Java User Interfaces with Swing in SunOne

An introduction to the Swing API and code generation in the SunOne development environment.

By Marc-André Laverdière

Ordre du Jour

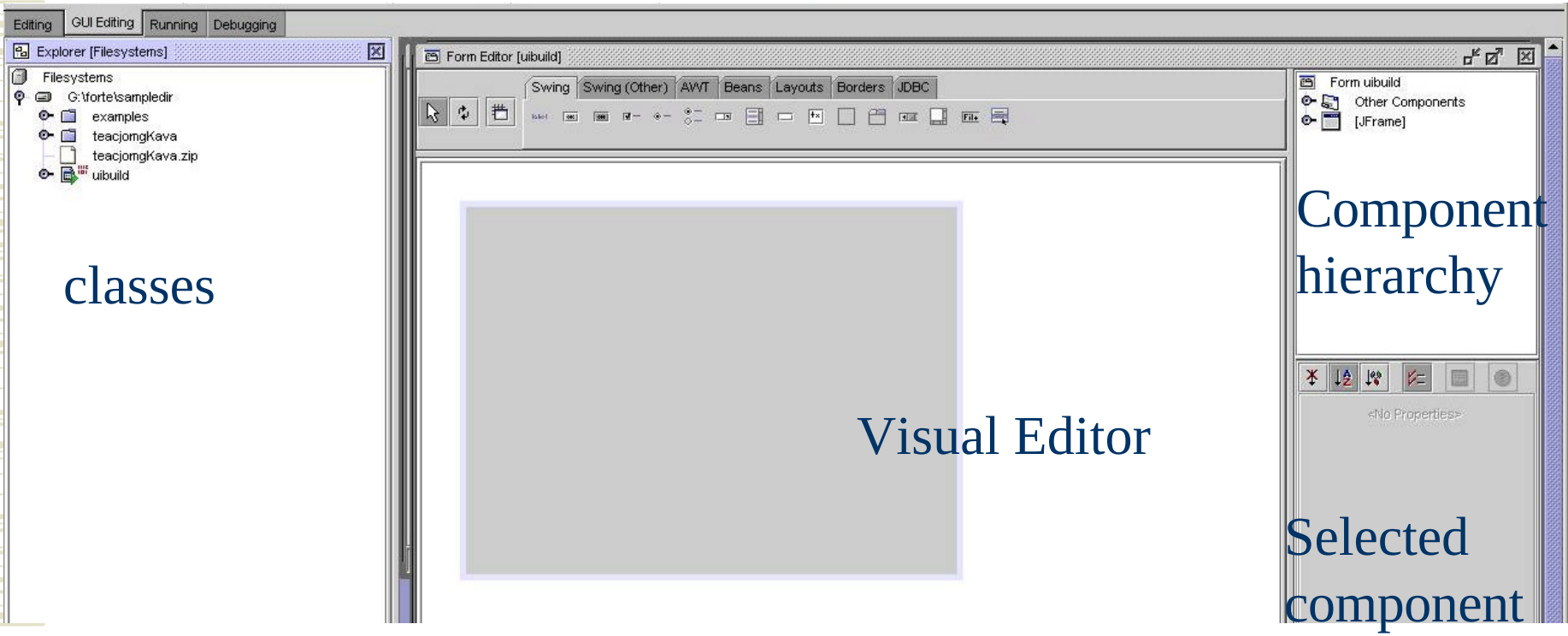
- What is Swing?
- Introduction to the drag-and-drop GUI development in SunOne
- First glance at Swing components in SunOne
- Basics of Swing
  - Class hierarchies
  - AWT Mode of Operation and Consequences
  - Event Handling
- API
- References

What is Swing?

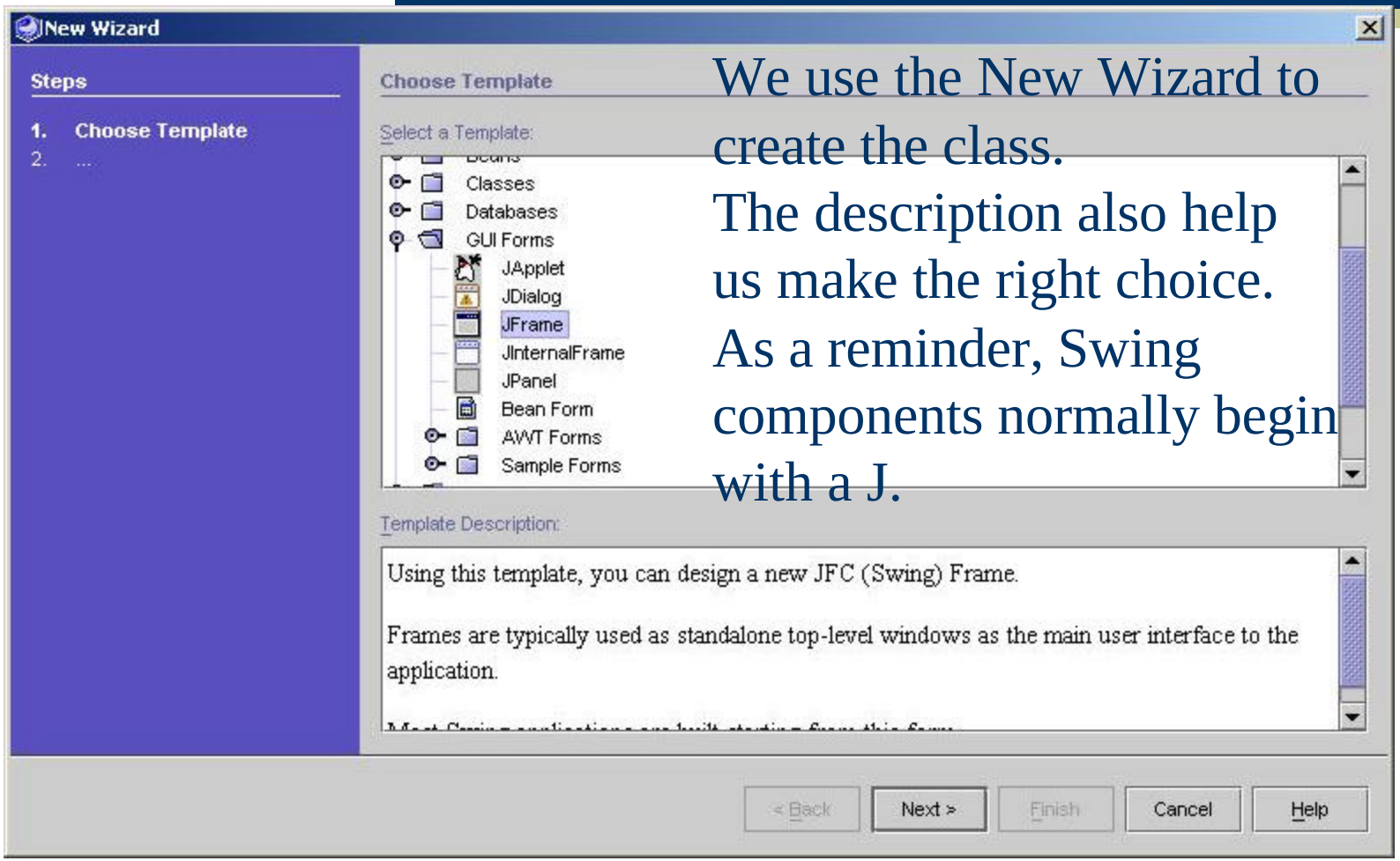
- Swing is a platform-independent, high-level API to build user interfaces
- Swing won't allow you to draw a canvas or graphic objects, you need AWT for that.
- Packages: java.awt.*, java.awt.event.*, javax.swing.*.
- We will focus on the most commonly used elements. If you want to learn more about a specific component, tutorials are all over the 'net.

Introduction to the drag-and-drop GUI development in SunOne

Code / GUI editing tabs



Introduction to the drag-and-drop GUI development in SunOne

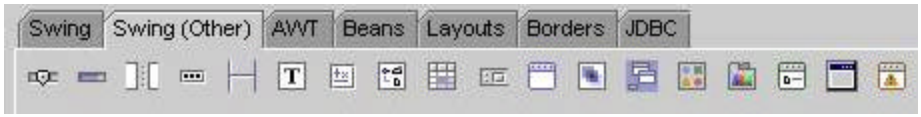


We use the New Wizard to create the class.
The description also help us make the right choice.
As a reminder, Swing components normally begin with a J.

Introduction to the drag-and-drop GUI development in SunOne

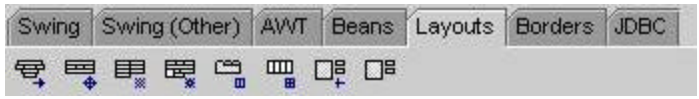
- The symbols, from left to right:
  - JLabel
  - JButton
  - JToggleButton
  - JCheckBox
  - JRadioButton
  - ButtonGroup
  - JComboBox
  - JList
  - JTextField
  - JTextArea
  - JPanel
  - JTabbedPane
  - JScrollBar
  - JScrollPane
  - JMenuBar
  - JPopupMenu

Introduction to the drag-and-drop GUI development in SunOne



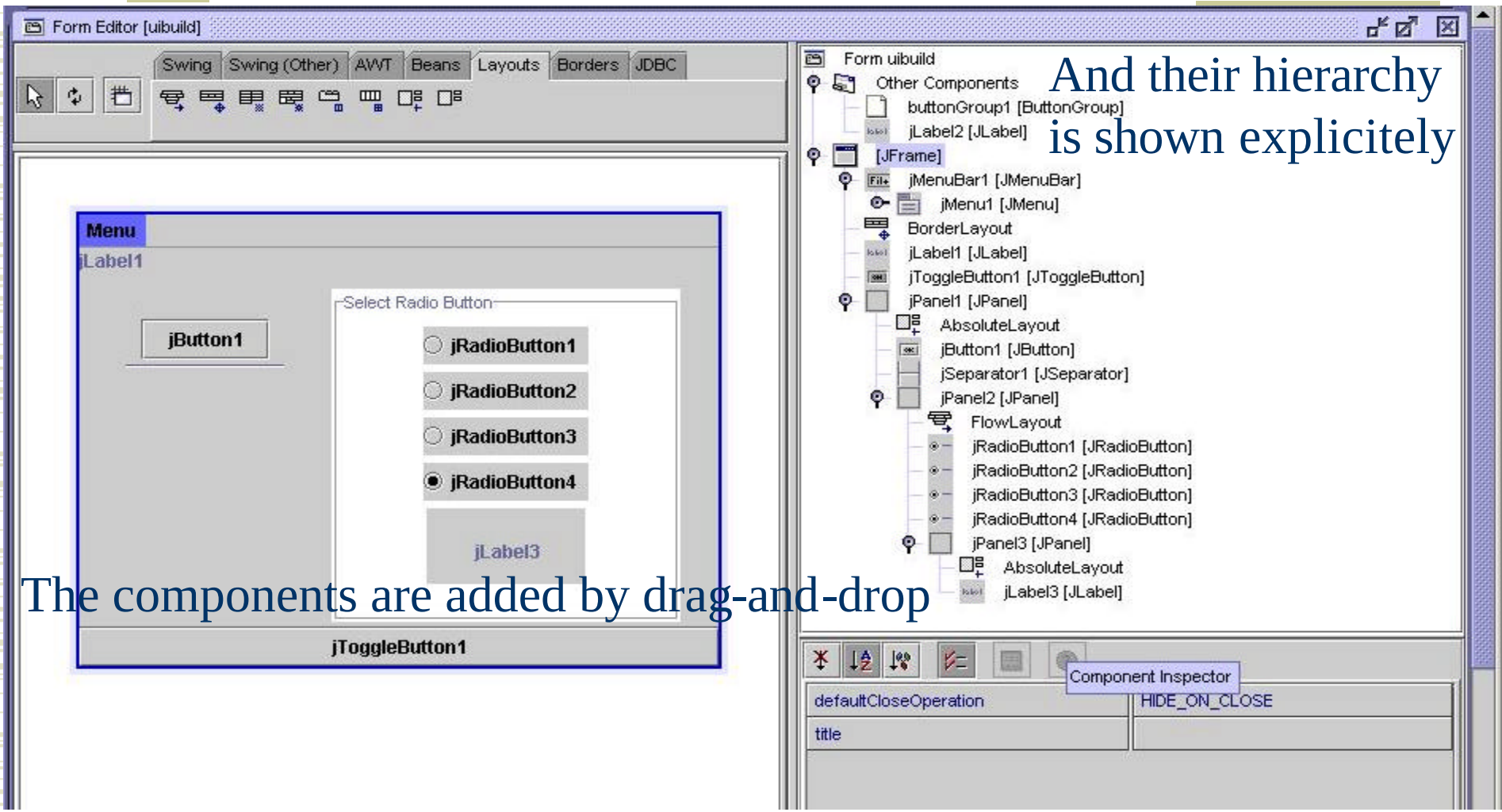
- The symbols, from left to right:
  - JSlider
  - JProgressBar
  - JSplitPane
  - JPasswordField
  - JSeparator
  - JTextPane
  - JEditorPane
  - JTree
  - JTable
  - JToolBar
  - JInternalFrame
  - JLayeredPane
  - JDesktopPane
  - JOptionPane
  - JColorChooser
  - JFileChooser
  - Jframe
  - JDialog

Introduction to the drag-and-drop GUI development in SunOne



- The symbols, from left to right:
  - FlowLayout
  - BorderLayout
  - GridLayout
  - GridBagLayout
  - CardLayout
  - BoxLayout
  - AbsoluteLayout
  - NullLayout

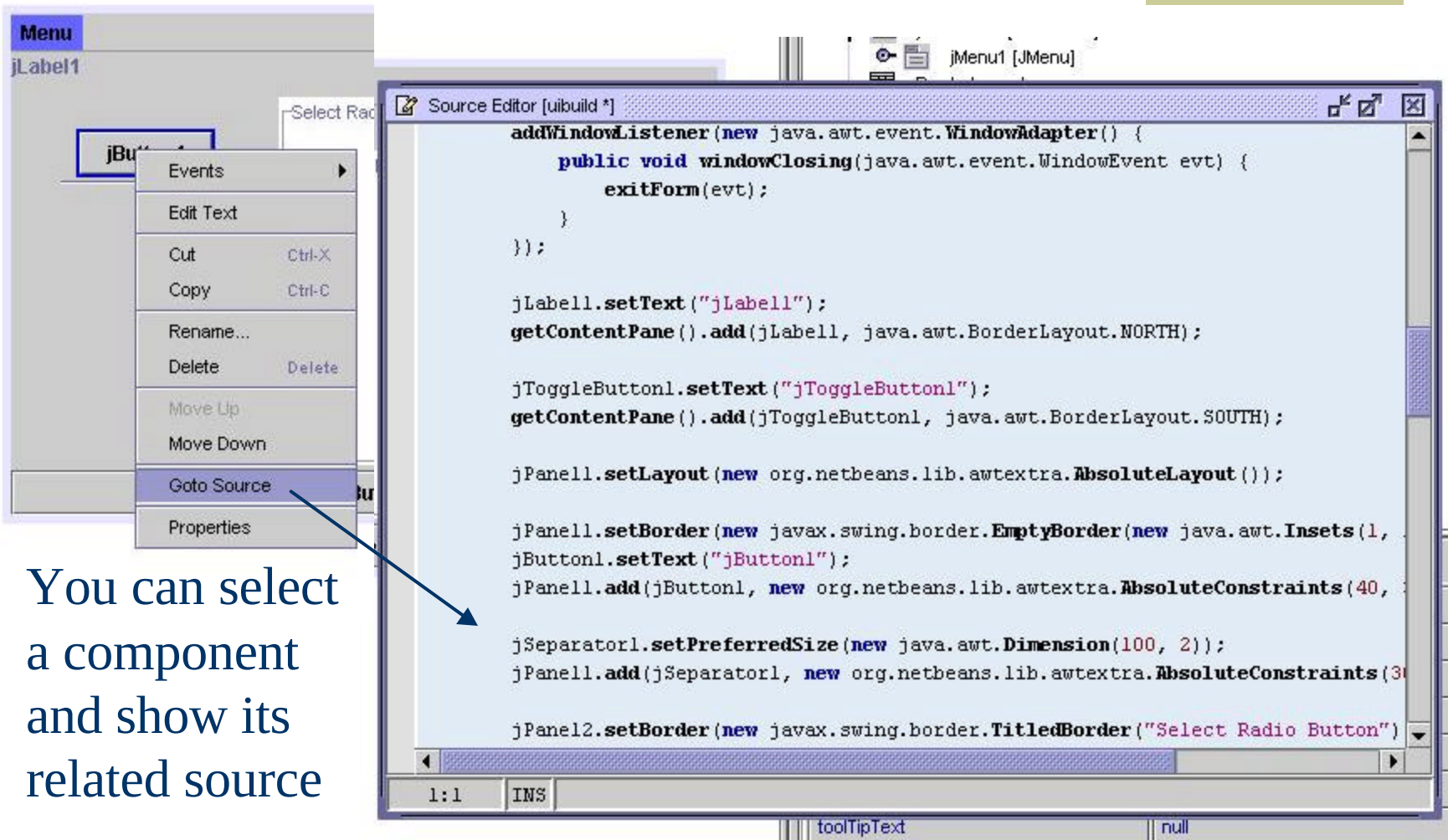
Introduction to the drag-and-drop GUI development in SunOne



And their hierarchy is shown explicitely

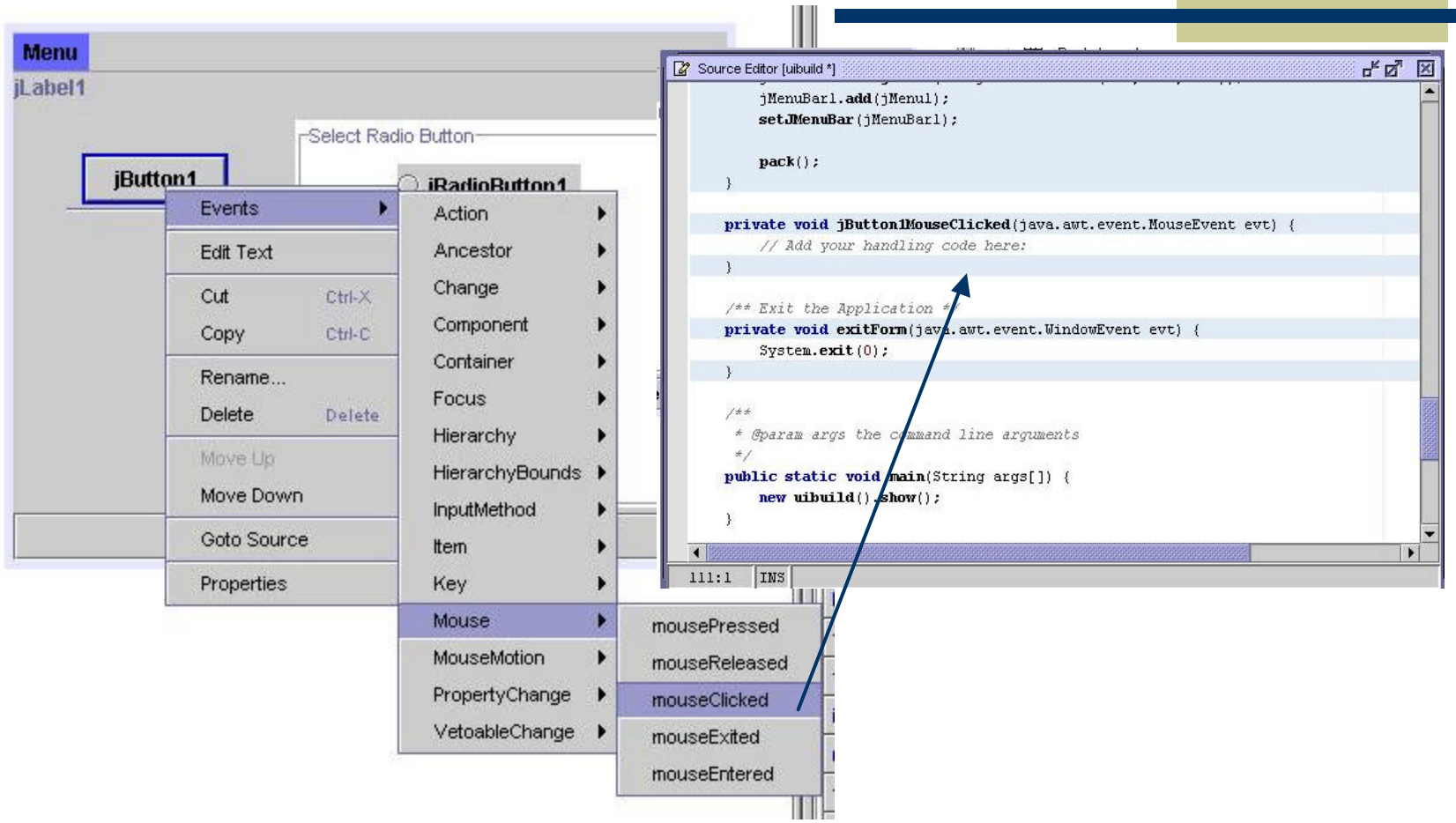
The components are added by drag-and-drop

Introduction to the drag-and-drop GUI development in SunOne

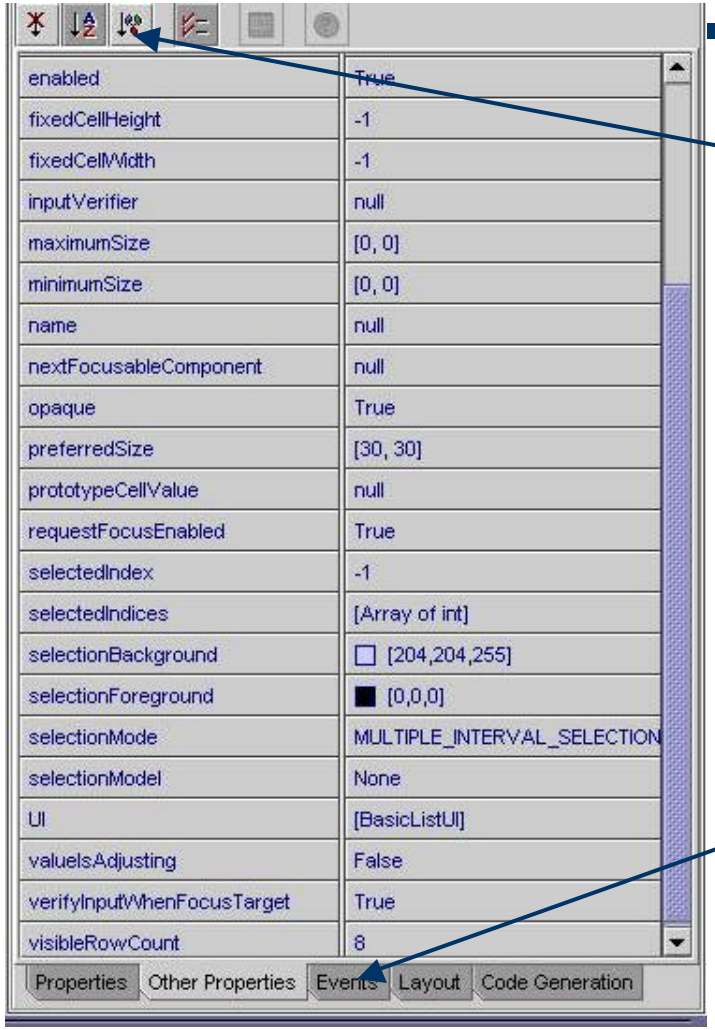


You can select a component and show its related source

Introduction to the drag-and-drop GUI development in SunOne



Introduction to the drag-and-drop GUI development in SunOne

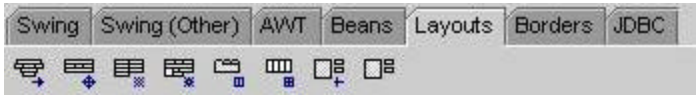


Sorting of options

Groups Properties, event editing and options

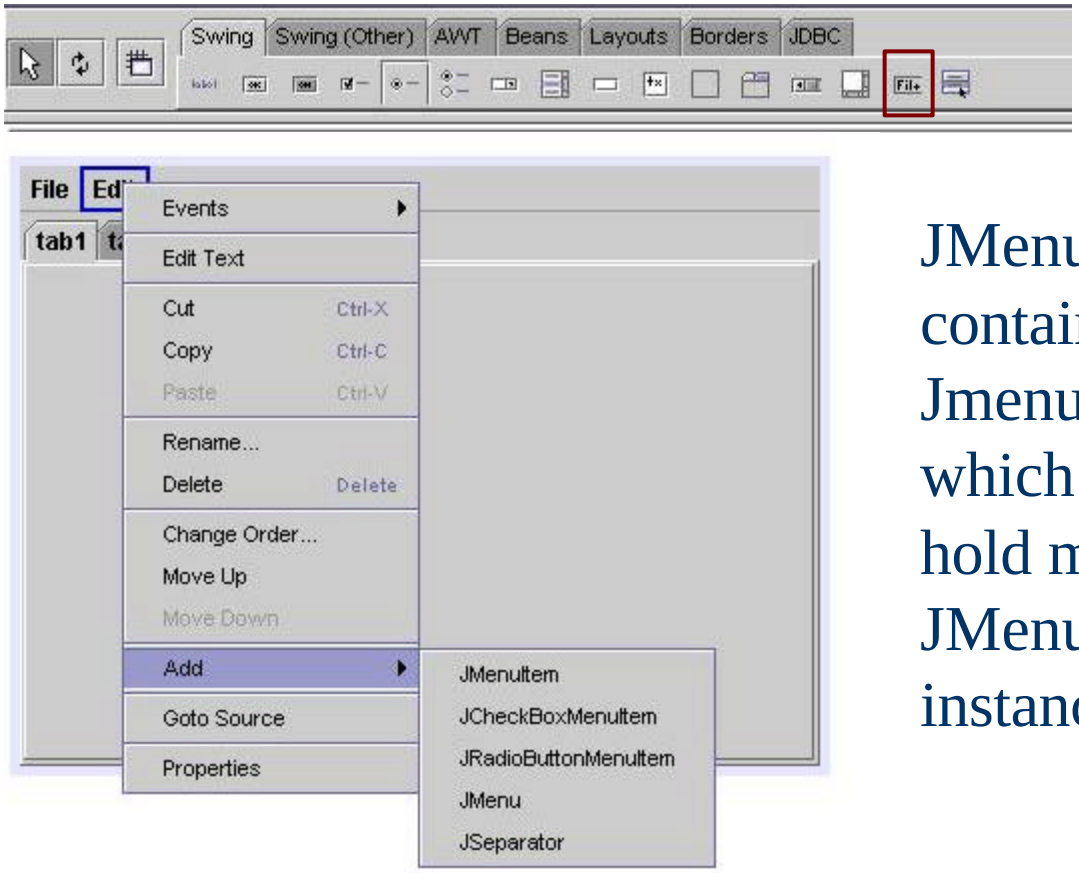
First glance at Swing components in SunOne

Layouts



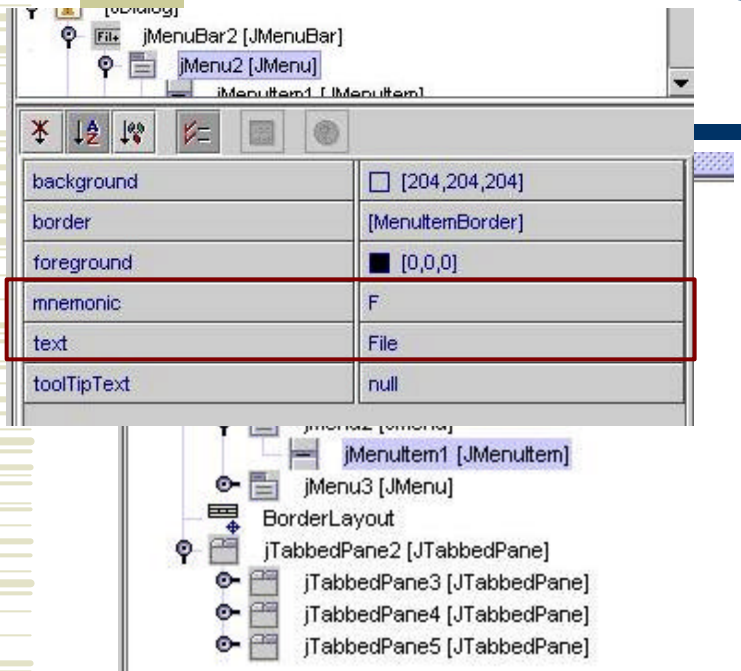
- FlowLayout: components are added on a row, one after the other. They are put on the line below if not enough room remains.
- BorderLayout: components are added at the edges and center of the area.
- GridLayout: creates a ‘table’ in which to put components.
- GridBagLayout: similar to GridLayout. Each component has its own size, and items can be inserted in any order.
- CardLayout: many layouts (such as JPanel) are stacked on each other. Only the top one is visible and we can decide which one to show.
- BoxLayout: organizes components vertically or horizontally.
- AbsoluteLayout: allow to put components wherever you want them to go.
- Null Layout: behaves just like AbsoluteLayout.

JMenuBar



JMenuBar can contain many Jmenu instances, which in turn may hold many JMenuItem instances.

JMenuBar



Bad example: keep Alt+? for mnemonics

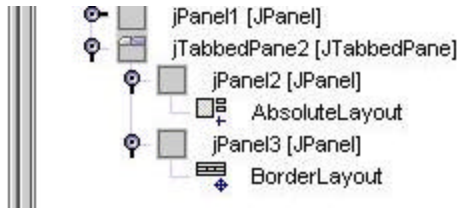
Many components support an ‘accelerator’ key sequence (AKA keyboard shortcuts). Make sure to define them with the wizard provided.
Also, mnemonics are supported, enabling key sequences such as Alt-F.
Also, set the toolTipText property. It shows a bubble that helps users and that enables screen readers.

JPanel

- JPanel is a generic ‘holder’ component.
- Each JPanel has a layout associated. As such, when mixing layout styles, do so with embedded JPanel instances.
- JPanel can hold pretty much anything within itself.
- JPanel is the likely candidate when using borders.

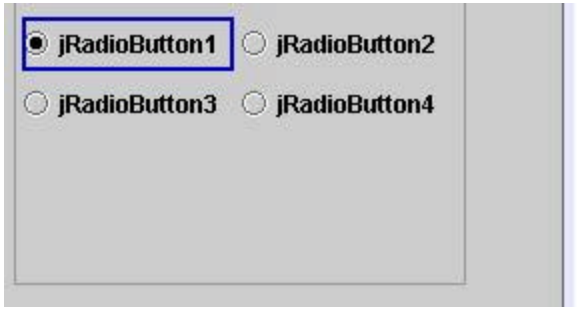
JTabbedPane

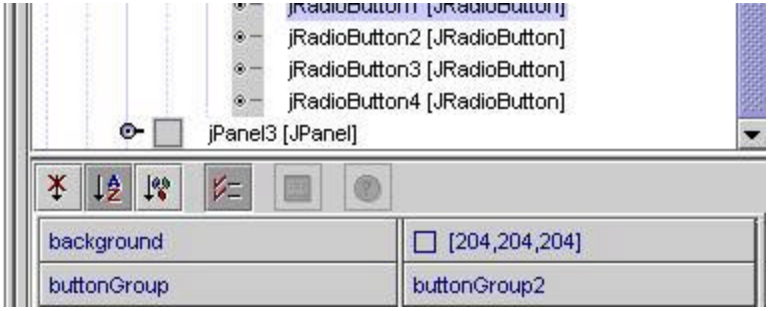
- JTabbedPane is also a ‘holder’. Each subcomponent of a JTabbedPane is in its own tab.
- You should use a JPanel for every tab you want to show.



JRadioButton

- JRadioButton is the classical radio button: only one in a group may be selected.
- In order to group them, you need a ButtonGroup to which it is linked.





JCheckBox

- Many checkboxes can be selected at any given time.
- They should be grouped together with a ButtonGroup, but this is not required.

JLabel

- A simple text label
- Rarely needed. Most of the time, the other components come with their own label.

JButton

- The average button

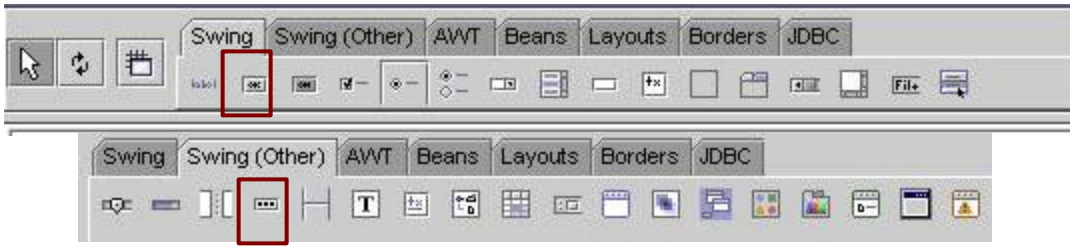
JComboBox

- Scrolldown list
- The choices are put manually in the code with the .addItem() call, which accepts a String object.

JList

- Selection list
- The choices are normally provided in the constructor of the object, by passing an array of String objects.
- The scrolldown bar is added automatically if need be.
- JList objects allow single entry selection or multiple-entry selection modes.

JTextField + JPasswordField



- Allows to prompt for user input for a single line of text.
- JPasswordField is identical, but won't display the characters typed.

JTextPane

- Supports multi-line text, as well as multi-text property setting.
- What you'd use if you felt like re-doing Notepad.
- Doesn't automatically integrate a scrollbar.

JEditorPane

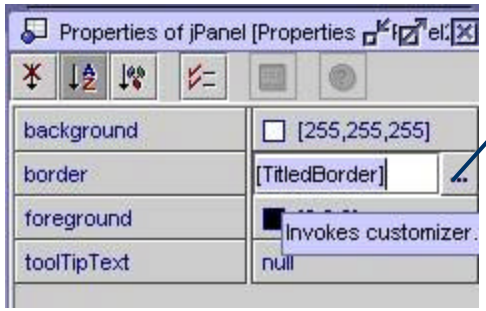
- Similar to JTextPane, but has an integrated support for HTML, including URL objects.
- Also features a HTML parser

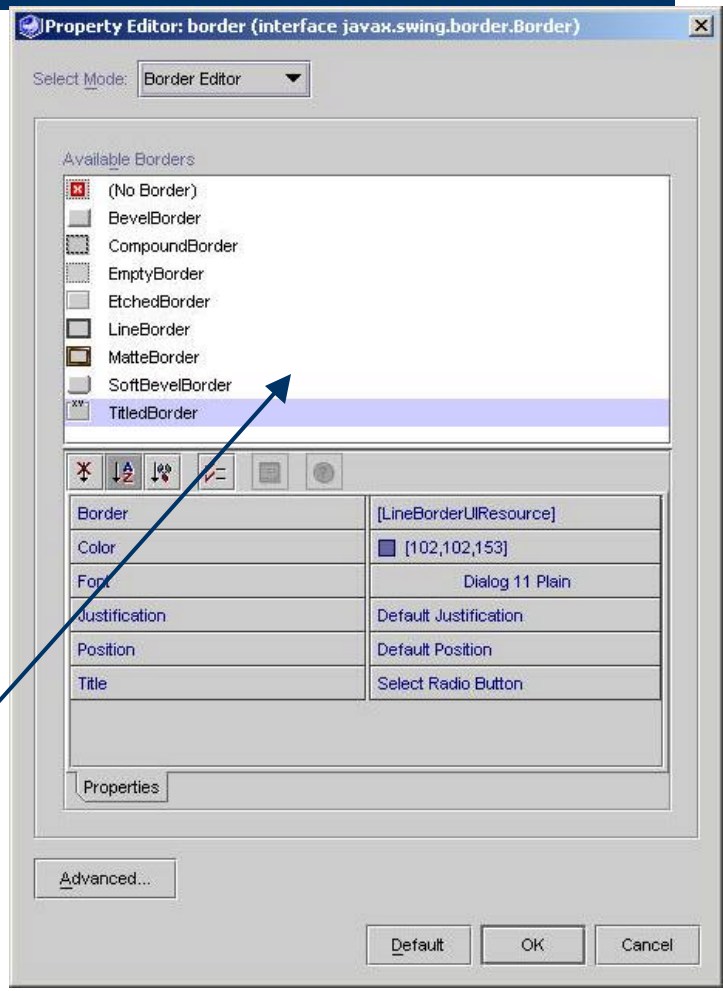
JScrollPane

- JScrollPane is another ‘holder’ object.
- It will make relevant scrollbars appear when needed by the held object.
- Normally combined with a JTextPane or a JEditorPane.

Borders

- These can be put around holder components such as a JPanel.
- Use the properties window to set them.

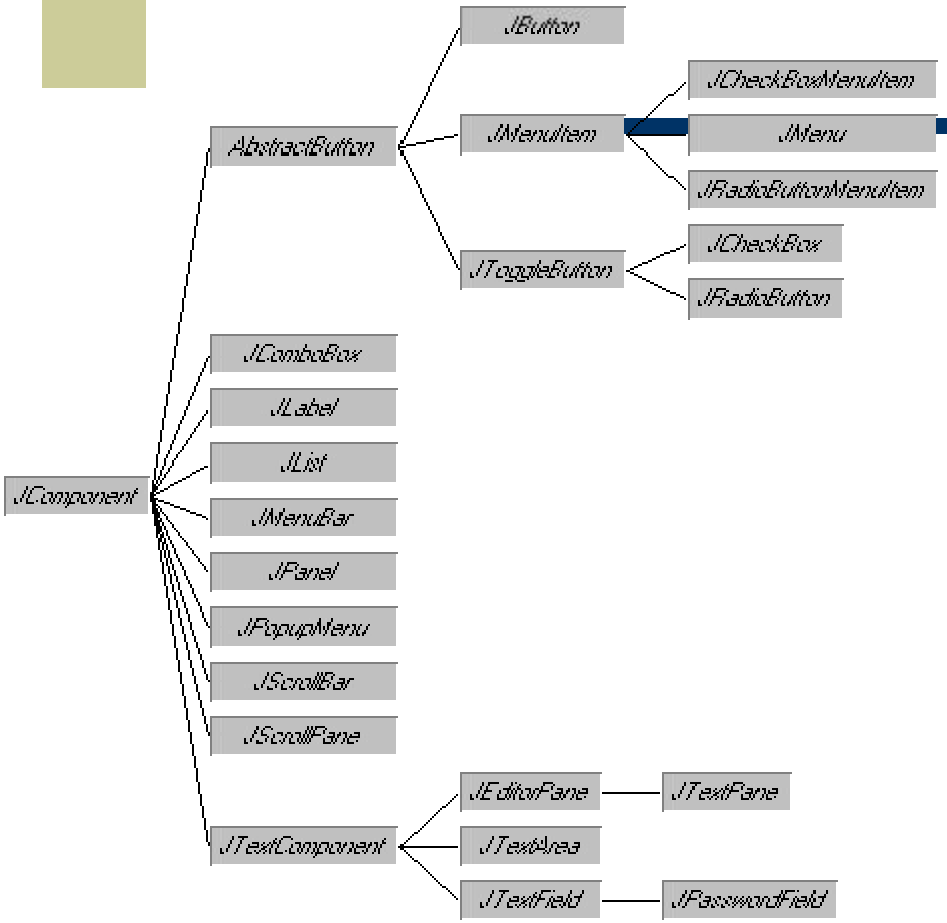




Basics of Swing

- Swing is a Java-based implementation over the Abstract Windowing Toolkit (AWT) package
- It offers easier cross-platform development, since it doesn’t rely directly on the underlying platform’s windowing system.
- Beware of usability issues this introduces when developing multi-platform software.
- Remember that GUI programming is inherently messy code-wise, but try to avoid the monolithic code structure, even if this is what is encouraged by SunOne’s code generator.

Class Hierarchies

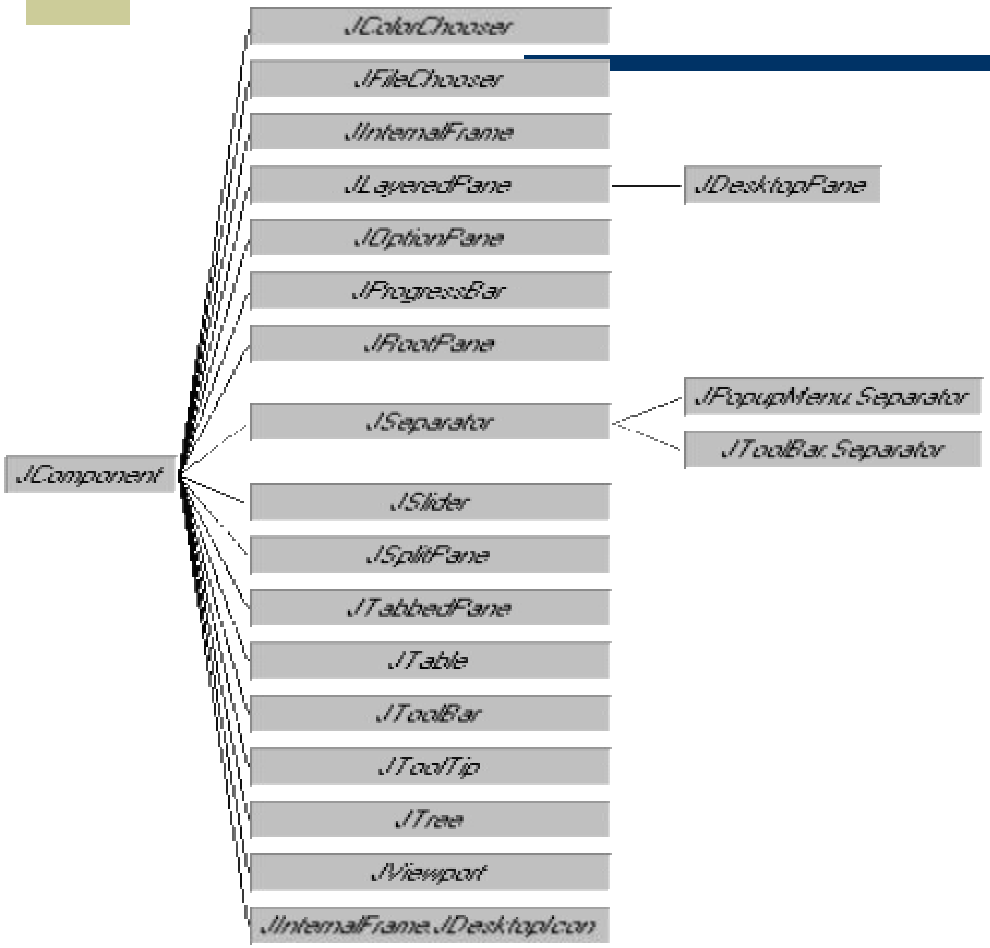


This hierarchy shows how the basic Swing objects are structured in a inheritance hierarchy.

Keep in mind that JComponent inherits from java.awt.Container.

http://developer.java.sun.com/developer/onlineTraining/GUI/Swing1/shortcourse.html

Class Hierarchies

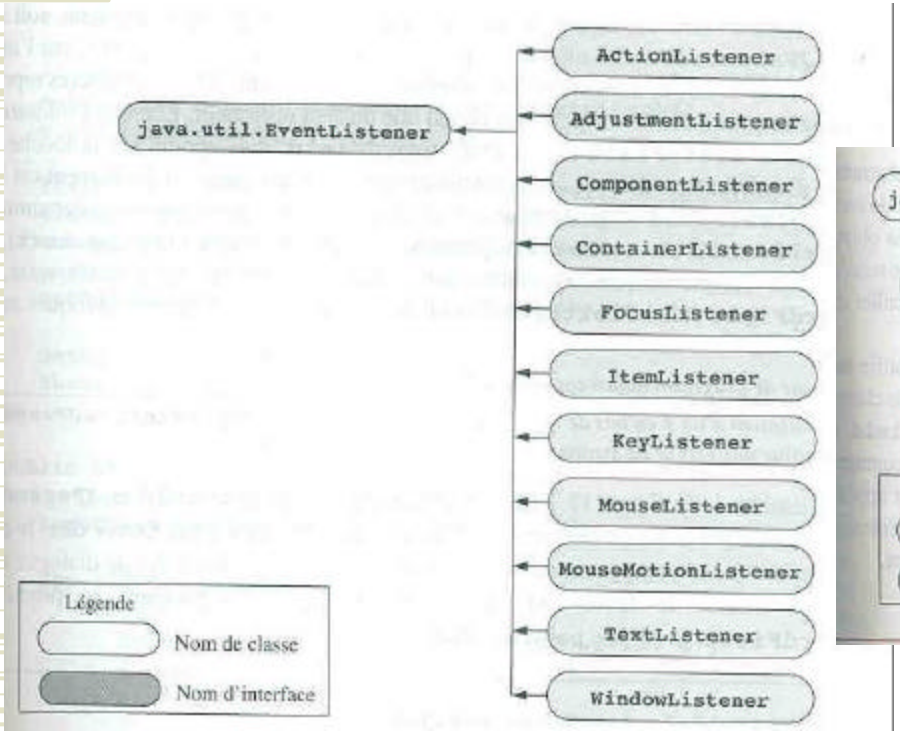


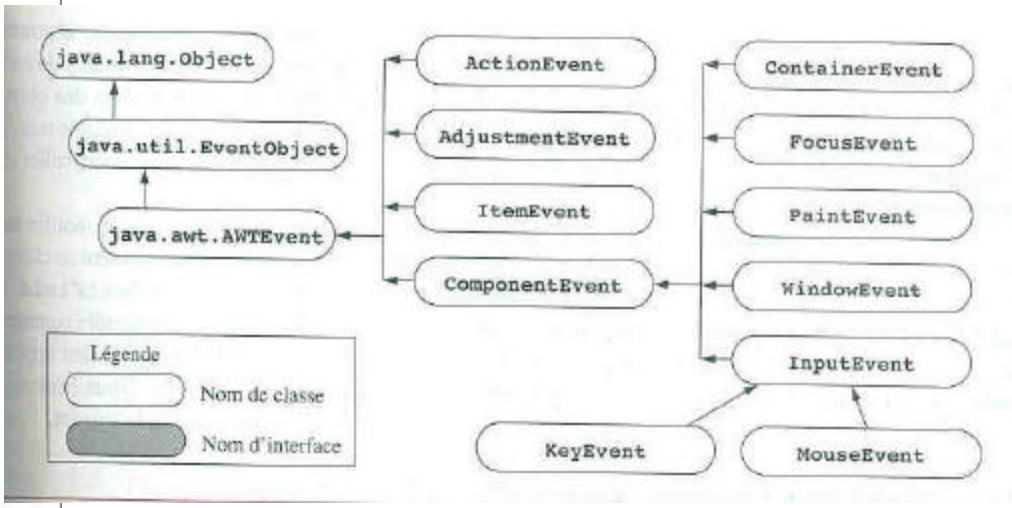
This hierarchy complements the previous one.
It shows newer Swing components.

http://developer.java.sun.com/developer/onlineTraining/GUI/Swing1/shortcourse.html

Class Hierarchies

Events & Listeners





Comment Programmer en Java[tm] troisième édition, Deitel & Deitel, Les Éditions Reynald Goulet Inc., 1999, p.565

AWT Mode of Operation and Consequences

- AWT components all run in a single thread of execution.
- This thread both handles the display of components on the screen (painting) and event handling.
- Remember that Swing is (mostly) a Java wrapping over AWT.
- This imposes tough real-time constraints on UI designers, especially for programs running on older machines.

AWT Mode of Operation and Consequences

- Rule of thumb A: keep the user interfaces very simple, making judicious use of object properties such as visible.
- Rule of thumb B: Design event-handling routines so that they are executed very quickly. If this is impossible, then multithreading is necessary. Failure to do so may result in a screen that is ‘frozen’ to the user, which is a major usability issue.

Event Handling

- Event handling is where a lot of work will be invested.
- The more complex the functionality to implement, the trickier will be the related event handler.
- AWT has many event types, but only a few are relevant for Swing components, and only a few are useful. Also, there are event types specific to Swing components.
- There are many interfaces usable, but it is recommended to inherit adapters.
- Try to put event handlers into separate files in order to facilitate maintainability
- Avoid embedded anonymous subclasses for event handling, this is the worst thing maintenance-wise. Some authors show code examples that do that, but keep in mind that it is done in order to shorten the code sample.

Event Handling

- Events are generated from input devices or special software constructs such as timers.
- The object targeted by the event will load the designated event handler for the type of event received, or will ignore the event if none are registered.
- The event handler will call the appropriate method related to the event, and some methods are meant to ‘work together’ for certain tasks.
- Event handlers can be assigned to many components, in order to centralize decision logic and facilitate maintenance.

Event Handling Hierarchy

- All event objects are derived from java.util.EventObject
- The getSource() method has a reference to Swing component to which the event was directed. You should cast this in the appropriate type in order to distinguish among many events on the same listener.

API

- Events’ API
- Event-Handling API
- Components’ API

Events’ API

MouseEvent

An event which indicates that a mouse action occurred in a component. A mouse action is considered to occur in a particular component if and only if the mouse cursor is over the unobscured part of the component's bounds when the action happens. Component bounds can be obscurred by the visible component's children or by a menu or by a top-level window. This event is used both for mouse events (click, enter, exit) and mouse motion events (moves and drags).

http://java.sun.com/j2se/1.4/docs/api/

MouseEvent

Method Summary	
int	**getButton**() Returns which, if any, of the mouse buttons has changed state.
int	**getClickCount**() Returns the number of mouse clicks associated with this event.
static String	**getMouseModifiersText**(int modifiers) Returns a String describing the modifier key(s), such as "Shift", or "Ctrl+Shift".
Point	**getPoint**() Returns the x,y position of the event relative to the source component.
int	**getX**() Returns the horizontal x position of the event relative to the source component.
int	**getY**() Returns the vertical y position of the event relative to the source component.
boolean	**isPopupTrigger**() Returns whether or not this mouse event is the popup menu trigger event for the platform.
String	**paramString**() Returns a parameter string identifying this event.
void	**translatePoint**(int x, int y) Translates the event's coordinates to a new position by adding specified x (horizontal) and y (vertical) offsets.

http://java.sun.com/j2se/1.4/docs/api/

MouseEvent

Field Summary

static int	**BUTTON1** Indicates mouse button #1; used by getButton().
static int	**BUTTON2** Indicates mouse button #2; used by getButton().
static int	**BUTTON3** Indicates mouse button #3; used by getButton().
static int	**MOUSE_CLICKED** The "mouse clicked" event.
static int	**MOUSE_DRAGGED** The "mouse dragged" event.
static int	**MOUSE_ENTERED** The "mouse entered" event.
static int	**MOUSE_EXITED** The "mouse exited" event.
static int	**MOUSE_FIRST** The first number in the range of ids used for mouse events.
static int	**MOUSE_LAST** The last number in the range of ids used for mouse events.
static int	**MOUSE_MOVED** The "mouse moved" event.
static int	**MOUSE_PRESSED** The "mouse pressed" event.
static int	**MOUSE_RELEASED** The "mouse released" event.
static int	**MOUSE_WHEEL** The "mouse wheel" event.
static int	**NOBUTTON** Indicates no mouse buttons; used by getButton().

http://java.sun.com/j2se/1.4/docs/api/

ActionEvent

•A semantic event which indicates that a component-defined action occured. This high-level event is generated by a component (such as a Button) when the component-specific action occurs (such as being pressed). The event is passed to every every ActionListener object that registered to receive such events using the component's addActionListener method.

•The object that implements the ActionListener interface gets this ActionEvent when the event occurs. The listener is therefore spared the details of processing individual mouse movements and mouse clicks, and can instead process a "meaningful" (semantic) event like "button pressed".

http://java.sun.com/j2se/1.4/docs/api/

ItemEvent

•A semantic event which indicates that an item was selected or deselected. This high-level event is generated by an ItemSelectable object (such as a List) when an item is selected or deselected by the user. The event is passed to every ItemListener object which registered to receive such events using the component's addItemListener method.

•The object that implements the ItemListener interface gets this ItemEvent when the event occurs. The listener is spared the details of processing individual mouse movements and mouse clicks, and can instead process a "meaningful" (semantic) event like "item selected" or "item deselected".

http://java.sun.com/j2se/1.4/docs/api/

ItemEvent

Field Summary

static int	**DESELECTED** This state-change-value indicates that a selected item was deselected.
static int	**ITEM_FIRST** The first number in the range of ids used for item events.
static int	**ITEM_LAST** The last number in the range of ids used for item events.
static int	**ITEM_STATE_CHANGED** This event id indicates that an item's state changed.
static int	**SELECTED** This state-change value indicates that an item was selected.

Method Summary

Object	**getItem**() Returns the item affected by the event.
ItemSelectable	**getItemSelectable**() Returns the originator of the event.
int	**getStateChange**() Returns the type of state change (selected or deselected).
String	**paramString**() Returns a parameter string identifying this item event.

http://java.sun.com/j2se/1.4/docs/api/

ListSelectionEvent

An event that characterizes a change in the current selection. The change is limited to a row interval. ListSelectionListeners will generally query the source of the event for the new selected status of each potentially changed row.

That means that, in general, the ListSelectionEvent handler will not normally rely on the event's information, but will consider the event as a notification that there is a need to check on the state of the list.

http://java.sun.com/j2se/1.4/docs/api/

WindowEvent

A low-level event that indicates that a window has changed its status. This low-level event is generated by a Window object when it is opened, closed, activated, deactivated, iconified, or deiconified, or when focus is transfered into or out of the Window.
The event is passed to every WindowListener or WindowAdapter object which registered to receive such events using the window's addWindowListener method. (WindowAdapter objects implement the WindowListener interface.) Each such listener object gets this WindowEvent when the event occurs.

It is useful to monitor this event especially in cases that there are components to redraw or when the window is closed. Swing (or SunOne Code generator) does so automatically, so you don’t need to bother about it

http://java.sun.com/j2se/1.4/docs/api/

KeyEvent

An event which indicates that a keystroke occurred in a component.
This low-level event is generated by a component object (such as a text field) when a key is pressed, released, or typed. The event is passed to every KeyListener or KeyAdapter object which registered to receive such events using the component's addKeyListener method. (KeyAdapter objects implement the KeyListener interface.) Each such listener object gets this KeyEvent when the event occurs.
"Key typed" events are higher-level and generally do not depend on the platform or keyboard layout. They are generated when a Unicode character is entered, and are the preferred way to find out about character input. In the simplest case, a key typed event is produced by a single key press (e.g., 'a'). Often, however, characters are produced by series of key presses (e.g., 'shift' + 'a'), and the mapping from key pressed events to key typed events may be many-to-one or many-to-many. Key releases are not usually necessary to generate a key typed event, but there are some cases where the key typed event is not generated until a key is released (e.g., entering ASCII sequences via the Alt-Numpad method in Windows).

These are handled for you for the shortcuts and so on. There is little need to bother about that event, unless you were to implement a ‘on-the-fly’ text entry like Rational.

http://java.sun.com/j2se/1.4/docs/api/

KeyEvent

KeyEvent has very many constants that we won't detail. But we'll see the methods. Consult the API documentation for the list of keys and subtleties.

Method Summary	
char	**getKeyChar**() Returns the character associated with the key in this event.
int	**getKeyCode**() Returns the integer keyCode associated with the key in this event.
int	**getKeyLocation**() Returns the location of the key that originated this key event.
static String	**getKeyModifiersText**(int modifiers) Returns a String describing the modifier key(s), such as "Shift", or "Ctrl+Shift".
static String	**getKeyText**(int keyCode) Returns a String describing the keyCode, such as "HOME", "F1" or "A".
boolean	**isActionKey**() Returns whether the key in this event is an "action" key.
String	**paramString**() Returns a parameter string identifying this event.
void	**setKeyChar**(char keyChar) Set the keyChar value to indicate a logical character.
void	**setKeyCode**(int keyCode) Set the keyCode value to indicate a physical key.
void	**setModifiers**(int modifiers) Set the modifiers to indicate additional keys that were held down (e.g.

http://java.sun.com/j2se/1.4/docs/api/

Event Handler API

- SunOne will automatically register default event handlers for you, and create a method for you to add code in.
- You should avoid to put your event handling code directly, for maintainability reasons.
- The rule of thumb is to inherit the appropriate handler into a new class and make an indirection to this class.

MouseInputAdapter

The adapter which receives mouse events and mouse motion events. The methods in this class are empty; this class is provided as a convenience for easily creating listeners by extending this class and overriding only the methods of interest.

http://java.sun.com/j2se/1.4/docs/api/

MouseInputAdapter

Method Summary	
void	**mouseClicked**(MouseEvent e) Invoked when the mouse button has been clicked (pressed and released) on a component.
void	**mouseDragged**(MouseEvent e) Invoked when a mouse button is pressed on a component and then dragged.
void	**mouseEntered**(MouseEvent e) Invoked when the mouse enters a component.
void	**mouseExited**(MouseEvent e) Invoked when the mouse exits a component.
void	**mouseMoved**(MouseEvent e) Invoked when the mouse button has been moved on a component (with no buttons down).
void	**mousePressed**(MouseEvent e) Invoked when a mouse button has been pressed on a component.
void	**mouseReleased**(MouseEvent e) Invoked when a mouse button has been released on a component.

http://java.sun.com/j2se/1.4/docs/api/

KeyAdapter

An abstract adapter class for receiving keyboard events. The methods in this class are empty. This class exists as convenience for creating listener objects.

Method Summary	
void	**keyPressed**(KeyEvent e) Invoked when a key has been pressed.
void	**keyReleased**(KeyEvent e) Invoked when a key has been released.
void	**keyTyped**(KeyEvent e) Invoked when a key has been typed.

http://java.sun.com/j2se/1.4/docs/api/

ActionListener

The listener interface for receiving action events. The class that is interested in processing an action event implements this interface, and the object created with that class is registered with a component, using the component's addActionListener method. When the action event occurs, that object's actionPerformed method is invoked.

Method Summary	
void	**actionPerformed**(ActionEvent e) Invoked when an action occurs.

http://java.sun.com/j2se/1.4/docs/api/

ItemListener

The listener interface for receiving item events. The class that is interested in processing an item event implements this interface. The object created with that class is then registered with a component using the component's addItemListener method. When an item-selection event occurs, the listener object's itemStateChanged method is invoked.

Method Summary	
void	**itemStateChanged**(ItemEvent e) Invoked when an item has been selected or deselected by the user.

http://java.sun.com/j2se/1.4/docs/api/

ListSelectionListener

The listener that's notified when a lists selection value changes.

Method Summary	
void	**valueChanged**(ListSelectionEvent e) Called whenever the value of the selection changes.

http://java.sun.com/j2se/1.4/docs/api/

WindowAdapter

An abstract adapter class for receiving window events. The methods in this class are empty. This class exists as convenience for creating listener objects.
Extend this class to create a WindowEvent listener and override the methods for the events of interest. (If you implement the WindowListener interface, you have to define all of the methods in it. This abstract class defines null methods for them all, so you can only have to define methods for events you care about.)

http://java.sun.com/j2se/1.4/docs/api/

WindowAdapter

Method Summary

void	**windowActivated**(WindowEvent e) Invoked when a window is activated.
void	**windowClosed**(WindowEvent e) Invoked when a window has been closed.
void	**windowClosing**(WindowEvent e) Invoked when a window is in the process of being closed.
void	**windowDeactivated**(WindowEvent e) Invoked when a window is de-activated.
void	**windowDeiconified**(WindowEvent e) Invoked when a window is de-iconified.
void	**windowGainedFocus**(WindowEvent e) Invoked when the Window is set to be the focused Window, which means that the Window, or one of its subcomponents, will receive keyboard events.
void	**windowIconified**(WindowEvent e) Invoked when a window is iconified.
void	**windowLostFocus**(WindowEvent e) Invoked when the Window is no longer the focused Window, which means that keyboard events will no longer be delivered to the Window or any of its subcomponents.
void	**windowOpened**(WindowEvent e) Invoked when a window has been opened.
void	**windowStateChanged**(WindowEvent e) Invoked when a window state is changed.

http://java.sun.com/j2se/1.4/docs/api/

Components' API

- In this section, we focus on the components previously introduced
- We will list the important methods and data members, as well as associated relevant events
- We do not cover the API in depth, especially since most functionality is covered in SunOne. Furthermore, there is no need to cover methods extensively, since most of the set---() methods are generated automatically in SunOne.
- Remember to have the toolTipText property set for all components.
- For full details, refer to http://java.sun.com/j2se/1.4/docs/api/

JMenuBar API

- Relevant event: ActionEvent on mouse click/activation
- Associated handler: ActionListener.actionPerformed
- Behavior: when activated, the submenu will be shown
- Useful properties: enabled, mnemonic,text

JMenu API

- Relevant event: ActionEvent on mouse click/activation
- Associated handler: ActionListener.actionPerformed
- Behavior: Show submenu if needed otherwise launch action of handler.
- “An implementation of a menu -- a popup window containing JMenuItems that is displayed when the user selects an item on the JMenuBar. In addition to JMenuItems, a JMenu can also contain JSeparators.
- In essence, a menu is a button with an associated JPopupMenu. When the "button" is pressed, the JPopupMenu appears. If the "button" is on the JMenuBar, the menu is a top-level window. If the "button" is another menu item, then the JPopupMenu is "pull-right" menu.“
- Useful properties: enabled, mnemonic, text

http://java.sun.com/j2se/1.4/docs/api/

JPanel API

- Relevant event: no specific event
- Associated handler: none
- Behavior: No observable behavior
- Useful properties: preferredSize

http://java.sun.com/j2se/1.4/docs/api/

JTabbedPane API

- Relevant event: no specific event
- Associated handler: none
- Behavior: Clicking on a tab automatically brings forward the JComponent object associated to it, and hides the other ones.
- Useful properties: tabPlacement
- Note: in order to change the name of the tabs, you can do so in the source code by changing the first parameter of the addTab() method.

http://java.sun.com/j2se/1.4/docs/api/

JRadioButton API

- Relevant event: ItemEvent, use getSource()
- Associated handler: ItemListener
- Behavior: All the radio buttons in the same ButtonGroup object are mutually exclusive.
- Useful properties: buttonGroup, enabled, selected, mnemonic, horizontalAlignment, horizontalTextPosition, text, verticalAlignment, verticalTextPosition

http://java.sun.com/j2se/1.4/docs/api/

JCheckBox API

- Relevant event: ItemEvent, use getSource() and SELECTED
- Associated handler: ItemListener
- Behavior: no particular behavior.
- Useful properties: buttonGroup, enabled, selected, mnemonic, horizontalAlignment, horizontalTextPosition, text, verticalAlignment, verticalTextPosition

http://java.sun.com/j2se/1.4/docs/api/

JLabel API

- Relevant event: no specific event
- Associated handler: none
- Behavior: no particular behavior.
- Useful properties: enabled, selected, mnemonic, horizontalAlignment, horizontalTextPosition, text, verticalAlignment, verticalTextPosition

http://java.sun.com/j2se/1.4/docs/api/

JButton API

- Relevant event: ActionEvent
- Associated handler: ActionListener
- Behavior: no particular behavior.
- Useful properties: enabled, horizontalAlignment, horizontalTextPosition, icon, preferredSize, text, verticalAlignment, verticalTextPosition

http://java.sun.com/j2se/1.4/docs/api/

JComboBox API

- Relevant event: ItemEvent, use the list's getSelectedIndex()
- Associated handler: ItemListener
- Behavior: no particular behavior.
- Useful properties: enabled, maximumRowCount, model, preferedSize, SelectedIndex
- Note: the strings held in the JComboBox are set with the model property

http://java.sun.com/j2se/1.4/docs/api/

JList API

- Relevant event: ListSelectionEvent, use the list's getSelectedIndex() or getSelectedValues()
- Associated handler: ListSelectionListener
- Behavior: there are different selection methods available, SINGLE_INTERVAL_SELECTION, MULTIPLE_INTERVAL_SELECTION, SINGLE_SELECTION, set in selectionMode
- Useful properties: enabled, layoutOrientiation, model, preferedSize, selectedIndex, selectionMode
- Note: the strings held in the JComboBox are set with the model property. Also, you need to embed it within a JScrollPane if you want scrollbars

http://java.sun.com/j2se/1.4/docs/api/

JTextField +JPasswordField API

- Relevant event: ActionEvent, when pressing the ‘enter’ key.
- Associated handler: ActionListener
- Behavior: JPasswordField hides the actual contents of the field. To fetch data in the fields: JPasswordField.getPassword(), ActionEvent.getActionCommand()
- Useful properties: editable, enabled, text

http://java.sun.com/j2se/1.4/docs/api/

JTextPane API

- Relevant event: no specific event
- Associated handler: none
- Behavior: JTextPane allows to have text with multiple font styles in it. Its semantics are a bit complex, and it is useful only if you need to do serious text entry.
- Useful properties: editable, enabled, text
- Note: You need to embed it within a JScrollPane if you want scrollbars. You should use it with a DefaultStyledDocument.

http://java.sun.com/j2se/1.4/docs/api/

JEditorPane API

- Relevant event: HyperlinkEvent
- Associated handler: HyperlinkListener
- Behavior: JEditorPane is the superclass of JTextPane, with HTML and RTF support. It also loads hyperlinks from URL objects

http://java.sun.com/j2se/1.4/docs/api/

JScrollPane API

- Relevant event: no specific event
- Associated handler: none
- Behavior: components objects within a JScrollPane now are able to be scrolled up and down, left and right if needed.
- Useful properties: enabled, horizontalScrollBarPolicy, verticalScrollBarPolicy, wheelScrollingEnabled

http://java.sun.com/j2se/1.4/docs/api/

References

- Sun’s tutorials and references:
  - http://developer.java.sun.com/developer/onlineTraining/GUI/Swing1/shortcourse.html
  - http://developer.java.sun.com/developer/onlineTraining/GUI/Swing2/shortcourse.html
  - http://java.sun.com/docs/books/tutorial/uiswing/events/eventsandcomponents.html
- *Java How to Program*, Deitel & Deitel, examples: http://www.deitel.com/books/downloads.html
  - You really should download the slides for chapters 11-15.