Abstract Window Toolkit

The Java JDK provides the AWT writing GUI applications or applets.

Applications extend `java.awt.Frame`.

Applets extend `java.applet.Applet`.

Subclasses of `java.awt.Component` can be used to build complicated interfaces.
E.g. `Button`, `Label`, `Scrollbar`, `Panel`.

Any `Panel` can be configured with a `LayoutManager`.
E.g. `BorderLayout`, `GridLayout`.

Any `Component` can be added to the `Panel` in the given layout.

The *event handler* is a *thread* that manages user events such as mouse clicks and keystrokes.

Event `Listener`s can be registered with a `Component`.

ColorFrame

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

public class ColorFrame extends Frame
{
   Canvas c;

   public static void main(String[] args)
   {
      Frame f = new ColorFrame();
      f.show();
   }

   public ColorFrame()
   {
      setSize(200,200);

      c = new Canvas();
      c.setBackground(new Color(0,0,0));

      Scrollbar scrollBar = new Scrollbar(Scrollbar.VERTICAL,0,5,0,260);

      setLayout(new BorderLayout());
      add("Center", c);
      add("East", scrollBar);
```


ColorFrame (cont)

```
      /* listener for scrollbar events */
      scrollBar.addAdjustmentListener(new AdjustmentListener() {
        public void adjustmentValueChanged(AdjustmentEvent e)
         {
            c.setBackground(new Color(e.getValue(),0,0));
         }
      });

      /* add event listener to quit when window is closed */
      addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent e)
         {
            System.exit(0);
         }
      });
   }
}
```


Highlights

Main program first instantiates the frame then shows it.

A `Frame` *is a* `Panel` which *is a* `Component`.

Inner classes for event handling are very useful.

An event listener either needs to implement the `*Listener` interface or extend the `*Adapter` object.

Creating Components

GUI AWT interfaces are built up by combining `Components`.

Not every component you might want is implemented for you.

When implementing your own component:
Make your component bahave as other components behave

An Extended Example

Looking at example component that demonstrates:

- the *Java event model*,
- use of `Components`, and
- good Object Oriented design principles.

`ValueSlider` Example

Say we want a scrollbar that has an editable text area.

This new scrollbar should be have just like `java.awt.Scrollbar`s do:

- It implements the `Adjustable` interface.
- It allows registration of an `AdjustmentListener`.

Additionally, its value can be changed by editing the text area.

Extending `Panel`

To make a composite component, extend `Panel`

```
public class ValueSlider extends Panel
{
   public TextField field;
   public Scrollbar slider;
   public int value;

   public ValueSlider(int orient, int init, int step, int min, int max)
   {
      value = init;

      setLayout(new BorderLayout());

      slider = new Scrollbar(orient, init, step, min, max);
      field = new TextField(Integer.toString(init));

      add("Center",slider);
      add("South",field);
   }
}
```

`ValueSlidersFrame`

```
public class ValueSlidersFrame extends Frame
{
   public ValueSlidersFrame()
   {
      setSize(150,500);

      setLayout(new GridLayout(1,3));

      add(new ValueSlider1(Scrollbar.VERTICAL,0,5,0,260));
      add(new ValueSlider2(Scrollbar.VERTICAL,0,5,0,260));

      ValueSlider3 vs3 = new ValueSlider3(Scrollbar.VERTICAL,0,5,0,260);

      add(vs3);

      vs3.addAdjustmentListener(new AdjustmentListener() {
         public void adjustmentValueChanged(AdjustmentEvent e)
         {
            System.out.println(e.getValue());
         }
      });
   }
}
```


Setting the Value

We want to be able to set the slider value:

```
   public void setValue(int val)
   {
      slider.setValue(val);
      value = slider.getValue();
      field.setText(Integer.toString(value));
   }

   public int getValue()
   {
      return value;
   }
```


Listening for Events

Need to listen for events on `field` and `slider`.

```
public class ValueSlider
                  extends Panel
                  implements AdjustmentListener, ActionListener
{
   ...
   public ValueSlider(...)
   {
      ...

      slider.addAdjustmentListener(this);
      field.addActionListener(this);
   }
}
```


Implement Listener Methods

Must implement `adjustmentValueChanged(...)` and `actionPerformed(...)`

```
   public void actionPerformed(ActionEvent e)
   {
      try {
         setValue(Integer.parseInt(field.getText()));
      } catch (NumberFormatException nfe) {
         setValue(value);
      }
   }

   public void adjustmentValueChanged(AdjustmentEvent e)
   {
      setValue(e.getValue());
   }
```

Allow AdjustmentListeners

Need to allow AdjustmentListeners to be added to listen to this slider's events.

The AWTEventMulticaster can be used to make a single Listener out of two Listeners. It is a Listener itself!

When it gets an event, it will send it to both listeners:

```
protected AWTEventMulticaster(EventListener a, EventListener b);

static public AdjustmentListener add(SomeListener a, SomeListener b);
static public AdjustmentListener remove(SomeListener l, SomeListener oldl);
```

Where Some is Adjustment, Action, etc.

Allowing Listeners (cont)

```
public class ValueSlider
                 extends Panel
                 implements AdjustmentListener, ActionListener, Adjustable
{
   AdjustmentListener listener = null;

   ...

   public void addAdjustmentListener(AdjustmentListener l)
   {
      if (listener == null)
         listener = l;
      else
         listener = AWTEventMulticaster.add(listener,l);
   }

   public void removeAdjustmentListener(AdjustmentListener l)
   {
      if (listener != null)
         listener = AWTEventMulticaster.remove(listener,l);
   }
}
```


Dispatching Events

```
public void dispatchEvent(int value)
{
   if (listener != null)
      listener.adjustmentValueChanged(
         new AdjustmentEvent(
                this,
                AdjustmentEvent.ADJUSTMENT_VALUE_CHANGED,
                AdjustmentEvent.TRACK,
                value));
}
```

Need to call dispatchEvent(...) at end of actionPerformed(...) and adjustmentValueChanged(...)

Also need to implement rest of methods in Adjustable interface. Each should just call corresponding method on slider. E.g.

```
public int getMinimum()
   { return slider.getMinimum(); }
```


Result

Now we can use ValueSliders where ever we previously used Scrollbars.

As an exercise, modify ColorFrame to use the ValueSlider.