



Programare avansată
Interfața grafică cu
utilizatorul (GUI)

Interfețe cu utilizatorul

Human Machine Interface

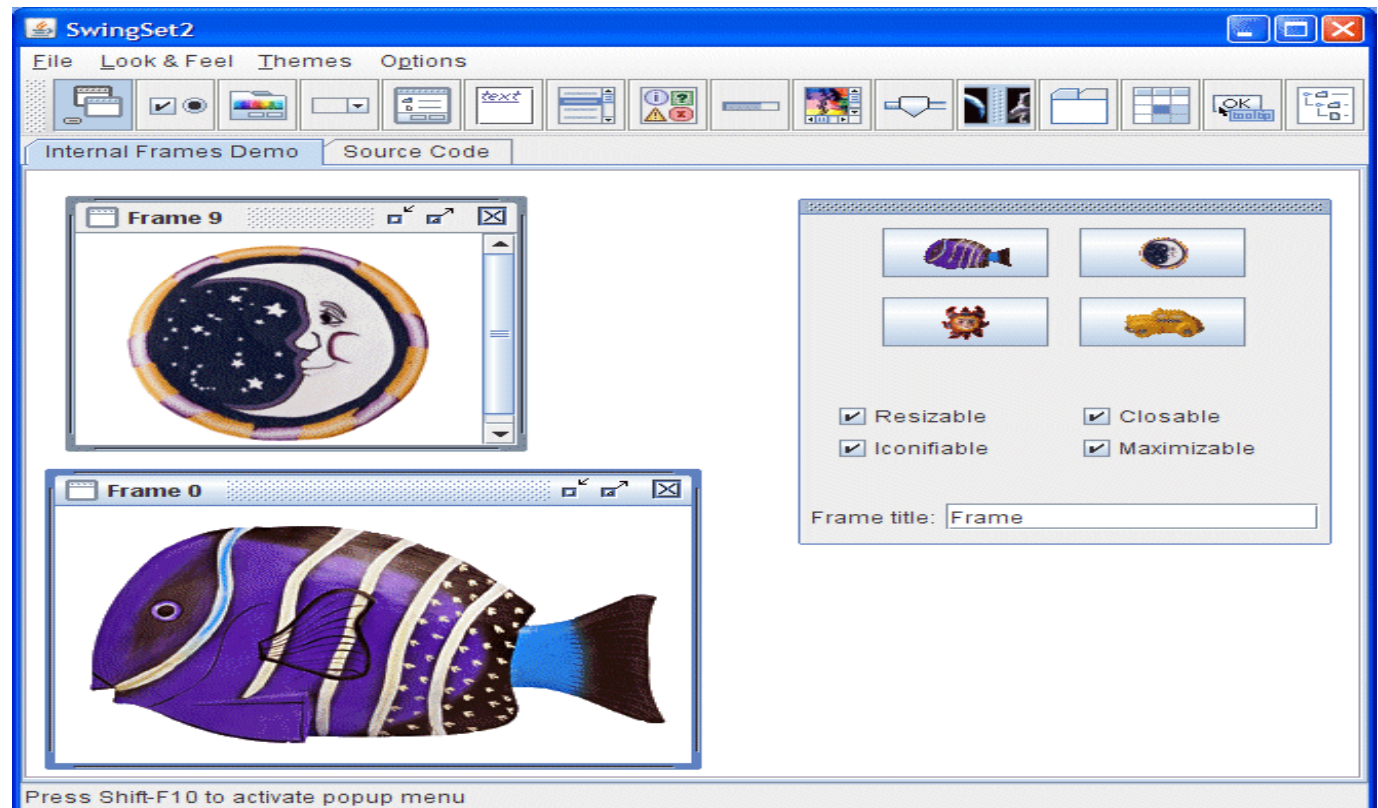
Modalitățile prin care un sistem (software) interacționează cu utilizatorii săi (umani).

- Linia de comandă
- Grafice (Graphical User Interface - GUI)
- Tactile (Touch User Interface - TUI)
- Multimedia (voce, animație, etc.)
- Inteligente (recunoașterea gesturilor, conversaționale, etc)

Interfețe grafice

Comunicarea **vizuală** între un program și utilizatori.

- **AWT**(Abstract Windowing Toolkit)
- **Swing** - parte din JFC (Java Foundation Classes)
- **SWT** (IBM)
- **Java FX**
- **XUL**
- ...
- Java 2D
- Java 3D



Etapele creării unei aplicații cu interfață grafică

❏ Design

- Crearea suprafețelor de afișare (containere)
- Crearea și asezarea componentelor



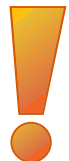
❏ Funcționalitate

- Definirea unor acțiuni
- “Legarea” componentelor de acțiuni
- ”Ascultarea” și tratarea evenimentelor



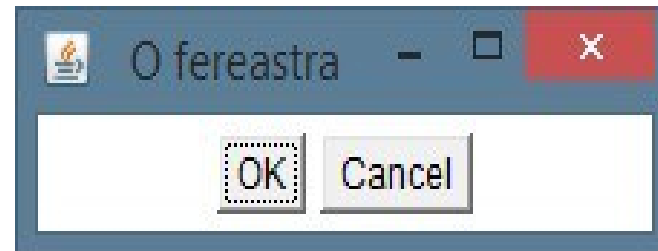
❏ Considerente

- Programatic – Declarativ – Vizual
- Separare dintre nivelul GUI și logica aplicației



Modelul AWT

```
import java.awt.*;  
public class ExempluAWT1 {  
    public static void main (String args []) {  
        // Crearea ferestrei - un obiect de tip Frame  
        Frame f = new Frame("O fereastră");  
  
        // Setarea modului de dispunere a componentelor  
        f.setLayout (new FlowLayout());  
  
        // Crearea a doua butoane  
        Button b1 = new Button("OK");  
        Button b2 = new Button("Cancel");  
        // Adaugarea butoanelor  
        f.add(b1);  
        f.add(b2);  
        f.pack();  
  
        // Afisarea ferestrei  
        f.setVisible(true);  
    }  
}
```



Componente AWT

- ✓ Button
- ✓ Canvas
- ✓ Checkbox
- ✓ CheckBoxGroup
- ✓ Choice
- ✓ Container
- ✓ Label
- ✓ List
- ✓ Scrollbar
- ✓ TextComponent
- ✓ TextField
- ✓ TextArea



Dependente de sistemul de operare (*peer*)

Infrastructura AWT

- **Component**

- proprietăți comune tuturor componentelor
(location, x, y, size, height, width, bounds, foreground, background, font, visible, enabled,...)

- **Container**

- metode comune tuturor containerelor

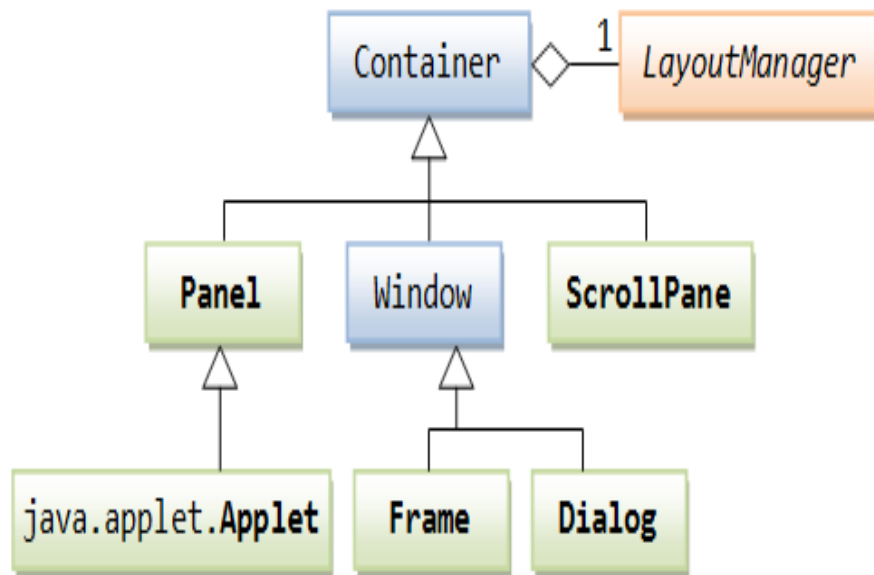
- **LayoutManager**

- interfața pentru orice gestionar de poziționare

- **AWTEvent**

- superclasa pentru evenimente

Ferestre și panel-uri



```
Frame f = new Frame("O fereastră");
// Adaugam un buton direct pe fereastră
f.add(new Button("Hello"));
```

```
// Adaugam doua componente pe un panel
Panel panel = new Panel();
panel.add(new Label("Nume:"));
panel.add(new TextField());
```

```
// Adaugam panel-ul pe fereastră
// si, indirect, cele doua componente
f.add(panel);
```

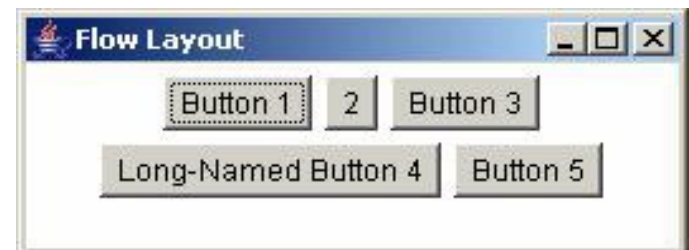
```
class Fereastra extends Frame {
    // Constructorul
    public Fereastra(String titlu) {
        super(titlu);
        ...
    }
}
...
Fereastra f = new Fereastra("O fereastră");
f.setVisible(true);
```


Gestionarea poziționării

```
import java.awt.*;  
public class TestLayout {  
    public static void main ( String args [])  
  
        Frame f = new Frame("Grid Layout");  
        f.setLayout (new GridLayout (3, 2));  
  
        Button b1 = new Button (" Button 1");  
        Button b2 = new Button ("2");  
        Button b3 = new Button (" Button 3");  
        Button b4 = new Button ("Long - Named Button 4");  
        Button b5 = new Button (" Button 5");  
        f.add(b1); f.add (b2); f. add(b3); f.add(b4); f.add(b5);  
        f.pack ();  
        f.setVisible(true);  
    }  
}
```



```
Frame f = new Frame("Flow Layout");  
f.setLayout (new FlowLayout ());
```



LayoutManager

Un ***gestionar de poziționare*** este un obiect care controlează dimensiunea și aranjarea (poziția) componentelor unui container.

Fiecare obiect de tip *Container* are asociat un gestionar de poziționare.

Toate clasele care instanțiază obiecte pentru gestionarea poziționării implementează interfață ***LayoutManager***.

La instanțierea unui container se creează implicit un gestionar de poziționare asociat acestuia:

- ferestre: *BorderLayout*
- panel-uri: *FlowLayout*

Gestionari de poziționare “clasici”

`FlowLayout, BorderLayout, GridLayout,
CardLayout, GridBagLayout`

Metoda *setLayout*

```
container.setLayout(new FlowLayout());
```

Dimensionarea componentelor

```
preferredSize, minimumSize, maximumSize
```

Poziționarea absolută

```
container.setLayout(null);  
Button b = new Button("Buton");  
b.setSize(10, 10);  
b.setLocation (0, 0);  
container.add(b);
```

BorderLayout

```
import java.awt .*;  
public class TestBorderLayout {  
    public static void main ( String args []) {  
  
        Frame f = new Frame (" Border Layout ");  
        // Apelul de mai jos poate sa lipseasca  
        f.setLayout (new BorderLayout());  
  
        f.add(new Button(" Nord "), BorderLayout.NORTH );  
        f.add(new Button(" Sud"), BorderLayout.SOUTH );  
        f.add(new Button(" Est"), BorderLayout.EAST );  
        f.add(new Button(" Vest "), BorderLayout.WEST );  
        f.add(new Button(" Centru "), BorderLayout.CENTER );  
        f.pack ();  
        f.setVisible(true);  
    }  
}
```



GridBagLayout

```
GridBagLayout gridBag = new GridBagLayout();  
container.setLayout(gridBag);
```

```
GridBagConstraints c = new GridBagConstraints();  
//Specificam restrictiile  
c.fill = GridBagConstraints.HORIZONTAL;  
c.gridx = 0;  
c.gridy = 0;
```

- `gridx, gridy`
- `gridwidth, gridheight`
- `fill`
- `insets`
- `anchor`
- `weightx, weighty`

. . .

```
gridBag.setConstraints(componenta, c);  
container.add(componenta);
```



Interacțiunea cu utilizatorul

Event-Driven Programming

Eveniment: apăsarea unui buton, modificarea textului, închiderea unei ferestre, etc.

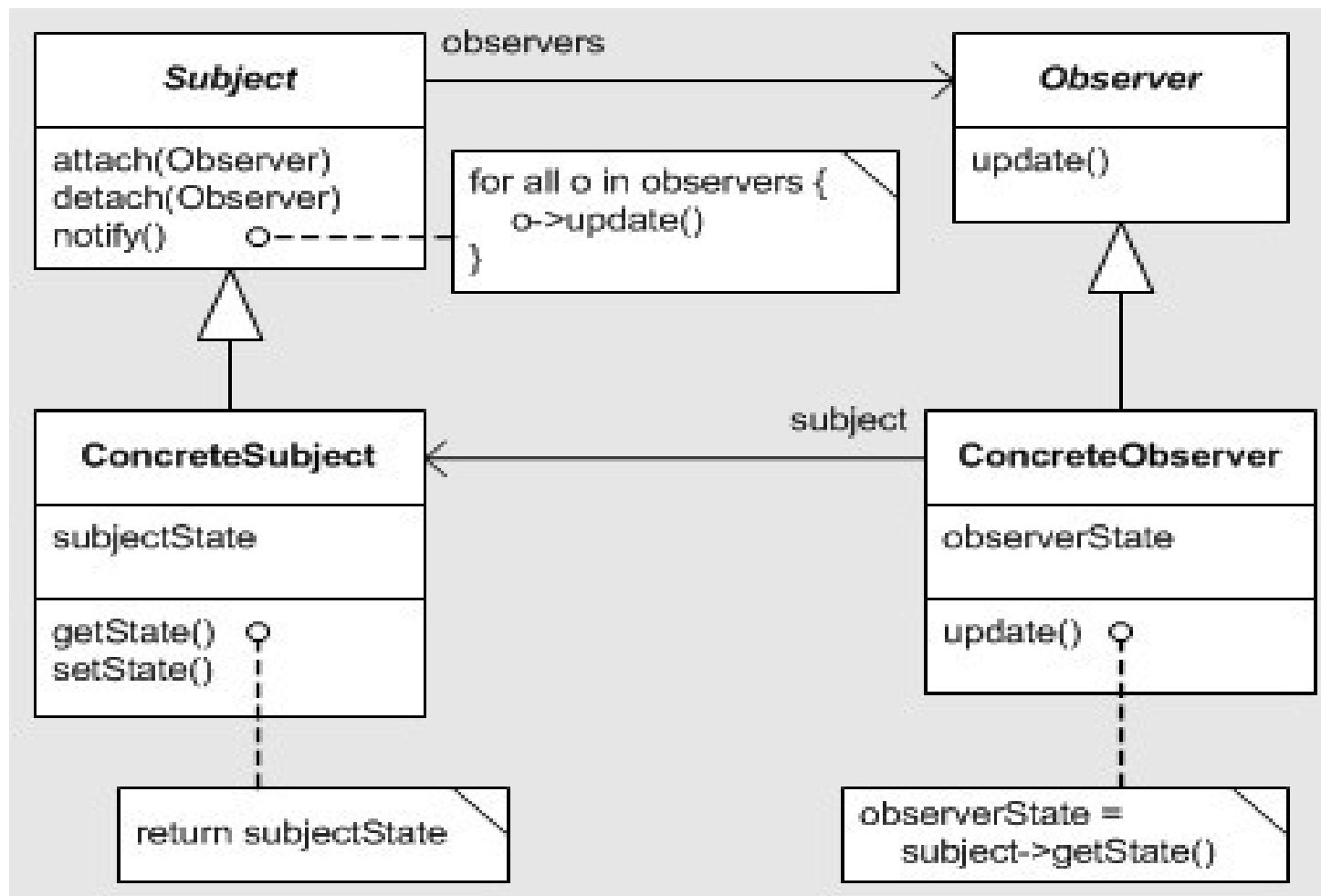
Sursă: componenta care generează un eveniment.

Listener: responsabil cu interceptarea evenimentelor (consumator de evenimente).



Observer Design Pattern

Observarea stării unei entități în cadrul unui sistem
(*Publish-Subscribe*)



Button - ActionEvent - ActionListener

```
class Fereastra extends Frame {
    public Fereastra ( String titlu ) {
        super (titlu);
        setLayout (new FlowLayout ());
        setSize (200, 100) ;
        Button b1 = new Button ("OK");
        Button b2 = new Button ("Cancel");
        add(b1); add(b2);
        listener = new MyButtonListener (this);
        b1.addActionListener ( listener );
        b2.addActionListener ( listener );
        // Ambele butoane sunt "ascultate" de obiectul listener,
        // instanta a clasei MyButtonListener, definita mai jos
    }
}

class MyButtonListener implements ActionListener {
    private Fereastra frame;
    public MyButtonListener (Fereastra frame) {
        this.frame = frame;
    }
    // Metoda interfetei ActionListener
    public void actionPerformed (ActionEvent e) {
        frame.setTitle ("Ati apasat " + e.getActionCommand ());
    }
}
```


Tipuri de evenimente

Low-level	Semantice
ComponentEvent <i>ascundere, deplasare, redimensionare, afișare componente</i>	ActionEvent <i>apăsarea unui buton, apăsarea tastei enter într-un textfield, etc.</i>
ContainerEvent <i>adăugare, eliminare componente în/din containere</i>	AdjustmentEvent <i>ajustarea valorii unei componente, de exemplu un scrollbar</i>
FocusEvent <i>obținere, pierdere focus</i>	ItemEvent <i>schimbarea stării unei componente: selectarea unor articole într-o listă, apăsarea unui checkbox</i>
KeyEvent <i>apăsare, eliberare taste</i>	TextEvent <i>schimbarea textului într-o componentă de tipul textfield, textarea</i>
MouseEvent <i>operațiuni cu mouse-ul: click, drag, etc.</i>	. . .
WindowEvent <i>operațiuni asupra ferestrelor: minimizare, redimensionare, etc.</i>	

Relația “Componentă - *Listener*”

many-to-many

Component	ComponentListener FocusListener KeyListener MouseListener
Container	ContainerListener
Window	WindowListener
Button List MenuItem TextField	ActionListener
Choice Checkbox List	ItemListener
Scrollbar	AdjustmentListener
TextField TextArea	TextListener

Metode *handler*

ActionListener

`actionPerformed(ActionEvent e)`

ItemListener

`itemStateChanged(ItemEvent e)`

TextListener

`textValueChanged(TextEvent e)`

MouseListener

`mouseClicked(MouseEvent e)`
`mouseEntered(MouseEvent e)`
`mouseExited(MouseEvent e)`
`mousePressed(MouseEvent e)`
`mouseReleased(MouseEvent e)`

MouseMotionListener

`mouseDragged(MouseEvent e)`
`mouseMoved(MouseEvent e)`

WindowListener

`windowActivated(WindowEvent e)`
`windowClosed(WindowEvent e)`
`windowClosing(WindowEvent e)`
`windowDeactivated(WindowEvent e)`
`windowDeiconified(WindowEvent e)`
`windowIconified(WindowEvent e)`
`windowOpened(WindowEvent e)`

...

...

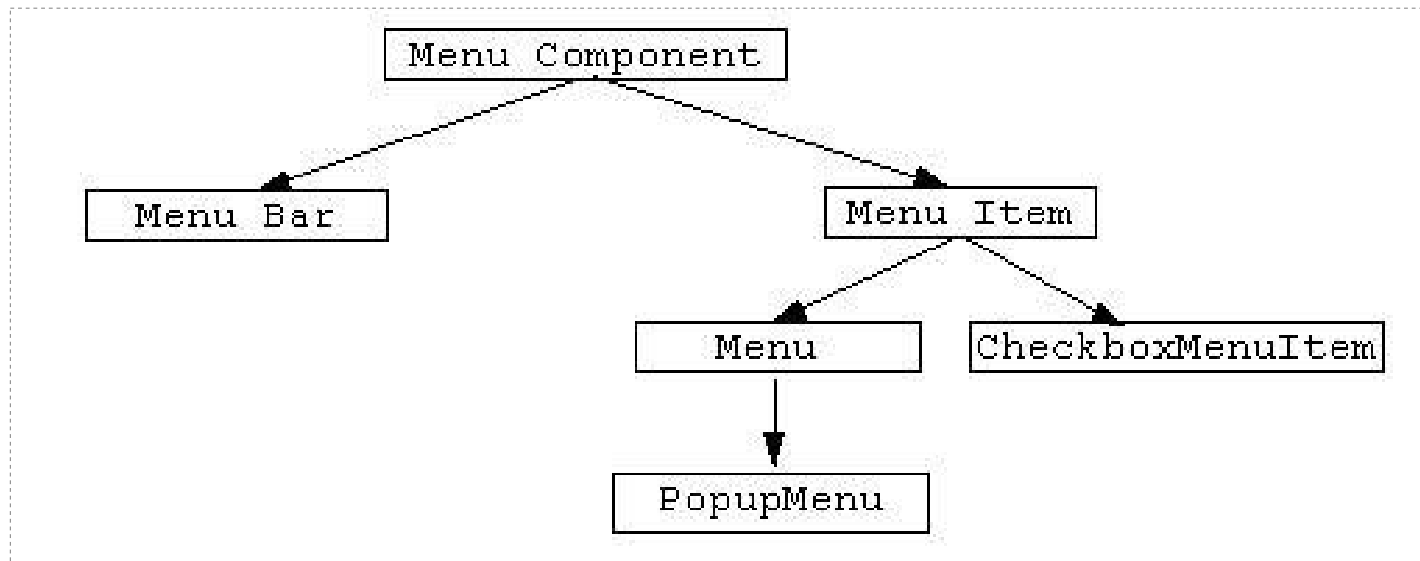
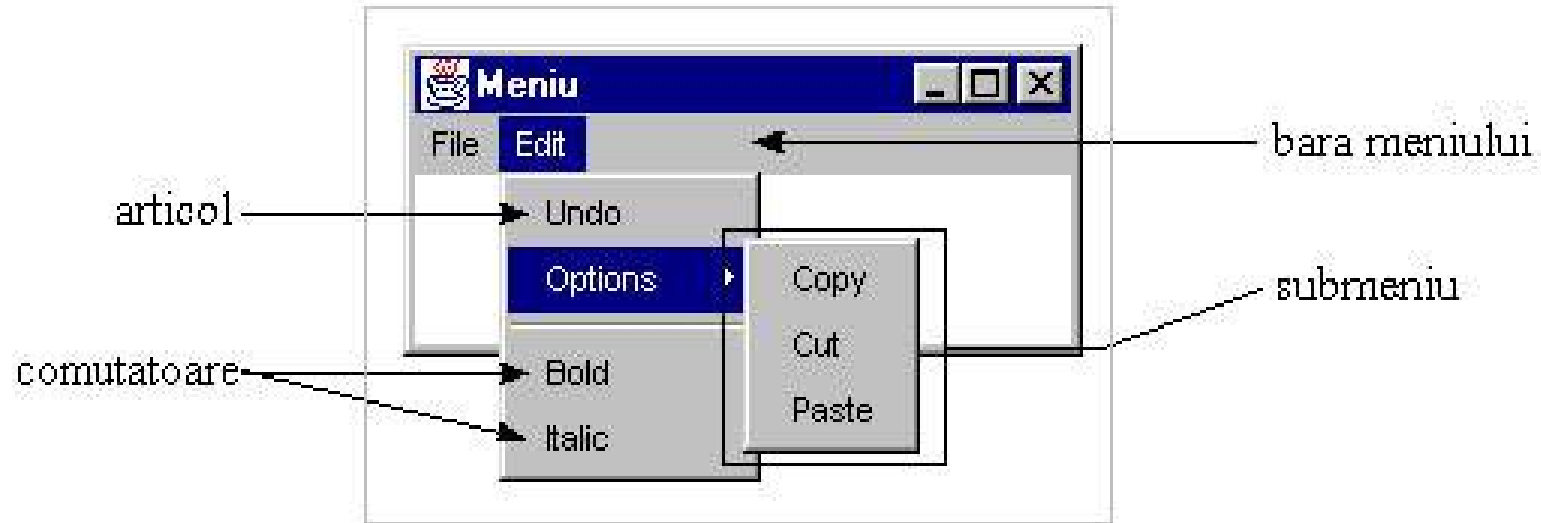
Adaptori și clase anonime

```
class Fereastra extends Frame implements WindowListener {
    public Fereastra (String titlu) {
        super (titlu);
        this.addWindowListener(this);
    }
    // Metodele interfetei WindowListener
    public void windowOpened ( WindowEvent e) {}
    public void windowClosing ( WindowEvent e) {
        // Terminare program
        System.exit (0);
    }
    public void windowClosed ( WindowEvent e) {}
    public void windowIconified ( WindowEvent e) {}
    public void windowDeiconified ( WindowEvent e) {}
    public void windowActivated ( WindowEvent e) {}
    public void windowDeactivated ( WindowEvent e) {}
}
```

Un **adaptor** este o clasă abstractă care implementează o anumită interfață fără a specifica cod nici unei metode a interfeței.

```
this.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});
```

Folosirea meniurilor



Swing

- **Extinde** tehnologia AWT, preluând conceptele și mecanismele de bază
- Aduce un **set nou de componente**, mult mai complex, înlocuindu-l complet pe cel din AWT
- Aduce **portabilitatea** la nivelul interfeței grafice; aceasta nu mai depinde de sistemul de operare
- Introduce o **arhitectură cu model separabil**
- Dezvoltarea interfeței grafice devine:

"Programare orientată pe componente"

Java Foundation Classes

Java Foundation Classes (JFC) reprezintă un cadru de lucru pentru dezvoltarea de aplicații portabile având interfață grafică desktop. Conține mai multe biblioteci, toate incluse în platforma Java SE:

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

Paleta de componente 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

Convenția "J"

`java.awt.Frame` - `javax.swing.JFrame`

`java.awt.Button` - `javax.swing.JButton`

`java.awt.Label` - `javax.swing.JLabel`

Noi gestionari de poziționare:

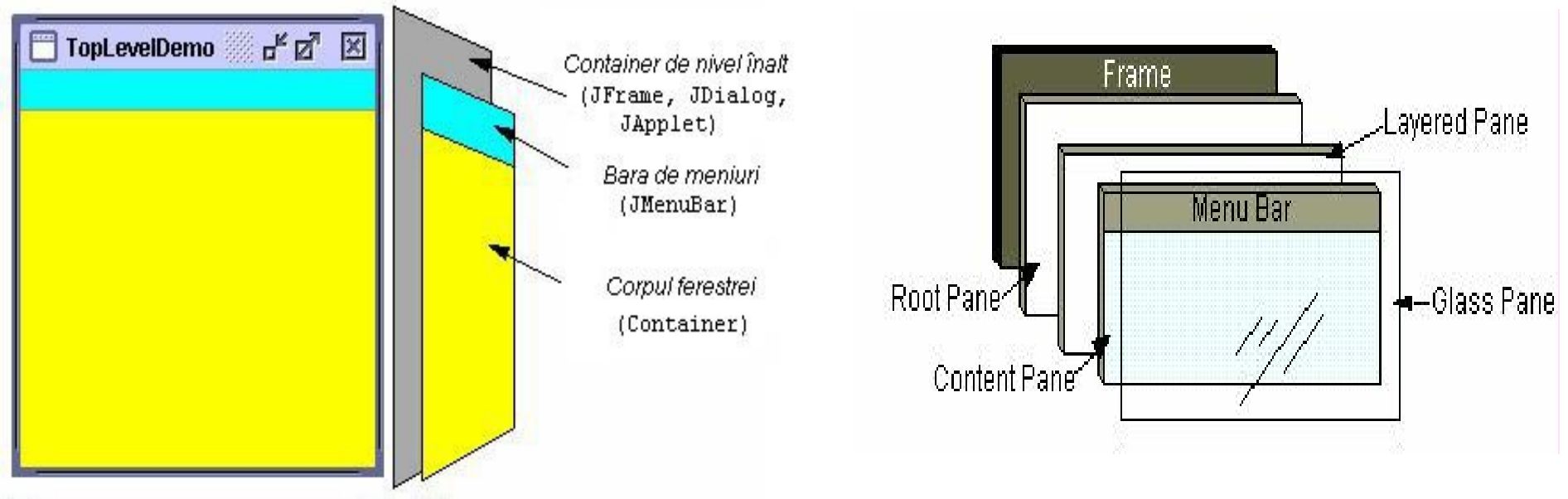
`BoxLayout`, `SpringLayout`, `GroupLayout`, `OverlayLayout`, etc.

Folosirea HTML

```
JButton simplu = new JButton("Text simplu");
```

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

Folosirea ferestrelor în Swing



```
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"));
```

Ferestre interne

Aplicațiile pot fi clasificate ca:

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

`JInternalFrame`, `DesktopPane`



Clasa *JComponent*

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

JComponent extinde clasa Container.

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

Arhitectura Swing

MVC (Model-View-Controller)

- *Model* - datele aplicației
- *Prezentare* - reprezentarea vizuală
- *Control* - transformarea acțiunilor în evenimente

Arhitectură cu model separabil

Model + (Prezentare, Control)

Reprezentare - Model

Componentă	Interfața care descrie modelul
JList	ListModel ListSelectionModel
JTable	TableModel TableColumnModel ListSelectionModel
JTree	TreeModel TreeSelectionModel
JEditorPane JTextPane JTextField	Document
...	...

Crearea unui model = implementarea interfeței
JList - ListModel, DefaultListModel, AbstractListModel

Exemplu *JList*

Componentele sunt instanțiate pe baza unui model.

Acesta poate fi reprezentat fie de o structură de date simplă:

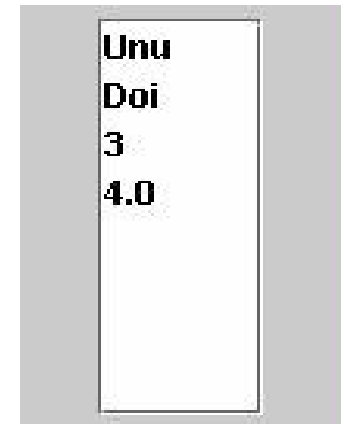
```
Object elemente[] = {"Unu", "Doi", new Integer(3), 4.0};
```

```
JList lista = new JList(elemente);
```

fie de o instanță *model* specifică

```
DefaultListModel model = new DefaultListModel();  
model.addElement("Unu");  
model.addElement("Doi");  
model.addElement(new Integer(3));  
model.addElement(4.0);
```

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



Exemplu *JTable*

```
class MyTableModel extends AbstractTableModel {  
    private String[] coloane = {"Nume", "Varsta", "Student"};  
    private 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);  
    }  
}
```

Nume	Varsta	Student
Ionescu	20	true
Popescu	80	false

Personalizarea reprezentării

SwingSet2

File Look & Feel Themes Options Multiscreen

Table Demo Source Code

☒ Reordering allowed ☒ Row selection
☒ Horiz. Lines ☐ Column selection
☒ Vert. Lines

Inter-cell spacing:
Row height:

Selection mode: Multiple ranges
Autoresize mode: Subsequent columns

Printing
Header: JTable Printing
Footer: Page {0}
☒ Fit Width

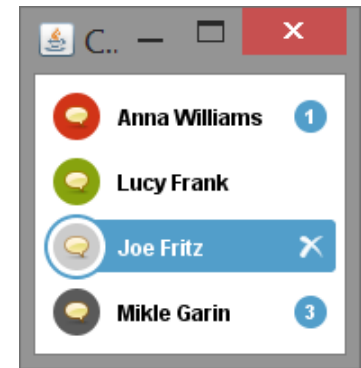
First Name	Last Name	Favorite Color	Favorite Movie	Favorite Number	Favorite Food
Mike	Albers	Green	Brazil	44	
Mark	Andrews	Blue	Curse of the Demon	3	
Brian	Beck	Black	The Blues Brothers	2.718	
Lara	Bunni	Red	Airplane (the whol...	15	
Roger	Brinkley	Blue	The Man Who Kne...	13	
Brent	Christian	Black	Blade Runner (Dir...	23	
Mark	Davidson	Dark Green	Brazil	27	
Jeff	Dinkins	Blue	The Lady Vanishes	8	
Ewan	Dinkins	Yellow	A Bug's Life	2	
Amy	Fowler	Violet	Reservoir Dogs	3	
Hania	Gajewska	Purple	Jules et Jim	5	

Press Shift-F10 to activate popup menu

Conceptul de *CellRenderer*

Un **renderer** este responsabil cu afișarea articolelor unei componente, de exemplu înregistrările dintr-o listă.

```
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());
```



Conceptul de *CellEditor*

Un **editor** este responsabil cu editarea articolelor unei componente, de exemplu informația din celulele unui tabel.

```
public class MyCellEditor extends AbstractCellEditor  
    implements TableCellEditor {
```

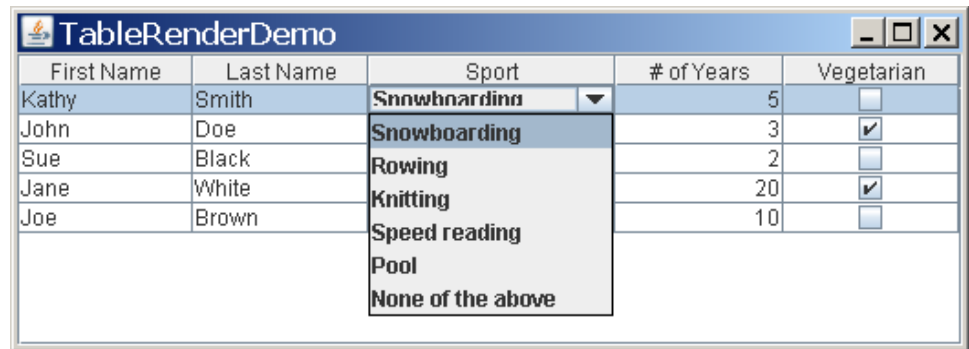
```
    public Component getTableCellEditorComponent(...) {
```

```
        // Returneaza componenta
```

```
        // de tip editor
```

```
        ...
```

```
    }
```



First Name	Last Name	Sport	# of Years	Vegetarian
Kathy	Smith	Snowboarding	5	<input type="checkbox"/>
John	Doe	Snowboarding	3	<input checked="" type="checkbox"/>
Sue	Black	Rowing	2	<input type="checkbox"/>
Jane	White	Knitting	20	<input checked="" type="checkbox"/>
Joe	Brown	Speed reading	10	<input type="checkbox"/>

```
    public Object getCellEditorValue() {
```

```
        // Returneaza valoarea editata
```

```
        ...
```

```
    }
```

```
}
```

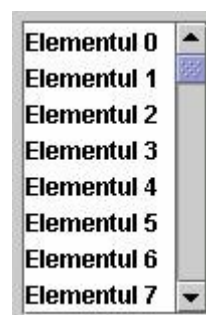
Containere Swing

❏ Containere de nivel înalt

JFrame, JDialog, JApplet

❏ Containere intermediare

- JPanel
- JScrollPane
- JTabbedPane
- JSplitPane
- JLayeredPane
- JDesktopPane
- JRootPane



Look and Feel

Schimbarea aspectului general al interfeței cu utilizatorul prin intermediul unor **teme grafice**:

- `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();
```

The Java Tutorial

- **Trail: Graphical User Interfaces**

<http://docs.oracle.com/javase/tutorial/ui/index.html>

- **Trail: Creating a GUI With JFC/Swing**

<http://docs.oracle.com/javase/tutorial/uiswing/index.html>