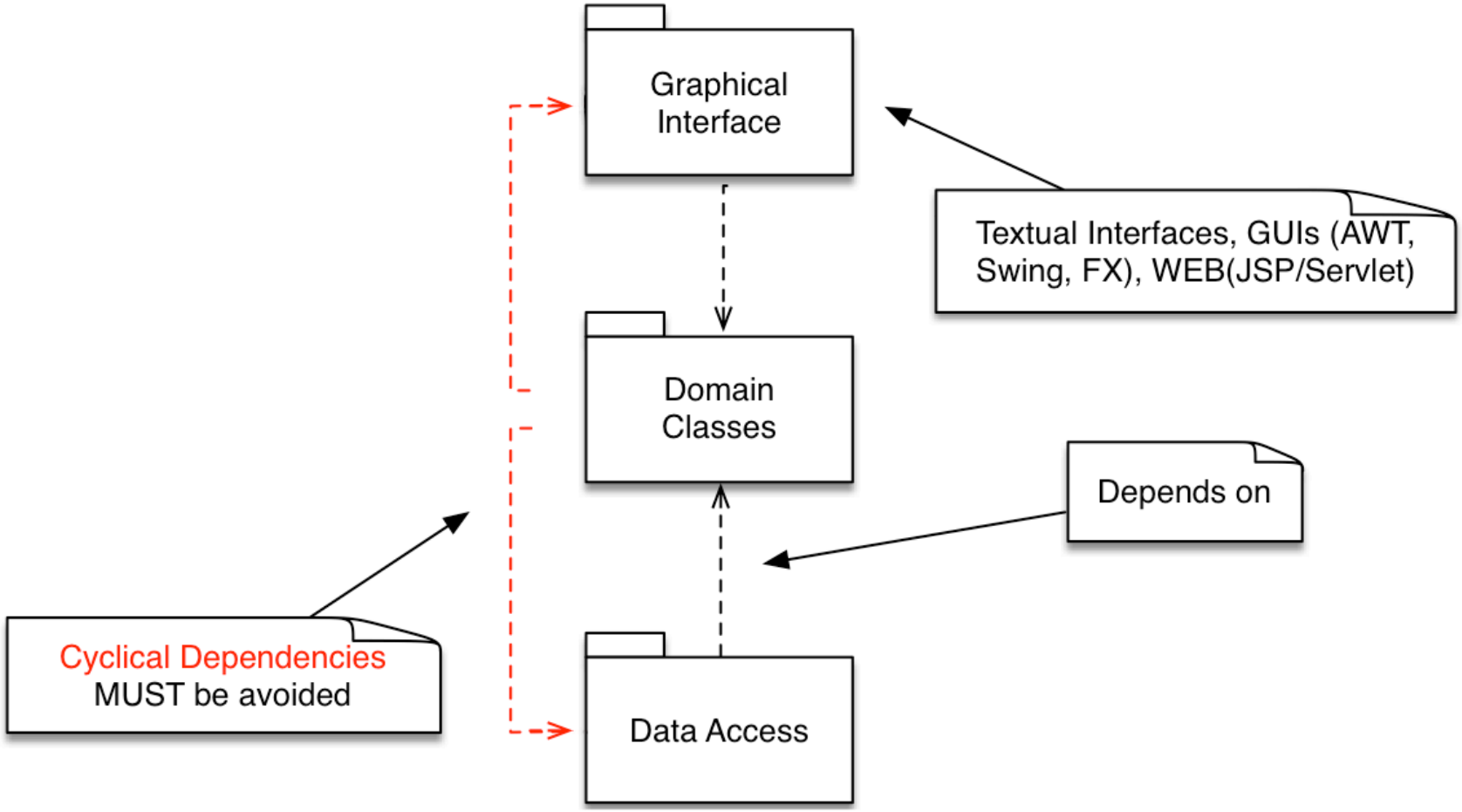
Java Swing

Università di Modena e Reggio Emilia

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

ING

Software Design



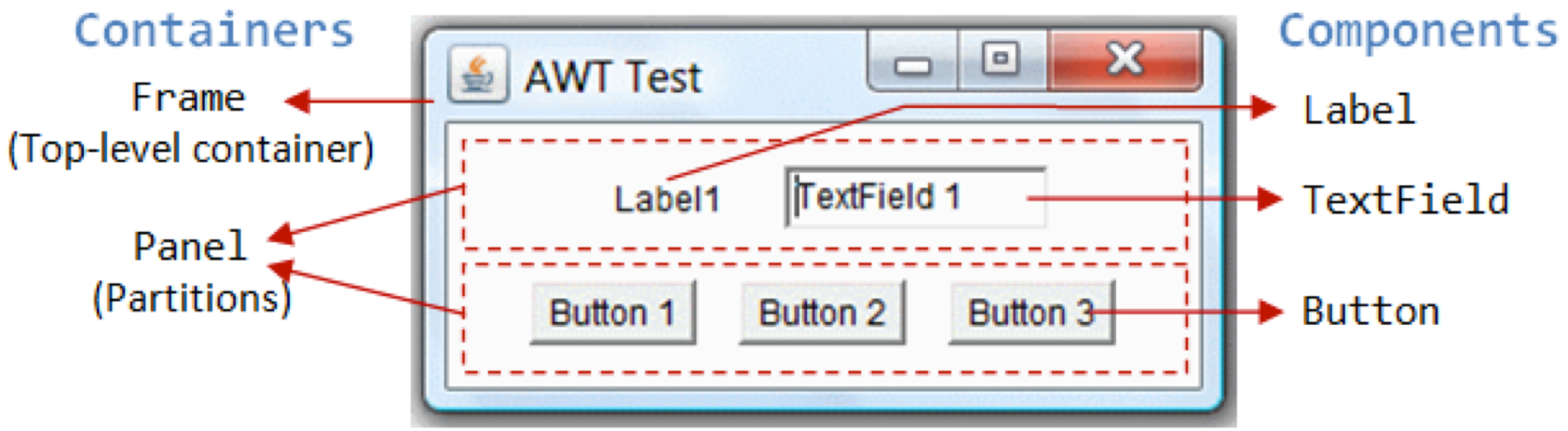
Package java.awt.*

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

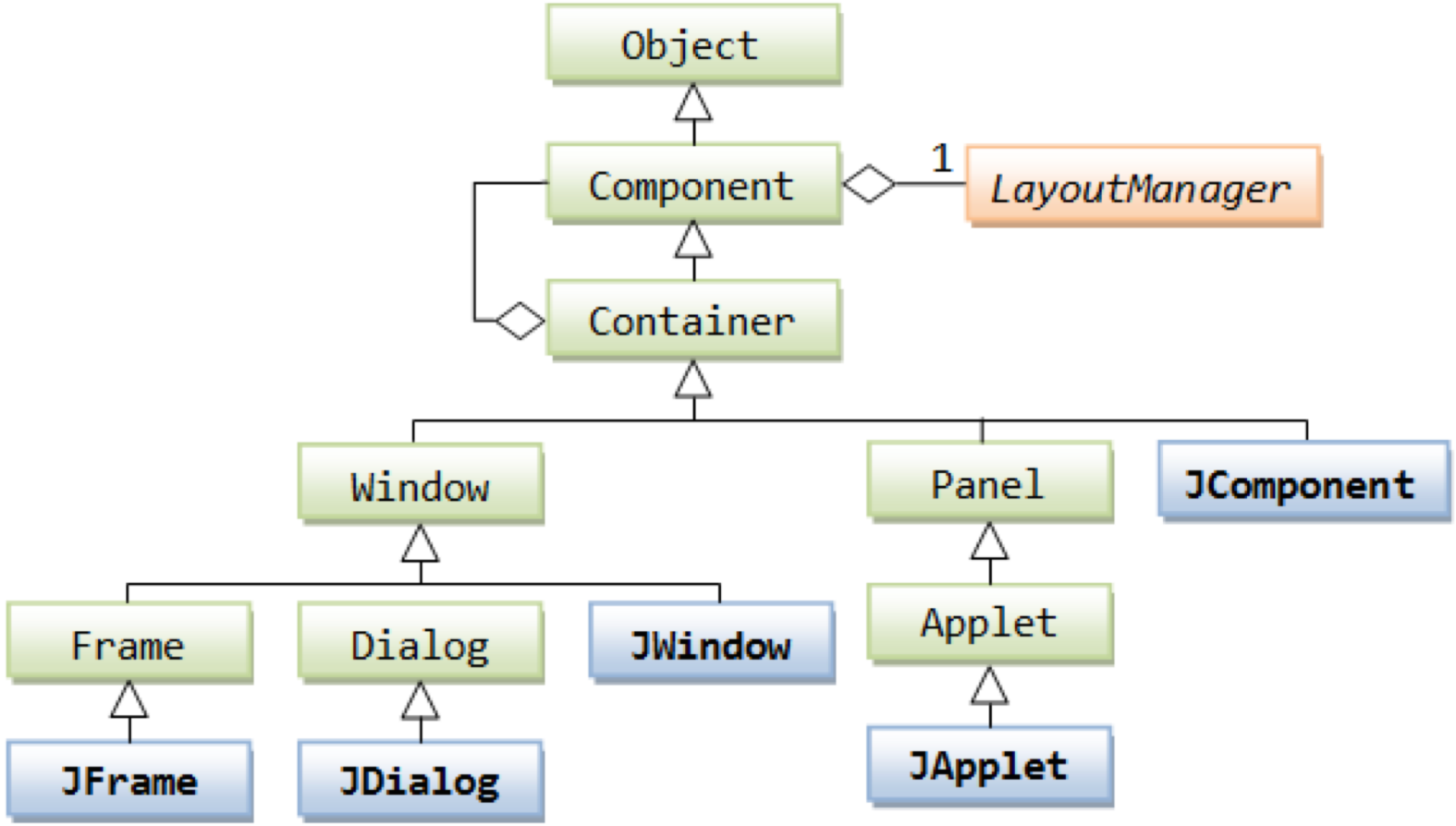
Package javax.swing.*

- 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

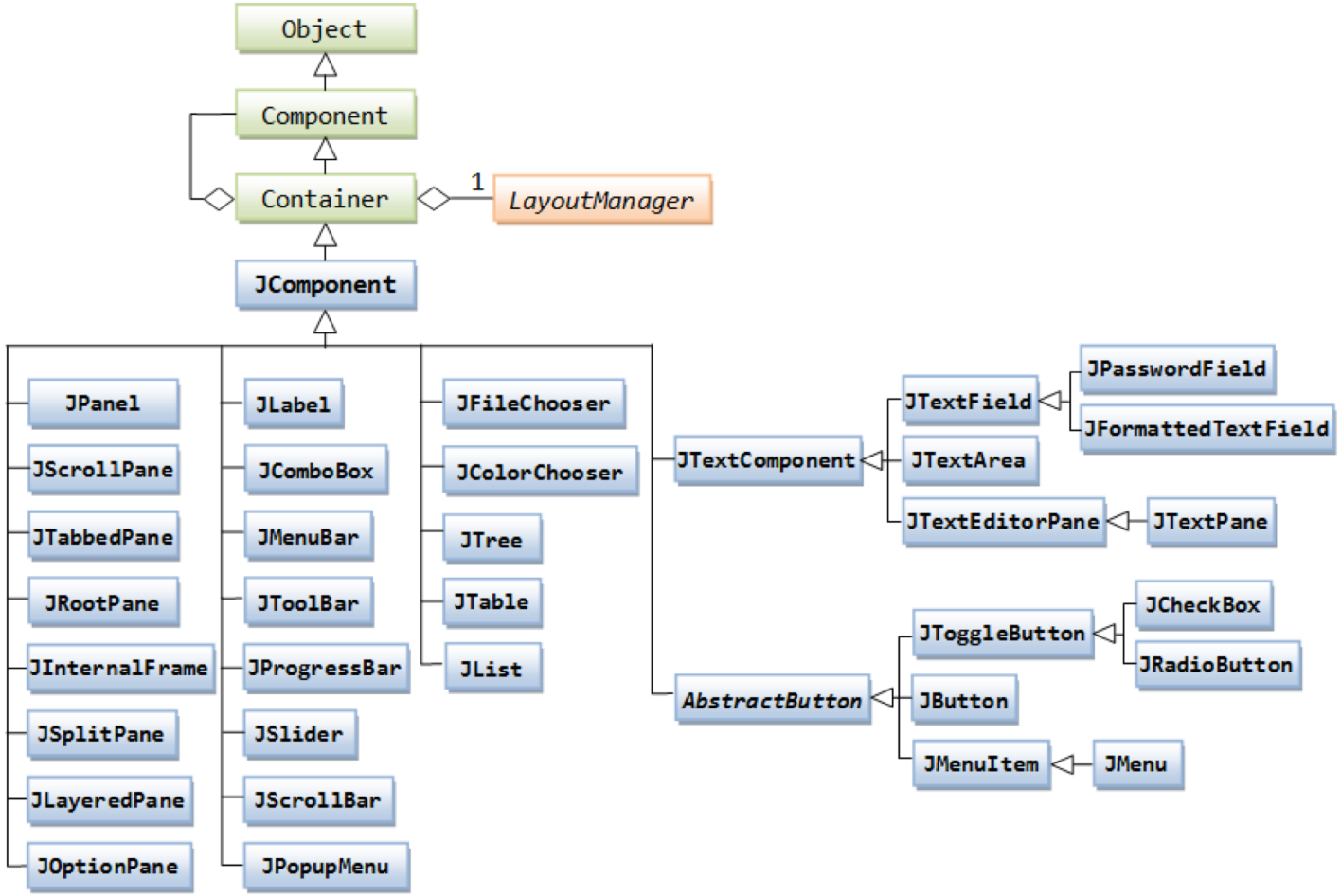
Containers and components



Class hierarchy (Containers)



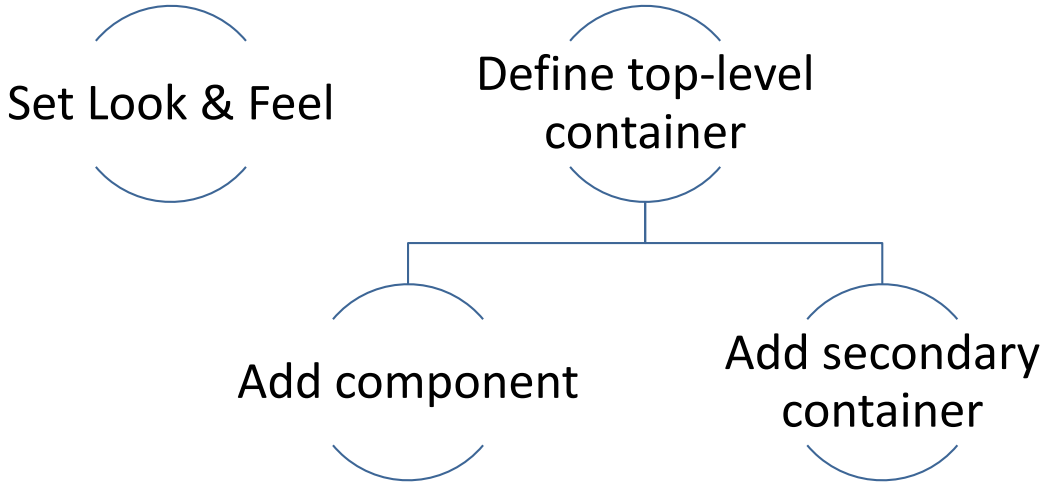
Class hierarchy (Components)



ING

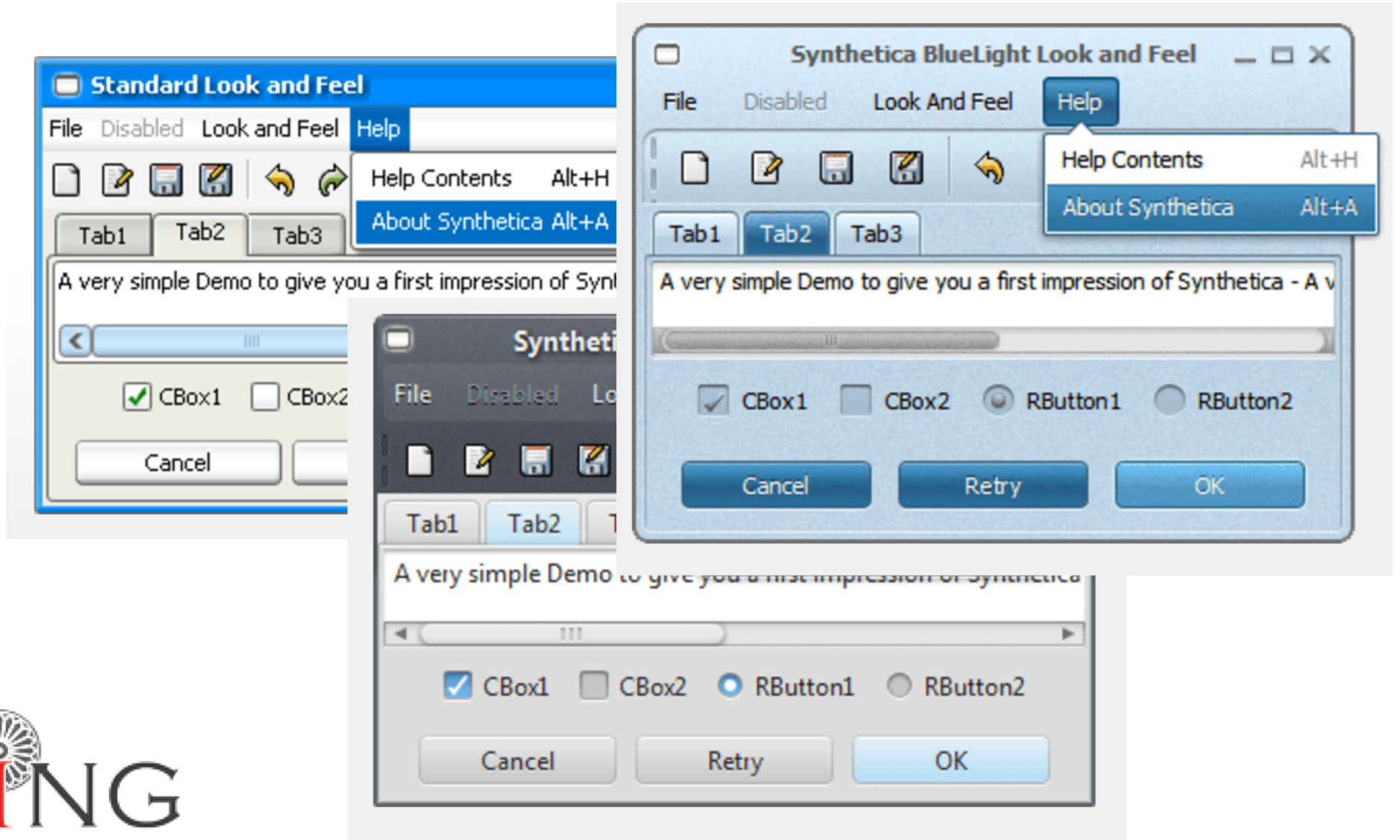
Graphical Programming

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



ING

Look & Feel

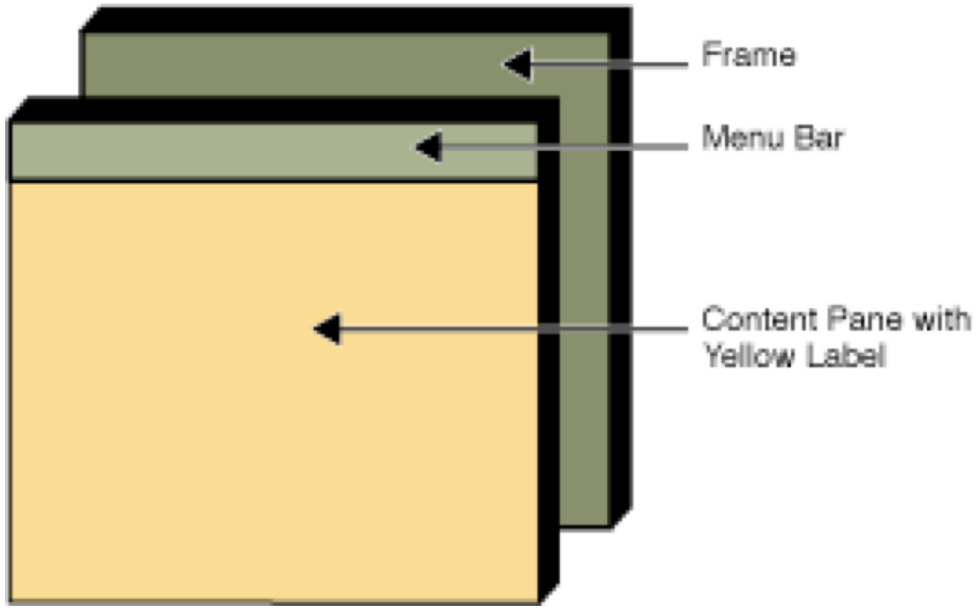

Standard Look and Feel
File Disabled Look and Feel Help
Help Contents Alt+H
About Synthetica Alt+A
Tab1 Tab2 Tab3
A very simple Demo to give you a first impression of Synt
CBox1 CBox2
Cancel
Synthetica BlueLight Look and Feel
File Disabled Look And Feel Help
Help Contents Alt+H
About Synthetica Alt+A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

Look & Feel

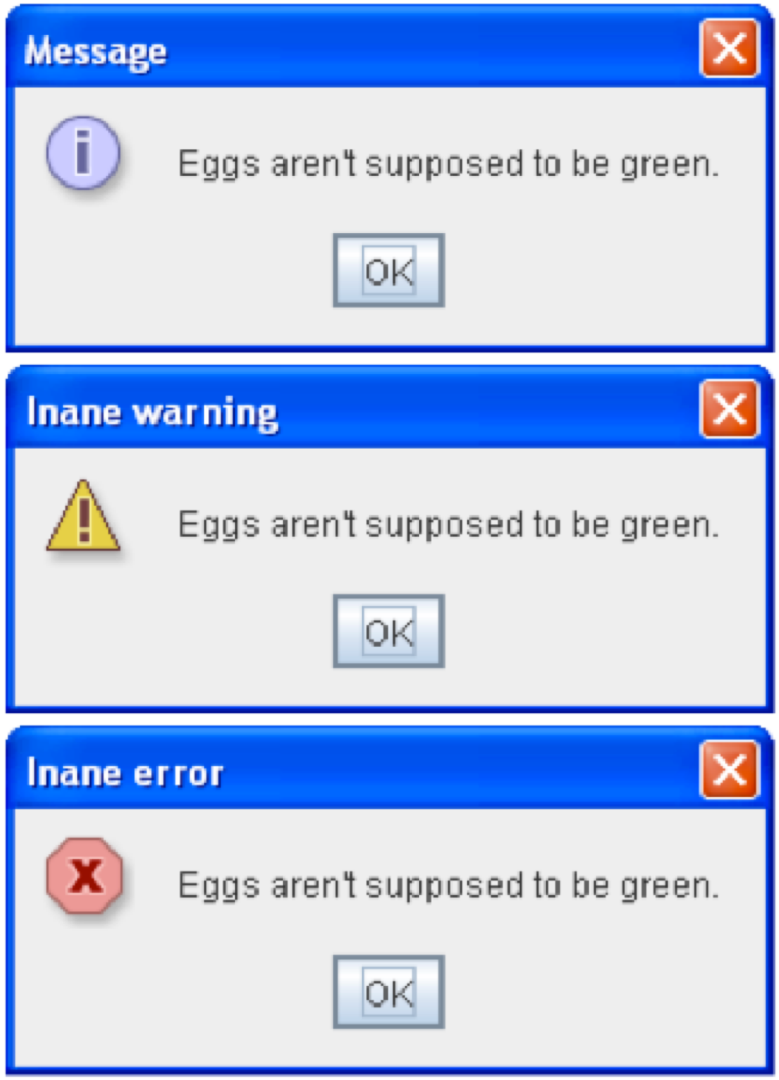
- UIManager manages the current look and feel
  - http://www.jyloo.com/synthetica/themes/

```
/* 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");
```

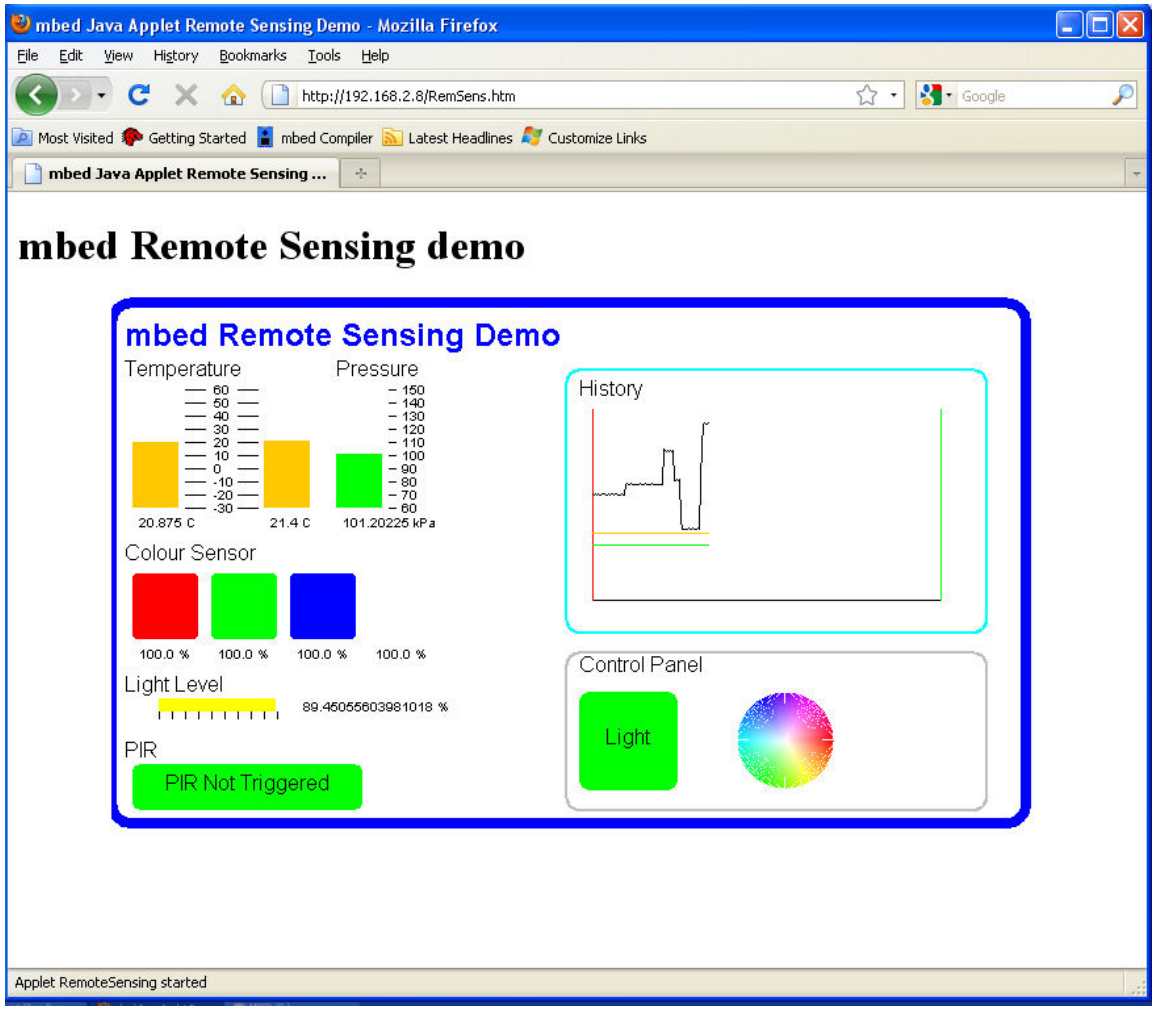
Top-level container: JFrame



Top-level container: JDialog

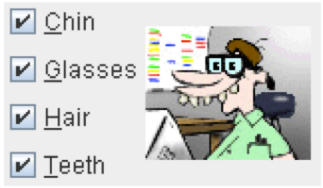


Top-level container: JApplet *(deprecated)*

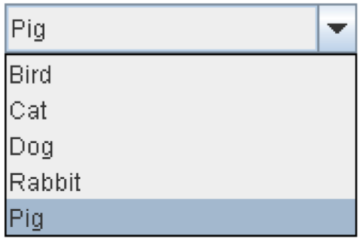


Components, a visual guide

JButton

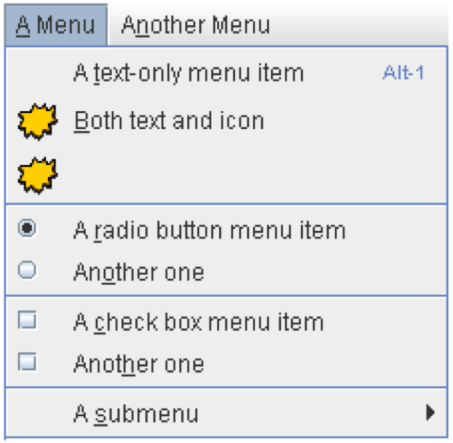


JCheckBox



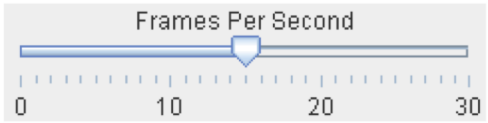
JComboBox

JList



JMenu

JRadioButton



JSlider

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

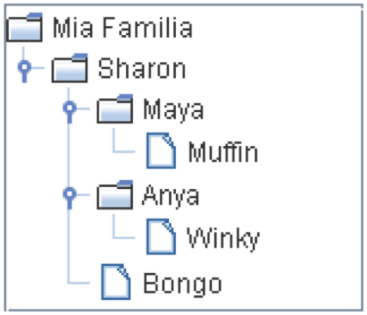
Components, a visual guide

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

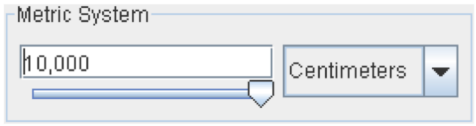
JProgressBar

JSeparator

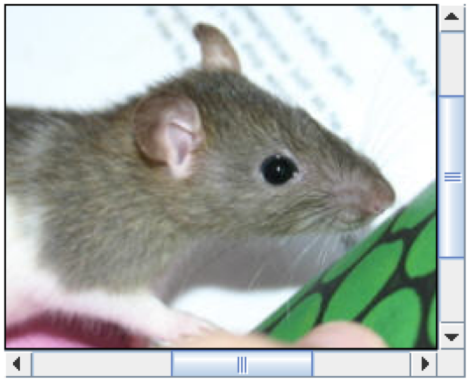
JToolTip

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

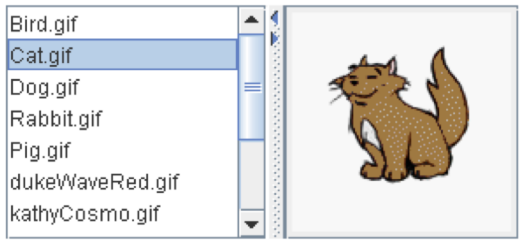
Components, a visual guide



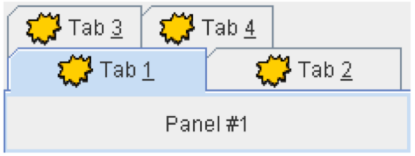
JPanel



JScrollPane



JSplitPane



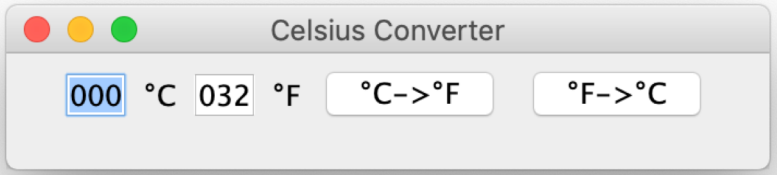
JTabbedPane

JToolBar

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

A complete example

```
public class CelsiusConverterBareMinimum extends JFrame {
	private JButton CFButton, FCButton;
	private JTextField fahrenheitTF, celsiusTF;

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

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

		setContentPane(p1);
		setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
		setSize(350, 75);
		setVisible(true);
	}
}
```



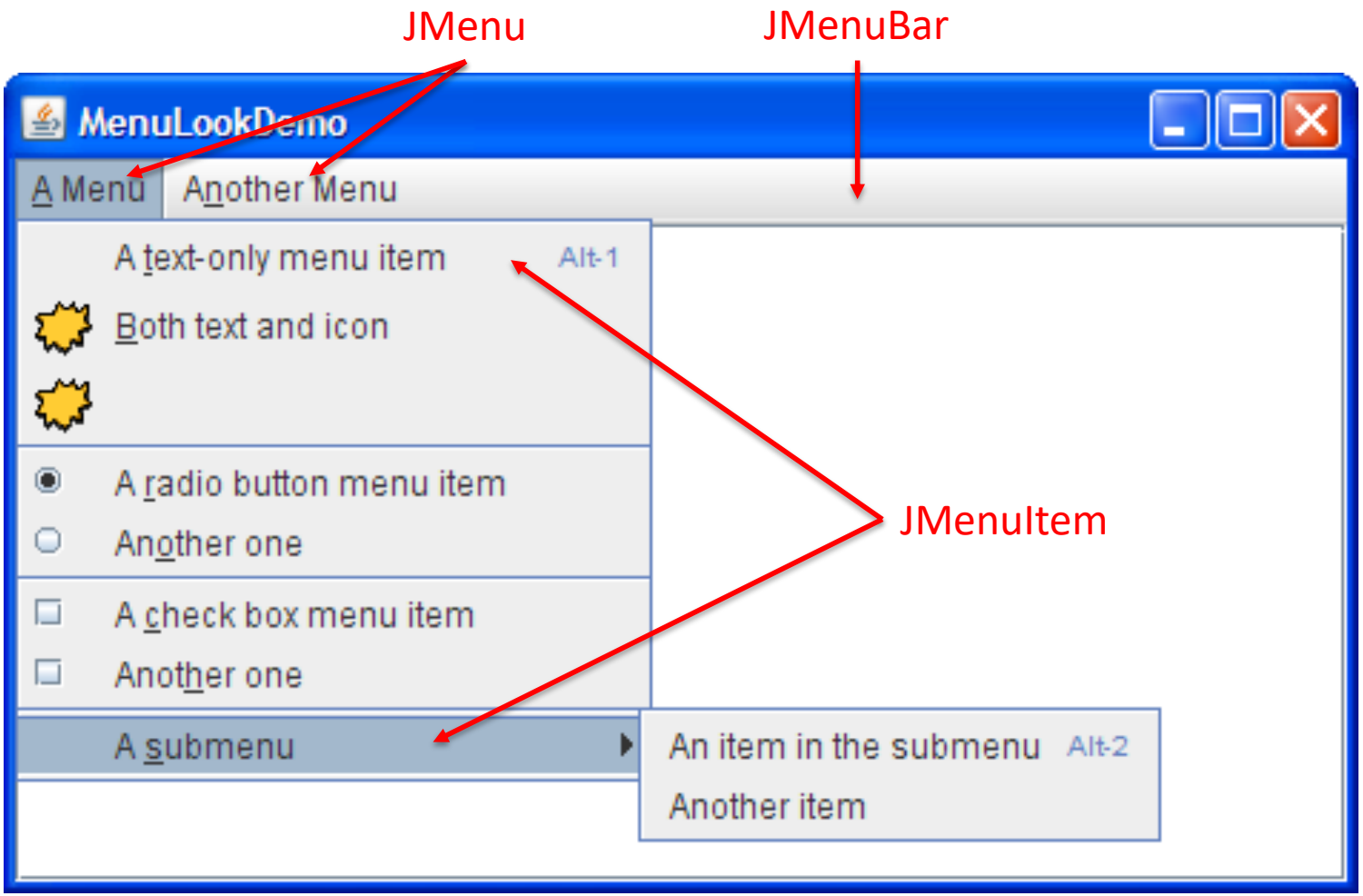
JFrame basic methods

- getContentPane()
  - retrieves the primary JPanel
- setContentPane(Container c)
  - sets the primary JPanel
- add(Component c)
  - add a component to the primary JPanel
- 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



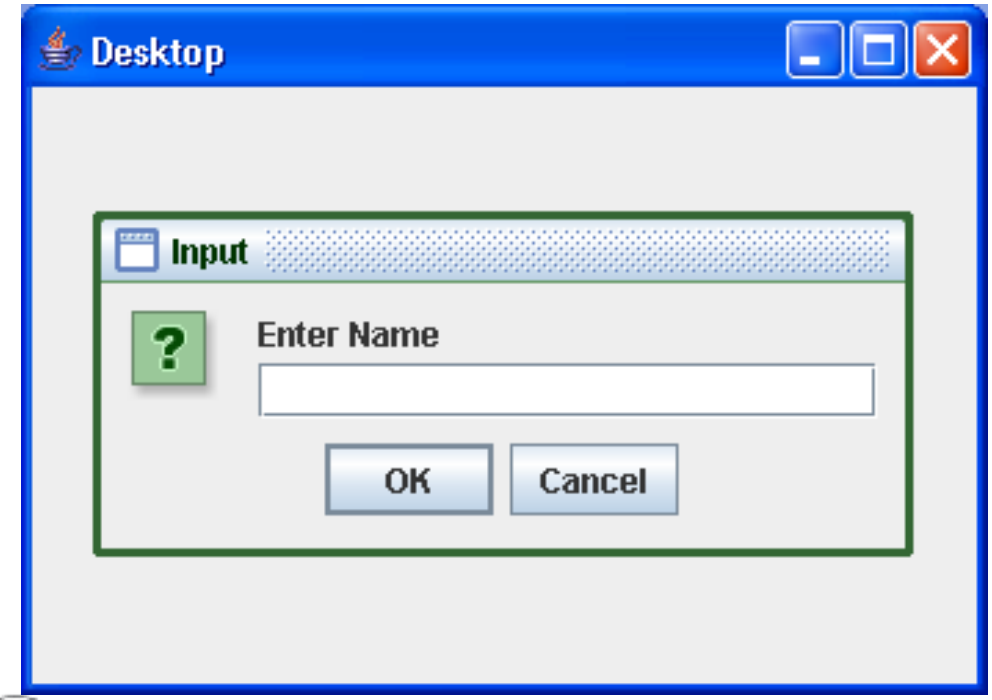
JFrame MenuBar

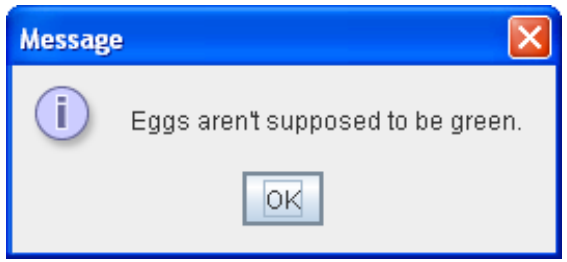
- In composing menus, three components are hierarchically involved: JMenuBar, JMenu, JMenuItem

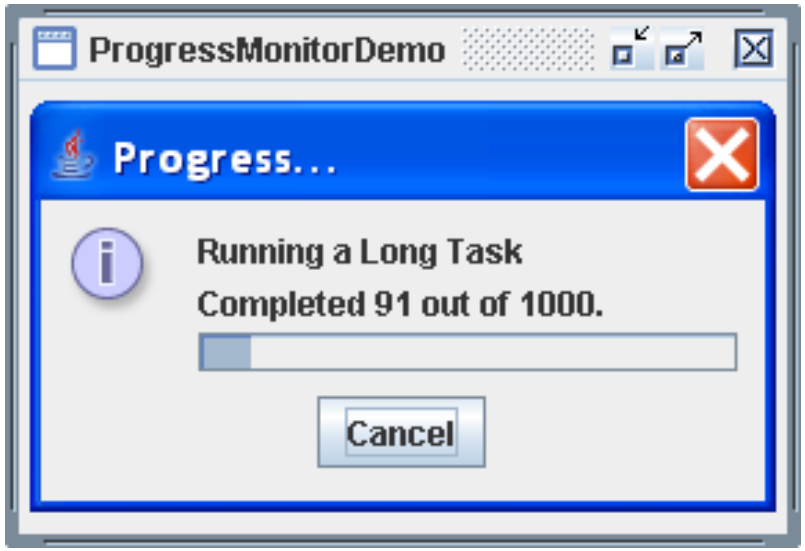
```
JMenuItem openFile = new JMenuItem("Open");
JMenuItem closeFile = new JMenuItem("Close");

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, feedback and advise to their users!







ING

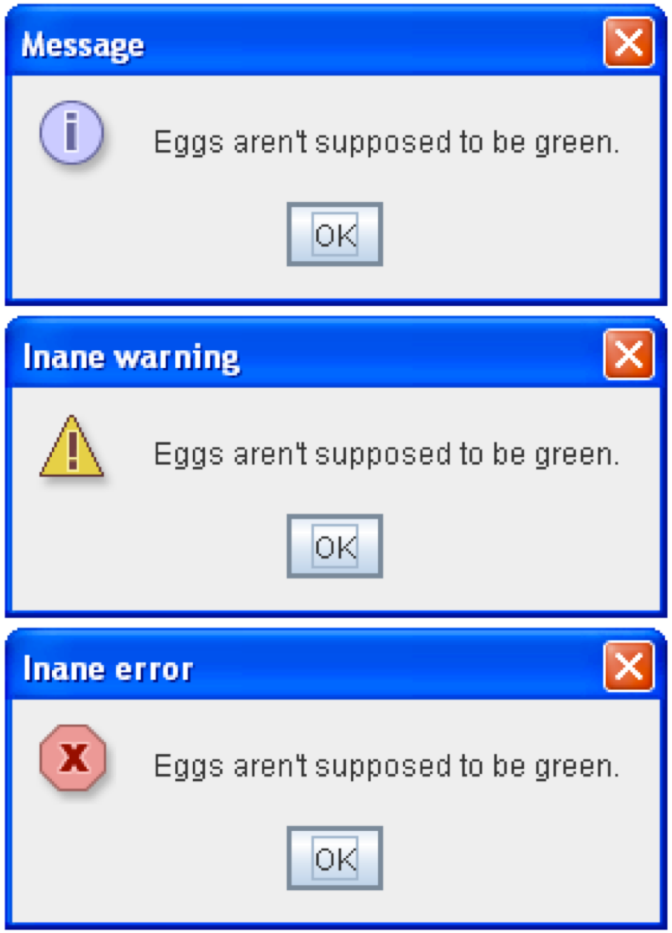
JDialog

- Dialogs are a better choice than instantiating multiple JFrames!
  - Every dialog is dependent on a top-level container.
  - Dialogs are all instances of JDialog, and can be generally composed following the same guidelines seen for JFrames. Nevertheless, the majority is built automatically 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 with icons, buttons, title and 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);
```

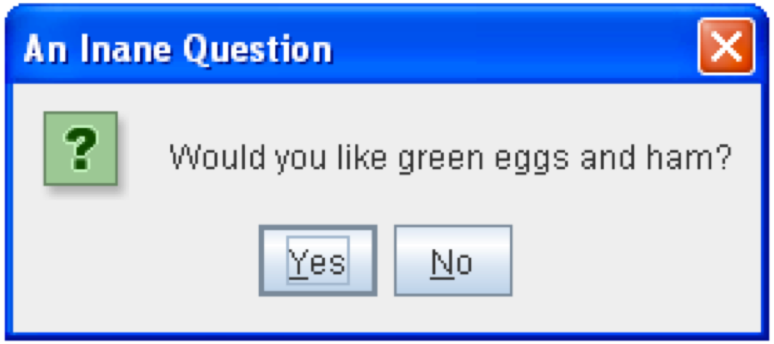
ING

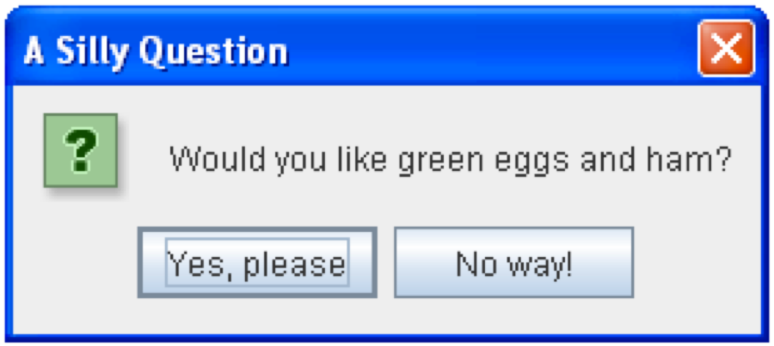
JOptionPane.showOptionDialog()

- 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]);
```

JOptionPane.showConfirmationDialog()



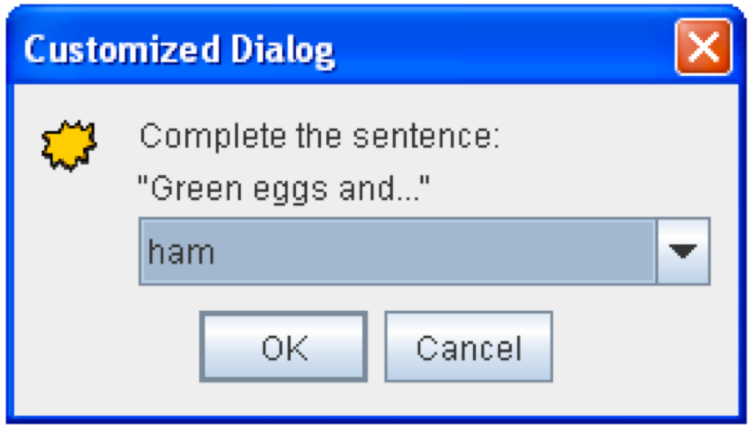



```
//default icon, custom title
int n = JOptionPane.showConfirmDialog(
    frame,
    "Would you like green eggs and ham?",
    "An Inane Question",
    JOptionPane.YES_NO_OPTION);
```

```
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()

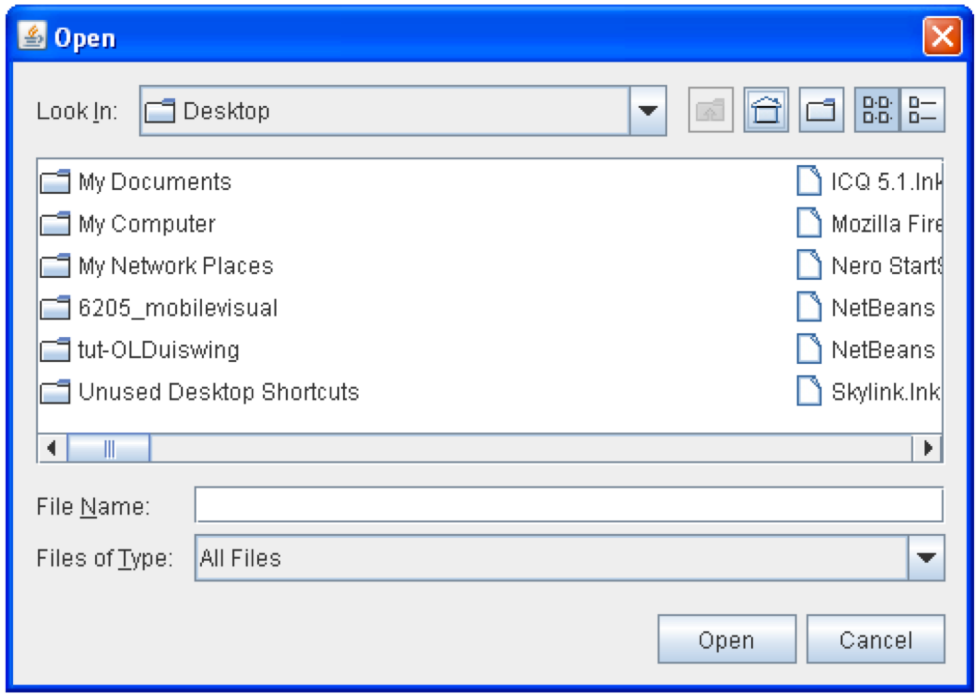



```
Object[] possibilities = {"ham", "spam", "yam"};
String s = (String)JOptionPane.showInputDialog(
                    frame,
                    "Complete the sentence:\n"
                    + "\"Green eggs and...\"",
                    "Customized Dialog",
                    JOptionPane.PLAIN_MESSAGE,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

JFileChooser

- Provides a GUI for navigating the file system

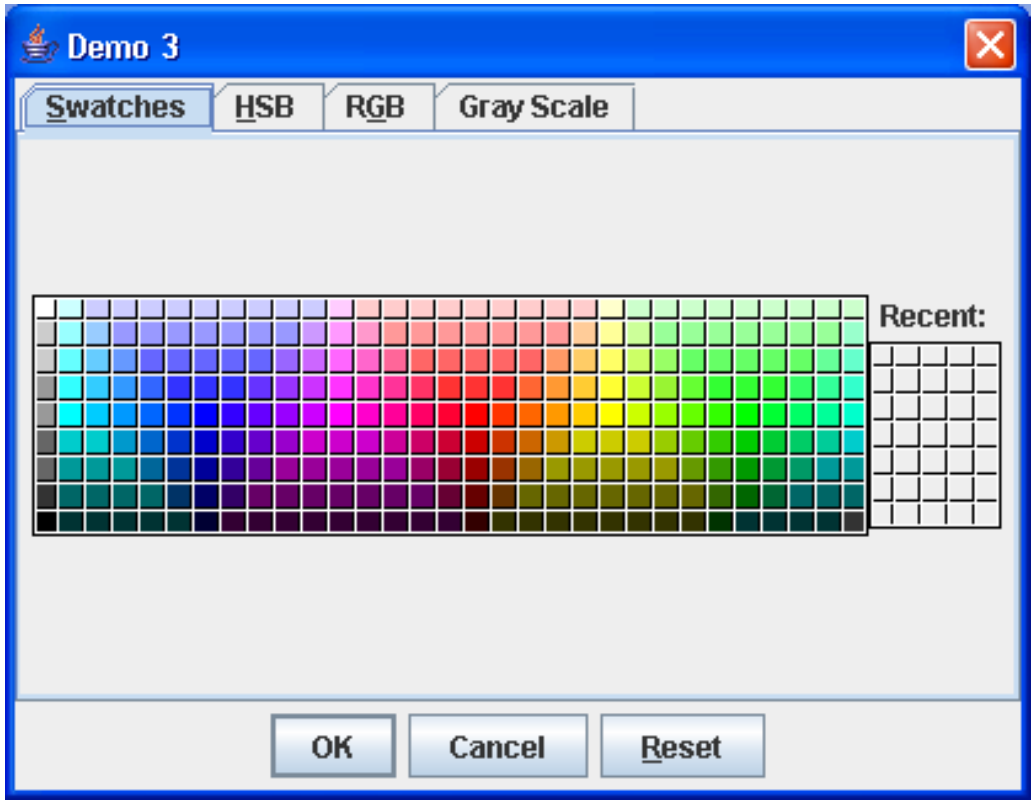
```
JFileChooser open = new JFileChooser();
int option = open.showOpenDialog(this);
if (option == JFileChooser.APPROVE_OPTION) {
    /* do stuff */
}
```



JColorChooser

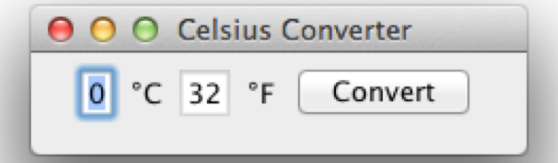
- Provides a GUI for navigating color spaces

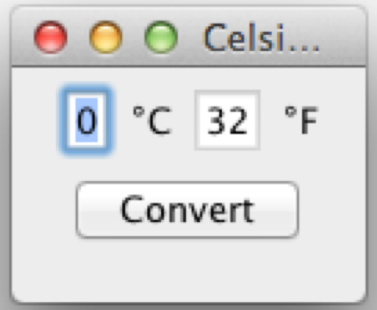
```
Color c = JColorChooser.showDialog(Component parent,
String title, Color initialColor)
```



What is a layout?

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





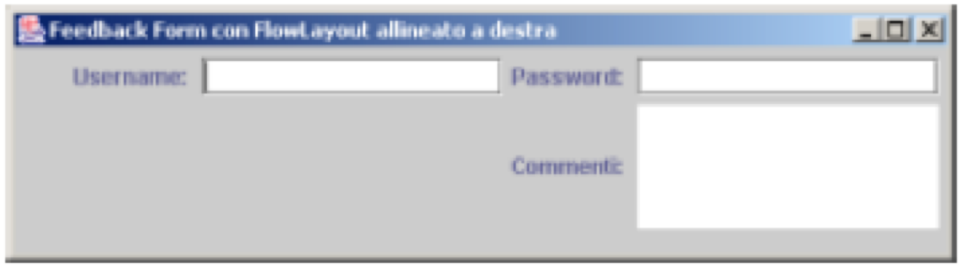
ING

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 layouts
  - Layout managers are passed to JPanel via the Jpanel constructor
- Methodology:

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

Layout Manager - FlowLayout



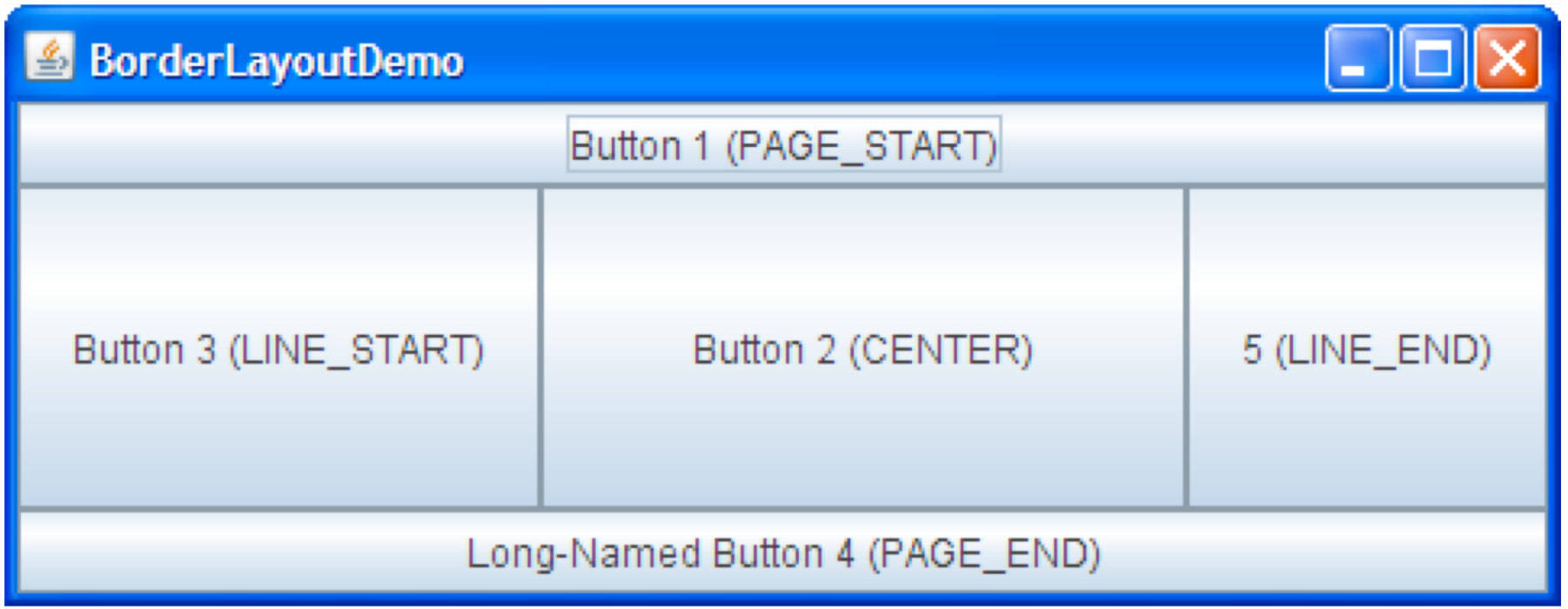
(FlowLayout.RIGHT)

(FlowLayout.CENTER,20,20)

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 (FlowLayout.LEFT, FlowLayout.RIGHT, FlowLayout.CENTER)
  - hgap: Horizontal space between components (default: 3 pixel)
  - vgap: Vertical space between components (default: 3 pixel)

Layout Manager - BorderLayout



ING

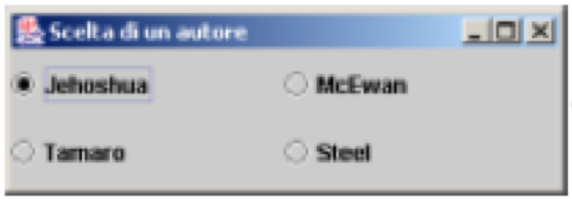
Layout Manager - BorderLayout

- Splits a container 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());
```

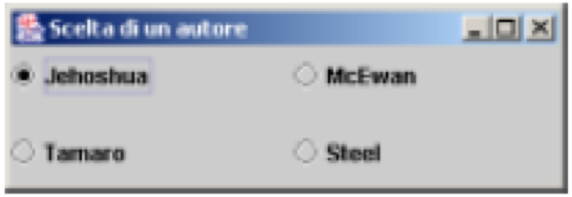
Layout Manager - GridLayout

4 row, 1 columns:
distance 0



2 row, 2 columns:
distance 0 pixel

(Distanza min = 0)



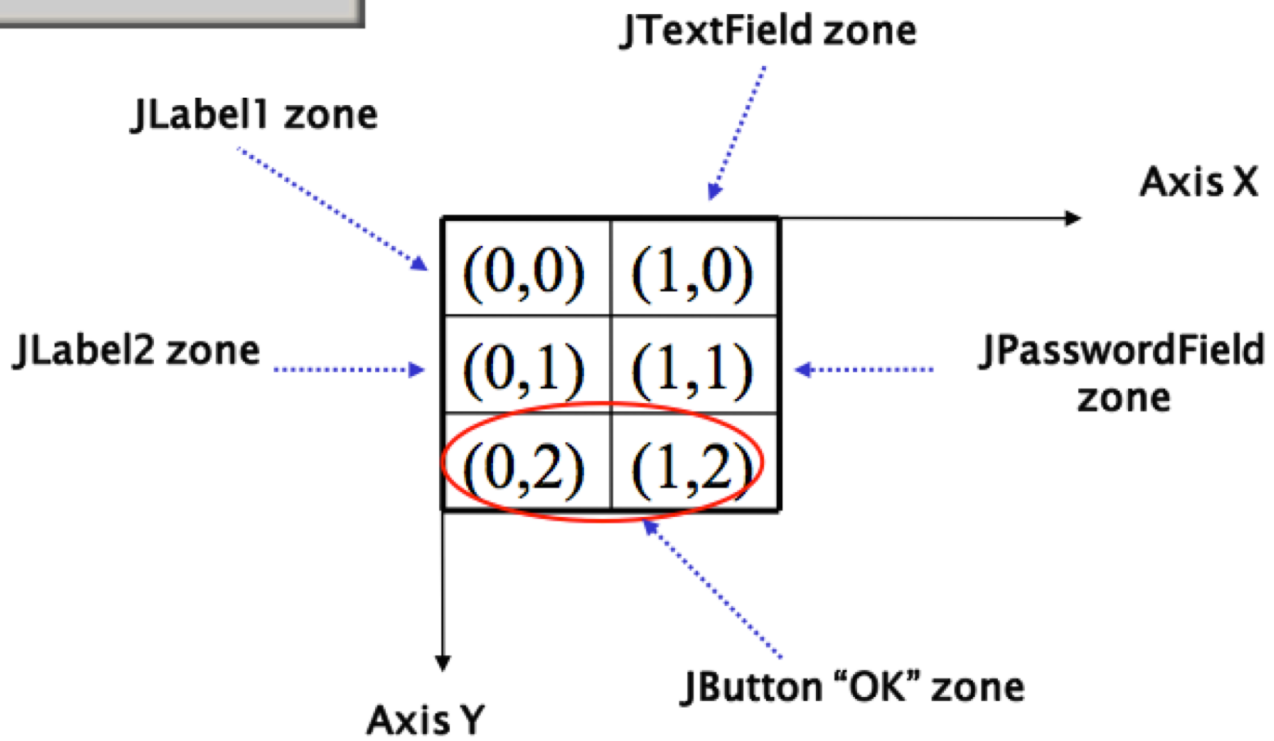
2 row, 2 columns:
distance 10 pixel

(Min distance = 10)

Layout Manager - GridLayout

- Splits the visual area in a grid of rows and columns
  - Starts filling from the top-left cell
- 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)

Layout Manager - GridBagLayout



Layout Manager - GridBagLayout

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

```
JPanel p = 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...
p.add(theComponent, c);
```

Layout Manager - CardLayout

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

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

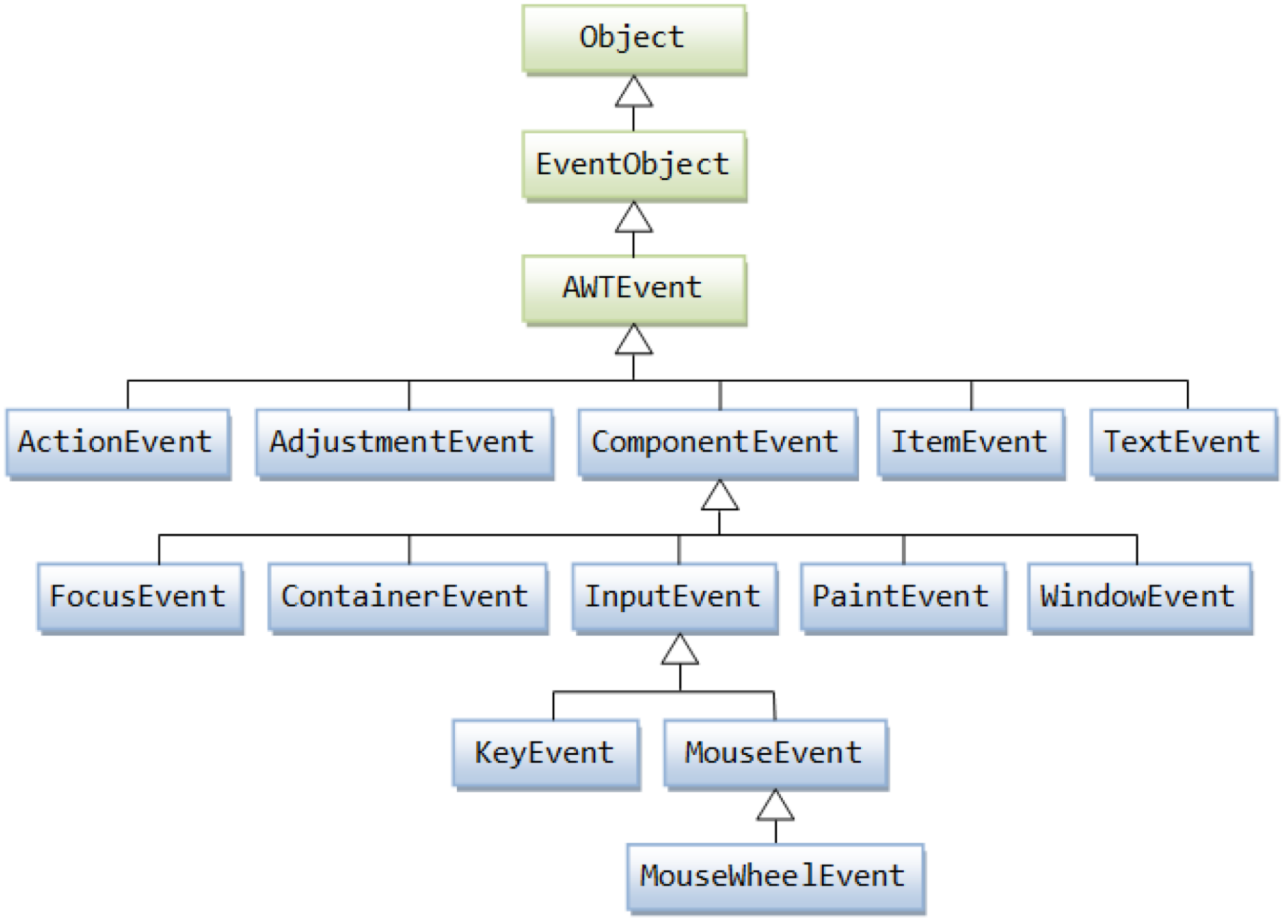
Java Swing Events

Università di Modena e Reggio Emilia

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

ING

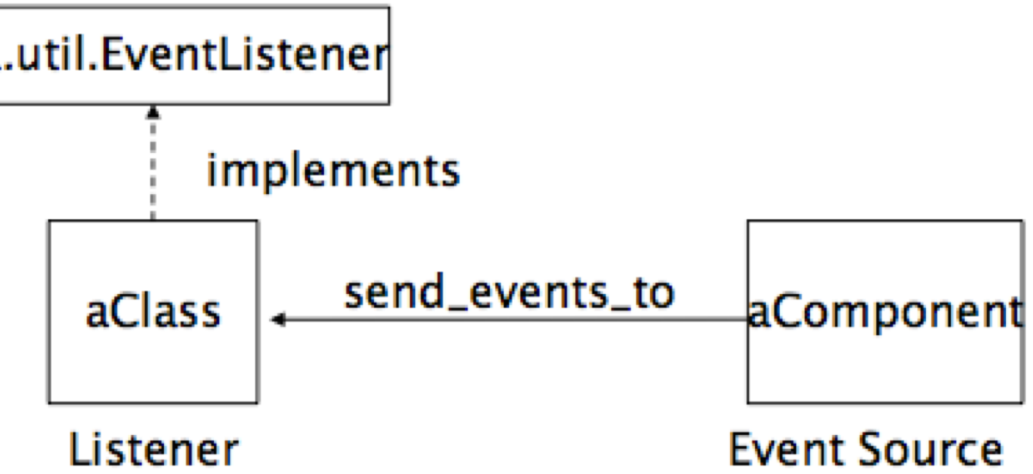
EventObject



Event Delegation Model

- Events are classified by a subtype of AWTEvent
  - 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 to listeners as a parameter)
- Listeners must implement appropriate interfaces to make the callback mechanism possible

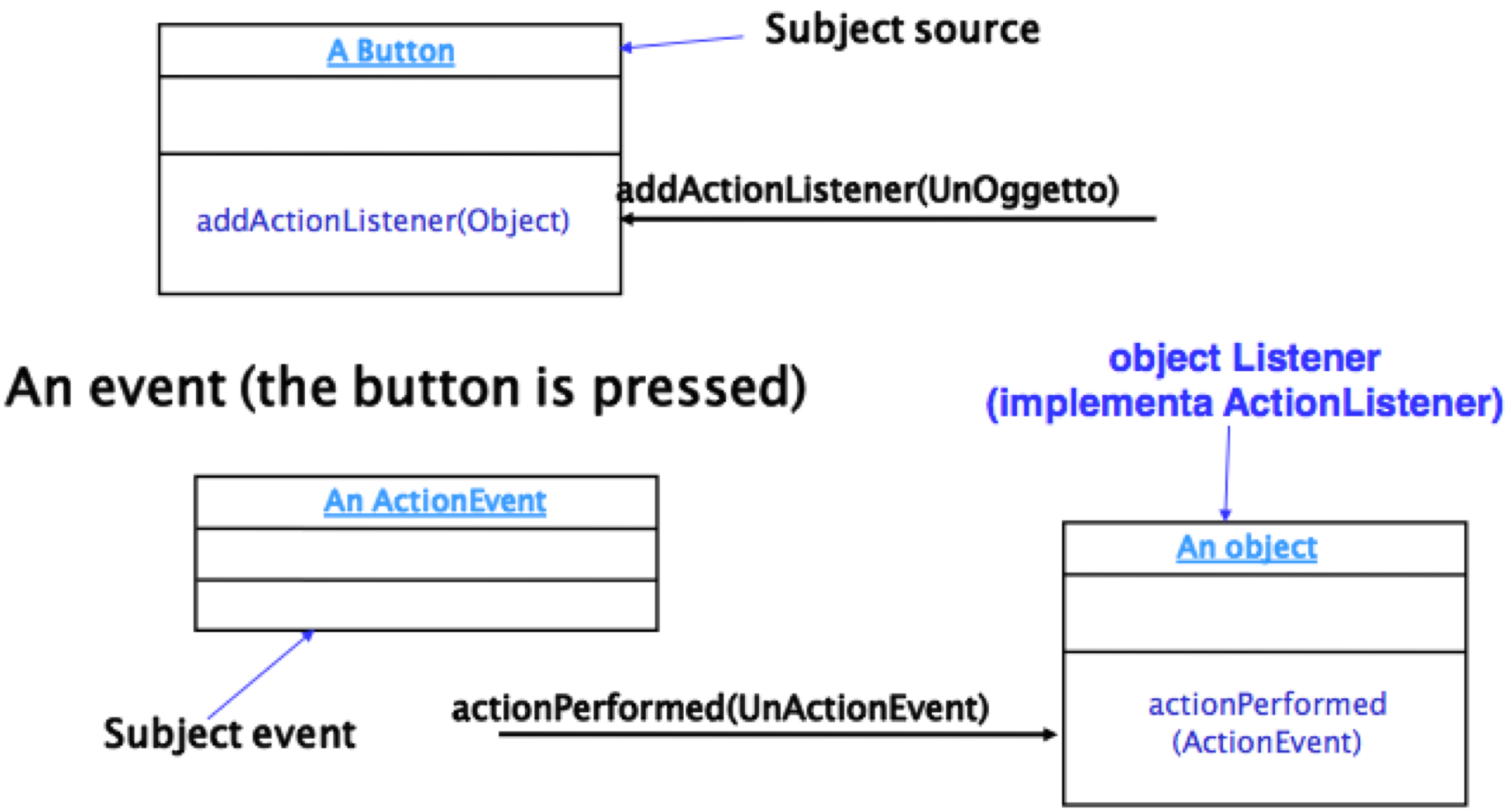
Event Delegation Model



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
actionPerformed
(ActionEvent)
Subject event
actionPerformed(UnActionEvent)

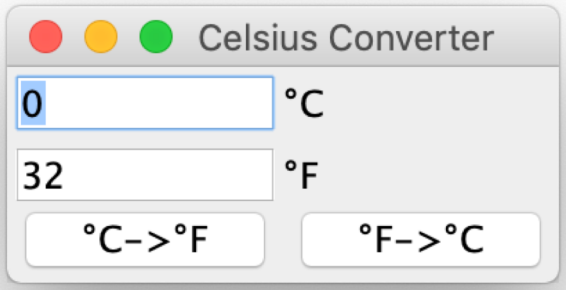
Event Delegation Model

- Events are organized by type
- Events need specific listener methods

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

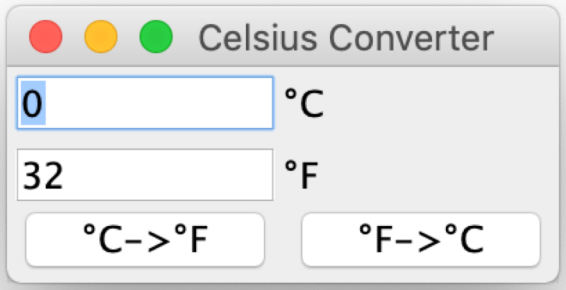
A complete example

```
public class CelsiusConverterFull extends JFrame implements ActionListener {
        private JButton CFButton;
        private JButton FCButton;
        private JTextField fahrenheitTF;
        private JTextField celsiusTF;

        public CelsiusConverterFull() {
                super("Celsius Converter");
                celsiusTF = new JTextField("0");
                fahrenheitTF = new JTextField("32");
                CFButton = new JButton("°C->°F");
                CFButton.addActionListener(this);
                FCButton = new JButton("°F->°C");
                FCButton.addActionListener(this);

                JPanel p1 = new JPanel(new GridLayout(2, 2));
                p1.add(celsiusTF); p1.add(new JLabel("°C"));
                p1.add(fahrenheitTF); p1.add(new JLabel("°F"));

                JPanel p2 = new JPanel(new GridLayout(1, 2));
                p2.add(CFButton); p2.add(FCButton);

                JPanel p3 = new JPanel(new BorderLayout());
                p3.add(p1, BorderLayout.NORTH);  p3.add(p2, BorderLayout.SOUTH);

                setContentPane(p3);
                setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
                setSize(200, 100);
                setVisible(true);
        }
```



A complete example

```
public void actionPerformed(ActionEvent e) {
        if (e.getSource() == CFButton) {
                int tempFahr = (int)((Double.parseDouble(celsiusTF.getText())) * 1.8 + 32);
                fahrenheitTF.setText(Integer.toString(tempFahr));
        }
        if (e.getSource() == FCButton) {
                int tempCelsius = (int)(((Double.parseDouble(fahrenheitTF.getText())) - 32) * 0.555);
                celsiusTF.setText(Integer.toString(tempCelsius));
        }
}

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



How to manage events in Java

- Events are managed similarly to exceptions:
  - sources of events (JButton, JTextField, etc..) select their receivers
  - Example: *button.addActionListener(receiver)*
  - receivers of events declare which events are able to deal with (one or more) by implementing the needed interfaces
- Pay attention! Receivers implement interfaces, so they must implement all methods of those interfaces (see interface MouseListener)!

ING

How to manage events in Java

- Generally, JComponents 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 sourceofEvents = new JComponent();
    sourceOfEvents.addListener(this);

    void method1OfTheInterfaceWithEvents() {...}
    void method2OfTheInterfaceWithEvents() {...}
    void methodnOfTheInterfaceWithEvents() {...}
}
```

Delegate events to external classes

```
class MyListener implements InterfaceWithEvents {
    void method1OfTheInterfaceWithEvents() {...}
    void method2OfTheInterfaceWithEvents() {...}
} //end class


class Frame extends JFrame {
    MyListener l = new MyListener();
    JComponent sourceofEvents = new JComponent();
    sourceOfEvents.addListener(l);
} //end class
```

Dealing with multiple sources

- Frequently, multiple components register to the same listener. For example, a group of buttons within the same JFrame. It is needed a mechanism for recognizing the actual source of each event.
- getSource() and object references
  - if (e.getSource() == buttonSelfDestruction) {}
- getActionCommand() and strings
  - If (e.getActionCommand() == “destroy”) {}
- Event classes
  - If (e instanceof(KeyEvent)) {}

Event Interfaces

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

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)