**GUI SWING - Part 3**

**Effective use of Nested Panels :**

In this section, we will discuss how to nest panels effectively to get a desired layout of GUI components. It is possible, but very difficult, to place all GUI components on a single JPanel or other types of containers. A better approach is to use multiple panels, placing panels inside other panels.

We will introduce other useful Swing components in this section. Please keep in mind that we limit the discussion to the most basic use of these components. They are actually far more capable than what we present here. However, the materials presented in this section should be enough to let you use them in most common situations and should serve as a good starting point from which you can explore more advanced uses of these components on your own.

**Now go over the code for the complete program Ch14NestedPanels1!**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**JCheckBox**

Graphical user interface, text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Now go over the code for the complete program!**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**JComboBox**

The JComboBox class presents a combo box. This class is similar to the JRadioButton class in that it also allows the user to select one item from a list of possible choices. The

Text

Description automatically generated

Also, we can call the getSelectedIndex method to retrieve the position of the selected item. The first item in the list is at position 0. Here’s the Ch14JComboBoxSample class:

Graphical user interface, application

Description automatically generated

**Now go over the code for the complete program Ch14JComboBoxSample.java**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**JList :**

Table

Description automatically generated with medium confidence

Graphical user interface, application

Description automatically generated

Text

Description automatically generated

**Now go over the code for the complete program Ch14JListSample.java**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**JSplitPane  - From the Oracle Java API :**

JSplitPane is used to divide two (and only two) Components.

The two Components in a split pane can be aligned left to right using JSplitPane.HORIZONTAL\_SPLIT, or top to bottom using JSplitPane.VERTICAL\_SPLIT.

The preferred way to change the size of the Components is to invoke setDividerLocation where location is either the new x or y position, depending on the orientation of the JSplitPane.

To resize the Components to their preferred sizes invoke resetToPreferredSizes.

When the user is resizing the Components the minimum size of the Components is used to determine the maximum/minimum position the Components can be set to. If the minimum size of the two components is greater than the size of the split pane the divider will not allow you to resize it. To alter the minimum size of a JComponent, see [JComponent.setMinimumSize(java.awt.Dimension)](https://docs.oracle.com/en/java/javase/17/docs/api/java.desktop/javax/swing/JComponent.html" \l "setMinimumSize(java.awt.Dimension)).

When the user resizes the split pane the new space is distributed between the two components based on the resizeWeight property. A value of 0, the default, indicates the right/bottom component gets all the space, where as a value of 1 indicates the left/top component gets all the space.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Text

Description automatically generated**

**Graphical user interface, text, application, email

Description automatically generated**

**Graphical user interface, text, application, email

Description automatically generated**

**Graphical user interface, text, application, email

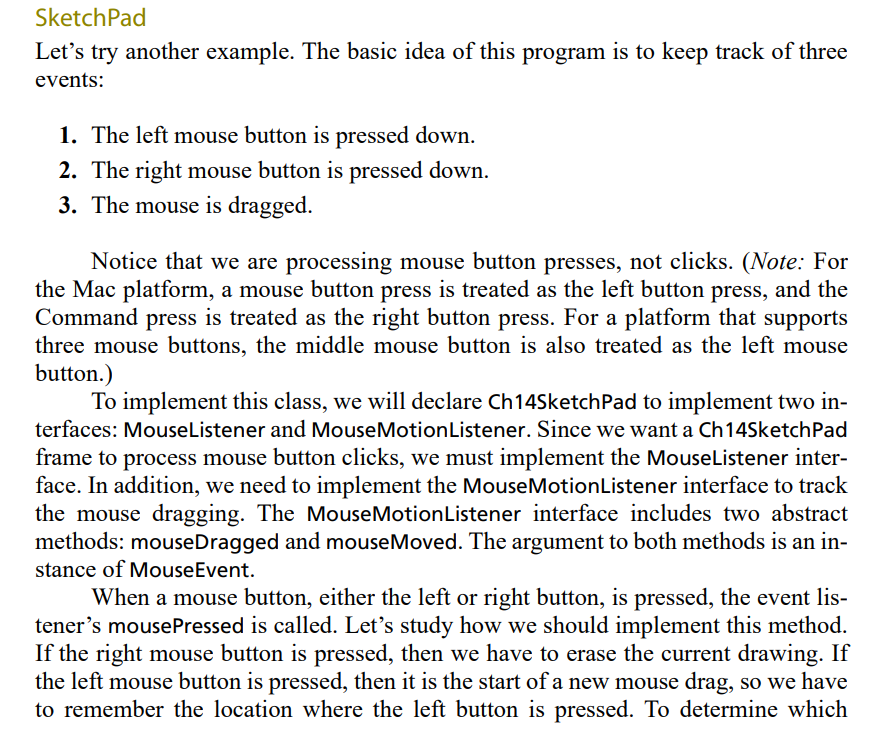
Description automatically generated**

**Text

Description automatically generated**

**Now go over the code for the complete program**

**Then do :**

****

**Graphical user interface, text, application

Description automatically generated**

**Graphical user interface, text

Description automatically generated**

**Graphical user interface, text, application, email

Description automatically generated**

The complete program for SketchPad is in the other file of code.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

If time allows do : A Complete CardLayout program is below for extra credit, and ...

Also, the First link below, is REALLY GOOD!!

<https://docs.oracle.com/javase/tutorial/uiswing/examples/layout/index.html#CardLayoutDemo>

and

<https://docs.oracle.com/javase/tutorial/uiswing/layout/card.html>

---------------------------------------------------------------------------------------------------------

**The following code does run. It is Extra Credit**.

////package layout;

/\*

 \* CardLayoutDemo.java

 \*

 \*/

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

public class CardLayoutDemo implements ItemListener {

    JPanel cards; //a panel that uses CardLayout

    final static String BUTTONPANEL = "Card with JButtons";

    final static String TEXTPANEL = "Card with JTextField";

    public void addComponentToPane(Container pane) {

        //Put the JComboBox in a JPanel to get a nicer look.

        JPanel comboBoxPane = new JPanel(); //use FlowLayout

        String comboBoxItems[] = { BUTTONPANEL, TEXTPANEL };

        JComboBox cb = new JComboBox(comboBoxItems);

        cb.setEditable(false);

        cb.addItemListener(this);

        comboBoxPane.add(cb);

        //Create the "cards".

        JPanel card1 = new JPanel();

        card1.add(new JButton("Button 1"));

        card1.add(new JButton("Button 2"));

        card1.add(new JButton("Button 3"));

        JPanel card2 = new JPanel();

        card2.add(new JTextField("TextField", 20));

        //Create the panel that contains the "cards".

        cards = new JPanel(new CardLayout());

        cards.add(card1, BUTTONPANEL);

        cards.add(card2, TEXTPANEL);

        pane.add(comboBoxPane, BorderLayout.PAGE\_START);

        pane.add(cards, BorderLayout.CENTER);

    }

    public void itemStateChanged(ItemEvent evt) {

        CardLayout cl = (CardLayout)(cards.getLayout());

        cl.show(cards, (String)evt.getItem());

    }

    /\*\*

     \* Create the GUI and show it.  For thread safety,

     \* this method should be invoked from the

     \* event dispatch thread.

     \*/

    private static void createAndShowGUI() {

        //Create and set up the window.

        JFrame frame = new JFrame("CardLayoutDemo");

        frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        //Create and set up the content pane.

        CardLayoutDemo demo = new CardLayoutDemo();

        demo.addComponentToPane(frame.getContentPane());

        //Display the window.

        frame.pack();

        frame.setVisible(true);

    }

    public static void main(String[] args) {

        /\* Use an appropriate Look and Feel \*/

        try {

            //UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");

            UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");

        } catch (UnsupportedLookAndFeelException ex) {

            ex.printStackTrace();

        } catch (IllegalAccessException ex) {

            ex.printStackTrace();

        } catch (InstantiationException ex) {

            ex.printStackTrace();

        } catch (ClassNotFoundException ex) {

            ex.printStackTrace();

        }

        /\* Turn off metal's use of bold fonts \*/

        UIManager.put("swing.boldMetal", Boolean.FALSE);

        //Schedule a job for the event dispatch thread:

        //creating and showing this application's GUI.

        javax.swing.SwingUtilities.invokeLater(new Runnable() {

            public void run() {

                createAndShowGUI();

            }

        });

    }

}