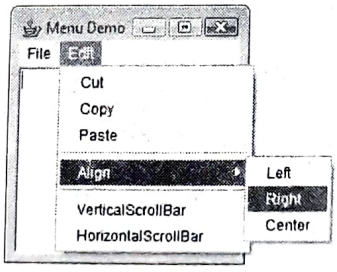
▲ **Figure 21: Displaying the Menu Items on a Frame Window**

Now, when you click the New menu item, a message is displayed in the text area in the frame window, as shown in Figure 22:



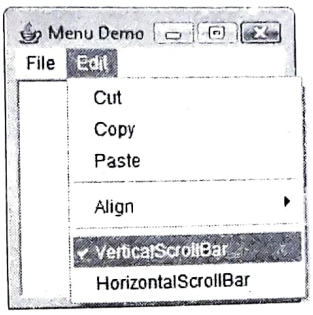
▲ **Figure 22: Displaying a Text Message**

Apart from this, submenus have also been added to the menus, as shown in Figure 23:



▲ **Figure 23: Displaying Submenus of a Menu Item**

Figure 23 displays the Left, Right, and Center submenus of the Align menu. The Menu Demo frame window also contains a checked menu item, as shown in Figure 24:



▲ **Figure 24: Displaying a Checked Menu Item**

The check mark appears when you select the VerticalScrollBar menu item (Figure 24).

Next, let's discuss the implementation of dialog boxes in a frame window.

3.7 Working with Dialog Boxes

AWT supports a special window class, the `Dialog` class, which you can use to create dialog boxes. Unlike a normal frame window, a dialog box does not have the minimize and maximize buttons. Table 36 lists noteworthy constructors of the `Dialog` class:

Table 36: Noteworthy Constructors of the Dialog Class

Constructor	Description
`Dialog(Dialog owner)`	Creates an initially invisible, nonmodal `Dialog` object (that is, a user cannot close the dialog box from the screen before interacting with the program) with an empty title and the given owner dialog box.
`Dialog(Dialog owner, String title)`	Creates an initially invisible, nonmodal `Dialog` object with the given owner dialog box and title.
`Dialog(Dialog owner, String title, boolean modal)`	Creates an initially invisible `Dialog` object with the given owner dialog box, title, and modality.
`Dialog(Dialog owner, String title, boolean modal GraphicsConfiguration gc)`	Creates an initially invisible `Dialog` object with the given owner dialog box, title, modality, and graphics configuration.
`Dialog(Frame owner)`	Creates an initially invisible, nonmodal `Dialog` object with an empty title and the given owner frame.
`Dialog(Frame owner, boolean modal)`	Creates an initially invisible `Dialog` object with an empty title and the given owner frame and modality.
`Dialog(Frame owner, String title)`	Creates an initially invisible, nonmodal `Dialog` object with the given owner frame and title.
`Dialog(Frame owner, String title, boolean modal)`	Creates an initially invisible `Dialog` object with the given owner frame, title, and modality.

Table 36: Noteworthy Constructors of the Dialog Class

Constructor	Description
`Dialog(Frame owner, String title, boolean modal, GraphicsConfiguration gc)`	Creates an initially invisible `Dialog` object with the given owner frame, title, modality, and graphics configuration.

Table 37 lists noteworthy methods of the Dialog class:

Table 37: Noteworthy Methods of the Dialog Class

Method	Description
`void addNotify()`	Displays a dialog box by connecting it to a native screen.
`AccessibleContext getAccessibleContext()`	Gets the `AccessibleContext` associated with a dialog box.
`String getTitle()`	Gets the title of a dialog box.
`void hide()`	Hides a dialog box.
`boolean isModal()`	Indicates whether or not a dialog box is modal (that is, the user must close the dialog box from the screen before interacting with the program).
`boolean isResizable()`	Indicates whether or not a dialog box can be resized by a user.
`boolean isUndecorated()`	Indicates whether or not a dialog box is undecorated.
`protected String paramString()`	Gets the parameter string representing the state of a dialog box.
`void setModal(boolean b)`	Specifies whether or not a dialog box is modal.
`void setResizable(boolean resizable)`	Specifies whether or not a dialog box can be resized by the user.
`void setTitle(String title)`	Sets the title of a dialog box.
`void setUndecorated(boolean undecorated)`	Disables or enables decorations for a dialog box.
`Dialog.ModalityType getModalityType()`	Returns the modality type of a dialog box.
`void setModalityType (Dialog.ModalityType)`	Sets the modality type of a dialog box.
`void setVisible(boolean bool)`	Shows or hides a dialog box depending on the value of the bool parameter.
`void toBack()`	Sends a window to the back if the current window is visible.

There are a few things to note in Tables 36 and 37. One is that you can create either modal dialog boxes or nonmodal dialog boxes, depending on the `Dialog` constructor you use, or if you use the `setModal()` method. Another point is that you can use the `setResizeable()` method to create a dialog box that the user can resize.

Let's create a simple frame window, which contains a menu bar and in which, when you click the New submenu item, a dialog box appears. Listing 21 creates the `SampleDialog` class, which creates a dialog box (you can find the `DialogDemo.java` file on the CD in the code\chapter8 folder):

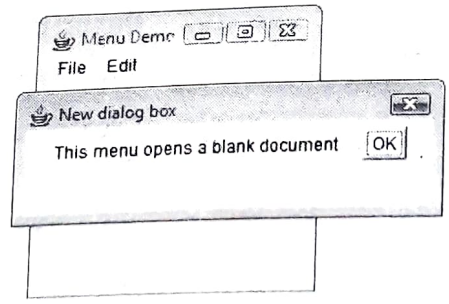
▶ **Listing 21: The DialogDemo.java File**

```
import java.awt.*;
import java.awt.event.*;
import java.awt.datatransfer.*;
import java.io.*;
class DialogDemo extends Frame
{
  String msg=" ";
  CheckboxMenuItem i13,i14;
  TextArea text;
  DialogDemo()
  {
      MenuBar mbar=new MenuBar();
      text=new TextArea(400,200);
      add(text);
      text.setEditable(true);
      Menu file=new Menu("File");
      MenuItem i1,i2,i3,i4,i5;
      file.add(i1=new MenuItem("New..."));
      file.add(i2=new MenuItem("Open..."));
      file.add(i3=new MenuItem("Save..."));
      file.add(i4=new MenuItem("-"));
      file.add(i5=new MenuItem("Quit..."));
      mbar.add(file);
      Menu edit=new Menu("Edit");
      MenuItem i6,i7,i8,i9,sep;
      edit.add(i6=new MenuItem("Cut"));
      edit.add(i7=new MenuItem("Copy"));
      edit.add(i8=new MenuItem("Paste"));
      edit.add(i9=new MenuItem("-"));
      Menu sub=new Menu("Align");
      MenuItem i10,i11,i12;
      sub.add(i10=new MenuItem("Left"));
      sub.add(i11=new MenuItem("Right"));
      sub.add(i12=new MenuItem("Center"));
      edit.add(sub);
      edit.add(sep=new MenuItem("-"));
      i13=new CheckboxMenuItem("VerticalScrollBar");
      edit.add(i13);
      i14=new CheckboxMenuItem("HorizontalScrollBar");
      edit.add(i14);
      mbar.add(edit);
      Mymenuhandler handler=new Mymenuhandler(this);
      i1.addActionListener(handler);
      i2.addActionListener(handler);
      i3.addActionListener(handler);
      i4.addActionListener(handler);
      i5.addActionListener(handler);
      i6.addActionListener(handler);
      i7.addActionListener(handler);
      i8.addActionListener(handler);
```

```
            i9.addActionListener(handler);
            i10.addActionListener(handler);
            i11.addActionListener(handler);
            i12.addActionListener(handler);
            i13.addItemListener(handler);
            i14.addItemListener(handler);
            MyWindowAdapter adapter=new MyWindowAdapter(this);
            addWindowListener(adapter);
            setMenuBar(mbar);
        }
        public void paint(Graphics g)
        {
        }
        public static void main(String args[])
        {
            DialogDemo cle = new DialogDemo();
            cle.setVisible(true);
            cle.setTitle("Menu Demo");
            cle.setSize(200,200);
        }
    }
    class MyWindowAdapter extends WindowAdapter
    {
     DialogDemo md;
     public MyWindowAdapter(DialogDemo md)
     {
         this.md=md;
     }
     public void windowClosing(WindowEvent we)
     {
         System.exit(0);
     }
    }
    class SampleDialog extends Dialog implements ActionListener
    {
     SampleDialog(Frame parent,String title)
     {
         super(parent,title,false);
         setLayout(new FlowLayout());
         setSize(300,100);
         add(new Label("This menu opens a blank document"));
         Button b;
         add(b=new Button("OK"));
         b.addActionListener(this);
    }
    public void actionPerformed(ActionEvent ae)
    {
        dispose();
    }
}
class Mymenuhandler implements ActionListener,ItemListener
{
 DialogDemo mdemo;
 FileDialog fd;
 String pm="you selected : ",msg=" ";
 String filename;
 public Mymenuhandler(DialogDemo mdemo)
 {
```

```
        this.mdemo=mdemo;
    }
    public void actionPerformed(ActionEvent ae)
    {
        String arg=(String)ae.getActionCommand();
        if(arg.equals("New..."))
        {
            msg=pm+"new";
            SampleDialog d=new SampleDialog(mdemo,"New dialog box");
            d.setVisible(true);
        }
        else if(arg.equals("Open..."))
        msg=pm+"Open";
        else if(arg.equals("Save..."))
        msg=pm+"save";
        else if(arg.equals("Quit..."))
        msg=pm+"quit";
        else if(arg.equals("Edit"))
        msg=pm+"Edit";
        else if(arg.equals("Cut"))
        msg=pm+"Cut";
        else if(arg.equals("Copy"))
        msg=pm+"Copy";
        else if(arg.equals("Paste"))
        msg=pm+"Paste";
        else if(arg.equals("Left"))
        msg=pm+"Left";
        else if(arg.equals("Right"))
        msg=pm+"Right";
        else if(arg.equals("Center"))
        msg=pm+"Center";
        else if(arg.equals("VerticalScrollBar"))
        msg=pm+"Vertical Scroll Bar";
        else if(arg.equals("HorizontalScrollBar"))
        msg=pm+"Horizontal Scroll Bar";
        mdemo.text.setText(msg);
        mdemo.repaint();
    }
    public void itemStateChanged(ItemEvent ie)
    {
        mdemo.repaint();
    }
    }
```

In Listing 21, the menu items, File and Edit, are first added to a menu bar and then added to a frame window. The instance of the `SampleDialog` class is created when you select a File menu item. Therefore, when you click the New menu item, a dialog box appears, as shown in Figure 25:



▲ **Figure 25: Displaying a Dialog Box**

Figure 25 shows the New dialog box when you select the New menu item from the File menu.

Next, let's now discuss how to create images in Java by using the java.awt package.

8.8 Working with Images

Images are objects of the `Image` class, which is a part of the `java.awt` package. They are implemented by using the classes found in the `java.awt.image` package. AWT excels at handling images, and you will see what it is capable of in this section. We will take a look at various ways to load images in formats such as GIF and JPG, resize images, wait until images are fully loaded before displaying them, draw images offscreen before displaying them (a process called *double buffering*), and animate images.

Images are manipulated using the classes found in the `java.awt.image` package. Table 38 lists noteworthy fields of the `Image` class:

Table 38: Noteworthy Fields of the Image Class

Field	Description
`protected float accelerationPriority`	Indicates the priority for loading an image in a frame window.
`static int SCALE_AREA_AVERAGING`	Indicates the use of the area-averaging image-scaling algorithm.
`static int SCALE_DEFAULT`	Indicates the use of the default image-scaling algorithm.
`static int SCALE_FAST`	Indicates the selection of an image-scaling algorithm that gives higher priority to the scaling speed of an image.
`static int SCALE_REPLICATE`	Indicates the use of an image-scaling algorithm in the `ReplicateScaleFilter` class.
`static int SCALE_SMOOTH`	Indicates the selection of an image-scaling algorithm that gives higher priority to image smoothness.
`static Object UndefinedProperty`	Returns the `UndefinedProperty` object whenever a property that is not defined for a particular image is requested.

The constructor of the Image class is Image(), which creates an Image object.

Table 39 lists noteworthy methods of the Image class:

Table 39: Noteworthy Methods of the Image Class

Method	Description
`abstract void flush()`	Flushes all the resources used by an image.
`abstract Graphics getGraphics()`	Creates a graphics context for drawing an offscreen image.
`abstract int getHeight(ImageObserver observer)`	Indicates the height of an image.

Table 39: Noteworthy Methods of the Image Class

Method	Description
`abstract Object getProperty(String name, ImageObserver observer)`	Gets a property of an image of the specified name.
`Image getScaledInstance(int width, int height, int Hints)`	Creates a scaled version of an image.
`abstract ImageProducer getSource()`	Gets the object that produces the pixels for an image.
`abstract int getWidth (ImageObserver observer)`	Indicates the width of an image.

Listing 22 shows how to load an image saved on your computer (you can find the `ImageDemo.java` file on the CD in the code\chapter8 folder):

▶ **Listing 22: The ImageDemo.java File**

```
import java.awt.*;
import java.awt.event.*;
public class ImageDemo extends Frame
{
  Image image;
  public ImageDemo()
  {
      this.setLayout (null);
      Toolkit toolkit = Toolkit.getDefaultToolkit();
      image=toolkit.getImage("java.jpg");
      addWindowListener(new WindowAdapter() {
          public void windowClosing(WindowEvent e)
          {
              System.exit(0);
              }
      });
  }

  public void paint(Graphics graphics) {
      graphics.drawImage(image, 0, 0, null);
  }
  public static void main(String[] args) {
      ImageDemo i = new ImageDemo();
      i.setTitle("Image Demo");
      i.setSize(300,300);
      i.setVisible(true);
  }
}
```

In Listing 22, the `ImageDemo` class creates an instance of the `Image` class, that is, `image`. The instance of the Toolkit class is then created and the `getImage()` method is invoked by passing the name of the file to be loaded. The `paint()` method is used to draw the loaded

image by using the `drawImage()` method. Figure 26 shows the loaded image in a frame window:

▲ **Figure 26: Displaying an Image on a Frame Window**

Figure 26 loads an image in a frame window.

TEST YOUR KNOWLEDGE

Q3. Write a program to load an image in a frame window.

Ans.

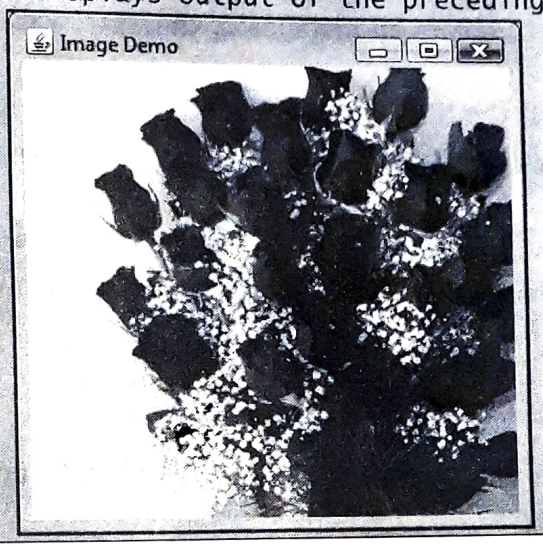
```
import java.awt.*;
import java.awt.event.*;
public class TestYourKnowledge3 extends Frame {
 Image image;
 public TestYourKnowledge3()
 {
 this.setLayout (null);
 Toolkit toolkit = Toolkit.getDefaultToolkit();
 image=toolkit.getImage("flower.jpg");
 addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
            System.exit(0);
            }
    });
 }

 public void paint(Graphics graphics) {
    graphics.drawImage(image, 0, 0, null);
 }
 public static void main(String[] args) {
    TestYourKnowledge3 i = new TestYourKnowledge3();
    i.setTitle("Image Demo");
    i.setSize(300,300);
    i.setVisible(true);
 }
}
```

Execution:

```
D:\code\chapter8>javac TestYourKnowledge3.java
D:\code\chapter8>java TestYourKnowledge3
```

Output:

The following figure displays output of the preceding program:



With this, we come to the end of the chapter. Before moving on to the next chapter, let's recap the main points covered in this chapter by a brief summary.

Summary

This chapter discusses the java.awt package, which contains classes, such as Component, Container, Panel, Window, and Frame, along with their constructors and methods. This chapter also shows how to create a frame window and set its features, such as size, visibility, and title, and add various components to it, such as labels, buttons, check boxes and radio buttons. Apart from this, this chapter discusses how to add a menu bar, menu items and submenus to a menu in a frame window. We also learn how to create a dialog box and to load images on a frame window.

Review Questions

Now, let's solve the following types of questions related to this chapter:

- ❒ True or False
- ❒ Multiple Choice
- ❒ Short Answers
- ❒ Debugging

■ True or False

State True or False for the following:

1. The java.awt package is used to create CUI-based applications. **False**
2. The Component class is the base class of the `java.awt` package. **True**
3. The `setVisible()` method is used to show or hide a frame window. **True**
4. The GridBagLayout manager is a type of layout manager. **True**

5. The `setBackground()` method is a method of the `Component` class. **True**
6. The TextArea class has only one constructor, which is `TextArea()`. **False**
7. The `setFont()` method is of the `Component` class. **True**
8. You need to create an instance of the `Image` class to load an image on a frame window. **True**
9. You can set the title of a frame window by invoking the `setTitle()` method. **True**
10. The Panel class is the subclass of the Container class and superclass of the Applet class. **True**
11. To work with images, you need to import the `java.awt.image` package. **True**
12. The Image class has only one constructor, which is Image(). **True**

■ Multiple Choice Questions

Answer the following multiple choice questions:

Q1. Which of the following packages is used to create GUI-based applications?

A. The java.awt package
B. The java.util package
C. The java.applet package
D. The java.lang package

Ans. Option A is correct.

Q2. Which of the following is not a method of the Container class?

A. getFont()
B. add(Component comp)
C. getLayout()
D. update(Graphics g)

Ans. Option A is correct.

Q3. Which of the following is not a constructor of the Label class?

A. Label()
B. Label(String str)
C. Label(String str, int alignment)
D. Label(int alignment, String str)

Ans. Option D is correct.

Q4. Which of the following is not an AWT component?

A. Push buttons
B. Labels
C. Text fields
D. Dialog boxes

Ans. Option D is correct.

Q5. Which of the following is the default layout of a frame window?

A. Card Layout
B. Border Layout
C. Flow Layout
D. Grid Layout

Ans. Option B is correct.

Q6. Which of the following methods is used to load an image on a frame window?

A. createImage()
B. getImage()
C. loadImage()
D. manageImage()

Ans. Option B is correct.

Q7. Which of the following methods is used to flush all resources of an image?

A. getGraphics()
B. getHeight(ImageObserver observer)
C. flush()
D. getSource()

Ans. Option C is correct.

Q8. Which of the following methods is used to set a label for a button in a frame window?

A. setLabel(str)
B. setLabel()
C. setLabel(String str)
D. setLabel(int i)

Ans. Option C is correct.

Q9. Which of the following is not a method of the TextField class?

A. getText()
B. getColumns()
C. setText(String str)
D. setLabel(String str)

Ans. Option D is correct.

Q10. Which of the following methods is used to delete a shortcut of a menu item?

A. deleteShortcut()
B. deleteShortcut(String str)
C. deleteShortcut(Shortcut s)
D. removeShortcut()

Ans. Option A is correct.

Q11. Which of the following is not a constructor of the List class?

A. List()
B. List(int rows)
C. List(int rows, boolean multipleMode)
D. List(String str)

Ans. Option D is correct.

Q12. Which of the following is not a method of the Button class?

A. getLabel()
B. setLabel(String str)
C. addActionListener(ActionListener al)
D. setText(String str)

Ans. Option D is correct.

■ Short Answers Questions

Answer the following short answer questions:

Q1. What do you mean by GUI-based applications?

Ans. A GUI-based application is an application in which a user interacts through various components, such as text fields and buttons. Java provides Abstract Window Toolkit (AWT) and the Swing Application Programming Interface (API) to create interactive GUI-based applications.

Q2. Explain noteworthy classes provided in the java.awt package.

Ans. The AWT API provides various classes, stored in the java.awt package, to create GUI components in the application. Following are the classes provided in the java.awt package:

- **Component:** Serves as an abstract superclass for various AWT components, such as buttons and check boxes.

- **Container**: Contains AWT components, such as buttons and check boxes.
- **Panel**: Creates a panel for a Windows application.
- **Window**: Provides the container where a Windows application can attach other components, including other panels. The Window class is a subclass of the Container class.
- **Frame**: Creates a frame for a Windows application.

Q3. What is the role of the Component class?

Ans. The Component class is the base class of the java.awt package on which all the visual components are based. For example, the Button class is directly derived from the Component class. The Component class also contains many methods that are used to create GUI-based applications.

Q4. What is the role of the Container class?

Ans. The Container class is an abstract class that cannot be instantiated and some of its methods must be implemented by its subclass.

Q5. What is the role of the Panel class?

Ans. The Panel class is a subclass of the Container class and a superclass of the Applet class. A panel serves as a container for attaching other components or panels. You can add a title bar, menu bar, or border to a panel window.

Q6. What is the role of the Window class?

Ans. The Window class is a subclass of the Container class. An object of the Window class is a top-level window with no menu bar or borders. The WindowAdapter class or WindowListener interface responds to events, such as WINDOW_OPENED and WINDOW_CLOSED, of the Window class.

Q7. What is the role of the Frame class?

Ans. The Frame class is a subclass of the Window class and contains a title bar, menu bar, and borders; therefore, when you create an object of the Frame class within an applet a warning message is displayed during execution. The default layout of a frame window is BorderLayout. A frame window generates events, such as WINDOW_OPENED, WINDOW_CLOSED, WINDOW_ACTIVATED, and WINDOW_DEACTIVATED.

Q8. What is the role of the Button class?

Ans. The Button class is used to create and add the push buttons to a frame window that can be used to perform specific functions, such as calculating the sum of two numbers on clicking the button or changing the background color of the frame window.

Q9. How to create a drop-down list in a frame window?

Ans. The Choice class helps you to create a drop-down list. This class has only one constructor, Choice(), which is used to create an empty choice list.

Q10. What are the various types of layout managers used to arrange Java components in a frame window?

Ans. Java provides various layout managers, which are used to arrange components, such as buttons and radio buttons, in a frame window. A layout manager

implements the LayoutManager interface of the java.awt package to arrange these components in the frame window. Following are the layout managers used to arrange components:

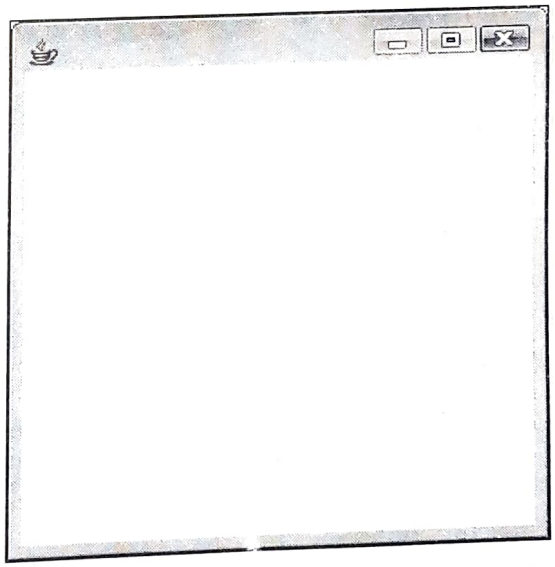
- The FlowLayout Manager
- The BorderLayout Manager
- The GridLayout Manager
- The GridBagLayout Manager
- The CardLayout Manager

Debugging Exercises

Q1. What will be the output of the following program?

```
import java.awt.*;
class Program1 extends Frame
{
 public static void main (String args [])
 {
     Program1 f = new Program1();
     f.setSize (300,300);
     f.setVisible (true); // making the window visible
 }
}
```

Ans. When you compile and execute the preceding program, a frame window appears, as shown in Figure 27:



▲ **Figure 27: Displaying a Blank Frame Window**

Q2. What will be the output of the following program?

```
import java.awt.*;
class Program2 extends Frame
{
public static void main (String args[])
{
    Program2 f = new Program2();
    f.setSize (300,300);
```

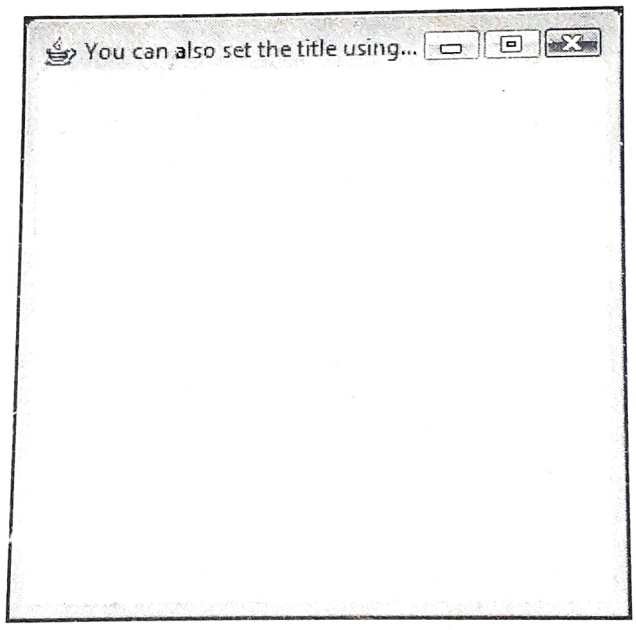
```
    }
}
```

Ans. When you compile and execute the preceding program, it compiles and executes successfully, but nothing is displayed, as the setVisible() method is not set to be true.

Q3. What will be the output of the following program?

```
import java.awt.*;
class Program3 extends Frame
{
 public static void main(String args[])
 {
     Program3 f = new Program3();
     f.setSize (300,300);
     f.setVisible (true);
     f.setTitle ("You can also set the title using setTitle() ");
 }
}
```

Ans. After compilation and execution of the preceding program, the output is displayed, as shown in Figure 28:



▲ **Figure 28: Displaying a Frame Window with a Title**