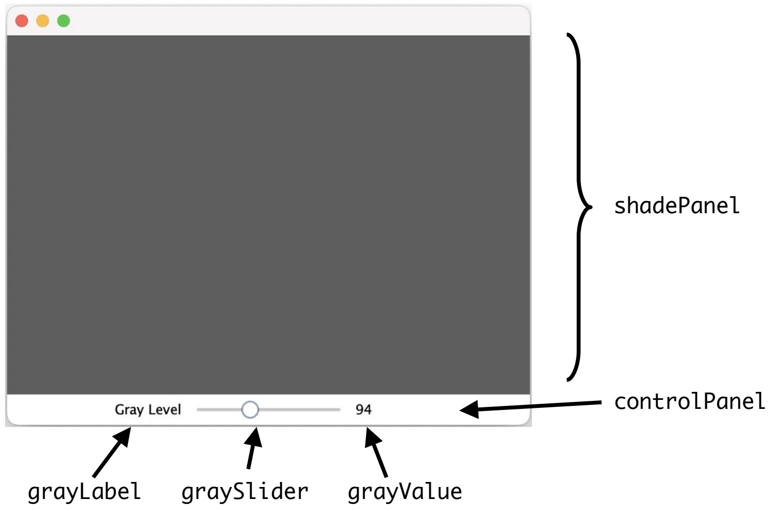
# Lab 6 — GUI Programming

1. Download **lab06_files.zip** from our class web page and open the file **GrayViewer.java**. We will create a GUI program to display shades of gray from 0-255 (where 0 is black and 255 is white), as shown below. The window will be divided into two regions: the *shade panel*, and the *control panel* with a slider for adjusting the gray level.



Each panel will be represented as a JPanel. The control panel will contain a JLabel called `grayLabel` to display the text "Gray Level", a JSlider called `graySlider` for the slider control, and another JLabel called `grayValue` to display the numerical value corresponding to the gray level (94 in the example above). As a first step, add the following code to the GrayViewer constructor to create these five objects:

```
this.shadePanel = new JPanel();
this.controlPanel = new JPanel();
this.grayLabel = new JLabel("Gray Level");
this.graySlider = new JSlider(0, 255);  // slider ranges from 0 to 255
this.grayValue = new JLabel("127");     // middle value of 0-255 range
```

2. Next, we will add `grayLabel`, `graySlider`, and `grayValue` to the control panel in a FlowLayout arrangement:

```
this.controlPanel.setLayout(new FlowLayout());
this.controlPanel.add(this.grayLabel);
this.controlPanel.add(this.graySlider);
this.controlPanel.add(this.grayValue);
```

Likewise, we will add `shadePanel` and `controlPanel` to the top-level JFrame (`this`) in a BorderLayout arrangement, with `shadePanel` going in the CENTER and `controlPanel` going in the SOUTH:

```
this.setLayout(new BorderLayout());
this.add(this.shadePanel, BorderLayout.CENTER);
this.add(this.controlPanel, BorderLayout.SOUTH);
```

Compile and run GrayViewer. The result should look something like the window shown above, although changing the slider won't actually do anything yet.

3. If you like, you can change the background color of the control panel to make it a little more visible:

```
this.controlPanel.setBackground(Color.YELLOW);
```

4. Our next task is to make the slider control responsive. To do this, we need to create a *listener* object and attach it to `graySlider`. This listener object will be an instance of a new class called GrayViewerListener. To be able to respond to sliders, our GrayViewerListener class must implement Java's ChangeListener interface (which is

available in the `javax.swing.event` package). This interface requires the following method:

```
void stateChanged(ChangeEvent event) { ... }
```

Whenever the slider is moved (even just a little), it generates an *event* object of type ChangeEvent, which gets passed into the above method by the system. Let's start by having this method just print out a message when the slider changes. Open the file **GrayViewerListener.java** and add the code shown below, and then compile it.

```
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;  // ChangeListener, ChangeEvent

public class GrayViewerListener implements ChangeListener {

    public void stateChanged(ChangeEvent event) {
        System.out.print("Slider moved! ");  // use print instead of println here
    }
}
```

5. To make the slider respond, we next create a GrayViewerListener and attach it to the slider. Add the following code to the GrayViewer constructor:

   ```
   GrayViewerListener listener = new GrayViewerListener();
   this.graySlider.addChangeListener(listener);
   ```

   Now when you move the slider, you should see the message "Slider moved!" printed out over and over.

6. However, rather than printing a message, what we really want to happen is for the `shadePanel` to update its shade of gray based on the new position of `graySlider`, and the `grayValue` label to reflect this change. To accomplish this, the GrayViewerListener will need access to these objects. When we construct a new GrayViewerListener, we will pass these three objects into the constructor, which will store them as instance variables inside the GrayViewerListener. That way, the stateChanged method will be able to refer to them directly.

   Add the new instance variables and constructor to your GrayViewerListener class, and replace the print statement in the stateChanged method, as shown below. The new version of stateChanged retrieves the position value from the slider, then updates the background color of `shadePanel` and the text label of `grayValue` accordingly:

   ```
   public class GrayViewerListener implements ChangeListener {

       private JPanel shadePanel;
       private JSlider graySlider;
       private JLabel grayValue;

       // constructor
       public GrayViewerListener(JPanel shadePanel, JSlider graySlider, JLabel grayValue) {
           this.shadePanel = shadePanel;
           this.graySlider = graySlider;
           this.grayValue = grayValue;
       }

       public void stateChanged(ChangeEvent event) {
           int v = this.graySlider.getValue();
           this.shadePanel.setBackground(new Color(v, v, v));  // equal amounts of red, green, blue = gray
           this.grayValue.setText(Integer.toString(v));
       }
   }
   ```

   Also, now that the GrayViewerListener constructor takes three objects, we need to update the line in GrayViewer that creates the GrayViewerListener:

   ```
   GrayViewerListener listener = new GrayViewerListener(this.shadePanel, this.graySlider, this.grayValue);
   ```

   Compile and run the new version of your code. The slider should now work the way we want.

7. Notice that when the GrayViewer program first starts up, the slider position is at 127 but the shade panel is white, and moving the slider even a tiny amount abruptly changes the shade. We can fix this by defining a helper method in GrayViewer to update the shade panel (similar to the stateChanged method in GrayViewerListener), and then calling this method in the GrayViewer constructor. Add the following method definition to the GrayViewer class:

```
public void updatePanelColor() {
    int v = this.graySlider.getValue();
    this.shadePanel.setBackground(new Color(v, v, v));
    this.grayValue.setText(Integer.toString(v));
}
```

   and add the following line at the end of the GrayViewer constructor:

```
this.updatePanelColor();
```

   This way, the GrayViewer program will update the panel color immediately to reflect the initial slider value when it starts up.

8. However, if you look closely, you will see that the control panel is still slightly wonky. That is, changing the slider causes the “Gray Level” JLabel to move back and forth slightly. This is due to the FlowLayout. We can fix this by making the control panel use a GridLayout (with 1 row and 3 columns) instead of a FlowLayout. Change the call to setLayout in the GrayViewer constructor to the following:

```
this.controlPanel.setLayout(new GridLayout(1, 3));
```

   and then test it out. Unfortunately, the GridLayout aligns the “Gray Level” JLabel on the left, by default. Fortunately, we can easily change the alignment by providing a second argument to the JLabel constructor:

```
this.grayLabel = new JLabel("Gray Level", SwingConstants.RIGHT);
```

   That puts the JLabel next to the slider and prevents it from moving around as the slider is changed.

9. At this point, everything should be working properly. The final step is to simplify the code by making GrayViewerListener be a private inner class of GrayViewer. The advantage of this is that as an inner class, GrayViewerListener will have direct access to the instance variables of GrayViewer, including `shadePanel`, `graySlider`, and `grayValue`. Thus we will no longer need to pass these variables into the GrayViewerListener constructor. In fact, we will no longer need to explicitly define the constructor at all. Add the definition below to the GrayViewer class. Notice that stateChanged now just calls the GrayViewer’s updatePanelColor helper method whenever the slider is moved.

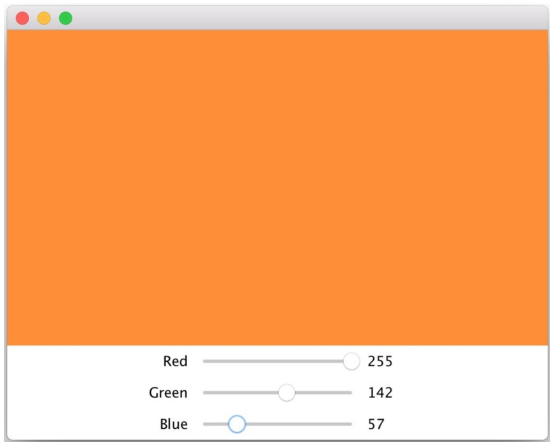
```
private class GrayViewerListener implements ChangeListener {

    public void stateChanged(ChangeEvent event) {
        updatePanelColor();
    }
}
```

   We also need to remove the instance variables `this.shadePanel`, `this.graySlider`, and `this.grayValue` from the call to the GrayViewerListener constructor in GrayViewer:

```
GrayViewerListener listener = new GrayViewerListener();  // removed instance variables
```

   After making these changes, recompile and test the program. Everything should work the same as before.

10. Using your completed GrayViewer program as a guide, create a new program called **ColorViewer** that displays arbitrary RGB colors in a similar way. Your program should have *three* separate sliders for specifying the Red, Green, and Blue color levels, which get mixed together to produce the color displayed, as shown below:



11. Following the example of MyPicture.java, define a new class called ***YourName*Fish** that extends JPanel and draws an image of a fish or some other aquatic creature in the graphics window. Your aquatic creature can be as colorful and elaborate as you like (be creative!). However, your drawing should consist of just your creature, with no additional features in the background. It should fit inside a bounding box that is no more than 350 pixels wide and 350 pixels high, relative to position (0, 0) in the upper left corner of the screen. Next week, we will implement an animated fish tank (the "Javaquarium") containing everyone's creatures swimming around. For now, you can use **TestFish.java** to test your code.