CSE2101: Object Oriented Programming-II (Java)

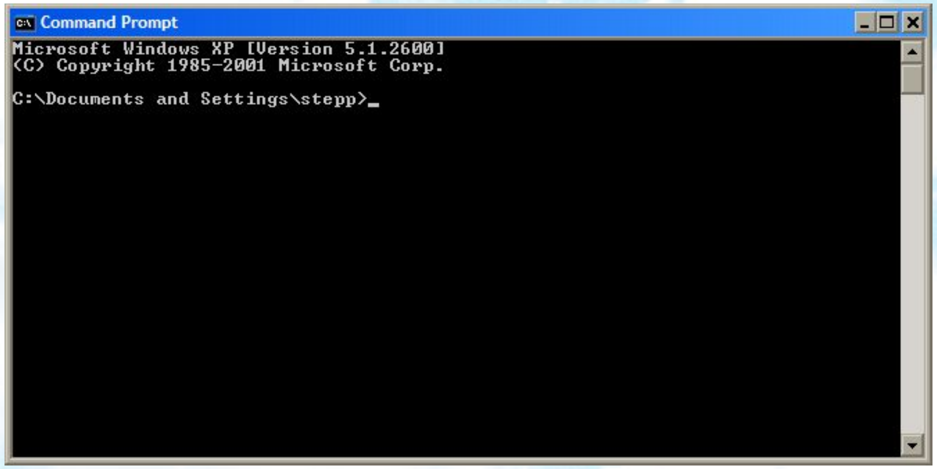
Lecture 10

Java Swing

Lecture 10

Motivation

- Learn how to create a graphical user interface (GUI) using Java and Swing
- Understand Java graphical component hierarchy
- Learn Java's event-driven model
- Banish the evil black box to the fiery depths from which it came



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\stepp>_
```

Java GUI: AWT and Swing

- Sun's initial idea: create a set of classes/methods that can be used to write a multi-platform GUI (Abstract Windowing Toolkit, or AWT)
  - problem: not powerful enough; limited; a bit clumsy to use

- Second edition (JDK v1.2): **Swing**
  - a newer library written from the ground up that allows much more powerful graphics and GUI construction

- Drawback: Both exist in Java now; easy to get them mixed up; still have to use both sometimes!

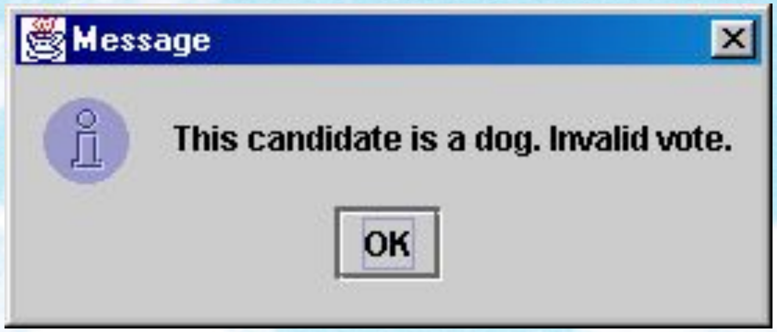
Swing and AWT components - a quick Reminder

- Mix Swing and AWT components as little as possible (not at all in most cases)

- Put ‘J’ in front of everything AWT provides to get Swing’s counterpart
  - AWT: Button
  - Swing: JButton

Simplest GUI programming: JOptionPane

An option pane is a simple dialog box for graphical input/output

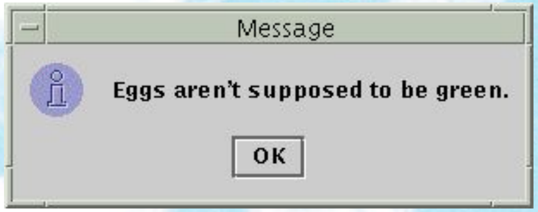


This candidate is a dog. Invalid vote.

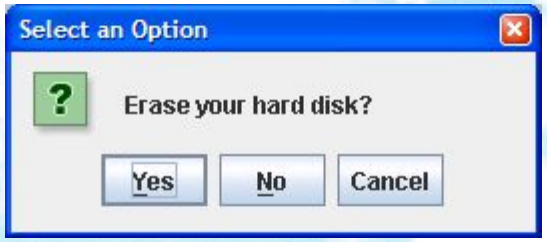
- advantages:
  - simple
  - flexible (in some ways)
  - looks better than the black box of doom
- disadvantages:
  - created with static methods;
    not very object-oriented
  - not very powerful (just simple dialog boxes)

Types of JOptionPane

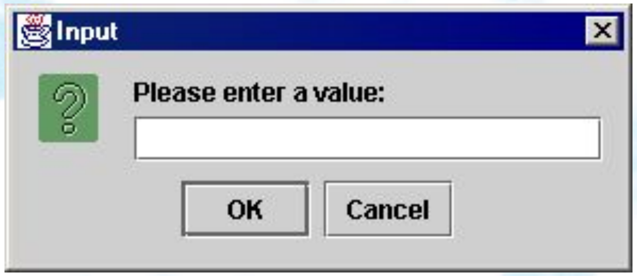
- public static void showMessageDialog(Component parent, Object message) Displays a message on a dialog with an OK button.



- public static int showConfirmDialog(Component parent, Object message) Displays a message and list of choices Yes, No, Cancel



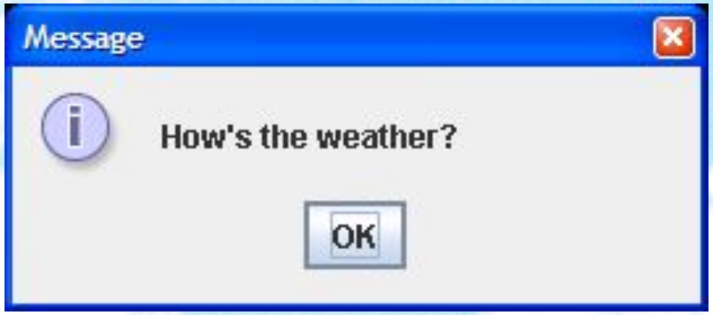
- public static String showInputDialog(Component parent, Object message) Displays a message and text field for input, and returns the value entered as a String.



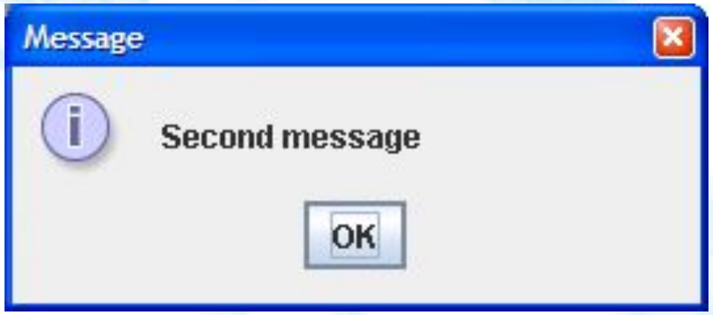
JOptionPane examples 1

- showMessageDialog analogous to System.out.println for displaying a simple message

```
import javax.swing.*;

class MessageDialogExample {
  public static void main(String[] args) {
    JOptionPane.showMessageDialog(null,
        "How's the weather?");
    JOptionPane.showMessageDialog(null,
        "Second message");
  }
}
```



How's the weather?

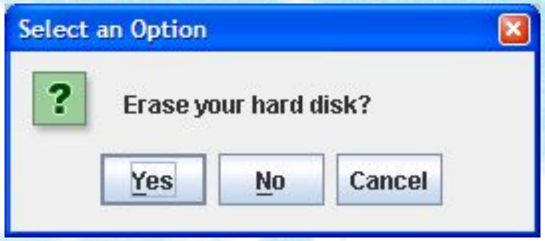


Second message

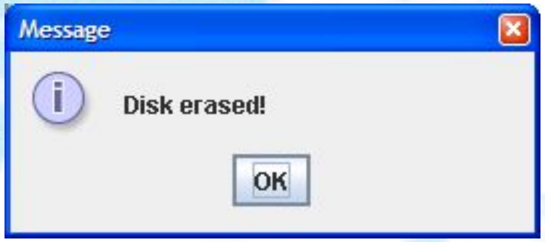
JOptionPane examples 2

- showConfirmDialog analogous to a System.out.print that prints a question, then reading an input value from the user (can only be one of the provided choices)

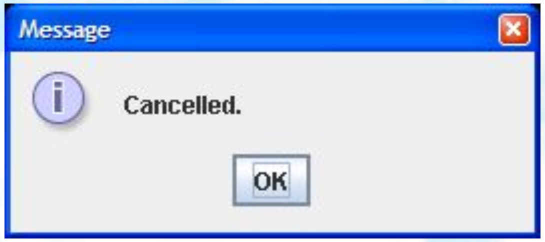
```
import javax.swing.*;

class ConfirmDialogExample {
 public static void main(String[] args) {
  int choice = JOptionPane.showConfirmDialog(null,
               "Erase your hard disk?");
  if (choice == JOptionPane.YES_OPTION) {
   JOptionPane.showMessageDialog(null, "Disk erased!");
  } else {
   JOptionPane.showMessageDialog(null, "Cancelled.");
  }
 }
}
```


Select an Option

Erase your hard disk?


Message

Disk erased!

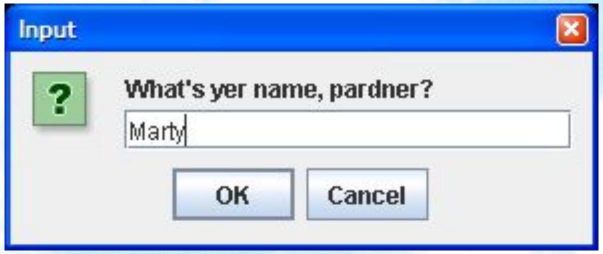

Message

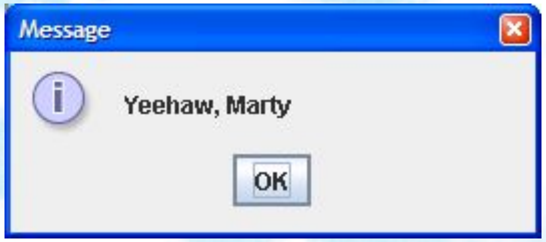
Cancelled.

JOptionPane examples 3

- showInputDialog analogous to a System.out.print that prints a question, then reading an input value from the user (can be any value)

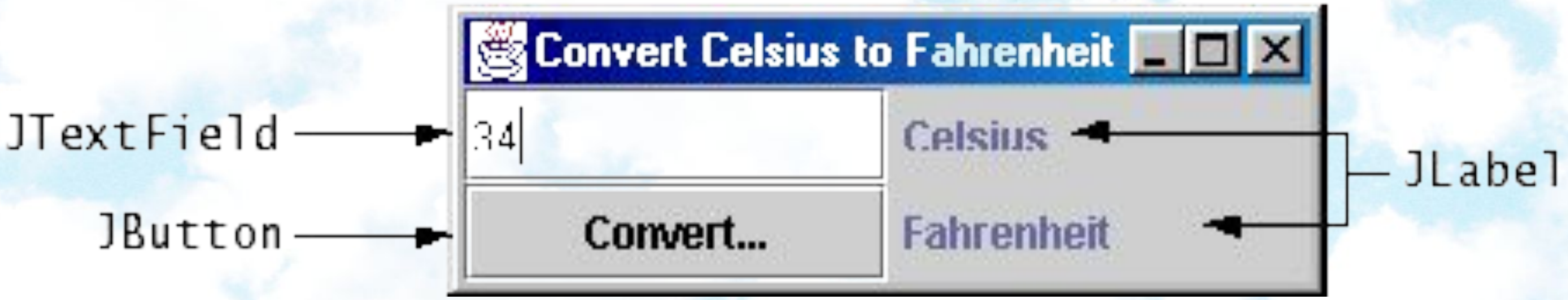
```
import javax.swing.*;

class InputDialogExample {
 public static void main(String[] args) {
   String name = JOptionPane.showInputDialog(null,
                "What's yer name, pardner?");
   JOptionPane.showMessageDialog(null, "Yeehaw, " + name);
 }
}
```





Onscreen GUI elements

- **windows**: actual first-class citizens of desktop; also called top-level containers
  examples: frame, dialog box
- **components**: GUI widgets
  examples: button, text box, label
- **containers**: logical grouping for components
  example: panel



Swing Component Hierarchy

```
java.lang.Object
 +--java.awt.Component
      +--java.awt.Container
            |
            +--javax.swing.JComponent
            |      +--javax.swing.JButton
            |      +--javax.swing.JLabel
            |      +--javax.swing.JMenuBar
            |      +--javax.swing.JOptionPane
            |      +--javax.swing.JPanel
            |      +--javax.swing.JTextArea
            |      +--javax.swing.JTextField
            |
            +--java.awt.Window
                  +--java.awt.Frame
                        +--javax.swing.JFrame
```

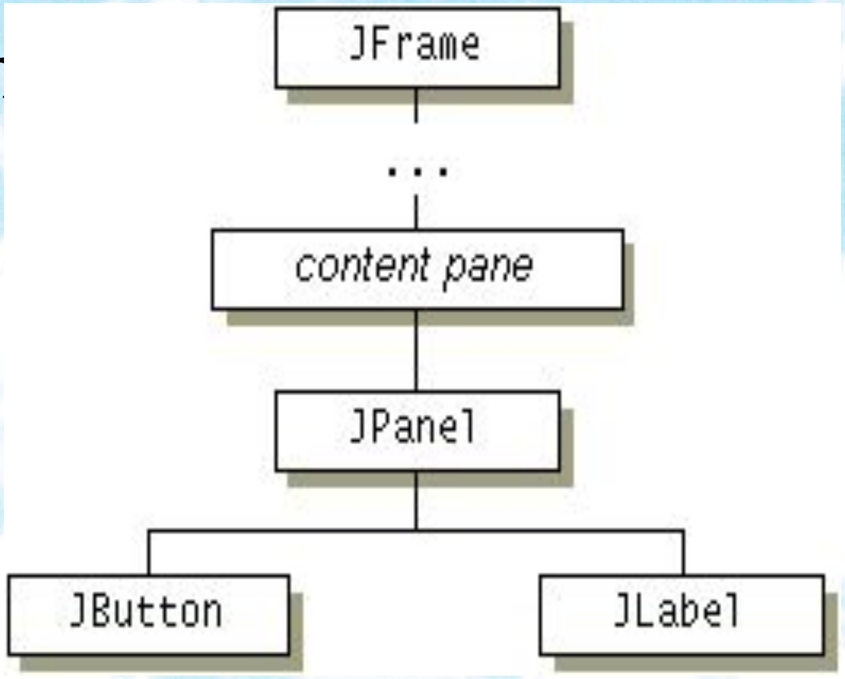
- **import java.awt.*;**
  import javax.swing.*;

Components and Containers

- Components
  - The building blocks
  - Variety of uses and complexities
- Containers
  - The cement
  - Hierarchical organisation
  - Distinction is not always drawn

Containment hierarchies

- Top level containers
  - Intermediate contain
    - Atomic components



Top-level containers

- At the root of every containment hierarchy
- All Swing programs have at least one
- Content panes
- Types of top-level containers
  - Frames
  - Dialogs
  - Applets

Top Level Container (Frame)

- //this won’t compile…
- public static void main(String[] args)
- {
- JFrame frame = new JFrame(“A JFrame"); //Just like any
  //other class
- // do things with frame
- frame.setJMenuBar(menuBar);
- frame.setContentPane(contentPane);
- frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

- // set frame size
- frame.pack();
-
- // realize frame
- frame.setVisible(true);
- } // end main

Dialog boxes

- More limited than frames
- Modality
- Types of dialogs
  - JOptionPane
  - ProgressMonitor
  - JColorChooser
  - JDialog

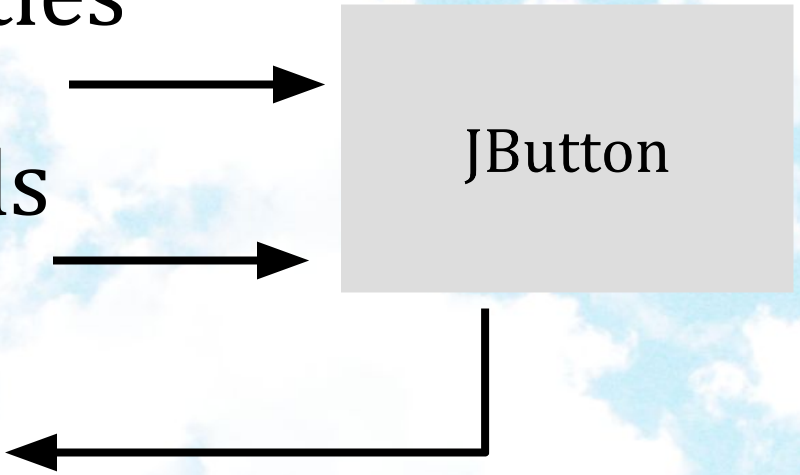
GUI Component API

- Java: GUI component = class

- Properties

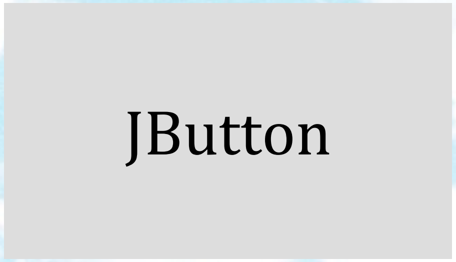
- Methods

- Events



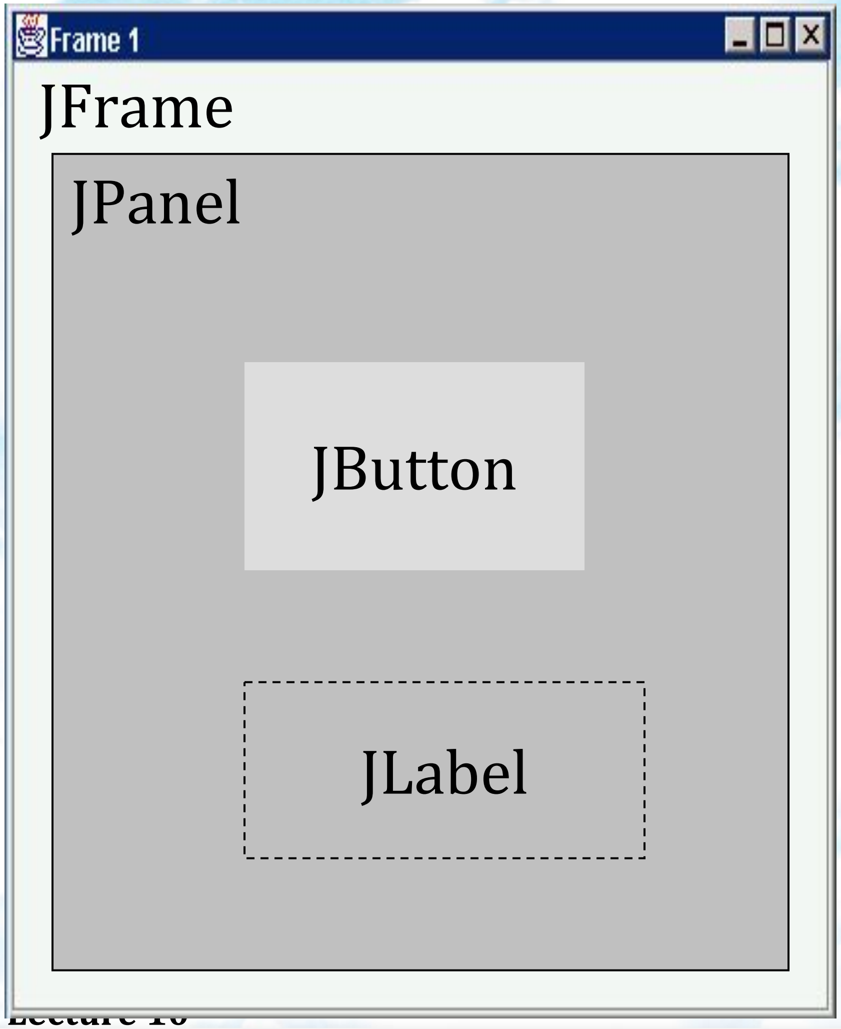
Using a GUI Component

1. Create it
   - Instantiate object: b = new JButton(“press me”);
2. Configure it
   - Properties: b.text = “press me”; [avoided in java]
   - Methods: b.setText(“press me”);
3. Add it
   - panel.add(b);
4. Listen to it
   - Events: Listeners

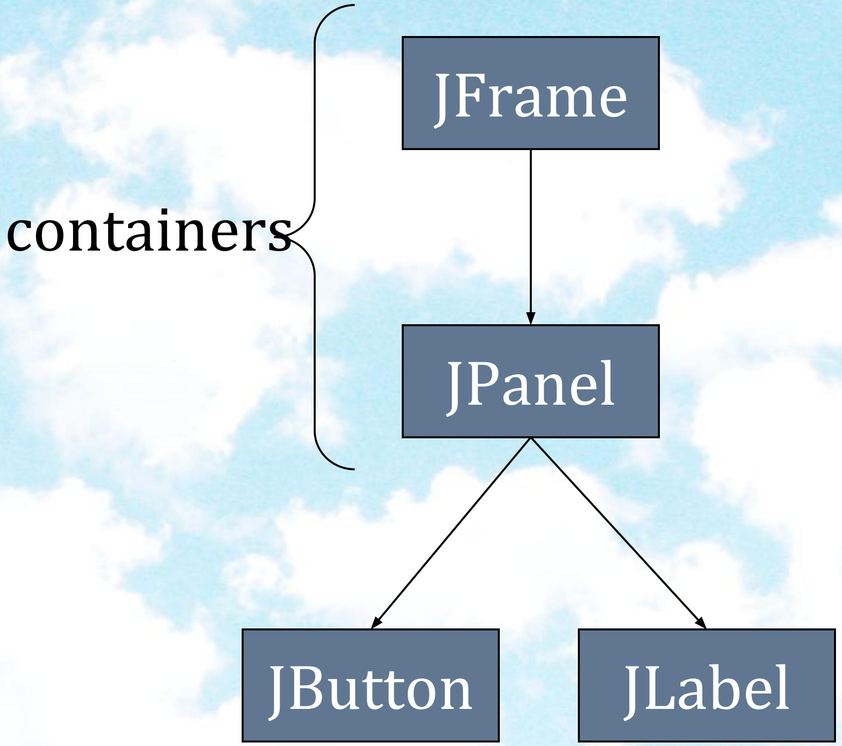


Anatomy of an Application GUI

GUI



Internal structure



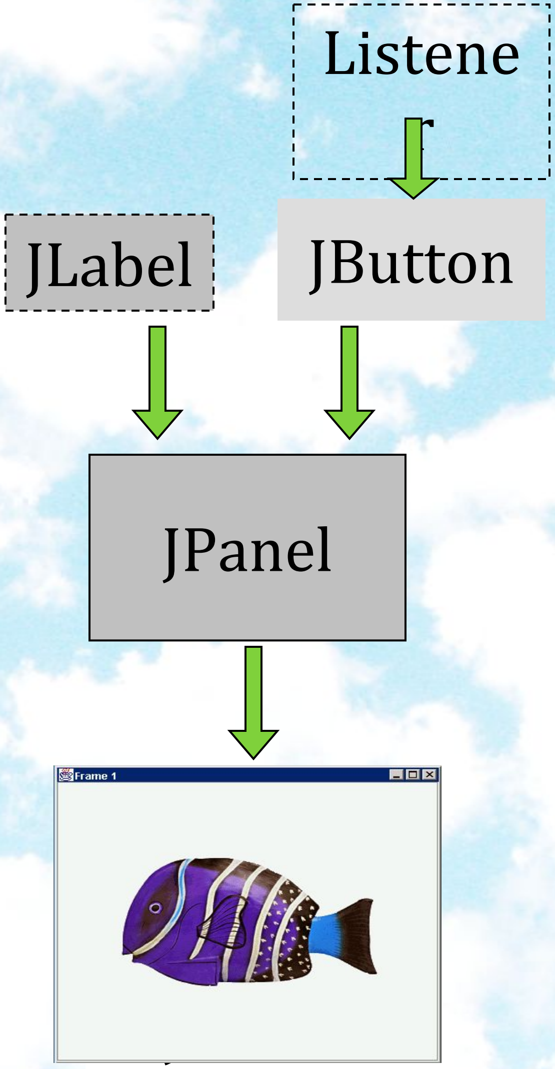
Using a GUI Component 2

1. Create it
2. Configure it
3. Add children (if container)
4. Add to parent (if not JFrame)
5. Listen to it

order important

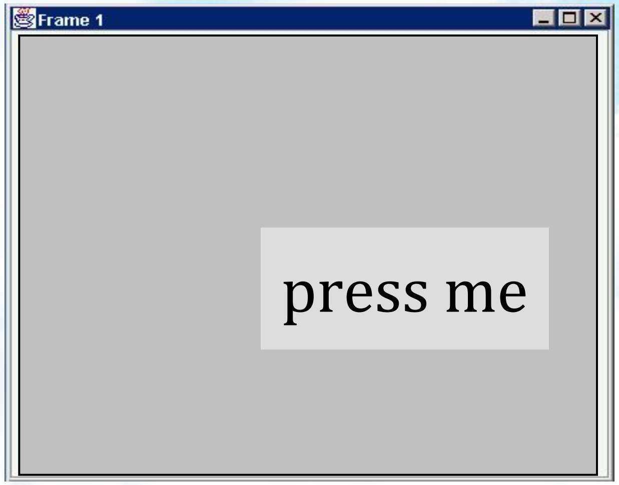
Build from bottom up

- Create:
    - Frame
    - Panel
    - Components
    - Listeners
- Add: (bottom up)
    - listeners into components
    - components into panel
    - panel into frame



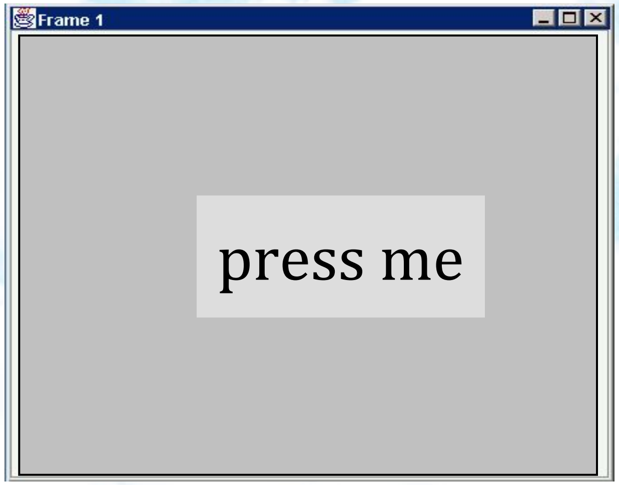
Code

```
JFrame f = new JFrame("title");
JPanel p = new JPanel( );
JButton b = new JButton("press me");

p.add(b);                    // add button to panel
f.setContentPane(p);   // add panel to frame

f.show();
```



Application Code

```
import javax.swing.*;

class hello {
     public static void main(String[] args){
       JFrame f = new JFrame(“title”);
       JPanel p = new JPanel();
       JButton b = new JButton(“press me”);
       p.add(b);                    // add button to panel
       f.setContentPane(p);    // add panel to frame

       f.setvisible(true);
     }
}
```



Methods of all Swing components

- public int getWidth()
  public int getHeight()
  Allow access to the component's current width and height in pixels.

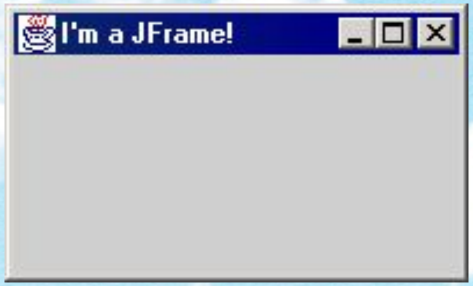
- public boolean isEnabled()
  Returns whether the component is enabled (can be interacted with).

- public boolean isVisible()
  Returns whether the component is visible (can be seen on the screen).

More JComponent methods

- `public void setBackground(Color c)`
  Sets the background color of the component to be the given color.
- `public void setFont(Font f)`
  Sets the font of the text on the given component to be the given font.
- `public void setEnabled(boolean b)`
  Sets whether the component is enabled (can be interacted with).
- `public void setVisible(boolean b)`
  Sets whether the component is visible (can be seen on the screen). Set to `true` to show the component, or set to `false` to hide the component.

JFrame



A frame is a graphical window that can be used to hold other components

- `public JFrame()` or `public JFrame(String title)`
  Creates a frame with an optional title.

- `public void setTitle(String text)`
  Puts the given text in the frame's title bar.

- `public void setDefaultCloseOperation(int op)`
  Makes the frame perform the given action when it closes. Common value: `JFrame.EXIT_ON_CLOSE`

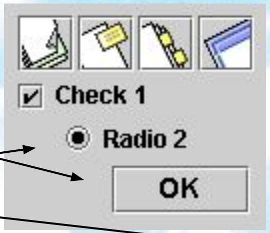
- `public void add(Component comp)`
  Places the given component or container inside the frame.
  - *How would we add more than one component to the frame?*

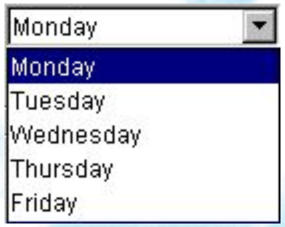
- `public void pack()`
  Resizes the frame to fit the components inside it.

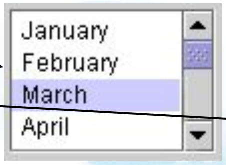
- *NOTE*: Call `setVisible(true)` to make a frame appear on the screen after creating it.

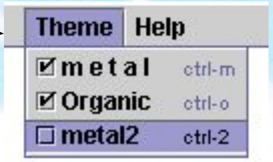
Atomic components (1)

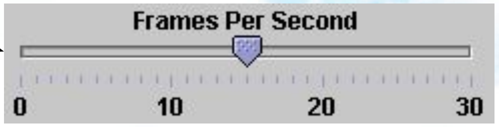
- Buttons
- Combo boxes
- Lists
- Menus
- Sliders
- Text Fields
- Labels

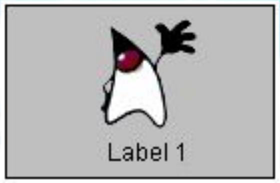


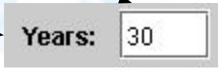






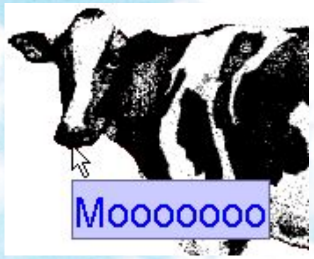


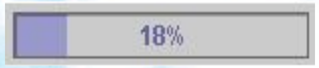


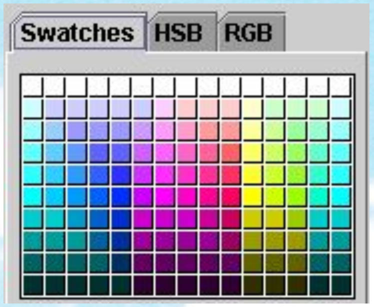


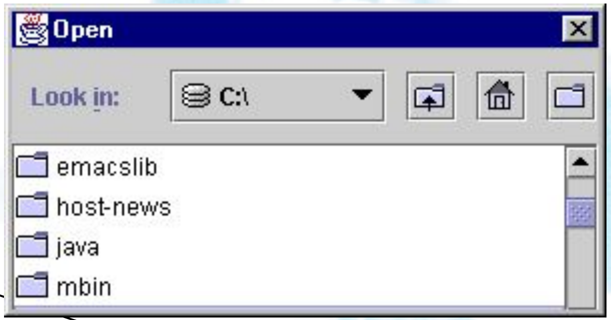
Atomic Components (2)

- Tool tips
- Progress bars
- Colour choosers
- File choosers
- Tables
- Text
- Trees

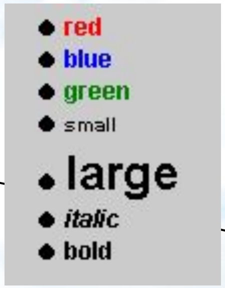




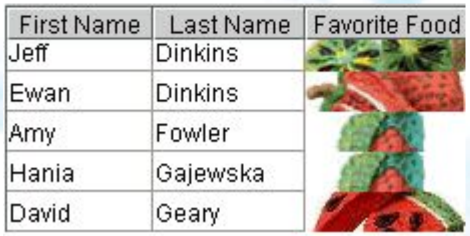




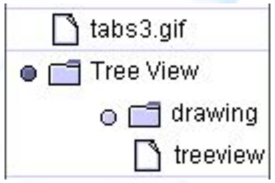
emacslib
host-news
java
mbin



- red
- blue
- green
- small
- large
- *italic*
- **bold**



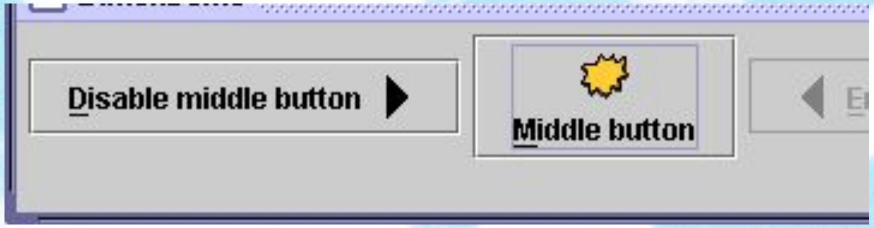
First Name	Last Name	Favorite Food
Jeff	Dinkins	
Ewan	Dinkins	
Amy	Fowler	
Hania	Gajewska	
David	Geary	



tabs3.gif
Tree View
drawing
treeview

JButton, JLabel

The most common component— a button is a clickable onscreen region that the user interacts with to perform a single command



A text label is simply a string of text displayed on screen in a graphical program. Labels often give infor-mation or describe other components

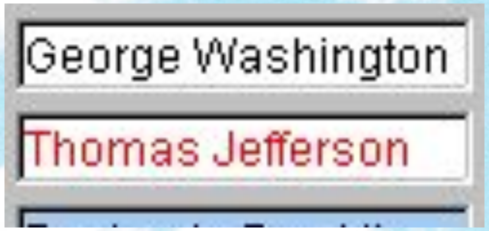
- `public JButton(String text)`
  `public JLabel(String text)`
  Creates a new button / label with the given string as its text.

- `public String getText()`
  Returns the text showing on the button / label.

- `public void setText(String text)`
  Sets button / label's text to be the given string.

JTextField, JTextArea

A text field is like a label, except that the text in it can be edited and modified by the user. Text fields are commonly used for user input, where the user types information in the field and the program reads it


George Washington

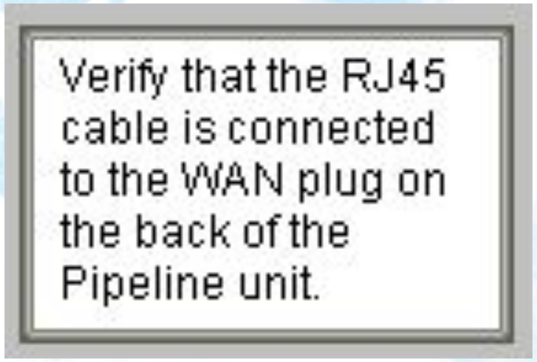
Thomas Jefferson

A text area is a multi-line text field

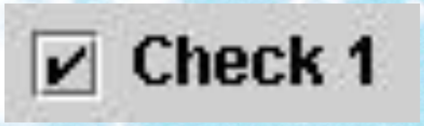
- `public JTextField(int columns)`
- `public JTextArea(int lines, int columns)`
  Creates a new text field that is the given number of columns (letters) wide.

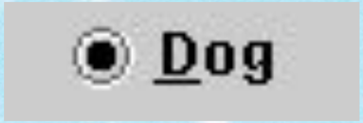
- `public String getText()`
  Returns the text currently in the field.

- `public void setText(String text)`
  Sets field's text to be the given string.


Verify that the RJ45 cable is connected to the WAN plug on the back of the Pipeline unit.

JCheckBox, JRadioButton





A check box is a toggleable button with two states: checked and unchecked

A radio button is a button that can be selected; usually part of a group of mutually-exclusive radio buttons (1 selectable at a time)

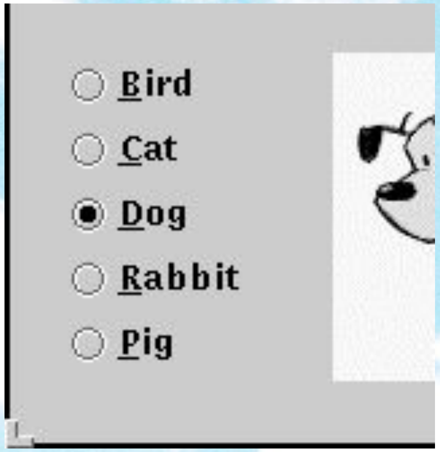
- `public JCheckBox / JRadioButton(String text)`
  `public JCheckBox(String text, boolean isChecked)`
  Creates checked/unchecked check box with given text.

- `public boolean isSelected()`
  Returns true if check box is checked.

- `public void setSelected(boolean selected)`
  Sets box to be checked/unchecked.

ButtonGroup

A logical group of radio buttons that ensures that only one is selected at a time



- `public ButtonGroup()`
- `public void add(JRadioButton button)`

- The `ButtonGroup` is not a graphical component, just a logical group; the `RadioButton`s themselves are added to the container, not the `ButtonGroup`

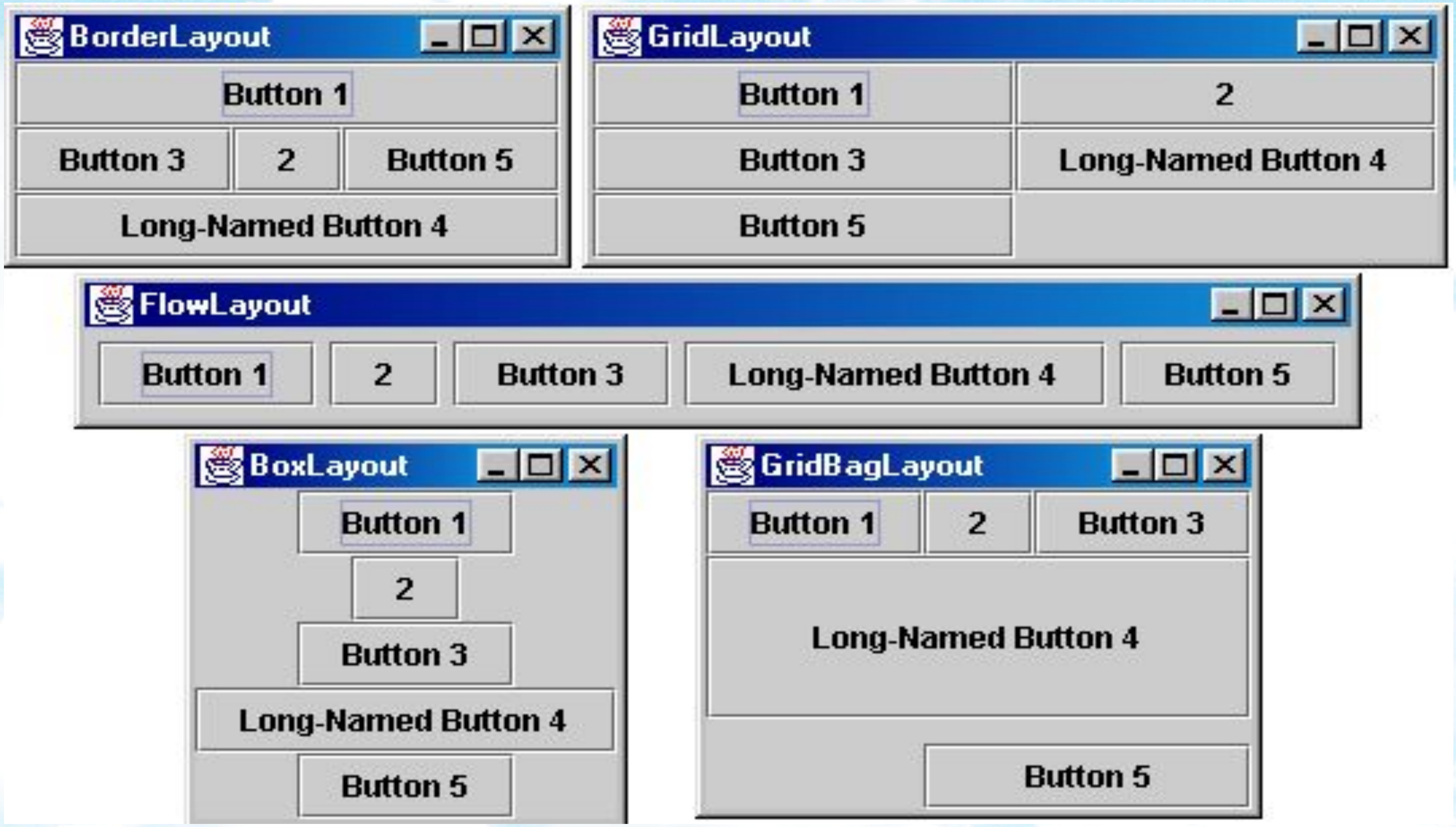
Problem: positioning, resizing

How does the programmer specify where each component sits in the window, how big each component should be, and what the component should do if the window is resized/moved/maximized/etc?

- **Absolute positioning** (C++, C#, others): Specify exact pixel coordinates for every component
- ***Layout managers*** (Java): Have special objects that decide where to position each component based on some criteria

Containers with layout

- The idea: Place many components into a container(s), then add the container(s) to the JFrame



Container

A container is an object that holds components; it also governs their positions, sizes, and resize behavior

- `public void add(Component comp)`
  `public void add(Component comp, Object info)`
  Adds a component to the container, possibly giving extra information about where to place it.

- `public void remove(Component comp)`
  Removes the given component from the container.

- `public void setLayout(LayoutManager mgr)`
  Uses the given layout manager to position the components in the container.

- `public void validate()`
  You should call this if you change the contents of a container that is already on the screen, to make it re-do its layout.

`JPanel`

A panel is our container of choice; it inherits the methods from the previous slide and defines these additional methods (among others):

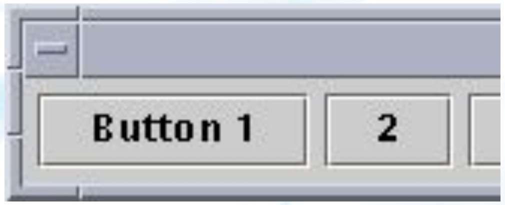
- `public JPanel()`
  Constructs a panel with a default flow layout.

- `public JPanel(LayoutManager mgr)`
  Constructs a panel that uses the given layout manager.

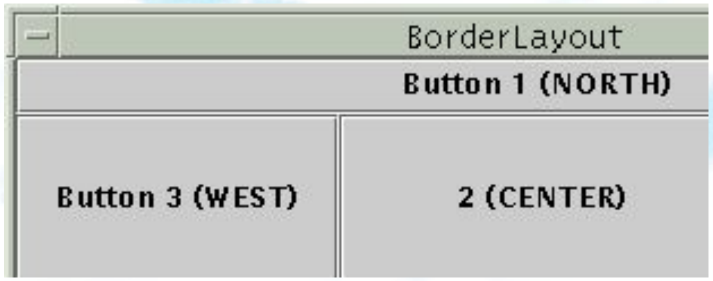
Preferred size of components

- Swing component objects each have a certain size they would "like" to be--just large enough to fit their contents (text, icons, etc.)
- This is called the *preferred size* of the component
- Some types of layout managers (e.g. `FlowLayout`) choose to size the components inside them to the preferred size; others (e.g. `BorderLayout`, `GridLayout`) disregard the preferred size and use some other scheme

Buttons at preferred size:



Not preferred size:

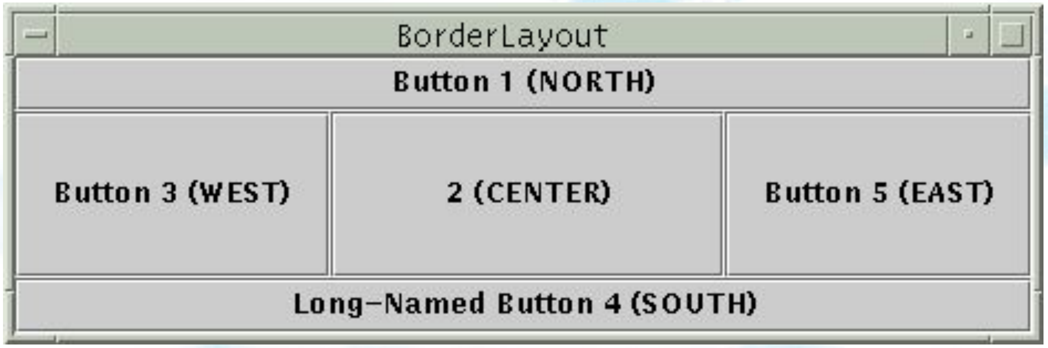


BorderLayout

```
public BorderLayout()
```

- divides container into five regions: `NORTH`, `SOUTH`, `WEST`, `EAST`, `CENTER`
- `NORTH` and `SOUTH` regions expand to fill region horizontally, and use preferred size vertically
- `WEST` and `EAST` regions expand to fill region vertically, and use preferred size horizontally
- `CENTER` uses all space not occupied by others

```
Container panel = new JPanel(new BorderLayout());
panel.add(new JButton("Button 1 (NORTH)", BorderLayout.NORTH);
```

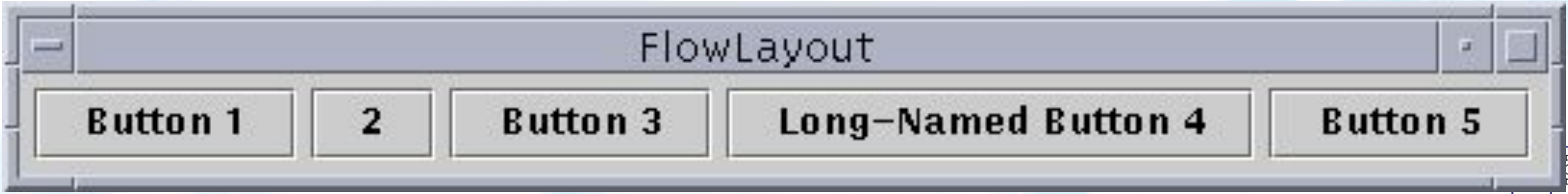


FlowLayout

```
public FlowLayout()
```

- treats container as a left-to-right, top-to-bottom "page" or "paragraph"
- components are given their preferred size both horizontally and vertically
- components are positioned in order added
- if too long, components wrap around to next line

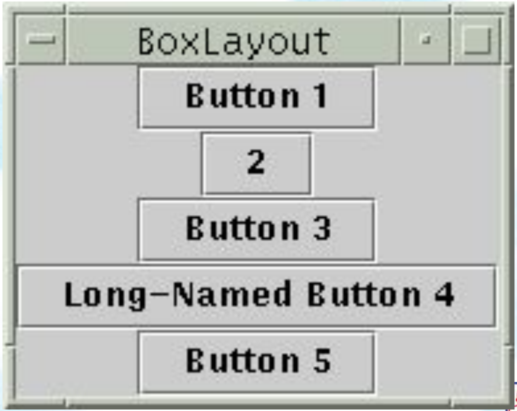
```
Container panel = new JPanel(new FlowLayout());
panel.add(new JButton("Button 1"));
```



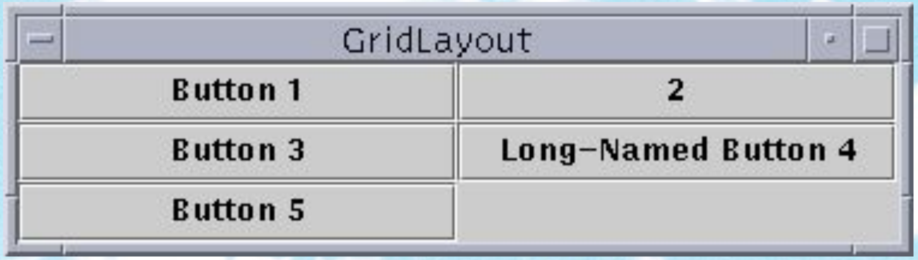
BoxLayout

```
Box.createHorizontalBox()
Box.createVerticalBox()
```

- aligns components in container in a single row or column
- components use preferred sizes and align based on their preferred alignment
- preferred way to construct a container with box layout: `Box.createHorizontalBox();` or `Box.createVerticalBox();`



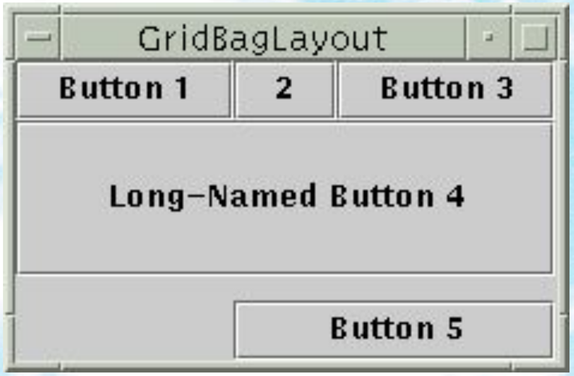
GridLayout



- Grid of cells - all same size
- Components take all space in a cell
- Gaps
  - default = 5
  - use setter methods hGap and vGap
  - or via arguments to constructor
- Re-sizing
  - Cells resize to be as large as possible in given window / container

GridBagLayout (1)

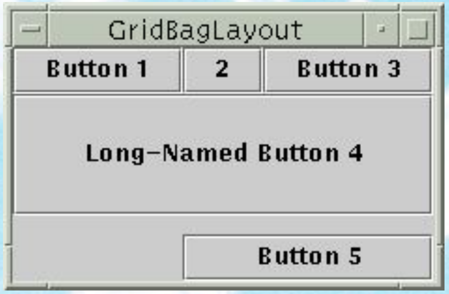
- Very flexible (and complex!)
- Rows can have different heights
- Columns can have different lengths
- Uses cells in a grid



```
GridBagLayout gridbag = new GridBagLayout();
GridBagConstraints c = new GridBagConstraints();

JPanel pane = new JPanel();
pane.setLayout(gridbag);

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

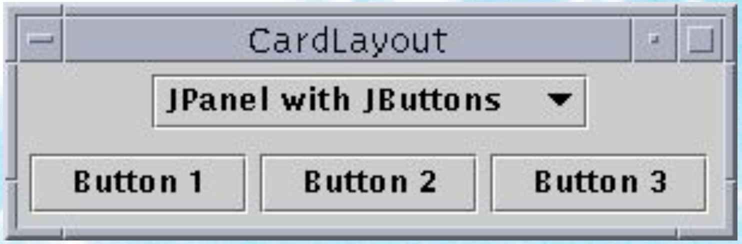
GridBagLayout (2)

- Constraints
  - set in an instance of a gridBagConstraints Object
  - *gridx and gridy* - The row and column of the upper left of the component
  - *Anchor* - Where to display within cell when component is smaller than it
  - *fill* - How to size component when cell is larger than components requested size
  - *insets* - External padding - min space between component and cell edges
  - *ipadx, ipady* - Internal padding - What to add to min size of components
  - *weightx and weighty* - How to distribute extra space (padding)
  - *gridwidth and gridheight* - Number of columns or rows the component uses



- Example explained very well here:
  - http://java.sun.com/docs/books/tutorial/uiswing/layout/gridbagExample.html

CardLayout



- Manages objects (usually JPanels) in sets
- Works much like tabbed pane
- Choose cards by
  - Asking for card in order added to container
  - Going backwards or forwards
  - Specifying card by name

Choosing Layout Managers (1)

- In order to display a component in as much space as it can get, consider:
  - BorderLayout
    - Component in centre
  - GridBagLayout
    - fill=GridBagConstraints.BOTH
  - BoxLayout
    - Component specifies very large preferred/maximum sizes

Choosing Layout Managers (2)

- To display a few components in a compact row:
  - JPanel’s default FlowLayout
  - BoxLayout

- Display a few components of the same size in rows and columns
  - GridLayout

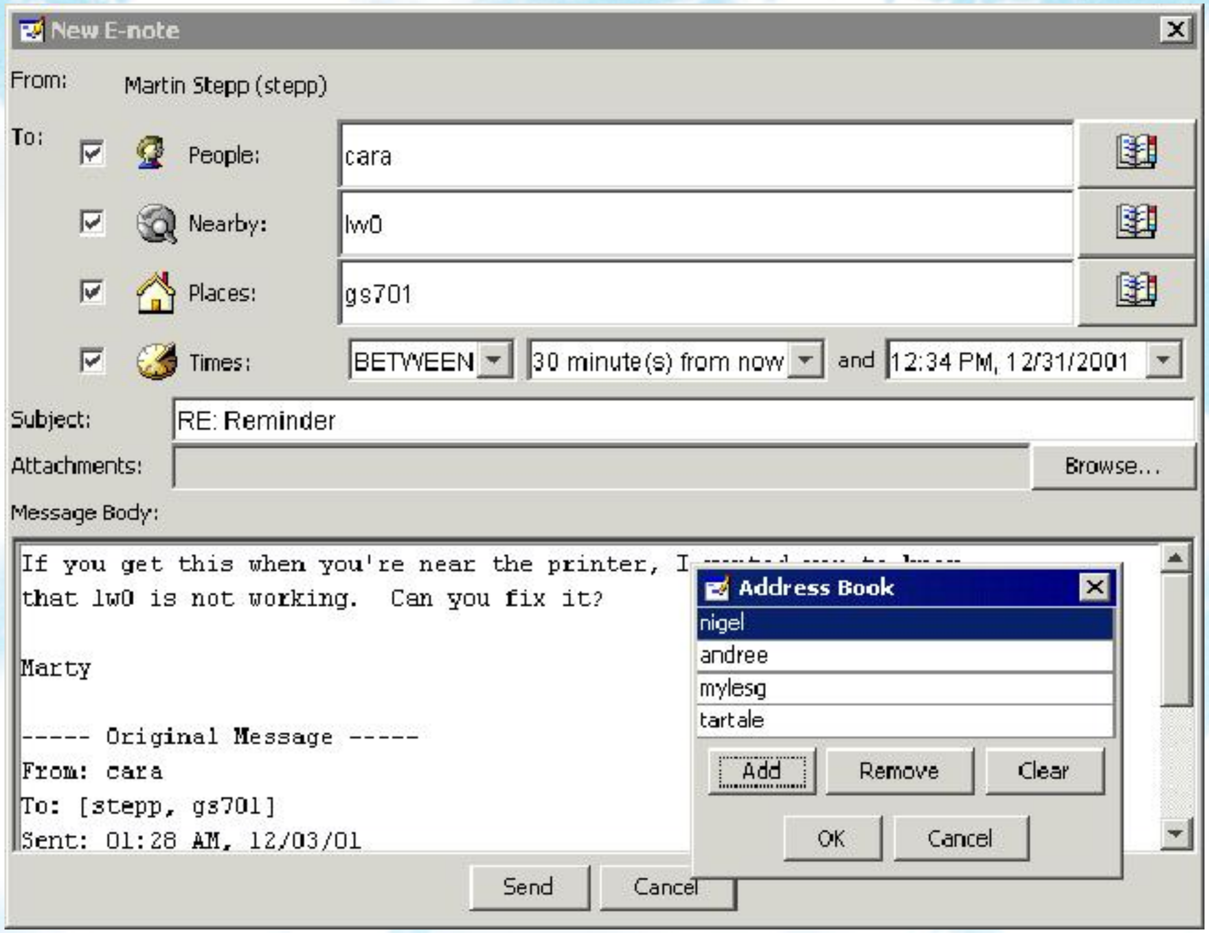
Choosing layout managers (3)

- Display a few components in a row or column, with different spacing between them and custom component sizes
  - BoxLayout

- Display a complex layout that has many components
  - GridBagLayout
  - *Using JPanel grouping and hierarchies*

Problem with Layout Managers

How would you create a complex window like this, using the layout managers shown?



New E-note

From: Martin Stepp (stepp)

To:

✓ People:	cara	
✓ Nearby:	lw0	
✓ Places:	gs701	
✓ Times:	BETWEEN · 30 minute(s) from now · and · 12:34 PM, 12/31/2001	

Subject: RE: Reminder

Attachments: Browse...

Message Body:

```
If you get this when you're near the printer, I
that lw0 is not working.  Can you fix it?

Marty

----- Original Message -----
From: cara
To: [stepp, gs701]
Sent: 01:28 AM, 12/03/01
```

Send · Cancel

Address Book

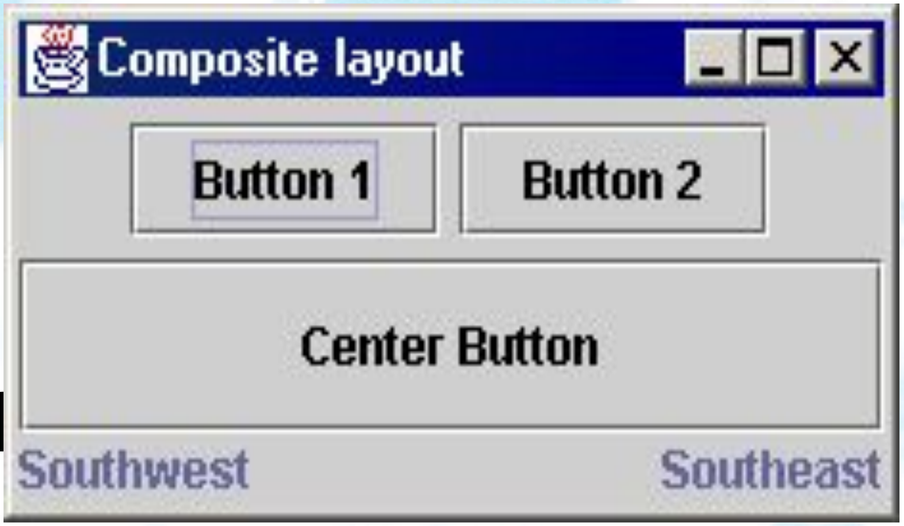
- nigel
- andree
- mylesg
- tartale

Add · Remove · Clear

OK · Cancel

Solution: composite layout

- create panels within panels
- each panel has a different layout, and by combining the layouts, more complex / powerful layout can be achieved
- example:
  - how many panels?
  - what layout in eacl



Event-driven Programming

- program's execution is indeterminate
- on-screen components cause *events* to occur when they are clicked / interacted with
- events can be handled, causing the program to respond, *driving* the execution thru events (an "event-driven" program)

Java Event Hierarchy

```
java.lang.Object
  +--java.util.EventObject
        +--java.awt.AWTEvent
              +--java.awt.event.ActionEvent
              +--java.awt.event.TextEvent
              +--java.awt.event.ComponentEvent
                    +--java.awt.event.FocusEvent
                    +--java.awt.event.WindowEvent
                    +--java.awt.event.InputEvent
                          +--java.awt.event.KeyEvent

  +--java.awt.event.MouseEvent
```

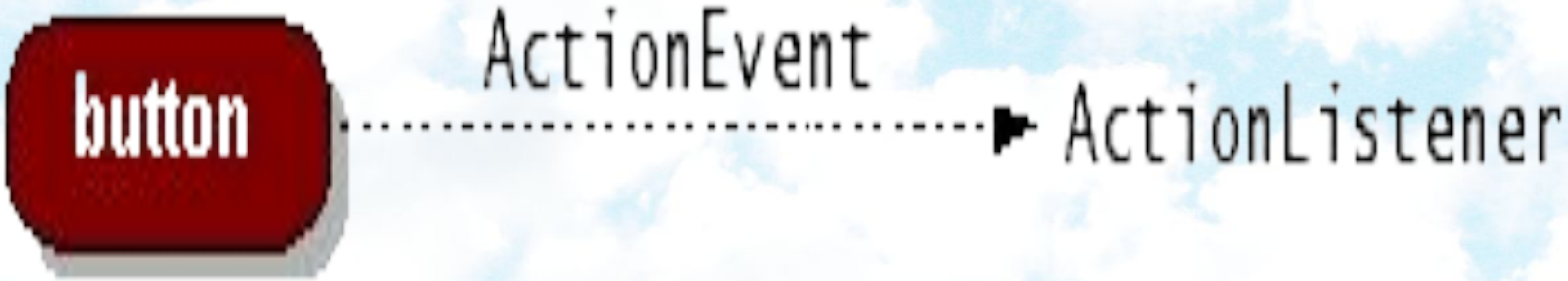
- **`import java.awt.event.*;`**

Some typical component events and listeners

Act that results in event	**Listener**
User clicks a button, presses return while typing in a text field, or chooses a menu item	ActionListener
User closes a window	WindowListener
User presses a mouse button while the cursor is over a component	MouseListener
User moves the mouse over a component	MouseMotionListener
Component becomes visible	ComponentListener
Component gets the keyboard focus	FocusListener
Table or list selection changes	ListSelectionListener

Listening for Events

- attach *listener* to component
- listener's appropriate method will be called when event occurs (e.g. when the button is clicked)
- for Action events, use `ActionListener`



Writing an ActionListener

```
// part of Java; you don’t write this
public interface ActionListener {
  void actionPerformed(ActionEvent event);
}

// Prints a message when the button is clicked.
public class MyActionListener
             implements ActionListener {
  public void actionPerformed(ActionEvent
  event){
    System.out.println("Event occurred!");
  }
}
```

Attaching an ActionListener

```
JButton button = new JButton("button 1");
ActionListener listener = new MyActionListener();
button.addActionListener(listener);
```

- now `button` will print `"Event occurred!"` when clicked
- `addActionListener` method exists in many Swing components

References

- The Java Tutorial: Visual Index to the Swing Components. https://docs.oracle.com/javase/tutorial/uiswing/components/index.html
- The Java Tutorial: Laying Out Components Within a Container. https://docs.oracle.com/javase/tutorial/uiswing/layout/index.html
- The Java Tutorial: How to Write Action Listeners. https://docs.oracle.com/javase/tutorial/uiswing/events/actionlistener.html

Thank you