

```
1: /**
2:  * This application allows the user to generate different colors using sliders
3:  * representing Red, Green, and Blue values and outputs the color with a circle
4:  * in the middle of the window as well as the numeric representation and a bar
5:  * graph visually representing the ratio of component primary colors. The
6:  * numeric representations can be either decimal, binary, octal, or hexadecimal.
7:  *
8:  * @name      ColorFactory (Homework Assignment 06)
9:  * @author    Ravi S. Ramphal
10:  * @class     CCSF CS111B
11:  * @date      2017.07.27
12:  * @version   1.0
13:  */
14:
15: import java.awt.*;
16: import java.awt.event.*;
17:
18: class Circle extends Canvas
19: {
20:     int circleSize;
21:
22:     // ===== paint() =====
23:
24:     public void paint (Graphics graphicsContext)
25:     {
26:         int x = (getWidth() / 2) - (circleSize / 2);
27:         int y = (getHeight() / 2) - (circleSize / 2);
28:         graphicsContext.fillOval(x, y, circleSize, circleSize);
29:     }
30:
31:     // ===== Circle() =====
32:
33:     public Circle (int size)
34:     {
35:         circleSize = size;
36:     }
37: }
38:
39: class Bars extends Canvas
40: {
41:     final double SCALE = 0.4;
42:
43:     final int BAR_WIDTH = (int) (20.0 * SCALE); // relative to 256 then scaled
44:     final int BAR_GAP = (int) (10.0 * SCALE); // relative to 256 then scaled
45:
46:     public int redValue = 0;
47:     public int blueValue = 0;
48:     public int greenValue = 0;
49:
50:     int horizontalOffset = 100;
51:     int verticalOffset = 10;
52:
53:     // ===== paint() =====
54:
55:     public void paint (Graphics graphicsContext)
56:     {
57:         graphicsContext.setColor(Color.RED);
58:         graphicsContext.fillRect(
59:             horizontalOffset + BAR_GAP,
60:             verticalOffset + (int) (255.0 * SCALE) - (int) (redValue * SCALE),
61:             BAR_WIDTH,
62:             (int) (redValue * SCALE)
63:         );
64:
65:         graphicsContext.setColor(Color.GREEN);
66:         graphicsContext.fillRect(
67:             horizontalOffset + (2 * BAR_GAP) + BAR_WIDTH,
68:             verticalOffset + (int) (255.0 * SCALE) - (int) (greenValue * SCALE),
```

```
69:         BAR_WIDTH,
70:         (int) (greenValue * SCALE)
71:     );
72:
73:     graphicsContext.setColor(Color.BLUE);
74:     graphicsContext.fillRect(
75:         horizontalOffset + (3 * BAR_GAP) + (2 * BAR_WIDTH),
76:         verticalOffset + (int) (255.0 * SCALE) - (int) (blueValue * SCALE),
77:         BAR_WIDTH,
78:         (int) (blueValue * SCALE)
79:     );
80: }
81:
82: // ===== updateBars() =====
83:
84: public void updateBars (int red, int green, int blue)
85: {
86:     redValue    = red;
87:     greenValue  = green;
88:     blueValue   = blue;
89:     repaint();
90: }
91: }
92:
93: public class ColorFactory extends Frame implements AdjustmentListener,
94:                               ItemListener
95: {
96:     final String TITLE = "Color Factory";
97:     final int DIRECTION = Scrollbar.HORIZONTAL;
98:     final int CIRCLE_SIZE = 200;
99:
100:     Dimension screenSize;
101:     Dimension windowSize;
102:
103:     int redValue    = 0;
104:     int greenValue  = 0;
105:     int blueValue   = 0;
106:
107:     Label header = new Label(TITLE, Label.CENTER);
108:
109:     Panel columns = new Panel();
110:
111:     Panel        outputPanel    = new Panel();
112:     CheckboxGroup outputGroup    = new CheckboxGroup();
113:     Checkbox     decimal        = new Checkbox("Decimal", outputGroup, true );
114:     Checkbox     binary         = new Checkbox("Binary",   outputGroup, false);
115:     Checkbox     octal          = new Checkbox("Octal",    outputGroup, false);
116:     Checkbox     hex            = new Checkbox("Hex",      outputGroup, false);
117:     Label        output         = new Label(getOutput(), Label.CENTER);
118:
119:     Circle display = new Circle(CIRCLE_SIZE);
120:
121:     Panel rgbPanel = new Panel();
122:
123:     Panel sliderPanel = new Panel();
124:
125:     Label    redLabel    = new Label("Red  ", Label.RIGHT);
126:     Scrollbar redSlider  = new Scrollbar(DIRECTION, redValue, 1, 0, 256);
127:     Label    redNumber   = new Label("  " + redValue, Label.LEFT);
128:
129:     Label    greenLabel  = new Label("Green ", Label.RIGHT);
130:     Scrollbar greenSlider = new Scrollbar(DIRECTION, greenValue, 1, 0, 256);
131:     Label    greenNumber = new Label("  " + greenValue, Label.LEFT);
132:
133:     Label    blueLabel   = new Label("Blue  ", Label.RIGHT);
134:     Scrollbar blueSlider  = new Scrollbar(DIRECTION, blueValue, 1, 0, 256);
135:     Label    blueNumber  = new Label("  " + blueValue, Label.LEFT);
136:
```

```
137: Panel barsPanel = new Panel();
138: Bars bars = new Bars();
139:
140: // ===== leftPad() =====
141:
142: private static String leftPad (String input, int width, char padder)
143: {
144:     // cache number of characters in input
145:     int inputWidth = input.length();
146:
147:     // if user desires a width shorter than input width, return unchanged
148:     if (inputWidth >= width) return input;
149:
150:     // initialize a StringBuilder with capacity set to desired width
151:     StringBuilder output = new StringBuilder(width);
152:
153:     // repeat characters to fill missing width
154:     for (int i = 0; i < (width - inputWidth); i++)
155:     {
156:         output.append(padder);
157:     }
158:
159:     // append original input to end of repeated characters and return String
160:     return output.append(input).toString();
161: }
162:
163: // ===== setWindowSize() =====
164:
165: private void setWindowSize ()
166: {
167:     Toolkit toolkit = Toolkit.getDefaultToolkit();
168:     screenSize = toolkit.getScreenSize();
169:     windowSize = new Dimension(screenSize.width / 2, screenSize.height / 3);
170:     setSize(windowSize);
171: }
172:
173: // ===== setupOutput() =====
174:
175: private void setupOutput ()
176: {
177:     decimal.addItemListener(this);
178:     binary.addItemListener(this);
179:     octal.addItemListener(this);
180:     hex.addItemListener(this);
181:
182:     output.setFont(new Font("Dialog", Font.BOLD, 12));
183:
184:     outputPanel.setLayout(new GridLayout(5, 1, 0, 0));
185:
186:     outputPanel.add(decimal);
187:     outputPanel.add(binary);
188:     outputPanel.add(octal);
189:     outputPanel.add(hex);
190:     outputPanel.add(output);
191:
192:     columns.add(outputPanel);
193: }
194:
195: // ===== setupDisplay() =====
196:
197: private void setupDisplay ()
198: {
199:     columns.add(display);
200: }
201:
202: // ===== setupRGB() =====
203:
204: private void setupRGB ()
```

```
205:    {
206:        Font rgbFont = new Font("Dialog", Font.BOLD, 12);
207:
208:        sliderPanel.setLayout(new GridLayout(3, 3));
209:
210:        redLabel.setFont(rgbFont);
211:        redSlider.addAdjustmentListener(this);
212:        redNumber.setFont(rgbFont);
213:        sliderPanel.add(redLabel);
214:        sliderPanel.add(redSlider);
215:        sliderPanel.add(redNumber);
216:
217:        greenLabel.setFont(rgbFont);
218:        greenSlider.addAdjustmentListener(this);
219:        greenNumber.setFont(rgbFont);
220:        sliderPanel.add(greenLabel);
221:        sliderPanel.add(greenSlider);
222:        sliderPanel.add(greenNumber);
223:
224:        blueLabel.setFont(rgbFont);
225:        blueSlider.addAdjustmentListener(this);
226:        blueNumber.setFont(rgbFont);
227:        sliderPanel.add(blueLabel);
228:        sliderPanel.add(blueSlider);
229:        sliderPanel.add(blueNumber);
230:
231:        rgbPanel.setLayout(new GridLayout(2, 1, 0, 0));
232:
233:        rgbPanel.add(sliderPanel);
234:        rgbPanel.add(bars);
235:
236:        columns.add(rgbPanel);
237:    }
238:
239:    // ===== updateColorValues() =====
240:
241:    public void updateColorValues ()
242:    {
243:        redValue    = redSlider.getValue();
244:        greenValue   = greenSlider.getValue();
245:        blueValue    = blueSlider.getValue();
246:    }
247:
248:    // ===== updateNumbers() =====
249:
250:    public void updateNumbers ()
251:    {
252:        redNumber.setText("" + redValue);
253:        greenNumber.setText("" + greenValue);
254:        blueNumber.setText("" + blueValue);
255:    }
256:
257:    // ===== updateDisplayColor() =====
258:
259:    public void updateDisplayColor ()
260:    {
261:        display.setForeground(new Color(redValue, greenValue, blueValue));
262:    }
263:
264:    // ===== updateBars() =====
265:
266:    public void updateBars ()
267:    {
268:        bars.updateBars(redValue, greenValue, blueValue);
269:    }
270:
271:    // ===== getOutput() =====
272:
```

```
273: public String getOutput ()
274: {
275:     Checkbox selectedOutput = outputGroup.getSelectedCheckbox();
276:
277:     if (selectedOutput == decimal)
278:     {
279:         return (
280:             redValue  + ", " +
281:             greenValue + ", " +
282:             blueValue
283:         );
284:     }
285:     else if (selectedOutput == binary)
286:     {
287:         return (
288:             leftPad(Integer.toBinaryString(redValue), 8, '0') + ", " +
289:             leftPad(Integer.toBinaryString(greenValue), 8, '0') + ", " +
290:             leftPad(Integer.toBinaryString(blueValue), 8, '0')
291:         );
292:     }
293:     else if (selectedOutput == octal)
294:     {
295:         return (
296:             Integer.toOctalString(redValue)  + ", " +
297:             Integer.toOctalString(greenValue) + ", " +
298:             Integer.toOctalString(blueValue)
299:         );
300:     }
301:     else if (selectedOutput == hex)
302:     {
303:         return (
304:             "#" +
305:             leftPad(Integer.toHexString(redValue), 2, '0') +
306:             leftPad(Integer.toHexString(greenValue), 2, '0') +
307:             leftPad(Integer.toHexString(blueValue), 2, '0')
308:         ).toUpperCase();
309:     }
310:
311:     return "";
312: }
313:
314: // ===== updateOutput() =====
315:
316: public void updateOutput ()
317: {
318:     output.setText(getOutput());
319: }
320:
321: // ===== adjustmentValueChanged() =====
322:
323: public void adjustmentValueChanged (AdjustmentEvent event)
324: {
325:     updateColorValues();
326:     updateOutput();
327:     updateDisplayColor();
328:     updateNumbers();
329:     updateBars();
330: }
331:
332: // ===== itemStateChanged() =====
333:
334: public void itemStateChanged (ItemEvent event)
335: {
336:     updateOutput();
337: }
338:
339: // ===== ColorFactory() =====
340:
```

```
341: public ColorFactory ()
342: {
343:     setWindowSize();
344:     setTitle(TITLE);
345:     setLayout(new BorderLayout(0, 0));
346:
347:     header.setFont(new Font("Dialog", Font.BOLD, 24));
348:     add(header, BorderLayout.PAGE_START);
349:
350:     columns.setLayout(new GridLayout(1, 3));
351:     add(columns, BorderLayout.CENTER);
352:
353:     setupOutput();
354:     setupDisplay();
355:     setupRGB();
356:
357:     setVisible(true);
358: }
359:
360: // ===== closeFrame() =====
361:
362: public static void closeFrame (Frame frame)
363: {
364:     frame.addWindowListener(new WindowAdapter ()
365:     {
366:         public void windowClosing (WindowEvent event)
367:         {
368:             System.exit(0);
369:         }
370:     });
371: }
372:
373: // ===== main() =====
374:
375: public static void main (String ... args) {
376:     ColorFactory colorFactory = new ColorFactory();
377:     closeFrame(colorFactory);
378: }
379: }
```