GUIs continued

- Checkboxes
- Radio Buttons
- Read-Only Text Fields
- Lists
- Combo Boxes
- Images
- Hotkeys and Tool Tips
- Sliders

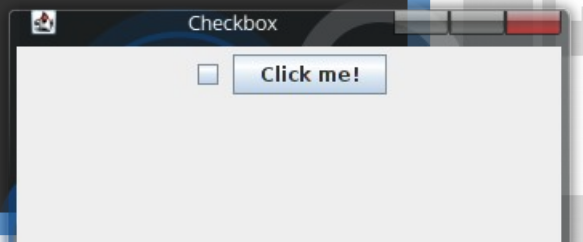
Checkboxes

- Checkboxes are used as a GUI representation of a boolean value
- They can either be checked or unchecked.
- Swing Provides a **JCheckBox** class
- A JCheckBox object has an isSelected() method that returns true/false depending on whether the box is checked or not.

```java
public class CheckBoxExample {
    private static class ButtonActionListener implements ActionListener {
        private JCheckBox checkbox;

        public ButtonActionListener(JCheckBox checkbox) {
            this.checkbox = checkbox;
        }

        public void actionPerformed(ActionEvent arg0) {
            if (this.checkbox.isSelected()) {
                System.out.println("The checkbox is checked");
            }
            else {
                System.out.println("The checkbox is not checked");
            }
        }
    }

    public static void main(String[] args) {
        JFrame window = new JFrame();
        window.setTitle("Checkbox");
        window.setSize(350, 250);
        window.setLayout(new FlowLayout());
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JCheckBox checkbox = new JCheckBox();
        JButton button = new JButton("Click me!");
        button.addActionListener(new ButtonActionListener(checkbox));

        window.add(checkbox);
        window.add(button);
        window.setVisible(true);
    }
}
```

Create an actionListener
And assign it to the button

Pass the checkbox
to the action listener
as a constructor argument

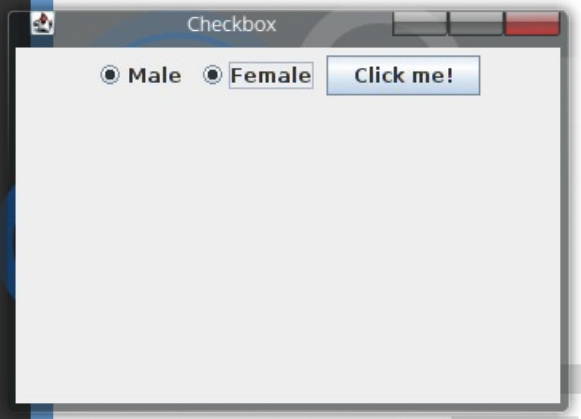


Radio Buttons

- Radio buttons are similar to checkboxes:
- They can be checked or unchecked
- The difference between radio buttons and checkboxes is that radio buttons appear in *groups*
- In each group, only one can be selected at a time
- Swing provides a **JRadioButton** class and a **ButtonGroup** class
- The ButtonGroup class is used to keep track of which buttons belong to which group

```
public static void main(String[] args) {
        JFrame window = new JFrame();
        window.setTitle("Checkbox");
        window.setSize(350, 250);
        window.setLayout(new FlowLayout());
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


        JRadioButton male = new JRadioButton("Male");
        JRadioButton female = new JRadioButton("Female");

        window.add(male);
        window.add(female);


        JButton button = new JButton("Click me!");


        window.add(button);
        window.setVisible(true);
    }
```

Because there is no ButtonGroup it's possible to check both radio buttons at the same time!



ButtonGroups

- By using a button group you can make it so only one button in the group can be selected at any time

```
public static void main(String[] args) {
    JFrame window = new JFrame();
    window.setTitle("Checkbox");
    window.setSize(350, 250);
    window.setLayout(new FlowLayout());
    window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


    JRadioButton male = new JRadioButton("Male");
    JRadioButton female = new JRadioButton("Female");

    window.add(male);
    window.add(female);

    ButtonGroup group = new ButtonGroup();
    group.add(male);
    group.add(female);

    JButton button = new JButton("Click me!");


    window.add(button);
    window.setVisible(true);
}
```

Note that the radio buttons are added to the group and the window

The ButtonGroup is not A visible element and does not get added to the window

Radio buttons and action listeners

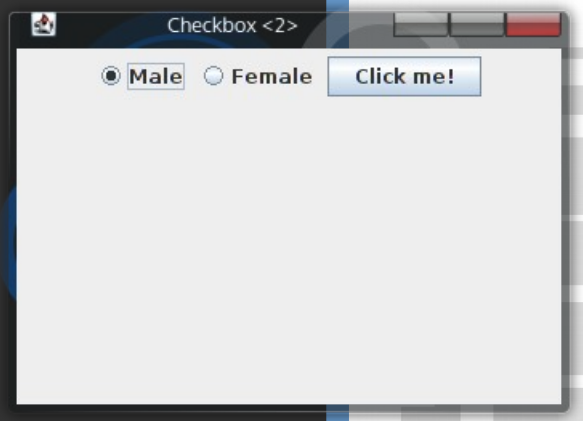
- Because a **ButtonGroup** contains all the elements in a group, it's possible to loop through the RadioButtons it contains to find out which is selected:
- To do this, import java.util.Enumeration
- And use the following code:

```
for (Enumeration<AbstractButton> buttons = group.getElements();
             buttons.hasMoreElements();) {
        AbstractButton button = buttons.nextElement();

        if (button.isSelected()) {
            System.out.println(button.getText());
        }
    }
```

```
private static class ButtonActionListener implements ActionListener {
    private ButtonGroup group;

    public ButtonActionListener(ButtonGroup group) {
        this.group = group;
    }

    public void actionPerformed(ActionEvent arg0) {
        for (Enumeration<AbstractButton> buttons = group.getElements();
            buttons.hasMoreElements();) {
                AbstractButton button = buttons.nextElement();

              if (button.isSelected()) {
                    System.out.println(button.getText());
          }
        }

    }
}
```

Use a ButtonGroup
In the constructor for the
ActionListener

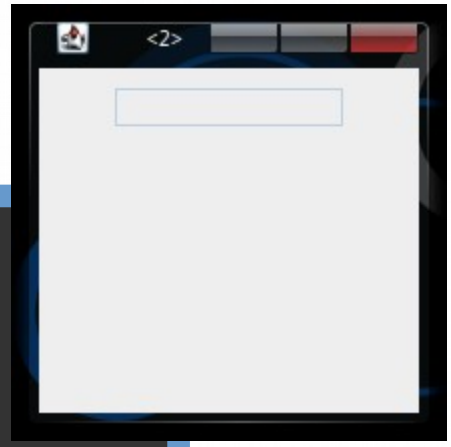
```java
public class RadioExample2 {
     private static class ButtonActionListener implements ActionListener {
          private ButtonGroup group;

          public ButtonActionListener(ButtonGroup group) {
               this.group = group;
          }

          public void actionPerformed(ActionEvent arg0) {
               for (Enumeration<AbstractButton> buttons = group.getElements();
                buttons.hasMoreElements();) {
                 AbstractButton button = buttons.nextElement();

                 if (button.isSelected())  System.out.println(button.getText());
             }
          }
     }

     public static void main(String[] args) {
          JFrame window = new JFrame();
          window.setTitle("Checkbox");
          window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

          JRadioButton male = new JRadioButton("Male");
          JRadioButton female = new JRadioButton("Female");
          window.add(male);
          window.add(female);

          ButtonGroup group = new ButtonGroup();
          group.add(male);
          group.add(female);

          JButton button = new JButton("Click me!");
          button.addActionListener(new ButtonActionListener(group));

          window.add(button);
          window.setVisible(true);
     }
}
```



Read-Only Text Fields

- It's often useful to make a text-field which is only editable after something else has happened
- Or make it uneditable after a particular action.
- To set a JtextField as read-only you can use the *setEditable(boolean);* method. Pass true/fasle to set the text box as read-only or editable.

Read-only text fields



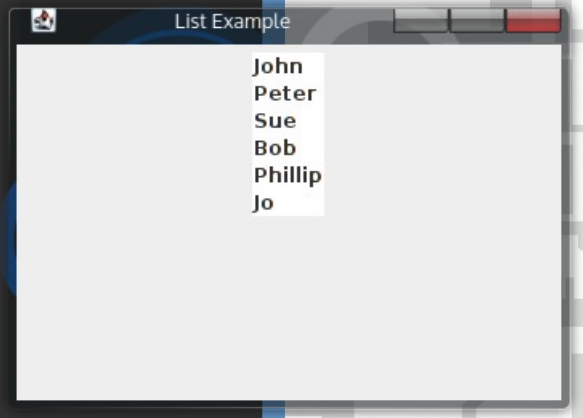
```
JFrame window = new JFrame();
window.setSize(350, 250);
window.setLayout(new FlowLayout());
window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

JTextField text = new JTextField(10);
text.setEditable(false);
window.add(text);

window.setVisible(true);
```

setEditable to false

Lists

- A *list* is a component that displays a list of items and allows the user to select items from the list.
- The **JList** component is used for creating lists.
- When an instance of the **JList** class is created, an array of Strings is passed into the constructor.

```
JFrame window = new JFrame();
window.setTitle("List Example");
window.setSize(350, 250);
window.setLayout(new FlowLayout());
window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


String[] names = {"John", "Peter", "Sue", "Bob", "Phillip", "Jo"};
JList list = new JList(names);


window.add(list);


window.setVisible(true);
```



List Selection modes

- A list is an interactive component. You can select one or more entries depending on the Selection Mode:
  - Single Selection Mode: Only one item can be selected at a time
  - Single Interval Selection Mode: Multiple items can be selected but they must be continuous, e.g from and to
  - Multiple Interval Selection Mode: In this mode, multiple selections can be made
  - To select multiple items you have to hold the Ctrl key.

List Selection Modes

Single selection mode allows only one item to be selected at a time.

Multiple interval selection mode allows multiple items to be selected with no restrictions.

Single interval selection mode allows a single interval of contiguous items to be selected.

List Selection Modes

- You can change a Jlist component's selection mode with the **setSelectionMode()** method
- The method accepts a constant as an argument:
  - ListSelectionModel.SINGLE_SELCTION
  - ListSelectionModel.SINGLE_INTERVAL_SELECTION
  - ListSelectionModel.MULTIPLE_INTERVAL_SELECTION

List Selection Modes

```
JFrame window = new JFrame();
window.setTitle("List Example");
window.setSize(350, 250);
window.setLayout(new FlowLayout());
window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


String[] names = {"John", "Peter", "Sue", "Bob", "Phillip", "Jo"};
JList list = new JList(names);


window.add(list);


window.setVisible(true);
```

List Events

- When an item in a Jlist object is selected it generates a *list selection event*
- This event is handled by an instance of a ***ListSelectionListener*** class it must implement the ***ListSelectionListener*** interface.
- It must have a method named ***valueChanged.*** This method must take an argument of the ListSelectionEvent type
- Use the ***addListSelectionListener()*** method of the ***JList*** class to register the listener with the object

List Events

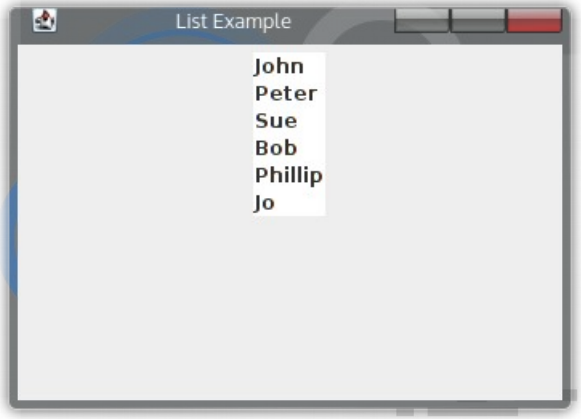
- You can use:
  - getSelectedIndex() to get the integer index of the selection
  - getSelectedValue() to return the value of the selected object in the list
    - You must cast the return value to a string order to use it correctly

List Events

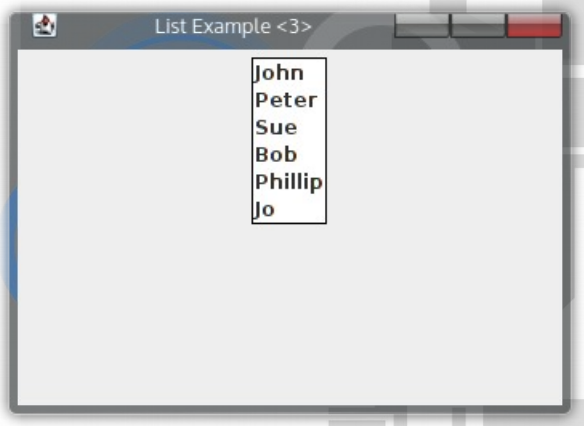
```
public static class MySelectionListener implements ListSelectionListener
    private JList list;

    public MySelectionListener(JList list) {
        this.list = list;
    }
    public void valueChanged(ListSelectionEvent e) {
                System.out.println(list.getSelectedValue());
    }
}

public static void main(String[] args) {
    JFrame window = new JFrame();
    window.setTitle("List Example");
    window.setSize(350, 250);
    window.setLayout(new FlowLayout());
    window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


    String[] names = {"John", "Peter", "Sue", "Bob", "Phillip", "Jo"};
    JList list = new JList(names);
    list.addListSelectionListener(new MySelectionListener(list));
    list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

    window.add(list);
    window.setVisible(true);
}
```

Bordered Lists

- By Default, a list looks quite ugly and gets a little lost on the background
- To make it stand out you can add
- A border using




```
list.setBorder(BorderFactory.createLineBorder(Color.BLACK));
```



Adding a scroll bar to a list

- By default, a list component is large enough to display all of the items it contains.
- Sometimes a list component contains too many items to be displayed at once.
  - E.g. displaying a list of years
- Most GUI applications display a scroll bar on list components that contain a large number of items.

- List components do not automatically display a scroll bar.

Adding a Scroll Bar To a List

- To display a scroll bar on a list component, follow these general steps.
  - 1. Set the number of visible rows for the list component.
  - 2. Create a scroll pane object and add the list component to it.
  - 3. Add the scroll pane object to any other containers, such as panels.

```
JFrame window = new JFrame();
window.setTitle("List Example");
window.setSize(350, 250);
window.setLayout(new FlowLayout());
window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


ArrayList<String> years = new ArrayList<String>();
for (int i = 1950; i <= 2015; i++) {
    years.add(Integer.toString(i));
}
JList list = new JList(years.toArray());
list.setVisibleRowCount(5);
JScrollPane scrollPane = new JScrollPane(list);

window.add(scrollPane);


window.setVisible(true);
```

Create an Array of years 1950 to 2015 and add them to the list

Set the number of visible rows on the list

Add the list to a JScrollPane

Add the scroll pane to the frame

Replacing the items in a list

- The ***setListData*** method allows adding of items to an existing JList component.
- list.setListData({“new”, “list”, “items”});
- This replaces any elements currently displayed in the list.

```
String[] names = {"John", "Peter", "Sue", "Bob", "Phillip", "Jo"};
JComboBox nameBox = new JComboBox(names);
```

Combo Boxes

- A combo box presents a drop-down list of items that the user may select from.
- The ***JComboBox*** class is used to create a combo box.
- Pass an array of strings that are to be displayed as the items in the drop-down list to the constructor.
- A combo box only allows selection

  of one item at a time. Only the selection

  is displayed at first


```
String[] names = {"John", "Peter", "Sue", "Bob", "Phillip", "Jo"};
JComboBox nameBox = new JComboBox(names);
```

Combo Boxes

- The button displays the item that is currently selected.
- The first item in the list is automatically selected when the combo box is displayed.
- When the user clicks on the button, the drop-down list appears and the user may select another item.

Combo box events

- When an item in a JComboBox object is selected, it generates an action event.
- Handle action events with an action event listener class, which must have an actionPerformed method.
- When the user selects an item in a combo box, the combo box executes its action event listener's actionPerformed method, passing an ActionEvent object as an argument.

Combo Boxes

- To read the currently selected value from a combo box you can use the methods:
  - **getSelectedItem()** to retrieve the text of the selected entry
    - Note: it returns an object rather than a string so should be cast to a string!
  - **getSelectedIndex()** to retrieve the index of the selected entry

```
    private static class MyActionListener implements ActionListener {
        private JComboBox box;

        public MyActionListener(JComboBox box) {
            this.box = box;

        }

        public void actionPerformed(ActionEvent arg0) {
            System.out.println(box.getSelectedItem());
        }
    }

    public static void main(String[] args) {
        JFrame window = new JFrame();
        window.setTitle("List Example");
        window.setSize(350, 250);
        window.setLayout(new FlowLayout());
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


        String[] names = {"John", "Peter", "Sue", "Bob", "Phillip", "Jo"};
        JComboBox nameBox = new JComboBox(names);
        nameBox.addActionListener(new MyActionListener(nameBox));
        window.add(nameBox);

        window.setVisible(true);
    }
```

Editable Combo Boxes

- There are two types of combo boxes:
  - uneditable – allows the user to only select items from its list.
  - editable – combines a text field and a list.
    - It allows the selection of items from the list
    - And allows the user to type input into the text field
- The setEditableMethod sets the edit mode for the component

```
JComboBox nameBox = new JComboBox(names);
nameBox.setEditable(true);
```

Editable Combo Boxes

- The user is not restricted to the values that appear in the list, and may type any input into the text field.
- As such, Editable Combo Boxes cannot be used to validate user input.

Displaying images in labels and buttons

- Labels can display text, an image, or both.
- To display an image, create an instance of the ImageIcon class, which reads the image file.
- The constructor accepts the name of an image file.
- The supported file types are JPEG, GIF, and PNG.
- The name can also contain path information.

```
ImageIcon image = new ImageIcon("Smiley.gif");
//or
ImageIcon image = new ImageIcon("C:\\Chapter 12\\Images\\Smiley.gif");
```

Displaying images in labels and buttons

- To display the image in a label pass the ImageIcon into the constructor of the JLabel
- Alternatively, you can set both image and text on a label, by passing text as the constructor argument and then call the ***setIcon()*** method with the path of the image

```
ImageIcon image = new ImageIcon("Smiley.gif");
JLabel label = new JLabel(image);
//or
JLabel label = new JLabel("Have a nice day!");
label.setIcon(image);
```

Images in labels


```
ImageIcon image = new ImageIcon("Smiley.gif");
JLabel label = new JLabel(image);
```


```
ImageIcon image = new ImageIcon("smiley.gif");
JLabel label = new JLabel("Have a nice day!");
label.setIcon(image);
```

Images in labels

- Images will be displayed at their *natural* width/height in pixels.
- If an image is 100x100 pixels the label will automatically be sized to show the entire image

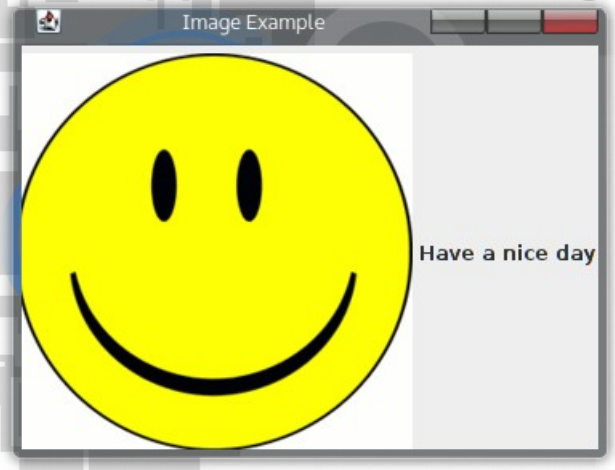
Images in Labels

- Text is displayed to the right of images by default
- Text Alignment can be modified by passing one of the following to the method

  label.setHorizontalTextPosition();

  - SwingConstants.LEFT
  - SwingContants.RIGHT
  - SwingConstants.CENTER

Images in labels and buttons




```
ImageIcon image = new ImageIcon("smiley.gif");
JLabel label = new JLabel("Have a nice day!");
label.setIcon(image);
label.setHorizontalTextPosition(SwingConstants.RIGHT);
```

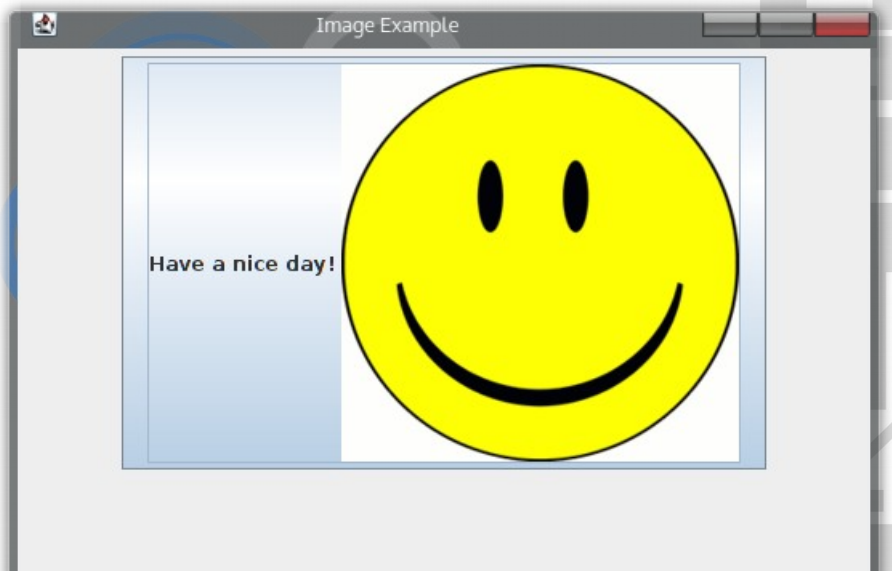

```
ImageIcon image = new ImageIcon("smiley.gif");
JLabel label = new JLabel("Have a nice day!");
label.setIcon(image);
label.setHorizontalTextPosition(SwingConstants.CENTER);
```


```
ImageIcon image = new ImageIcon("smiley.gif");
JLabel label = new JLabel("Have a nice day!");
label.setIcon(image);
label.setHorizontalTextPosition(SwingConstants.LEFT);
```

Images in Buttons

- Creating a button with an image is the same as creating a label with an image:

```
ImageIcon image = new ImageIcon("smiley.gif");
JButton button = new JButton("Have a nice day!");
button.setIcon(image);
button.setHorizontalTextPosition(SwingConstants.LEFT);
```



Hot Keys

- A hot key is a keyboard key that you press in combination with the Alt key to quickly access a component.
- These are sometimes referred to as hot keys.
- A hot key is assigned to a component through the component's ***setMnemonic()*** method
- The argument passed to the method is an integer code that represents the key you wish to assign.

Hot Keys

- The key codes are predefined constants in the KeyEvent class (java.awt.event package).
- These constants take the form:
- KeyEvent.VK_*x*, where *x* is a key on the keyboard.
- The letters VK in the constants stand for “virtual key”.
- To assign the A key as a mnemonic, use KeyEvent.VK_A.
- Example:
- **`JButton exitButton = new JButton("Exit");`**
- **`exitButton.setMnemonic(KeyEvent.VK_X);`**
- This would enable pressing Alt+X to trigger the button press.

Hot Keys

- If the letter is in the component's text, the first occurrence of that letter will appear underlined.
- If the letter does not appear in the component's text, then no letter will appear underlined.

Hot Keys

```
JButton exitButton = new JButton("Exit");
exitButton.setMnemonic(KeyEvent.VK_X);
```

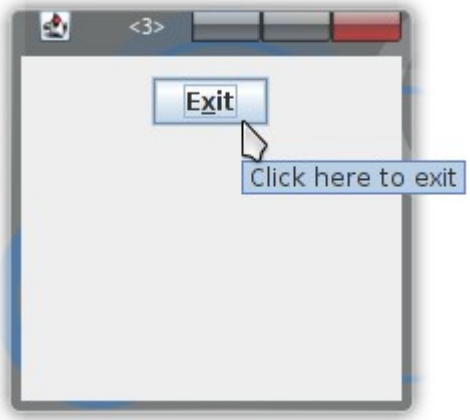
Tool Tips

- A *tool tip* is text that is displayed in a small box when the mouse is held over a component for a short time.
- The box usually gives a short description of what the component does.
- Most GUI applications use tool tips as concise help to the user.

Tool Tips

- Assign a tool tip to a component with the ***setToolTipText()*** method.

```
JButton exitButton = new JButton("Exit");
exitButton.setMnemonic(KeyEvent.VK_X);
exitButton.setToolTipText("Click here to exit");
```



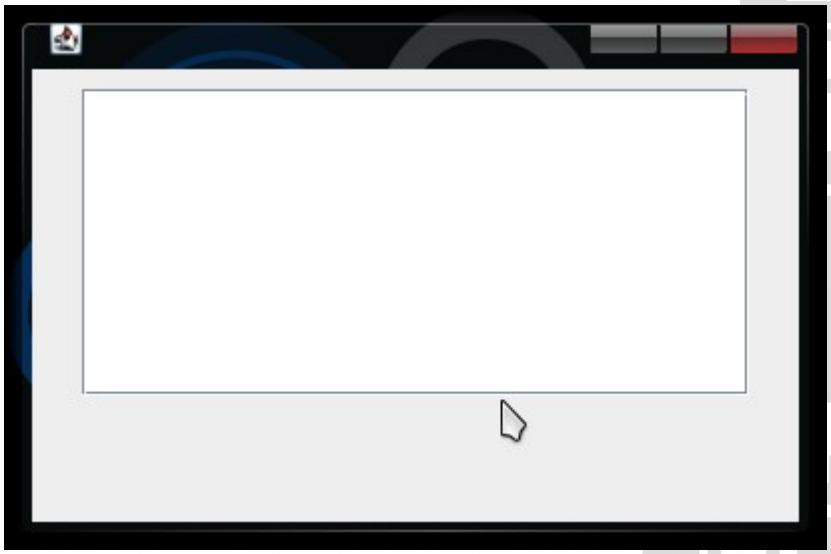
Text Areas

- The ***JTextField*** class is used to create text fields.
- A text field is a component that allows the user to enter a single line of text.
- A text area is like a text field that can accept multiple lines of input.
- You use the ***JTextArea*** class to create a text area.
- The general format of two of the class's constructors:
- **`JTextArea(int rows, int columns)`**
- **`JTextArea(String text, int rows, int columns)`**

Text Areas

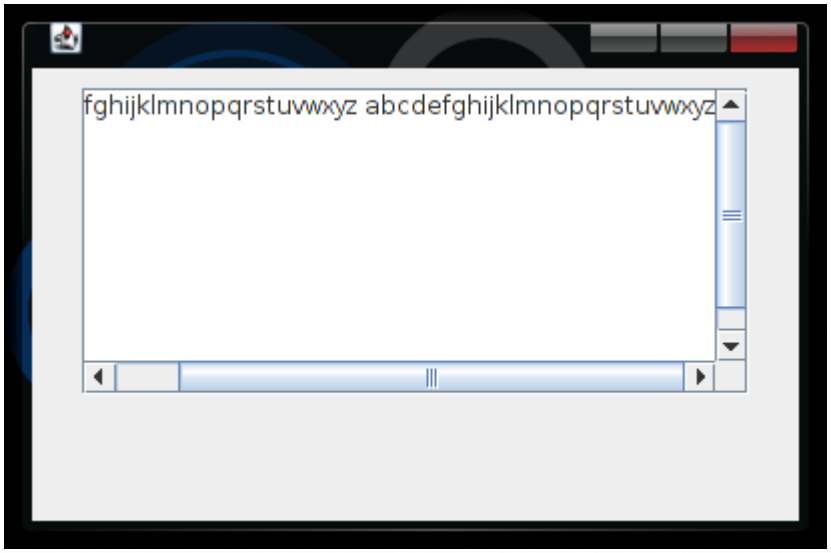
- Like Lists, Text Areas do not automatically display scroll text
- You must add a text area to a scroll pane

```
JTextArea textArea = new JTextArea(10, 30);
JScrollPane scrollPane = new JScrollPane(textArea);
window.add(scrollPane);
```



Text Areas – Word Wrapping

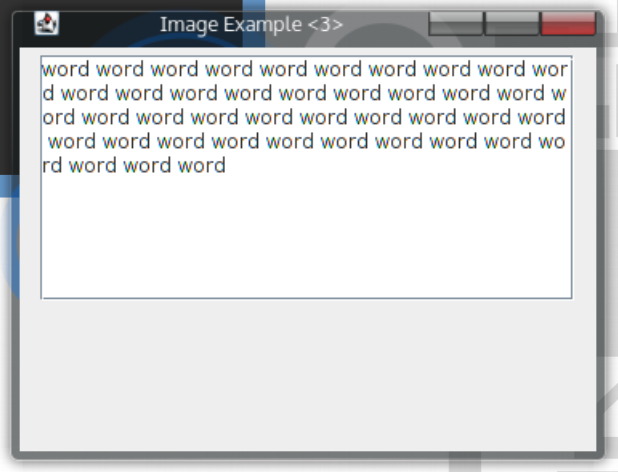
- By default, Text Areas do not perform line wrapping. This means that a horizontal scrollbar appears when the width is used up:



Text Areas – Wrapping

- You can enable wrapping to change this behaviour.
- There are two types of word wrapping:
  - Character wrapping. The text breaks when the number of characters run out

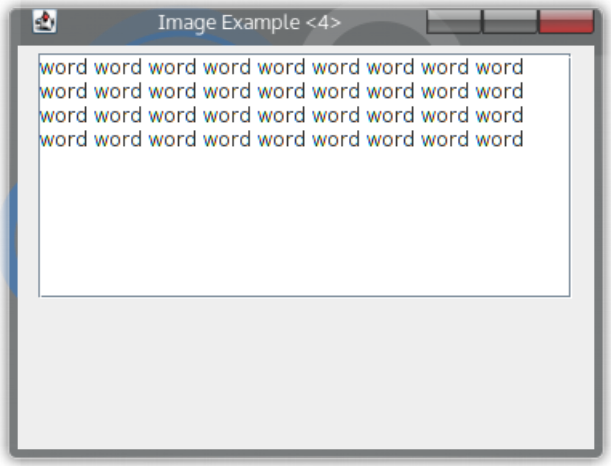
```
JTextArea textArea = new JTextArea(10, 30);
JScrollPane scrollPane = new JScrollPane(textArea);
textArea.setLineWrap(true);
window.add(scrollPane);
```



Text Areas - Wrapping

- For most programs, this isn't very user friendly
- It's better to break on words (at spaces)

```
JTextArea textArea = new JTextArea(10, 30);
JScrollPane scrollPane = new JScrollPane(textArea);
textArea.setLineWrap(true);
textArea.setWrapStyleWord(true);
window.add(scrollPane);
```

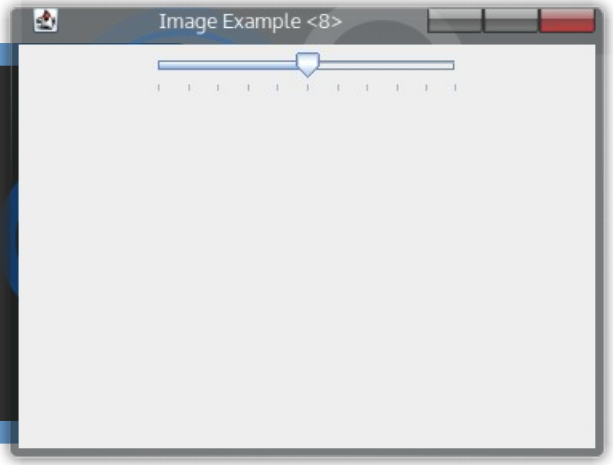


Sliders

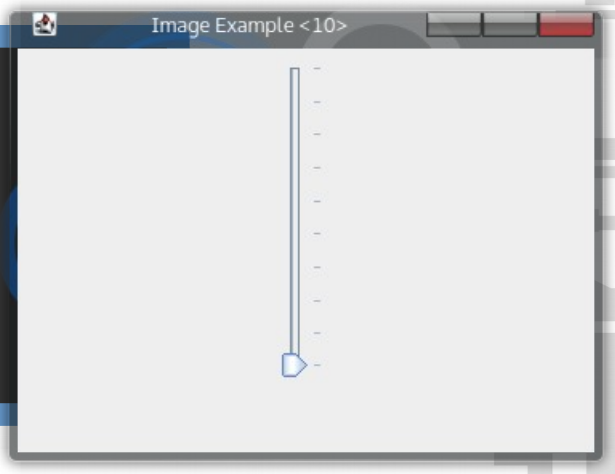
- A slider is a component that allows the user to graphically adjust a number within a range
- Sliders are created from the ***JSlider*** class
- They display a "slider button" that can be dragged along a track
- Sliders require a minimum value and a maximum value

Sliders

```
JSlider slider = new JSlider(JSlider.HORIZONTAL,
                      -100, 100, 0);
//Show the 'ticks'
slider.setPaintTicks(true);
//Show a tick every 20 intervals
slider.setMinorTickSpacing(20);
```




```
JSlider slider = new JSlider(JSlider.VERTICAL,
                      -100, 100, 0);
//Show the 'ticks'
slider.setPaintTicks(true);
//Show a tick every 20 intervals
slider.setMinorTickSpacing(20);
```



Sliders

- A slider can be assigned a **ChangeListener** that gets triggered when the slider value is changed.
- A changeListener has a StateChanged method which takes a ChangeEvent as an argument.

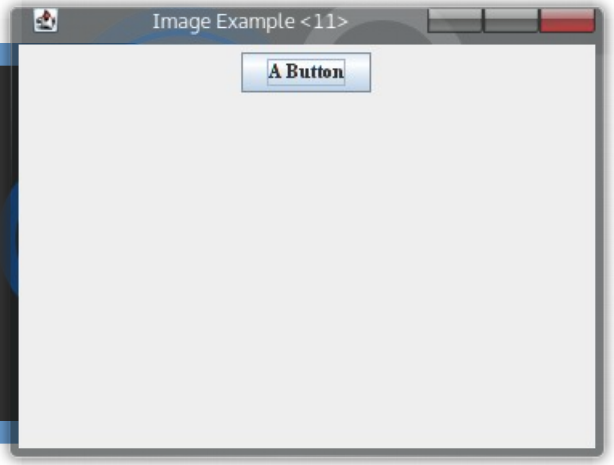
```
private static class MyChangeListener implements ChangeListener {
        private JSlider slider;

        public MyChangeListener(JSlider slider) {
            this.slider = slider;

        }

        public void stateChanged(ChangeEvent arg0) {
            System.out.println(this.slider.getValue());
        }
    }

    public static void main(String[] args) {
        JFrame window = new JFrame();
        window.setTitle("Image Example");
        window.setSize(370, 280);
        window.setLayout(new FlowLayout());
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


        JSlider slider = new JSlider(JSlider.HORIZONTAL,
                -100, 100, 0);
        slider.setPaintTicks(true);
        slider.setMinorTickSpacing(20);
        slider.addChangeListener(new MyChangeListener(slider));

        window.add(slider);

        window.setVisible(true);
    }
```

Fonts

- Most components have a *setFont()* method that allows you to change the font on the component. This uses the Font class.
- The font class constructor takes three arguments:
  - The font name
  - Font options, e.g. bold or italic
    - Use constants Font.BOLD and Font.ITALIC
  - The font size in points

Fonts

```
JButton button = new JButton("A Button");
Font font = new Font("Times New Roman", Font.BOLD, 12);
button.setFont(font);
window.add(button);
```




```
JButton button = new JButton("A Button");
window.add(button);
```