Interfața grafică cu utilizatorul - Swing

Programare Orientată pe Obiecte

Ferestre interne

- Aplicațiile pot fi împărțite în:
  - SDI (Single Document Interface)
  - MDI (Multiple Document Interface)
- Clase:

JInternalFrame

DesktopPane – container care va fi apoi plasat pe o fereastră de tip JFrame. Folosirea clasei DesktopPane este necesară deoarece aceasta ”ştie” cum să gestioneze ferestrele interne, având în vedere că acestea se pot suprapune şi la un moment dat doar una singură este activă.

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

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 şi dimensionare
  - setPreferredSize,
  ...
- Controlul opacității - setOpaque
- Asocierea de acțiuni tastelor
- Double-Buffering

Facilități oferite de clasa JComponent (1)

```
import javax . swing .*;
import javax . swing . border .*;
import java . awt .*;
import java . awt. event .*;
class Fereastra extends JFrame {
    public Fereastra ( String titlu ) {
        super ( titlu );
        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 );
        add( panel );

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

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

Facilități oferite de clasa JComponent (2)

```
        // Controlul opacitatii
        JButton btn1 = new JButton (" Opaque ");
        btn1 . setOpaque ( true ); // implicit
        panel .add( btn1 );
        JButton btn2 = new JButton (" Transparent ");
        btn2 . setOpaque ( false );  //dependent de Look&Feel !!
        panel .add( btn2 );
        // ToolTips
        label1 . setToolTipText (" Eticheta coborata ");
        label2 . setToolTipText (" Eticheta ridicata ");
        btn1 . setToolTipText (" Buton opac ");
        btn2 . setToolTipText ("<html><b> Apasati </b> <font color =red >F2</font> " +
                " cand butonul are <u> focusul </u> </html>");
        /* 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 ();
    }
}
class TestJComponent {
    public static void main ( String args []) throws Exception{
        new Fereastra (" Facilitati JComponent "). show ();
        UIManager.setLookAndFeel( "com.sun.java.swing.plaf.motif.MotifLookAndFeel");
    }
}
```

Folosirea componentelor

Componente atomice

- Etichete: JLabel
- Butoane simple sau cu două stări:

JButton, JCheckBox, JRadioButton;

- Componente pentru progres şi derulare:

JSlider, JProgressBar, JScrollBar

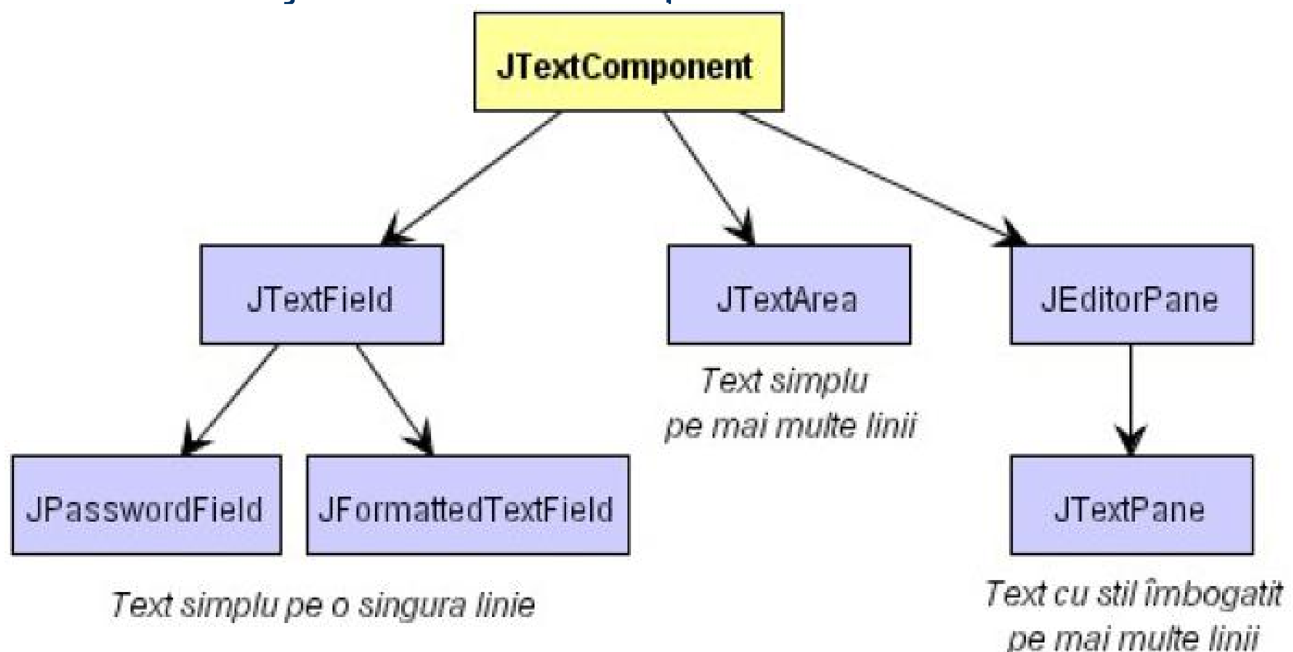
- Separatori: JSeparator

Componente editare de text

Facilităţi: undo şi redo, tratarea evenimentelor generate de cursor (caret), etc.

Arhitectura JTextComponent:

- Model - Document
- Reprezentare
- ’Controller’ - editor kit, permite scrierea şi citirea textului şi definirea de acţiuni necesare editării



Tratarea evenimentelor

- ActionEvent

  ActionListener :

  - actionPerformed
- CaretEvent: generat la deplasarea cursorului ce gestionează poziția curentă în text

  CaretListener :

  - caretUpdate
- DocumentEvent: generat la orice schimbare a textului

  DocumentListener :

  – insertUpdate

  – removeUpdate

  – changedUpdate
- PropertyChangeEvent: eveniment comun tuturor componentelor de tip JavaBean, fiind generat la orice schimbare a unei proprietăți a componentei.

  PropertyChangeListener:

  - propertyChange

Containere

1. Containere de nivel înalt - JFrame, JDialog, JApplet
2. Containere intermediare

– JPanel

– JScrollPane

– JTabbedPane

– JSplitPane

– JDesktopPane

– JRootPane: container utilizat în fundal de JFrame, JDialog, JWindow, JApplet şi JInternalFrame

–JLayeredPane: permite componentelor să se suprapună una peste alta atunci când este necesar

JPanel

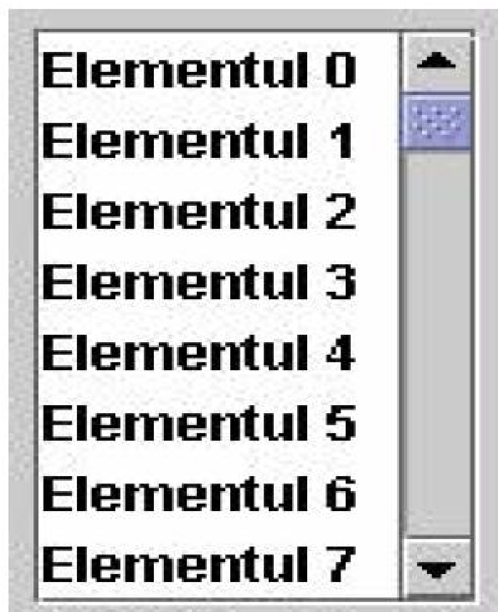
- Permite gruparea componentelor.

```
JPanel p = new JPanel(new BorderLayout());
…
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.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();
panel2.setOpaque(true);
panel2.add(new JButton("OK"));
tabbedPane.addTab("Tab 2", icon,
 panel2,"Aici avem un buton");
tabbedPane.setMnemonicAt(1,
KeyEvent.VK_2);
```

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.add(sp2);
```

Dialoguri - Clasa JDialog

- JOptionPane: Permite crearea unor dialoguri simple, folosite pentru afişarea unor mesaje, realizarea unor interogări de confirmare/renunțare, etc. sau chiar pentru introducerea unor valori
  - 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: monitorizarea progresului unei operații consumatoare de timp

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)
- Fiecărui obiect corespunzător unei clase ce descrie o componentă Swing îi este asociat un obiect care gestionează datele sale şi care implementează o interfață care reprezintă modelul componentei respective.
- Fiecare componentă are un model inițial implicit, însă are posibilitatea de a-l înlocui cu unul nou atunci când este cazul. Metodele care accesează modelul unui obiect sunt: setModel, respectiv getModel, cu argumente specifice fiecărei componente în parte.
- Crearea unui model = implementarea interfeței

JList - ListModel

DefaultListModel, AbstractListModel

Folosirea modelelor

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

Folosirea modelelor - exemplu

```
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 ();
        add(lst , BorderLayout . CENTER );
        // La apasara butonului schimbam modelul
        JButton btn = new JButton (" Schimba modelul ");
        add(btn , BorderLayout . SOUTH );
        btn . addActionListener ( this );
        // Cream obiectele corespunzatoare celor doua modele
        model1 = new Model1 ();
        model2 = new Model2 ();
        lst . setModel ( model1 );
        pack ();
    }
```

Folosirea modelelor - exemplu

```
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 ();
    }
}
```

Observații

- Multe componente Swing furnizează metode care să obțină starea obiectului fără a mai fi nevoie să obținem instanța modelului şi să apelăm metodele acesteia. Un exemplu este metoda getValue a clasei JSlider care este de fapt un apel de genul:

  getModel().getValue().
- In multe situații însă, mai ales pentru clase cum ar fi JTable sau JTree, folosirea modelelor aduce flexibilitate sporită programului şi este recomandată utilizarea lor.

Tratarea evenimentelor: informativ (lightweight)

- Modelele trimit un eveniment prin care sunt informați ascultătorii că a survenit o anumită schimbare a datelor, fără a include în eveniment detalii legate de schimbarea survenită. Obiectele de tip listener vor trebui să apeleze metode specifice componentelor pentru a afla ce anume s-a schimbat.
- ChangeListener - ChangeEvent

  Modele: BoundedRangeModel, ButtonModel şi 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());
  }
});
```

Tratarea evenimentelor: informativ (lightweight) (2)

- Pentru uşurința programării, pentru a nu lucra direct cu instanța modelului, unele clase permit înregistrarea ascultătorilor direct pentru componenta în sine, singura diferență față de varianta anterioară constând în faptul că sursa evenimentului este acum de tipul componentei şi nu de tipul modelului.

```
JSlider slider = new JSlider();
slider.addChangeListener(new ChangeListener() {
    public void stateChanged(ChangeEvent e) {
        // Sursa este de tip JSlider
        JSlider s = (JSlider)e.getSource();
        System.out.println("Valoare noua: " +
                                    s.getValue());
    }
});
```

Tratarea evenimentelor

2. Consistent(statefull): Modele pun la dispoziție interfețe specializate şi tipuri de evenimente specifice ce includ toate informațiile legate de schimbarea datelor.

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

Tratarea evenimentelor

```
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());
    }
 }
});
```

Sau:

```
JList list = new JList(culori);
list.addListSelectionListener(new ListSelectionListener() {
    public void valueChanged (ListSelectionEvent e) {
    …
});
```

Componente pentru selectare
Clasa JList

- Object elemente[] = {"Unu", new Integer(2)};
  JList lista = new JList(elemente);
- Vector elemente = new Vector();
  elemente.add(“Unu”); elemente.add(new Integer(2));
  JList lista = new JList(elemente);

Obs: modificarea elementelor vectorului pe baza căruia a fost creata inițial lista nu duce la modificarea elementelor listei!! Pentru actualizarea elementelor listei se va apela funcția **setListData(vector)** din clasa JList.

- DefaultListModel model = new DefaultListModel();

model.addElement("Unu");

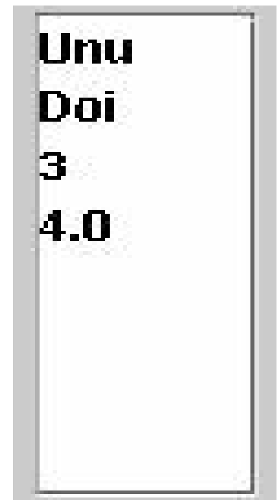
model.addElement(new Integer(2));

JList lista = new JList(model);

Obs: modificarea elementelor modelului pe baza căruia a fost creata inițial lista se reflectă automat în listă!!

Clasa JList

```
ModelLista model = new ModelLista();
JList lista = new JList(model);
...
class ModelLista extends AbstractListModel {
  Object elemente[] = {"Unu", "Doi", new Integer(3),
new Double(4)};
  public int getSize() {
     return elemente.length;
 }
  public Object getElementAt(int index) {
    return elemente[index];
  }
}
```



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

Obiecte de tip Renderer

- Un renderer este responsabil cu afişarea 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

- similară cu JList, cu deosebirea că permite doar selectarea unui singur articol, acesta fiind şi singurul permanent vizibil.
- Inițializarea se face folosind fie un vector fie un model de tipul ComboBoxModel
- fiecare element poate fi de asemenea reprezentat diferit prin intermediul unui obiect ce implementează aceeaşi intefață ca şi în cazul listelor:ListCellRenderer.
- JComboBox permite şi editarea explicită a valorii elementului, acest lucru fiind controlat de metoda setEditable.
- Evenimentele generate de obiectele JComboBox sunt de tip ItemEvent generate la navigarea prin listă, respectiv ActionEvent generate la selectarea efectivă a unui articol.

Clasa JSpinner

• oferă posibilitatea de a selecta o anumită valoare (element) dintr-un domeniu prestabilit, lista elementelor nefiind însă vizibilă. Este folosit atunci când domeniul din care poate fi făcută selecția este foarte mare sau chiar nemărginit; de exemplu: numere intregi intre 1950 si 2050.
• se bazează exclusiv pe folosirea unui model. Acesta este un obiect de tip SpinnerModel (SpinnerListModel, SpinnerNumberModel sau SpinnerDateModel)
• permit şi specificarea unui anumit tip de editor pentru valorile elementelor sale. Acesta este instalat automat pentru fiecare din modelele standard amintite mai sus
• evenimentele generate de obiectele de tip JSpinner sunt de tip ChangeEvent, generate la schimbarea stării componentei.