Coding a Bar Chart Creator

The Frame Class

Our program needs the following user-interface components:

- A text field for entering a bar label
- A text field for entering a bar value
- A button for adding a new bar with the given label and value
- A button for removing the last bar
- A component for drawing the chart

The chart component needs to provide two methods that support the button commands:

```java
public void append(String label, double value)
public void removeLast()
```

For now, let us assume that those methods have been implemented. In the frame class, we need to provide action listeners that call them:

```java
public class ChartFrame extends JFrame
{
    private JTextField labelField;
    private JTextField valueField;
    private JButton addButton;
    private JButton removeButton;
    private ChartComponent chart;

    class AddBarListener implements ActionListener
    {
        public void actionPerformed(ActionEvent event)
        {
```

Worked Example 10.1  Coding a Bar Chart Creator

In this Worked Example, we will develop a simple program for creating bar charts. The user enters labels and values for the bars, and the program displays the chart.

We also allow the user to fix mistakes by removing the last bar. Admittedly, the user interface is a bit limited. Worked Example 11.1 will improve on it, allowing users to use the mouse to edit the chart.
String label = labelField.getText();
double value = Double.parseDouble(valueField.getText());
chart.append(label, value);
}
}

class RemoveBarListener implements ActionListener
{
    public void actionPerformed(ActionEvent event)
    {
        chart.removeLast();
    }
}

private void createButtons()
{
    addButton = new JButton("Add");
    addButton.addActionListener(new AddBarListener());
    removeButton = new JButton("Remove last");
    removeButton.addActionListener(new RemoveBarListener());
}

The remainder of the frame class is straightforward. We need to construct the text field and the chart, place everything in a panel, and add the panel to the frame.

public class ChartFrame extends JFrame
{
    private static final int FRAME_WIDTH = 400;
    private static final int FRAME_HEIGHT = 400;

    private static final int CHART_WIDTH = 300;
    private static final int CHART_HEIGHT = 300;

    private static final String DEFAULT_LABEL = "Description";
    private static final double DEFAULT_VALUE = 100;

    public ChartFrame()
    {
        chart = new ChartComponent();
        chart.setPreferredSize(new Dimension(CHART_WIDTH, CHART_HEIGHT));
        chart.append(DEFAULT_LABEL, DEFAULT_VALUE);
        createTextFields();
        createButtons();
        createPanel();
        setSize(FRAME_WIDTH, FRAME_HEIGHT);
    }

    private void createTextFields()
    {
        final int LABEL_FIELD_WIDTH = 20;
        labelField = new JTextField(LABEL_FIELD_WIDTH);
        labelField.setText(DEFAULT_LABEL);
        final int VALUE_FIELD_WIDTH = 10;
        valueField = new JTextField(VALUE_FIELD_WIDTH);
        valueField.setText("" + DEFAULT_VALUE);
private void createPanel() {
    JPanel panel = new JPanel();
    panel.add(labelField);
    panel.add(valueField);
    panel.add(addButton);
    panel.add(removeButton);
    panel.add(chart);
    add(panel);
}

Now let us turn to the bar chart itself. Unlike the chart of 10.4.3, this chart draws the labels in addition to the bars. A bar chart consists of multiple bars, each of which has a label and a value. It is best to make a class for a bar that holds both the label and the value. Then the chart component stores an array of bars:

```java
public class Bar {
    private String label;
    private double value;

    public Bar(String aLabel, double aValue) {
        label = aLabel;
        value = aValue;
    }
}
```

```java
public class ChartComponent extends JComponent {
    private ArrayList<Bar> bars;
    private double maxValue;
    
    public void draw(Graphics g, int y, double scale) {
        final int GAP = 2;
        g.fillRect(0, y, (int) (value * scale), HEIGHT);
        g.setColor(Color.WHITE);
        g.drawString(label, GAP, y + HEIGHT - GAP);
        g.setColor(Color.BLACK);
    }
}
```

To draw the chart, we ask that each bar draw itself. In general, this is a good strategy when you have an image that is made up of parts. Turn each part into an object with a draw method. A bar needs to know where it is situated in the chart—that is, the vertical displacement. It also needs to know how to scale itself in order to fit into the component. After all, the bar values are in units that make sense to the user (perhaps dollars or meters). Finally, in order to do any drawing, the draw method needs a Graphics object.

Here is the draw method for the Bar class:

```java
public void draw(Graphics g, int y, double scale) {
    final int GAP = 2;
    g.fillRect(0, y, (int) (value * scale), HEIGHT);
    g.setColor(Color.WHITE);
    g.drawString(label, GAP, y + HEIGHT - GAP);
    g.setColor(Color.BLACK);
}
```

Note that we draw the label in white on the black bar. When we are done, we need to restore the black color so that the next bar will be drawn in black.
When you look at this method, it becomes clearer why it is a good idea for the bar to draw itself, instead of placing the drawing instructions for all bars into the chart’s `paintComponent` method. In this arrangement, the `ChartComponent` class need not to query the bar’s data (that is, the label and width), and the bar class doesn’t need to provide methods that reveal them. The `ChartComponent`’s `paintComponent` method simply asks each bar to draw itself. It’s responsible for stacking the bars atop each other, and for fitting them horizontally inside the component.

```java
public void paintComponent(Graphics g)
{
    final int GAP = 5;
    int y = GAP;
    double scale = getWidth() / maxValue;
    for (Bar b : bars)
    {
        b.draw(g, y, scale);
        y = y + Bar.HEIGHT + GAP;
    }
}
```

The `append` and `removeLast` methods add and remove a bar, then invoke `repaint` so that the changed chart is displayed.

```java
public void append(String label, double value)
{
    bars.add(new Bar(label, value));
    if (value > maxValue) { maxValue = value; }
    repaint();
}

public void removeLast()
{
    int n = bars.size();
    if (n == 0) { return; }
    bars.remove(n - 1);
    repaint();
}
```

Finally, we need a routine viewer class that shows the frame. This completes the program.

```java
import javax.swing.JFrame;

/**
 * This program displays an editable bar chart.
 */
public class ChartViewer
{
}
```

It is worth reflecting on the division of labor between the component and frame classes. The `ChartComponent` class knows how to draw a chart, and it has methods for modifying the chart data. But it has no notion of a user interface. The same class could be used if we had a different user interface, perhaps with voice recognition instead of the text fields and buttons.

The `ChartFrame`, on the other hand, is all about the user interface. It handles the text fields and buttons. As soon as it knows what the user wants to do, it hands the work off to the chart component.

This is a useful division of labor, giving you guidance if the program needs to be enhanced. For a fancier rendering of the chart, improve the chart component. For more control over the chart’s appearance, add user-interface components to the frame.
public static void main(String[] args) {
    JFrame frame = new ChartFrame();
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
}

public class ChartFrame extends JFrame {

    private static final int FRAME_WIDTH = 400;
    private static final int FRAME_HEIGHT = 400;

    private static final int CHART_WIDTH = 300;
    private static final int CHART_HEIGHT = 300;

    private static final String DEFAULT_LABEL = "Description";
    private static final double DEFAULT_VALUE = 100;

    private JTextField labelField;
    private JTextField valueField;
    private JButton addButton;
    private JButton removeButton;
    private ChartComponent chart;

    public ChartFrame() {
        chart = new ChartComponent();
        chart.setPreferredSize(new Dimension(CHART_WIDTH, CHART_HEIGHT));
        chart.append(DEFAULT_LABEL, DEFAULT_VALUE);

        createTextFields();
        createButtons();
        createPanel();

        setSize(FRAME_WIDTH, FRAME_HEIGHT);
    }

    private void createTextFields() {
        final int LABEL_FIELD_WIDTH = 20;
        labelField = new JTextField(LABEL_FIELD_WIDTH);
        labelField.setText(DEFAULT_LABEL);

        final int VALUE_FIELD_WIDTH = 10;
        valueField = new JTextField(VALUE_FIELD_WIDTH);
        valueField.setText("" + DEFAULT_VALUE);
    }

    private void createButtons() {
        addButton = new JButton("Add");
        removeButton = new JButton("Remove");
    }

    private void createPanel() {
        JPanel panel = new JPanel();
        panel.add(labelField);
        panel.add(valueField);
        panel.add(addButton);
        panel.add(removeButton);
        add(panel);
    }

    public ChartComponent getChart() {
        return chart;
    }
}

/*
 * A frame that allows users to edit a bar chart.
 */

import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class ChartFrame extends JFrame {

    private static final int FRAME_WIDTH = 400;
    private static final int FRAME_HEIGHT = 400;

    private static final int CHART_WIDTH = 300;
    private static final int CHART_HEIGHT = 300;

    private static final String DEFAULT_LABEL = "Description";
    private static final double DEFAULT_VALUE = 100;

    private JTextField labelField;
    private JTextField valueField;
    private JButton addButton;
    private JButton removeButton;
    private ChartComponent chart;

    public ChartFrame() {
        chart = new ChartComponent();
        chart.setPreferredSize(new Dimension(CHART_WIDTH, CHART_HEIGHT));
        chart.append(DEFAULT_LABEL, DEFAULT_VALUE);

        createTextFields();
        createButtons();
        createPanel();

        setSize(FRAME_WIDTH, FRAME_HEIGHT);
    }

    private void createTextFields() {
        final int LABEL_FIELD_WIDTH = 20;
        labelField = new JTextField(LABEL_FIELD_WIDTH);
        labelField.setText(DEFAULT_LABEL);

        final int VALUE_FIELD_WIDTH = 10;
        valueField = new JTextField(VALUE_FIELD_WIDTH);
        valueField.setText("" + DEFAULT_VALUE);
    }

    private void createButtons() {
        addButton = new JButton("Add");
        removeButton = new JButton("Remove");
    }

    private void createPanel() {
        JPanel panel = new JPanel();
        panel.add(labelField);
        panel.add(valueField);
        panel.add(addButton);
        panel.add(removeButton);
        add(panel);
    }

    public ChartComponent getChart() {
        return chart;
    }
}
class AddBarListener implements ActionListener
{
    public void actionPerformed(ActionEvent event)
    {
        String label = labelField.getText();
        double value = Double.parseDouble(valueField.getText());
        chart.append(label, value);
    }
}

class RemoveBarListener implements ActionListener
{
    public void actionPerformed(ActionEvent event)
    {
        chart.removeLast();
    }
}

public void createButtons()
{
    addButton = new JButton("Add");
    addButton.addActionListener(new AddBarListener());
    removeButton = new JButton("Remove last");
    removeButton.addActionListener(new RemoveBarListener());
}

public void createPanel()
{
    JPanel panel = new JPanel();
    panel.add(labelField);
    panel.add(valueField);
    panel.add(addButton);
    panel.add(removeButton);
    panel.add(chart);
    add(panel);
}

worked_example_1/ChartComponent.java

import java.awt.Color;
import java.awt.Graphics;
import java.util.ArrayList;
import javax.swing.JComponent;

/**
 * A component that draws a chart.
 */
public class ChartComponent extends JComponent
{
    private ArrayList<Bar> bars;
    private double maxValue;

    /**
     * Constructs an empty chart.
     */
    public ChartComponent()
    {

}
Coding a Bar Chart Creator

```java
bars = new ArrayList<Bar>();
maxValue = 1;
}

/**
  * Appends a bar to this chart.
  * @param label the label for the bar
  * @param value the value of the bar
  */
public void append(String label, double value)
{
bars.add(new Bar(label, value));
if (value > maxValue) { maxValue = value; }
repaint();
}

/**
  * Removes the last bar of this chart.
  */
public void removeLast()
{
  int n = bars.size();
  if (n == 0) { return; }
  bars.remove(n - 1);
  repaint();
}

public void paintComponent(Graphics g)
{
  final int GAP = 5;
  int y = GAP;
  double scale = getWidth() / maxValue;
  for (Bar b : bars)
  {
    b.draw(g, y, scale);
    y = y + Bar.HEIGHT + GAP;
  }
}
```

worked_example_1/Bar.java

```java
import java.awt.Color;
import java.awt.Graphics;

/**
  * A bar of a bar chart.
  */
public class Bar
{
  private String label;
  private double value;
  public static final int HEIGHT = 15;

  /**
   * Constructs a bar with a given label and value.
   * @param label the label for this bar
   * @param value the value of this bar
   */
```
public Bar(String aLabel, double aValue)
{
    label = aLabel;
    value = aValue;
}

/**
 * Draws this bar.
 * @param g the Graphics object
 * @param y the top of the bar
 * @param scale the horizontal scale factor for fitting the bar into the component
 */
public void draw(Graphics g, int y, double scale)
{
    final int GAP = 2;
    g.fillRect(0, y, (int) (value * scale), HEIGHT);
    g.setColor(Color.WHITE);
    g.drawString(label, GAP, y + HEIGHT - GAP);
    g.setColor(Color.BLACK);
}