

JAVA Swing

Metodi Avanzati di Programmazione
Laurea Triennale in Informatica
Università degli Studi di Bari Aldo Moro
Docente: Pierpaolo Basile

SWING

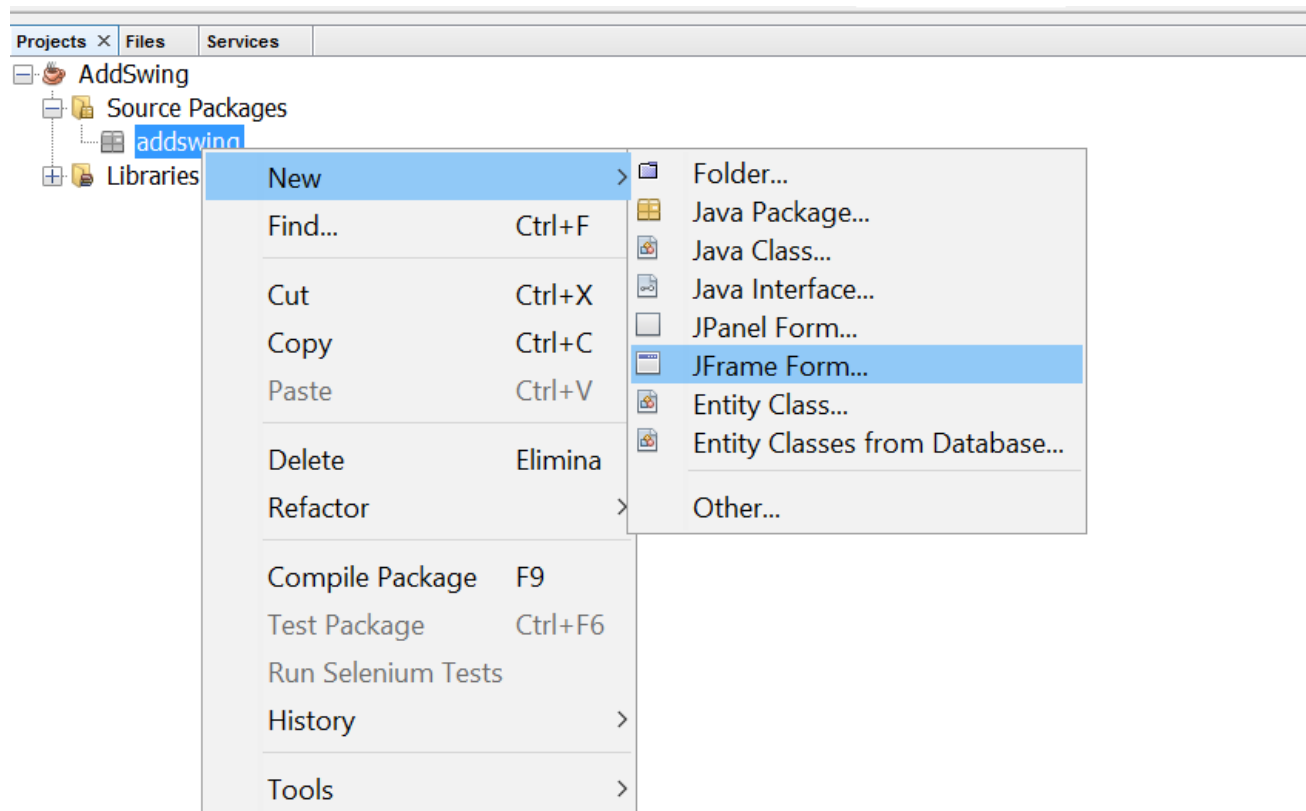
- E' il framework di JAVA che permette la realizzazione di interfacce grafiche (GUI)
 - Finestre, Form, Dialog
 - Menu, Pulsanti, Check-box, Combo-box
 - Alberi, Tabelle
 - Layout, Look&Feel
- Package di riferimento
 - *javax.swing, javax.swing.event*

NetBeans & SWING

- NetBeans mette a disposizione degli strumenti che facilitano la creazione delle GUI
 - drag and drop dei componenti
 - auto-generazione del codice
 - strumenti per la gestione del layout dei componenti

La prima applicazione SWING

- Aggiungere un nuovo JFrame



La prima applicazione SWING

The screenshot shows the NetBeans IDE 8.1 interface for creating a Swing application. The main window is titled "AddSwing - NetBeans IDE 8.1". The menu bar includes File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, and Help. The toolbar shows various icons for file operations and running the application. The left sidebar contains the "Projects" window, showing the project structure: "AddSwing" (Source Packages), "addswing" (Add.java), and "Libraries". The central area is the "Design" view, showing a blank canvas with a light gray background. The right sidebar contains the "Palette" window, which lists various Swing components organized into categories: "Swing Containers" (Panel, Tabbed Pane, Split Pane, Scroll Pane, Tool Bar, Desktop Pane, Internal Frame, Layered Pane), "Swing Controls" (Label, Button, Toggle Button, Check Box, Radio Button, Button Group, Combo Box, List, Text Field, Text Area, Scroll Bar, Slider, Progress Bar, Formatted Field, Password Field, Spinner, Separator, Text Pane, Editor Pane, Tree, Table), "Swing Menus", "Swing Windows", "Swing Fillers", "AWT", "Beans", and "Java Persistence". Below the Palette is the "Properties" window, which has tabs for "Properties", "Binding", "Events", and "Code". The "Properties" tab is active, showing a list of properties for the selected component (JFrame). The "Events" tab is also visible. Three red boxes with arrows point to the "Swing Controls" section of the Palette, the "Properties" window, and the "Events" tab in the Properties window.

Paletta di componenti SWING

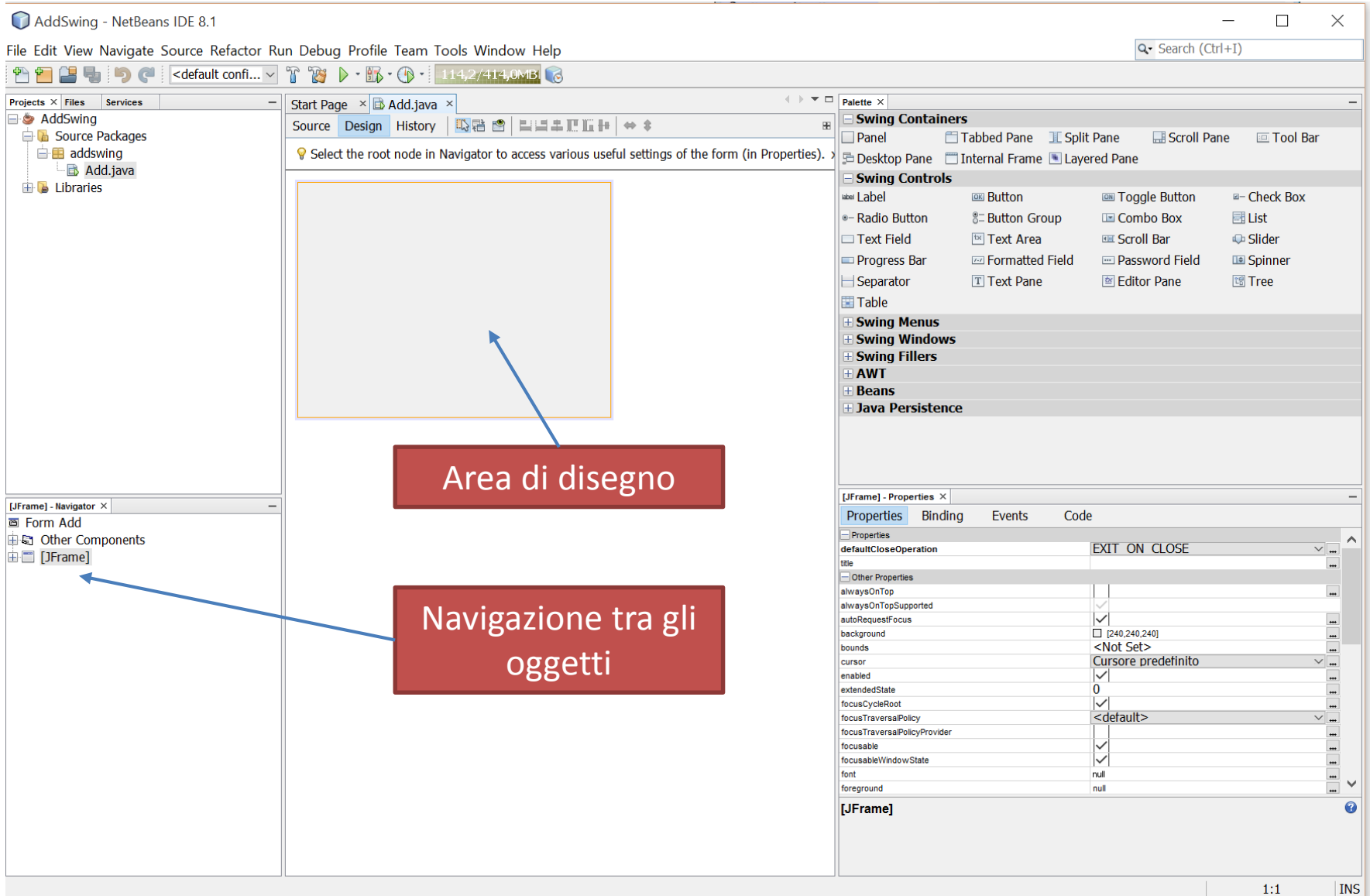
Proprietà degli oggetti

Eventi

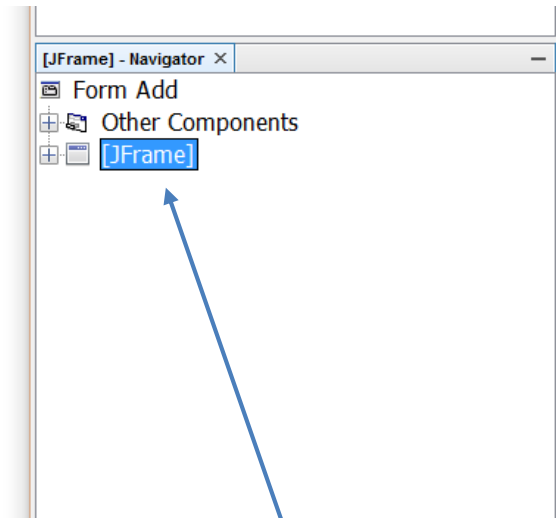
Property	Value
defaultCloseOperation	EXIT ON CLOSE
title	
alwaysOnTop	
alwaysOnTopSupported	
autoRequestFocus	
background	[240,240,240]
bounds	<Not Set>
cursor	Cursore predefinito
enabled	
extendedState	0
focusCycleRoot	
focusTraversalPolicy	<default>
focusTraversalPolicyProvider	
focusable	
focusableWindowState	
font	null
foreground	null

[JFrame]

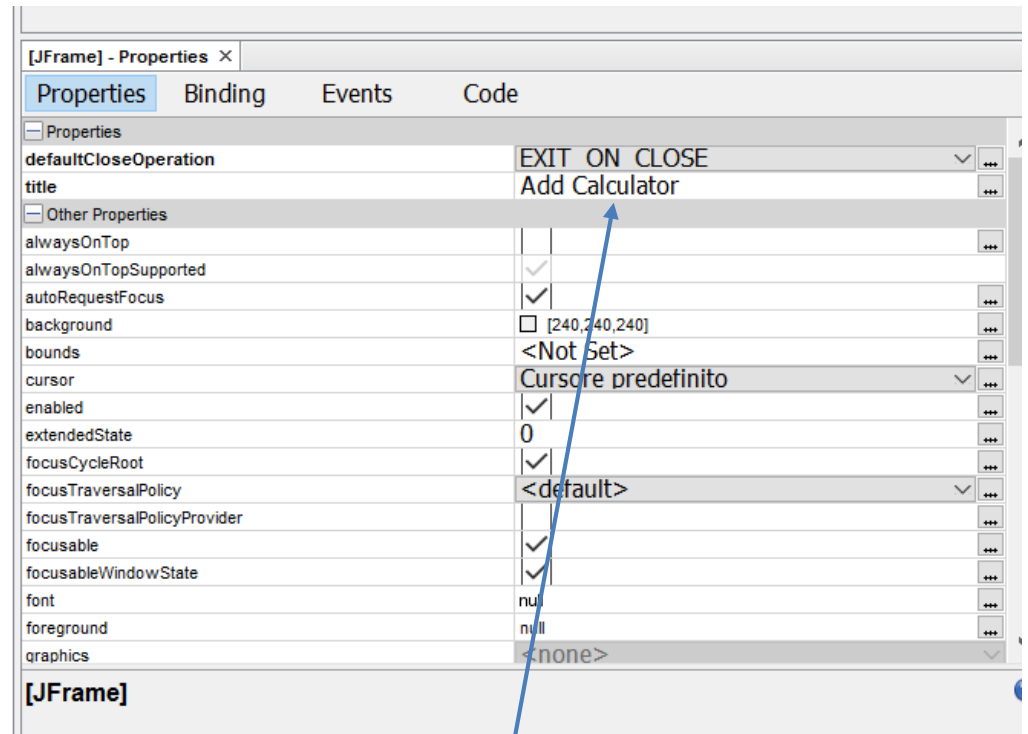
La prima applicazione SWING



La prima applicazione SWING



Seleziona il JFrame
tra i componenti



Modifica la proprietà
titolo (title)

La prima applicazione SWING

- Drag and drop delle componenti nell'area di disegno
- Selezionare le componenti dalla Palette
 - 2 JTextField per l'inserimento del testo (due operandi)
 - jtfOperand1, jtfOperand2
 - 1 JLabel per la stampa del risultato
 - jlResult
 - 1 JButton per effettuare la somma
 - jbAdd

La prima applicazione SWING

The screenshot shows the NetBeans IDE with the following components:

- Projects:** AddSwing, Source Packages, addswing, Add.java, Libraries.
- Source:** Add.java (Design mode). The Navigator window displays a tree hierarchy of components in the opened form.
- Design:** The main window is in Design mode, showing a JFrame with two text fields and a button. A red box highlights the text fields, with an arrow pointing to a callout box.
- Properties:** The Properties window shows the properties for the selected JTextField. The 'columns' property is set to 15, and the 'text' property is set to LEADING.

Callout Box:

JTextField per gli operandi

- cambiare la dimensione
- rimuovere il testo

Properties Window:

Properties	Binding	Events	Code
editable			<input checked="" type="checkbox"/>
background			[255,255,255]
columns			15
document			<default>
font			Tahoma 16 Plain
foreground			[0,0,0]
horizontalAlignment			LEADING
text			
tooltipText			
UI			<default>
UIClassID			TextFieldUI
action			<none>
alignmentX			0.5
alignmentY			0.5
autoscrolls			<input checked="" type="checkbox"/>
baselineResizeBehavior			CENTER OFFSET
border			[XPFIBorder]
caret			<default>

La prima applicazione SWING

The screenshot displays the NetBeans IDE environment for creating a Java Swing application. The main window is titled 'Add.java' and is in the 'Design' view. The design view shows a graphical user interface with two text input fields at the top, a 'Result:' label, and an 'Add' button below it. A red box highlights the 'Add' button, and a blue arrow points from a red text box to it. Another blue arrow points from the same red text box to the 'jbAdd [JButton]' component in the 'Other Components' list.

Le componenti presenti nel JForm

The 'Other Components' list shows the following components:

- Other Components
 - [JFrame]
 - jtfOperand1 [JTextField]
 - jtfOperand2 [JTextField]
 - jlResult [JLabel]
 - jbAdd [JButton]

The 'Palette' on the right lists various Swing components:

- Swing Containers
 - Panel
 - Tabbed Pane
 - Split Pane
 - Desktop Pane
 - Internal Frame
 - Layered Pane
- Swing Controls
 - Label
 - Button
 - Toggle Button
 - Radio Button
 - Button Group
 - ComboBox
 - Text Field
 - Text Area
 - ScrollPane
 - Progress Bar
 - Formatted Field
 - Password Field
 - Separator
 - Text Pane
 - Editor
 - Table
- Swing Menus
- Swing Windows
- Swing Fillers
- AWT
- Beans
- Java Persistence

The 'Properties' window at the bottom right shows the properties for the selected 'jbAdd [JButton]' component:

Properties	Binding	Events	Code
action			<none>
background			[240,240,240]
font			Tahoma 16
foreground			[0,0,0]
icon			<none>
mnemonic			
text			Add
toolTipText			
Other Properties			

La prima applicazione SWING

- La gestione degli eventi
 - quando il bottone (jbAdd) viene premuto l'applicazione deve calcolare la somma
 - leggere i valori dai JTextField
 - sommare i valori
 - modificare il testo della JLabel con il risultato
 - selezionare il componente (jbAdd) e il tab «Events» dalle proprietà dell'oggetto

La prima applicazione SWING

- Gestione dell'evento

The screenshot illustrates the process of setting an event listener for a JButton in NetBeans. The 'Form Add' window contains several components: jtfOperand1 [JTextField], jtfOperand2 [JTextField], jlResult [JLabel], and jbAdd [JButton]. A context menu is open over jbAdd, showing the 'Events' tab selected. The 'Events' tab in the 'jbAdd [JButton] - Properties' window is also selected, showing a list of events. The 'actionPerformed' event is selected in the list.

Properties	Binding	Events	Code
Events			
actionPerformed			<none>
ancestorAdded			<none>
ancestorMoved			<none>
ancestorMoved			<none>
ancestorRemoved			<none>
ancestorResized			<none>
caretPositionChanged			<none>
componentAdded			<none>
componentHidden			<none>
componentMoved			<none>
componentRemoved			<none>
componentResized			<none>
componentShown			<none>
focusGained			<none>
focusLost			<none>

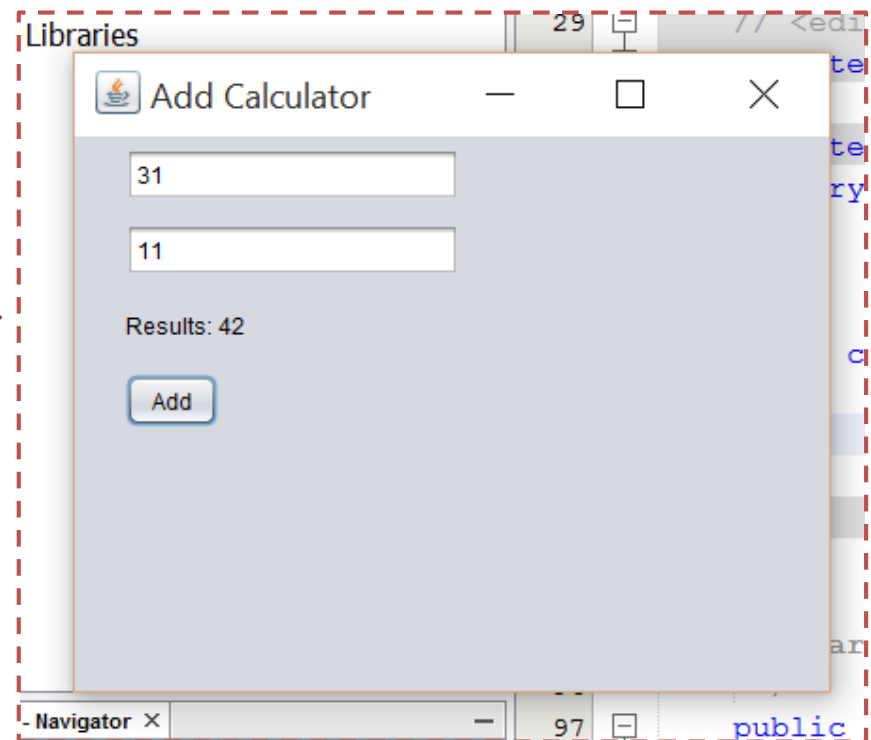
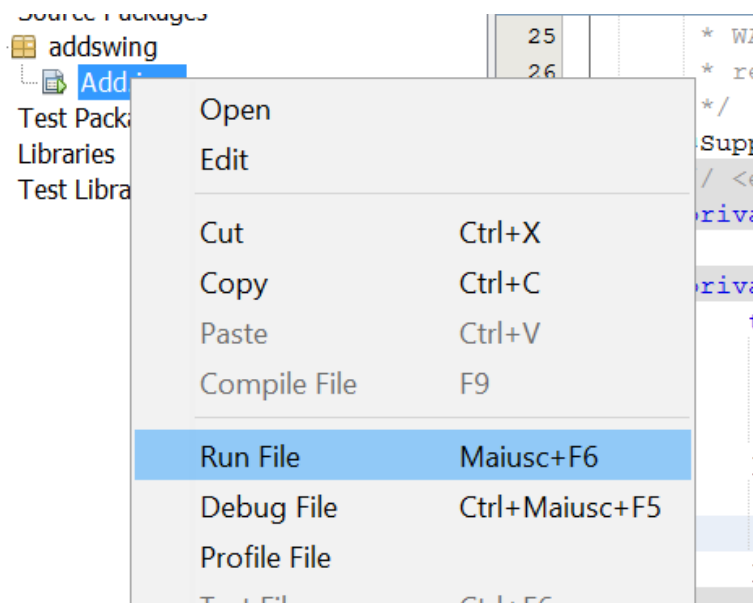
```
23      * WARNING: Do NOT modify this code. The content of this method is always
24      * regenerated by the Form Editor.
25      */
26      @SuppressWarnings("unchecked")
27      Generated Code
80
81      private void jbAddActionPerformed(java.awt.event.ActionEvent evt) {
82          // TODO add your handling code here:
83      }
84
```

La prima applicazione SWING

```
private void jbAddActionPerformed(java.awt.event.ActionEvent evt) {  
    try {  
        int op1 = Integer.parseInt(jtfOperand1.getText());  
        int op2 = Integer.parseInt(jtfOperand2.getText());  
        jlResult.setText("Results: " + (op1 + op2));  
    } catch (NumberFormatException nfex) {  
        JOptionPane.showMessageDialog(this, "No valid number " + nfex.getMessage(),  
            "Error", JOptionPane.ERROR_MESSAGE);  
    }  
}
```

- Lettura del testo dalle *JTextField* con il metodo *getText*
- Conversione delle stringhe in interi
- Stampa del risultato
- Blocco *try-catch* per catturare eventuali errori di formato degli interi
 - visualizzazione di un messaggio di errore tramite *JOptionPane*

Eseguire l'applicazione

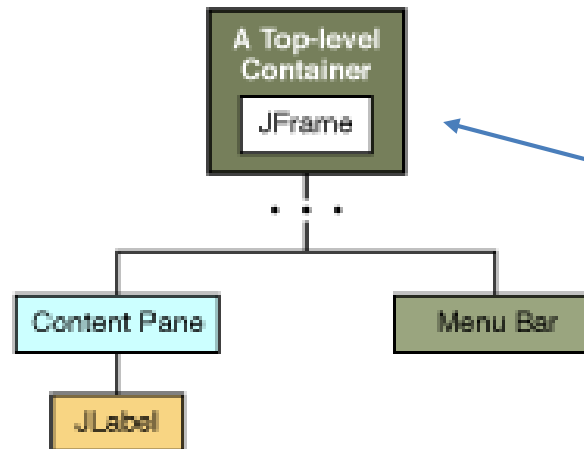
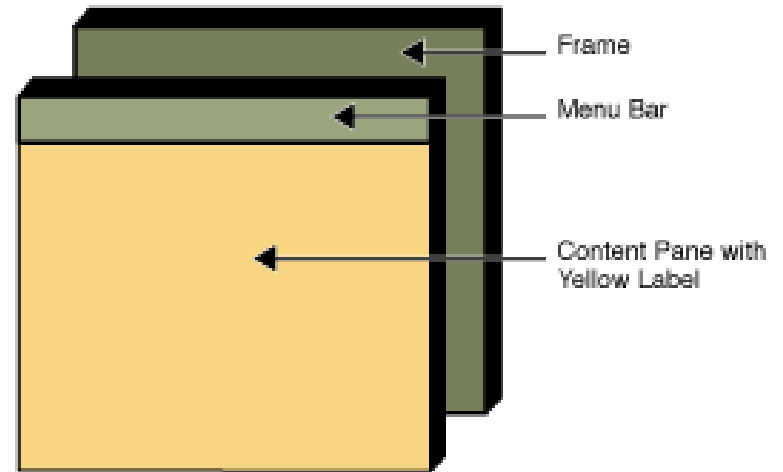
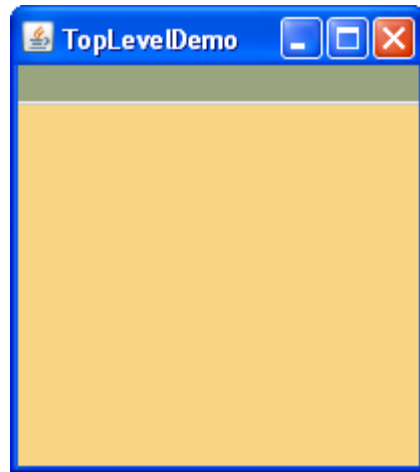


CONTENITORI DI ALTO LIVELLO

Contenitori

- JAVA Swing mette a disposizione tre tipi di contenitori JFrame, JDialog, JApplet
 - ogni componente deve far parte di una gerarchia di componenti connessi ad un contenitore radice (top-level)
 - un componente può appartenere ad un solo contenitore
 - ogni contenitore top-level è associato ad una vista
 - ovvero gli oggetti effettivamente visibili su schermo
 - è possibile aggiungere un menu ad un contenitore top-level

Contenitori



Gerarchia delle componenti

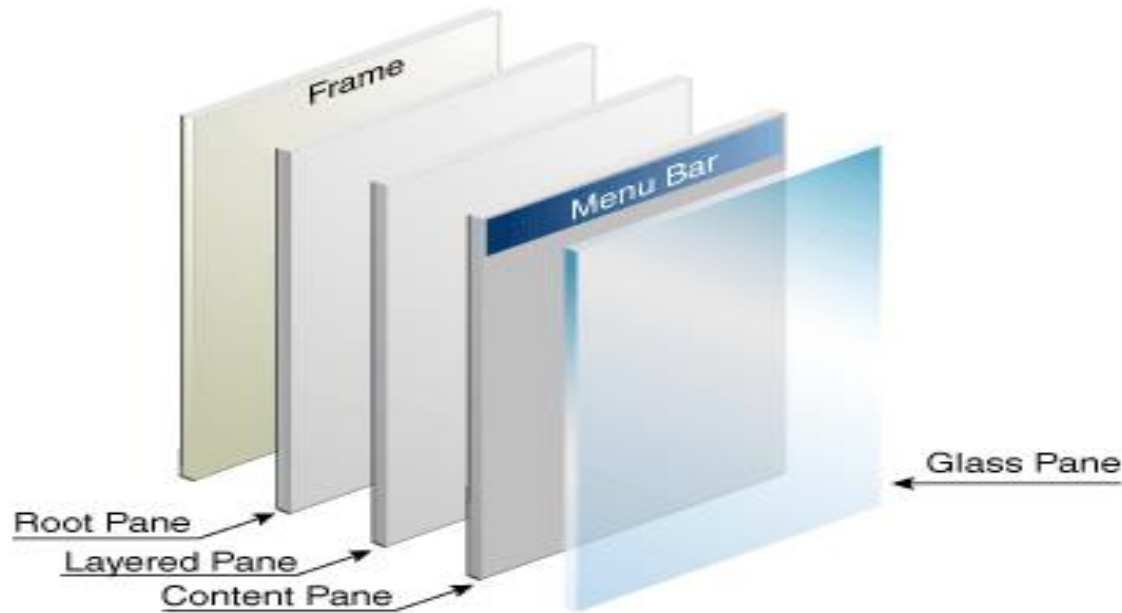
Contenitori

- Ogni programma SWING deve avere almeno un contenitore top-level
 - questo contenitore è la radice della gerarchia delle componenti (es. il JForm)
- Generalmente un'applicazione SWING avrà un JForm come contenitore top-level e radice

Aggiungere componenti

- Attraverso il metodo *add* è possibile aggiungere le componenti ad un contenitore
 - *frame.**getContentPane().add**(yellowLabel, BorderLayout.CENTER);*
 - *getContentPane()* restituisce un oggetto JComponent
- Aggiungere un menu
 - creare un oggetto di tipo JMenuBar
 - aggiungerlo al container
frame.setJMenuBar(greenMenuBar);

Root Pane



Root Pane

Gestisce il Content Pane e l'eventuale Menu

Layered Pane

Gestisce la disposizione dei componenti

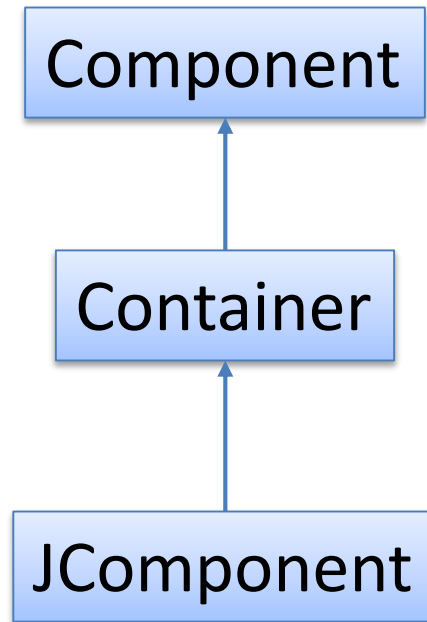
Content Pane

Contiene gli altri oggetti disposti secondo il layer

Glass Pane

Intercetta eventi esterni (es. mouse)

JComponent



JComponent

- eredita da Container che a sua volta eredita da Component
- mette a disposizione una set di metodi per modificare l'aspetto e il comportamento del componente

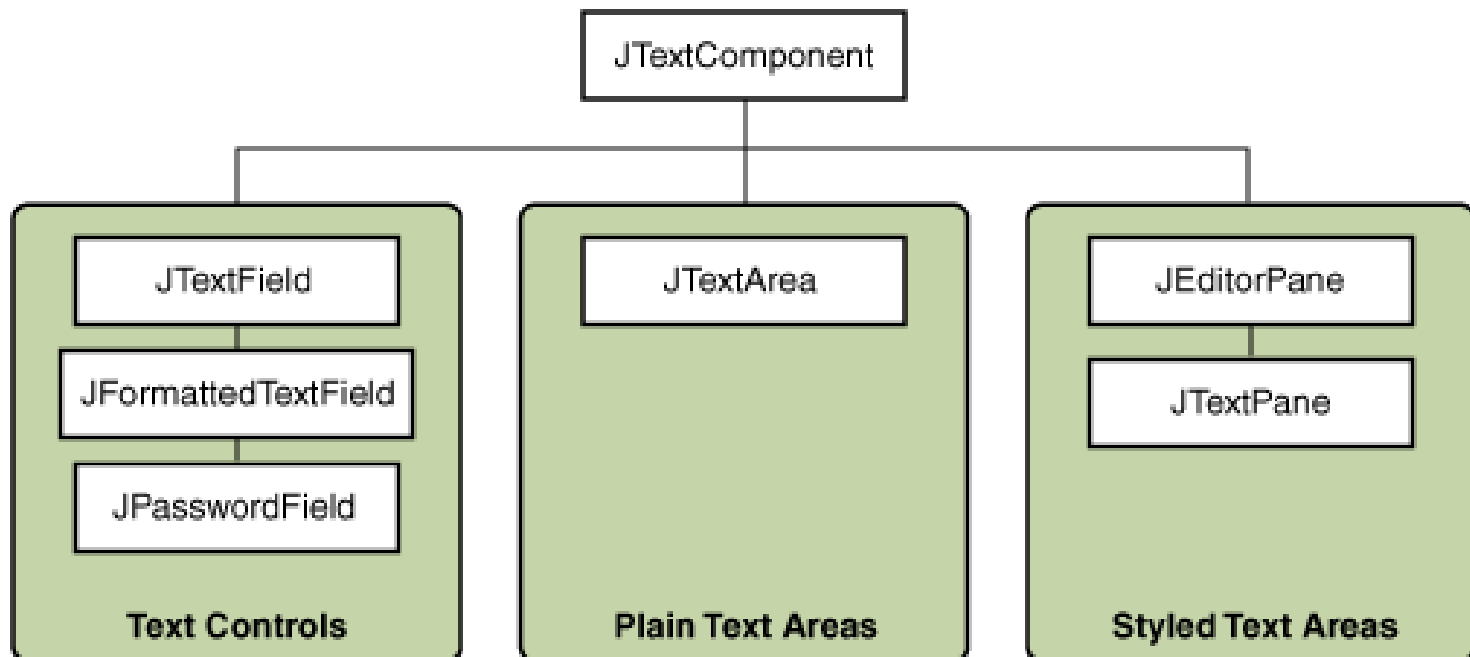
JComponent

- Modifica dell'aspetto del componente
 - colore, bordo, tipo cursore
- Monitoraggio dello stato
 - gestione di un PopupMenu, cambiare nome, visibile/non visibile, abilita/disabilita, modifica del ToolTip, nome
- Gestione degli eventi
- Disegno degli oggetti
- Gestione delle gerarchia di oggetti: aggiungi, rimuovi
- Modifica del Layout (disposizione degli oggetti)
- Dimensione e posizione del componente

COMPONENTI PER IL TESTO

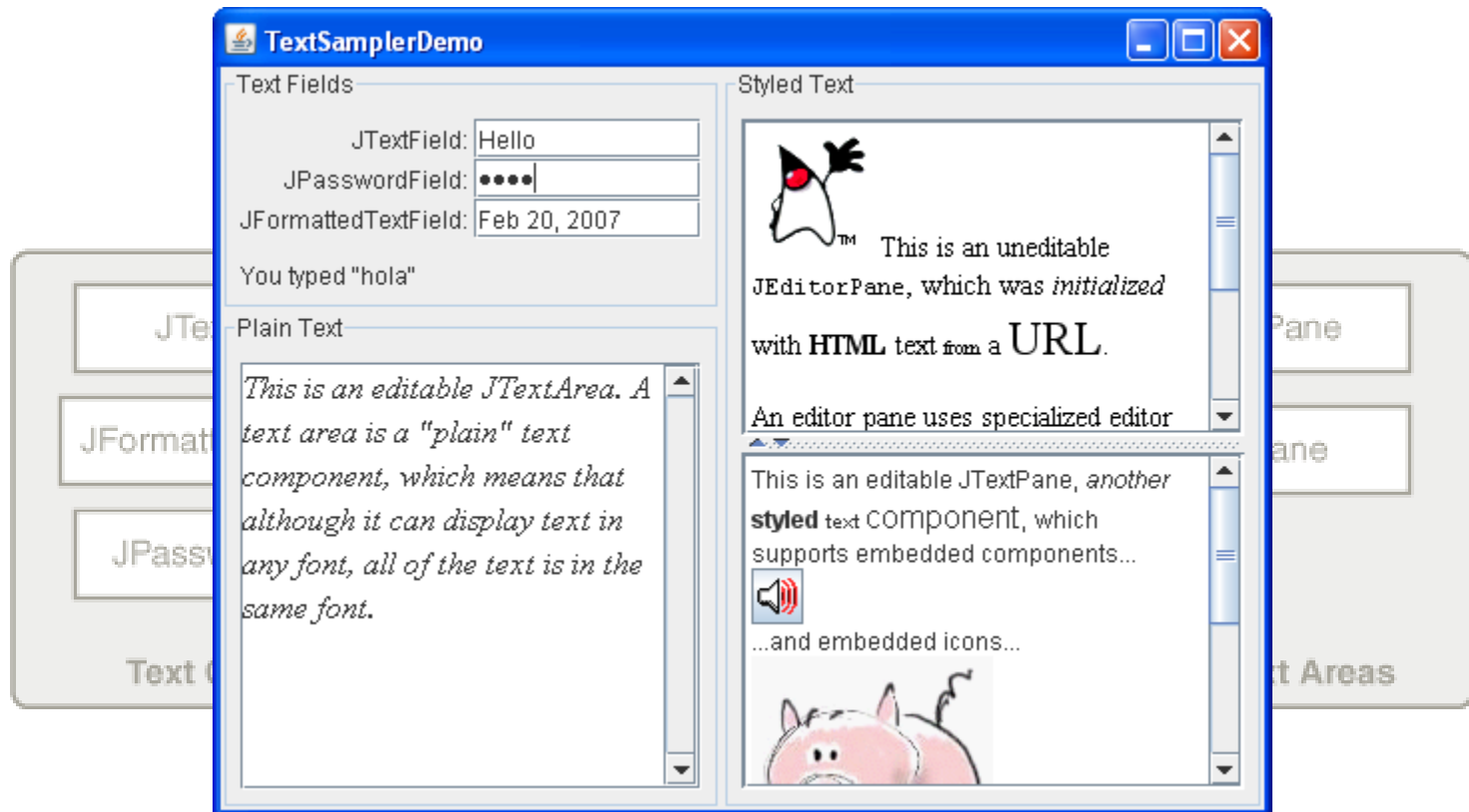
Componenti per il testo

- Permettono di visualizzare e modificare il testo



Componenti per il testo

- Permettono di visualizzare e modificare il testo



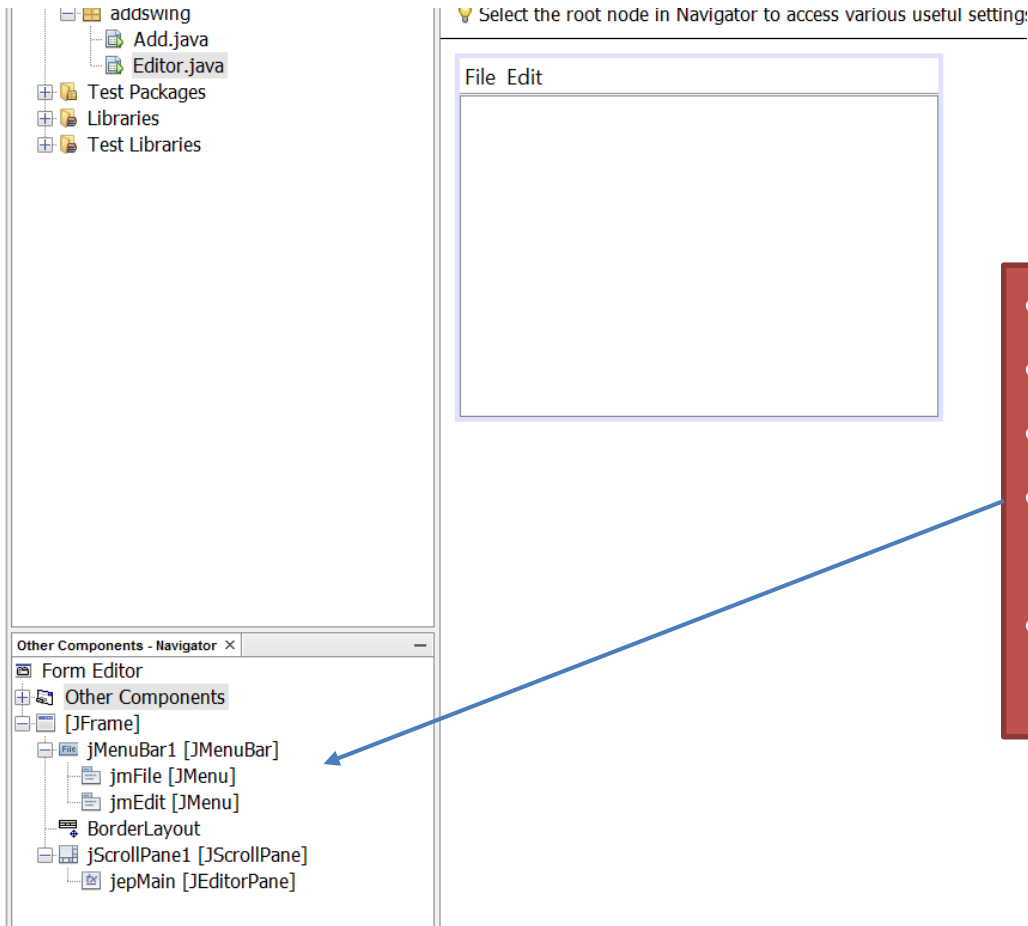
Componenti per il testo

Group	Description	Swing Classes
Text Controls	Also known simply as text fields, text controls can display only one line of editable text. Like buttons, they generate action events. Use them to get a small amount of textual information from the user and perform an action after the text entry is complete.	JTextField and its subclasses JPasswordField and JFormattedTextField
Plain Text Areas	JTextArea can display multiple lines of editable text. Although a text area can display text in any font, all of the text is in the same font. Use a text area to allow the user to enter unformatted text of any length or to display unformatted help information.	JTextArea
Styled Text Areas	A styled text component can display editable text using more than one font. Some styled text components allow embedded images and even embedded components. Styled text components are powerful and multi-faceted components suitable for high-end needs, and offer more avenues for customization than the other text components. Because they are so powerful and flexible, styled text components typically require more initial programming to set up and use. One exception is that editor panes can be easily loaded with formatted text from a URL, which makes them useful for displaying uneditable help information.	JEditorPane and its subclass JTextPane

JTextComponent

- Un modello di dati, detto *document*, che gestisce il contenuto
- Una vista che si occupa della visualizzazione del contenuto
- Un controller, detto *editor kit*, che legge e scrive il testo e permette le funzionalità di editing
- Un cursore che permette la navigazione nel contenuto

Un semplice editor



- Selezionare il BorderLayout
- Aggiungere una menu bar
- Aggiungere due menu (File, Edit)
- Aggiungere un JScrollPane
 - aggiungere un JEditorPane
- Aggiungere una toolbar a NORTH del frame

Un semplice editor (code)

```
private Map<Object, Action> editorActionTable;

private void initActionTable() {
    editorActionTable = new HashMap<>();
    Action[] actions = jepMain.getActions();
    for (Action action : actions) {
        editorActionTable.put(action.getValue(Action.NAME), action);
    }
}

private Action getEditorActionByName(String action) {
    return editorActionTable.get(action);
}

protected UndoManager undo = new UndoManager();

protected UndoAction undoAction = new UndoAction("Undo");

protected RedoAction redoAction = new RedoAction("Redo");

protected class MyUndoableEditListener implements UndoableEditListener {
    @Override
    public void undoableEditHappened(UndoableEditEvent e) {
        undo.addEdit(e.getEdit());
    }
}
```

Un semplice editor (code)

```
private Map<Object, Action> editorActionTable;

private void initActionTable() {
    editorActionTable = new HashMap<>();
    Action[] actions = jepMain.getActions();
    for (Action action : actions) {
        editorActionTable.put(action.getValue(Action.NAME), action);
    }
}

private Action getEditorActionByName(String action) {
    return editorActionTable.get(action);
}
```

```
protected UndoManager undo = new UndoManager();
```

- Creazione di una Map contenente le azioni di default dell'editor

```
protected RedoAction redoAction = new RedoAction("Redo");

protected class MyUndoableEditListener implements UndoableEditListener {

    @Override
    public void undoableEditHappened(UndoableEditEvent e) {
        undo.addEdit(e.getEdit());
    }

}
```

Un semplice editor (code)

```
private Map<Object, Action> editorActionTable;

private void initActionTable() {
    editorActionTable = new HashMap<>();
    Action[] actions = jepMain.getActions();
    for (Action action : actions) {
        editorActionTable.put(action.getValue(Action.NAME), action);
    }
}
```

- UndoManager gestisce le operazioni di undo/redo
- UndoAction e RedoAction sono due nuove Action che andremo a definire
- MyUndoableEventListener è il listener che gestisce gli eventi di undo/redo

```
protected UndoManager undo = new UndoManager();

protected UndoAction undoAction = new UndoAction("Undo");

protected RedoAction redoAction = new RedoAction("Redo");

protected class MyUndoableEditListener implements UndoableEditListener {

    @Override
    public void undoableEditHappened(UndoableEditEvent e) {
        undo.addEdit(e.getEdit());
    }

}
```

Action

- Quando la stessa funzionalità è accessibile da più controlli è utile creare delle Action
 - Per esempio la funzione copy può avvenire: tramite un menu, tramite una toolbar, tramite una combinazione di tasti. In questo caso è possibile creare un'unica Action che può essere associata ad ognuno di questi controlli
- Per creare un'azione è sufficiente estendere la classe astratta `AbstractAction`
- E' necessario implementare il metodo `actionPerformed` con la logica dell'azione, inoltre tramite una Action si possono modificare diverse proprietà:
 - Il testo che deve essere visualizzato nell'eventuale voce di menu o come tooltip
 - Le icone associate all'azione
 - Abilitare e disabilitare l'azione contemporaneamente su tutti i controlli a cui è associata l'azione
 - L'azione può essere direttamente aggiunta a vari controlli

Un semplice editor (action)

```
protected class UndoAction extends AbstractAction {

    public UndoAction(String name) {
        super(name);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        try {
            undo.undo();
        } catch (CannotUndoException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(), "Undo exception", JOptionPane.ERROR_MESSAGE);
        }
    }

}

protected class RedoAction extends AbstractAction {

    public RedoAction(String name) {
        super(name);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        try {
            undo.redo();
        } catch (CannotRedoException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(), "Redo exception", JOptionPane.ERROR_MESSAGE);
        }
    }

}
```

```

private void init() {
    initActionTable();
    undoAction.putValue(Action.SMALL_ICON, new ImageIcon("img/general/Undo16.gif"));
    undoAction.putValue(Action.LARGE_ICON_KEY, new ImageIcon("img/general/Undo24.gif"));
    jmEdit.add(undoAction);
    jToolBar1.add(undoAction);
    redoAction.putValue(Action.SMALL_ICON, new ImageIcon("img/general/Redo16.gif"));
    redoAction.putValue(Action.LARGE_ICON_KEY, new ImageIcon("img/general/Redo24.gif"));
    jmEdit.add(redoAction);
    jToolBar1.add(redoAction);
    jmEdit.add(new JSeparator());
    jToolBar1.add(new JSeparator());
    Action cutaction = getEditorActionByName(DefaultEditorKit.cutAction);
    cutaction.putValue(Action.NAME, "Cut");
    cutaction.putValue(Action.SMALL_ICON, new ImageIcon("img/general/Cut16.gif"));
    cutaction.putValue(Action.LARGE_ICON_KEY, new ImageIcon("img/general/Cut24.gif"));
    jmEdit.add(cutaction);
    jToolBar1.add(cutaction);
    Action copyaction = getEditorActionByName(DefaultEditorKit.copyAction);
    copyaction.putValue(Action.NAME, "Copy");
    copyaction.putValue(Action.SMALL_ICON, new ImageIcon("img/general/Copy16.gif"));
    copyaction.putValue(Action.LARGE_ICON_KEY, new ImageIcon("img/general/Copy24.gif"));
    jmEdit.add(copyaction);
    jToolBar1.add(copyaction);
    Action pasteaction = getEditorActionByName(DefaultEditorKit.pasteAction);
    pasteaction.putValue(Action.NAME, "Paste");
    pasteaction.putValue(Action.SMALL_ICON, new ImageIcon("img/general/Paste16.gif"));
    pasteaction.putValue(Action.LARGE_ICON_KEY, new ImageIcon("img/general/Paste24.gif"));
    jmEdit.add(pasteaction);
    jToolBar1.add(pasteaction);

    Keymap keymap = jepMain.getKeymap();
    KeyStroke keydown = KeyStroke.getKeyStroke(KeyEvent.VK_N, Event.CTRL_MASK);
    keymap.addActionForKeyStroke(keydown, getEditorActionByName(DefaultEditorKit.downAction));
    KeyStroke keyup = KeyStroke.getKeyStroke(KeyEvent.VK_P, Event.CTRL_MASK);
    keymap.addActionForKeyStroke(keyup, getEditorActionByName(DefaultEditorKit.upAction));

    jepMain.getDocument().addUndoableEditListener(new MyUndoableEditListener());
}

```

Un semplice editor (action)

```
private void init() {
    initActionTable();
    undoAction.putValue(Action.SMALL_ICON, new ImageIcon("img/general/Undo16.gif"));
    undoAction.putValue(Action.LARGE_ICON_KEY, new ImageIcon("img/general/Undo24.gif"));
    jmEdit.add(undoAction);
    jToolBar1.add(undoAction);
    redoAction.putValue(Action.SMALL_ICON, new ImageIcon("img/general/Redo16.gif"));
    redoAction.putValue(Action.LARGE_ICON_KEY, new ImageIcon("img/general/Redo24.gif"));
    jmEdit.add(redoAction);
    jToolBar1.add(redoAction);
    jmEdit.add(new JSeparator());
    jToolBar1.add(new JSeparator());
    Action cutAction = new EditorAction("Cut", "img/general/Cut16.gif", "img/general/Cut24.gif");
    Action copyAction = new EditorAction("Copy", "img/general/Copy16.gif", "img/general/Copy24.gif");
    Action pasteAction = new EditorAction("Paste", "img/general/Paste16.gif", "img/general/Paste24.gif");
    Action deleteAction = new EditorAction("Delete", "img/general/Delete16.gif", "img/general/Delete24.gif");
    Action selectAllAction = new EditorAction("Select All", "img/general/SelectAll16.gif", "img/general/SelectAll24.gif");
    Action findAction = new EditorAction("Find", "img/general/Find16.gif", "img/general/Find24.gif");
    Action replaceAction = new EditorAction("Replace", "img/general/Replace16.gif", "img/general/Replace24.gif");
    Action aboutAction = new EditorAction("About", "img/general/About16.gif", "img/general/About24.gif");
    Action quitAction = new EditorAction("Quit", "img/general/Quit16.gif", "img/general/Quit24.gif");
    Action helpAction = new EditorAction("Help", "img/general/Help16.gif", "img/general/Help24.gif");
    Action undoAction = new EditorAction("Undo", "img/general/Undo16.gif", "img/general/Undo24.gif");
    Action redoAction = new EditorAction("Redo", "img/general/Redo16.gif", "img/general/Redo24.gif");
    Action cutAction.putValue(Action.SMALL_ICON, new ImageIcon("img/general/Cut16.gif"));
    Action cutAction.putValue(Action.LARGE_ICON_KEY, new ImageIcon("img/general/Cut24.gif"));
    Action copyAction.putValue(Action.SMALL_ICON, new ImageIcon("img/general/Copy16.gif"));
    Action copyAction.putValue(Action.LARGE_ICON_KEY, new ImageIcon("img/general/Copy24.gif"));
    Action pasteAction.putValue(Action.SMALL_ICON, new ImageIcon("img/general/Paste16.gif"));
    Action pasteAction.putValue(Action.LARGE_ICON_KEY, new ImageIcon("img/general/Paste24.gif"));
    Action deleteAction.putValue(Action.SMALL_ICON, new ImageIcon("img/general/Delete16.gif"));
    Action deleteAction.putValue(Action.LARGE_ICON_KEY, new ImageIcon("img/general/Delete24.gif"));
    Action selectAllAction.putValue(Action.SMALL_ICON, new ImageIcon("img/general/SelectAll16.gif"));
    Action selectAllAction.putValue(Action.LARGE_ICON_KEY, new ImageIcon("img/general/SelectAll24.gif"));
    Action findAction.putValue(Action.SMALL_ICON, new ImageIcon("img/general/Find16.gif"));
    Action findAction.putValue(Action.LARGE_ICON_KEY, new ImageIcon("img/general/Find24.gif"));
    Action replaceAction.putValue(Action.SMALL_ICON, new ImageIcon("img/general/Replace16.gif"));
    Action replaceAction.putValue(Action.LARGE_ICON_KEY, new ImageIcon("img/general/Replace24.gif"));
    Action aboutAction.putValue(Action.SMALL_ICON, new ImageIcon("img/general/About16.gif"));
    Action aboutAction.putValue(Action.LARGE_ICON_KEY, new ImageIcon("img/general/About24.gif"));
    Action quitAction.putValue(Action.SMALL_ICON, new ImageIcon("img/general/Quit16.gif"));
    Action quitAction.putValue(Action.LARGE_ICON_KEY, new ImageIcon("img/general/Quit24.gif"));
    Action helpAction.putValue(Action.SMALL_ICON, new ImageIcon("img/general/Help16.gif"));
    Action helpAction.putValue(Action.LARGE_ICON_KEY, new ImageIcon("img/general/Help24.gif"));
    Action undoAction.putValue(Action.SMALL_ICON, new ImageIcon("img/general/Undo16.gif"));
    Action undoAction.putValue(Action.LARGE_ICON_KEY, new ImageIcon("img/general/Undo24.gif"));
    Action redoAction.putValue(Action.SMALL_ICON, new ImageIcon("img/general/Redo16.gif"));
    Action redoAction.putValue(Action.LARGE_ICON_KEY, new ImageIcon("img/general/Redo24.gif"));
    jmEdit.add(cutAction);
    jmEdit.add(copyAction);
    jmEdit.add(pasteAction);
    jmEdit.add(deleteAction);
    jmEdit.add(selectAllAction);
    jmEdit.add(findAction);
    jmEdit.add(replaceAction);
    jmEdit.add(aboutAction);
    jmEdit.add(quitAction);
    jmEdit.add(helpAction);
    jmEdit.add(undoAction);
    jmEdit.add(redoAction);
    jToolBar1.add(cutAction);
    jToolBar1.add(copyAction);
    jToolBar1.add(pasteAction);
    jToolBar1.add(deleteAction);
    jToolBar1.add(selectAllAction);
    jToolBar1.add(findAction);
    jToolBar1.add(replaceAction);
    jToolBar1.add(aboutAction);
    jToolBar1.add(quitAction);
    jToolBar1.add(helpAction);
    jToolBar1.add(undoAction);
    jToolBar1.add(redoAction);
}
```

- Inizializzazione della tabella delle action di default dell'editor
- Modifica delle proprietà delle action (SMALL/LARGE ICON)
- Aggiungere le action ai menu e alla toolbar

Un semplice editor (action)

```
jToolBar1.add(cutaction);  
Action copyaction = getEditorActionByName(DefaultEditorKit.copyAction);  
copyaction.putValue(Action.NAME, "Copy");  
copyaction.putValue(Action.SMALL_ICON, new ImageIcon("img/general/Copy16.gif"));  
copyaction.putValue(Action.LARGE_ICON_KEY, new ImageIcon("img/general/Copy24.gif"));  
jmEdit.add(copyaction);  
jToolBar1.add(copyaction);  
Action pasteaction = getEditorActionByName(DefaultEditorKit.pasteAction);  
pasteaction.putValue(Action.NAME, "Cut");  
pasteaction.putValue(Action.SMALL_ICON, new ImageIcon("img/general/Paste16.gif"));  
pasteaction.putValue(Action.LARGE_ICON_KEY, new ImageIcon("img/general/Paste24.gif"));
```

- Modifica delle proprietà delle action (NAME, SMALL e LARGE ICON) delle action di default dell'editor

Un semplice editor (key stroke)

```
Keymap keymap = jepMain.getKeymap();
KeyStroke keydown = KeyStroke.getKeyStroke(KeyEvent.VK_N, Event.CTRL_MASK);
keymap.addActionForKeyStroke(keydown, getEditorActionByName(DefaultEditorKit.downAction));
KeyStroke keyup = KeyStroke.getKeyStroke(KeyEvent.VK_P, Event.CTRL_MASK);
keymap.addActionForKeyStroke(keyup, getEditorActionByName(DefaultEditorKit.upAction));

jepMain.getDocument().addUndoableEditListener(new MyUndoableEditListener());
```

- Aggiunge delle nuove azioni collegate alla pressione di alcune combinazioni di tasti
- In questo caso colleghiamo la pressione di alcune combinazioni di tasti ad eventi di default dell'editor

Un semplice editor (undo/redo...)

```
Keymap keymap = jepMain.getKeymap();
KeyStroke keydown = KeyStroke.getKeyStroke(KeyEvent.VK_N, Event.CTRL_MASK);
keymap.addActionForKeyStroke(keydown, getEditorActionByName(DefaultEditorKit.downAction));
KeyStroke keyup = KeyStroke.getKeyStroke(KeyEvent.VK_P, Event.CTRL_MASK);
keymap.addActionForKeyStroke(keyup, getEditorActionByName(DefaultEditorKit.upAction));

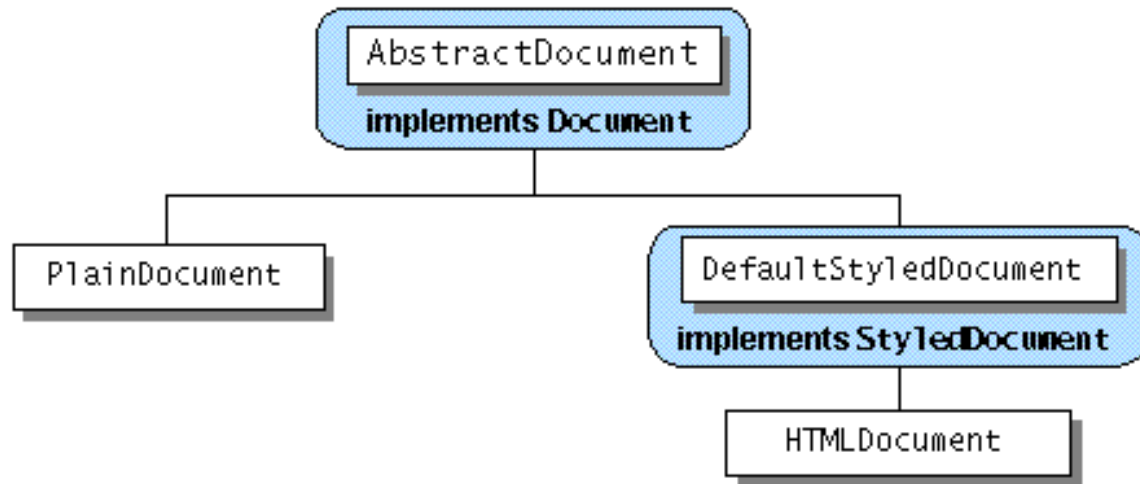
jepMain.getDocument().addUndoableEditListener(new MyUndoableEditListener());
}
```

- Aggiunge il listener per il undo/redo al Document dell'editor pane

Document

- Il modello dei dati di un TextComponent è il Document (interfaccia Document)
 - contiene il testo suddiviso in oggetti di tipo Element
 - supporta l'editing del testo (aggiungi/rimuovi)
 - notifica degli eventi di modifica del testo
 - gestisci gli oggetti di tipo Position che tracciano delle porzioni di testo anche se vengono modificate
 - fornisce informazioni su testo (es. lunghezza)

Document



StyledDocument è un Document con informazioni sullo stile del testo (utilizzato da JTextPane)

PlainDocument è il Document di default di JTextField, JTextArea, JPasswordField

DocumentListener

- Document notifica tutti i DocumentListener associati ad esso
 - inserimenti
 - rimozioni
 - modifica dello stile
- Aggiunta di un DocumentListener ad un document
 - `doc.addDocumentListener(new MyDocumentListener());`
- CaretListener permette di ricevere tutti gli eventi di modifica del cursore, deve essere aggiunto ad un JTextComponent

DocumentListener

```
protected class MyDocumentListener implements DocumentListener {  
  
    public void insertUpdate(DocumentEvent e) {  
        displayEditInfo(e);  
    }  
  
    public void removeUpdate(DocumentEvent e) {  
        displayEditInfo(e);  
    }  
  
    public void changedUpdate(DocumentEvent e) {  
        displayEditInfo(e);  
    }  
  
    private void displayEditInfo(DocumentEvent e) {  
        Document document = (Document) e.getDocument();  
        int changeLength = e.getLength();  
        System.out.println(e.getType().toString() + ": "  
            + changeLength + " character"  
            + ((changeLength == 1) ? ". " : "s. ")  
            + " Text length = " + document.getLength()  
            + ".");  
    }  
}
```

DocumentListener

```
protected class MyDocumentListener implements DocumentListener {

    public void init() {
        jmEdit.add(undoAction);
        jmEdit.add(redoAction);
        jmEdit.add(new JSeparator());
        editorActionMap = createActionTable(jepMain);
        jmEdit.add(getActionByName(DefaultEditorKit.cutAction));
        jmEdit.add(getActionByName(DefaultEditorKit.copyAction));
        jmEdit.add(getActionByName(DefaultEditorKit.pasteAction));

        InputMap inputMap = jepMain.getInputMap();
        KeyStroke key1 = KeyStroke.getKeyStroke(KeyEvent.VK_N, Event.CTRL_MASK);
        inputMap.put(key1, DefaultEditorKit.downAction);
        KeyStroke key2 = KeyStroke.getKeyStroke(KeyEvent.VK_P, Event.CTRL_MASK);
        inputMap.put(key2, DefaultEditorKit.upAction);

        jepMain.getDocument().addUndoableEditListener(new MyUndoableEditListener());
        jepMain.getDocument().addDocumentListener(new MyDocumentListener());
    }

    + " Text length = " + document.getLength()
    + ".");
}
```

Un semplice editor (Open File)

- Inserire un MenuItem nel Menu File e rinominarlo in jmiOpen
- Aggiungere l'evento ActionPerformed a jmiOpen

```
private void jmiOpenActionPerformed(java.awt.event.ActionEvent evt) {  
    JFileChooser chooser = new JFileChooser();  
    chooser.setMultiSelectionEnabled(false);  
    chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);  
    if (chooser.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {  
        File file = chooser.getSelectedFile();  
        try {  
            BufferedReader reader = new BufferedReader(new FileReader(file));  
            StringBuilder sb = new StringBuilder();  
            while (reader.ready()) {  
                sb.append(reader.readLine()).append("\n");  
            }  
            reader.close();  
            jepMain.setText(sb.toString());  
        } catch (IOException ioex) {  
            JOptionPane.showMessageDialog(this, "Error to load file:\n" + ioex.getMessage(),  
                "Error", JOptionPane.ERROR_MESSAGE);  
        }  
    }  
}
```

JFileChooser apre una finestra di dialogo per la selezione di file o directory

Un semplice editor (Open File)

- Lettura del contenuto testuale dal file
 - BufferedReader+StringBuilder
- Utilizzare il metodo setText di JEditorPane per impostare il testo

```
private void jmiOpenActionPerformed(java.awt.event.ActionEvent evt) {  
    JFileChooser chooser = new JFileChooser();  
    chooser.setMultiSelectionEnabled(false);  
    chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);  
    if (chooser.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {  
        File file = chooser.getSelectedFile();  
        try {  
            BufferedReader reader = new BufferedReader(new FileReader(file));  
            StringBuilder sb = new StringBuilder();  
            while (reader.ready()) {  
                sb.append(reader.readLine()).append("\n");  
            }  
            reader.close();  
            jepMain.setText(sb.toString());  
        } catch (IOException ioex) {  
            JOptionPane.showMessageDialog(this, "Error to load file:\n" + ioex.getMessage(),  
                "Error", JOptionPane.ERROR_MESSAGE);  
        }  
    }  
}
```

Implementare la funzione salva

- Visualizzare un JFileChooser in modalità save
- Usare un FileWriter e il metodo getText di JEditorPane

```
private void jmiSaveActionPerformed(java.awt.event.ActionEvent evt) {  
    JFileChooser chooser = new JFileChooser();  
    chooser.setMultiSelectionEnabled(false);  
    chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);  
    if (chooser.showSaveDialog(this) == JFileChooser.APPROVE_OPTION) {  
        File file = chooser.getSelectedFile();  
        try {  
            FileWriter writer = new FileWriter(file);  
            writer.append(jepMain.getText());  
            writer.close();  
        } catch (IOException ioex) {  
            JOptionPane.showMessageDialog(this, "Error to save file:\n" + ioex.getMessage(),  
                "Error", JOptionPane.ERROR_MESSAGE);  
        }  
    }  
}
```

Esercizio...

- Prima di creare un nuovo documento o aprire un documento ricordare all'utente di salvare solo nel caso in cui il documento sia stato modificato dall'ultimo salvataggio
 - Utilizzare un `DocumentListener` per tenere traccia delle modifiche
 - Utilizzare `JOptionPane.showConfirmDialog` per visualizzare una finestra di dialogo
`YES_NO_CANCEL`

...Esercizio

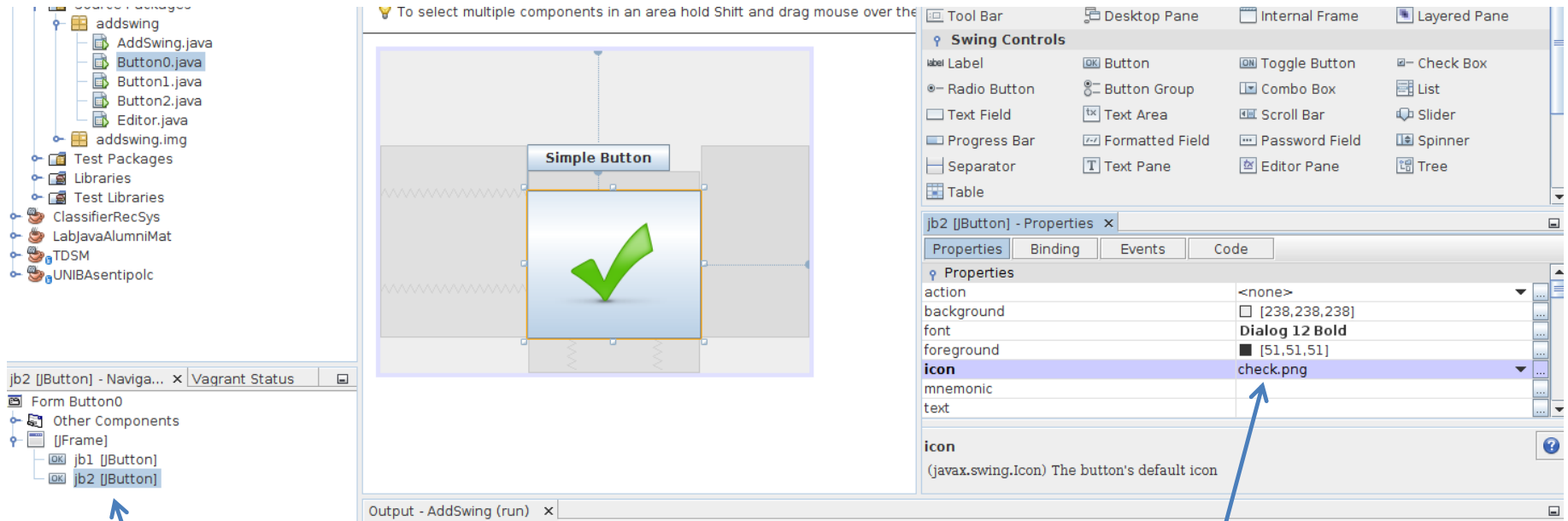
```
int option=JOptionPane.showConfirmDialog(this, "Save?", "Save", JOptionPane.YES_NO_CANCEL_OPTION);  
if (option==JOptionPane.YES_OPTION) {  
  
} else if (option==JOptionPane.NO_OPTION) {  
  
} else if (option==JOptionPane.CANCEL_OPTION) {  
  
}
```


**BUTTONS, CHECK BOXES, RADIO
BUTTONS**

Buttons

- Ereditano da **AbstractButton**
 - **JButton**: un classico pulsante
 - **JCheckBox**: una check box
 - **JRadioButton**: un singolo pulsante di un gruppo di pulsanti di opzione
 - **JMenuItem**: una voce di un menu
 - **JCheckBoxMenuItem**: una voce di un menu che è una check box
 - **JRadioButtonMenuItem**: una voce di un menu che è un pulsante di opzione
 - **JToggleButton**: permette di creare un bottone a due stati utilizzando due check box o due radio button

JButton



Inserire due
bottoni

Associare un'icona a jb2 e
cancellare il testo
(copiare un file png in un package
del progetto)

JButton

The screenshot displays an IDE interface with several components:

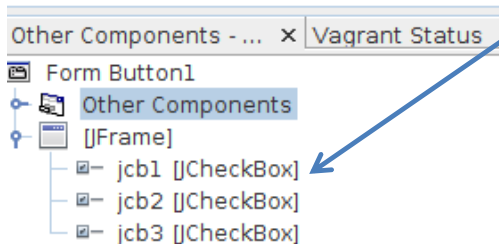
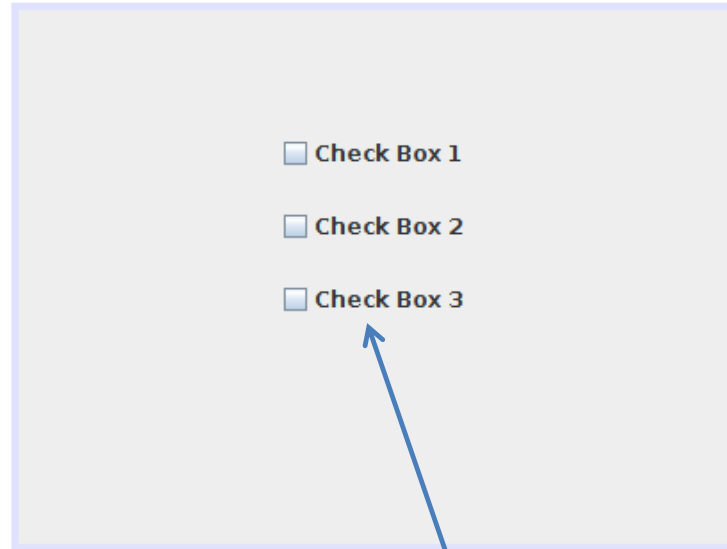
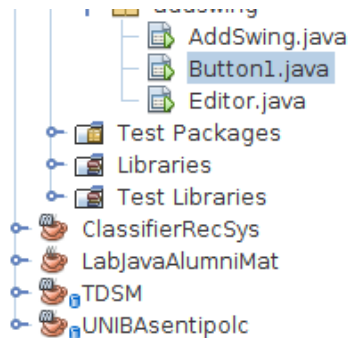
- Left Panel (Project Explorer):** Shows a project structure with folders like 'addswing' and 'Test Packages', and files such as 'AddSwing.java', 'Button0.java', 'Button1.java', 'Button2.java', and 'Editor.java'.
- Center Panel (Design View):** Displays a visual representation of a button labeled 'Simple Button'.
- Right Panel (Swing Controls):** Lists various Swing components including Button, Toggle Button, Check Box, Radio Button, Button Group, Combo Box, List, Text Field, Text Area, Scroll Bar, Slider, Progress Bar, Formatted Field, Password Field, Spinner, Separator, Text Pane, Editor Pane, Tree, and Table.
- Properties Window:** Shows the 'jb2 [JButton] - Properties' window with tabs for Properties, Binding, Events, and Code.
- Code Editor:** Contains the following Java code snippet:

```
private void jb2ActionPerformed(java.awt.event.ActionEvent evt) {  
    JOptionPane.showMessageDialog(null, "Ok");  
}
```
- Bottom Panel:** Includes a 'Form Button0' section, a 'Other Components' section with 'JFrame', and an 'Output - AddSwing (run)' section.

A blue arrow points from the 'jb2 [JButton]' component in the 'Other Components' section to the 'jb2ActionPerformed' method in the code editor.

Associare un evento actionPerformed a jb2

JCheckBox...



Inserire tre check box, cambiare nome variabile (jcb1, ...) e la proprietà *text* (Check Box 1, ...)

...JCheckBox

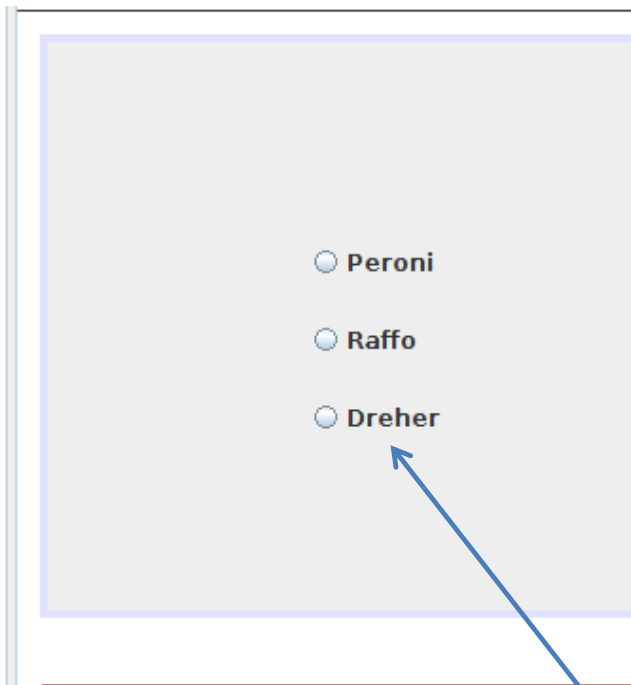
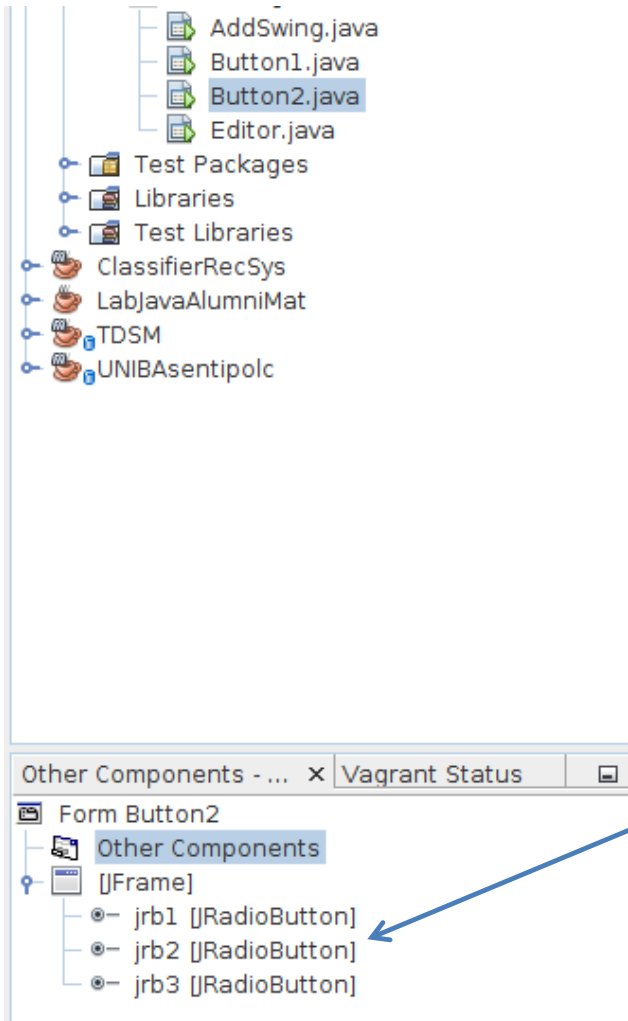
Visualizzare un messaggio quando la check box viene selezionata

```
private void jcb1ActionPerformed(java.awt.event.ActionEvent evt) {  
    if (jcb1.isSelected()) {  
        JOptionPane.showMessageDialog(this, "Check Box 1 is selected!");  
    }  
}
```

```
private void jcb2ActionPerformed(java.awt.event.ActionEvent evt) {  
    if (jcb2.isSelected()) {  
        JOptionPane.showMessageDialog(this, "Check Box 2 is selected!");  
    }  
}
```

```
private void jcb3ActionPerformed(java.awt.event.ActionEvent evt) {  
    if (jcb3.isSelected()) {  
        JOptionPane.showMessageDialog(this, "Check Box 3 is selected!");  
    }  
}
```

JRadioButton



Inserire tre radio button, cambiare nome variabile (jrb1, ...) e la proprietà *text* (Peroni, Raffo, Dreher)

JRadioButton

```
public Button2() {  
    initComponents();  
    init();  
}
```

```
private void init() {  
    BeerAction ba=new BeerAction();  
    jrb1.addActionListener(ba);  
    jrb2.addActionListener(ba);  
    jrb3.addActionListener(ba);  
    ButtonGroup beers=new ButtonGroup();  
    beers.add(jrb1);  
    beers.add(jrb2);  
    beers.add(jrb3);  
}
```

```
protected class BeerAction implements ActionListener {
```

```
    @Override
```

```
    public void actionPerformed(ActionEvent e) {
```

```
        switch (e.getActionCommand()) {
```

```
            case "Peroni":JOptionPane.showMessageDialog(null, "Scommetto che vieni da Bari.");
```

```
            break;
```

```
            case "Raffo":JOptionPane.showMessageDialog(null, "Scommetto che vieni da Taranto.");
```

```
            break;
```

```
            case "Dreher":JOptionPane.showMessageDialog(null, "Scommetto che non capisci nulla in fatto di birre!");
```

```
            break;
```

```
        }
```

```
    }
```

```
}
```

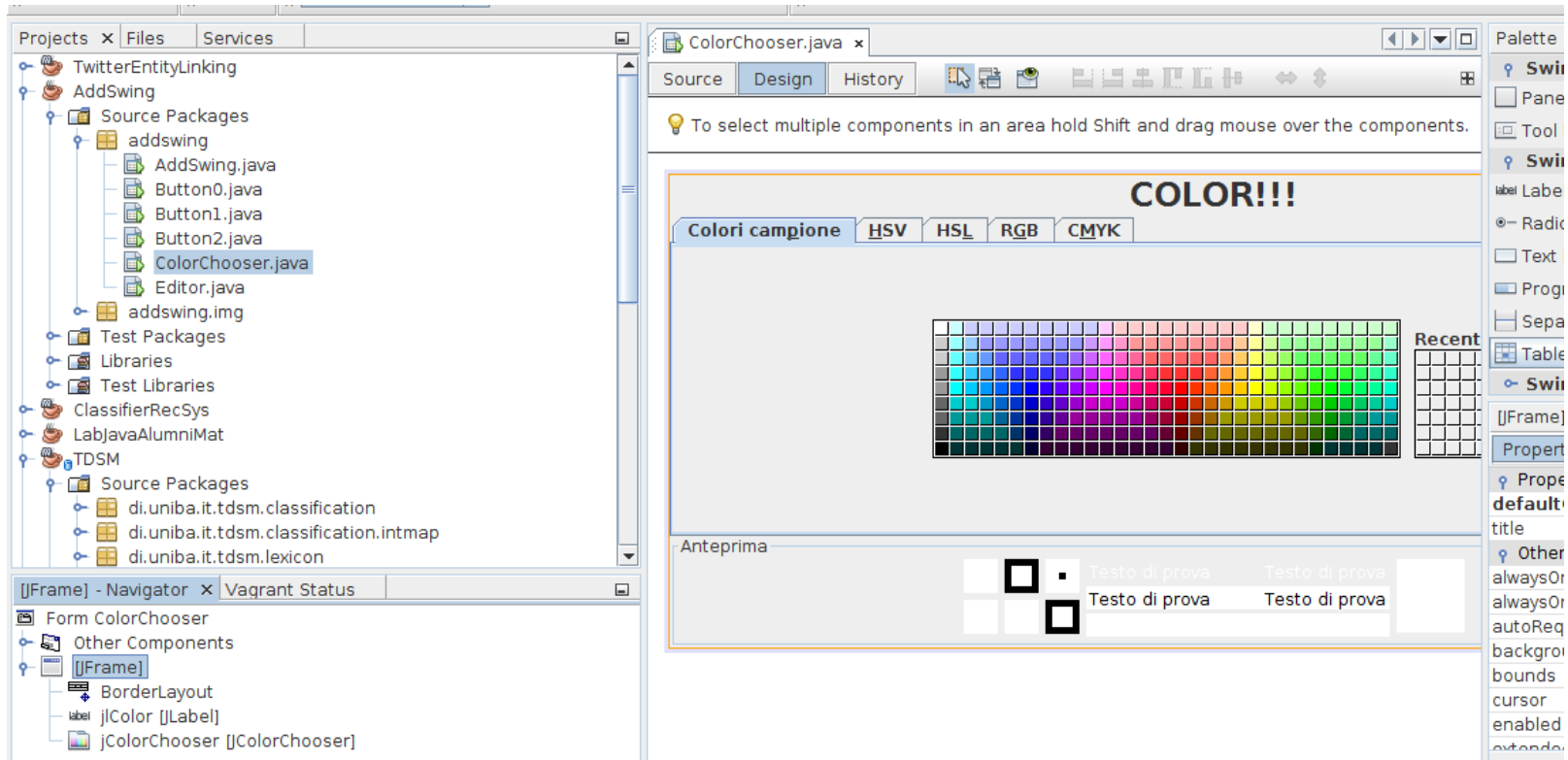
1. Creare un metodo init
2. Creiamo un nuovo ActionListener e lo associamo ai bottini
3. I radio button vanno raggruppati in un Button Group

COLOR CHOOSER

Color Chooser

- Implementato dalla classe JColorChooser
- Permette di selezionare un colore utilizzando diverse modalità
- ColorSelectionModel è il modello che contiene le informazioni sul colore selezionato
 - ogni JColorChooser ha un'istanza del selection model
- Implementare una classe ChangeListener per intercettare la selezione di un nuovo colore

Color Chooser



- Impostare il BorderLayout
- Aggiungere una JLabel (jColor) a nord e impostare la proprietà text («COLOR!!!»), aumentare la dimensione del font
- Aggiungere un JColorChooser (jColorChooser) al centro

Color Chooser

```
L
[-]
public ColorChooser() {
    initComponents();
    init();
}

[-]
private void init() {
    jColorChooser.setColor(Color.black);
    jColorChooser.getSelectionModel().addChangeListener(new MyColorChangeListener());
}

[-]
protected class MyColorChangeListener implements ChangeListener {
    @Override
    public void stateChanged(ChangeEvent e) {
        ColorSelectionModel selModel = (ColorSelectionModel) e.getSource();
        jlColor.setForeground(selModel.getSelectedColor());
    }
}
```

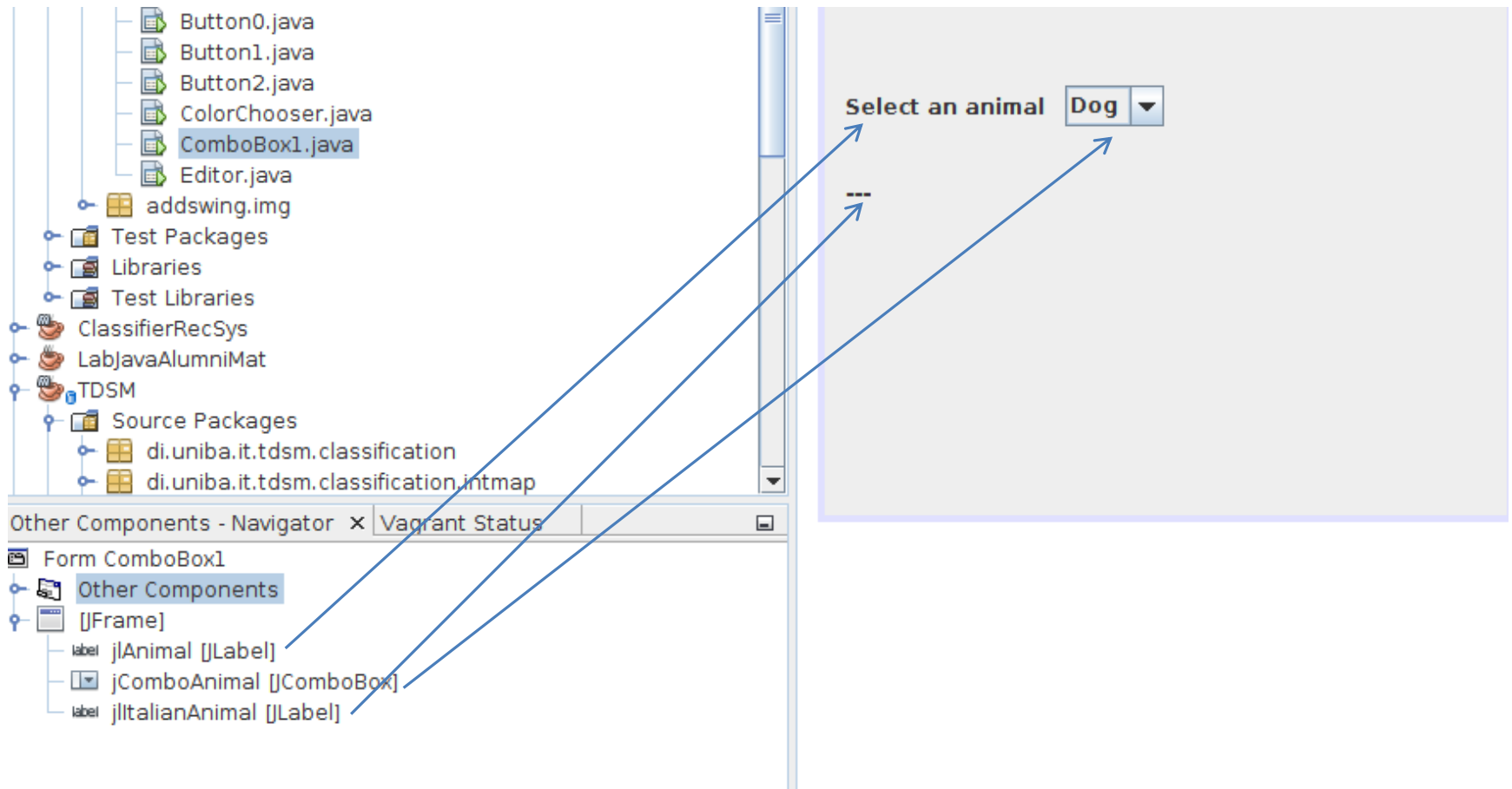
Intercettiamo la selezione di un nuovo color e cambiamo il colore della label (jlColor)

COMBO BOX

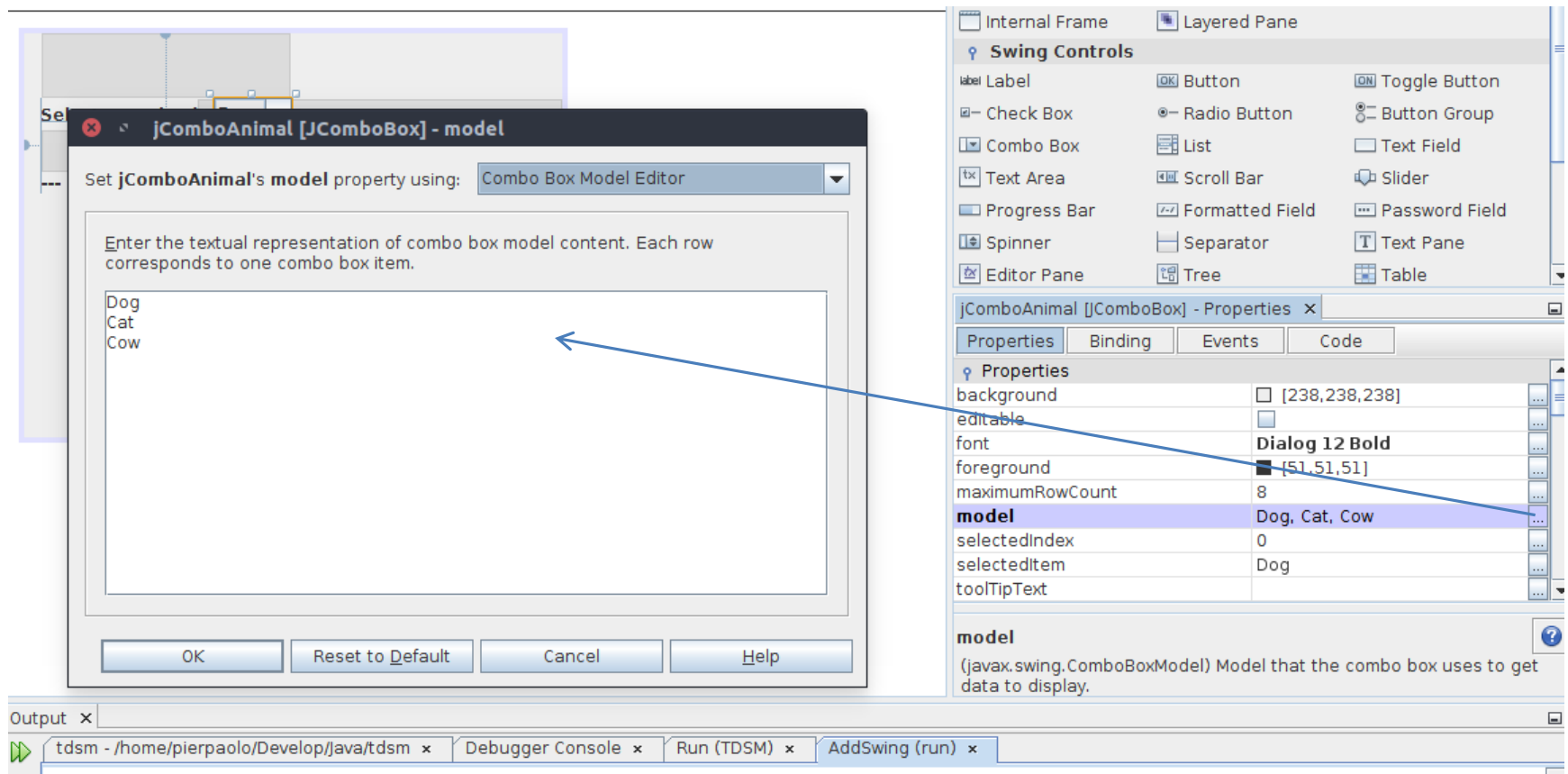
Combo Box

- Una combo box permette di effettuare una selezione tra diverse opzioni possibili
 - Editabile: permette di inserire un valore non presente nella lista delle opzioni possibili
 - Non editabile: permette la selezione solo tra le opzioni possibili
- Queste funzionalità sono implementate nella classe JComboBox

Combo Box



Combo Box

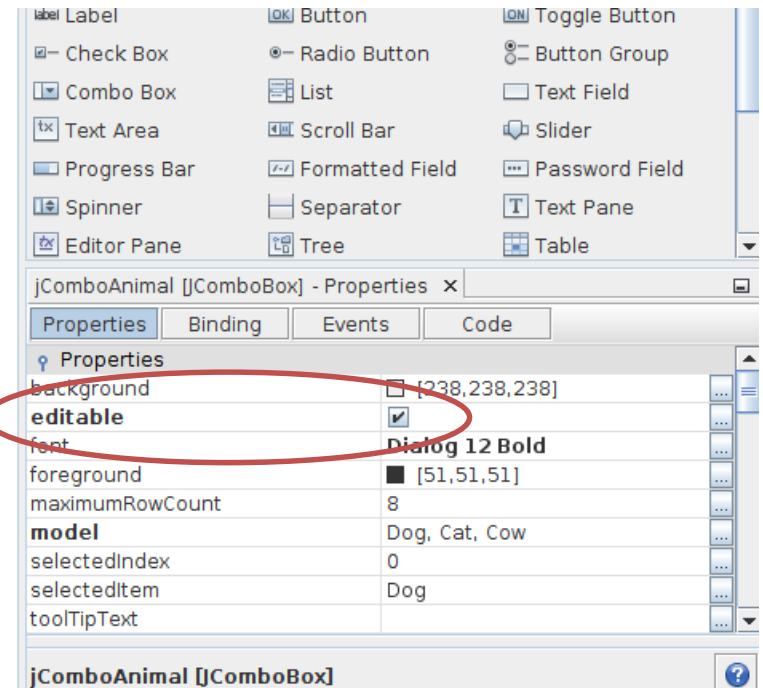
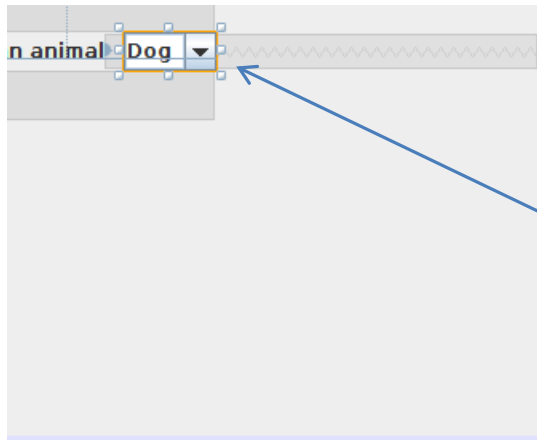


Combo Box

```
private void jComboAnimalActionPerformed(java.awt.event.ActionEvent evt) {  
    String animal = jComboAnimal.getSelectedItem().toString();  
    switch (animal) {  
        case "Dog":  
            jlItalianAnimal.setText("Cane");  
            break;  
        case "Cat":  
            jlItalianAnimal.setText("Gatto");  
            break;  
        case "Cow":  
            jlItalianAnimal.setText("Mucca");  
            break;  
        default:  
            break;  
    }  
}
```

Aggiungiamo un evento actionPerformed alla JComboBox

Combo Box Editabile



Rendiamo la combo box editabile

Combo box Editabile

```
private void jComboAnimalActionPerformed(java.awt.event.ActionEvent evt) {  
    String animal = jComboAnimal.getSelectedItem().toString();  
    switch (animal) {  
        case "Dog":  
            jlItalianAnimal.setText("Cane");  
            break;  
        case "Cat":  
            jlItalianAnimal.setText("Gatto");  
            break;  
        case "Cow":  
            jlItalianAnimal.setText("Mucca");  
            break;  
        default:  
            jlItalianAnimal.setText("Non conosco " + animal);  
            break;  
    }  
}
```

Combo box Render

- Gli oggetti nella combo box vengono visualizzati utilizzando un render di default
 - Visualizza stringhe e icone, per gli altri oggetti chiama il metodo toString()
- E' possibile modificare il render realizzando una classe che implementi
 - ListCellRender per le combo box non editabili
 - ComboBoxEditor per le combo box editabili

LIST

List

- Permette di selezionare degli oggetti in una lista, visualizzati anche su più colonne
 - Selezione singola
 - Selezione multipla
 - Selezione di intervalli multipli
- In genere va inserita in uno ScrollPane
- Implementata dalla class JList

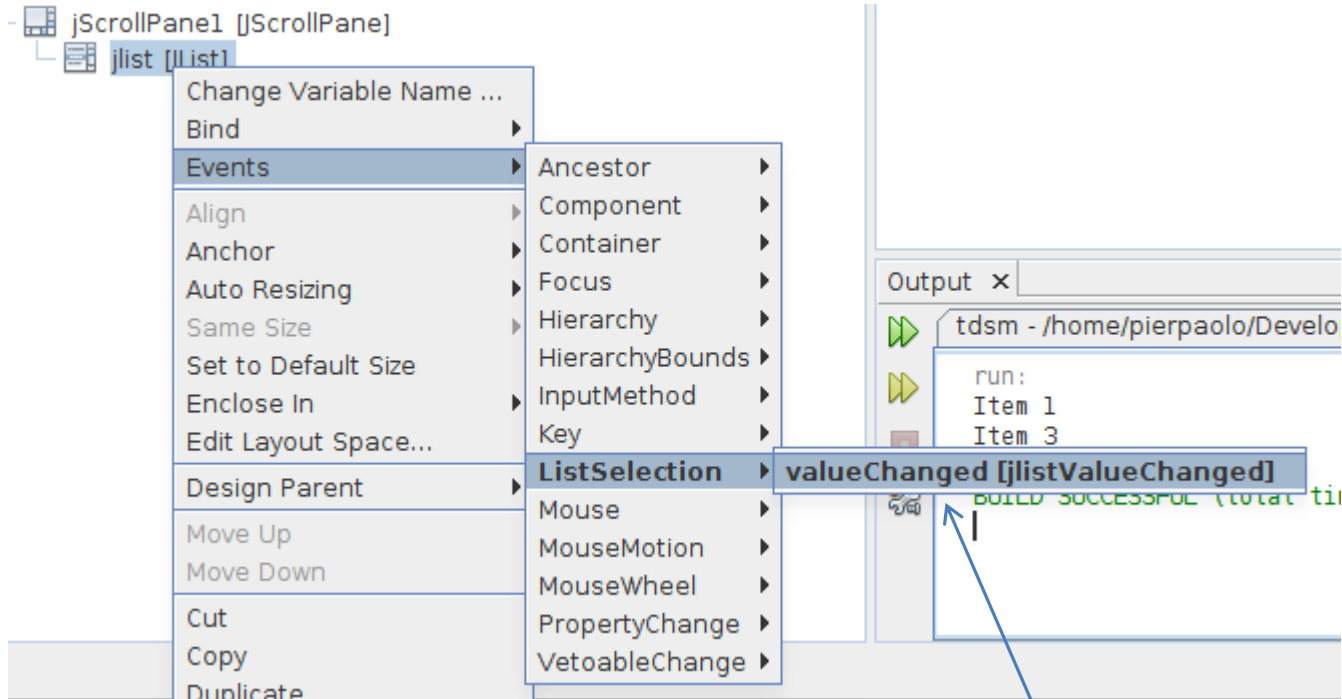
List

The image shows a screenshot of an IDE (Integrated Development Environment) with three main panes. The left pane displays the project structure, including source packages, test packages, and libraries. The center pane shows a visual representation of a list with items 'Item 1' through 'Item 6'. The right pane shows the 'Properties' window for the 'JList' component, with the 'model' and 'selectionMode' properties highlighted. A red box at the bottom left contains the text 'ScrollPane che contiene la JList'. A red box at the bottom right contains the text 'Modello dei dati e modalità di selezione'.

ScrollPane che contiene la JList

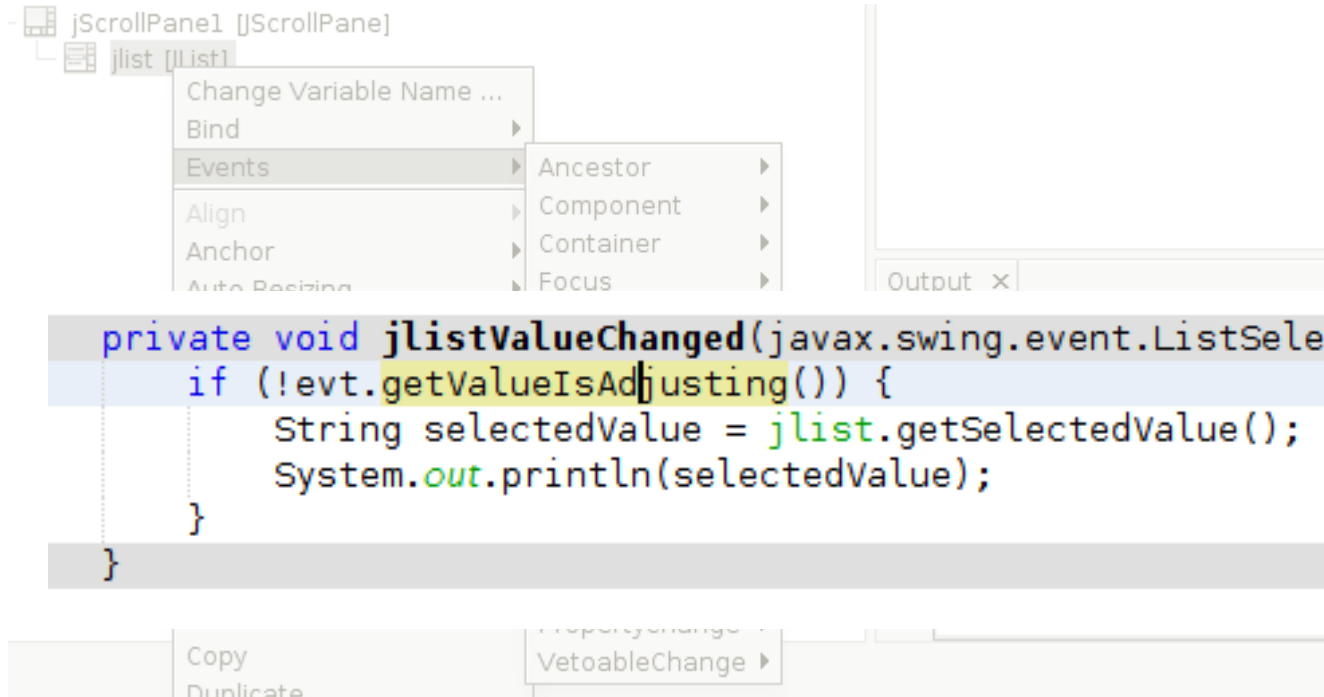
Modello dei dati e modalità di selezione

List (gestione eventi)



Gestiamo l'evento di modifica della selezione

List (gestione eventi)



```
private void jlistValueChanged(javax.swing.event.ListSelectionEvent evt) {  
    if (!evt.getValueIsAdjusting()) {  
        String selectedValue = jlist.getSelectedValue();  
        System.out.println(selectedValue);  
    }  
}
```

List

- E' possibile aggiungere e rimuovere elementi dalla lista

```
DefaultListModel m = (DefaultListModel) jlist.getModel();  
m.remove(0);  
m.removeElement(''Item 10'');  
m.addElement(''Item 20'');  
m.size();  
m.add(2, ''Item 3'');
```

DIALOG

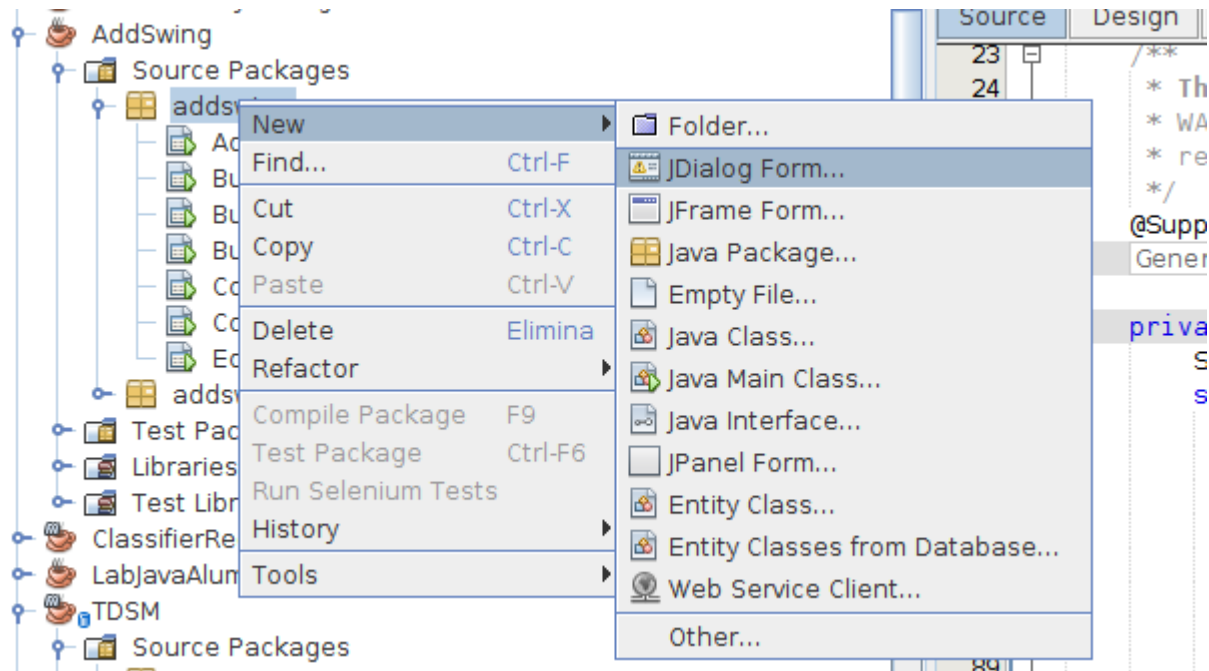
Dialog

- Una finestra di dialogo è una sottofinestra temporanea che permette di notificare qualcosa
- JOptionPane mette a disposizione delle finestre per
 - Visualizzare messaggi
 - Fare delle scelte (YES/NO/CANCEL)
 - Inserire valori
- E' possibile realizzare delle finestre di dialogo estendendo la classe JDialog

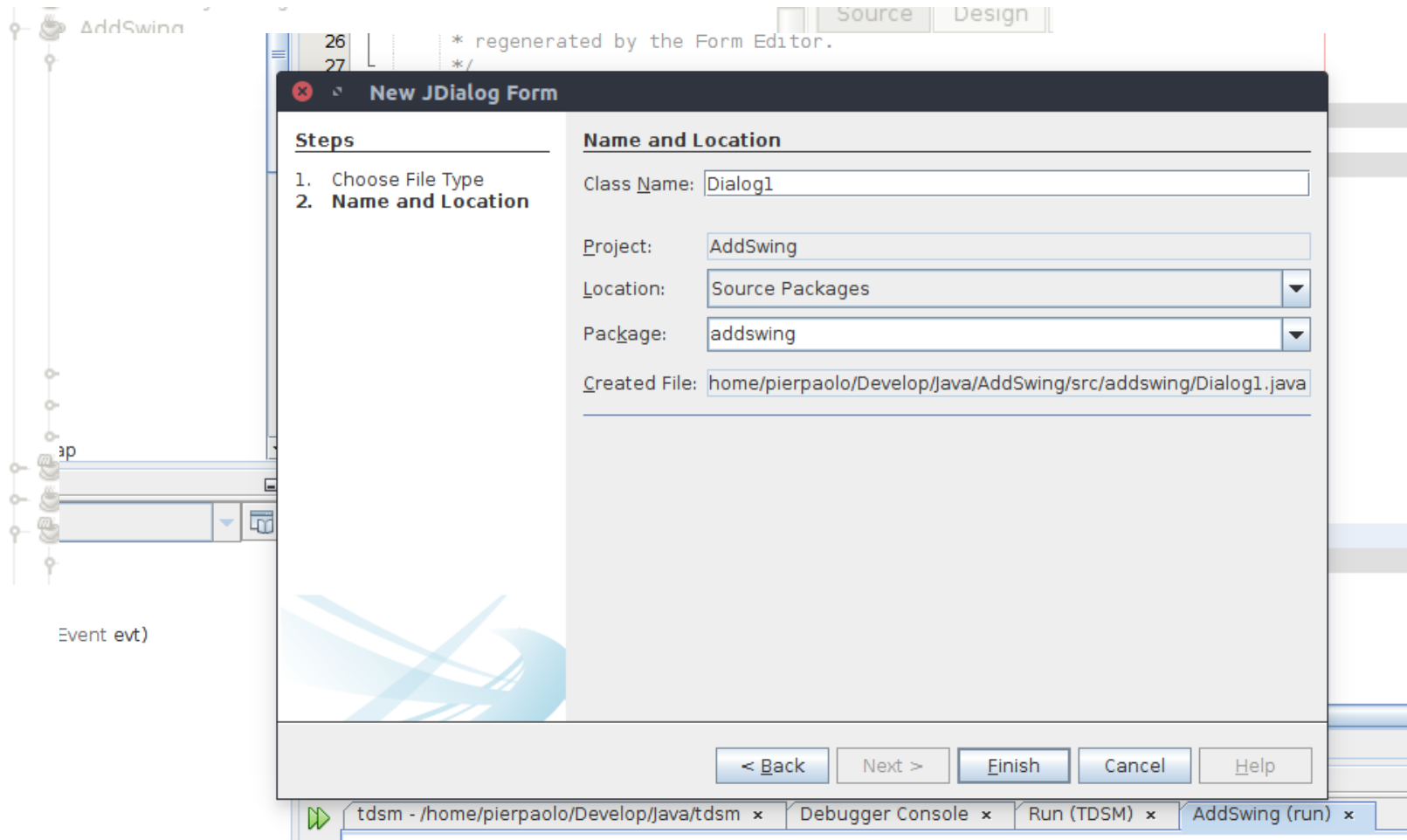
Dialog

- Dipendono da un Frame component
- Può essere modale
 - non è possibile interagire con altre finestre fino a quando non viene chiusa la dialog
- JFileChooser e JColorChooser sono degli esempi di dialog

Dialog per l'inserimento di Person



Dialog per l'inserimento di Person



La classe Person

```
L  */
   public class Person {

       private String name;

       private String surname;

       private String birthplace;

       public Person(String name, String surname, String birthplace) {
           this.name = name;
           this.surname = surname;
           this.birthplace = birthplace;
       }

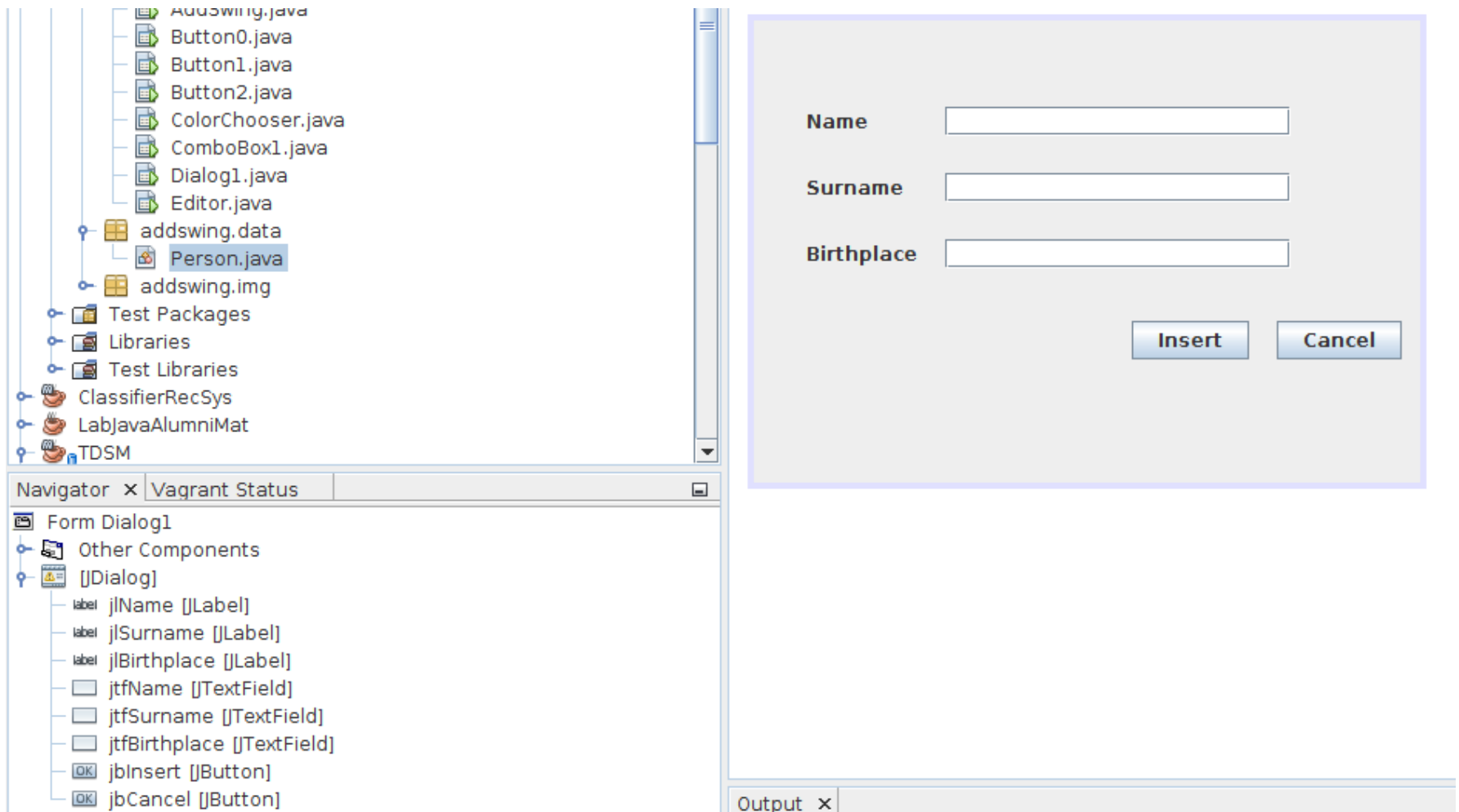
       public String getName() {
           return name;
       }

       public void setName(String name) {
           this.name = name;
       }

       public String getSurname() {
           return surname;
       }

       public void setSurname(String surname) {
```


Dialog per Person



Dialog per Person

```
private Person person = null;

public Person getPerson() {
    return person;
}
```

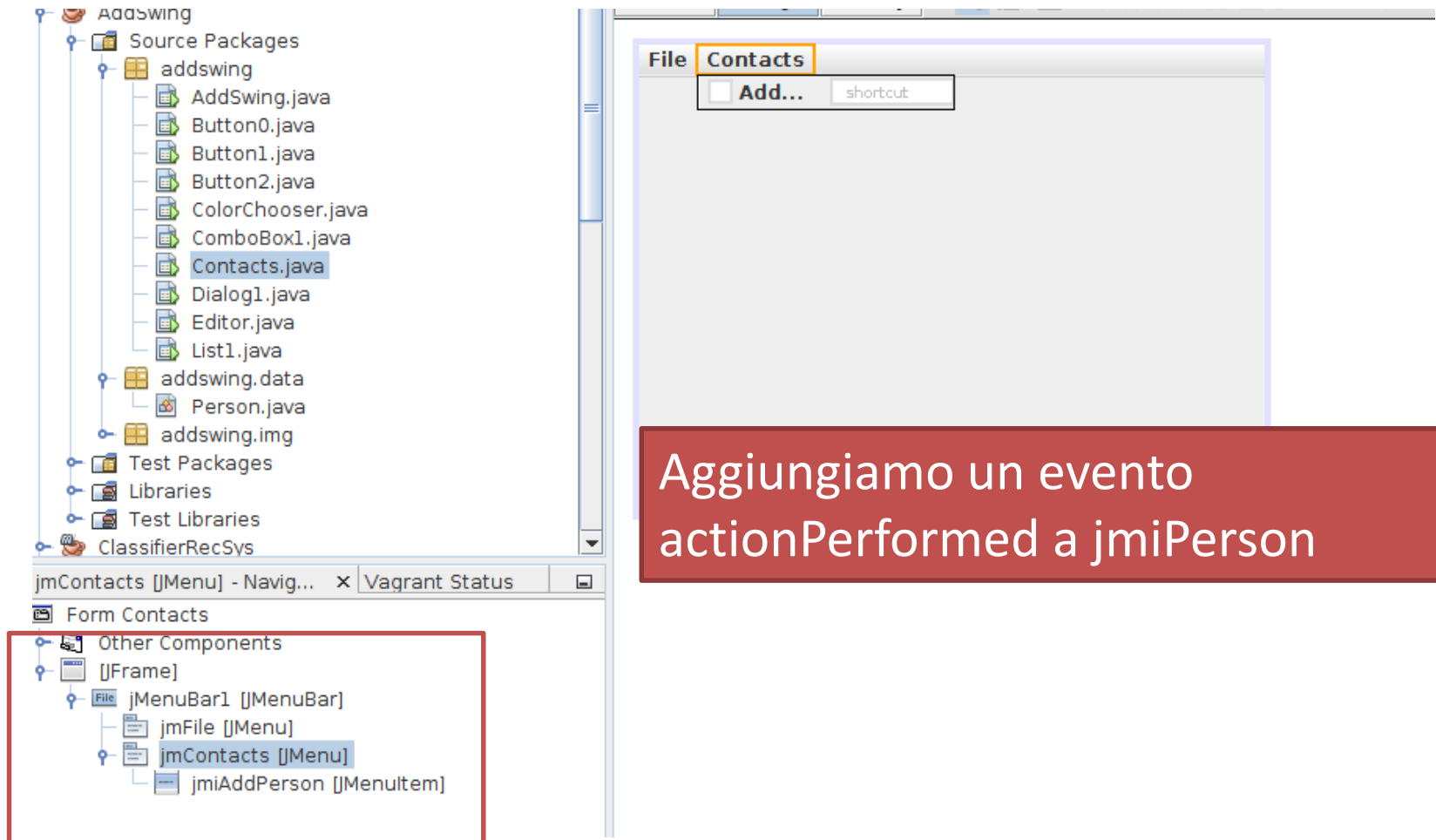
```
/**
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
```

Generated Code

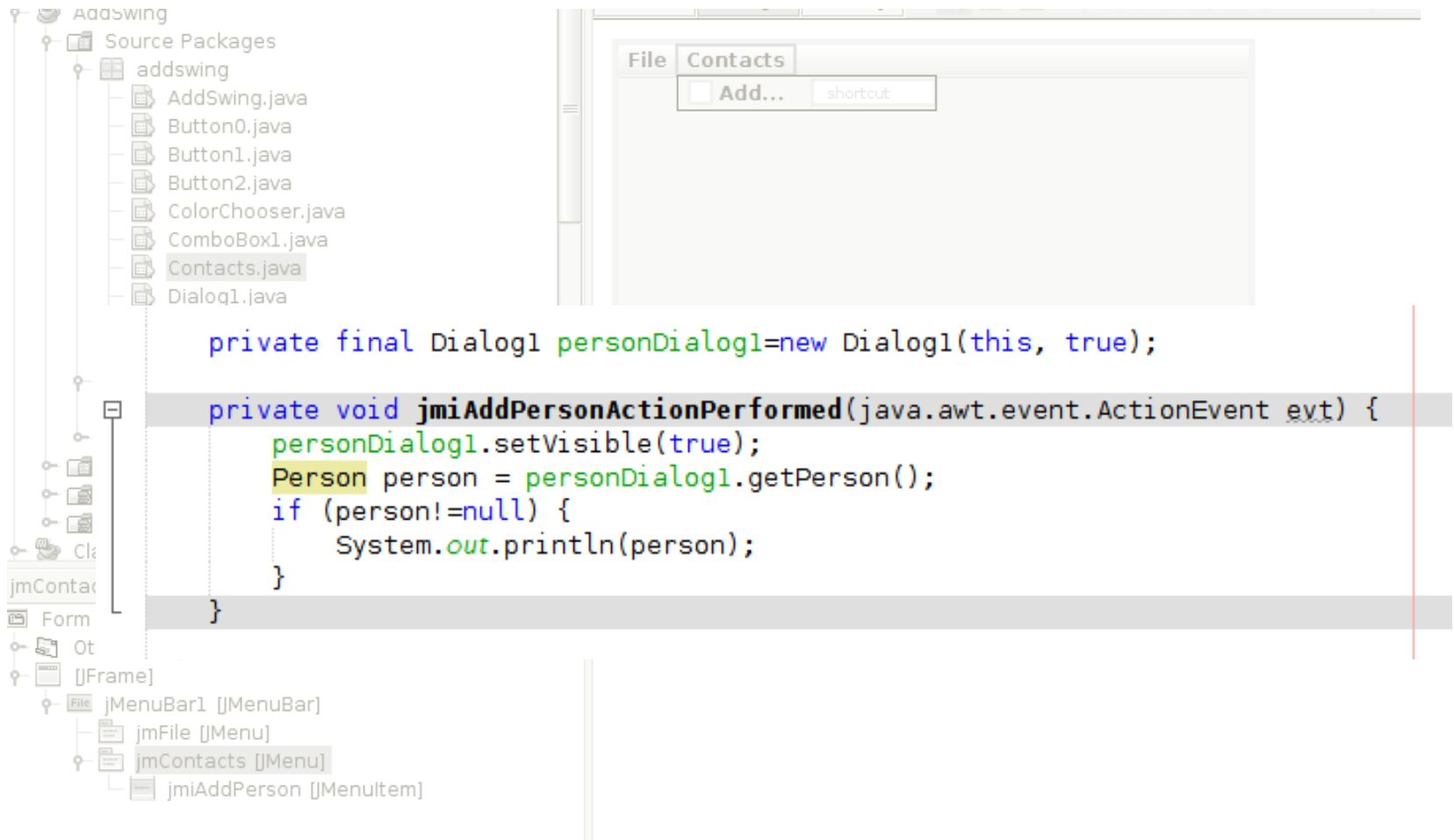
```
private void jbInsertActionPerformed(java.awt.event.ActionEvent evt) {
    person = new Person(jtfName.getText(), jtfSurname.getText(), jtfBirthplace.getText());
    this.dispose();
}

private void jbCancelActionPerformed(java.awt.event.ActionEvent evt) {
    person=null;
    this.dispose();
}
```

Uso della Dialog



Uso della Dialog



The image displays a Java Swing application named 'AddSwing'. On the left, the 'Source Packages' view shows the project structure, including a 'contacts' package with files like 'AddSwing.java', 'Button0.java', 'Button1.java', 'Button2.java', 'ColorChooser.java', 'ComboBox1.java', 'Contacts.java', and 'Dialog1.java'. Below this, the 'JmContact' form is visible, showing a 'File' menu with 'jmFile' and 'jmContacts' items, and a 'jmAddPerson' menu item. On the right, a 'Contacts' dialog box is shown with an 'Add...' button and a 'shortcut' label. The main code area shows the implementation of the 'jmAddPersonActionPerformed' method, which creates a 'personDialog1' instance and calls 'setVisible(true)' on it. The code is as follows:

```
private final Dialog1 personDialog1=new Dialog1(this, true);

private void jmAddPersonActionPerformed(java.awt.event.ActionEvent evt) {
    personDialog1.setVisible(true);
    Person person = personDialog1.getPerson();
    if (person!=null) {
        System.out.println(person);
    }
}
```