



Advanced Programming Graphical User Interface (GUI)

Human-Machine Interfaces

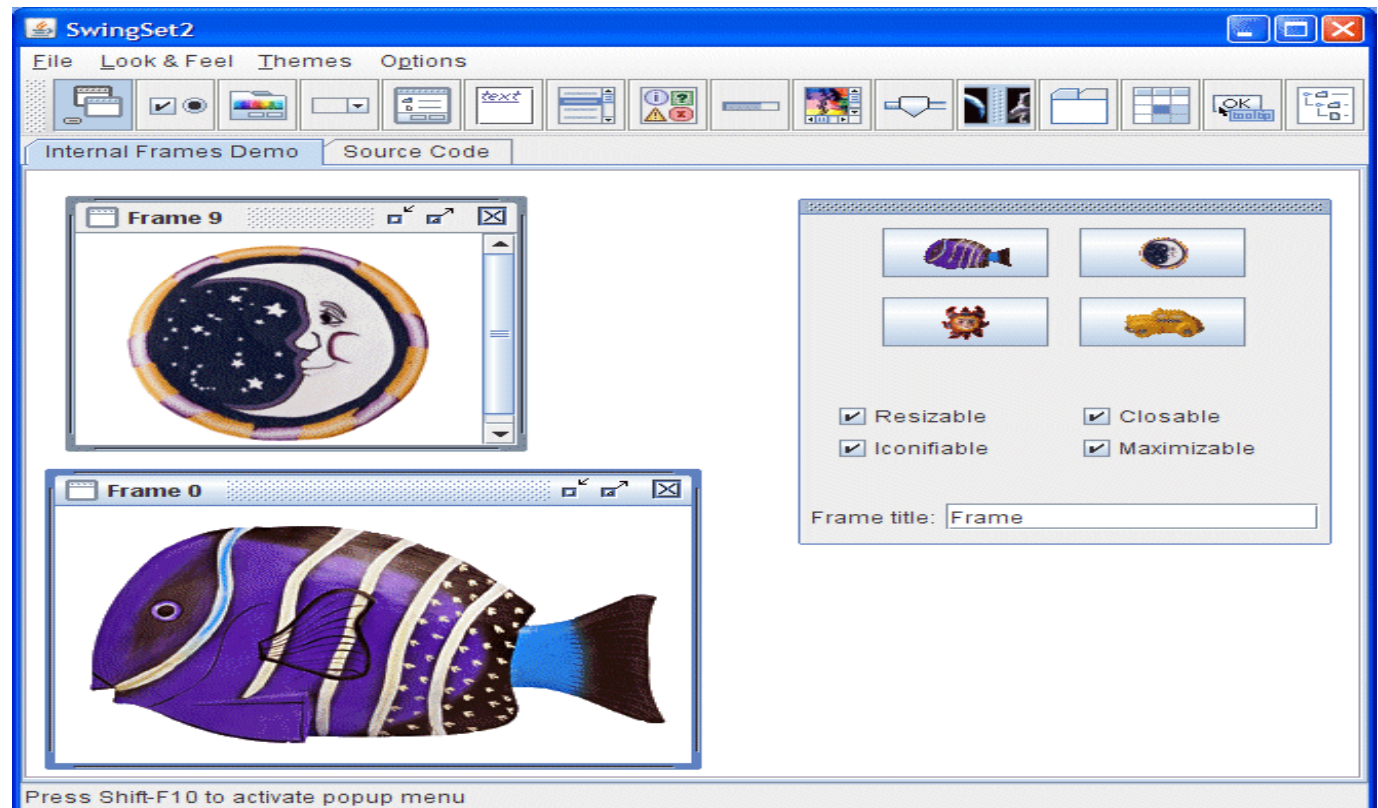
The ways in which a *software system* interacts with its *users*.

- Command Line
- Graphical User Interface - GUI
- Touch User Interface - TUI
- Multimedia (voice, animation, etc.)
- Intelligent (gesture recognition, conversational, etc.)

Graphical User Interfaces

Visual communication between software and users.

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



The Stages of Creating a GUI Application

❏ Design

- Create the **containers**
- Create and arrange the **components**



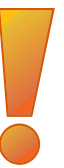
❏ Functionality

- Define the user-components **interaction**
- Attach **actions** to components
- Create the action **handlers**



❏ Considerations

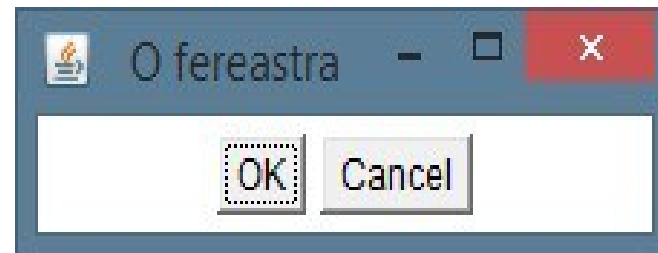
- Programatic – Declarative – Visual
- Separation between the GUI and application logic



AWT Library

```
import java.awt.*;  
public class AWTEExample {  
    public static void main (String args []) {  
        // Create the window (frame)  
        Frame f = new Frame("O fereastră");  
  
        // Set the layout of the frame  
        f.setLayout (new FlowLayout());  
  
        // Create the components  
        Button b1 = new Button("OK");  
        Button b2 = new Button("Cancel");  
  
        // Add the components to the frame  
        f.add(b1);  
        f.add(b2);  
        f.pack();  
  
        // Show the frame  
        f.setVisible(true);  
    }  
}
```

AWT is the original
Java GUI library.



AWT Components

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



AWT Components are **platform-dependend**,
each of them having an underlying **native peer**.

AWT Infrastructure

- **Component**

- A component is an object having a graphical representation that can be displayed on the screen and that can interact with the user. Properties common to all components are:

location, x, y, size, height, width, bounds, foreground, background, font, visible, enabled,...

- **Container**

- A generic component containing other components.

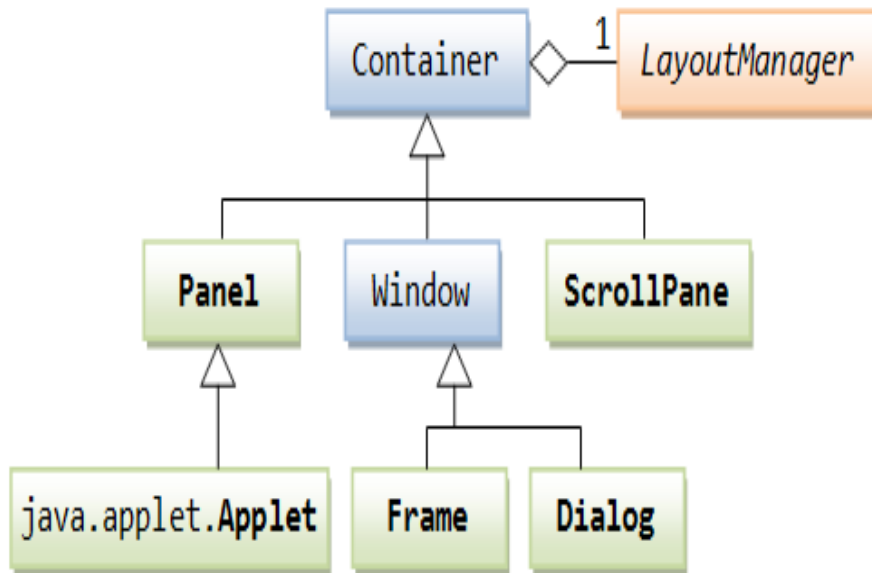
- **LayoutManager**

- The interface for classes that know how to lay out Containers.

- **AWTEvent**

- The root event class for all AWT events

Frames and Panels



```
Frame f = new Frame("Hello Frame");
// Add a button on the frame
f.add(new Button("Hello"));
```

```
// Create a panel
Panel panel = new Panel();
panel.add(new Label("Name:"));
panel.add(new TextField());
```

```
// Add the panel on the frame
f.add(panel);
```

```
class MyFrame extends Frame {
```

```
    // Constructor
```

```
    public MyFrame(String title) {
        super(title);
```

```
        ...
```

```
    }
```

```
}
```

```
...
```

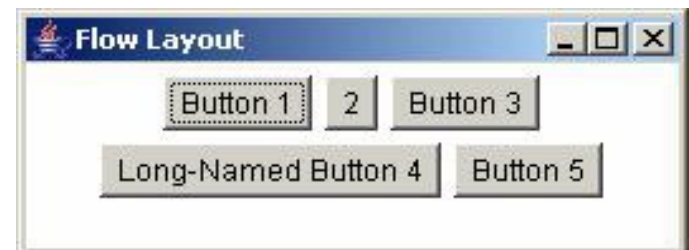
```
MyFrame f = new MyFrame("My very special frame");
f.setVisible(true);
```


Arranging the Components

```
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

A **layout manager** is an object that controls the size and arrangement (position) of components inside a container.

Each *Container* object has a layout manager.

All classes that instantiate objects for managing positioning implements *LayoutManager* interface.

Upon instantiation of a container it is created an implicit layout manager associated with it:

- frames: *BorderLayout*
- panels: *FlowLayout*

“Classical” Layout Managers

`FlowLayout`, `BorderLayout`, `GridLayout`,
`CardLayout`, `GridBagLayout`

Setting a layout manager

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

Controlling the dimensions of the components

```
preferredSize, minimumSize, maximumSize
```

Absolute positioning

```
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 ");  
        // This is the default for frames  
        f.setLayout (new BorderLayout());  
  
        f.add(new Button(" North "), BorderLayout.NORTH );  
        f.add(new Button(" South"), BorderLayout.SOUTH );  
        f.add(new Button(" East"), BorderLayout.EAST );  
        f.add(new Button(" West "), BorderLayout.WEST );  
        f.add(new Button(" Center "), BorderLayout.CENTER );  
        f.pack ();  
        f.setVisible(true);  
    }  
}
```



GridBagLayout

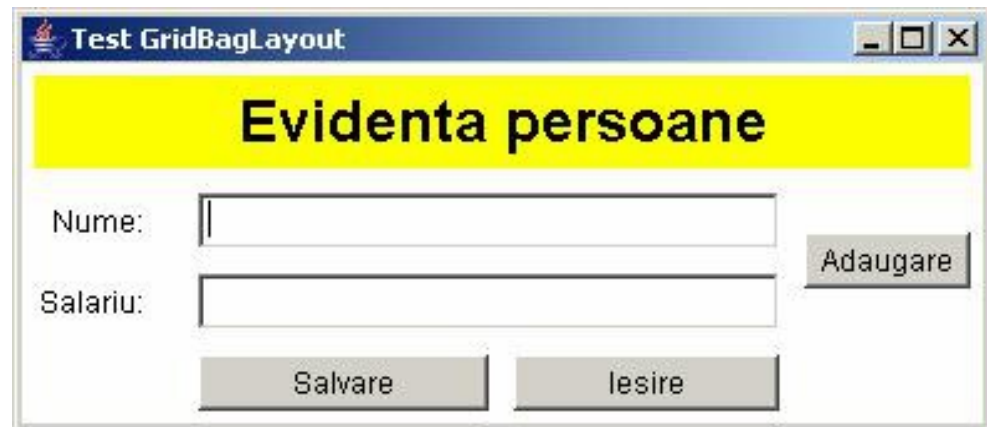
```
GridBagLayout gridBag = new GridBagLayout();  
container.setLayout(gridBag);
```

```
GridBagConstraints c = new GridBagConstraints();  
//Define the constraints  
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);
```



The screenshot shows a Java Swing window titled "Test GridBagLayout". Inside the window is a form titled "Evidenta persoane" on a yellow background. Below the title, there are two text input fields. The first field is labeled "Nume:" and the second is labeled "Salariu:". To the right of the "Salariu:" field is a button labeled "Adaugare". At the bottom of the form are two buttons labeled "Salvare" and "Iesire".

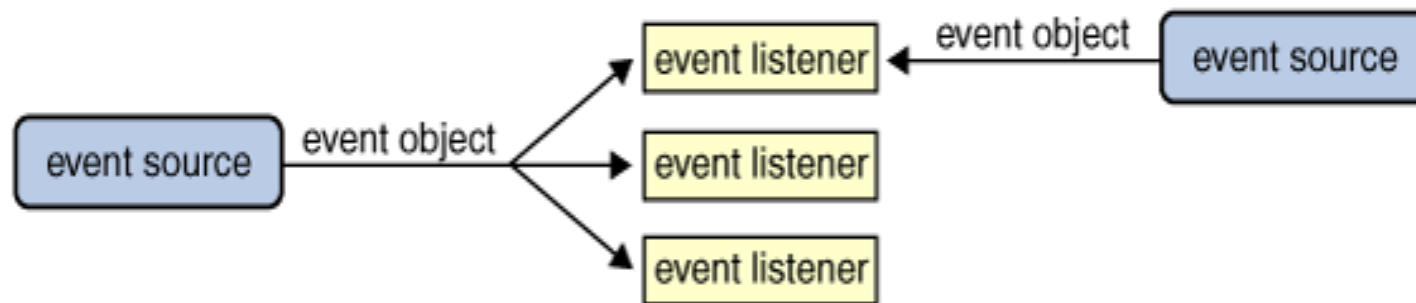
User Interactions

Event-Driven Programming

Event: clicking a button, altering the text, checking an option, closing a frame, etc.

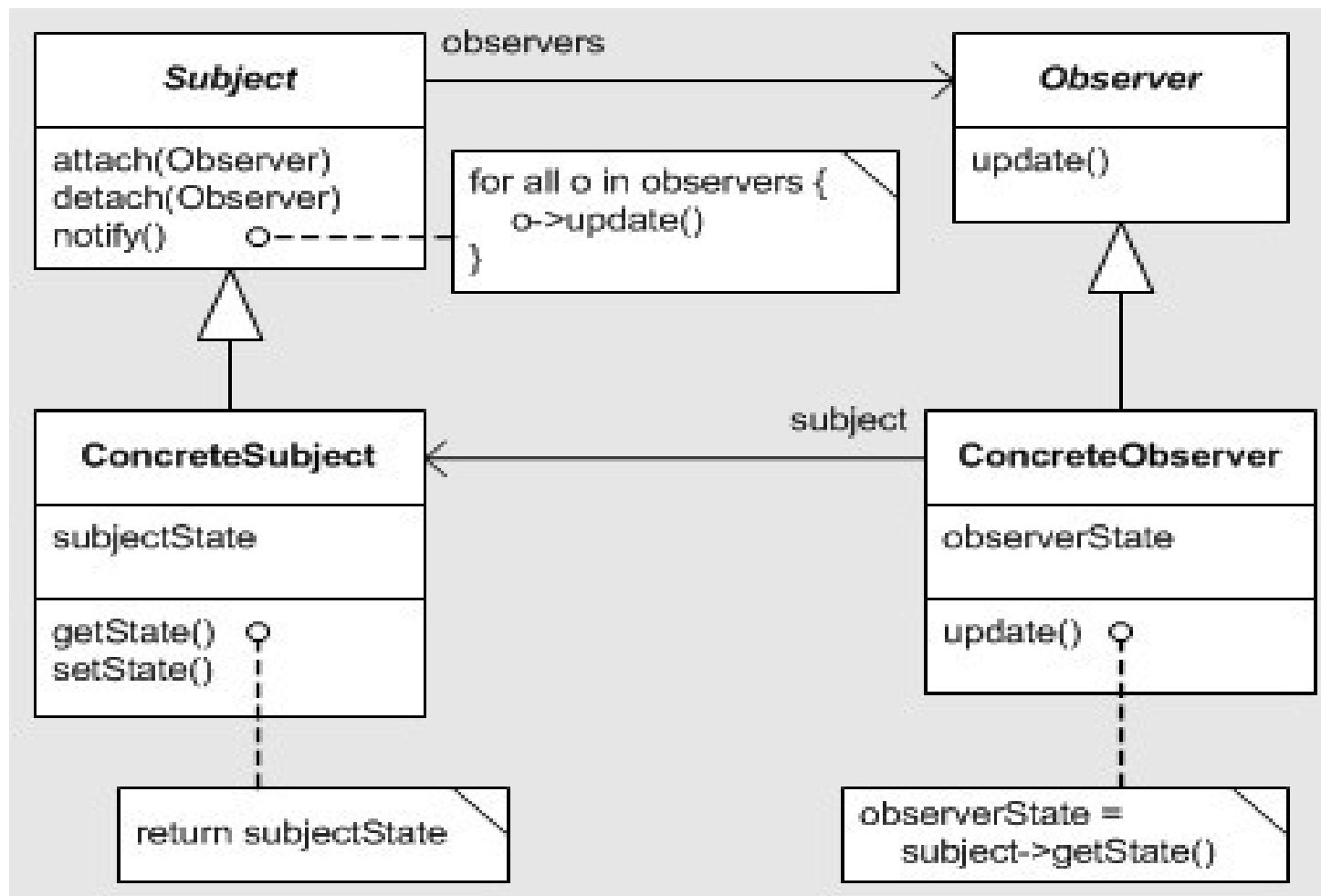
Source: the component that generates an event.

Listener: the responsible for receiving and handling (consuming) events.



Observer Design Pattern

Observing the state of an entity within a system
(*Publish-Subscribe*)



Button - ActionEvent - ActionListener

```
class MyFrame extends Frame {
    public MyFrame ( String title ) {
        super (title);
        setLayout (new FlowLayout ());
        setSize (200, 100) ;
        Button b1 = new Button ("OK");
        Button b2 = new Button ("Cancel");
        add(b1); add(b2);
        MyButtonListener listener = new MyButtonListener (this);
        b1.addActionListener ( listener );
        b2.addActionListener ( listener );
        // The events generated by the two buttons
        // are intercepted by the listener object
    }
}

class MyButtonListener implements ActionListener {
    private MyFrame frame;
    public MyButtonListener (MyFrame frame) {
        this.frame = frame;
    }
    // ActionListener interface has only one method
    public void actionPerformed (ActionEvent e) {
        frame.setTitle ("You pressed the button " + e.getActionCommand());
    }
}
```


Using Anonymous Classes

```
class MyFrame extends Frame {  
    public MyFrame ( String title ) {  
        ...  
        button.addActionListener( new ActionListener() {  
            @Override  
            public void actionPerformed(ActionEvent e) {  
                MyFrame.this.setTitle(  
                    "You pressed the button " + e.getActionCommand());  
            }  
        });  
        ...  
    }  
}
```

Using Lambda Expressions

```
...  
button.addActionListener( (ActionEvent e) -> {  
    MyFrame.this.setTitle(  
        "You pressed the button " + e.getActionCommand());  
    });  
...  
}
```

Using Method References

```
class MyFrame extends Frame {  
    public MyFrame ( String title ) {  
        ...  
        button.addActionListener( this::onButtonPressed );  
        checkbox.addItemListener( this::onItemChanged );  
        ...  
    }  
  
    //Your own, suggestively called, methods  
  
    private void onButtonPressed(ActionEvent e) {  
        this.setTitle("You pressed the button");  
    }  
  
    private void onItemChanged(ItemEvent e) {  
        this.setTitle("Checkbox state: " + check.getState());  
    }  
}
```

Event Types

Low-level	Semantic
ComponentEvent <i>hiding, moving, resizing, displaying components</i>	ActionEvent <i>pressing a button, pressing 'enter' in a text editing component, etc.</i>
ContainerEvent <i>adding, removing components in/from a container</i>	AdjustmentEvent <i>adjusting the value of a scrollbar, etc.</i>
FocusEvent <i>getting, losing the focus</i>	ItemEvent <i>changing the state of a component: selecting some items in a list, selecting or deselecting a checkbox, etc.</i>
KeyEvent <i>pressing, releasing a key</i>	TextEvent <i>changing the text in a component</i>
MouseEvent <i>mouse clicking, dragging, etc.</i>	. . .
WindowEvent <i>frame minimizing, resizing, etc.</i>	

Component-Listener Relationship

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

Handler Methods

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)`

...

...

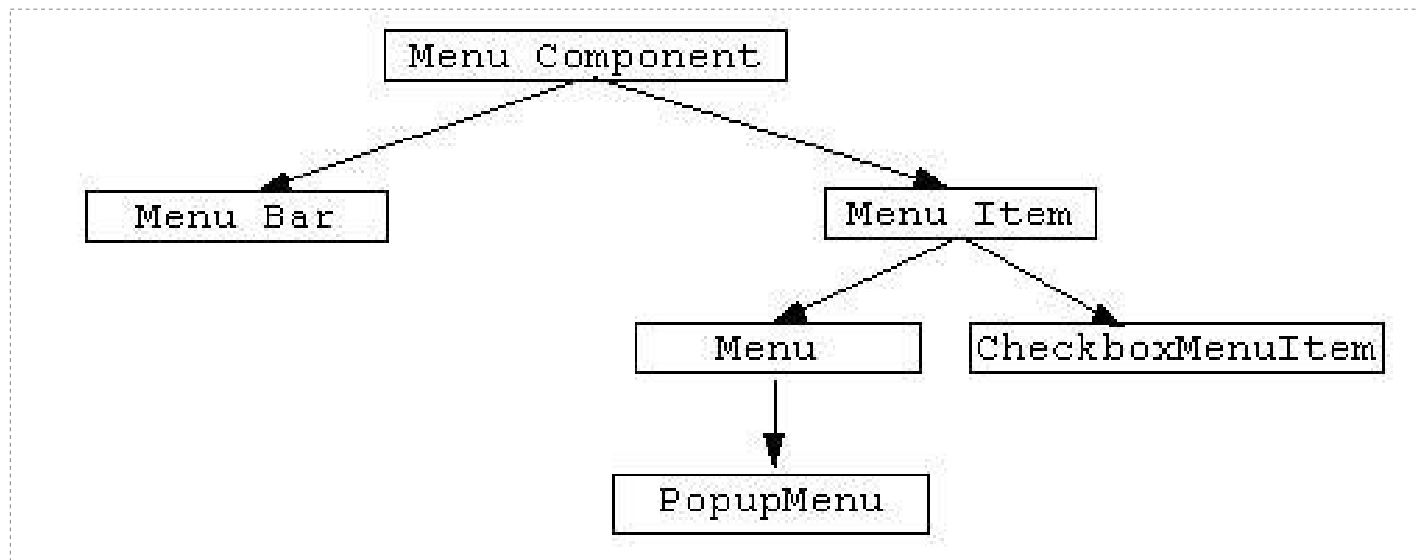
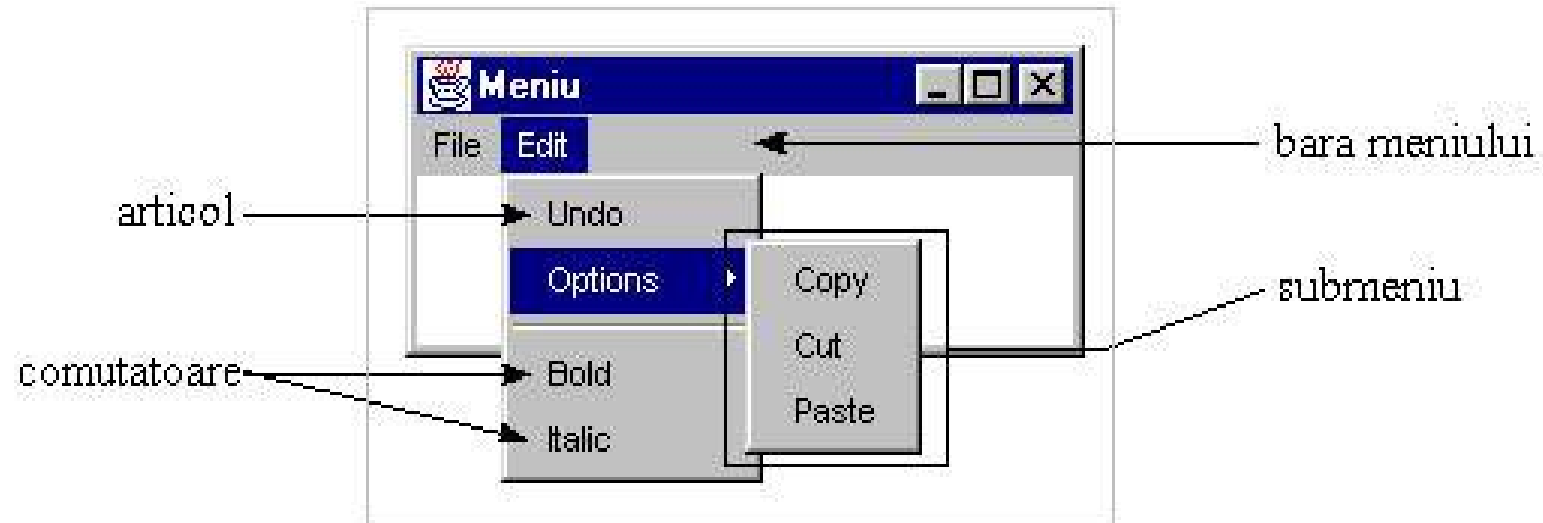
Using Adapters

```
class MyFrame extends Frame implements WindowListener {
    public MyFrame (String titlu) {
        super (titlu);
        this.addWindowListener(this);
    }
    // We are interested only in one of method of WindowListener
    public void windowOpened ( WindowEvent e) {}
    public void windowClosing ( WindowEvent e) {
        // Terminate the 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) {}
}
```

An **adapter** class provides the default implementation of all methods in an event listener interface.

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

Menus



Swing

- **Extends** the core concepts and mechanisms of AWT; *we still have components, containers, layout managers, events and event listeners.*
- **Replaces completely** the AWT component set, providing a new set of components, capable of sorting, printing, drag and drop and other “cool” features.
- Brings **portability** to the GUI level; no more *native peers*, all components are “pure”.
- Based on **Separable Model-and-View** design pattern.
- **"Component Oriented Programming"**

Java Foundation Classes

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

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

Swing Components

- **Atomic Components**

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

- **Complex Components**

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

- **Text Editing Components**

JTextField, JFormattedTextField, JPasswordField, JTextArea, JEditorPane, JTextPane

- **Menus**

JMenuBar, JMenu, JPopupMenu, JMenuItem, JCheckboxMenuItem, JRadioButtonMenuItem

- **Intermediate Containers**

JPanel, JScrollPane, JSplitPane, JTabbedPane, JDesktopPane, JToolBar

- **High-Level Containers**

JFrame, JDialog, JWindow, JInternalFrame, JApplet



Similarities and Differences with AWT

"J" Convention

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

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

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

New Layout Managers

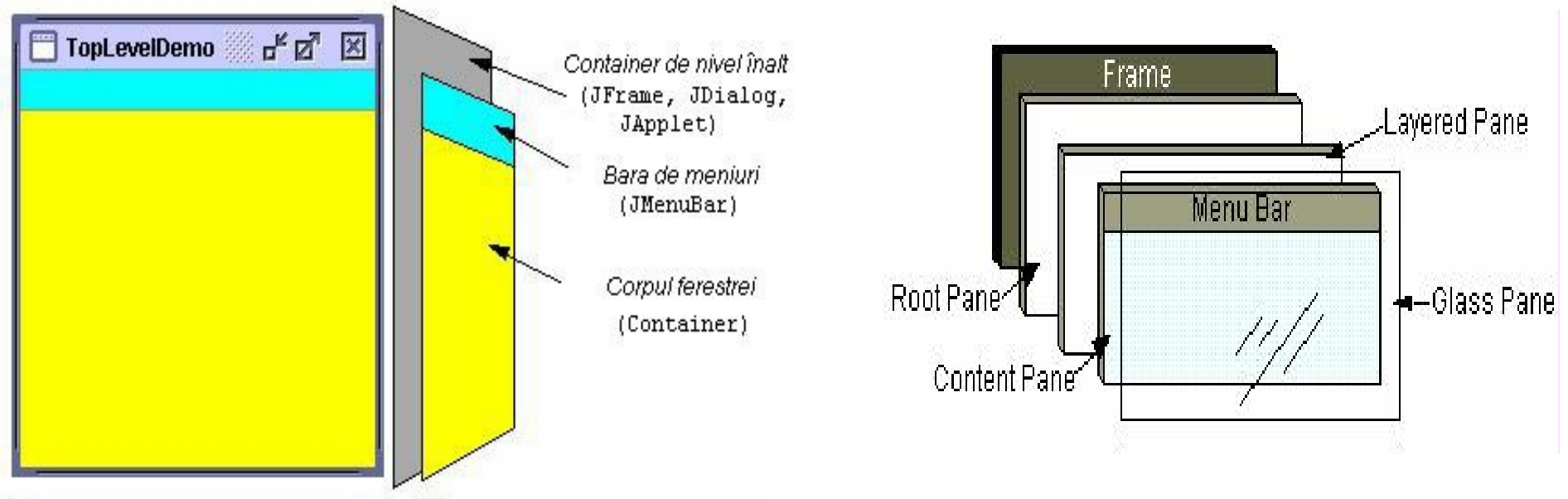
`BoxLayout`, `SpringLayout`, `GridLayout`, `OverlayLayout`, etc.

HTML Aware Components

`JButton simple = new JButton("Dull text");`

`JButton html = new JButton("<html><u>Cool</u> <i>text</i></html>");`

Swing *JFrames*



```
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"));
```

Internal Frames

GUI applications can be designed either as:

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

Multiple-document interface (MDI) applications enable you to display multiple documents at the same time, with each document displayed in its own window.

JInternalFrame

DesktopPane



JComponent

JComponent is the base class for all Swing components, except top-level containers: JFrame, JDialog, JApplet.

JComponent extends Container

- ★ Support for **tool tips** - `setToolTip`
- ★ Support for **borders** - `setBorder`
- ★ Enhanced support for **sizing and positioning**
`setPreferredSize, ...`
- ★ **Opacitiy** control - `setOpaque`
- ★ **Keyboard bindings**
- ★ “Pluggable” **look and feel**
- ★ Double-Buffering, Support for accessibility, etc.

Swing Architecture

Swing architecture is “rooted” in the MVC design:

- *Model* – the data for the application
- *View* – the visual representation of the data
- *Controller* – takes user input on the view and translates that to changes in the model.

Separable Model Architecture

Model + (Presentation, Control)

Presentation - Model

Component Class	Interface describing the Model
JList	ListModel ListSelectionModel
JTable	TableModel TableColumnModel ListSelectionModel
JTree	TreeModel TreeSelectionModel
JEditorPane JTextPane JTextField	Document
...	...

Creating a model involves either implementing the interface or extending an abstract support class

JList - **ListModel**, **DefaultListModel**, **AbstractListModel**

Example: *JList*

Components are created based on a model:

The model could be a standard data structure:

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

```
JList list = new JList(elements);
```

or a *model instance* specific to that component:

```
DefaultListModel model = new DefaultListModel();
```

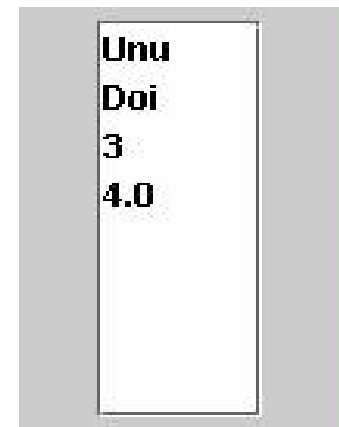
```
model.addElement("Unu");
```

```
model.addElement("Doi");
```

```
model.addElement(new Integer(3));
```

```
model.addElement(4.0);
```

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



Example: *JTable*

```
class MyTableModel extends AbstractTableModel {  
    private String[] columns = {"Nume", "Varsta", "Student"};  
    private Object[][] elements = {  
        {"Ionescu", new Integer(20), Boolean.TRUE},  
        {"Popescu", new Integer(80), Boolean.FALSE}};  
  
    public int getColumnCount() {  
        return columns.length;  
    }  
  
    public int getRowCount() {  
        return elements.length;  
    }  
  
    public Object getValueAt(int row, int col) {  
        return elements[row][col];  
    }  
  
    public String getColumnName(int col) {  
        return columns[col];  
    }  
  
    public boolean isCellEditable(int row, int col) {  
        // Doar numele este editabil  
        return (col == 0);  
    }  
}
```

Nume	Varsta	Student
Ionescu	20	true
Popescu	80	false

Customizing the View

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

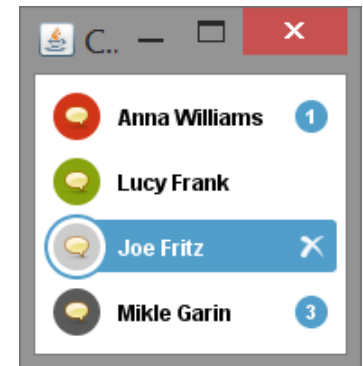
First Name	Last Name	Favorite Color	Favorite Movie	Favorite Number	Favorite Food
Mike	Albers	Green	Brazil	44	
Mark	Andrews	Blue	Curse of the Demon	3	
Brian	Beck	Black	The Blues Brothers	2.718	
Lara	Bunni	Red	Airplane (the whol...	15	
Roger	Brinkley	Blue	The Man Who Kne...	13	
Brent	Christian	Black	Blade 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

The *CellRenderer* Concept

A **renderer** is responsible for displaying the content of components, for example: the appearance of items in a list.

```
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());
```



The *CellEditor* Concept

An **editor** is responsible for the editing of individual items of components, such as the cells in a table.

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

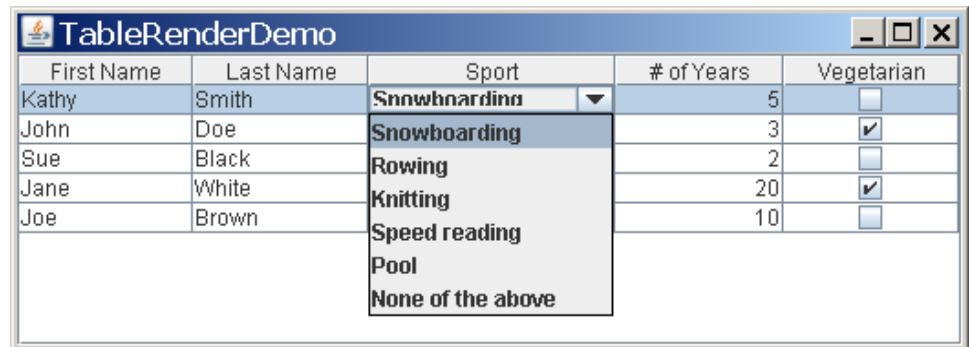
```
    public Component getTableCellEditorComponent(...) {
```

```
        //Returns the component
```

```
        //responsible for editing
```

```
        ...
```

```
    }
```



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() {
```

```
        // Returns the cell value
```

```
        ...
```

```
    }
```

```
}
```

Swing Containers

❏ High-Level Containers

JFrame, JDialog, JApplet

❏ Intermediate Containers

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



Look and Feel

The architecture of Swing is designed so that you may change the "look and feel" (L&F) of your application's GUI. "**Look**" refers to the appearance of GUI widgets and "**feel**" refers to the way the widgets behave.

- `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>