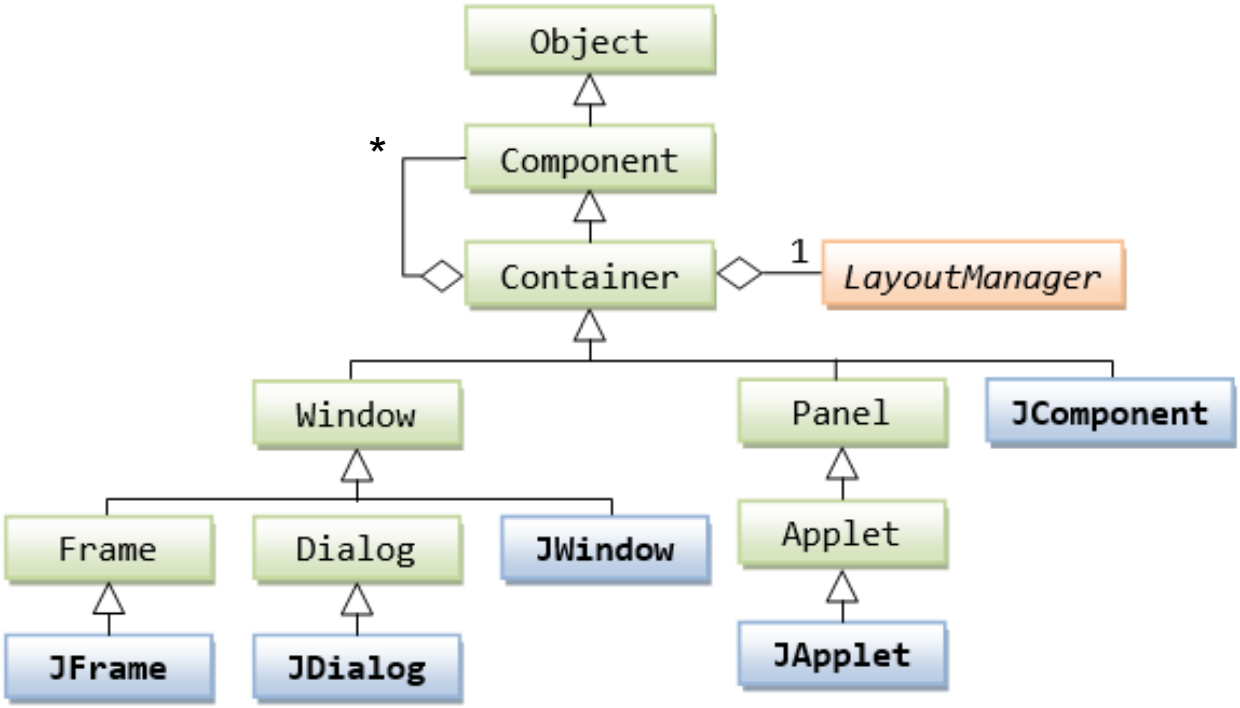
EECS 1720
Building Interactive Systems

Lecture 13/14 :: Graphical User Interfaces (GUI) - 2

Windows, Components & Layouts

YORK UNIVERSITÉ UNIVERSITY

Simplified class hierarchy (Swing Parents):

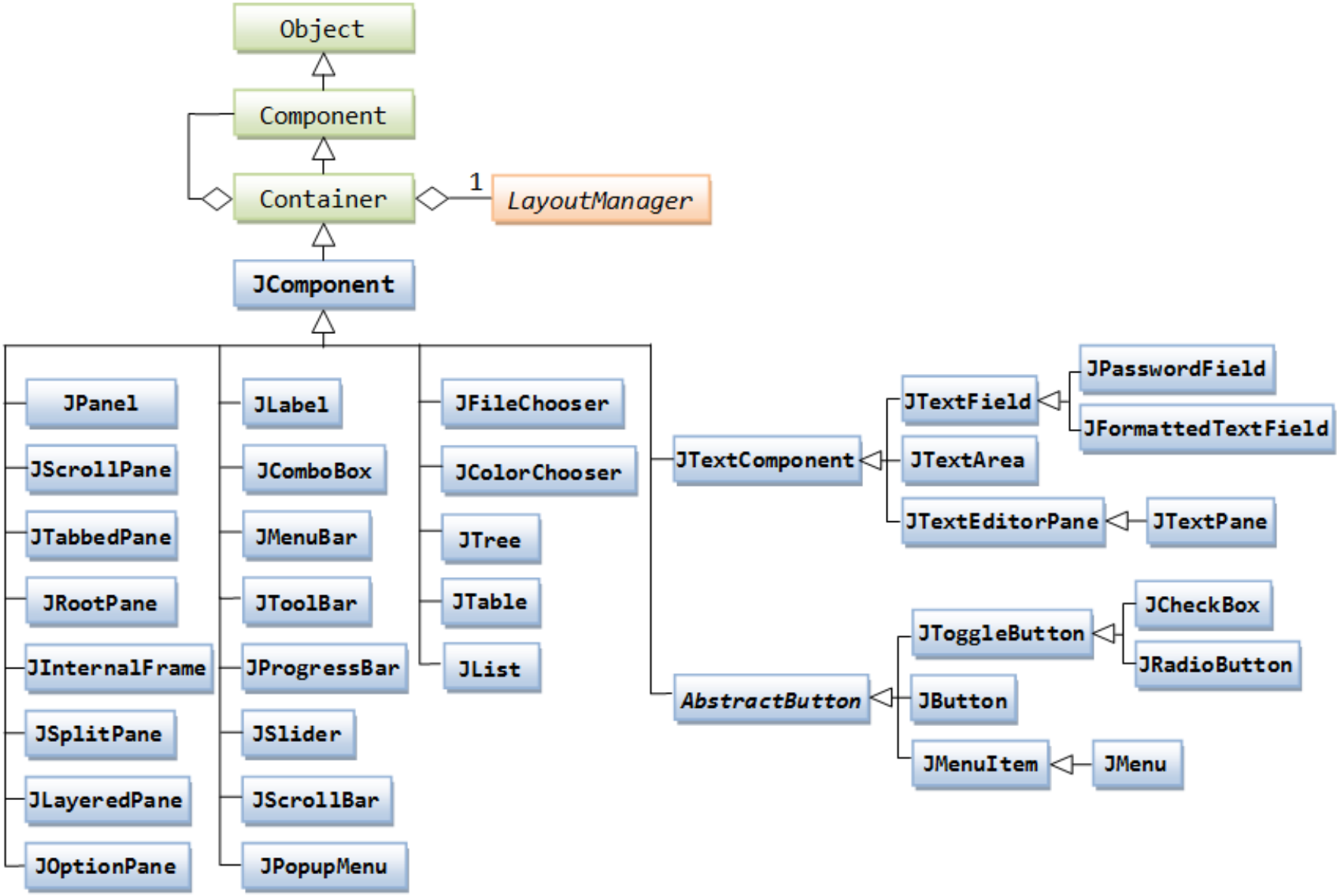


Container **is-a** Component
Container **has-a** LayoutManager
Container **has-a** Component (in fact can have many)
JFrame, JDialog, JComponent **is-a** Container

- JFrame
  - Contains title bar, window management buttons & special areas (panes) to hold static and interactive graphical components/content
    - JRootPane (manages a content pane)
    - JDialog is a specialized type of window (popup to show message or prompt for some input/confirmation)
    - JWindow is a general version of JFrame – without window typical dressings/management

- Organizing content (panes/panels)
  - JFrame
    - holds a set of containers
    - one of these is containers is called “contentPane”
      - holds other JComponents
        - JPanels, JButtons, JScrollPane, JTabbedPane, JSplitPane, … etc.
        - All swing classes are containers (can hold other JComponents), which can be organized spatially with or without a LayoutManager (which can automate how components move around/or are placed)

Class hierarchy (Swing – JComponent's)



JFrame – can be instantiated or extended

Fields

Modifier and Type	Field and Description
protected AccessibleContext	accessibleContext The accessible context property.
static int	EXIT_ON_CLOSE The exit application default window close operation.
protected JRootPane	rootPane The JRootPane instance that manages the contentPane and optional menuBar for this frame, as well as the glassPane.
protected boolean	rootPaneCheckingEnabled If true then calls to add and setLayout will be forwarded to the contentPane.

Fields inherited from class java.awt.Frame

CROSSHAIR_CURSOR, DEFAULT_CURSOR, E_RESIZE_CURSOR, HAND_CURSOR, ICONIFIED, MAXIMIZED_BOTH, MAXIMIZED_HORIZ, MAXIMIZED_VERT, MOVE_CURSOR, N_RESIZE_CURSOR, NE_RESIZE_CURSOR, NORMAL, NW_RESIZE_CURSOR, S_RESIZE_CURSOR, SE_RESIZE_CURSOR, SW_RESIZE_CURSOR, TEXT_CURSOR, W_RESIZE_CURSOR, WAIT_CURSOR

Fields inherited from class java.awt.Component

BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface javax.swing.WindowConstants

DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, HIDE_ON_CLOSE

Fields inherited from interface java.awt.image.ImageObserver

ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

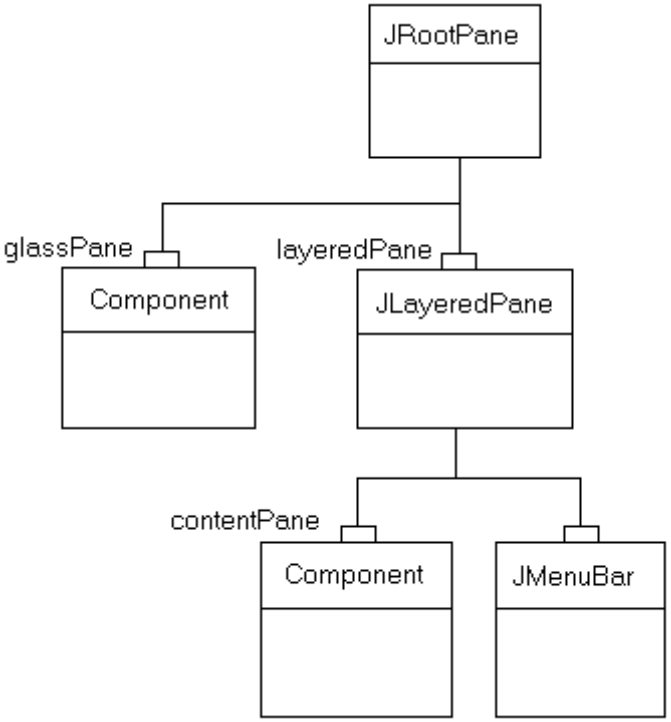
Constructors

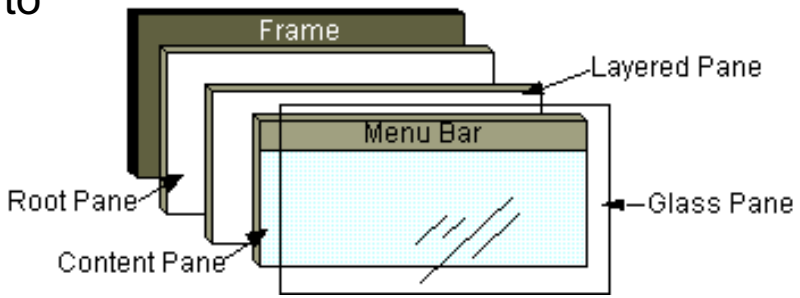
Constructor and Description
JFrame() Constructs a new frame that is initially invisible.
JFrame(GraphicsConfiguration gc) Creates a Frame in the specified GraphicsConfiguration of a screen device and a blank title.
JFrame(String title) Creates a new, initially invisible Frame with the specified title.
JFrame(String title, GraphicsConfiguration gc) Creates a JFrame with the specified title and the specified GraphicsConfiguration of a screen device.

JFrame

- **JFrame** has-a JRootPane object

- JRootPane has several parts:
  - layered pane
    - positions both the content pane and optional menu bar
  - menu bar (optional)
  - **content pane**
    - container for root pane's visible components (excl. menu bar)
    - this is where most of your GUI components will go
  - glass pane
    - transparent, can be switched on to intercept interaction events





Analogy (fishing tackle box)

Box = JFrame

compartment(s) = JPanels / Panes

Content pane

- The JFrame has-a Container called “contentPane”
- This is essentially a JPanel that is used to hold a set of other JComponents

- the JFrame renders (paints) itself by telling is sub-containers to render themselves … similarly, each sub-container tells its sub components to render etc.

- Some of these may/may not be subject to a **layout manager** (which can automate how components position/re-position themselves when the JFrame moves/resizes)

Content pane (getter/setter exists in JFrame)

getContentPane

`public Container getContentPane()`

Returns the contentPane object for this frame.

Specified by:

`getContentPane` in interface `RootPaneContainer`

Returns:

the contentPane property

See Also:

`setContentPane(java.awt.Container)`, `RootPaneContainer.getContentPane()`

setContentPane

`public void setContentPane(Container contentPane)`

Sets the contentPane property. This method is called by the constructor.

Swing's painting architecture requires an opaque `JComponent` in the containment hierarchy. This is typically provided by the content pane. If you replace the content pane it is recommended you replace it with an opaque `JComponent`.

Specified by:

`setContentPane` in interface `RootPaneContainer`

Parameters:

contentPane - the contentPane object for this frame

Throws:

`IllegalComponentStateException` - (a runtime exception) if the content pane parameter is null

See Also:

`getContentPane()`, `RootPaneContainer.setContentPane(java.awt.Container)`, `JRootPane`

YORK UNIVERSITÉ UNIVERSITY

Adding multiple components to content pane

- Add implicitly:

```
// adding to this JFrame
// (adds components to its contentPane field)
this.add(heading);
this.add(iconLabel);
this.add(textButton);
this.add(iconButton);
this.add(iconTextButton);
```

- OR.. get reference to/ create a content pane, explicitly add to it, then (re)set it as the JFrame's content pane

```
Container pane = this.getContentPane();
// Container pane = new JPanel();  // generic pane
pane.add(heading);
pane.add(iconLabel);
pane.add(textButton);
pane.add(iconButton);
pane.add(iconTextButton);

// re-set content pane
this.setContentPane(pane);
```

Absolute Positioning (no Layout Manager)

```
JFrame app = new JFrame();
app.setLayout(null);
```

- All Containers and GUI components can be positioned directly in the user space (the coord system of a window)!
  - Generally achieved through use of:
    - Insets & Dimension objects, and methods like:
    - setBounds(…); setSize(…); setPreferredSize(…); …

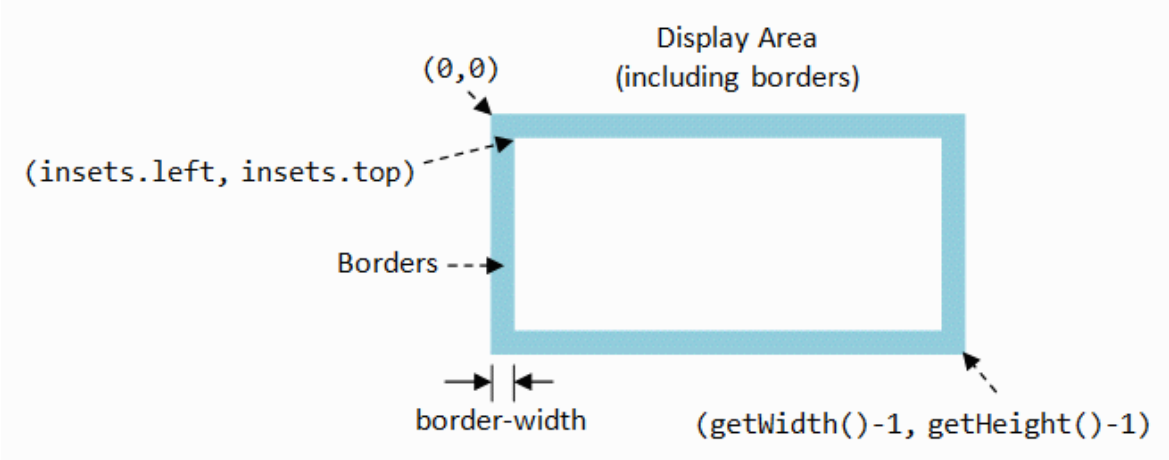
Absolute positioning of a JComponent

```
// only works if no layout manager used (i.e. null)
JFrame app = new JFrame();
app.setLayout(null);


// Text based JLabel
JLabel textLabel = new JLabel("Text Label");
textLabel.setBounds(50, 100, 200, 50);


// Text based JButtons
JButton textButton = new JButton("Text Button 1");
textButton.setBounds(260, 100, 200, 50);
```

Insets & Dimension

- java.awt.Insets
  - a representation of the borders of a container

```
Insets myInsets = new Insets(top,left,bottom,right);
JFrame f = new JFrame();
Insets myInsets2 = f.getInsets();
```



  - specifies the space that a container must leave at each of its edges (the space can be a border, a blank space, or a title.

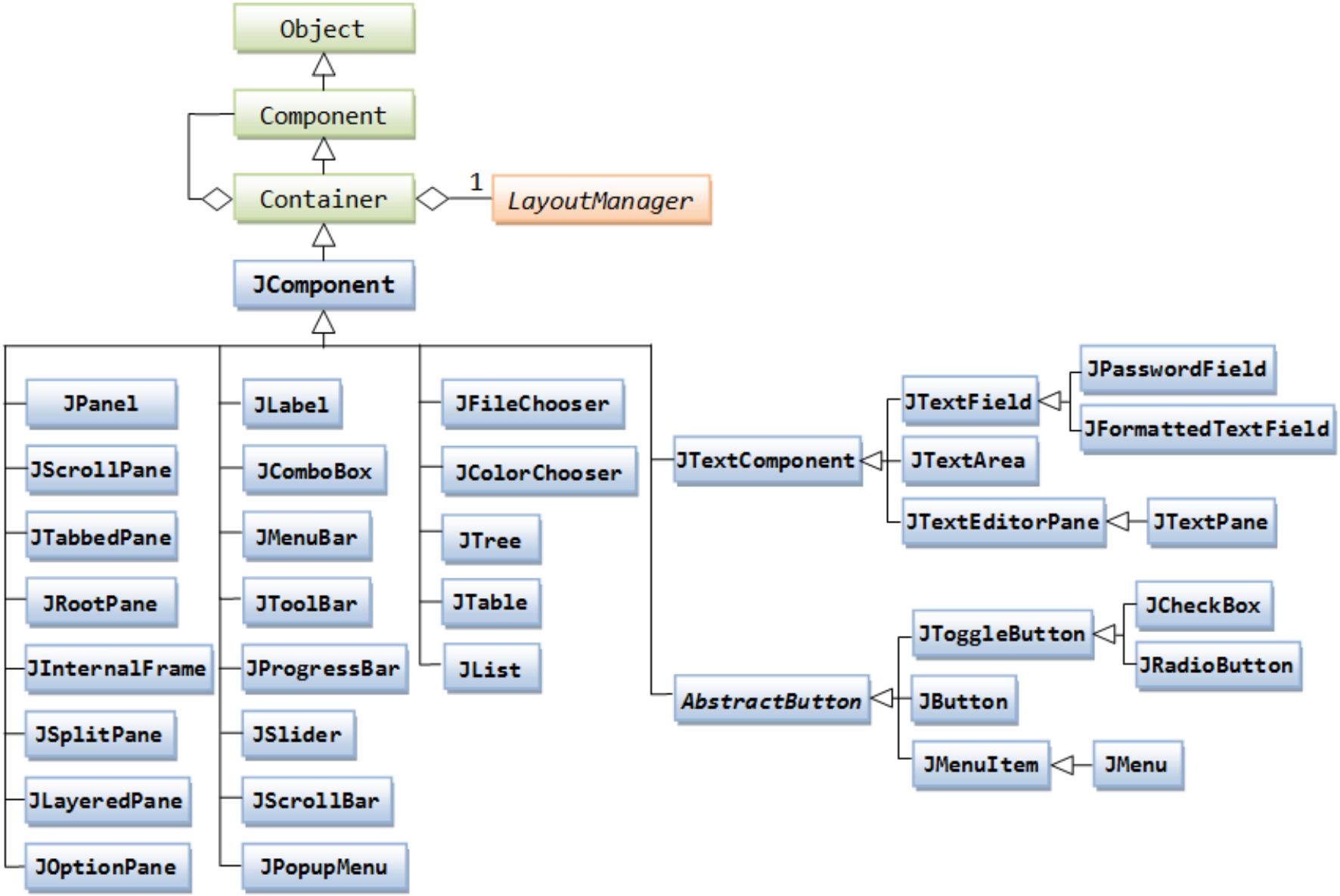
Insets & Dimension

- java.awt.Dimension
  - encapsulates the width and height of a component (in integer precision) in a single object.

```
// a Swing gui component
Jbutton b1 = new Jbutton(“one”);

// uses this size to draw component if layout manager not used
Dimension sizePref = b1.getPreferredSize();
System.out.println(“B1 (pref): width = “ + sizePref.width + “ “ +
                                    “height = “ + sizePref.height);

// actual size of component
Dimension sizeAct = b1.getSize();
System.out.println(“B1 (act): width = “ + sizePref.width + “ “ +
                                    “height = “ + sizePref.height);
```

Swing GUI components

Class hierarchy (Swing)



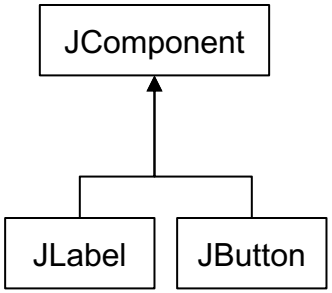
Common GUI controls

- JLabel, JButton & Icon/ImageIcon
- JTextField vs. JTextArea
- JToggleButtons: JCheckBox / JRadioButton

JLabel

Constructors

Constructor and Description
`JLabel()` Creates a `JLabel` instance with no image and with an empty string for the title.
`JLabel(Icon image)` Creates a `JLabel` instance with the specified image.
`JLabel(Icon image, int horizontalAlignment)` Creates a `JLabel` instance with the specified image and horizontal alignment.
`JLabel(String text)` Creates a `JLabel` instance with the specified text.
`JLabel(String text, Icon icon, int horizontalAlignment)` Creates a `JLabel` instance with the specified text, image, and horizontal alignment.
`JLabel(String text, int horizontalAlignment)` Creates a `JLabel` instance with the specified text and horizontal alignment.



JButton

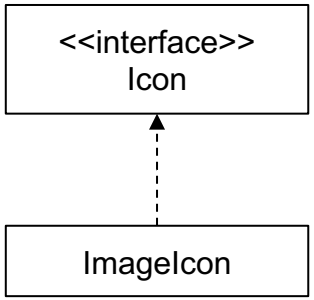
Constructors

Constructor and Description
`JButton()` Creates a button with no set text or icon.
`JButton(Action a)` Creates a button where properties are taken from the `Action` supplied.
`JButton(Icon icon)` Creates a button with an icon.
`JButton(String text)` Creates a button with text.
`JButton(String text, Icon icon)` Creates a button with initial text and an icon.

https://docs.oracle.com/javase/tutorial/uiswing/components/button.html

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

public class LabelsButtons extends JFrame {

	public LabelsButtons(String name) {
		super(name); // create parent sub-object (JFrame with String arg - sets window title to String)

		// Text based JLabel
		JLabel textLabel = new JLabel("Text Label");
		textLabel.setBounds(50, 100, 200, 50);

		// Text based JButtons
		JButton textButton = new JButton("Text Button 1");
		textButton.setBounds(260, 100, 200, 50);

		JButton textButton2 = new JButton("Text Button 2");
		textButton2.setForeground(Color.BLUE);
		textButton2.setToolTipText("press me!");

		// get content pane from top level container (this is-a JFrame)
		Container pane = this.getContentPane();

		// add components to pane
		pane.add(textLabel);
		pane.add(textButton);
		pane.add(textButton2);

		// set pane as this content pane
		this.setContentPane(pane);

		// setup JFrame object for display
		this.setSize(480,400);			// frame size 480 width and 400 height
		this.setResizable(true);		// allow/restrict window resizing
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // close on pressing 'x' button
		this.setVisible(true);			// now frame will be visible, by default not visible

	}
	public static void main(String[] args) {
		// Create GUI layout
		LabelsButtons frame = new LabelsButtons("Labels & Buttons");
	}
}
```

Using Icons on JLabels & JButtons




```
public class ImageIcon
extends Object
implements Icon, Serializable, Accessible
```

An implementation of the Icon interface that paints Icons from Images. Images that are created from a URL, filename or byte array are preloaded using MediaTracker to monitor the loaded state of the image.

Constructors

Constructor and Description
`ImageIcon()` Creates an uninitialized image icon.
`ImageIcon(byte[] imageData)` Creates an ImageIcon from an array of bytes which were read from an image file containing a supported image format, such as GIF, JPEG, or (as of 1.3) PNG.
`ImageIcon(byte[] imageData, String description)` Creates an ImageIcon from an array of bytes which were read from an image file containing a supported image format, such as GIF, JPEG, or (as of 1.3) PNG.
`ImageIcon(Image image)` Creates an ImageIcon from an image object.
`ImageIcon(Image image, String description)` Creates an ImageIcon from the image.
`ImageIcon(String filename)` Creates an ImageIcon from the specified file.
`ImageIcon(String filename, String description)` Creates an ImageIcon from the specified file.
`ImageIcon(URL location)` Creates an ImageIcon from the specified URL.
`ImageIcon(URL location, String description)` Creates an ImageIcon from the specified URL.

YORK UNIVERSITÉ UNIVERSITY

JLabels & JButtons (with icons!)

```
// Icon based Jlabel
Icon fileIcon = new ImageIcon("images/3d_file.png");

JLabel iconLabel = new JLabel("olah", fileIcon, JLabel.CENTER );
iconLabel.setBounds(200, 50, 250, 50);
```

// alignment

```
// Icon based Button
Icon trashIcon = new ImageIcon("images/3d_trash_can.png");

JButton iconButton = new JButton(trashIcon) ;
iconButton.setBounds(200, 150, trashIcon.getIconWidth(),
                                    trashIcon.getIconHeight());
```

YORK UNIVERSITÉ UNIVERSITY

Modifying Fonts (on text based JComponents):

- AWT has a Font class (used also by Swing & JavaFX)

```
java.awt.Font
    Font(String name, int style, int size)        // constructor
```

Dialog
SansSerif
Serif
Monospaced
DialogInput

Font.PLAIN
Font.BOLD
Font.ITALIC

Can also combine:
Font.BOLD | Font.ITALIC

Integer (point size of the font)

E.g. 12 = 12pt
28 = 28pt

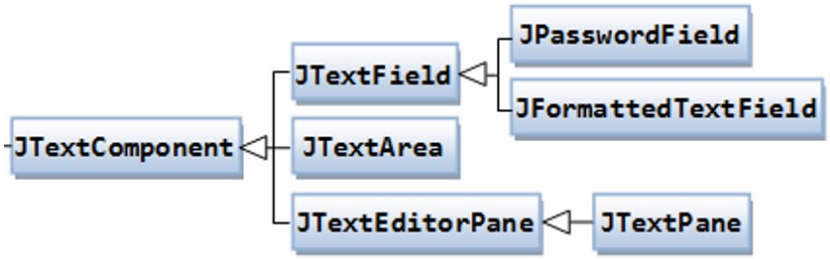
```
// Text based JLabel
JLabel textLabel = new JLabel("Label with Font");

textLabel.setFont(new Font("SansSerif", Font.BOLD|Font.ITALIC, 18));

textLabel.setForeground(Color.GREEN);
textLabel.setBounds(50, 100, 200, 50);
textLabel.setToolTipText("I am a Label!");


// Text based JButton
JButton textButton2 = new JButton("Text Button 2");
textButton2.setForeground(Color.BLUE);

textButton2.setFont(new Font("Monospaced", Font.PLAIN, 22));

textButton2.setToolTipText("press me!");
textButton2.setBounds(260, 160, 200, 50);
```

Aside:

- Toolkit class can be queried to see what fonts exist on your system:

```
// import from java.awt.Toolkit

// What fonts are available ?
String[] fonts = Toolkit.getDefaultToolkit().getFontList();

for (String f : fonts)
        System.out.println(f);
```

JTextField / JTextArea



Constructors

Constructor and Description
`JTextField()` Constructs a new TextField.
`JTextField(Document doc, String text, int columns)` Constructs a new JTextField that uses the given text storage model and the given number of columns.
`JTextField(int columns)` Constructs a new empty TextField with the specified number of columns.
`JTextField(String text)` Constructs a new TextField initialized with the specified text.
`JTextField(String text, int columns)` Constructs a new TextField initialized with the specified text and columns.

Constructors

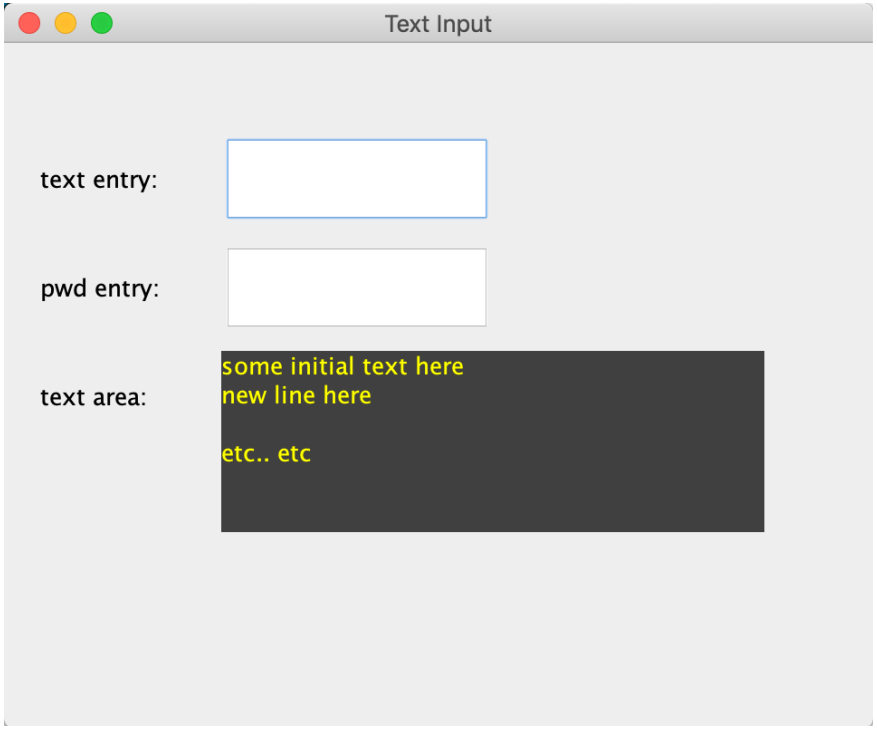
Constructor and Description
`JPasswordField()` Constructs a new JPasswordField, with a default document, null starting text string, and 0 column width.
`JPasswordField(Document doc, String txt, int columns)` Constructs a new JPasswordField that uses the given text storage model and the given number of columns.
`JPasswordField(int columns)` Constructs a new empty JPasswordField with the specified number of columns.
`JPasswordField(String text)` Constructs a new JPasswordField initialized with the specified text.
`JPasswordField(String text, int columns)` Constructs a new JPasswordField initialized with the specified text and columns.

Constructors

Constructor and Description
`JTextArea()` Constructs a new TextArea.
`JTextArea(Document doc)` Constructs a new JTextArea with the given document model, and defaults for all of the other arguments (null, 0, 0).
`JTextArea(Document doc, String text, int rows, int columns)` Constructs a new JTextArea with the specified number of rows and columns, and the given model.
`JTextArea(int rows, int columns)` Constructs a new empty TextArea with the specified number of rows and columns.
`JTextArea(String text)` Constructs a new TextArea with the specified text displayed.
`JTextArea(String text, int rows, int columns)` Constructs a new TextArea with the specified text and number of rows and columns.

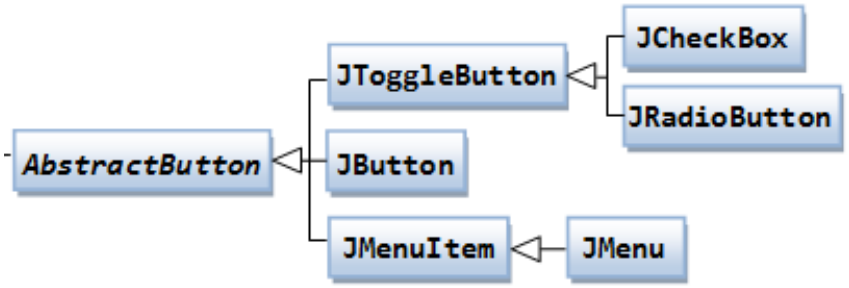
YORK UNIVERSITÉ UNIVERSITY

https://docs.oracle.com/javase/tutorial/uiswing/components/text.html




```
// Text Field
JLabel tfLabel = new JLabel("text entry:");
tfLabel.setBounds(20, 50, 100, 50);

JTextField textField = new JTextField(10);
textField.setBounds(120, 50, 150, 50);


// Password Field
JLabel pfLabel = new JLabel("pwd entry:");
pfLabel.setBounds(20, 110, 100, 50);

JPasswordField passwordField = new JPasswordField(10);
passwordField.setBounds(120, 110, 150, 50);


// TextArea
JLabel taLabel = new JLabel("text area:");
taLabel.setBounds(20, 170, 100, 50);

JTextArea textArea = new JTextArea("some initial text here\nnew line here\n\netc.. etc");
textArea.setBackground(Color.DARK_GRAY);
textArea.setForeground(Color.YELLOW);
textArea.setBounds(120, 170, 300, 100);
```

Checkboxes and RadioButtons



- JCheckBox
  - act independently (set each to on or off - i.e. checked/not)

- JRadioButton
  - act as a group (mutually exclusive, if one is on, the others in the group are all set to off)

JCheckBox

```
JCheckBox chinButton = new JCheckBox("Chin");
chinButton.setSelected(true);

JCheckBox glassesButton = new JCheckBox("Glasses");
glassesButton.setSelected(true);

JCheckBox hairButton = new JCheckBox("Hair");
hairButton.setSelected(true);

JCheckBox teethButton = new JCheckBox("Teeth");
teethButton.setSelected(true);
```

```
// ...
Container pane = this.getContentPane();
pane.add(chinButton);
pane.add(glassesButton);
pane.add(hairButton);
pane.add(teethButton);
this.setContentPane(pane);
```

JCheckBox

Other constructors allow ImageIcon

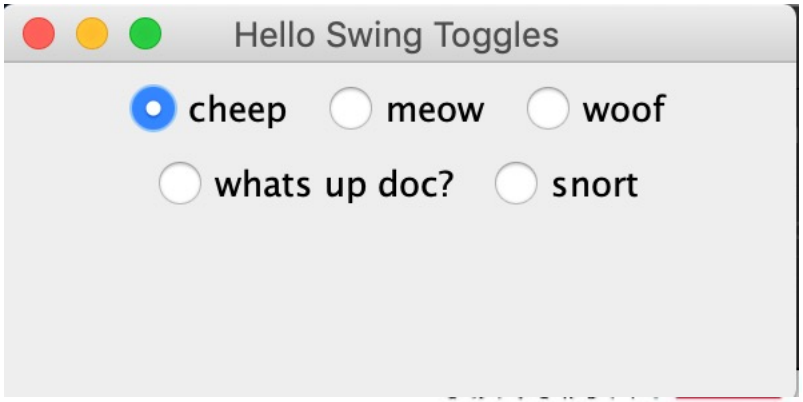
```
JCheckBox chinButton = new JCheckBox("Chin");
chinButton.setSelected(false);

JCheckBox glassesButton = new JCheckBox("Glasses");
glassesButton.setSelected(true);

JCheckBox hairButton = new JCheckBox("Hair");
hairButton.setSelected(false);

JCheckBox teethButton = new JCheckBox("Teeth");
teethButton.setSelected(true);


// ...
Container pane = this.getContentPane();
pane.add(chinButton);
pane.add(glassesButton);
pane.add(hairButton);
pane.add(teethButton);
this.setContentPane(pane);
```

JRadioButton (& ButtonGroup)

```
JRadioButton birdButton = new JRadioButton("cheep");
birdButton.setSelected(true);

JRadioButton catButton = new JRadioButton("meow");
JRadioButton dogButton = new JRadioButton("woof");
JRadioButton rabbitButton = new JRadioButton("whats up doc?");
JRadioButton pigButton = new JRadioButton("snort");

//Group the radio buttons (used to figure out which is clicked)
ButtonGroup group = new ButtonGroup();
group.add(birdButton);
group.add(catButton);
group.add(dogButton);
group.add(rabbitButton);
group.add(pigButton);
```

JRadioButton (& ButtonGroup)

```
// ...

Container pane = this.getContentPane();

pane.add(birdButton);
pane.add(catButton);
pane.add(dogButton);
pane.add(rabbitButton);
pane.add(pigButton);


this.setContentPane(pane);
```



Using Layout Managers

Layout Managers

- Preferred way of arranging components
  - makes it easier to adjust look and feel, etc.
  - they can also be used by different containers and other programs

- Responsible for 2 things:
  - Automatically calculates the minimum/preferred/maximum sizes for a container
  - Lay out the container's children
    - (somewhat automatic given hints/settings of the manager, and the type of layout it is attempting to maintain on the components)

Layout Managers

- BorderLayout - North, South, East West + Centre
- BoxLayout - Vertical/Horizontal Boxes
- FlowLayout - Left to right, new row when too wide
- GridLayout - regularly spaced grid
- GridBagLayout - irregularly spaced grid (more complex)

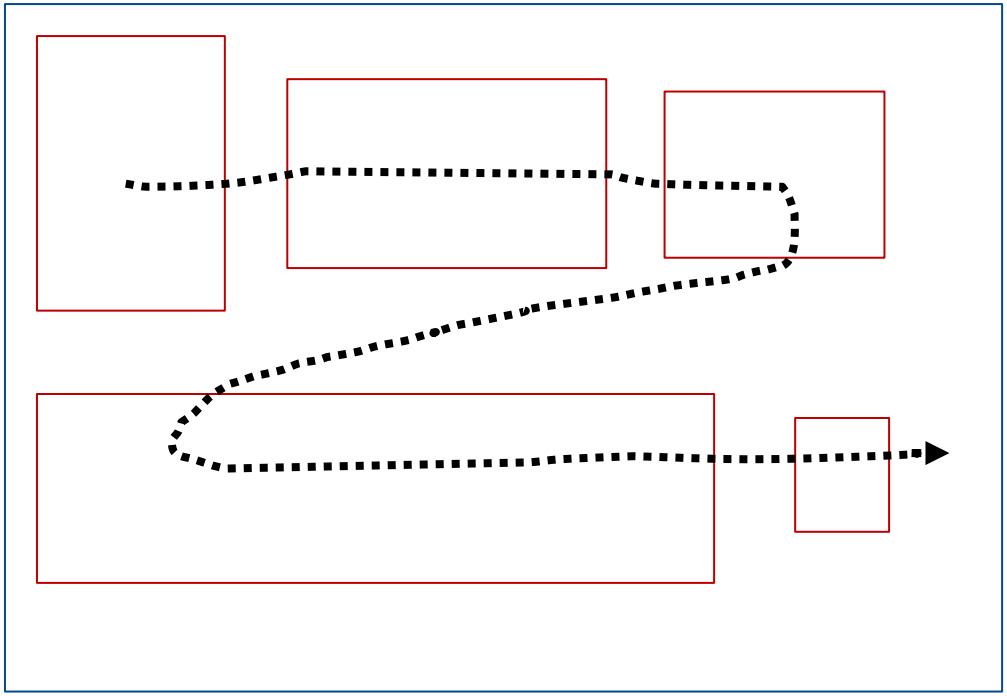
- Others:
  - CardLayout - (when part of screen's UI's freq. changes)
  - GroupLayout - (separate horizontal & vertical layouts)
  - SpringLayout - (allows for dependencies between components and child dialog/popup windows)

How are they configured?

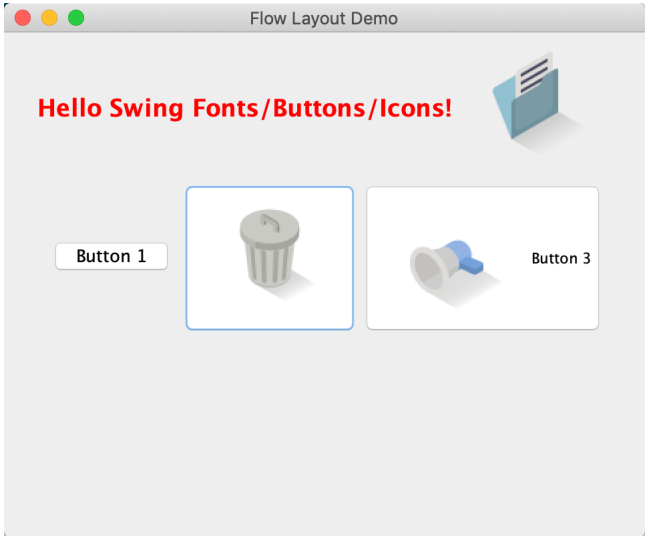
- How do they work?
  - Set layout manager:
    - specify when constructing a JPanel
    - or use **setLayout()** method for an existing JPanel or content pane (obtained from a top-level container)

FlowLayout

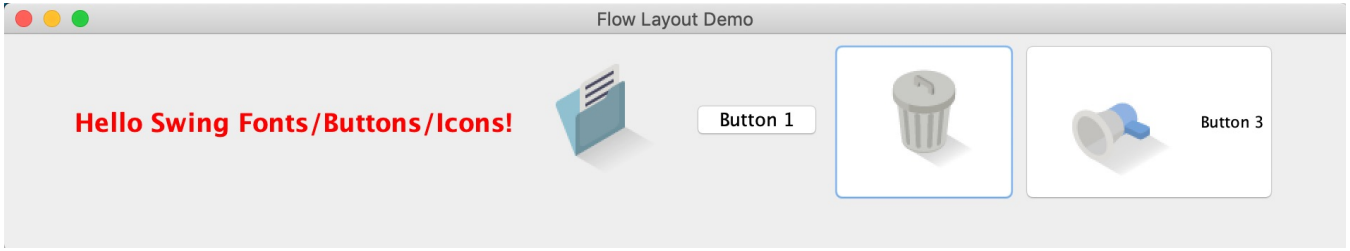
- Simplest to use (adds components left to right)
  - If a component won't fit in size of window, placement of the next component continues on the next row



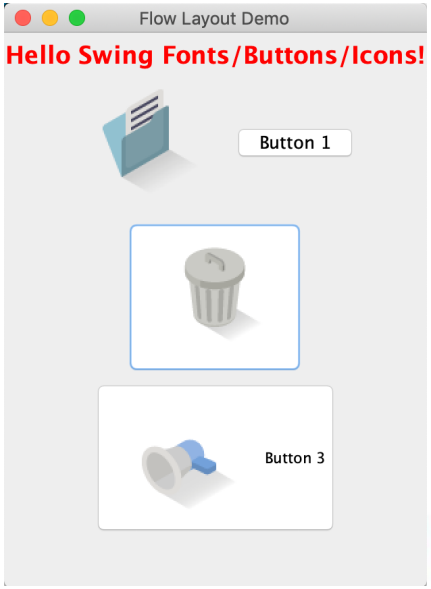
FlowLayout




```
// in constructor:
super(name);
this.setLayout(new FlowLayout());

// no need to use setBounds(..)
// for each new component added
```

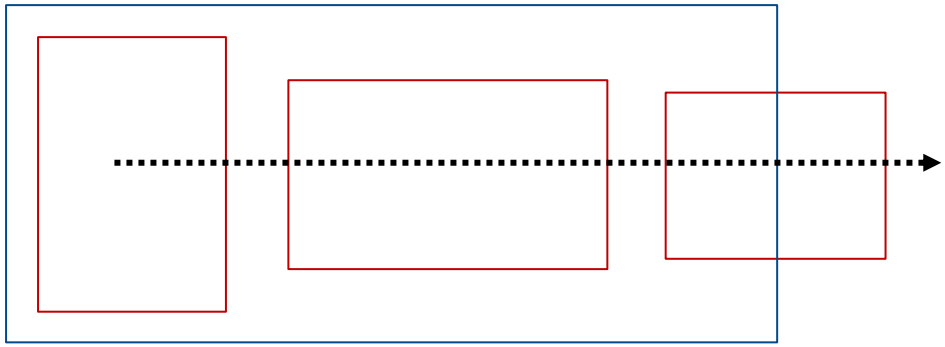


resizing window
re-arranges components

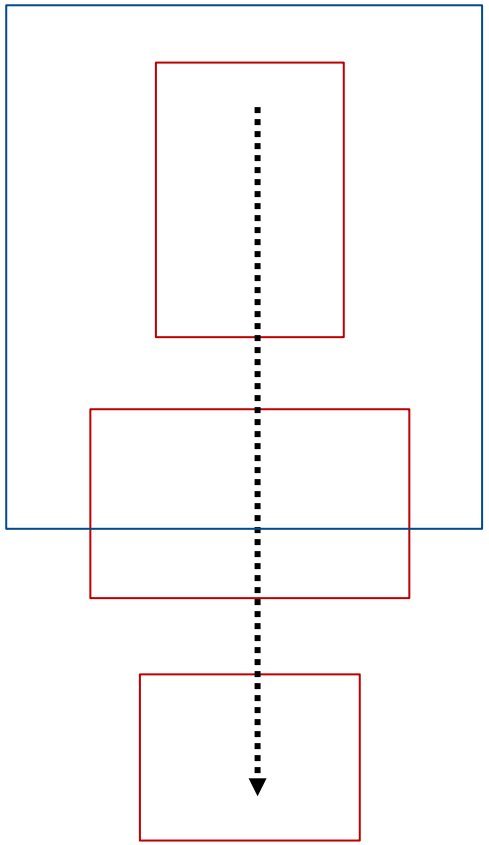


```
public class FlowLayoutDemo extends JFrame {

	// constructor
	public FlowLayoutDemo(String name) {
		super(name); // create parent sub-object
		this.setLayout(new FlowLayout()); // set to FlowLayout

		// setup components
		JLabel heading = new JLabel("Hello Swing Fonts/Buttons/Icons!");
		heading.setFont(new Font("SansSerif", Font.BOLD, 18));
		heading.setForeground(Color.RED);

		JLabel iconLabel = new JLabel( new ImageIcon("src/images/3d_file.png") );
		JButton textButton = new JButton("Button 1");
		JButton iconButton = new JButton(new ImageIcon("src/images/3d_trash_can.png")) ;
		JButton iconTextButton = new JButton("Button 3", new ImageIcon("src/images/advertising.png") );

		Container pane = this.getContentPane();
		pane.add(heading);
		pane.add(iconLabel);
		pane.add(textButton);
		pane.add(iconButton);
		pane.add(iconTextButton);

		// set content pane back into this object
		this.setContentPane(pane);

		// setup JFrame object for display
		this.setSize(480,400); // frame size 480 width and 400 height
		this.setResizable(true); // allow/restrict window resizing
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // close on pressing 'x' button
		this.setVisible(true); // now frame will be visible, by default not visible
	}

	public static void main(String[] args) {
		FlowLayoutDemo frame = new FlowLayoutDemo("Flow Layout Demo");
	}

}
```

BoxLayout

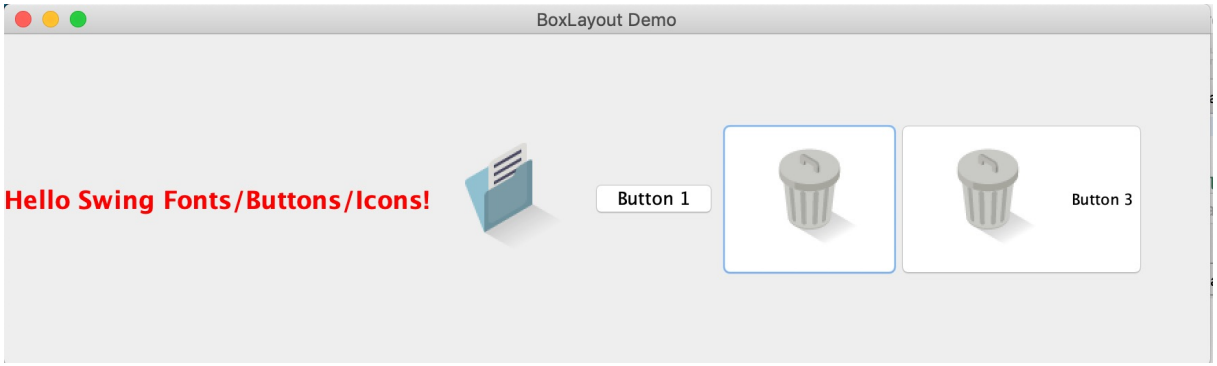


OR

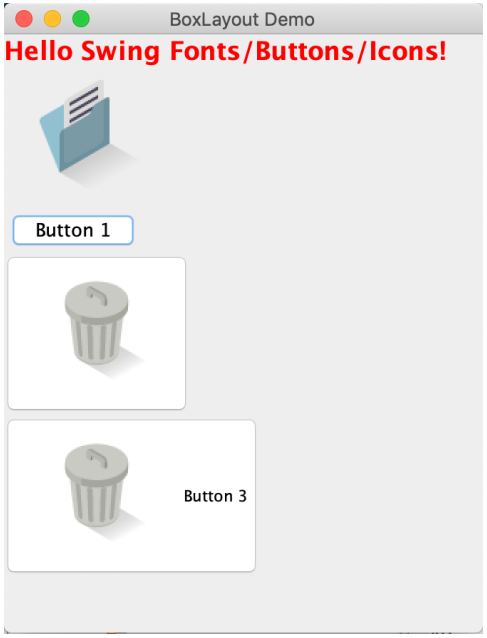


```
this.setLayout(
    new BoxLayout(this.getContentPane(),
                  BoxLayout.X_AXIS)      );
```

BoxLayout.X_AXIS
BoxLayout.Y_AXIS
BoxLayout.PAGE_AXIS
BoxLayout.LINE_AXIS



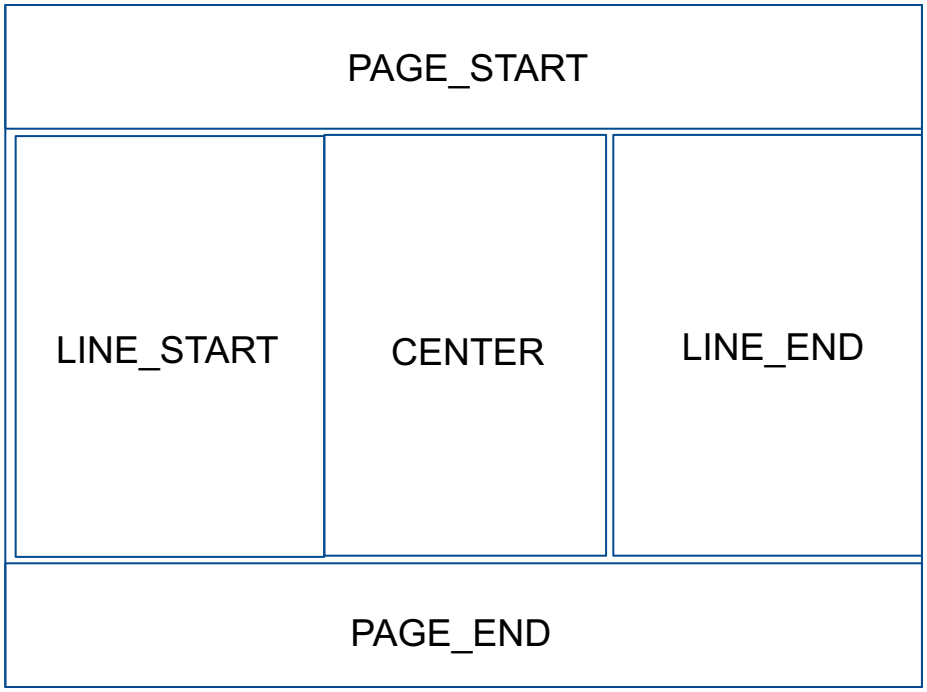

```
this.setLayout(new BoxLayout(this.getContentPane(), BoxLayout.X_AXIS));
this.setLayout(new BoxLayout(this.getContentPane(), BoxLayout.LINE_AXIS));
```

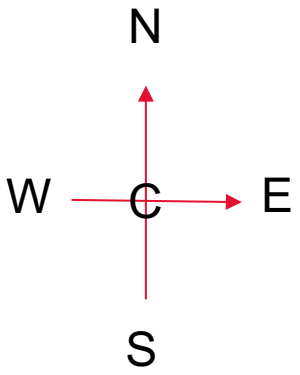



```
this.setLayout(new BoxLayout(this.getContentPane(),
                                        BoxLayout.Y_AXIS));
this.setLayout(new BoxLayout(this.getContentPane(),
                                        BoxLayout.PAGE_AXIS));
```

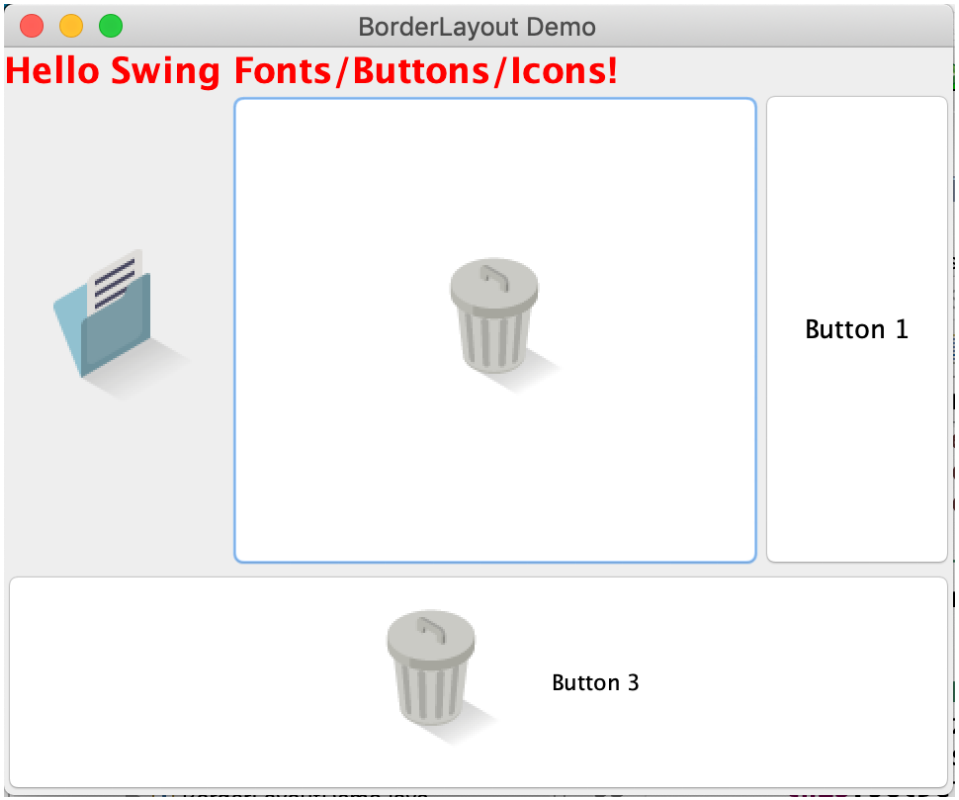
```
public class BoxLayoutDemo extends JFrame {

	public BoxLayoutDemo(String name) {

		super(name);
		this.setLayout(new BoxLayout(this.getContentPane(), BoxLayout.Y_AXIS));

		// setup components
		JLabel heading = new JLabel("Hello Swing Fonts/Buttons/Icons!");
		heading.setFont(new Font("SansSerif", Font.BOLD, 18));
		heading.setForeground(Color.RED);

		JLabel iconLabel = new JLabel( new ImageIcon("src/images/3d_file.png") );
		JButton textButton = new JButton("Button 1");
		JButton iconButton = new JButton(new ImageIcon("src/images/3d_trash_can.png")) ;
		JButton iconTextButton = new JButton("Button 3", new ImageIcon("src/images/3d_trash_can.png") );

		Container pane = this.getContentPane();
		pane.add(heading);
		pane.add(iconLabel);
		pane.add(textButton);
		pane.add(iconButton);
		pane.add(iconTextButton);
		this.setContentPane(pane);

		// setup JFrame object for display
		this.setSize(480,400); // frame size 480 width and 400 height
		this.setResizable(true); // allow/restrict window resizing
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // close on pressing 'x' button
		this.setVisible(true); // now frame will be visible, by default not visible

	}


	public static void main(String[] args) {

		BoxLayoutDemo frame = new BoxLayoutDemo("BoxLayout Demo");

	}

}
```

BorderLayout

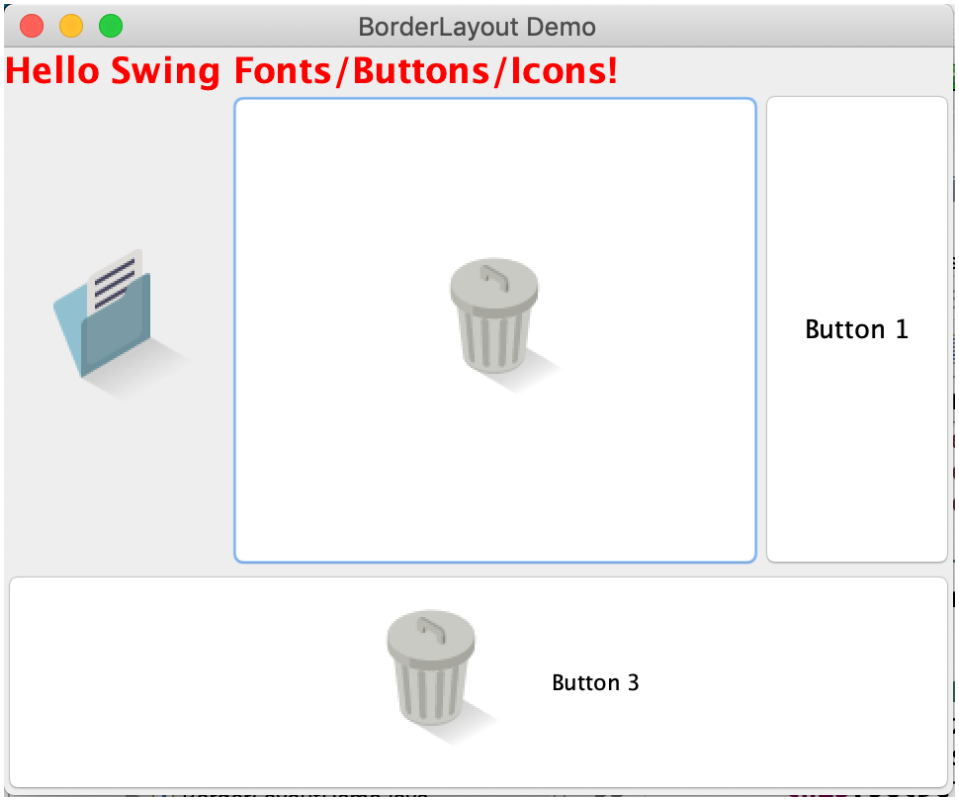





```
this.setLayout(new BorderLayout());

// but have to explicitly pin JComponents
// to these anchor points
```

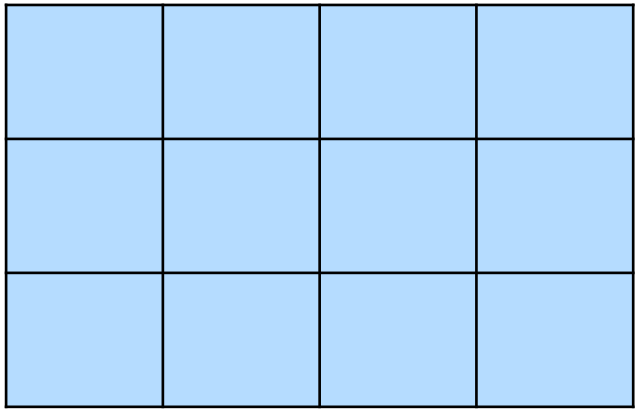


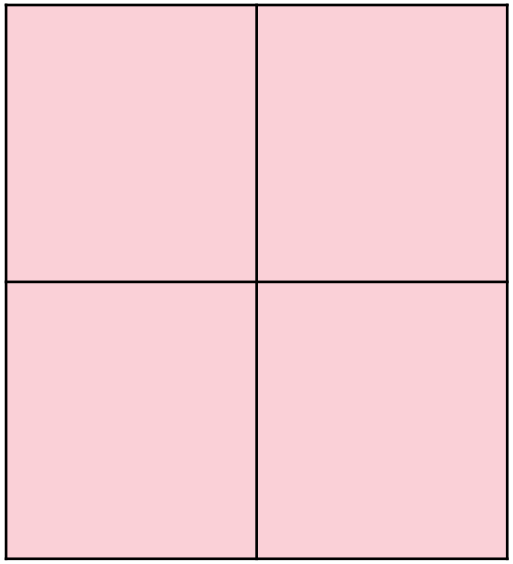

```
// done when adding components
pane.add(heading, BorderLayout.PAGE_START);
pane.add(iconLabel, BorderLayout.LINE_START);
pane.add(textButton, BorderLayout.LINE_END);
pane.add(iconButton, BorderLayout.CENTER);
pane.add(iconTextButton, BorderLayout.PAGE_END);
```




```
// similarly..
pane.add(heading, BorderLayout.NORTH);
pane.add(iconLabel, BorderLayout.WEST);
pane.add(textButton, BorderLayout.EAST);
pane.add(iconButton, BorderLayout.CENTER);
pane.add(iconTextButton, BorderLayout.SOUTH);
```

GridLayout

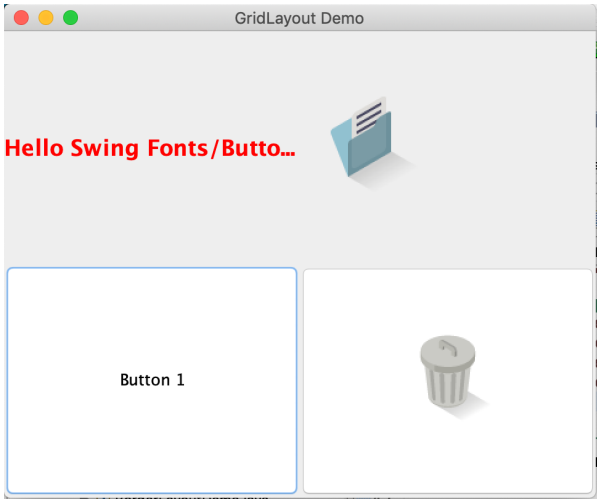




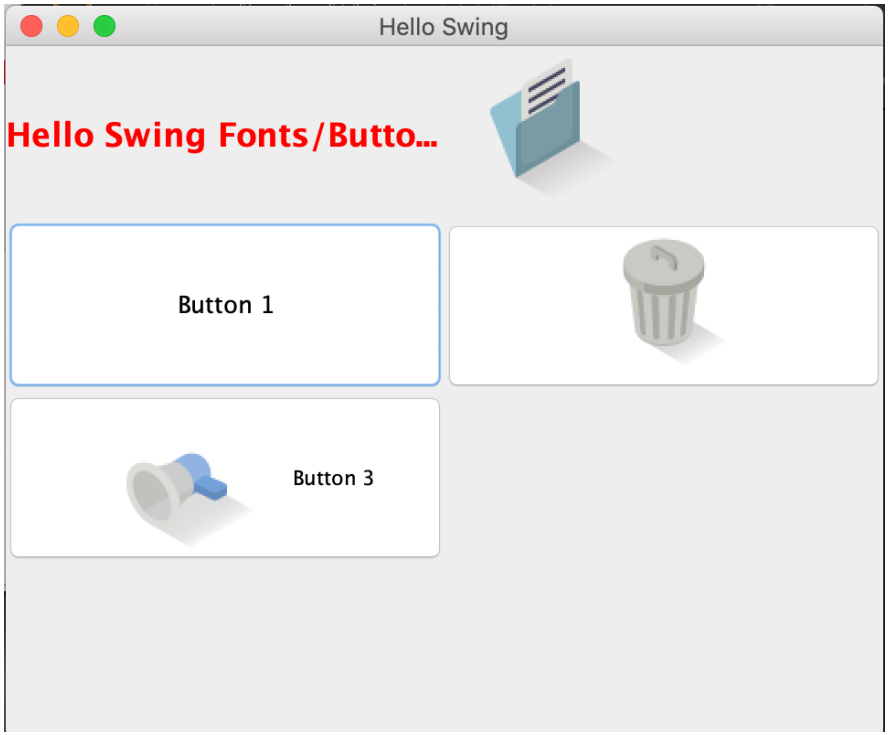
```
this.setLayout(new GridLayout(3,4));
```

```
this.setLayout(new GridLayout(2,2));
```

```
// always regular (uses all space)

// tries to fit elements as best it can
// sometimes modifies grid slightly
```



// using 2x2 grid (4 components)



// using 4x4 grid (5 components)

GridLayout - limitations

- A little unpredictable when components do not match number of grid locations

GridBagLayout

GridBagLayout is one of the most flexible & complex

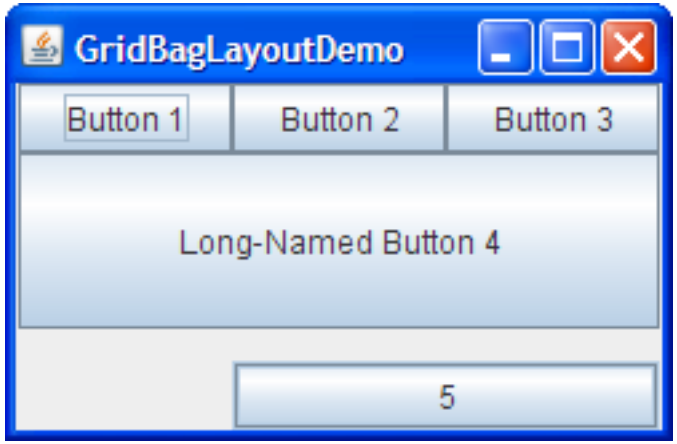
A GridBagLayout places components in a grid of rows and columns, allowing specified components to span multiple rows or columns.

Not all rows necessarily have the same height.

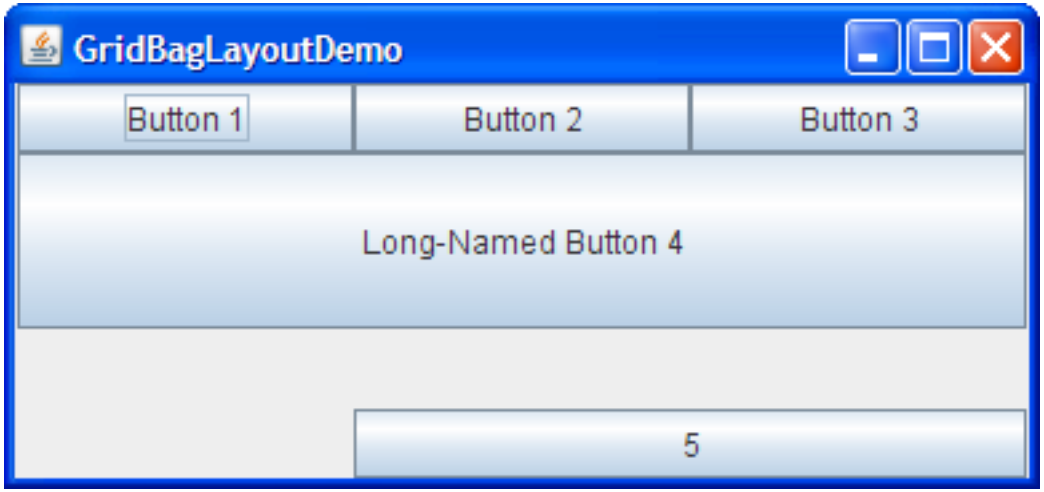
Similarly, not all columns necessarily have the same width.

Essentially, GridBagLayout places components in rectangles (cells) in a grid, and then uses the components' preferred sizes to determine how big the cells should be.

Allows for Irregular grid/ different properties per cell/component



Note this is (an older) windows look and feel (while the prev examples are mac)



From the Java Tutorials: https://docs.oracle.com/javase/tutorial/uiswing/layout/gridbag.html

GridBagLayout

The way the program specifies the size and position characteristics of its components is by specifying *constraints* for each component.

```
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);
```

GridBagConstraints

(instance variables) 🡐 i.e. class fields

gridx, **gridy**	cell at upper left of component (e.g. 0,0 is 1st in top row)
gridwidth, **gridheight**	number of columns, rows (cells) used by component
fill	Fill to fit … used if components component size > display area NONE | HORIZONTAL | VERTICAL
anchor	Anchors to … used when component size < display area

FIRST_LINE_START PAGE_START FIRST_LINE_END

LINE_START CENTER LINE_END

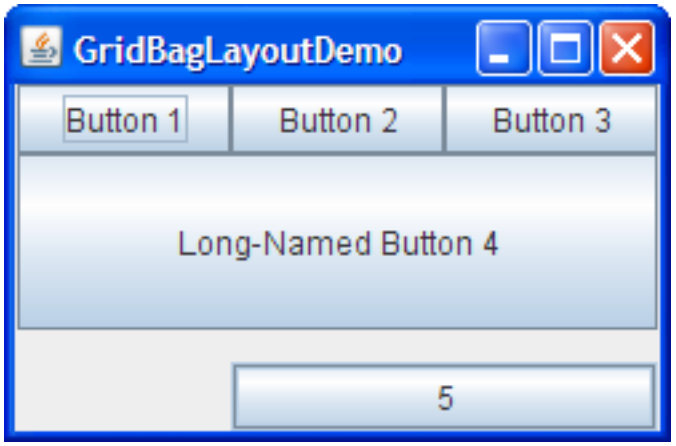
LAST_LINE_START PAGE_END LAST_LINE_END

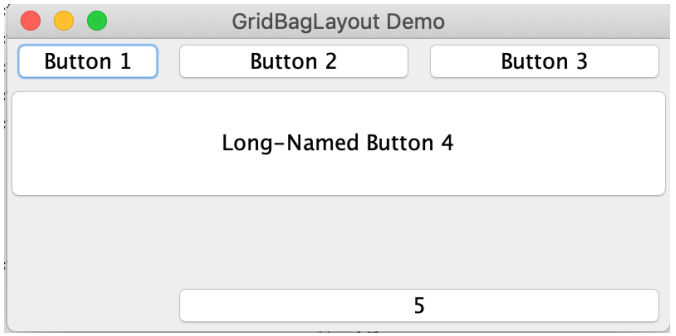
GridBagConstraints

insets	external padding of the component
ipadx, **ipady**	specifies the internal padding: how much to add to the size of the component.
weightx, **weighty**	used to determine how to distribute space among columns (weightx) and among rows (weighty) this is important for specifying resizing behavior.

More info here: https://docs.oracle.com/javase/tutorial/uiswing/layout/gridbag.html

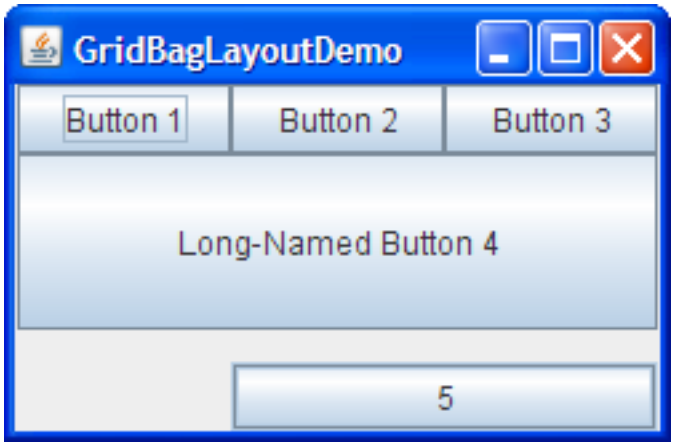
Example

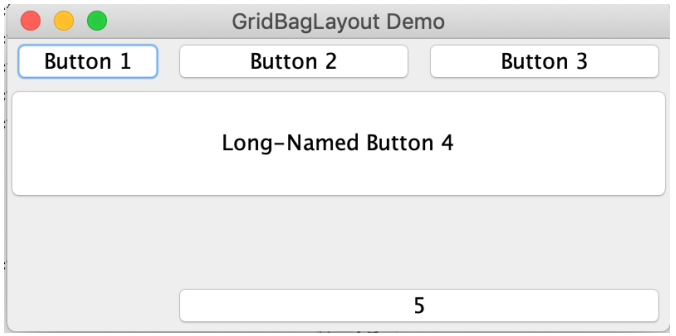





```
pane.setLayout(new GridBagLayout());
GridBagConstraints c = new
GridBagConstraints();

c.fill = GridBagConstraints.HORIZONTAL;
c.gridx = 0;
c.gridy = 0;
pane.add(button, c);

button = new JButton("Button 2");
c.fill = GridBagConstraints.HORIZONTAL;
c.weightx = 0.5;
c.gridx = 1;
c.gridy = 0;
pane.add(button, c);

// …
button = new JButton("Long-Named Button 4");
c.fill = GridBagConstraints.HORIZONTAL;
c.ipady = 40; //make this component tall
c.weightx = 0.0;
c.gridwidth = 3;
c.gridx = 0;
c.gridy = 1;
pane.add(button, c);
```

Example






```
// …

button = new JButton("5");

c.fill = GridBagConstraints.HORIZONTAL;

c.ipady = 0;        //reset to default
c.weighty = 1.0;    //request any extra
                    // vertical space


c.anchor = GridBagConstraints.PAGE_END;

//bottom of space

c.insets = new Insets(10,0,0,0); //top padding

c.gridx = 1;        //aligned with button 2
c.gridwidth = 2;    //2 columns wide
c.gridy = 2;        //third row

pane.add(button, c);
```

Other Layouts?

- Generally can do most with the above subset
- You can also place containers within containers
  - Sub containers can be specified with different layout managers!

Using Panels to organize GUI components

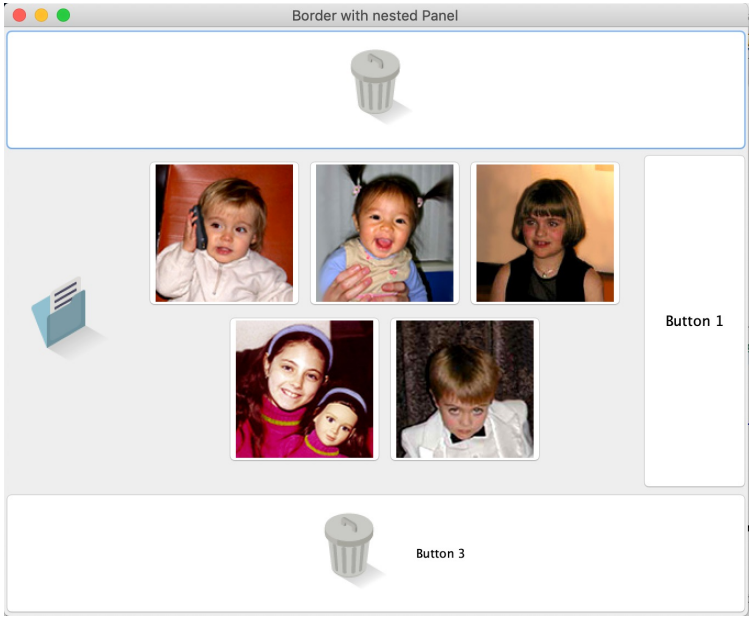
- Assume we have icons organized into a BorderLayout configuration on ContentPane.

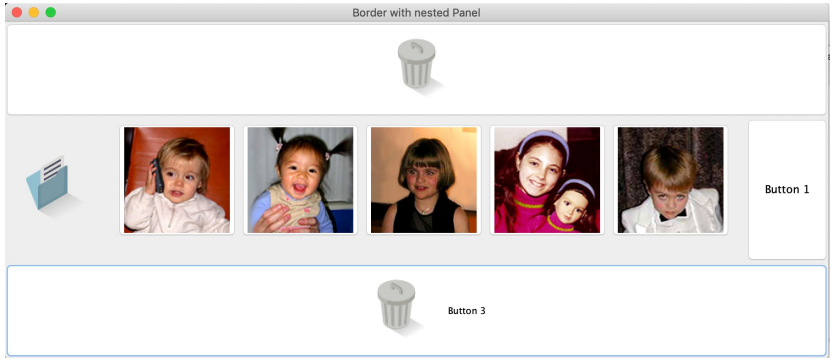
- Can “nest” JPanels within the top-level containers
  - **JPanels** can thus act as “**sub-containers**” and be used to pool GUI controls/elements
  - These **JPanels** can have their own separate layout managers
    - The JPanel bounds/size are controlled by the layout manager of the parent (container) – i.e. the ContentPane of Jframe
    - The GUI components within the JPanel are then controlled (within this bounds) to utilize the layout manager of the Jpanel

Creating a JPanel & populating it

```
// create a new panel (sub container within content pane)
JPanel jp = new JPanel();

JButton iconButton2 = new JButton(new ImageIcon("src/images/Adele.jpg")) ;
JButton iconButton3 = new JButton(new ImageIcon("src/images/Alexi.jpg")) ;
JButton iconButton4 = new JButton(new ImageIcon("src/images/Laine.jpg")) ;
JButton iconButton5 = new JButton(new ImageIcon("src/images/Maya.jpg")) ;
JButton iconButton6 = new JButton(new ImageIcon("src/images/Cosmo.jpg")) ;
```

Border with Nested Panel Demo (FlowLayout)



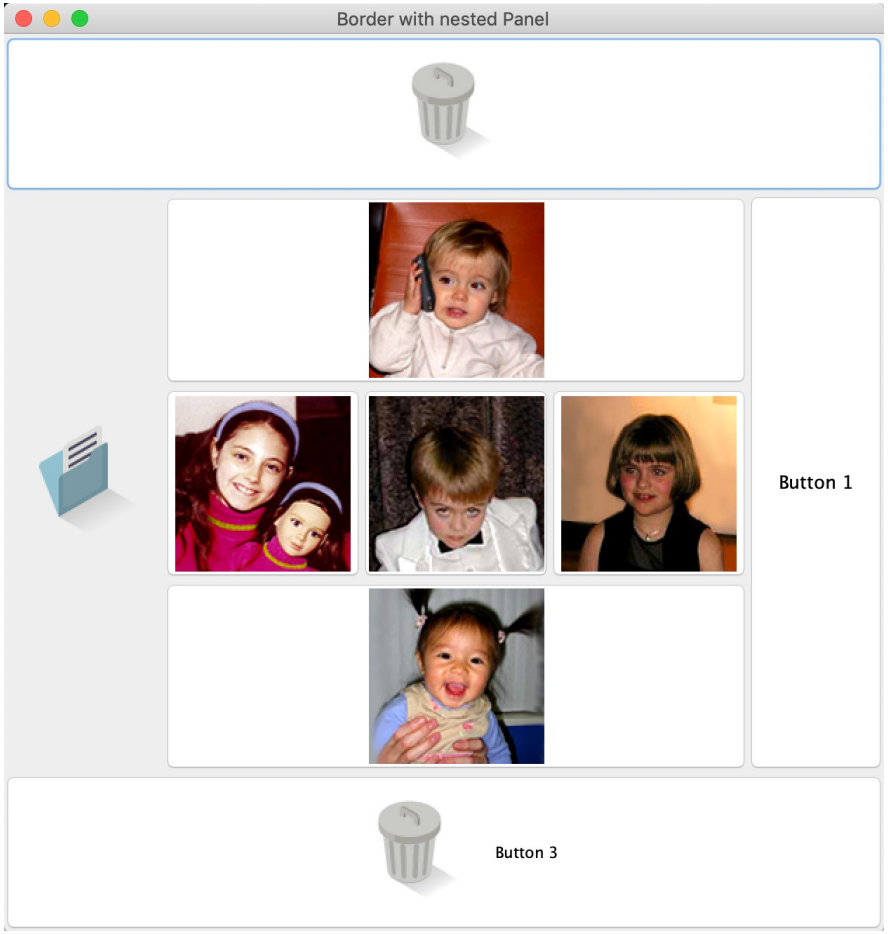



```
// create a new panel (sub container within content pane)
JPanel jp = new JPanel();

JButton iconButton2 = new JButton(new ImageIcon("src/images/Adele.jpg")) ;
JButton iconButton3 = new JButton(new ImageIcon("src/images/Alexi.jpg")) ;
JButton iconButton4 = new JButton(new ImageIcon("src/images/Laine.jpg")) ;
JButton iconButton5 = new JButton(new ImageIcon("src/images/Maya.jpg")) ;
JButton iconButton6 = new JButton(new ImageIcon("src/images/Cosmo.jpg")) ;

jp.setLayout(new FlowLayout());

jp.add(iconButton2);
jp.add(iconButton3);
jp.add(iconButton4);
jp.add(iconButton5);
jp.add(iconButton6);
```

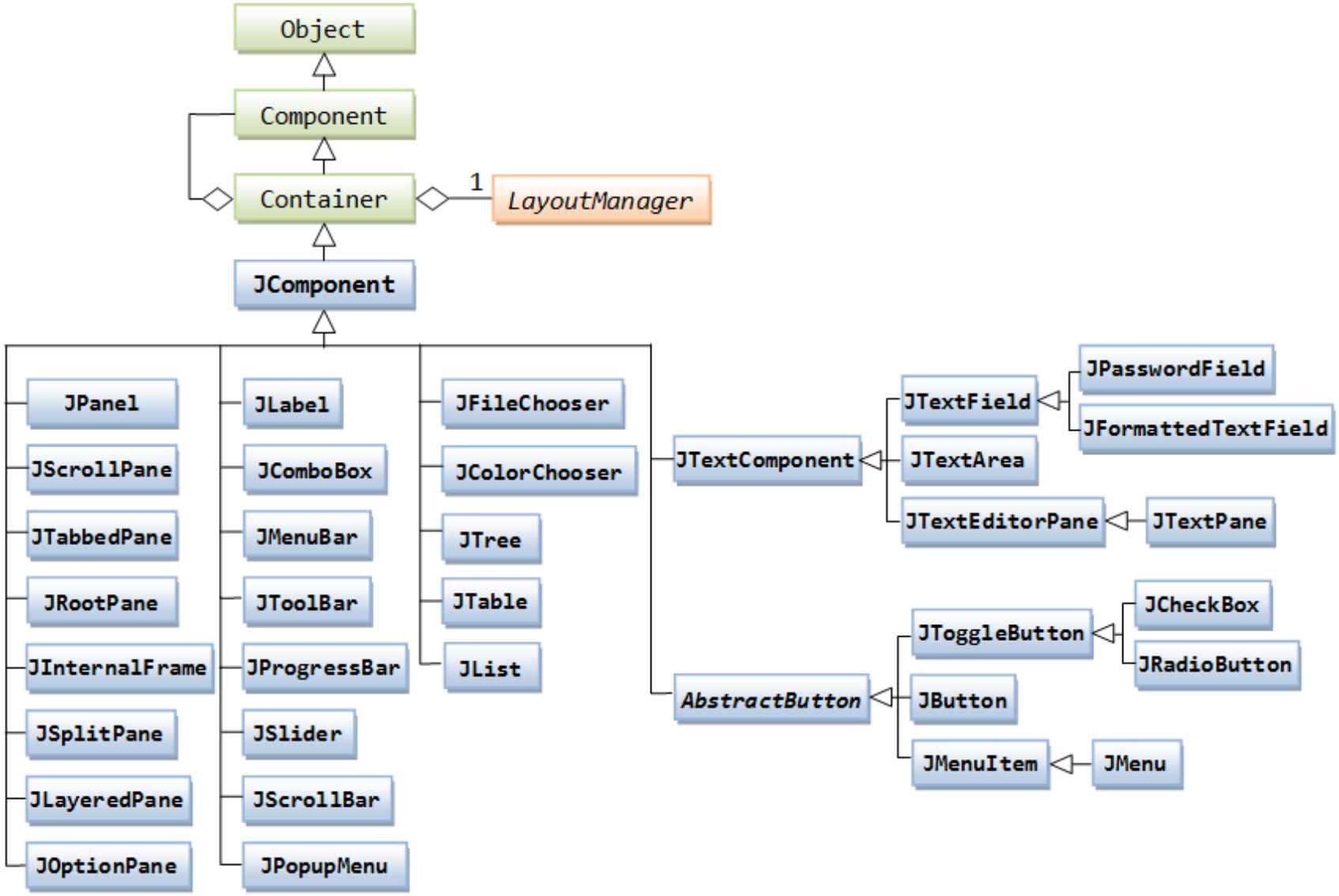
YORK UNIVERSITÉ UNIVERSITY

Border with Nested Panel Demo (BorderLayout)

```
// create a new panel (sub container within content pane)
JPanel jp = new JPanel();

JButton iconButton2 = new JButton(new ImageIcon("src/images/Adele.jpg")) ;
JButton iconButton3 = new JButton(new ImageIcon("src/images/Alexi.jpg")) ;
JButton iconButton4 = new JButton(new ImageIcon("src/images/Laine.jpg")) ;
JButton iconButton5 = new JButton(new ImageIcon("src/images/Maya.jpg")) ;
JButton iconButton6 = new JButton(new ImageIcon("src/images/Cosmo.jpg")) ;

jp.setLayout(new BorderLayout());

jp.add(iconButton2);
jp.add(iconButton3);
jp.add(iconButton4);
jp.add(iconButton5);
jp.add(iconButton6);

jp.add(iconButton2,BorderLayout.NORTH);
jp.add(iconButton3,BorderLayout.SOUTH);
jp.add(iconButton4,BorderLayout.EAST);
jp.add(iconButton5,BorderLayout.WEST);
jp.add(iconButton6,BorderLayout.CENTER);
```



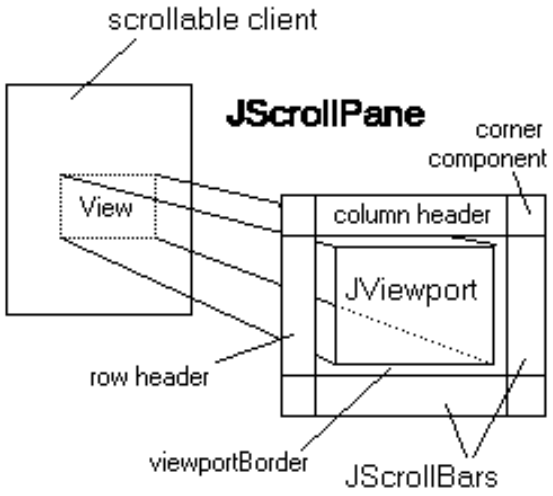
Other types of Pane Components?



JScrollPane

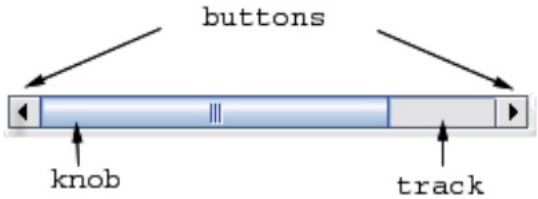
- provides a scrollable view of a component
- use when screen real estate is limited
  - i.e. use a scroll pane to display a component that is large or one whose size can change dynamically.

```
public class JScrollPane
extends JComponent
implements ScrollPaneConstants, Accessible
```



https://docs.oracle.com/javase/8/docs/api/javax/swing/JScrollPane.html

https://docs.oracle.com/javase/8/docs/api/javax/swing/ScrollPaneConstants.html



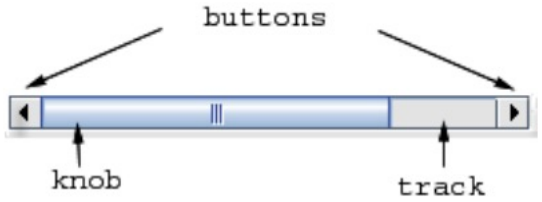
Constructors

Constructor and Description
JScrollPane() Creates an empty (no viewport view) JScrollPane where both horizontal and vertical scrollbars appear when needed.
JScrollPane(**Component** view) Creates a JScrollPane that displays the contents of the specified component, where both horizontal and vertical scrollbars appear whenever the component's contents are larger than the view.
JScrollPane(**Component** view, int vsbPolicy, int hsbPolicy) Creates a JScrollPane that displays the view component in a viewport whose view position can be controlled with a pair of scrollbars.
JScrollPane(int vsbPolicy, int hsbPolicy) Creates an empty (no viewport view) JScrollPane with specified scrollbar policies.

From **ScrollPaneContants** interface:

vsbPolicy → vertical scroll policy

hsbPolicy → horizontal scroll policy



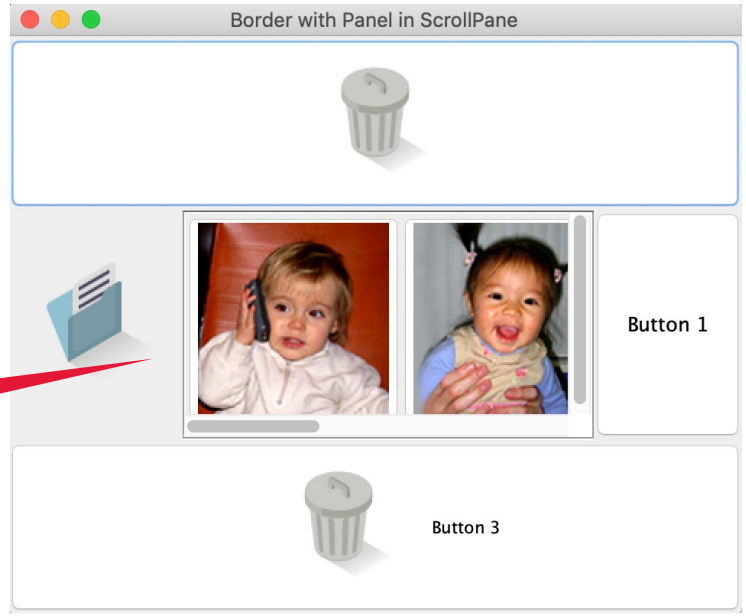
Policy	Description
VERTICAL_SCROLLBAR_AS_NEEDED HORIZONTAL_SCROLLBAR_AS_NEEDED	The default. The scroll bar appears when the viewport is smaller than the client and disappears when the viewport is larger than the client.
VERTICAL_SCROLLBAR_ALWAYS HORIZONTAL_SCROLLBAR_ALWAYS	Always display the scroll bar. The knob disappears if the viewport is large enough to show the whole client.
VERTICAL_SCROLLBAR_NEVER HORIZONTAL_SCROLLBAR_NEVER	Never display the scroll bar. Use this option if you don't want the user to directly control what part of the client is shown, or if you want them to use only non-scroll-bar techniques (such as dragging).

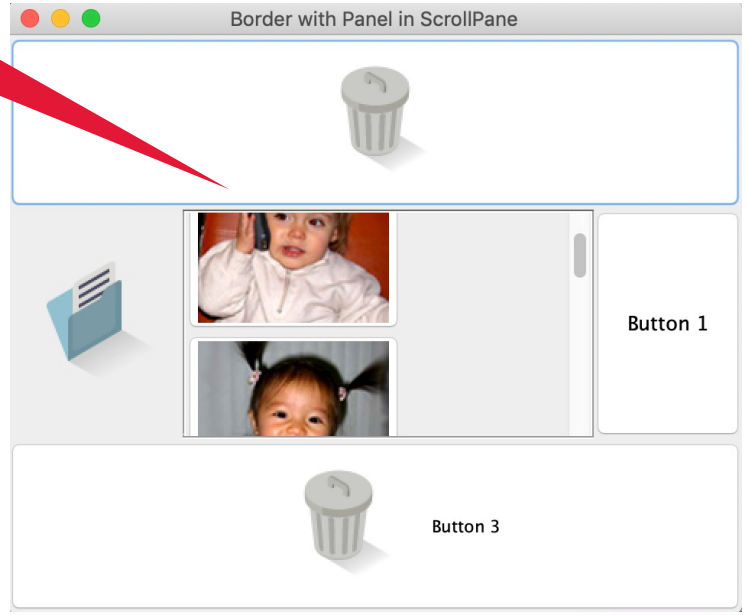
Examples

By default, vertical/horizontal scrollbars automatically appear if content (view component) is larger in either dimension

Panel uses BoxLayout along X_AXIS

Panel uses BoxLayout along Y_AXIS






```
// create a new panel (sub container within content pane)
JPanel jp = new JPanel();

// create icon components (images) to panel (not shown)

jp.setLayout(new BoxLayout(jp,BoxLayout.Y_AXIS));

// create icon components (images) to panel (not shown)

JScrollPane scrollPane = new JScrollPane(jp);

Container pane = this.getContentPane();
pane.add(scrollPane, BorderLayout.CENTER);
// ...

// set content pane back into this JFrame
this.setContentPane(pane);
```

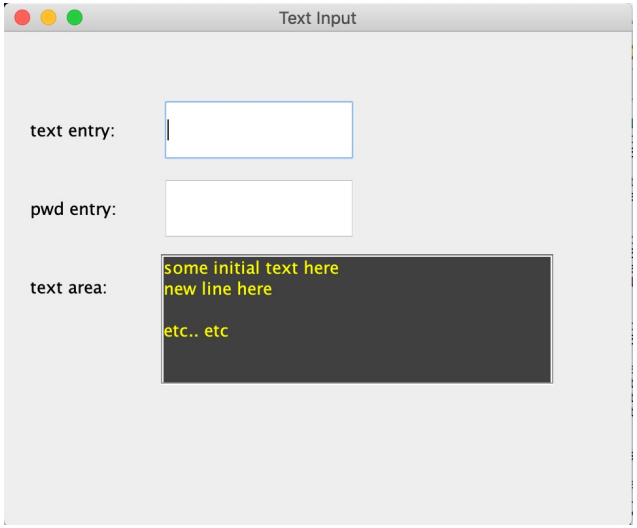
Setting scroll policies

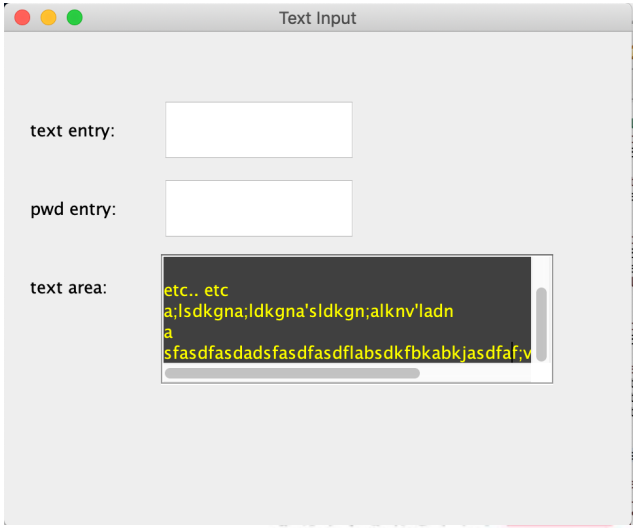
```
JScrollPane scrollPane = new JScrollPane(jp,
        ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS,
        ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS);
```

```
// OR
JScrollPane scrollPane = new JScrollPane(jp);

scrollPane.setVerticalScrollBarPolicy(
           ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);

scrollPane.setHorizontalScrollBarPolicy(
           ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS);
```

Examples

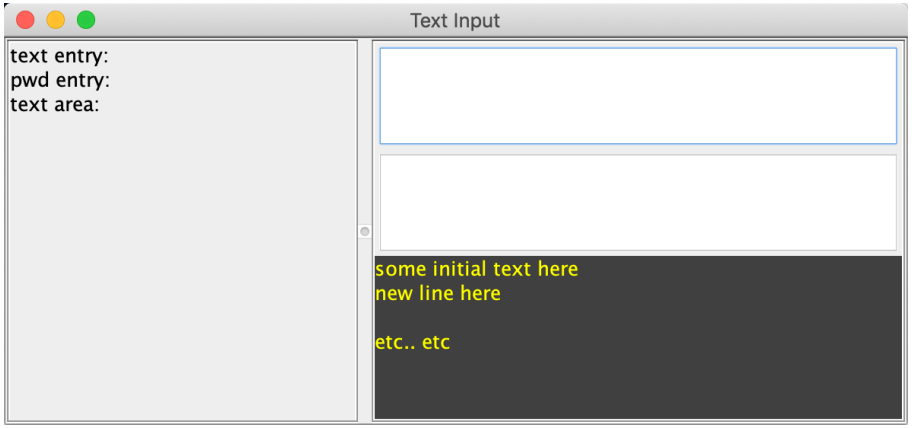
```
JTextArea textArea = new JTextArea("some initial text here\nnew line here\n\netc.. etc");
textArea.setBackground(Color.DARK_GRAY);
textArea.setForeground(Color.YELLOW);
textArea.setBounds(120, 170, 300, 100);

JScrollPane scrollPane = new JScrollPane(textArea);
scrollPane.setBounds(120, 170, 300, 100);


Container pane = this.getContentPane();

pane.add(tfLabel);
pane.add(textField);
pane.add(pfLabel);
pane.add(passwordField);
pane.add(taLabel);
pane.add(scrollPane);

this.setContentPane(pane);
```





Previous example for TextArea (no layout manager so scrollPane needs to be positioned with setBounds method)

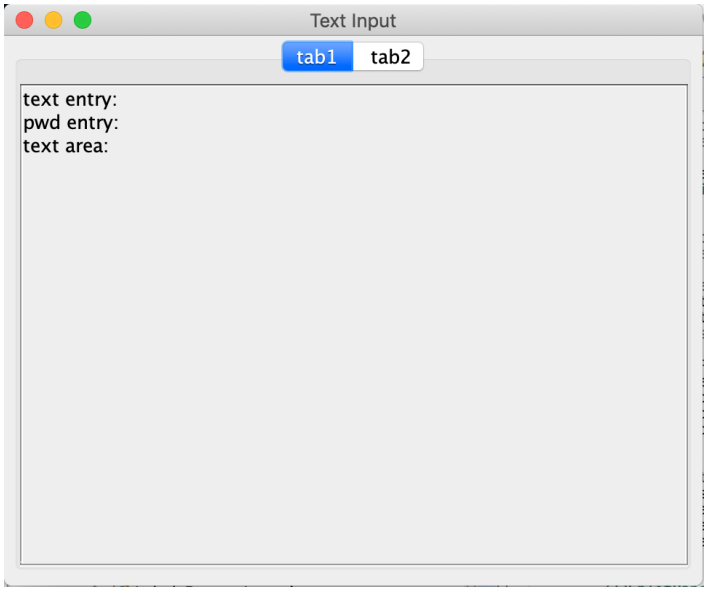
JSplitPane

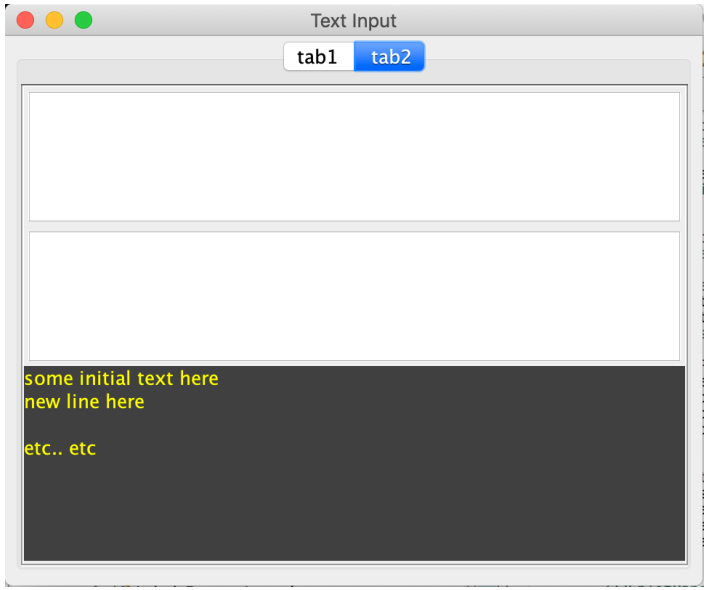



```
JPanel leftPanel = new JPanel();
leftPanel.setLayout(new BoxLayout(leftPanel, BoxLayout.Y_AXIS));
leftPanel.add(tfLabel);
leftPanel.add(pfLabel);
leftPanel.add(taLabel);
JScrollPane scrollPaneLeft = new JScrollPane(leftPanel);

JPanel rightPanel = new JPanel();
rightPanel.setLayout(new BoxLayout(rightPanel, BoxLayout.Y_AXIS));
rightPanel.add(textField);
rightPanel.add(passwordField);
rightPanel.add(textArea);
JScrollPane scrollPaneRight = new JScrollPane(rightPanel);

JSplitPane split = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
                                  scrollPaneLeft, scrollPaneRight);
```

YORK UNIVERSITÉ UNIVERSITY

JTabbedPane






```
JTabbedPane tabbedPane = new JTabbedPane();
tabbedPane.add("tab1", scrollPaneLeft);
tabbedPane.add("tab2", scrollPaneRight);

Container pane = this.getContentPane();

pane.add(tabbedPane,BorderLayout.CENTER);

this.setContentPane(pane);
```

YORK UNIVERSITÉ UNIVERSITY

More Documentation on Swing Components?

- How to tutorials (very useful):

  https://docs.oracle.com/javase/tutorial/uiswing/components/index.html

→ Next week: Events and Event Handling!