Components

Labels, buttons, text fields and text areas, check boxes, radio buttons, combo boxes, scrollable lists

Labels

- A **label** is text that is not interactive – you cannot even select it. Commonly used for prompts or to simply report information.
- Labels cannot have listeners.
- To change the text on a label during program execution, use setText().
- You can specify font/color etc.
- Swing labels are of type **JLabel**.
- Sample code:

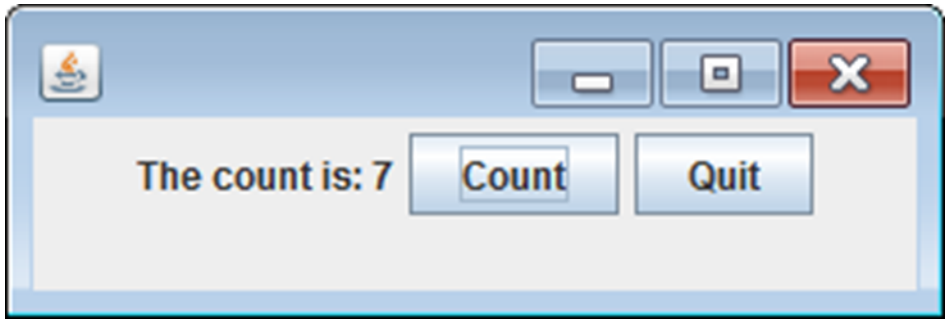
```
private static final String MESSAGE = "The count is: ";
private int count = 0;
private JLabel lblCount;

    lblCount = new JLabel(MESSAGE + count);

    lblCount.setText(MESSAGE + ++count);
```

Buttons

- **Buttons** are clickable components.
- Buttons need to register ActionListeners if they are to respond to clicks.
- Button text, color can be set.
- Buttons can be disabled.
- Swing buttons are of type **JButton**.

Button and label example

The lesson on events had a good button and label example.



- Declarations:

```
// Variables for initializing strings
private int count = 0;
private final String MESSAGE = "The count is: ";
// Components
private JLabel lblCount =
                        new JLabel(MESSAGE + count);
private JButton btnCount = new JButton("Count");
private JButton btnQuit = new JButton("Quit");
```

Button and label example (cont)

- Adding to the content pane:

```
Container c = getContentPane();
c.add(lblCount);
c.add(btnCount);
c.add(btnQuit);
```

- Registering listeners for the buttons:

```
btnCount.addActionListener(this);
btnQuit.addActionListener(this);
```

- Responding to events:

```
public void actionPerformed(ActionEvent e) {
   if (e.getSource().equals(btnQuit))
      System.exit(0);
   if (e.getSource().equals(btnCount))
       lblCount.setText(MESSAGE + ++count);
}
```

Images, fonts, colors

Buttons and labels have some characteristics in common.

- **Images and text:** You can set the text and add images when you create the button or label. For example:

```
myButton = new JButton("My message",
                                new ImageIcon("myIcon.gif"));
```

- **Fonts:** You can set the font on a button or label. For example:

```
myButton.setFont(new Font("Serif", Font.BOLD, 14));
```

- **Colors:** You can set background or foreground colors. For example:

```
myButton.setBackgroundColor(Color.RED);
```

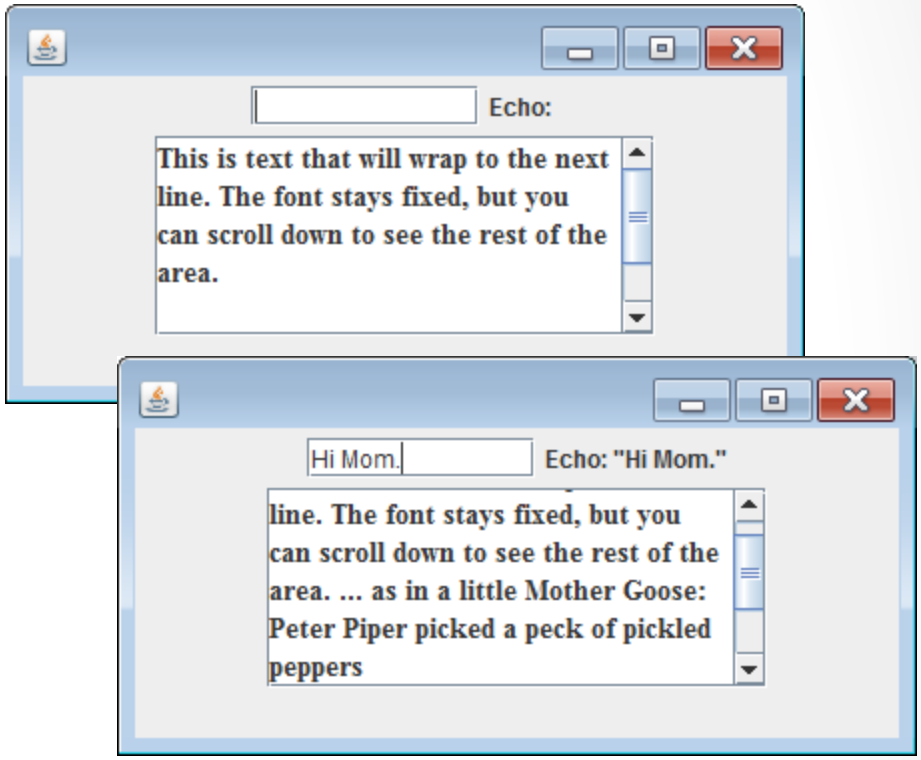
Text fields and text areas

- Text fields and text areas are components for entering free form text. Text fields and text areas can also display output information.
- Text fields and text areas can contain formatted text.
- Password fields are like text fields except the characters typed by the user are displayed as asterisks.
- Swing types are **JTextField**, **JTextArea**, and **JPasswordField**.
- JTextAreas are usually placed in JScrollPanes to enable scrolling.

Text example

This GUI has 4 widgets:

- A text field
- A label whose text varies
- A text area
- A scroll pane to hold the text area.



Text example code

The instance variables in TextDemo are initialized outside the constructor.

```
public class TextDemo extends JFrame
                      implements ActionListener{
   private JLabel lblUserInput = new JLabel("Echo: ");
   private JTextField txtUserInput = new JTextField(10);
   private JTextArea txtComment
          = new JTextArea("This is text that will wrap"
      + " to the next line. The font stays fixed, but you"
      + " can scroll down to see the rest of the area.\n");
   private JScrollPane scrollPane
         = new JScrollPane(txtComment);

    // Rest of code goes here
```

Text example code (cont)

```
// Declaration and instance variables go here
// Constructor (beginning)
public TextDemo() {
   Container c = this.getContentPane();
   c.setLayout(new FlowLayout());

   txtUserInput.addActionListener(this);

   scrollPane.setVerticalScrollBarPolicy(
           JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
   scrollPane.setPreferredSize(new Dimension(250, 100));

   txtComment.setFont(new Font("Serif", Font.BOLD, 14));
   txtComment.setLineWrap(true);
   txtComment.setWrapStyleWord(true);
```

Text example code (cont 2)

```
// Declaration and instance variables go here
// public TextDemo() {
   // First part of constructor body
   c.add(txtUserInput);
   c.add(lblUserInput);
   c.add(scrollPane);

   setDefaultCloseOperation(EXIT_ON_CLOSE);
   setSize(400,200);
   setVisible(true);
}
public void actionPerformed(ActionEvent e) {
    JTextField userEntry = (JTextField)e.getSource();
    lblUserInput.setText("Echo: \""
                         + userEntry.getText() + "\"");
}
```

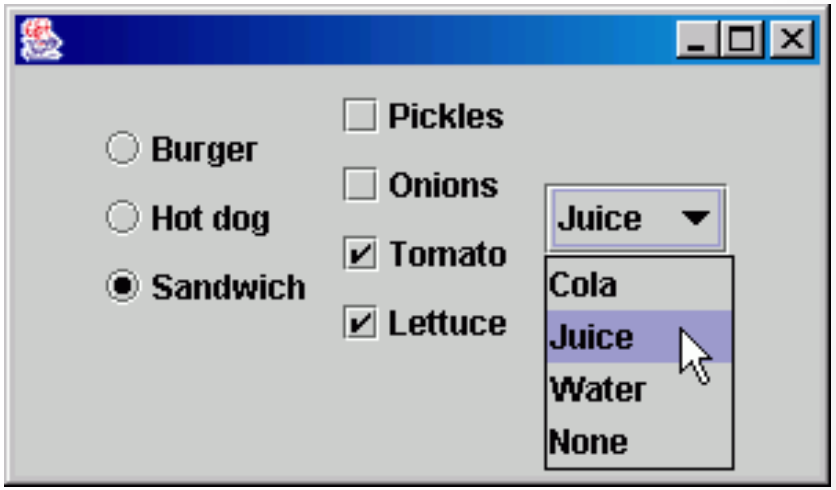
Text example discussion

You typically place text areas inside scroll panes in order to give them scroll bars. The scroll pane and text area here illustrate 4 features:

- **setVerticalScrollBarPolicy()** guarantees a vertical scrollbar column, even when you can see all of the text without scrolling.
- **setPreferredSize()** establishes a size for the scroll pane. This depends on whether the window is large enough.
- **setFont()** fixes the font style and size for the text area. The user cannot change these settings.
- **setLineWrap()** and **setWrapStyleWord()** make the text in the text area wrap at the right edge, with no breaks inside words.

Check boxes, radio buttons, combo boxes

- A **check box** toggles a check on and off when it is clicked. Check boxes register ItemListeners instead of ActionListeners. Swing type is **JCheckBox**.
- A **radio button** has a similar toggle, but it is always grouped with other radio buttons for the user to select a single one among all in the group. Swing/AWT types: **JRadioButton** and **ButtonGroup**.
- A **combo box** gives a drop-down list of choices. Swing type: **JComboBox**



Checkbox/radio button/ combo box code

Instance variable declarations.

```
public class ChoiceCheckDemo extends JFrame {
    private String[] drinks = {"Cola", "Juice","Water","None"};
    private JComboBox cmbxDrink = new JComboBox(drinks);
    private ButtonGroup grpEntree = new ButtonGroup();

    private JCheckBox ckbxPickle = new JCheckBox("Pickles");
    private JCheckBox ckbxLettuce = new JCheckBox("Lettuce");
    private JCheckBox ckbxOnion = new JCheckBox("Onions");
    private JCheckBox ckbxTomato = new JCheckBox("Tomato");
    private JRadioButton btnBurger = new JRadioButton("Burger");
    private JRadioButton btnDog = new JRadioButton("Hot dog");
    private JRadioButton btnSandwich =
                                    new JRadioButton("Sandwich");

    JPanel pnlFood = new JPanel();
    JPanel pnlCondiment = new JPanel();
```

Checkbox/radio button/ combo box code (cont)

Setting up the widgets.

```
public ChoiceCheckDemo() {

    btnBurger.setSelected(true);
    grpEntree.add(btnBurger);
    grpEntree.add(btnDog);
    grpEntree.add(btnSandwich);
    pnlFood.setLayout(new GridLayout(3,1));
    pnlFood.add(btnBurger);
    pnlFood.add(btnDog);
    pnlFood.add(btnSandwich);

    pnlCondiment.setLayout(new GridLayout(4,1));
    pnlCondiment.add(ckbxPickle);
    pnlCondiment.add(ckbxOnion);
    pnlCondiment.add(ckbxTomato);
    pnlCondiment.add(ckbxLettuce);
```

Checkbox/radio button/ combo box code (cont 2)

Finishing the constructor.

```
    // public ChoiceCheckDemo() {

        Container c = getContentPane();
        c.setLayout(new FlowLayout());


        c.add(pnlFood);
        c.add(pnlCondiment);
        c.add(cmbxDrink);

        setSize(300,175);
        setVisible(true);
    }
```

Checkbox/radio button/ combo box code - discussion

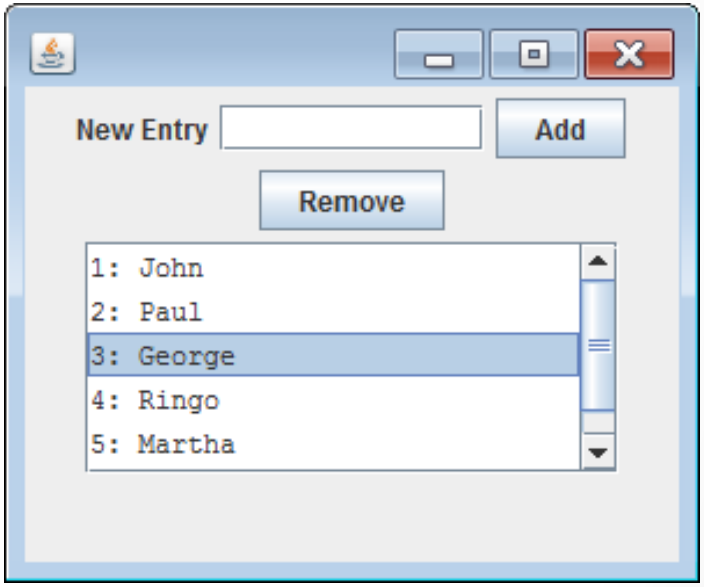
- The combo box, cmbxDrink, is initialized by an array of strings, which become the menu items on the GUI. The most recently selected item serves as the box label on the menu.
- Checkboxes are stand-alone components. Panels provide a convenient mechanism for grouping related checkboxes on the GUI.
- Radio buttons are created as separate components. When you group them together using a ButtonGroup, they act as a single component. In that case, exactly one of the buttons can be selected at a time.
- ***CheckBox listeners***: Combo boxes and radio buttons can register ActionListeners. Checkboxes, however, register ItemListeners. An application with a working JCheckBox should:
  - implement ItemListener
  - define the method void itemStateChanged(ItemEvent e)

Scrollable lists

- A **scrollable list** is a list with scrollbars and in which you can select items. The scrollable list items are strings that reflect the underlying actual data. Three types are required to create a scrollable list:
  - **DefaultListModel** -- a list for the GUI containing the items in the actual list. DefaultListModel elements are Objects. With the list, you can add and remove elements:
    - void addElement(Object)
    - void remove(int)
  - **JList** -- holds the DefaultListModel. JLists are constructed from the DefaultListModel. Two important JList methods are:
    - void setFont(Font) -- sets the font characteristics.
    - int getSelectedIndex() – gets the smallest selected cell index; *the selection* when only a single item is selected in the list.
  - **JScrollPane** – to display the JList in an area with vertical and horizontal scroll bars.

Scrollable list example

- This example has:
  - A label
  - A text field for text entry.
  - Two buttons:
    - Add the entry in the text field.
    - Remove the selected item from the list.
  - Scrollable list, which requires:
    - Backend list for data
    - Default list model
    - GUI list for display
    - Scroll pane to hold display



Scrollable list code

4 variables for the list, including the backend array for actual data.

```
public class ScrollableListDemo extends JFrame
                            implements ActionListener {

  // Instance variables
  private ArrayList<String> backendArray
                             = new ArrayList<String>();
  private DefaultListModel model = new DefaultListModel();
  private JList jList = new JList(model);
  private JScrollPane scrollPane = new JScrollPane(jList);
  private JLabel lblName = new JLabel("New Entry");
  private JTextField txtName = new JTextField(10);
  private JButton btnAdd = new JButton("Add");
  private JButton btnRemove = new JButton("Remove");
```

Scrollable list code (cont)

```
public ScrollableListDemo() {
    Container c = this.getContentPane();
    fillModel(); // Private method call
    jList.setFont(new Font("Courier", Font.PLAIN, 12));
    jList.setSelectionMode(
                        ListSelectionModel.SINGLE_SELECTION);
    scrollPane.setPreferredSize(new Dimension(230,100));
    btnAdd.addActionListener(this);
    btnRemove.addActionListener(this);

    c.setLayout(new FlowLayout());
    c.add(lblName); c.add(txtName); c.add(btnAdd);
    c.add(btnRemove); c.add(scrollPane);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setSize(300,250); setVisible(true);
  }
```

Scrollable list code (cont 2)

```
private void fillModel(){
   // model is DefaultListModel
   model.clear();
   for (int k = 0; k < backendArray.size(); k++)
      model.addElement((k+1) + ": " + backendArray.get(k));
 }

public void actionPerformed(ActionEvent e) {
   if (e.getSource().equals(btnAdd)) {
       backendArray.add(txtName.getText());
       txtName.setText("");
   }
   if (e.getSource().equals(btnRemove)) {
       int index = jList.getSelectedIndex();
       if (index >= 0)
           backendArray.remove(index);
   }
   fillModel();
}
```

Scrollable list code - discussion

- The model is tightly coupled with the backend list. *Whenever the backend list changes, the model is cleared and filled anew with the list.* That is the purpose of fillModel().
- The JList method getSelectedIndex() returns the index of the list item selected. If nothing is selected, the return value is -1.
- The ScrollPane for scrollable lists is the same as that for text areas.
- ***Default list model***: GUI lists typically reflect real data. That requires two lists, separating the backend (the "model" part of MVC architecture) from what the user sees (the "view" part of MVC architecture).

Reference

The Java™ Tutorials: https://docs.oracle.com/javase/tutorial/
See the section on Creating Graphical User Interfaces.