The Java Series

GUI Building with Swing

What is JFC?

The Java Foundation Classes (JFC) are a comprehensive set of GUI components and services to simplify the development and deployment of commercial-quality desktop applications.

It s an effort to provide a complete CLASS LIBRARY to build modern GUIs out-of-the-box.

With JFC you ll get most of what you need when developing any kind of user interface.

What s in JFC

Swing: A complete set of graphical components

Pluggable look & feel.

Java 2D: To render, manipulate and transform, complex 2D images and text.

Drag & Drop programmability.

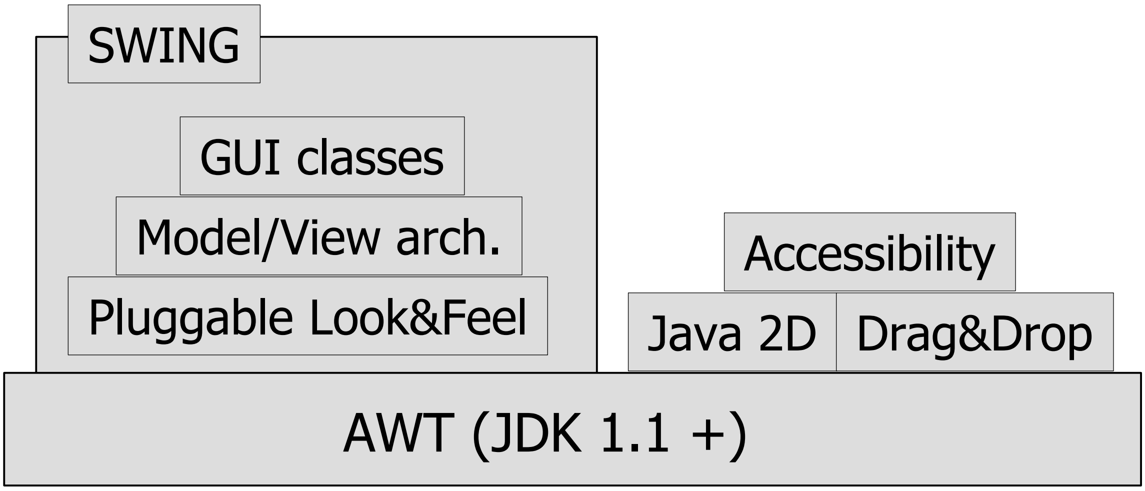
Accessibility.

SWING IS FOR GUI BUILDING

What about AWT?

- Provides the basic functionality for GUI building.
- Provides a minimum set of components.
- Complex GUI require complex applications.
- Components difficult to customize extend.
- JFC extends AWT
- JFC provides more components and more functionality.
- AWT provides:
  - The Event Model.
  - The Component/Container conceptualization.

AWT & JFC


SWING

GUI classes

Model/View arch.

Pluggable Look&Feel

Accessibility

Java 2D

Drag&Drop

AWT (JDK 1.1 +)

What do you get with Swing

AWT leaves the final implementation to the interpreter/OS. Although functionally the same, components look different in different platforms.

Swing controls look & feel. Your GUI looks the same everywhere.

Some AWT components have limited capabilities and customizability. With Swing out-of-the-box:

- ToolTips, Keyboard Navigation, Properties, etc.

Swing provides more components:

- Tables, Trees, ToolBars, etc.

This Talk

We are going to see how to use and customize Swing components.

Two ways to use Swing (two parts of the lecture):

- A high level set of graphical components ready to use. Easy to use in your programs.
- An architecture upon which to build and customize components to any degree. Very flexible.

The Event Model is the one from AWT so we are going to use it in the exact same way as in AWT.

Remember AWT (1)

- GUI Building includes two tasks:
  - Building the Interface.
  - Handling Events.
- The Hierarchy is based on the **Component** class.
- There are two types of Components:
  - Containers (Windows, Frames, Panels,)
  - Everything else (Buttons, Lists,)
- A Container:
  - contains a set of other Components.
  - has a Layout Manager to place the Components within.
- A **Frame** is a Container which is a top level window.

```
import java.awt.*;

public class MyApplication {

  public static void main (String[] args) {

     Choice l = new Choice();
     l.addItem("Item 1");
     l.addItem("Item 2");
     l.addItem("Item 3");


     TextArea ta = new TextArea(5,20);
     TextField tf = new TextField();
     Label lb = new Label("This is an example");

     Frame f = new Frame("Some sample");
     f.setLayout(new GridLayout(2,2));
     f.setLocation(100,100);
     f.add (lb, new Dimension(1,1));
     f.add (l,  new Dimension(1,2));
     f.add (tf, new Dimension(2,1));
     f.add (ta, new Dimension(2,2));

     f.pack();
     f.show();
 }
}
```

A few components. Each component has its own methods, constructors, etc.. specific to the function they perform.

See API documentation for details on each different component

Remember AWT (2)

The Event Model

GUI Building is **Event Driven**.

Listeners are objects which are notified whenever a certain event happens.

Different Listeners can listen to different events (mouse move, button clicks, list selection, windows closing, etc)

Each component has a list of Listeners for each type of events may happen.

If you want to do something when a certain event happens:

Create your listener class/object

Register it with the component you are interested in

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

public class MyActionListener
       implements ActionListener {

   public void actionPerformed(ActionEvent e) {
     System.out.println("A button has been pressed");
   }
}
```

```
import java.awt.*;

public class MyApplication {

  public static void main (String[] args) {

     Button b = new Button ("Press me");
     MyActionListener alistener = new MyActionListener();
     b.addActionListener(alistener);

     Frame f = new Frame("Some sample");
     f.setLayout(new FlowLayout());
     f.setLocation(100,100);
     f.add (b);

     f.pack();
     f.show();
 }
}
```

PART 1:
Ready-to-use Swing

Every AWT component is reimplemented in Swing:

- Just add a J to its class name.
- AWT Button is improved in Swing s JButton.
- AWT List is improved in Swing s JList.

Swing s components have more methods

```
JButton b1 = new JButton (new
   Image( picture.gif ))
b1.setToolTipText( Click here to open );
b1.setMinimunSize(30,10);
```

The Swing Hierarchy

(PARTIAL)

AWT

- Object
  - Component
    - Button
    - Checkbox
    - Choice
    - Label
    - List
    - Scrollbar
    - Container
      - Panel
      - Window
        - Frame

SWING

- Object
  - Component
  - Container
    - JComponent
      - JBComboBox
      - JLabel
      - JList
      - JProgressBar
      - JScrollPane
      - JSplitPane
      - JTabbedPane
      - JTable
      - JTree
      - JAbstractButton
        - JButton
        - JToggleButton
      - JTextComponent
    - Window
      - Frame
        - JWindow
        - JDialog
        - JFrame

The Main Differences

JComponent from Container. Every Swing object is a Container. Flexibility.

JComponent provides general functionality:

- ToolTips, Keyboard Navigation,

Two kinds of JComponent:

- Top-Level Containers (JFrame, JWindow)
- Lightweight Components (the rest, including JPanel)

Top-Level Containers:

- The Contain ONE JPanel.
- Can t add JComponents directly.
- JComponents are added to the JPanel.
- The Layout Manager is associated with the JPanel

Using Swing

If you have jdk 1.2: it comes with it you don t have to do anything.

Otherwise:

download swing for jdk 1.1 from:

`http://java.sun.com/products/jfc/download.html`

untar the file swing.jar

point your classpath to it

Using java at CERN:

jdk -listversions

setenv JDKVERSION 1.2

javac MyApplication.java

java MyApplication.java

Scenario 1

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

public class MyApplication {

  public static void main (String args[]) {

        // Create the frame
        JFrame frame = new JFrame ("My Application");

        // Create some components
        JButton b1 = new JButton("A Button");
        b1.setToolTipText("This is the left button");

        JButton b2 = new JButton(new ImageIcon("middle.gif"));
        b2.setToolTipText("This is the middle button");

        JLabel label  = new JLabel(new ImageIcon("alb.gif"));

        JTextField text = new JTextField(20);

        Vector items = new Vector();
        for (int i=1; i<20; i++) {
                items.addElement("This is item "+i);
        }
        JList list = new JList(items);
        list.setToolTipText("Select one item");
        JScrollPane listPane = new JScrollPane(list);

        //Lay out the content pane.
        JPanel contentPane = new JPanel();
        contentPane.setLayout(new FlowLayout());
        contentPane.setPreferredSize(new Dimension(300, 300));
        contentPane.add(b1);
        contentPane.add(b2);
        contentPane.add(label);
        contentPane.add(text);
        contentPane.add(listPane);
        frame.setContentPane(contentPane);
        frame.pack();
        frame.show();
  }
}
```

Create some components

Create a JPanel

Set a Layout Manager

Add to the JPanel

Associate the JPanel to the Frame

A Common GUI Technique

In Scenario 1, if we want to open another frame identical we have build it again.

But remember this is OO!!

We can create our own class defining the frame as we want it and then

INSTANTIATE AS MANY TIMES AS WE WANT

Extend the already defined GUI classes.

Define constructors, redefine methods.

Implement more functionality:

- For instance, you don t need to define extra listeners objects if the same frame is defined as a listener.

Scenario 1.2

We extend JPanel

```
public class FirstSample extends JPanel {

  public FirstSample() {
        super();
        // Create some components
        JButton b1 = new JButton("A Button");

        .. .. ..

        add(label);
        add(text);
        add(listPane);
   }
   public static void main (String args[]) {
        JFrame frame = new JFrame("First Sample");
        frame.setContentPane(new FirstSample());
        frame.pack();
        frame.show();

        JFrame frame2 = new JFrame("Another First Sample");
        frame2.setContentPane(new FirstSample());
        frame2.pack();
        frame2.show();
   }
}
```

Use the constructor to add components

Always call the parent s constructor

Now we create as many instances as we want

Interesting JComponents

JButton, JList, JLabel, JTextXXX, JComboBox, JRadioButton, JCheckBox extend functionality existing in AWT Components.

JProgressBar, JSlider, JTable, JToolBar, JTree provide new components.

JInternalFrame, JScrollPane, JSplitPane, JTabbedPane provide new ways to combine components.

See the doc:

wwwinfo.cern.ch/support/java/docs/api

The Java Tutorial:

wwwinfo.cern.ch/support/java/docs

Scenario 2: JToolBar

Same technique as before

```
import java.util.*;
import javax.swing.*;
import java.awt.*;
public class ToolBarSample extends JPanel {
  public ToolBarSample () {
        super();
        // Create a toolbar
        JToolBar mybar = new JToolBar();
        mybar.add(new JButton(new ImageIcon("open.gif")));
        mybar.add(new JButton(new ImageIcon("save.gif")));

        JButton cut = new JButton(new ImageIcon("cut.gif"));
        cut.setToolTipText("Cut Selection");
        JButton copy = new JButton(new ImageIcon("copy.gif"));
        copy.setToolTipText("Copy Selection");
        mybar.add(cut);
        mybar.add(copy);

        // Create some components
        .. .. ..

        add("North", mybar);
        add("South", label);
        add("Center", text);
   }

  public static void main (String args[]) {
        // Create the frame and the content
        JFrame frame = new JFrame ("My Application");
        ToolBarSample tb = new ToolBarSample();

        frame.setContentPane(tb);
        frame.pack();
        frame.show();
 }

}
```

Create a JToolBar object

Add buttons to it

And ready to go

JScrollPane

A generic container to put anything you d like to be scrollable:

Images, Data, Text,

You can put any component in an JScrollPane and Swing will take care of everything.

The Component inside the JScrollPane must be Scrollable (interface).

Most of Swing components are Scrollable

See next scenario with and without the ScrollPane

Scenario 3: Scrolling

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

public class ScrollSample extends JPanel {

  public ScrollSample() {
        JTextArea text = new JTextArea();
        JScrollPane textPane = new JScrollPane(text);

        //Lay out the pane.
        setLayout(new BorderLayout());
        setPreferredSize(new Dimension(300, 300));
        add("Center", textPane);
   }

  public static void main (String args[]) {

        // Create the frame
        JFrame frame = new JFrame ("My Application");
        ScrollSample scroll = new ScrollSample();
        frame.setContentPane(scroll);
        frame.pack();
        frame.show();
   }
}
```

Create the component

Put into a JScrollPane

Instantiate it and insert into a top level frame

JTable

A Table is a grid of cells.

The JTable class provides the basic functionality.

The data in the table is separated from the JTable object itself.

To create a table:

- Instantiate a JTable object
- Create a class to hold the data
- Instantiante a data object
- Associate the Table object with the data object

The Data class must be derived from **AbstractTableModel**

Then you can use generic and specific Listeners as in any other component

}

Scenario 4: Tables

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

class MyTableModel extends AbstractTableModel {
          public int getColumnCount() { return 10; }
          public int getRowCount() { return 10;}
          public Object getValueAt(int row, int col)
                    { return new Integer(row*col); }
}

public class TableSample extends JPanel {

  public TableSample() {
      super();

      MyTableModel dataModel = new MyTableModel();
      JTable table = new JTable(dataModel);
      table.setPreferredScrollableViewportSize
            (new Dimension(300, 100));
      JScrollPane scrollpane = new JScrollPane(table);

        //Lay out the content pane.
        setLayout(new FlowLayout());
        add(scrollpane);
   }
   public static void main (String args[]) {

        // Create the frame
        JFrame frame = new JFrame ("Table Sample");

        frame.setContentPane(new TableSample());
        frame.pack();
        frame.show();
   }
}
```

Define the data class

See AbstractTableModel doc

Create a data instance

Association with data can be done in the constructor

JTree

The principles are the same as JTable

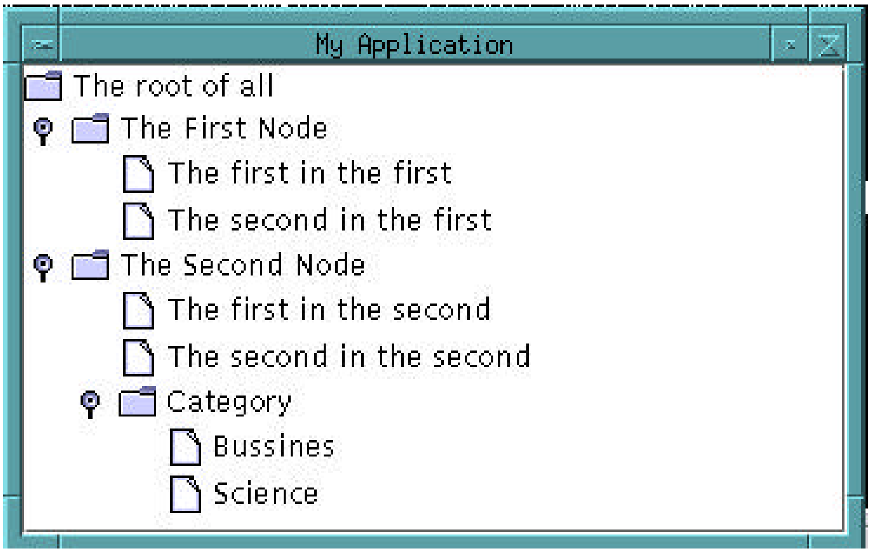
Create your data + Associate with tree.

A tree is made of nodes.

Tree data is created through the DefaultMutableTreeNode class.

We need to:

- Define one root node.
- Add nodes to each other to build the tree structure.



Scenario 5: Trees

```
public class TreeSample extends JPanel {
  public TreeSample() {
        super();
        // Create the tree structure
        DefaultMutableTreeNode top =
             new DefaultMutableTreeNode("The root of all");
        DefaultMutableTreeNode primo =
           new  DefaultMutableTreeNode ("The First Node");
        top.add(primo);
        primo.add(new DefaultMutableTreeNode ("The firs .. ..
        primo.add(new DefaultMutableTreeNode ("The second ..

        DefaultMutableTreeNode second =
           new  DefaultMutableTreeNode ("The Second Node");
        top.add(second);
        second.add(new DefaultMutableTreeNode ("The first ..
        second.add(new Defau                              nd ..

        DefaultMutableTreeNo
           new  DefaultMutableTreeNode ("Category");
        second.add(category);
        category.add(new DefaultMutableTreeNode("Bussines"));
        category.add(new DefaultMutableTreeNode ("Science"));

        JTree tree = new JTree(top);
        JScrollPane scrollpane = new JScrollPane(tree);

        //Lay out the content pane.
        setLayout(new BorderLayout());
        setPreferredSize(new Dimension(350,300));
        add("Center", scrollpane);
   }
} .. .. ..
```

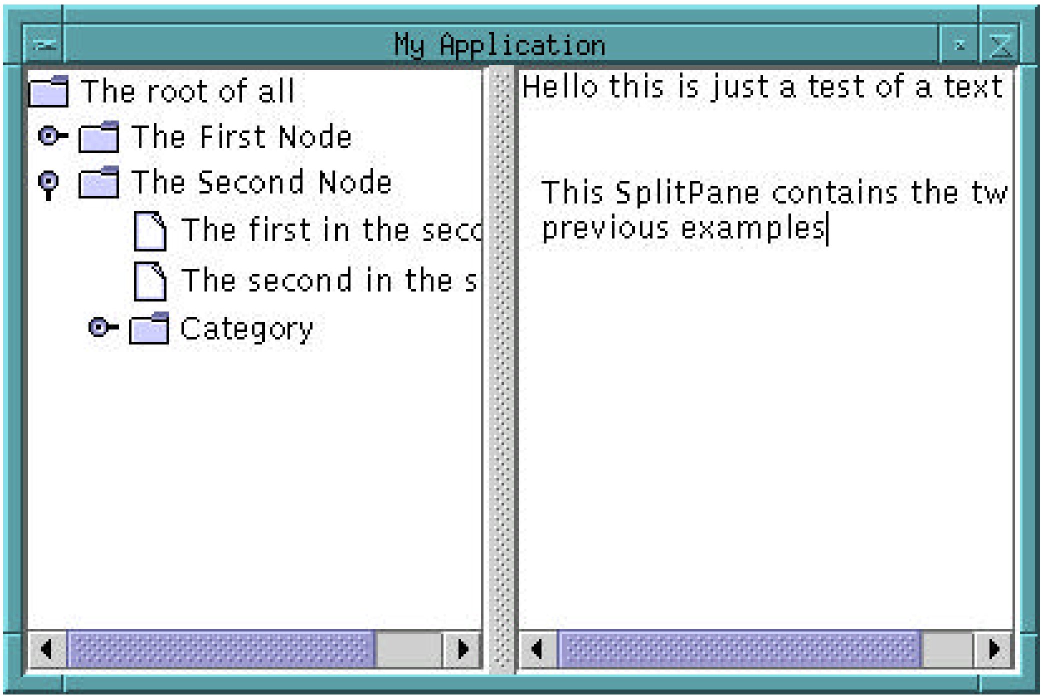
Create the root node

Create the structure by adding nodes to each other

Now create the Tree object with root

JSplitPanel

To share the same physical space simultaneously between two containers



Hello this is just a test of a text

This SplitPane contains the tw
previous examples

To use it:

- Create the two components
- Create a JSplitPanel
- Insert them into the JSplitPanel

Scenario 6: Split Panels

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

public class SplitSample extends JPanel {

  public SplitSample() {
        super();

        JSplitPane pane =
           new JSplitPane(JSplitPane.HORIZONTAL_SPLIT);

        TreeSample tree = new TreeSample();
        ScrollSample text = new ScrollSample();

        pane.setLeftComponent(tree);
        pane.setRightComponent(text);
        pane.setDividerLocation(150);
        pane.setDividerSize(10);

        //Lay out the content pane.
        setPreferredSize(new Dimension(350, 300));

        setLayout(new GridLayout(1,1));
        add(pane, new Dimension(1,1));

   }
}

.. .. ..
```

Create the Split Panel

Create the components to add

We are using the previous examples

This is OO!!!

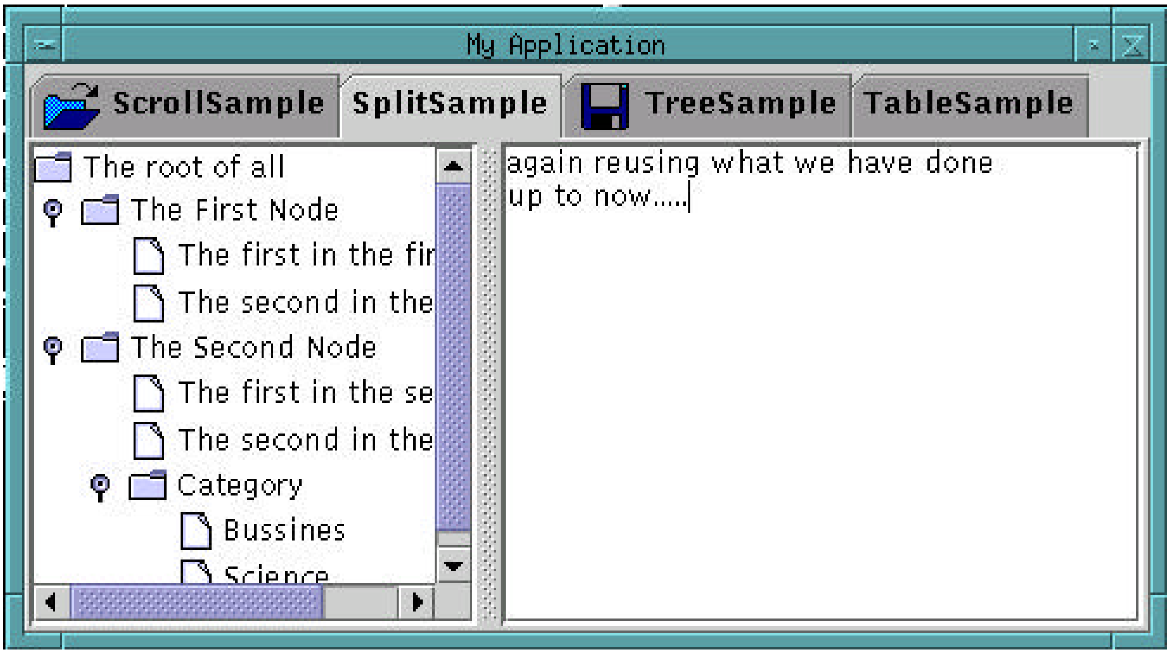
Insert them and configure the Split Panel

JTabPanel

To share the same physical space between any number of components

The principles like the SplitPanel.

Create objects + add them.



again reusing what we have done
up to now.....

Scenario 7: Tab Panels

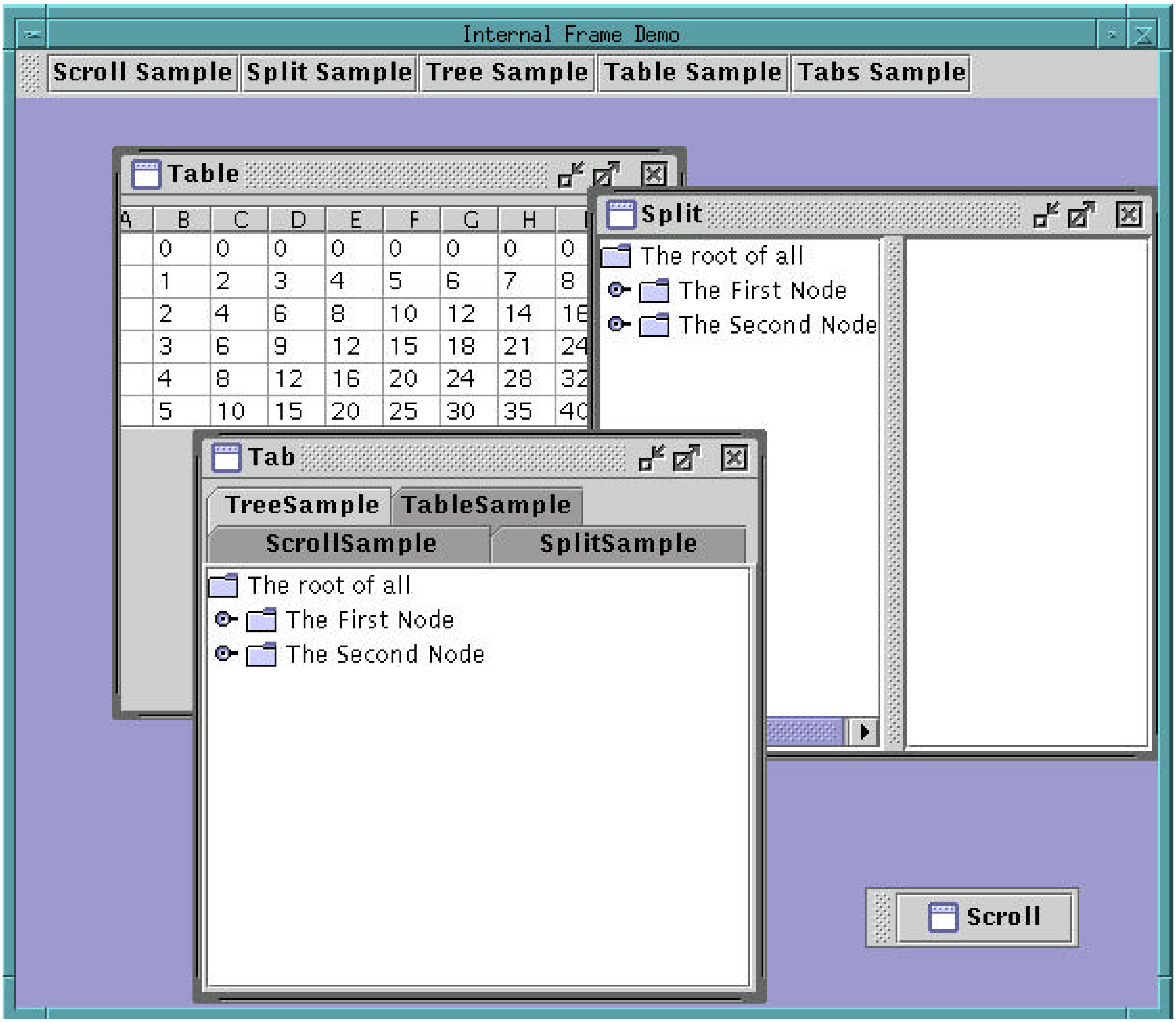
```
import java.util.*;
import javax.swing.*;
import javax.swing.tree.*;
import java.awt.*;

public class TabbedSample extends JPanel {

  public TabbedSample() {
      super();

      JTabbedPane tpane = new JTabbedPane();

      tpane.addTab("ScrollSample", new ImageIcon("open.gif"),
                   new ScrollSample(),
                    "The Previous Scroll Sample");
      tpane.addTab("SplitSample", null,
                   new SplitSample(), null);
      tpane.addTab("TreeSample", new ImageIcon("save.gif"),
                   new TreeSample(), "The Tree Sample");
      tpane.addTab("TableSample", null,
                   new TableSample(), "The Table Sample");

        //Add the tabbed pane to this panel.
        setLayout(new GridLayout(1, 1));
        add(tpane);
   }

.. .. ..
}
```

Create the TabbedPanel

Create and add other components.
We are using our previous examples.
Can specify infos when adding

Internal Frame

A desktop within a window:



A	B	C	D	E	F	G	H	I
	0	0	0	0	0	0	0	0
	1	2	3	4	5	6	7	8
	2	4	6	8	10	12	14	16
	3	6	9	12	15	18	21	24
	4	8	12	16	20	24	28	32
	5	10	15	20	25	30	35	40

Scenario 8: Desktops

Extend JFrame

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

public class InternalFrameSample extends JFrame implements ActionListener{

  JDesktopPane desktop;
  JButton b1, b2, b3, b4, b5;
  int offsetx=50; int offsety=50;
  public InternalFrameSample() {
      super("Internal Frame Demo");

      desktop = new JDesktopPane();

      JToolBar toolbar = new JToolBar();
      b1 = new JButton ("Scroll Sample");
      b2 = new JButton ("Split Sample");
      b3 = new JButton ("Tree Sample");
      b4 = new JButton ("Table Sample");
      b5 = new JButton ("Tabs Sample");

      toolbar.add(b1);  b1.addActionListener(this);
      toolbar.add(b2);  b2.addActionListener(this);
      toolbar.add(b3);  b3.addActionListener(this);
      toolbar.add(b4);  b4.addActionListener(this);
      toolbar.add(b5);  b5.addActionListener(this);

      //Add the desktop pane to this panel.
      desktop.setLayout(new BorderLayout());
      desktop.add("North", toolbar);
      setContentPane(desktop);
   }

   public void actionPerformed (ActionEvent e) {
      JInternalFrame frame;
      if (e.getSource() == b1) {
         createInternalFrame("Scroll", new ScrollSample());
      } else if (e.getSource() == b2) {
         createInternalFrame("Split", new SplitSample());
      } else if (e.getSource() == b3) {
         createInternalFrame("Tree", new TreeSample());
      } else if (e.getSource() == b4) {
         createInternalFrame("Table", new TableSample());
      } else if (e.getSource() == b5) {
         createInternalFrame("Tab", new TabbedSample());
      }
   }
```

Create a toolbar with some buttons

```
 public void createInternalFrame(String title, Container what) {
          JInternalFrame fr = new JInternalFrame(title, true, true, true, true)
;
          fr.setContentPane(what);
          fr.setSize(300,300);
          fr.setLocation(offsetx,offsety);
          offsetx = offsetx+20;
          offsety = offsety+20;

          desktop.add(fr);
          try {
           fr.setSelected(true);
          } catch (java.beans.PropertyVetoException e2) {}
   }


   public static void main (String args[]) {
       InternalFrameSample frame = new InternalFrameSample();
       frame.pack();
       frame.show();
   }
```

Just put a panel in an InternalFrame and add it to the desktop

InternalFrameSample objects are also ActionListeners (for the toolbar buttons in this case)

PART 2: Customizable Swing

We are just going to see the principles behind the Swing architecture.

Everything is Swing is open .

It is designed in a modular way, separating the functionalities of every component.

When customizing and Swing component you choose which part you want to modify and this does not affect the rest.

This is OO!!!

Customizable Swing

Swing is a huge library.

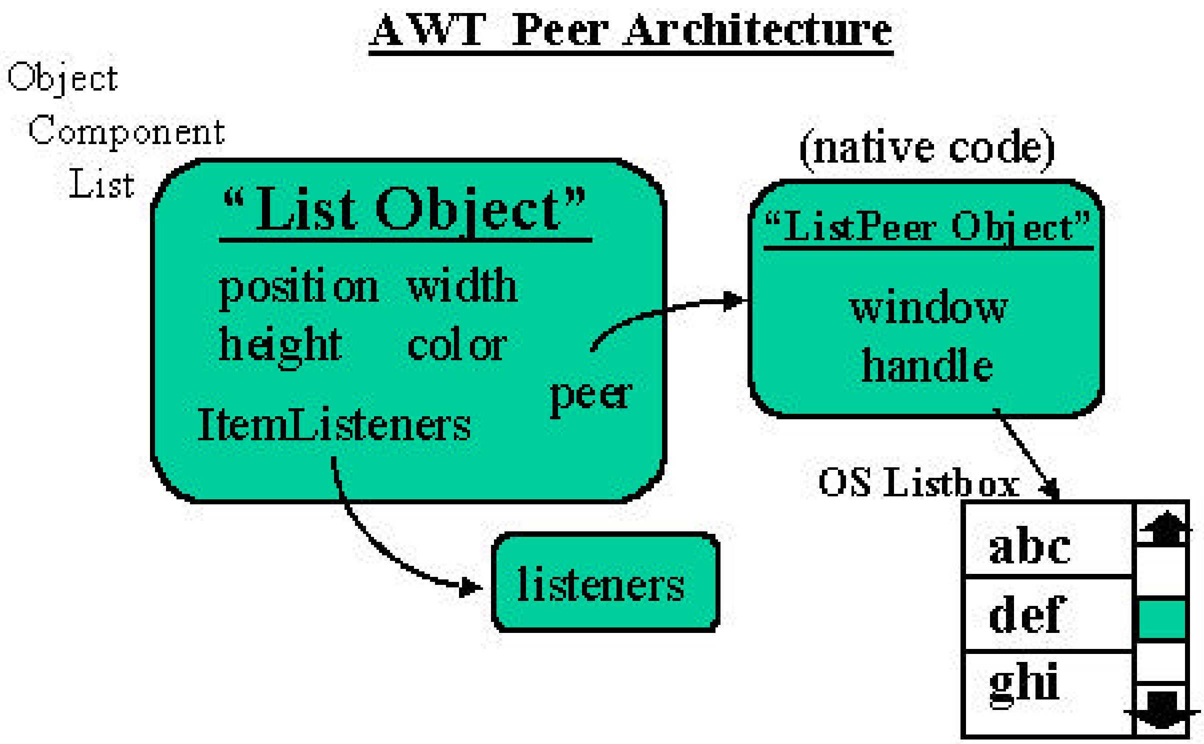
Behind every component there is a set of objects working to achieve its mission.

This implies having a big amount of classes, interfaces, etc interrelated.

See the doc:

wwwinfo/support/java/docs/api

AWT Peer Architecture



For every component AWT provides two objects:

A Logical Object: Containing the high level properties and behaviors (data + painting)

A Peer Object: Containing the low level interface with the platform. Final drawing and behavior is delegated to the platform.

Swing Architecture

The implementation of every component is separated into:

- The Model of the component: containing the data, status...
- The View of the component: containing the graphics and event handling.

This way we separate the data from the UI.

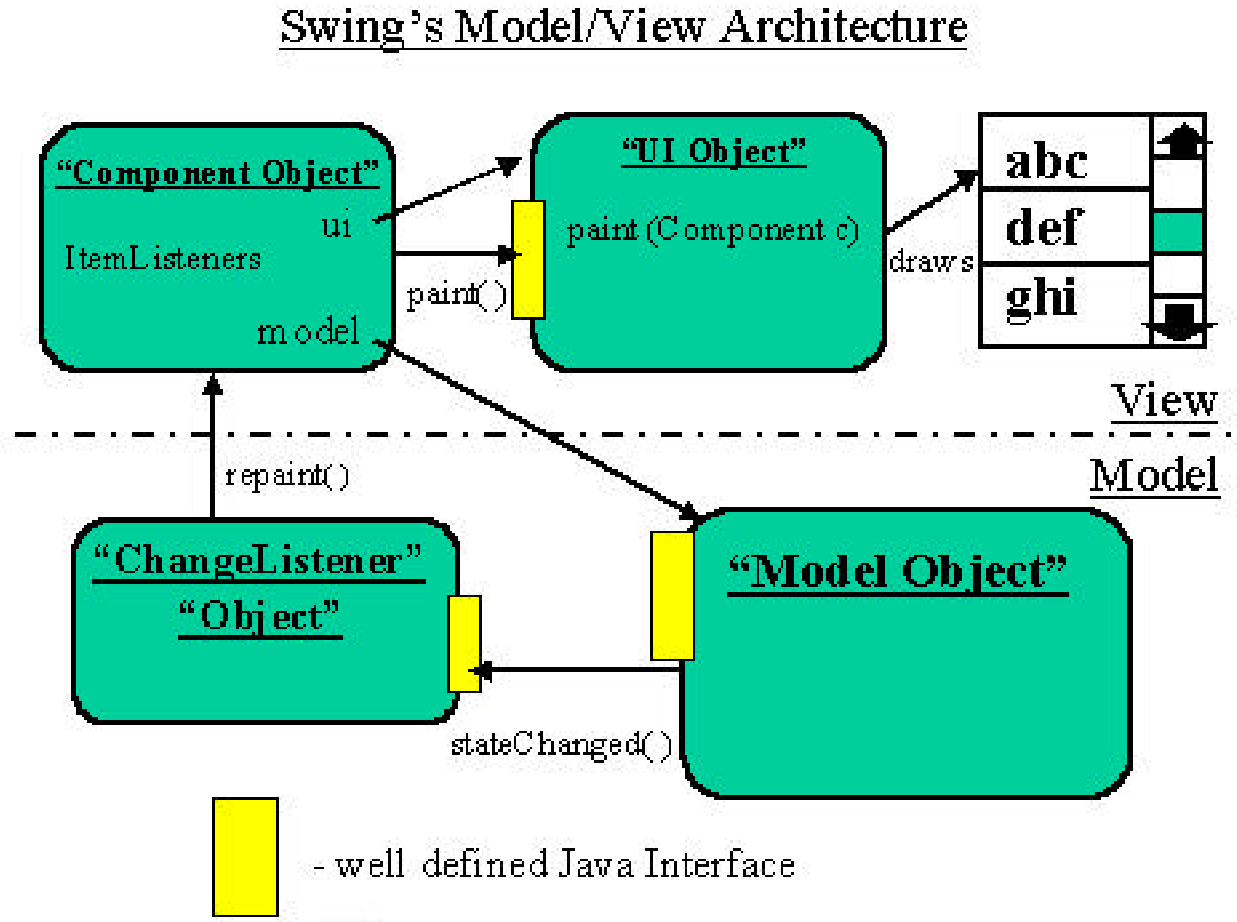
For every component 4 objects:

- 2 implementing the model
- 2 implementing the view

When we want to change something we change the relevant object.

In AWT everything is in one object.

Swing Architecture



Model Object: data + status. Notifies ChangeList.

ChangeListener: what to do when model changes

Notifies component about changes.

Component Object: graphical properties.

Gets data from model. Asks UI to repaint.

UI Object: Final physical drawing (not the OS)

Swing Architecture

Remember the table:

- We created our own TableModel containing the data.
- We created a component object and associated to it.

There are default implementations of these objects for each component.

Depending on what we change we customize:

- The model object: Changes the data
- The changeListener object: Changes when the UI is updated after a change on the model.
- The Component object: Changes physical properties (position, size, contained components .)
- The UI object: Changes the look and feel.

Swing Architecture

See the doc:

- JComponent.getUIClassID()
- JTable (introduction)
- JTable constructors
- JTable.createDefaultDataModel()
- JTable.createDefaultRenderers()
- JTable.getCellRenderer()
- JTable.getCellEditor()
- JTable.getUI()
- swing.plaf.*

Observe:

- Complete independence of functionalities.
- Complete customizability.
- Complete modularity.

Pluggable Look and Feel

There are three sets of UI Objects already defined:

- Metal Look & Feel
- Motif Look & Feel
- Windows Look & Feel

These are the UI objects which are part of the View part of every component.

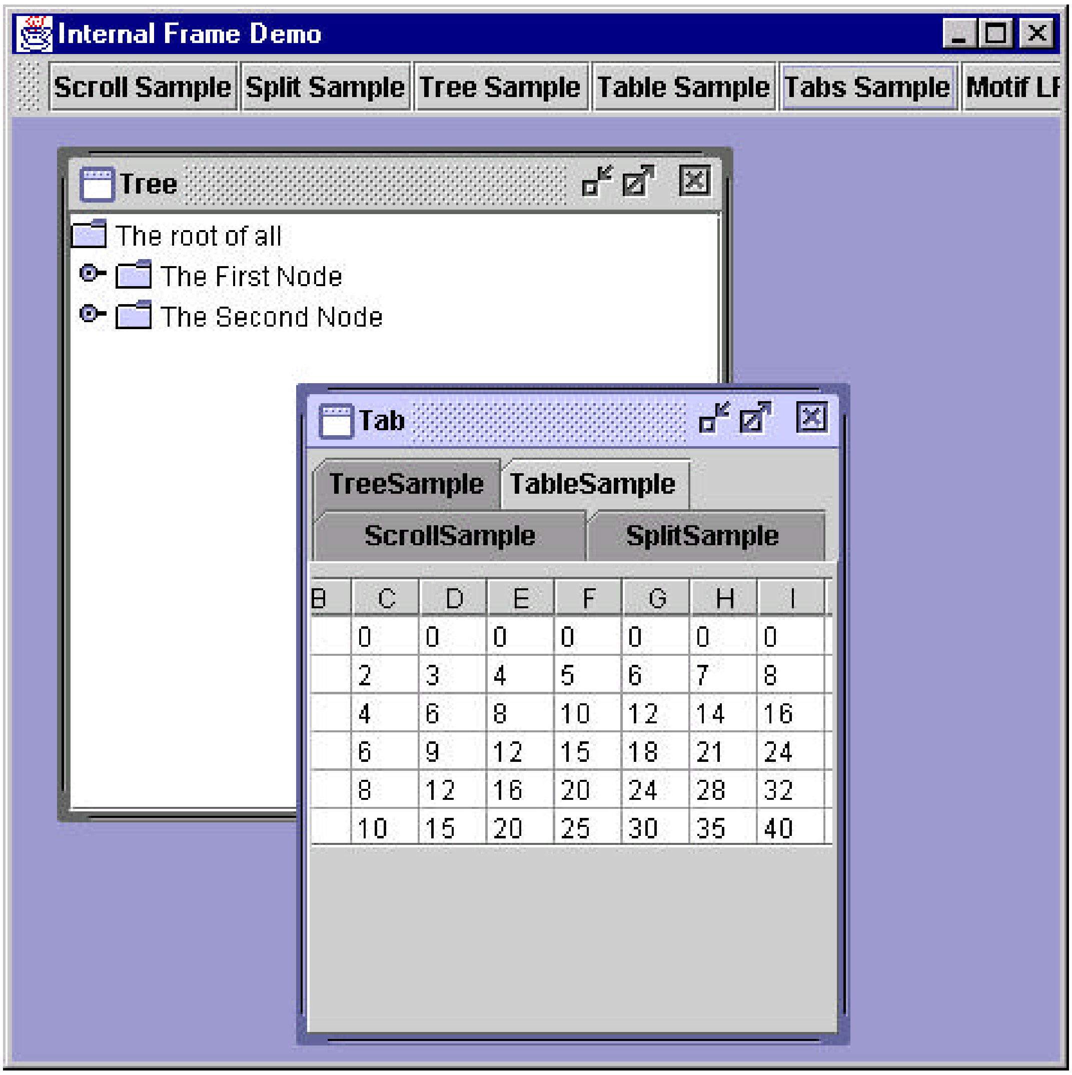
Since they are just objects we can:

- Change the UI Objects associated with any component
- Build our own UI Objects to create our own Look & Feel.

The most important:

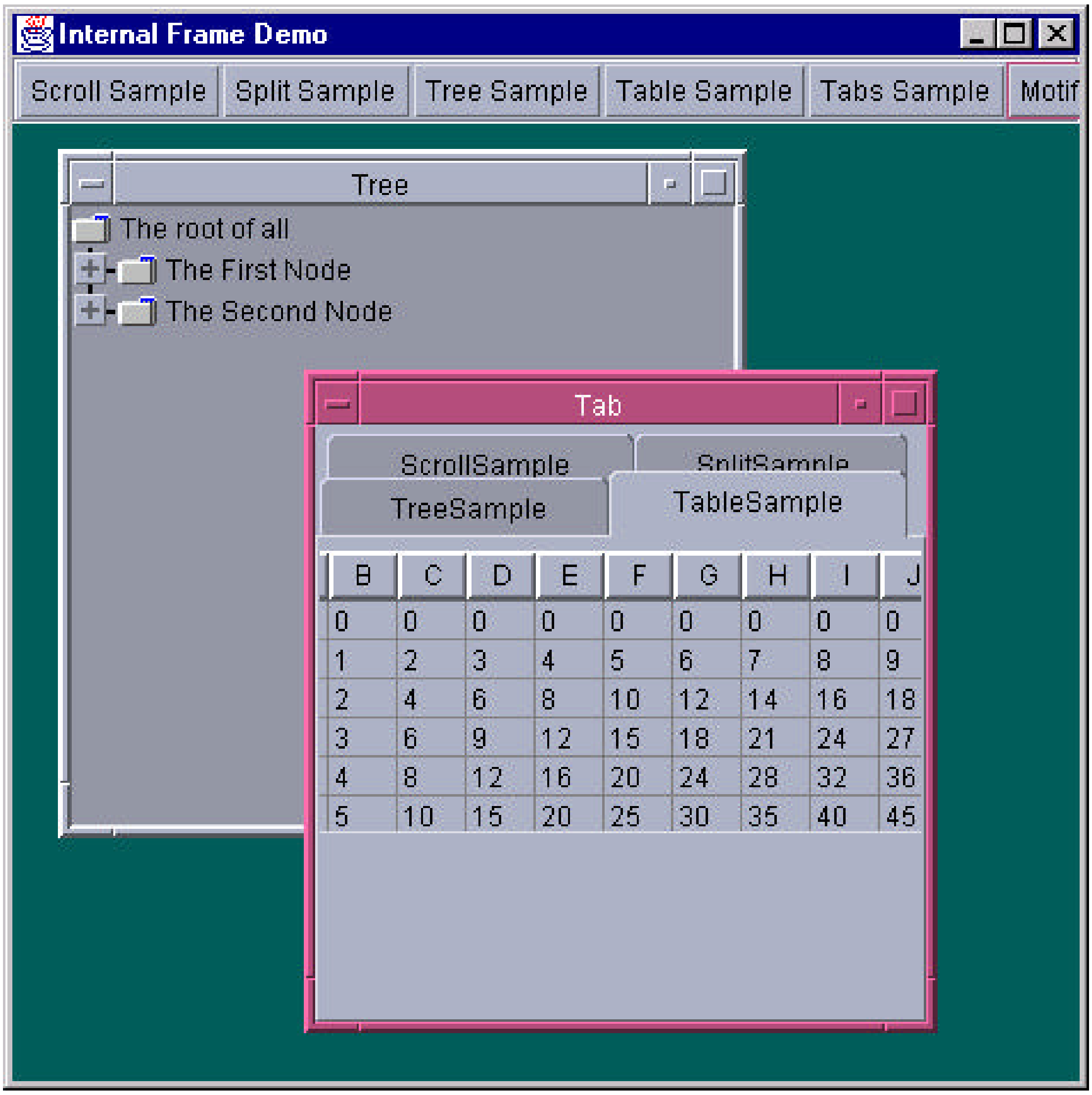
- With this architecture we have complete independence.
- We can design our application and THEN decide on the LF

Scenario 9: Look & Feel



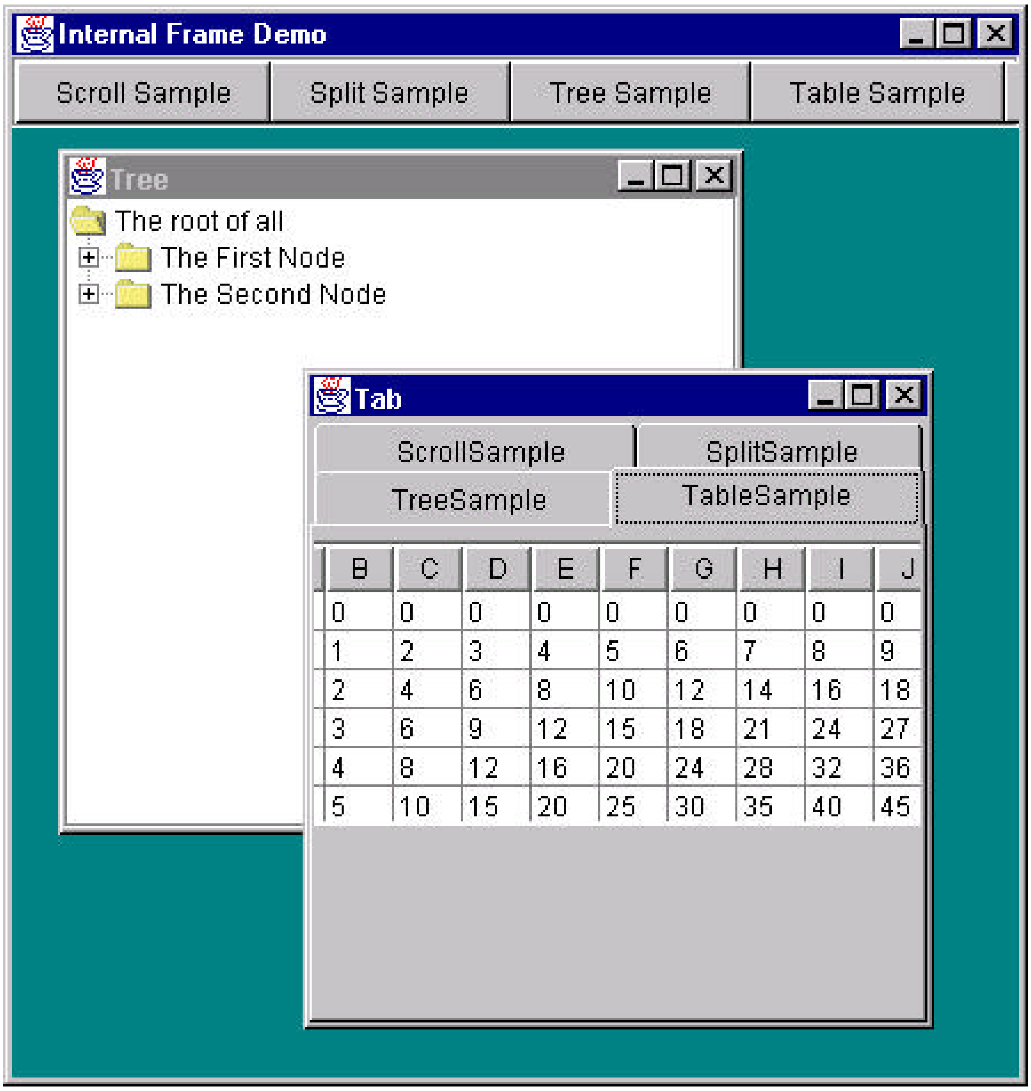
B	C	D	E	F	G	H	I
	0	0	0	0	0	0	0
	2	3	4	5	6	7	8
	4	6	8	10	12	14	16
	6	9	12	15	18	21	24
	8	12	16	20	24	28	32
	10	15	20	25	30	35	40

Scenario 9: L&F



B	C	D	E	F	G	H	I	J
0	0	0	0	0	0	0	0	0
1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45

Scenario 9: L&F



B	C	D	E	F	G	H	I	J
0	0	0	0	0	0	0	0	0
1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45

Scenario 9

```
public class InternalFrameDemo extends JFrame {

.. .. ..

public void actionPerformed (ActionEvent e) {

        } else if (e.getSource() == b6) {
            changeLF(1);
        } else if (e.getSource() ==
            changeLF(2);
        } else if (e.getSource() == b8) {
            changeLF(3);
        }
   }

   public void changeLF(int what) {
     String lf ="";
     if (what==1) { lf = "com.sun.java.swing.plaf.motif.MotifLookAndFeel";}
     else if (what==2) { lf = "javax.swing.plaf.metal.MetalLookAndFeel";}
     else if (what==3) {
           lf = "com.sun.java.swing.plaf.windows.WindowsLookAndFeel";
     }
     try {
        UIManager.setLookAndFeel(lf);
        SwingUtilities.updateComponentTreeUI(this);
     } catch(Exception e) { e.printStackTrace(); }
  }
```

When clicking a button

Change the look and feel of this frame

Summary

- Swing extends AWT
- New components:
  - Lightweight components
    - JButton, JTree,
  - Containers
    - Split, Tabbed, Desktop
- Customizability through separation:
  - Model part (data + status)
  - View part (behavior + UI)

Swing

Free.

Is now part of JDK 1.2.

When established will be homogenous.

In JDK 1.1.x is a separate library.

Modularization implies overhead (slow performance for heavy UIs)

Recommendation:

- Swing out-of-the-box is enough for most of the needs.
- Learn model/view of the particular component you want to customize.