

## 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.*;
6  import javax.swing.*;
7  import java.awt.event.KeyEvent;
8  import java.awt.event.KeyListener;
9
10 public class ImageEditorPanel extends JPanel implements KeyListener {
11     boolean quit = false;
12
13     Color[][] pixels;
14
15     public ImageEditorPanel() {
16         BufferedImage imageIn = null;
17         try {
18             // the image should be in the main project folder, not in \src or \bin
19             imageIn = ImageIO.read(new File("Barack.jpg"));
20         } catch (IOException e) {
21             System.out.println(e);
22             System.exit(1);
23         }
24         pixels = makeColorArray(imageIn);
25         setPreferredSize(new Dimension(pixels[0].length, pixels.length));
26         setBackground(Color.BLACK);
27         addKeyListener(this);
28     }
29
30     public void paintComponent(Graphics g) {
31         // paints the array pixels onto the screen
32         for (int row = 0; row < pixels.length; row++) {
33             for (int col = 0; col < pixels[0].length; col++) {
34                 g.setColor(pixels[row][col]);
35                 g.fillRect(col, row, 1, 1);
36             }
37         }
38     }
39
40     public void run() {
41         while(!quit) {
42             repaint();
43         }
44         pixels = flipHorizontal(pixels);
45         repaint();
46     }
47
48     public Color[][] contrast(Color[][] inputArr) {
```

```
49     final int MIDDLE_NUM = 127;
50     final double CONTRAST = 0.5;
51     int newRed = 0;
52     int newGreen = 0;
53     int newBlue = 0;
54     Color[][] outputArr = new Color[inputArr.length][inputArr[0].length];
55     for(int row = 0; row < inputArr.length; row++) {
56         for(int col = 0; col < inputArr[0].length; col++) {
57             Color c = inputArr[row][col];
58             int red = c.getRed();
59             int green = c.getGreen();
60             int blue = c.getBlue();
61             if(red <= MIDDLE_NUM) {
62                 newRed = (int)(red * CONTRAST);
63             } else {
64                 newRed = (int)(red + (255 - red) * (CONTRAST));
65             }
66             if(green <= MIDDLE_NUM) {
67                 newGreen = (int)(green * CONTRAST);
68             } else {
69                 newGreen = (int)(green + (255 - green) * (CONTRAST));
70             }
71             if(blue <= MIDDLE_NUM) {
72                 newBlue = (int)(blue * CONTRAST);
73             } else {
74                 newBlue = (int)(blue + (255 - blue) * (CONTRAST));
75             }
76             outputArr[row][col] = new Color(newRed, newGreen, newBlue);
77         }
78     }
79     return outputArr;
80 }
81
82 public Color[][] postarizeFilter(Color[][] inputArr) {
83     Color[][] outputArr = new Color[inputArr.length][inputArr[0].length];
84     Color[] palette = {
85         new Color(114, 219, 139),
86         new Color(104, 151, 227),
87         new Color(217, 195, 230),
88         new Color(26, 4, 2),
89     };
90     for(int row = 0; row < inputArr.length; row++) {
91         for(int col = 0; col < inputArr[0].length; col++) {
92             Color c = inputArr[row][col];
93             Color newC = getNearestColor(c, palette);
94             outputArr[row][col] = newC;
95         }
96     } return outputArr;
97 }
98
```

```
99     public Color getNearestColor(Color c,Color[] palette) {
100         Color nearest = null;
101         double min = Integer.MAX_VALUE;
102         for(Color p: palette) {
103             int dRed = c.getRed() - p.getRed();
104             int dGreen = c.getGreen() - p.getGreen();
105             int dBlue = c.getBlue() - p.getBlue();
106             double loss = Math.sqrt(Math.pow(dRed,2) + Math.pow(dGreen,2) + Math.pow(dBlue,2));
107             if (loss < min) {
108                 min = loss;
109                 nearest = p;
110             }
111         }
112         return nearest;
113     }
114
115     public Color[][] colorNeg(Color[][] inputArr) {
116         Color[][] outputArr = new Color[inputArr.length][inputArr[0].length];
117         for(int row = 0; row < inputArr.length; row++) {
118             for(int col = 0; col < inputArr[0].length; col++) {
119                 Color c = inputArr[row][col];
120                 int red = 255 - c.getRed();
121                 int green = 255 - c.getGreen();
122                 int blue = 255 - c.getBlue();
123                 Color newC = new Color(red, green, blue);
124                 outputArr[row][col] = newC;
125             }
126         }
127         return outputArr;
128     }
129
130     public Color[][] grayScale(Color[][] inputArr) {
131         Color[][] outputArr = new Color[inputArr.length][inputArr[0].length];
132         for(int row = 0; row < inputArr.length; row++) {
133             for(int col = 0; col < inputArr[0].length; col++) {
134                 Color c = inputArr[row][col];
135                 int red = c.getRed();
136                 int green = c.getGreen();
137                 int blue = c.getBlue();
138                 int average = (red + green + blue) / 3;
139                 Color newC = new Color(average, average, average);
140                 outputArr[row][col] = newC;
141             }
142         }
143         return outputArr;
144     }
145
146     //Multiple - pixel algorithm template
147     public Color[][] multiPixelAlgo(Color[][] inputArr) {
148         final int RADIUS = 3;
```

```
149     Color[][] outputArr = new Color[inputArr.length][inputArr[0].length];
150     for(int row = 0; row < inputArr.length; row++) {
151         for(int col = 0; col < inputArr[0].length; col++) {
152             int pixelNum = 0;
153             int totalRed = 0;
154             int totalGreen = 0;
155             int totalBlue = 0;
156             // initialize some variables
157             // visit the neighbors centered at row, col
158             for(int row2 = row - RADIUS; row2 <= row + RADIUS; row2++) {
159                 for(int col2 = col - RADIUS; col2 <= col + RADIUS; col2++) {
160                     if(row2 >= 0 && row2 < inputArr.length && col2 >= 0 && col2 <
inputArr[0].length) {
161                         //do some work with this neighbor
162                         Color c = inputArr[row2][col2];
163                         totalRed += c.getRed();
164                         totalGreen += c.getGreen();
165                         totalBlue += c.getBlue();
166                         pixelNum++;
167                     }
168                 }
169             }
170             int avgRed = totalRed / pixelNum;
171             int avgGreen = totalGreen / pixelNum;
172             int avgBlue = totalBlue / pixelNum;
173             Color newC = new Color(avgRed, avgGreen, avgBlue);
174             outputArr[row][col] = newC;
175         }
176     }
177     return outputArr;
178 }
179
180 //Single - pixel algorithm template
181 public Color[][] singlePixelAlgo(Color[][] inputArr) {
182     Color[][] outputArr = new Color[inputArr.length][inputArr[0].length];
183     for(int row = 0; row < inputArr.length; row++) {
184         for(int col = 0; col < inputArr[0].length; col++) {
185             Color c = inputArr[row][inputArr[0].length - col - 1];
186             //based on the values of Color c, create a new color
187             outputArr[row][col] = c;
188         }
189     }
190     return outputArr;
191 }
192
193 public Color[][] flipHorizontal(Color[][] inputArr) {
194     Color[][] outputArr = new Color[inputArr.length][inputArr[0].length];
195     for(int row = 0; row < inputArr.length; row++) {
196         for(int col = 0; col < inputArr[0].length; col++) {
197             Color c = inputArr[row][inputArr[0].length - col - 1];
```

```
198         outputArr[row][col] = c;
199     }
200 }
201 return outputArr;
202 }
203
204 public Color[][] flipVertical(Color[][] inputArr) {
205     Color[][] outputArr = new Color[inputArr.length][inputArr[0].length];
206     for(int row = 0; row < inputArr.length; row++) {
207         for(int col = 0; col < inputArr[0].length; col++) {
208             Color c = inputArr[inputArr.length - row - 1][col];
209             outputArr[row][col] = c;
210         }
211     }
212     return outputArr;
213 }
214
215 public Color[][] brighten(Color[][] inputArr) {
216     Color[][] outputArr = new Color[inputArr.length][inputArr[0].length];
217     for(int row = 0; row < inputArr.length; row++) {
218         for(int col = 0; col < inputArr[0].length; col++) {
219             Color c = inputArr[row][col];
220             outputArr[row][col] = c.brighter();
221         }
222     }
223     return outputArr;
224 }
225
226 public Color[][] makeColorArray(BufferedImage image) {
227     int width = image.getWidth();
228     int height = image.getHeight();
229     Color[][] result = new Color[height][width];
230
231     for (int row = 0; row < height; row++) {
232         for (int col = 0; col < width; col++) {
233             Color c = new Color(image.getRGB(col, row), true);
234             result[row][col] = c;
235         }
236     }
237     // System.out.println("Loaded image: width: " +width + " height: " + height);
238     return result;
239 }
240
241 public void keyPressed(KeyEvent e) {
242     //unused
243 }
244 public void keyReleased(KeyEvent e) {
245     //unused
246 }
247 public void keyTyped(KeyEvent e) {
```

```
248     if (e.getKeyChar() == 'v') {
249         pixels = flipVertical(pixels);
250     }
251     if (e.getKeyChar() == 'h') {
252         pixels = flipHorizontal(pixels);
253     }
254     if (e.getKeyChar() == 'p') {
255         pixels = postarizeFilter(pixels);
256     }
257     if (e.getKeyChar() == 'n') {
258         pixels = colorNeg(pixels);
259     }
260     if (e.getKeyChar() == 'c') {
261         pixels = contrast(pixels);
262     }
263     if (e.getKeyChar() == 'a') {
264         pixels = brighten(pixels);
265     }
266     if (e.getKeyChar() == 'g') {
267         pixels = grayScale(pixels);
268     }
269     if (e.getKeyChar() == 'b') {
270         pixels = multiPixelAlgo(pixels);
271     }
272 }
273 }
274
```