CNT 4714: Enterprise Computing Spring 2010

Introduction To GUIs and Event-Driven Programming In Java – Part 4

Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
http://www.cs.ucf.edu/courses/cnt4714/spr2010

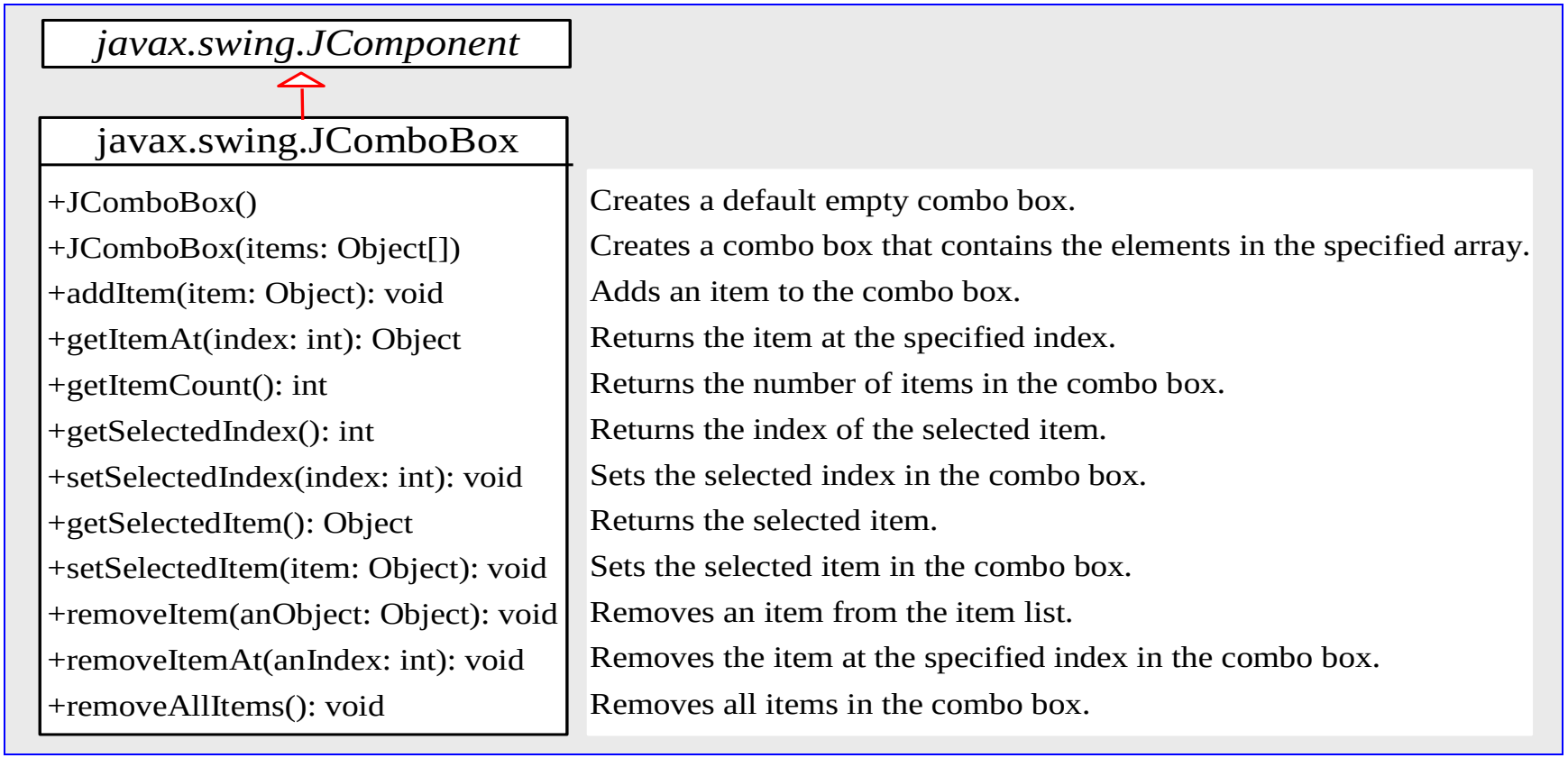
School of Electrical Engineering and Computer Science
University of Central Florida

Combo Boxes

- A combo box, also known as a choice list or drop-down list, contains a list of items from which the user can choose.
- It is useful in limiting a user's range of choices and avoids the cumbersome validation of data input.
- The UML diagram for the `JComboBox` class is shown on the next page.

JComboBox

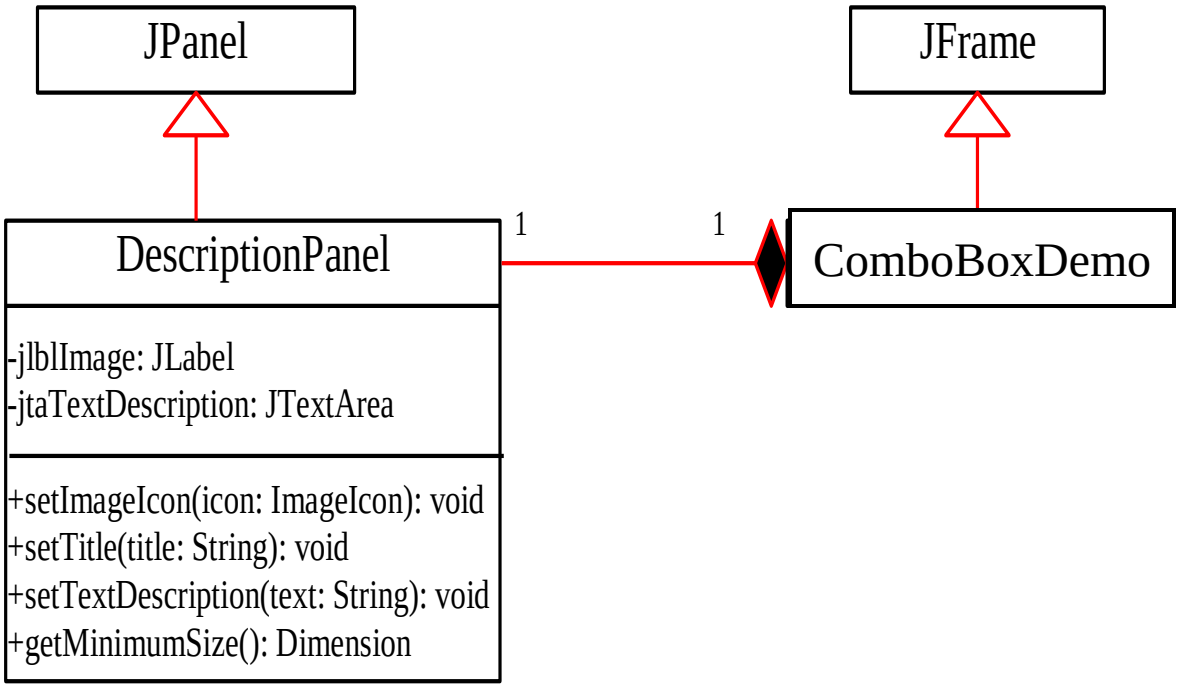
- JComboBox inherits all the properties from JComponent. A JComboBox can generate an ActionEvent and an ItemEvent, among many other events.



Comments on JComboBox

- JComboBox inherits all the properties from JComponent. A JComboBox can generate an ActionEvent and an ItemEvent, among many other events.
- Whenever a new item is selected, an ActionEvent is fired. Whenever a new item is selected, JComboBox generates an ItemEvent twice, once for deselecting the previously selected item, and the other for selecting the currently selected item.
- Note that no ItemEvent is fired if the current item is reselected.
- To respond to an ItemEvent, you need to implement the itemStateChanged(ItemEvent e) handler for processing a choice.
- To get data from a JComboBox menu, you can use getSelectedItem() to return the currently selected item, or e.getItem() method to get the item from the itemStateChanged(ItemEvent e) handler.
- The program on the following pages illustrates the JComboBox class.

Comments on JComboBox



Example – ComboBoxDemo

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

public class ComboBoxDemo extends JFrame {
  // Declare an array of Strings for flag titles
  private String[] flagTitles = {"Canada", "China", "Denmark",
    "France", "Germany", "India", "Norway", "United Kingdom",
    "United States of America"};

  // Declare an ImageIcon array for the national flags of 9 countries
  private ImageIcon[] flagImage = {
    new ImageIcon("E:/image/ca.gif"),
    new ImageIcon("E:/image/china.gif"),
    new ImageIcon("E:/image/denmark.gif"),
    new ImageIcon("E:/image/fr.gif"),
    new ImageIcon("E:/image/germany.gif"),
    new ImageIcon("E:/image/india.gif"),
    new ImageIcon("E:/image/norway.gif"),
    new ImageIcon("E:/image/uk.gif"),
    new ImageIcon("E:/image/us.gif")
  };

  // Declare an array of strings for flag descriptions
  private String[] flagDescription = new String[9];

  // Declare and create a description panel
  private DescriptionPanel descriptionPanel = new DescriptionPanel();

  // Create a combo box for selecting countries
  private JComboBox jcbo = new JComboBox(flagTitles);
```

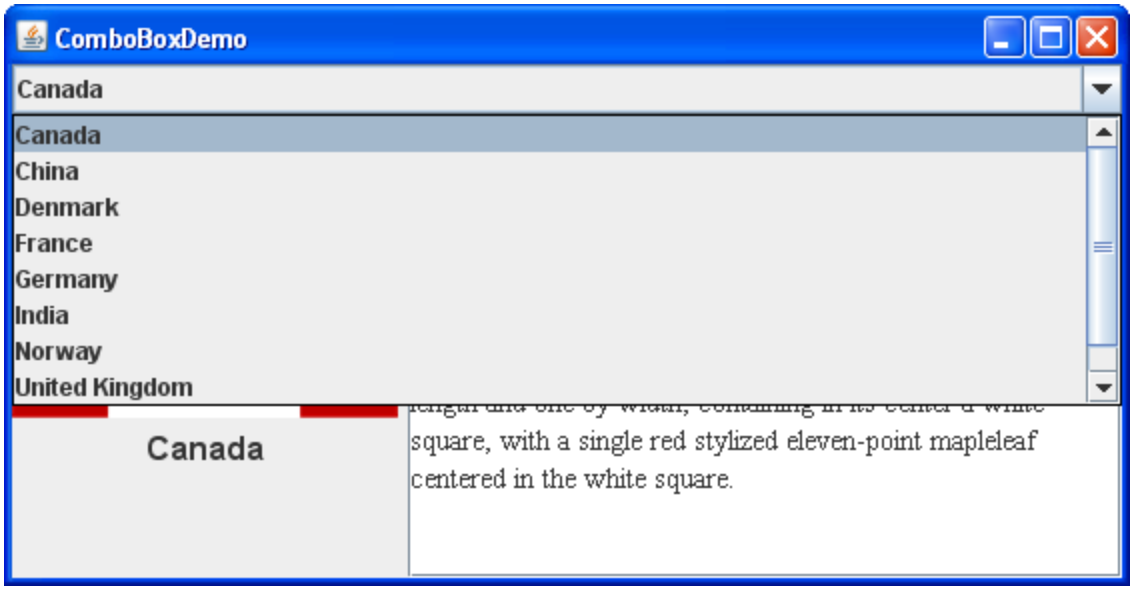

Example – ComboBoxDemo

```
public static void main(String[] args) {
    ComboBoxDemo frame = new ComboBoxDemo();
    frame.pack();
    frame.setTitle("ComboBoxDemo");
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }
  public ComboBoxDemo() {
    // Set text description
    flagDescription[0] = "The Maple Leaf flag \n\n" +
      "The Canadian National Flag was adopted by the Canadian " +
      "Parliament on October 22, 1964 and was proclaimed into law " +
      "by Her Majesty Queen Elizabeth II (the Queen of Canada) on " +
      "February 15, 1965. The Canadian Flag (colloquially known " +
      "as The Maple Leaf Flag) is a red flag of the proportions " +
      "two by length and one by width, containing in its center a " +
      "white square, with a single red stylized eleven-point " +
      "mapleleaf centered in the white square.";
    flagDescription[1] = "Description for China ... ";
    flagDescription[2] = "Description for Denmark ... ";
    flagDescription[3] = "Description for France ... ";
    flagDescription[4] = "Description for Germany ... ";
    flagDescription[5] = "Description for India ... ";
    flagDescription[6] = "Description for Norway ... ";
    flagDescription[7] = "Description for UK ... ";
    flagDescription[8] = "Description for US ... ";
    // Set the first country (Canada) for display
    setDisplay(0);
```

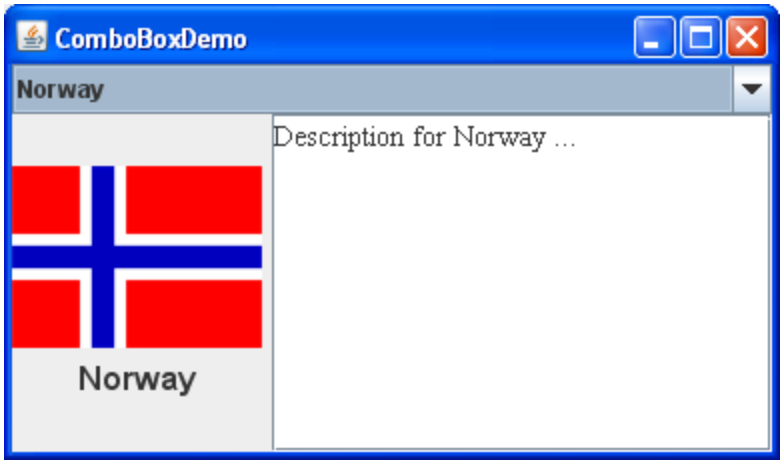

```
    // Add combo box and description panel to the lis
    add(jcbo, BorderLayout.NORTH);
    add(descriptionPanel, BorderLayout.CENTER);

    // Register listener
    jcbo.addItemListener(new ItemListener() {
      /** Handle item selection */
      public void itemStateChanged(ItemEvent e) {
        setDisplay(jcbo.getSelectedIndex());
      }
    });
  }

  /** Set display information on the description panel */
  public void setDisplay(int index) {
    descriptionPanel.setTitle(flagTitles[index]);
    descriptionPanel.setImageIcon(flagImage[index]);
    descriptionPanel.setDescription(flagDescription[index]);
  }
}
```

Example – ComboBoxDemo

The JComboBox



User clicks on the JComboBox slider

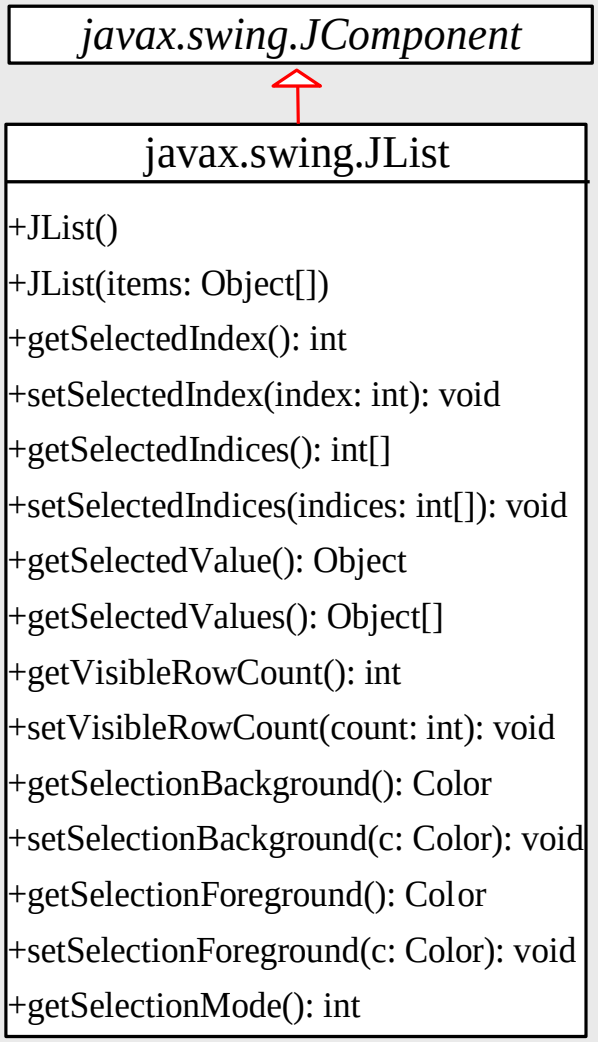


User selected Norway

Lists

- A list is a component that basically performs the same function as a combo box but enables the user to choose a single value or multiple values simultaneously.
- The Swing `JList` is a very versatile component.
- The UML (again a partial UML) for the `JList` class is shown on the next page.

JList



Creates a default empty list.

Creates a list that contains the elements in the specified array.

Returns the index of the first selected item.

Selects the cell at the specified index.

Returns an array of all of the selected indices in increasing order.

Selects the cells at the specified indices.

Returns the first selected item in the list.

Returns an array of the values for the selected cells in increasing index order.

Returns the number of visible rows displayed without a scrollbar. (default: 8)

Sets the preferred number of visible rows displayed without a scrollbar.

Returns the background color of the selected cells.

Sets the background color of the selected cells.

Returns the foreground color of the selected cells.

Sets the foreground color of the selected cells.

Returns the selection mode for the list.

Lists

- `selectionMode` is one of three values (`SINGLE_SELECTION`, `SINGLE_INTERVAL_SELECTION`, `MULTIPLE_INTERVAL_SELECTION`) as defined in `javax.swing.ListSelectionModel` that indicate whether a single item, single-interval item, or multiple-interval item can be selected.
  - Single allows only one item to be selected.
  - Single interval allows multiple selections, but the selected items must be contiguous.
  - Multiple interval allows selection of multiple contiguous items without restriction.

Example – ListDemo

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

public class ListDemo extends JFrame {
  final int NUMBER_OF_FLAGS = 9;

  // Declare an array of Strings for flag titles
  private String[] flagTitles = {"Canada", "China", "Denmark",
    "France", "Germany", "India", "Norway", "United Kingdom",
    "United States of America"};

  // The list for selecting countries
  private JList jlst = new JList(flagTitles);

  // Declare an ImageIcon array for the national flags of 9 countries
  private ImageIcon[] imageIcons = {
    new ImageIcon("E:/image/ca.gif"),
    new ImageIcon("E:/image/china.gif"),
    new ImageIcon("E:/image/denmark.gif"),
    new ImageIcon("E:/image/fr.gif"),
    new ImageIcon("E:/image/germany.gif"),
    new ImageIcon("E:/image/india.gif"),
    new ImageIcon("E:/image/norway.gif"),
    new ImageIcon("E:/image/uk.gif"),
    new ImageIcon("E:/image/us.gif")
  };

  // Arrays of labels for displaying images
  private JLabel[] jlblImageViewer = new JLabel[NUMBER_OF_FLAGS];
```


Example – ListDemo

```
public static void main(String[] args) {
    ListDemo frame = new ListDemo();
    frame.setSize(650, 500);
    frame.setTitle("ListDemo");
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }

  public ListDemo() {
    // Create a panel to hold nine labels
    JPanel p = new JPanel(new GridLayout(3, 3, 5, 5));

    for (int i = 0; i < NUMBER_OF_FLAGS; i++) {
      p.add(jlblImageViewer[i] = new JLabel());
      jlblImageViewer[i].setHorizontalAlignment
        (SwingConstants.CENTER);
    }

    // Add p and the list to the frame
    add(p, BorderLayout.CENTER);
    add(new JScrollPane(jlst), BorderLayout.WEST);
```

The list is added to a scroll pane so that it can be scrolled when the number of icons in the list extends beyond the viewing area.

```
// Register listeners
    jlst.addListSelectionListener(new ListSelectionListener() {
      /** Handle list selection */
      public void valueChanged(ListSelectionEvent e) {
        // Get selected indices
        int[] indices = jlst.getSelectedIndices();

        int i;
        // Set icons in the labels
        for (i = 0; i < indices.length; i++) {
          jlblImageViewer[i].setIcon(imageIcons[indices[i]]);
        }

        // Remove icons from the rest of the labels
        for (; i < NUMBER_OF_FLAGS; i++) {
          jlblImageViewer[i].setIcon(null);
        }
      }
    });
  }
}
```

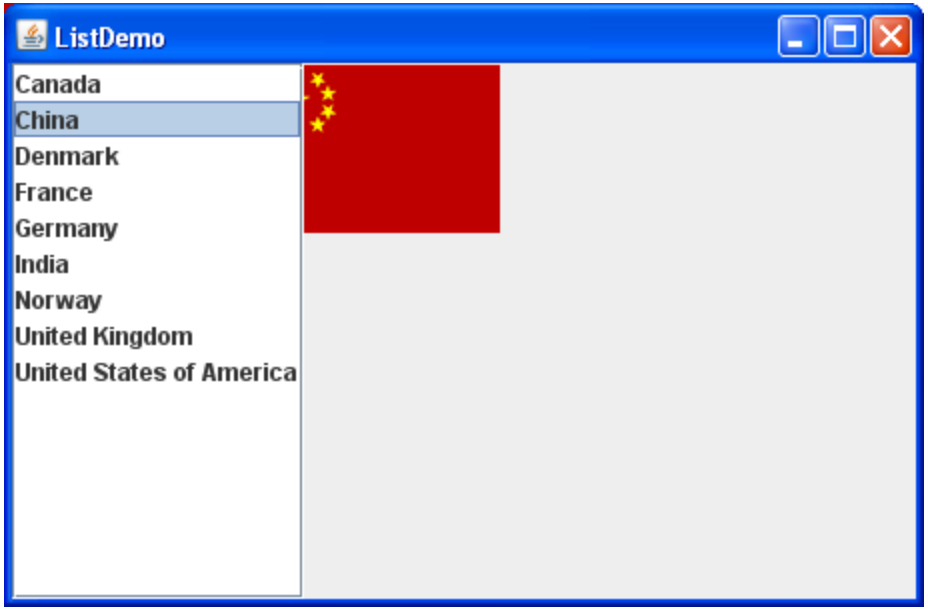
An anonymous inner class listener listens to `ListSelectionEvent` for handling the selection of the country names in the list.

`ListSelectionEvent` and `ListSelectionListener` are defined in the `javax.swing.event` package, so this package must be imported into the program.

By default, the selection mode of the list is multiple-interval, which lets the user select multiple non-contiguous items.

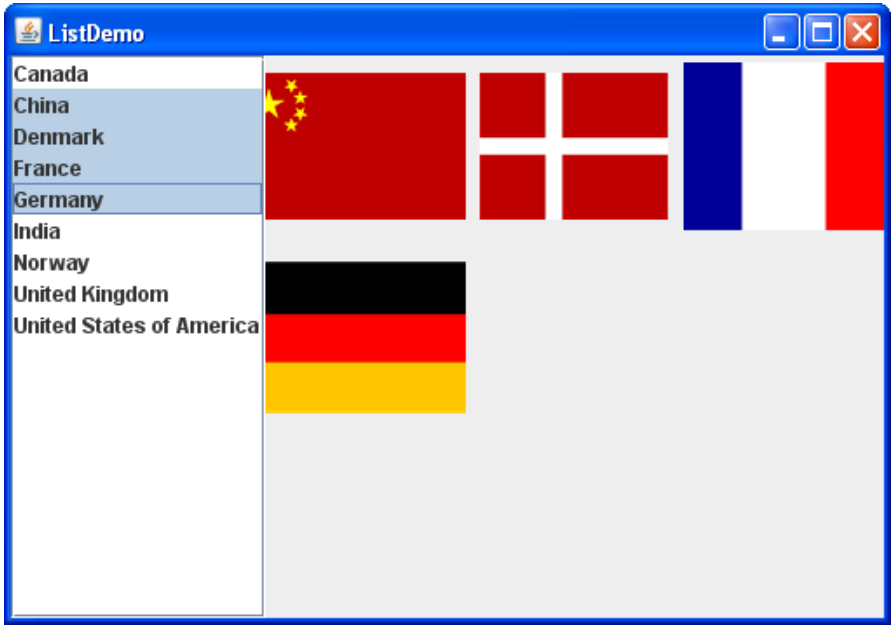
Example – `ListDemo`

Initial GUI

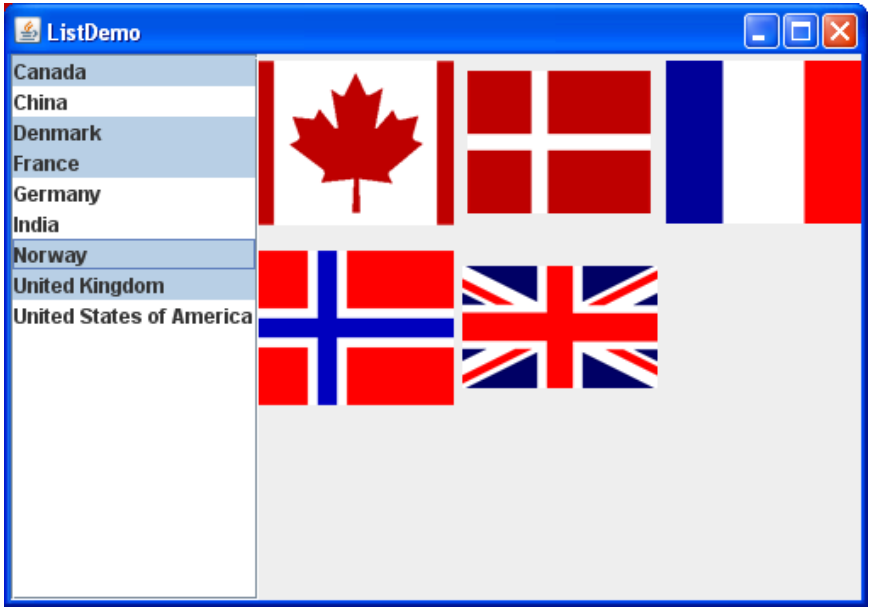


User selects a single item

Example – `ListDemo`



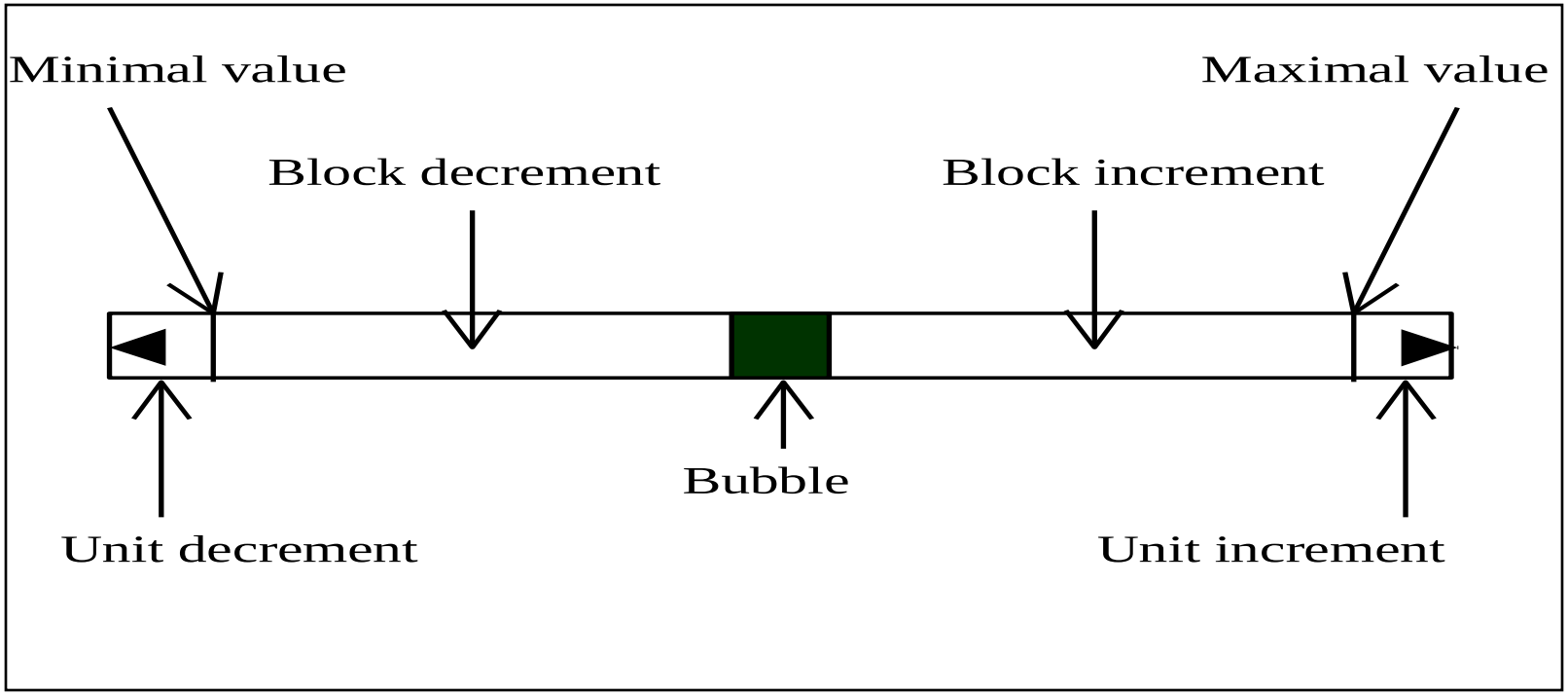
User selects multiple contiguous items



User selects multiple non-contiguous items

Scroll Bars

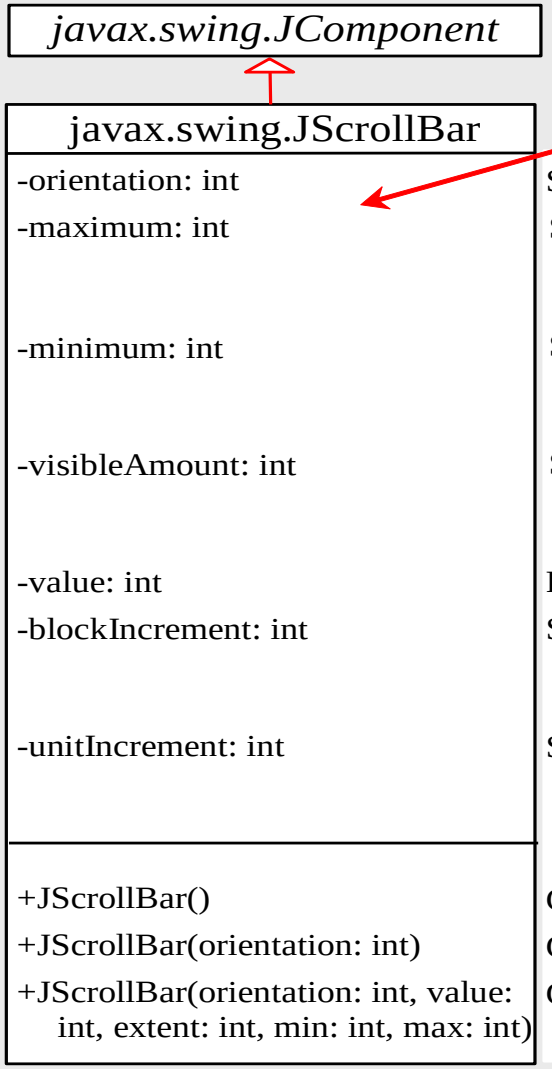
- `JScrollBar` is a component that enables the user to select from a range of values as shown below:



Scroll Bars

- Normally, the user changes the value of the scroll bar by making a gesture with the mouse. For example, the user can drag the scroll bar's bubble up and down, or click in the scroll bar's unit-increment or block-increment areas.
- Keyboard gestures can also be mapped to the scroll bar.
- By convention, the Page Up and Page Down keys are equivalent to clicking in the scroll bar's block increment and block decrement areas.
- The width of the scroll bar's track corresponds to the `maximum+visibleAmount`. When a scroll bar is set to its `maximum` value, the left side of the bubble is at `maximum`, and the right side is at `maximum + visibleAmount`.
- A partial UML for the `JScrollBar` class is given on the next page.

JScrollBar



The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

-orientation: int	Specifies horizontal or vertical style, default is horizontal.
-maximum: int	Specifies the maximum value the scroll bar represents when the bubble reaches the right end of the scroll bar for horizontal style or the bottom of the scroll bar for vertical style.
-minimum: int	Specifies the minimum value the scroll bar represents when the bubble reaches the left end of the scroll bar for horizontal style or the top of the scroll bar for vertical style.
-visibleAmount: int	Specifies the relative width of the scroll bar's bubble. The actual width appearing on the screen is determined by the maximum value and the value of visibleAmount.
-value: int	Represents the current value of the scroll bar.
-blockIncrement: int	Specifies value added (subtracted) when the user activates the block-increment (decrement) area of the scroll bar, as shown in Figure 13.30.
-unitIncrement: int	Specifies the value added (subtracted) when the user activates the unit-increment (decrement) area of the scroll bar, as shown in Figure 13.30.
+JScrollBar()	Creates a default vertical scroll bar.
+JScrollBar(orientation: int)	Creates a scroll bar with the specified orientation.
+JScrollBar(orientation: int, value: int, extent: int, min: int, max: int)	Creates a scrollbar with the specified orientation, value, extent, minimum, and maximum.

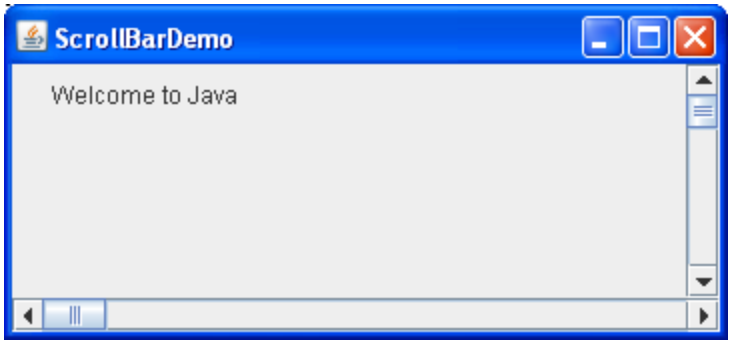
Comments on Scroll Bars

- When the user changes the value of the scroll bar, the scroll bar generates an instance of `AdjustmentEvent`, which is passed to every registered listener.
- An object that wishes to be notified of changes to the scroll bar's value must implement the `adjustmentValueChanged` method in the `AdjustmentListener` interface defined in the `java.awt.event` package.
- The following scroll bar demo program uses both horizontal and vertical scroll bars to control a message displayed in a panel (this is the same example that we previously controlled the message movement using buttons).

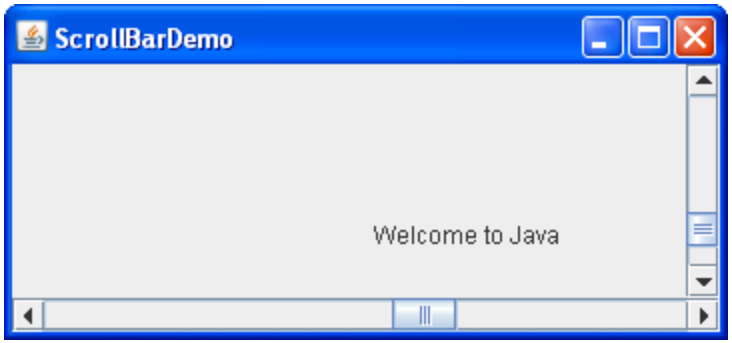
Example – ScrollBarDemo

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

public class ScrollBarDemo extends JFrame {
  // Create horizontal and vertical scroll bars
  private JScrollBar jscbHort =
    new JScrollBar(JScrollBar.HORIZONTAL);
  private JScrollBar jscbVert =
    new JScrollBar(JScrollBar.VERTICAL);

  // Create a MessagePanel
  private MessagePanel messagePanel =
    new MessagePanel("Welcome to Java");

  public static void main(String[] args) {
    ScrollBarDemo frame = new ScrollBarDemo();
    frame.setTitle("ScrollBarDemo");
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.pack();
    frame.setVisible(true);
  }

  public ScrollBarDemo() {
    // Add scroll bars and message panel to the frame
    setLayout(new BorderLayout());
    add(messagePanel, BorderLayout.CENTER);
    add(jscbVert, BorderLayout.EAST);
    add(jscbHort, BorderLayout.SOUTH);
```


```
// Register listener for the scroll bars
    jscbHort.addAdjustmentListener(new AdjustmentListener() {
      public void adjustmentValueChanged(AdjustmentEvent e) {
        // getValue() and getMaximumValue() return int, but for better
        // precision, use double
        double value = jscbHort.getValue();
        double maximumValue = jscbHort.getMaximum();
        double newX = (value * messagePanel.getWidth() /
          maximumValue);
        messagePanel.setXCoordinate((int)newX);
      }
    });
    jscbVert.addAdjustmentListener(new AdjustmentListener() {
      public void adjustmentValueChanged(AdjustmentEvent e) {
        // getValue() and getMaximumValue() return int, but for better
        // precision, use double
        double value = jscbVert.getValue();
        double maximumValue = jscbVert.getMaximum();
        double newY = (value * messagePanel.getHeight() /
          maximumValue);
        messagePanel.setYCoordinate((int)newY);
      }
    });
  }
}
```

Initial GUI

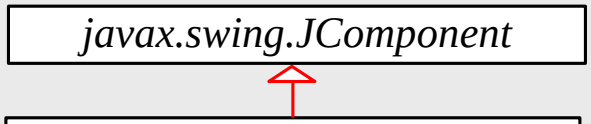


GUI after user changes position
of both scroll bars

Sliders

- `JSlider` is similar to `JScrollBar`, but `JSlider` has more properties and can appear in many forms.
- `JSlider` lets the user graphically select a value by sliding a knob within a bounded interval. The slider can show both major tick marks and minor tick marks between them.
- The number of pixels between tick marks is controlled by the `majorTickSpacing` and `minorTickSpacing` properties.
- Sliders can be displayed horizontally or vertically, with or without tick marks, and with or without labels.
- The values of a vertical scroll bar increase from top to bottom, but the value so a vertical slider decrease from top to bottom.
- The more commonly used constructors and properties are shown in the UML diagram on the next page.

JSlider



The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

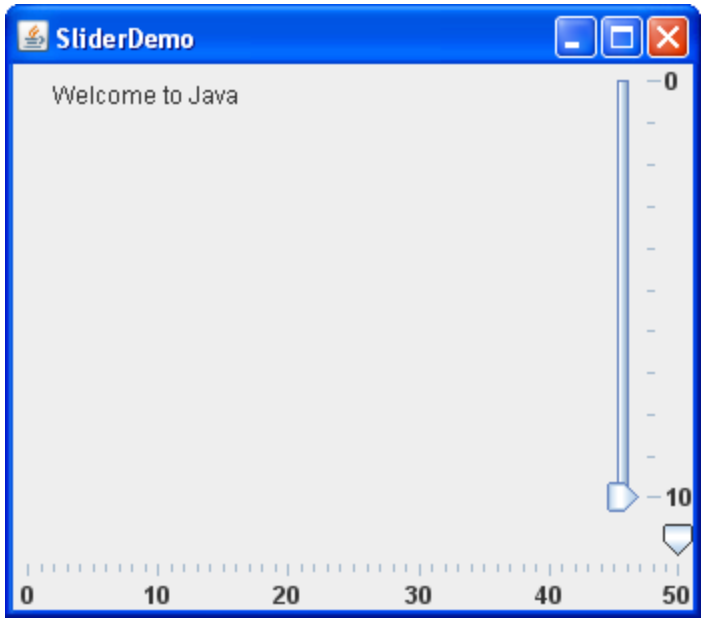
javax.swing.JSlider	
-maximum: int	The maximum value represented by the slider (default: 100).
-minimum: int	The minimum value represented by the slider (default: 0).
-value: int	The current value represented by the slider.
-orientation: int	The orientation of the slider (default: JSlider.HORIZONTAL).
-paintLabels: boolean	True if the labels are painted at tick marks (default: false).
-paintTicks: boolean	True if the ticks are painted on the slider (default: false).
-paintTrack: boolean	True if the track is painted on the slider (default: true).
-majorTickSpacing: int	The number of units between major ticks (default: 0).
-minorTickSpacing: int	The number of units between minor ticks (default: 0).
-inverted: boolean	True to reverse the value-range, and false to put the value range in the normal order (default: false).
+JSlider()	Creates a default horizontal slider.
+JSlider(min: int, max: int)	Creates a horizontal slider using the specified min and max.
+JSlider(min: int, max: int, value: int)	Creates a horizontal slider using the specified min, max, and value.
+JSlider(orientation: int)	Creates a slider with the specified orientation.
+JSlider(orientation: int, min: int, max: int, value: int)	Creates a slider with the specified orientation, min, max, and value.

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

public class SliderDemo extends JFrame {
  // Create horizontal and vertical sliders
  private JSlider jsldHort = new JSlider(JSlider.HORIZONTAL);
  private JSlider jsldVert = new JSlider(JSlider.VERTICAL);

  // Create a MessagePanel
  private MessagePanel messagePanel =
    new MessagePanel("Welcome to Java");

  public static void main(String[] args) {
    SliderDemo frame = new SliderDemo();
    frame.setTitle("SliderDemo");
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.pack();
    frame.setVisible(true);
  }

  public SliderDemo() {
    // Add sliders and message panel to the frame
    setLayout(new BorderLayout(5, 5));
    add(messagePanel, BorderLayout.CENTER);
    add(jsldVert, BorderLayout.EAST);
    add(jsldHort, BorderLayout.SOUTH);
```


Example – SliderDemo

```
// Set properties for sliders
    jsldHort.setMaximum(50);
    jsldHort.setPaintLabels(true);
    jsldHort.setPaintTicks(true);
    jsldHort.setMajorTickSpacing(10);
    jsldHort.setMinorTickSpacing(1);
    jsldHort.setPaintTrack(false);
    jsldVert.setInverted(true);
    jsldVert.setMaximum(10);
    jsldVert.setPaintLabels(true);
    jsldVert.setPaintTicks(true);
    jsldVert.setMajorTickSpacing(10);
    jsldVert.setMinorTickSpacing(1);
    // Register listener for the sliders
    jsldHort.addChangeListener(new ChangeListener() {
      /** Handle scroll bar adjustment actions */
      public void stateChanged(ChangeEvent e) {
        // getValue() and getMaximumValue() return int, but for better
        // precision, use double
        double value = jsldHort.getValue();
        double maximumValue = jsldHort.getMaximum();
        double newX = (value * messagePanel.getWidth() /
          maximumValue);
        messagePanel.setXCoordinate((int)newX);
      }
    });
    jsldVert.addChangeListener(new ChangeListener() {
      /** Handle scroll bar adjustment actions */
      public void stateChanged(ChangeEvent e) {
        // getValue() and getMaximumValue() return int, but for better
        // precision, use double
        double value = jsldVert.getValue();
        double maximumValue = jsldVert.getMaximum();
        double newY = (value * messagePanel.getHeight() /
          maximumValue);
        messagePanel.setYCoordinate((int) newY);
      }
    });
  }
}
```


Example – `SliderDemo`



Initial GUI

After user moves both sliders

Creating Multiple Windows

- Occasionally, you may want to create multiple windows in an application. The application opens a new window to perform a specified task.
- The new windows are called subwindows, and the main frame is called the main window.
- To create a subwindow from an application, you need to create a subclass of `JFrame` that defines the task of the new window and tells the new window what to do. You can then create an instance of this subclass in the application and launch the new window by setting the frame instance to be visible.
- The following example program creates a main window with a text area in the scroll pane and a button named Show Histogram. When the user clicks the button, a new window appears that displays a histogram to show occurrences of the letters in the text area.

Creating Multiple Windows

Step 1: Create a subclass of `JFrame` (called a `SubFrame`) that tells the new window what to do. For example, all the GUI application programs extend `JFrame` and are subclasses of `JFrame`.

Step 2: Create an instance of `SubFrame` in the application. Example:

```
SubFrame subFrame = new  SubFrame("SubFrame Title");
```

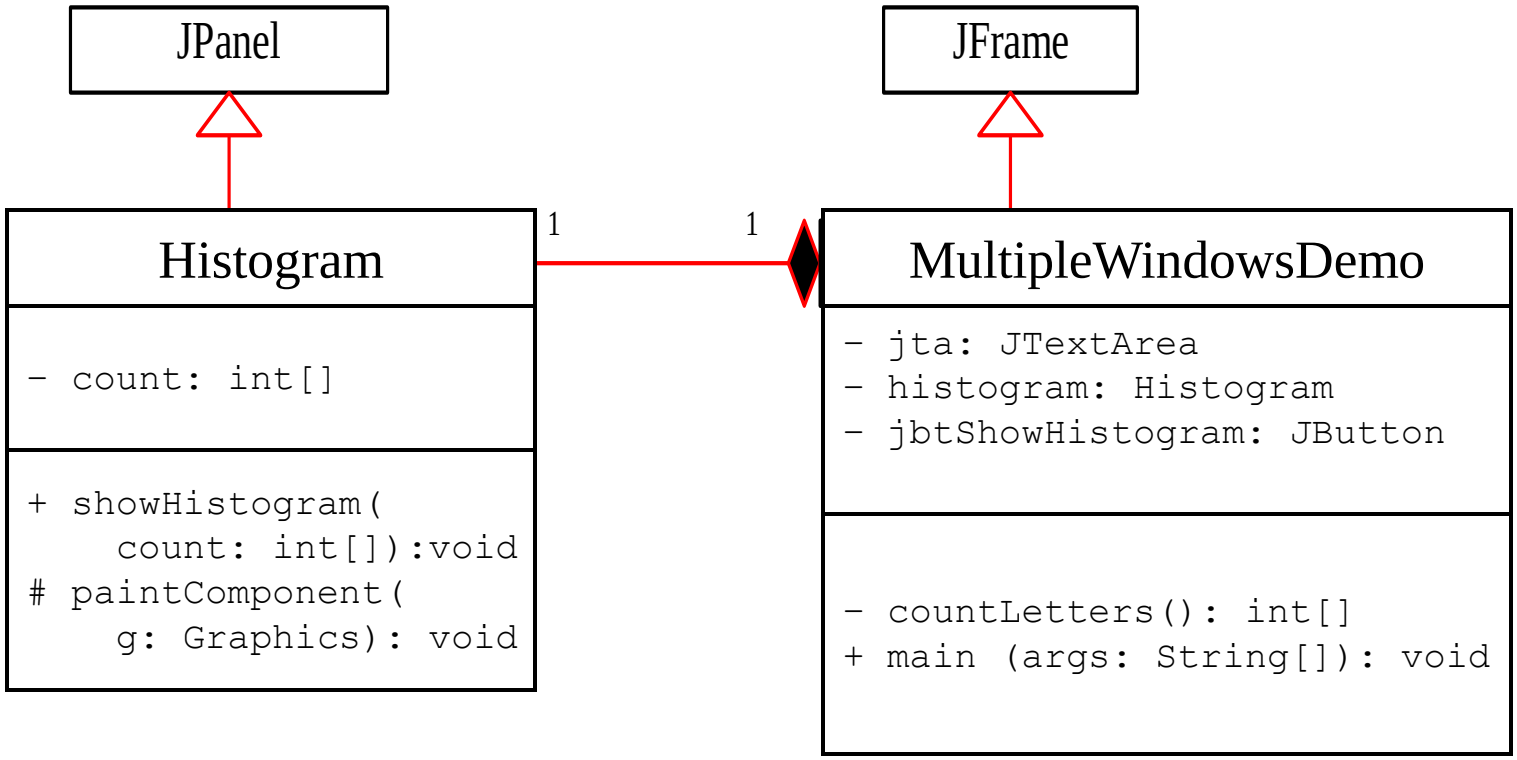
Step 3: Create a `JButton` for activating the `subFrame`.

```
add(new JButton("Activate SubFrame"));
```

Step 4: Override the `actionPerformed()` method as follows:

```
public actionPerformed(ActionEvent e) {
  String actionCommand = e.getActionCommand();
  if (e.target instanceof Button) {
    if ("Activate SubFrame".equals(actionCommand)) {
      subFrame.setVisible(true);
    }
  }
}
```


UML for Multiple Windows Example



Example – Histogram

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

public class Histogram extends JPanel {
  // Count the occurrence of 26 letters
  private int[] count;

  /** Set the count and display histogram */
  public void showHistogram(int[] count) {
    this.count = count;
    repaint();
  }

  /** Paint the histogram */
  protected void paintComponent(Graphics g) {
    if (count == null) return; // No display if count is null

    super.paintComponent(g);

    // Find the panel size and bar width and interval dynamically
    int width = getWidth();
    int height = getHeight();
    int interval = (width - 40) / count.length;
    int individualWidth = (int)(((width - 40) / 24) * 0.60);

    // Find the maximum count. The maximum count has the highest bar
    int maxCount = 0;
    for (int i = 0; i < count.length; i++) {
      if (maxCount < count[i])
        maxCount = count[i];
    }
```

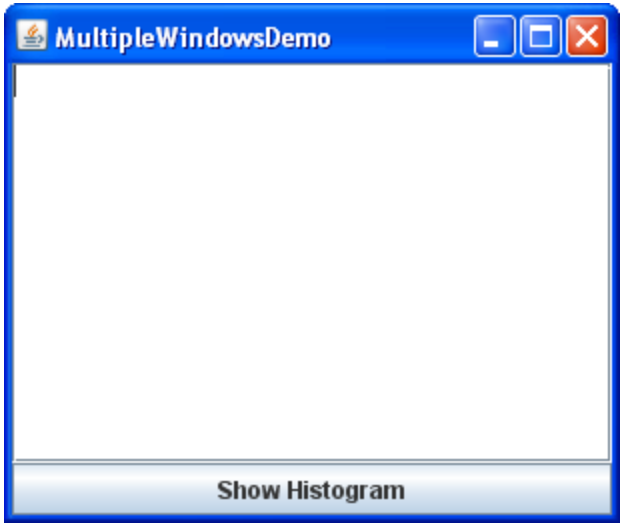

```
// x is the start position for the first bar in the histogram
    int x = 30;

    // Draw a horizontal base line
    g.drawLine(10, height - 45, width - 10, height - 45);
    for (int i = 0; i < count.length; i++) {
      // Find the bar height
      int barHeight =
        (int)(((double)count[i] / (double)maxCount) * (height - 55));

      // Display a bar (i.e. rectangle)
      g.drawRect(x, height - 45 - barHeight, individualWidth,
        barHeight);

      // Display a letter under the base line
      g.drawString((char)(65 + i) + "", x, height - 30);

      // Move x for displaying the next character
      x += interval;
    }
  }

  /** Override getPreferredSize */
  public Dimension getPreferredSize() {
    return new Dimension(300, 300);
  }
}
```


Example – MulitpleWindowsDemo

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

public class MultipleWindowsDemo extends JFrame {
  private JTextArea jta;
  private JButton jbtShowHistogram = new JButton("Show Histogram");
  private Histogram histogram = new Histogram();

  // Create a new frame to hold the histogram panel
  private JFrame histogramFrame = new JFrame();

  public MultipleWindowsDemo() {
    // Store text area in a scroll pane
    JScrollPane scrollPane = new JScrollPane(jta = new JTextArea());
    scrollPane.setPreferredSize(new Dimension(300, 200));
    jta.setWrapStyleWord(true);
    jta.setLineWrap(true);

    // Place scroll pane and button in the frame
    add(scrollPane, BorderLayout.CENTER);
    add(jbtShowHistogram, BorderLayout.SOUTH);

    // Register listener
    jbtShowHistogram.addActionListener(new ActionListener() {
      /** Handle the button action */
      public void actionPerformed(ActionEvent e) {
        // Count the letters in the text area
        int[] count = countLetters();

        // Set the letter count to histogram for display
        histogram.showHistogram(count);
```


Example – MulitpleWindowsDemo

```
// Show the frame
        histogramFrame.setVisible(true);
      }
    });

    // Create a new frame to hold the histogram panel
    histogramFrame.add(histogram);
    histogramFrame.pack();
    histogramFrame.setTitle("Histogram");
  }
  /** Count the letters in the text area */
  private int[] countLetters() {
    // Count for 26 letters
    int[] count = new int[26];
    // Get contents from the text area
    String text = jta.getText();
    // Count occurrence of each letter (case insensitive)
    for (int i = 0; i < text.length(); i++) {
      char character = text.charAt(i);
      if ((character >= 'A') && (character <= 'Z')) {
        count[(int)character - 65]++; // The ASCII for 'A' is 65
      }
      else if ((character >= 'a') && (character <= 'z')) {
        count[(int)character - 97]++; // The ASCII for 'a' is 97
      }
    }
    return count; // Return the count array
  }
  public static void main(String[] args) {
    MultipleWindowsDemo frame = new MultipleWindowsDemo();
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setTitle("MultipleWindowsDemo");
    frame.pack();
    frame.setVisible(true);
  }
}
```

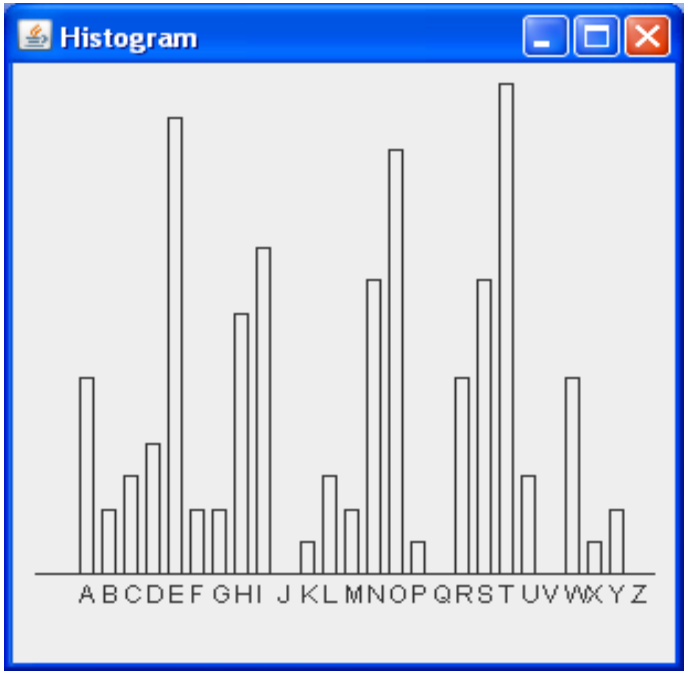
The initial first window

There isn't one until the user clicks the Show Histogram button!

The initial second window

Example – `MultipleWindowsDemo`

The first window after the user has entered the text



The second window after user has entered text in the first window