UNIT-3 SWINGS

JLIST:

A component that displays a list of objects and allows the user to select one or more items. A separate model, ListModel, maintains the contents of the list. The JList component is similar to a set of checkboxes or radio buttons, except that the items are placed inside a single box and are selected by clicking on the items themselves, not on buttons. If you permit multiple selection for a list box, the user can select any combination of the items in the box.J

List provides a scrollable set of items from which one or more may be selected. JList can be populated from an Array or Vector. JList does not support scrolling directly—instead, the list must be associated with a scrollpane. The view port used by the scrollpane can also have a user-defined border. JList actions are handled using ListSelectionListener.

To construct this list component, you first start out with an array of strings, then pass the array to the JList constructor:

String[] words= { "quick", "brown", "hungry", "wild", ... };
JList wordList = new JList(words);
List boxes do not scroll automatically. To make a list box scroll, you must insert it into a scroll pane:

JScrollPane scrollPane = new JScrollPane(wordList);
By default, a user can select multiple items. This requires some knowledge of mouse technique: To add more items to a selection, press the CTRL key while clicking on each item. To select a contiguous range of items, click on the first one, then hold down the SHIFT key and click on the last one.

You can also restrict the user to a more limited selection mode with the setSelectionMode method:

```
wordList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION); \
  // select one item at a time
```

PROGRAM:

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

```
import javax.swing.*;

class ListExample extends JFrame
 {

            public ListExample()
            {



                        JPanel p = new JPanel(new BorderLayout());
                        String      listData[] ={
                                    "Item 1",
                                    "Item 2",
                                    "Item 3",
                                    "Item 4"
                        };

                        JList listbox = new JList( listData );
                        p.add( listbox, BorderLayout.CENTER );
           add(p);

            }

            public static void main( String args[] )
            {

                        ListExample mainFrame       = new ListExample();
          mainFrame.setSize(400,400);
                        mainFrame.setVisible(true);
            }
}
```

We cannot directly edit the collection of list values. Instead, you must access the model and then add or remove elements. For adding & removing values , a default list model is used. We should construct a DefaultListModel object, fill it with the initial values, and associate it with the list.

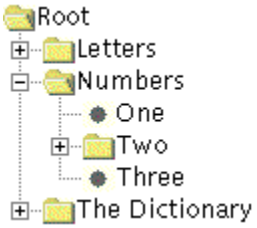
```
DefaultListModel model = new DefaultListModel();
model.addElement("quick");
model.addElement("brown");
. . .
JList list = new JList(model);
```

Now we can add or remove values from the model object. The model object then notifies the list of the changes, and the list repaints itself.

model.removeElement("quick");

JTREES:

With the JTree class, you can display hierarchical data. A JTree object does not actually contain your data; it simply provides a view of the data. Like any non-trivial Swing component, the tree gets data by querying its data model. Here is a picture of a tree:



As the preceding figure shows, JTree displays its data vertically. Each row displayed by the tree contains exactly one item of data, which is called a *node*. Every tree has a *root* node from which all nodes descend. By default, the tree displays the root node, but you can decree otherwise. A node can either have children or not. We refer to nodes that can have children — whether or not they currently *have* children — as *branch* nodes. Nodes that can not have children are *leaf* nodes.

you establish the parent/child relationships between the nodes. Start
with the root node, and use the add method to add the children:
DefaultMutableTreeNode root = new DefaultMutableTreeNode("World");
DefaultMutableTreeNode country = new DefaultMutableTreeNode("USA");
root.add(country);
DefaultMutableTreeNode state = new DefaultMutableTreeNode("California");
country.add(state);

Finally, construct a JTree with the tree model.
DefaultTreeModel treeModel = new DefaultTreeModel(root);
JTree tree = new JTree(treeModel);

PROGRAM:

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

class mytree extends JFrame
```

```
{

        public mytree()
        {


                JPanel p = new JPanel(new BorderLayout());
          DefaultMutableTreeNode root = new DefaultMutableTreeNode("World");
           DefaultMutableTreeNode country = new DefaultMutableTreeNode("USA");
           root.add(country);
           DefaultMutableTreeNode state = new DefaultMutableTreeNode("California");
           country.add(state);

DefaultTreeModel model = new DefaultTreeModel(root);
JTree tree = new JTree(model);
       p.add( tree, BorderLayout.CENTER );

        add(p);


        }

        public static void main( String args[] )
        {

                mytree mainFrame= new mytree();
                mainFrame.setSize(400,400);
                mainFrame.setVisible(true);
        }
}
```

To add or remove nodes following methods are used:
Model.insertNodeInto(newNode, selectedNode, selectedNode.getChildCount());
Model.removeNodeFromParent(selectedNode);

JTABLES:

The JTable component displays a two-dimensional grid of objects. Of course, tables are very common in user interfaces. The Swing team has put a lot of effort into the table control. Tables are inherently complex, but—perhaps more successfully than with other Swing classes. As with the tree control, a JTable does not store its own data, but obtains its data from a *table model.* The JTable class has a constructor that wraps a two-dimensional array of objects into a default model. the data of the table is stored as a two-dimensional array of Object values:

Object[][] cells = { {"abc", “21-a”, 24,987}, {“def”,”21-b”,25, 976}},
You supply the column names in a separate array of strings:

String[] columnNames = { "Name", "Address", "Age", "Phone no" };

Then, you construct a table from the cell and column name arrays. Finally, add scroll bars in the usual way, by wrapping the table in a JScrollPane.
JTable table = new JTable(cells, columnNames);
JScrollPane pane = new JScrollPane(table);

PROGRAM:

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

class mytable extends JFrame
 {

        public mytable()
        {



                JPanel p = new JPanel(new GridLayout(2,4));

          Object[][] cells = { {"abc", “21-a”, 24,987}, {“def”,”21-b”,25, 976}},
         String[] columnNames = { "Name", "Address", "Age", "Phone no" };
          JTable table = new JTable(cells, columnNames);
           JScrollPane pane = new JScrollPane(table);

         p.add( table );

            add(p);


        }

        public static void main( String args[] )
        {

                 Mytable  t2 = new mytable();
                t2.setSize(400,400);
                 t2.setVisible(true);
```

```
        }
}
```

STYLED TEXT COMPONENTS:

A styled text component can display editable text using more than one font. Some styled text components allow embedded images and even embedded components. Styled text components are powerful and multi-faceted components suitable for high-end needs, and offer more avenues for customization than the other text components.

Two Swing classes support styled text: JEditorPane and its subclass JTextPane. The JEditorPane class is the foundation for Swing's styled text components and provides a mechanism through which you can add support for custom text formats. If you want unstyled text, use a text area instead.

Because they are so powerful and flexible, styled text components typically require more initial programming to set up and use. JTextpane is used in the following way

```
JTextPane pane = new JTextPane();
SimpleAttributeSet set = new SimpleAttributeSet();
StyleConstants.setItalic(set, true);
StyleConstants.setBackground(set, Color.blue);
Document doc = pane.getStyledDocument();
doc.insertString(doc.getLength(), "MYJAVA ", set);
```

JEditorPane, is also used to display and edit text in HTML and RTF format.

We can use the setPage method to load a new document.

```
JEditorPane ed= new JEditorPane();

ed.setPage(url);
```

the parameter is either string or URL object.

We can also add hyperlinks to our editor pane. To listen to hyperlink clicks, add a Hyperlink listener. The HyperlinkListener interface has a single method, hyperlinkUpdate, that is called when the user moves over or clicks on a link. The method has a parameter of type HyperlinkEvent.

You need to call the getEventType method to find out what kind of event occurred. There are three possible return values:

```
HyperlinkEvent.EventType.ACTIVATED
HyperlinkEvent.EventType.ENTERED
HyperlinkEvent.EventType.EXITED
```

The first value indicates that the user clicked on the hyperlink. In that case, you typically want to open the new link.

```
editorPane.addHyperlinkListener(new
  HyperlinkListener()
  {
    public void hyperlinkUpdate(HyperlinkEvent event)
    {
      if (event.getEventType()
        == HyperlinkEvent.EventType.ACTIVATED)
      {
        try
        {
          editorPane.setPage(event.getURL());
        }
        catch (IOException e)
        {
          editorPane.setText("Exception: " + e);
        }
      }
    }
  });
```

The event handler simply gets the URL and updates the editor pane. When the component is editable, then hyperlinks are not active. Therefore , we write

```
Ed.setEditable(false);
```

PROGRESS INDICATORS:

There are three classes for indicating the progress of a slow activity. A JProgressBar is a Swing component that indicates progress. A ProgressMonitor is a dialog box that contains a progress bar. A ProgressMonitorInputStream displays a progress monitor dialog box while the stream is read.

Progress Bars

A progress bar is a simple component just a rectangle that is partially filled with color to indicate the progress of an operation. By default, progress is indicated by a string "n%". You construct a progress bar much as you construct a slider, by supplying the minimum and maximum value and an optional orientation:

```
progressBar = new JProgressBar(0, 1000);
progressBar = new JProgressBar(SwingConstants.VERTICAL, 0, 1000);
progressBar.setStringPainted(true);
```

progressBar.setValue() method is used to set or get the current value of the progress bar. The value is constrained by the minimum and maximum values.You can also set the minimum and maximum with the setMinimum and setMaximum methods.

The SimulatedActivity class implements a thread that increments a value current 10 times per second. When it reaches a target value, the thread finishes. To terminate the thread before it has reached its target, you interrupt it.

```
class SimulatedActivity implements Runnable
{
  . . .
  public void run()
  {
    try
    {
      while (current < target && !interrupted())
      {
        Thread.sleep(100);
        current++;
      }
    }
    catch (InterruptedException e)
    {
    }
  }

  int current;
  int target;
}
```

Progress Monitors

A progress bar is a simple component that can be placed inside a window. In contrast, a ProgressMonitor is a complete dialog box that contains a progress bar. The dialog box contains a Cancel button. If you click it, the monitor dialog box is closed. In addition, your program can query whether the user has canceled the dialog box and terminate the monitored action.

You construct a progress monitor by supplying the following:

- The parent component over which the dialog box should pop up;
- An object (which should be a string, icon, or component) that is displayed on the dialog box;
- An optional note to display below the object;
- The minimum and maximum values.

However, the progress monitor cannot measure progress or cancel an activity by itself. You still need to periodically set the progress value by calling the setProgress method. (This is the equivalent of the setValue method of the JProgressBar class.) As you update the progress value,

you should also call the isCanceled method to see if the program user has clicked the Cancel button.

When the monitored activity has concluded, call the close method to dismiss the dialog box. You can reuse the same dialog box by calling start again.

Here's the statement that creates the progress monitor:

```
progressMonitor = new ProgressMonitor(ProgressMonitorDemo.this,
                                      "Running a Long Task",
                                      "", 0, task.getLengthOfTask());
```

This code uses ProgressMonitor's only constructor to create the monitor and initialize several arguments. Methods used in ProgressMonitor class:

- ProgressMonitor(Component parent, Object message, String note, int min, int max)

  constructs a progress monitor dialog box.

Parameters:		
	parent	The parent component over which this dialog box pops up
	message	The message object to display in the dialog box
	note	The optional string to display under the message. If this value is null, then no space is set aside for the note, and a later call to setNote has no effect
	min, max	The minimum and maximum values of the progress bar

- void setNote(String note)

  changes the note text.

- void setProgress(int value)

  sets the progress bar value to the given value.

- void close()

  closes this dialog box.

- boolean isCanceled()

  returns true if the user canceled this dialog box.

Monitoring the Progress of Input Streams

The Swing package contains a useful stream filter, ProgressMonitorInputStream, that automatically pops up a dialog box that monitors how much of the stream has been read. For example, suppose you read text from a file. You start out with a FileInputStream:

FileInputStream in = new FileInputStream(f);
Normally, you would convert in to an InputStreamReader:

InputStreamReader reader = new InputStreamReader(in);
However, to monitor the stream, first turn the file input stream into a stream with a progress monitor:

ProgressMonitorInputStream progressIn = new ProgressMonitorInputStream(parent, caption, in);

- constructs an input stream filter with an associated progress monitor dialog box.

Parameters:	parent	The parent component over which this dialog box pops up
	message	The message object to display in the dialog box
	in	The input stream that is being monitored

COMPONENT ORGANIZERS:

These are those components that help organize other components. These include the split pane, a mechanism for splitting an area into multiple parts whose boundaries can be adjusted, the tabbed pane, which uses tab dividers to allow a user to flip through multiple panels, and the desktop pane, which can be used to implement applications that display multiple internal frames.

Split Panes

Split panes split a component into two parts, with an adjustable boundary in between.The outer pane is split vertically, with a text area on the bottom and another split pane on the top. That pane is split horizontally, with a list on the left and a label containing an image on the right.

You construct a split pane by specifying the orientation, one of JSplitPane.HORIZONTAL_SPLIT or JSplitPane.VERTICAL_SPLIT, followed by the two components. For example,

JSplitPane innerPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, planetList, planetImage);
If you like, you can add "one-touch expand" icons to the splitter bar. To add this capability, call

innerPane.setOneTouchExpandable(true);

The "continuous layout" feature continuously repaints the contents of both components as the user adjusts the splitter. That looks classier, but it can be slow. You turn on that feature with the call

innerPane.setContinuousLayout(true);

Tabbed Panes

Tabbed panes are a familiar user interface device to break up a complex dialog box into subsets of related options. You can also use tabs to let a user flip through a set of documents or images

To create a tabbed pane, you first construct a JTabbedPane object, then you add tabs to it.

```
JTabbedPane tabbedPane = new JTabbedPane();
tabbedPane.addTab(title, icon, component);
```

The last parameter of the addTab method has type Component. To add multiple components into the same tab, you first pack them up in a container, such as a JPanel.

The icon is optional; for example, the addTab method does not require an icon:

```
tabbedPane.addTab(title, component);
```

You can also add a tab in the middle of the tab collection with the insertTab method:

```
tabbedPane.insertTab(title, icon, component, tooltip, index);
```

To remove a tab from the tab collection, use

```
tabPane.removeTabAt(index);
```

.