Programming of Interactive Systems

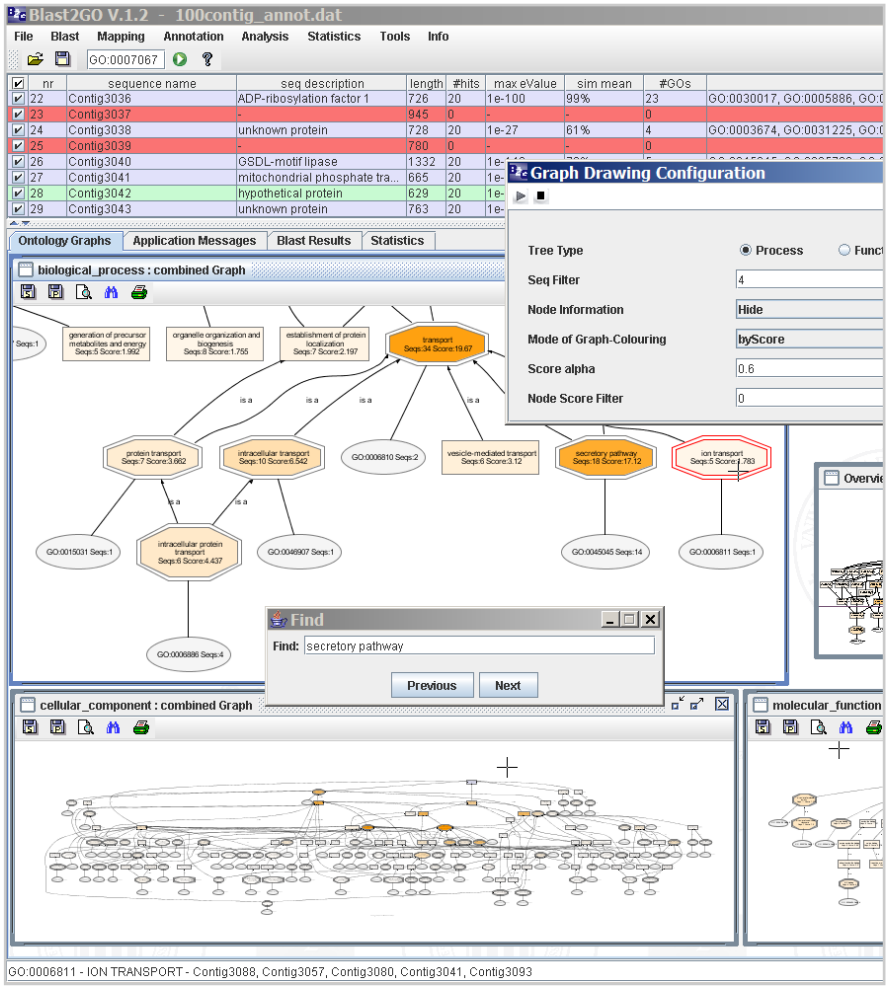
Anastasia.Bezerianos@lri.fr

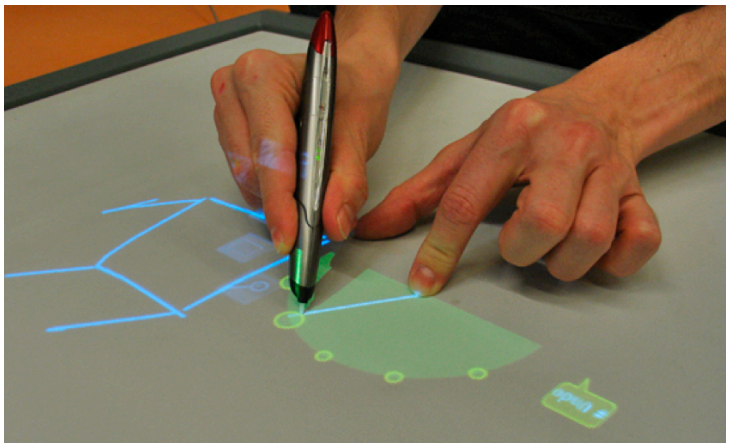
Week 2 :
a. Intro to Swing

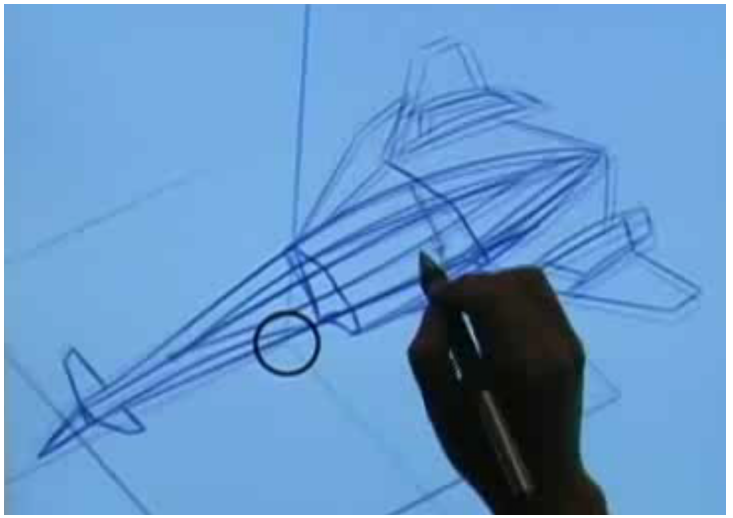
Anastasia.Bezerianos@lri.fr

(part of this class is based on previous classes from Anastasia,
and of T. Tsandilas, S. Huot, M. Beaudouin-Lafon, N.Roussel, O.Chapuis)

interactive systems



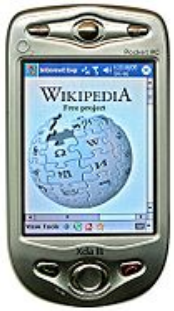




graphical interfaces

A **graphical user interface** or **GUI**, is an interface that allows users to interact with electronic devices through graphical icons and visual components (widgets)

GUIs: input events (and their result) are specified w.r.t. output (on what widget they act on)



events

Event: An object that represents a user's interaction with a GUI component; can be "handled" to create interactive components

Types of UI events:

- Mouse: move/drag/click button press/release,
  Keyboard: key press/release sometimes with
  modifiertd like shift/control/alt,
  Touchscreen: finger tap/drag,
  Window: resize/minimize/restore/close, etc.

interface toolkits

libraries of interactive objects (« widgets », e.g. buttons) that we use to construct interfaces

functions to help programming of GUIs

usually also handle input events

This week we focus on a specific toolkit, Swing.
Later in the class we will discuss toolkits more …

Swing

Swing toolkit (and AWT)

- Original Java GUI was the Abstract Window Toolkit (AWT), that was platform dependent.
- Swing was introduced in 1997 to fix the problems with AWT, with higher level components, with pluggable look and feel.
- Swing is built on AWT.

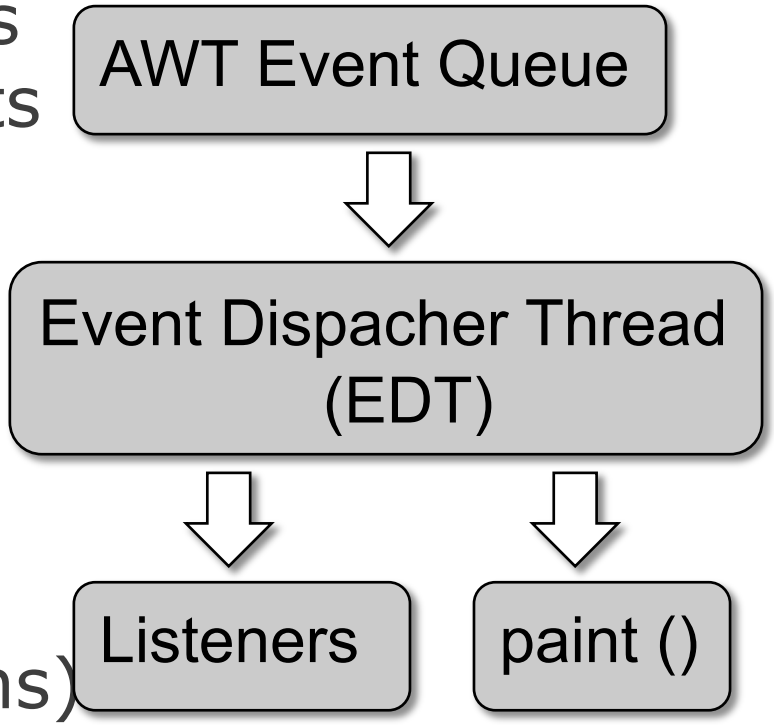
Swing toolkit (and AWT)

3 threads in JVM:

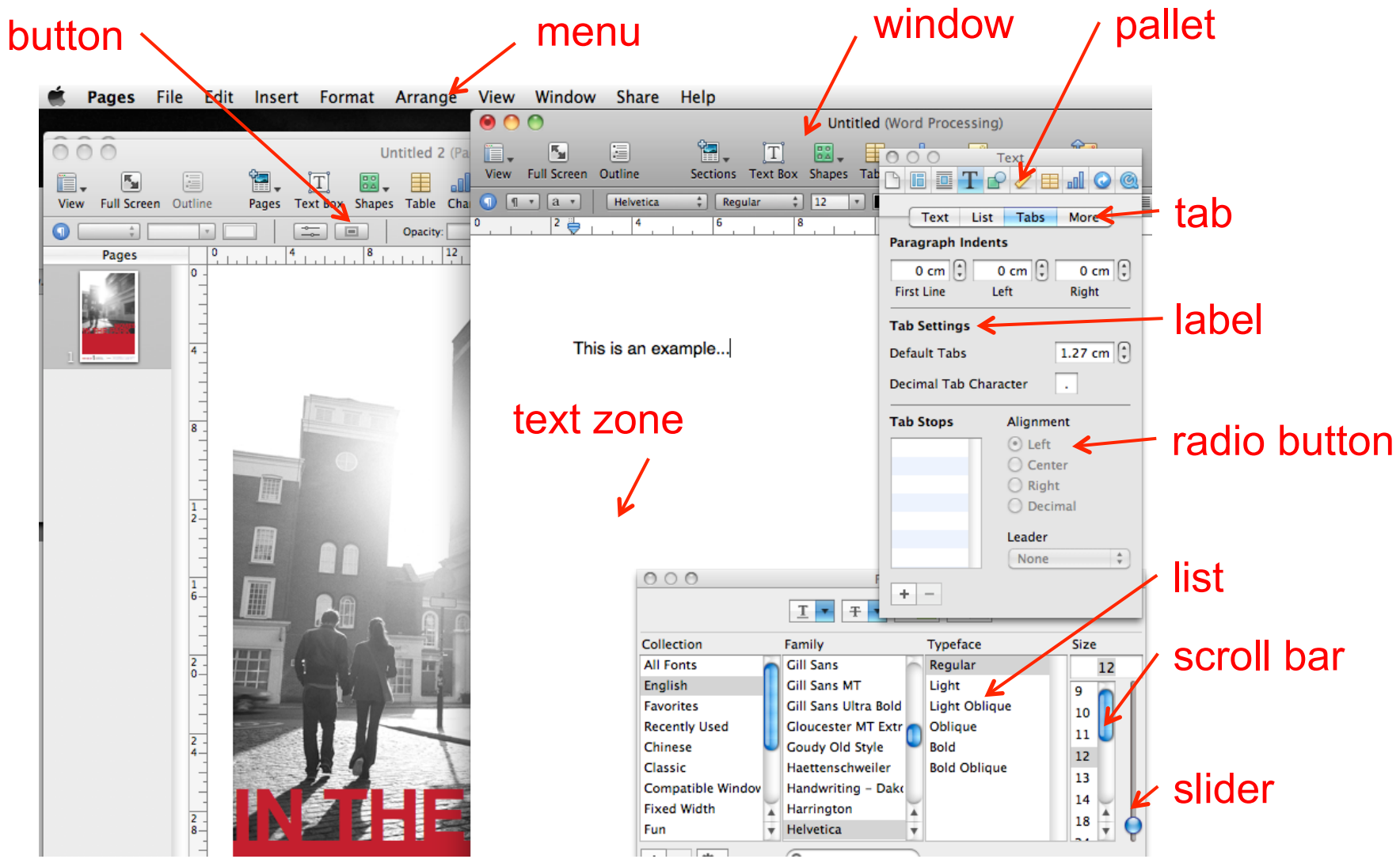
- main () : your program

- toolkit thread that receives (from OS) **events** and puts them in a queue

- EDT manages the queue: sends events to *listeners* (functions dealing with events) and calls paint methods (drawing functions)



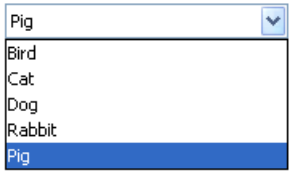
« widgets » (window gadget)



Swing widgets (atomic interactive)

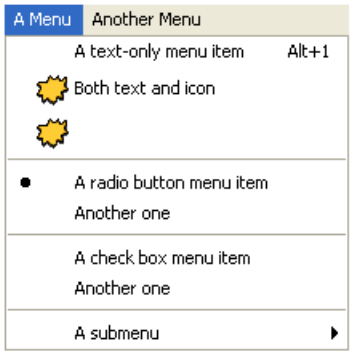
JButton

JCheckBox

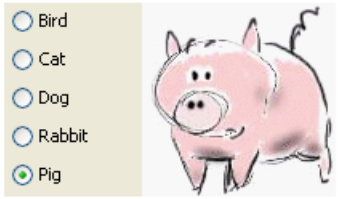


JComboBox

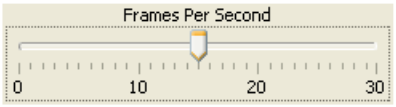
JList



JMenu



JRadioButton



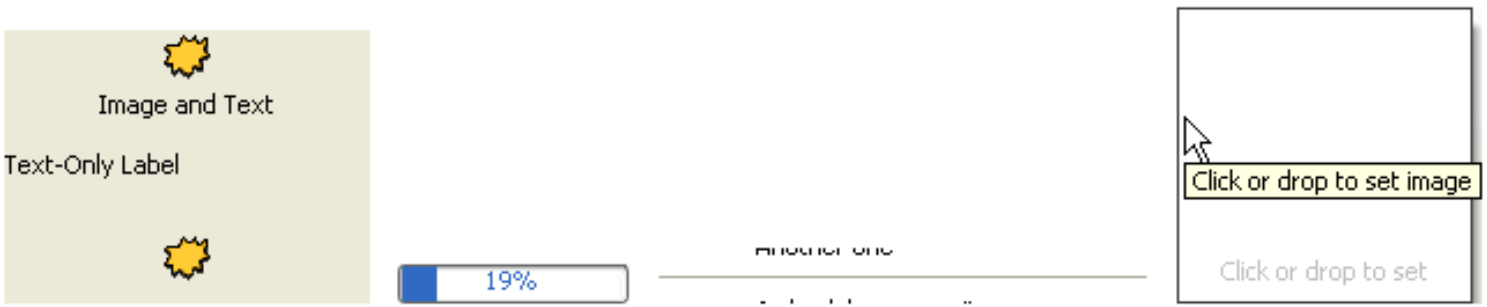
JSlider

JSpinner

JTextField

JPasswordField

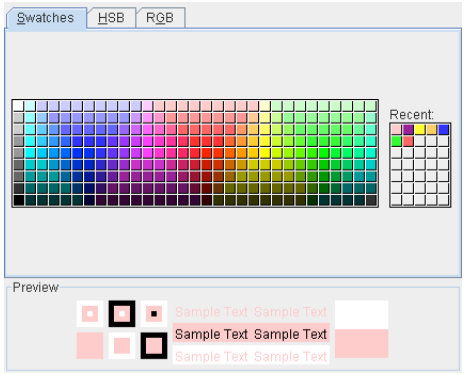
Swing widgets (non-editable)



JLabel JProgressBar JSeparator JToolTip

Swing widgets (more complex)

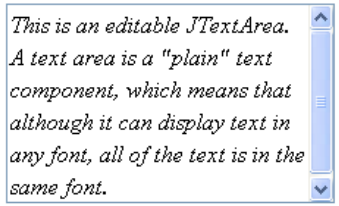
JFileChooser



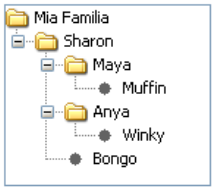
JColorChooser

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

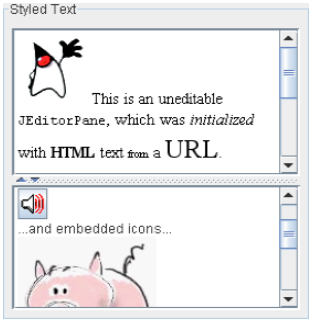
JTable



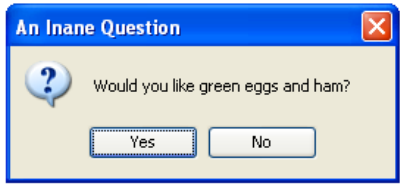
JTextArea



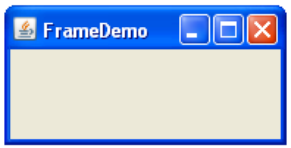
JTree



JEditorPane and JTextPane



JDialog



JFrame

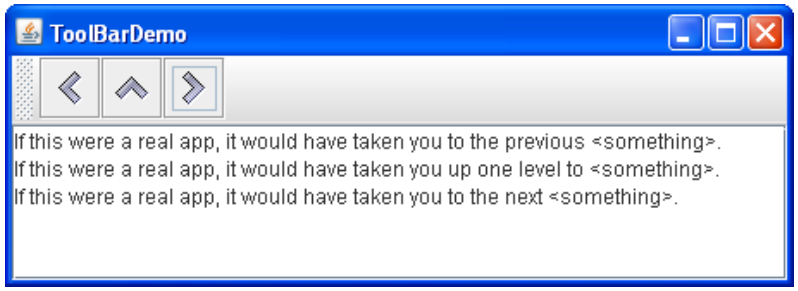
widget complexity

- Simple widgets
  - buttons, scroll bars, labels, …
- Composite/complex widgets
  - contain other widgets (simple or complex)
  - dialog boxes, menus, color pickers, …

widget tree

Hierarchical representation of the widget structure

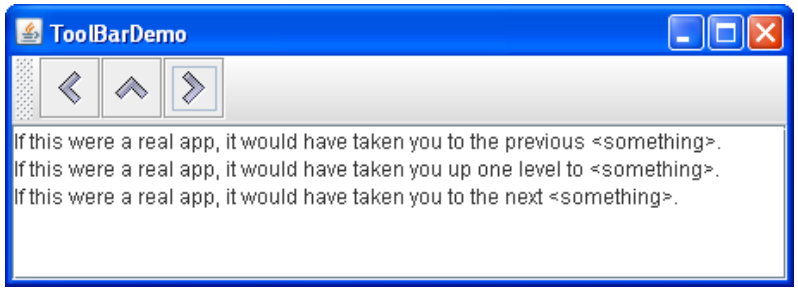
- a widget can belong to only one « container »



widget tree

Hierarchical representation of the widget structure

- a widget can belong to only one « container »

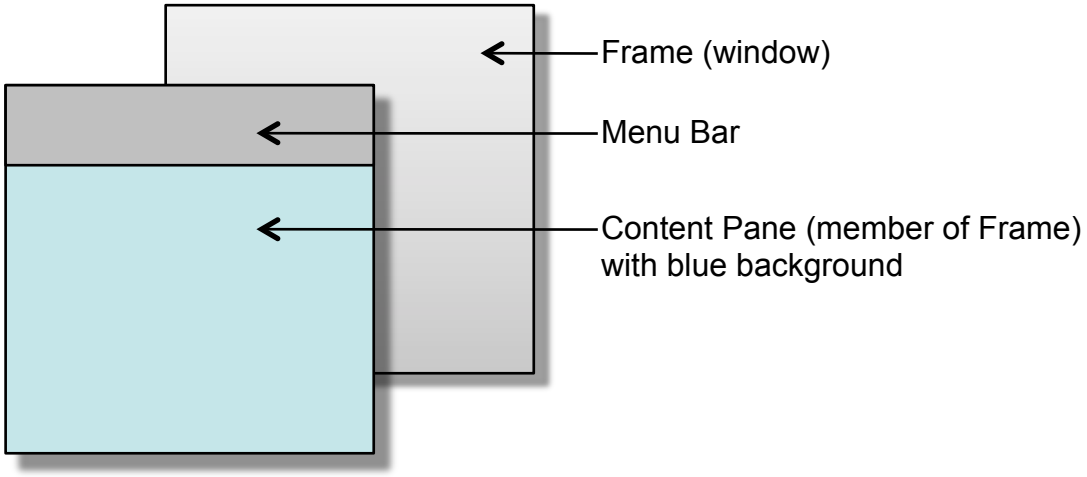


Swing widget classes

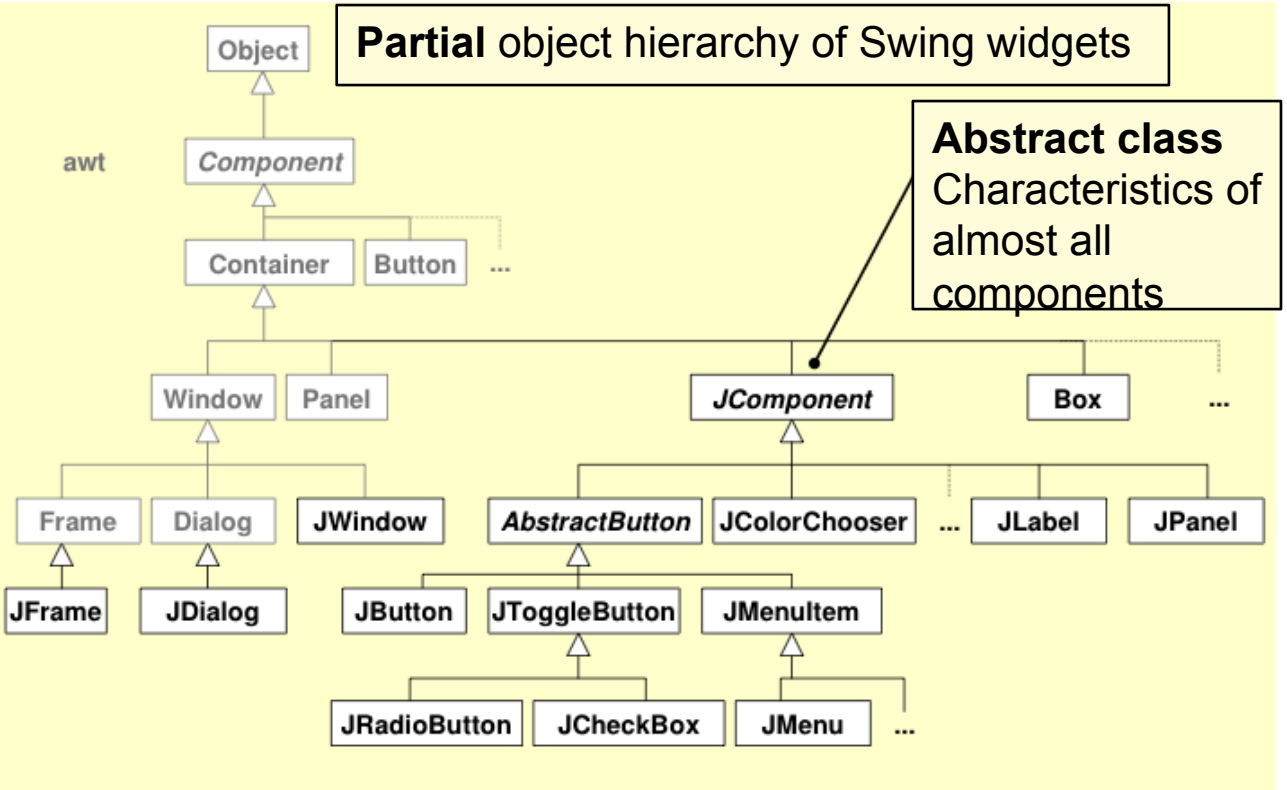
A GUI application has a top-level (container) widget that includes all others

In Swing there are 3 types: JFrame, JDialog and JApplet

They all contain other widgets (simple or complex), that are declared in the field **content pane**



Swing widget classes



http://docs.oracle.com/javase/tutorial/ui/features/components.html

AWT (older) is more connected to the graphics system. Later extended with Swing (less use of the graphics system).

Swing JFrame

a window with a basic bar

```
public static void main(String[] args) {
   JFrame jf = new JFrame("Tada!");
   jf.setVisible(true);
   jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    System.out.println("finished ? ! ?");
   System.out.println("no, still running …");
}
```

Useful functions

public JFrame();
public JFrame(String name);
public Container getContentPane();
public void setJMenuBar(JMenuBar menu);
public void setTitle(String title);
public void setIconImage(Image image);

This program does not terminate after "no, still running …"

Swing JDialog

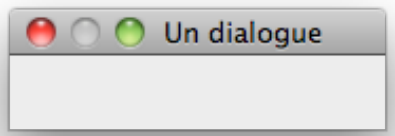
a message window (dialog) can be “modal” (blocks interaction)

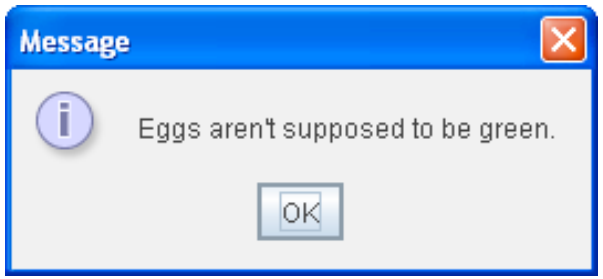
usually attached to another window
(when that closes, so does the dialog)

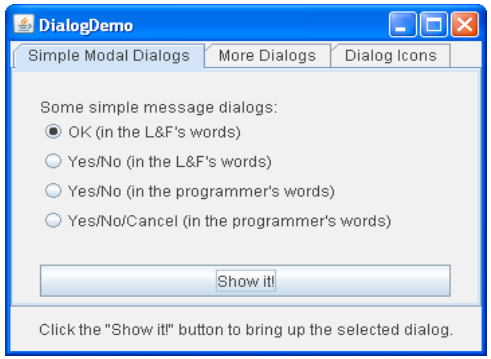
```
public static void main(String[] args) {
    JFrame jf = new JFrame("tada!");
    jf.setVisible(true);
    jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JDialog jd = new JDialog(jf,"A dialog",true);
    jd.setVisible(true);
}
```

modal

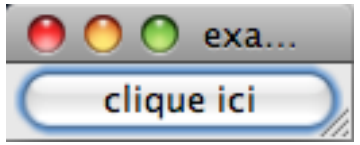
attached to







Swing layouts



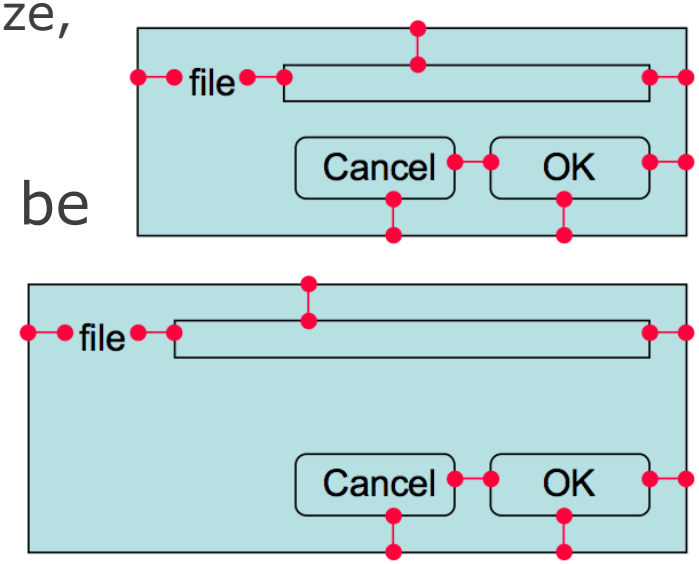

```
import javax.swing.*;

public class SwingDemo1 {

        public static void main(String[] args)
        {
                JFrame frame = new JFrame();

                frame.setTitle("example 1");

                frame.getContentPane().add(new JLabel("Swing Demo 1"));

                frame.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);

                frame.getContentPane().add(new JButton("clique ici"));

                frame.setSize(100,50);

                frame.setVisible(true);
        }
}
```

where is the label?

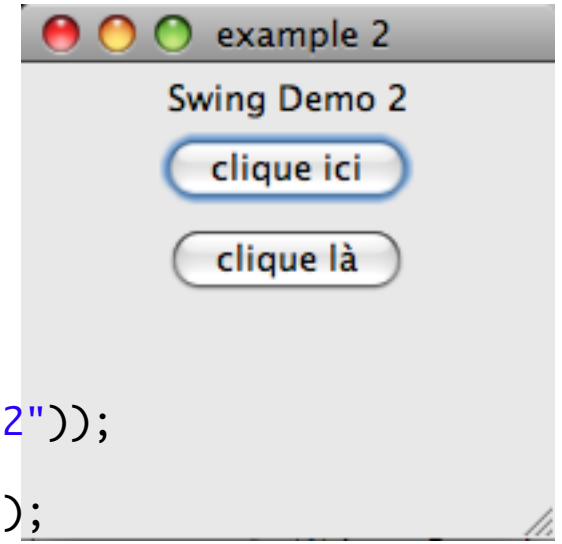
widget placement

UI toolkits control widget placement:

- should be independent of widget size
  (menu at least as big as its largest item,
  change of scrollbar size with document size,
  adjusting text flow)
- done in *layout managers* that can be added to container widgets



```
import javax.swing.*;
import java.awt.*;

public class SwingDemo2 extends JFrame {

        public void init()
        {
            this.setTitle("example 2");

            getContentPane().add(new JLabel("Swing Demo 2"));

            Container contentPane = this.getContentPane();
            contentPane.setLayout(new FlowLayout());

            this.setDefaultCloseOperation(EXIT_ON_CLOSE);

            contentPane.add(new JButton("clique ici"));
            contentPane.add(new JButton("clique là"));
        }

        public static void main(String[] args)
        {
                JFrame frame = new SwingDemo2();

                frame.init();

                frame.setSize(200,200);
                frame.setVisible(true);
        }
}
```



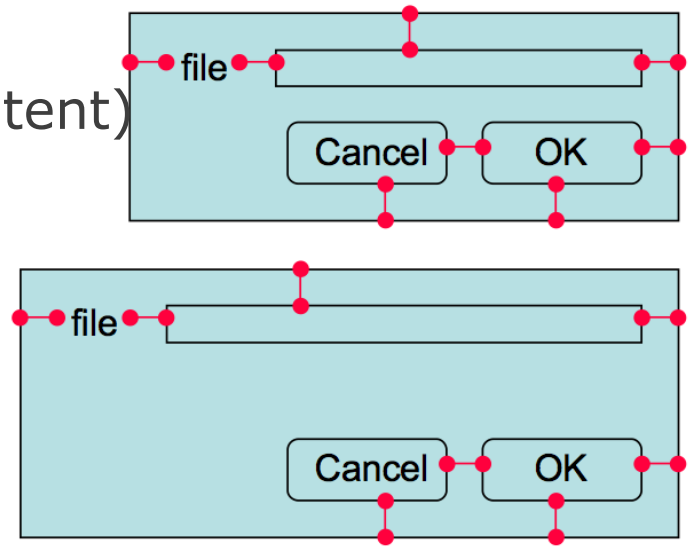
widget placement

general guides

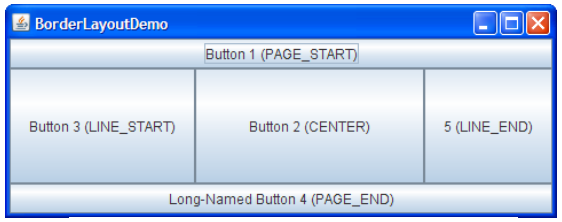
- embed geometry of a «child» widget to its parent
- parent controls the placement of its children

layout algorithm

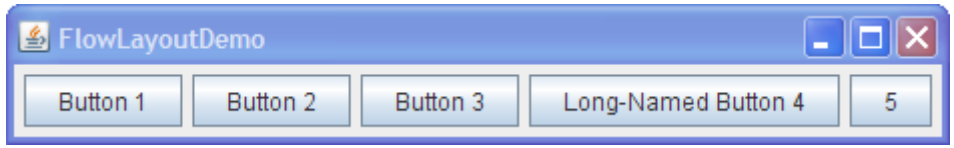
- natural size for each child (to fit content)
- size and position imposed by parent
- constraints: grid, form, etc.



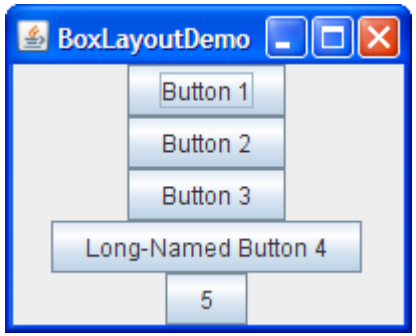
layout managers (in Swing)



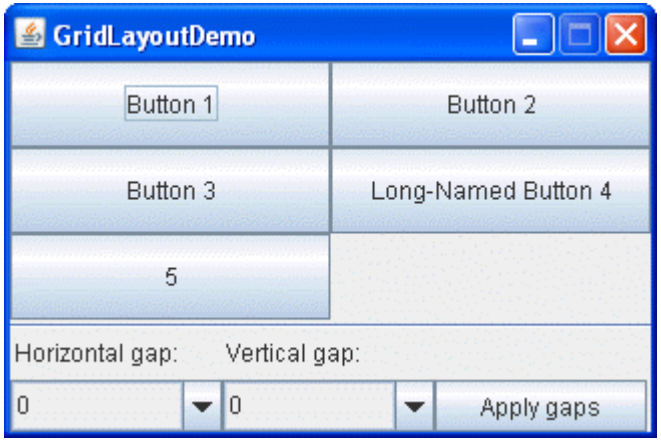
BorderLayout



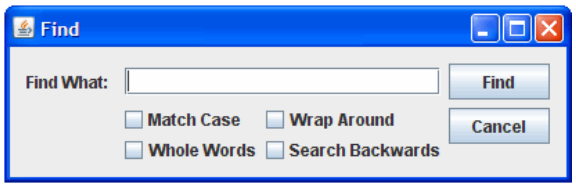
FlowLayout



BoxLayout



GridLayout



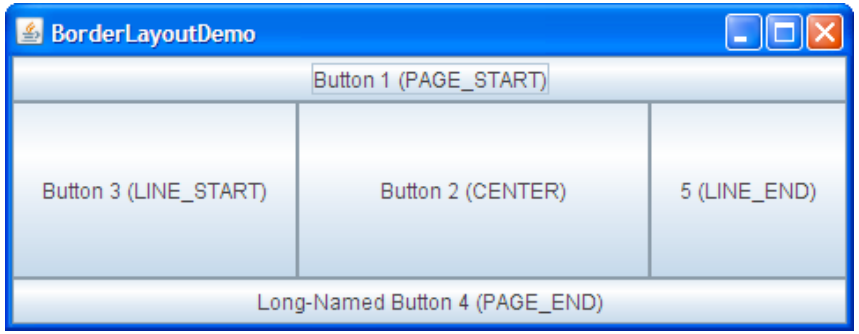
GroupLayout

http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html

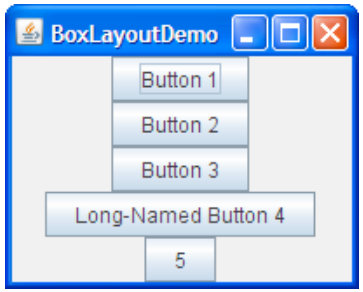
layout managers (in Swing)

BorderLayout : 5 containers:

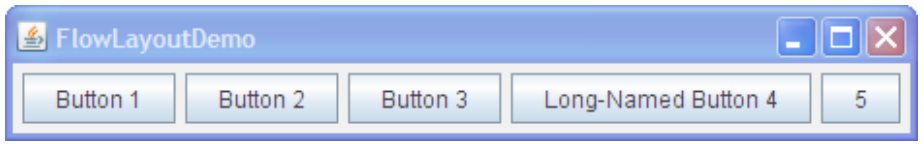
North, South, East, West & Center.



BoxLayout : line or column arrangement

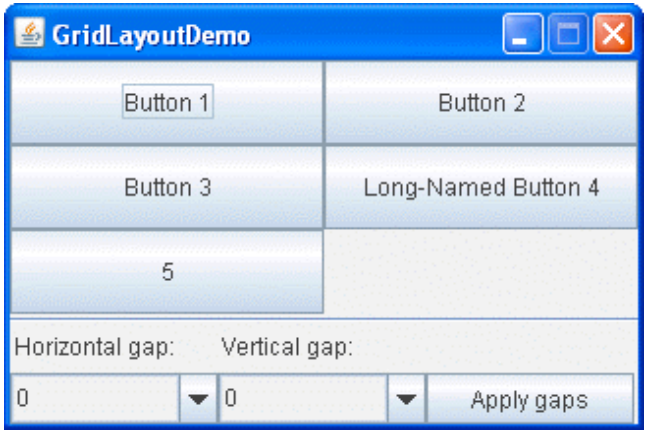


FlowLayout : (defautl) on a line and adaptable

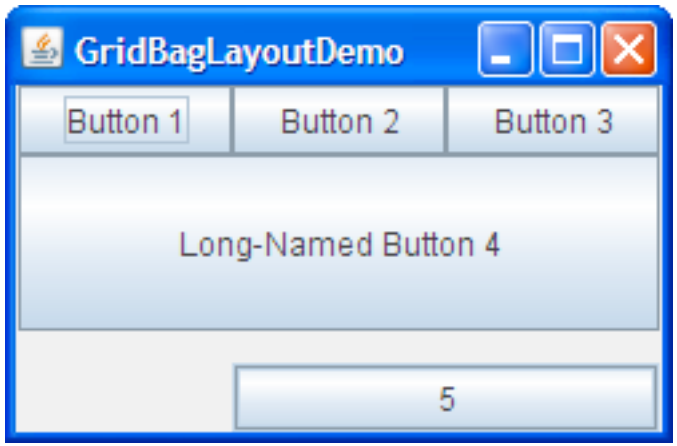


layout managers (in Swing)

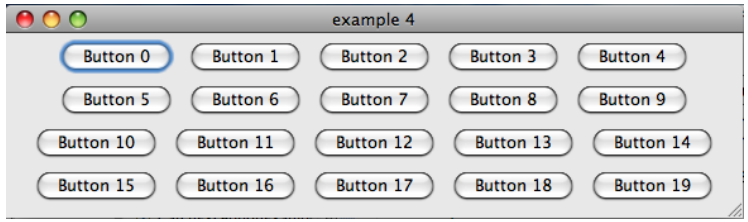
GridLayout : grid



GridBagLayout : sophisticated grid



```
import javax.swing.*;
import java.awt.*;

public class SwingDemo4 extends JFrame {

    public void init()
    {
       Container cp = getContentPane();

       this.setTitle("example 4");
       this.setDefaultCloseOperation(EXIT_ON_CLOSE);

       cp.setLayout(new FlowLayout());
       for(int i = 0; i < 20; i++)
          cp.add(new JButton("Button " + i));
    }

    public static void main(String[] args)
    {
       SwingDemo4 frame = new SwingDemo4();

       frame.init();

       frame.setSize(200,700);
       frame.setVisible(true);
    }
}
```



```
import javax.swing.*;
import java.awt.*;

public class SwingDemo5 extends JFrame {

     public void init() {
          Container cp = getContentPane();

          this.setTitle("example 5");
          this.setDefaultCloseOperation(EXIT_ON_CLOSE);

          cp.setLayout(new GridLayout(7,3));
          for(int i = 0; i < 20; i++)
               cp.add(new JButton("Button " + i));
     }

     public static void main(String[] args)
     {
               SwingDemo5 frame = new SwingDemo5();

               frame.init();

               frame.setSize(200,700);
               frame.setVisible(true);
     }
}
```

Inspiré de: Bruce Eckel, Thinking in Java, 2e édition

Swing: layout example

Based on slides by David Bonnet, Cédric Fleury and Arnaud Prouzeau

building the interface

Always start by **laying out** the widgets in the window

- Handle the functionality (discussed next) with the event listeners **after**.

Use **JPanels** to **structure** and **sub-divide** the layout.

Assign LayoutManagers to JPanels to define a specific layout.

building the interface

Example of structure and resulting code:

```
Window
JPanel
   JLabel "A"
   JTextField
JPanel
   JLabel "B"
   JTextField
```

Structure

```
Container panel = getContentPane();

JPanel panelA = new JPanel();
panel.add(panelA);
panelA.add(new JLabel("A"));
panelA.add(new JTextField(5));

JPanel panelB = new JPanel();
panel.add(panelB);
panelB.add(new JLabel("B"));
panelB.add(new JTextField(5));
```

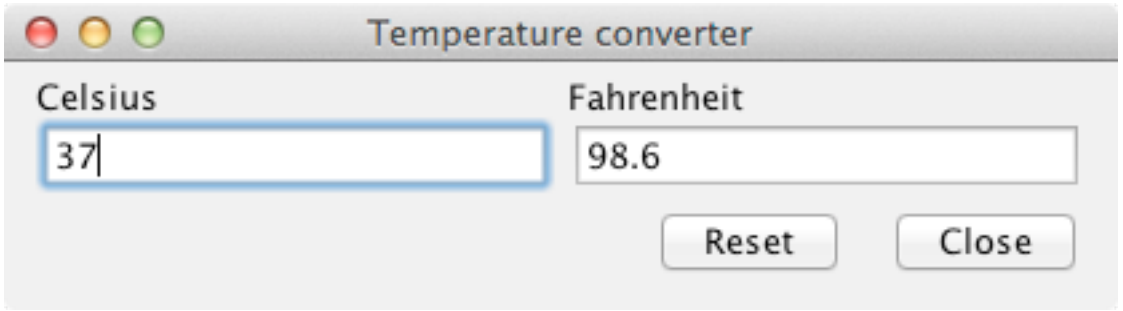
Code

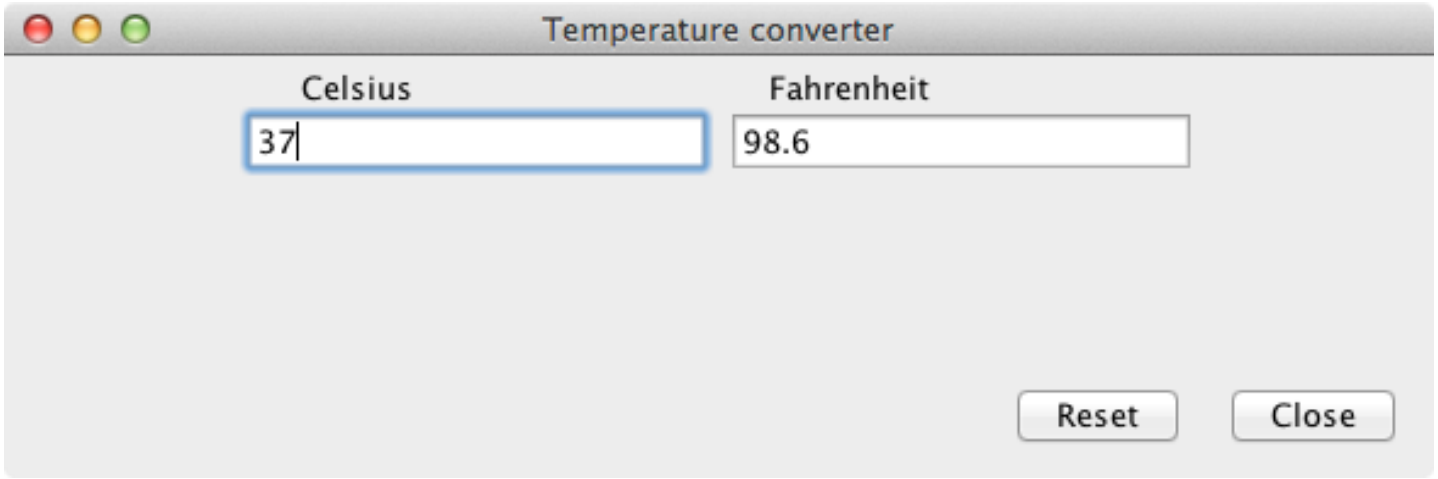
In-class exercise

In-class assignment

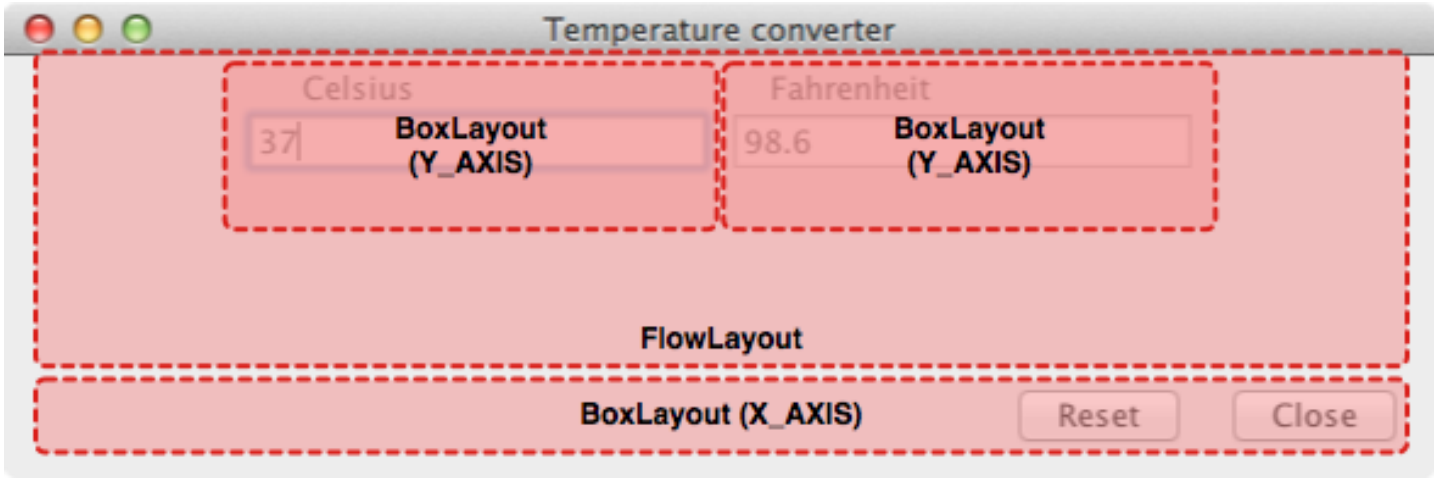
Go to our website:
http://www.lri.fr/~anab/teaching/HCID-ProgIS-2017/

and look at TA 1 parts 1-3, we will work on these together step by step



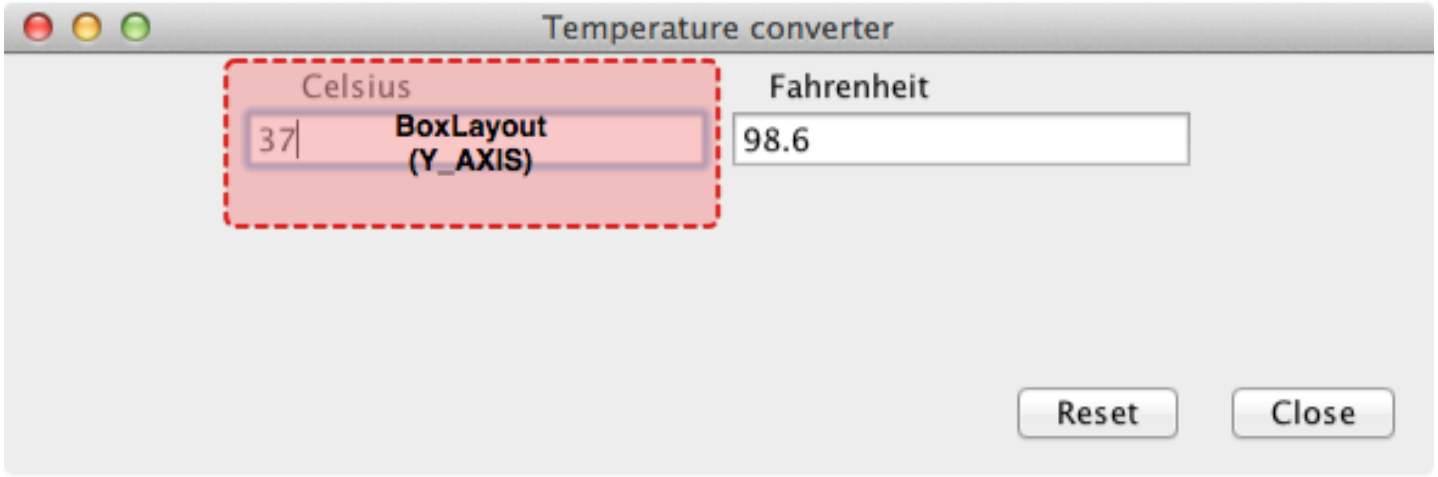


Before writing any code for the layout, **identify a structure** that sub-divides nicely into rectangular areas.

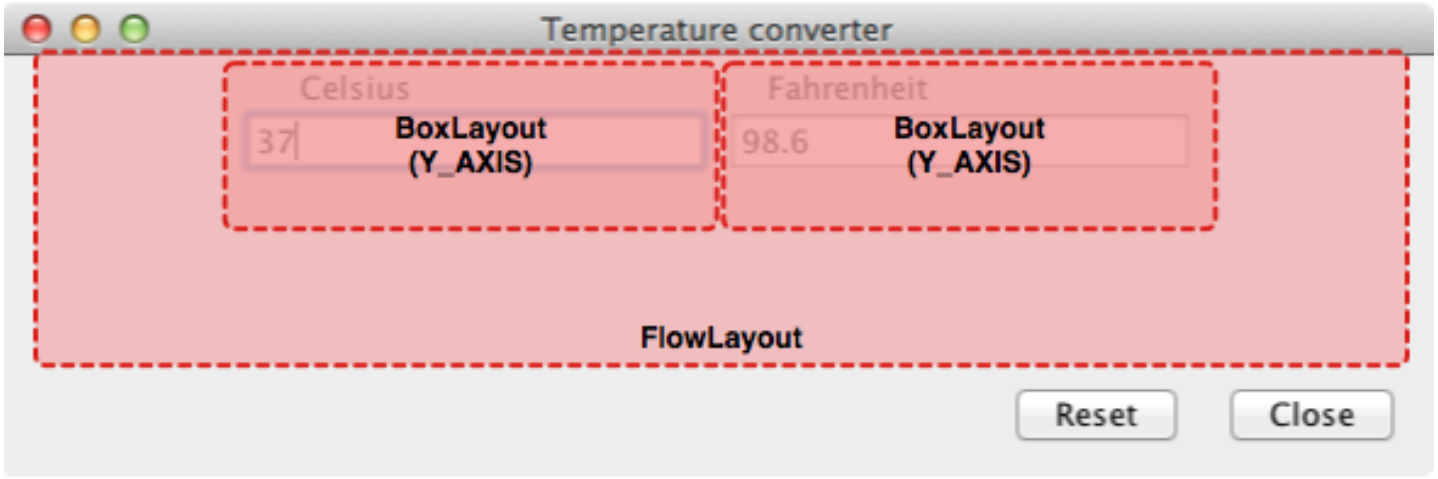


In this example, we have sub-divided the layout into different JPanels and chosen a specific layout (BoxLayout and FlowLayout) for each of them.

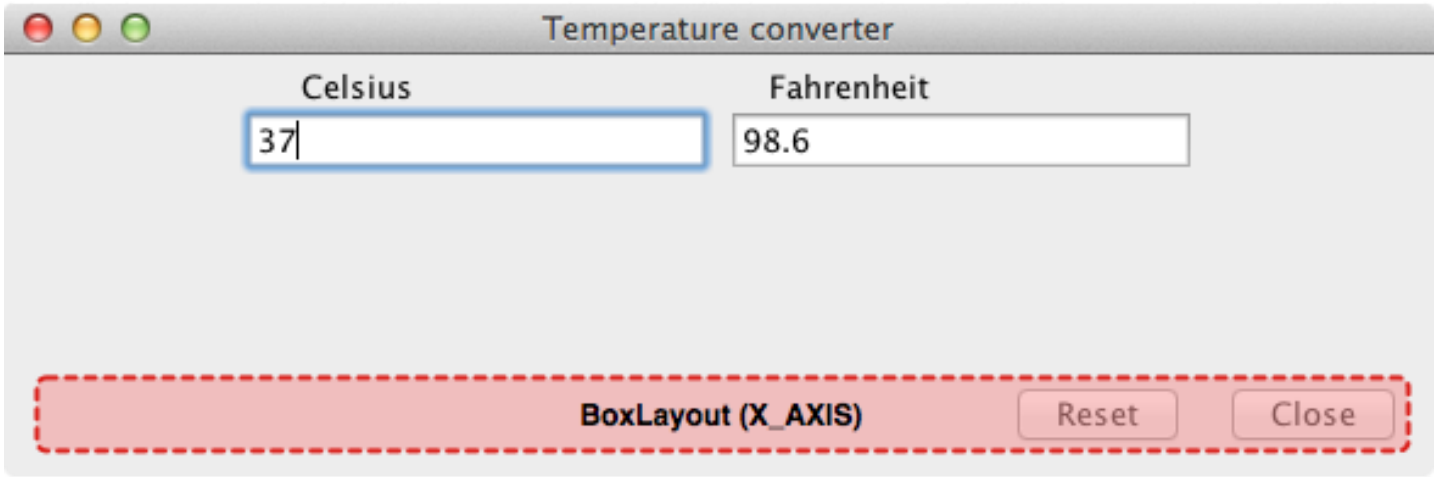
Note that there is not a unique solution.



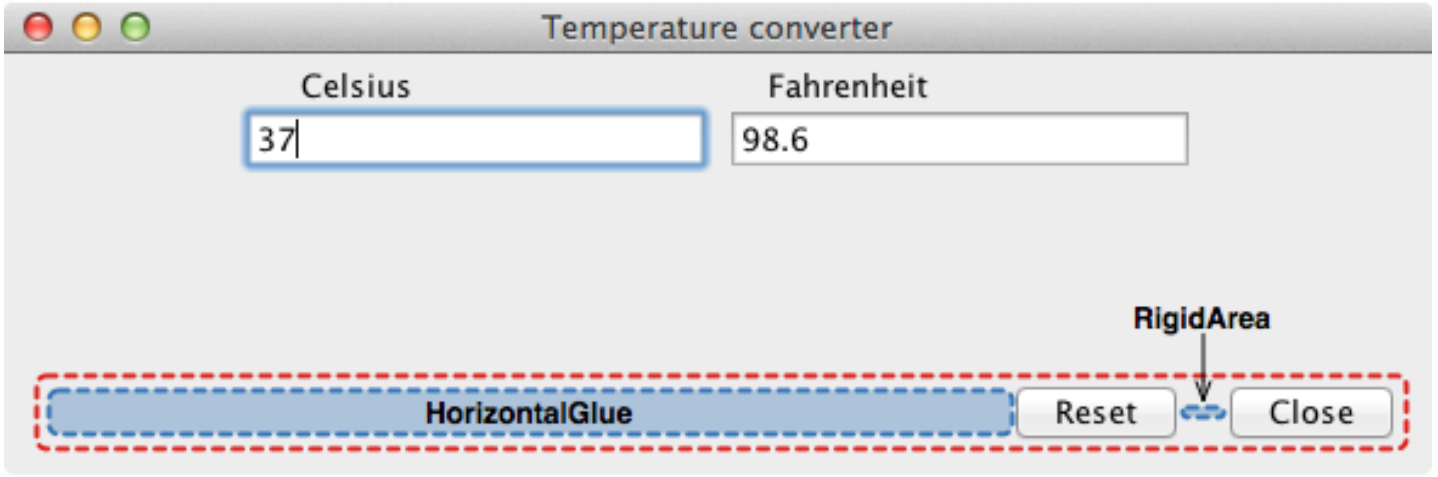

```
JPanel paneC = new JPanel();
paneC.setLayout(new BoxLayout(paneC, BoxLayout.Y_AXIS));
paneC.add(labelC);
paneC.add(textFieldC);
```



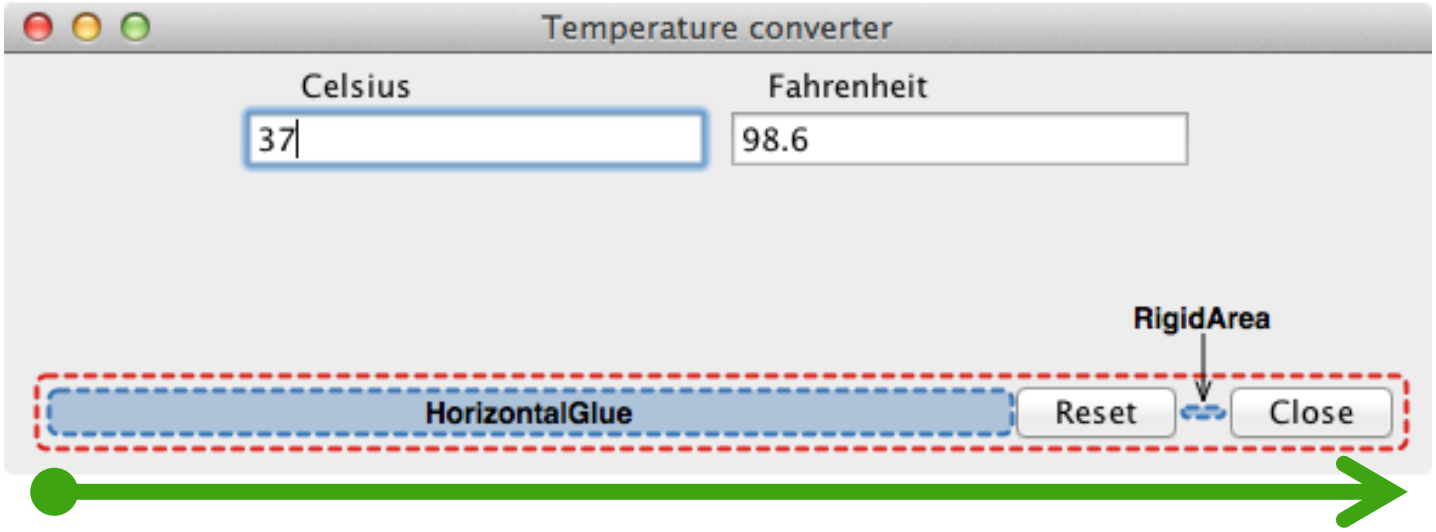

```
JPanel tempPane = new JPanel();
tempPane.add(paneC);
tempPane.add(paneF);
```




```
JPanel buttonPane = new JPanel();
buttonPane.setLayout(new BoxLayout(buttonPane, BoxLayout.X_AXIS));
buttonPane.setBorder(BorderFactory.createEmptyBorder(5, 10, 10, 10));
buttonPane.add(Box.createHorizontalGlue());
buttonPane.add(buttonReset);
buttonPane.add(Box.createRigidArea(new Dimension(10, 0)));
buttonPane.add(buttonClose);
```

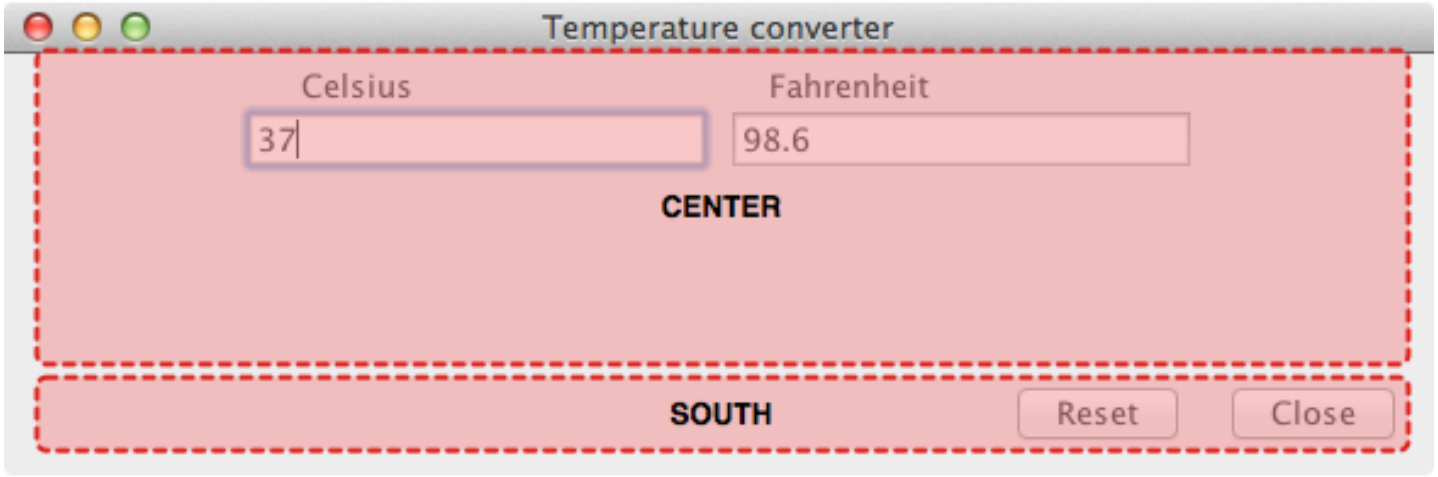



```
JPanel buttonPane = new JPanel();
buttonPane.setLayout(new BoxLayout(buttonPane, BoxLayout.X_AXIS));
buttonPane.setBorder(BorderFactory.createEmptyBorder(5, 10, 10, 10));
buttonPane.add(Box.createHorizontalGlue());
buttonPane.add(buttonReset);
buttonPane.add(Box.createRigidArea(new Dimension(10, 0)));
buttonPane.add(buttonClose);
```




```
JPanel buttonPane = new JPanel();
buttonPane.setLayout(new BoxLayout(buttonPane, BoxLayout.X_AXIS));
buttonPane.setBorder(BorderFactory.createEmptyBorder(5, 10, 10, 10));
buttonPane.add(Box.createHorizontalGlue());
buttonPane.add(buttonReset);
buttonPane.add(Box.createRigidArea(new Dimension(10, 0)));
buttonPane.add(buttonClose);
```

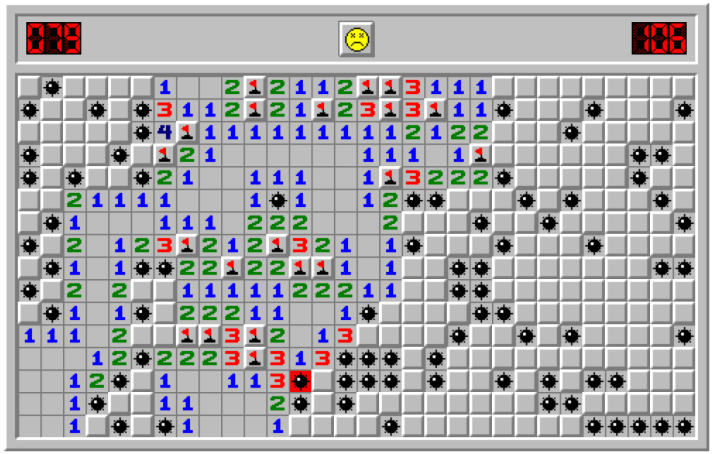
Insertion order is important: items are added from **left to right** for horizontal layouts and **top to bottom** for vertical layouts.




```
Container mainPane = getContentPane();
mainPane.setLayout(new BorderLayout());
mainPane.add(tempPane, BorderLayout.CENTER);
mainPane.add(buttonPane, BorderLayout.SOUTH);
```

In-class assignment

Consider your minesweeper,
how would you layout your widgets?



Swing: treating events

event listeners (Java)

Event: A msg (in Java an object) that represents a user's interaction with a GUI component; can be "handled" to create interactive components

Listener: An object that waits for events and responds to them.

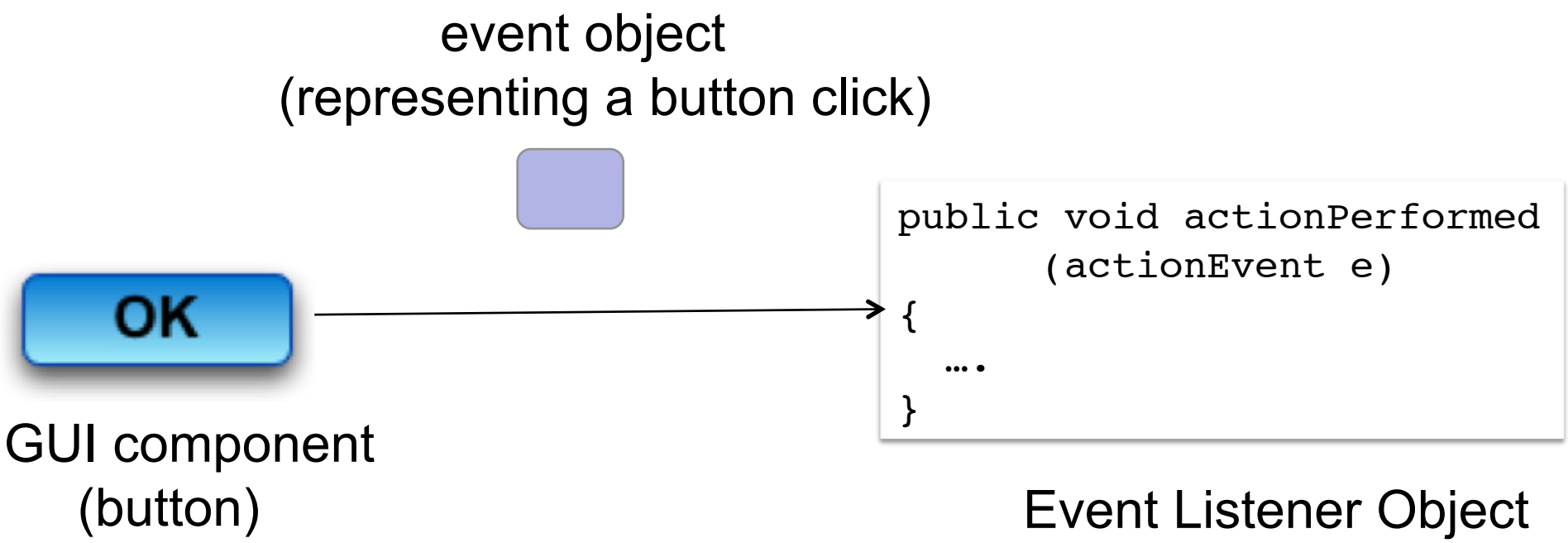
- To handle an event, attach a listener to a component.
- The listener will be notified when the event occurs (e.g. button click)

event listeners (Java)

methods of type `AddListener` create the **listener** object that treats events

when a widget changes state, it triggers a predefined method of the **listener** object (e.g. `actionPerformed, mouseClicked`)

event listeners (Java)



A GUI component sending an event to its registered listener

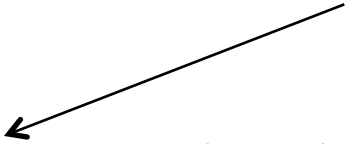
event listeners (Java)

A listener can be:

- a separate object that treats events
- the same object as your window
- an object inside your main window

event listeners (Java)

a Listener implements
a Java interface



```
public class ClickListener implements ActionListener
{
    public void actionPerformed(ActionEvent e){
            JButton button = (JButton)e.getSource();
            …
    }
}

…
ClickListener listener = new ClickListener();
JButton button = new JButton(''Click me'');
button.addActionListener(listener);
…
```

(aside) Java Interface

A Java interface is:
a group of related methods with empty bodies

When we *implement* one (or more interfaces)
we need to provide code for its methods

event listeners (Java)

A listener can be:

- a separate object that treats events
- the same object as your window
- an object inside your main window

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


public class SwingDemoEvent0 extends JFrame implements ActionListener{

        public void actionPerformed(ActionEvent event) {
                System.exit(0);
        }

        public final void init() {
                JButton quitButton = new JButton("Quit");
                quitButton.addActionListener(this);
                getContentPane().add(quitButton);
        }

        public static void main(String[] args) {
                SwingDemoEvent0 frame = new SwingDemoEvent0();
                frame.init();
                frame.setTitle("Quit button");
                frame.setSize(100, 100);
                frame.setVisible(true);
        }
}
```

events and listeners (Java)

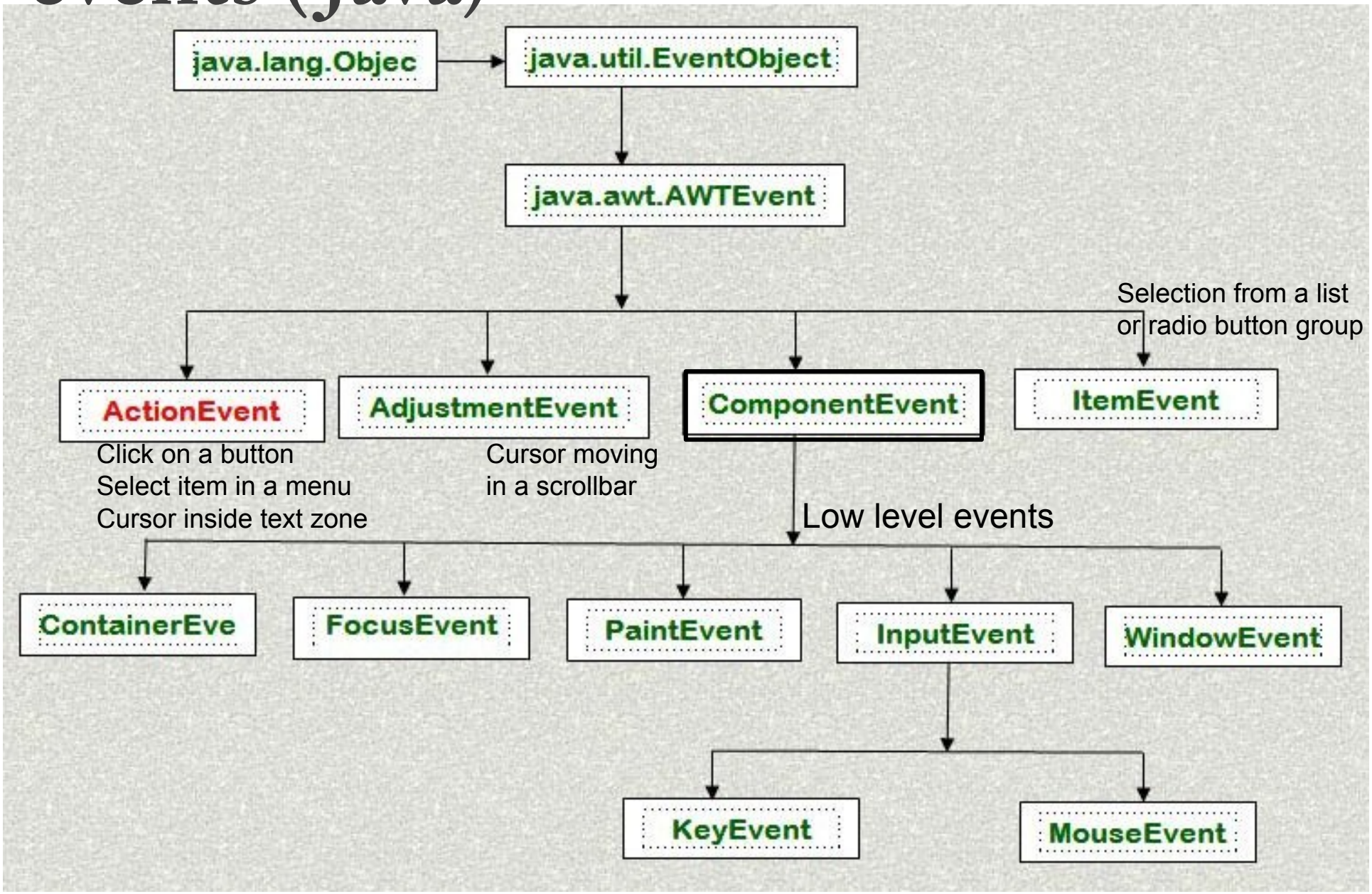
Each event has a source (e.g. JButton, JRadioButton, JCheckBox, JToggleButton,JMenu, JRadioButtonMenuItem, JTextField)

Can get it with the function **getSource()**

Listeners implement the interface that corresponds to their specific event
e.g. ActionEvent => ActionListener :

```
public interface ActionListener extends EventListener {
    /** Invoked when an action occurs.*/
    public void actionPerformed(ActionEvent e)
}
```

events (Java)


java.lang.Objec
java.util.EventObject
java.awt.AWTEvent
Selection from a list
or radio button group
ActionEvent
AdjustmentEvent
ComponentEvent
ItemEvent
Click on a button
Select item in a menu
Cursor inside text zone
Cursor moving
in a scrollbar
Low level events
ContainerEve
FocusEvent
PaintEvent
InputEvent
WindowEvent
KeyEvent
MouseEvent

events and listeners (Java)

all events inherit from the class EventObject

all listeners correspond to an interface that inherits from EventListener

a class receiving notification events of some type needs to implement the corresponding interface:

ActionEvent	ActionListener
MouseEvent	MouseListener
KeyEvent	KeyListener
...	

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;

public class SwingDemoEvent1 extends JFrame implements ComponentListener {
    JTextArea display;

    public void init() {
          this.setTitle("example event 1");
          this.setDefaultCloseOperation(EXIT_ON_CLOSE);
          Container cp = getContentPane();
          cp.setLayout(new FlowLayout());
          display = new JTextArea();
          display.setEditable(false);
          cp.add(display);
          this.addComponentListener(this);
    }

    protected void displayMessage(String message) {
          display.append(message + "\n");
          display.setCaretPosition(display.getDocument().getLength())
    }

   // ComponentListener methods
   public void componentHidden(ComponentEvent e) {
          displayMessage(e.getComponent().getClass().getName() + " --- Hidden")
   }
   public void componentMoved(ComponentEvent e) { ...
   public void componentResized(ComponentEvent e) { ...
   public void componentShown(ComponentEvent e) { ...

   public static void main(String[] args) {
          SwingDemoEvent1 frame = new SwingDemoEvent1();
          frame.init();
          frame.setSize(250,700);
          frame.setVisible(true);
   }
}
```

events and listeners (Java)

listeners need to be registered (added) to widgets

a listener can be added to multiple widgets

- e.g. one listener handles events from multiple buttons

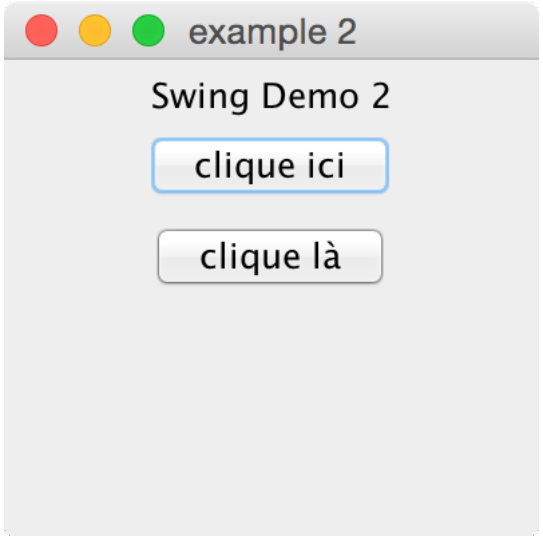
a widget can have many listeners

- e.g. one for “click” events and for “enter inside” button events

event listeners (Java)

A listener can be:

- a separate object that treats events
- the same object as your window
- an object inside another




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

public class SwingDemo2 extends JFrame {

    JButton b1 = new JButton("Clique ici");
    JButton b2 = new JButton("Clique la");
    JTextField txt = new JTextField(10);

    class ButtonListener implements ActionListener // INNER CLASS DEF.
    {
            public void actionPerformed(ActionEvent e) {
                        String name = ((JButton)e.getSource()).getText();
                        txt.setText(name);
            }
    } // END OF INNER CLASS DEFINITION

    ButtonListener bl = new ButtonListener();


    public void init() {

            b1.addActionListener(bl);
            b2.addActionListener(bl);

            Container cp = this.getContentPane();

            this.setTitle("example 2");

            cp.add(new JLabel("Swing Demo 2"));
            cp.setLayout(new FlowLayout());

            cp.add(b1);
            cp.add(b2);
            cp.add(txt);
   }
```

inner class

```
    public static void main(String[] args)
    {
                SwingDemo2 frame = new SwingDemo2();

                frame.init();

                frame.setSize(200,200);
                frame.setVisible(true);
    }

} // end of SwingDemo2 class definition
```

event listeners (Java)

Anonymous Inner classes

“new <class-name> () { <body> }”

this construction does 2 things:

- creates a new class without name, that is a subclass of <class-name> defined by <body>
- creates a (unique) instance of this new class and returns its value

this (inner) class has access to variables and methods of the class inside which it is defined

event listeners (Java)

Anonymous Inner classes

```
…
button.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                  …
            }
});
…
panel.addMouseListener(new MouseAdapter(){
            public void mouseClicked(MouseEvent e){
                  …
            }
});
```

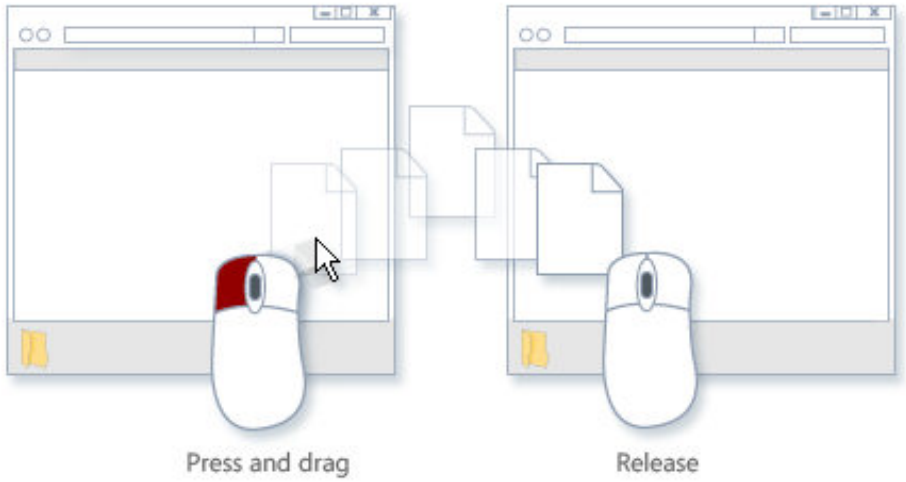
The functions and events are predefined

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

public class SwingDemo3 extends JFrame {

    JButton b1 = new JButton("Clique ici");
    JTextField txt = new JTextField(10);


      public void init() {

              b1.addActionListener(new ActionListener() {
                 public void actionPerformed(ActionEvent e) {
                     txt.setText( "Button clicked"); }
              });

              Container cp = this.getContentPane();

              this.setTitle("example 3");

              cp.add(new JLabel("Swing Demo 3"));
              cp.setLayout(new FlowLayout());

              cp.add(b1);
              cp.add(txt);
     }


     public static void main(String[] args)
     {
                 SwingDemo3 frame = new SwingDemo3();

                 frame.init();

                 frame.setSize(200,200);
                 frame.setVisible(true);
     }

} // end of SwingDemo3 class definition
```

anonymous inner class

« drag-and-drop » to think about

What are the affected « widgets »?
What are the events?



How to describe this interaction with a « event listener » ?