Object Oriented Programming

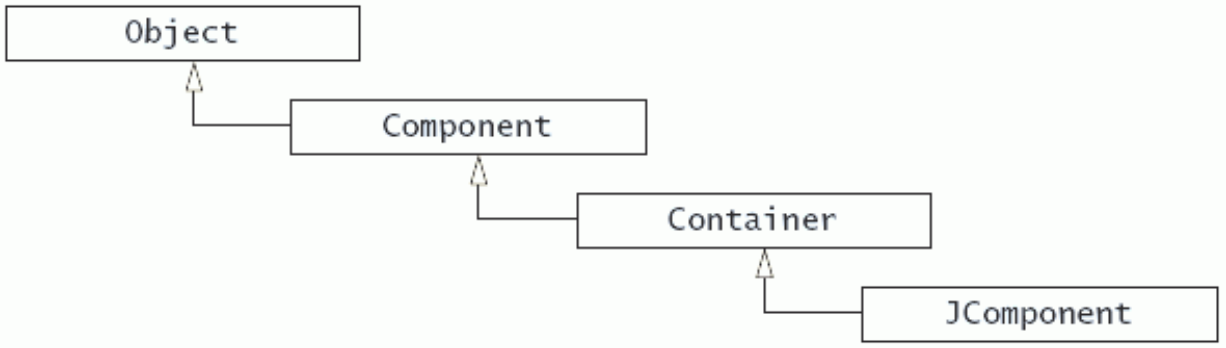
1. Graphical User Interfaces

GUI

- A Graphical User Interface (GUI – pronounced "goo-ee") presents a user-friendly mechanism for interacting with a program
  - Gives a program a distinctive "look" and "feel"
  - Allows users to be somewhat familiar with a program before they ever use it
  - Reduces the time to learn usage

Containers and Components (again)

- Class **`Component`** declares the common attributes and behaviors of all its subclasses
  - Important methods: **`paint(), repaint()`**
- Class **`Container`** manages a collection of related components
  - In applications with **`JFrame`** and in applets we attach components to the content pane – a container
  - Important methods: **`add(), setLayout()`**



The `Container` Class

- Any class that is a descendent class of the class `Container` is considered to be a container class
  - The `Container` class is found in the `java.awt` package, not in the Swing library
- Any object that belongs to a class derived from the `Container` class (or its descendents) can have components added to it
- The classes `JFrame` and `JPanel` are descendent classes of the class `Container`
  - Therefore they and any of their descendents can serve as a container

The `JComponent` Class

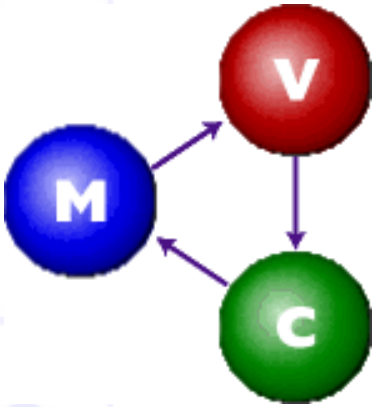
- Declares common attributes and behaviors for its subclasses, such as:
  - *a pluggable look-and-feel* – used to customize the appearance of components
  - shortcut keys for direct access to GUI components through the keyboard
  - common event-handling capabilities for cases where several GUI components initiate the same actions in a program
  - brief descriptions of a GUI's component purpose (called *tool tips*)

The `JComponent` Class

- Provides support for user interface localization-customizing the interface to display in different language and use local cultural conventions
- Any descendent class of the class `JComponent` is called a *component class*
  - Any `JComponent` object or *component* can be added to any container class object
  - Because it is derived from the class `Container`, a `JComponent` can also be added to another `JComponent`

Swing and the Model-View-Controller Architecture

- Swing architecture is rooted in the *model-view-controller* (*MVC*) design that dates back to SmallTalk.
- MVC architecture calls for a visual application to be broken up into three separate parts:
  - A *model* that represents internally the data for the application
  - The *view* that is the visual representation of that data.
  - A *controller* that takes user input on the view and translates that to changes in the model.



The Model

- Most programs are supposed to do work, not just be "another pretty face"
  - but there are some exceptions
  - useful programs existed long before GUIs
- The Model is the part that does the work – it *models* the actual problem being solved
- The Model should be independent of both the Controller and the View
  - *But it can provide services (methods) for them to use*
- Independence gives flexibility, robustness

The Controller

- The Controller decides what the model is to do
- Often, the user is put in control by means of a GUI
  - in this case, the GUI and the Controller are often the same
- The Controller and the Model can almost always be separated (what to do versus how to do it)
- The design of the Controller depends on the Model
- The Model should *not* depend on the Controller

The View

- Typically, the user has to be able to see, or *view*, what the program is doing
- The View shows what the Model is doing
  - The View is a *passive* observer; it should not affect the model
- The Model should be independent of the View, but (but it can provide access methods)
- The View should ***not*** display what the Controller ***thinks*** is happening

Combining Controller and View

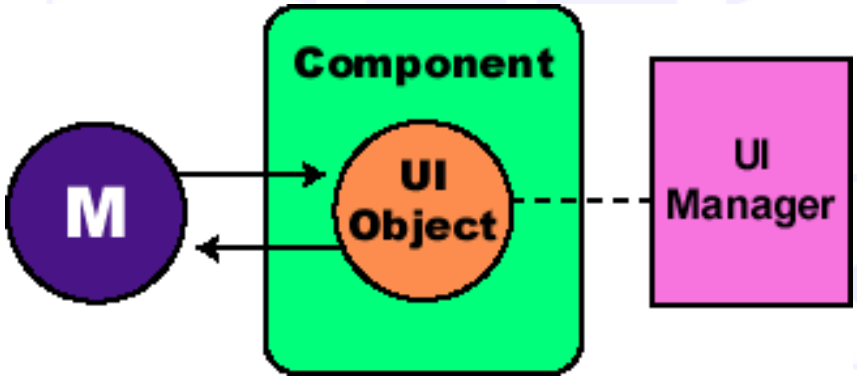
- Sometimes the Controller and View are combined, especially in small programs
- Combining the Controller and View is appropriate if they are very interdependent
- The Model should still be independent
- *Never* mix Model code with GUI code!

Separation of concerns

- As always, you want code independence
- The Model should not be contaminated with control code or display code
- The View should represent the Model as it really is, not some remembered status
- The Controller should *talk to* the Model and View, not *manipulate* them
  - The Controller can set variables that the Model and View can read

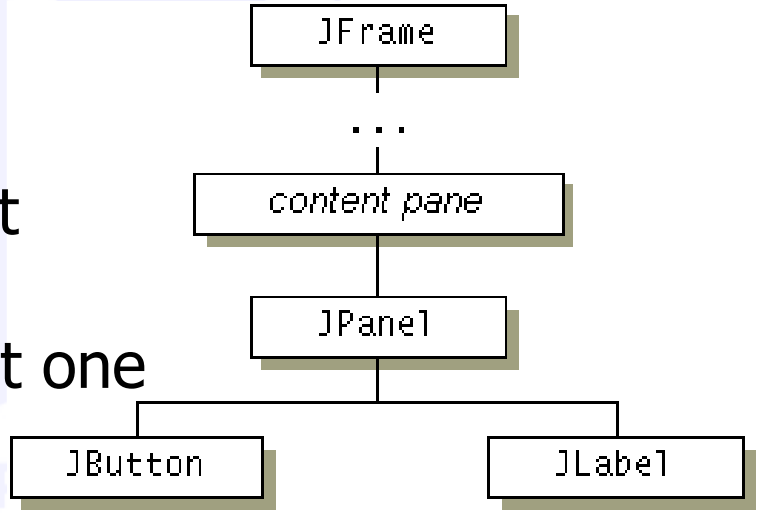
Swing Separable Model Architecture

- The *view* and *controller* parts of a component required a tight coupling
  - These two entities were collapsed into a single UI (user-interface) object
  - This new quasi-MVC design is sometimes referred to a *separable model architecture*.



Containment Hierarchies

- Top level containers
  - Intermediate containers
    - 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

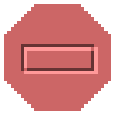


Dialog boxes

- More limited than frames
- Modality
  - Modal dialog boxes temporarily halt the program – the user cannot continue until the dialog has been closed
- Types of dialogs
  - `JOptionPane`
  - `ProgressMonitor`
  - `JColorChooser`
  - `JDialog`

Showing dialogs

- JOptionPane.showXYZDialog(...)
  - Option and Message dialogs



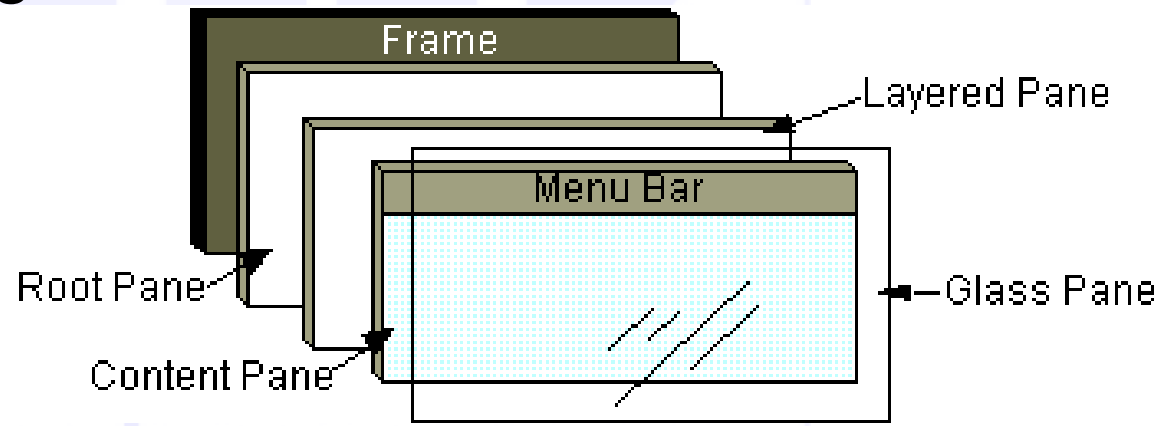
```
JOptionPane.showMessageDialog(frame, "Error!", "An error message", JOptionPane.ERROR_MESSAGE);
```

```
JOptionPane.showOptionDialog(frame, "Save?", "A save dialog", JOptionPane.YES_NO_CANCEL_OPTION);
```

  - Input, Confirm
- Customization
  - showOptionDialog - Fairly customizable
  - JDialog - Totally customizable

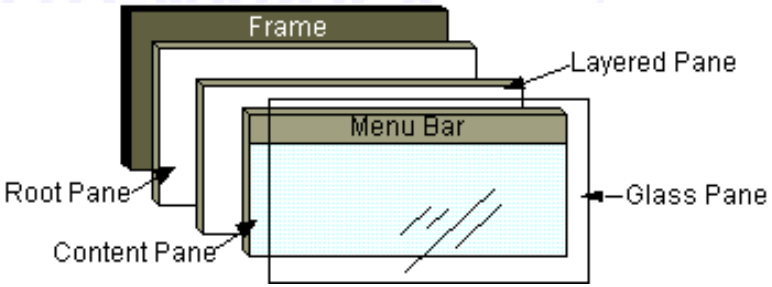
Intermediate containers – Panels (or 'panes')

- Root panes
  - The content pane
  - Layered panes
  - Glass panes

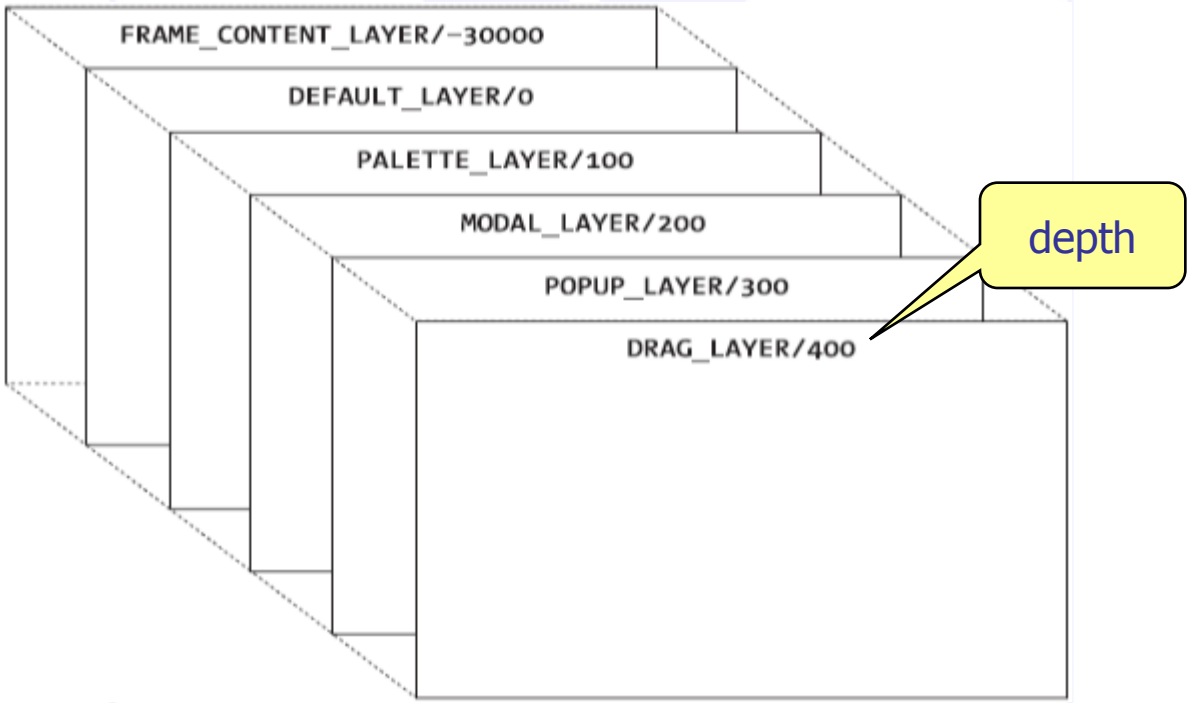


Root panes

- 'Invisibly' attached to top-level container
- Created by Swing on realizing frame
- Manages everything between top-level container and components
- Places menu bar and content pane in an instance of **`JLayeredPane`**

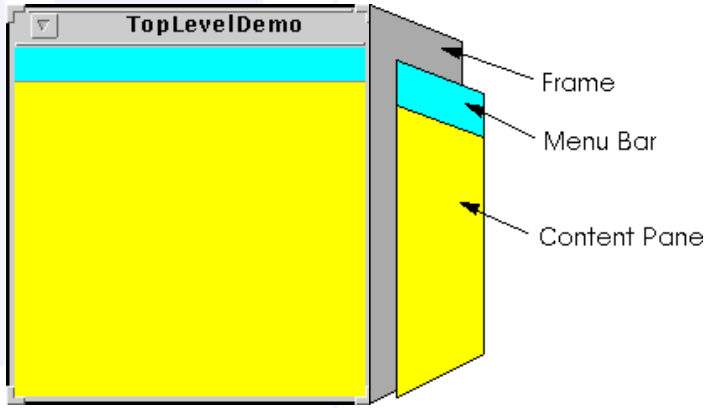


JLayeredPanes

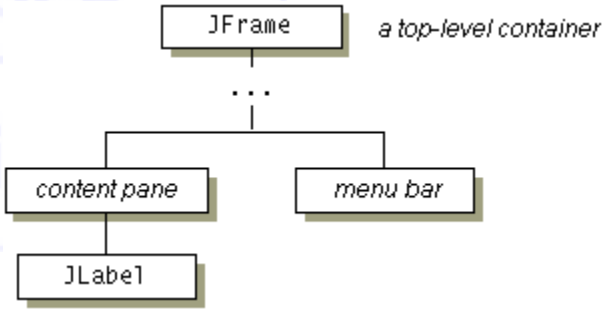


Content panes

- Usually use a **`JPanel`**
- Contains everything except menu bar for most Swing applications
- Can be explicitly, or implicitly created,
  - simplified code




```
//Create a panel and add components to it.
JPanel contentPane = new JPanel();
contentPane.add(someComponent);
contentPane.add(anotherComponet);
//Make it the content pane.
contentPane.setOpaque(true);
topLevelContainer.setContentPane(contentPane)
;
```



Glass panes

- Not structured into components
  - event catching
  - painting
- Used for 'weird' interface behavior, rarely used.
- Either created explicitly or root version used

Objects in a Typical GUI

- Almost every GUI built using Swing container classes will be made up of three kinds of objects:
  1. The *container* itself, probably a *panel* or *window-like* object
  2. The *components* added to the container such as labels, buttons, and panels
  3. A *layout manager* to position the components inside the container

Tip: Code a GUI's Look and Actions Separately

- The task of designing a Swing GUI can be divided into two main subtasks:
  1. Designing and coding the appearance of the GUI on the screen
  2. Designing and coding the actions performed in response to user actions
- In particular, it is useful to implement the `actionPerformed()` method as a *stub*, until the GUI looks the way it should

```
public void actionPerformed(ActionEvent e)
{}
```

- This philosophy is at the heart of the technique used by the *Model-View-Controller* pattern

Using Inheritance to Customize Frames

- Use inheritance for complex frames to make programs easier to understand
- Design a subclass of **`JFrame`**
- Store the components as instance fields
- Initialize them in the constructor of your subclass
- If initialization code gets complex, simply add some helper methods

Layout Management

- Up to now, we have had limited control over layout of components
  - When we used a panel, it arranged the components from the left to the right
- User-interface components are arranged by placing them inside containers
- Each container has a *layout manager* that directs the arrangement of its components
- Some useful layout managers:
  - border layout
  - flow layout
  - grid layout
  - box layout

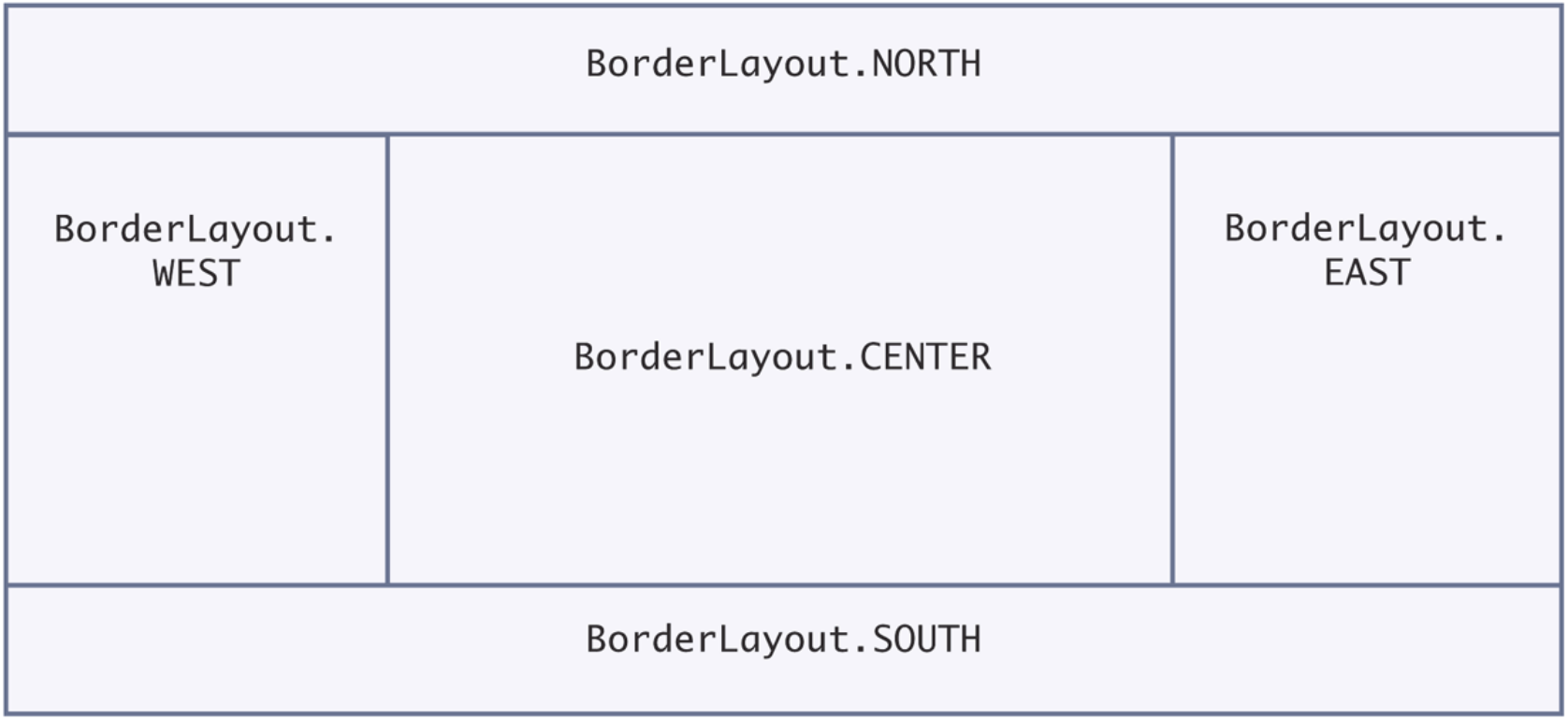
Layout Management

- By default, **`JPanel`** places components from left to right and starts a new row when needed
- Panel layout carried out by **`FlowLayout`** layout manager
- Can set other layout managers

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

Border Layout

- Border layout groups container into five areas: center, north, west, south and east
  - Components expand to fill space in the border layout



Border Layout

- Default layout manager for a frame (technically, the frame's content pane)
- When adding a component, specify the position like this:

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

- Expands each component to fill the entire allotted area
- If that is not desirable, place each component inside a panel

`FlowLayout` Layout Manager

- The `FlowLayout` layout manager arranges com-ponents in order from left to right and top to bottom across a container
- Constructors:

```
public FlowLayout();
public FlowLayout(int align);
public FlowLayout(int align, int horizontalGap,
                              int verticalGap);
```

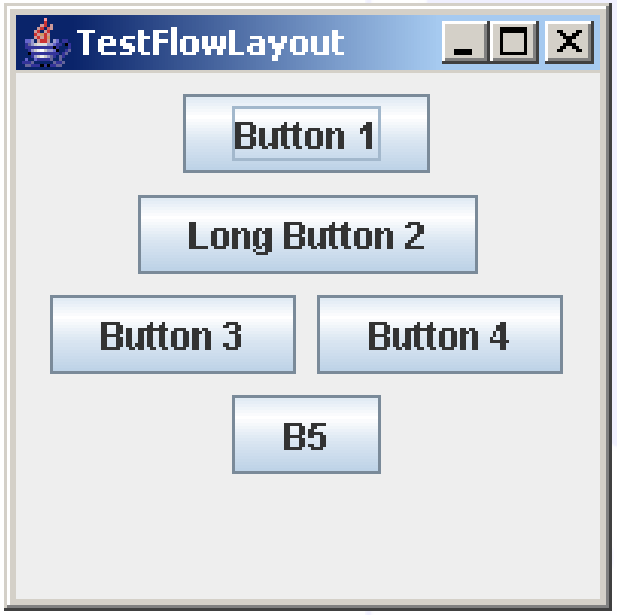
- The alignment can be `LEFT`, `RIGHT`, or `CENTER`
- Default layout manager for `JPanel`

Grid Layout

- Arranges components in a grid with a fixed number of rows and columns
- Resizes each component so that they all have same size
- Expands each component to fill the entire allotted area
- Add the components, row by row, left to right:

```
JPanel numberPanel = new JPanel();
numberPanel.setLayout(new GridLayout(4, 3));
numberPanel.add(button7);
numberPanel.add(button8);
numberPanel.add(button9);
numberPanel.add(button4);
. . .
```

Examples: `FlowLayout` and `GridLayout`



BlueJ: TestFlowLayout

BlueJ: TestGridLayout

Grid Bag Layout

- Tabular arrangement of components
  - Columns can have different sizes
  - Components can span multiple columns
- Quite complex to use
- Fortunately, you can create acceptable-looking layouts by nesting panels
  - Give each panel an appropriate layout manager
  - Panels don't have visible borders
  - Use as many panels as needed to organize components

`BoxLayout` Layout Manager

- The `BoxLayout` layout manager arranges components within a container in a single row or a single column.
- The spacing and alignment of each element on each row or column can be individually controlled.
- Containers using `BoxLayout` managers can be nested inside each other to produce complex layouts
- Constructor:

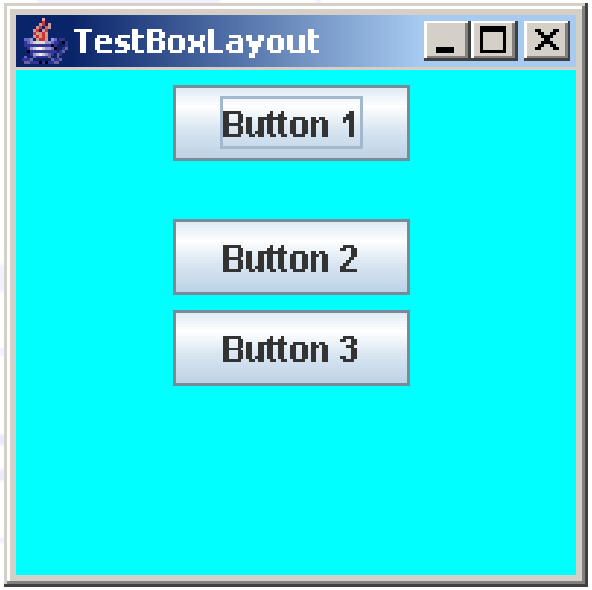
```
public BoxLayout(Container c, int direction);
```

- The direction can be `X_AXIS` or `Y_AXIS`
- Rigid areas and glue regions can be used to space out components within a `BoxLayout`

Example: Creating a `BoxLayout`

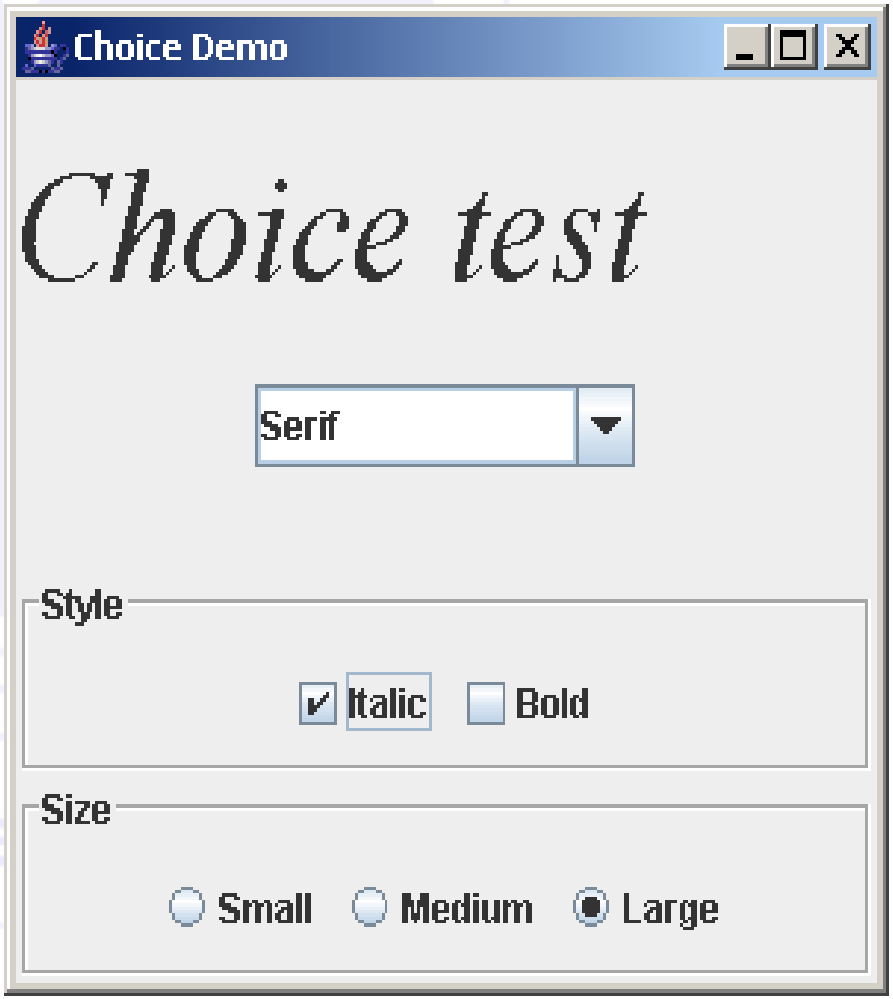
```
JFrame jf = new JFrame("TestBoxLayout");
jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
jf.setSize(new Dimension( 200, 200));
jf.setLocation(300, 300);
// Create a new panel
JPanel p = new JPanel();
// Set the layout manager
p.setLayout(new BoxLayout(p, BoxLayout.Y_AXIS));
// Add buttons
// leave some vertical space before button
p.add( Box.createRigidArea(new Dimension(0,5)) );
addAButton( "Button 1", p );
// vertical space between buttons
p.add( Box.createRigidArea(new Dimension(0,20)) );
addAButton( "Button 2", p );
p.add( Box.createRigidArea(new Dimension(0,5)) );
addAButton( "Button 3", p );
p.setBackground(Color.cyan);
// Add the new panel to the existing container
jf. add( p );
jf.setVisible(true);
```

```
private static void addAButton(
   String text, Container container)  {
  JButton button = new
                        JButton(text);
  button.setAlignmentX(
    Component.CENTER_ALIGNMENT);
  container.add(button);
}
```



Choices

- Radio buttons
- Check boxes
- Combo boxes



Radio Buttons

- For a small set of mutually exclusive choices, use radio buttons or a combo box
- In a radio button set, only one button can be selected at a time
- When a button is selected, previously selected button in set is automatically turned off

Radio Buttons

- Button group does not place buttons close to each other on container
- It is your job to arrange buttons on screen
- **`isSelected()`**: called to find out if a button is currently selected or not

```
if(largeButton.isSelected()) size = LARGE_SIZE;
```

- Call **`setSelected(true)`** on a radio button in group before making the enclosing frame visible

Borders

- Place a border around a panel to group its contents visually
- **`EtchedBorder`**: three-dimensional etched effect
- Can add a border to any component, but most commonly to panels:

```
Jpanel panel = new JPanel ();
panel.setBOrder(new EtchedBorder ());
```

- **`TitledBorder`**: a border with a title:

```
Panel.setBorder(new TitledBorder(new EtchedBorder(),
                "Size"));
```

Check Boxes

- Two states: checked and unchecked
- Use one checkbox for a binary choice
- Use a group of check boxes when one selection does not exclude another
- Example: "bold" and "italic" when choosing a font style
- Construct by giving the name in the constructor:

```
JCheckBox italicCheckBox = new JCheckBox("Italic");
```

- Don't place into a button group

Combo Boxes

- For a large set of choices, use a combo box
  - Uses less space than radio buttons
- "Combo": combination of a list and a text field
  - The text field displays the name of the current selection

Combo Boxes

- If combo box is editable, user can type own selection
  - Use `setEditable()` method
- Add strings with `addItem()` method:

```
JComboBox facenameCombo = new JComboBox();
facenameCombo.addItem("Serif");
facenameCombo.addItem("SansSerif");
. . .
```

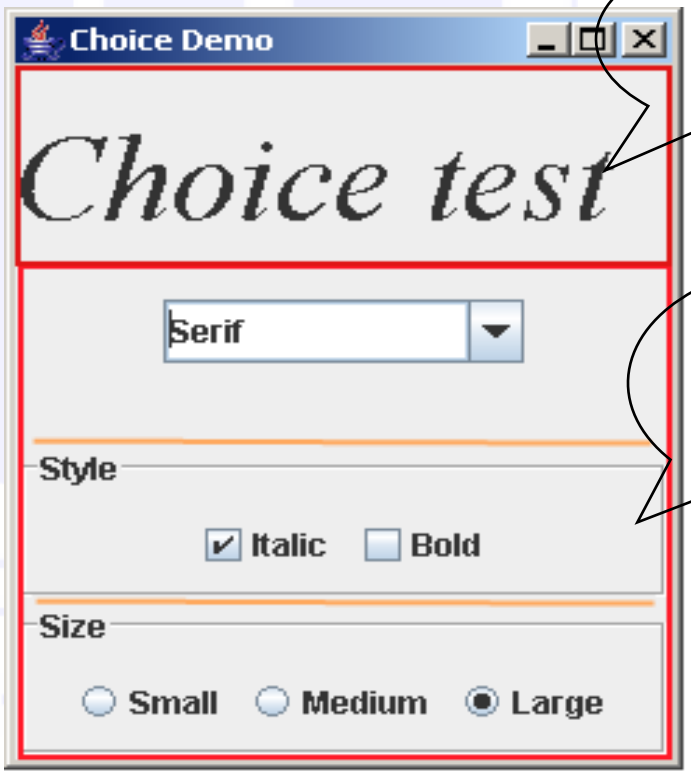
- Get user selection with `getSelectedItem()` (return type is `Object`)

```
String selectedString =
   (String) facenameCombo.getSelectedItem();
```

- Select an item with `setSelectedItem()`

Radio Buttons, Check Boxes, and Combo Boxes

- They generate an **`ActionEvent`** whenever the user selects an item
- An example: **`ChoiceFrame`**
  - All components notify the same listener object
  - When user clicks on any component, we ask each component for its current content
  - Then redraw text sample with the new font



BlueJ: ChoiceFrameViewer

Layout Management

- Step 1: Make a sketch of your desired component layout
- Step 2: Find groupings of adjacent components with the same layout
- Step 3: Identify layouts for each group
- Step 4: Group the groups together
- Step 5: Write the code to generate the layout

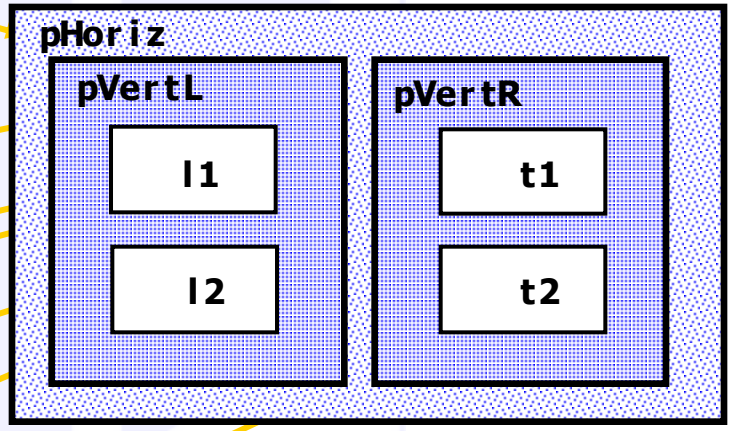
Combining Layout Managers to Produce a Result

- To create just the look you want, it is sometimes useful to create multiple containers inside each other, each with its own layout manager
- For example, a top-level panel might use a horizontal box layout, and that panel may contain two or more panels using vertical box layouts
- The result is complete control of component spacing in *both* dimensions

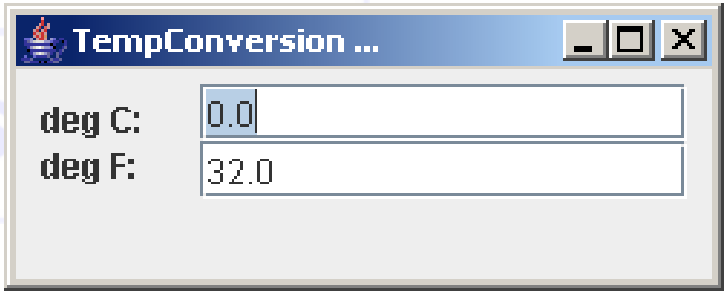
Example: Nested Containers and Layouts

```
// Create a new high-level panel
JPanel pHoriz = new JPanel();
pHoriz.setLayout(new BoxLayout(pHoriz,
BoxLayout.X_AXIS));
add( pHoriz );
// Create two subordinate panels
JPanel pVertL = new JPanel();
JPanel pVertR = new JPanel();
pVertL.setLayout(new BoxLayout(pVertL,
BoxLayout.Y_AXIS));
pVertR.setLayout(new BoxLayout(pVertR,
BoxLayout.Y_AXIS));
// Add to pHoriz with a horizontal space
between panels
pHoriz.add( pVertL );
pHoriz.add( Box.createRigidArea(new
Dimension(20,0)) );
pHoriz.add( pVertR );
// Create degrees Celsius field
l1 = new JLabel("deg C:", JLabel.RIGHT);
pVertL.add( l1 );
t1 = new JTextField("0.0",15);
t1.addActionListener( cHnd );
pVertR.add( t1 );
// Create degrees Fahrenheight field
l2 = new JLabel("deg F:", JLabel.RIGHT);
pVertL.add( l2 );
t2 = new JTextField("32.0",15);
t2.addActionListener( fHnd );
pVertR.add( t2 );
```

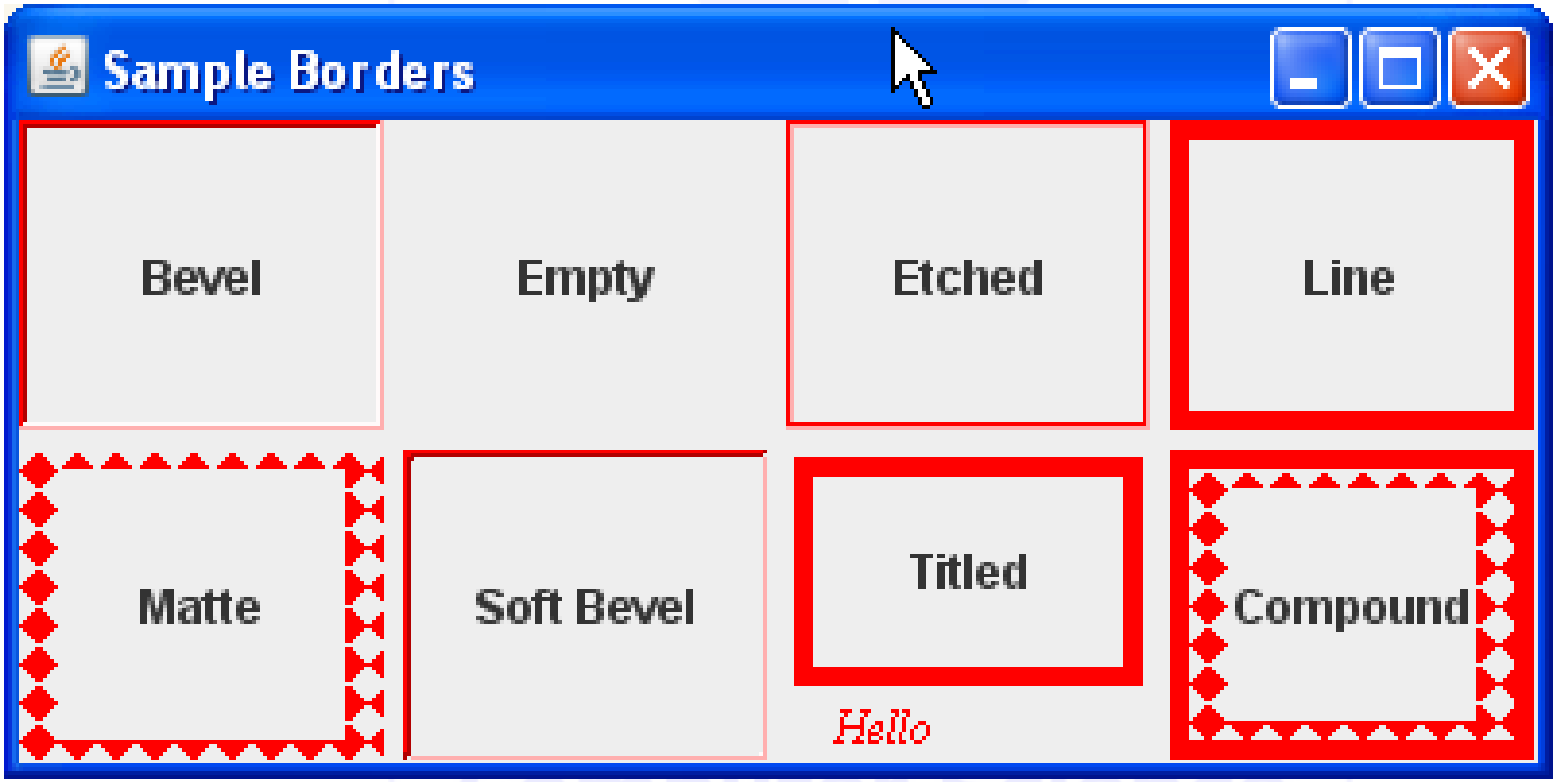
Structure:



Result:

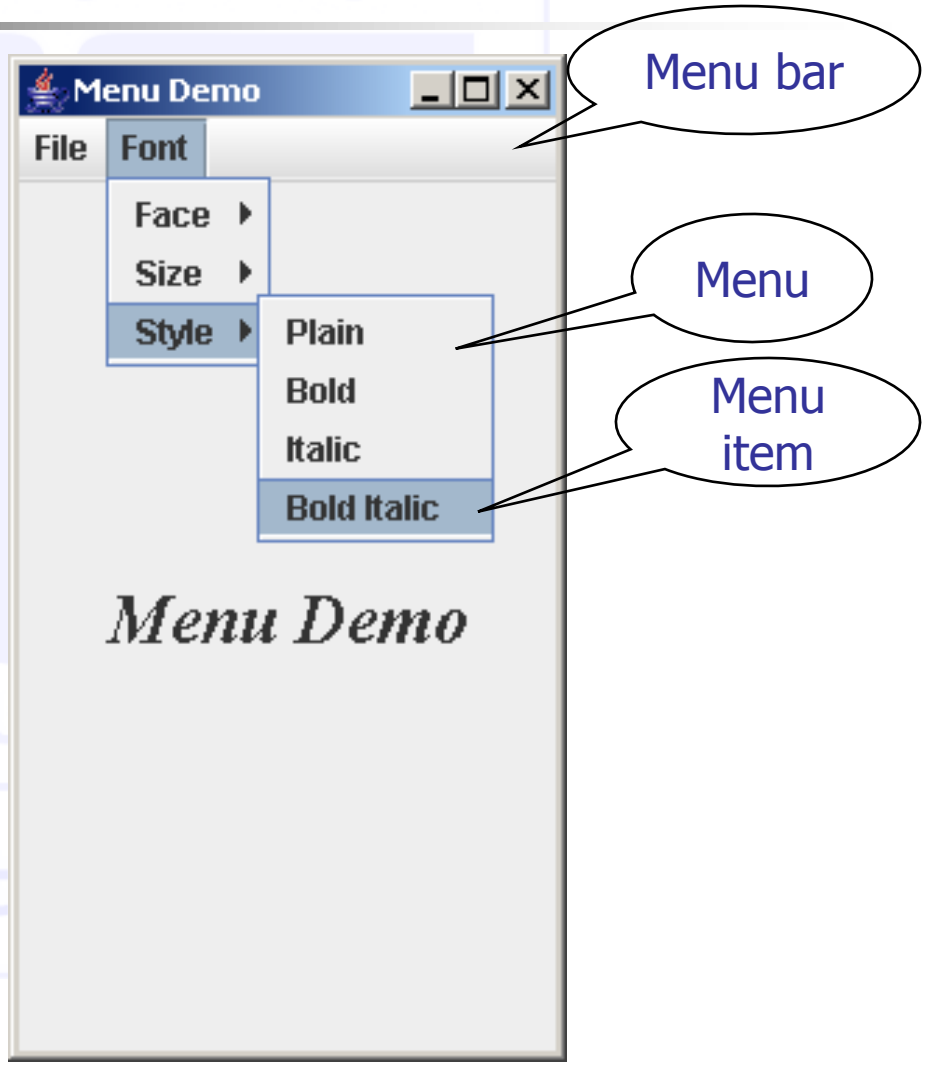


Borders (Swing)

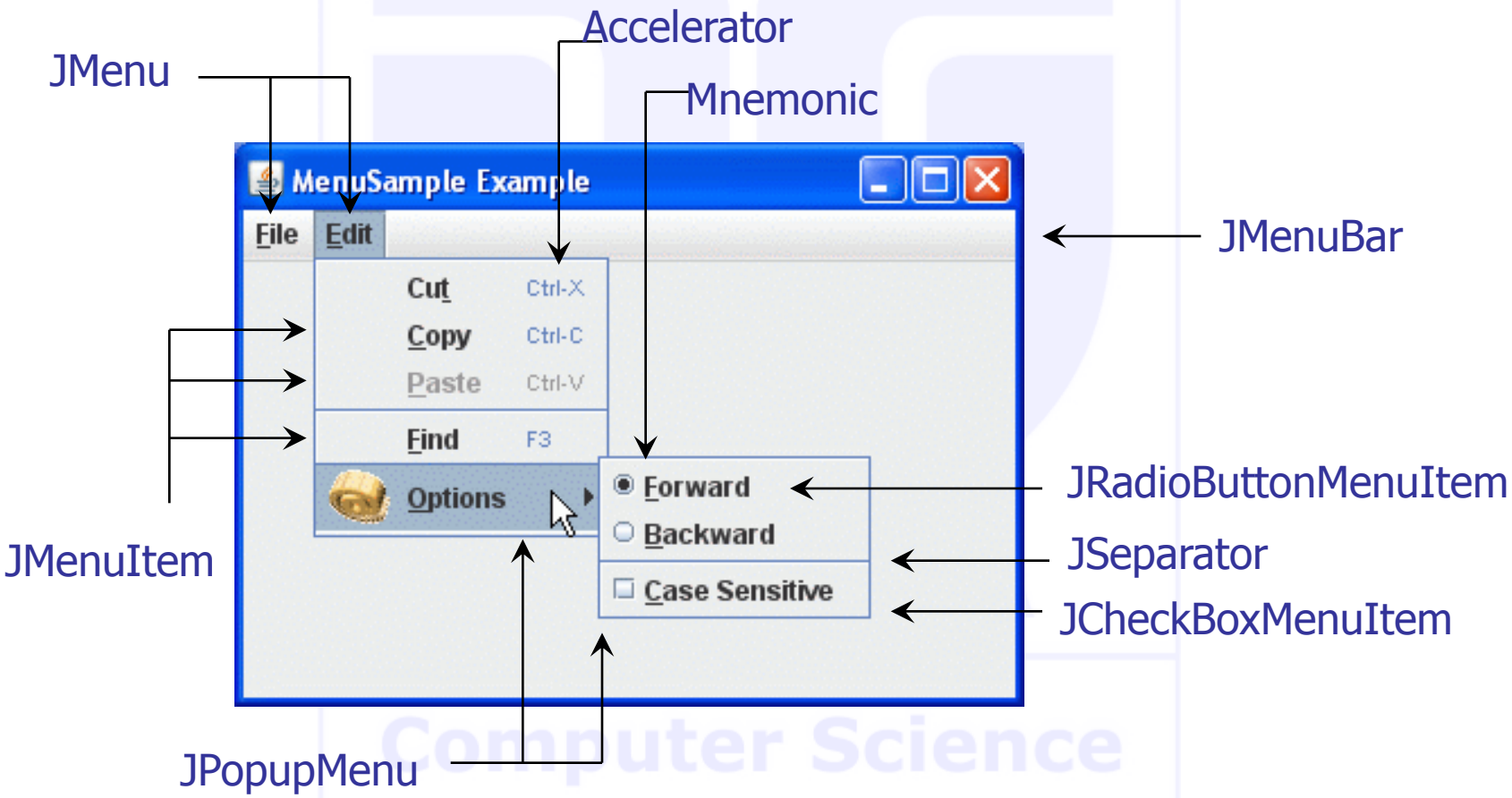


Menus

- A frame contains a menu bar
- The menu bar contains menus
- A menu contains submenus and menu items
  - Pull-Down Menus



Menus (Swing)


JMenu
Accelerator
Mnemonic
MenuSample Example
File
Edit
Cut Ctrl-X
Copy Ctrl-C
Paste Ctrl-V
Find F3
Options
Forward
Backward
Case Sensitive
JMenuBar
JRadioButtonMenuItem
JSeparator
JCheckBoxMenuItem
JMenuItem
JPopupMenu

Menu Items

- Add menu items and submenus with the **`add()`** method:

```
JMenuItem fileExitItem = new JMenuItem("Exit");
fileMenu.add(fileExitItem);
```

- A menu item has no further submenus
- Menu items generate action events
- Add a listener to each menu item:

```
fileExitItem.addActionListener(listener);
```

- Add action listeners only to menu items, not to menus or the menu bar

A Sample Program

- Builds up a small but typical menu
- Traps action events from menu items
- To keep program readable, use a separate method for each menu or set of related menus
  - `createFaceItem()`: creates menu item to change the font face
  - `createSizeItem()`
  - `createStyleItem()`

BlueJ MenuFrameViewer

Text Areas

- Use a **`JTextArea`** to show multiple lines of text
- You can specify the number of rows and columns:

```
final int ROWS = 10;
final int COLUMNS = 30;
JTextArea textArea = new JTextArea(ROWS, COLUMNS);
```

- The number of characters per line for a **`JTextField`** or **`JTextArea`** object is the number of *em* spaces
- An *em space* is the space needed to hold one uppercase letter **`M`** (widest in the alphabet)
  - A line specified to hold 20 **`M`** 's will almost always be able to hold more than 20 characters

Text Areas

- `setText()`: to set the text of a text field or text area
- `append()`: to add text to the end of a text area
- Use `newline` characters to separate lines:

```
textArea.append(account.getBalance() + "\n");
```

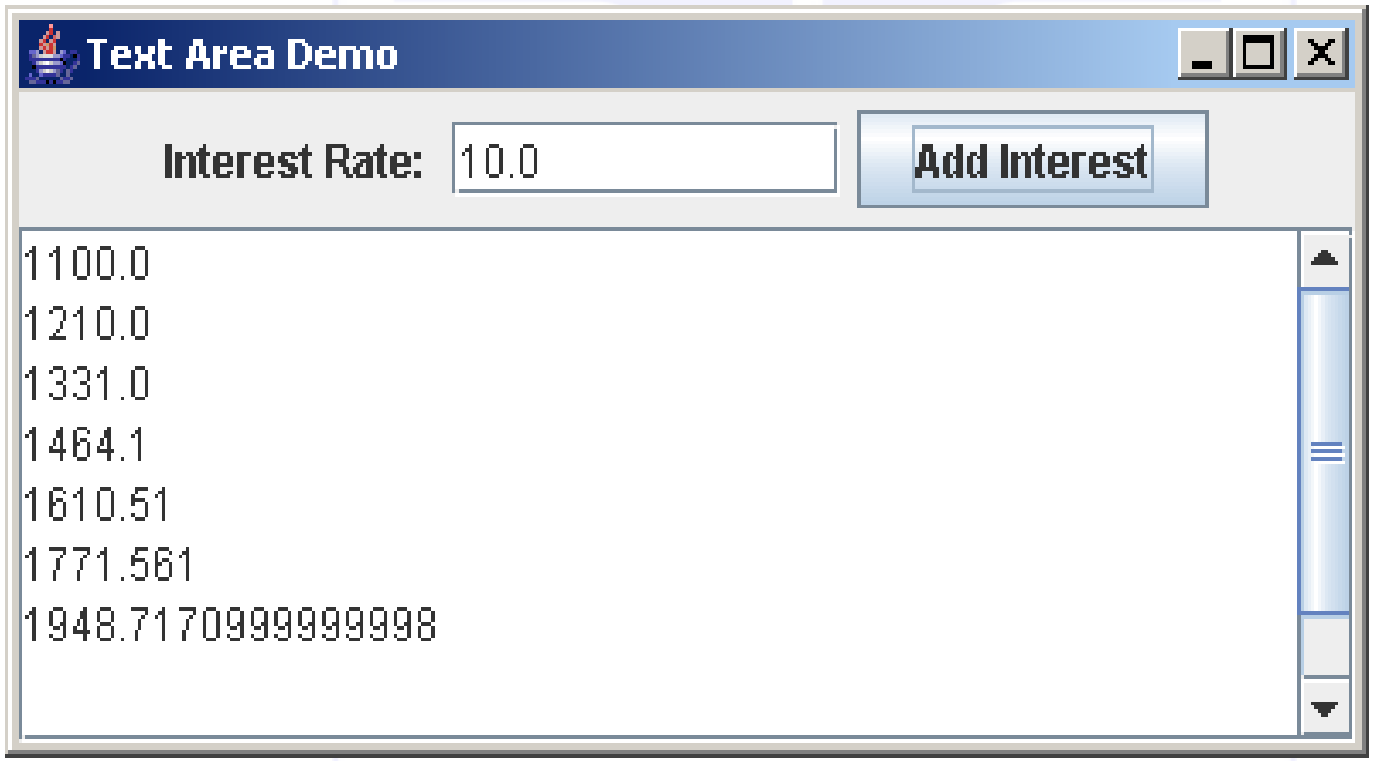
- To use for display purposes only:

```
textArea.setEditable(false);
// program can call setText and append to change it
```

- To add scroll bars to a text area:

```
JTextArea textArea = new JTextArea(ROWS, COLUMNS);
JScrollPane scrollPane = new JScrollPane(textArea);
```

Text Areas



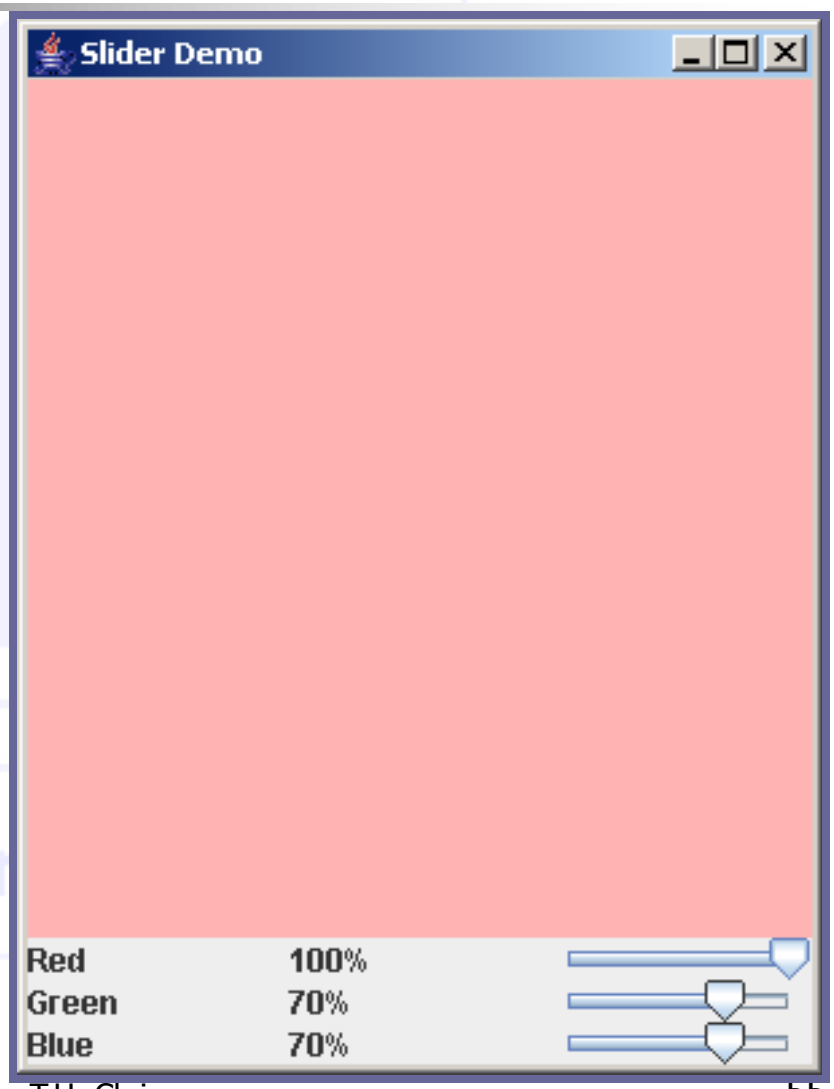
BlueJ TextAreaViewer

Exploring the Swing Documentation

- For more sophisticated effects, explore the Swing documentation
- The documentation can be quite intimidating at first glance
- Next example will show how to use the documentation to your advantage

Example: A Color Mixer

- It should be fun to mix your own colors, with a slider for the red, green, and blue values
- There are over 50 methods in `JSlider` class and over 250 inherited methods
- Some method descriptions look scary



How do I construct a `JSlider`?

- Look at the Java version 5.0 API documentation
- There are six constructors for the `JSlider` class
- Learn about one or two of them
- Strike a balance somewhere between the trivial and the bizarre
- Too limited: `public JSlider()`
  - Creates a horizontal slider with the range 0 to 100 and an initial value of 50
- Bizarre: `public JSlider(BoundedRangeModel brm)`
  - Creates a horizontal slider using the specified `BoundedRangeModel`
- Useful: `public JSlider(int min, int max, int value)`
  - Creates a horizontal slider using the specified min, max, and value.

How Can I Get Notified When the User Has Moved a `JSlider`?

- There is no `addActionListener` method
- There is a method

```
public void addChangeListener(ChangeListener l)
```

- Click on the `ChangeListener` link to learn more
- It has a single method:

```
void stateChanged(ChangeEvent e)
```

How Can I Get Notified When the User Has Moved a `JSlider`?

- Apparently, method is called whenever user moves the slider
- What is a `ChangeEvent`?
  - It inherits `getSource()` method from superclass `EventObject`
  - `getSource()`: tells us which component generated this event

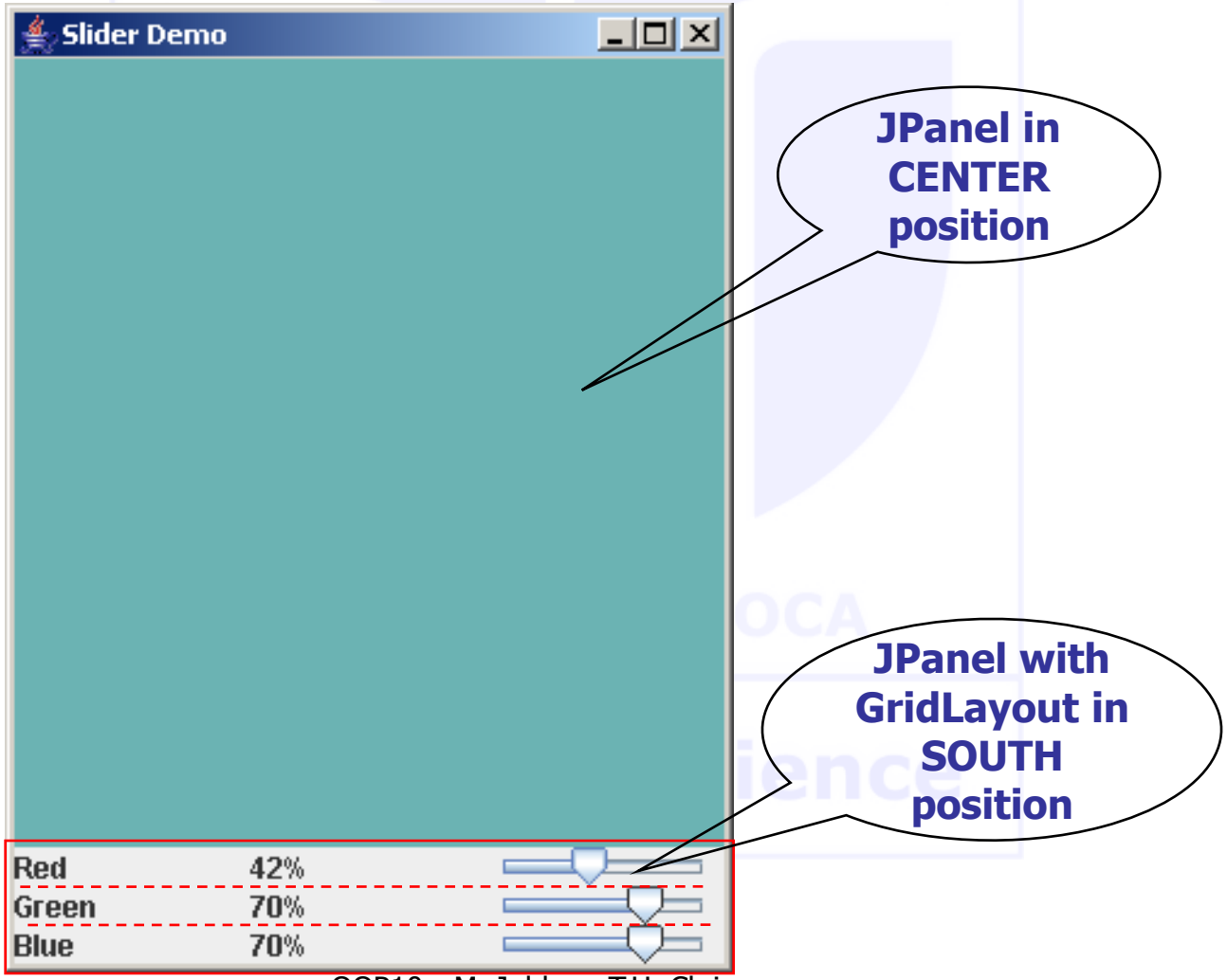
How Can I Tell to Which Value the User Has Set a `JSlider`?

- Now we have a plan:
  - Add a change event listener to each slider
  - When slider is changed, `stateChanged()` method is called
  - Find out the new value of the slider
  - Re-compute color value
  - Repaint color panel
- Need to get the current value of the slider
- Look at all the methods that start with get; you find:

```
public int getValue()
```

Returns the slider's value.

The Components of the `SliderFrame`



Icons

- `JLabels`, `JButtons`, and `JMenuItems` can have icons
  - An *icon* is just a small picture (usually)
  - It is not required to be small
- An icon is an object of the `ImageIcon` class
  - It is based on a digital picture file such as `.gif`, `.jpg`, or `.tiff`
- Labels, buttons, and menu items may display a string, an icon, a string and an icon, or nothing

Icons

- The class **`ImageIcon`** is used to convert a picture file to a Swing icon

```
ImageIcon dukeIcon = new
   ImageIcon("duke_waving.gif");
```

  - The picture file must be in the same directory as the class in which this code appears, unless a complete or relative path name is given
  - Note that the name of the picture file is given as a string
- An icon can be added to a label using the **`setIcon`** method as follows:

```
JLabel dukeLabel = new JLabel("Mood check");
dukeLabel.setIcon(dukeIcon);
```

Icons

- Instead, an icon can be given as an argument to the **JLabel** constructor:

```
JLabel dukeLabel = new JLabel(dukeIcon);
```

- Text can be added to the label as well using the **setText** method:

```
dukeLabel.setText("Mood check");
```

- Icons and text may be added to **JButtons** and **JMenuItems** in the same way as they are added to a **JLabel**

```
JButton happyButton = new JButton("Happy");
ImageIcon happyIcon = new ImageIcon("smiley.gif");
happyButton.setIcon(happyIcon);
```

Icons

- Button or menu items can be created with just an icon by giving the **`ImageIcon`** object as an argument to the **`JButton`** or **`JMenuItem`** constructor

```
ImageIcon happyIcon = new ImageIcon("smiley.gif");
JButton smileButton = new JButton(happyIcon);
JMenuItem happyChoice = new JMenuItem(happyIcon);
```

  - A button or menu item created without text should use the **`setActionCommand()`** method to explicitly set the action command, since there is no string

Summary

- GUI
- Containers and Components
- MVC and Swing
- Layout management
- Radio buttons
- Check boxes
- Combo boxes
- Grouping buttons
- Menus
- Text areas
- Exploring documentation: using JSlider
- Icons – setting icons and text