1.00 Lecture 19

Swing Event Model, Continued
Layout Managers

Reading for next time: Big Java: review Swing sections

Event Listeners

- **You may select any object, as long as it implements ActionListener, to be the event listener. Either:**
  - **Add an actionPerformed method to GUI element class**
    - **Often make the containing panel listen to its buttons, etc., as in both examples in class so far**
  - **Create new class as listener**
  - **Create 'inner class' as listener (covered next class)**
- **Next exercise, ComboBox, has two event sources and we must listen and distinguish between the two types of event**
  - **Example displays fonts selected by user**
  - **Font family, font size are chosen; font style is BOLD**
- **Download CFrame, ComboPanel**

CFrame

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

public class CFrame extends JFrame {

   public static void main(String[] args) {
        CFrame frame = new CFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVibible(true);
   }

   public CFrame() {
        setTitle("ComboBox Example");
        setSize(400, 200);
        ComboPanel panel = new ComboPanel();
        Container contentPane = getContentPane();
        contentPane.add(panel);
   }

}       // Why no data members in this class?
```

ComboPanel

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

public class ComboPanel extends JPanel implements ActionListener {
   private String curFamily = "Monospaced";
   private int curSize = 10;
   private JLabel showFont;
   private JComboBox comboFamily;
   private JComboBox comboSize;

   public ComboPanel() {
        String[] fontFamily = { "Monospaced", "Serif", "SansSerif" };
        String[] fontSize = { "10", "12", "14", "18", "24", "36" };
        comboFamily = new JComboBox(fontFamily);
        comboSize = new JComboBox(fontSize);
        showFont = new JLabel();
        showFont.setFont(new Font(curFamily, Font.BOLD, curSize));
        showFont.setText(curFamily + " " + Font.BOLD + " " + curSize);

        // Add necessary components to the ComboPanel using add():
        //      Add the 3 objects you just created
        // Add action listeners for the components that generate events
        //      using addActionListener()     }
```

ComboPanel, p.2

```
  public void actionPerformed(ActionEvent e) {
     if (e.getSource().equals(comboFamily))
         ?? = (String) comboFamily.getSelectedItem();
     else
         ?? = Integer.parseInt((String) comboSize.getSelectedItem());
     showFont.setFont(new Font(curFamily, Font.BOLD, curSize));
     showFont.setText(curFamily + " " + Font.BOLD + " " + curSize);
   }
}
       // Complete actionPerformed()

       // Combo boxes return Objects from getSelectedItem()
       // so you need to cast/convert them to Strings and then
       // the desired data type/object
```

The Java Event Model, Again

- **How do GUIs *interact* with users? How do applications recognize when the user has done something?**
- **In Java this depends on 3 related concepts:**
  - **events: objects that represent a user action with the system**
  - **event sources: in Swing, these are components that can recognize user action, like a button or an editable text field**
  - **event listeners: objects that can respond when an event occurs**

Events

- Events are instances of simple classes (tin other words, they are objects) that supply information about what happened.
- For example, instances of `MouseEvent` have `getX()` and `getY()` methods that will tell you where the mouse event (e.g., mouse press) occurred.
- All event listener methods take an event as an argument.

Event Sources, Listeners

- Event sources
  - Event sources generate events
  - The ones you will be most interested are subclasses of `JComponents` like `JButtons` and `JPanels`
  - You find out the kind of events they can generate by reading the Javadoc
- Event listeners
  - An object becomes an event listener when its class implements an event listener interface.
  - The event listener gets called when the event occurs if we register the event listener with the event source

How do I Set Up to Receive an Event?

1. **Figure out what type of event you are interested in and what component it comes from.**
2. **Decide which object is going to *handle* (act on) the event.**
3. **Determine the correct listener interface for the type of event you are interested in.**
4. **Write the appropriate listener method(s) for the class of the handler object.**
5. **Use an `add`*EventType*`Listener()` method to register the listener with the event source**

Exercise

- **Mark up the next three slides:**
  - **Find where steps 1, 2, 3, 4 and 5 occur from the previous slide**
  - **Circle these steps and label them**

Exercise: Hello Application

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

public class Hello extends JFrame
  implements ActionListener
{
  private JButton button;
  private int state = 0;

  public static void main (String args[]) {
    Hello hello = new Hello();
    hello.setVisible( true );
  }
```

The Hello Application, 2

```
  public Hello() {
    setDefaultCloseOperation( EXIT_ON_CLOSE );
    button = new JButton( "Hello" );
    button.setFont( new Font( "SansSerif",
                              Font.BOLD, 24 ) );
    button.addActionListener( this );
    getContentPane().add( button, "Center" );
    setSize( 200, 200 );
  }
```

The Hello Application, 3

```
public void actionPerformed( ActionEvent e ) {
  if ( state == 0 ) {
    button.setText( "Goodbye" );
    state++;
  } else {
    System.exit( 0 );
  }
}
```

Event Types

- **Semantic events vs low-level events**
  - **Semantic events are generally meaningful, often a set of low-level events**
    - **ActionEvent: user action on object (button click, etc.)**
    - **AdjustmentEvent: value adjusted (scroll bar, etc.)**
    - **ItemEvent: selectable item changed (combo box)**
    - **TextEvent: value of text changed**
  - **You can often just use ActionEvent, especially if a button is present to 'Compute', etc.**
  - **Low level events:**
    - **Mouse press, mouse move, key release, etc.**
    - **There are 7 of these**

Event Types, Interfaces

Event type	Interface name	Methods in interface
ActionEvent	ActionListener	void actionPerformed(ActionEvent e)
AdjustmentEvent	AdjustmentListener	void adjustmentValuePerformed(AdjustmentEvent e)
ItemEvent	ItemListener	void itemStateChanged(ItemEvent e)
TextEvent	TextListener	void textValueChanged(TextEvent e)
ComponentEvent	ComponentListener	void componentHidden(ComponentEvent e) void componentMoved(ComponentEvent e) void componentResized(ComponentEvent e) void componentShown(ComponentEvent e)
FocusEvent	FocusListener	void focusGained(FocusEvent e) void focusLost(FocusEvent e)
KeyEvent	KeyListener	void keyPressed(KeyEvent e) void keyReleased(KeyEvent e) void keyTyped(KeyEvent e)
ContainerEvent	ContainerListener	void componentAdded(ContainerEvent e) void componentRemoved(ContainerEvent e)
WindowEvent	WindowListener	(7 methods—see text)
MouseEvent	MouseListener	(7 methods—see text)

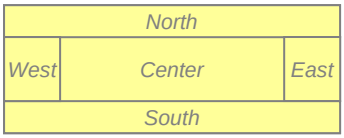
Layout Management

- Layout management is the process of determining the size and location of a container's components.
- Java containers do not handle their own layout. They delegate that task to their layout manager, an instance of another class.
- Each type (class) of layout manager enforces a different *layout policy*.
- Layout proceeds bottom-up: it finds the size of individual elements, then sizes their containers until the frame is sized
- If you do not like a container's default layout manager, you can change it.

```
Container contentPane = getContentPane();
contentPane.setLayout( new FlowLayout() );
```

BorderLayout

"A border layout lays out a container, arranging and resizing its components to fit in five regions: north, south, east, west, and center. Each region may contain no more than one component, and is identified by a corresponding constant: NORTH, SOUTH, EAST, WEST, and CENTER." - javadoc



`BorderLayout` is the default layout manager for `contentPane`

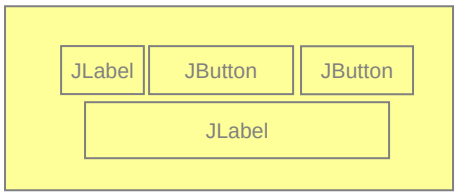
BorderLayout

- Selecting a `LayoutManager` affects how we add components.
- Below we set '`panel`' to use a `BorderLayout` and we add two components, `button` and `label`, to it:

```
// button is an existing JButton
// label is an existing JLabel
JPanel panel = new JPanel();    // default FlowLayout
panel.setLayout(new BorderLayout());
panel.add(button, BorderLayout.NORTH);
panel.add(label, BorderLayout.SOUTH);
// The second argument to add(…) must be BorderLayout.
// NORTH, SOUTH, EAST, WEST, or CENTER.
```

FlowLayout

"A flow layout arranges components in a left-to-right flow, much like lines of text in a paragraph. Flow layouts are typically used to arrange buttons in a panel. It will arrange buttons left to right until no more buttons fit on the same line. Each line is centered." - javadoc



`FlowLayout` is the default layout manager for `JPanel`.

Adding Components with FlowLayout

- Below we set `panel` to use a `FlowLayout` and we add two components, `button` and `label`, to it:

```
// Same assumptions: button and label exist
JPanel panel = new JPanel();
// Panel's layout mgr is FlowLayout already but
// we set it here as an example
panel.setLayout(new FlowLayout());
panel.add(button);
panel.add(label);
// When adding to FlowLayout, order matters
// panel.add(label);
// panel.add(button);
```

Why Use Layout Management

1. **Often you do not know how large your application will be. Even if you call `setSize()`, the user can still physically resize the window of an application.**
2. **Java knows better than you how large components should be. It is hard to gauge the size of a `JLabel`, for instance, except by trial and error. And if you get the size correct on one system and then run it on another with a different set of fonts, the `JLabel` will not be correctly sized.**
3. **Once you lay out a GUI, you may want to make changes that will compromise a layout done by hand. If you use layout management, the new layout happens automatically, but if you are laying out the buttons by hand, you have an annoying task ahead of you.**

JComponent Size

- **Components communicate their layout needs to their enclosing container's layout manager via the methods:**
  - `public Dimension getMinimumSize()`
  - `public Dimension getPreferredSize()`
  - `public Dimension getMaximumSize()`
- **There are three corresponding set methods that allow you to change a component's *size hints*.**
  - `public Dimension setMinimumSize(Dimension d)`
  - `public Dimension setPreferredSize(Dimension d)`
  - `public Dimension setMaximumSize(Dimension d)`
- **Where a Dimension argument, d, is created via:**
  - `Dimension d = new Dimension(int width, int height)`

Exercise: Layout

- First, download and run `ClockFrame` 'as is'.
  - Resize the frame and see how it behaves
- Update `ClockPanel`:
  - `ClockPanel` already uses a `BorderLayout`
  - The buttons and labels are added to a second panel, which is then added to the `ClockPanel` at `SOUTH`
  - Comment out the second panel: `JPanel panel= new JPanel();`
  - Add the buttons and labels directly to `ClockPanel`:
  - Put `tickButton` to the NORTH, `resetButton` SOUTH, `hourLabel` WEST, and `minuteLabel` EAST

Exercise: Layout, p.2

- When you are finished, run `ClockFrame`.
  - Resize your application and see how it behaves
  - The result won't look good. Just make sure you understand how to update the `LayoutManager`s and invoke `add()`
- Optional:
  - `Replace setLayout(new BorderLayout); with setLayout(new FlowLayout);`
  - Change the `add()` calls to omit the second argument (NORTH, SOUTH, ...)
  - See what happens (it's not pretty either)