UNIT 4: SWINGS

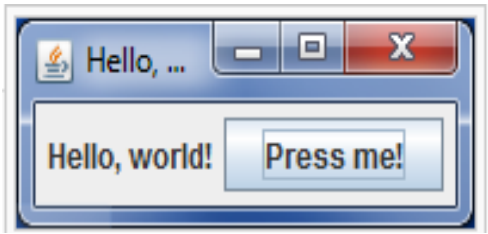
Swing is a GUI widget toolkit for Java . It is part of Oracle's Java Foundation Classes (JFC) — an API for providing a graphical user interface (GUI) for Java programs.

Swing is a set of classes that provides more powerful and flexible components than are possible with the AWT (Abstract Window Toolkit).

In addition to the familiar components, such as buttons, check boxes, and labels, Swing supplies several exciting additions, including tabbed panes, scroll panes, trees, and tables.

Even familiar components such as buttons have more capabilities in Swing. For example, a button may have both an image and a text string associated with it. Also, the image can be changed as the state of the button changes.

Unlike AWT components, Swing components are not implemented by platform-specific code. Instead, they are written entirely in Java and, therefore, are platform-independent. The term *lightweight* is used to describe such elements.




```
Import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.SwingUtilities;
 public class SwingExample implements Runnable {
  @Override
    public void run() {
        // Create the window
        JFrame f = new JFrame("Hello, !");
        // Sets the behavior for when the window is closed
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // Add a layout manager so that the button is not placed on top of the label
        f.setLayout(new FlowLayout());
        // Add a label and a button
        f.add(new JLabel("Hello, world!"));
        f.add(new JButton("Press me!"));
        // Arrange the components inside the window
        f.pack();
        // By default, the window is not visible. Make it visible.
        f.setVisible(true);
    }
   public static void main(String[] args) {
        SwingExample se = new SwingExample();
        // Schedules the application to be run at the correct time in the event queue.
        SwingUtilities.invokeLater(se);
    }}
```

The number of classes and interfaces in the Swing packages is substantial, and this chapter provides an overview of just a few.

Class	Description
AbstractButton	Abstract superclass for Swing buttons.
ButtonGroup	Encapsulates a mutually exclusive set of buttons.

ImageIcon	Encapsulates an icon.
JApplet	The Swing version of Applet.
JButton	The Swing push button class.
JCheckBox	The Swing check box class.
JComboBox	Encapsulates a combo box (an combination of a drop-down list and text field).
JLabel	The Swing version of a label.
JRadioButton	The Swing version of a radio button.
JScrollPane	Encapsulates a scrollable window.
JTabbedPane	Encapsulates a tabbed window.
JTable	Encapsulates a table-based control.
JTextField	The Swing version of a text field.
JTree	Encapsulates a tree-based control.

DIFFERENCE BETWEEN AWT & SWING

AWT	SWING
AWT uses Applet for running GUI Applications.	Swing uses JApplet for running GUI Applications.
AWT provides less features & components than swing.	Swing has got variety of components & features which are not present in AWT
It is platform dependent code.	This is platform independent code.
AWT has predefined formats of appearance & behaviour of elements.	Swing provides a facility of different appearance & behaviour of the same elements.

JAPPLET

Fundamental to Swing is the JApplet class, which extends Applet. Applets that use Swing must be subclasses of JApplet.

JApplet is rich with functionality that is not found in Applet. For example, JApplet supports various "panes," such as the contentpane, the glass pane, and the root pane. For the examples in this chapter, we will not be using most of JApplet's enhanced features.

However, one difference between Applet and JApplet is important to this discussion, because it is used by the sample applets in this chapter.

When adding a component to an instance of JApplet, do not invoke the add( ) method of the applet. Instead, call add( ) for the content pane of the JApplet object. The content pane can be obtained via the method shown here: **Container getContentPane( )**
The add( ) method of Container can be used to add a component to a content pane.
Its form is shown here: **void add(comp)**

Here, comp is the component to be added to the content pane.

ICONS

In Swing, icons are encapsulated by the ImageIcon class, which paints an icon from an image. Two of its constructors are shown here:

ImageIcon(String filename)
ImageIcon(URL url)

The first form uses the image in the file named filename. The second form uses the image in the resource identified by url.

The ImageIcon class implements the Icon interface that declares the methods shown here:

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

LABELS

Swing labels are instances of the JLabel class, which extends JComponent. It can display text and/or an icon. Some of its constructors are shown here:

JLabel(Icon i)
Label(String s)
JLabel(String s, Icon i, int align)

Here, s and i are the text and icon used for the label.
The align argument is either LEFT,RIGHT, CENTER, LEADING, or TRAILING. These constants are defined in the SwingConstants interface, along with several others used by the Swing classes.

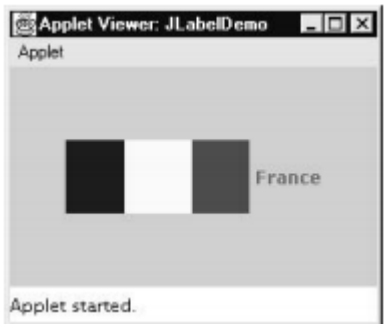
The icon and text associated with the label can be read and written by the following methods:
Icon getIcon( )
String getText( )
void setIcon(Icon i)
void setText(String s)
Here, i and s are the icon and text, respectively.

Program:
The following example illustrates how to create and display a label containing both an icon and a string. The applet begins by getting its content pane. Next, an ImageIcon object is created for the file france.gif. This is used as the second argument to the JLabel constructor. The first and last arguments for the JLabel constructor are the label text and the alignment. Finally, the label is added to the content pane.

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

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

Output from this applet is shown here:



TEXT FIELDS

The Swing text field is encapsulated by the JTextComponent class, which extends JComponent. It provides functionality that is common to Swing text components.

One of its subclasses is JTextField, which allows you to edit one line of text. Some of its constructors are shown here:

JTextField( )
JTextField(int cols)
JTextField(String s, int cols)
JTextField(String s)

Here, s is the string to be presented, and cols is the number of columns in the text field.

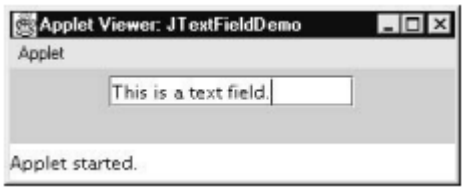
Program:

The following example illustrates how to create a text field. The applet begins by getting its content pane, and then a flow layout is assigned as its layout manager. Next, a JTextField object is created and is added to the content pane.

```
import java.awt.*;
import javax.swing.*;
/*
<applet code="JTextFieldDemo" width=300 height=50>
```

```
</applet>
*/
public class JTextFieldDemo extends JApplet {
JTextField jtf;
public void init() {
// Get content pane
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());
// Add text field to content pane
jtf = new JTextField(15);
contentPane.add(jtf);
}
}
```

Output from this applet is shown here:



BUTTONS

Swing buttons provide features that are not found in the Button class defined by the AWT.

For example, you can associate an icon with a Swing button. Swing buttons are subclasses of the AbstractButton class, which extends JComponent. AbstractButton contains many methods that allow you to control the behavior of buttons, check boxes, and radio buttons.

For example, you can define different icons that are displayed for the component when it is disabled, pressed, or selected. Another icon can be used as a rollover icon, which is displayed when the mouse is positioned over that component.

The following are the methods that control this behavior:
void setDisabledIcon(Icon di)
void setPressedIcon(Icon pi)
void setSelectedIcon(Icon si)
void setRolloverIcon(Icon ri)

Here, di, pi, si, and ri are the icons to be used for these different conditions.
The text associated with a button can be read and written via the following methods:
String getText( )
void setText(String s)

Here, s is the text to be associated with the button.
Concrete subclasses of AbstractButton generate action events when they are pressed. Listeners register and unregister for these events via the methods shown here:

void addActionListener(ActionListener al)
void removeActionListener(ActionListener al)

Here, al is the action listener.

The JButton Class

The JButton class provides the functionality of a push button. JButton allows an icon, a string, or both to be associated with the push button. Some of its constructors are shown here:

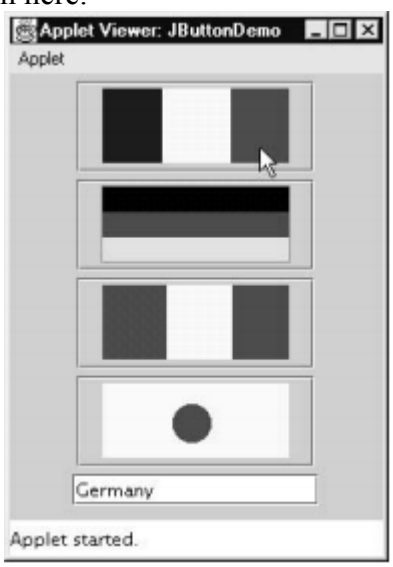
JButton(Icon i)
JButton(String s)
JButton(String s, Icon i)
Here, s and i are the string and icon used for the button.

Program:
The following example displays four push buttons and a text field. Each button displays an icon that represents the flag of a country. When a button is pressed, thename of that country is displayed in the text field. The applet begins by getting its content pane and setting the layout manager of that pane. Four image buttons are created and added to the content pane. Next, the applet is registered to receive action events that are generated by the buttons. A text field is then created and added to the applet. Finally, a handler for action events displays the command string that is associated with the button. The text field is used to present this string.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="JButtonDemo" width=250 height=300>
</applet>
*/
public class JButtonDemo extends JApplet
implements ActionListener {
JTextField jtf;
public void init() {
// Get content pane
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());

// Add buttons to content pane
ImageIcon france = new ImageIcon("france.gif");
JButton jb = new JButton(france);
jb.setActionCommand("France");
jb.addActionListener(this);
contentPane.add(jb);

ImageIcon germany = new ImageIcon("germany.gif");
jb = new JButton(germany);
```

```
jb.setActionCommand("Germany");
jb.addActionListener(this);
contentPane.add(jb);

ImageIcon italy = new ImageIcon("italy.gif");
jb = new JButton(italy);
jb.setActionCommand("Italy");
jb.addActionListener(this);
contentPane.add(jb);

ImageIcon japan = new ImageIcon("japan.gif");
jb = new JButton(japan);
jb.setActionCommand("Japan");
jb.addActionListener(this);
contentPane.add(jb);

// Add text field to content pane
jtf = new JTextField(15);
contentPane.add(jtf);
}
public void actionPerformed(ActionEvent ae) {
jtf.setText(ae.getActionCommand());
}
}
```

Output from this applet is shown here:



CHECK BOXES

The JCheckBox class, which provides the functionality of a check box, is a concrete implementation of AbstractButton.

Its immediate superclass is JToggleButton, which provides support for two-state buttons. Some of its constructors are shown here:

JCheckBox(Icon i)
JCheckBox(Icon i, boolean state)

JCheckBox(String s)
JCheckBox(String s, boolean state)
JCheckBox(String s, Icon i)
JCheckBox(String s, Icon i, boolean state)

Here, i is the icon for the button. The text is specified by s. If state is true, the check box is initially selected. Otherwise, it is not.

The state of the check box can be changed via the following method:

void setSelected(boolean state)
Here, state is true if the check box should be checked.

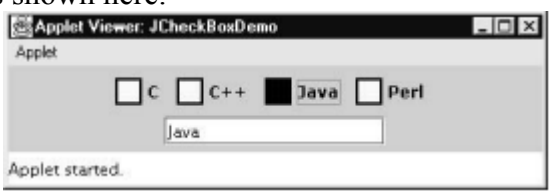
Program:
The following example illustrates how to create an applet that displays four check boxes and a text field. When a check box is pressed, its text is displayed in the text field. The content pane for the JApplet object is obtained, and a flow layout is assigned as its layout manager. Next, four check boxes are added to the content pane, and icons are assigned for the normal, rollover, and selected states. The applet is then registered to receive item events. Finally, a text field is added to the content pane.

When a check box is selected or deselected, an item event is generated. This is handled by itemStateChanged( ). Inside itemStateChanged( ), the getItem( ) method gets the JCheckBox object that generated the event. The getText( ) method gets the text for that check box and uses it to set the text inside the text field.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="JCheckBoxDemo" width=400 height=50>
</applet>
*/
public class JCheckBoxDemo extends JApplet
implements ItemListener {
JTextField jtf;
public void init() {

// Get content pane
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());

// Create icons
ImageIcon normal = new ImageIcon("normal.gif");
ImageIcon rollover = new ImageIcon("rollover.gif");
ImageIcon selected = new ImageIcon("selected.gif");

// Add check boxes to the content pane
JCheckBox cb = new JCheckBox("C", normal);
cb.setRolloverIcon(rollover);
cb.setSelectedIcon(selected);
cb.addItemListener(this);
```

```
contentPane.add(cb);

cb = new JCheckBox("C++", normal);
cb.setRolloverIcon(rollover);
cb.setSelectedIcon(selected);
cb.addItemListener(this);
contentPane.add(cb);

cb = new JCheckBox("Java", normal);
cb.setRolloverIcon(rollover);
cb.setSelectedIcon(selected);
cb.addItemListener(this);
contentPane.add(cb);

cb = new JCheckBox("Perl", normal);
cb.setRolloverIcon(rollover);
cb.setSelectedIcon(selected);
cb.addItemListener(this);
contentPane.add(cb);

// Add text field to the content pane
jtf = new JTextField(15);
contentPane.add(jtf);
}
public void itemStateChanged(ItemEvent ie) {
JCheckBox cb = (JCheckBox)ie.getItem();
jtf.setText(cb.getText());
}
}
```

Output from this applet is shown here:



RADIO BUTTON

Radio buttons are supported by the JRadioButton class, which is a concrete implementation of AbstractButton.

Its immediate superclass is JToggleButton, which provides support for two-state buttons. Some of its constructors are shown here:

```
JRadioButton(Icon i)
JRadioButton(Icon i, boolean state)
JRadioButton(String s)
JRadioButton(String s, boolean state)
JRadioButton(String s, Icon i)
JRadioButton(String s, Icon i, boolean state)
```

Here, i is the icon for the button. The text is specified by s. If state is true, the button is initially selected. Otherwise, it is not.

Radio buttons must be configured into a group. Only one of the buttons in that group can be selected at any time. For example, if a user presses a radio button that is in a group, any previously selected button in that group is automatically deselected.

The **ButtonGroup** class is instantiated to create a button group. Its default constructor is invoked for this purpose. Elements are then added to the button group via the following method:

void add(AbstractButton ab)

ab is a reference to the button to be added to the group.

Program:

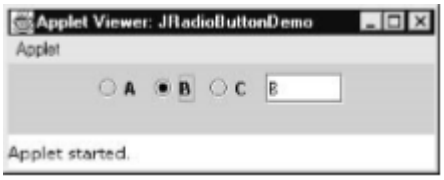
The following example illustrates how to use radio buttons. Three radio buttons and one text field are created. When a radio button is pressed, its text is displayed in the text field. First, the content pane for the JApplet object is obtained and a flow layout is assigned as its layout manager. Next, three radio buttons are added to the content pane. Then, a button group is defined and the buttons are added to it. Finally, a text field is added to the content pane.

Radio button presses generate action events that are handled by **actionPerformed().** The **getActionCommand()** method gets the text that is associated with a radio button and uses it to set the textfield.

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

// Get content pane
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());

// Add radio buttons to content pane
JRadioButton b1 = new JRadioButton("A");
b1.addActionListener(this);
contentPane.add(b1);

JRadioButton b2 = new JRadioButton("B");
b2.addActionListener(this);
contentPane.add(b2);

JRadioButton b3 = new JRadioButton("C");
b3.addActionListener(this);
contentPane.add(b3);
```

```
// Define a button group
ButtonGroup bg = new ButtonGroup();
bg.add(b1);
bg.add(b2);
bg.add(b3);

// Create a text field and add it to the content pane
tf = new JTextField(5);
contentPane.add(tf);
}
public void actionPerformed(ActionEvent ae)
{ tf.setText(ae.getActionCommand());
}
}
```

Output from this applet is shown here:



COMBO BOXES

Swing provides a combo box (a combination of a text field and a drop-down list) through the JComboBox class, which extends JComponent. A combo box normally displays one entry. However, it can also display a drop-down list that allows a user to select a different entry. You can also type your selection into the text field. Two of JComboBox's constructors are shown here:

JComboBox( )
JComboBox(Vector v)
Here, v is a vector that initializes the combo box.

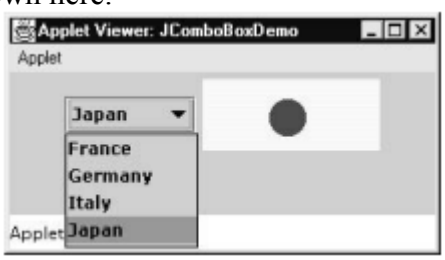
Items are added to the list of choices via the addItem( ) method, whose signature is shown here:
void addItem(Object obj)
Here, obj is the object to be added to the combo box.

Program:
The following example contains a combo box and a label. The label displays an icon. The combo box contains entries for "France", "Germany", "Italy", and "Japan". When a country is selected, the label is updated to display the flag for that country.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="JComboBoxDemo" width=300 height=100>
</applet>
```

```
*/
public class JComboBoxDemo extends JApplet
implements ItemListener {

JLabel jl;
ImageIcon france, germany, italy, japan;

public void init() {

// Get content pane
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());

// Create a combo box and add it to the panel
JComboBox jc = new JComboBox();
jc.addItem("France");
jc.addItem("Germany");
jc.addItem("Italy");
jc.addItem("Japan");
jc.addItemListener(this);
contentPane.add(jc);

// Create label
jl = new JLabel(new ImageIcon("france.gif"));
contentPane.add(jl);
}
public void itemStateChanged(ItemEvent ie) {
String s = (String)ie.getItem();
jl.setIcon(new ImageIcon(s + ".gif"));
}
}
```

Output from this applet is shown here:



TABBED PANES

A tabbed pane is a component that appears as a group of folders in a file cabinet. Each folder has a title. When a user selects a folder, its contents become visible. Only one of the folders may be selected at a time. Tabbed panes are commonly used for setting configuration options.

Tabbed panes are encapsulated by the **JTabbedPane** class, which extends JComponent. We will use its default constructor. Tabs are defined via the following method:

void addTab(String str, Component comp)

Here, str is the title for the tab, and comp is the component that should be added to the tab. Typically, a **JPanel** or a subclass of it is added.

The general procedure to use a tabbed pane in an applet is outlined here:

1. Create a JTabbedPane object.
2. Call addTab( ) to add a tab to the pane. (The arguments to this method define the title of the tab and the component it contains.)
3. Repeat step 2 for each tab.
4. Add the tabbed pane to the content pane of the applet.

Program:

The following example illustrates how to create a tabbed pane. The first tab is titled "Cities" and contains four buttons. Each button displays the name of a city. The second tab is titled "Colors" and contains three check boxes. Each check box displays the name of a color. The third tab is titled "Flavors" and contains one combo box. This enables the user to select one of three flavors.

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

class CitiesPanel extends JPanel {
public CitiesPanel() {
JButton b1 = new JButton("New York");
add(b1);
JButton b2 = new JButton("London");
add(b2);
JButton b3 = new JButton("Hong Kong");
add(b3);
JButton b4 = new JButton("Tokyo");
add(b4);
}
}

class ColorsPanel extends JPanel {
public ColorsPanel() {
JCheckBox cb1 = new JCheckBox("Red");
```

```
add(cb1);
JCheckBox cb2 = new JCheckBox("Green");
add(cb2);
JCheckBox cb3 = new JCheckBox("Blue");
add(cb3);
}
}

class FlavorsPanel extends JPanel {
public FlavorsPanel() {
JComboBox jcb = new JComboBox();
jcb.addItem("Vanilla");
jcb.addItem("Chocolate");
jcb.addItem("Strawberry");
add(jcb);
}
}
```

Output from this applet is shown in the following three illustrations:

SCROLL PANES

A scroll pane is a component that presents a rectangular area in which a component may be viewed. Horizontal and/or vertical scroll bars may be provided if necessary. Scroll panes are implemented in Swing by the JScrollPane class, which extends JComponent. Some of its constructors are shown here:

JScrollPane(Component comp)
JScrollPane(int vsb, int hsb)
JScrollPane(Component comp, int vsb, int hsb)

Here, comp is the component to be added to the scroll pane. vsb and hsb are int constants that define when vertical and horizontal scroll bars for this scroll pane are shown.

These constants are defined by the ScrollPaneConstants interface. Some examples of these constants are described as follows:

Constant	Description
HORIZONTAL_SCROLLBAR_ALWAYS	Always provide horizontal

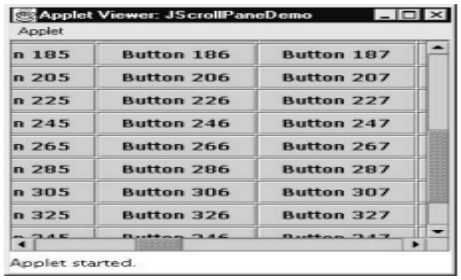
	scroll bar
HORIZONTAL_SCROLLBAR_AS_NEEDED	Provide horizontal scroll bar, if needed
VERTICAL_SCROLLBAR_ALWAYS	Always provide vertical scroll bar
VERTICAL_SCROLLBAR_AS_NEEDED	Provide vertical scroll bar, if needed

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

Program:
The following example illustrates a scroll pane. First, the content pane of the JApplet object is obtained and a border layout is assigned as its layout manager. Next, a JPanel object is created and four hundred buttons are added to it, arranged into twenty columns. The panel is then added to a scroll pane, and the scroll pane is added to the content pane. This causes vertical and horizontal scroll bars to appear. You can use the scroll bars to scroll the buttons into view.

```
import java.awt.*;
import javax.swing.*;
/*
<applet code="JScrollPaneDemo" width=300 height=250>
</applet>
*/
public class JScrollPaneDemo extends JApplet {
public void init() {
// Get content pane
Container contentPane = getContentPane();
contentPane.setLayout(new BorderLayout());
// Add 400 buttons to a panel
JPanel jp = new JPanel();
jp.setLayout(new GridLayout(20, 20));
int b = 0;
for(int i = 0; i < 20; i++) {
for(int j = 0; j < 20; j++) {
jp.add(new JButton("Button " + b));
++b;
}
}
// Add panel to a scroll pane
int v = ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED;
int h = ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED;
JScrollPane jsp = new JScrollPane(jp, v, h);
// Add scroll pane to the content pane
contentPane.add(jsp, BorderLayout.CENTER);
```

}
}
Output from this applet is shown here:



TREES

A tree is a component that presents a hierarchical view of data. A user has the ability to expand or collapse individual subtrees in this display. Trees are implemented in Swing by the JTree class, which extends JComponent. Some of its constructors are shown here:

JTree(Hashtable ht)
JTree(Object obj[ ])
JTree(TreeNode tn)
JTree(Vector v)

The first form creates a tree in which each element of the hash table ht is a child node.
Each element of the array obj is a child node in the second form.
The tree node tn is the root of the tree in the third form.
Finally, the last form uses the elements of vector v as child nodes.

Method	**Description**
void addTreeExpansionListener(TreeExpansionListener tel)	when a node is expanded
void removeTreeExpansionListener(TreeExpansionListener tel)	when a node is collapsed.
TreePath getPathForLocation(int x, int y)	translate a mouse click on a specific point of the tree to a tree path. Here, x and y are the coordinates at which the mouse is clicked.
TreePath	encapsulates information about a path to a particular node in a tree.
TreeNode	obtain information about a tree node.
DefaultMutableTreeNode(Object obj) obj is the object to be enclosed in this tree node.	The MutableTreeNode interface extends TreeNode. It declares methods that can insert and remove child nodes or change the parent node. The DefaultMutableTreeNode class implements the MutableTreeNode interface. It represents a node in a tree.

void add(MutableTreeNode child)	To create a hierarchy of tree nodes Here, child is a mutable tree node that is to be added as a child to the current node.
TreePath getPath( )	returns a TreePath object that describes the path to the changed node.
void treeCollapsed(TreeExpansionEvent tee)	called when a subtree is hidden
void treeExpanded(TreeExpansionEvent tee) tee is the tree expansion event.	method is called when a subtree becomes visible.
addMouseListener( )	To receive mouse events from the tree
doMouseClicked( )	processes mouse clicks.

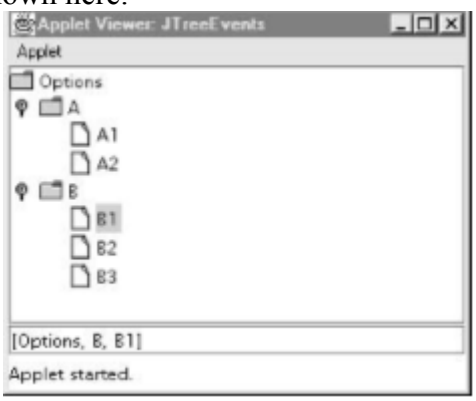
Here are the steps that you should follow to use a tree in an applet:
1. Create a JTree object.
2. Create a JScrollPane object. (The arguments to the constructor specify the tree and the policies for vertical and horizontal scroll bars.)
3. Add the tree to the scroll pane.
4. Add the scroll pane to the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

Program:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.tree.*;
/* <applet code="JTreeEvents" width=400 height=200>
</applet> */
public class JTreeEvents extends JApplet { JTree tree;
JTextField jtf;
public void init() {

// Get content pane
Container contentPane = getContentPane();

// Set layout manager
contentPane.setLayout(new BorderLayout());

// Create top node of tree
DefaultMutableTreeNode top = new DefaultMutableTreeNode("Options");

// Create subtree of "A"
DefaultMutableTreeNodea=newDefaultMutableTreeNode("A");
top.add(a);
DefaultMutableTreeNode a1 = new DefaultMutableTreeNode("A1");
a.add(a1);
DefaultMutableTreeNode a2 = new DefaultMutableTreeNode("A2");
a.add(a2);
```

```
// Create subtree of "B"
DefaultMutableTreeNodeb=newDefaultMutableTreeNode("B");
top.add(b);
DefaultMutableTreeNode b1 = new DefaultMutableTreeNode("B1");
b.add(b1);
DefaultMutableTreeNode b2 = new DefaultMutableTreeNode("B2");
b.add(b2);
DefaultMutableTreeNode b3 = new DefaultMutableTreeNode("B3");
b.add(b3);

// Create tree tree = new JTree(top);
// Add tree to a scroll pane
int v = ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED;
int h = ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED;
JScrollPane jsp = new JScrollPane(tree, v, h);

// Add scroll pane to the content pane
contentPane.add(jsp, BorderLayout.CENTER);

// Add text field to applet
jtf = new JTextField("", 20);
contentPane.add(jtf, BorderLayout.SOUTH);

// Anonymous inner class to handle mouse clicks
tree.addMouseListener(new MouseAdapter()
{
public void mouseClicked(MouseEvent me) { doMouseClicked(me); } });
}
void doMouseClicked(MouseEvent me)
{ TreePath tp = tree.getPathForLocation(me.getX(), me.getY());
if(tp != null)
    jtf.setText(tp.toString());
else jtf.setText(""); }
}
```

Output from this applet is shown here:



TABLES

A table is a component that displays rows and columns of data.

You can drag the cursor on column boundaries to resize columns. You can also drag a column to a new position.

Tables are implemented by the JTable class, which extends JComponent. One of its constructors is shown here:

JTable(Object data[ ][ ], Object colHeads[ ])

Here, data is a two-dimensional array of the information to be presented, and

colHeads is a one-dimensional array with the column headings.

Here are the steps for using a table in an applet:

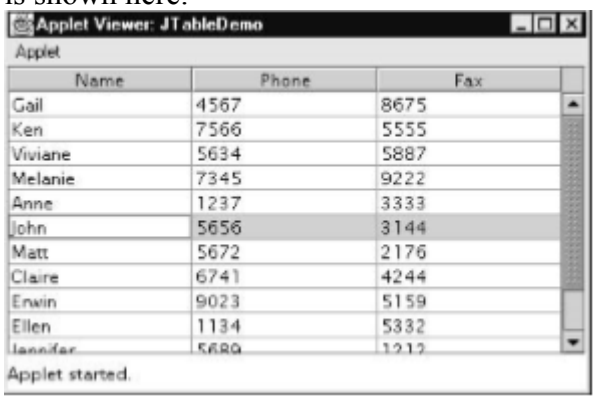
1. Create a JTable object.
2. Create a JScrollPane object. (The arguments to the constructor specify the table and the policies for vertical and horizontal scroll bars.)
3. Add the table to the scroll pane. 4. Add the scroll pane to the content pane of the applet.

Program:
The following example illustrates how to create and use a table. The content pane of the JApplet object is obtained and a border layout is assigned as its layout manager. A one dimensional array of strings is created for the column headings. This table has three columns. A two-dimensional array of strings is created for the table cells. You can see that each element in the array is an array of three strings. These arrays are passed to the JTable constructor. The table is added to a scroll pane and then the scroll pane is added to the content pane.

```
import java.awt.*;
import javax.swing.*;
/* <applet code="JTableDemo" width=400 height=200>
</applet> */
public class JTableDemo extends JApplet {
public void init() {

// Get content pane
Container contentPane = getContentPane();

// Set layout manager
contentPane.setLayout(new BorderLayout());

// Initialize column headings
final String[] colHeads = { "Name", "Phone", "Fax" };

// Initialize data
final Object[][] data = { { "Gail", "4567", "8675" },
{ "Ken", "7566", "5555" }, { "Viviane", "5634", "5887" },
{ "Melanie", "7345", "9222" }, { "Anne", "1237", "3333" },
```

```
{ "John", "5656", "3144" }, { "Matt", "5672", "2176" },
{ "Claire", "6741", "4244" }, { "Erwin", "9023", "5159" },
{ "Ellen", "1134", "5332" }, { "Jennifer", "5689", "1212" },
{ "Ed", "9030", "1313" }, { "Helen", "6751", "1415" } };
// Create the table
JTable table = new JTable(data, colHeads);

// Add table to a scroll pane
int v = ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED;
int h = ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED;
JScrollPane jsp = new JScrollPane(table, v, h);

// Add scroll pane to the content pane
contentPane.add(jsp, BorderLayout.CENTER);
}
}
```

Output from this applet is shown here:



EXPLORING THE SWINGS

Swing is a large system, and it has many features that you will want to explore on your own.

For example, Swing provides toolbars, tooltips, and progress bars. Also, Swing components can provide a pluggable look and feel, which means that it is easy to substitute another appearance and behavior for an element.

This can be done dynamically. You may even design your own look and feel.

Frankly, the Swing approach to GUI components might replace the AWT classes some time in the future, so familiarizing yourself with it now is a good idea.

Swing is just one part of the Java Foundation Classes (JFC). You may want to explore other JFC features.

The Accessibility API can be used to build programs that are usable by people with disabilities.

The Java 2-D API provides advanced capabilities for working with shapes, text, and images. The Drag-and-Drop API allows information to be exchanged between Java and non-Java programs.