

# Swing

- JFC (Java Foundation Classes)
- Componentele Swing
- Asemănări și deosebiri cu AWT
- Folosirea ferestrelor
- Ferestre interne
- Arhitectura modelului Swing
- Folosirea modelelor
- Tratarea evenimentelor
- Folosirea componentelor
- Containere
- Desenarea
- Look and Feel

## JFC (Java Foundation Classes)

- Componente Swing
- Look-and-Feel
- Accessibility API
- Java 2D API
- Drag-and-Drop
- Internaționalizare

# Swing API

<code>javax.accessibility</code>	<code>javax.swing.plaf</code>
<code>javax.swing.text.html</code>	<code>javax.swing</code>
<code>javax.swing.plaf.basic</code>	<code>javax.swing.text.parser</code>
<code>javax.swing.border</code>	<code>javax.swing.plaf.metal</code>
<code>javax.swing.text.rtf</code>	<code>javax.swing.colorchooser</code>
<code>javax.swing.plaf.multi</code>	<code>javax.swing.tree</code>
<code>javax.swing.event</code>	<code>javax.swing.table</code>
<code>javax.swing.undo</code>	<code>javax.swing.filechooser</code>
<code>javax.swing.text</code>	

Cel mai important: **javax.swing**

# Componentele Swing

- **Componente atomice**

JLabel, JButton, JCheckBox, JRadioButton, JToggleButton, JScrollBar, JSlider, JProgressBar, JSeparator

- **Componente complexe**

JTable, JTree, JComboBox, JSpinner, JList, JFileChooser, JColorChooser, JOptionPane

- **Componente pentru editare de text**

JTextField, JFormattedTextField, JPasswordField, JTextArea, JEditorPane, JTextPane

- **Meniuri**

JMenuBar, JMenu, JPopupMenu, JMenuItem, JCheckboxMenuItem, JRadioButtonMenuItem

- **Containere intermediare**

JPanel, JScrollPane, JSplitPane, JTabbedPane, JDesktopPane, JToolBar

- **Containere de nivel înalt**

JFrame, JDialog, JWindow, JInternalFrame, JApplet

# Asemănări și deosebiri cu AWT

Tehnologia Swing **extinde** AWT.

```
import javax.swing.*;
import java.awt.*; //Font, Color, ...
import java.awt.event.*;
```

## Convenția ”J”

java.awt.Button - javax.swing.JButton  
java.awt.Label - javax.swing.JLabel,  
etc.

Noi gestionari de poziționare: BorderLayout  
și SpringLayout

## Folosirea HTML

```
JButton simplu = new JButton("Text simplu");
JButton html = new JButton(
    "<html><u>Text</u> <i>formatat</i></html>");
```

---

Listing 1: Aplicația rescrisă folosind Swing

---

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

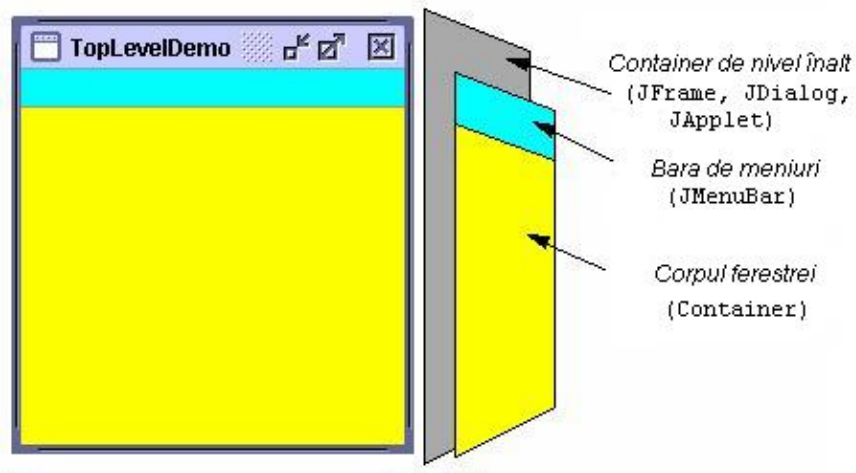
public class ExempluSwing extends JFrame implements
    ActionListener {
    public ExempluSwing(String titlu) {
        super(titlu);
        // Metoda setLayout nu se aplica direct ferestrei
        getContentPane().setLayout(new FlowLayout());

        // Componentele au denumiri ce incep cu litera J
        // Textul poate fi si in format HTML
        getContentPane().add(new JLabel(
            "<html><u>Hello</u> <i>Swing</i></html>"));
        JButton b = new JButton("Close");
        b.addActionListener(this);

        // Metoda add nu se aplica direct ferestrei
        getContentPane().add(b);
        pack();
        show();
    }
    public void actionPerformed(ActionEvent e) {
        // Tratarea evenimentelor se face ca in AWT
        System.exit(0);
    }
    public static void main(String args[]) {
        new ExempluSwing("Hello");
    }
}
```

---

# Folosirea ferestrelor



```
Frame f = new Frame();  
f.setLayout(new FlowLayout());  
f.add(new Button("OK"));
```

```
JFrame jf = new JFrame();  
jf.getContentPane().setLayout(new FlowLayout());  
jf.getContentPane().add(new JButton("OK"));
```

```
jf.setDefaultCloseOperation(  
    WindowConstants.HIDE_ON_CLOSE);  
    WindowConstants.DO_NOTHING_ON_CLOSE  
    JFrame.EXIT_ON_CLOSE
```

## Ferestre interne

Aplicațiile pot fi împărțite în:

- **SDI** (Single Document Interface)
- **MDI** (Multiple Document Interface)

Clase:

**JInternalFrame**

**DesktopPane** - container



---

Listing 2: Folosirea ferestrelor interne

---

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

class FereastraPrincipala extends JFrame {
    public FereastraPrincipala(String titlu) {
        super(titlu);
        setSize(300, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        FereastraInterna fin1 = new FereastraInterna();
        fin1.setVisible(true);
        FereastraInterna fin2 = new FereastraInterna();
        fin2.setVisible(true);

        JDesktopPane desktop = new JDesktopPane();
        desktop.add(fin1);
        desktop.add(fin2);
        setContentPane(desktop);

        fin2.moveToFront();
    }
}

class FereastraInterna extends JInternalFrame {
    static int n = 0; //nr. de ferestre interne
    static final int x = 30, y = 30;

    public FereastraInterna() {
        super("Document #" + (++n),
            true, //resizable
            true, //closable
            true, //maximizable
            true); //iconifiable
        setLocation(x*n, y*n);
        setSize(new Dimension(200, 100));
    }
}

public class TestInternalFrame {
    public static void main(String args[]) {
        new FereastraPrincipala("Test ferestre interne").show();
    }
}
```

---

## Clasa JComponent

JComponent este superclasa tuturor componentelor Swing, mai puțin JFrame, JDialog, JApplet.

JComponent extinde clasa Container.

Facilități:

- ToolTips - setToolTip
- Chenare - setBorder
- Suport pentru plasare și dimensionare - setPreferredSize, ...
- Controlul opacității - setOpaque
- Asocierea de acțiuni tastelor
- Double-Buffering

---

Listing 3: Facilități oferite de clasa JComponent

---

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

class Fereastră extends JFrame {
    public Fereastră(String titlu) {
        super(titlu);
        getContentPane().setLayout(new FlowLayout());
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Folosirea chenarelor
        Border lowered, raised;
        TitledBorder title;

        lowered = BorderFactory.createLoweredBevelBorder();
        raised = BorderFactory.createRaisedBevelBorder();
        title = BorderFactory.createTitledBorder("Borders");

        final JPanel panel = new JPanel();
        panel.setPreferredSize(new Dimension(400,200));
        panel.setBackground(Color.blue);
        panel.setBorder(title);
        getContentPane().add(panel);

        JLabel label1 = new JLabel("Lowered");
        label1.setBorder(lowered);
        panel.add(label1);

        JLabel label2 = new JLabel("Raised");
        label2.setBorder(raised);
        panel.add(label2);

        // Controlul opacității
        JButton btn1 = new JButton("Opaque");
        btn1.setOpaque(true); // implicit
        panel.add(btn1);

        JButton btn2 = new JButton("Transparent");
        btn2.setOpaque(false);
        panel.add(btn2);

        // ToolTips
```

```

label1.setToolTipText("Eticheta coborata");
label2.setToolTipText("Eticheta ridicata");
btn1.setToolTipText("Buton opac");
// Textul poate fi HTML
btn2.setToolTipText("<html><b>Apasati <font color=red>F2
    </font> " +
        "cand butonul are <u>focusul</u>");

// Asocierea unor actiuni (KeyBindings)
/* Apasarea tastei F2 cand focusul este pe butonul al
doilea
va determina schimbarea culorii panelului */
btn2.getInputMap().put(KeyStroke.getKeyStroke("F2"),
    "schimbaCuloare");
btn2.getActionMap().put("schimbaCuloare", new
    AbstractAction() {
        private Color color = Color.red;
        public void actionPerformed(ActionEvent e) {
            panel.setBackground(color);
            color = (color == Color.red ? Color.blue : Color.red)
                ;
        }
    });

pack();
}
}

public class TestJComponent {
    public static void main(String args[]) {
        new Fereastră("Facilitati JComponent").show();
    }
}

```

---

## Arhitectura modelului Swing

### MVC (model-view-controller).

- *Modelul* - datele aplicației.
- *Prezentarea* - reprezentare vizuală
- *Controlul* - transformarea acțiunilor în evenimente

### Arhitectură cu model separabil

Model + (Prezentare, Control)

## Folosirea modelelor

Model	Componentă
ButtonModel	JButton, JToggleButton, JCheckBox, JRadioButton, JMenu, JMenuItem, JCheckBoxMenuItem, JRadioButtomMenuItem
JComboBox	ComboBoxModel
BoundedRangeModel	JProgressBar, JScrollBar, JSlider
JTabbedPane	SingleSelectionModel
ListModel	JList
ListSelectionModel	JList
JTable	TableModel
JTable	TableColumnModel
JTree	TreeModel
JTree	TreeSelectionModel
Document	JEditorPane, JTextPane, JTextArea, JTextField, JPasswordField

Crearea unui model = implementarea interfeței

JList - ListModel

DefaultListModel, AbstractListModel

---

Listing 4: Folosirea mai multor modele pentru o componenta

---

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

class Fereastra extends JFrame implements ActionListener {
    String data1[] = {"rosu", "galben", "albastru"};
    String data2[] = {"red", "yellow", "blue"};
    int tipModel = 1;
    JList lst;
    ListModel model1, model2;

    public Fereastra(String titlu) {
        super(titlu);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Lista initiala nu are nici un model
        lst = new JList();
        getContentPane().add(lst, BorderLayout.CENTER);

        // La apasara butonului schimbam modelul
        JButton btn = new JButton("Schimba modelul");
        getContentPane().add(btn, BorderLayout.SOUTH);
        btn.addActionListener(this);

        // Cream obiectele corespunzatoare celor doua modele
        model1 = new Model1();
        model2 = new Model2();
        lst.setModel(model1);

        pack();
    }

    public void actionPerformed(ActionEvent e) {
        if (tipModel == 1) {
            lst.setModel(model2);
            tipModel = 2;
        }
        else {
            lst.setModel(model1);
            tipModel = 1;
        }
    }
}
```

```
// Clasele corespunzatoare celor doua modele
class Model1 extends AbstractListModel {
    public int getSize() {
        return data1.length;
    }
    public Object getElementAt(int index) {
        return data1[index];
    }
}

class Model2 extends AbstractListModel {
    public int getSize() {
        return data2.length;
    }
    public Object getElementAt(int index) {
        return data2[index];
    }
}

}

public class TestModel {
    public static void main(String args[]) {
        new Fereastra("Test Model").show();
    }
}

```

---

# Tratarea evenimentelor

## 1. Informativ (lightweight) ChangeListener - ChangeEvent

Modele: BoundedRangeModel, ButtonModel  
și SingleSelectionModel

```
JSlider slider = new JSlider();
BoundedRangeModel model = slider.getModel();
model.addChangeListener(new ChangeListener() {
    public void stateChanged(ChangeEvent e) {
        // Sursa este de tip BoundedRangeModel
        BoundedRangeModel m =
            (BoundedRangeModel)e.getSource();
        // Trebuie sa interogam sursa asupra schimbarii
        System.out.println(
            "Schimbare model: " + m.getValue());
    }
});
```

## 2. Consistent (statefull)

Model	Tip Eveniment
ListModel	ListDataEvent
ListSelectionModel	ListSelectionEvent
ComboBoxModel	ListDataEvent
TreeModel	TreeModelEvent
TreeSelectionModel	TreeSelectionEvent
TableModel	TableModelEvent
TableColumnModel	TableColumnModelEvent
Document	DocumentEvent
Document	UndoableEditEvent

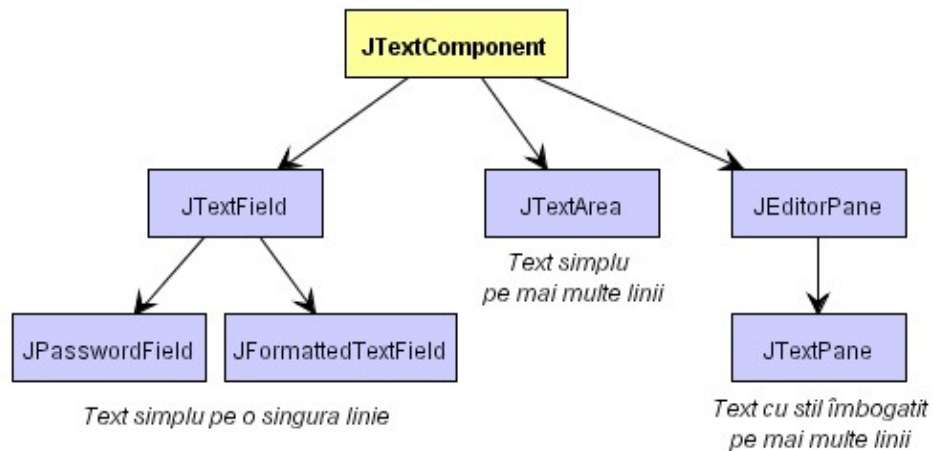
```
String culori[] = {"rosu", "galben", "albastru"};
JList list = new JList(culori);
ListSelectionModel sModel = list.getSelectionModel();
sModel.addListSelectionListener(
    new ListSelectionListener() {
        public void valueChanged(ListSelectionEvent e) {
            // Schimbarea este continuta in eveniment
            if (!e.getValueIsAdjusting()) {
                System.out.println("Selectie curenta: " +
                    e.getFirstIndex());
            }
        }
    }
);
```

# Folosirea componentelor

## Componente atomice

- Etichete: `JLabel`
- Butoane simple sau cu două stări:  
`JButton`, `JCheckBox`, `JRadioButton`,  
`JToggleButton`;
- Componente pentru progres și derulare: `JSlider`, `JProgressBar`, `JScrollbar`
- Separatori: `JSeparator`

# Componente editare de text



Facilități: *undo* și *redo*, tratarea evenimentelor generate de cursor (caret), etc.

Arhitectura **JTextComponent**:

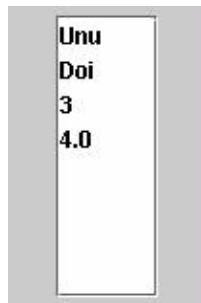
- **Model** - Document
- **Reprezentare**
- **'Controller'** - *editor kit*

## Tratarea evenimentelor

- **ActionEvent**  
ActionListener  
- actionPerformed
- **CaretEvent**  
CaretListener  
- caretUpdate
- **DocumentEvent**  
DocumentListener  
– insertUpdate  
– removeUpdate  
– changedUpdate
- **PropertyChangeEvent**  
PropertyChangeListener  
- propertyChange

# Componente pentru selectare

## Clasa JList



## Inițializarea

```
Object elemente[] = {"Unu", new Integer(2)};  
JList lista = new JList(elemente);
```

```
DefaultListModel model = new DefaultListModel();  
model.addElement("Unu");  
model.addElement(new Integer(2));  
JList lista = new JList(model);
```

```
ModelLista model = new ModelLista();  
JList lista = new JList(model);
```

## Tratarea evenimentelor

```
class Test implements ListSelectionListener {
    ...
    public Test() {
        ...
        // Stabilim modul de selectie
        list.setSelectionMode(
            ListSelectionModel.SINGLE_SELECTION);
            // sau SINGLE_INTERVAL_SELECTION
            //      MULTIPLE_INTERVAL_SELECTION

        // Adaugam un ascultator
        ListSelectionModel model = list.getSelectionModel();
        model.addListSelectionListener(this);
        ...
    }

    public void valueChanged(ListSelectionEvent e) {
        if (e.getValueIsAdjusting()) return;
        int index = list.getSelectedIndex();
        ...
    }
}
```

## Obiecte de tip `Renderer`

Un *renderer* este responsabil cu afișarea articolelor unei componente.

```
class MyCellRenderer extends JLabel
    implements ListCellRenderer {

    public MyCellRenderer() {
        setOpaque(true);
    }

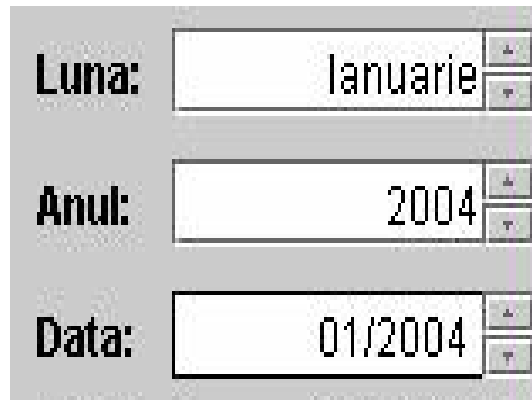
    public Component getListCellRendererComponent(
        JList list, Object value, int index,
        boolean isSelected, boolean cellHasFocus) {
        setText(value.toString());
        setBackground(isSelected ?
            Color.red : Color.white);
        setForeground(isSelected ?
            Color.white : Color.black);
        return this;
    }
}

...
list.setCellRenderer(new MyCellRenderer());
```

## Clasa JComboBox



## Clasa JSpinner



# Tabele

Nume	Varsta	Student
Ionescu	20	true
Popescu	80	false

## Inițializarea

```
String[] coloane = {"Nume", "Varsta", "Student"};
Object[][] elemente = {
    {"Ionescu", new Integer(20), Boolean.TRUE},
    {"Popescu", new Integer(80), Boolean.FALSE}};
JTable tabel = new JTable(elemente, coloane);
```

## Folosirea unui model

```
ModelTabel model = new ModelTabel();
JTable tabel = new JTable(model);
...
class ModelTabel extends AbstractTableModel {
    String[] coloane = {"Nume", "Varsta", "Student"};
    Object[][] elemente = {
        {"Ionescu", new Integer(20), Boolean.TRUE},
        {"Popescu", new Integer(80), Boolean.FALSE}};

    public int getColumnCount() {return coloane.length;}
    public int getRowCount() {return elemente.length;}
    public Object getValueAt(int row, int col) {
        return elemente[row][col];
    }
    public String getColumnName(int col) {
        return coloane[col];
    }
    public boolean isCellEditable(int row, int col) {
        // Doar numele este editabil
        return (col == 0);
    }
}
```

# Tratarea evenimentelor

## 1. Generate de editarea celulelor

```
public class Listener implements TableModelListener {
    public void tableChanged(TableModelEvent e) {
        // Aflam celula care a fost modificata
        int row = e.getFirstRow();
        int col = e.getColumn();
        TableModel model = (TableModel)e.getSource();
        Object data = model.getValueAt(row, col);
        ...
    }
}
...
tabel.getModel().addTableModelListener(new Listener());
```

## 2. Generate de selectarea liniilor

```
class Listener implements ListSelectionListener {
    public void valueChanged(ListSelectionEvent e) {
        if (e.getValueIsAdjusting()) return;
        ListSelectionModel model =
            (ListSelectionModel)e.getSource();
        if (model.isSelectionEmpty()) {
            // Nu este nici o linie selectata
            ...
        } else {
            int index = model.getMinSelectionIndex();
            // Linia cu numarul index este prima selectata
            ...
        }
    }
}

...
tabel.setSelectionMode(
    ListSelectionModel.SINGLE_SELECTION);
ListSelectionModel model = tabel.getSelectionModel();
model.addListSelectionListener(new Listener());
```

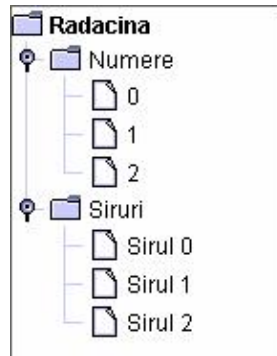
## Folosirea unui renderer

```
public class MyRenderer extends JLabel
    implements TableCellRenderer {
    public Component getTableCellRendererComponent(...) {
        ...
        return this;
    }
}
```

## Folosirea unui editor

```
public class MyEditor
    extends AbstractCellEditor
    implements TableCellEditor {
    public Object getCellEditorValue() {
        // Returneaza valoarea editata
        ...
    }
    public Component
        getTableCellEditorComponent(...) {
        // Returneaza componenta de tip editor
        ...
    }
}
```

# Arbori



```
String text = "<html><b>Radacina</b></html>";
DefaultMutableTreeNode root =
    new DefaultMutableTreeNode(text);
DefaultMutableTreeNode numere =
    new DefaultMutableTreeNode("Numere");
DefaultMutableTreeNode siruri =
    new DefaultMutableTreeNode("Siruri");
for(int i=0; i<3; i++) {
    numere.add(
        new DefaultMutableTreeNode(new Integer(i)));
    siruri.add(
        new DefaultMutableTreeNode("Sirul " + i));
}
root.add(numere);
root.add(siruri);
JTree tree = new JTree(root);
```

## Tratarea evenimentelor

```
class Listener implements TreeSelectionListener {
    public void valueChanged(TreeSelectionEvent e) {
        // Obtinem nodul selectat
        DefaultMutableTreeNode node =
            (DefaultMutableTreeNode)
                tree.getLastSelectedPathComponent();

        if (node == null) return;
        // Obtinem informatia din nod
        Object nodeInfo = node.getUserObject();
        ...
    }
}

...
// Stabilim modul de selectie
tree.getSelectionModel().setSelectionMode(
    TreeSelectionMode.SINGLE_TREE_SELECTION);
// Adaugam un ascultator
tree.addTreeSelectionListener(new Listener());
```

# Personalizarea nodurilor

## TreeCellRenderer

- `setRootVisible`
- `setShowsRootHandles`
- `putClientProperty`

```
tree.putClientProperty("JTree.lineStyle", "Angled");  
// sau "Horizontal", "None"
```

- Specificarea unei iconițe

```
ImageIcon leaf = createImageIcon("img/leaf.gif");  
ImageIcon open = createImageIcon("img/open.gif");  
ImageIcon closed = createImageIcon("img/closed.gif");
```

```
DefaultTreeCellRenderer renderer =  
    new DefaultTreeCellRenderer();  
renderer.setLeafIcon(leaf);  
renderer.setOpenIcon(open);  
renderer.setClosedIcon(closed);
```

```
tree.setCellRenderer(renderer);
```

# Containere

1. **Containere de nivel înalt** - JFrame, JDialog, JApplet
2. **Containere intermediare**
  - JPanel
  - JScrollPane
  - JTabbedPane
  - JSplitPane
  - JLayeredPane
  - JDesktopPane
  - JRootPane

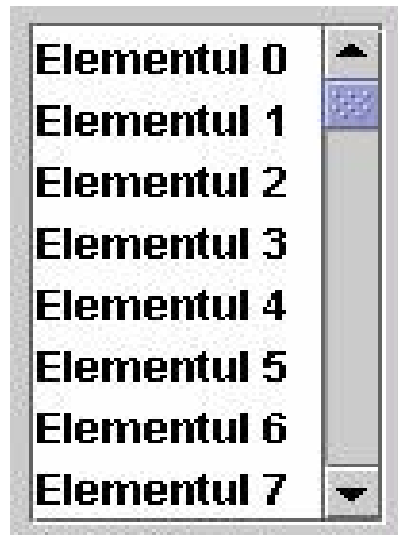
# JPanel

Permite **gruparea** componentelor.

```
JPanel p = new JPanel(new BorderLayout());  
/* Preferabil, deoarece nu mai este construit si  
   un obiect de tip FlowLayout (implicit)  
*/  
p.add(new JLabel("Hello"));  
p.add(new JButton("OK"));  
...
```

# JScrollPane

Oferă suport pentru **derulare**



```
String elemente[] = new String[100];  
for(int i=0; i<100; i++)  
    elemente[i] = "Elementul " + i;
```

```
JList lista = new JList(elemente);  
JScrollPane sp = new JScrollPane(lista);  
frame.getContentPane().add(sp);
```

# JTabbedPane

Permite **suprapunerea** mai multor containere.



```
JTabbedPane tabbedPane = new JTabbedPane();  
ImageIcon icon = new ImageIcon("smiley.gif");
```

```
JComponent panel1 = new JPanel();  
panel1.setOpaque(true);  
panel1.add(new JLabel("Hello"));  
tabbedPane.addTab("Tab 1", icon, panel1,  
                 "Aici avem o eticheta");  
tabbedPane.setMnemonicAt(0, KeyEvent.VK_1);
```

```
JComponent panel2 = new JPanel();
```

```
...
```

# JSplitPane

Oferă suport pentru **separarea** componentelor.



```
JList list;  
JPanel panel;  
JTextArea text;  
...  
JSplitPane sp1 = new JSplitPane(  
    JSplitPane.HORIZONTAL_SPLIT, list, panel);  
JSplitPane sp2 = new JSplitPane(  
    JSplitPane.VERTICAL_SPLIT, sp1, text);  
  
frame.getContentPane().add(sp2);
```

# Dialoguri - Clasa JDialog

- **JOptionPane**

```
JOptionPane.showMessageDialog(frame,  
    "Eroare de sistem !", "Eroare",  
    JOptionPane.ERROR_MESSAGE);
```

```
JOptionPane.showConfirmDialog(frame,  
    "Doriti inchiderea aplicatiei ? ", "Intrebare",  
    JOptionPane.YES_NO_OPTION,  
    JOptionPane.QUESTION_MESSAGE);
```

- **JFileChooser**
- **JColorChooser**
- **ProgressMonitor**

## Desenarea

Metoda **paint** e responsabilă cu:

- **paintComponent**
- **paintBorder**
- **paintChildren**
- "double-buffering"

## Afișarea imaginilor

```
ImageIcon img = new ImageIcon("smiley.gif");  
JLabel label = new JLabel(img);
```

## Transparența

Permite crearea de componente care nu au formă rectangulară.

## Dimensiunile componentelor

```
Insets insets = getInsets();
int currentWidth = getWidth() -
    insets.left - insets.right;
int currentHeight = getHeight() -
    insets.top - insets.bottom;
```

## Contexte grafice

```
public void paintComponent(Graphics g) {
    Graphics2D g2d = (Graphics2D)g;
    // Desenam apoi cu g2d

    // g este refolosit !!
    // 1. Revenirea la starea initiala
    g2d.translate(x, y); // modificam contextul
    ...
    g2d.translate(-x, -y); // revenim la starea initiala

    // 2. Folosirea unei copii
    Graphics2D g2d = (Graphics2D)g.create();
    g2d.translate(x, y);
    ...
    g2d.dispose();
}
```

# Look and Feel

- `javax.swing.plaf.metal.MetalLookAndFeel`
- `com.sun.java.swing.plaf.windows.WindowsLookAndFeel`
- `com.sun.java.swing.plaf.mac.MacLookAndFeel`
- `com.sun.java.swing.plaf.motif.MotifLookAndFeel`
- `com.sun.java.swing.plaf.gtk.GTKLookAndFeel`

```
UIManager.setLookAndFeel(  
    "com.sun.java.swing.plaf.motif.MotifLookAndFeel");  
SwingUtilities.updateComponentTreeUI(f);  
f.pack();
```

```
java -Dswing.defaultlaf=  
    com.sun.java.swing.plaf.gtk.GTKLookAndFeel App
```

```
// In lib/swing.properties  
# Swing properties  
swing.defaultlaf=  
    com.sun.java.swing.plaf.windows.WindowsLookAndFeel
```