

src\ImageEditorPanel.java

```
1  import java.awt.image.BufferedImage;
2  import java.io.IOException;
3  import java.io.File;
4  import javax.imageio.ImageIO;
5  import java.awt.Color;
6  import java.awt.Graphics;
7  import java.awt.Dimension;
8  import java.awt.event.KeyEvent;
9  import java.awt.event.KeyListener;
10 import javax.swing.JPanel;
11
12 @SuppressWarnings("serial")
13
14 public class ImageEditorPanel extends JPanel implements KeyListener {
15
16     Color[][] pixels;
17
18     public ImageEditorPanel() {
19         BufferedImage imageIn = null;
20         try {
21             imageIn = ImageIO.read(new File("City.jpg"));
22         } catch (IOException e) {
23             System.out.println(e);
24             System.exit(1);
25         }
26         pixels = makeColorArray(imageIn);
27         setPreferredSize(new Dimension(pixels[0].length, pixels.length));
28         setBackground(Color.BLACK);
29         addKeyListener(this);
30     }
31
32     public void paintComponent(Graphics g) {
33         for (int row = 0; row < pixels.length; row++) {
34             for (int col = 0; col < pixels[0].length; col++) {
35                 g.setColor(pixels[row][col]);
36                 g.fillRect(col, row, 1, 1);
37             }
38         }
39     }
40
41     public Color[][] makeColorArray(BufferedImage image) {
42         int width = image.getWidth();
43         int height = image.getHeight();
44         Color[][] result = new Color[height][width];
45         for (int row = 0; row < height; row++) {
46             for (int col = 0; col < width; col++) {
47                 Color c = new Color(image.getRGB(col, row), true);
48                 result[row][col] = c;
49             }
50         }
51         return result;
52     }
53 }
```

```
54     public Color[][] horizontalFlip(Color[][] oldArr) {
55         Color[][] newArr = new Color[oldArr.length][oldArr[0].length];
56         for (int r = 0; r < oldArr.length; r++) {
57             for (int c = 0; c < oldArr[r].length; c++) {
58                 newArr[r][newArr[r].length - c - 1] = oldArr[r][c];
59             }
60         }
61         return newArr;
62     }
63
64     public Color[][] verticalFlip(Color[][] oldArr) {
65         Color[][] newArr = new Color[oldArr.length][oldArr[0].length];
66         for (int r = 0; r < oldArr.length; r++) {
67             for (int c = 0; c < oldArr[r].length; c++) {
68                 newArr[newArr.length - r - 1][c] = oldArr[r][c];
69             }
70         }
71         return newArr;
72     }
73
74     public Color[][] grayscale(Color[][] oldArr) {
75         final int NUM_COLORS = 3;
76         Color[][] newArr = new Color[oldArr.length][oldArr[0].length];
77         for (int r = 0; r < oldArr.length; r++) {
78             for (int c = 0; c < oldArr[r].length; c++) {
79                 Color col = oldArr[r][c];
80                 double red = col.getRed();
81                 double blue = col.getBlue();
82                 double green = col.getGreen();
83                 int gray = (int) ((red + blue + green) / NUM_COLORS);
84                 Color grayColor = new Color(gray, gray, gray);
85                 newArr[r][c] = grayColor;
86             }
87         }
88         return newArr;
89     }
90
91     public Color[][] blur(Color[][] oldArr) {
92         int radius = 7;
93         int total = 0;
94         Color[][] newArr = new Color[oldArr.length][oldArr[0].length];
95         for (int r = 0; r < oldArr.length; r++) {
96             for (int c = 0; c < oldArr[r].length; c++) {
97                 int redTotal = 0;
98                 int greenTotal = 0;
99                 int blueTotal = 0;
100                for (int i = r - radius; i <= r + radius; i++) {
101                    for (int j = c - radius; j <= c + radius; j++) {
102                        if ((i < oldArr.length) && (i > 0) && (j < oldArr[r].length) && (j >
103                            0)) {
104                            Color col = oldArr[i][j];
105                            redTotal = redTotal + col.getRed();
106                            greenTotal = greenTotal + col.getGreen();
107                            blueTotal = blueTotal + col.getBlue();
108                            total++;
109                        }
110                    }
111                }
112            }
113        }
114    }
```

```
109         }
110     }
111 }
112 redTotal = (int) (redTotal / total);
113 greenTotal = (int) (greenTotal / total);
114 blueTotal = (int) (blueTotal / total);
115 Color newColor = new Color(redTotal, greenTotal, blueTotal);
116 newArr[r][c] = newColor;
117 total = 0;
118     }
119 }
120 return newArr;
121 }
122
123 public Color[][] contrast(Color[][] oldArr) {
124     final int DIVIDER = 127;
125     final double POSITIVE_SHIFT = 1.3;
126     final double NEGATIVE_SHIFT = 0.7;
127     final int COLOR_MAX = 255;
128     Color[][] newArr = new Color[oldArr.length][oldArr[0].length];
129     for (int r = 0; r < oldArr.length; r++) {
130         for (int c = 0; c < oldArr[r].length; c++) {
131             Color col = oldArr[r][c];
132             int red = col.getRed();
133             int blue = col.getBlue();
134             int green = col.getGreen();
135             if (red >= DIVIDER) {
136                 red = (int) (red * POSITIVE_SHIFT);
137             } else {
138                 red = (int) (red * NEGATIVE_SHIFT);
139             }
140             if (red > COLOR_MAX) {
141                 red = COLOR_MAX;
142             }
143             if (green >= DIVIDER) {
144                 green = (int) (green * POSITIVE_SHIFT);
145             } else {
146                 green = (int) (green * NEGATIVE_SHIFT);
147             }
148             if (green > COLOR_MAX) {
149                 green = COLOR_MAX;
150             }
151             if (blue >= DIVIDER) {
152                 blue = (int) (blue * POSITIVE_SHIFT);
153             } else {
154                 blue = (int) (blue * NEGATIVE_SHIFT);
155             }
156             if (blue > COLOR_MAX) {
157                 blue = COLOR_MAX;
158             }
159             Color grayColor = new Color(red, green, blue);
160             newArr[r][c] = grayColor;
161         }
162     }
163     return newArr;
164 }
```

```
165
166 public Color[][] posterize(Color[][] oldArr) {
167     final Color col1 = new Color(62, 47, 91); // Deep Purple
168     final Color col2 = new Color(242, 130, 0); // Orange
169     final Color col3 = new Color(243, 239, 224); // Off White
170     final Color col4 = new Color(230, 161, 215); // Light Purple
171     Color[][] newArr = new Color[oldArr.length][oldArr[0].length];
172     for (int r = 0; r < oldArr.length; r++) {
173         for (int c = 0; c < oldArr[r].length; c++) {
174             Color col = oldArr[r][c];
175             int d1 = (int) (Math.sqrt(Math.pow((col.getRed() - col1.getRed()), 2)
176                 + Math.pow((col.getGreen() - col1.getGreen()), 2)
177                 + Math.pow((col.getBlue() - col1.getBlue()), 2)));
178             int d2 = (int) (Math.sqrt(Math.pow((col.getRed() - col2.getRed()), 2)
179                 + Math.pow((col.getGreen() - col2.getGreen()), 2)
180                 + Math.pow((col.getBlue() - col2.getBlue()), 2)));
181             int d3 = (int) (Math.sqrt(Math.pow((col.getRed() - col3.getRed()), 2)
182                 + Math.pow((col.getGreen() - col3.getGreen()), 2)
183                 + Math.pow((col.getBlue() - col3.getBlue()), 2)));
184             int d4 = (int) (Math.sqrt(Math.pow((col.getRed() - col4.getRed()), 2)
185                 + Math.pow((col.getGreen() - col4.getGreen()), 2)
186                 + Math.pow((col.getBlue() - col4.getBlue()), 2)));
187             int color = Math.min(Math.min(d1, d2), Math.min(d3, d4));
188             if (color == d1) {
189                 newArr[r][c] = col1;
190             }
191             if (color == d2) {
192                 newArr[r][c] = col2;
193             }
194             if (color == d3) {
195                 newArr[r][c] = col3;
196             }
197             if (color == d4) {
198                 newArr[r][c] = col4;
199             }
200         }
201     }
202     return newArr;
203 }
204
205 public Color[][] vintage(Color[][] oldArr) {
206     final int BRGHTNESS = 1;
207     final double CONTRAST = 3.5;
208     final int COLOR_MAX = 255;
209     final int TINT = 50;
210     Color[][] newArr = new Color[oldArr.length][oldArr[0].length];
211     for (int r = 0; r < oldArr.length; r++) {
212         for (int c = 0; c < oldArr[r].length; c++) {
213             Color col = oldArr[r][c];
214             int red = (int) CONTRAST * col.getRed() + BRGHTNESS + TINT;
215             int green = (int) CONTRAST * col.getGreen() + BRGHTNESS;
216             int blue = (int) CONTRAST * col.getBlue() + BRGHTNESS;
217             if (red > COLOR_MAX) {
218                 red = COLOR_MAX;
219             }
220             if (green > COLOR_MAX) {
```

```
221         green = COLOR_MAX;
222     }
223     if (blue > COLOR_MAX) {
224         blue = COLOR_MAX;
225     }
226     Color grayColor = new Color(red, green, blue);
227     newArr[r][c] = grayColor;
228 }
229 }
230 return newArr;
231 }
232
233 @Override
234 public void keyTyped(KeyEvent e) {
235     if (e.getKeyChar() == 'p') {
236         pixels = posterize(pixels);
237     }
238     if (e.getKeyChar() == 'c') {
239         pixels = contrast(pixels);
240     }
241     if (e.getKeyChar() == 'b') {
242         pixels = blur(pixels);
243     }
244     if (e.getKeyChar() == 'h') {
245         pixels = horizontalFlip(pixels);
246     }
247     if (e.getKeyChar() == 'j') {
248         pixels = verticalFlip(pixels);
249     }
250     if (e.getKeyChar() == 'g') {
251         pixels = grayscale(pixels);
252     }
253     if (e.getKeyChar() == 'v') {
254         pixels = vintage(pixels);
255     }
256     repaint();
257 }
258
259 @Override
260 public void keyPressed(KeyEvent e) {
261     // unused
262 }
263
264 @Override
265 public void keyReleased(KeyEvent e) {
266     // unused
267 }
268 }
```