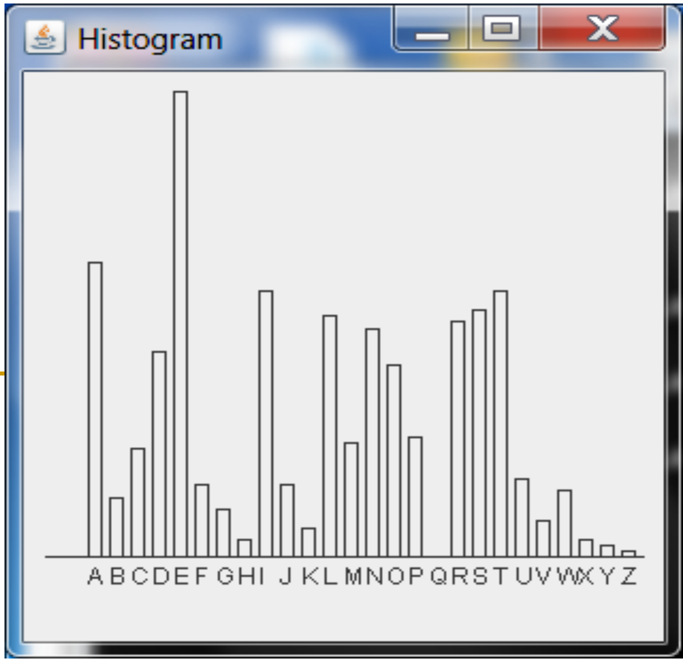
Events, Event Source, Event Listeners



MOTIVATION

- A graphical user interface (GUI) makes a system user-friendly and easy to use. Creating a GUI requires creativity and knowledge of how GUI components work. Since the GUI components in Java are very flexible and versatile, you can create a wide assortment of useful user interfaces.
- Previous chapters briefly introduced several GUI components. This chapter introduces the frequently used GUI components in detail.

OBJECTIVES

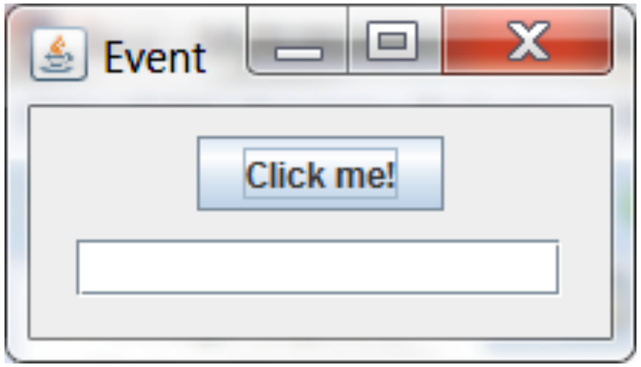
- To create listeners for JButton using inner class
- To create listeners for **JCheckBox**, **JRadioButton**, and **JTextField**
- To enter multiple-line texts using **JTextArea and JScrollPane**
- To select a single item using **JComboBox**
- To select a single or multiple items using **JList**
- To select a range of values using **JScrollBar**
- To select a range of values using **JSlider** and explore differences between **JScrollBar** and **JSlider**
- To display multiple windows in an application

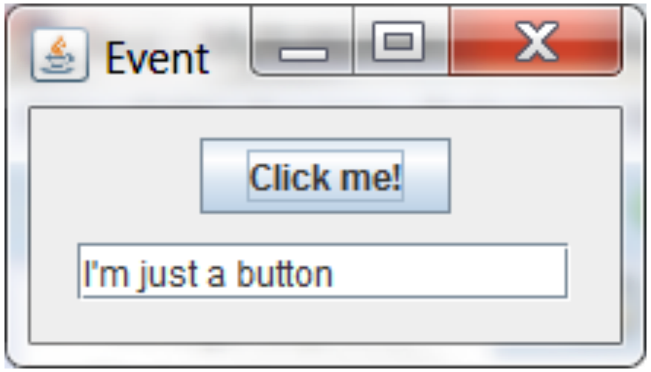
EVENT HANDLING

- What are user interface events? Some examples:
  - button clicks
  - mouse drags
  - key presses
  - Menu selections
- An event-driven program needs to inform the system about the events it is interested in.
- When one of those events occurs, the program will be notified.
- The program will then be able to respond to that event by executing the corresponding event-handling code.

HANDLING BUTTON-CLICK EVENTS

- Suppose we would like the JFrame to display a message when the user clicks on the button.





I'm just a button

EVENTS, EVENT SOURCES AND LISTENERS

- Important components involved in event-handling:
  - Events
    - Events are represented as `Event` objects.
    - For example, a button-click event is represented as an `ActionEvent` object.
  - Event sources
    - An **event source** is a component (e.g. a `JButton` object) closely associated with the event which sends notifications to **event listeners**.
    - For example, the `JButton` object associated with a button-click event.

EVENTS, EVENT SOURCES AND LISTENERS

- Event listeners
  - Every program must indicate which events it needs to receive. It does that by installing **event listener** objects.
  - An event listener belongs to a class that is provided by the application programmer. Its methods describe the actions to be taken when an event occur.

HANDLING BUTTON-CLICK EVENTS

- The following steps can be used for handling events:
  1. Declare class as an event listener
  2. Register event listener with event source
  3. Write event-handling code

HANDLING BUTTON-CLICK EVENTS

- Declare class as an event listener :

```
public class BtnHandler extends JFrame implements
    ActionListener {
    public JTextField text;
    public JButton btnOK;

    public BtnHandler() {   } //the constructor
    public actionPerformed(..) {   } //event handling
    public static void main() {   } // main method
}
```

HANDLING BUTTON-CLICK EVENTS

- Register event listener with event source :

```
public BtnHandler()
   {
         text = new JTextField(15);
         btnOK = new JButton("Click me please....");
         add(btnOK);
         add(text);
         // register event listener with event source here
         btnOK.addActionListener(this);


   }
```

HANDLING BUTTON-CLICK EVENTS

- Write event-handling code :

```
public void actionPerformed(ActionEvent e)
{
      text.setText("I'm just a button");
}
}
```

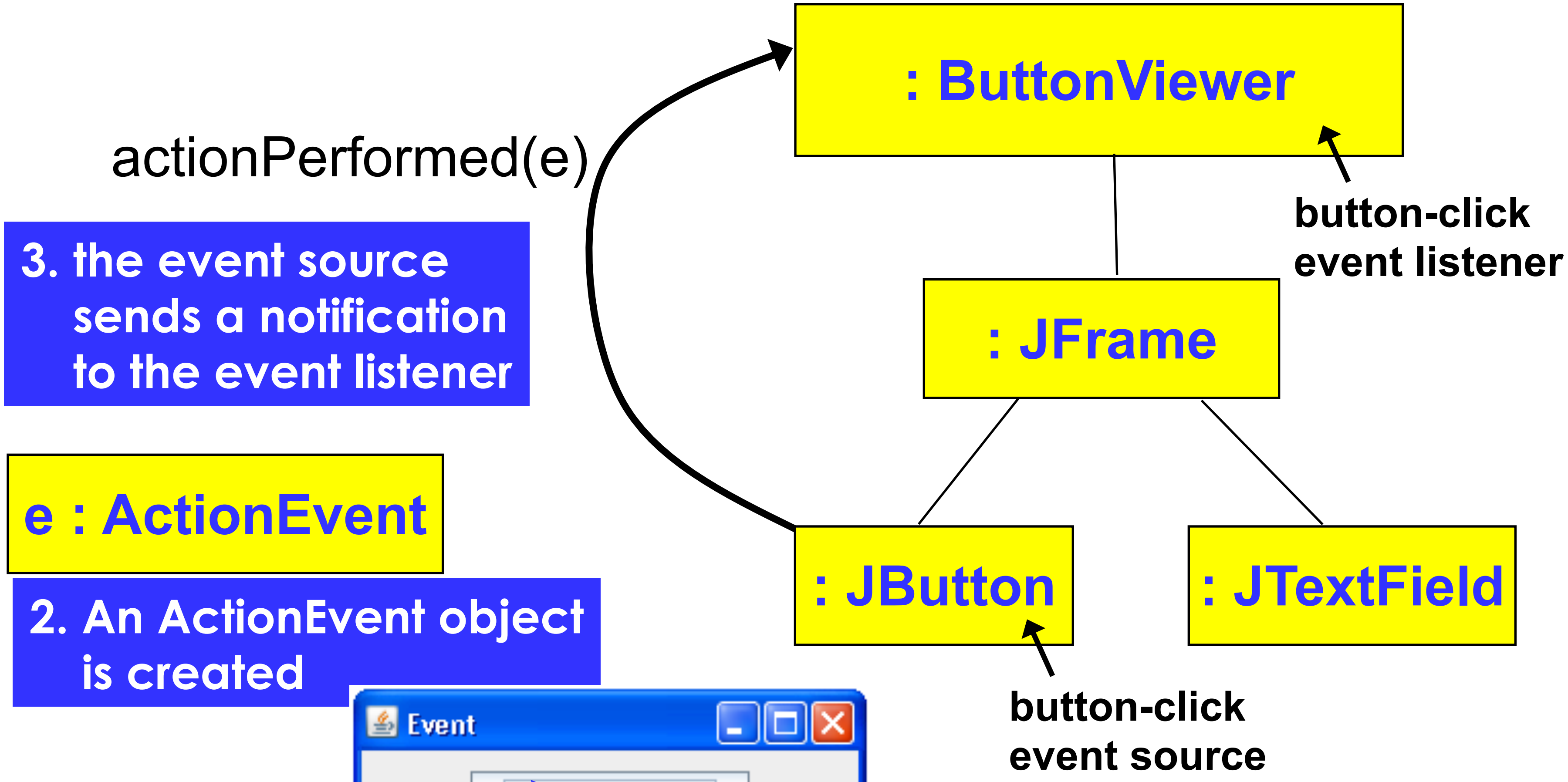
Execution starts here

```
....
....

public static void main (String[] args) {
        BtnHandler frame = new BtnHandler();

        frame.setTitle("Event");
        frame.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 10));
        frame.setSize(250,130);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
}
```

HANDLING BUTTON-CLICK EVENTS

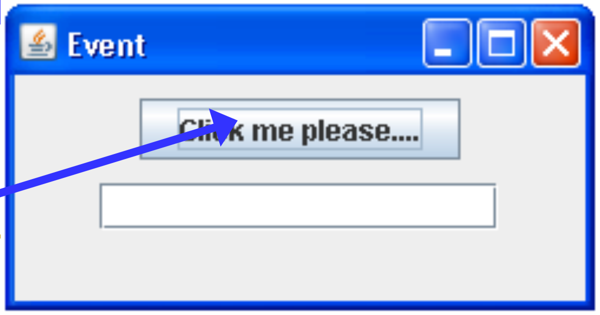
- The components involved:
  - Event
    - The `ActionEvent` object
  - Event source
    - The “Click me please …” `JButton` object
      - When the user clicks on the button, the `JButton` object sends an `ActionEvent` object to all event listeners.
  - Event listener
    - The `BtnHandler` object
    - This object will receive an `ActionEvent` object from the `JButton` object when a button-click event occurs.

4. the event listener executes actionPerformed() method



3. the event source sends a notification to the event listener

2. An ActionEvent object is created



2. a button-click event has occurred!

1. User clicks button

Using inner class for implementing EventListener

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class ButtonViewer {
  private static final int FRAME_WIDTH = 200;
  private static final int FRAME_HEIGHT = 120;
  private static JButton button;
  private static JTextField text;
  public static void main(String[] args) {
  JFrame frame = new JFrame();
// The button to trigger an event
    button = new JButton("Click me!");
    // The textfield for displaying the message
    text = new JTextField(15);
    frame.add(button);
    frame.add(text);
```

```
/** An action listener that prints a message :
ClickListerner is an inner class
    */
    class ClickListener implements ActionListener {
       public void actionPerformed(ActionEvent event) {
       Object obj = event.getSource();
       if (obj == button) {
          text.setText("I'm just a button");
       }
       }
    }
    //create an event listener object(specifically
ClickListener) named listener
    ActionListener listener = new ClickListener();
    //Attach an ActionListener to each button.
    button.addActionListener(listener);
    frame.setTitle("Event");
    frame.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 10));
    frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);

  }

}
```

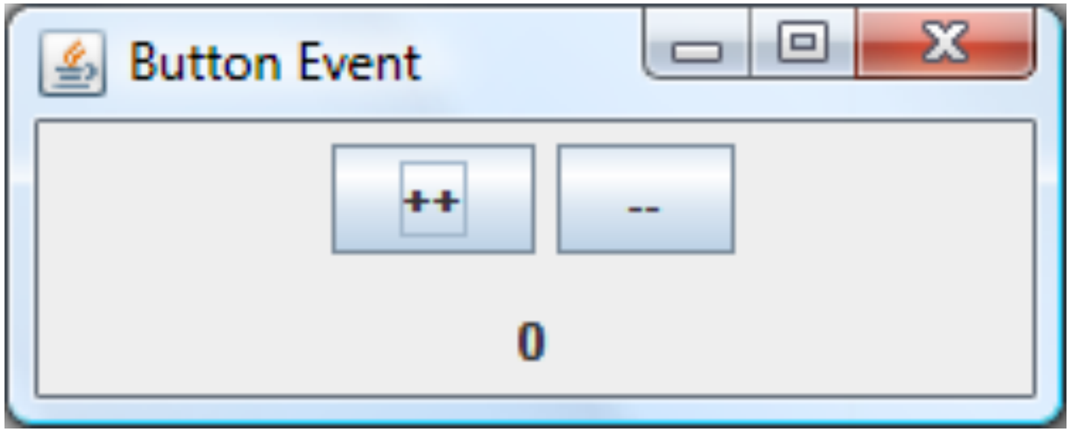
User Action	Event Source	Event Object Created	Type of Listener
Click on a button	JButton	ActionEvent	ActionListener
Select a new item in a combo box	JComboBox	ItemEvent, ActionEvent	ItemListener, ActionListener
Select an item from a list	JList	ListSelectionEvent	ListSelectionListener
Click on a checkbox	JCheckBox	ItemEvent, ActionEvent	ItemListener, ActionListener
Click on a radio button	JRadioButton	ItemEvent, ActionEvent	ItemListener, ActionListener
Drag slider knob	JSlider	ChangeEvent	ChangeListener
Press enter in a textfield	JTextField	ActionEvent	ActionListener

ActionEvent

- An Action Event occurs, whenever an action is performed by the user
- Examples:
  - User clicks a **button**
  - User chooses a **menu item**
  - User presses <Enter> in a **text field**
- Action Listener defines what should be done when an action event occurs, through actionPerformed message

Identifying Event Source

- To identify which object triggers the event, use **getSource()** method
- Example:

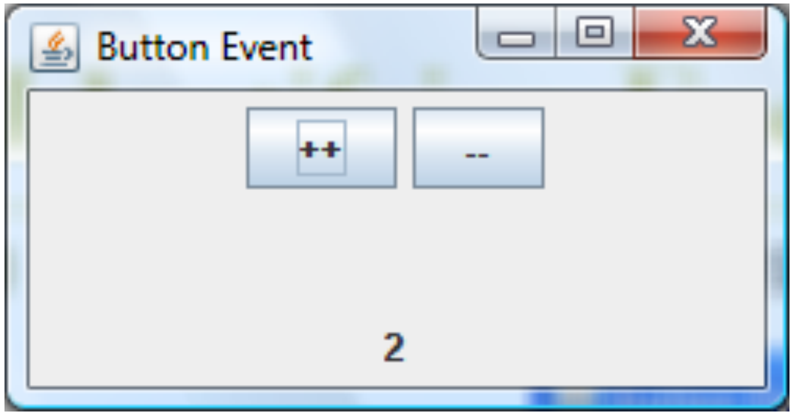


IdentifyingSource

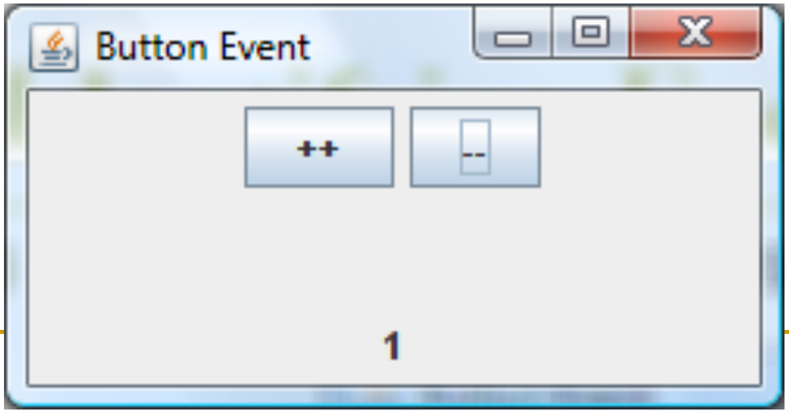
Run

Identifying Event Source

- After button "++" is clicked twice:



- After button "--" is clicked once:



Identifying Event Source

```
public class EvtPlusMinus extends JFrame implements ActionListener
{
        private JButton btnPlus, btnMinus;
        private JLabel lblValue;
        private int value = 0;
        private static JPanel p1, p2;

        public EvtPlusMinus() {
                btnPlus = new JButton("++");
                btnMinus = new JButton("--");
                lblValue = new JLabel("0");
                p1 = new JPanel();
                p2 = new JPanel();
                p1.add(btnPlus);
                p1.add(btnMinus);
                p2.add(lblValue);
                btnPlus.addActionListener(this);
                btnMinus.addActionListener(this);
        }
```

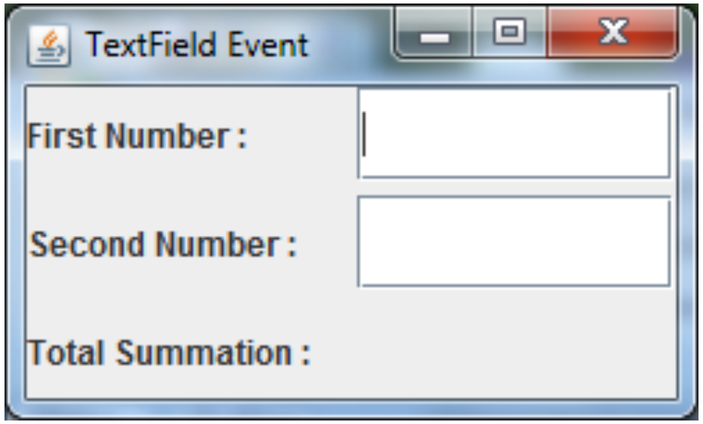
Identifying Event Source

```
public void actionPerformed(ActionEvent e) {
        Object obj = e.getSource();

        if (obj == btnPlus) {
                value++;
                lblValue.setText("" + value);
        }
        else if (obj == btnMinus) {
                value--;
                lblValue.setText("" + value);
        }
}
```

```
    public static void main (String[] args) {
            JFrame frame1 = new EvtPlusMinus();
            frame1.setLayout(new BorderLayout());
            frame1.add(p1,BorderLayout.NORTH);
            frame1.add(p2,BorderLayout.SOUTH);

            frame1.setTitle("Button Event");
            frame1.setSize(250,130);
            frame1.setLocation(300, 200);
            frame1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame1.setVisible(true);
    }

}
```

ActionEvent with JTextField

- Sometimes it is not necessary to identify the event source
- Example:


TextField Event

First Number :

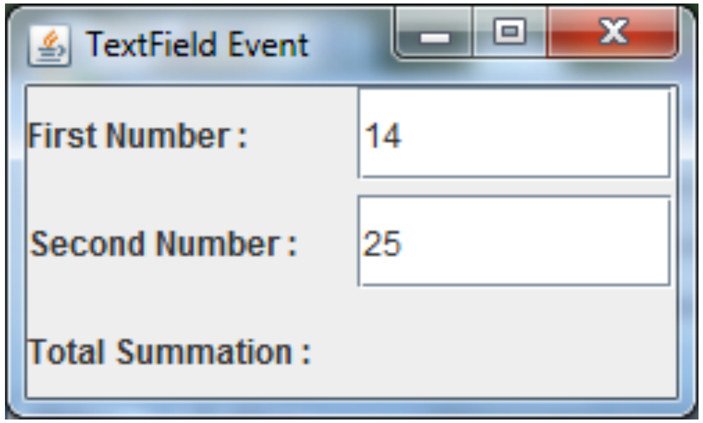
Second Number :

Total Summation :

Source Run

ActionEvent with JTextField

- After user input, say 14 and 25 :

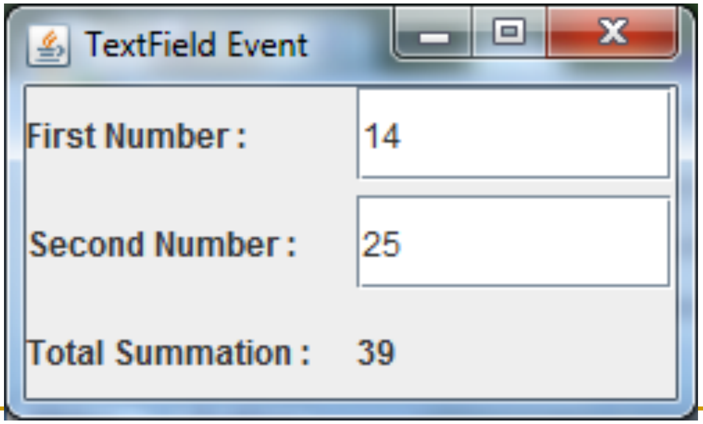


First Number : 14

Second Number : 25

Total Summation :

- After “Enter” key is pressed :



First Number : 14

Second Number : 25

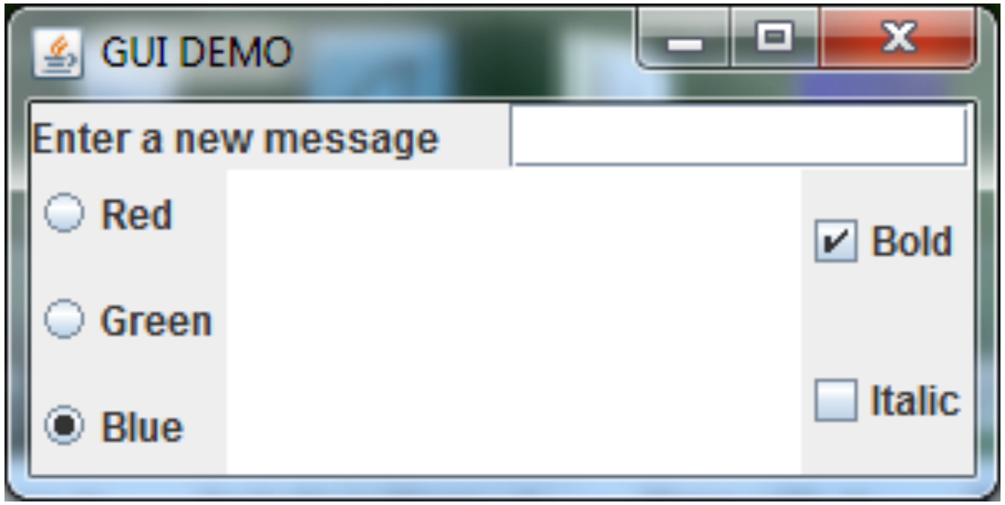
Total Summation : 39

ActionEvent with JTextField

```
public class EvtTextCalc extends JFrame implements
ActionListener  {
         JTextField txtNum1, txtNum2;
        JLabel lblAnswer;
        public EvtTextCalc()
        {
                txtNum1 = new JTextField(10);
                txtNum2 = new JTextField(10);
                add(new JLabel("First Number : "));
                add(txtNum1);
                add(new JLabel("Second Number : "));
                add(txtNum2);
                add(new JLabel("Total Summation : "));
                lblAnswer = new JLabel("");
                add(lblAnswer);
                txtNum1.addActionListener(this);
                txtNum2.addActionListener(this);
        }
```

```
:
public void actionPerformed(ActionEvent e)
{
        String str;
        int val1, val2, total;

        str = txtNum1.getText();
        if (str.equals(""))
                val1 = 0;
        else
                val1 = Integer.parseInt(str);
        str = txtNum2.getText();
        if (str.equals(""))
                val2 = 0;
        else
                val2 = Integer.parseInt(str);
        total = val1 + val2;
        lblAnswer.setText("" + total);
}
:
```

Events for JCheckBox, JRadioButton, and JTextField



GuiDemo

Run

Multiple Events: ActionEvent and ItemEvent

```
public class GuiDemo extends JFrame implements ActionListener,
ItemListener  {
            private JTextField text;
            private JRadioButton red, green, blue;
            private JLabel msg;
            private JCheckBox bold, italic;

            public GuiDemo() { …}  // the constructor
            public void actionPerformed(ActionEvent evt) { …} //Event handling
            public void itemStateChanged(ItemEvent evt) { …} //Event handling
            public static void main () { …} //main method
}
```

Multiple Events

- Register event listener (class GuiDemo –this) with event source (JRadioButton and JCheckBox objects) :

```
public GuiDemo() {
    …
        // register event listener with event source here
        red = new JRadioButton("Red");
        red.addItemListener(this);
        green = new JRadioButton("Green");
        green.addItemListener(this);
        blue = new JRadioButton("Blue", true);
        blue.addItemListener(this);
        bold = new JCheckBox("Bold", true);
        bold.addActionListener(this);
        italic = new JCheckBox("Italic", false);
        italic.addActionListener(this);
    …
```

Multiple Events

- Write event-handling code for ItemEvent :

```
public void itemStateChanged(ItemEvent evt) {
        if (red.isSelected())
                msg.setForeground(Color.RED);
        if (green.isSelected())
                msg.setForeground(Color.GREEN);
        if (blue.isSelected())
                msg.setForeground(Color.BLUE);
}
```

Multiple Events

- Write event-handling code for Action Event :

```
public void actionPerformed(ActionEvent evt) {
        int fontStyle = Font.PLAIN;
        if (bold.isSelected())
                fontStyle += Font.BOLD;
        if (italic.isSelected())
                fontStyle += Font.ITALIC;
        Font myFont = new Font("Serif", fontStyle, 20);
        String str = text.getText();
        msg.setFont(myFont);
        msg.setText(""+str);
  }
```

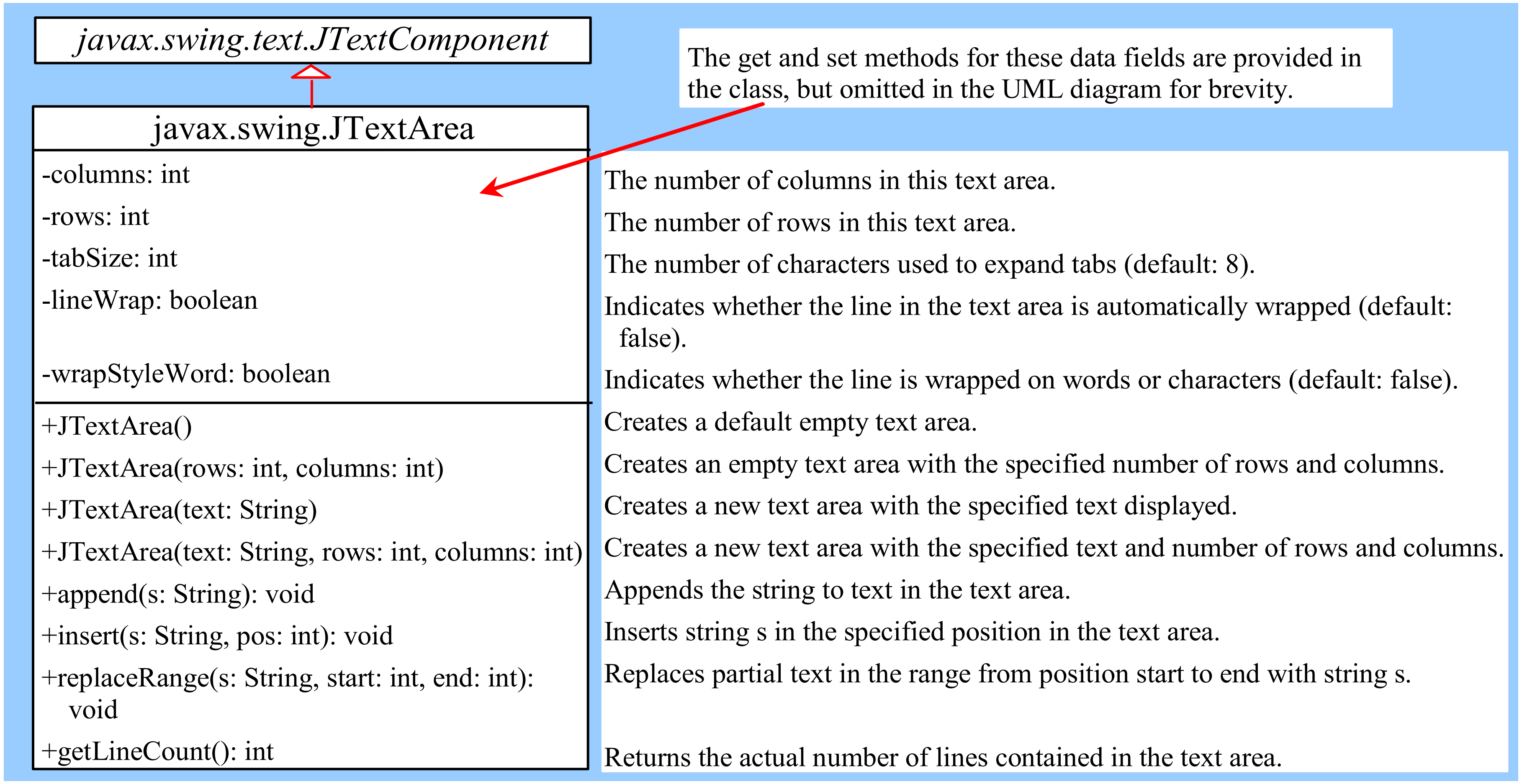
Execution starts here

```
….
….

public static void main (String[] args) {
        GuiDemo frame = new GuiDemo();

        frame.setTitle("GUI DEMO");
        frame.setSize(300,150);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
}
```

Advanced GUI and Event Handlings

JTextArea

If you want to let the user enter multiple lines of text, you cannot use text fields unless you create several of them. The solution is to use `JTextArea`, which enables the user to enter multiple lines of text.



javax.swing.text.JTextComponent

javax.swing.JTextArea

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

-columns: int	The number of columns in this text area.
-rows: int	The number of rows in this text area.
-tabSize: int	The number of characters used to expand tabs (default: 8).
-lineWrap: boolean	Indicates whether the line in the text area is automatically wrapped (default: false).
-wrapStyleWord: boolean	Indicates whether the line is wrapped on words or characters (default: false).
+JTextArea()	Creates a default empty text area.
+JTextArea(rows: int, columns: int)	Creates an empty text area with the specified number of rows and columns.
+JTextArea(text: String)	Creates a new text area with the specified text displayed.
+JTextArea(text: String, rows: int, columns: int)	Creates a new text area with the specified text and number of rows and columns.
+append(s: String): void	Appends the string to text in the text area.
+insert(s: String, pos: int): void	Inserts string s in the specified position in the text area.
+replaceRange(s: String, start: int, end: int): void	Replaces partial text in the range from position start to end with string s.
+getLineCount(): int	Returns the actual number of lines contained in the text area.

`JTextArea` Constructors

- `JTextArea(int rows, int columns)`

  Creates a text area with the specified number of rows and columns.

- `JTextArea(String s, int rows, int columns)`

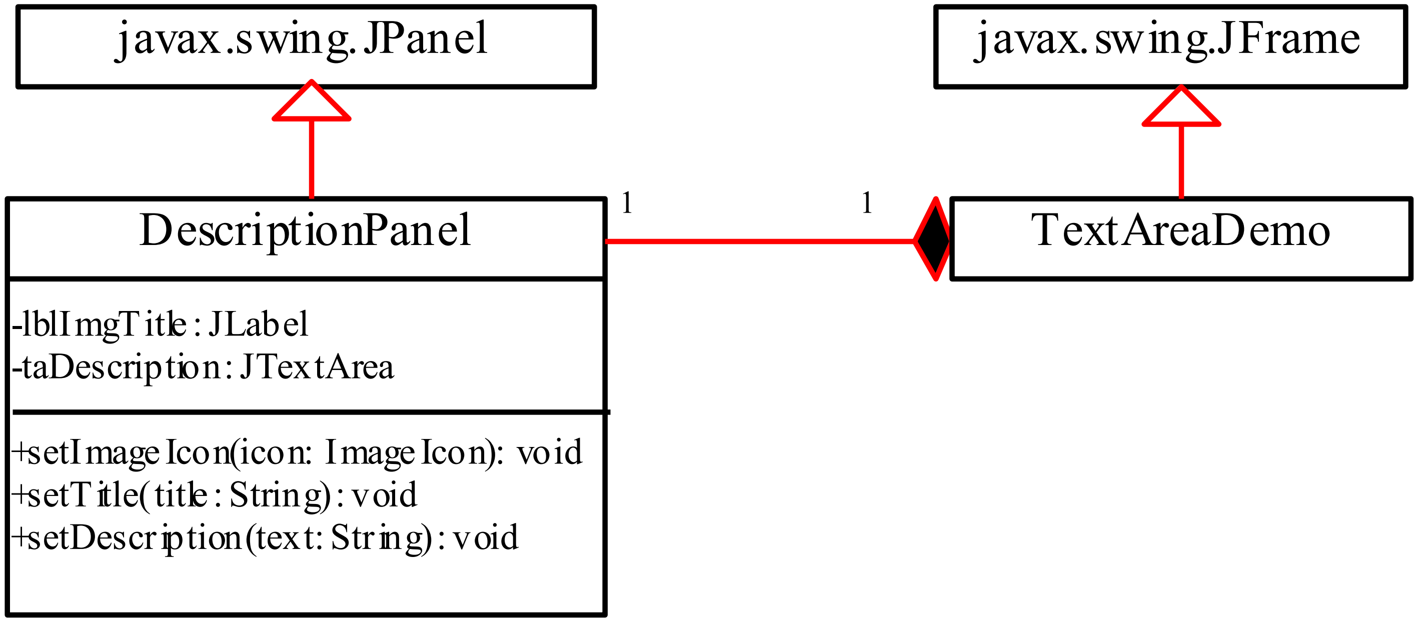
  Creates a text area with the initial text and the number of rows and columns specified.

JTextArea Properties

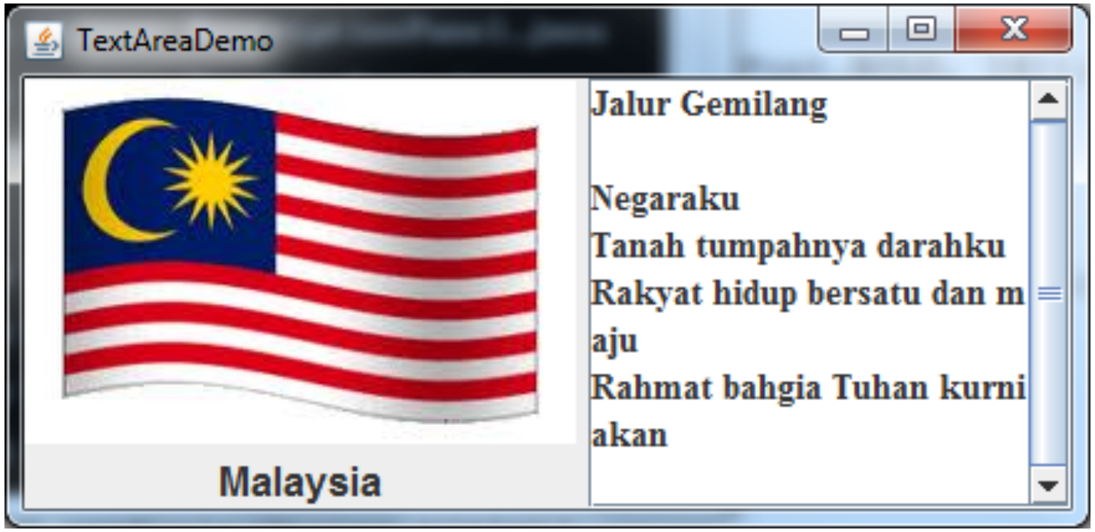
- text
- editable
- columns
- lineWrap
- wrapStyleWord
- rows
- lineCount
- tabSize

Example: Using Text Areas

- This example gives a program that displays an image in a label, a title in a label, and a text in a text area.


javax.swing.JPanel
javax.swing.JFrame
DescriptionPanel
-lblImgTitle: JLabel
-taDescription: JTextArea
+setImageIcon(icon: ImageIcon): void
+setTitle(title: String): void
+setDescription(text: String): void
1
1
TextAreaDemo

Example: JTextArea and ScrollPane Demo



Jalur Gemilang

Negaraku
Tanah tumpahnya darahku
Rakyat hidup bersatu dan m
aju
Rahmat bahgia Tuhan kurni
akan

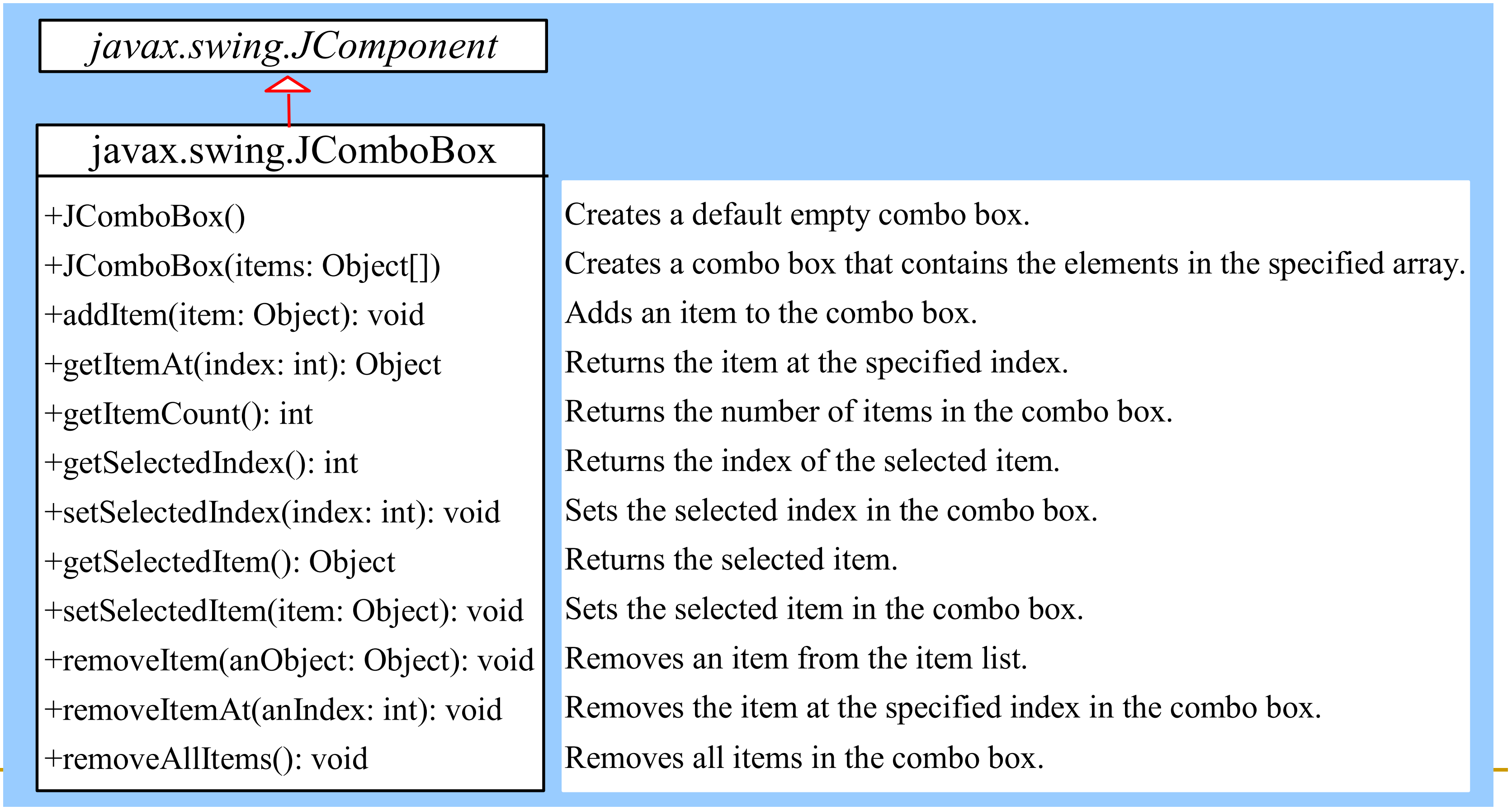
DescriptionPanel

TextAreaDemo

Run

JComboBox

A *combo box* is a simple list of items from which the user can choose. It performs basically the same function as a list, but can get only one value.



+JComboBox()	Creates a default empty combo box.
+JComboBox(items: Object[])	Creates a combo box that contains the elements in the specified array.
+addItem(item: Object): void	Adds an item to the combo box.
+getItemAt(index: int): Object	Returns the item at the specified index.
+getItemCount(): int	Returns the number of items in the combo box.
+getSelectedIndex(): int	Returns the index of the selected item.
+setSelectedIndex(index: int): void	Sets the selected index in the combo box.
+getSelectedItem(): Object	Returns the selected item.
+setSelectedItem(item: Object): void	Sets the selected item in the combo box.
+removeItem(anObject: Object): void	Removes an item from the item list.
+removeItemAt(anIndex: int): void	Removes the item at the specified index in the combo box.
+removeAllItems(): void	Removes all items in the combo box.

JComboBox Methods

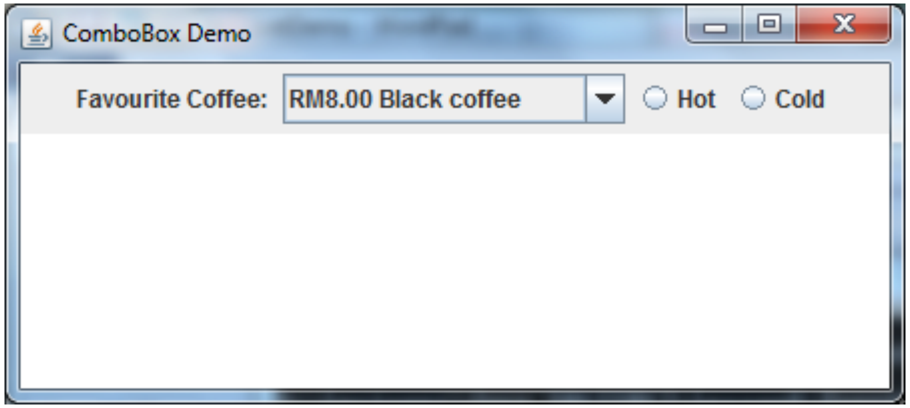
To add an item to a `JComboBox jcbo`, use

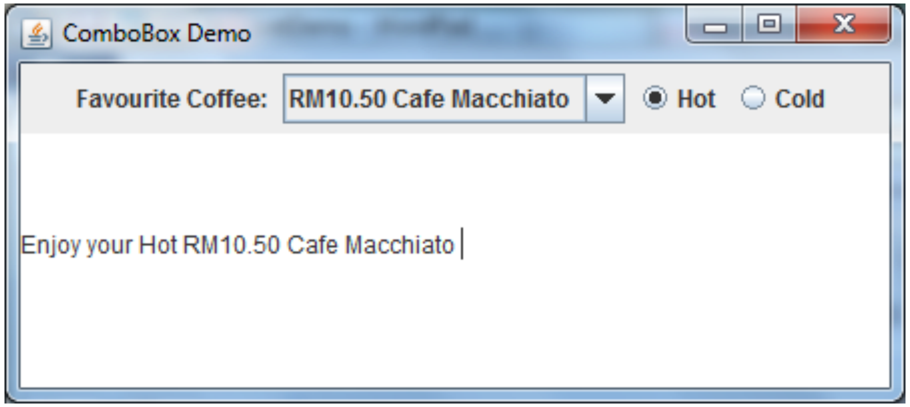
```
jcbo.addItem(Object item)
```

To get an item from `JComboBox jcbo`, use

```
jcbo.getItem()
```

Example: ComboBoxDemo





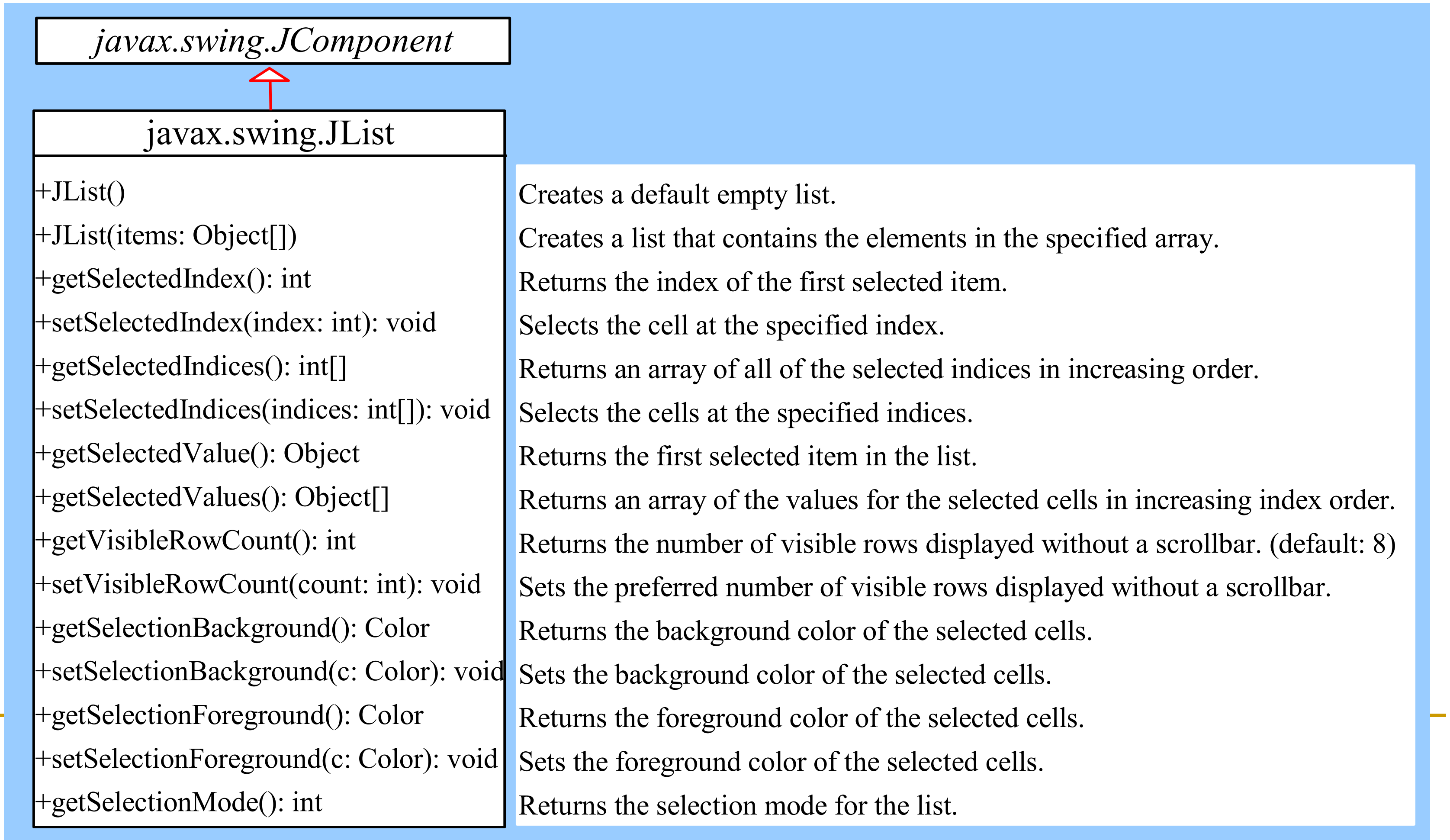
Enjoy your Hot RM10.50 Cafe Macchiato

ComboBoxDemo

Run

JList

A *list* is a component that performs basically the same function as a combo box, but it enables the user to choose a single value or multiple values.



javax.swing.JComponent

javax.swing.JList

+JList()	Creates a default empty list.
+JList(items: Object[])	Creates a list that contains the elements in the specified array.
+getSelectedIndex(): int	Returns the index of the first selected item.
+setSelectedIndex(index: int): void	Selects the cell at the specified index.
+getSelectedIndices(): int[]	Returns an array of all of the selected indices in increasing order.
+setSelectedIndices(indices: int[]): void	Selects the cells at the specified indices.
+getSelectedValue(): Object	Returns the first selected item in the list.
+getSelectedValues(): Object[]	Returns an array of the values for the selected cells in increasing index order.
+getVisibleRowCount(): int	Returns the number of visible rows displayed without a scrollbar. (default: 8)
+setVisibleRowCount(count: int): void	Sets the preferred number of visible rows displayed without a scrollbar.
+getSelectionBackground(): Color	Returns the background color of the selected cells.
+setSelectionBackground(c: Color): void	Sets the background color of the selected cells.
+getSelectionForeground(): Color	Returns the foreground color of the selected cells.
+setSelectionForeground(c: Color): void	Sets the foreground color of the selected cells.
+getSelectionMode(): int	Returns the selection mode for the list.

`JList` Constructors

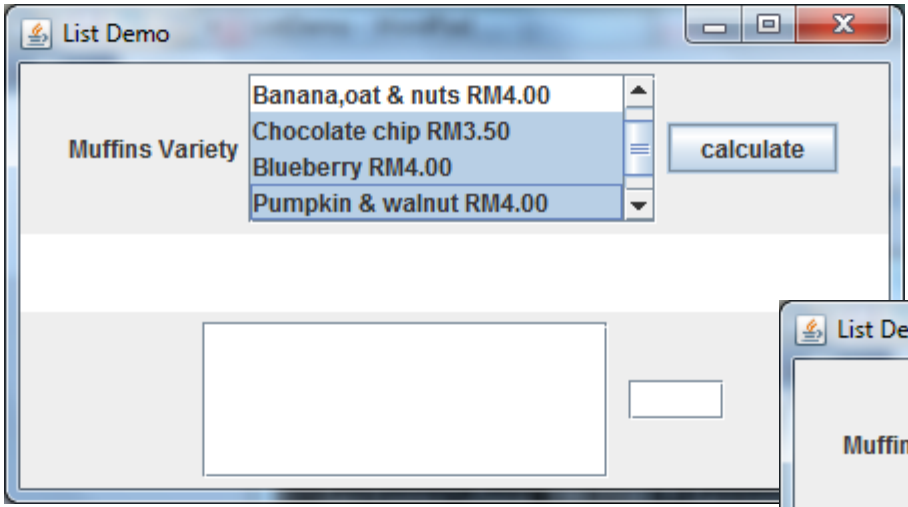
- `JList()`

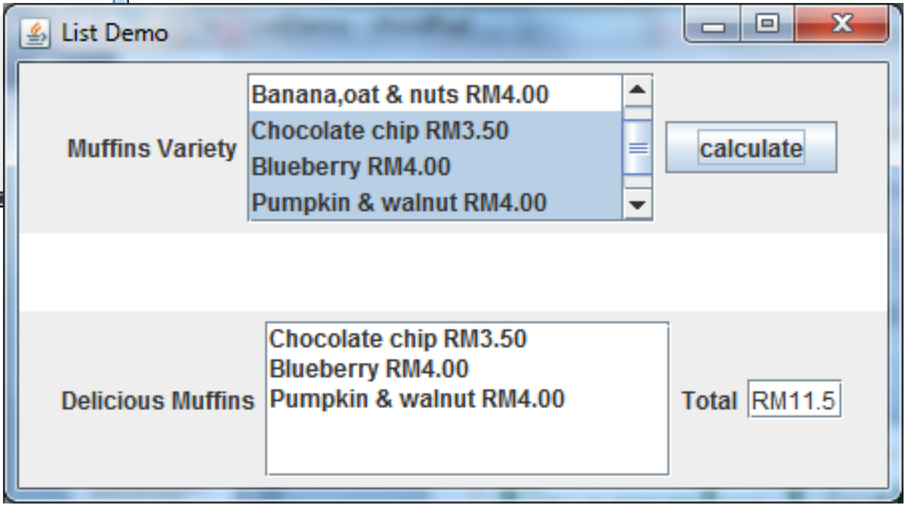
  Creates an empty list.

- `JList(Object[] stringItems)`

  Creates a new list initialized with items.

Example: ListDemo



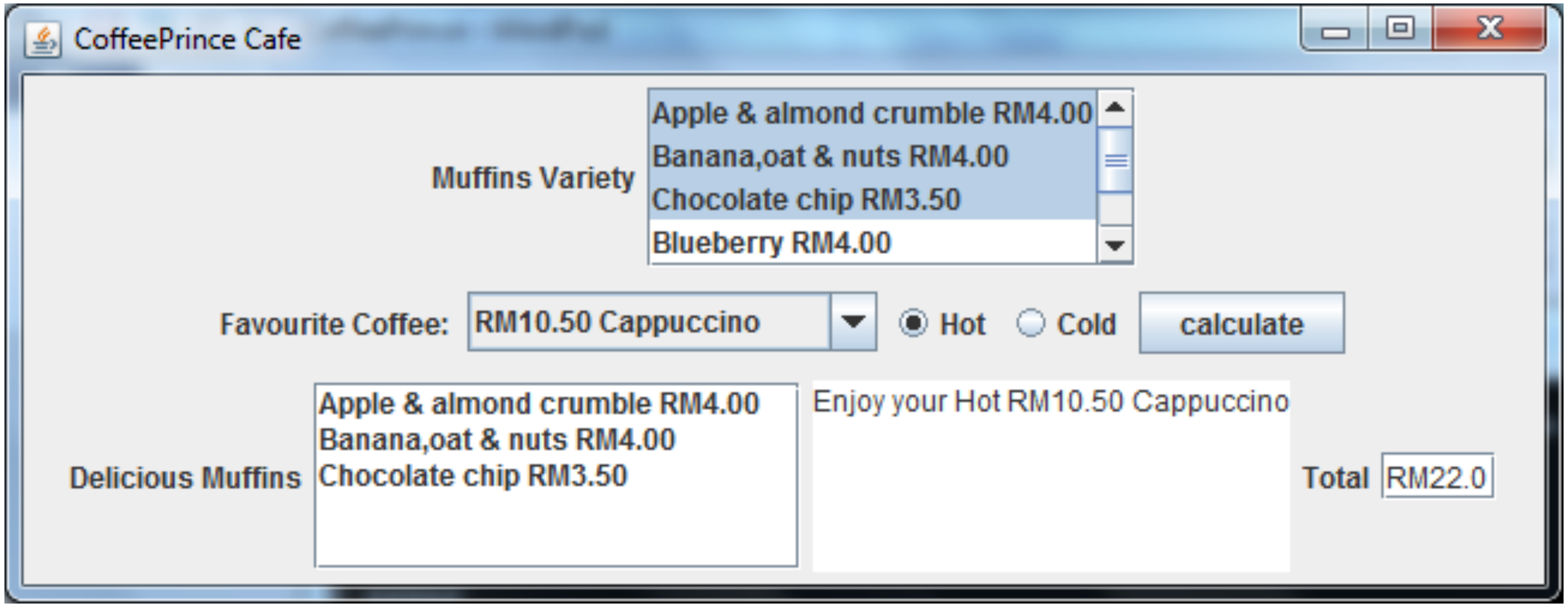


ListDemo

Run

Exercise: ComboBox and List

- Modify previous programs to produce:



Muffins Variety: Apple & almond crumble RM4.00, Banana,oat & nuts RM4.00, Chocolate chip RM3.50 (selected); Blueberry RM4.00

Favourite Coffee: RM10.50 Cappuccino [✓ Hot] Cold

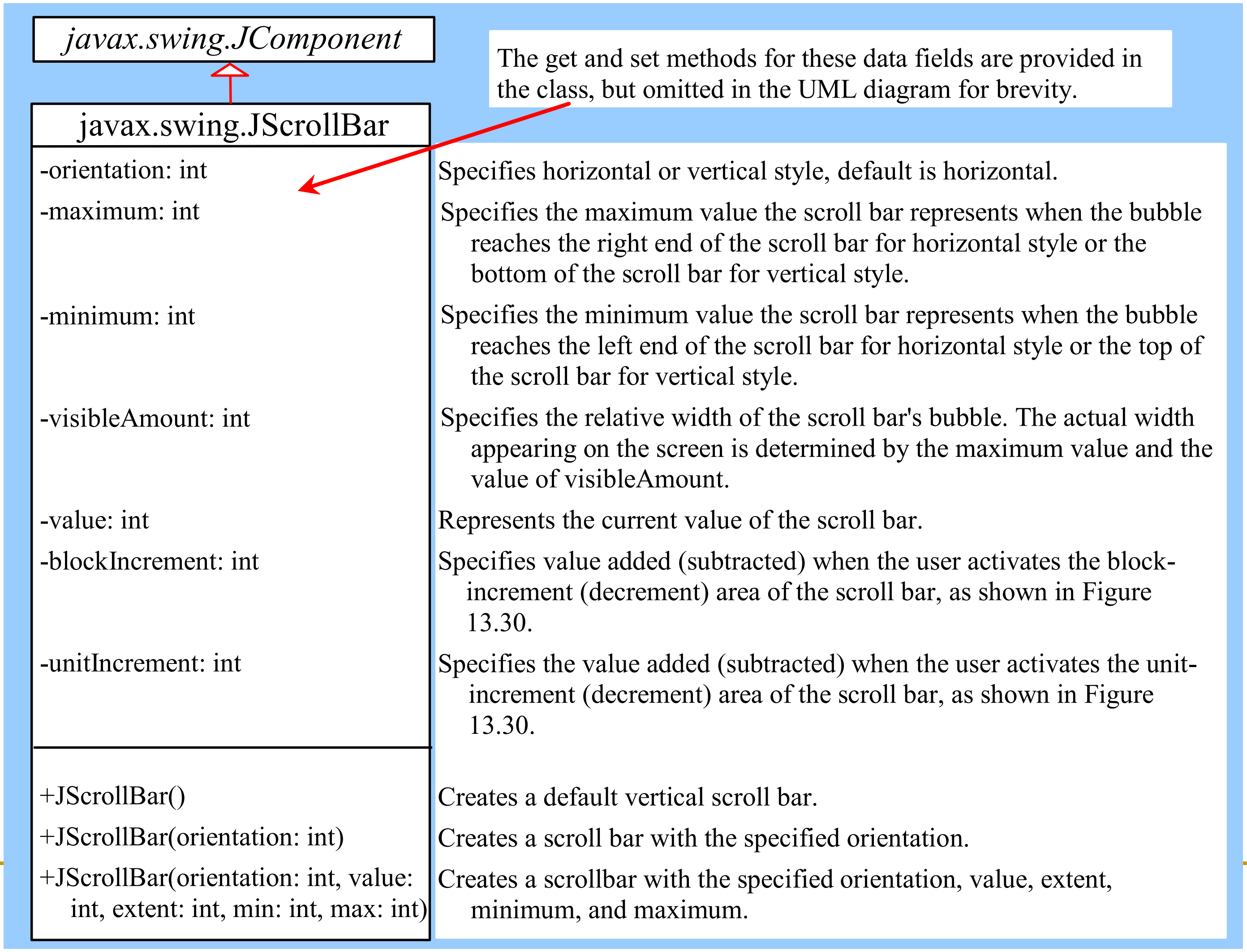
Delicious Muffins:
Apple & almond crumble RM4.00
Banana,oat & nuts RM4.00
Chocolate chip RM3.50

Enjoy your Hot RM10.50 Cappuccino

Total RM22.0

JScrollBar

A *scroll bar* is a control that enables the user to select from a range of values. The scrollbar appears in two styles: *horizontal* and *vertical*.



javax.swing.JComponent

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

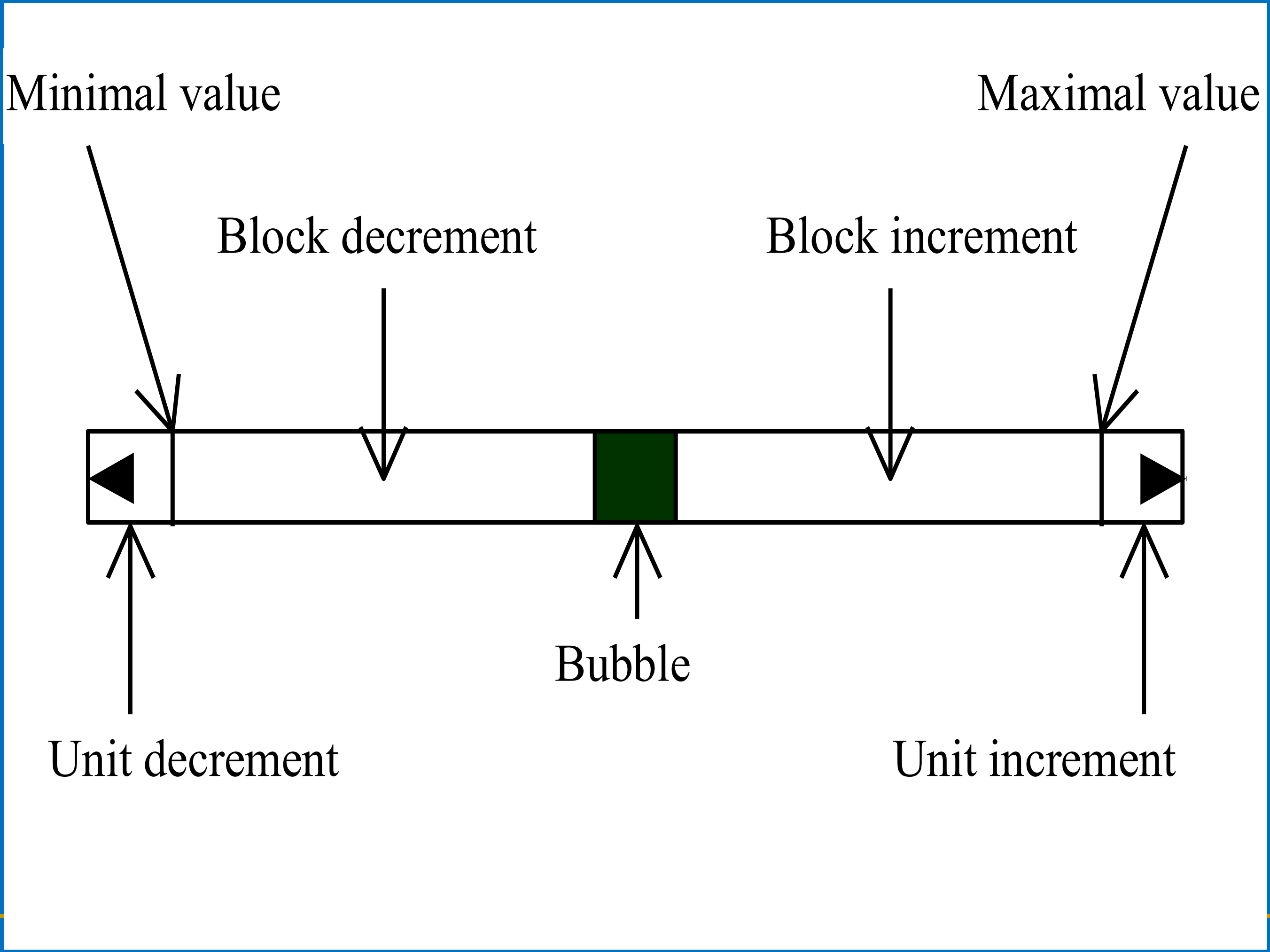
javax.swing.JScrollBar

-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.

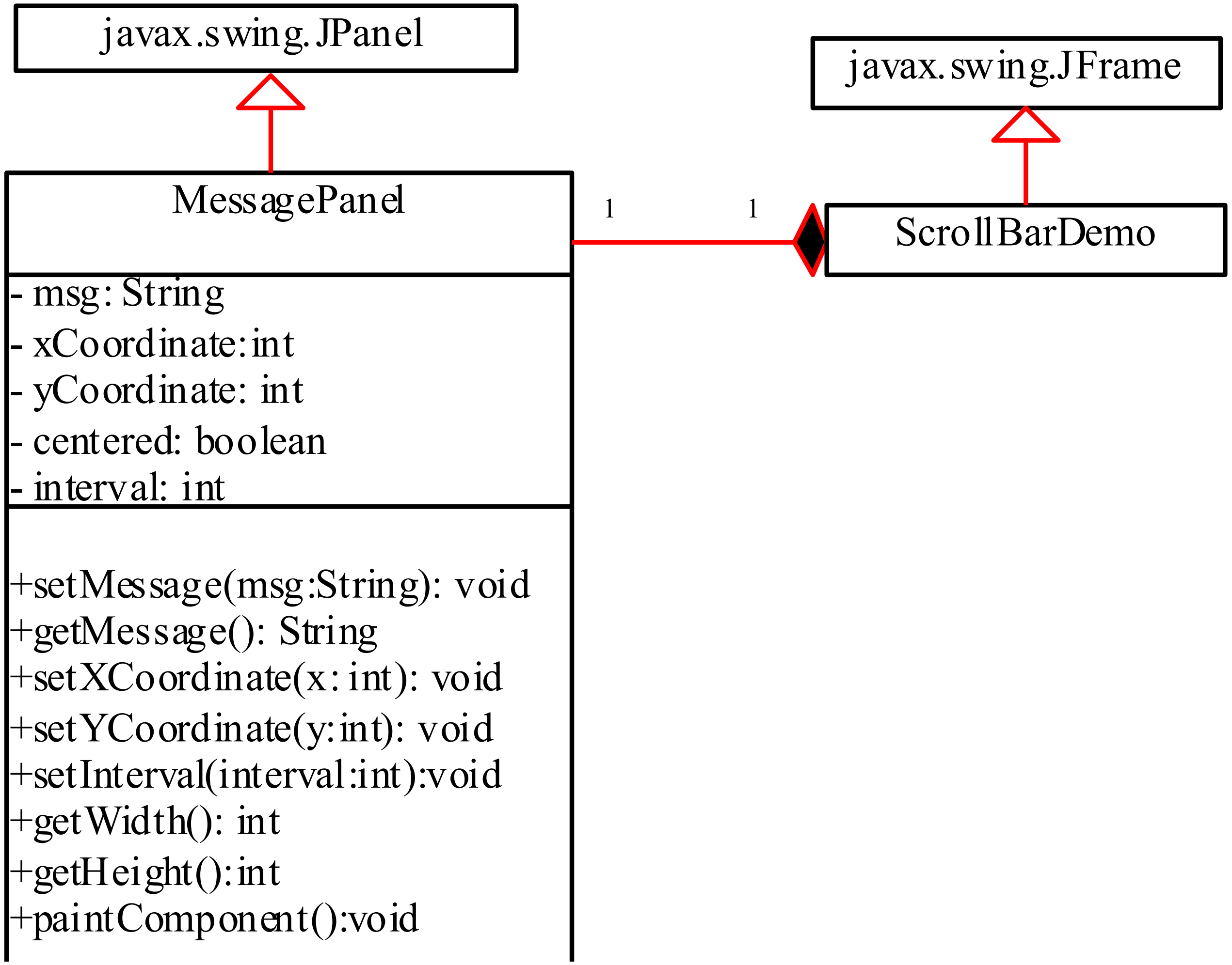
Class JScrollBar

- java.lang.Object
  - java.awt.Component
    - java.awt.Container
      - javax.swing.JComponent
        - javax.swing.JScrollBar
- All Implemented Interfaces:
  - Adjustable, ImageObserver, MenuContainer, Serializable, Accessible
- Direct Known Subclasses: JScrollPane.ScrollBar

Scroll Bar Properties

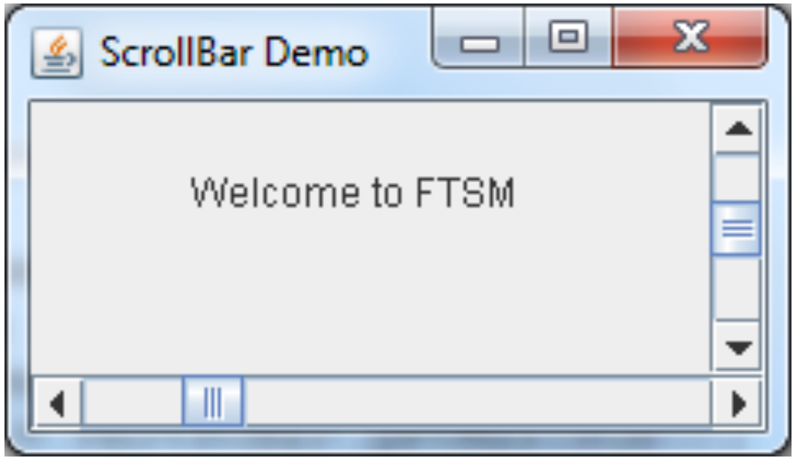


Example: ScrollBar



Example: Using Scrollbars

This example uses horizontal and vertical scrollbars to control a message displayed on a panel. The horizontal scrollbar is used to move the message to the left or the right, and the vertical scrollbar to move it up and down.



Welcome to FTSM

ScrollBarDemo

MessagePanel

Run

SLIDERS

- Sliders can be represented as `JSlider` objects.
- To create a `JSlider` object:

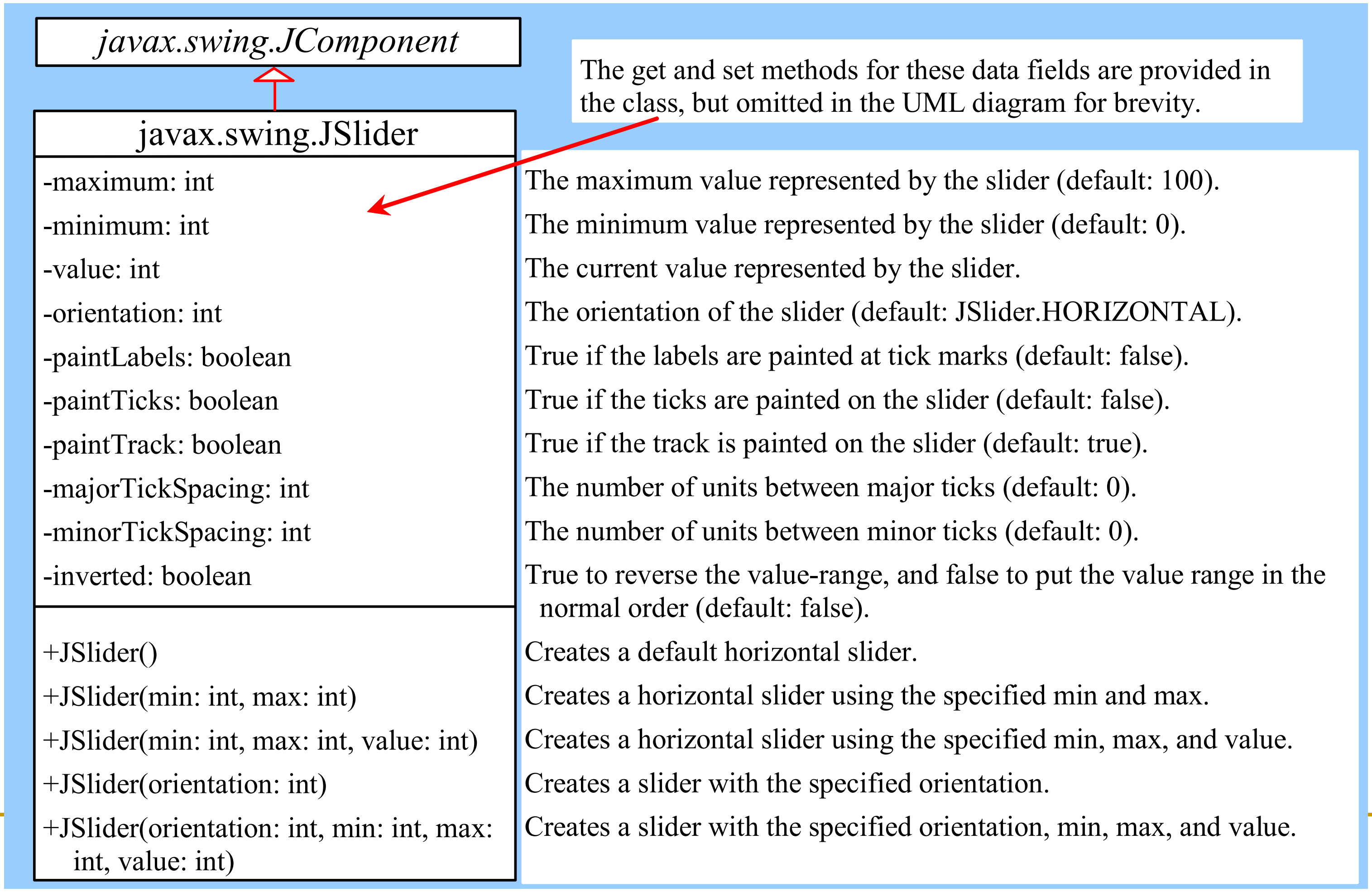
```
public JSlider()
public JSlider(int d)
public JSlider(int d, int min, int max, int val)
```

where

`d`:	direction of slider
	Possible values:
	`JSlider.HORIZONTAL`
	`JSlider.VERTICAL`
`min..max`:	range of values for slider
`val`:	initial value

JSlider

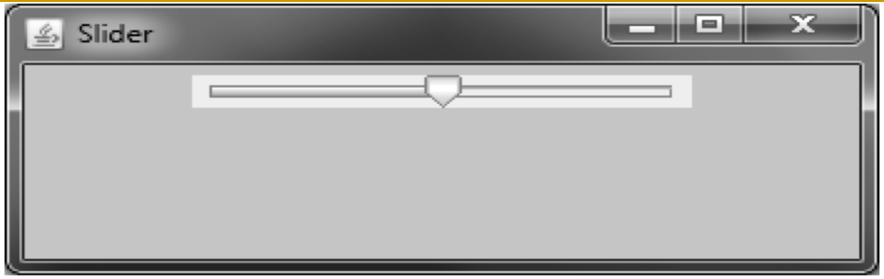
JSlider is similar to JScrollBar, but JSlider has more properties and can appear in many forms.



javax.swing.JComponent

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

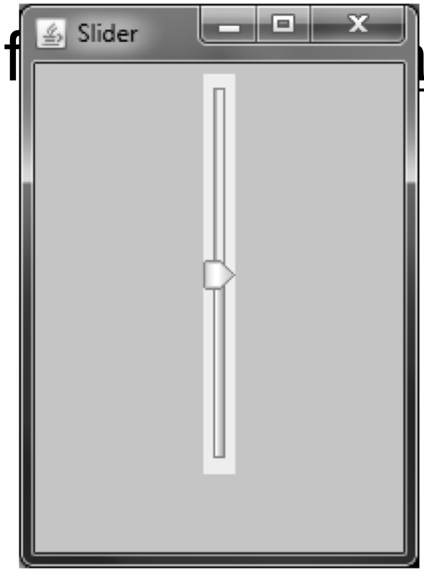
javax.swing.JSlider	
-maximum: int	The maximum value represented by the slider (default: 100).
-minimum: int	The minimum value represented by the slider (default: 0).
-value: int	The current value represented by the slider.
-orientation: int	The orientation of the slider (default: JSlider.HORIZONTAL).
-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.



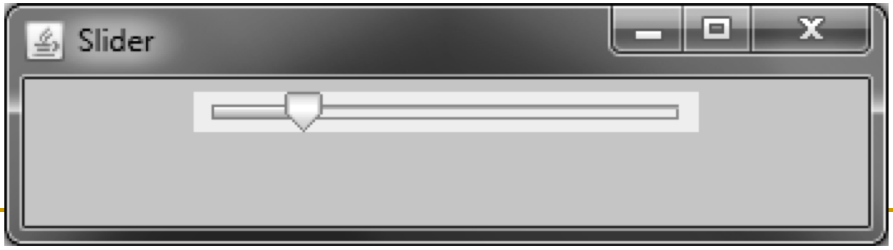
- Examples:

```
new JSlider()
```

  - creates a slider with the f ult properties:
    - horizontal
    - range 0..100
    - initial value: 50

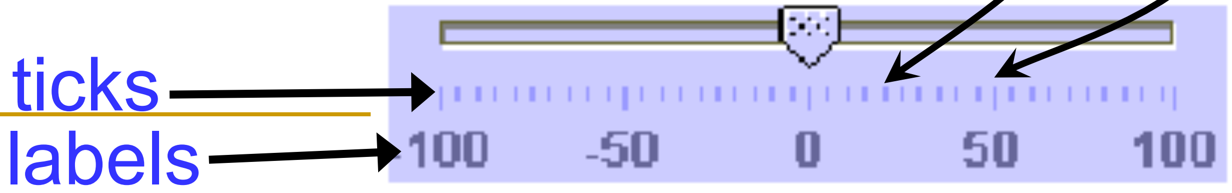


```
new JSlider(JSlider.VERTICAL)
new JSlider(JSlider.HORIZONTAL, 1, 100, 20)
```



METHODS FOR `JSlider` OBJECTS

- **void setMajorTickSpacing(int spacing)**
  - Sets spacing between major ticks
- **void setMinorTickSpacing(int spacing)**
  - Sets spacing between minor ticks
- **void setPaintTicks(boolean status)**
  - Sets status on whether ticks are to be displayed
- **void setPaintLabels(boolean status)**
  - Sets status on whether labels are to be displayed



```
public class SliderDemo extends JFrame {
    public SliderDemo() {
        Container pane = getContentPane();
        pane.setBackground(Color.white);
        pane.setLayout(new FlowLayout());
        JSlider slider = new JSlider(JSlider.HORIZONTAL, 0, 100, 1);
        slider.setMajorTickSpacing(25);
        slider.setMinorTickSpacing(1);
        slider.setPaintTicks(true);
        slider.setPaintLabels(true);
        pane.add(new JLabel("Volume:"));
        pane.add(slider);
        pane.add(new JLabel("Brightness:"));
        pane.add(new JSlider());
        pane.add(new JButton("ON/OFF"));
    }
//main  here
}
```

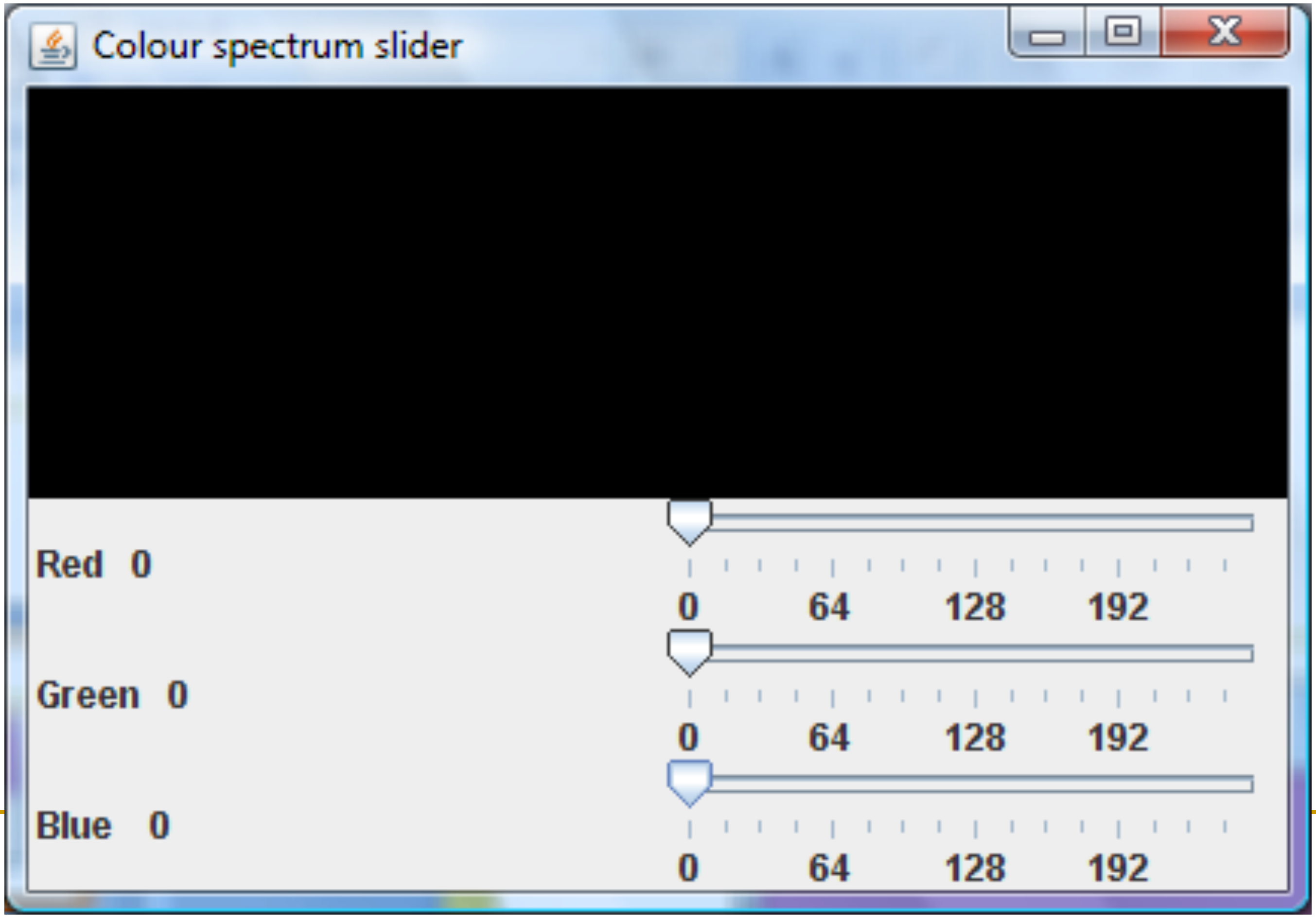
Run

ChangeEvent

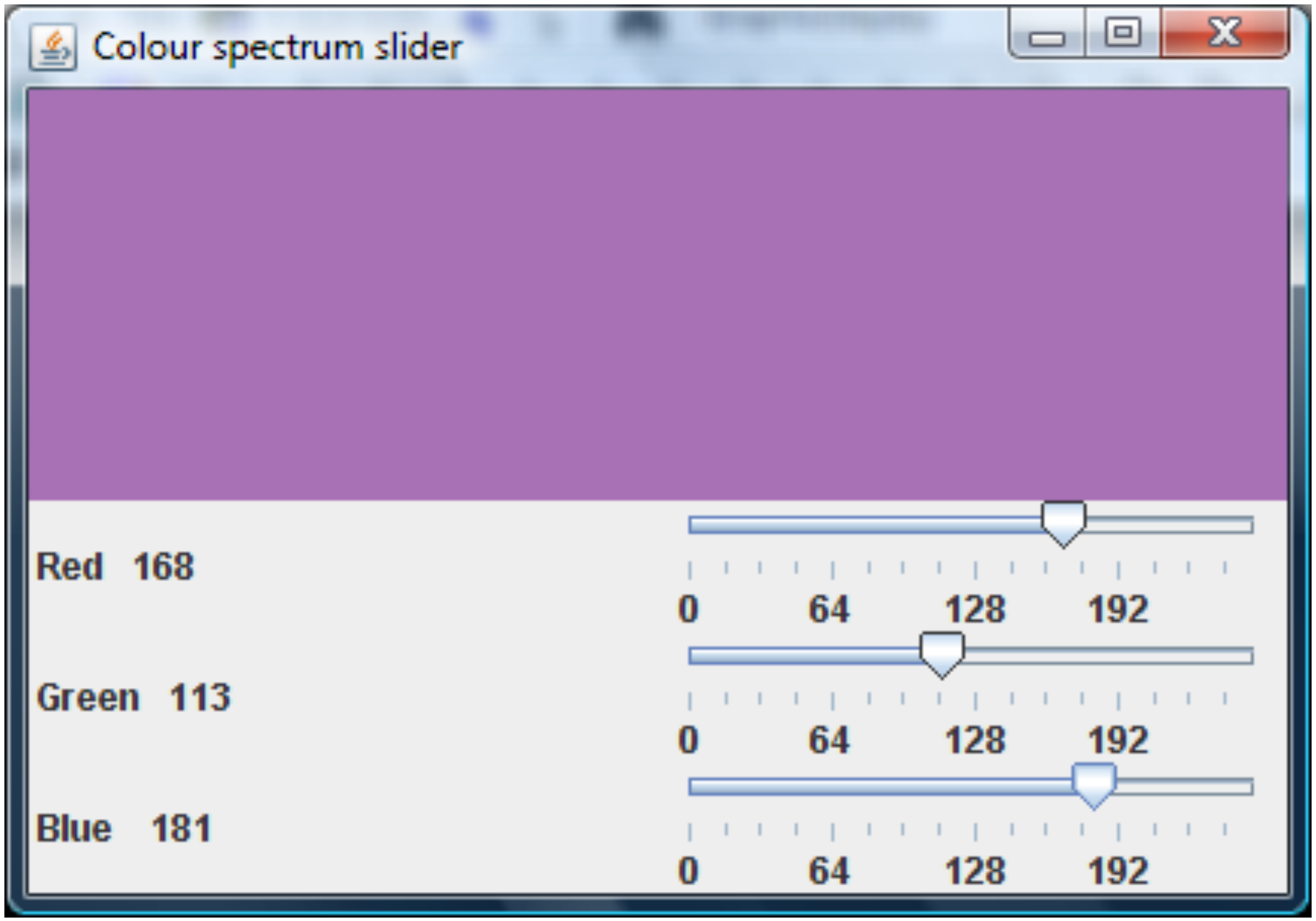
- Change events are created when there are changes to the event source.
- The Swing components that fire change events include **JSlider**
- Change Listener defines what should be done when an item StateChanged message
- Change event is included in javax.swing.event package

ChangeEvent

- Example:



ChangeEvent



ColorSpectrumSlider

Run

ChangeEvent

```
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.event.*;
public class ColourSlider extends JFrame implements ChangeListener
    {
    private JLabel rLabel, gLabel, bLabel;
    private JSlider red, green, blue;
    private Color colour;
    private JPanel c;
```

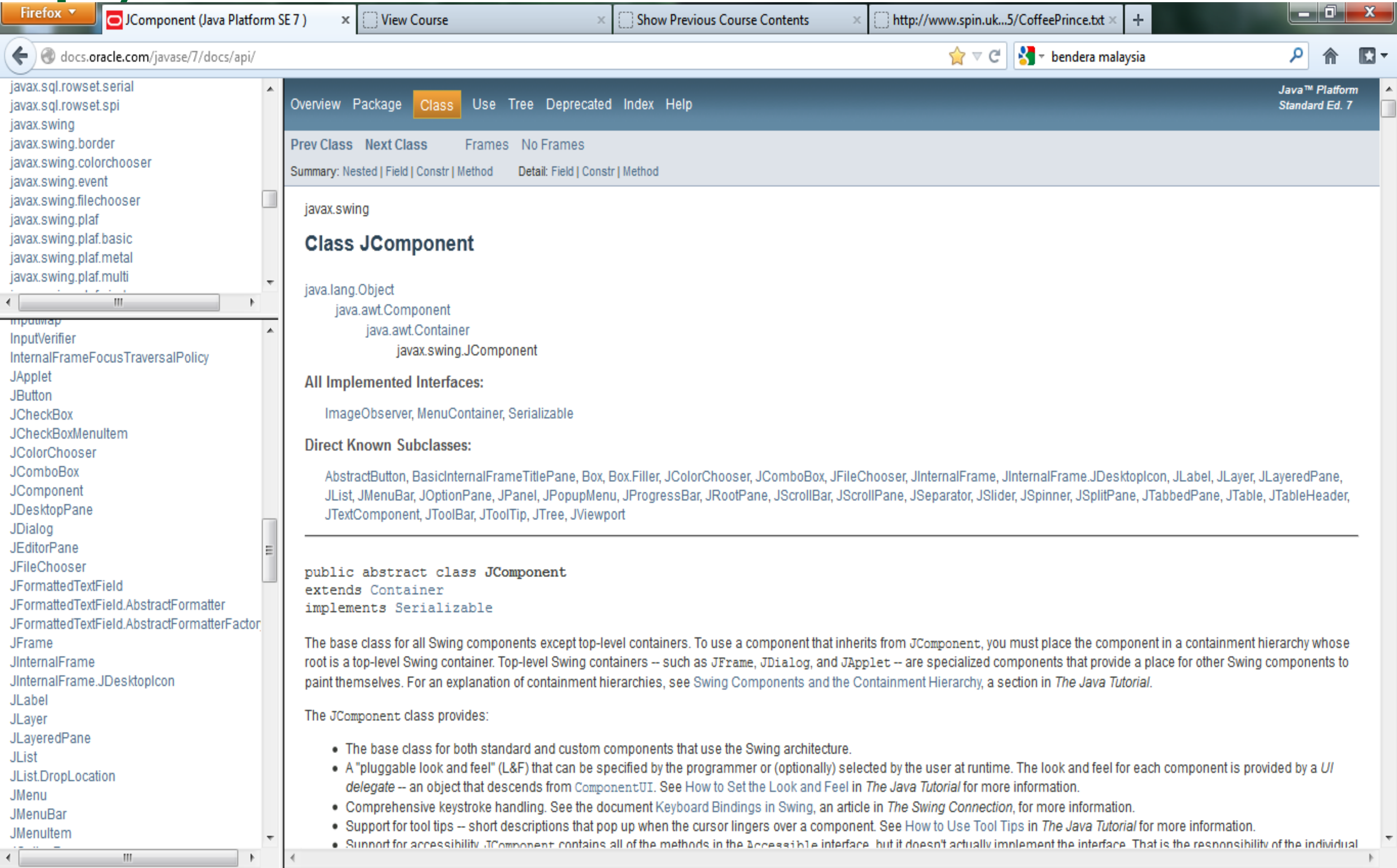
```
public static void main (String[] args) {
        ColourSlider frame = new ColourSlider();
        frame.setTitle("Colour spectrum slider");
        frame.setSize(430,300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
}
```

```
public ColourSlider() {
    Container pane = getContentPane();
    pane.setBackground(Color.white);
    pane.setLayout(new BorderLayout());
     JPanel p = new JPanel();
    p.setLayout(new GridLayout(3, 2));
    rLabel = new JLabel(" Red 0");
    p.add(rLabel);
    red = new JSlider(JSlider.HORIZONTAL, 0, 255, 0);
    red.setMajorTickSpacing(64);
    red.setMinorTickSpacing(16);
    red.setPaintTicks(true);
    red.setPaintLabels(true);
    red.addChangeListener(this); //register event source
    p.add(red);
    gLabel = new JLabel(" Green 0");
    p.add(gLabel);
    green = new JSlider(JSlider.HORIZONTAL, 0, 255, 0);
    green.setMajorTickSpacing(64);
    green.setMinorTickSpacing(16);
    green.setPaintTicks(true);
    green.setPaintLabels(true);
    green.addChangeListener(this); //register event source
```

```
  p.add(green);
  bLabel = new JLabel(" Blue 0");
  p.add(bLabel);
  blue = new JSlider(JSlider.HORIZONTAL, 0, 255, 0);
  blue.setMajorTickSpacing(64);
  blue.setMinorTickSpacing(16);
  blue.setPaintTicks(true);
  blue.setPaintLabels(true);
  blue.addChangeListener(this); //register event source
  p.add(blue);
  pane.add(p, BorderLayout.SOUTH);


  c = new JPanel();
  colour = new Color(0, 0, 0);
  c.setBackground(colour);
  pane.add(c, BorderLayout.CENTER);
} // end of constructor Slider
```

```
    public void stateChanged(ChangeEvent e) {
            int r, g, b;
            r = red.getValue();
            g = green.getValue();
            b = blue.getValue();
            rLabel.setText(" Red    " + r);
            gLabel.setText(" Green    " + g);
            bLabel.setText(" Blue     " + b);
            colour = new Color(r, g, b);
            c.setBackground(colour);
            c.repaint();  //method inherited from class Component
    }
} // end of program
```

http://docs.oracle.com/javase/7/docs/api/


Firefox
JComponent (Java Platform SE 7)
View Course
Show Previous Course Contents
http://www.spin.uk...5/CoffeePrince.txt
docs.oracle.com/javase/7/docs/api/
bendera malaysia
Overview Package Class Use Tree Deprecated Index Help
Java™ Platform Standard Ed. 7
Prev Class Next Class Frames No Frames
Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

javax.sql.rowset.serial
javax.sql.rowset.spi
javax.swing
javax.swing.border
javax.swing.colorchooser
javax.swing.event
javax.swing.filechooser
javax.swing.plaf
javax.swing.plaf.basic
javax.swing.plaf.metal
javax.swing.plaf.multi

InputVerifier
InternalFrameFocusTraversalPolicy
JApplet
JButton
JCheckBox
JCheckBoxMenuItem
JColorChooser
JComboBox
JComponent
JDesktopPane
JDialog
JEditorPane
JFileChooser
JFormattedTextField
JFormattedTextField.AbstractFormatter
JFormattedTextField.AbstractFormatterFactor
JFrame
JInternalFrame
JInternalFrame.JDesktopIcon
JLabel
JLayer
JLayeredPane
JList
JList.DropLocation
JMenu
JMenuBar
JMenuItem

javax.swing

Class JComponent

java.lang.Object
  java.awt.Component
    java.awt.Container
      javax.swing.JComponent

All Implemented Interfaces:

ImageObserver, MenuContainer, Serializable

Direct Known Subclasses:

AbstractButton, BasicInternalFrameTitlePane, Box, Box.Filler, JColorChooser, JComboBox, JFileChooser, JInternalFrame, JInternalFrame.JDesktopIcon, JLabel, JLayer, JLayeredPane, JList, JMenuBar, JOptionPane, JPanel, JPopupMenu, JProgressBar, JRootPane, JScrollBar, JScrollPane, JSeparator, JSlider, JSpinner, JSplitPane, JTabbedPane, JTable, JTableHeader, JTextComponent, JToolBar, JToolTip, JTree, JViewport

```
public abstract class JComponent
extends Container
implements Serializable
```

The base class for all Swing components except top-level containers. To use a component that inherits from `JComponent`, you must place the component in a containment hierarchy whose root is a top-level Swing container. Top-level Swing containers -- such as `JFrame`, `JDialog`, and `JApplet` -- are specialized components that provide a place for other Swing components to paint themselves. For an explanation of containment hierarchies, see Swing Components and the Containment Hierarchy, a section in *The Java Tutorial*.

The `JComponent` class provides:

- The base class for both standard and custom components that use the Swing architecture.
- A "pluggable look and feel" (L&F) that can be specified by the programmer or (optionally) selected by the user at runtime. The look and feel for each component is provided by a *UI delegate* -- an object that descends from `ComponentUI`. See How to Set the Look and Feel in *The Java Tutorial* for more information.
- Comprehensive keystroke handling. See the document Keyboard Bindings in Swing, an article in *The Swing Connection*, for more information.
- Support for tool tips -- short descriptions that pop up when the cursor lingers over a component. See How to Use Tool Tips in *The Java Tutorial* for more information.
- Support for accessibility. `JComponent` contains all of the methods in the `Accessible` interface, but it doesn't actually implement the interface. That is the responsibility of the individual

Class JComponent

- The following are a number of member method inherited from Class `JComponent:`

Rectangle	getBounds(Rectangle rv)
	`Stores the bounds of this component into "return value" rv and returns rv.`

Graphics	getGraphics()
	`Returns this component's graphics context, which lets you draw on a component.`

int	getHeight()
	`Returns the current height of this component.`

Point	getLocation(Point rv)
	`Stores the x,y origin of this component into "return value" rv and returns rv.`

Dimension	getSize(Dimension rv)
	`Stores the width/height of this component into "return value" rv and returns rv.`

int	getWidth()
	`Returns the current width of this component.`

int	getX()
	`Returns the current x coordinate of the component's origin.`

int	getY()
	`Returns the current y coordinate of the component's origin.`

void	paint(Graphics g) Invoked by Swing to draw components.
protected void	paintComponent(Graphics g) Calls the UI delegate's paint method, if the UI delegate is non-null.
void	reshape(int x, int y, int w, int h) Moves and resizes this component.
void	setBackground(Color bg) Sets the background color of this component.
void	setBorder(Border border) Sets the border of this component.
void	setFont(Font font) Sets the font for this component.
void	setForeground(Color fg) Sets the foreground color of this component.
void	setVisible(boolean aFlag) Makes the component visible or invisible.
void	update(Graphics g) Calls paint.

Class Container

- The following are a number of member methods inherited from class `Container`:

Component	add(Component comp)
	`Appends the specified component to the end of this container.`

Component	add(Component comp, int index) `Adds the specified component to this container at the given position.`

void	add(Component comp, Object constraints)
	`Adds the specified component to the end of this container.`

Component	add(String name, Component comp)
	`Adds the specified component to this container.`

Component	getComponent(int n)
	`Gets the nth component in this container.`

Component	getComponentAt(int x, int y) `Locates the component that contains the x,y position.`

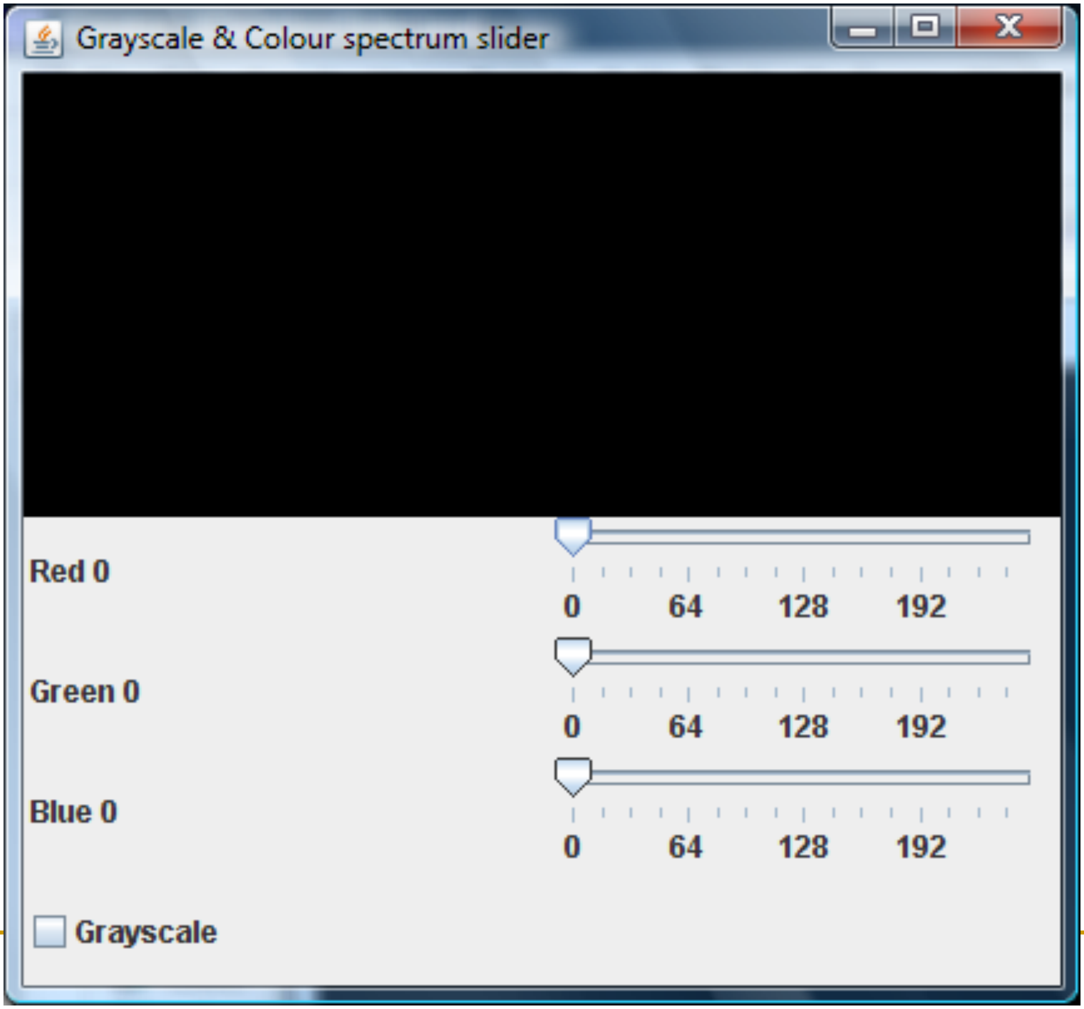
int	getComponentCount()
	`Gets the number of components in this panel.`

Component[]	getComponents()
	`Gets all the components in this container.`

LayoutManager	getLayout() Gets the layout manager for this container.
void	paint(Graphics g) Paints the container.
void	paintComponents(Graphics g) Paints each of the components in this container.
void	remove(Component comp) Removes the specified component from this container.
void	remove(int index) Removes the component, specified by index, from this container.
void	removeAll() Removes all the components from this container.
void	setLayout(LayoutManager mgr) Sets the layout manager for this container.

Multiple Events

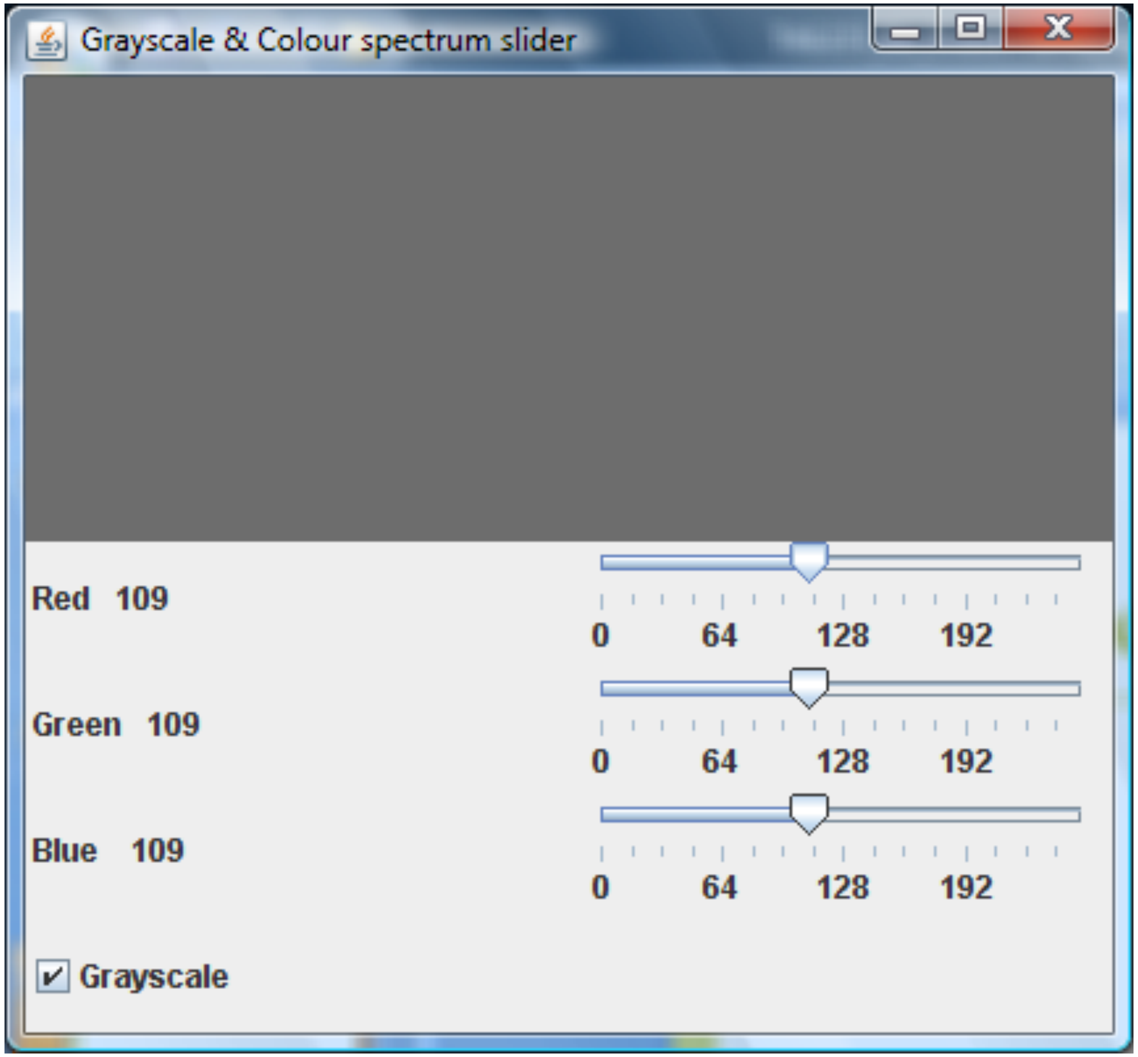
- A modification of the same appliaction:



CheckBoxSlider

Run

Multiple Events



Multiple Events

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

public class CheckboxSlider extends JFrame implements
    ChangeListener, ItemListener {
    private JLabel rLabel, gLabel, bLabel;
    private JSlider red, green, blue;
    private Color colour;
    private JPanel c;
    private JCheckBox cbGrayscale;
    private boolean grayscale = false;
```

Multiple Events

```
public CheckboxSlider() {
  Container pane = getContentPane();
    pane.setBackground(Color.white);
    pane.setLayout(new BorderLayout());

    JPanel p = new JPanel();
    p.setLayout(new GridLayout(4,2,5,5));
   …
   …
    cbGrayscale = new JCheckBox("Grayscale", false);
    cbGrayscale.addItemListener(this);
    p.add(cbGrayscale);
}
```

```
public void itemStateChanged(ItemEvent e) {
        if (cbGrayscale.isSelected())
                grayscale = true;
        else
                grayscale = false;
}
public void stateChanged(ChangeEvent e) {
        int r, g, b;
        if (grayscale) {
                JSlider slider = (JSlider) e.getSource();
                r = g = b = slider.getValue();
                red.setValue(r);
                green.setValue(g);
                blue.setValue(b);
        }
        else {
                r = red.getValue();
                g = green.getValue();
                b = blue.getValue();
        }
        ::::: //same as in Slider
}
```

Other Events

- KeyEvent
- MouseEvent
- FocusEvent
- ComponentEvent
- ContainerEvent
- WindowEvent
- AdjustmentEvent

Other Event Listeners

- KeyListener
- MouseListener
- MouseMotionListener
- FocusListener
- ComponentListener
- ContainerListener
- WindowListener
- AdjustmentListener

Creating Multiple Windows

The following slides show step-by-step how to create an additional window from an application or applet.

Creating Additional Windows, Step 1

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`.

Creating Additional Windows, Step 2

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

Example:

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

Creating Additional Windows, Step 3

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

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

Creating Additional Windows, Step 4

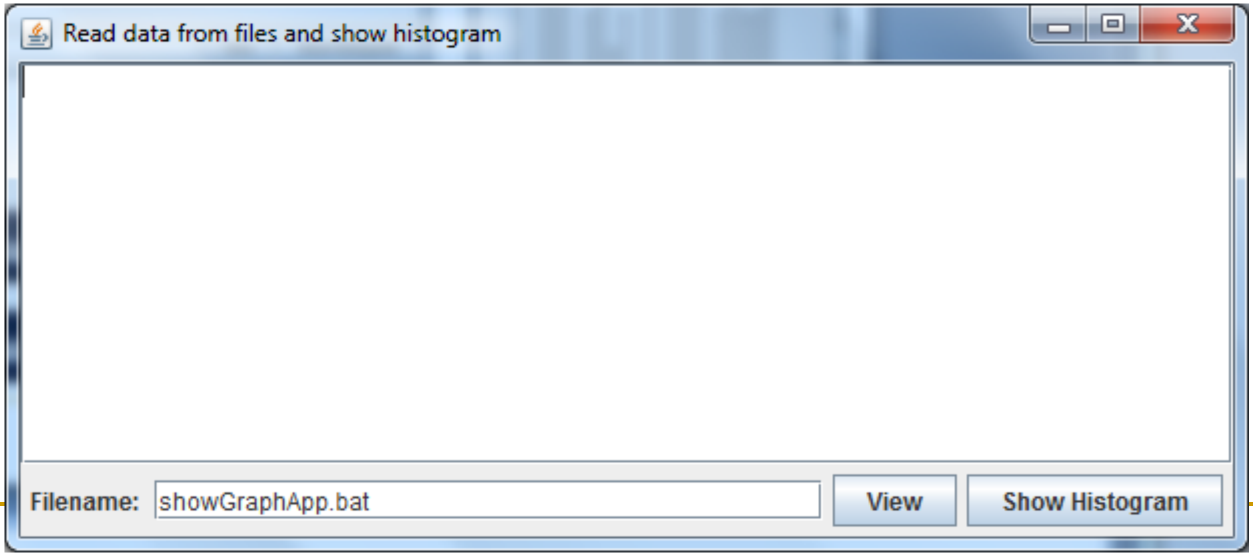
Step 4: Override the `actionPerformed()` method as follows:

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

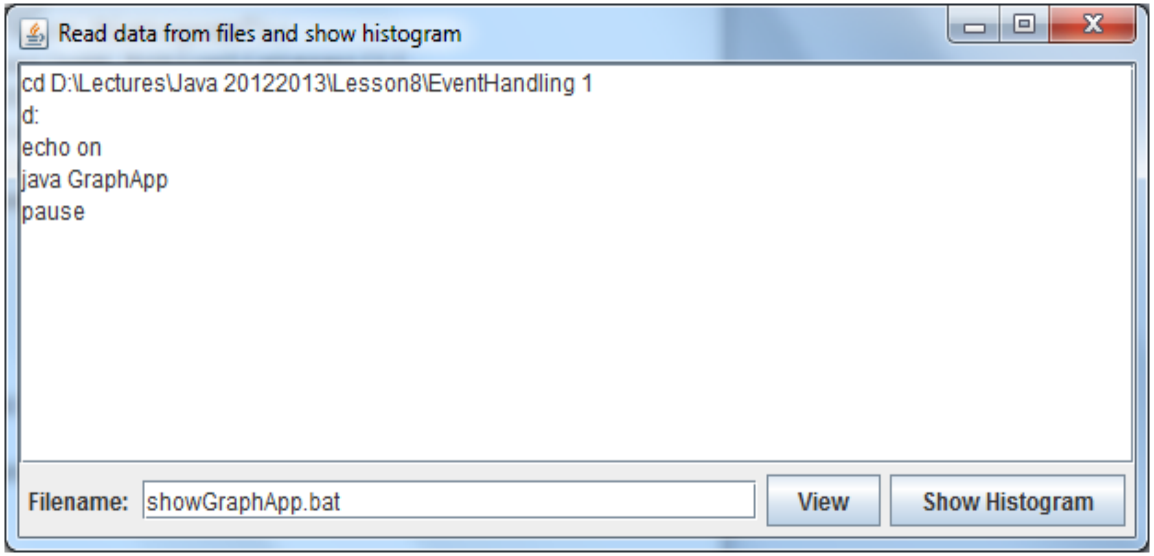
Example: Creating Multiple Windows

This program will :

- display the content of the filename in a text area when user entered the filename in the textfield and click the button “View”
- create another window to display a histogram of letter counts from the text area

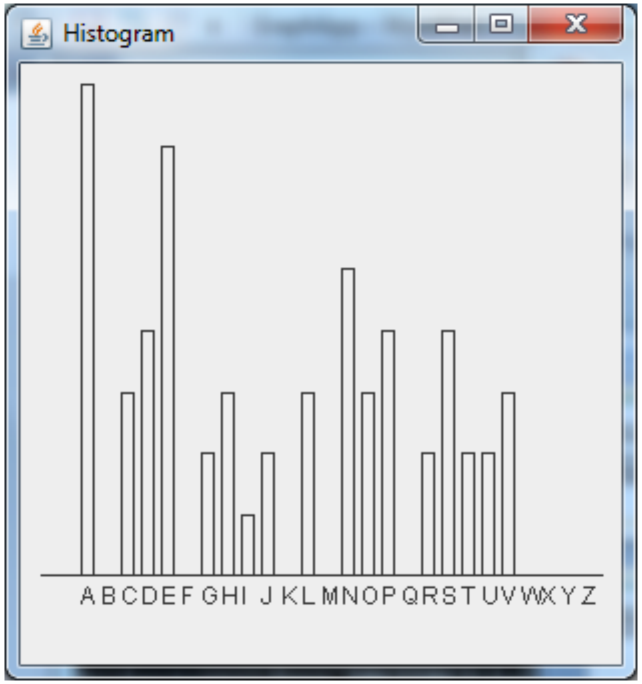


Example, cont.



```
cd D:\Lectures\Java 20122013\Lesson8\EventHandling 1
d:
echo on
java GraphApp
pause
```

Filename: showGraphApp.bat

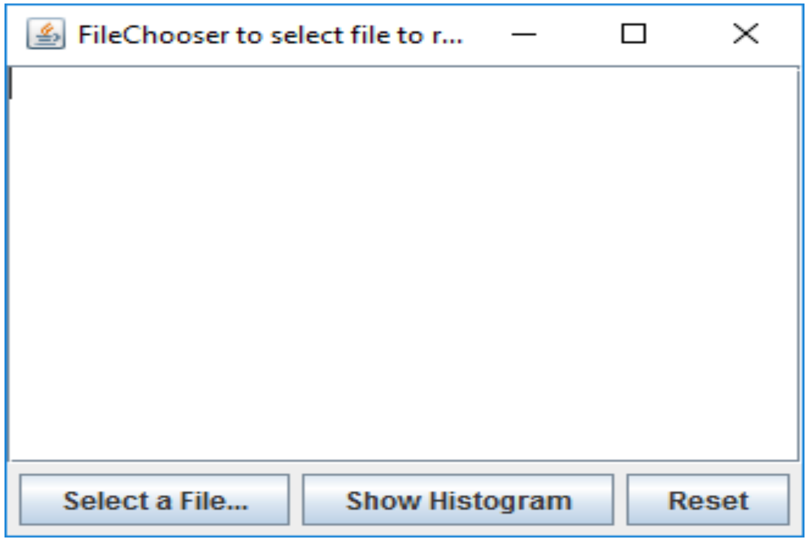


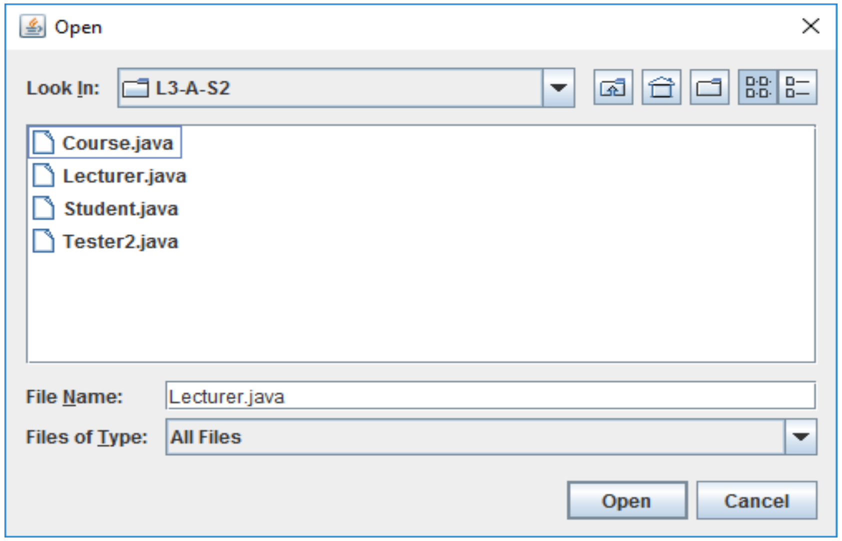
GraphApp

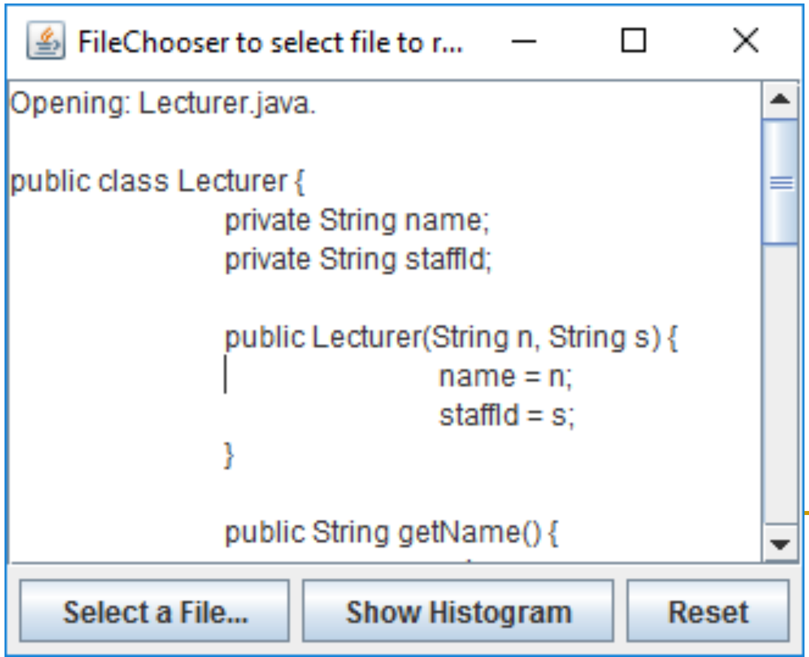
Run

Histogram

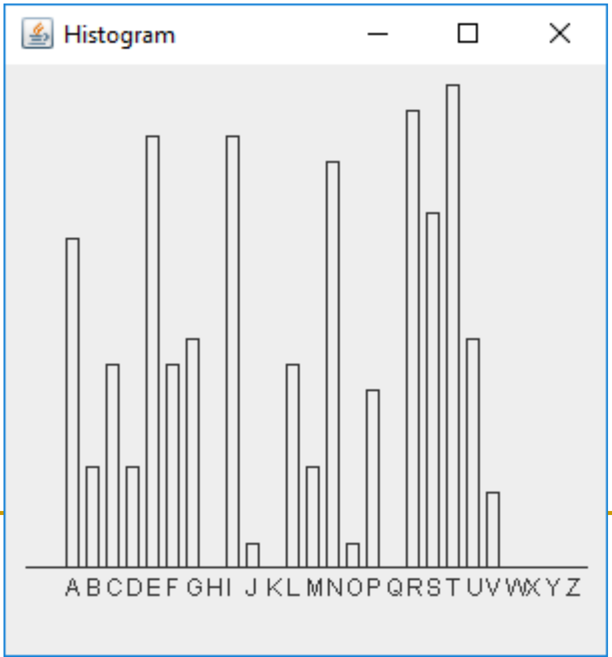
Graph App Using FileChooser







```
Opening: Lecturer.java.

public class Lecturer {
        private String name;
        private String staffId;

        public Lecturer(String n, String s) {
                name = n;
                staffId = s;
        }

        public String getName() {
```



Program

Run