Chapter 14

GUI and Event-Driven Programming

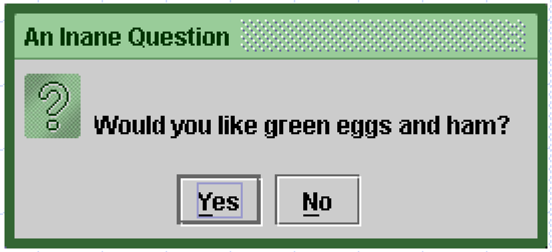
Graphical User Interface

- In Java, GUI-based programs are implemented by using classes from the **javax.swing** and **java.awt** packages.
- The Swing classes provide greater compatibility across different operating systems. They are fully implemented in Java, and behave the same on different operating systems.

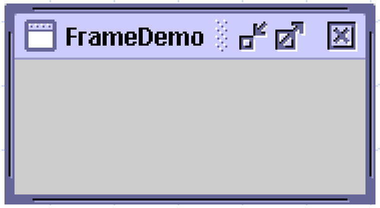
Swing Components

- Top-Level Containers
  The components at the top of any Swing containment hierarchy.
  General-Purpose Containers
  Intermediate containers that can be used under many different circumstances.
- Special-Purpose Containers
  Intermediate containers that play specific roles in the UI.
- Basic Controls
  Atomic components that exist primarily to get input from the user; they generally also show simple state.
- Uneditable Information Displays
  Atomic components that exist solely to give the user information.
- Interactive Displays of Highly Formatted Information
  Atomic components that display highly formatted information that (if you choose) can be modified by the user.

Top-Level Containers

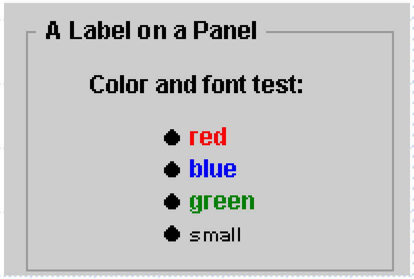


Dialog



Frame

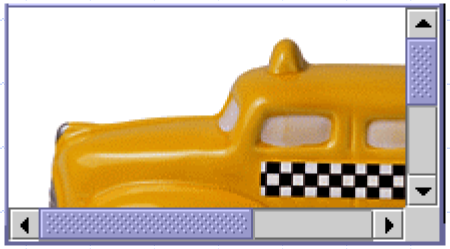
General-Purpose Containers

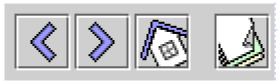

A Label on a Panel

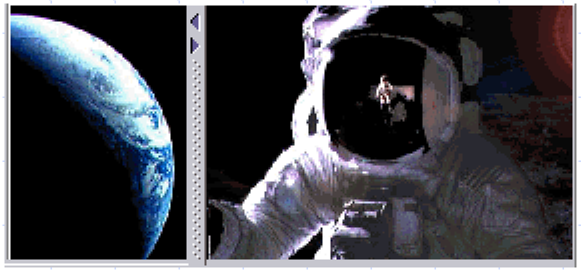
Color and font test:

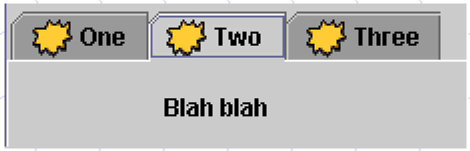
- red
- blue
- green
- small

Panel


Scroll pane


Tool bar

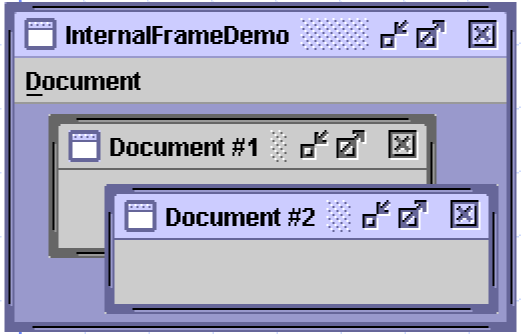

Split pane



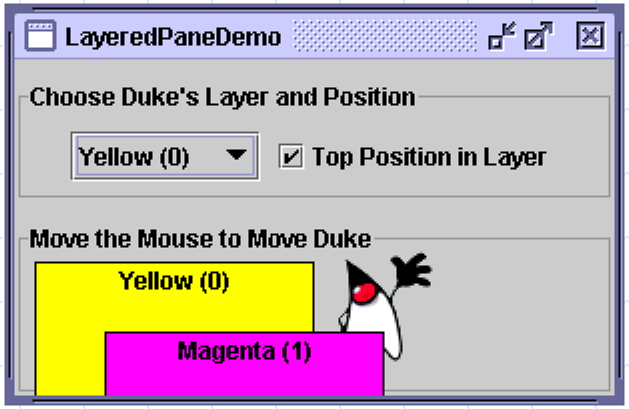
Blah blah

Tabbed pane

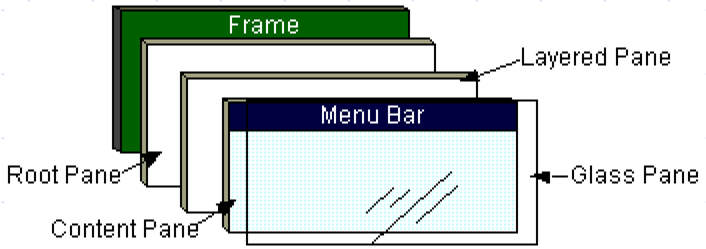
Special-Purpose Containers



Internal frame

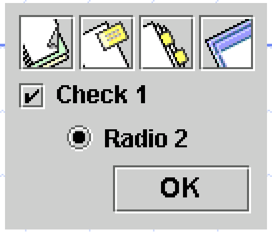


Layered pane

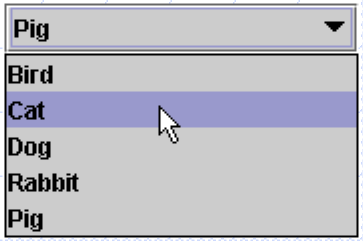


Root pane

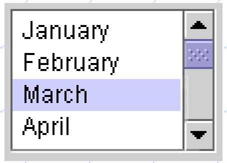
Basic Controls



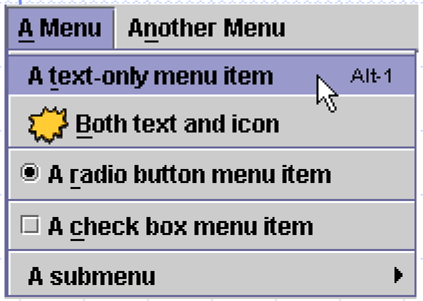
Buttons



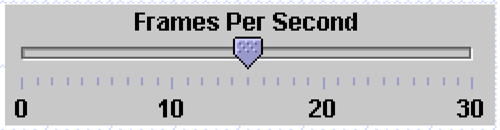
Combo Box



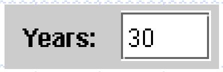
List



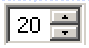
Menu



Slider

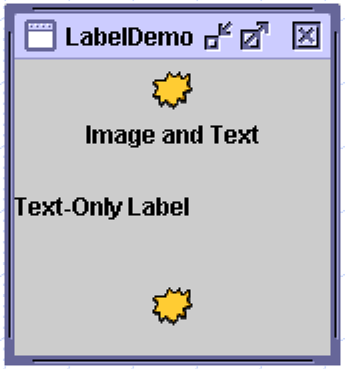


Text field
or
Formatted text field

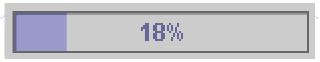


Spinner

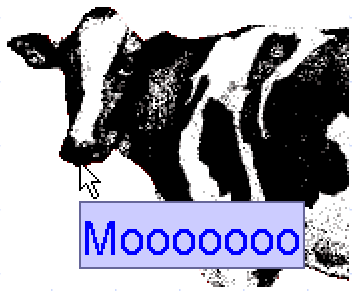
Uneditable Information Displays



Label

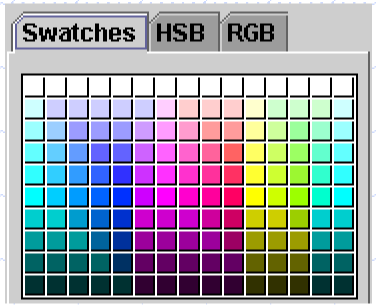


Progress bar

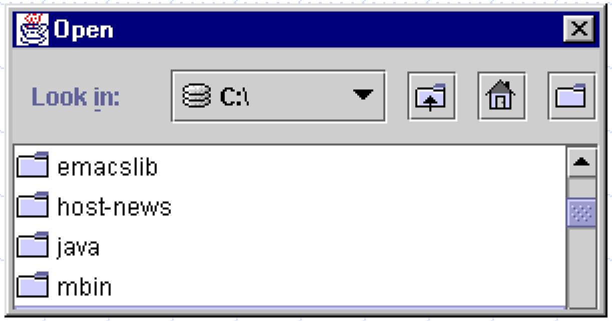


Tool tip

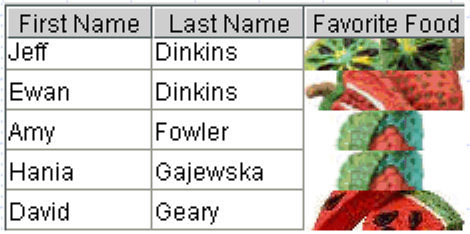
Interactive Displays of Highly Formatted Information



Color chooser

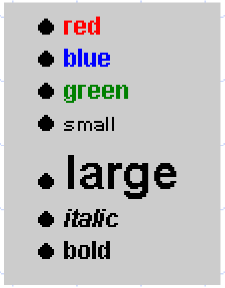


File chooser



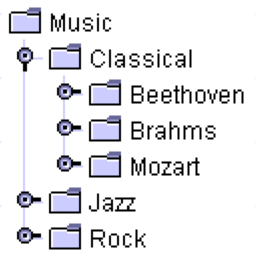
First Name	Last Name	Favorite Food
Jeff	Dinkins	
Ewan	Dinkins	
Amy	Fowler	
Hania	Gajewska	
David	Geary	

Table



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

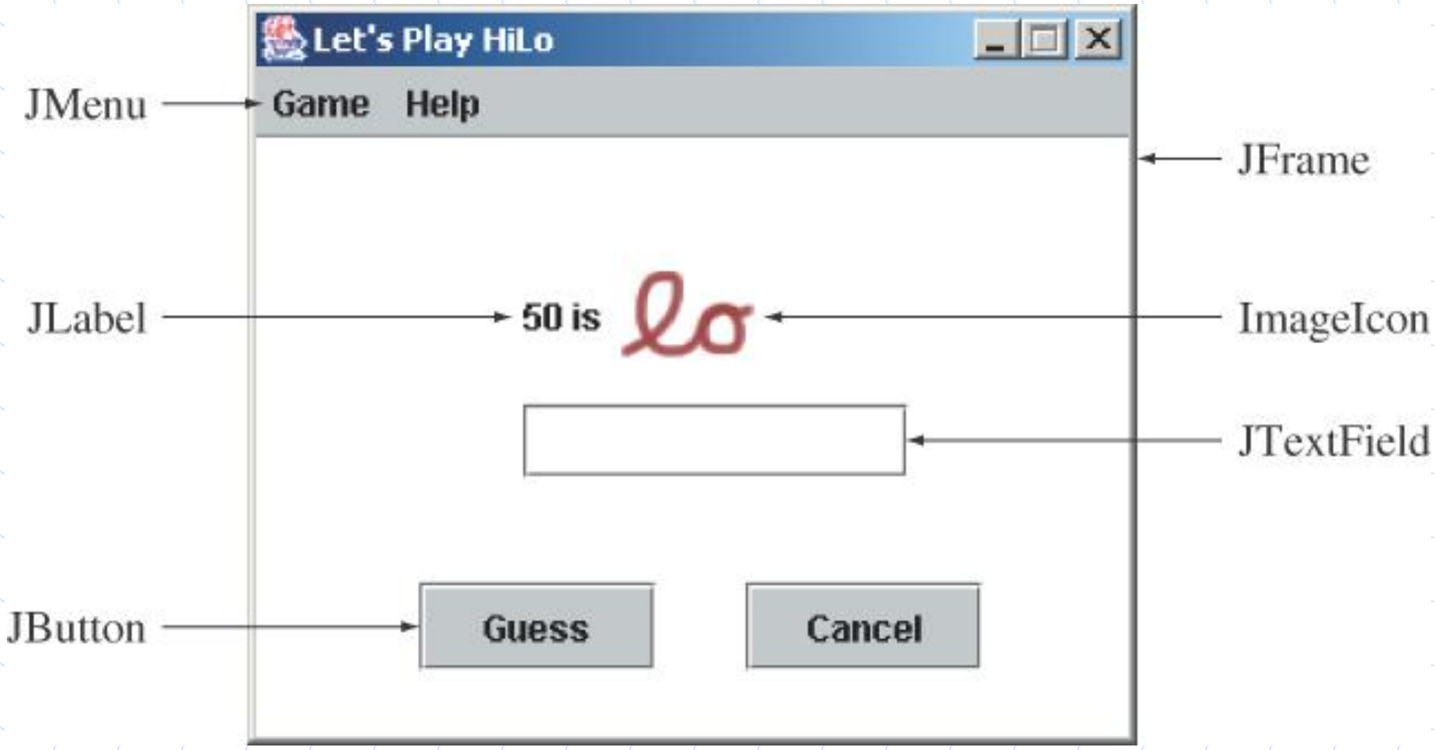
Text



Tree

Sample GUI Objects

- Various GUI objects from the **javax.swing** package.

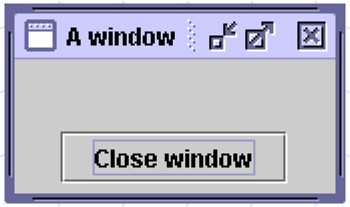


How to Make Frames (Main Windows)

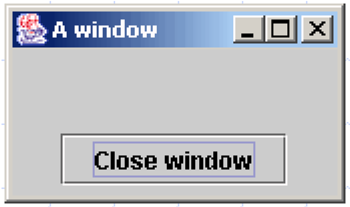
- //1. Optional: Specify who draws the window decorations.
  - **JFrame.setDefaultLookAndFeelDecorated(true);**
- //2. Create the frame.
  - **JFrame frame = new JFrame("FrameDemo");**
- //3. Optional: What happens when the frame closes?
  - **frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE**);
- //4. Create components and put them in the frame. *...create emptyLabel...*
  - **frame.getContentPane**().**add(emptyLabel**, **BorderLayout.CENTER**);
- //5. Size the frame.
  - **frame.pack**();
- //6. Show it.
  - **frame.setVisible(true**);

Specifying Window Decorations

```
JFrame.setDefaultLookAndFeelDecorated(true);
```

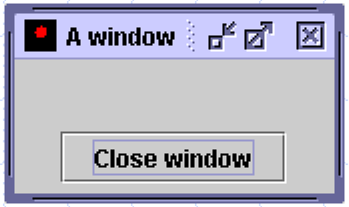


```
JFrame.setDefaultLookAndFeelDecorated(false);
```



Creating and Setting Up a Frame

JFrame()
JFrame(String)
	Create a frame that is initially invisible.
	The String argument is the title for the frame.

void setDefaultCloseOperation(int) int getDefaultCloseOperation()	Set or get the operation that occurs when the user pushes the close button on this frame. Possible choices are: DO_NOTHING_ON_CLOSE HIDE_ON_CLOSE DISPOSE_ON_CLOSE EXIT_ON_CLOSE
void setIconImage(Image) Image getIconImage()	Set or get the icon that represents the frame. 

Creating and Setting Up a Frame (2)

void setTitle(String) String getTitle()	Set or get the frame's title.
void setUndecorated(boolean) boolean isUndecorated()	Set or get whether the window system should provide decorations for this frame. Works only if the frame is not yet displayable (hasn't been packed or shown).
static void setDefaultLookAndFeelDecorated(boolean) static boolean isDefaultLookAndFeelDecorated()	Determine whether subsequently created JFrames should have their Window decorations (such as borders, widgets for closing the window, title) provided by the current look and feel.

Setting the Window Size and Location

void pack()	Size the window so that all its contents are at or above their preferred sizes.
void setSize(int, int) void setSize(Dimension) Dimension getSize()	Set or get the total size of the window. The integer arguments to setSize specify the width and height, respectively.
void setBounds(int, int, int, int) void setBounds(Rectangle) Rectangle getBounds()	Set or get the size and position of the window. The window's upper left corner is at the x, y location specified by the first two args, and has the width and height specified by the last two args.
void setLocation(int, int) Point getLocation()	Set or get the location of the upper left corner of the window. The parameters are the x and y values, respectively.

Creating a Plain **JFrame**

```
import javax.swing.*;
class SalamFrame {
   JFrame frame;
     public  void createAndShowGUI() {
   JFrame.setDefaultLookAndFeelDecorated(true);
   frame = new JFrame("Salam");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   JLabel label = new JLabel("Assalamo Alaikom");
       frame.getContentPane().add(label);
   frame.pack();
       frame.setVisible(true);
   }
}
```

Displaying the Window

```
import javax.swing.*;
class myMain {
     public  static void main(String[] rags) {
       SalamFrame myFrame = new SalamFrame();
       myFrame.createAndShowGUI();
    } //end main.
}
```

Subclassing **JFrame**

- To create a customized frame window, we define a subclass of the **JFrame** class.

- The **JFrame** class contains rudimentary functionalities to support features found in any frame window.

Creating a Subclass of **JFrame**

- To define a subclass of another class, we declare the subclass with the reserved word **extends**.

```
import javax.swing.*;

class myJFrameSubclass1 extends JFrame {
    . . .
}
```

Creating a Plain **JFrame**

```
import javax.swing.*;
class SalamJFrame extends JFrame {

    public  SalamJFrame() {
        super(“Salam”);
        JFrame.setDefaultLookAndFeelDecorated(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel label = new JLabel(“Assalamo Alaikom");
        getContentPane().add(label);
        pack();
    //  setVisible(true);
    }
}
```

Displaying the Window

```
import javax.swing.*;
class myMain {
     public  static void main(String[] rags) {
       SalamJFrame myFrame = new SalamJFrame();
       myFrame.setVisible(true);
   } //end main.
}
```

Customizing myJFrameSubclass1

- An instance of myJFrameSubclass1 will have the following default characteristics:
  - The title is set to **My First Subclass**.
  - The program terminates when the close box is clicked.
  - The size of the frame is 300 pixels wide by 200 pixels high.
  - The frame is positioned at screen coordinate (150, 250).
- These properties are set inside the default constructor.

Creating a Plain **JFrame**

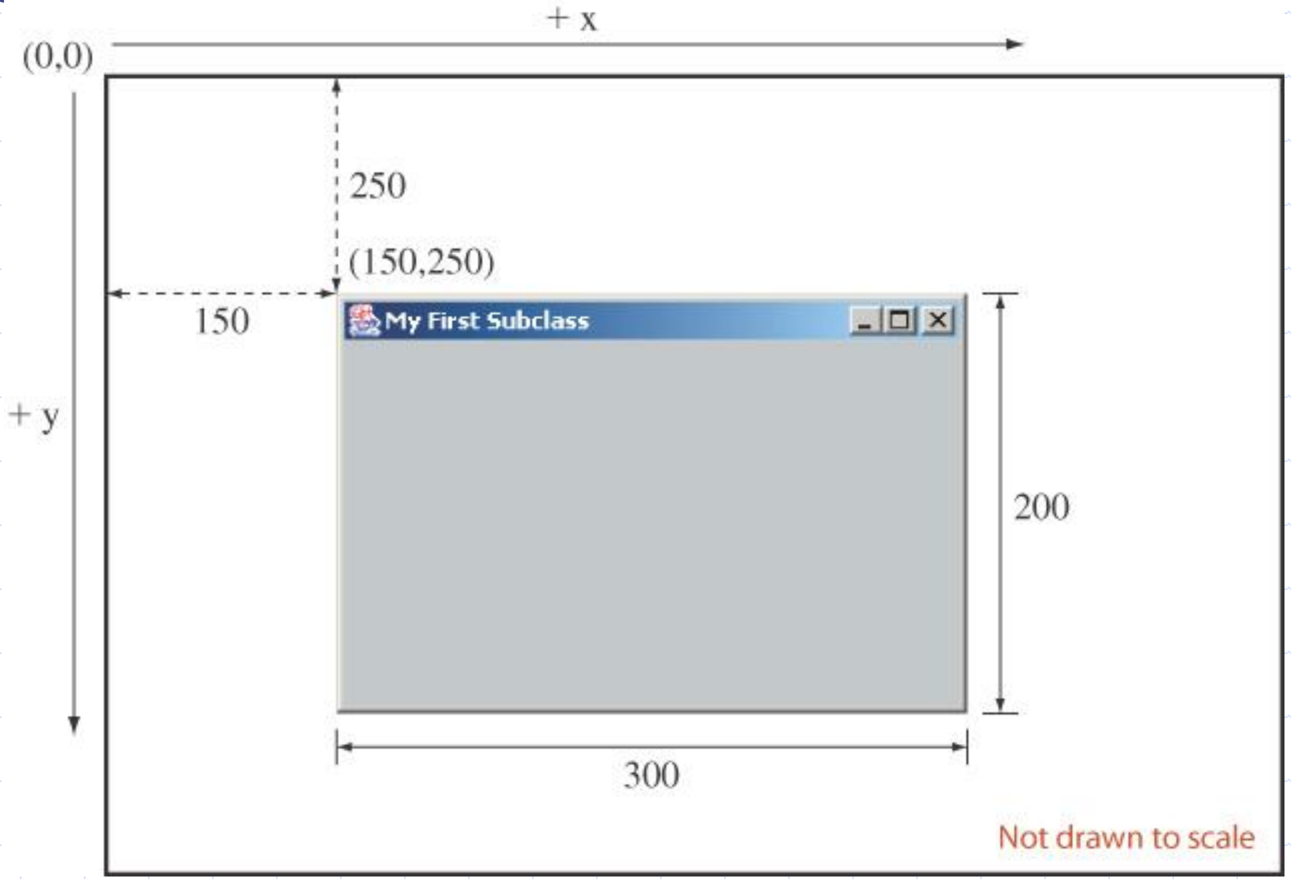
```
import javax.swing.*;
class SalamJFrame extends JFrame {

    public  SalamJFrame() {
       super("My First Subclass");
       JFrame.setDefaultLookAndFeelDecorated(true);
       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       JLabel label = new JLabel("Assalamo Alaikom");
       getContentPane().add(label);
       setBounds(150, 250, 300, 200);
       pack();
   //  setVisible(true);
    }
}
```

Displaying the Window

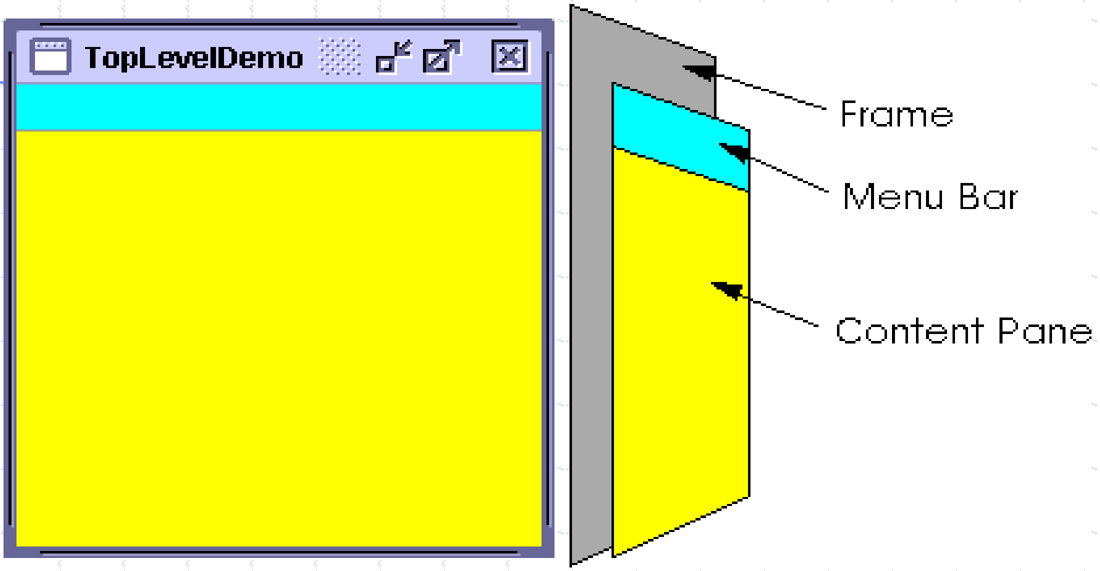
```
import javax.swing.*;
class myMain {
     public  static void main(String[] rags) {
   SalamJFrame myFrame = new SalamJFrame();
      myFrame.setVisible(true);
   } //end main.
}
```

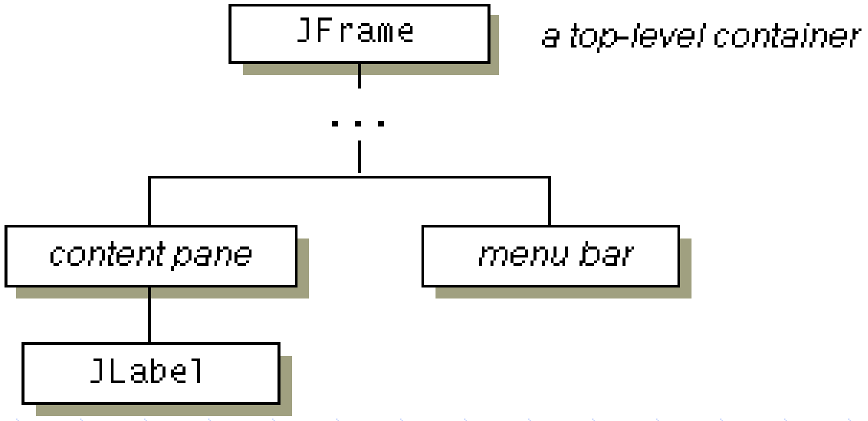
Displaying myJFrameSubclass1

- Here's how a **myJFrameSubclass1** frame window will appear on the screen.



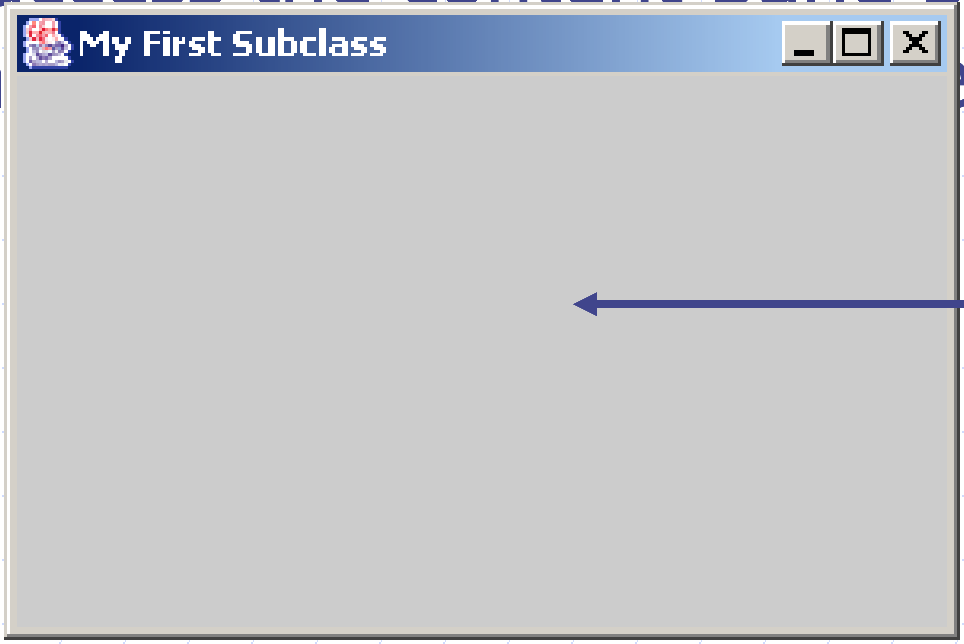
Use of Top Level Container





The Content Pane of a Frame

- The content pane is where we put GUI objects such as buttons, labels, scroll bars, and others.
- We access the content pane by calling the fram ethod.

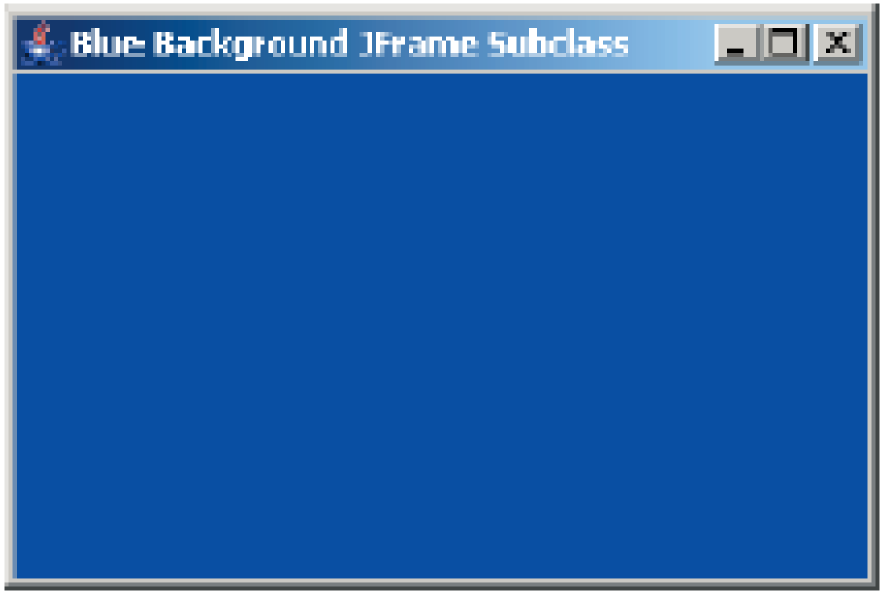


This gray area is the content pane of this frame.

Changing the Background Color

- Here's how we can change the background color of a content pane to blue:

```
  Container contentPane = myFrame.getContentPane();
  contentPane.setBackground(Color.BLUE);
```



Adding Components to the Content Pane

```
myFrame.getContentPane().add(myLabel,BorderLayout.CENTER);
```

```
JPanel contentPane = new JPanel(new BorderLayout());
contentPane.setBorder(someBorder);
contentPane.add(someComponent, BorderLayout.CENTER);
contentPane.add(anotherComponent, BorderLayout.PAGE_END);
//Make it the content pane.
contentPane.setOpaque(true);
myFrame.setContentPane(contentPane);
```

Adding Components to the Content Pane

```
myFrame.getContentPane().add(myLabel,BorderLayout.CENTER);
```

```
JPanel contentPane = myFrame.getContentPane();
contentPane.setBorder(someBorder);
contentPane.add(someComponent, BorderLayout.CENTER);
contentPane.add(anotherComponent, BorderLayout.PAGE_END);
//Make it the content pane.
contentPane.setOpaque(true);
```

Placing GUI Objects on a Frame

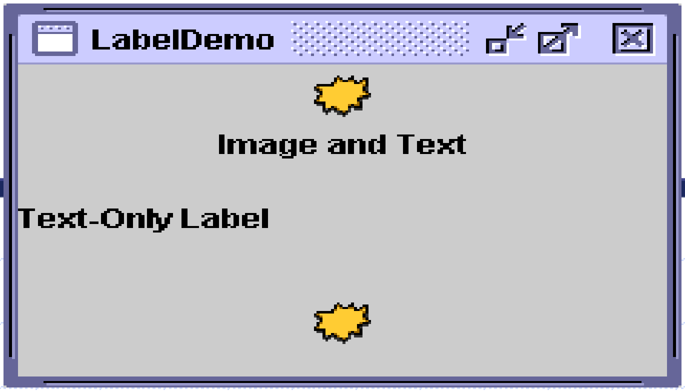
- There are two ways to put GUI objects on the content pane of a frame:
  - Use a *layout manager*
    - FlowLayout
    - BorderLayout
    - GridLayout
  - Use *absolute positioning*
    - null layout manager

JPanel API

Method	Purpose
void add(Component) void add(Component, int) void add(Component, Object) void add(Component, Object, int) void add(String, Component)	Add the specified component to the panel. When present, the int parameter is the index of the component within the container. By default, the first component added is at index 0, the second is at index 1, and so on. The Object parameter is layout manager dependent and typically provides information to the layout manager regarding positioning and other layout constraints for the added component. The String parameter is similar to the Object parameter.
int getComponentCount()	Get the number of components in this panel.
Component getComponent(int) Component getComponentAt(int, int) Component getComponentAt(Point) Component[] getComponents()	Get the specified component or components. You can get a component based on its index or *x*, *y* position.
void remove(Component) void remove(int) void removeAll()	Remove the specified component(s).

Use of Labels

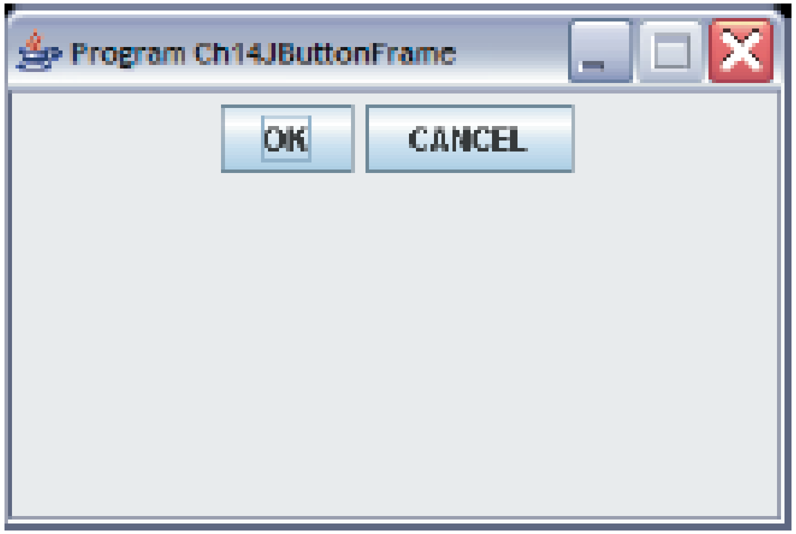
```
ImageIcon icon = createImageIcon("images/middle.gif");
label1 = new JLabel("Image and Text", icon,JLabel.CENTER);
//Set the position of the text, relative to the icon:
label1.setVerticalTextPosition(JLabel.BOTTOM);
label1.setHorizontalTextPosition(JLabel.CENTER);
label2 = new JLabel("Text-Only Label");
label3 = new JLabel(icon);
```



Placing a Button

- A **JButton** object is a GUI component that represents a pushbutton.
- Here's an example of how we place a button with **FlowLayout**.

```
contentPane.setLayout(
        new FlowLayout());
okButton
    = new JButton("OK");
cancelButton
    = new JButton("CANCEL");
contentPane.add(okButton);
contentPane.add(cancelButton);
```



Event Handling

- An action involving a GUI object, such as clicking a button, is called an *event.*
- The mechanism to process events is called *event handling*.
- The event-handling model of Java is based on the concept known as the *delegation-based event model.*
- With this model, event handling is implemented by two types of objects:
  - event source objects
  - event listener objects

Event Source Objects

- An event source is a GUI object where an event occurs. We say an event source generates events.
  - **Buttons,**
  - **text boxes,**
  - **list boxes,**
  - **menus**

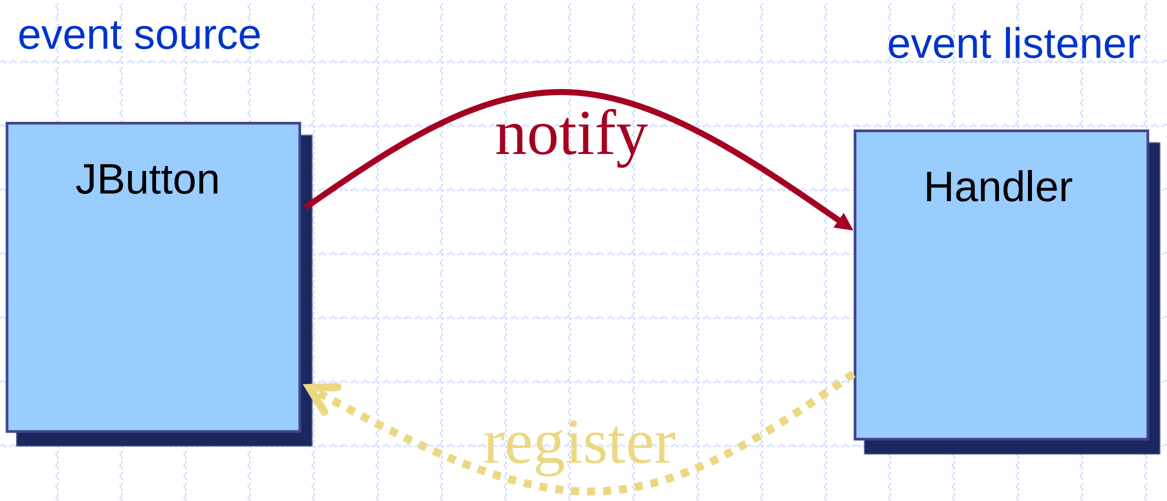
  are common event sources in GUI-based applications.

Event Listener Objects

- An event listener object is an object that includes a method that gets executed in response to the generated events.

- A listener must be associated, or registered, to a source, so it can be notified when the source generates events.

Connecting Source and Listener

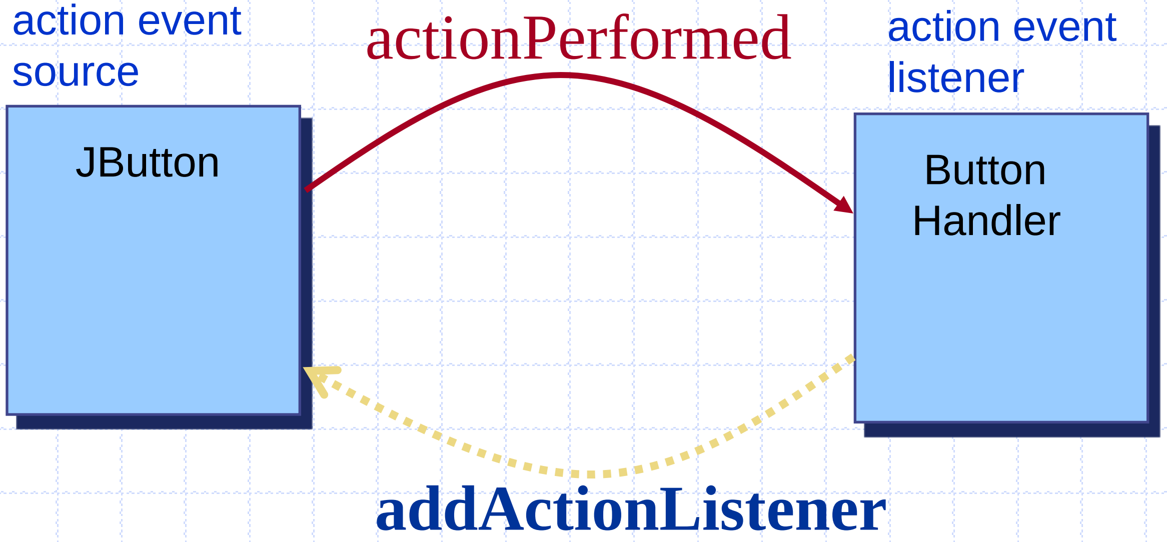


A listener must be **registered** to a event source. Once registered, it will get **notified** when the event source generates events.

Event Types

- Registration and notification are specific to event types
    - **Mouse listener handles mouse events**
    - **Item listener handles item selection events**
    - **and so forth**
- Among the different types of events, the action event is the most common.
    - Clicking on a button generates an action event
    - Selecting a menu item generates an action event
    - and so forth
- Action events are generated by action event sources and handled by action event listeners.

Handling Action Events



```
JButton button = new JButton("OK");
ButtonHandler handler = new ButtonHandler( );


button.addActionListener(handler);
```

ActionListener Interface

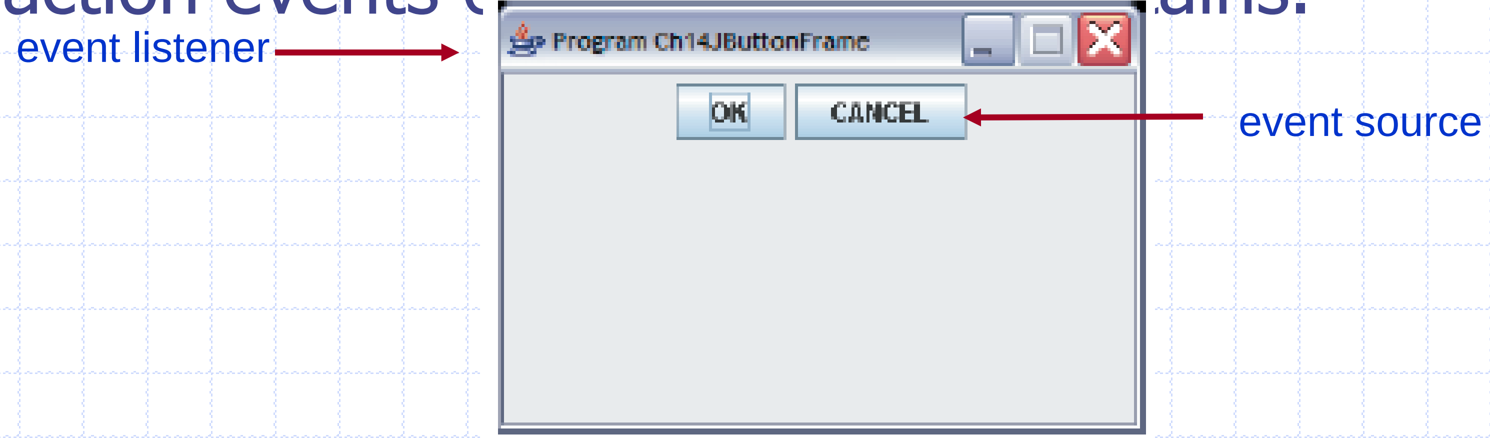
- When we call the **addActionListener** method of an event source, we must pass an instance of a class that implements the **ActionListener** interface.
- The ActionListener interface includes one method named **actionPerformed**.
- A class that implements the ActionListener interface must therefore provide the method body of **actionPerformed**.
- Since actionPerformed is the method that will be called when an action event is generated, this is the place where we put a code we want to be executed in response to the generated events.

The ButtonHandler Class

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

class ButtonHandler implements ActionListener {
    . . .
    public void actionPerformed(ActionEvent event) {
        JButton clickedButton = (JButton) event.getSource();

        JRootPane rootPane = clickedButton.getRootPane( );
        Frame     frame    = (JFrame) rootPane.getParent();

        frame.setTitle("You clicked " + clickedButton.getText());
    }
}
```

Container as Event Listener

- Instead of defining a separate event listener such as ButtonHandler, it is much more common to have an object that contains the event sources be a listener.
  - Example: We make this frame a listener of the action events of the buttons it contains.



Ch14JButtonFrameHandler

```
. . .
class myJButtonFrameHandler extends JFrame
                                 implements ActionListener {

   . . .

   public void actionPerformed(ActionEvent event) {
        JButton clickedButton
                    = (JButton) event.getSource();


        String  buttonText = clickedButton.getText();


        setTitle("You clicked " + buttonText);
    }
}
```

GUI Classes for Handling Text

- The Swing GUI classes **JLabel**, **JTextField**, and **JTextArea** deal with text.

- A **JLabel** object displays uneditable text (or image).
- A **JTextField** object allows the user to enter a single line of text.
- A **JTextArea** object allows the user to enter multiple lines of text. It can also be used for displaying multiple lines of uneditable text.

JTextField

- We use a JTextField object to accept a single line to text from a user. An action event is generated when the user presses the ENTER key.
- The getText method of JTextField is used to retrieve the text that the user entered

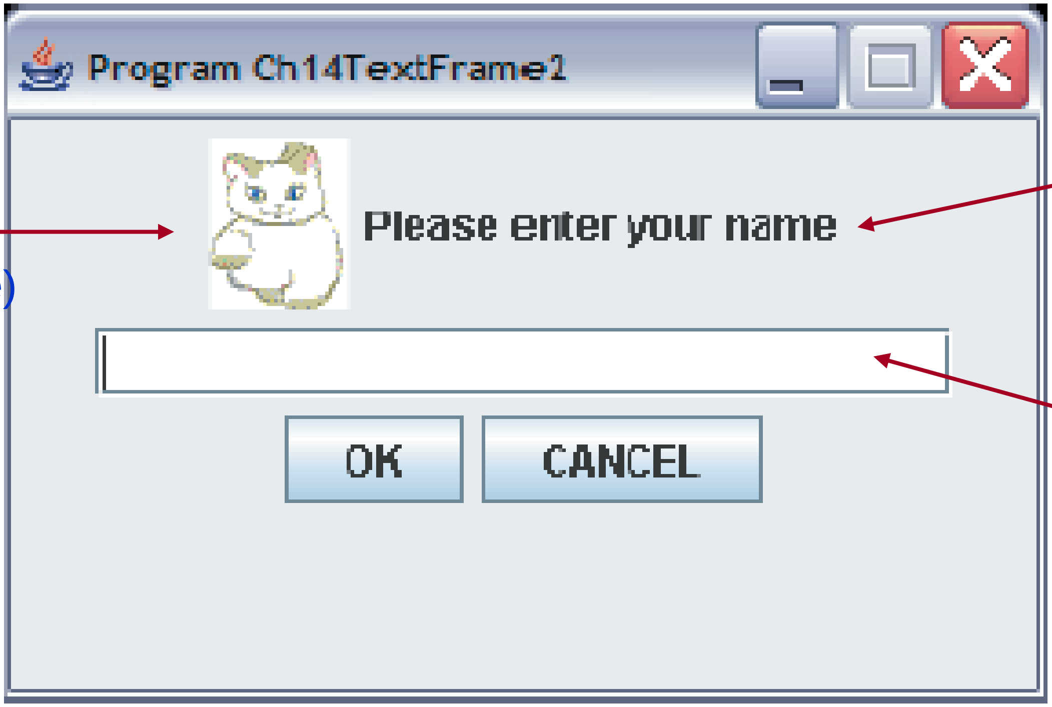
```
JTextField input = new JTextField( );
input.addActionListener(eventListener);
contentPane.add(input);
```

JLabel

- We use a JLabel object to display a label.
- A label can be a text or an image.
- When creating an image label, we pass ImageIcon object instead of a string.

```
JLabel textLabel = new JLabel("Please enter your name");
contentPane.add(textLabel);


JLabel imgLabel = new JLabel(new ImageIcon("cat.gif"));
contentPane.add(imgLabel);
```

Ch14TextFrame2



JLabel
(with an image)

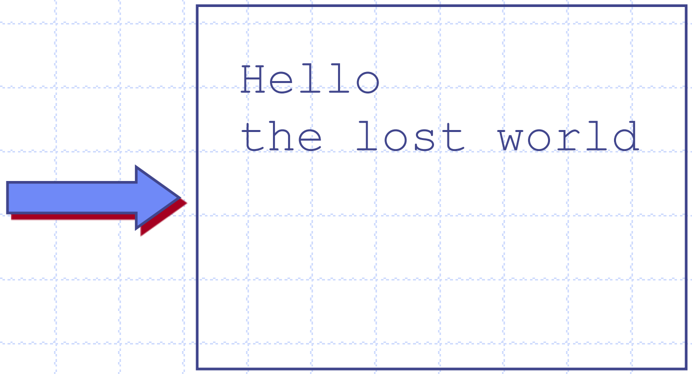
JLabel
(with a text)

JTextField

JTextArea

- We use a JTextArea object to display or allow the user to enter multiple lines of text.
- The setText method assigns the text to a JTextArea, replacing the current content.
- The append method appends the text to the current text.

```
JTextArea textArea
      = new JTextArea( );
. . .
textArea.setText("Hello\n");
textArea.append("the lost ");
textArea.append("world");
```

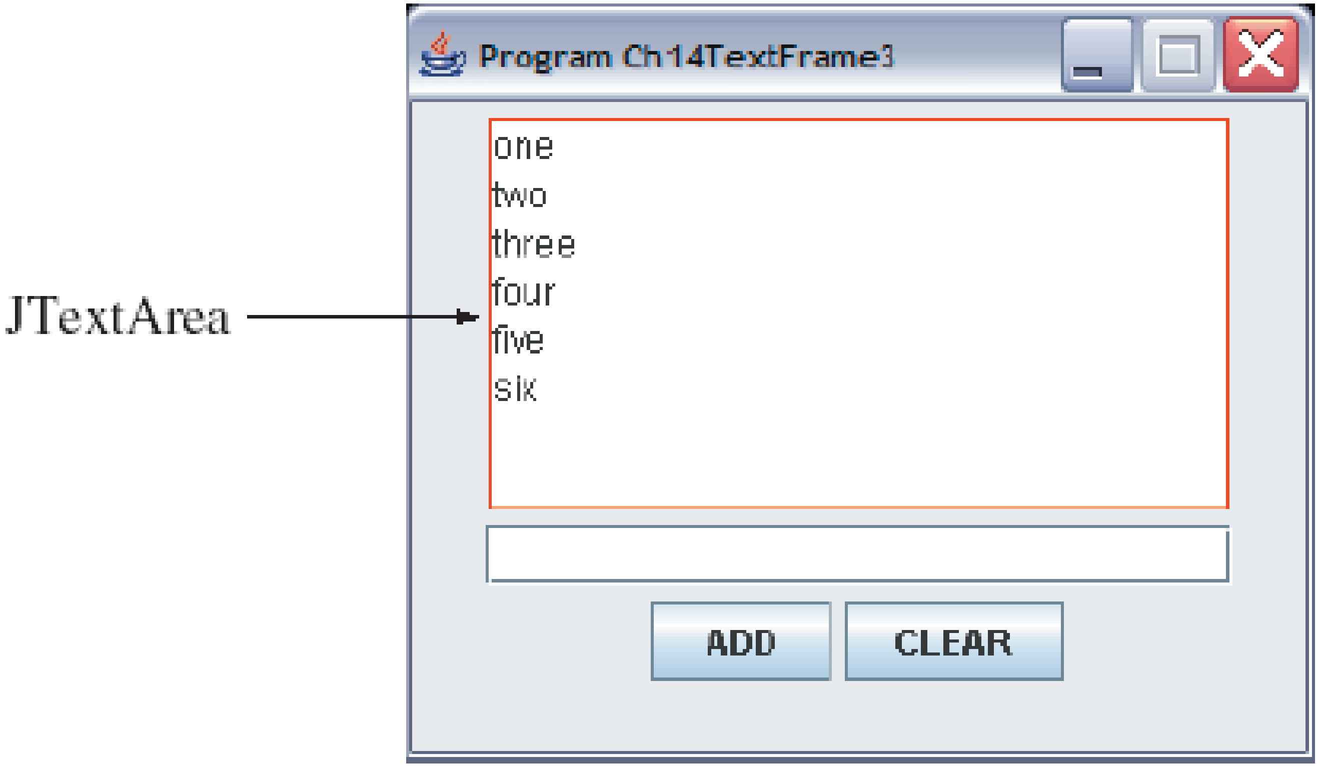


```
Hello
the lost world
```

JTextArea

Ch14TextFrame3

- The state of a **Ch14TextFrame3** window after six words are entered.



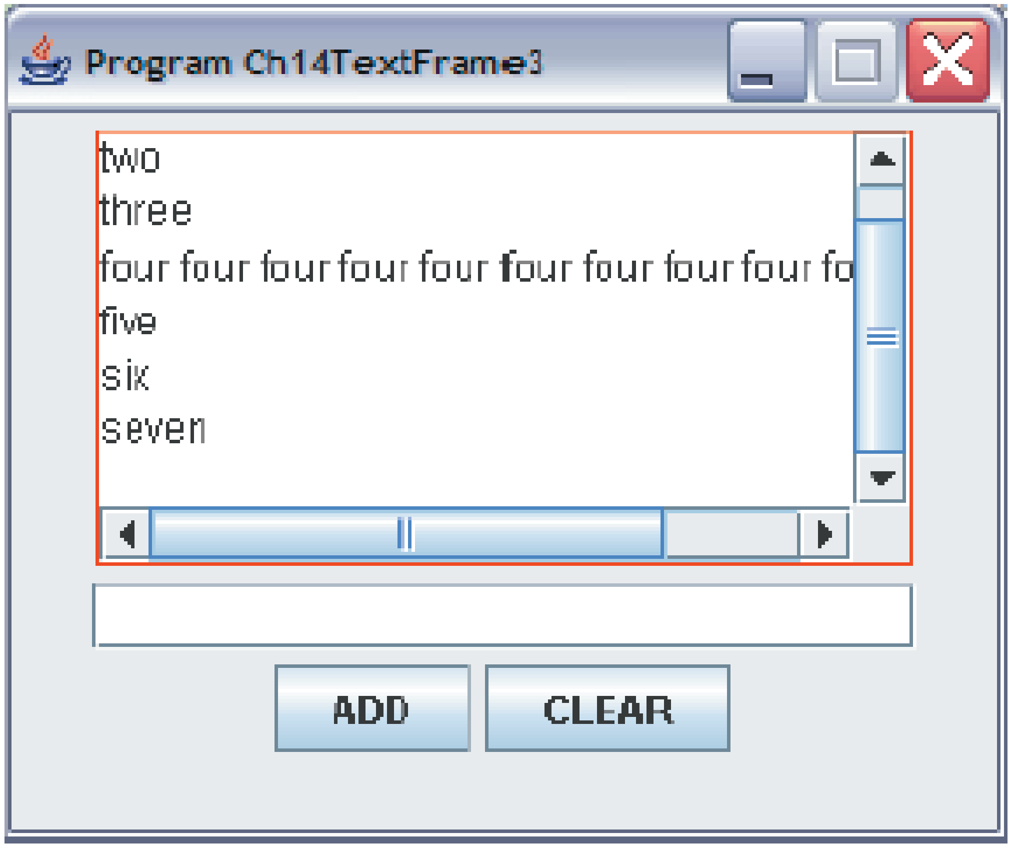
Adding Scroll Bars to JTextArea

- By default a JTextArea does not have any scroll bars. To add scroll bars, we place a JTextArea in a JScrollPane object.

```
JTextArea   textArea   = new JTextArea();
. . .
JScrollPane scrollText = new JScrollPane(textArea);
. . .
contentPane.add(scrollText);
```

Ch14TextFrame3 with Scroll Bars

- A sample Ch14TextFrame3 window when a JScrollPane is used.



```
two
three
four four four four four four four four four fo
five
six
seven
```

Layout Managers

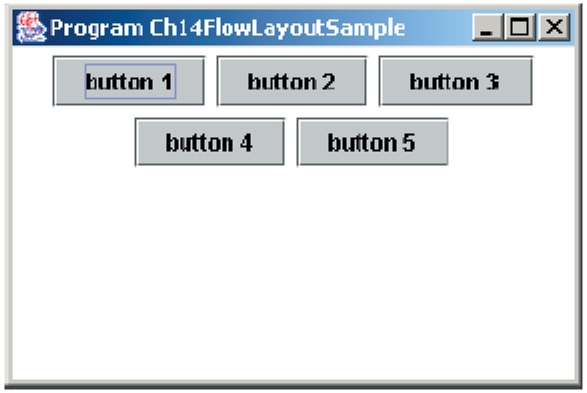
- The layout manager determines how the GUI components are added to the container (such as the content pane of a frame)
- Among the many different layout managers, the common ones are
  - FlowLayout (see Ch14FlowLayoutSample.java)
  - BorderLayout (see Ch14BorderLayoutSample.java)
  - GridLayout (see

FlowLayout

- In using this layout, GUI components are placed in left-to-right order.
  - When the component does not fit on the same line, left-to-right placement continues on the next line.
- As a default, components on each line are centered.
- When the frame containing the component is resized, the placement of components is adjusted accordingly.

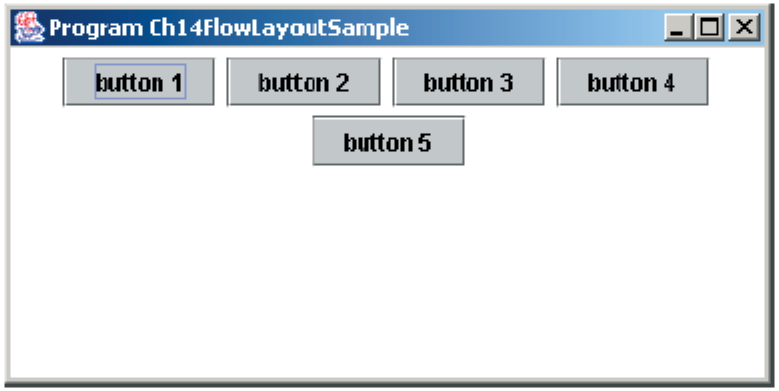
FlowLayout Sample

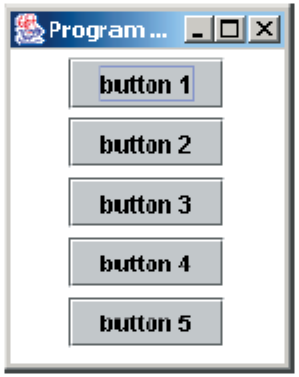
This shows the placement of five buttons by using FlowLayout.



Center alignment is used as a default. It can be set to a different alignment at the time a FlowLayout is created.

When the frame first appears on the screen.





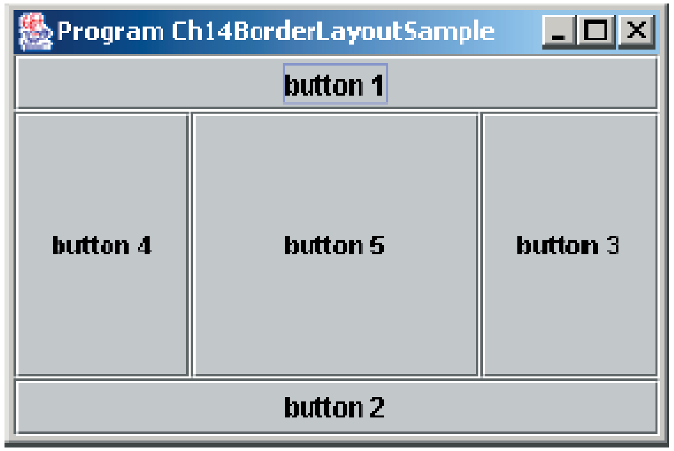
After the frame's width is widened and shortened.

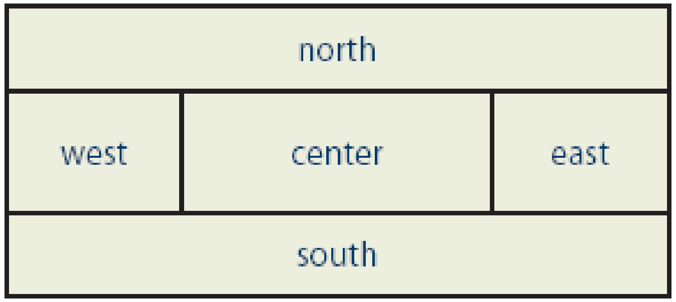
BorderLayout

- This layout manager divides the container into five regions: center, north, south, east, and west.
- The north and south regions expand or shrink in height only
- The east and west regions expand or shrink in width only
- The center region expands or shrinks on both height and width.
- Not all regions have to be occupied.

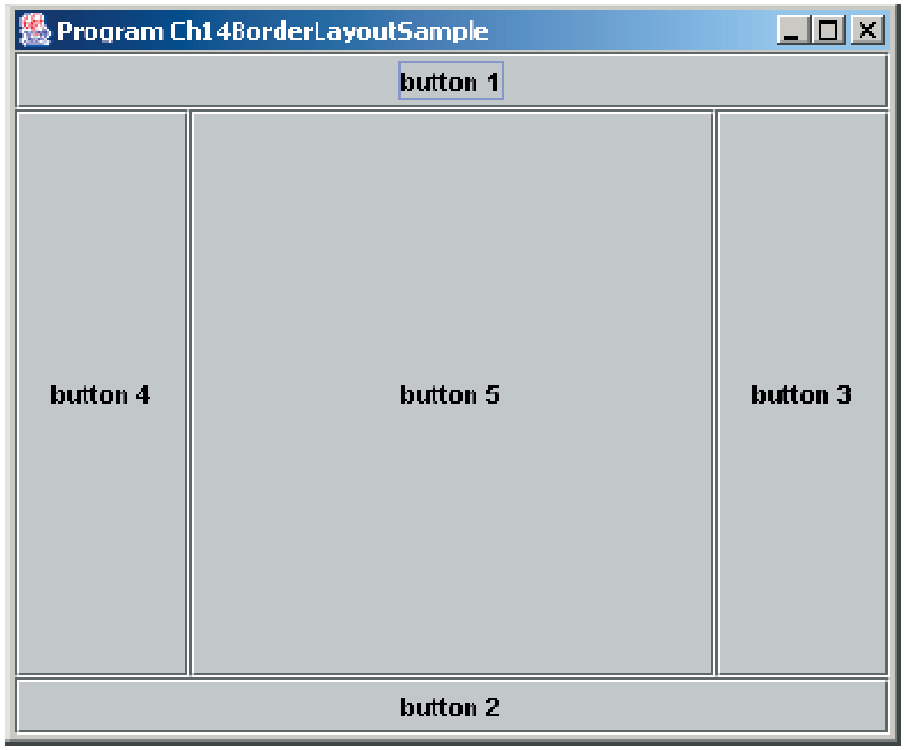
BorderLayout Sample

When the frame first appears on the screen.





After the frame is resized.

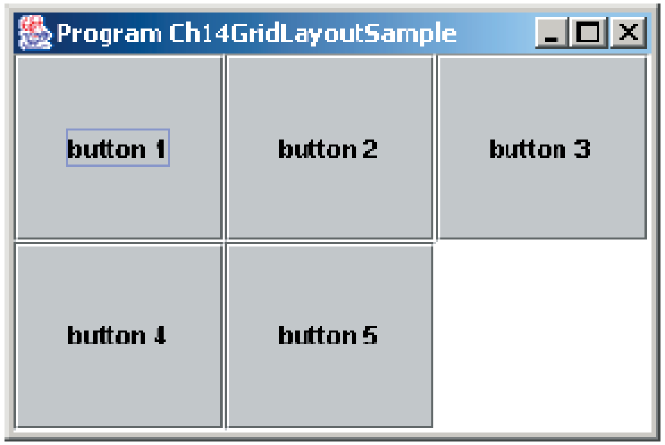


GridLayout

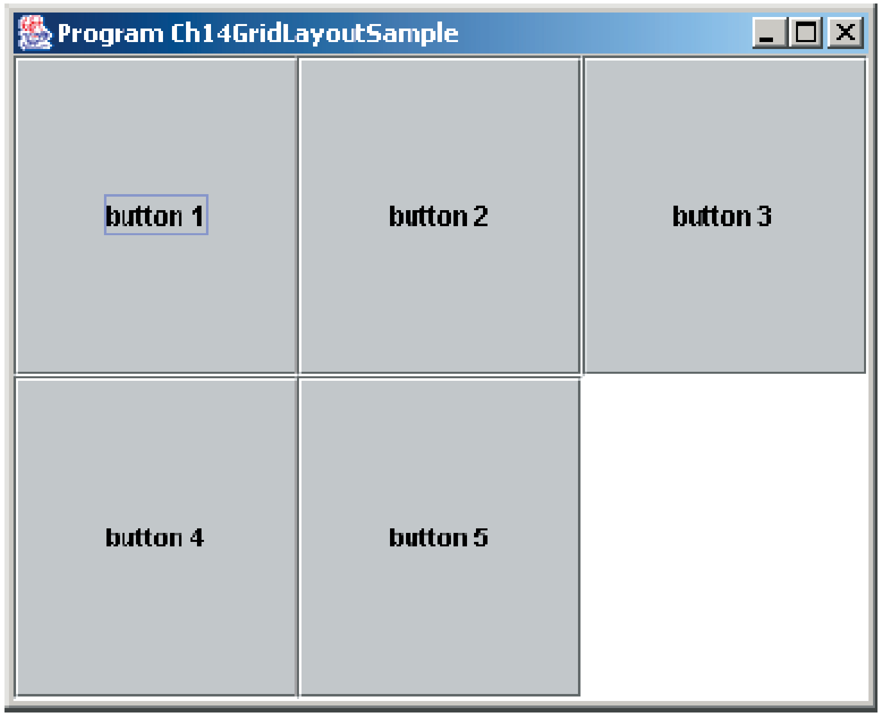
- This layout manager placesGUI components on equal-size N by M grids.
- Components are placed in top-to-bottom, left-to-right order.
- The number of rows and columns remains the same after the frame is resized, but the width and height of each region will change.

GridLayout Sample

When the frame first appears on the screen.



After the frame is resized.



Nesting Panels

- It is possible, but very difficult, to place all GUI components on a single JPanel or other types of containers.
- A better approach is to use multiple panels, placing panels inside other panels.
- To illustrate this technique, we will create two sample frames that contain nested panels.
- Ch14NestedPanels1.java provides the user interface for playing Tic Tac Toe.
- Ch14NestedPanels2.java provides the user interface for playing HiLo.

Other Common GUI Components

- JCheckBox
  - see Ch14JCheckBoxSample1.java and Ch14JCheckBoxSample2.java
- JRadioButton
  - see Ch14JRadioButtonSample.java
- JComboBox
  - see Ch14JComboBoxSample.java
- JList
  - see Ch14JListSample.java
- JSlider
  - see Ch14JSliderSample.java

Menus

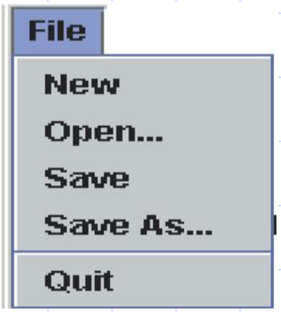
- The javax.swing package contains three menu-related classes: JMenuBar, JMenu, and JMenuItem.
- JMenuBar is a bar where the menus are placed. There is one menu bar per frame.
- JMenu (such as File or Edit) is a group of menu choices. JMenuBar may include many JMenu objects.
- JMenuItem (such as Copy, Cut, or Paste) is an individual menu choice in a JMenu object.
- Only the JMenuItem objects generate events.

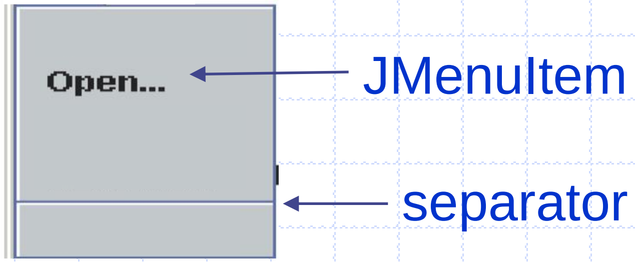
Menu Components

JMenuBar

File Edit View Help

JMenu





Sequence for Creating Menus

1. Create a JMenuBar object and attach it to a frame.
2. Create a JMenu object.
3. Create JMenuItem objects and add them to the JMenu object.
4. Attach the JMenu object to the JMenuBar object.

Handling Mouse Events

- Mouse events include such user interactions as
  - moving the mouse
  - dragging the mouse (moving the mouse while the mouse button is being pressed)
  - clicking the mouse buttons.
- The MouseListener interface handles mouse button
  - mouseClicked, mouseEntered, mouseExited, mousePressed, and mouseReleased
- The MouseMotionListener interface handles mouse movement
  - mouseDragged and mouseMoved.
- See Ch14TrackMouseFrame and Ch14SketchPad