PROGRAMMING III
JAVA LANGUAGE

COURSE 10

PREVIOUS COURSE CONTENT

- **Graphical User Interfaces**

- **Abstract Windows Toolkit**
  - Components
  - Containers
  - Layout Managers
  - Action Management
  - Drawing Components

COURSE CONTENT

- Introduction
- Arhitecture
- Root pane container model
- Swing threads
- Components

AWT - ABSTRACT WINDOWS TOOLKIT

- **First Java API used for building GUI applications**

- **Offers**
  - A robust mechanism for event handling
  - Layout managers

- **Disadvantages**
  - Portability
    - Limitation regarding the way in which components are rendered (ex. on Windows OS uses DirectX to render components)
  - Facilities
    - does not have support for some components like icons and tooltips

SWING

- A set of custom graphical components which look-and-feel is settled at runtime
- First official version in 1998
- Included in Java distributions > 1.1.5
- Corrects AWT problems
- The components names start with J

NEWS SWING VS. AWT

- **Actions**
  - synchronized actions model
- **Tooltips**
- **Timer**
  - Useful for animations
- **Event dispatcher thread**
- **Client properties**
- **Keyboard shortcuts**
  - Focus management, mnemonics and menu accelerators, keymaps
- **Borders**
- **Icons**
- **Cursors**
- **Double Buffering**
- **New layout managers**
  - box, spring, grouped
- **Simple dialog windows**
- **Components**
  - JFileChooser, JColorChooser, JTable (TableModel), ...

COMPONENTS AWT/SWING

AWT COMPONENTS

- **Heavyweight**
  - Associated with native components named peers
  - Same behavior but the look and feel is platform dependent
  - package: `java.awt`

SWING COMPONENTS

- **Lightweight**
  - The components are rendered by JVM (Java Virtual Machine)
  - package: `javax.swing`

COMPONENTS AWT/SWING

LIGHTWEIGHT

- Can contain transparent pixels
- Can have a different form from rectangular because they contain transparent pixels
- Mouse events are treated through parent components

HEAVYWEIGHT

- Are opaque
- Are rectangular
- Mouse events are not treated through parent components

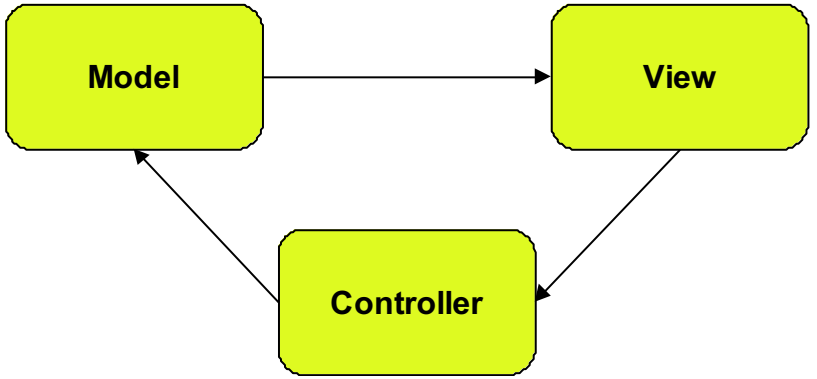
Remark: In Java version < 1.7 when a lightweight component was above a heavyweight component , the heavyweight component was always render above

COURSE CONTENT

- Introduction
- **Arhitecture**
- Root pane container model
- Swing threads
- Components

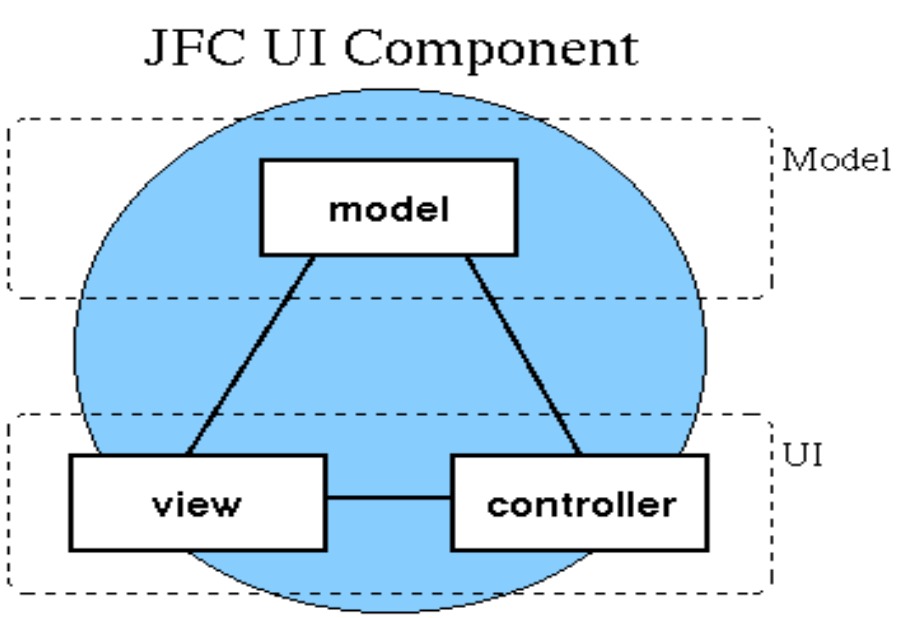
MODEL VIEW CONTROLLER

- **Architectural pattern which separates the presentation to logical level**
- **Swing components that use MVC pattern**
  - Model
    - Encapsulated date state for each component
  - View
    - The way in which the component is visible on screen
  - Controller
    - the way in which the component interacts with events (keyboard, mouse, focus, etc)



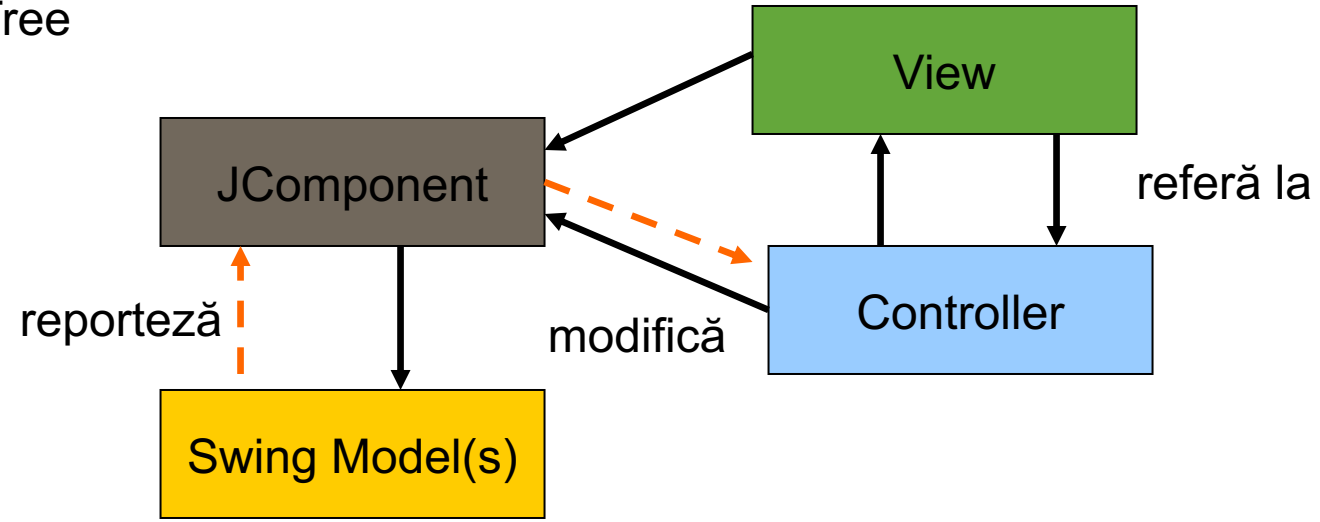
MODEL VIEW CONTROLLER

- **The pattern is a little different from the classical one**
  - Doe not exists a clear separation between View and Controller
  - The separation from Model is clear



MODEL VIEW CONTROLLER

- **MV/C**
  - For more simple components the default model is used
- **MVC**
  - Complex components
    - JList
    - JTable
    - JTree



MODEL

- **Each component has its model class**
  - For JTable is Table Model
- **The model provides data to the component**
- **When the model is modified all listeners are notified by an event**
- **What models ca be used**
  - Default
  - Custom implementation, if it assures a more efficient data control inside the component

VIEW

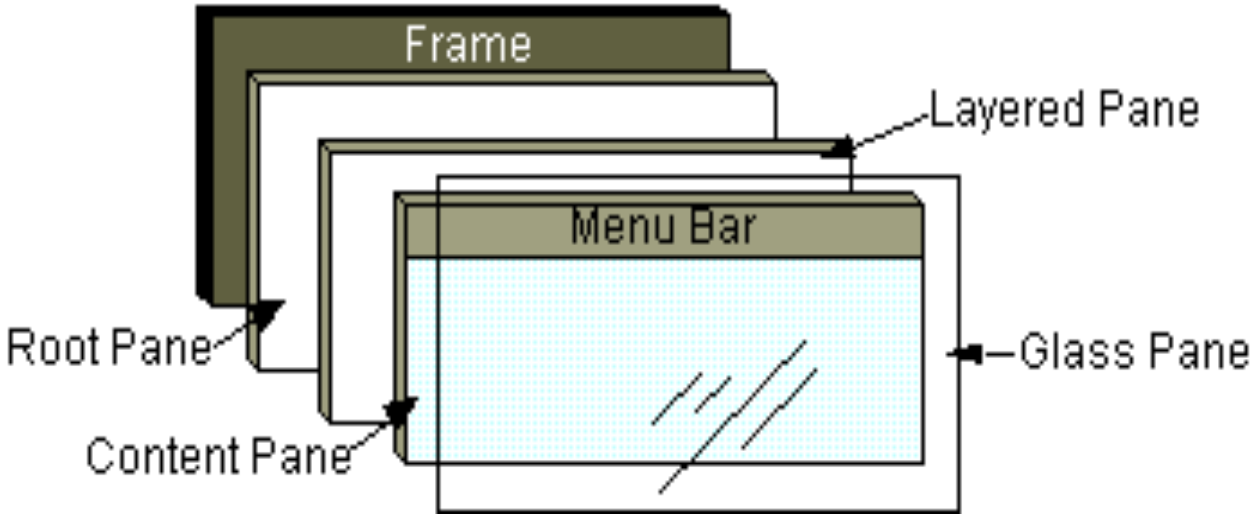
- **Each component has its own view**
  - Responsible with component rendering
  - Named UI Delegates
    - ButtonUI, SliderUI, etc
- **Some of the view properties can be found in component class**
  - Example
    - Font properties
    - Background properties
    - Size properties

COURSE CONTENT

- Introduction
- Arhitecture
- **Root pane container model**
- Swing threads
- Components

ROOT PANE

- **Manages**
  - Content pane
  - Menu bar
  - Other containers



COURSE CONTENT

- Introduction
- Arhitecture
- Root pane container model
- **Swing threads**
- Components

SWING THREADS

- **A well-written Swing program uses concurrency to create a user interface that never "freezes"**
  - the program is always responsive to user interaction, no matter what it's doing

- **Swing threads**
  - Initial threads
    - the threads that execute initial application code.
  - The event dispatch thread
    - where all event-handling code is executed. Most code that interacts with the Swing framework must also execute on this thread.
  - Worker threads
    - also known as background threads, where time-consuming background tasks are executed

- **The programmer does not need to provide code that explicitly creates these threads: they are provided by the runtime or the Swing framework**

INITIAL THREAD

- **What they do?**
  - Create a Runnable object that initializes the GUI and schedule that object for execution on the event dispatch thread
  - Once the GUI is created, the program is primarily driven by GUI events, each of which causes the execution of a short task on the event dispatch thread
  - Application code can schedule additional tasks on the event dispatch thread (if they complete quickly, so as not to interfere with event processing) or a worker thread (for long-running tasks)

INITIAL THREAD

- **Are created by invoking the methods**
  - `javax.swing.SwingUtilities.invokeLater`
    - simply schedules the task to an initial thread and returns
    - `public static void invokeLater( Runnable r)`
    - Example

```
Runnable doWorkRunnable = new Runnable() {
    public void run() { doWork(); }
};
SwingUtilities.invokeLater(doWorkRunnable);
```

  - `javax.swing.SwingUtilities.invokeAndWait`
    - waits initial thread for the task to finish before returning
    - `public static void invokeAndWait( Runnable r)`

EVENT DISPATCH THREAD

- **It is automatically started by JVM when an application contains graphical components**
  - Is a background thread used in Java to process events from the Abstract Window Toolkit (AWT) graphical user interface event queue
- **Responsible for method call**
  - `paint()`
  - `actionPerformed()`
  - all other methods that are used for events handling
- Allows the safe modification of the components

WORKER THREADS AND SWINGWORKER

- **Execute a long-running task**
  - uses one of the worker threads (background threads)
  - `javax.swing.SwingWorker. SwingWorker`

UNIQUE THREAD RULE

- **AWT components methods are thread safe**
  - the concurrent access to the components doe not affect their state
- **Swing components are NOT thread safe**
  - Once a Swing component has been realized, all code that might affect or depend on the state of that component should be executed in the event-dispatching thread
- **Some methods like `repaint()`, `revalidate()`, `inavalidate()`, methods that modifies the listeners list are thread safe being handled by *event thread***

UNIQUE THREAD RULE

- **Resolves the problem of modification of the components by an external execution thread**
- **Example:**

```
public class MyApplication {
public static void main(String[] args) {
    JFrame f = new JFrame("Labels");
    // Add components to
    // the frame here... M   f.pack();
    f.show();
    // Don't do any more GUI work here...
    }
}
```

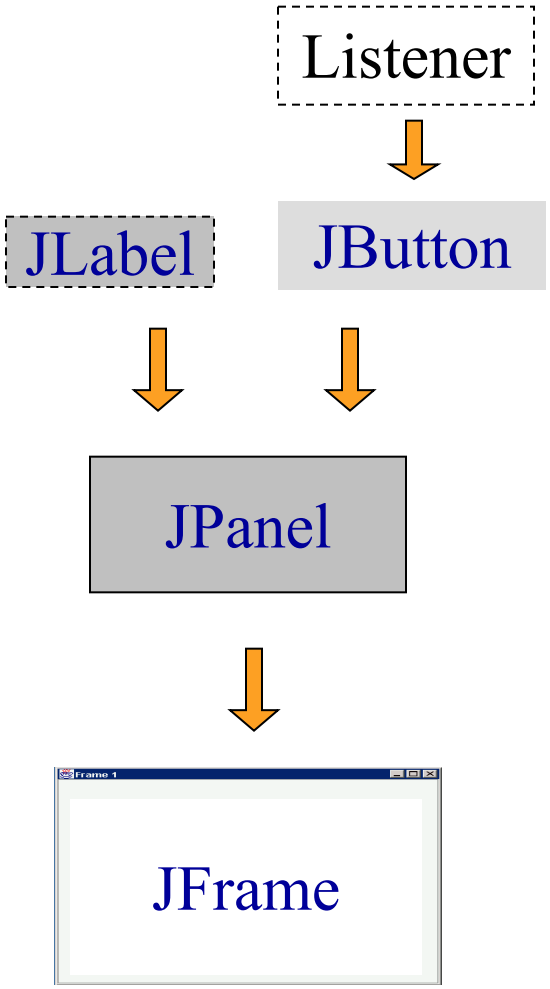
  - the code runs on the "main" thread.
  - `f.pack()` call realizes the components under the `JFrame`
  - the components in the GUI are shown with the `setVisible()` (or show) call. Technically the setVisible call is unsafe, because the components have already been realized by the pack call. However, because the program doesn't already have a visible GUI, it's exceedingly unlikely that a paint request will occur before `setVisible()` returns.
  - The main thread executes no GUI code after the setVisible call. This means that all GUI work moves from the main thread to the event-dispatching thread, and the example is, in practice, thread safe.

COURSE CONTENT

- ❑ Introduction
- ❑ Arhitecture
- ❑ Root pane container model
- ❑ Swing threads
- ❑ **Components**

HOW TO CREATE AN APPLICATION

- **Creation**
  - `Frame/JFrame`
  - `Panel/JPanel`
  - Components
  - Listeners
- **Adding**
  - Listeners at components
  - Components in panels
  - Panels in frames



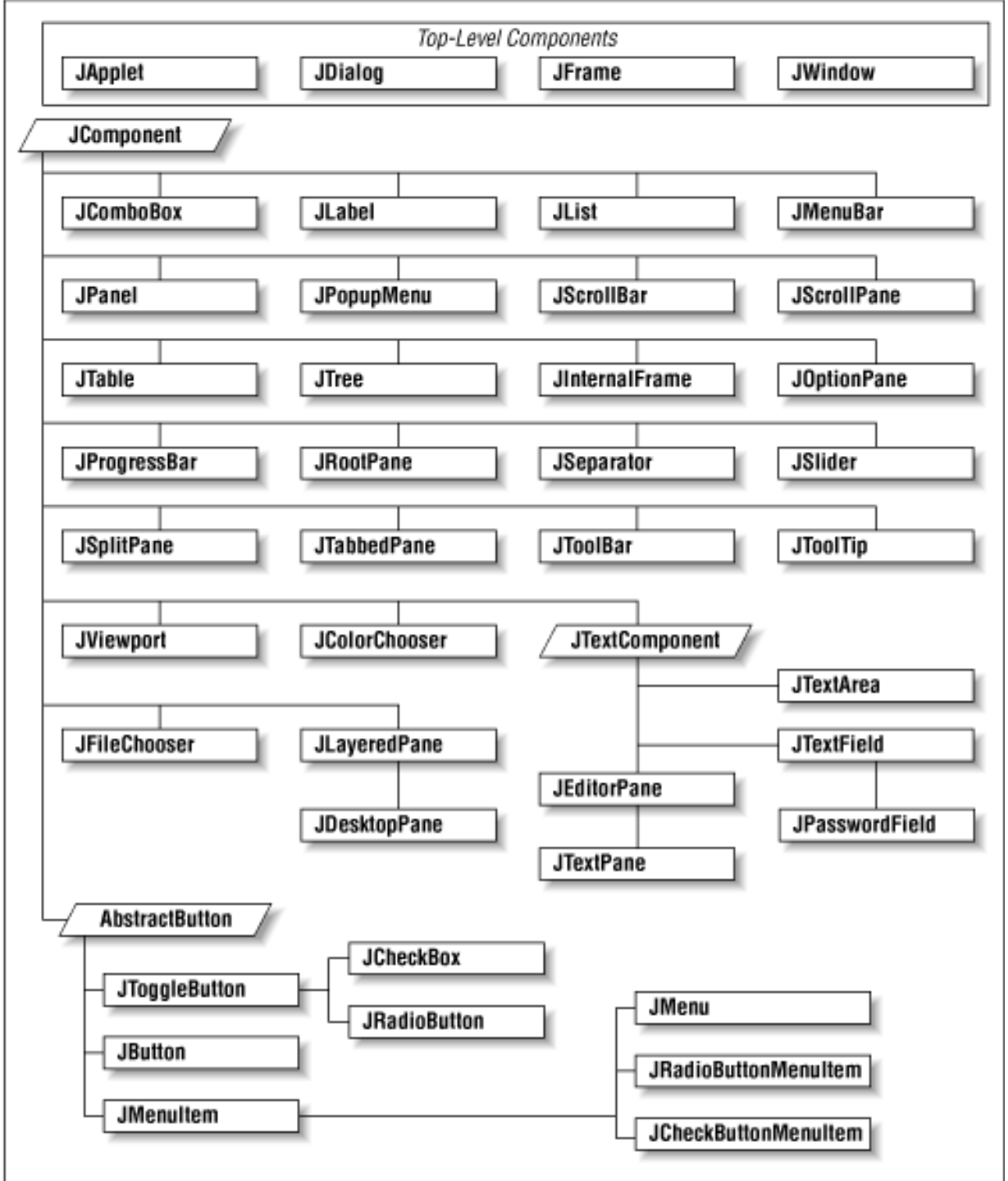
HOW TO CREATE AN APPLICATION

```
public class GUIExample extends JFrame {
        public GUIExample(String title) {
                super(title);
                //add components
                setVisible(true);
                pack();
        }

        public static void main (String args[]) {
                SwingUtilities.invokeLater( new Runnable(){
                        public void run() {
                                new GUIExample("Ex titlu");
                        }
                });
            }
}
```

Call constructor of JFrame and sets the application name

Make the application visible

Apply the layout(s) manager(s)

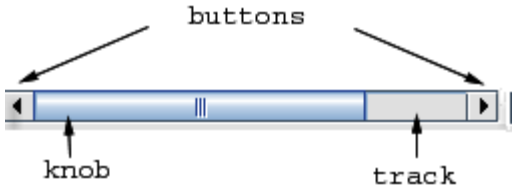

Top-Level Components
JApplet
JDialog
JFrame
JWindow
JComponent
JComboBox
JLabel
JList
JMenuBar
JPanel
JPopupMenu
JScrollBar
JScrollPane
JTable
JTree
JInternalFrame
JOptionPane
JProgressBar
JRootPane
JSeparator
JSlider
JSplitPane
JTabbedPane
JToolBar
JToolTip
JViewport
JColorChooser
JTextComponent
JTextArea
JFileChooser
JLayeredPane
JTextField
JEditorPane
JDesktopPane
JPasswordField
JTextPane
AbstractButton
JCheckBox
JToggleButton
JRadioButton
JMenu
JButton
JRadioButtonMenuItem
JMenuItem
JCheckButtonMenuItem

INTERMEDIATE CONTAINERS

- **Intermediate containers**
  - `JPane`
  - `JScrollPane`
  - `JSplitPane`
  - `JTabbedPane`
- **Used for components grouping**
- **Default layout `Flowyout`**

JPANE

- `JPane`
- **Base intermediary container for any graphical interface**
- **Most used to organize group of components**
- **Methods**
  - add layout manager
  - components management (add/remove)
  - adding border



JSCROLLPANE

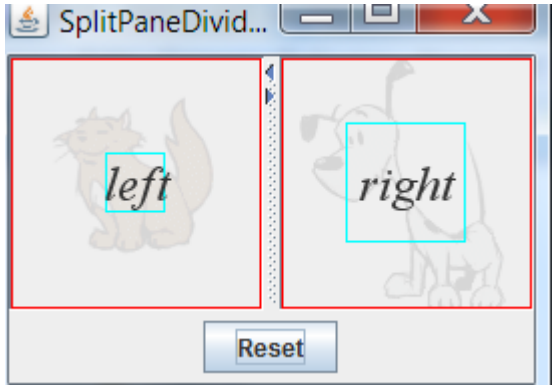
- **`JScrollPane`**
- **Allows other components to scroll inside a fix dimension area**
- **Example**

```
JTextArea textArea = new JTextArea(5, 5);

JScrollPane scrollableTextArea = new
                                    JScrollPane(textArea);
scrollableTextArea.setHorizontalScrollBarPolicy(
                    JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
scrollableTextArea.setVerticalScrollBarPolicy(
                    JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

add(scrollableTextArea);
```

JSPLITPANE

- **`JSplitPane`**

- **Contains to side by side panels separated by a divider, that allows simulating visualization of to components near each other**



- **Methods for adding components**
  - `setTopComponent()`
  - `setLeftComponent()`
  - `setBottomComponent()`
  - `setRightComponent()`

- **Example**

```
Panel panel1 = new JPanel();
JPanel panel2 = new JPanel();
JSplitPane splitPane = new JSplitPane(
     JSplitPane.HORIZONTAL_SPLIT, panel1, panel2);
splitPane.setDividerLocation(0.25);
```

JTABEDPANE

- `JTabedPane`
- A stack of components layered on many over layered layers
- Methods
  - `addTab()`
  - `removeTabAt()`
  - `setSelectedIndex()`
- Not present in AWT

JTABEDPANE

- **Example**

```
JTabbedPane tabbedPane = new JTabbedPane();
ImageIcon icon = createImageIcon("images/middle.gif");

JComponent panel1 = makeTextPanel("Panel #1");
tabbedPane.addTab("Tab 1", icon, panel1,"Does nothing");
tabbedPane.setMnemonicAt(0, KeyEvent.VK_1);

JComponent panel2 = makeTextPanel("Panel #2");
tabbedPane.addTab("Tab 2", icon, panel2, "Does twice as
                                          much nothing");
tabbedPane.setMnemonicAt(1, KeyEvent.VK_2);

tabbedPane.setSelectedIndex(2);
add(tabbedPane);
```

SIMPLE COMPONENTS

- Labels
- Buttons
- Borders
- Lists
- Drop down lists
- Spinner

LABELS

- **`JLabel`**
  - can render HTML code
- **Useful to display**
  - text
  - images
- **Example**

```
label1 = new JLabel("Image and Text", icon, JLabel.CENTER);
label1.setVerticalTextPosition(JLabel.BOTTOM);
label1.setHorizontalTextPosition(JLabel.CENTER);

label2 = new JLabel("Text-Only Label");

label3 = new JLabel(icon);

JLabel l = new JLabel("<html><center><b>Label</b>
     <br><font color=#ff00ff>HTML Format</font></html>");
```

BUTTONS

- `JButton`
- `JCheckbox`
- `JRadioButton`
- `JMenuItem`
- `JCheckBoxMenuItem`
- `JRadioButtonMenuItem`
- `JToggleButton`

BUTTONS

- **Can realize actions**
- **Can be grouped**
  - `ButtonGroup`
- **Can contain**
  - Images
  - HTML text
  - Mnemonics

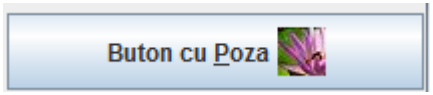
BUTTONS STATES

- Selected
- Pressed
- Rollover
- Armed
- Enabled

EXAMPLES

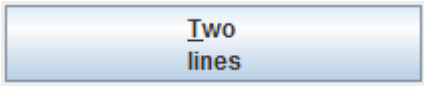
- **Simple buttons**

```
new JButton("Simple button")
```



- **Buttons with images**

```
btnPicture = new JButton("Button cu Poza");
btnPicture.setIcon(new ImageIcon(getImage(getCodeBase(),
                                 "../img/butoane/icon1.JPG")));
btnPicture.setHorizontalTextPosition(SwingConstants.LEFT);
btnPicture.setMnemonic(KeyEvent.VK_P);
```



- **Buttons using HTML**

```
new JButton("<html><b><u>T</u>wo</b><br>lines</html>"));
```

- **Adding listeners (behavior)**

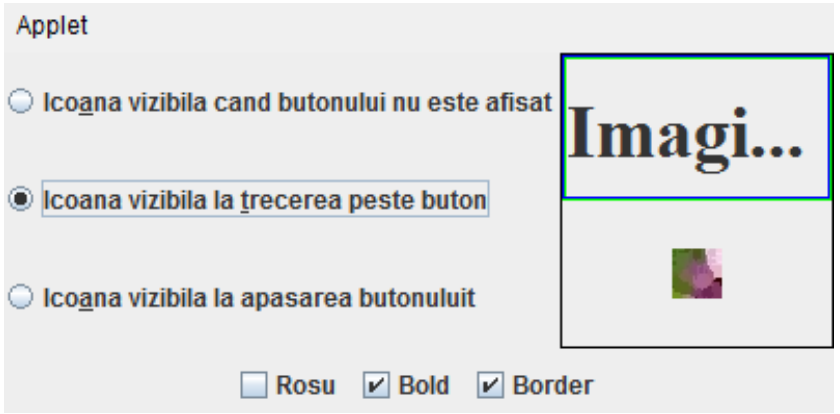
```
btnPicture.addActionListener(this);
```

EXAMPLES

- **Buttons rendered in different way from default behavior**

```
btn = new FancyButton(
    new ImageIcon(getImage(getCodeBase(), "../img/butoane/icon1.JPG")),
    new ImageIcon(getImage(getCodeBase(), "../img/icon2.JPG")),
    new ImageIcon(getImage(getCodeBase(), "../img/butoane/icon3.JPG")));

btn.setText("FancyButton");
btn.setIcon(new ImageIcon(
                getImage(getCodeBase(),"../img/butoane/icon1.JPG")));


public class FancyButton extends JButton {
    public FancyButton(Icon icon, Icon pressed, Icon rollover) {
            super(icon);
            setFocusPainted(false);,
            setRolloverEnabled(true);
            setRolloverIcon(rollover);
            setPressedIcon(pressed);
            setBorderPainted(false);
            setContentAreaFille(false);
    }
```

EXAMPLES



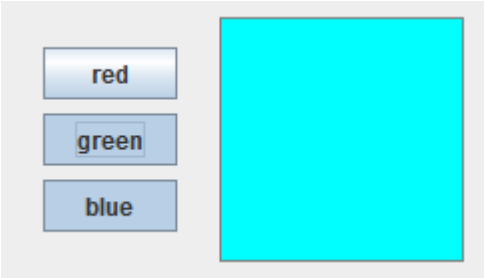
- **Radio/CheckboxButtons**

```
r1 = new JRadioButton(
                "Icoana vizibila cand butonului nu este afisat");
r1.setActionCommand("1");
r1.setSelected(true);
panelButone.add(r1);

r2 = new JRadioButton("Icoana vizibila la trecerea peste buton");
r2.setActionCommand("2");
panelButone.add(r2);

r3 = new JRadioButton("Icoana vizibila la apasarea butonuluit");
panelButone.add(r3);

ButtonGroup group = new ButtonGroup();
group.add(r1); group.add(r2); group.add(r3);

JCheckBox  cb1 = new JCheckBox("Rosu");
```

EXAMPLES

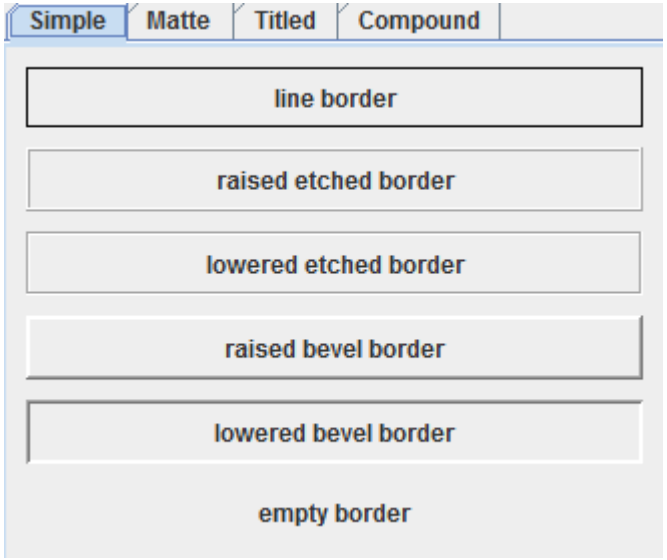
- **ToolgeButton**

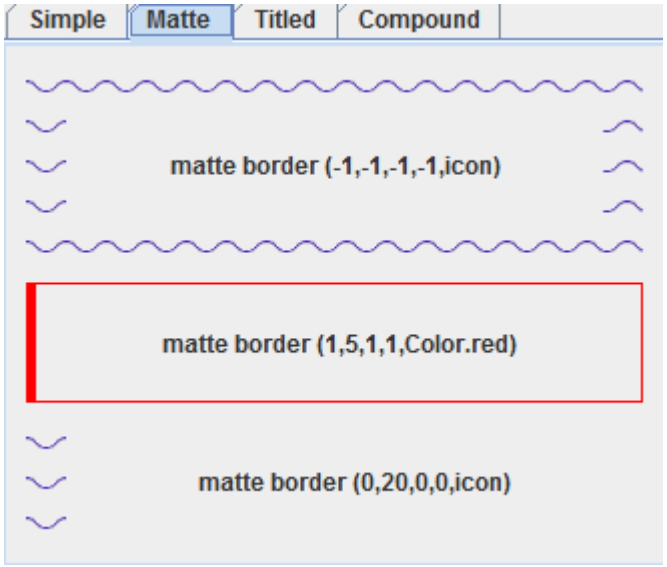
```
JToggleButton redButton = new JToggleButton("red");
```



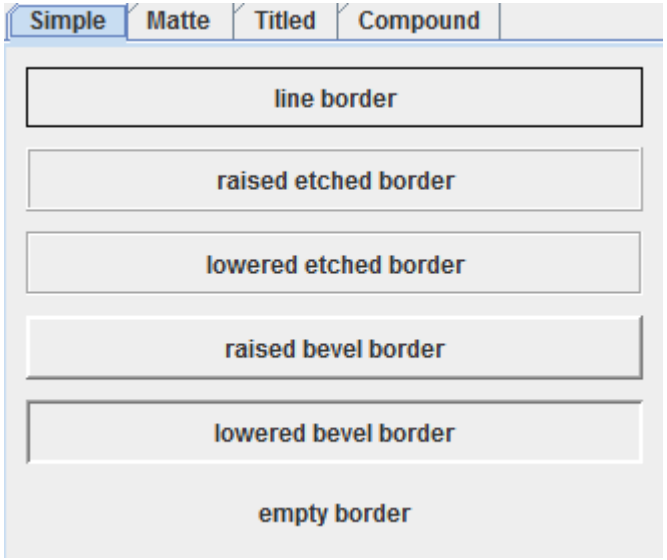
BORDERS

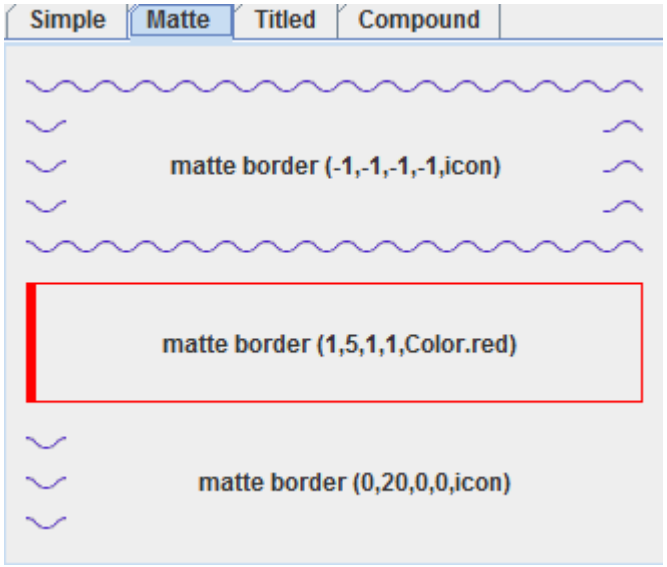
- **`Borders`**
  - Components derived from `JComponent`
  - cannot have listeners
- **Borders Types**
  - `CompoundBorder`
  - `EmptyBorder`
  - `EtchedBorder`
  - `LineBorder`
  - `MatteBorder`
  - `SoftBevelBorder`
  - `TitledBorder`
- **Border Factory**
- **`setBorder()`**

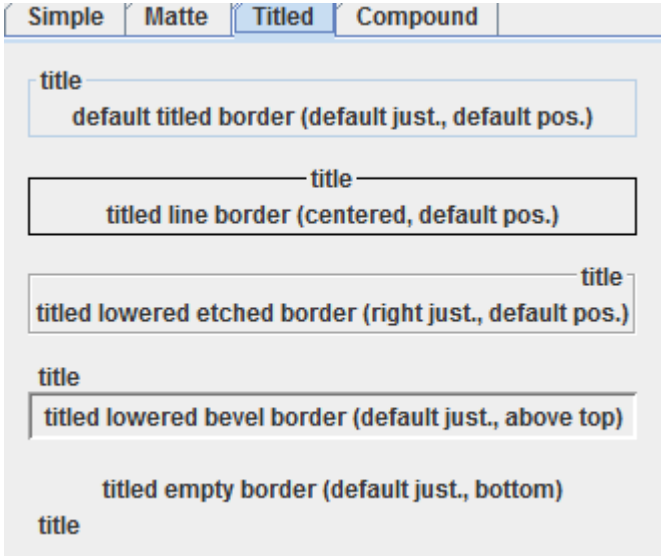


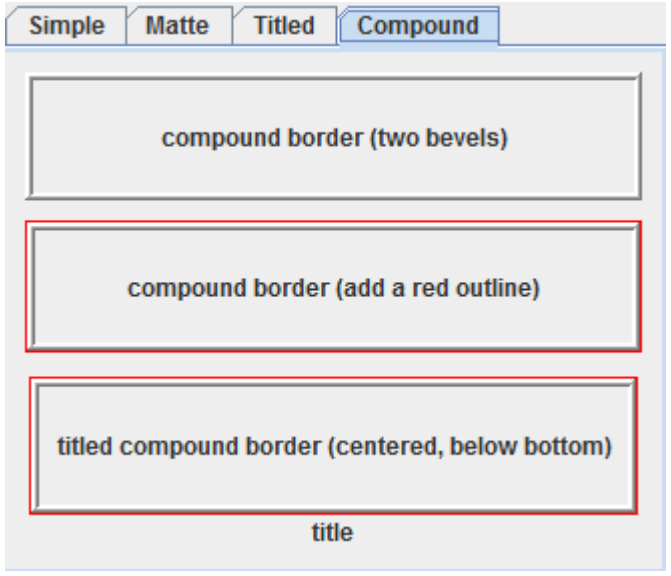


BORDERS


Simple
Matte
Titled
Compound
line border
raised etched border
lowered etched border
raised bevel border
lowered bevel border
empty border


Simple
Matte
Titled
Compound
matte border (-1,-1,-1,-1,icon)
matte border (1,5,1,1,Color.red)
matte border (0,20,0,0,icon)


Simple
Matte
Titled
Compound
title
default titled border (default just., default pos.)
title
titled line border (centered, default pos.)
title
titled lowered etched border (right just., default pos.)
title
titled lowered bevel border (default just., above top)
titled empty border (default just., bottom)
title


Simple
Matte
Titled
Compound
compound border (two bevels)
compound border (add a red outline)
titled compound border (centered, below bottom)
title

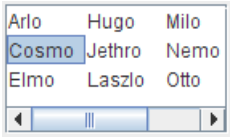
BORDERS

- **Example**

```
Border outline =
        BorderFactory.createLineBorder(Color.black);

JLabel northLabel = new JLabel("NORTH");

northLabel.setHorizontalAlignment(
                                SwingConstants.CENTER);

northLabel.setBorder(outline);
```

JLIST

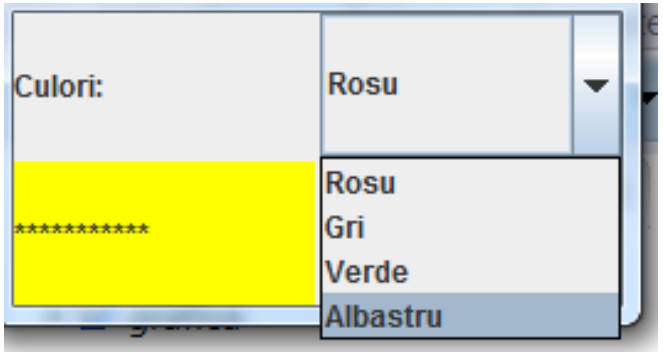
- **Allows selection one or more elements from a list**
  - `SINGLE_SELECTION`
  - `SINGLE_INTERVAL_SELECTION`
  - `MULTIPLE_INTERVAL_SELECTION`
- **Objects are displayed by calling `toString()` method**
  - exception: Icon class



- **Model**
  - `DefaultListModel`
  - `AbstractListModel`
- **Events**
  - `ListSelectionListener`

JCOMBOBOX

- **`JComboBox`**
  - Combines a button with a list (internal popup)
- **Data Models**
  - `DefaultComboBoxModel`
  - `MutableComboBoxModel`
    - allows operation of add/remove/update
- **Events**
  - `ChangeListener`
- **Types**
  - Combobox
  - Editable Combobox



EXAMPLE

```
final JComboBox<MyColor> cb = new JComboBox<MyColor>();

cb.addItem(new MyColor("Rosu", Color.red));
cb.addItem(new MyColor("Gri",Color.gray));
cb.addItem(new MyColor("Verde",Color.green));
cb.addItem(new MyColor("Albastru",Color.blue));

cb.addItemListener(new ItemListener() {
        @Override
        public void itemStateChanged(ItemEvent ie) {
                MyColor color = (MyColor) cb.getSelectedItem();
                lText.setBackground(color.color);
                lText.updateUI();
        }
});
```

JSPINNER

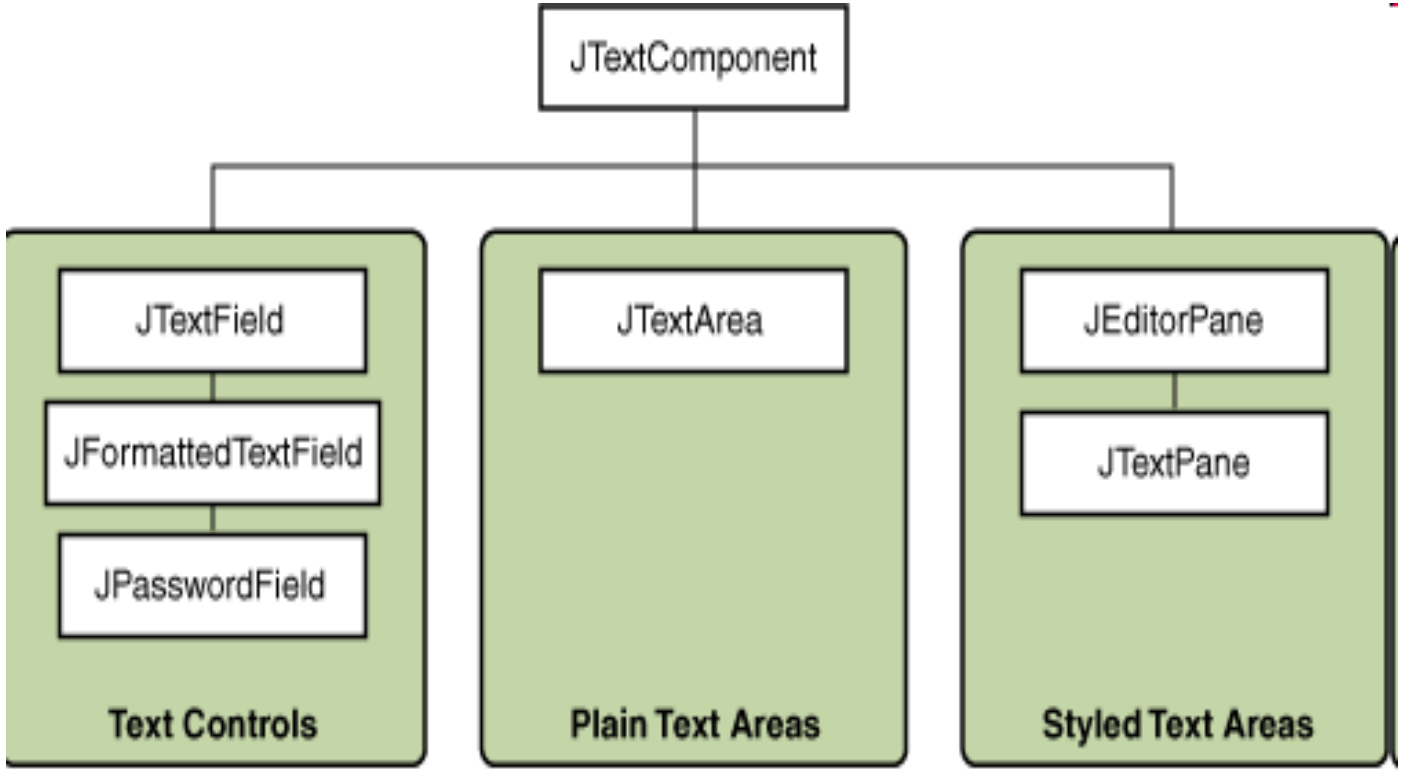
- **`JSpinner`**
- **Is a text line that allows selecting a value from a sequence of values**
- **Selection**
  - Using arrows
  - Adding directly a value
- **Models**
  - `SpinnerListModel`
  - `AbstractSpinnerModel`
  - `SpinnerDateModel`
  - `SpinnerModel`
  - `SpinnerNumberModel`

COMPLEX COMPONENTS

- Text comonents
- Tables
- Trees

TEXT COMPONENTS

- Display and allow text editing



TEXT COMPONENTS

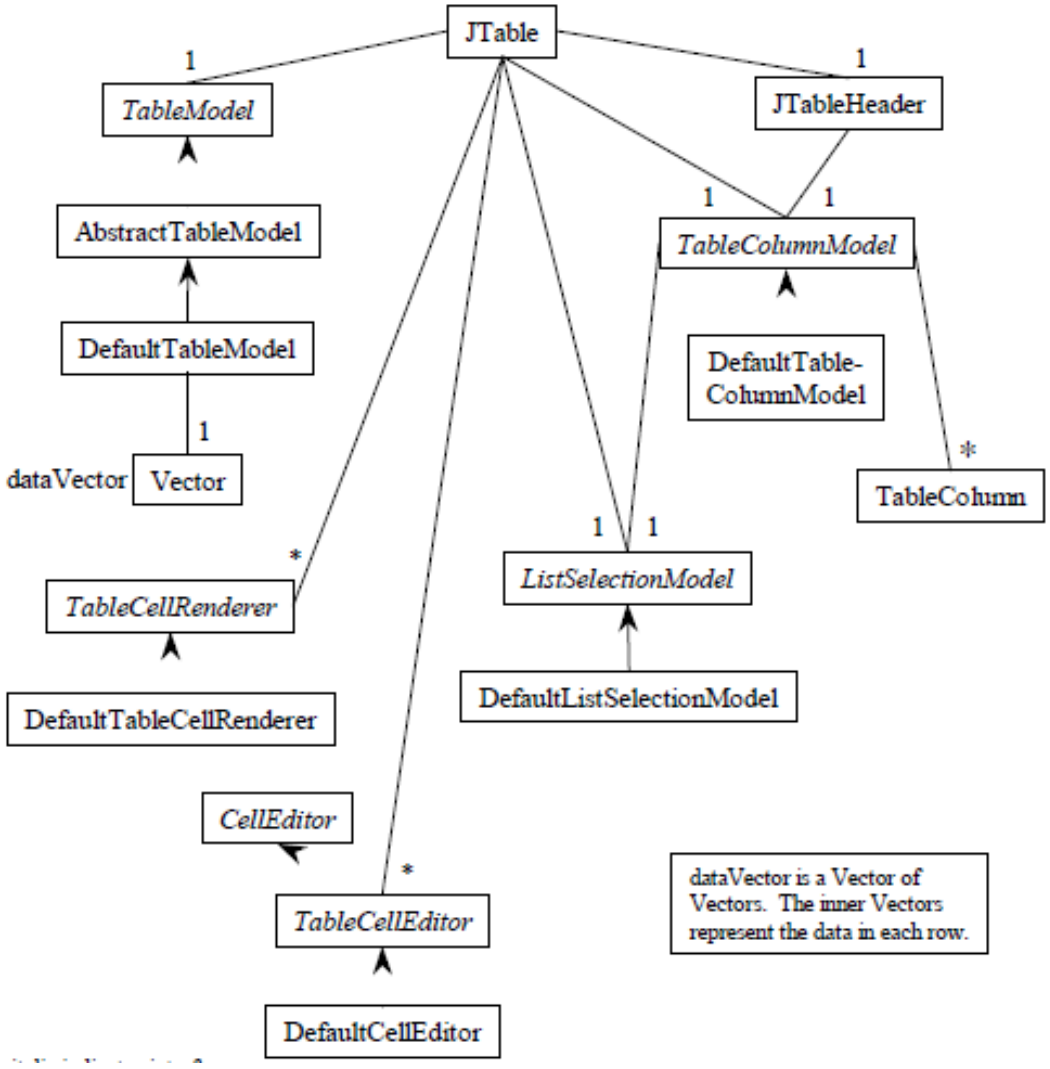
- **The content is managed by an instance of Document interface**
  - `PlainDocument`
  - `StyledDocument`

- **Copy/paste system is already implemented in superclass `JTextComponent`**
  - `copy()`
  - `cut()`
  - `select(int pozInt, int pozFin)`
  - `selectAll()`

TABLES

- `JTable`
- **Allows displaying data in a tabular format**
- **Properties**
  - The user can
    - select table lines, columns, table header
    - reorder the columns by table headers movement
    - column resize
    - edit cell values
    - sort columns based on content
    - filter columns based on content
  - The program can
    - modify cell values
    - add/delete/move columns
    - customize rendering mode of the table
    - customize edit mode of the table

TABLES. RELATED CLASSES



TABLES

- **`JTable` has many properties that can be customized, like cell rendering and editing, but also contains default values for them**
- **A `JTable` component is formed from**
  - Data lines
  - Data columns
  - Columns header
  - An editor if the cells will be editable
  - A `TableModel`, that is a subclass of `AbstractTableModel`, that will contain the data
  - A `TableColumnModel`, usually `DefaultTableColumnModel`, which controls the behavior of table columns and gives access to them
  - A `ListSelectionModel`, usually `DefaultListSelectionModel`, that contains information about the lines selected into the table
  - A `TableCellRenderer`, usually `DefaultTableCellRender`, that gives information about how the cells are rendered
  - A `MultipleTableColumns`, that contains information about each column
  - A `JTableHeader` that display the header

TABLEMODEL

- **`TableModel`**
  - Manages the dates displayed in the table
  - Methods
    - `Class getColumnClass(int columnIndex)`
      - Used by renderer and editor
    - `boolean isCellEditable(int rowIndex, int columnIndex)`
    - `Object getValueAt(int rowIndex, int columnIndex)`
    - `void setValueAt(Object aValue, int rowIndex, int columnIndex)`
      - caled by JTable at editing time
    - `void addTableModelListener (TableModelListener l)`
      - notifications for table data and structure

TABLEMODEL

- **`AbstractTableModel`**
  - defaults
    - Object class reported is `Object`
    - The columns have a default name if none is specified
    - The cells are not editable
  - the following methods have to be overwritten
    - `int getRowCount();`
    - `int getColumnCount();`
    - `Object getValueAt(int rowIndex, int columnIndex)`

TABLEMODEL

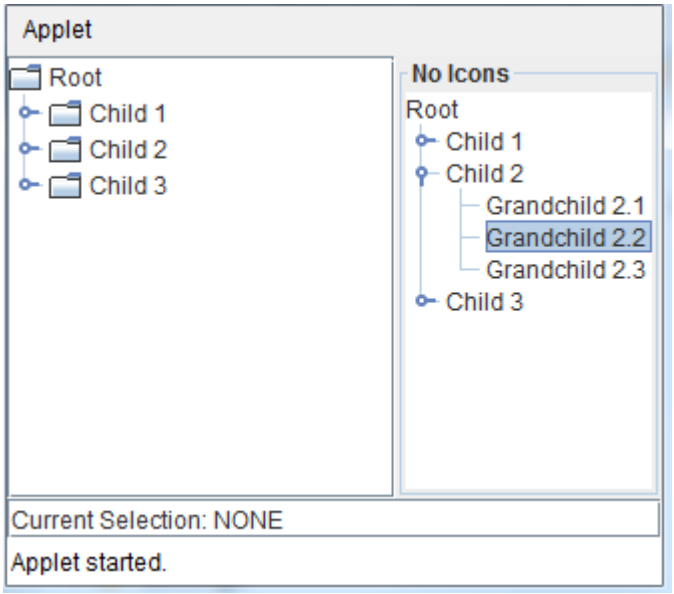
- **`DefaultTableModel`**
  - Stores the data in vectors
    - each column has its own vector
  - defaults
    - Object class reported is `Object`
    - The columns have a default name if none is specified
    - The cells are not editable

COLUMNMODEL

- **Contains information about**
  - Headers values
  - Dimensions
  - Rendering mode
  - Editing mode
  - Allows columns resize

JTREE

- **`JTree`**
  - Allows the visualization of tree structures
  - Has only one root node
  - Selection mechanism similar with `JList` selection mechanism



DEFAULT DIALOGUES

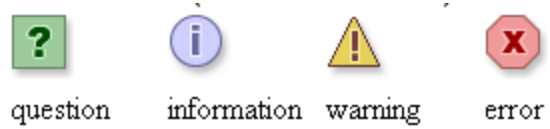
- `JOption Pane`
- `JFileChooser`
- `JColorChooser`

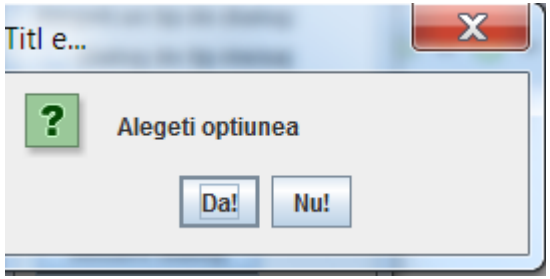
JOPTIONPANE

- `JOptionPane`
  - Easy way to create simple dialogs
    - Display a message
    - Ask a question
    - Input a value
  - Modal windows
    - Blocks application until an answer is given

JOPTIONPANE

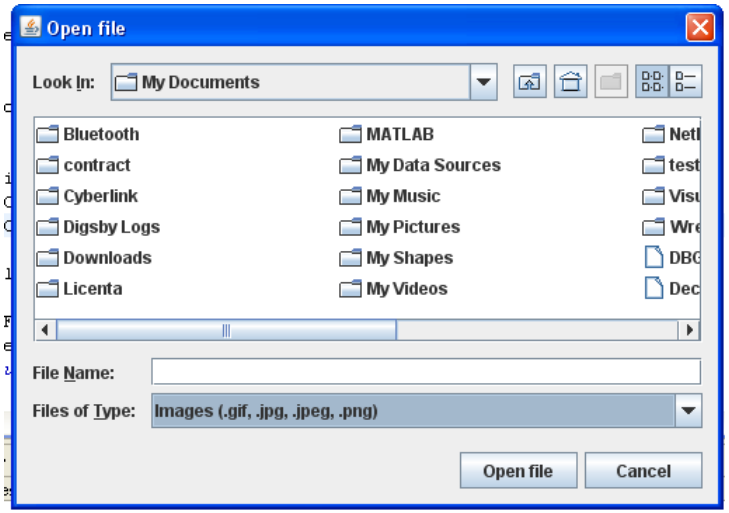
- **Types**
  - Message
    - `showConfirmDialog()`
  - Confirm
    - `showInputDialog()`
  - Input
    - `showMessageDialog()`
  - Option
    - `showOptionDialog()`





JFILECHOOSER

- **JFileChooser**
  - Component that allows navigation through file system
    - Can open files
    - Can save files



  - Example

```
JFileChooser files = new JFileChooser(DEFAULT_DIRECTORY);
int result = files.showSaveDialog(frame);
File f = files.getSelectedFile();
```

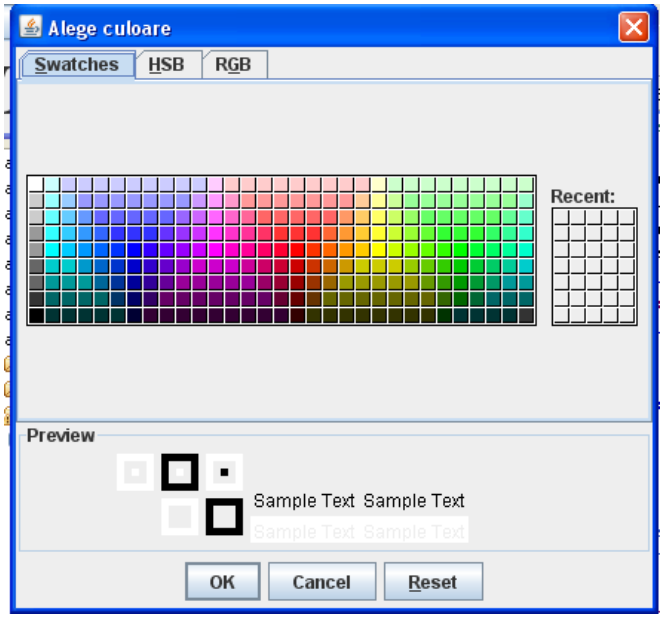
JFILECHOOSER

- **Files filters**
  - `FileNameExtensionFilter`
  - Create your own filter by extending the class `FileFilter`
  - Example

```
FileFilter filter = new FileNameExtensionFilter(
                        “Text files (*.txt)”, ” txt” );

files.addChoosableFileFilter(filter);
```

JCOLORCHOOSER

- **JColorChooser**
  - Allows color choosing
  - Available modes
    - Swatches
    - HSV —Hue-Saturation-Value
    - RGB —Red-Green-Blue
    - HSL —Hue-Saturation-Lightness
    - CMYK – Crayn-Magenda-Yellow-Black

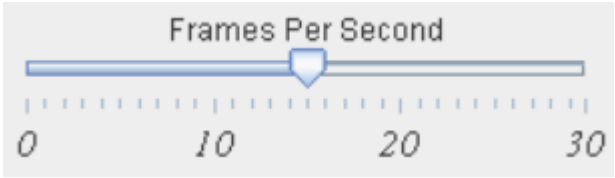


COMPONENTS FOR PROGRESS AND SCROLL

- `JSlider`
- `JScrollBar`
- `JProgressBar`
- `JToolTip`

SLIDER

- **JSlider**
  - Used when we have to chose values into a known numerical interval
  - Properties
    - orientation
    - intent
      - the number of missing values by using page up, page down keys
    - minorTickSpacing
    - majorTickSpacing
    - paintTicks
    - paintLabels
    - inverted



PROGRESSBAR

- `JProgressBar`
- Used to display the progress of a time consuming operation
- The activity monitorization is done by the method `setValue()`

JSCROLLBAR

- `JScrollBar`

- **Added to the components in order to scroll in a more facile way the content**

- **Properties**
  - Orientation
  - The place where the indicator is initially displayed
  - the size of the indicator

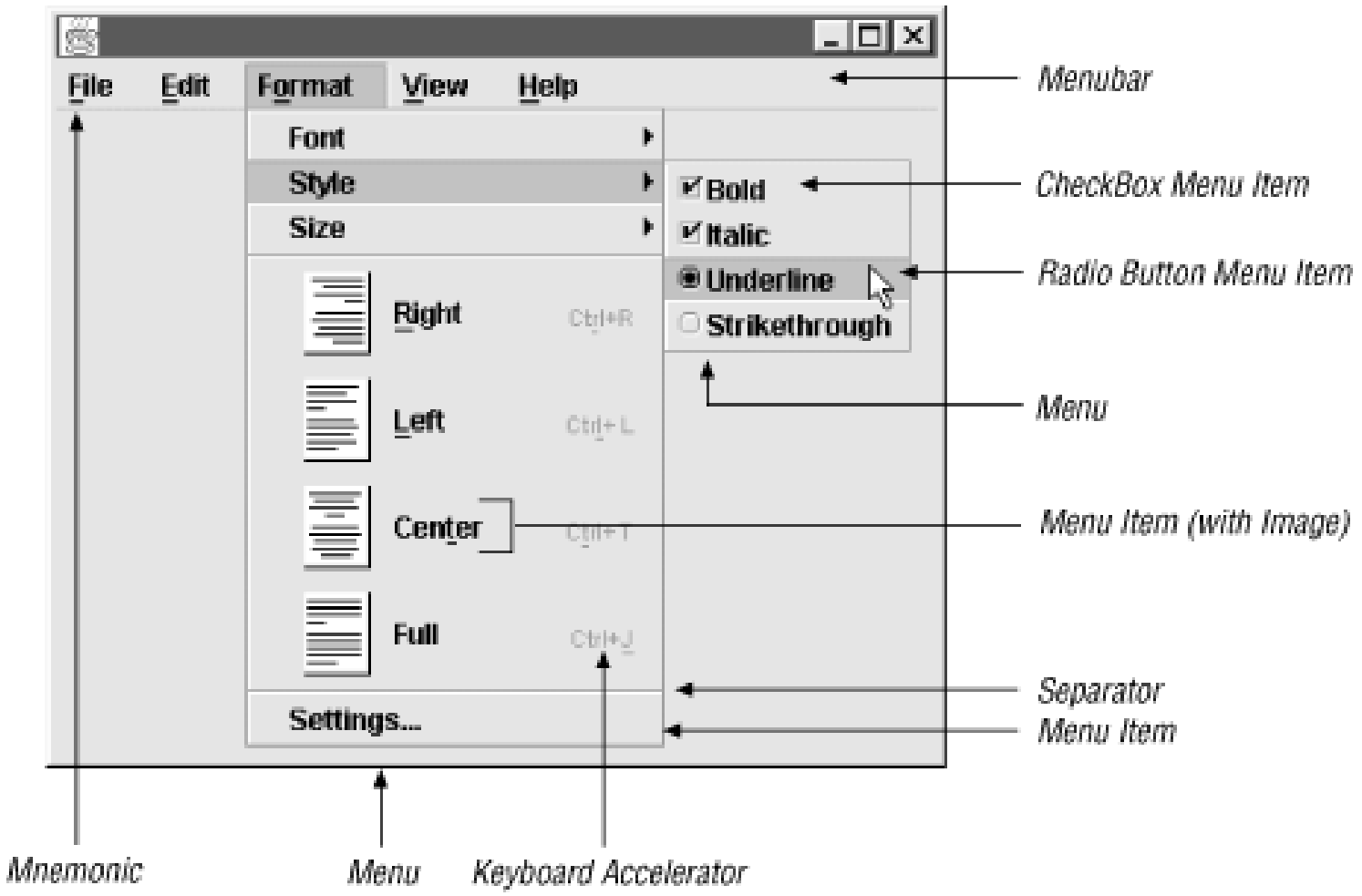
TOOL TIP

- `JToolTip`
- Windows that allows association of contextual information to application components
- Visible when the mouse is over the component
- `void setToolTipText( String text)`
- They become active when the mouse remain 750 ms over the component
- It remains active 4000ms
- If we enter, and get out from the component it activates in 500 ms
- The time periods can be modified through class `ToolTipManager`
  - `setInitialDelay(), setDismissDelay(), setReshowDelay()`

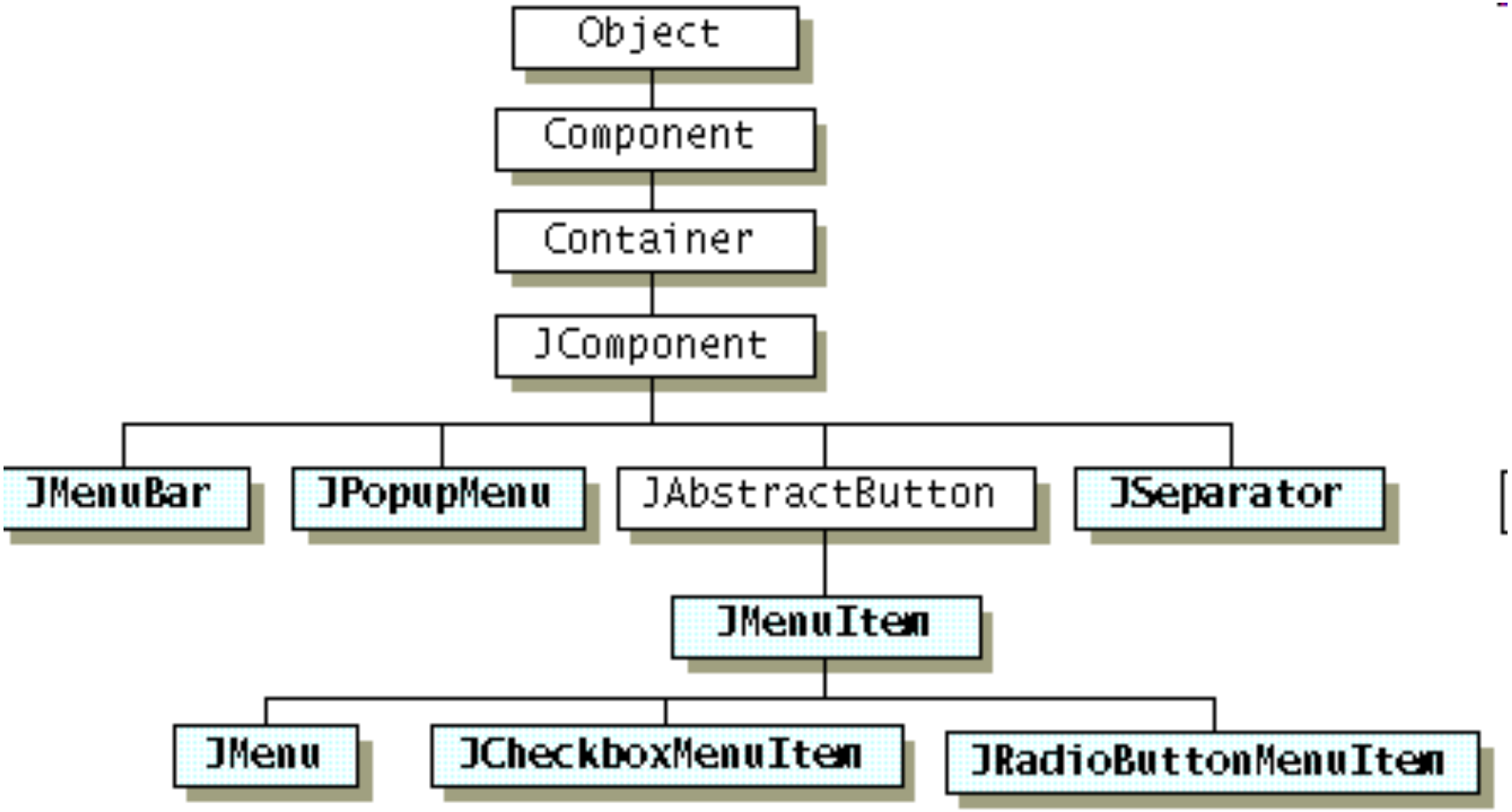
MENUS

- **Swing allows creation of**
  - Menu bars
    - `JMenuBar`
  - Menus
    - `JMenu, JMenuItem, JSeparator, JCheckBoxMenuItem, JRadioButtonMenuItem`
  - Popup menus
    - `JPopupMenu`
  - Toolbars
    - `JToolBar`

MENUS


File
Edit
Format
View
Help
Font
Style
Size
Right
Ctrl+R
Left
Ctrl+L
Center
Ctrl+T
Full
Ctrl+J
Settings...
Bold
Italic
Underline
Strikethrough
Menubar
CheckBox Menu Item
Radio Button Menu Item
Menu
Menu Item (with Image)
Separator
Menu Item
Mnemonic
Menu
Keyboard Accelerator

MENUS. CLASS HIERACHY



MENUS

- **Events**
  - Mouse
  - Keyboard
    - Key combinations (mnemonics0
      - usually fit letter from submenu
    - Accelerators
      - Allows direct access of a submenu

BUILDING A MENU

- **Building menu bar**

```
JMenuBar bar = new JMenuBar();
```

- **Building the menu**

```
JMenu meniu = new JMenu("Example");
```

- **Building menu items**

```
meniu.add(new JMenuItem("intrentryare 1"));
meniu.addSeparator();
meniu.add(new JMenuItem("entry 2"));
```

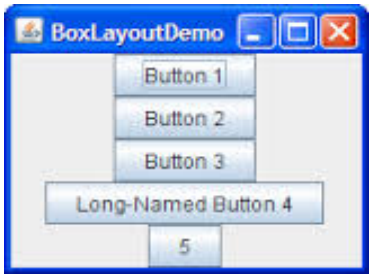
- **Adding the menu to menubar**

```
bar.add(meniu);
```

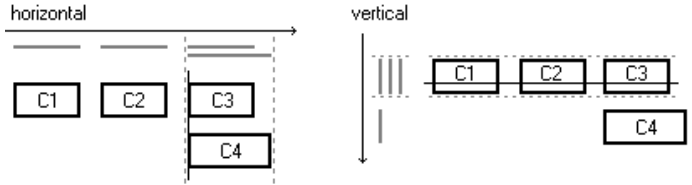
- **Adding the menubar to frame**

```
JFrame frame = new JFrame("Title");
frame.setJMenuBar(bar);
```

MORE LAYOUTS



- **`BoxLayout`**
  - The components are displayed like a stack and can be paced on vertical or horizontal
- **`SpringLayout`**
  - Defines directed relations (constraints) between components edges
- **`GroupLayout`**
  - Hieratical groups components in order place them into container
  - horizontal layout = sequential group { c1, c2, parallel group (LEFT) { c3, c4 } }
  - vertical layout = sequential group { parallel group (BASELINE) { c1, c2, c3 }, c4 }



LOOK & FEEL

- **Swing allows the modification of the way components look**
  - Look - the way in which the components look
  - Fell - the way in which the components behaves

- **Types**
  - `CrossPlatformLookAndFeel`
    - Java L&F - Metal
  - `SystemLookAndFeel`
    - L&F native for the operation system on which the application runs
  - `Synth`
    - Create your own L&F using XML files
  - Multiplexing
    - Using multiple L&Fs same time

LOOK & FEEL

- **Example of usage**
  - Java L&F

```
UIManager.setLookAndFeel( UIManager.
        getCrossPlatformLookAndFeelClassName());
```

  - Platform L&F

```
UIManager.setLookAndFeel( UIManager.
                getSystemLookAndFeelClassName());
```

  - Specifying on command line of the L&F

```
java -
Dswing.defaultlaf=com.sun.java.swing.plaf.windo
ws.WindowsLookAndFeel MyAp
```

OTHER FACILITIES

- Focus Manager
- Dialogs
- Printing
- Splash Screens

DIALOGS

- **Dialog Box**
  - Is a top level window that has a title, margins and collects information from users
  - Modal or modelees
    - Modal
      - blocks the input at applications windows until it is closed
    - Modeless
      - allows to work with other application's windows while it is opened

DIALOGS TYPES

- **Modeless**
  - A modeless dialog box does not block any other window while it is visible
- **Document-modal**
  - A document-modal dialog box blocks all windows from the same document, except windows from its child hierarchy. In this context, a document is a hierarchy of windows that share a common ancestor, called the document root, which is the closest ancestor window without an owner
- **Application-modal**
  - An application-modal dialog box blocks all windows from the same application, except windows from its child hierarchy. If several applets are launched in a browser environment, the browser is allowed to treat them either as separate applications or as a single application. This behavior is implementation-dependent.
- **Toolkit-modal**
  - A toolkit-modal dialog box blocks all windows that run in the same toolkit, except windows from its child hierarchy. If several applets are launched, all of them run with the same toolkit. Hence, a toolkit-modal dialog box shown from an applet may affect other applets and all windows of the browser instance that embeds the Java runtime environment for this toolkit.
- **Exclusion mode**
  - Any top-level window can be marked not to be blocked by modal dialogs.

CONCLUSIONS

- **Advantages**
  - Portability
    - Contains few elements platform specific
  - Behavior
    - Allows a more flexible behavior to the components because they are not so peered with operating system
  - Properties
    - Supports more properties like icons, tooltips ...
  - Look and Feel
    - Allows application to look similar on all platforms
- **Disadvantages**
  - Applets portability
    - Most of the browsers does not include Swing classes so it have to install a Java plug-in
  - Performance
    - Swing components are slower that AWT components
  - Look & Feel
    - In case the components use the L&F of the operating system the components could look different

BIBLIOGRAPHY

- https://docs.oracle.com/javase/tutorial/uiswing/TOC.html