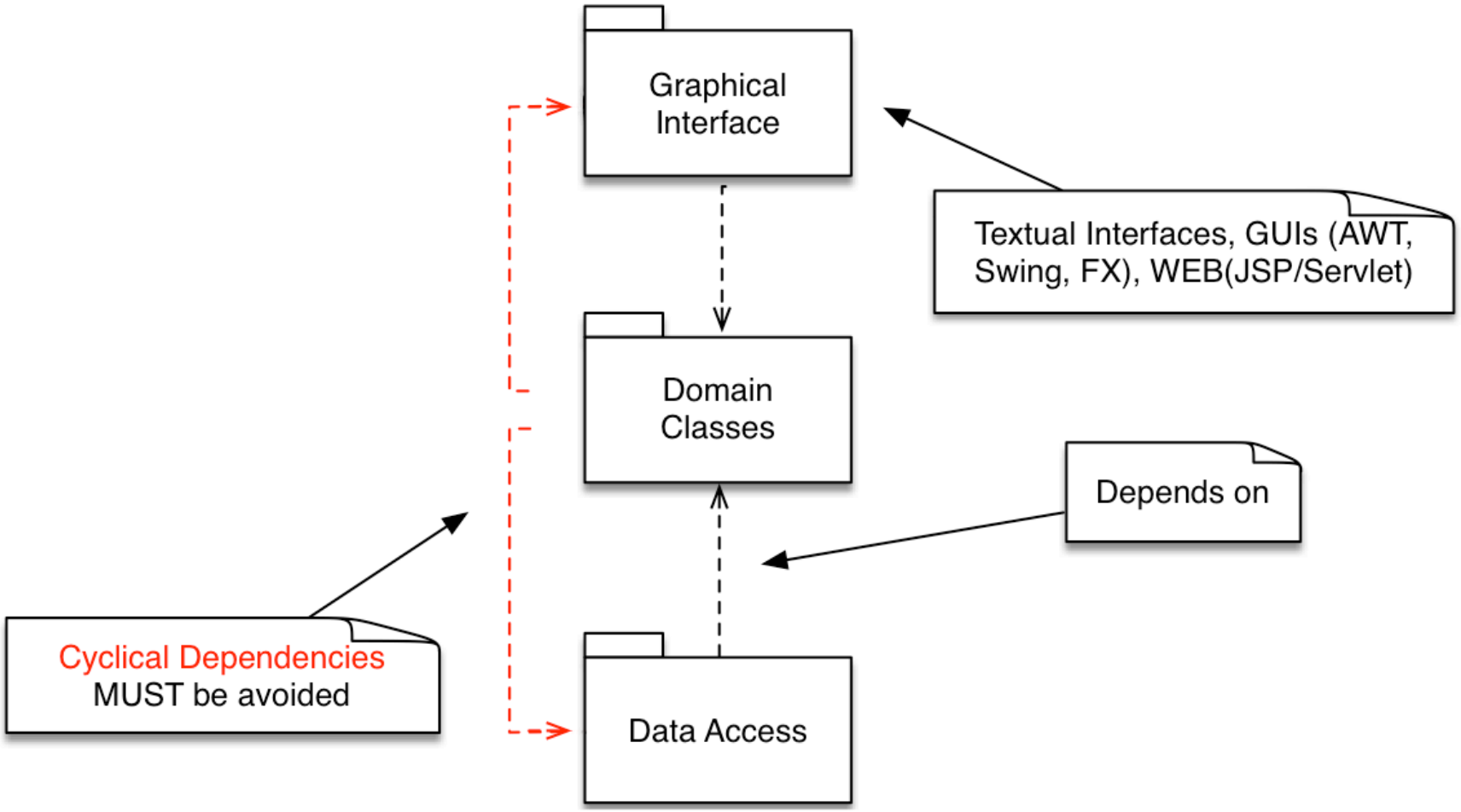
Java Swing

Università di Modena e Reggio Emilia

Prof. Nicola Bicocchi (nicola.bicocchi@unimore.it)

ING

Software Design



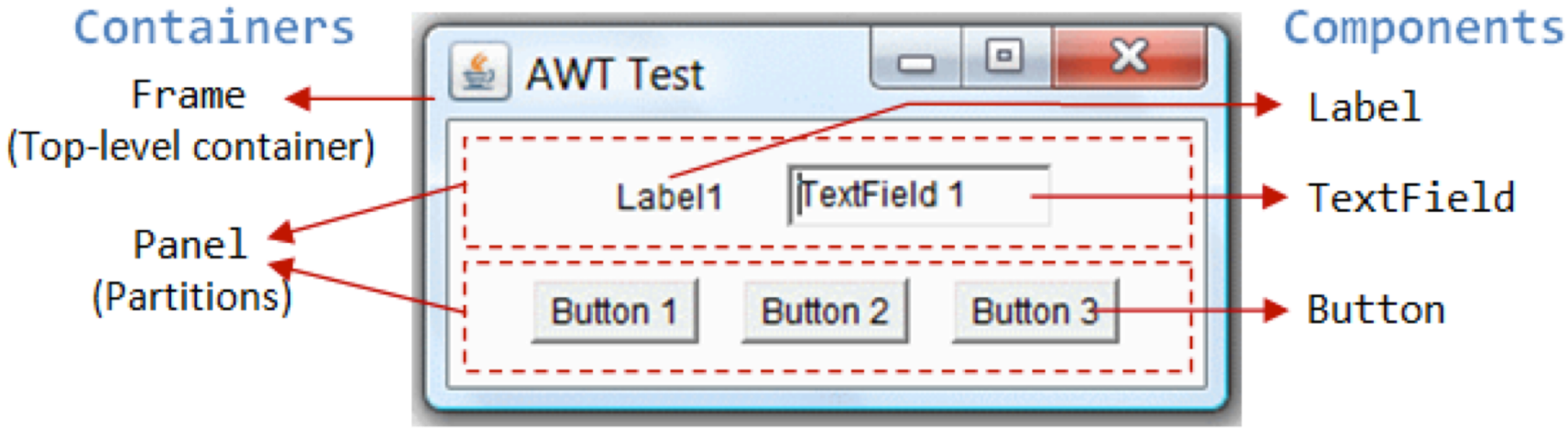
Package java.awt.*

- Provides:
  - Components *(button, checkbox, scrollbar, etc.)*
  - Containers *(they are still components)*
  - Event management:
    - System-generated events
    - UI-generated events
  - Layout management

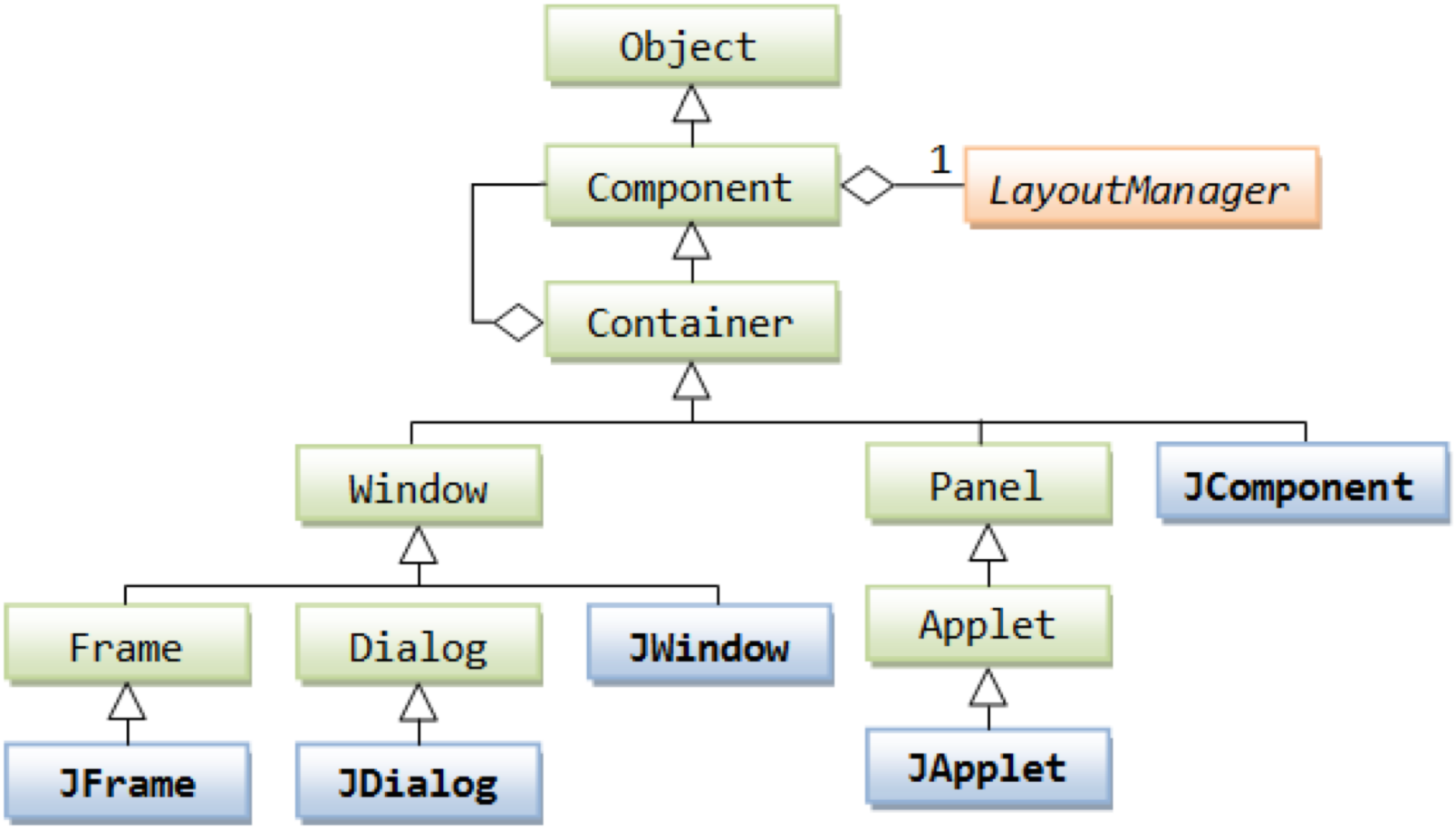
Package javax.swing.*

- Contains the same components of java.awt, but with different names (JButton, JFrame, etc.)
- All these components derive from JComponent
- Advantages:
  - provides a series of components with the same appearance and behavior on all platforms
  - look and feel changeable at runtime
- Swing is an extension of AWT

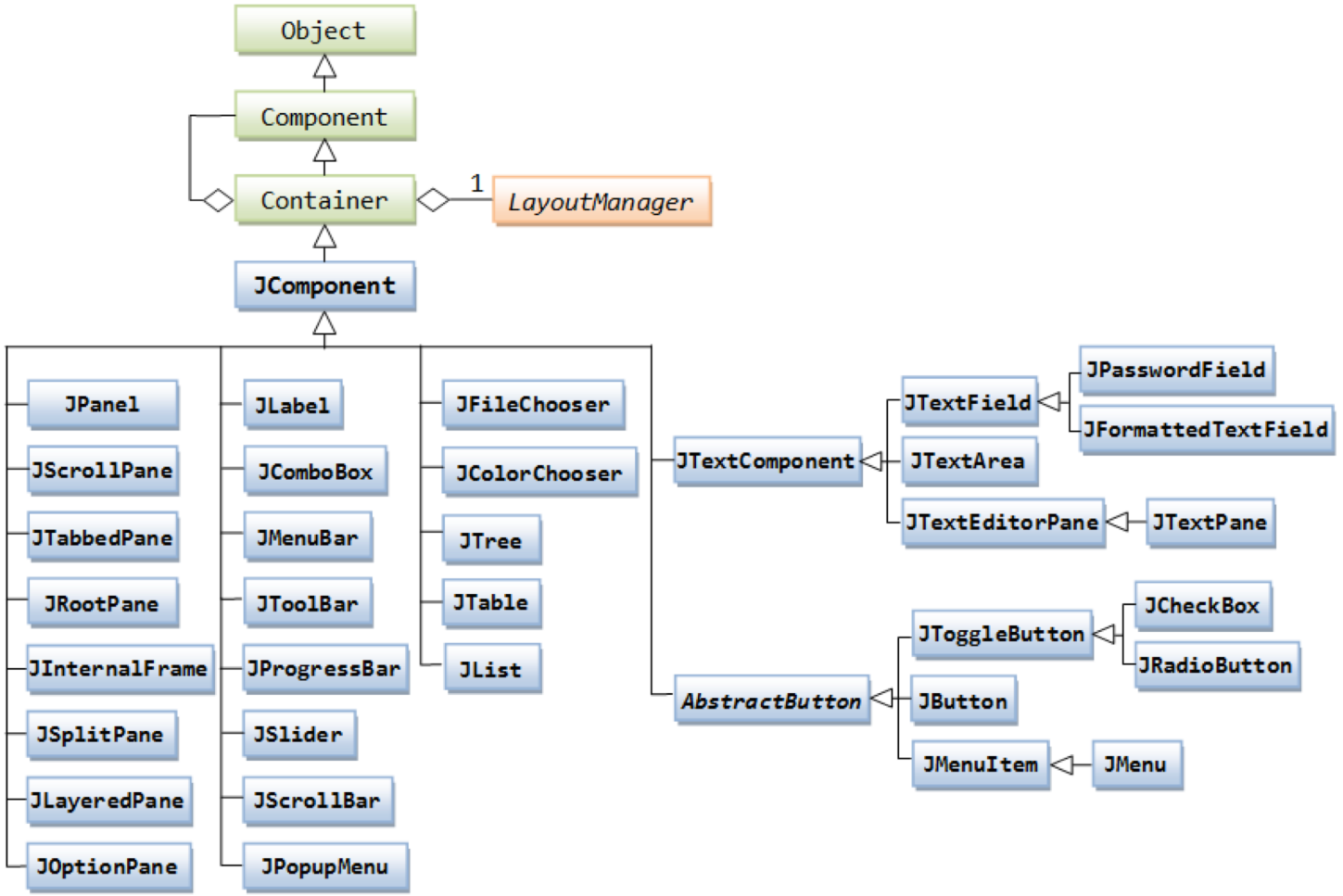
Containers and components



Class hierarchy (Containers)

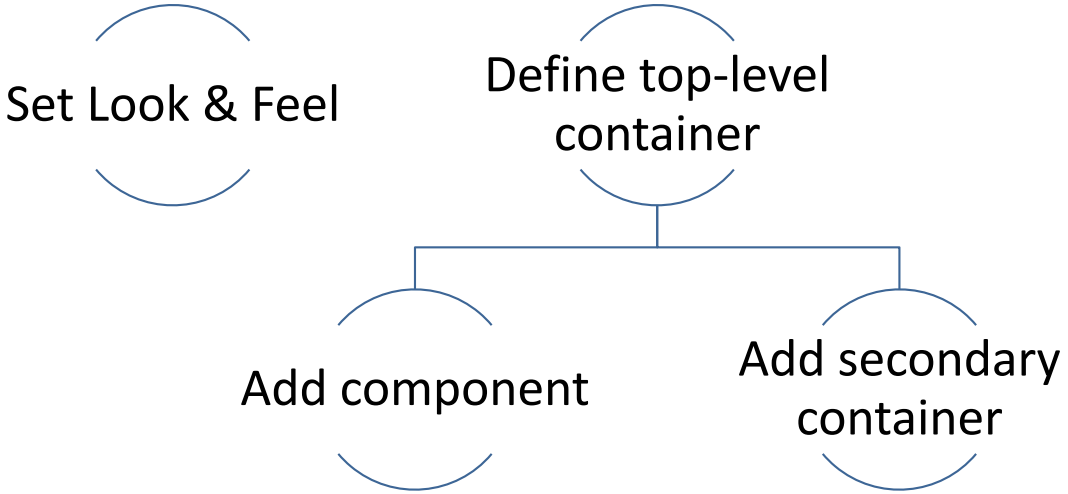


Class hierarchy (Components)



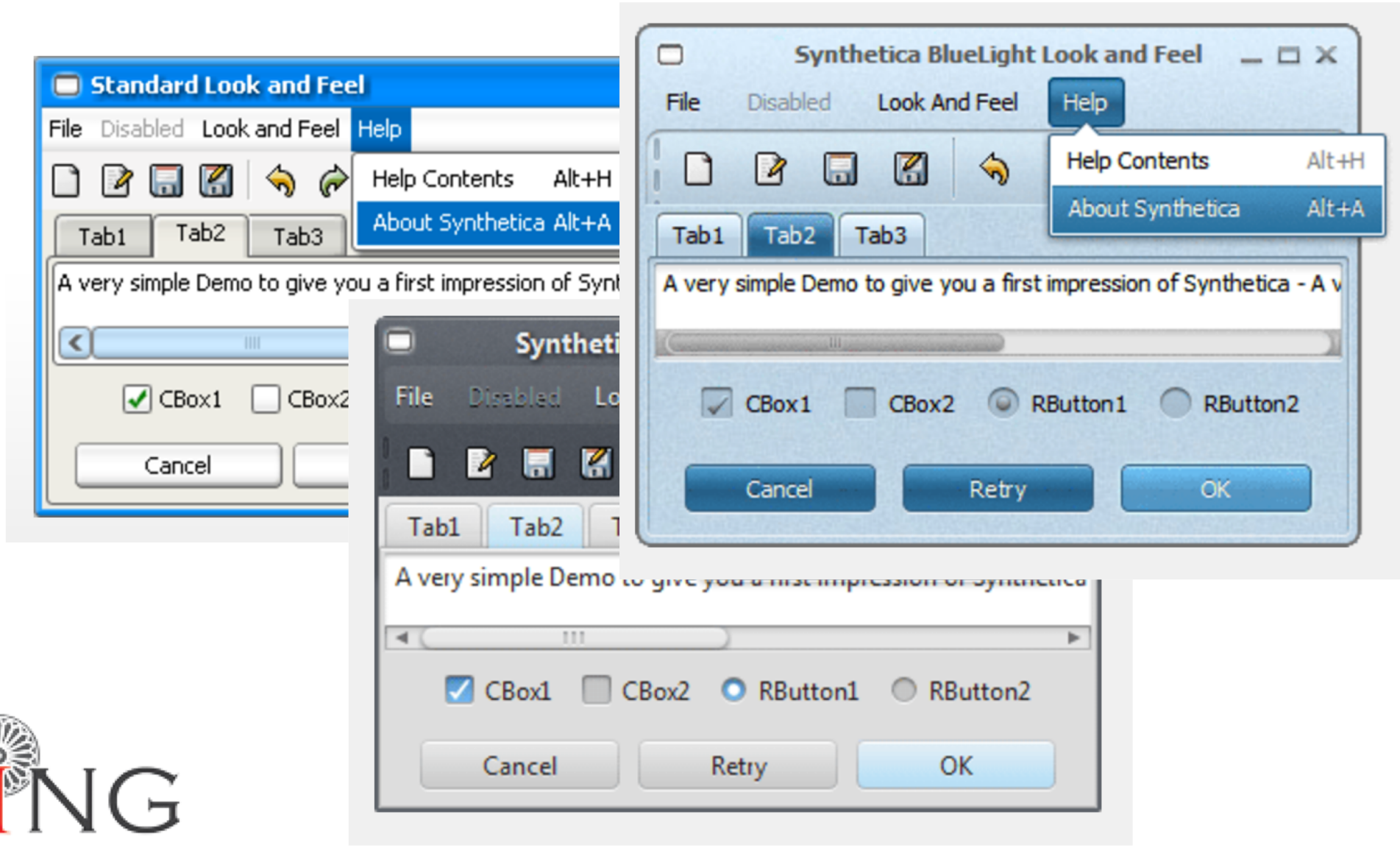
Graphical Programming

- Set a Look & Feel (= Style)
  - Microsoft Windows, Mac, Java Metal
- Define one (or more) top-level container
  - **JFrame**, JDialog, JApplet
- Add components to the containers
  - JButton, JComboBox, JSlider, …
- Add secondary containers



ING

Look & Feel

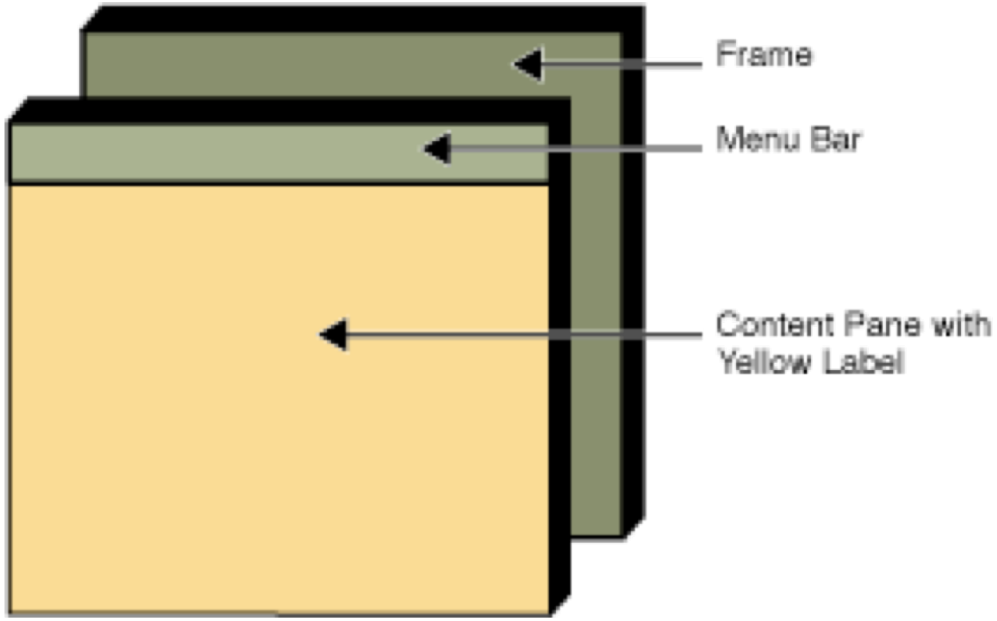

Standard Look and Feel
File Disabled Look and Feel Help
Help Contents Alt+H
About Synthetica Alt+A
Tab1 Tab2 Tab3
A very simple Demo to give you a first impression of Synt
CBox1 CBox2
Cancel
Synthetica BlueLight Look and Feel
File Disabled Look And Feel Help
Help Contents Alt+H
About Synthetica Alt+A
Tab1 Tab2 Tab3
A very simple Demo to give you a first impression of Synthetica - A v
CBox1 CBox2 RButton1 RButton2
Cancel Retry OK
Syntheti
File Disabled Lo
Tab1 Tab2
A very simple Demo
CBox1 CBox2 RButton1 RButton2
Cancel Retry OK

Look & Feel

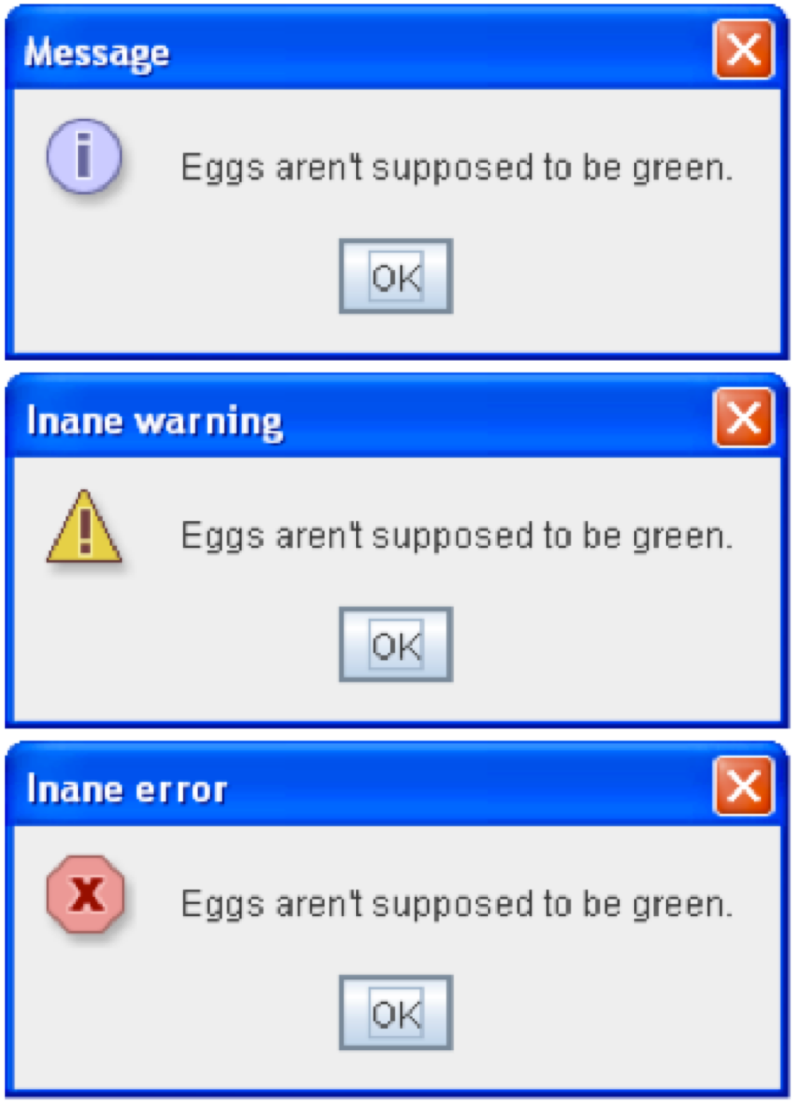
- UIManager manages the current look and feel. For examples, look at http://www.jyloo.com/synthetica/themes/

```
// Set Metal Look and Feel
UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");

// Set Motif Look and Feel
UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");

// Set Windows Look and Feel
UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WIndowsLookAndFeel");
```

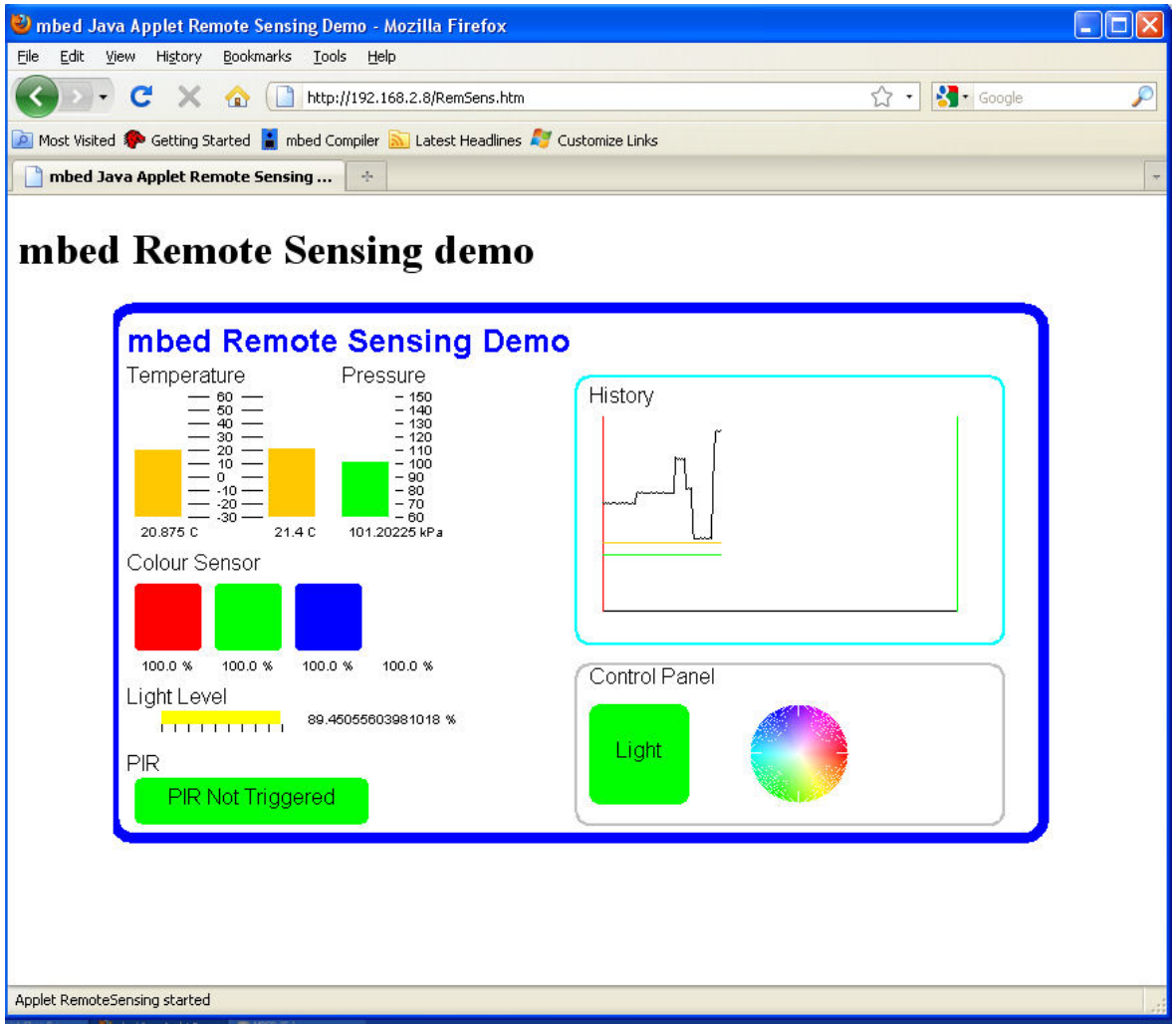
Top-level container: JFrame



Top-level container: JDialog



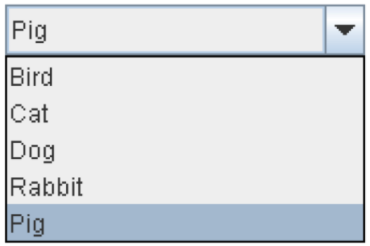
Top-level container: JApplet *(deprecated)*



Components, a visual guide

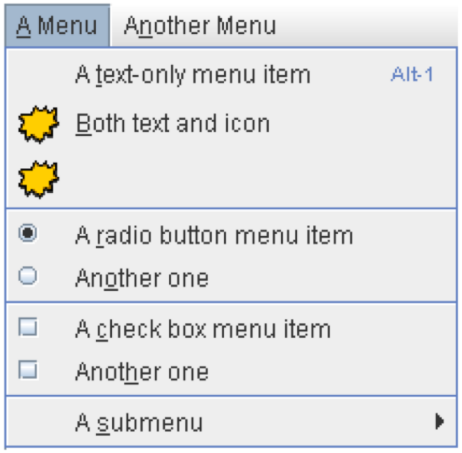
JButton

JCheckBox

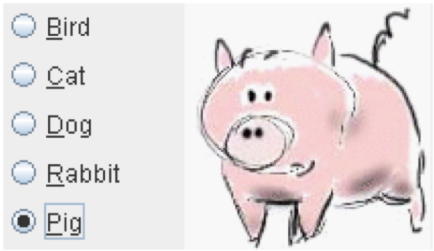


JComboBox

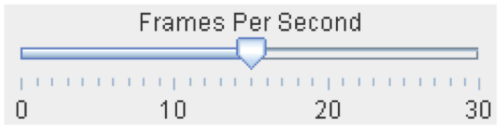
JList



JMenu



JRadioButton



JSlider

http://web.mit.edu/6.005/www/sp14/psets/ps4/java-6-tutorial/components.html

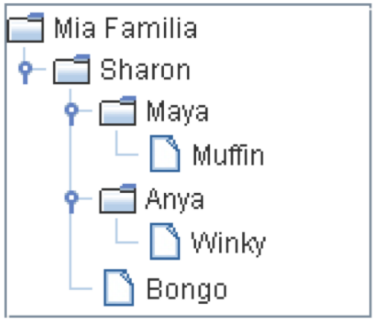
Components, a visual guide

Host	User	Password	Last Modified
Biocca Games	Freddy	!#asf6Awwzb	Mar 16, 2006
zabble	ichabod	Tazb!34$fZ	Mar 6, 2006
Sun Developer	fraz@hotmail.co...	AasW541!fbZ	Feb 22, 2006
Heirloom Seeds	shams@gmail....	bkz[ADF78!	Jul 29, 2005
Pacific Zoo Shop	seal@hotmail.c...	vbAf124%z	Feb 22, 2006

JTable

This is an editable JTextArea. A text area is a "plain" text component, which means that although it can display text in any font, all of the text is in the same font.

JTextArea



JTree

JSpinner

JTextField

JPasswordField

http://web.mit.edu/6.005/www/sp14/psets/ps4/java-6-tutorial/components.html

Components, a visual guide

JLabel

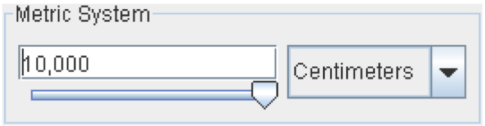
JProgressBar

JSeparator

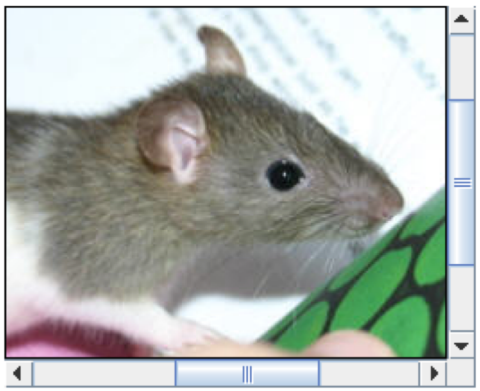
JToolTip

http://web.mit.edu/6.005/www/sp14/psets/ps4/java-6-tutorial/components.html

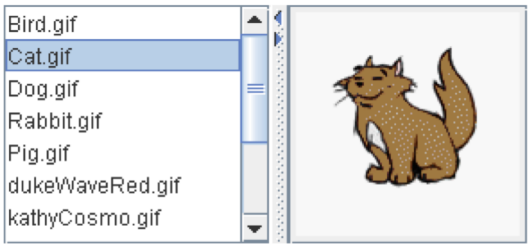
Components, a visual guide



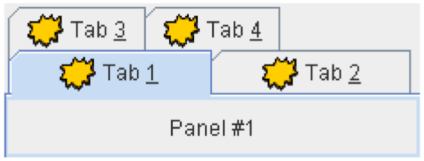
JPanel



JScrollPane



JSplitPane



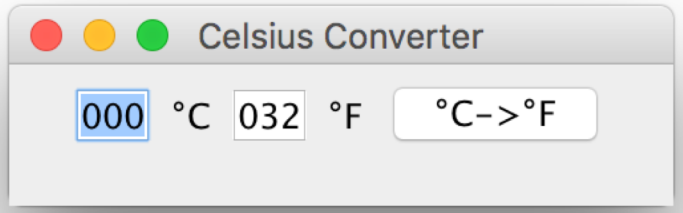
JTabbedPane

JToolBar

http://web.mit.edu/6.005/www/sp14/psets/ps4/java-6-tutorial/components.html

A complete example

```
public class CelsiusConverterBasic extends JFrame {
	private static final long serialVersionUID = 1L;
	private JButton CFButton;
	private JTextField fahrenheitTF, celsiusTF;

	public CelsiusConverterBasic() {
		super("Celsius Converter");
		celsiusTF = new JTextField("000");
		fahrenheitTF = new JTextField("032");
		CFButton = new JButton("°C->°F");

		JPanel p1 = new JPanel();
		p1.add(celsiusTF);
		p1.add(new JLabel("°C"));
		p1.add(fahrenheitTF);
		p1.add(new JLabel("°F"));
		p1.add(CFButton);

		add(p1);
		setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
		setSize(250, 75);
		setVisible(true);
	}
}
```



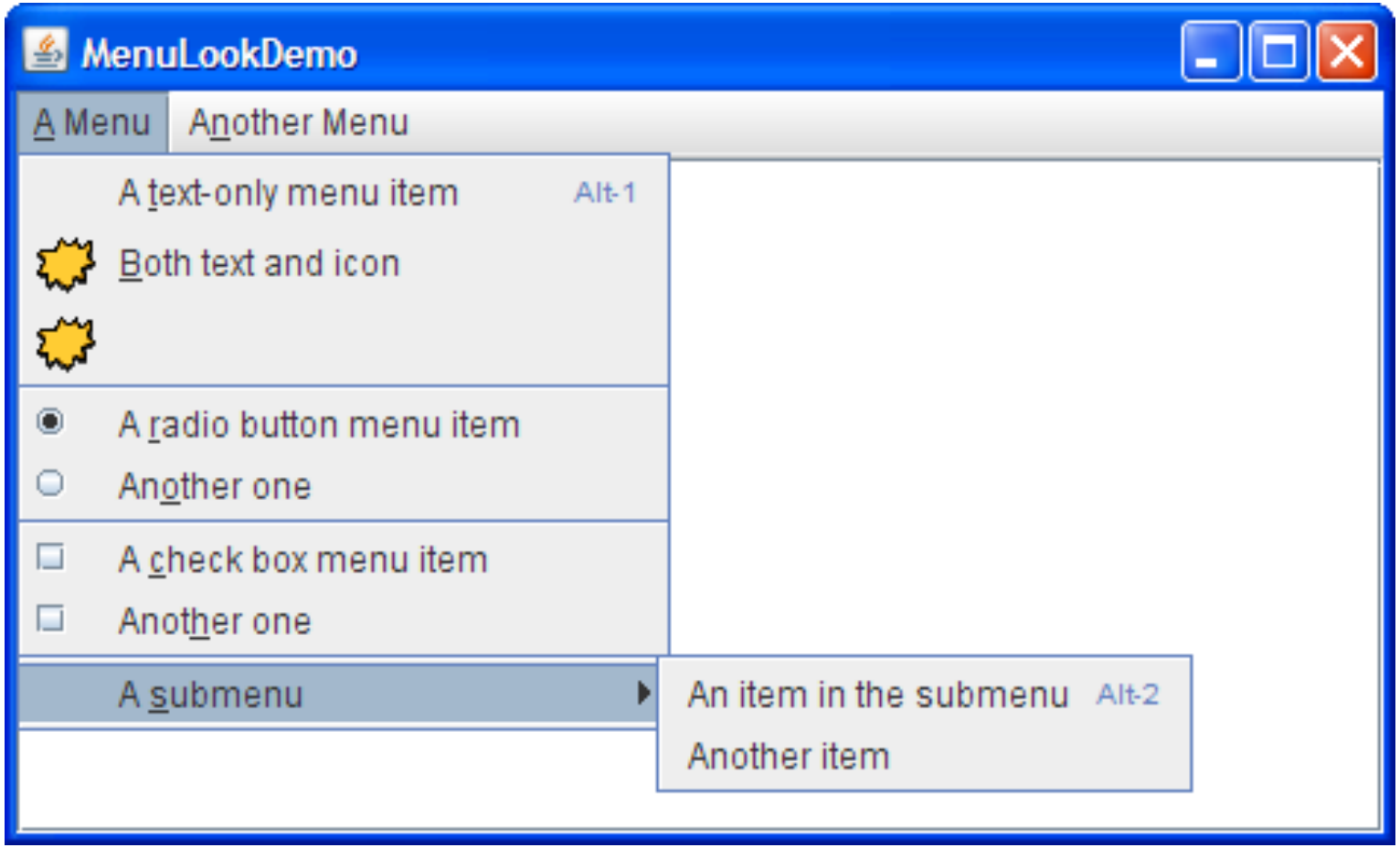
JFrame basic methods

- setcontentpane(Container c)
  - sets the ContentPane. It is a a secondary container, usually a JPanel
- add(Component c)
  - add a component to ContentPane
- setDefaultCloseOperation(WindowConstants)
  - EXIT_ON_CLOSE
  - DO_NOTHING_ON_CLOSE
  - DISPOSE_ON_CLOSE
  - HIDE_ON_CLOSE
- setSize(int base, int height)
  - defines the dimensions of the component
- setVisible(boolean visibility)
  - defines the visibility status of the component

Running it!

```
// Ok
public static void main(String[] args) {
    new CelsiusConverter();
}

// Better
public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
        @Override
        public void run() {
            new CelsiusConverter();
        }
    });
}
```

JFrame MenuBar



ING

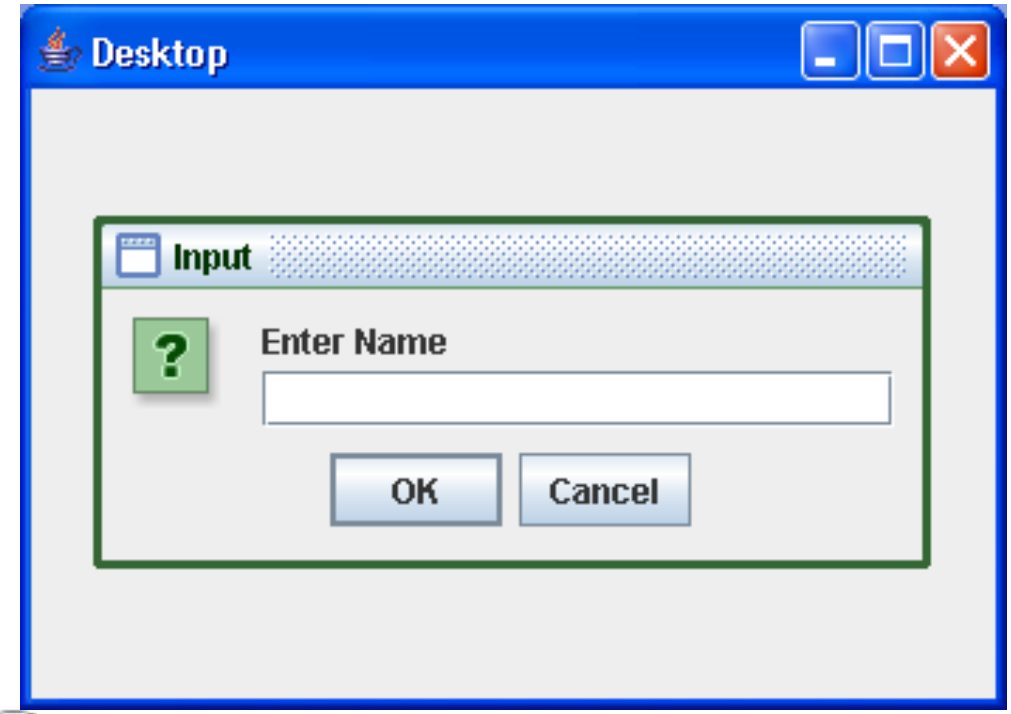
JFrame MenuBar

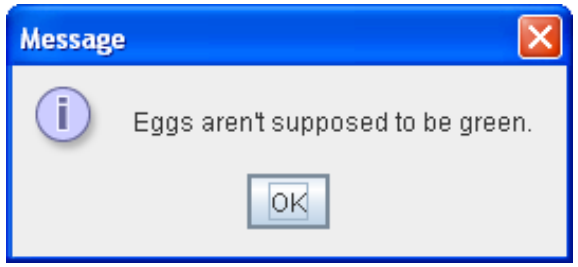
- Three components are involved in a hierarchical fashion:
  - JMenuBar, JMenu, JMenuItem

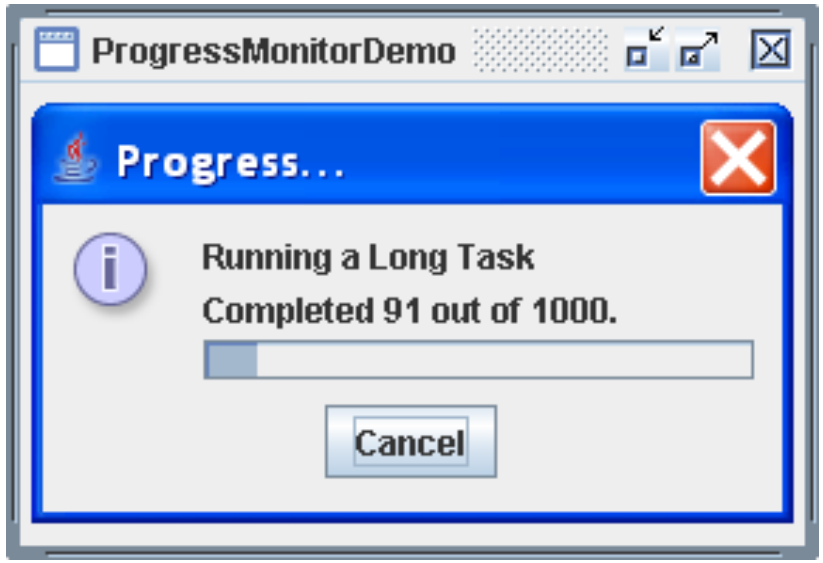
```
JMenuItem openFile = new JMenuItem("Open");
JMenuItem closeFile = new JMenuItem("Close");

JMenu file = new JMenu("File");
file.add(openFile);
file.add(closeFile);

JMenuBar menuBar = new JMenuBar();
menuBar.add(file)

setJMenuBar(menuBar);
```

JDialog

- Applications need to provide information, advise the user, etc.







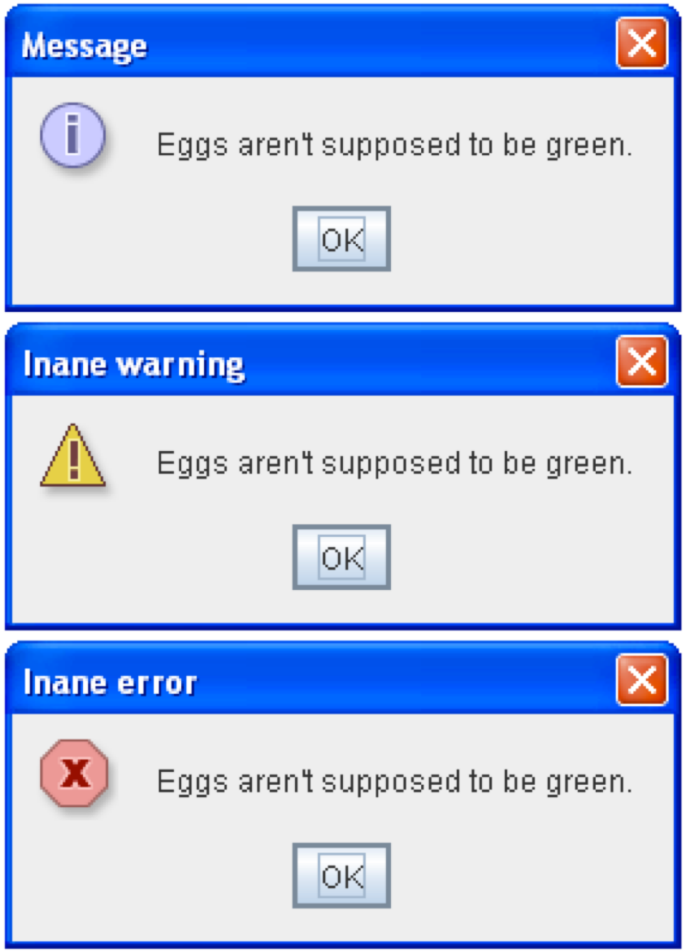
JDialog

- Dialogs are a better choice than instantiating other JFrames!
  - Every dialog is dependent on a top-level container.
  - Dialogs are all instances of JDialog, even though the majority is done using helper classes (e.g., JOptionPane).

How to make Dialogs

- Specializing JDialog (top-level container) and defining your own layouts. Same principle as specializing JFrame.
- Using JOptionPane. JOptionPane provides support for laying out standard dialogs, providing icons, specifying the dialog title and text, and customizing the button text.

JOptionPane.showMessageDialog()




```
//default title and icon
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.");
```

```
//custom title, warning icon
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.",
    "Inane warning",
    JOptionPane.WARNING_MESSAGE);
```

```
//custom title, error icon
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.",
    "Inane error",
    JOptionPane.ERROR_MESSAGE);
```

ING

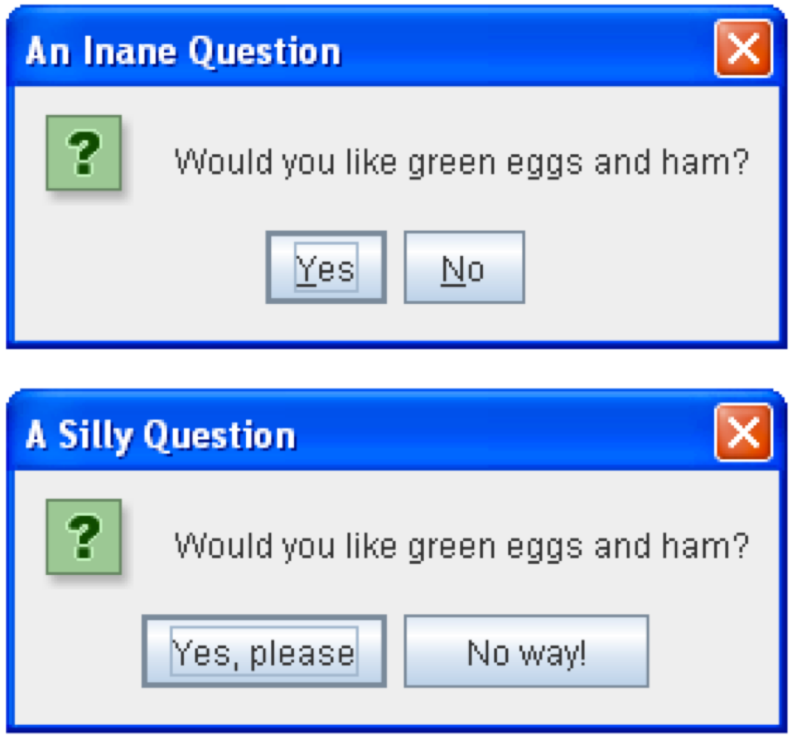
JOptionPane.showOptionDialog()

- Displays a modal dialog with the specified buttons, icons, message, title, and so on. With this method, you can change the text that appears on the buttons of standard dialogs. You can also perform many other kinds
- of customization.


```
//Custom button text
Object[] options = {"Yes, please",
                    "No, thanks",
                    "No eggs, no ham!"};
int n = JOptionPane.showOptionDialog(frame,
    "Would you like some green eggs to go "
    + "with that ham?",
    "A Silly Question",
    JOptionPane.YES_NO_CANCEL_OPTION,
    JOptionPane.QUESTION_MESSAGE,
    null,
    options,
    options[2]);
```

ING

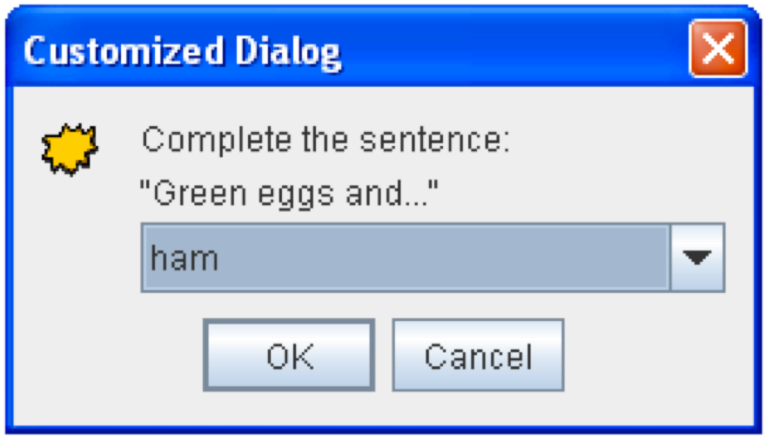
JOptionPane.showConfirmationDialog()




```
//default icon, custom title
int n = JOptionPane.showConfirmDialog(
    frame,
    "Would you like green eggs and ham?",
    "An Inane Question",
    JOptionPane.YES_NO_OPTION);


Object[] options = {"Yes, please",
                    "No way!"};
int n = JOptionPane.showOptionDialog(frame,
    "Would you like green eggs and ham?",
    "A Silly Question",
    JOptionPane.YES_NO_OPTION,
    JOptionPane.QUESTION_MESSAGE,
    null,     //do not use a custom Icon
    options,  //the titles of buttons
    options[0]); //default button title
```

ING

JOptionPane.showInputDialog()

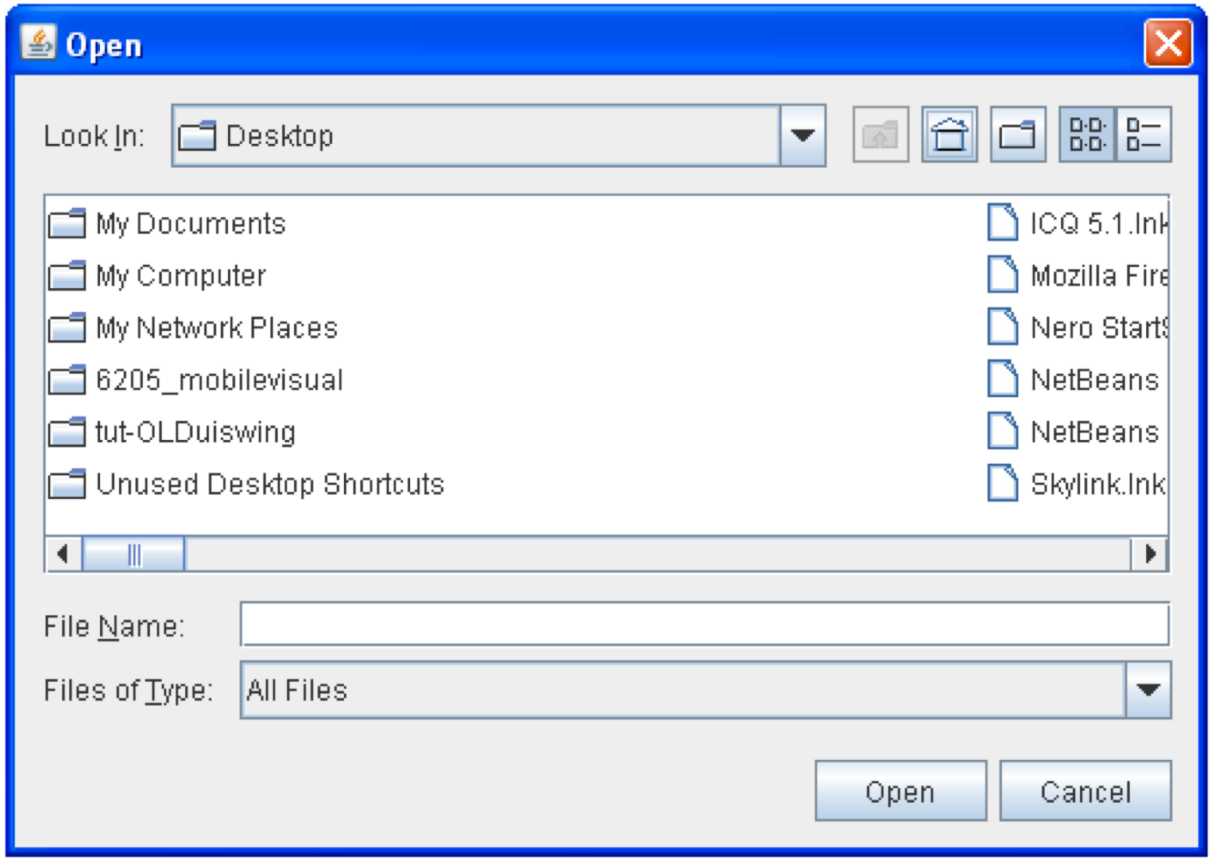



```
Object[] possibilities = {"ham", "spam", "yam"};
String s = (String)JOptionPane.showInputDialog(
                    frame,
                    "Complete the sentence:\n"
                    + "\"Green eggs and...\"",
                    "Customized Dialog",
                    JOptionPane.PLAIN_MESSAGE,
                    icon,
                    possibilities,
                    "ham");

//If a string was returned, say so.
if ((s != null) && (s.length() > 0)) {
    setLabel("Green eggs and... " + s + "!");
    return;
}

//If you're here, the return value was null/empty.
setLabel("Come on, finish the sentence!");
```

ING

JFileChooser

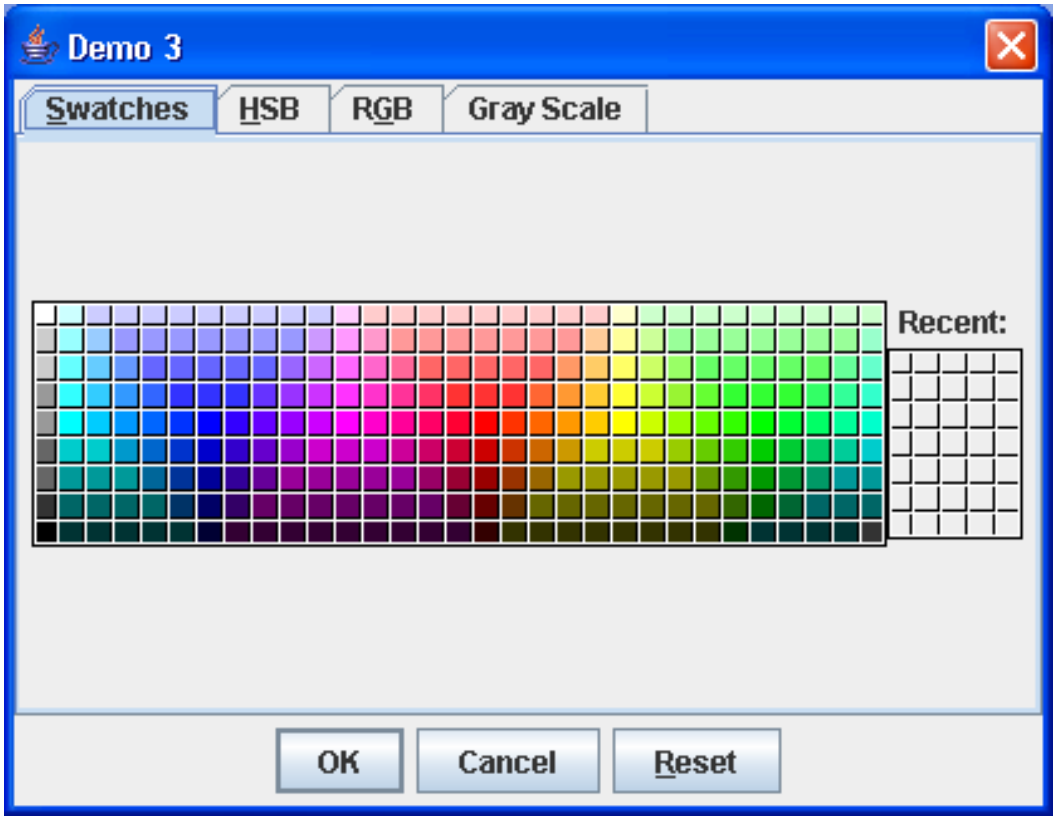
- Provides a GUI for navigating the file system
  - `int result = fileChooser.showOpenDialog(Component parent);`



ING

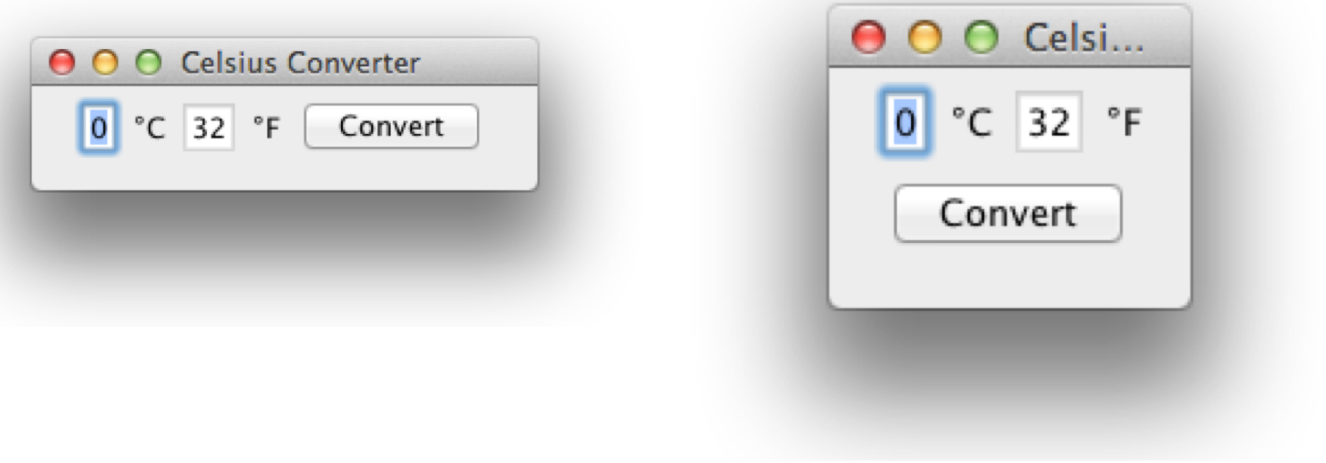
JColorChooser

- Provides a GUI for navigating color spaces
  - `Color c = JColorChooser.showDialog(Component parent, String title, Color initialColor)`



What is a layout?

- Default GUIs, when resized, allow the automatic relocation of components:
  - this is a necessity: Java runs to many different platforms. Android, for example, runs on either 65” TVs or 6” smartphones (very different screen size!)



ING

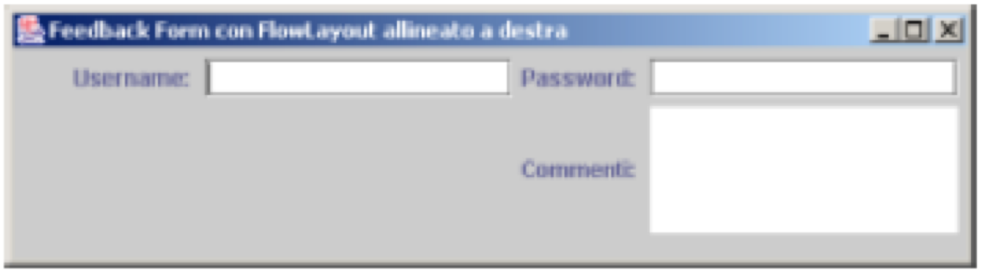
Layout Manager

- A layout manager determines the disposal of the components in a container
  - *Flow, Border, Grid, GridBag, Card* Layouts
- JPanels are containers supporting layouts
  - Different panels can have different layout managers
  - Layout managers are passed to JPanel constructor
- Methodology:

```
JPanel p = new JPanel(new GridLayout(2,2));
p.add(JButton);
```

ING

Layout Manager - FlowLayout



(FlowLayout.RIGHT)

(FlowLayout.CENTER,20,20)

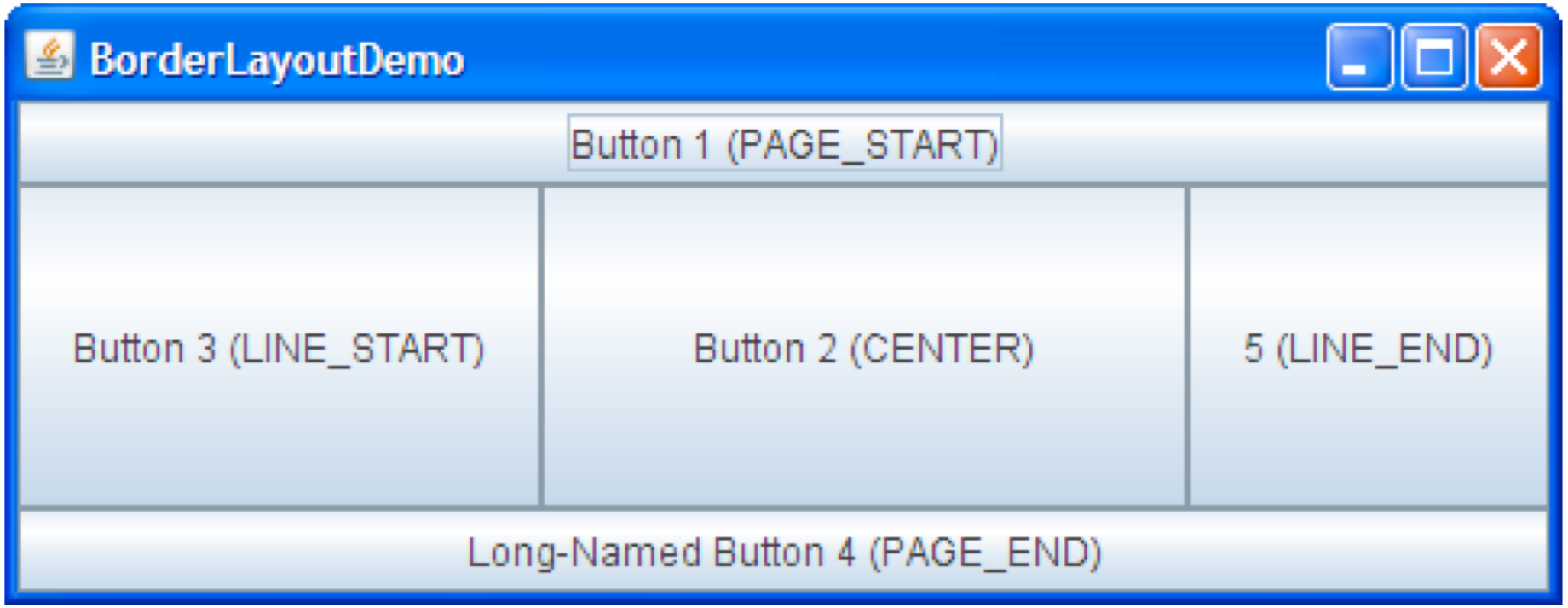
ING

Layout Manager - FlowLayout

- Default layout (i.e., new JPanel())
  - Disposes components from left to right
- Constructors:
  - `FlowLayout f = new FlowLayout();`
  - `FlowLayout f = new FlowLayout(int align);`
  - `FlowLayout f = new FlowLayout(int align, int hgap, int vgap);`
- Constructors parameters:
  - align: Alignment of basis (FlowLayout.LEFT, FlowLayout.RIGHT, FlowLayout.CENTER)
  - hgap: Horizontal space between components (default: 3 pixel)
  - vgap: Vertical space between components (default: 3 pixel)

ING

Layout Manager - BorderLayout



ING

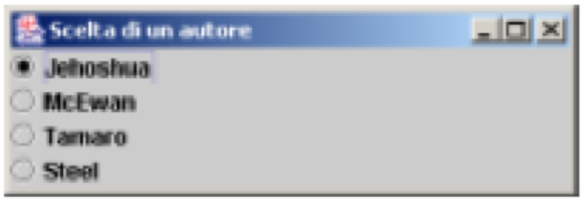
Layout Manager - BorderLayout

- Splits into five areas (PAGE_START, PAGE_END, LINE_START, LINE_END, CENTER).
- Constructors:
  - `BorderLayout b = new BorderLayout();`
  - `BorderLayout b = new BorderLayout(int1, int2);`
    - int1, int2 are the spaces between the components related horizontal and vertical
- The filling is “targeted”:

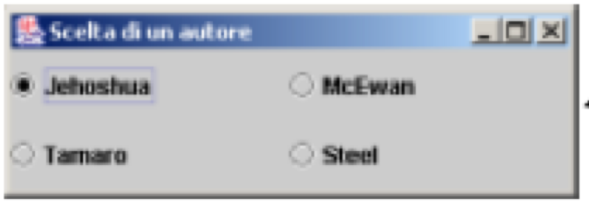
```
JPanel p = new JPanel(new BorderLayout());
p.add(BorderLayout.PAGE_START, new JButton());
p.add(BoarderLayout.PAGE_END, new JButton());
```

ING

Layout Manager - GridLayout

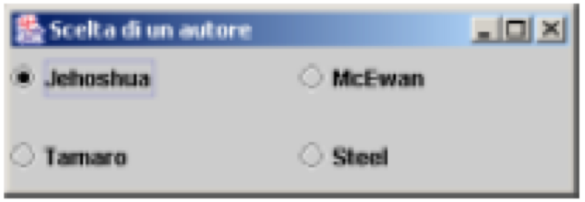


4 row, 1 columns:
distance 0



2 row, 2 columns:
distance 0 pixel

(Distanza min = 0)



2 row, 2 columns:
distance 10 pixel

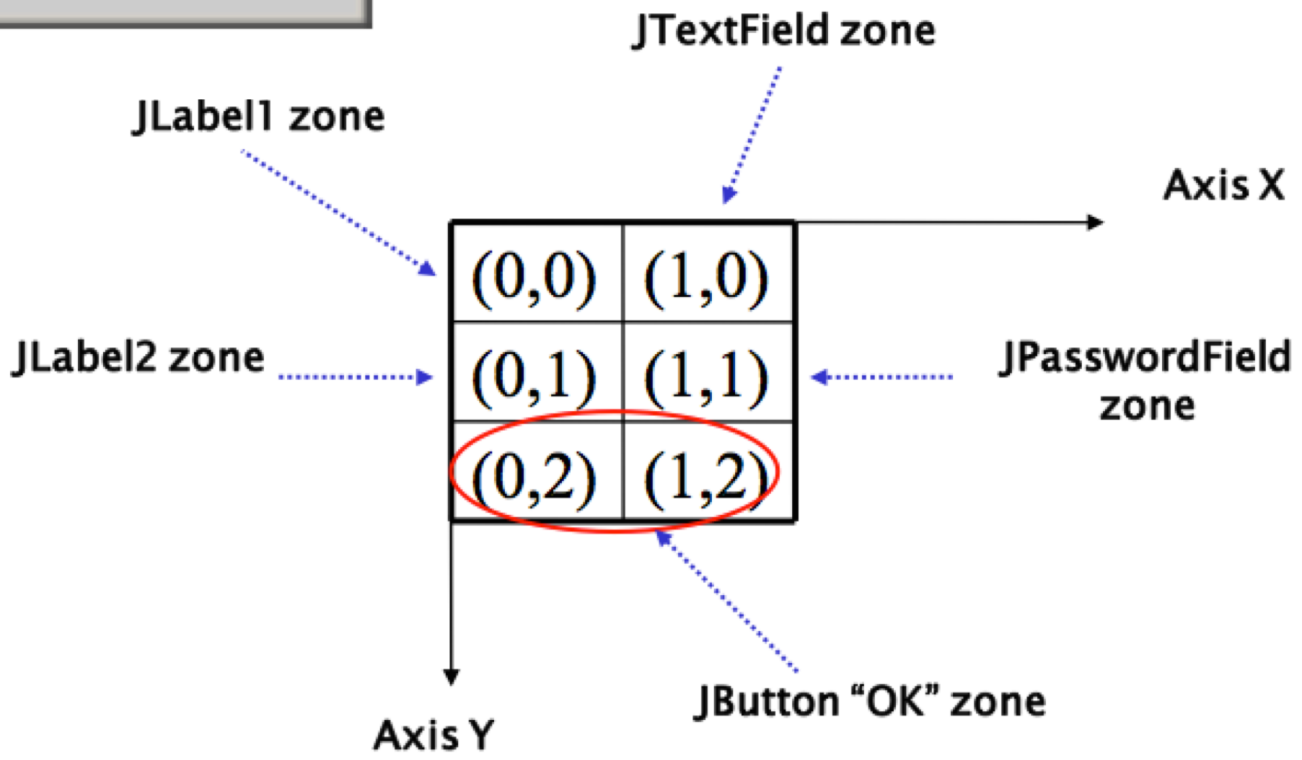
(Min distance = 10)

Layout Manager - GridLayout

- Splits the visual area in a grid of rows and columns
  - Starts from the box in the top left
- Constructors:
  - `GridLayout g = new GridLayout(int rows, int cols);`
  - `GridLayout g = new GridLayout(rows, cols, hgap, vgap);`
- Constructors parameters:
  - rows: number of row; cols: number of columns;
  - hgap: Spacing (in pixels) between two horizontal boxes (default: 0 pixel)
  - vgap: spacing (in pixel) between two vertical boxes (default: 0 pixel)

ING

Layout Manager - GridBagLayout



Layout Manager - GridBagLayout

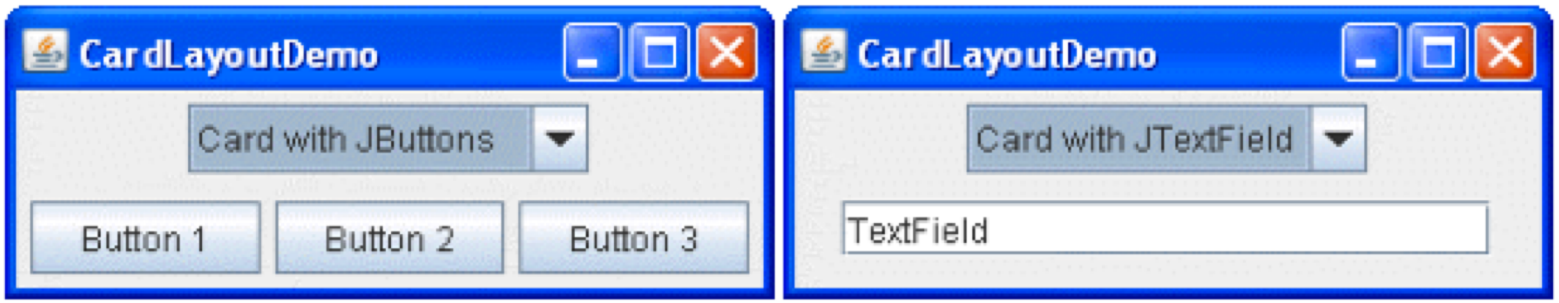
- Extension of GridLayout. Makes it possible to adjust the elements of the grid
- Methodology:

```
JPanel pane = new JPanel(new GridBagLayout());
GridBagConstraints c = new GridBagConstraints();

//For each component to be added to this container:
//...Create the component...
//...Set instance variables in the GridBagConstraints
instance...
pane.add(theComponent, c);
```

Layout Manager - CardLayout

- CardLayout allows to have different panels in the frame, but only one showed at time
  - the panels are called cards
- Methodology:

```
JPanel p = new JPanel(new CardLayout());
p.add("Panel1", new JPanel());
p.add("Panel2", new JPanel());
```



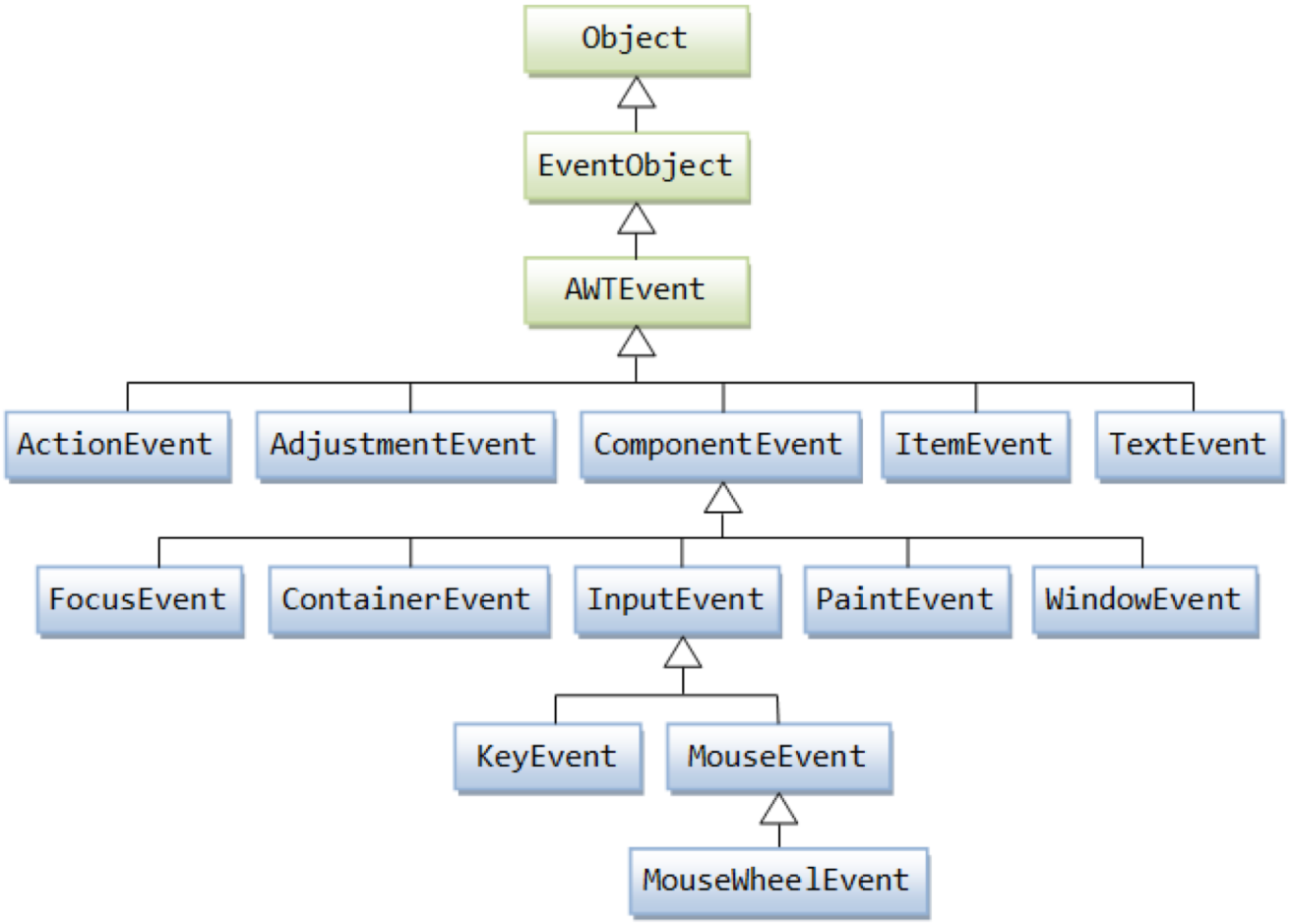
Java Swing Events

Università di Modena e Reggio Emilia

Prof. Nicola Bicocchi (nicola.bicocchi@unimore.it)

ING

EventObject



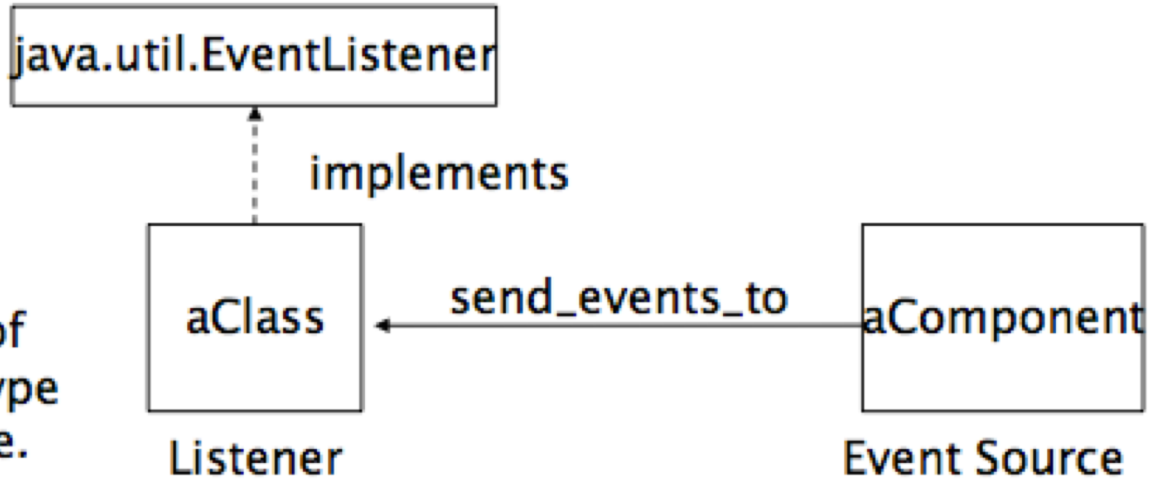
ING

Event Delegation Model

- Events are classified by type
  - MouseEvent, KeyEvent, ActionEvent
- Events are generated in components (source)
- Listeners (target) can be registered to components
- Whenever an event occurs, the event thread send a message to all the registered listeners (the event is passed as a parameter)
- Listeners must implement appropriate interfaces to make the callback mechanism possible

ING

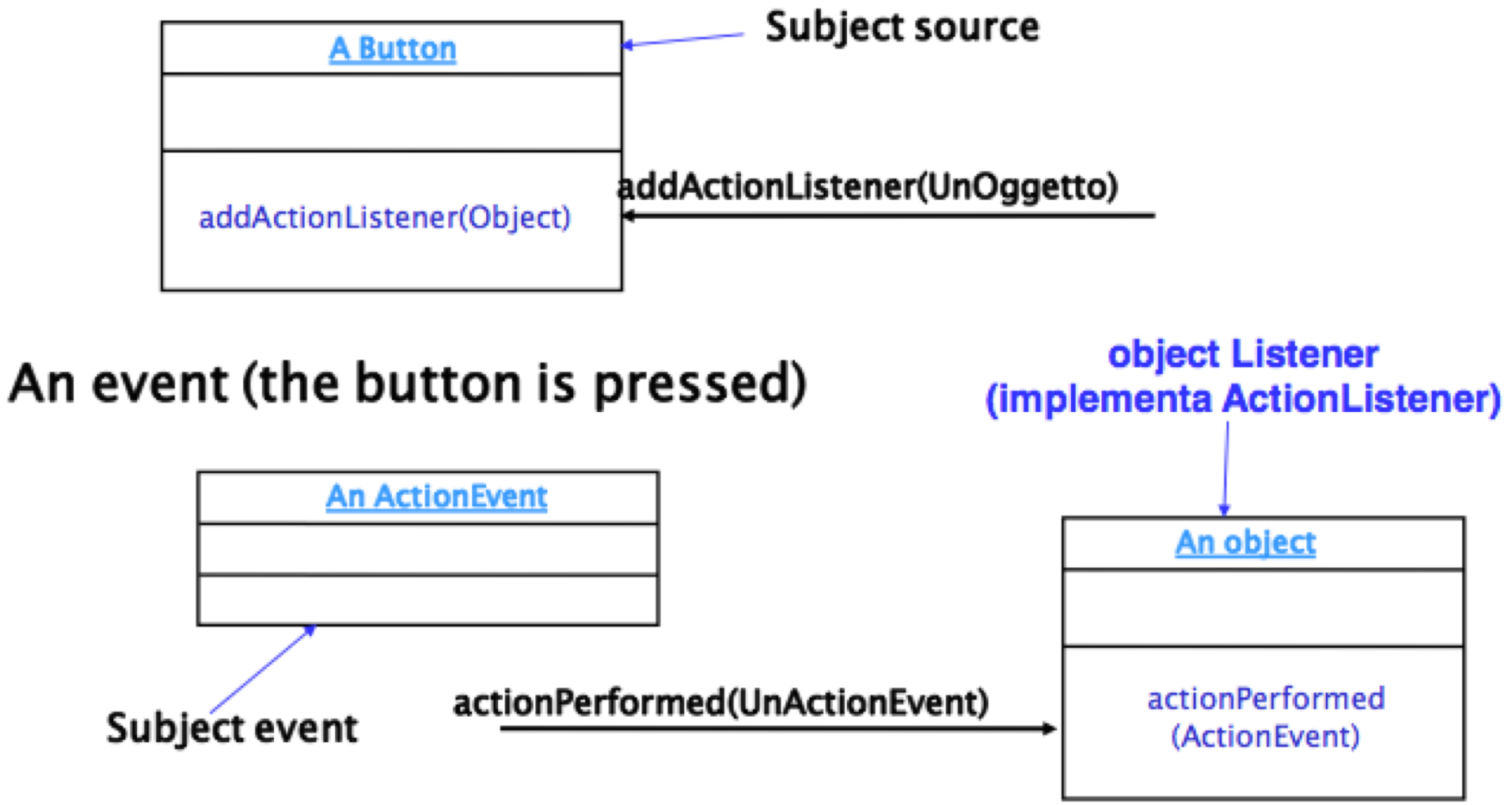
Event Delegation Model



Multiple listeners can register to be notified of events of a particular type from a particular source. Also, the same listener can listen to notifications from different objects.

ING

Example

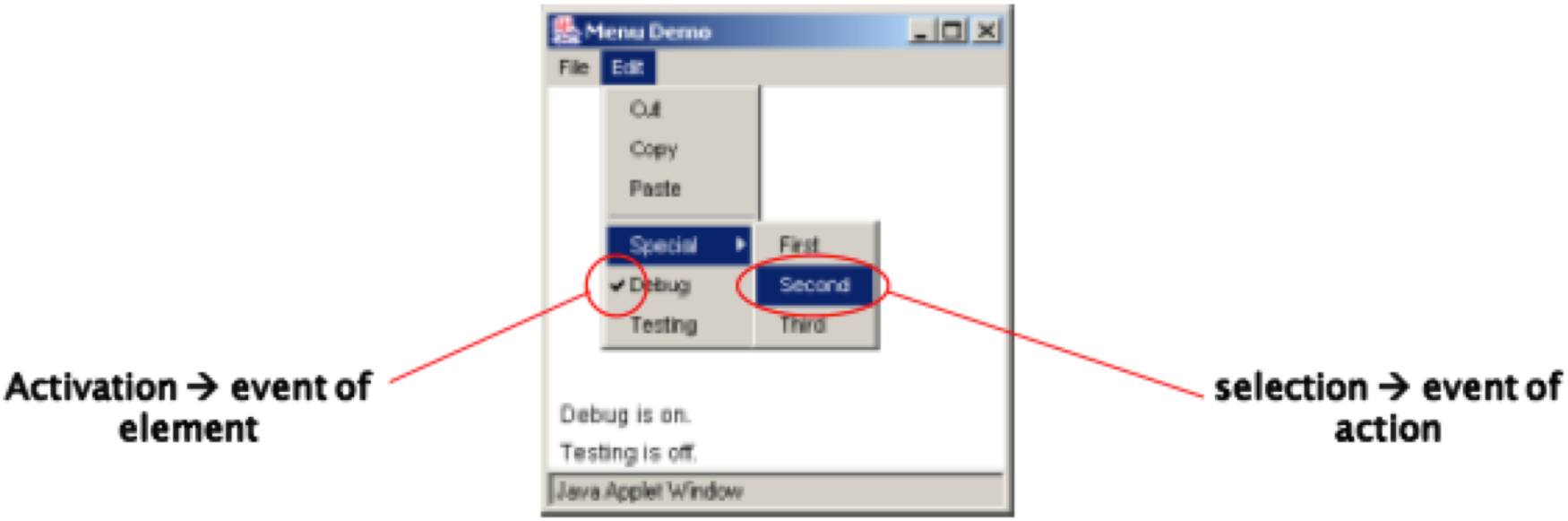

A Button
addActionListener(Object)
Subject source
addActionListener(UnOggetto)
An event (the button is pressed)
object Listener
(implementa ActionListener)
An ActionEvent
An object
Subject event
actionPerformed(UnActionEvent)
actionPerformed
(ActionEvent)

Event Delegation Model

- Events are organized by type and need specific listeners

User Action	Event Triggered	Event Listener interface
Click a Button, JButton	ActionEvent	ActionListener
Open, iconify, close Frame, JFrame	WindowEvent	WindowListener
Click a Component, JComponent	MouseEvent	MouseListener
Change texts in a TextField, JTextField	TextEvent	TextListener
Type a key	KeyEvent	KeyListener
Click/Select an item in a Choice, JCheckbox, JRadioButton, JComboBox	ItemEvent, ActionEvent	ItemListener, ActionListener

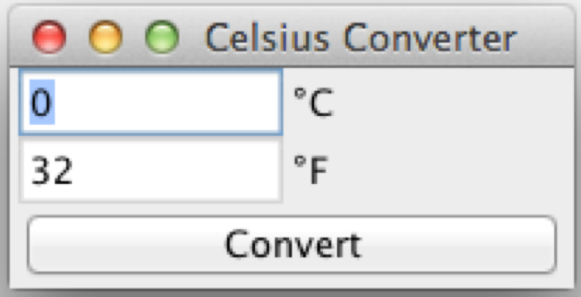
“selection” and “activation”



ING

A complete example

```
import java.awt.*;
import java.awt.event.*;

import javax.swing.*;

public class CelsiusConverter extends JFrame implements ActionListener {
      private JButton convertButton;
      private JTextField fahrenheitLabel;
      private JTextField tempTextField;

      public CelsiusConverter() {
            super("Celsius Converter");
            tempTextField = new JTextField("0");
            fahrenheitLabel = new JTextField("32");
            fahrenheitLabel.setEditable(false);
            convertButton = new JButton("Convert");
            convertButton.addActionListener(this);

            JPanel p = new JPanel();
            p.setLayout(new GridLayout(2, 2));
            p.add(tempTextField); p.add(new JLabel("°C"));
            p.add(fahrenheitLabel); p.add(new JLabel("°F"));
```



A complete example

```
        setLayout(new BorderLayout());
        add(p, BorderLayout.CENTER);
        add(convertButton, BorderLayout.SOUTH);

        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        setSize(200, 100);
        setVisible(true);
    }

    public void actionPerformed(ActionEvent e) {
        int tempFahr = (int)
            ((Double.parseDouble(tempTextField.getText())) * 1.8 + 32);
        fahrenheitLabel.setText(Integer.toString(tempFahr));
    }

    public static void main(String[] args) {
        new CelsiusConverter();
    }
}
```

How to manage events in Java

- The principle underlying the events is quite similar to the exceptions :
  - the receivers declare which event are able to deal with (one or more) by implementing the needed interfaces
  - the components that are source of events (JButton, JTextField, etc..) select their receivers
    - button.addActionListener(receiver)
- Pay attention! You're implementing interfaces, so you must implement all methods of those interfaces!

How to manage events in Java

- Components can:
  1. Handle events on their own
     - *In case of large number of components*
  2. Delegate events to their container
     - *In case of small/medium number of components*
  3. Delegate events to external classes
     - *Rarely used (produces unneeded classes)*

Handle events on their own

```
JButton btn = new Jbutton();
btn.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
    // do something
}});
```

Delegate events to their container

```
class FrameWithEvents extends JFrame implements InterfaceWithEvents {
    JComponent componentSourceofEvents = new JComponent();
    componentSourceOfEvents.addListener(this);

    void methodOfTheInterfaceWithEvents() {...}
    void anotehrMethodOfTheInterfaceWithEvents() {...}
}
```

Delegate events to external classes

```
class MyListener  implements InterfaceWithEvents {
    void methodOfTheInterfaceWithEvents() {...}
    void anotehrMethodOfTheInterfaceWithEvents() {...}
} //end class


class Frame extends JFrame {
    MyListener listener = new MyListener();
    JComponent componentSourceofEvents = new JComponent();
    componentSourceOfEvents.addListener(listener);
} //end class
```

Dealing with multiple sources

- getSource() and object references
  - if (e.getSource() == buttonSelfDestruction) {}
- getActionCommand() and custom strings
  - If (e.getActionCommand() == “destroy”) {}
- Event classes
  - If (e instanceof(KeyEvent)) {}

ING

Event Interfaces

- ActionListener
  - void actionPerformed (ActionEvent evt)
- FocusListener
  - void focusGained (FocusEvent evt)
  - void focusLost (FocusEvent evt)
- ItemListener
  - void itemStateChanged (ItemEvent evt)

ING

Event Interfaces

- MouseListener
  - void mouseClicked (MouseEvent evt)
  - void mouseEntered (MouseEvent evt)
  - void mouseExited (MouseEvent evt)
  - void mousePressed (MouseEvent evt)
  - void mouseReleased (MouseEvent evt)
- MouseMotionListener
  - void mouseDragged (MouseEvent evt)
  - void mouseMoved (MouseEvent evt)

Event Interfaces

- KeyListener
  - void keyPressed (KeyEvent evt)
  - void keyReleased(KeyEvent evt)
  - void keyTyped(KeyEvent evt)
- WindowListener
  - void windowActivated(WindowEvent evt)
  - void windowClosed (WindowEvent evt)
  - void windowClosing (WindowEvent evt)
  - void windowDeactivated (WindowEvent evt)
  - void windowDeiconified (WindowEvent evt)
  - void windowIconified (WindowEvent evt)
  - void windowOpened (WindowEvent evt)