Swing Never Ends

There are many components to Swing that we won't have time to cover in class. However, once you get the hang of some components it is pretty easy to learn the others, as they use a similar model. See the Oracle documentation for examples on the full suite of components:

https://docs.oracle.com/javase/tutorial/uiswing/components/componentlist.html

Here we cover a few other common components.

Check Boxes and Radio Buttons

The JCheckBox and JRadioButton classes are used to create checkboxes (on or off) and radio buttons (only one can be selected from a group). They are implemented pretty much like regular JButtons. Put the buttons into a ButtonGroup to set the radio buttons that should be grouped together. The following example demonstrates a few buttons, groups, and a button to show how to programmatically change the settings.

```
public class GUI_Demo extends JFrame implements ActionListener
{
    public GUI_Demo()
    {
        super();
        setSize(810,640);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLayout(new FlowLayout());

        ButtonGroup group1 = new ButtonGroup();
        JRadioButton rb1 = new JRadioButton("Pepperoni");
        JRadioButton rb2 = new JRadioButton("Cheese");
        rb1.addActionListener(this);
        rb2.addActionListener(this);
        group1.add(rb1);
        group1.add(rb2);
        this.add(rb1);
        this.add(rb2);

        ButtonGroup group2 = new ButtonGroup();
        JRadioButton rb3 = new JRadioButton("Large");
        JRadioButton rb4 = new JRadioButton("Extra Large");
        rb3.addActionListener(this);
        rb4.addActionListener(this);
        group2.add(rb3);
        group2.add(rb4);
        this.add(rb3);
        this.add(rb4);

        JCheckBox check = new JCheckBox("Delivery");
        check.addActionListener(this);
```

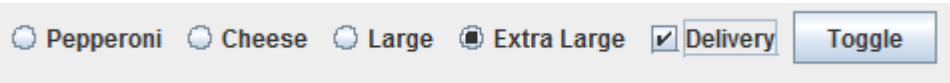
```
        this.add(check);

        // Example to get and set buttons
        JButton toggle = new JButton("Toggle");
        toggle.addActionListener(new ActionListener()
        {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (rb4.isSelected())
                    rb3.setSelected(true);
                else
                    rb4.setSelected(true);
                if (check.isSelected())
                    check.setSelected(false);
                else
                    check.setSelected(true);
            }
        }
        );
        this.add(toggle);

        setVisible(true);
    }

    public static void main(String[] args) {
        GUI_Demo app = new GUI_Demo();
    }

    @Override
    public void actionPerformed(ActionEvent e) {
         System.out.println(e.getActionCommand());
    }
}
```



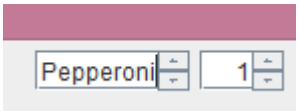
Spinners

A spinner is a list that shows one (or a couple) items at a time and the user can go up and down to pick an item. A list displays multiple items at a time and the user clicks on one or more of them.

Here is an example of two spinners, one that uses a list, and the other that uses a numeric model. There are other spinner models, such as one for selecting dates.

```
        // Spinner for pizzas
        String[] pizzaTypes = {"Pepperoni", "Combo", "Hawaiian"};
        SpinnerListModel pizzaModel = new SpinnerListModel(pizzaTypes);
        JSpinner spinner = new JSpinner(pizzaModel);
        this.add(spinner);
```

```
// Initial value of 1, min of 0, max of 20, step of 1
SpinnerModel quantityModel = new
          SpinnerNumberModel(1, 0, 20, 1);
JSpinner spinnerQ = new JSpinner(quantityModel);
this.add(spinnerQ);
```



To detect if the spinner changes, we have to implement a ChangeListener. We do this basically the same way we made an ActionListener. A simple approach is to put the ChangeListener on the same class, and then implement the StateChanged method:

```
public class GUI_Demo extends JFrame implements ActionListener,
ChangeListener


spinner.addChangeListener(this);
spinnerQ.addChangeListener(this);


    public void stateChanged(ChangeEvent e)
    {
        JSpinner spinner = (JSpinner) e.getSource();
        SpinnerModel model = spinner.getModel();
        if (model instanceof SpinnerNumberModel)
        {
            System.out.println((int) model.getValue());
        }
        else if (model instanceof SpinnerListModel)
        {
            System.out.println(model.getValue().toString());
        }
    }
```

There is another ChangeListener in javafx.beans.value, you don't want that one, you want the one from javax.swing.event.ChangeListener.

Lists

A list is like a spinner but we have a few more properties to configure. The following creates a list of pizza toppings:

```
String[] pizzas = {"Pepperoni", "Sausage", "Olives", "Pineapple",
    "Bacon", "Tomatoes", "Onion", "Garlic"};
JList<String> pizzaList = new JList(pizzas);
// Choose SINGLE_SELECTION, SINGLE_INTERVAL_SELECTION, OR
// MULTIPLE_INTERVAL_SELECTION

pizzaList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
// The line below makes the list display the max number of
// items possible
// that fit in the space on screen
pizzaList.setVisibleRowCount(-1);
this.add(pizzaList);
```

Note that if we want to fix the number of rows, then set the visible row count to a positive integer and put the list inside a JScrollPane.

```
// If you put a number in here, use a Scroll Pane
pizzaList.setVisibleRowCount(4);
JScrollPane sp = new JScrollPane(pizzaList);
this.add(sp);
```

Finally, to get the items that are changed, add a ListSelectionListener. This interface requires you to implement the valueChanged method.

```
public class GUI_Demo extends JFrame implements ActionListener,
ListSelectionListener


pizzaList.addListSelectionListener(this);


    @Override
    public void valueChanged(ListSelectionEvent e)
    {
      if (!e.getValueIsAdjusting())
      {
        for (String s : pizzaList.getSelectedValuesList())
        {
            System.out.println(s);
        }
      }
    }
```

The getValueIsAdjusting method checks if we have completed the selection event (i.e. mouse up). If you leave this out then you'll likely get multiple outputs for when you click and release the mouse.

If we want to programmatically add or remove items from the list then we should create a ListModel:

```
DefaultListModel listModel = new DefaultListModel();
listModel.addElement("Pepperoni");
listModel.addElement("Ham");
listModel.addElement("Olives");
listModel.addElement("Pineapple");
JList pizzaList = new JList(listModel);

pizzaList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
pizzaList.setVisibleRowCount(-1);
this.add(pizzaList);



// Example to remove or add item
JButton listEdit = new JButton("Edit list");
listEdit.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e) {
        listModel.addElement("Onions");
        //listModel.removeElementAt(0);
    }
}
);
this.add(listEdit);
```

Sliders

The JSlider class implements a slider that can be either horizontal or vertical. To get events, use a ChangeListener just like a JSpinner.

```
public class GUI_Demo extends JFrame implements ActionListener,
ChangeListener
{
    public GUI_Demo()
    {
        super();
        setSize(810,640);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLayout(new FlowLayout());

        // Horizontal slider with min=0, max=100, initial = 10
        JSlider slider = new JSlider(JSlider.HORIZONTAL, 0, 100, 10);
        slider.addChangeListener(this);
        slider.setMajorTickSpacing(20);
        slider.setPaintTicks(true);
        // Add some labels
        Hashtable labelTable = new Hashtable(); // java.util.Hashtable
```

```
        labelTable.put(0, new JLabel("Stop"));
        labelTable.put(50, new JLabel("Medium"));
        labelTable.put(100, new JLabel("Fast"));
        slider.setLabelTable(labelTable);
        slider.setPaintLabels(true);
        this.add(slider);
        setVisible(true);
    }

    @Override
    public void stateChanged(ChangeEvent e) {
        JSlider source = (JSlider) e.getSource();
        if (!source.getValueIsAdjusting())
            System.out.println(source.getValue());

    }
```

Tabbed Panes

Lastly, we briefly show how to create tabbed panes. It is fairly simple, we create a JTabbedPane object and add items (e.g. panels) to each one.

```
public GUI_Demo()
{
    super();
    setSize(810,640);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setLayout(new BorderLayout());

    JTabbedPane tabbedPane = new JTabbedPane();
    // set up pane11 and add it to JTabbedPane
    JLabel label1 = new JLabel( "panel one");
    JPanel panel1 = new JPanel(); // create first panel
    panel1.add( label1 ); // add label to panel
    tabbedPane.addTab( "Tab One", null, panel1, "First Panel" );

    // set up panel2 and add it to JTabbedPane
    JLabel label2 = new JLabel( "panel two");
    JPanel panel2 = new JPanel(); // create second panel
    panel2.setBackground( Color.YELLOW ); // background to yellow
    panel2.add( label2 ); // add label to panel
    tabbedPane.addTab( "Tab Two", null, panel2, "Second Panel" );

    // set up panel3 and add it to JTabbedPane
    JLabel label3 = new JLabel( "panel three" );
    JPanel panel3 = new JPanel(); // create third panel
    panel3.setLayout( new BorderLayout() ); // use borderlayout
    panel3.add( new JButton( "North" ), BorderLayout.NORTH );
    panel3.add( new JButton( "West" ), BorderLayout.WEST );
    panel3.add( new JButton( "East" ), BorderLayout.EAST );
    panel3.add( new JButton( "South" ), BorderLayout.SOUTH );
    panel3.add( label3, BorderLayout.CENTER );
```

```
        tabbedPane.addTab( "Tab Three", null, panel3, "Third Panel" );
        this.add(tabbedPane, BorderLayout.CENTER);
        setVisible(true);
    }
```