Introduction to JFC Swing

Agenda

- Introduction
- About JFC and Swing
- Swing Components
- Borders
- Layout Management
- Events Handling

AWT to Swing

- AWT: Abstract Windowing Toolkit
  - import java.awt.*
- Swing: new with Java2
  - import javax.swing.*
  - Extends AWT

Getting started with Swing

- Swing, like the rest of the Java API is subdivided into packages:

- At the start of your code - always
  - **import javax.swing.*;**
  - **import javax.swing.event.*;**

- Most Swing programs also need
  - **import java.awt.*;**
  - **import java.awt.event.*;**

A typical Swing program

- Consists of multiple parts
  - Containers
  - Components
  - Events
  - Graphics
  - (Threads)

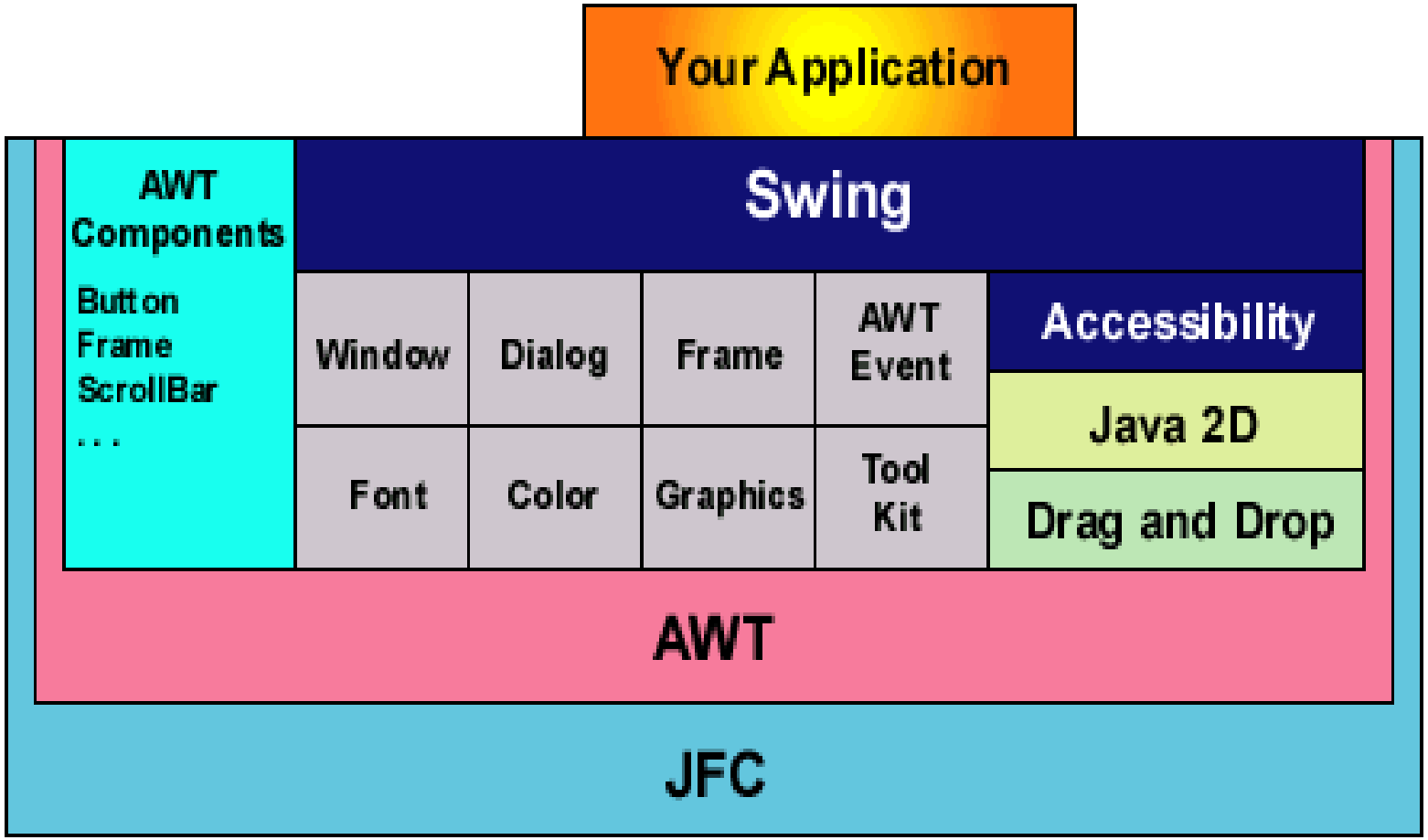
About JFC and Swing

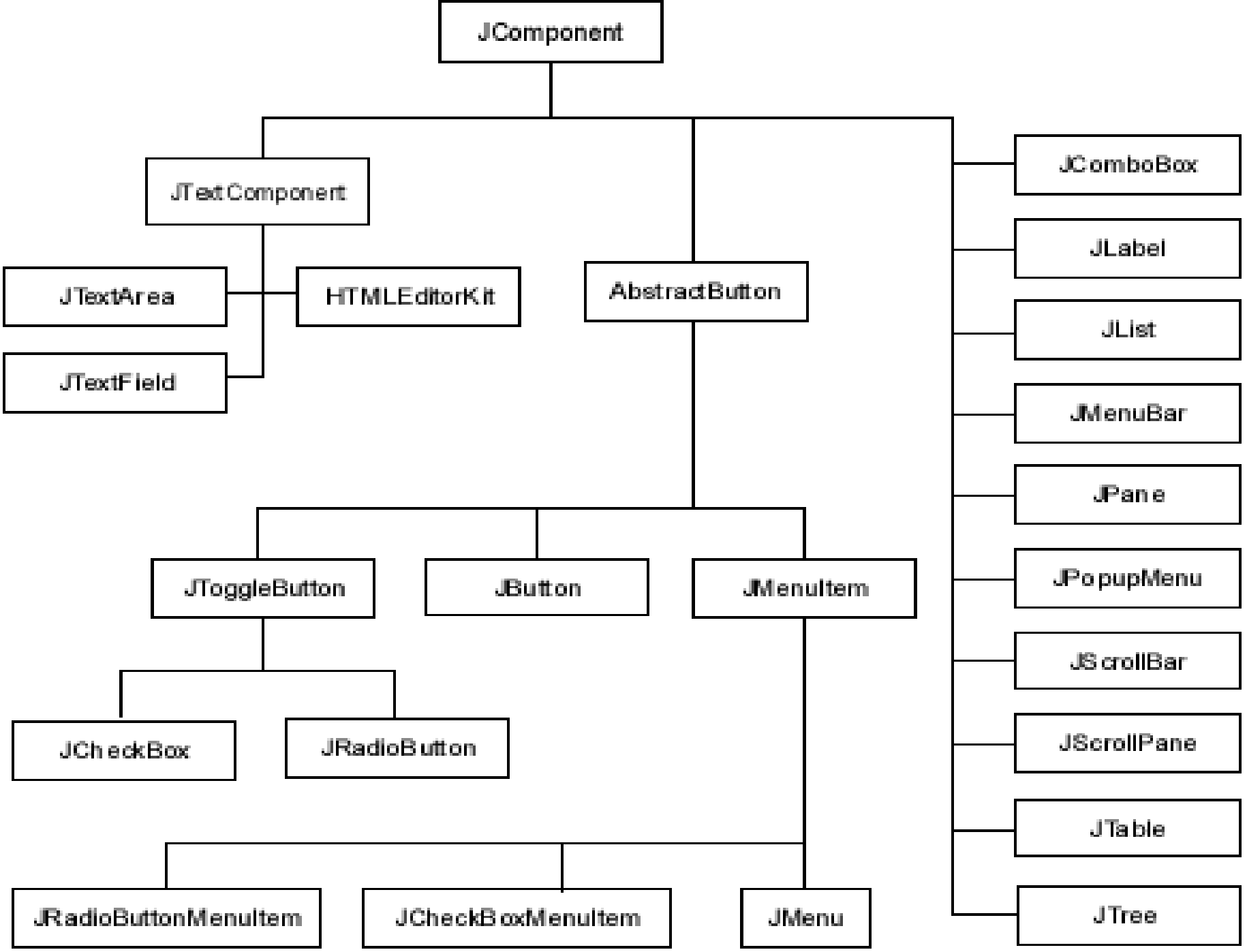
- JFC – Java™ Foundation Classes
- Encompass a group of features for constructing graphical user interfaces (GUI).
- Implemented without any native code.
- “Swing” is the codename of the project that developed the first JFC components .
- The name “Swing” is frequently used to refer to new components and related API.

Swing Components

- Swing provides many standard GUI **components** such as buttons, lists, menus, and text areas, which you combine to create your program's GUI.
- Swing provides **containers** such as windows and tool bars.
  - top level: frames, dialogs
  - intermediate level: panel, scroll pane, tabbed pane, ...

OTHER APIs

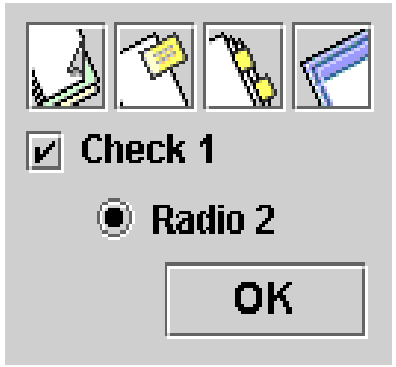

Your Application
AWT Components
Button
Frame
ScrollBar
. . .
Swing
Window
Dialog
Frame
AWT Event
Accessibility
Java 2D
Font
Color
Graphics
Tool Kit
Drag and Drop
AWT
JFC


JComponent
JTextComponent
JTextArea
HTMLEditorKit
JTextField
AbstractButton
JToggleButton
JButton
JMenuItem
JCheckBox
JRadioButton
JRadioButtonMenuItem
JCheckBoxMenuItem
JMenu
JComboBox
JLabel
JList
JMenuBar
JPane
JPopupMenu
JScrollBar
JScrollPane
JTable
JTree

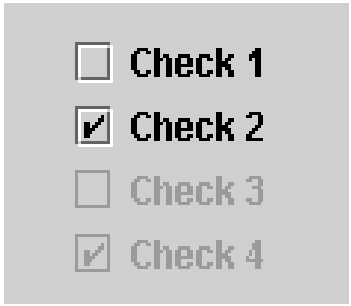
BUTTONS

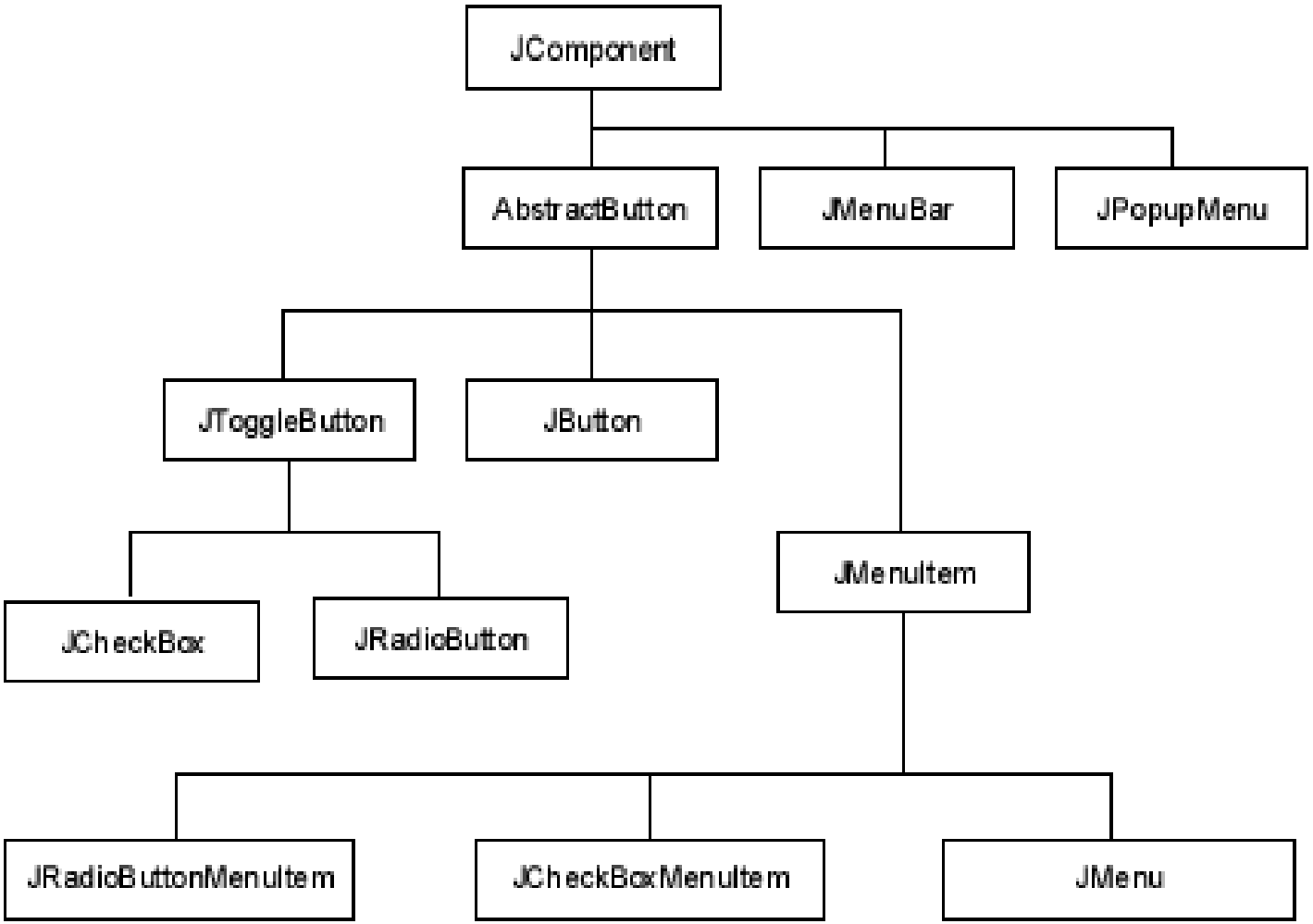

Toggle
Enabled
Disabled

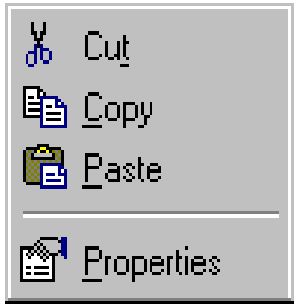
JComponent
AbstractButton
JToggleButton
JButton
JCheckBox
JRadioButton


Check 1
Radio 2
OK


Radio 1
Radio 2
Radio 3
Radio 4


Check 1
Check 2
Check 3
Check 4


JComponent
AbstractButton
JMenuBar
JPopupMenu
JToggleButton
JButton
JCheckBox
JRadioButton
JMenuItem
JRadioButtonMenuItem
JCheckBoxMenuItem
JMenu

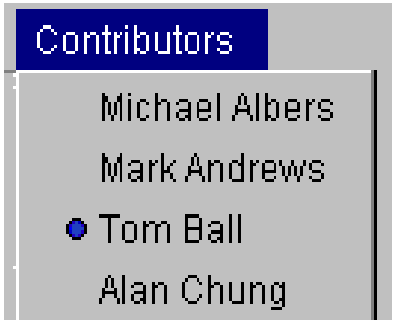

Cut
Copy
Paste
Properties


Text
Help
Settings
Font

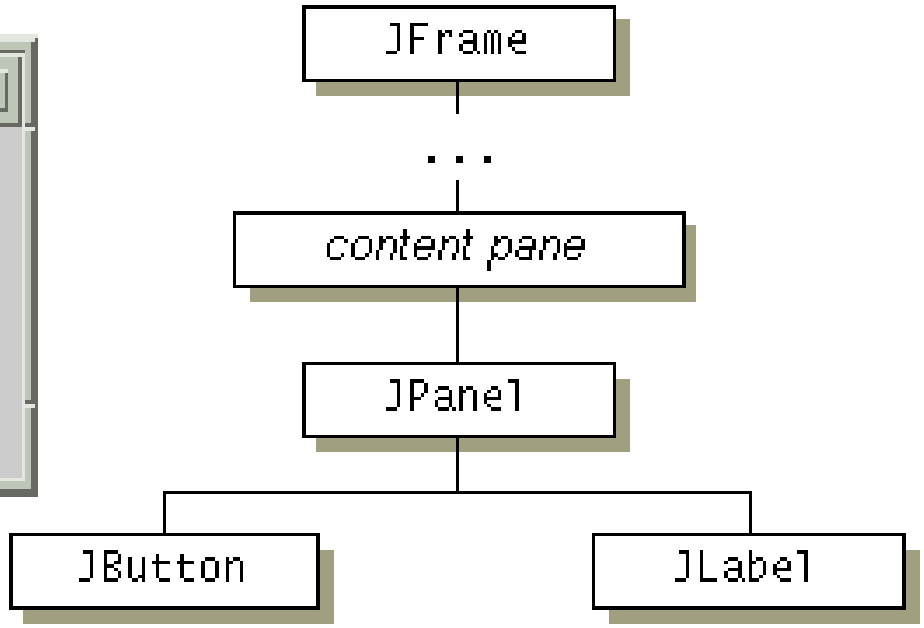
JMenuBar


Theme
Help
metal
ctrl-m
Organic
ctrl-o
metal2
ctrl-2


Radio 1
Radio 2
Radio 3


Contributors
Michael Albers
Mark Andrews
Tom Ball
Alan Chung

A simple Swing program - Containers



Containers

- Descendents of the **java.awt.Container** class
- Components that can contain other components.
- Use a layout manager to position and size the components contained in them.
- Components are added to a container using one of the various forms of its **add** method
  - Depending on which layout manager is used by the container

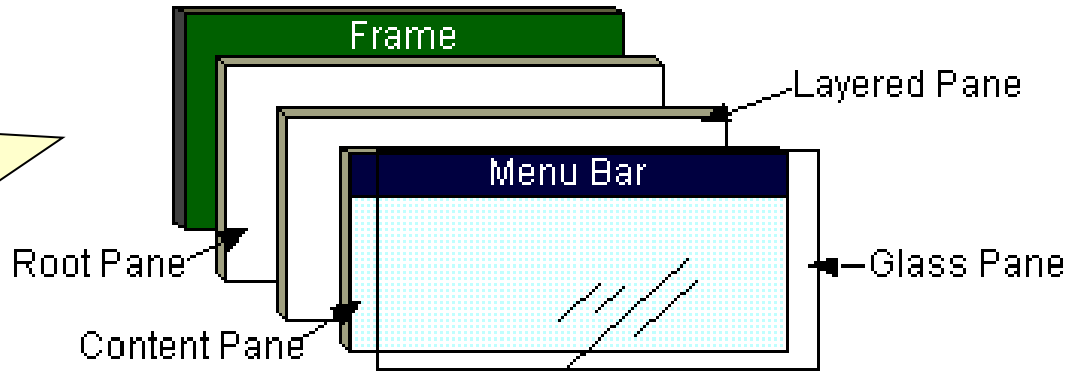
Top Level Containers

- Every program that presents a Swing GUI contains at least one top-level container.
- A Top level container provides the support that Swing components need to perform their painting and event-handling.
- Swing provides three top-level containers:
  - **JFrame** (Main window)
  - **JDialog** (Secondary window)
  - **JApplet** (An applet display area within a browser window)

Top Level Containers (cont)

- To appear on screen, every GUI component must be part of a **containment hierarchy**, with a top-level container as its root.
- Each top-level container has a **content pane** that contains visible components in that top-level container's GUI.

Don't add a component directly to a top-level container.



Frames

- Frame is a window that is not contained inside another window.
- For Swing GUI programs, use JFrame class to create widows.

JFrame

- A frame implemented as an instance of the JFrame class, is a window that has decorations such as a border, a title and buttons for closing and iconifying the window.
  - The decorations on a frame are platform dependent.
- Applications with a GUI typically use at least one frame.

Creating Frames

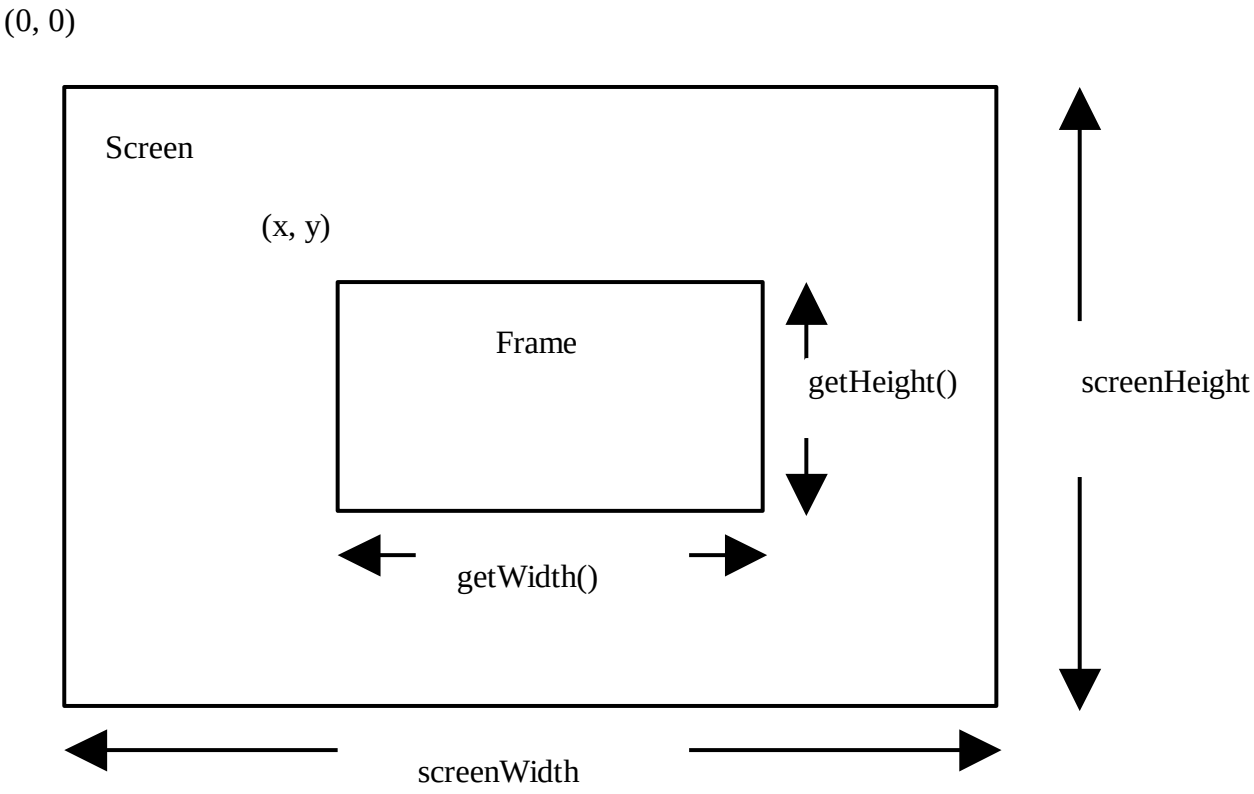
```
import javax.swing.*;

public class MyFrame {

  public static void main(String[] args) {
    JFrame frame = new JFrame("Test Frame");
    frame.setSize(400, 300);
    frame.setVisible(true);
    frame.setDefaultCloseOperation(
      JFrame.EXIT_ON_CLOSE);
  }
}
```

Centering Frames

By default, a frame is displayed in the upper-left corner of the screen.

To display a frame at a specified location, you can use the `setLocation(x, y)` method in the `JFrame` class. This method places the upper-left corner of a frame at location (x, y).

Centering Frames, cont.



Adding Components into a Frame

```
// Add a button into the frame
frame.getContentPane().
add( new JButton("OK"));
```

content pane

The content pane is a subclass of Container.

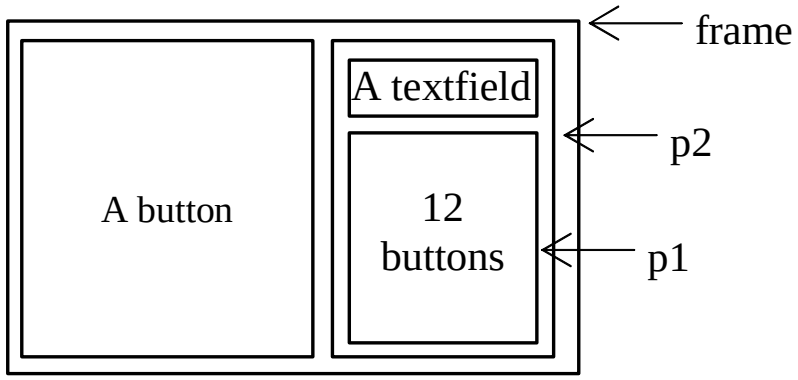
A JFrame object uses the content pane to hold components in the frame.

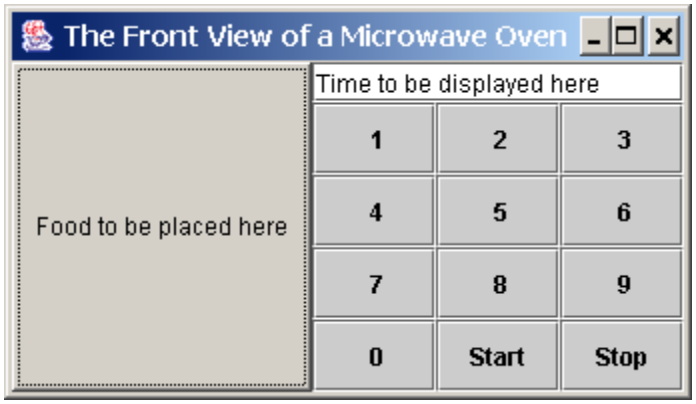
Intermediate Level Containers

- Also known as **panels** or **panes**
- Simplify the positioning of other components.
  - JPanel
- Play a visible and interactive role in a program's GUI
  - JScrollPane
  - JTabbedPane

Panel

Use panels to organize components. The program creates a user interface for a Microwave oven.



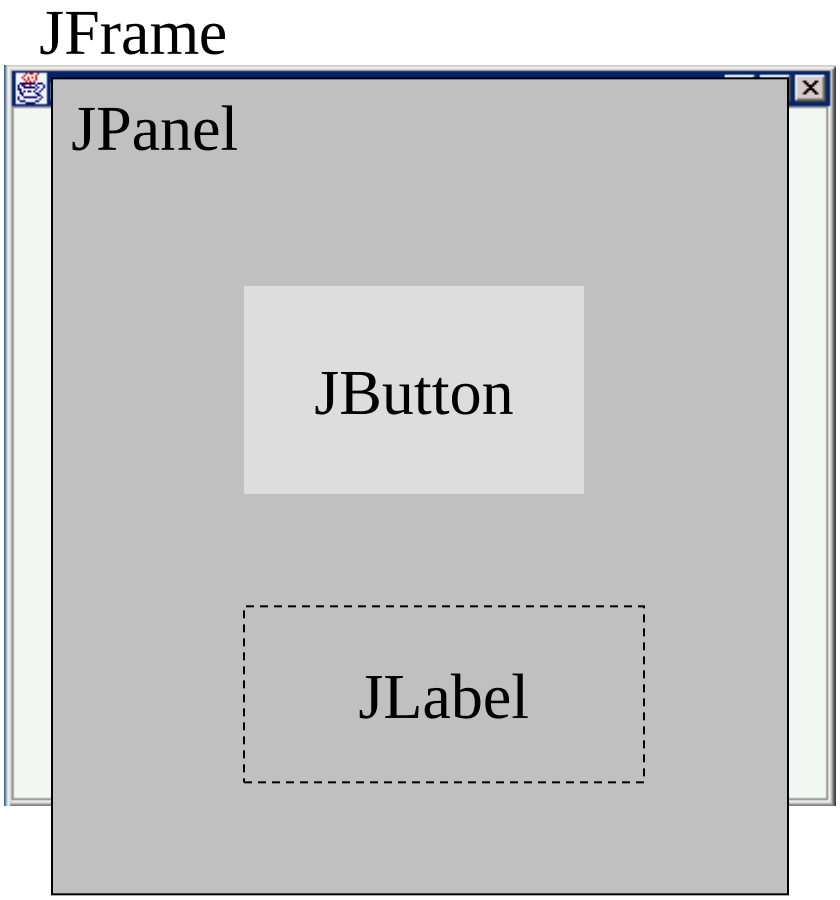


Using a GUI Component

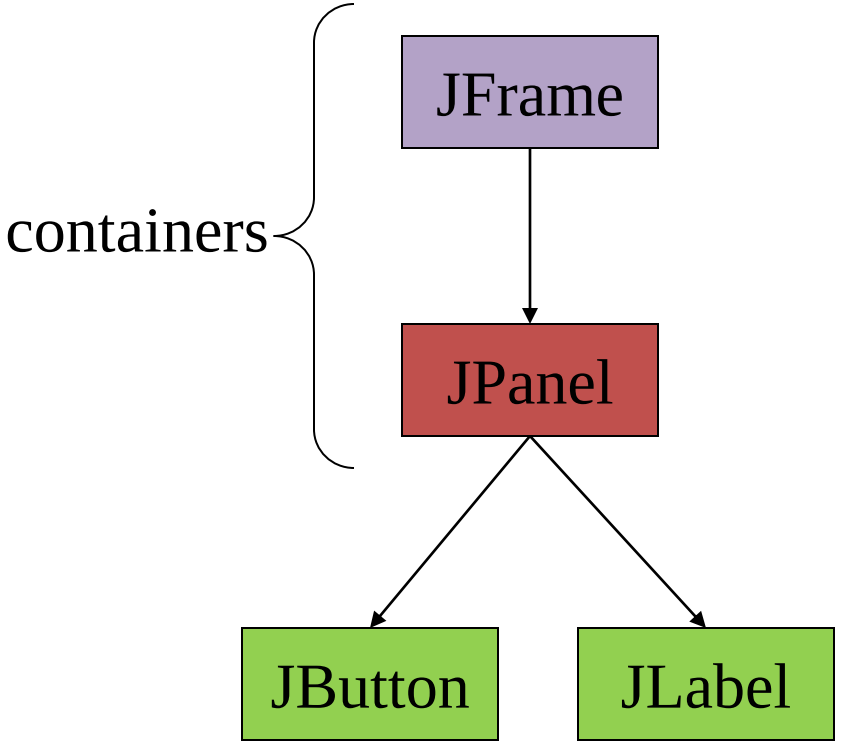
1. Create it
   - Instantiate object: b = **new JButton**(“press me”);
2. Configure it
   - Methods: **b.setText**(“press me”);
3. Add it
   - **panel.add(b)**;
4. Listen to it
   - **Events: Listeners**

Anatomy of an Application GUI

GUI

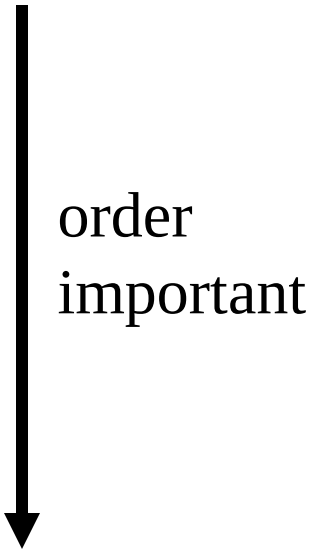


Internal structure



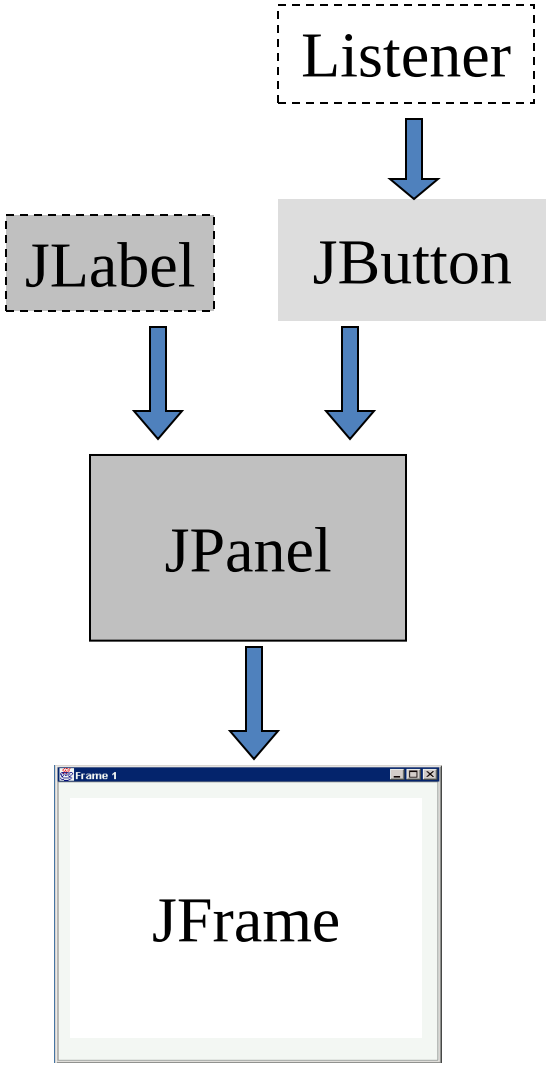
Using a GUI Component 2

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



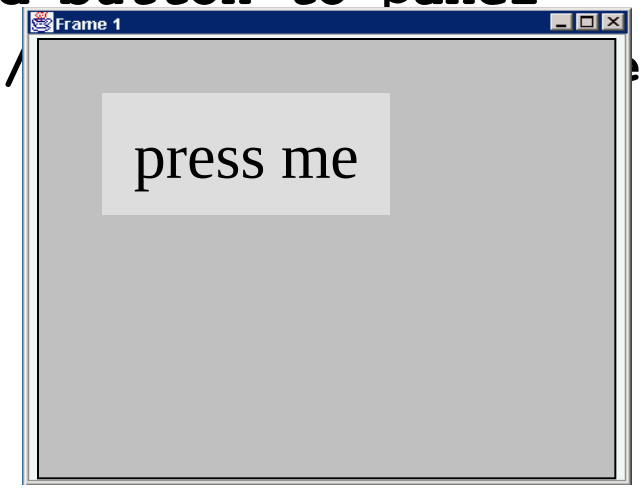
Build from bottom up

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



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);       //

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



Borders

- Every JComponent can have one or more borders.
- The class BorderFactory may be used to create standard borders

```
pane.setBorder(BorderFactory.

    createLineBorder(Color.black));
```

- Using a compound border, you can combine any two borders, which can themselves be compound borders

```
BorderFactory.createCompoundBorder(b
order1, border2);
```

Layout Management

- The process of determining the size and position of components.
- Layout management can be done using **absolute positioning**
  - Size and position of every component within the container must be specified.
  - Does not adjust well when the top-level container is resized.
  - Does not adjust well to differences between users and systems, such as font size.

Layout Management (cont)

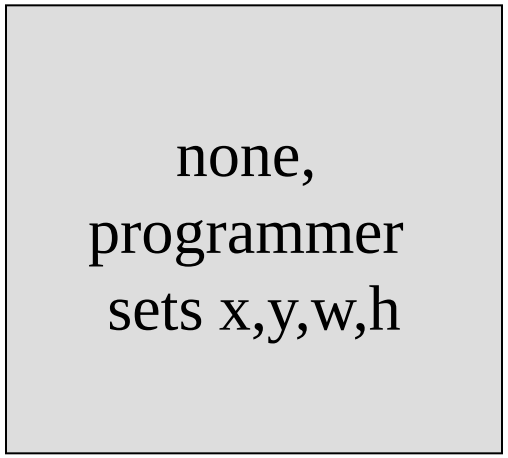
- Layout management is often performed using **layout mangers**
  - Components can provide size and position *hints* to layout managers, but layout managers have the final say on the size and position of those components.

Layout Management (cont)

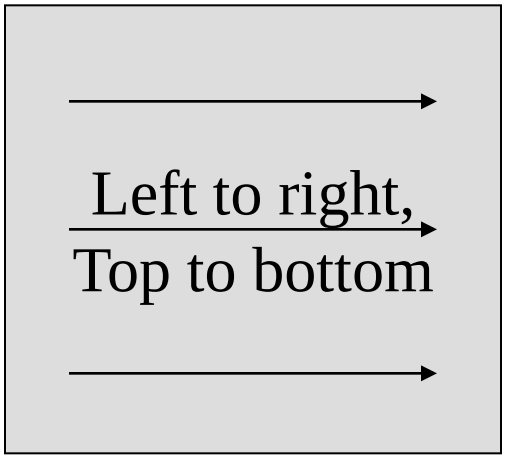
- The Java platform supplies five commonly used layout managers:
  - BorderLayout
  - BoxLayout
  - FlowLayout
  - GridLayout
  - GridBagLayout

Layout Manager Heuristics

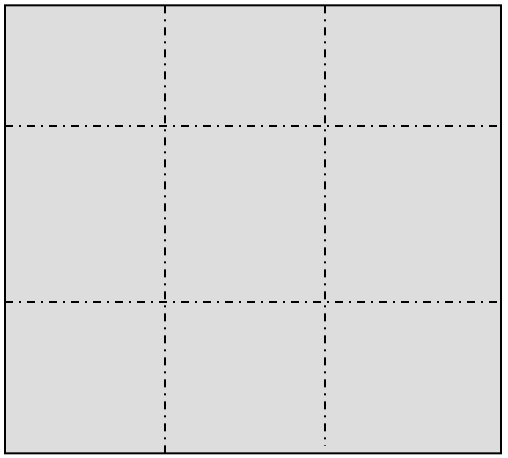
null



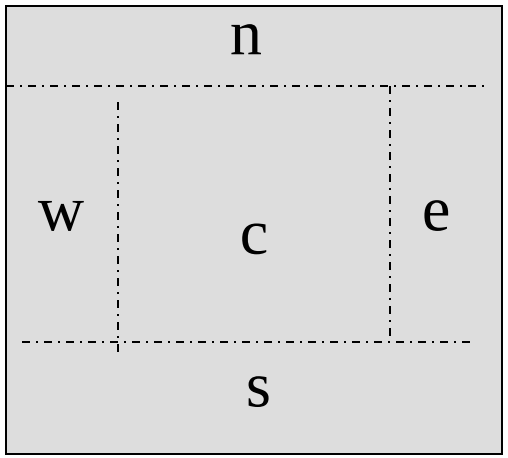
FlowLayout



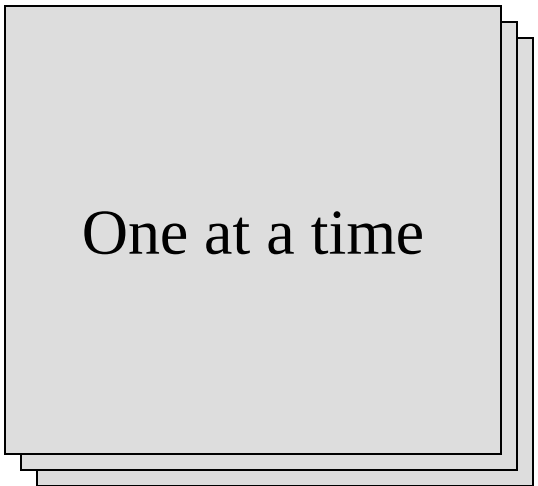
GridLayout



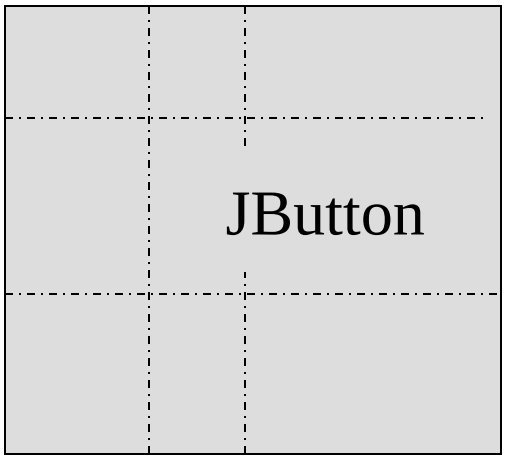
BorderLayout



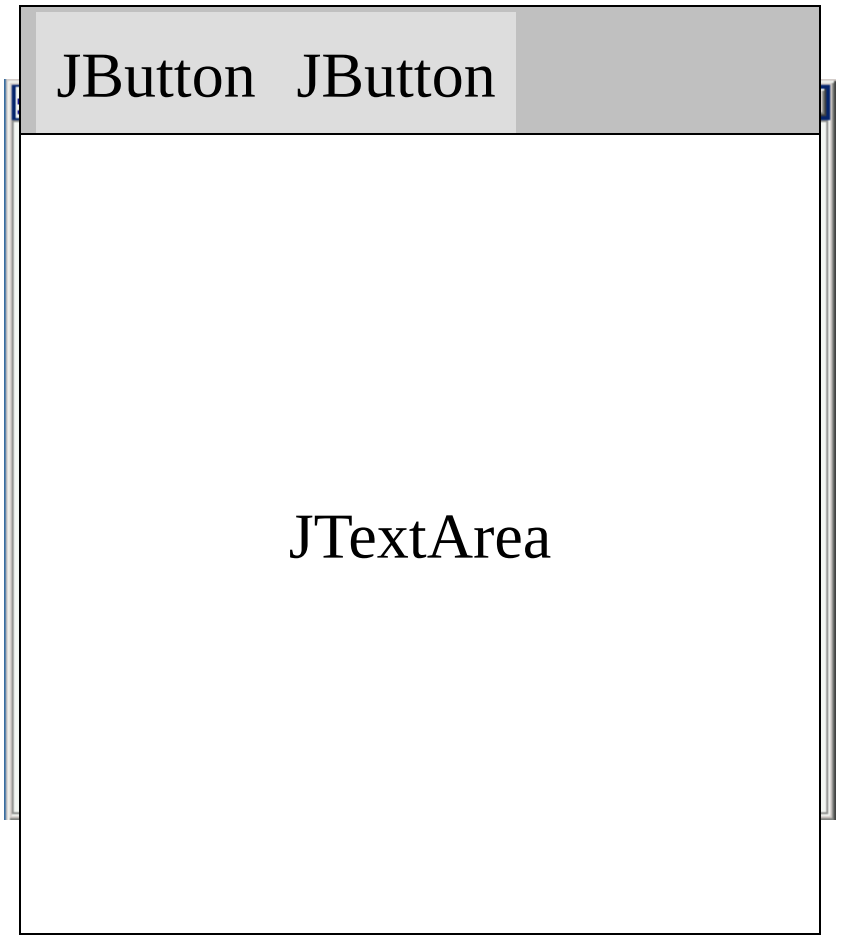
CardLayout



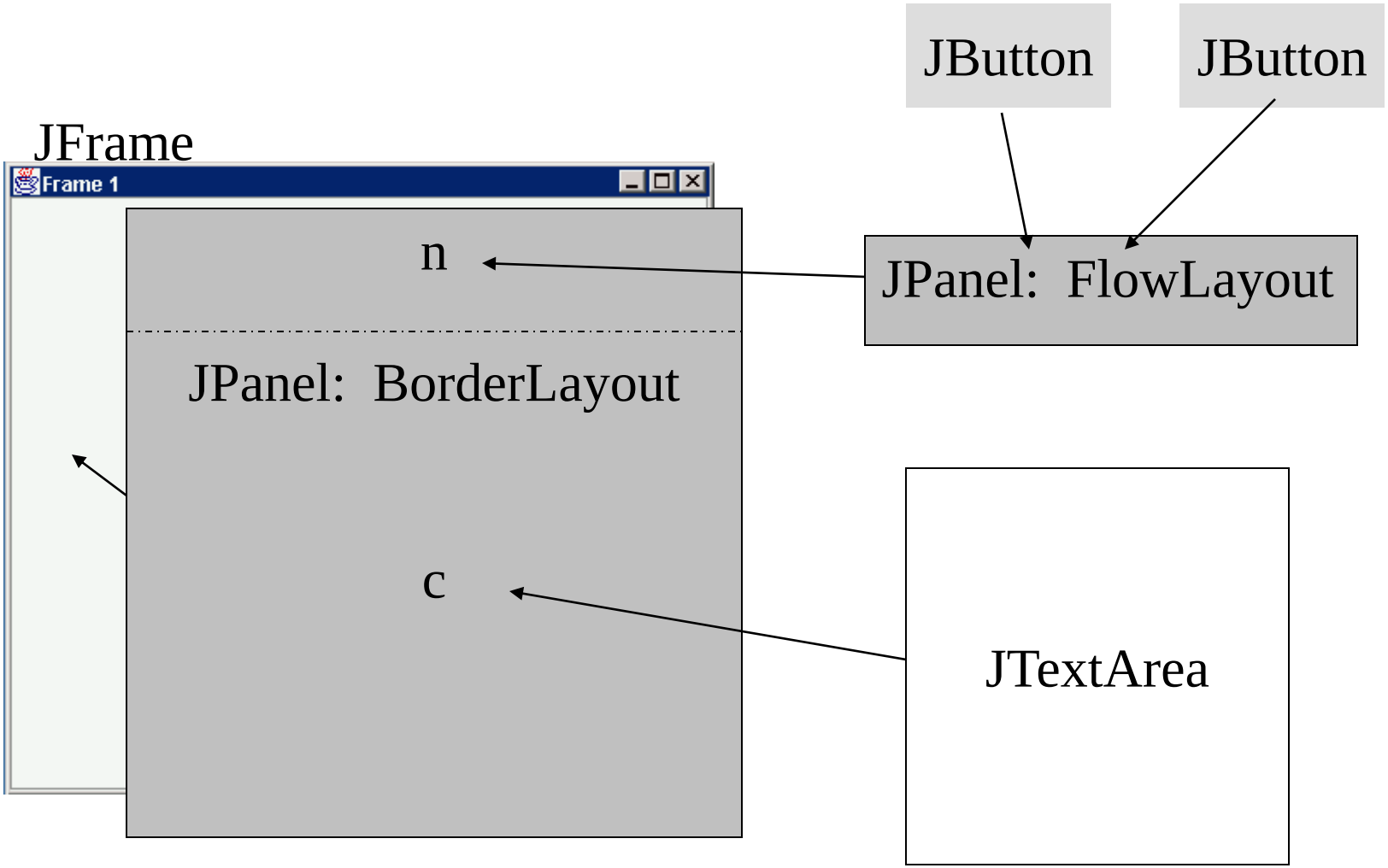
GridBagLayout



Combinations

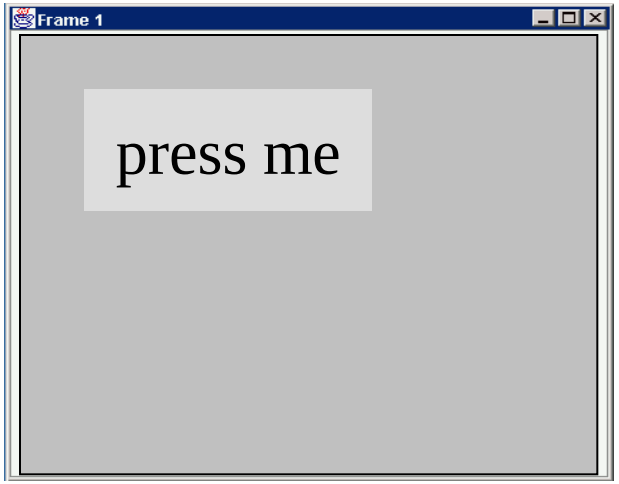


Combinations



Code: null layout

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

b.setBounds(new Rectangle(10,10,
  100,50));
p.setLayout(null);          // x,y layout
p.add(b);
f.setContentPane(p);
```



FlowLayout

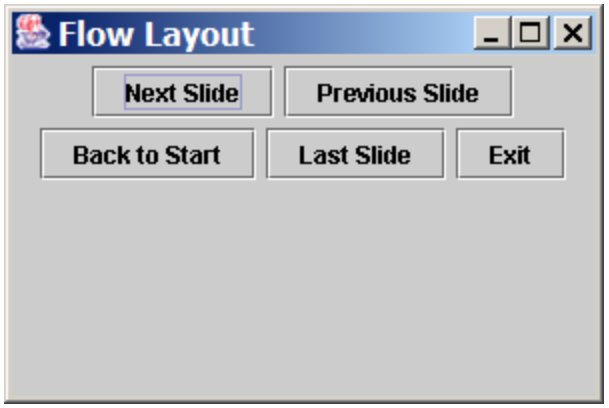
- Places components from left to right, starting new rows if necessary.
- Default LayoutManager of JPanel

FlowLayout

- Places components in a line as long as they fit, then starts the next line.
- Uses “best judgement” in spacing components. Centers by default.
- Lets each component assume its natural (preferred) size.
- Often used for placing buttons on panels.

FlowLayout (cont'd)

```
Container c = getContentPane();
c.setLayout (new FlowLayout() );
c.add (new JButton ("Next Slide"));
c.add (new JButton ("Previous Slide"));
c.add (new JButton ("Back to Start"));
c.add (new JButton ("Exit"));
```



Layout Management (cont)

- When using the *add* method to put a component in a container, the container's layout manager must be taken into account.
- A panel's default layout manager is FlowLayout.
  - Other layout managers can easily be set

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

  - Relative position (BorderLayout)

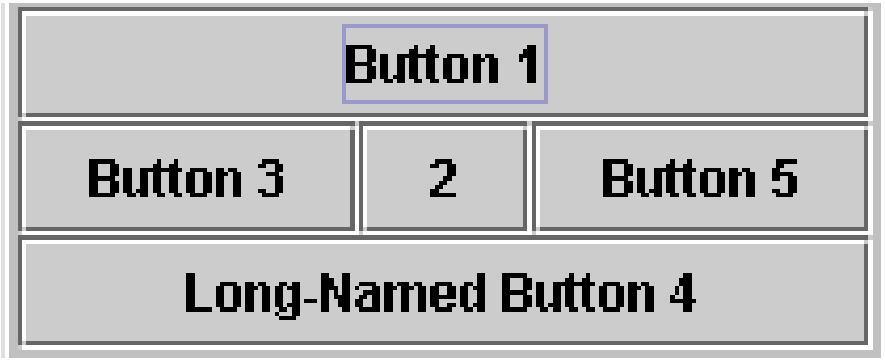
```
panel.add(component, BorderLayout.CENTER);
```

  - Order of addition (BoxLayout, GridLayout, ...)

```
panel.add(component);
```

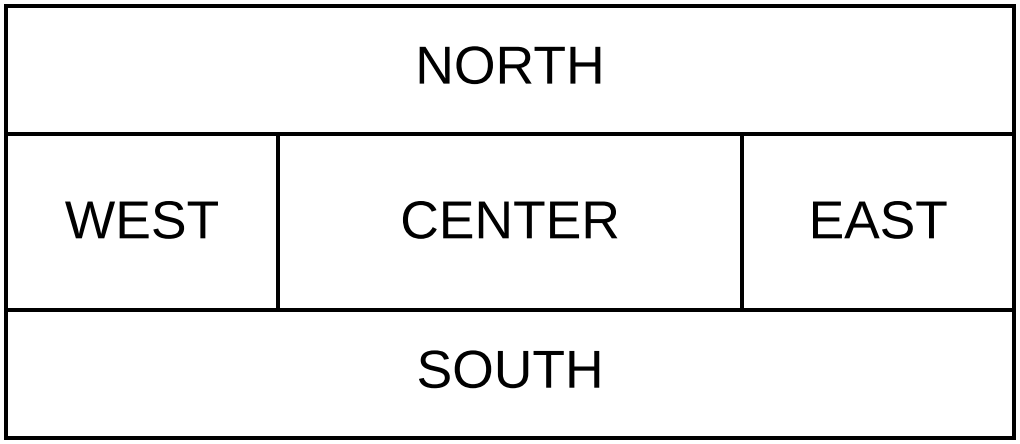
BorderLayout

- Has five areas available to hold components
  - north, south, east, west and center
- All extra space is placed in the center area
  - Only the center area is affected when the container is resized.
- Default layout manager of content panes.



BorderLayout

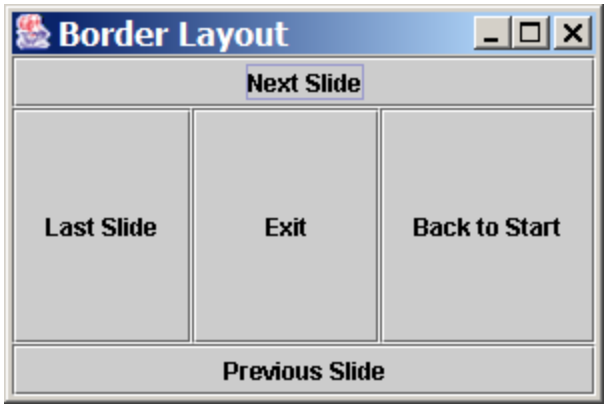
- Divides the area into five regions and adds a component to the specified region.



- Forces the size of each component to occupy the whole region.

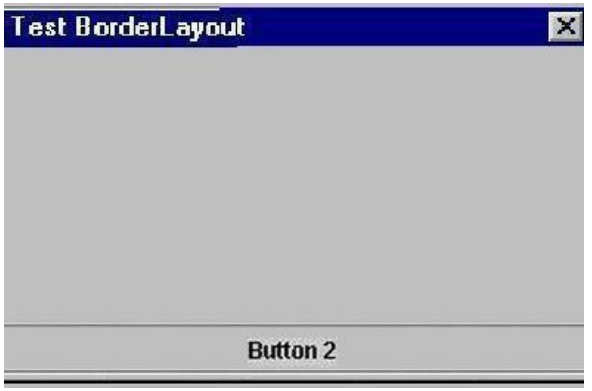
BorderLayout (cont'd)

```
Container c = getContentPane();
c.setLayout(new BorderLayout() );
c.add (new JButton ("Next Slide"), BorderLayout.NORTH);
c.add (new JButton ("Previous Slide"), BorderLayout.SOUTH);
c.add (new JButton ("Back to Start"), BorderLayout.EAST);
c.add (new JButton ("Last Slide"), BorderLayout.WEST);
c.add (new JButton ("Exit"), BorderLayout.CENTER);
```



Layout Management — Using Panel

- Potential problem with BorderLayout:
  - The button is stretched to fill the entire southern region of the frame
  - If you add another button to the southern region, it would just displace the first button



- Solution – use additional *panels*.
  - It acts as containers for interface elements and can themselves be arranged inside a larger panel
  - Use flow layout by default
  - To fix the problem of BorderLayout
    1. Create a panel
    2. Add components to the panel
    3. Add the panel to the larger container

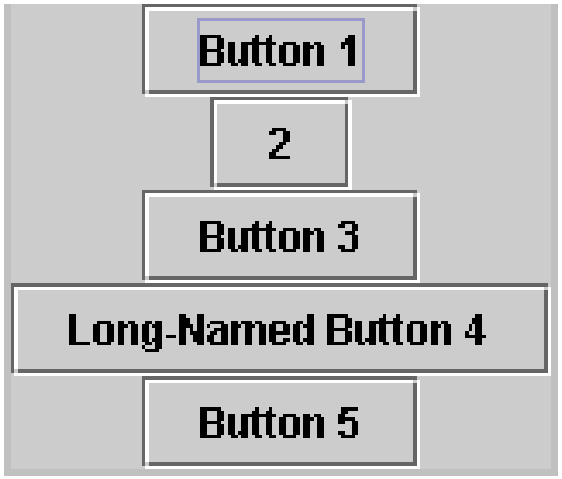

```
JPanel p = new JPanel();
p.add(button1);
p.add(button2);
P.add(button3);
frame.add(panel, BorderLayout.SOUTH);
```

Additional slides: Not for evaluation

BoxLayout

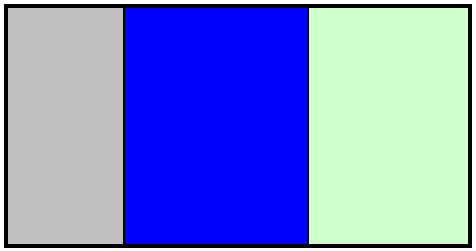
- Places components in a single row (left to right) or column (top to bottom).
- Respects component's maximum size and alignment hints.

```
BoxLayout layout = new BoxLayout(container, BoxLayout.Y_AXIS);
 container.setLayout(layout);
container.add(button);
```

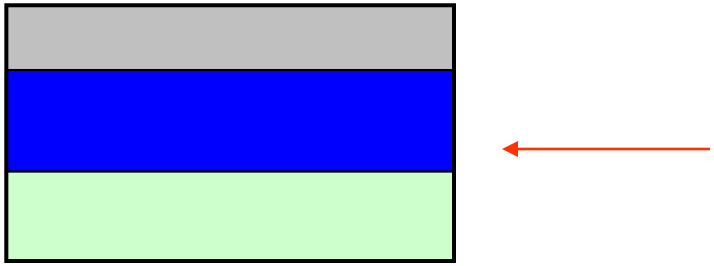


BoxLayout

- In a horizontal box, components are placed horizontally, left to right.



- In a vertical box, components are placed vertically, top to bottom.



"Horizontal" or "vertical" has nothing to do with the shape of the box itself.

GridLayout

- Places components in a requested number of rows and columns.
- Components are placed left-to-right and top-to-bottom.
- Forces all components to be the same size
  - as wide as the widest component's preferred width
  - as high as the highest component’s preferred height

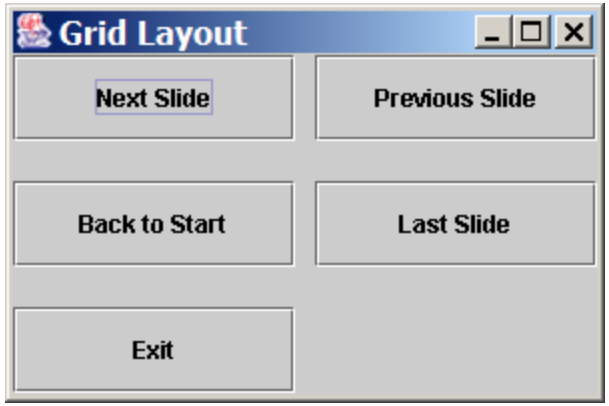
GridLayout

- Splits the panel into a grid with given number of rows and columns.
- Places components into the grid cells.
- Forces the size of each component to occupy the whole cell.
- Allows additional spacing between cells.

GridLayout (cont'd)

```
Container c = getContentPane();
c.setLayout (new GridLayout(3, 2,  10, 20 ));
c.add (new JButton ("Next Slide"));
c.add (new JButton ("Previous Slide"));
c.add (new JButton ("Back to Start"));
c.add (new JButton ("Last Slide"));
c.add (new JButton ("Exit"));
```

Extra space between the cells



Layout Management (cont)

- The following factors influence the amount of space between visible components in a container:
  - Layout manager
    - automatically, user specified, none
  - Invisible components
    - often used with BoxLayout
  - Empty borders
    - works best with components that have no default border such as panels and labels.

Example 1

```
import javax.swing.*;

public class HelloWorldSwing {

  public static void main(String[] args) {
    JFrame frame = new JFrame("HelloWorldSwing");
    final JLabel label = new JLabel("Hello
  World");
    frame.getContentPane().add(label);

  frame.setDefaultCloseOperation(JFrame.EXIT_ON_C
  LOSE);
    frame.pack();
    frame.setVisible(true);
  }
}
```

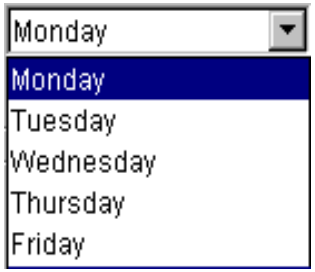
pack() causes a window to be sized to fit the preferred size and layouts of its sub-components

Example 2

In this example a custom frame is created

```
import javax.swing.*;

public class HelloWorldFrame extends JFrame {
  public HelloWorldFrame() {
    super("HelloWorldSwing");
    final JLabel label = new JLabel("Hello
  World");
    getContentPane().add(label);

  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    pack();
    setVisible(true);
  }
  public static void main(String[] args) {
    HelloWorldFrame frame = new
  HelloWorldFrame();
  }}
```

OTHER COMPONENTS



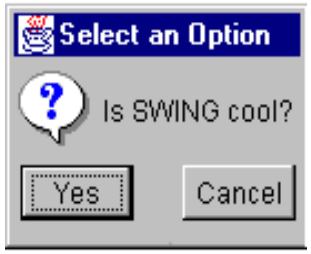
JComboBox

JApplet

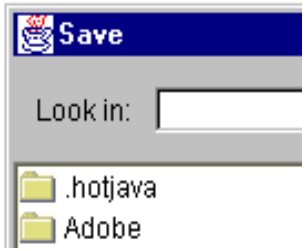
Border Interface

JColorChooser

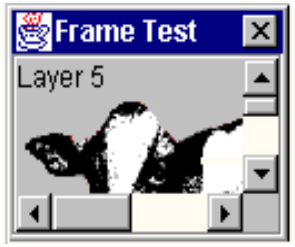
ImageIcon



JDialog

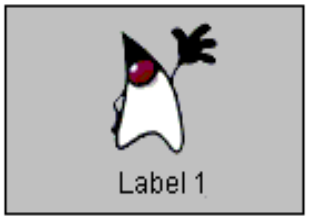


JFileChooser



JInternalFrame

OTHER COMPONENTS



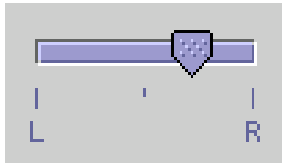
JLabel

JList

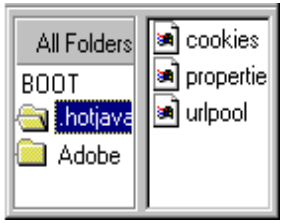
JScrollBar

JScrollPane

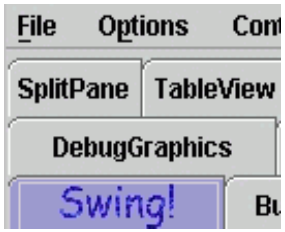
JOptionPane



JSlider



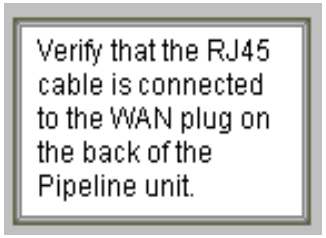
JSplitPane



JTabbedPane

OTHER COMPONENTS

JTable



JTextArea

JTextField

JToolBar

JToolTip

JTree

Menus

- You can add a JMenuBar object to JFrame or JApplet.
- You can add JMenu objects to a JMenuBar.
- You can add other JMenus, JMenuItems, JCheckBoxMenuItems, JRadioButtonMenuItems, etc. to a Jmenu.

Menus

Menu Bar

- JMenuBar()
- add(JMenu)

Menu

- JMenu(String)
- add(JMenuItem)

JMenuItem(String)
JMenuItem(String,int)

```
JMenuBar mb = new JMenuBar();          //create a menu bar
JMenu fileMenu = new JMenu ("File");   //create a menu
mb.add( fileMenu );                    //add menu to menu bar
setMenuBar( mb );        // add a menu bar to frame
fileMenu.setMnemonic( KeyEvent.VK_F ); // add a hotkey to menu

JMenuItem miOpen = new JMenuItem( "Open...", KeyEvent.VK_O );
JMenuItem miExit = new JMenuItem( "Exit" );

fileMenu.add( miOpen ); // add a menu item
fileMenu.addSeparator(); // add a menu separator
fileMenu.add( miExit );
```

JLabel

- Labels
  - Provide text instructions on a GUI
  - Read-only text
  - Programs rarely change a label's contents
  - Class **`JLabel`** (subclass of **`JComponent`**)

- Methods
  - Can declare label text in constructor
  - **`myLabel.setToolTipText( "Text" )`**
    - Displays **`"Text"`** in a tool tip when mouse over label
  - **`myLabel.setText( "Text" )`**
  - **`myLabel.getText()`**

JLabel

- **`Icon`**
  - Object that implements interface **`Icon`**

```
24        Icon bug = new ImageIcon( "bug1.gif" );
```

  - One class is **`ImageIcon`** (**`.gif`** and **`.jpeg`** images)

```
33        label3.setIcon( bug );
```

    - Assumed same directory as program
  - Display an icon with **`JLabel`**'s **`setIcon`** method
    - **`myLabel.setIcon( myIcon );`**
    - **`myLabel.getIcon //returns current Icon`**

```
1  // Fig. 12.4: LabelTest.java
2  // Demonstrating the JLabel class.
3  import javax.swing.*;
4  import java.awt.*;
5  import java.awt.event.*;
6
7  public class LabelTest extends JFrame {
8     private JLabel label1, label2, label3;
9
10    public LabelTest()
11    {
12       super( "Testing JLabel" );
13
14       Container c = getContentPane();
15       c.setLayout( new FlowLayout() );
16
17       // JLabel constructor with a string argument
18       label1 = new JLabel( "Label with text" );
19       label1.setToolTipText( "This is label1" );
20       c.add( label1 );
21
22       // JLabel constructor with string, Icon a
23       // alignment arguments
24       Icon bug = new ImageIcon( "bug1.gif" );
25       label2 = new JLabel( "Label with text and icon",
26                            bug, SwingConstants.LEFT );
27       label2.setToolTipText( "This is label2" );
28       c.add( label2 );
29
```

Create a **Container** object, to which we attach **JLabel** objects (subclass of **JComponent**).

Initialize text in **JLabel** constructor.

Set the tool tip text, and attach component to **Container c**.

Create a new **ImageIcon** (assumed to be in same directory as program).

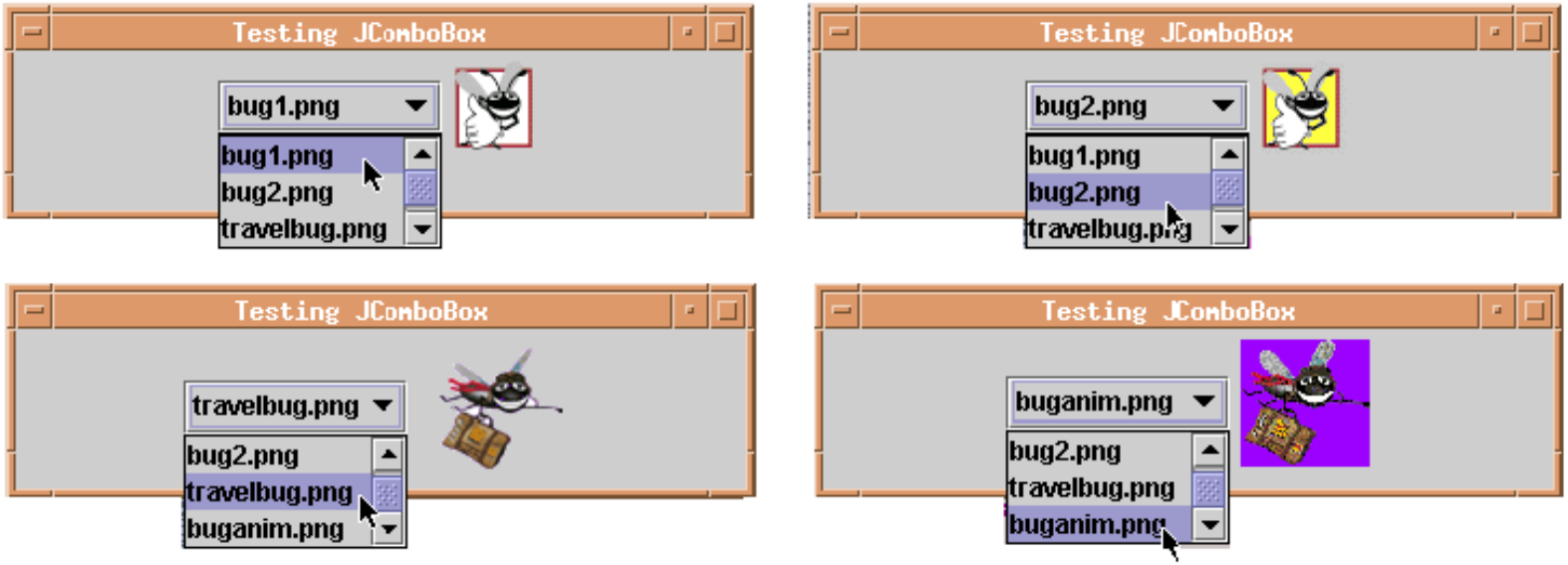
Set **ImageIcon** and alignment of text in **JLabel** constructor.

`JTextField` and `JPasswordField`

- They are single-line areas in which text can be entered by the user from the keyboard or text can simply be displayed
- When the user types data into them and presses the *`Enter`* key, an action event occurs. If the program registers an event listener, the listener processes the event and can use the data in the text field at the time of the event in the program.

More Examples …

- `JComboBox`: a drop-down list provides a list of items from which the user can make a selection. It generates `ItemEvent`



- `JList`: a list supports both *single selection* and *multiple-selection*. It generates `ListSelectionEvent`

JDialog

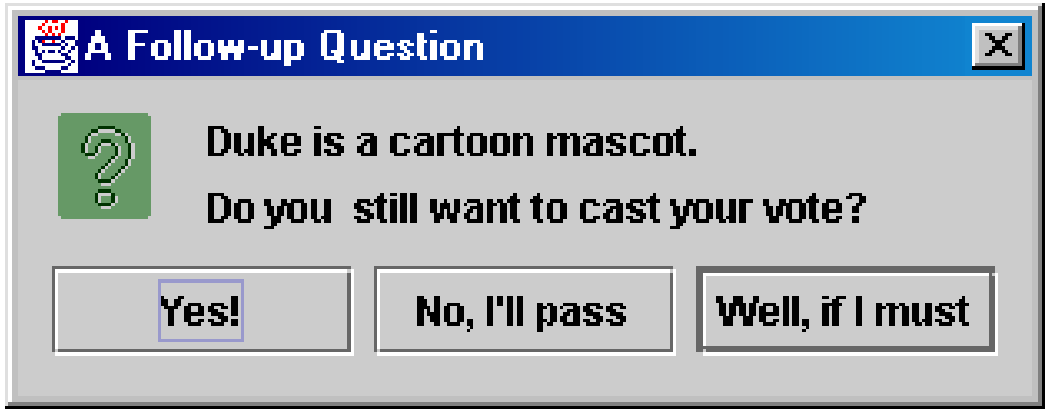
- Every dialog is dependent on a frame
  - Destroying a frame destroys all its dependent dialogs.
  - When the frame is iconified, its dependent dialogs disappear from the screen.
  - When the frame is deiconified, its dependent dialogs return to the screen.
- A dialog can be **modal**. When a modal dialog is visible it blocks user input to all other windows in the program.

Jdialog

- To create custom dialogs, use the JDialog class directly (as in the previous examples).
- Swing provides several standard dialogs
  - JProgressBar, JFileChooser, JColorChooser, ...
- The JOptionPane class can be used to create simple modal dialogs
  - icons, title, text and buttons can be customized.

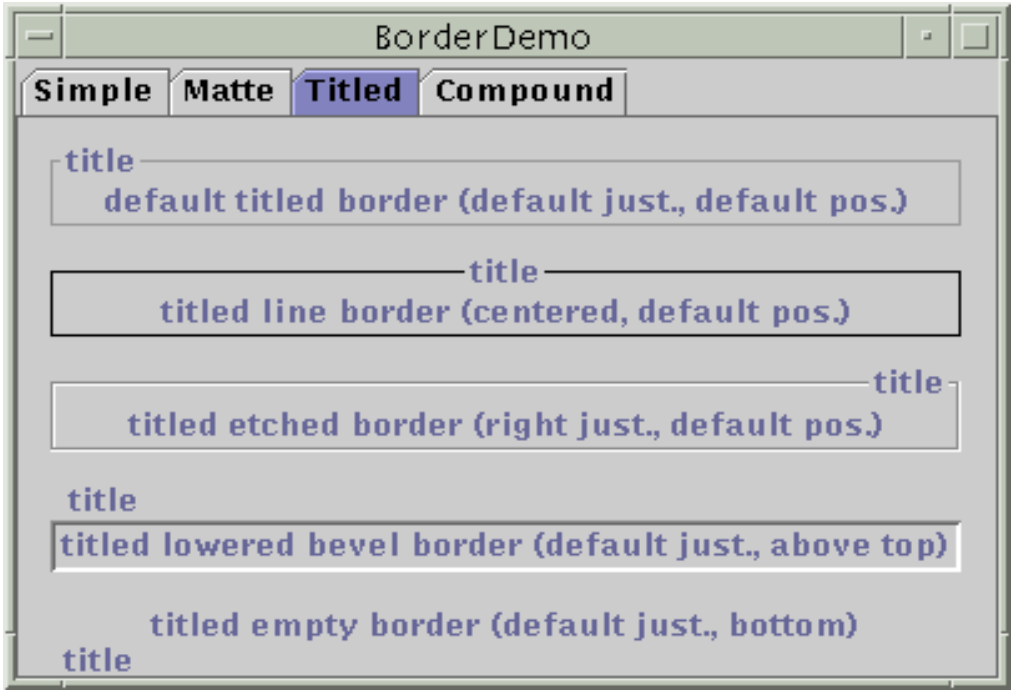
Example

```
Object[] options = {"Yes!", "No, I'll pass",
                    "Well, if I must"};
int n = JOptionPane.showOptionDialog(
        frame, "Duke is a cartoon mascot. \n"
  +
        "Do you still want to cast your
  vote?",
        "A Follow-up Question",
        JOptionPane.YES_NO_CANCEL_OPTION,
        JOptionPane.QUESTION_MESSAGE,
        null,
        options,
        options[2]);
```



Simple Borders

Titled Borders



Compound Border